



Estadística con R

Raquel Montes

Índice general

1. R	1
1.1. Introducción	1
1.2. Instalación de R	1
1.3. La consola y el editor de R	3
1.4. R-Studio	4
1.5. Introducción al lenguaje de R	6
1.5.1. Algunos tipos de objetos de R	6
1.5.2. Funciones más comunes en R	10
1.5.3. Operaciones lógicas	11
1.5.4. Definición de Funciones	11
1.5.5. La ayuda de R	12
1.6. Organización de datos en R	15
1.6.1. Introducir una hoja de datos	15
1.6.2. Almacenar datos: Las funciones save y load	17
1.6.3. Importar datos: La función read.table	19
1.6.4. Exportar datos: Función write.table	22
1.6.5. Filtrar datos	23

1.6.6. Almacenamiento de instrucciones y resultados: El script y la sesión de trabajo	24
2. Estadística descriptiva	27
2.1. Introducción	27
2.2. Distribuciones de frecuencias: La función table()	27
2.3. Gráficos	28
2.3.1. Diagrama de barras: La función barplot()	29
2.3.2. Diagrama de sectores: La función pie()	29
2.3.3. Histogramas: La función hist()	30
2.3.4. Diagrama de cajas: La función boxplot()	31
2.4. Medidas descriptivas	33
2.5. Descripción de datos bivariantes	34
2.6. Regresión lineal. La función plot() y la función lm().	34
2.6.1. Preliminares	35
2.6.2. Diagrama de dispersión. La función plot()	36
2.6.3. Ajuste de la recta de regresión	39
3. Variables Aleatorias	43
3.1. Introducción	43
3.2. Cálculo de probabilidades	45
3.2.1. Distribuciones discretas	45
3.2.2. Distribuciones continuas	45
3.3. Cálculo de cuantiles	46
3.3.1. Distribuciones discretas	47
3.3.2. Distribuciones continuas	47

3.4. Simulación de muestras	48
4. Inferencia Estadística	51
4.1. Estimación por Intervalos de confianza	51
4.1.1. De la media de una distribución normal con varianza desconocida	52
4.1.2. De la media de una distribución cualquiera, con muestras grandes	53
4.1.3. De una proporción	54
4.1.4. De la varianza de una distribución normal	54
4.2. Contrastes de hipótesis	55
4.2.1. Contrastes sobre medias: La función <code>t.test()</code>	55
4.2.2. Contrastes sobre proporciones: La función <code>prop.test()</code>	62
4.2.3. Contraste para la comparación de varianzas: La función <code>var.test()</code>	66

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.

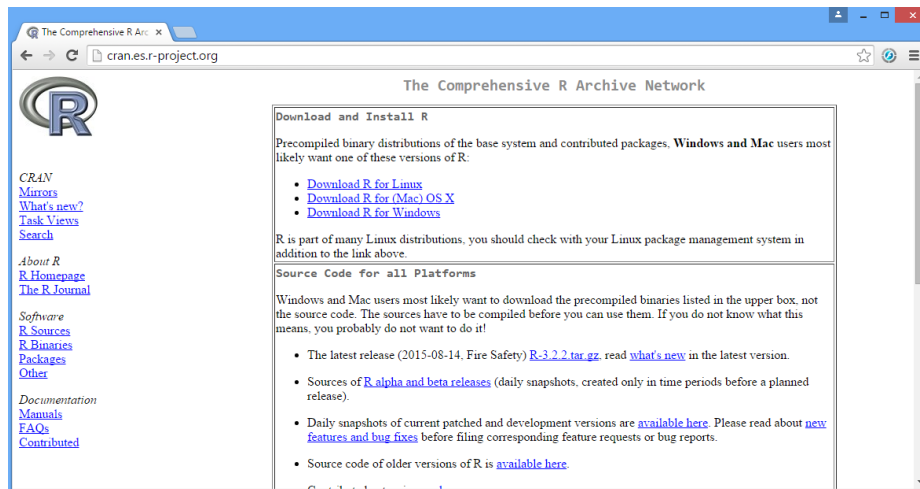


Figura 1.1: Elegimos el sistema operativo.

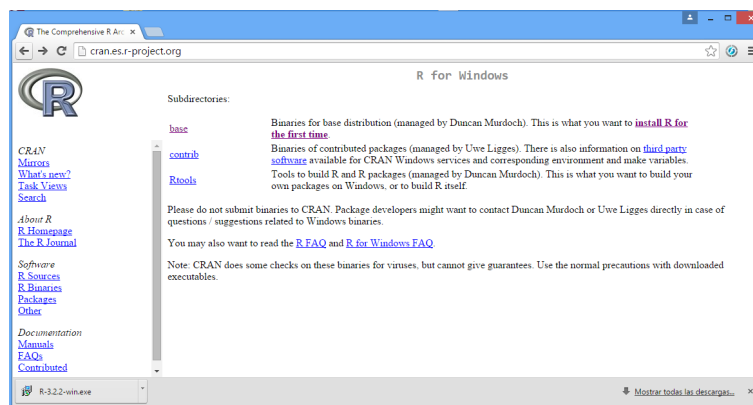


Figura 1.2: Pinchamos en *base*.

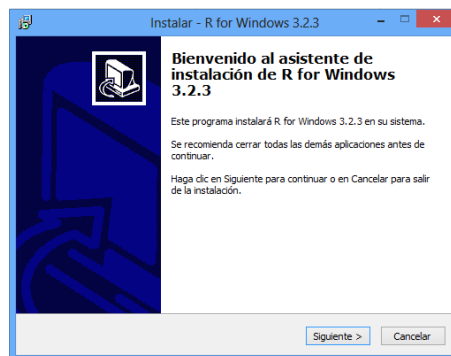


Figura 1.3: Abrimos el ejecutable.

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.

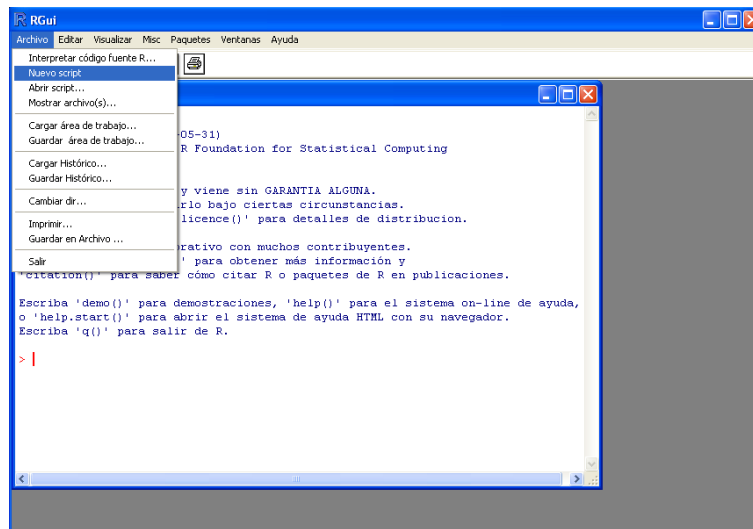


Figura 1.4: Consola de R y forma de acceder al editor

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 cual, el aspecto será el que aparece en la Figura 1.5.

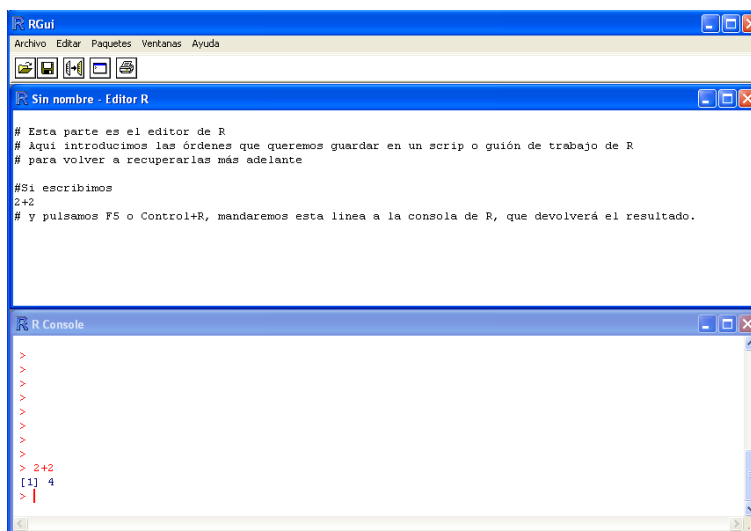


Figura 1.5: Consola y editor de R

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.

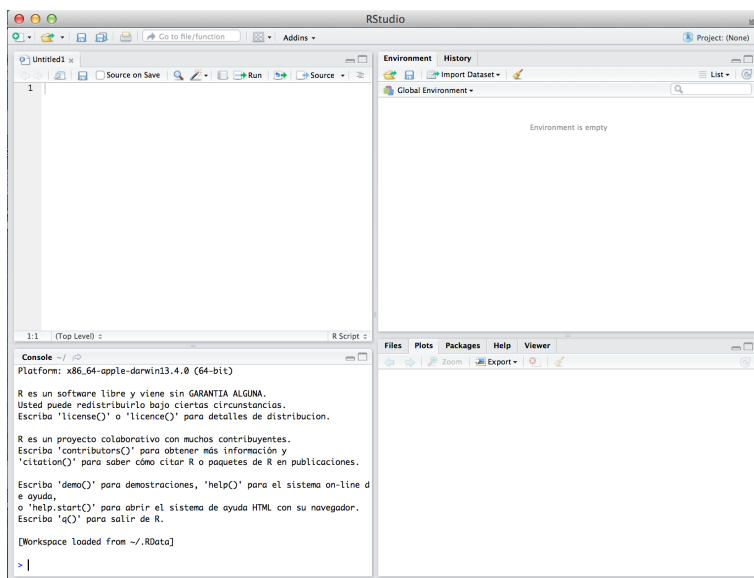


Figura 1.6: R-Studio

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
```

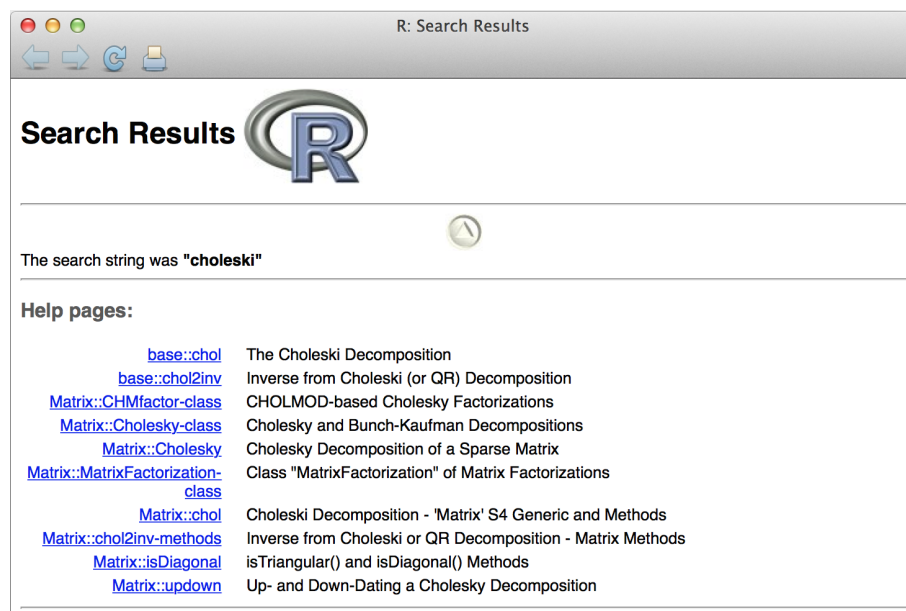


Figura 1.7: Información que da la ayuda de R a propósito de la entrada `choleski`

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.

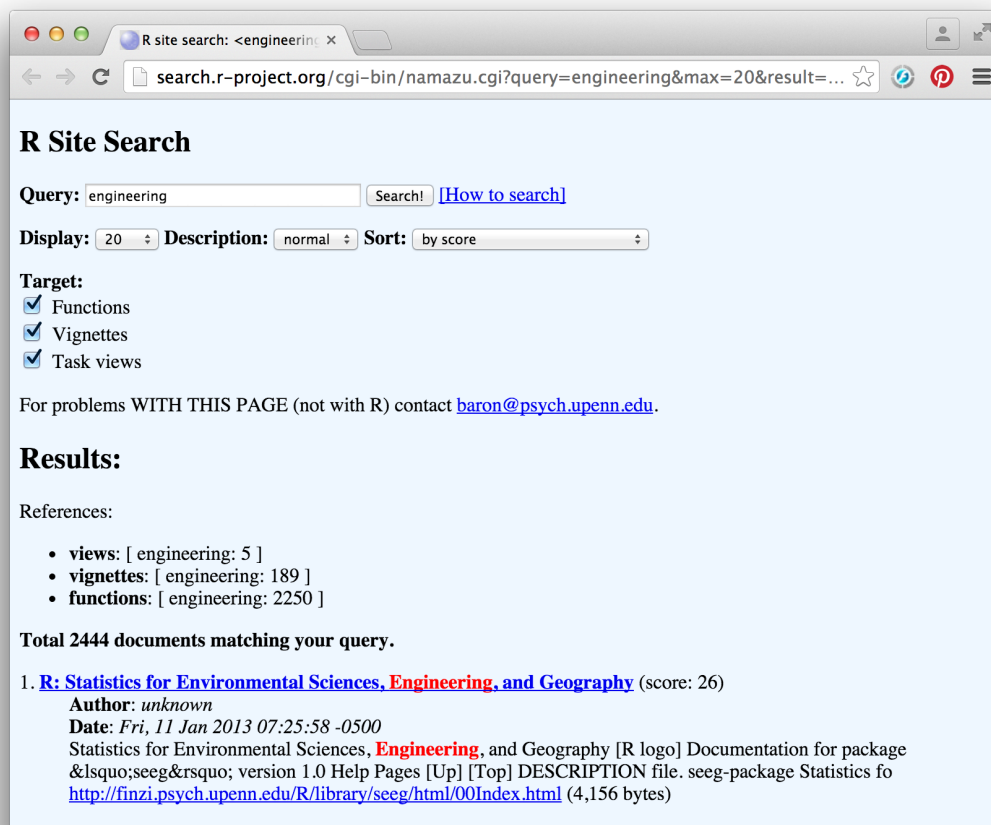


Figura 1.8: Información sobre funciones que, en algún lugar de la ayuda, incluyan la palabra *engineering*.

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 (1ª variable) del 5º internauta de la muestra, usamos

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

o también,

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

Supongamos que queremos los tiempos después de la mejora (2ª variable) de los 5º, 8º y 10º 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 3º al 8º 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 directorio 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.

Puesto	Título	Plataforma	Distribuidor	Género	PEGI
1	Grand Theft Auto V	PS3	Take 2	Action/Combat	18
2	Fifa 14 Move	PS3	Electr Arts	Sport	3
3	Call Of Duty Ghosts	PS3	Activision	Action/Combat	18
4	Last Dance 2014 WII	Wii	Ubi Soft	Dance	3
5	Animal Crossing New Leaf	3DS	Nintendo	Life Simulation	3
6	Grand Theft Auto V	Xbox360	Take 2	Action/Combat	18
7	The Last Of Us	PS3	Sony	Action/Combat	18
8	Last Dance 4	Wii	Ubi Soft	Dance	3
9	Call Of Duty: Black Ops II	PS3	Activision	Action/Combat	18
10	Pokemon Y	Nintendo 3DS	Nintendo	Graph-Adv/Rpg	3
11	Pokemon X	Nintendo 3DS	Nintendo	Graph-Adv/Rpg	3
12	Fifa 13 Move	PS3	Electr Arts	Sport	3
13	Far Cry 3	PS3	Ubi Soft	Action/Combat	18
14	New Super Mario Bros 2	Nintendo 3DS	Nintendo	Platform	3
15	Pro Evolution Soccer 2014	PS3	Konami	Sport	3
16	Assassins Creed Iv Black Flag	PS3	Ubi Soft	Action/Combat	18
17	God Of War Ascension	PS3	Sony	Action/Combat	18
18	Gran Turismo 6	PS3	Sony	Race/Rally	3
19	Fifa 14 PS4	PS4	Electr Arts	Sport	3
20	Uigi's Mansion 2	Nintendo 3DS	Nintendo	Platform	3

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 aparecen 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.

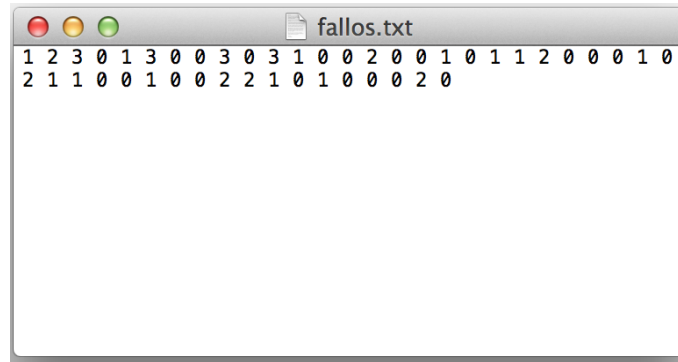


Figura 1.10: Archivo de texto. Variables separadas por tabulaciones y nombres de variables incluidos en la primera línea.

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 definirlo 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 leído 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

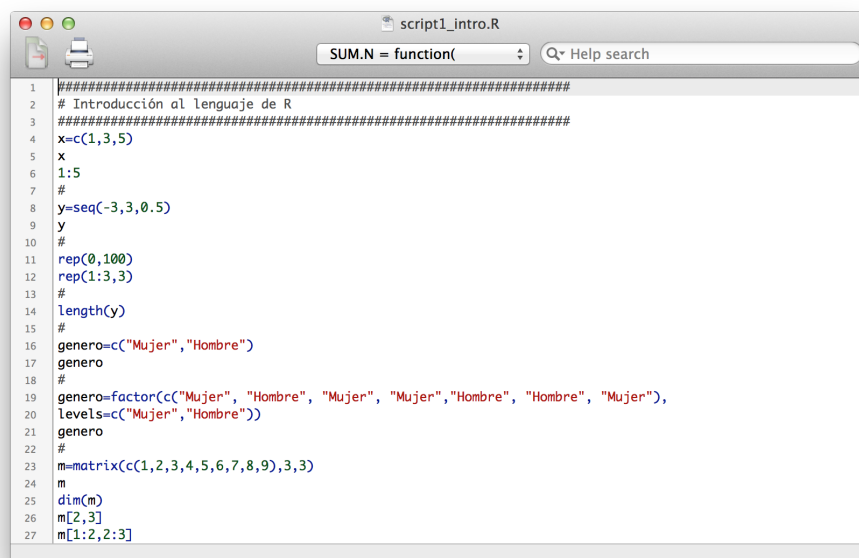
Ya hemos comentado que la forma más eficiente de trabajar con R es mediante un script del editor, en el que debemos ir introduciendo todos los pasos de nuestro trabajo. Sin embargo, puede que hasta ahora no hayamos tenido un ejemplo que ponga de manifiesto cómo esto, en efecto, facilita nuestro trabajo.

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 anteceditas 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 podríamos 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`.



```
1 #####
2 # Introducción al lenguaje de R
3 #####
4 x=c(1,3,5)
5 x
6 1:5
7 #
8 y=seq(-3,3,0.5)
9 y
10 #
11 rep(0,100)
12 rep(1:3,3)
13 #
14 length(y)
15 #
16 genero=c("Mujer","Hombre")
17 genero
18 #
19 genero=factor(c("Mujer", "Hombre", "Mujer", "Mujer", "Hombre", "Hombre", "Mujer"),
20 levels=c("Mujer", "Hombre"))
21 genero
22 #
23 m=matrix(c(1,2,3,4,5,6,7,8,9),3,3)
24 m
25 dim(m)
26 m[2,3]
27 m[1:2,2:3]
```

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.

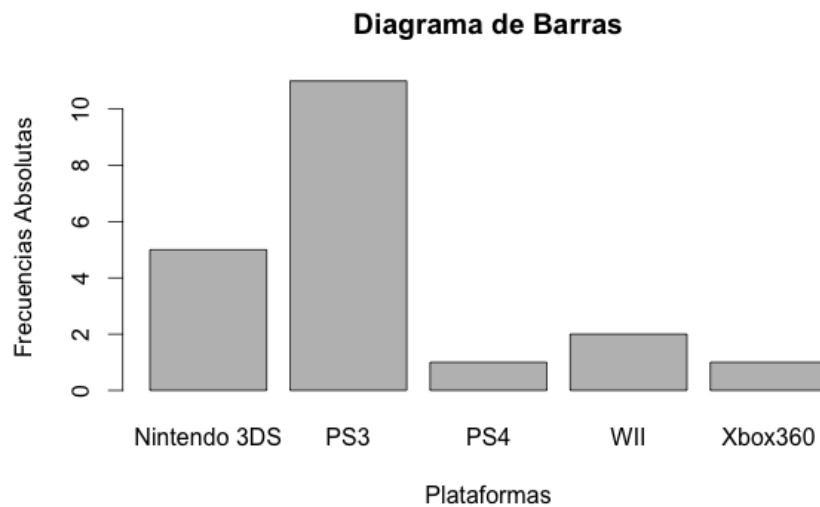


Figura 2.1: Diagrama de barras de la variable *Plataforma* de los datos *games*.

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.

Diagrama de Sectores de la variable Plataforma

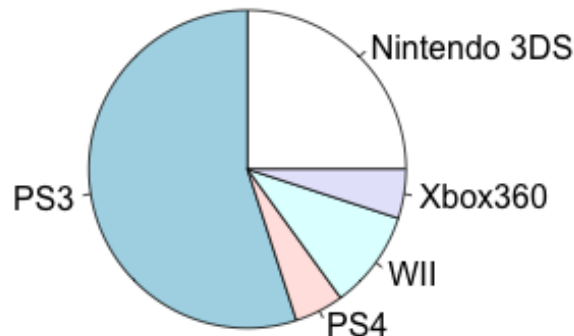


Figura 2.2: Diagrama de sectores de la variable *Plataforma* de los datos *games*

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)
```

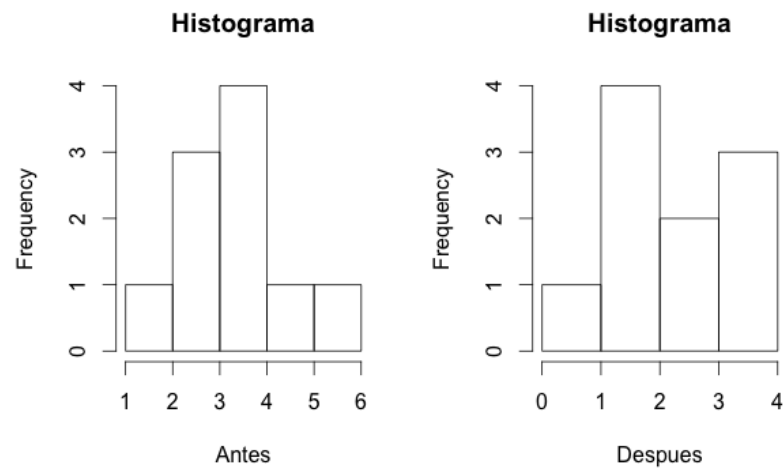


Figura 2.3: Histogramas de las variables *antes* y *despues*.

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()`.

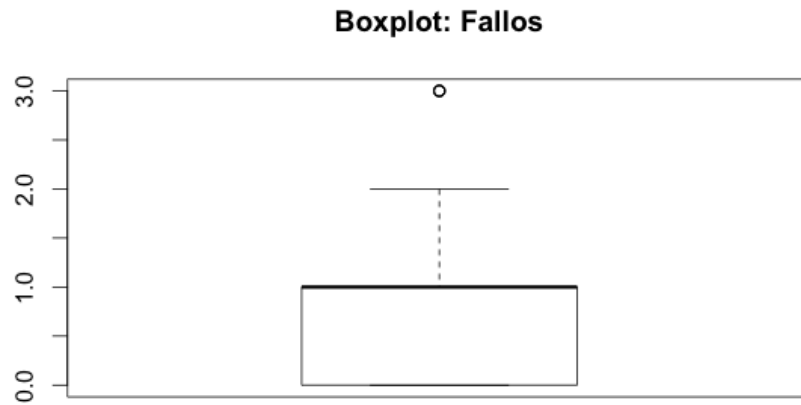


Figura 2.4: Boxplot de la variable *fallos*.

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).

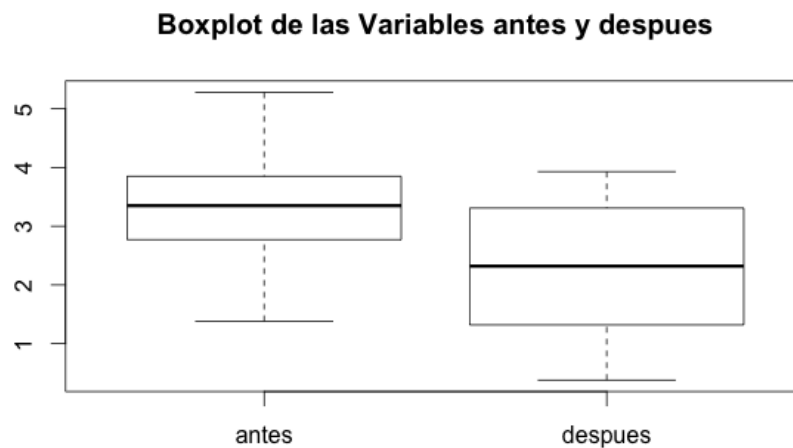


Figura 2.5: Boxplots de las variables *antes* y *despues*.

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, \dots, 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.

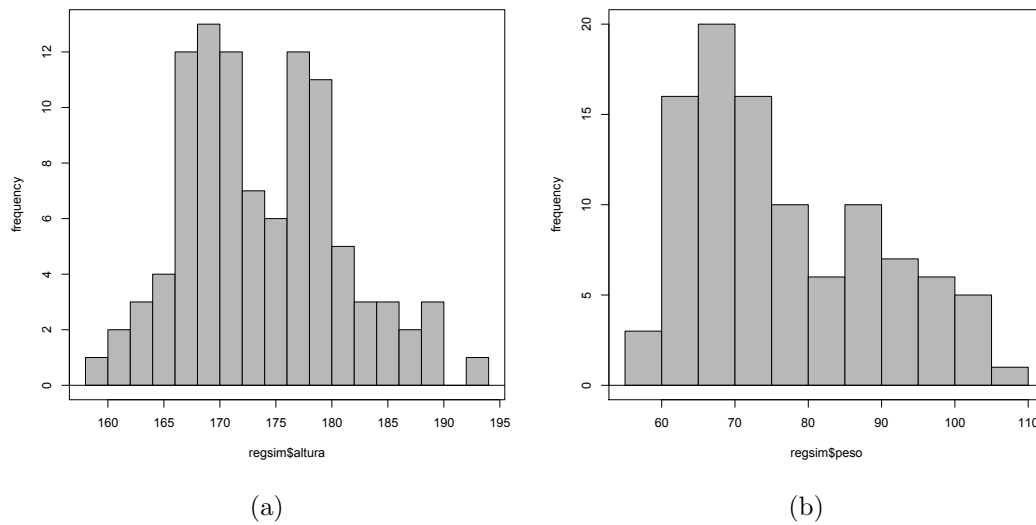


Figura 2.6: Histogramas de las variables altura y peso.

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).

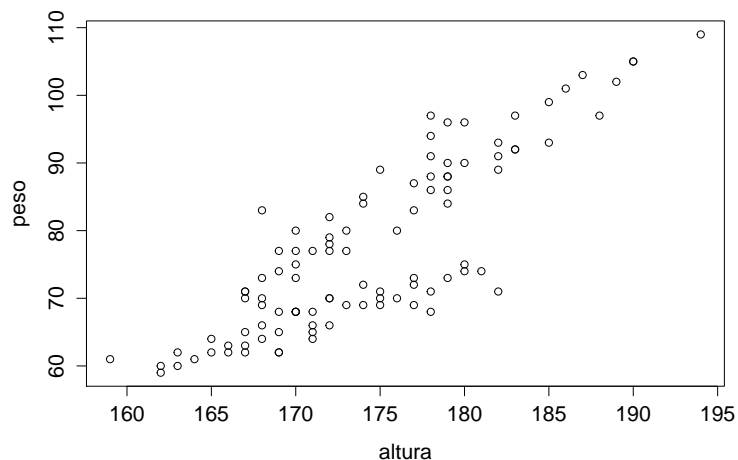


Figura 2.7: Diagrama de dispersión de las variables *altura* y *peso*.

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.

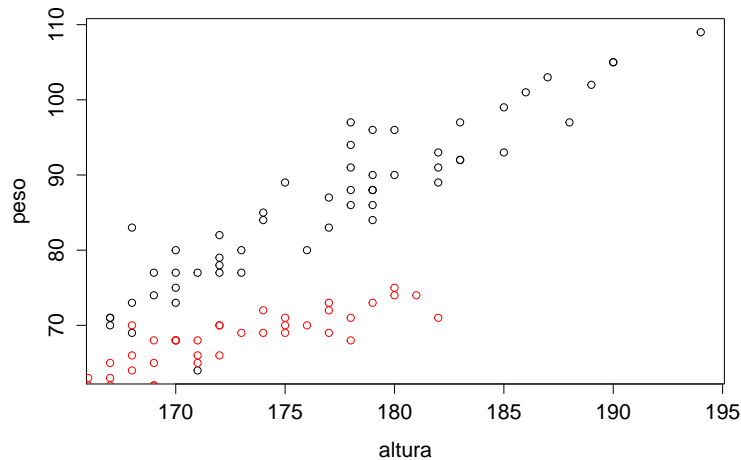


Figura 2.8: Diagrama de dispersión para las variables *altura* y *peso*, distinguiendo por *sexo*.

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 correspondiente 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")
```

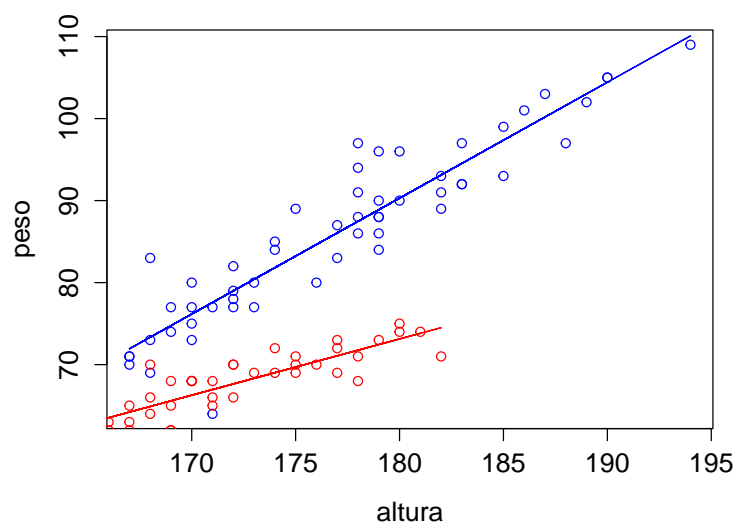


Figura 2.9: Diagrama de dispersión para las variables *altura* y *peso*, distinguiendo por *sexo* y rectas de regresión ajustadas.

.

Capítulo 3

Variables Aleatorias

3.1. Introducción

R y cualquier otro programa de software estadístico proporcionan una excelente manera de realizar cálculos asociados a las distribuciones de probabilidad más comunes.

Recordamos que según la naturaleza de la variable aleatoria pueden considerarse distribuciones de probabilidad discretas o continuas. Las principales distribuciones de probabilidad de variables discretas son: Bernoulli, Binomial, Geométrica, y de Poisson. Entre los modelos de variable continua destacan las distribuciones: Uniforme, Exponencial y Normal. Todas estas distribuciones y muchas más, están recogidas en R.

Incluso los modelos más sencillos, como el Binomial o el de Poisson, presentan dificultades en cuanto a lo tedioso que resulta realizar los cálculos. En otras distribuciones, como la Normal, el problema es que es imposible trabajar analíticamente, siendo absolutamente necesario la utilización de algún software matemático para obtener aproximaciones precisas de las probabilidades.

Utilizaremos 4 tipos de funciones:

1. Las funciones que R llama densidades, pero que en realidad son densidades (si la distribución es discreta) o funciones masa (si la distribución es continua). Todas estas funciones empiezan por la letra **d**.
2. Las funciones de distribución. Todas ellas empiezan por la letra **p**.

3. Las funciones cuantil, que empiezan por la letra **q**.
4. Las funciones de simulación de datos, que empiezan por la letra **r**.

Lo que sigue a esas letras que son el comienzo de cada función es la identificación de la distribución, por ejemplo

- **binom** para la binomial.
- **geom** para la geométrica.
- **pois** para la Poisson.
- **exp** para la exponencial.
- **norm** para la normal.
- etc.

Finalmente, los argumentos de cada una de las funciones tendrán que especificar, en cada caso, los parámetros concretos de la distribución, y determinar qué probabilidad (**prob**) o qué cuantil queremos, o cuántos datos simulados necesitamos (**m**). La siguiente tabla muestra algunos ejemplos de estas funciones, especificando los argumentos de cada función.

	F. Densidad	F. Distribución	F. Cuantil	Muestras aleatorias
Binom(n, p)	dbinom(x, n, p)	pbinom(x, n, p)	qbinom($prob, n, p$)	rbinom(m, n, p)
Geom(p)	dgeom(x, p)	pgeom(x, p)	qgeom($prob, p$)	rgeom(m, p)
Pois(λ)	dpois(x, λ)	ppois(x, λ)	qpois($prob, \lambda$)	rpois(m, λ)
$U(a, b)$	dunif(x, a, b)	punif(x, a, b)	qunif($prob, a, b$)	runif(m, a, b)
Beta(α, β)	dbeta(x, α, β)	pbeta(x, α, β)	qbeta($prob, \alpha, \beta$)	rbeta(m, α, β)
Exp(λ)	dexp(x, λ)	pexp(x, λ)	qexp($prob, \lambda$)	rexp(m, λ)
$N(\mu, \sigma)$	dnorm(x, μ, σ)	pnorm(x, μ, σ)	qnorm($prob, \mu, \sigma$)	rnorm(m, μ, σ)

Tabla 3.1: Resumen de las funciones asociadas a las distribuciones y sus argumentos

3.2. Cálculo de probabilidades

3.2.1. Distribuciones discretas

En el caso de las distribuciones discretas, estaremos, básicamente, interesados en el cálculo de dos tipos de probabilidades:

1. Las probabilidades que proporciona la función masa, que podríamos llamar probabilidades simples, del tipo $P(X = x)$.
2. Probabilidades acumuladas (dadas en términos de la función de distribución), del tipo $P(X \leq x)$.

Es evidente que las probabilidades acumuladas se pueden calcular a partir de las probabilidades simples, sin más que tener en cuenta que $P(X \leq x) = \sum_{x_i \leq x} P(X = x_i)$.

Por ejemplo, supongamos que tenemos una distribución binomial de parámetros $n = 10$ y $p = 0.25$ y deseamos calcular $P(X < 4)$. Entonces, dado que

$$P(X < 4) = P(X = 0, 1, 2, 3) = \sum_{x=0}^3 P(X = x),$$

sólo tenemos que calcular la probabilidad de 0, 1, 2 y 3 y sumarlas. Para ello podríamos ejecutar cualquiera de las ordenes:

```
sum(dbinom(0:3,10,0.25))  
pbinom(3,10,0.25)
```

3.2.2. Distribuciones continuas

En el caso de las distribuciones de tipo continuo sabemos que los valores concretos de la variable tienen probabilidad cero o, dicho de otra forma, no tienen masa de probabilidad, sino densidad de probabilidad. En estas variables no tiene sentido preguntarse por probabilidades del tipo $P(X = x)$ porque todas son cero. En su lugar, lo que nos preguntamos es por las probabilidades

de que las variables proporcionen valores en intervalos, es decir, probabilidades del tipo $P(a < X < b)$, y donde las desigualdades pueden ser estrictas o no, ya que el resultado final no varía.

Recordamos además que las probabilidades del tipo $P(a < X < b)$ se calculan como

$$P(a < X < b) = \int_a^b f(x)dx = F(b) - F(a),$$

donde $f(x)$ es la función de densidad de la variable y $F(x)$ es la función de distribución. En resumen, podremos calcular probabilidades del tipo $P(a < X < b)$ siempre que podamos obtener los valores de la función de distribución $F(x)$. Y recordemos que estas funciones de distribución en R son las que empiezan por la letra **p**.

Así, si tuviéramos una distribución normal de media 5 y desviación típica 2, y quisiéramos calcular $P(2 < X < 7.6)$ lo haríamos teniendo en cuenta que

$$P(2 < X < 7.6) = F(7.6) - F(2)$$

mediante

`pnorm(7.6,5,2)-pnorm(2,5,2)`

3.3. Cálculo de cuantiles

Recordemos que los cuantiles son medidas de posición relativas. El cuantil p , (siendo p un n° entre 0 y 1), Q_p , de una distribución de probabilidad es aquél que deja por debajo de sí una probabilidad p . Si, hipotéticamente, tuviéramos 100 datos, el cuantil p es el que, ordenados todos los valores de menor a mayor, ocuparía la posición $100p$. Eso es lo que permite interpretarlos como medidas de posición relativas, ya que permite analizar si un dato es *alto*, *medio* o *bajo* en su distribución.

Desde el punto de vista del cálculo, observemos que en la sección anterior hemos aprendido a calcular los valores $p = P(X \leq x)$. Lo que ahora tenemos que hacer es justo lo contrario, es calcular los valores x tales que $P(X \leq x) = p$.

3.3.1. Distribuciones discretas

A la hora de hablar de cuantiles en distribuciones discretas hay que recordar que es posible que no podamos encontrar algunos cuantiles concretos, precisamente por el carácter discreto de la variable. Es por eso que en el caso de distribuciones discretas la definición de cuantil debe afinar un poco más. Hablamos del cuantil p como del mayor valor x tal que $P(X \leq x) \geq p$. Por ejemplo:

- Si $X \sim B(15, 0.65)$, $Q_{0.05} = 7$, lo que en R se consigue mediante

```
qbinom(0.05, 15, 0.65)
```

- Si $X \sim Poisson(5.8)$, $Q_{50} = 6$, dado en R por

```
qpois(0.50, 5.8)
```

3.3.2. Distribuciones continuas

En el caso de las distribuciones continuas, dado $p \in (0, 1)$ siempre existe un valor x tal que $P(X \leq x)$ es exactamente igual a p , y ese valor es el percentil $100p$ o el cuantil p . A modo de ejemplos:

- Si $X \sim Exp(1/2)$, $Q_{0.05} = 0.1026$, dado por

```
qexp(0.05, 1/2)
```

- Si $X \sim N(0, 4.5)$, $Q_{0.95} = 7.402$, dado en R por

```
qnorm(0.95, 0, 4.5)
```

3.4. Simulación de muestras

Una de las aplicaciones más comunes de la Estadística es la de proporcionar un marco teórico para poder realizar simulaciones de procesos más o menos complejos. En esas simulaciones existe la necesidad de contar con datos inventados, pero inventados según un modelo proporcionado por una distribución de probabilidad que se suponga adecuado para el fenómeno que estamos simulando. Es por ello que la mayoría de los paquetes de software estadístico, entre ellos R, facilitan la posibilidad de obtener muestras aleatorias simples de las distribuciones más usuales.

Supongamos que estamos interesados en simular datos de una distribución normal. Utilizamos entonces, la función `rnorm()`, cuya sintaxis es muy sencilla: tan sólo tenemos que especificarle el número de valores que queremos simular y los parámetros de la distribución. Por ejemplo, para generar 250 valores de una $N(\mu = 100, \sigma = 10)$, usamos

```
rnorm(250,100,10)
```

Vamos a comprobar que, en efecto, esta muestra procede de una $N(100, 10)$. Para ello podemos comparar el histograma de la muestra con la función de densidad de esa $N(100, 10)$. El resultado aparece en la Figura 3.1. Hay que decir que el parecido es notable, aunque sería aún más notable si hubiéramos elegido aún más datos (más de 250).

El código es el siguiente:

```
muestra<-sort(rnorm(250,100,10))
```

genera la muestra de tamaño 250 de la $N(100, 10)$ y la ordena mediante la función `sort()`.

```
hist(muestra,freq=FALSE)
```

grafica el histograma de los datos de la muestra en escala de densidad.

```
lines(muestra,dnorm(muestra,100,10))
```

añade al histograma la gráfica de la función de densidad evaluada en los valores de la muestra.

Observamos que la función `lines()` se trata de una función muy similar a `plot()` en su estructura y utilidad, cuya principal diferencia radica en que `plot()` es lo que en R se conoce

como una función gráfica de primer nivel y `lines()` lo es de segundo nivel. Que una función gráfica sea de primer nivel quiere decir que por sí misma abrirá una ventana gráfica y mostrará su resultado. Que una función sea de segundo nivel implica que por sí misma no tiene autoridad para abrir una ventana de gráficos, tan sólo puede añadirse a una ventana ya abierta.

`hist()` es una función gráfica de primer nivel, por lo que realiza el histograma y lo muestra en una ventana de gráfico. A esa ventana se añade la función de densidad graficada mediante `lines()`.

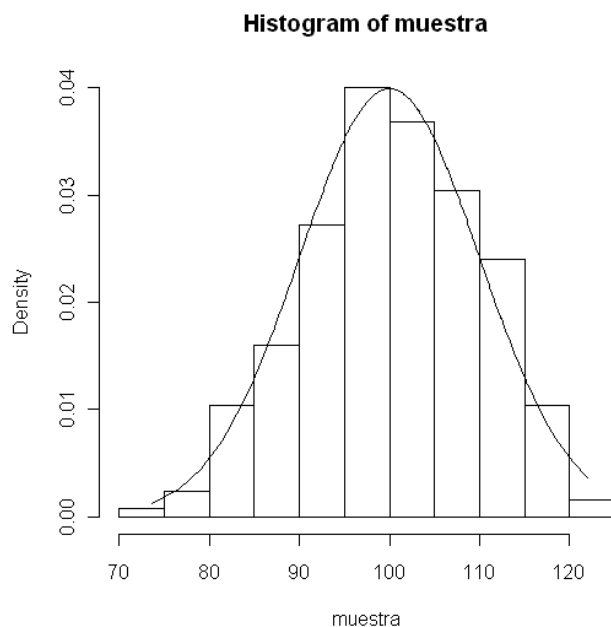


Figura 3.1: Histograma de una muestra junto con la función de densidad del modelo del que procede

.

Capítulo 4

Inferencia Estadística

En este capítulo se describe una manera de realizar estimaciones por intervalos de confianza y contrastes de hipótesis de algunos parámetros a través del código de R.

4.1. Estimación por Intervalos de confianza

Los intervalos de confianza guardan una relación biunívoca con los contrastes de hipótesis estadísticas. Ese es el motivo por el que R no tiene un comando específico para la construcción de intervalos de confianza, sino que éstos son proporcionados como parte de los resultados vinculados a los contrastes de hipótesis.

Sin embargo, es casi trivial la posibilidad de usar R como una simple calculadora para aplicar las fórmulas de los intervalos de confianza conocidos y eso es lo que vamos a hacer aquí.

En esta sección vamos a plasmar con ejemplos cómo podemos obtener algunos intervalos de confianza bilaterales. Lo vamos a hacer exclusivamente con código y sin la ayuda de ningún paquete adicional, para lo cual recordamos la sintaxis de algunas funciones:

- `mean(datos)` devuelve la media muestral de datos.
- `sd(datos)` devuelve la cuasi-desviación típica muestral de datos.
- `sqrt(x)` devuelve \sqrt{x} .
- `qnorm(a)` devuelve z_a .

- `qt(a, v)` devuelve $t_{a,v}$.
- `qchisq(a, v)` devuelve $\chi^2_{a,v}$.

4.1.1. De la media de una distribución normal con varianza desconocida

Recordemos que si denotamos x_1, \dots, x_n a una muestra de una distribución $N(\mu, \sigma)$, ambas desconocidas, un intervalo de confianza $(1 - \alpha)$ para μ viene dado por

$$\bar{x} \pm t_{n-1; \frac{\alpha}{2}} s / \sqrt{n}$$

donde \bar{x} es la media muestral, s es la cuasi-varianza muestral, n es el número de observaciones y $t_{n-1; \frac{\alpha}{2}}$ es el percentil $\frac{\alpha}{2}$ de la distribución t -Student con $n - 1$ grados de libertad.

Ejemplo: Los siguientes datos corresponden al nivel de glucosa en sangre, en ayunas, obtenidos de muestras de sangre de 30 pacientes diabéticos. Si asumimos que los datos siguen una distribución aproximadamente normal, ¿cómo podemos obtener un intervalo de confianza para la concentración media de glucosa en sangre, en ayunas para personas diabéticas, con $\alpha = 0.05$?

Tabla 4.1: Concentraciones de glucosa en sangre

97.68	105.82	104.58	121.84	107.77	95.79	111.19	98.47
106.05	101.05	96.83	104.17	94.20	94.42	104.17	97.13
103.55	101.05	98.40	107.68	97.13	102.79	100.56	98.95
103.57	105.14	96.19	102.55	106.45	113.16	103.47	107.28

Podemos cargar los datos, usando el comando `scan`

```
muestra <- scan()
 97.68 105.82 104.58 121.84 107.77  95.79 111.19  98.47
106.05 101.05  96.83 104.17  94.20  94.42 104.17  97.13
103.55 101.05  98.40 107.68  97.13 102.79 100.56  98.95
103.57 105.14  96.19 102.55 106.45 113.16 103.47 107.28
```

- Calculamos la cota inferior del intervalo, llamándola `ci`.

```
ci<-mean(muestra)-qt(0.975,31)*sd(muestra)/sqrt(32)
```

- Calculamos la cota superior del intervalo, llamándola `cs`.

```
cs<-mean(muestra)+qt(0.975,31)*sd(muestra)/sqrt(32)
```

- Unimos en un vector la cota inferior y la cota superior, haciéndolas aparecer en la ventana de resultados.

```
c(ci,cs)
```

4.1.2. De la media de una distribución cualquiera, con muestras grandes

Ejemplo: Consideramos 300 datos correspondientes al tiempo hasta el fallo (en años) de unas determinadas componentes electrónicas usadas en los marcapasos. Los datos se encuentran en el fichero `tiempos.fallo.rda`, en una hoja llamada `datos.tiempos.fallo`

Al cargarlos, podemos ver que la variable se llama *años* y realizando un histograma, podemos ver la forma que tienen los datos.

```
load("tiempos.fallo.rda")
names(datos.tiempos.fallo)
hist(datos.tiempos.fallo)
```

Sabemos que, visto el histograma, no es admisible pensar que la variable sigue una distribución normal, pero tenemos 300 datos, suficientes para poder aplicar el resultado basado en el teorema central del límite que determina que un intervalo para μ a un nivel de confianza α es

$$\bar{x} \pm z_{\frac{\alpha}{2}} s / \sqrt{n}$$

siendo n el tamaño de la muestra. El código es el siguiente:

```
ci<-mean(datos.tiempos.fallo$años)-qnorm(0.975)*sd(datos.tiempos.fallo$años)/sqrt(300)
cs<-mean(datos.tiempos.fallo$años)+qnorm(0.975)*sd(datos.tiempos.fallo$años)/sqrt(300)
c(ci,cs)
```

4.1.3. De una proporción

Ejemplo: Supongamos que un estudio para detectar alergias en menores, se comprueba que en una muestra de 300 niños y niñas, 21 sufren algún tipo de alergia alimenticia. Obtener un intervalo de confianza al 95 % para el porcentaje de menores alérgicos.

El intervalo, para un nivel α viene dado por

$$\hat{p} \pm z_{\frac{\alpha}{2}} \sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}$$

donde \hat{p} es la proporción muestral (en nuestro caso, $\frac{21}{300}$) y n es el tamaño de la muestra (en nuestro caso, 300).

El código para obtener el intervalo es el siguiente:

```
n=300
k=21
ci<-k/n-qnorm(0.975)*sqrt((k/n)*(1-k/n)/n)
cs<-k/n+qnorm(0.975)*sqrt((k/n)*(1-k/n)/n)
c(ci,cs)
```

4.1.4. De la varianza de una distribución normal

Ejemplo: Finalmente, vamos a obtener un intervalo de confianza para la varianza de la variable concentración de glucosa en sangre, en ayunas, en personas diabéticas.

Recordamos que dicho intervalo viene dado por

$$\left(\frac{(n-1)s^2}{\chi^2_{1-\frac{\alpha}{2};n-1}}, \frac{(n-1)s^2}{\chi^2_{\frac{\alpha}{2};n-1}} \right).$$

Usamos el código:

```
n=32
ci<-(n-1)*var(muestra)/qchisq(0.975,n-1)
cs<-(n-1)*var(muestra)/qchisq(0.025,n-1)
c(ci,cs)
```

4.2. Contrastes de hipótesis

A lo largo de este tema vamos a abordar la realización de contrastes de hipótesis paramétricas a través de diversos ejemplos.

4.2.1. Contrastes sobre medias: La función `t.test()`

En esta sección veremos cómo resolver problemas que involucran a la media de una población comparándola con un valor hipotético o a la media de dos poblaciones (independientes o apareadas), comparándolas entre sí. Lo que tienen en común estas pruebas es que todas ellas se basan en un estadístico de contraste que sigue una distribución *t* de *Student*. En R, el código necesario para llevar a cabo estos contrastes se basa en la misma función, la función `t.test()`. Su sintaxis básica es la siguiente:

```
t.test(x, y = NULL, alternative = c("two.sided", "less", "greater"), mu = 0,
paired = FALSE, var.equal = FALSE, conf.level = 0.95)
```

donde

- **x** es un vector de datos correspondiente a una de las muestras o a la única muestra del problema. Si estamos haciendo un test sobre la media de una población, **x** contiene la única muestra. Si estamos realizando un test de comparación de medias, **x** será la primera de las dos muestras.
- **y** corresponde a la segunda muestra en un test de comparación de medias. Si no es el caso y estamos realizando un test sobre una sola población, simplemente no se incluye.
- **alternative** especifica la dirección de la hipótesis alternativa. Como puede verse, tiene 3 posibles valores, “two.sided” (bilateral), “less” (unilateral a la izquierda) y “greater” (unilateral a la derecha).
- **mu** es el valor hipotético con el que se compara la media o la diferencia de medias en el contraste.
- **paired** especifica si las dos muestras **x** e **y**, en caso de que aparezcan, son apareadas o no.

- En el caso en el que aparecen dos muestras, `var.equal` especifica si podemos suponer varianzas iguales o no.
- `conf.level` es el nivel de confianza de los intervalos que se mostrarán asociados al test.

4.2.1.1. Para la media de una población

Ejemplo: Se sospecha que el nivel medio de glucosa en sangre, en ayunas, de las personas diabéticas es de 108 mg/100ml. Para contrastar esta suposición se utilizan análisis de sangre realizados a 45 personas diabéticas en ayunas. Los datos se encuentran en el fichero diabeticos.txt. ¿Es correcto asumir que la media del nivel de glucosa en ayunas es de 108 mg/100ml? (Utilícese un nivel de significación del 5%).

Fijémonos que nos piden claramente que confirmemos una afirmación: que la media es 108mg/100ml. Por lo tanto, si denotamos μ al nivel medio de glucosa en sangre, podemos plantear el contraste $H_0 : \mu = 108$ frente a $H_1 : \mu \neq 108$.

Dado que el tamaño muestral es generoso (45, superior a 30), no necesitamos la hipótesis de normalidad de los datos. Dicho esto, vemos que se trata de un contraste sobre la media de una distribución normal, contraste bilateral.

Abrimos el fichero de datos `diabeticos.txt`, por ejemplo con el bloc de notas, y observamos que los decimales están separados por una coma y que la primera línea de , lo que debemos indicar al utilizar el comando `read.table`. Y una vez que hemos importado los datos debemos especificar, cómo definimos el test.

```
datos.diabeticos<-read.table("diabeticos.txt",header=TRUE,dec=",")
t.test(datos.diabeticos$glucosa,alternative="two.sided",mu=108)
```

El resultado es el siguiente:

```
One Sample t-test
data:  diabeticos$glucosa
t = -1.6771, df = 44, p-value = 0.1006
alternative hypothesis: true mean is not equal to 108
95 percent confidence interval:
 103.5965 108.4035
```

```
sample estimates:
mean of x
      106
```

Analicemos el resultado con detalle:

- En primer lugar, nos recuerda que estamos analizando la variable `diabeticos$glucosa`.
- A continuación nos informa del valor del estadístico de contraste ($t = -1.6771$), de los grados de libertad ($df = 44$) y del p -valor ($p\text{-value} = 0.1006$). Ya podemos, por tanto, concluir: Dado que el p -valor no es inferior a 0.05, no tenemos suficientes evidencias en los datos para rechazar la hipótesis nula ($\mu = 108$) en favor de la alternativa ($\mu \neq 108$). Es decir, con los datos de la muestra no tenemos suficientes evidencias de que el nivel medio de concentración de glucosa en sangre, para diabéticos sea distinto de 108.
- Nos recuerda cuál era la hipótesis nula que habíamos planteado:
`alternative hypothesis: true mean is not equal to 108.`
- A continuación proporciona un intervalo de confianza bilateral, con un nivel de confianza del 95 %, para la media de la distribución normal que se le supone a los datos:
`95 percent confidence interval:`
`103.5965 108.4035`

Es importante recordar la relación que guarda el intervalo de confianza con el contraste de hipótesis. Observamos que el valor hipotético que hemos considerado para la media, 108, está dentro de este intervalo, luego éste es un valor de confianza para μ . Es otra forma de concluir que no hay datos que avalen que la media de la variable es significativamente distinta de 108, ya que éste es un valor bastante plausible para esta media. Si los datos fueran tales que el intervalo de confianza para μ dejara fuera al valor 108, tendríamos razones para pensar que el valor de μ es significativamente distinto de 108, pero no es el caso.

- Finalmente, proporciona los estadísticos muestrales utilizados, en este caso, la media muestral:

```
sample estimates:
mean of x
      106
```

Por lo tanto, y a modo de conclusión, podemos decir que no hay evidencias de que la concentración de glucosa en sangre, en el caso de las personas diabéticas, no sea 108.

4.2.1.2. Para la diferencia de medias de poblaciones independientes

Ejemplo: Un ingeniero industrial ha sintetizado en el laboratorio una feromona con la que pretende luchar contra una plaga de insectos. La feromona se aplica en trampas donde caen los insectos masivamente. Hasta ahora se trabajaba introduciendo otro producto que se supone que atraía al insecto, por lo que el ingeniero desearía demostrar que su feromona sintetizada es más efectiva que dicho producto. Para probar si esto ocurre, prepara 100 trampas con el producto tradicional y 100 con su feromona y las distribuye, contabilizando el número de insectos atrapados en cada una de las 200 trampas. Con esos datos, ¿puede concluir el ingeniero que su feromona es más efectiva que el producto tradicional? Los datos se encuentran en el fichero feronoma.txt.

Vamos a llamar μ_V a la media de las capturas con el viejo producto y μ_N a la media de las capturas con la nueva feromona. Lo que nos piden en el enunciado es que contrastemos la hipótesis nula $H_0 : \mu_V \geq \mu_N$ frente a la alternativa $H_1 : \mu_V < \mu_N$:

- En primer lugar, podemos suponer que las muestras son independientes. Nada hace pensar que los datos de la muestra bajo el producto antiguo hayan tenido nada que ver en la muestra bajo el producto nuevo ni al contrario.
- Con respecto al tamaño muestral, debe preocuparnos la hipótesis de normalidad: recordemos que si el tamaño de la muestras es pequeño, éstas deberían proceder de una distribución normal, pero no es el caso: ambas muestras tienen tamaños superiores a 30.
- Finalmente, deberemos plantearnos si podemos suponer o no que las varianzas son iguales.

Lo primero que tenemos que hacer para importar los datos (que se encuentran en un fichero de tipo texto) es ver cómo están almacenados. Si lo abrimos, por ejemplo con el bloc de notas, vemos que están separados por tabulaciones y que los nombres de las variables están en la primera fila. La Figura 4.1 (a) muestra el bloc de notas que contiene a los datos.

Es importante observar que los datos de las dos muestras aparecen en dos columnas paralelas. Esta no es una forma correcta de especificarlas, ya que parece que cada dato de la primera

Capturas, producto.tradicional	Capturas, feromona
54	70
45	72
54	71
40	63
50	64
40	60
51	73
44	56
47	64
49	64
49	62
50	63
47	65
53	66
56	61
42	65
48	65
45	76
46	57
50	62
52	65
51	56
54	58
44	59
46	63
48	72
49	61
50	69
46	72

Figura 4.1: Importando los datos del fichero *feromona.txt*

muestra está relacionado con otro dato de la segunda muestra y, en realidad, las muestras son independientes (de hecho podrían tener distinto tamaño muestral). Por este motivo, tenemos que especificar este hecho, para que R entienda que se trata de dos muestras independientes.

Podemos usar el código:

```
feromona<-read.table("feromona.txt",header=TRUE)
t.test(x=feromona$capturas.feromona, y=feromona$capturas.producto.tradicional,
alternative="greater",mu=0)
```

Cuyo resultado es el siguiente:

```
Welch Two Sample t-test
data: capturas by producto
t = 20.4367, df = 197.952, p-value < 2.2e-16
alternative hypothesis: true difference in means
is greater than 0
95 percent confidence interval:
14.25580 Inf
sample estimates:
mean in group capturas.feromona
64.94
mean in group capturas.producto.tradicional
49.43
```


Vamos a analizarlo punto por punto:

- Especifica que se trata de un test t para la variable *capturas* separada por el factor *producto*.
- Proporciona el valor del estadístico de contraste ($t = 20.4367$), los grados de libertad ($df = 197.952$), y el p -valor ($p\text{-value} < 2.2e-16$). Dado que el p -valor es inferior a 0.05, ya podemos concluir que tenemos evidencias en los datos para afirmar con un 95 % de confianza que la media de las capturas con la feromona es superior a la de las capturas con el viejo producto.
- Especifica cuál es nuestra hipótesis alternativa.
- Proporciona un intervalo de confianza para la diferencia de las medias. En este caso, la probabilidad de que el intervalo (14.25580 Inf) contenga a la diferencia de las medias es del 95 %. El cero no es, por tanto, un valor bastante plausible y por ello hemos rechazado la hipótesis nula en favor de la alternativa.
- Proporciona las dos medias muestrales.

En resumen, hemos concluido que podemos afirmar con un 95 % de confianza que la feromona es más efectiva que el viejo producto al aumentar significativamente el promedio de capturas.

4.2.1.3. Para la diferencia de medias apareadas

Ejemplo: En un programa de Control de Enfermedades Crónicas, la hipertensión está incluida como la primera patología a controlar. 15 pacientes hipertensos son sometidos al programa y su tensión asistólica es controlada antes y después de 6 meses de tratamiento. Los datos son los siguientes:

inicio	180	200	160	170	180	190	190	180	190	160	170	190	200	210	150
fin	140	170	160	140	170	150	140	150	190	170	120	160	170	160	150

Si los datos siguen una distribución normal, ¿podemos afirmar que es efectivo el tratamiento?

Vamos a denotar por μ_D al promedio de las medidas de tensión asistólica después del tratamiento y por μ_A al mismo promedio antes del tratamiento. Nos piden contrastar $H_0 : \mu_D \geq \mu_A$ frente a $H_1 : \mu_D < \mu_A$.

En este caso, el código sería el siguiente:

```
tension<-read.table("tension.txt",header=TRUE,dec=",")
t.test(x=tension$inicio, y=tension$fin, alternative="greater",mu=0,paired=TRUE)
```

Paired t-test

data: tension\$inicio and tension\$fin

t = 4.8316, df = 14, p-value = 0.0001332

alternative hypothesis: true difference in means is greater than 0

95 percent confidence interval:

16.09828 Inf

sample estimates:

mean of the differences

25.33333

Vamos a analizar los resultados con detalle:

- En las dos primeras líneas se nos informa que estamos realizando un test t para muestras apareadas sobre los datos relativos a las variables *fin* e *inicio* del conjunto de datos *tension*.
- En la siguiente línea aparece el valor del estadístico de contraste ($t = 4.8316$), de los grados de libertad ($df = 14$) y el p -valor ($p\text{-value} = 0.0001332$). Visto el valor de éste podemos concluir que existen evidencias en los datos de la muestra de que el tratamiento disminuye el promedio de la tensión asistólica. Observamos que el p -valor es muy bajo, luego las diferencias detectadas son muy significativas.
- A continuación aparece el tipo de hipótesis alternativa que hemos elegido.
- Posteriormente aparece un intervalo de confianza al 95 % para la diferencia de las medias. El hecho de que el cero no esté contenido en dicho intervalo es otra forma de ver que podemos rechazar la hipótesis nula en favor de la alternativa.
- Finalmente aparece el valor muestral de la diferencia de las medias.

En resumen, los datos muestran indicios de que el tratamiento disminuye la tensión asistólica en promedio.

4.2.2. Contrastes sobre proporciones: La función `prop.test()`

R realiza este tipo de contrastes de dos formas: mediante una prueba tipo χ^2 o mediante una prueba binomial exacta.

En el primero de los casos, el de las pruebas tipo χ^2 , lo que se hace es comparar las frecuencias de casos favorables en la muestra de los datos (frecuencias observadas, O_i) con la muestra de casos favorables que habría en una muestra con el mismo número de datos si la hipótesis nula fuera cierta (frecuencias esperadas, E_i). El estadístico utilizado para este contraste es

$$\chi^2 = \sum_i \frac{(O_i - E_i)^2}{E_i},$$

que, bajo el supuesto de que ninguna frecuencia esperada E_i es inferior a 5, se distribuye según una distribución χ^2 . En el caso de que alguna frecuencia esperada sea inferior a 5 se suele utilizar la corrección por continuidad de Yates, que da lugar al estadístico del contraste

$$\chi^2 = \sum_i \frac{(|O_i - E_i| - 0.5)^2}{E_i}.$$

La prueba binomial exacta parte del hecho de que, si la hipótesis nula fuera cierta, la distribución del número de casos favorables en la muestra sería binomial. Valorando el número observado de casos favorables dentro de la distribución binomial que se daría bajo la hipótesis nula, se obtiene el p -valor de la prueba.

Vamos a partir del hecho de que conocemos el número de éxitos y fracasos en la muestra. Si no es así y únicamente tenemos los datos en una hoja de datos, podemos rápidamente tabularla mediante la función `table()` a la que sólo hay que especificarle la hoja de datos a tabular y, si ésta tuviera más de una variable, cuál de ellas queremos tabular. La sintaxis de la función `prop.test` es la siguiente. Dicha sintaxis también nos servirá para los contrastes de comparación de dos proporciones:

```
prop.test(x, n, p = NULL, alternative = c("two.sided", "less", "greater"),
conf.level = 0.95, correct = TRUE)
```

Comentamos cada uno de los argumentos de la función:

- **x** puede especificar dos cosas. O bien el número de éxitos, o bien, mediante una matriz

de dos columnas, el número de éxitos y de fracasos en cada muestra.

- `n` especifica el número de datos de la muestra en el caso en que `x` sea el número de éxitos, y es ignorado en el caso en que `x` proporcione también el número de fracasos.
- `p` es el vector de probabilidades de éxito bajo la hipótesis nula. Debe ser un vector de la misma dimensión que el número de elementos especificado en `x`.
- `alternative` especifica la dirección de la hipótesis alternativa, tomando los valores "`two.sided`", "`greater`" o "`less`".
- `conf.level` es el nivel de confianza de los intervalos que se muestran entre los resultados.
- `correct` especifica si se usa la corrección por continuidad de Yates. Observamos que la opción por defecto es que sí se use esta corrección.

4.2.2.1. Para la proporción en una población

Ejemplo: En un determinado servicio de odontología se espera que aproximadamente el 75% de las visitas no requieran una extracción dentaria inmediata. En cierto año, de 1225 visitas, 926 no necesitaron una extracción inmediata. ¿Se puede decir que el porcentaje de ese año fue significativamente superior al porcentaje esperado?

Si denotamos por p la proporción de visitas que no requieren extracción, se nos está pidiendo que contrastemos $H_0 : p = 0.75$ frente a $H_1 : p > 0.75$.

En este caso, podemos resolver el contraste usando:

```
prop.test(x=926,n=1225,p=0.75,alternative="greater",correct=FALSE)
```

cuyo resultado es:

```
1-sample proportions test without continuity correction
data:  926 out of 1225, null probability 0.75
X-squared = 0.2288, df = 1, p-value = 0.3162
alternative hypothesis: true p is greater than 0.75
95 percent confidence interval:
0.7351821 1.0000000
```

```
sample estimates:  
p  
0.7559184
```

Analizamos estos resultados:

- Especifica en primer lugar que se trata de un test para la proporción en una muestra.
- Especifica a continuación el valor hipotético en la hipótesis nula.
- Proporciona el valor del estadístico de contraste y, lo que más nos interesa, el p -valor. En este caso el p -valor es bastante superior a 0.05, luego a la luz de estos datos no podemos rechazar la hipótesis nula en favor de la alternativa, es decir, no podemos concluir que el porcentaje de visitas que no requieren extracción esté por encima del 75 %.
- A continuación recuerda la hipótesis alternativa.
- Facilita un intervalo de confianza (en este caso unilateral a la derecha) para la proporción.
- Finalmente, muestra la proporción muestral.

Comentamos también que la prueba binomial exacta puede realizarse mediante la función `binom.test`, cuya sintaxis básica es similar a la de `prop.test`:

```
binom.test(x,n,p=0.5,alternative=c("two.sided","less","greater"), conf.level=0.95)
```

4.2.2.2. Para la diferencia de proporciones

Este tipo de contrastes se realiza también mediante la función `prop.test()`, pero previamente debemos comentar algo acerca de cómo introducir los datos. Los éxitos y los fracasos de cada muestra deben ir en dos filas de una matriz de dos columnas, es decir, con una estructura como la siguiente:

N° de éxitos de la muestra 1 (n11)	N° de fracasos de la muestra 1 (n12)
N° de éxitos de la muestra 2 (n21)	N° de fracasos de la muestra 2 (n22)

Para crear una matriz así se utiliza la función `matrix`, a la que tenemos que especificarle mediante un vector los elementos de la matriz, las dimensiones de la matriz y el sentido en el que vienen especificados los elementos. En el ejemplo nuestro caso sería

```
matrix(c(n11, n12, n21, n22),2,2,byrow=TRUE)
```

o bien

```
matrix(c(n11, n21, n12, n22),2,2,byrow=FALSE).
```

Las pruebas que R utiliza para este tipo de contrastes de nuevo se basan en el uso del estadístico χ^2 , comparando las frecuencias observadas en ambas muestras con las que aparecerían bajo la hipótesis nula, o también en la conocida como prueba exacta de Fisher.

Vamos a trabajar sobre el siguiente enunciado:

Ejemplo: A raíz de la alarma creada entre la opinión pública por la repercusión que tuvo el caso de un bloque de edificios con un transformador en su planta baja y donde un gran porcentaje de vecinos sufrió cáncer, se decide realizar un estudio para tratar de encontrar relación entre la cercanía de un transformador eléctrico y la incidencia del cáncer.

Para ello, se eligió una muestra aleatoria de edificios con transformadores en su planta baja durante un periodo de más de 10 años, contabilizando el número de habitantes en ellos, 2150, y todos los casos de cáncer detectados en los 5 últimos años, 37. Por su parte, se recolectó otra muestra aleatoria de control con edificios que no tuvieran ningún transformador eléctrico cercano, contabilizando también el número de personas, 2200, y el número de casos de cáncer en ellos en los últimos 5 años, 33. En ambas muestras, los expertos procuraron eliminar la posibilidad de ruidos, es decir, la presencia de otros factores que pudieran incidir en variar la incidencia del cáncer en alguna de las muestras.

A la luz de los datos de este estudio, ¿podemos afirmar que la cercanía de un transformador eléctrico aumenta la proporción de casos de cáncer? (Utilícese un nivel de significación del 5%)

Si llamamos p_{CT} a la proporción de casos de cáncer en los edificios con transformador cercano y p_{ST} a la proporción análoga en los edificios sin transformador, nos piden que contrastemos $H_0 : p_{CT} = p_{ST}$ frente a $H_1 : p_{CT} > p_{ST}$.

En este caso usaríamos:

```
tabla<-matrix(c(37,2113,33,2167),2,2,byrow=TRUE)
```

Finalmente, la aplicación de la función `prop.test` sería la siguiente:

```
prop.test(tabla, alternative='greater', correct=FALSE)
```

Los resultados son los siguientes:

```
2-sample test for equality of proportions without continuity
correction
```

```
data:  tabla
X-squared = 0.3352, df = 1, p-value = 0.2813
alternative hypothesis: greater
95 percent confidence interval:
-0.004071904 1.000000000
sample estimates:
prop 1 prop 2
0.01720930 0.01500000
```

Lo que realmente nos interesa es el p -valor, que aparece en la tercera línea, 0.2813, y que indica que no se puede rechazar la hipótesis nula en favor de la alternativa, es decir, no podemos, con los datos existentes, asegurar con un 95 % de confianza que la proporción de casos de cáncer sea superior en los edificios que tengan un transformador eléctrico en su planta baja.

Comentamos, por último, que la prueba exacta de Fisher se realiza con la función `fisher.test`, que tiene una sintaxis parecida a la de `prop.test`, aunque admite muchos más argumentos en función de las características de la tabla de contingencia a analizar.

4.2.3. Contraste para la comparación de varianzas: La función `var.test()`

Ejemplo: Un grupo de personas participa en un estudio nutricional que trata de analizar los niveles asimilados de vitamina C en sangre de fumadores y no fumadores. Los resultados en mg/l se encuentran en el fichero `vitaminac.txt`. Si asumimos que los datos son normales, ¿se puede concluir que el nivel de vitamina C asimilado es superior en los no fumadores?

Con el fin de contestar a la pregunta, planteamos las hipótesis $H_0 : \mu_1 \geq \mu_2$ versus $H_1 : \mu_1 < \mu_2$ donde estamos asumiendo que la primera muestra corresponde a los fumadores y la segunda a los no fumadores.

Observamos que el enunciado no menciona nada relativo a la igualdad de varianzas en ambos grupos, por lo que vamos a plantear en primer lugar un contraste de igualdad de varianzas (o desviaciones típicas). Si denotamos σ_F a la desviación típica del nivel de vitamina C asimilado en fumadores y σ_{NF} a la desviación típica del nivel de vitamina C asimilado en no fumadores, se trata de contrastar $H_0 : \sigma_F = \sigma_{NF}$ frente a $H_1 : \sigma_F \neq \sigma_{NF}$.

La función `var.test()` puede utilizarse con una sintaxis muy parecida a las anteriores:

```
var.test(x, y, ratio = 1, alternative = c("two.sided", "less", "greater"),
conf.level = 0.95)
```

1. `x` corresponde al vector de datos de la primera muestra.
2. `y` es el vector de datos de la segunda muestra.
3. `ratio` es el cociente hipotético con el que se compara. Habitualmente deseamos contrastar que las varianzas son distintas, o lo que es lo mismo, que su cociente es 1, así que la opción por defecto es precisamente `ratio=1`.
4. `alternative` especifica la hipótesis alternativa: `"two.sided"` para $H_1 : \sigma_1 \neq \sigma_2$, `"less"` para $H_1 : \sigma_1 < \sigma_2$ y `"greater"` para $H_1 : \sigma_1 > \sigma_2$.
5. `conf.level` es el nivel de confianza del intervalo para el cociente de varianzas que se muestra en las salidas.

Para la resolución del ejemplo anterior, tendríamos el siguiente código:

```
datos.vitaminac<-read.table("vitaminac.txt",header=TRUE,sep="\t",dec=",")
var.test(datos.vitaminac$fumadores,datos.vitaminac$nofumadores,
alternative="two.sided")
```

O alternativamente, podríamos haber utilizado la función `var.test()` de la forma


```
var.test(nivel ~ tabaco, alternative='two.sided', conf.level=.95,  
data=datos.vitaminac)
```

Los resultados son los siguientes:

```
F test to compare two variances  
data:  nivel by tabaco  
F = 0.3131, num df = 11, denom df = 10, p-value = 0.06976  
alternative hypothesis: true ratio of variances is not equal to 1  
95 percent confidence interval:  
 0.08544081 1.10400497  
sample estimates:  
ratio of variances  
 0.3131332
```

En la tercera línea podemos ver que aparece el valor del estadístico F , los grados de libertad en el numerador y el denominador y el p -valor. Ese $p\text{-value}=0.06976$ indica que no hay suficientes evidencias en los datos para rechazar la igualdad de varianzas. Y por lo tanto, a la vista de los resultados, realizaríamos el contraste de medias planteado asumiendo igualdad de varianzas.