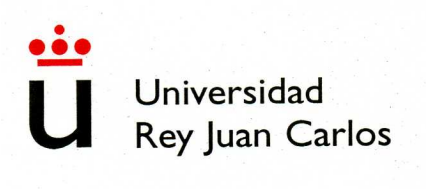


Estadística con R

Raquel Montes

.

Índice general

Capítulo 1

R

1.1. Introducción

R es un lenguaje de programación especialmente indicado para el análisis estadístico, que se maneja a través de una consola en la que se introduce el código.

R fue inicialmente diseñado por R. Gentleman y R. Ihaka, miembros del Departamento de Estadística de la Universidad de Auckland, en Nueva Zelanda. Sin embargo, una de las grandes ventajas de R es que hoy en día, es fruto del esfuerzo de miles de personas de todo el mundo, que colaboran en su desarrollo. El código de R está disponible como software libre bajo las condiciones de la licencia GNU-GPL, y puede ser instalado en sistemas operativos tipo Windows, Linux o MacOSX.

La página principal desde la que se puede acceder tanto a los archivos necesarios para su instalación como al resto de recursos del proyecto R es `http://www.r-project.org`.

1.2. Instalación de R

La descarga del archivo de instalación se realiza desde `http://cran.es.r-project.org/`. Si por ejemplo trabajamos en Windows, en dicha página debemos elegir la instalación *Windows*, la descarga de *base* y finalmente, el archivo de instalación (ver Figuras 1.1, 1.2, 1.3). Una vez concluida la instalación, podemos ejecutar el programa desde el icono que nos genera en el escritorio.

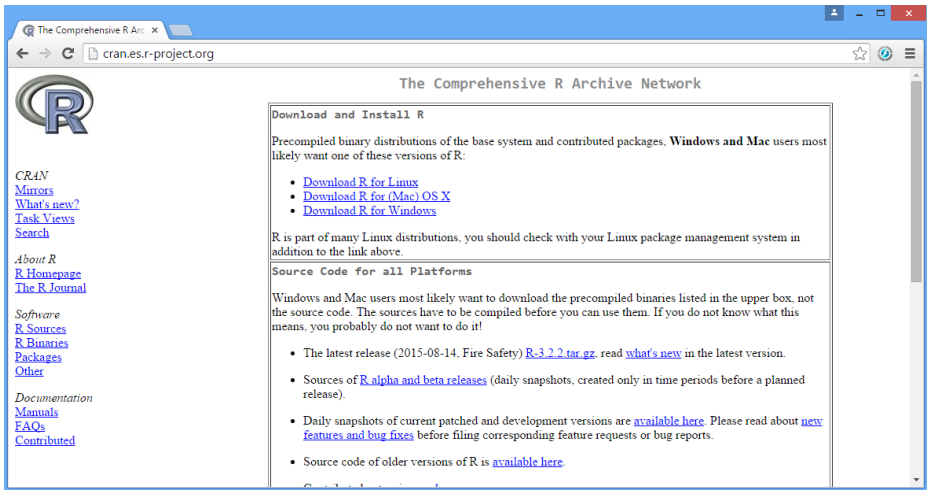


Figura 1.1: Elegimos el sistema operativo.

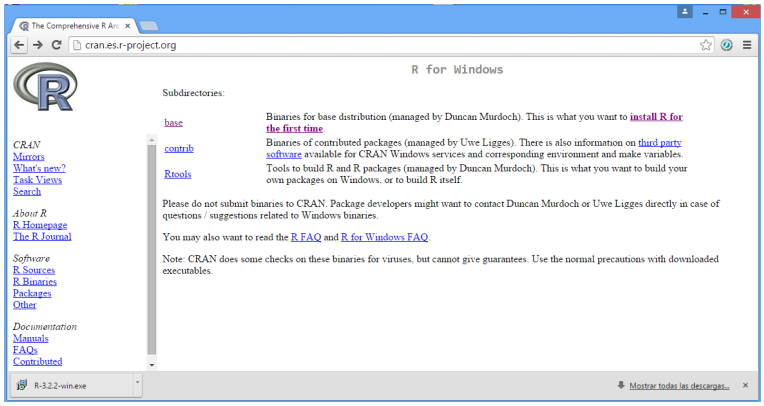


Figura 1.2: Pinchamos en *base*.

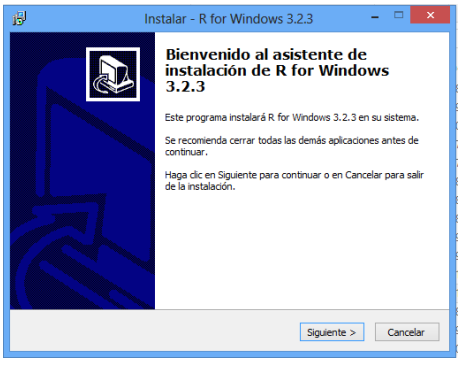


Figura 1.3: Abrimos el ejecutable.

1.3. La consola y el editor de R

Lo primero que nos aparece es una ventana, también llamada consola, donde podemos manejar R mediante la introducción de código. Por ejemplo, podemos escribir `2+2` en ella, pulsando Intro, lo que nos devolverá en la misma consola el valor `4`.

Sin embargo, esta no es la manera más eficiente de trabajar en R. Si estamos realizando un trabajo de mediana complejidad, será muy útil trabajar con todo el código que solicitemos a R en un entorno donde podamos corregirlo, retocarlo, repetirlo, guardarlo para continuar el trabajo en otro momento, etc. Esta es la función del editor de R. Podemos ver en la Figura 1.4 cómo acceder a un *nuevo script*, un documento en blanco del editor de R.

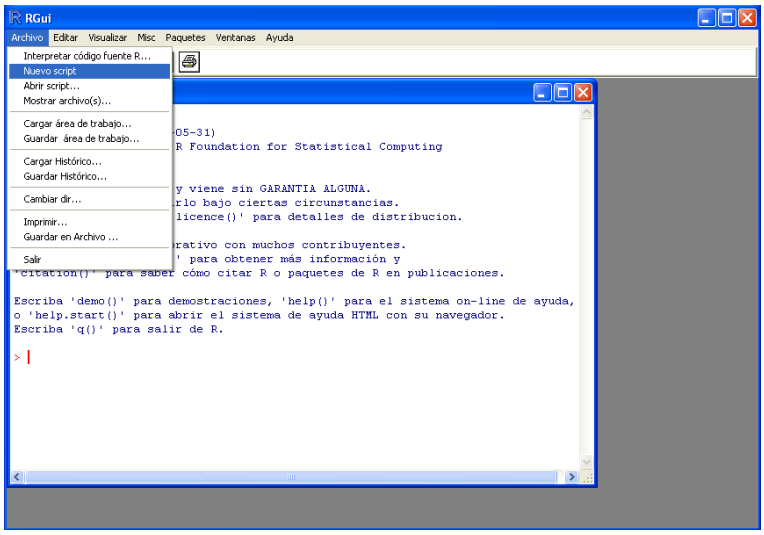


Figura 1.4: Consola de R y forma de acceder al editor

Una vez que hayamos seleccionado un *nuevo script* en el menú *Archivo*, para mayor comodidad, elegiremos la opción *Divida horizontalmente* (o *Divida verticalmente*) del menú *Ventana* de la consola, después de lo cuál, el aspecto será el que aparece en la Figura 1.5.

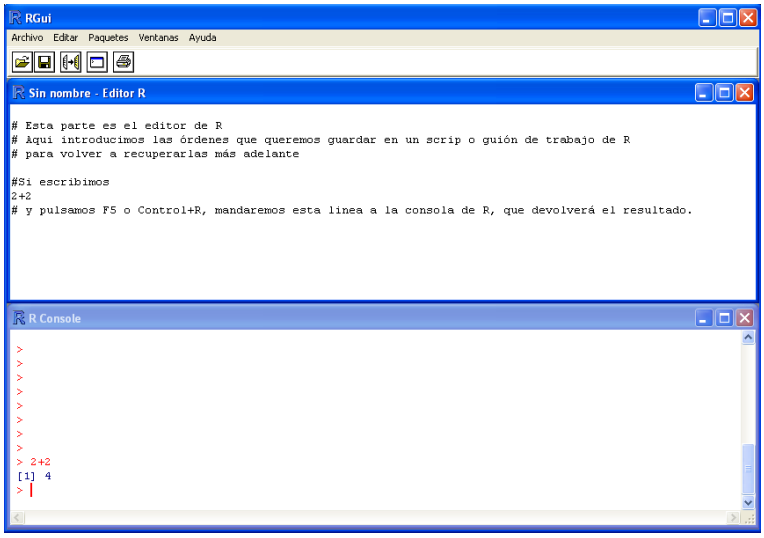


Figura 1.5: Consola y editor de R

En esa misma figura, observamos que ya aparecen algunas líneas en el editor. Puede verse que es posible incluir comentarios que R no leerá si utilizamos líneas que comiencen con el carácter #. Por el contrario, si escribimos cualquier orden no antecedida de # y queremos solicitar la respuesta a R, podemos hacerlo mediante la tecla *F5* situándonos en cualquier posición de esa línea (no necesariamente en el final) o la combinación de teclas *Control+R*. Asimismo, si seleccionamos con el ratón más de una línea, éstas pueden ser ejecutadas simultáneamente también con *F5* o *Control+R*.

Utilizar un script para trabajar es muy cómodo, ya que nos permite modificar nuestras líneas de código y guardarlas para el futuro. Para ello, utilizaremos la opción *Guardar* o *Guardar como* del menú *Archivo* de la consola. Evidentemente, después podremos recuperar el *script* previamente guardado mediante la opción *Abrir script* del mismo menú.

1.4. R-Studio

RStudio es un entorno de desarrollo integrado para el uso de R, disponible para Windows, Mac y Linux. Incluye una consola, editor de texto que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo.

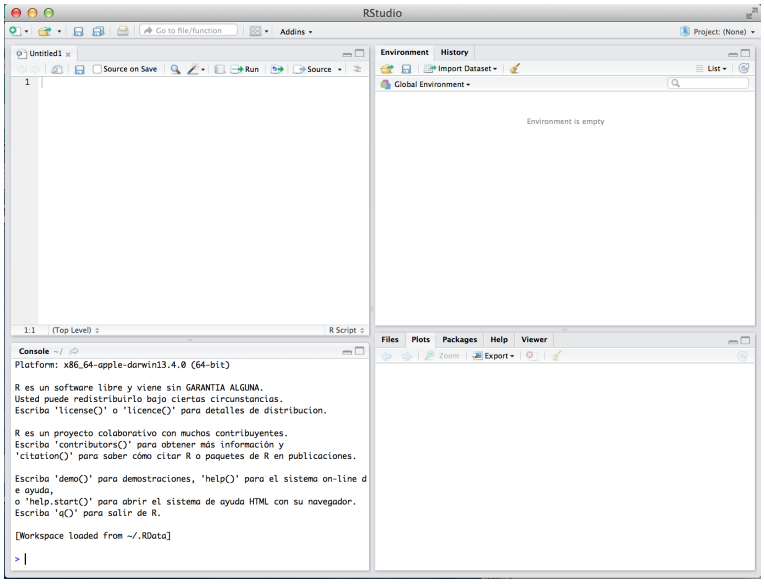


Figura 1.6: R-Studio

Podemos descargar e instalar R-Studio desde la página `https://www.rstudio.com/` y su uso es muy recomendable de cara a organizar el trabajo con R.

1.5. Introducción al lenguaje de R

1.5.1. Algunos tipos de objetos de R

En el lenguaje de R, los elementos u objetos que se vayan definiendo, bien por nosotros mismos, bien como resultado del programa, pueden y deben ser distinguidos para su uso correcto.

Concretamente, vamos a hablar de:

- Vectores.
- Matrices.
- Factores.
- Hojas de datos.

Otros tipos de objetos, como las listas, las variables indexadas (arrays), las funciones o los modelos, son también interesantes, aunque no los vamos a incluir aquí.

1.5.1.1. Vectores

Un vector en R puede contener una colección de números o de caracteres no numéricos. Para definir un vector, por ejemplo, el vector $x = (1, 3, 5)$, usaríamos la orden

```
x=c(1,3,5)
```

Así podremos llamar al vector x en el futuro. Observemos que se utiliza el operador `=` y que es la función de concatenación `c()` la que construye el vector.

También es posible definir un vector de números consecutivos, por ejemplo, el vector $(1, 2, 3, 4, 5)$ mediante

```
1:5
```

De forma más general, la función `seq()` permite definir secuencias desde un inicio hasta un fin con una determinada separación entre ellos. Por ejemplo,

```
y=seq(-3,3,0.5)
y
```

La función `rep()` para definir vectores como repetición de otros vectores. Por ejemplo,

```
rep(0,100)
rep(1:3,3)
```

Si queremos saber la longitud de un vector, usaremos `length()`. Por ejemplo,

```
length(y)
```

Un vector puede incluir caracteres en lugar de números, siempre que éstos estén entre comillas. Por ejemplo, podríamos definir el vector

```
genero=c("Mujer","Hombre")
genero
```

1.5.1.2. Factores

Los factores son un tipo especial de objetos que permiten analizar un conjunto de datos clasificados según las características que definen el factor.

Por ejemplo, podríamos definir el factor sexo, correspondiente a un grupo de 7 personas, de la siguiente manera:

```
genero=factor(c("Mujer", "Hombre", "Mujer", "Mujer","Hombre", "Hombre", "Mujer"),
levels=c("Mujer","Hombre"))
genero
```

1.5.1.3. Matrices

Una matriz se define mediante la función `matrix()` a la que hay que especificar sus elementos y su dimensión.

Por ejemplo, para definir la matriz

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

podemos usar

```
m=matrix(c(1,2,3,4,5,6,7,8,9),3,3)
```

Las dimensiones (número de filas y columnas) de la matriz pueden obtenerse mediante

```
dim(m)
```

Si queremos obtener elementos concretos de una matriz lo haremos utilizando corchetes para indicar filas y columnas. Por ejemplo,

```
m[2,3]
m[1:2,2:3]
m[,c(1,3)]
```

Esa misma sintaxis permite manejar los elementos de un vector.

Tanto para vectores como para matrices, funcionan las operaciones suma y diferencia sin más complicaciones. En el caso del producto, sin embargo, es importante distinguir entre la multiplicación elemento a elemento y el producto matricial.

```
m+m
m-m
m*m
m%*%m
```

1.5.1.4. Hojas de datos

Las hojas de datos constituyen la manera más eficiente en que R puede analizar un conjunto estadístico de datos. Habitualmente se configuran de tal manera que cada fila se refiere a un

elemento de la muestra que se analiza, mientras que cada columna hace referencia a las distintas variables analizadas. Con una nomenclatura estadística, diríamos que las filas son los casos y las columnas son las variables. Esa configuración hace que visualmente una hoja de datos parezca una matriz. Sin embargo, como objetos de R, son cosas distintas.

Vamos a ver cómo se construye una hoja de datos con los datos de 3 personas, que incluye el color de sus ojos como factor, su peso y su altura.

Empezaríamos definiendo el color de los ojos:

```
ojos=factor(c("Azules","Marrones","Marrones"),
levels=c("Azules","Marrones","Verdes","Negros"))
```

Supongamos que los pesos y las alturas son, respectivamente, 68, 75, 88 y 1.65, 1.79, 1.85. Entonces, definiríamos la hoja de datos mediante

```
hd=data.frame(Color.ojos=ojos,Peso=c(68,75,88),Altura=c(1.65,1.79,1.85))
```

Así, tendremos tres variables, llamadas `Color.ojos, Peso` y `Altura`. También podemos forzar a que una matriz se convierta en una hoja de datos y viceversa. Por ejemplo,

```
m2=as.matrix(hd)
hd2=as.data.frame(m)
```

convertirían `hd` en una matriz, que llamamos `m2` y `m` en una hoja de datos, que llamamos `hd2`. Si ponemos

```
names(hd2)
```

vemos los nombres que R ha elegido por defecto para las variables.

Si queremos modificar esos nombres de las variables, podemos usar de nuevo la función `names()`, forzando la asignación:

```
names(hd2)=c("Variable 1","Variable 2","Variable 3")
```

La manera en que podemos acceder a los elementos de una hoja de datos es doble:

1. Podemos usar el mismo método que para las matrices.
2. Podemos usar el operador `$`.

Para obtener los datos de la variable `Color.ojos`, por ejemplo, escribiríamos

```
hd$Color.ojos
```

Para saber el número de filas y de columnas de una hoja de datos utilizaremos las funciones `nrow()` y `ncol()`. Por ejemplo,

```
ncol(hd)
nrow(hd)
```

Cuando no estamos seguros de que un objeto de R, que sabemos que contiene datos, sea una hoja de datos o una matriz podemos usar funciones que nos informan sobre el carácter de los objetos, tales como, `is.vector(), is.matrix()` e `is.data.frame()`. Así, por ejemplo,

```
is.data.frame(m)
is.data.frame(hd2)
```

1.5.2. Funciones más comunes en R

Veamos algunas funciones de R típicas. Por ejemplo:

- `sum()` proporciona la suma de los elementos del argumento.
- `cumsum()` proporciona un vector con la suma acumulada del vector argumento.
- `rowSums()` y `colSums()` suman, por filas y por columnas respectivamente, los datos de una hoja de datos.
- `prod()` y `cumprod()` son el equivalente a `sum()` y `cumsum()` para el producto.
- `sqrt()` es la función raíz cuadrada.
- `log()` es la función logaritmo natural o neperiano.

- `log10()` es el logaritmo en base 10.
- `exp()` es la función exponencial.
- `max()` y `min()` proporcionan el máximo y el mínimo del argumento (habitualmente, un vector).
- `sort()` proporciona la ordenación de un vector de menor a mayor.

1.5.3. Operaciones lógicas

Algunos operadores lógicos utilizados con frecuencia, son:

- `<, >, <=` y `>=` son los operadores menor, mayor, menor o igual que y mayor o igual que, respectivamente. Por ejemplo,

  ```
  x>=1.5
  ```

- `==` es el operador de igualdad. Por ejemplo,

  ```
  hd$Color.ojos=="Marrones"
  ```

- `&` y `|` son los operadores y y o, respectivamente. Por ejemplo,

  ```
  (m>2)&(m<4)
  ```

1.5.4. Definición de Funciones

Una de las características más atractivas de R es la flexibilidad que tenemos para modificar funciones ya existentes y para crear otras nuevas. La estructura básica de una función es

```
nombre_funcion = function(argumento1, argumento2, ...){expresión}
```

La expresión es una fórmula matemática que calcula valores numéricos o crea objetos basados en los argumentos que previamente hemos especificado. Para utilizar una función, simplemente introducimos el nombre de ésta y especificamos el valor de los distintos argumentos.

Supongamos que estamos interesados en sumar los primeros n números naturales. La fórmula para calcular dicha suma es $n \times (n+1)/2$. Podemos entonces crear la función `SUM.N`

```
SUM.N = function(n){(n)*(n+1)/2}
```

y utilizar dicha función para calcular la suma de los 10 primeros números naturales, simplemente introduciendo

```
SUM.N(10)
```

La siguiente función suma los cuadrados de los valores de un vector o una matriz x:

```
sum.sq=function(x) {sum(x^2)}
```

1.5.5. La ayuda de R

Si deseamos obtener ayuda sobre el uso de alguna función cuyo nombre conocemos, podemos utilizar la ayuda de R simplemente escribiendo el nombre de esa función después de un signo de interrogación. Por ejemplo,

```
?sort
```

abrirá una ventana de nuestro explorador con todos los detalles sobre el uso de esa función, incluyendo interesantes ejemplos. Pero, ¿qué ocurre si queremos ayuda sobre un aspecto del que desconocemos qué función nos lo facilita? Supongamos, por ejemplo, que deseamos saber cómo se realiza la descomposición de Choleski de una matriz. En ese caso, si no sabemos qué función facilita esa descomposición, pondremos

```
??choleski
```

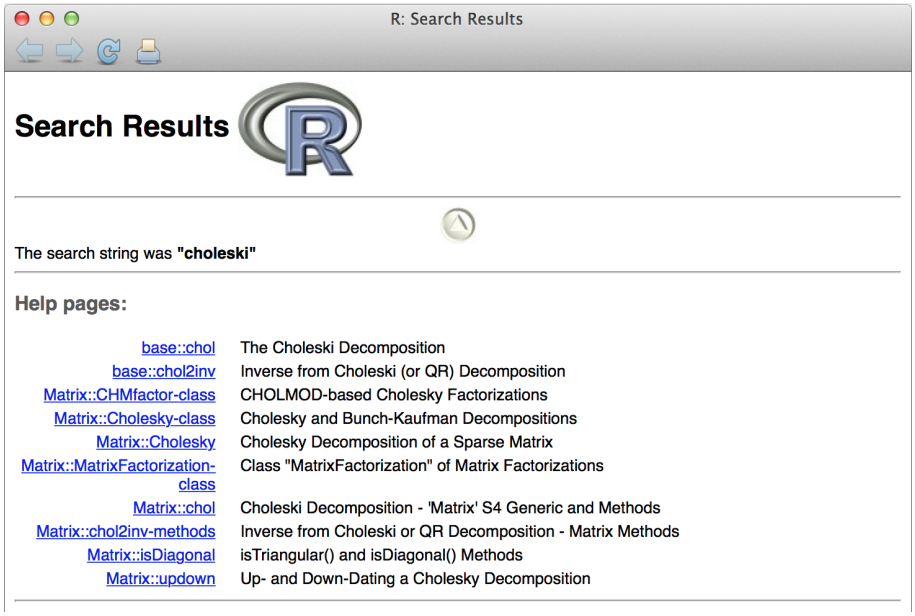


Figura 1.7: Información que da la ayuda de R a propósito de la entrada choleski

Eso abrirá una ventana de R con todas las funciones que incluyen la palabra Choleski en su ayuda. El resultado concreto podemos verlo en la Figura 1.7.

En esa ventana aparecen los nombres de las funciones junto con el paquete de R en el que se encuentra esa función. Por ejemplo, la función `chol()` está en el paquete base, que es el que por defecto se incorpora al arrancar R, su núcleo. Si queremos ayuda concreta sobre esta función, sólo tenemos que ejecutar `?chol`. Sin embargo, la función `Choleski()` se encuentra dentro del paquete `Matrix`, por lo que tendremos que cargar este paquete antes de pedir la ayuda, para ello utilizamos la función `library()`:

```
library(Matrix)
?Choleski
```

¿Qué ocurre si necesitamos ayuda sobre algo que está en una función de un paquete que nosotros no tenemos instalado? Tenemos que tener en cuenta que, al instalar R, tan sólo incorporamos una mínima parte de los paquetes que el proyecto CRAN tiene, gracias a la colaboración de los miles de desarrolladores de R, así que, si no encontramos ayuda en los paquetes instalados por defecto, puede que aún así, exista un paquete en CRAN donde haya algo al respecto. La cuestión es, ¿cómo acceder a esa información?

Si por ejemplo, buscamos información sobre funciones que, en algún lugar de la ayuda, incluyan la palabra `engineering`, tan sólo tenemos que teclear

```
RSiteSearch("engineering")
```

Eso abrirá una ventana de nuestro navegador donde podemos elegir el ámbito de nuestra consulta, como se muestra en la Figura 1.8.

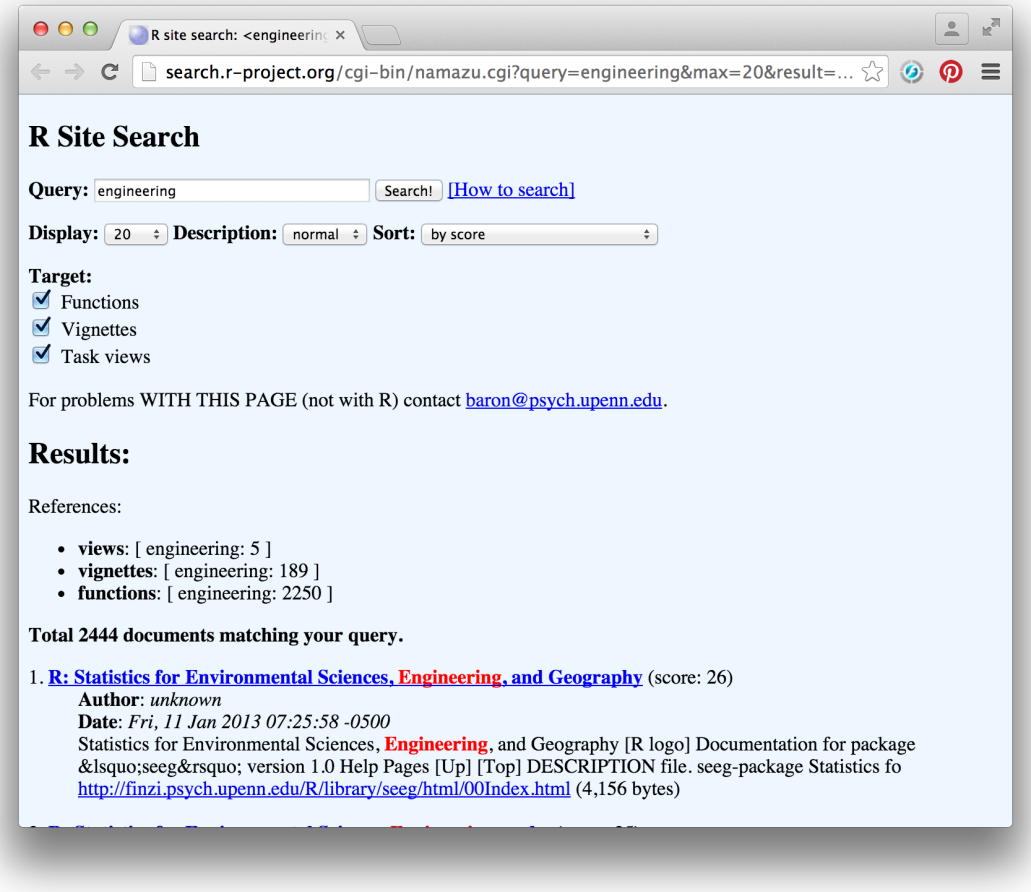


Figura 1.8: Información sobre funciones que, en algún lugar de la ayuda, incluyan la palabra *engineering*.

1.6. Organización de datos en R

Supongamos que tenemos información sobre n individuos, que se refiere a k variables. En estadística, la forma en que se organiza toda esta información es una matriz de dimensiones $n \times k$ donde cada fila representa un individuo o caso y cada columna representa una variable o dato.

Ejemplo: Los siguientes datos representan los tiempos de acceso de 10 usuarios a cierta página web antes (x) y después (y) de introducir una mejora en el diseño de dicha página.

x	y
3.48	1.99
1.38	0.38
3.27	2.65
4.27	3.54
2.75	1.46
5.28	3.93
3.43	3.31
3.85	2.71
2.77	1.21
2.97	1.32

Observamos que cada fila representa a un individuo (en este caso, internauta) y cada columna a una variable (tiempo de acceso antes o después de mejorar la página).

1.6.1. Introducir una hoja de datos

Para crear una hoja de datos, comenzamos introduciendo los datos de las variables x e y en forma de vector.

```
x = c(3.48, 1.38, 3.27, 4.27, 2.75, 5.28, 3.43, 3.85, 2.77, 2.97)
y = c(1.99, 0.38, 2.65, 3.54, 1.46, 3.93, 3.31, 2.71, 1.21, 1.32)
```

y a continuación definimos la hoja de datos:

```
tiempos = data.frame(antes=x,despues=y)
```

Así, hemos llamado a la hoja de datos `tiempos`. Por su parte, a la primera variable la hemos llamado `antes` y a la segunda `despues`. Si escribimos en la consola `tiempos`, visualizamos el resultado.

```
tiempos
```

Si ahora queremos trabajar con alguna de esas dos variables tenemos dos opciones:

1. Podemos referirnos a cualquiera de ellas poniendo el nombre de la hoja seguido del símbolo `$` y del nombre de la variable. Es decir,

   ```
   tiempos$antes
   tiempos$despues
   ```

2. Alternativamente, si no queremos escribir en demasiadas ocasiones `tiempos$`, podemos hacer que la hoja de datos se convierta en la hoja de datos activa mediante la función `attach`:

   ```
   attach(tiempos)
   antes
   despues
   detach(tiempos)
   ```

Si queremos referirnos a un elemento concreto de una hoja de datos, ya sabemos que también podemos identificarlo por su posición dentro de la matriz que constituye la hoja de datos. Por ejemplo, para saber el tiempo de acceso antes de la mejora (1^a variable) del 5^o internauta de la muestra, usamos

```
tiempos$antes[5]
```

o también,

```
tiempos[5,1]
```

Supongamos que queremos los tiempos después de la mejora (2^a variable) de los 5^o , 8^o y 10^o internautas, podemos usar

```
tiempos$despues[c(5,8,10)]
```

o bien

```
tiempos[c(5,8,10),2]
```

O por ejemplo, si deseamos ambos tiempos del 3^o al 8^o internauta,

```
tiempos[3:8,1:2]
```

o simplemente

```
tiempos[3:8,]
```

1.6.2. Almacenar datos: Las funciones save y load

Debemos tener presente que los datos que hemos introducido están sólo almacenados temporalmente, y que si cerramos R serán eliminados.

La función `save()` nos permite guardar varios objetos en archivos para poder utilizarlos con posterioridad. Su uso es muy sencillo, basta con indicarle qué objetos queremos guardar y el nombre del archivo con el que lo queremos guardar. Aunque no es imprescindible, es recomendable que este fichero tenga las extensiones propias de los ficheros de datos de R, que son `.RData` o `.rda`. Vamos a aplicarlo a la hoja de datos con los resultados de las dos tiempos:

```
setwd("D:/Estadistica/Practicas")
save(tiempos,file="datos_tiempos.RData")
```

1. La función `setwd()` permite cambiar el directorio donde estamos trabajando (working directory). Hay que tener en cuenta que los datos se guardarán en ese directorio.
2. Después hemos ejecutado la función `save()` para guardar `tiempos` en el fichero `datos_tiempos.RData`.

Si reiniciamos el programa o borramos todos los objetos mediante

```
rm(list=ls(all=TRUE))
```

(o eligiendo en el menú de la consola de R *Misc* → *Remover todos los objetos*), al poner `tiempos` daría error, porque tal objeto ya no existe. Sin embargo, al cargar el archivo `datos_tiempos.RData` mediante la función `load()`, recuperamos la hoja de datos:

```
load("datos_tiempos.RData").
```

El directorio de trabajo: Es importante que al comenzar a trabajar con datos en R, tengamos claro cuál es nuestro directorio de trabajo (o working directory). Suelen ser muchos los problemas que se encuentran cuando no se tiene este concepto claro, así que vamos a centrarnos un poco en él.

En primer lugar, lo más importante es situar a R en el directorio de trabajo de nuestro ordenador, es decir, el sitio donde localizaremos todos los datos, los resultados, los gráficos, etc., de nuestro análisis. Es recomendable situar cada trabajo en un directorio distinto.

Podemos situarnos en un directorio concreto de dos formas:

1. Mediante la función `setwd()`. Como argumento de esta función debemos escribir la ruta que conduce en nuestro ordenador al directorio de trabajo, entre comillas. Por ejemplo,

   ```
   setwd("D:/Estadistica/Practicas")
   ```

2. Utilizando la opción *Cambiar dir...* del menú *Archivo* de la consola de R.

Esta segunda opción es más sencilla la primera vez, pero cada vez que empecemos a trabajar tendremos que hacerlo de nuevo. Lo mejor es hacer que la primera línea de nuestro script siempre sea la que determina el directorio de trabajo mediante la función `setwd()`.

Mediante la función `getwd()` podemos saber si estamos en el directorio correcto. Por ejemplo, si yo abro R, y escribo `getwd()`, obtengo

```
[1] "C:/Documents and Settings/rmontes/Mis documentos"
```

Pero si ahora utilizo la opción *Cambiar dir...* del menú *Archivo* de la consola de R para cambiarme a mi directorio de trabajo donde guardo todo lo relativo a este curso y vuelvo a escribir `getwd()`, el resultado ahora es

```
[1] "D:/Estadistica/Practicas"
```

Así pues, un consejo en forma de esquema para evitarnos problemas con el directorio de trabajo:

1. Arrancamos R y buscamos nuestro directorio de trabajo mediante la opción *Cambiar dir...* del menú *Archivo* de la consola de R.
2. Escribimos `getwd()` y copiamos el resultado que sale entre comillas.
3. Escribimos la primera línea de nuestro script con `setwd()` incluyendo como argumento el resultado anterior que hemos copiado.

De ahora en adelante, ya sólo tendremos que ejecutar esa primera línea para situarnos en el directorio correcto.

Búsqueda de archivos y objetos. Las funciones dir() y objects(): Supongamos que estamos dentro del directorio de trabajo correcto (o eso pensamos) y queremos ver el nombre de los archivos de ese directorio. Tan sólo tenemos que ejecutar `dir()` y nos saldrán todos los archivos del directorio. Sin embargo, si sólo queremos saber el nombre de los archivos del directrorio que son archivos de datos de R, escribiremos `dir(pattern=".RData")` o `dir(pattern=".rda")`. Con la opción `pattern` le estamos pidiendo sólo los archivos que incluyan en su nombre la expresión que corresponde con las extensiones de los archivos de datos de R, `.RData` y `.rda`.

Con eso ya tenemos toda la información necesaria para cargar el conjunto de datos que queramos mediante la función `load()`. No olvidemos que debemos especificarle a esta función el nombre completo del archivo, entre comillas. Si usamos `dir()`, podemos copiar y pegar ese nombre sin problemas.

Al cargar un fichero de datos de R, este fichero incorpora a nuestro entorno de trabajo de R uno o varios objetos, habitualmente una o varias hojas de datos. Ejecutando `objects()` obtendremos todos los nombres de todos los objetos que en ese momento están definidos en nuestra sesión de R.

1.6.3. Importar datos: La función read.table

Introducir datos a mano puede convertirse en una tarea muy pesada, especialmente si el número de casos o de variables es medianamente alto. Por otra parte, es bastante común tener los datos

almacenados en algún tipo de formato electrónico, por lo que sería útil que nuestro programa estadístico, en este caso R, lea esos datos directamente.

Los formatos de archivo más habituales en los que nos podemos encontrar unos datos son los archivos tipo texto (con extensión *.txt*). Existen otros muchos formatos, pero casi siempre son convertibles a archivos de texto.

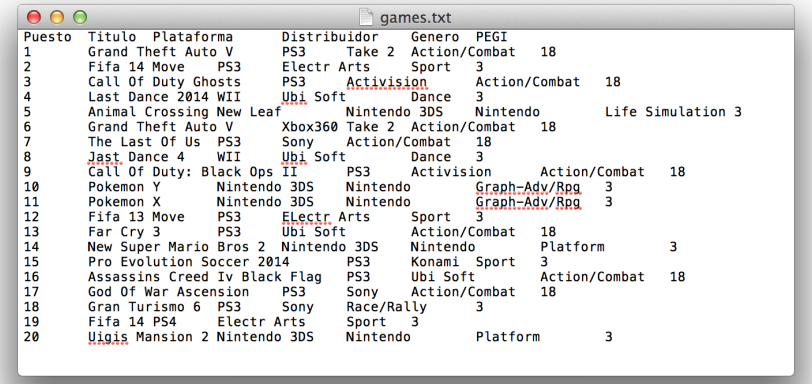


Figura 1.9: Archivo de texto. Variables separadas por tabulaciones y nombres de variables incluidos en la primera línea.

A la hora de importar datos, es necesario fijarse en tres cuestiones:

- si el archivo incluye, o no, los nombres de las variables,
- el carácter que separa las variables, y
- el carácter que distingue los decimales.

Ejemplo: En el archivo games.txt *aparecen datos relativos a los videojuegos más vendidos en España en 2013, según los datos aportados por ADESE (Asociación Española de Distribuidores y Editores de Software de Entretenimiento).*

Si abrimos este fichero, tiene el aspecto que aparece en la Figura 1.9. En ella podemos ver que, en efecto, se incluye el nombre de las variables y que éstas están separadas por tabulaciones. En este caso, no aparven cifras con decimales, por lo que podemos obviar este detalle.

La función que R utiliza para importar archivos de tipo texto es `read.table`. Esta función tiene multitud de opciones, pero nosotros vamos a destacar sólo las que creemos que son más importantes. Concretamente, la sintaxis de dicha función, en general, sería la siguiente:

```
read.table(archivo,header=FALSE,sep="",dec=".",na.strings="NA")
```

En esta línea:

- `archivo` sería el nombre del archivo que queremos importar. Opcionalmente, se puede importar desde el portapapeles, en ese caso, el valor debe ser `"Clipboard"`.
- `header` puede tomar el valor TRUE, si sabemos que la primera línea del archivo (cabecera) contiene los nombres de las variables, o el valor FALSE, si no lo hace.
- `sep` se refiere al carácter que separa los datos. En nuestro ejemplo son tabulaciones, luego debemos poner `"\t"`. El valor por defecto es vacío, que corresponde a uno o más espacios en blanco o a tabulaciones.
- `dec` se refiere al carácter que separa los números decimales. Hay que tener cuidado con él porque en español lo correcto es separar con comas, pero en el mundo anglosajón lo es hacerlo con puntos. De hecho, el punto es la opción por defecto.
- `na.strings` se refiere al carácter que en el archivo original identifica a los datos faltantes. Por defecto, se supone que un dato faltante aparecerá como `NA`, pero podemos poner cualquier otro. Si el dato faltante simplemente no aparece en el archivo original, será entendido como tal dato faltante sin necesidad de especificar nada más.

Por ejemplo, en el caso del archivo *games.txt* tendríamos lo siguiente:

```
juegos=read.table("games.txt",header=TRUE,sep="\t")
```

Ahora `juegos` ya es una hoja de datos manejable como hemos descrito en los apartados anteriores.

Es importante, observar que la función `read.table()` no es siempre la más adecuada para abrir ficheros de datos. Por ejemplo, en el fichero *fallos.txt* se recoge el número de fallos detectados en el desarrollo de un nuevo software. Los datos corresponden a los fallos observados por 45 usuarios, que prueban el funcionamiento de dicho software, antes de su comercialización.

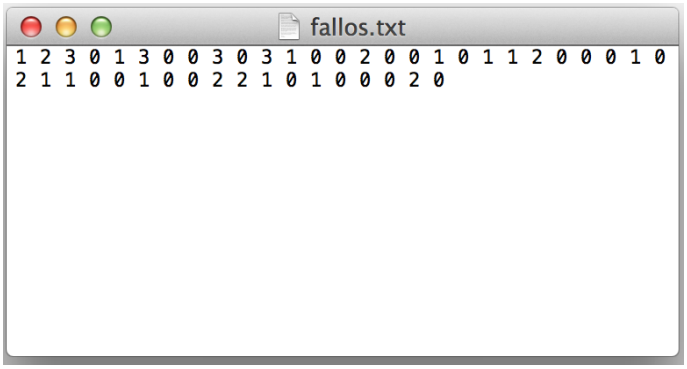


Figura 1.10: Archivo de texto. Variables separadas por tabulaciones y nombres de variables incluidos en la primera línea.

Si intentamos abrir este fichero *.txt* con `read.table`

```
fallos=read.table("fallos.txt")
fallos
```

el resultado no es el esperado, ya que en este caso, los datos no están ordenados en una tabla, con individuos en las filas y variables en las columnas. En realidad, los datos corresponden a una única variable, y podríamos definido así en el uso de `read.table`, sin embargo, es mucho más sencillo emplear el comando `scan()`:

```
fallos=scan("fallos.txt")
fallos
```

1.6.4. Exportar datos: Función write.table

Existe la posibilidad de exportar el conjunto de datos activo para que pueda ser leido por cualquier otro programa. El formato más sencillo en que podemos hacerlo mediante R es el formato de texto *.txt*.

La función `write.table` permite crear archivos de texto que contienen hojas de datos de R. La sintaxis de dicha función, con las opciones más habituales, es la siguiente:

```
write.table(hoja,file="fichero.txt",sep="\t",na="NA",dec=".",row.names=TRUE,
col.names=TRUE)
```

Vamos a comentar los detalles de cada argumento:

- `hoja` se refiere al nombre de la hoja de datos que queremos exportar.
- `fichero.txt` será el nombre del fichero donde queremos exportar los datos.
- `sep="\t"` quiere decir que los datos estarán separados por una tabulación. También podemos poner una coma, un espacio, etc.
- `na="NA"` se refiere a la forma en que se guardarán los dato faltantes. Si queremos que los deje en blanco, pondremos `na=""`.
- `dec="."` indica el carácter con el que se separan los decimales.
- `row.names` indicará si queremos que incluya en el fichero los nombres de las filas.
- `col.names` indicará si queremos que se incluyan los nombres de las variables.

Por ejemplo, si queremos exportar la hoja de datos `tiempos` que habíamos creado, en un fichero llamado `ejemplo_tiempos.txt`, con los datos separados por comas y con los nombres de las variables. El código sería:

```
write.table(tiempos,file="ejemplo_tiempos.txt",sep="\t",row.names=FALSE,
col.names=TRUE)
```

1.6.5. Filtrar datos

En ocasiones es necesario analizar, no todo el conjunto de datos, sino sólo un subconjunto de éste. En ese caso, lo que se hace es filtrar los datos mediante alguna condición dada por uno o varios valores de alguna variable.

Por ejemplo, supongamos que en el archivo de datos contenido en *games.txt* deseamos analizar sólo los datos correspondientes a WII.

Recordamos que la forma de referirnos a los elementos de una variable que conocemos hasta ahora es poniendo la posición que ocupan entre corchetes. Ahora vamos a hacer algo parecido, pero escribiendo entre los corchetes la condición que determina el filtro. En este caso, deseamos

quedarnos sólo con las filas que satisfacen `Plataforma=="WII"`, y todas las columnas (variables) de la hoja de datos. Entonces, utilizando la variable `Plataforma`, podemos hacerlo de la siguiente forma:

```
juegos.wii=juegos[juegos$Plataforma=="WII",]
```

Observamos que al no poner nada tras la coma que hay dentro del corchete estamos pidiendo a R que mantenga todas las variables.

1.6.6. Almacenamiento de instrucciones y resultados: El script y la sesión de trabajo

Ya hemos comentado que la forma más eficiente de trabajar con R es mediante un script del editor, en el que debemos ir introduciendo todos los pasos de nuestro trabajo. Sin embargo, puede que hasta ahora no hayamos tenido un ejemplo que ponga de manifiesto cómo esto, en efecto, facilita nuestro trabajo.

Por ejemplo, el *script* que aparece en la Figura 1.11 recoge todo el trabajo que hemos realizado en esta sección.

Observamos que aparecen algunas líneas antecedidas del símbolo `#`, para que R las imprima en la consola, pero no las ejecute. Podemos observar en la parte superior de la ventana del *script* que lo hemos guardado llamándolo `scrip_intro.R`, donde .R (o .r), es la extensión propia de los script de R. De esta forma, en el futuro podriamos seguir trabajando con este ejemplo sin necesidad de empezar todo de nuevo.

También podríamos estar interesados en guardar los objetos que se van generando en dichos análisis, por ejemplo, los conjuntos de datos, o cualquier resultado que obtengamos y que pueda ser almacenado poniéndole un nombre. Todos esos objetos que se van generando constituyen lo que en R se conoce como la sesión de trabajo. Por ejemplo, tras ejecutar el script que acabamos de comentar, los únicos objetos que hay en la sesión de trabajo son `"tiempos"`, `"juegos"` y `"juegos.wii"`.

Para guardar una sesión de trabajo (eso nos evitaría volver a tener que ejecutar el script) y almacenar estos objetos, simplemente seleccionamos el menú *Archivo* de R y la opción *Guardar área de trabajo*. Observamos que la extensión del fichero que genera es la de datos de R, *.RData*.

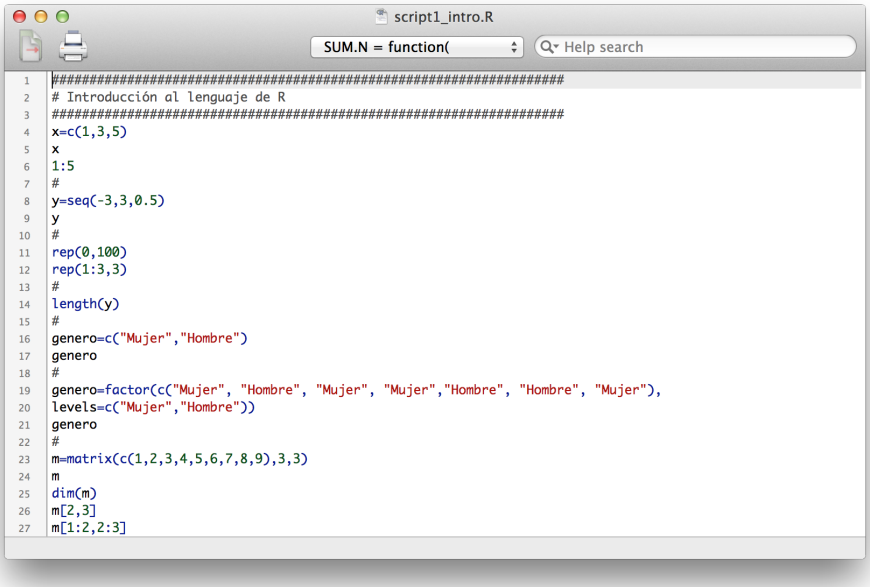


Figura 1.11: Ejemplo de script

Por último, si lo que deseamos es guardar los resultados que van apareciendo en la consola de R, tenemos que pinchar sobre la consola, seleccionamos *Archivo* y la opción *Guardar en archivo*. Esto generará un archivo de texto con todas las salidas.

.

Capítulo 2

Estadística descriptiva

2.1. Introducción

En los siguientes ejemplos, utilizaremos los conjuntos de datos almacenados en los ficheros: *games.txt*, *fallos.txt* y *datostiempos.txt*.

Comenzamos leyendo los datos de estos tres ficheros, con el siguiente código, ya conocido:

```
rm(list=ls(all=TRUE))
getwd()
setwd("D:/Estadistica/Practica")
dir()
juegos=read.table("games.txt", header = T, sep = "\t")
fallos=scan("fallos.txt")
tiempos=load("datos_tiempos.RData")
```

2.2. Distribuciones de frecuencias: La función table()

En los datos relativos a videojuegos, tenemos una serie de variables, la mayoría de tipo cualitativo. Para este tipo de variables, el resumen más conveniente es su distribución de frecuencias, que podemos obtener usando la función `table()`. Por ejemplo para la variable `Plataforma` escribimos

```
tabla.plat = table(juegos$Plataforma)
tabla.plat # frecuencias absolutas
prop.table(tabla.plat) # frecuencias relativas
```

Observamos que `fallos` es una variable cuantitativa discreta y que podemos obtener una tabla de frecuencias, exactamente como hicimos en el caso anterior, con la variable cualitativa `Plataforma`.

```
tabla.fallos = table(fallos)
tabla.fallos# frecuencias absolutas
prop.table(tabla.fallos)# frecuencias relativas
```

Sin embargo, podemos comprobar, que usar los mismos comandos para una variable cuantitativa continua, por ejemplo la variable `antes` de la hoja de datos `tiempos`, no tendría sentido,

```
tabla.antes = table(tiempos$antes)
tabla.antes
```

Debemos agrupar los datos previamente en intervalos. Por ejemplo, si queremos clasificar los datos de la variable `antes` en una tabla de 5 intervalos, podemos utilizar el comando `cut()`, de la siguiente forma:

```
attach(tiempos)
d=(max(antes)-min(antes))/5
limites = c(min(antes), min(antes)+d, min(antes)+2*d, min(antes)+3*d,
min(antes)+4*d, max(antes))
cortes = cut(antes,breaks=limites)
table(cortes)
```

2.3. Gráficos

Para plasmar en un gráfico la distribución de frecuencias de una variable cualitativa o cuantitativa discreta con pocos valores podemos usar diagramas de barras y diagramas de sectores.

2.3.1. Diagrama de barras: La función barplot()

Si estamos interesados en construir el diagrama de barras de una variable, por ejemplo, *Plataformas*, podemos usar la función `barplot()`,

```
barplot(tabla.plat, xlab="Plataformas", ylab="Frecuencias Absolutas",
main = "Diagrama de Barras")
```

Observamos que además podemos controlar cosas, como los títulos de los ejes, del gráfico, etc.

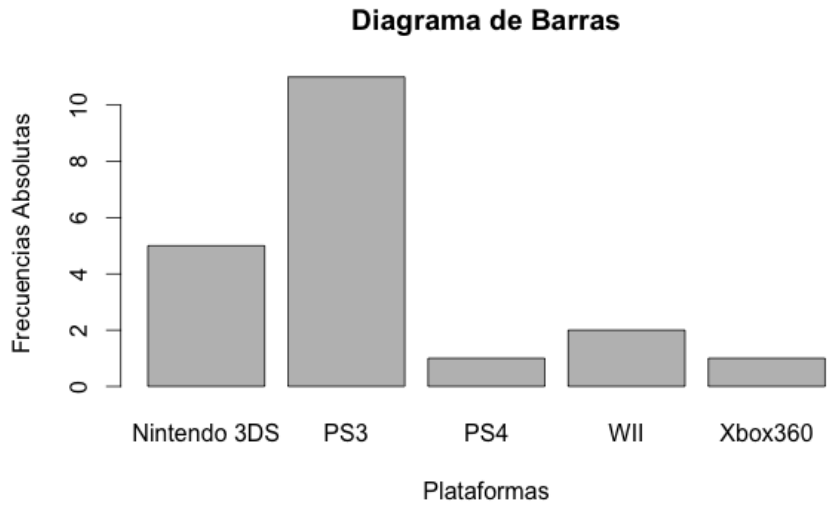


Figura 2.1: Diagrama de barras de la variable *Plataforma* de los datos *games*.

2.3.2. Diagrama de sectores: La función pie()

Su versión más básica es

```
pie(tabla.plat,main="Diagrama de Sectores de la variable Plataforma")
```

El resultado se muestra en la Figura 2.2.

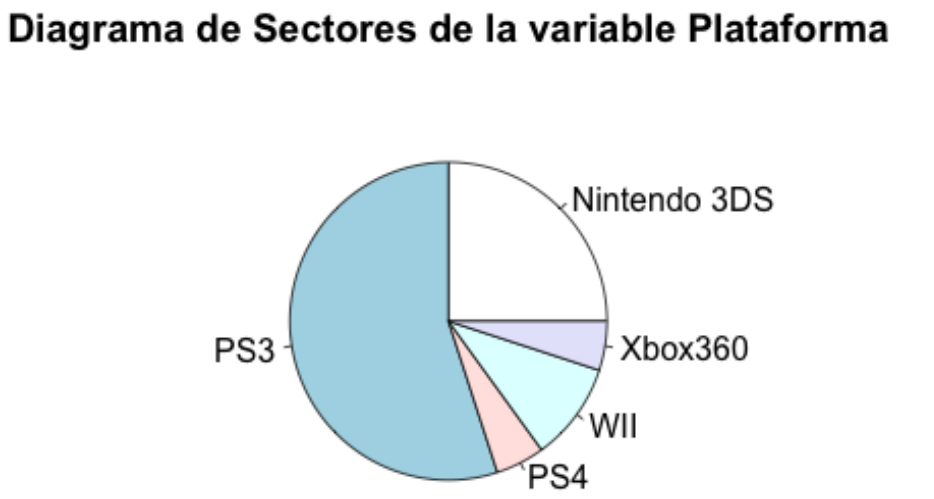


Figura 2.2: Diagrama de sectores de la variable *Plataforma* de los datos *games*

2.3.3. Histogramas: La función hist()

Como ya sabemos, los diagramas de barras o sectores no son adecuados para datos continuos. Frente a estas representaciones, el histograma aparece como la alternativa válida, ya que obliga a agrupar los valores en intervalos cuya frecuencia sí es relevante.

La función `hist()` nos permite representar el histograma y su sintaxis básica es la siguiente:

```
hist(x, breaks = "Sturges", freq = NULL, main = paste("Histogram of" , xname),
labels = FALSE)
```

donde

- `x` es el vector de datos.
- `breaks` puede especificar el número de intervalos que deseamos o los extremos de los intervalos que deseamos considerar, mediante un vector. Por defecto, asigna el número de intervalos por el método de Sturges.
- `freq` especifica si la escala del histograma es tal que el área de las barras es igual a la proporción de datos en cada intervalo (escala de densidad, con valor `freq = FALSE`) o su altura es simplemente el recuento de las frecuencias (escala de frecuencias, con valor `freq = TRUE`).

- `main` especifica el título del gráfico, mientras que xlab e ylab especifican las etiquetas de los ejes.
- `labels` añade una etiqueta a cada barra con el valor de las frecuencias.

Por ejemplo, para calcular los histogramas de frecuencias de las variables *antes* y *despues*, podemos escribir

```
par(mfrow=c(2,2))
hist(antes, freq=TRUE)
hist(despues, freq=TRUE)
```

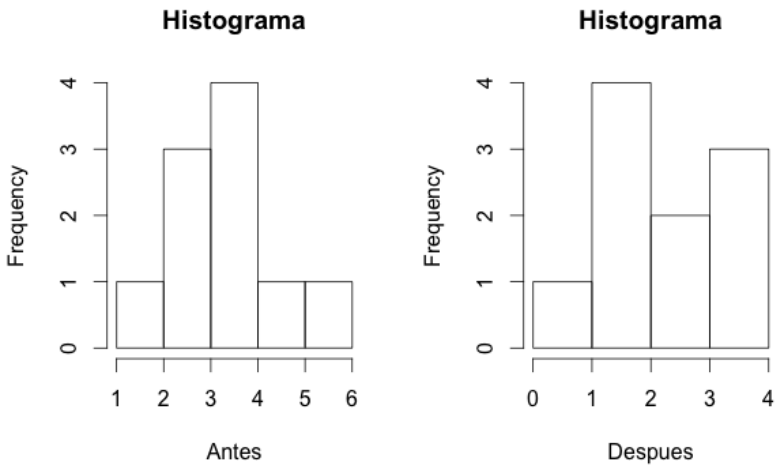


Figura 2.3: Histogramas de las variables *antes* y *despues*.

2.3.4. Diagrama de cajas: La función boxplot()

Este tipo de gráfico es válido para cualquier conjunto de datos, independientemente de la forma de su distribución de frecuencias.

La sintaxis básica de la función `boxplot()` obliga simplemente a especificar el conjunto de datos. Por ejemplo, usando

```
boxplot(fallos)
```

obtenemos el boxplot del conjunto de datos `fallos` (ver figura). También es posible añadir un título al gráfico y a los ejes, como en el caso de `hist()`.

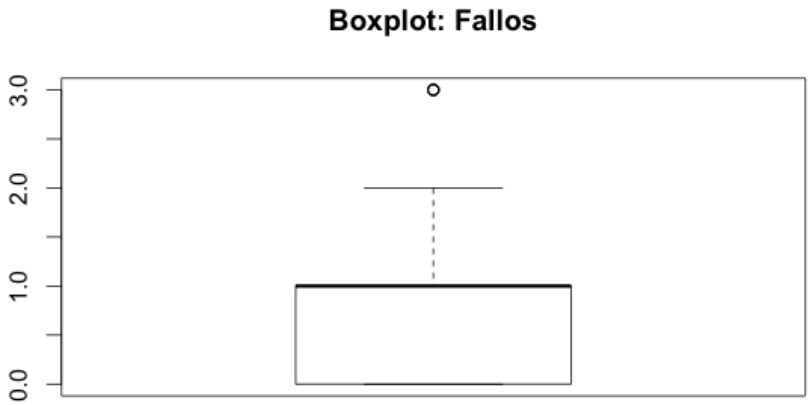


Figura 2.4: Boxplot de la variable *fallos*.

Observamos además que este tipo de gráfico es útil cuando queremos comparar valores de dos o más variables. Así, el código

```
boxplot(tiempos, main= "Boxplot de las Variables antes y despues")
```

nos permite representar en el mismo gráfico los diagramas de cajas de las dos variables relativas a los tiempos de acceso a la página web, y así poder compararlas (ver Figura 2.5).

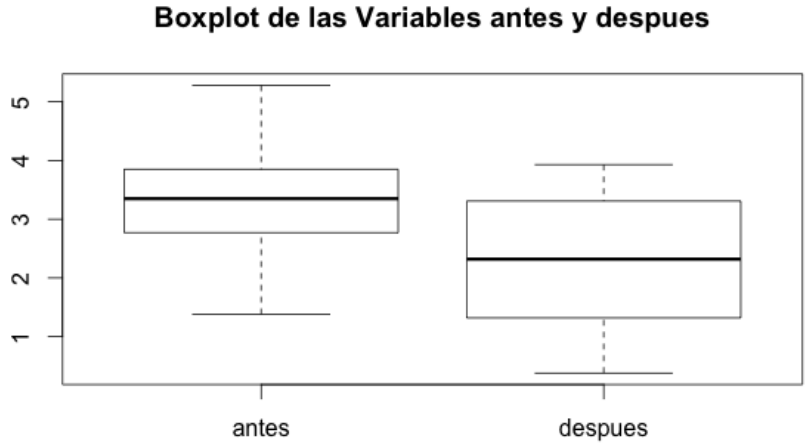


Figura 2.5: Boxplots de las variables *antes* y *despues*.

2.4. Medidas descriptivas

Las funciones `mean()`, `sd()` y `quantile()` proporcionan la media, la desviación típica y los cuantiles de cualquier muestra. Todas estas órdenes responden al mismo tipo de formato. Por ejemplo, si queremos calcular la media de las variables `antes` y `despues` escribimos

```
attach(tiempos)
mean(antes)
mean(despues)
```

De igual forma, la desviación típica se lograría mediante

```
sd(antes)
sd(despues)
```

Finalmente, para obtener los cuantiles necesitamos especificar las variables y las probabilidades de los cuantiles que deseamos mediante el argumento `probs`. Por ejemplo, para obtener los percentiles 5 y 95,

```
quantile(antes,probs=c(0.05,0.95))
quantile(despues,probs=c(0.05,0.95))
```

También puede resultarnos útil la función `summary()` que proporciona algunas de las medidas anteriores.

```
summary(antes)
summary(despues)
```

o simplemente

```
summary(tiempos)
```

2.5. Descripción de datos bivariantes

Consideramos ahora brevemente el caso en el que queremos describir la relación existente entre dos variables.

Cuando las dos variables a considerar son categóricas, o discretas, es habitual resumir los datos mediante tablas de frecuencia de doble entrada, absolutas o relativas. Para obtener estas tablas en R, podemos usar el comando `table()` que ya conocemos.

```
tabla.plat.edad = table(juegos$Plataforma, juegos$Edad)
tabla.plat.edad # frecuencias absolutas
prop.table(tabla.plat.edad) # frecuencias relativas
```

Las frecuencias de cada una de las variables, por separado, (frecuencias marginales) pueden calcularse sumando las filas o las columnas de la tabla conjunta y observamos que en R, pueden obtenerse usando el comando `addmargins()`.

```
addmargins(tabla.plat.edad)
addmargins(prop.table(tabla.plat.edad))
```

Para representar gráficamente estas tablas de frecuencias conjuntas, podemos utilizar diagramas de barras, usando el comando `barplot()`.

```
barplot(tabla.plat.edad, beside = T, xlab="Edad", legend=T)
barplot(t(tabla.plat.edad), beside = T, xlab="Plataformas", legend=T)
```

2.6. Regresión lineal. La función plot() y la función lm().

Tanto en el caso de dos variables (regresión simple) como en el de más de dos variables (regresión múltiple), el análisis de regresión lineal puede utilizarse para explorar y cuantificar la relación entre una variable llamada dependiente o criterio (Y) y una o más variables llamadas independientes o predictoras ($X_1, X_2, \ldots, X_k$). En esta sección, veremos únicamente el análisis de regresión lineal simple.

Para ilustrar los conceptos sobre regresión lineal, vamos a analizar la relación entre las variables *peso* y *altura* del fichero de datos `peso_altura.dat`.

2.6.1. Preliminares

Antes de abordar el análisis de regresión propiamente dicho, es conveniente hacer una breve descripción de los datos, para tener una idea de las características de éstos. Una vez que hemos importado el conjunto de datos de interés, podemos obtener algunas medidas descriptivas, usando por ejemplo

```
rm(list=ls(all=TRUE))
ls()
load("peso_altura.Rdata")
ls()
summary(pesoalt)
```

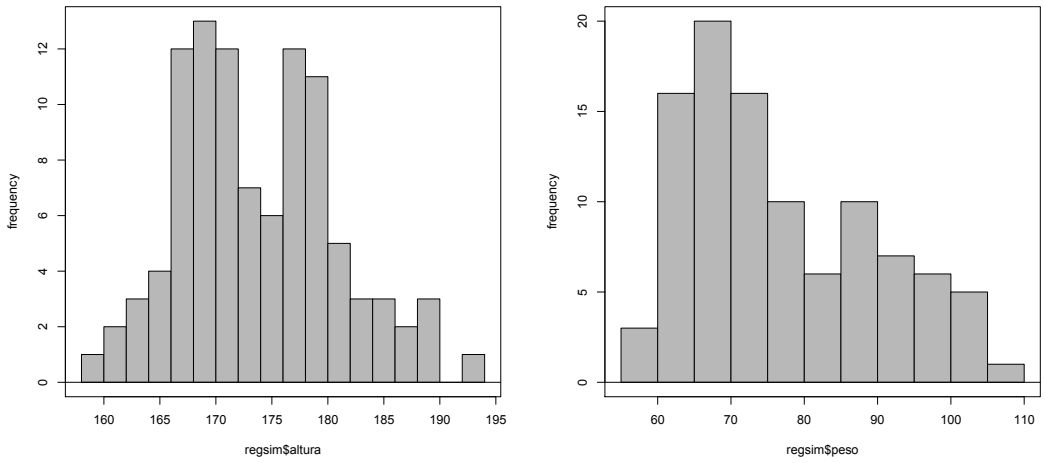
obteniendo

```
 sexo       altura           peso
 H:54   Min.   :159.0   Min.   : 59.00
 M:46   1st Qu.:169.0   1st Qu.: 68.00
        Median :173.5   Median : 73.50
        Mean   :174.3   Mean   : 77.37
        3rd Qu.:179.0   3rd Qu.: 88.00
        Max.   :194.0   Max.   :109.00
```

Observamos que además de las variables numéricas *altura* y *peso*, la hoja de datos contiene también la variable cualitativa *sexo*. Podemos además realizar gráficos de nuestros datos, con el fin de apreciar la forma de éstos, por ejemplo, la Figura 2.6 muestra los histogramas de las dos variables numéricas

```
par(mfrow=c(1,2))
hist(pesoalt$peso)
hist(pesoalt$altura)
```

Una primera visión de los histogramas permite detectar una bimodalidad tanto en la variable peso como en la altura, lo que típicamente constituye un indicio de mezcla de poblaciones.



(a) (b)

Figura 2.6: Histogramas de las variables altura y peso.

2.6.2. Diagrama de dispersión. La función plot()

Es muy conveniente realizar un diagrama de dispersión de las dos variables de interés, para hacernos una primera idea de la relación existente entre ambas, para ello, usamos la función `plot()`. Su sintaxis básica es la siguiente:

```
plot(x,y,type="l",main="Título",sub="Subtítulo",xlab="Eje X",ylab=Eje Y")
```

En esta expresión,

- `x` se refiere a las coordenadas en el eje de abscisas de los puntos que queremos representar, expresadas como un vector.
- `y` son las coordenadas en el eje de ordenadas.
- `type` especifica el tipo de gráfico que queremos. Las opciones más habituales son `"p"` si queremos simplemente puntos, `"l"` si queremos que una los puntos con una línea o `"b"`, si queremos que haga ambas cosas.
- `main` es el título del gráfico.
- `sub` es el subtítulo.

- `xlab` especifica el título del eje X.
- `ylab` especifica el título del eje Y.

Por ejemplo, usando,

```
par(mfrow=c(1,1))
plot(altura, peso)
```

el resultado permite intuir una posible relación lineal entre las dos variables (ver Figura 2.7).

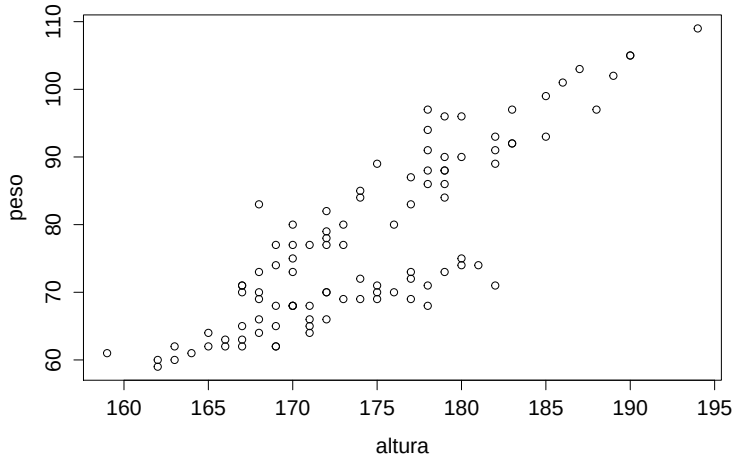


Figura 2.7: Diagrama de dispersión de las variables *altura* y *peso*.

Si se observa atentamente el diagrama de dispersión se puede entrever la existencia de dos poblaciones. En efecto, se están considerando conjuntamente los dos sexos, hombre y mujer, cuando los patrones de relación peso-altura no tienen porqué coincidir y de hecho no lo hacen. Para confirmarlo se representará el diagrama de dispersión pero diferenciando los individuos de ambos sexos.

```
pesoaltH=pesoalt[sexo=="H",]
pesoaltM=pesoalt[sexo=="M",]

plot(pesoaltH$altura, pesoaltH$peso, xlab="altura", ylab="peso")
points(pesoaltM$altura, pesoaltM$peso, col="red")
```

La figura 2.8 muestra los dos diagramas de dispersión para la *altura* y el *peso*, distinguiendo entre hombres y mujeres, respectivamente. Observamos que las dos rectas de ajuste se acomodan mucho mejor a sus respectivos grupos.

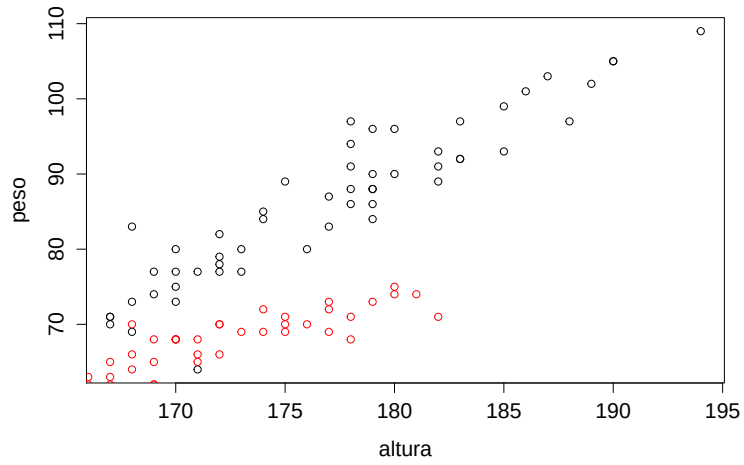


Figura 2.8: Diagrama de dispersión para las variables *altura* y *peso*, distinguiendo por *sexo*.

Podemos además obtener resúmenes numéricos de las variables peso y altura distinguiendo entre hombres y mujeres, usando por ejemplo:

```
summary(pesoaltH)
summary(pesoaltM)
```

Para confirmar cuantitativamente la existencia de una alta correlación podemos además calcular y contrastar el coeficiente de correlación lineal,

```
cor(pesoalt$altura, pesoalt$peso)
```

vemos que es positivo y relativamente alto, $r = 0.848$, lo que indica que existe relación directa entre las variables.

Calculamos de nuevo el coeficiente de correlación lineal para los nuevos conjuntos de datos,

```
cor(pesoaltH$altura, pesoaltH$peso)
cor(pesoaltM$altura, pesoaltM$peso)
```

que resulta 0.897 para las mujeres y 0.928 para los hombres, correlaciones más altas que las que se tenían para el ajuste conjunto.

2.6.3. Ajuste de la recta de regresión

A la vista de los resultados, optamos por realizar el análisis de regresión entre las variables *altura* y *peso* distinguiendo entre hombres y mujeres. En particular, realizaremos el análisis correpondiente al caso de los hombres, utilizando la función `lm()`, cuya sintaxis básica de la función es la siguiente:

```
lm(formula, data, subset)
```

- `formula` es la expresión que define el modelo. Observamos que en este tipo de expresiones, debe aparecer la variable dependiente seguida del símbolo ~ y la variable independiente.
- `data` es una opción adicional que puede especificar la hoja de datos que queremos manejar.
- `subset` es también un parámetro opcional en el que podemos especificar si sólo queremos utilizar un subconjunto de casos para ajustar el modelo.

Así, en el caso de los hombres, tendríamos

```
reg.H = lm(peso ~ altura, pesoalt, pesoalt$sexo=="H")
summary(reg.H)
```

Los resultados que aparecen en la ventana son los siguientes:

```
Call:
lm(formula = peso ~ altura, data = pesoaltH)
Residuals:
Min 1Q Median 3Q Max
-13.578 -2.091 -0.491 2.213 9.662
Coefficients:
Estimate Std. Error t value Pr(> |t|)
(Intercept) -164.09760 13.89222 -11.81 2.43e-16 ***
altura 1.41331 0.07837 18.03 < 2e-16 ***
```

```
---
Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 1
Residual standard error: 3.937 on 52 degrees of freedom
Multiple R-Squared: 0.8621, Adjusted R-squared: 0.8595
F-statistic: 325.2 on 1 and 52 DF, p-value: < 2,2e - 16
```

Destacamos los siguientes resultados:

1. La estimación del valor del parámetro a (intercept) es -164.097. Hipotéticamente, se interpretaría como el peso estimado si la altura fuera 0, lo que, obviamente, no tiene sentido.

2. La estimación de la pendiente de la recta b es 1.413.

   La recta ajustada aparece, por tanto, especificada a través de sus dos coeficientes: el término independiente o intercept y la pendiente de la recta:

   $$peso = -164.09760 + 1.41331 \times altura$$

   Así pues, por cada centímetro que se incremente la altura de los hombres, se espera que el peso se incremente en 1.41 Kg.

3. El coeficiente R^2 , que aparece en la penúltima línea, tiene un valor de 0.862, lo que nos indica que el 86.2 % de toda la variabilidad en el *peso* de los hombres, puede ser explicado por la *altura*.

De la misma manera, podemos obtener el ajuste correspondiente a los datos relativos a mujeres,

```
reg.M = lm(peso ~ altura, pesoalt, pesoalt$sexo==?M?)
summary(reg.M)
```

Y representar ambos ajustes gráficamente, usando

```
plot(pesoaltH$altura, pesoaltH$peso, xlab="altura", ylab="peso", col="blue")
points(pesoaltM$altura, pesoaltM$peso, col="red")
lines(pesoaltH$altura,reg.H$fitted, col="blue")
lines(pesoaltM$altura,reg.M$fitted, col="red")
```

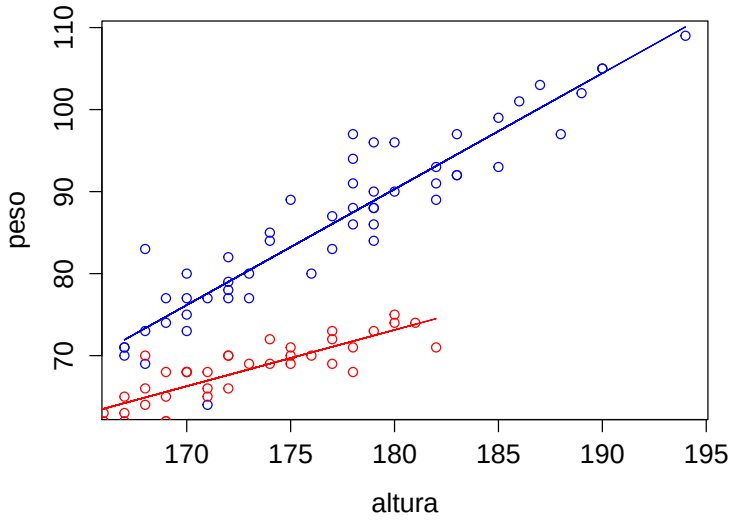


Figura 2.9: Diagrama de dispersión para las variables *altura* y *peso*, distinguiendo por *sexo* y rectas de regresión ajustadas.