

# Práctica 1. Fundamentos de R. Lectura y Creación de un fichero de R.

Christian J. Acal González y Miguel Ángel Montero Alonso



**UNIVERSIDAD  
DE GRANADA**

Todo el material para el conjunto de actividades de este curso ha sido elaborado y es propiedad intelectual del grupo **BioestadísticaR** formado por:

Antonio Martín Andrés  
Juan de Dios Luna del Castillo,  
Pedro Femia Marzo,  
Miguel Ángel Montero Alonso,  
Christian José Acal González,  
Pedro María Carmona Sáez,  
Juan Manuel Melchor Rodríguez,  
José Luis Romero Béjar,  
Manuela Expósito Ruíz,  
Juan Antonio Villatoro García.

Todos los integrantes del grupo han participado en todas las actividades, en su elección, construcción, correcciones o en su edición final, no obstante, en cada una de ellas, aparecerán uno o más nombres correspondientes a las personas que han tenido la máxima responsabilidad de su elaboración junto al grupo de **BioestadísticaR**.

Todos los materiales están protegidos por la Licencia Creative Commons **CC BY-NC-ND** que permite "descargar las obras y compartirlas con otras personas, siempre que se reconozca su autoría, pero no se pueden cambiar de ninguna manera ni se pueden utilizar comercialmente".

# Práctica 1. Fundamentos de R. Lectura y Creación de un fichero de R.

Christian J. Acal González y Miguel Ángel Montero Alonso

## 1.1 RStudio y su utilidad

### 1.1.1 Introducción

R es un programa estadístico que está orientado al análisis de datos. Entre otros aspectos, destaca por:

- Almacenamiento y manipulación de una base de datos.
- Gran variedad de técnicas estadísticas para el análisis de datos.
- Herramientas para el manejo sobre variables indexadas y para la construcción de gráficos avanzados.
- Lenguaje de programación abierto que permite la introducción de nuevas técnicas mediante la definición de funciones.

Para profesionales de las Ciencias de la Salud puede resultar que R sea bastante más complejo que otros programas estadísticos. Sin embargo, esto queda lejos de la realidad ya que entre las principales características por las que destaca R es por su enorme flexibilidad y por ser un programa de software libre que engloba un proyecto colaborativo, gratuito y abierto en el que muchos usuarios colaboran en el desarrollo y ampliación de funciones. R funciona mediante la ejecución de comandos y funciones que están indexadas en librerías o paquetes (*packages*) los cuales pueden extraerse del repositorio oficial de paquetes (Cran de R).

Asimismo, se han desarrollado diferentes interfaces con el fin de facilitar el manejo de R. En este contexto, surge el programa RStudio, un interfaz gratuito que permite acceder a todas las herramientas de R. Básicamente se trata de un software que facilita tanto la tarea de uso interactivo como la programación de scripts mediante el orden y la visualización de los procesos que son llevados a cabo en R de manera simultánea. Sin embargo, es importante tener en todo momento claro que R es el lenguaje de programación propiamente dicho, mientras que RStudio es un entorno que permite utilizar R de manera más cómoda e intuitiva gracias a la división, por defecto, de cuatro paneles que contienen las principales características de R. Para poder utilizar RStudio se requiere haber instalado R previamente.

### 1.1.2 Ventana principal de RStudio

Una vez haya finalizado todo el proceso de instalación del programa RStudio, en el escritorio se genera el icono que se muestra a continuación.



Figure 1: Icono de RStudio

Haciendo doble *click* sobre este icono se procede a arrancar el programa. Una vez iniciado el programa debe mostrarse la pantalla que aparece en la Figura 2

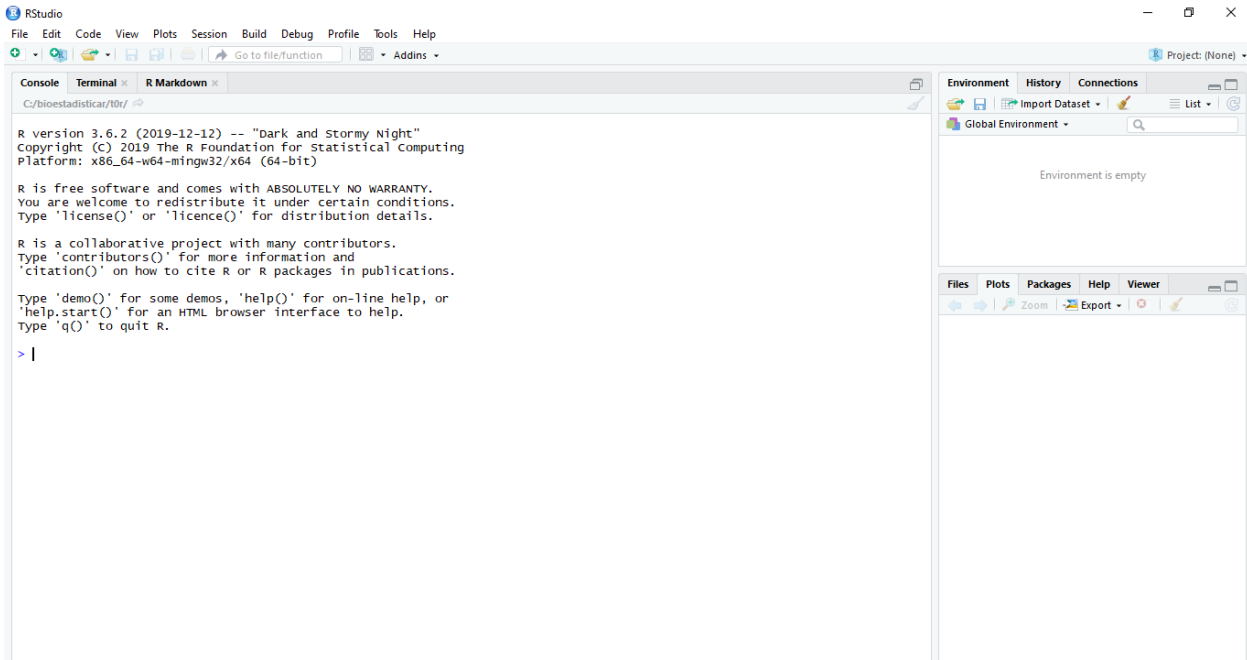


Figure 2: Pantalla que aparece cuando se abre RStudio

Se invita al usuario a que se dirija al menú **File** situado a la izquierda de la **Barra del menú principal** y siga la ruta **File** → **New File** → **R Script** de forma que se obtenga la pantalla que aparece en la Figura 3.

1. A este panel se le llama *Consola*. Al comienzo se muestra un breve mensaje inicial que contiene información sobre la versión actual del programa y algunas instrucciones básicas para su manejo. Este texto puede ser eliminado si se pincha en él y se ejecuta **Ctrl+I** con el teclado. Posteriormente aparece el símbolo '>' (denominado *prompt*) que indica que el programa está preparado para recibir una instrucción. A continuación del *prompt* el usuario puede escribir una expresión (p.ej. 1+1) y ejecutarla dándole al botón 'enter' o 'intro'. Automáticamente el programa devolverá el resultado. No obstante, a lo largo del curso todas las órdenes se escribirán y se ejecutarán desde el *script* (pantalla 4).
2. Este panel contiene una serie de funciones muy interesantes. Sin embargo, las pestañas que más se utilizarán a lo largo del curso serán:
  - *Plots*: Se guardan los gráficos que se generen desde el panel número 4.
  - *Packages*: Se puede instalar/cargar paquetes implementados en R.
  - *Help*: Se muestra información sobre una función concreta.
3. En esta pantalla se van almacenando todos los objetos que el usuario vaya creando. También se tiene disponible el historial de lo que se vaya ejecutando.
4. Este panel es conocido como el *Editor* o *Script*. En esta pantalla es donde se van a ir escribiendo los distintos códigos (comandos) de R para realizar los análisis. A pesar de que los comandos se pueden escribir también en la *consola* (pantalla 3), es recomendable utilizar esta sección para la implementación de los códigos debido a su flexibilidad: permite guardar el script de comandos de manera que pueda ser utilizado en otras sesiones, resalta los elementos por colores, posibilita modificar los comandos sobre la marcha, acepta escribir tantas líneas de código como se deseen, etc. A modo de ejemplo el usuario puede probar a escribir 1+1 y ejecutar dicha orden mediante **Ctrl+enter** o **Ctrl+intro** en el teclado. Automáticamente el programa devolverá el resultado en la *consola*. Para guardar el área de trabajo

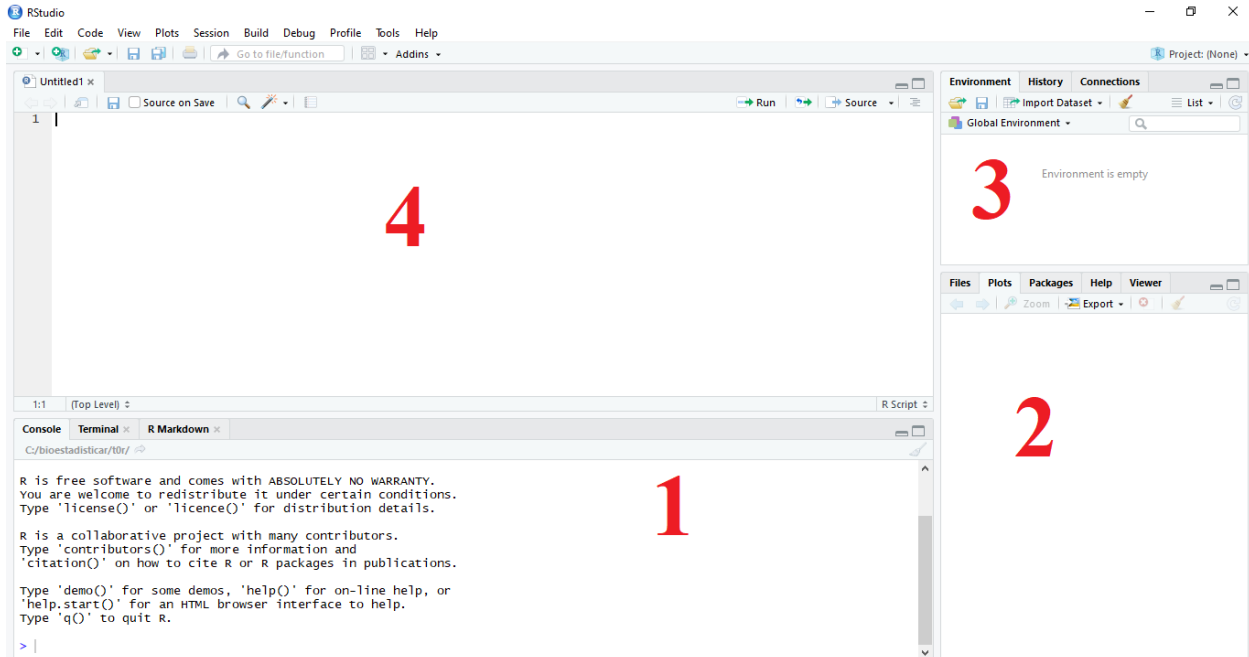


Figure 3: Pantalla final inicial

actual (*script*) en cualquier momento de la sesión o cargar un área de trabajo guardada anteriormente se utilizan las opciones **File** → **Save As...** y **File** → **Open File...** respectivamente.

## 1.2 El uso de R como calculadora

R es un lenguaje interpretado en el sentido de que se pueden escribir órdenes y R las ejecutará inmediatamente. De hecho, podemos considerar como primera aplicación de R su uso como calculadora, ya que es capaz de manejar las operaciones elementales fácilmente. El objetivo de este apartado es ayudar al estudiante en sus primeros pasos con R utilizándolo como una calculadora interactiva. Se utilizará siempre el entorno de **RStudio**.

### 1.2.1 Conceptos básicos de R

Los principales conceptos por los cuales se basa el programa estadístico R son el **objeto**, **función** y **expresión**. Estos tres conceptos quedan resumidos en la siguiente frase: El objeto es ‘todo lo que es’, la función es ‘todo lo que se ejecuta’ y la expresión es la ejecución de un modelo a través de un conjunto de funciones.

- **Objeto.** La explicación teórica de este concepto se escapa de los objetivos del presente curso pero pueden ser vistos como estructuras que combinan datos y funciones que operan sobre ellos y son muy útiles en un entorno pensado para el análisis estadístico de datos. A modo de resumen, todo lo que existe en R y que tiene un nombre propio es un objeto. Por ejemplo, los números, las variables, tablas, vectores, matrices, etc. son objetos.
- **Función.** *Grosso modo* es todo aquello que como primer paso recibe un *input* (entrada) que es procesada de manera interna y que devuelve una salida como fin de una sucesión. Cuando se habla de entrada se refiere a que toda función necesita variables, números u otros tipos de objetos para ponerse en marcha. El término ‘procesar’ se refiere a que ejecuta un algoritmo que tiene implementado intrínsecamente y que producirá una salida o resultado. Las funciones en R se caracterizan por un

nombre corto y que dé una idea de lo que hace la función. A modo de ejemplo la función **mean** devuelve la media aritmética de un vector de números.

- **Expresión.** Al igual que una oración está formada por palabras y signos de puntuación, cuando se habla de expresión matemática se está refiriendo al concepto que establece una relación entre objetos matemáticos, variables y constantes en la versión más simple. Una fórmula, un cálculo indicado o una función son sinónimos de expresiones matemáticas, ya que todas tienen la particularidad de que indicarán el valor que tiene una variable cuando se ejecuten las operaciones indicadas en la expresión.

## 1.2.2 Operaciones aritméticas con R

Como se ha comentado a lo largo de este tema, R se puede utilizar como una calculadora, ya que es capaz de realizar fácilmente las operaciones elementales. Las operaciones más frecuentes con su simbología asociada en R son:

Símbolo	Significado
+	Suma
-	Resta
*	Multiplicación
/	División
%/%	División entera
^	Potencia
sqrt(x)	Raíz cuadrada de x
exp(x)	Exponencial
sin(x)	Seno
cos(x)	Coseno
log(x)	Logaritmo neperiano de x
log10(x)	Logaritmo decimal de x
abs(x)	Valor absoluto de x
%%	Resto de una división
factorial(x)	Factorial
choose(m,n)	Número combinatorio
pi	El número $\pi$
round(x,digits)	Redondea a la cantidad indicada de decimales
ceiling(x)	Redondea al entero superior
floor(x)	Elimina la parte decimal

**Nota:** La coma decimal se designa con un punto.

En la Figura 4 se presentan algunos ejemplos de operaciones. Se recomienda repetir estas operaciones de manera que el usuario se familiarice con el entorno. Hay que escribir las órdenes en el *script*, ejecutarlas mediante **Ctrl+Enter** situando el cursor encima de la línea donde se encuentre la expresión y comprobar que salen los resultados correctos en la *consola*.

Al alumnado debe haberle llamado la atención la sentencia **#Esto es un ejemplo**. En R siempre que se quiera escribir algún tipo de comentario en el *script* que sirva como aclaración para el propio usuario, debe realizarse mediante el símbolo '#'. Todo lo que se escriba a continuación de este signo, R lo interpretará como comentario y no tendrá en cuenta esta sentencia aunque la ejecute en la consola. Nótese también que el [1] que aparece en la consola indica que es la primera línea de salida que devuelve el programa. Cuando los resultados sean más abultados (por ejemplo una matriz con 3 filas), estos índices pueden ayudar en la búsqueda de valores.

Obviamente R también acepta estructuras más complejas que una simple operación. Asimismo, conviene recordar que, en caso de querer alterar la jerarquía habitual entre las operaciones (primero multiplicación y

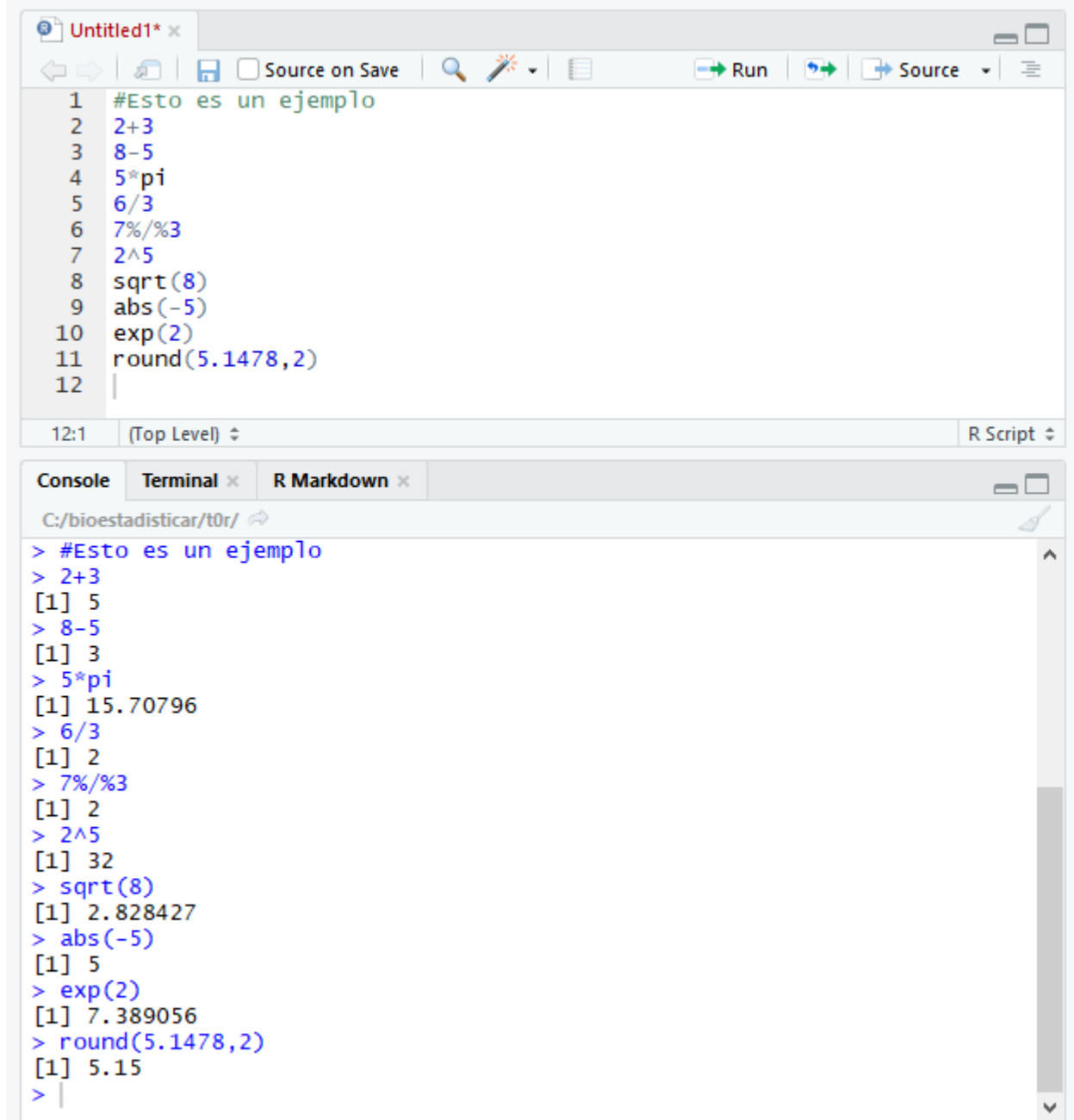


Figure 4: Algunas operaciones con RStudio

división y posteriormente suma y resta), ha de utilizarse paréntesis. En la siguiente salida figuran otra serie de ejemplos con múltiples operaciones en la misma expresión. En particular las expresiones que se están calculando son:  $2 + \frac{3}{4}$ ,  $\frac{2+3}{4}$ ,  $2 + \frac{3}{4} + 5$ ,  $\frac{2.4+3.7+0.5}{3}$ ,  $\frac{1+e^{2 \times 5}}{2 \times \sqrt{9}}$  y  $\frac{|-2 \times 5| + \ln(5)}{0.5^2}$ .

```
2+3/4
```

```
## [1] 2.75
```

```
(2+3)/4
```

```
## [1] 1.25
```

```
2+3/4+5
```

```
## [1] 7.75
```

```
(2.4+3.7+0.5)/3
```

```
## [1] 2.2
```

```
(1+exp(2*5))/(sqrt(9)*2)
```

```
## [1] 3671.244
```

```
(abs(-2*5)+log(5))/0.5^2
```

```
## [1] 46.43775
```

### 1.2.3 El uso de variables en R. El concepto de asignación

Una variable en el lenguaje de R es el nombre que recibe una entidad (numérica o de otro tipo) que puede ser modificada a lo largo de la sesión de R. Basta con elegir el nombre que tendrá y asignarle un valor. El concepto de asignación se refiere a la acción de otorgarle un valor (o valores) al correspondiente objeto de R. En referencia al nombre que se le otorgue a la variable, se recomienda que el nombre sea corto y preciso y que no disponga de caracteres especiales ni de espacios en blanco, puesto que de lo contrario el programa lanzará un mensaje de error avisando de que la definición de la variable no ha sido llevada a cabo (solo acepta los signos de puntuación de *punto* o *guión bajo*).

La asignación se puede realizar mediante `=` o `<-`. La única diferencia que radica entre ellas es la forma en las se almacenan internamente las variables. En consecuencia, se dispone de dos opciones para definir una variable. Por ejemplo, si se quiere asignar a la variable `x` el valor 5, se debe actuar de la siguiente forma.

```
#Asignación de variables
x=5
x<-5
```

Se invita a comprobar que se ha creado la variable `x` en la pestaña *Environment* que figura en el panel 3. Seguidamente, con el fin de probar que se ha almacenado correctamente se ordena mostrarla.

```
#Asignación de variables  
x=5  
x<-5  
x
```

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

Al mismo tiempo se pueden crear tantas variables como se deseen.

```
#Asignación de variables  
x=5  
x<-5  
x
```

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

```
y=8  
y
```

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

```
z<-1.5  
z
```

```
## [1] 1.5
```

Otro aspecto a tener en cuenta es que cada vez que se realice una asignación sobre una variable declarada anteriormente, R borra el valor antiguo y almacena el nuevo. Así, si ahora se define  $x=3$ , de manera automática R olvida que  $x$  anteriormente tomó el valor 5.

```
#Asignación de variables  
x=5  
x
```

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

```
x=3  
x
```

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

Es obvio que esta modificación se podría haber hecho directamente desde la línea donde se definió por primera vez la variable  $x$  sin necesidad de tener que crear otra línea de código.

## 1.2.4 Fórmulas con variables

Con las variables definidas se pueden realizar un sinfín de operaciones con ellas. La más simple sería realizar operaciones básicas con ellas mismas.



```
#Fórmulas con variables  
x=3.75  
exp(x)
```

```
## [1] 42.52108
```

```
x+5
```

```
## [1] 8.75
```

```
round(x,1)
```

```
## [1] 3.8
```

Sin embargo, el gran atractivo del uso de variables en R es la creación de nuevas variables mediante el uso de fórmulas a partir de las ya definidas. Es muy común encontrarse en situaciones donde se conozca el valor de una o varias variables y se desee determinar el valor de otra variable que dependa de los valores de las variables definidas anteriormente. En R, este cálculo es inmediato a través de la combinación del uso de fórmulas y la asignación de variables. A continuación se muestran algunos ejemplos con el fin de poner de manifiesto la potencia de este enfoque.

- Es conocido que 1 minuto es igual a 60 segundos. ¿Cuántos minutos serían 1740 segundos?

```
#Ejemplo de fórmulas  
seg=1740  
min=seg/60  
min
```

```
## [1] 29
```

- El área de un triángulo se define como la base por la altura partido entre 2. Si un triángulo tiene una base de 5cm y una altura de 10cm, ¿cuál es su área?

```
#Ejemplo de fórmulas  
base=5  
altura=10  
area=(base*altura)/2  
area
```

```
## [1] 25
```

- El índice de masa corporal de una persona se calcula como el peso partido por la altura en metros al cuadrado. Si una persona mide 90kg y mide 180cm, ¿cuál es su índice de masa corporal?

```
#Ejemplo de fórmulas  
peso=90  
altura.cm=180  
altura.m=altura.cm/100  
imc=peso/(altura.m^2)  
imc
```

```
## [1] 27.77778
```

## 1.2.5 Vectores en R

Hasta el momento, se han ido originando y manipulando variables que tomaban un único valor. Sin embargo, lo habitual en la práctica es trabajar con una base de datos en las que se disponga de varias observaciones por variable y no de un único valor. Extendiendo el ejemplo del apartado anterior donde se determinó el índice de masa corporal de una persona, se puede pensar que el objetivo sea determinar, por ejemplo, el *'imc'* de 10 personas. Realizar todo el proceso anterior de uno en uno es bastante tedioso a la par de lento e ineficiente. Como cabría esperar, R es capaz de realizar toda la serie de operaciones simultáneamente mediante la definición de **vectores de variables**.

Un vector en R es una colección de uno o más datos del mismo tipo. Es importante recalcar que un vector solo puede estar formado por elementos de un mismo tipo (numérico, texto o lógico), de manera que no es posible mezclar datos de diferentes tipos dentro de ellos. En caso de crear un vector formado por elementos numéricos (o lógicos) y cadenas de texto, el programa no reconocería los elementos numéricos (o lógicos) como tal y los transformaría a datos de tipo texto (más adelante se explicarán los distintos tipos de datos en R). Aunque hay diversas maneras de crear un vector en R, la forma más común es a través de la función **c()**, denominada función *concatenar*. Esta función combina (concatena) todos los elementos que recibe como argumentos. Para crear un vector debemos introducir todos los elementos dentro del paréntesis y **separados por coma**, es decir, `c(argumento1, argumento2, argumento3, ...)`. En caso de que los elementos sean cadenas de texto, cada elemento a su vez debe ser introducido entre **comillas** es decir, `"argumento1"`. En cambio, si el objetivo es formar un vector con elementos lógicos basta con utilizar T (true/verdadero) o F (false/falso), es decir, `c(T,F,T,T, ...)`.

En este tipo de estructura de datos se almacenan las observaciones de la base de datos. A continuación se muestran algunos ejemplos de vectores en R. Mostrar especial atención al último ejemplo donde se han mezclado datos numéricos y lógicos con datos de tipo texto y reflexionar sobre lo que sucede.

```
#Ejemplo de creación de vectores
peso=c(60,40.50,47.80,80,56.75)
peso
```

```
## [1] 60.00 40.50 47.80 80.00 56.75
```

```
ciudad=c("Granada","Granada","Cádiz","Gran Canarias")
ciudad
```

```
## [1] "Granada"      "Granada"      "Cádiz"        "Gran Canarias"
```

```
logico=c(T,F,T,F,F,T,T,T)
logico
```

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

```
mixto=c(80,T,"Granada",2,"Málaga")
mixto
```

```
## [1] "80"      "TRUE"    "Granada" "2"      "Málaga"
```

También se pueden crear vectores que son combinación de vectores.

```
#Ejemplo de creación de vectores
peso.niños=c(45.25,47.35,55.25,51)
peso_niñas=c(46.45,40.23,44.50,52,51.47)
peso=c(peso.niños,peso_niñas)
peso
```

```
## [1] 45.25 47.35 55.25 51.00 46.45 40.23 44.50 52.00 51.47
```

Por otro lado, se pueden crear secuencias numéricas mediante el símbolo ‘:’. Estas secuencias son consecutivas con incrementos o decrementos de 1 y pueden empezar con cualquier número, incluso si éste es negativo o tiene cifras decimales. Si el primer número de la secuencia tiene cifras decimales, estas serán respetadas al hacer los incrementos o decrementos. En contraste, si el último número es el que tiene cifras decimales, éste será redondeado.

```
#Ejemplo de creación de vectores
1:10
```

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

```
1:-10
```

```
## [1] 1 0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -10
```

```
2.25:6
```

```
## [1] 2.25 3.25 4.25 5.25
```

```
-12:0.75
```

```
## [1] -12 -11 -10 -9 -8 -7 -6 -5 -4 -3 -2 -1 0
```

En caso de querer realizar secuencias numéricas cuyos incrementos o decrementos no sean de una unidad, se puede utilizar la función `seq(elemento1,elemento2,elemento3)` donde *elemento1* hace referencia al inicio de la serie, *elemento2* representa el número final de la serie y *elemento3* denota la cantidad de incremento o decremento de la serie.

```
#Ejemplo de creación de vectores
seq(1,10,2)
```

```
## [1] 1 3 5 7 9
```

```
seq(1,-10,-2)
```

```
## [1] 1 -1 -3 -5 -7 -9
```

```
seq(-5.5,3,3)
```

```
## [1] -5.5 -2.5 0.5
```

```
seq(3,10.5,2)
```

```
## [1] 3 5 7 9
```

Finalmente, otra función muy utilizada es **rep(x,veces)**, que crea un vector donde aparece un elemento repetido tantas veces como se indique. Sus parámetros son el elemento x a replicar y el número de veces.

```
#Ejemplo de creación de vectores
```

```
rep("HOLA",5)
```

```
## [1] "HOLA" "HOLA" "HOLA" "HOLA" "HOLA"
```

```
rep(2,3)
```

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

## 1.2.6 Operaciones con vectores

Al igual que sucedía con las variables a las que se les asignaban un único valor, también se pueden realizar distintas operaciones con vectores de variables en R. La opción más sencilla sería realizar operaciones con ellas mismas. Las funciones más usuales con vectores son las que siguen

Función	Significado
+ - * /	Sumar/restar/multiplicar/dividir cada elemento del vector por una constante
exp(x)	Exponencial de todos los elementos del vector x
sqrt(x)	Raíz cuadrada de todos los elementos del vector x
^	Potencia de todos los elementos del vector x
length(x)	Longitud del vector (x)
sum(x)	Suma de todos los elementos del vector x
prod(x)	Producto de todos los elementos del vector x
max(x)/min(x)	Máximo/mínimo del vector x
cumsum(x)	Suma acumulada de los componentes del vector x
cumprod(x)	Producto acumulado de los componentes del vector x
diff(x)	Diferencias entre cada componente y su anterior
sort(x)	Ordena el vector de forma creciente
head(x)	Muestra los 6 primeros valores del vector
tail(x)	Muestra los 6 últimos valores del vector
rev(x)	Invierte el orden de las componentes del vector
unique(x)	Elementos sin repetir de los componentes del vector

```
#Operaciones con vectores
```

```
x=c(1,4,6,2,7,9,4,2,3,1,1,5,9)
```

```
x+2
```

```
## [1] 3 6 8 4 9 11 6 4 5 3 3 7 11
```

```
x^2
```

```
## [1] 1 16 36 4 49 81 16 4 9 1 1 25 81
```

```
sum(x)
```

```
## [1] 54
```

```
max(x)
```

```
## [1] 9
```

```
sort(x,decreasing=F)
```

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

```
sort(x,decreasing=T)
```

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

```
head(x)
```

```
## [1] 1 4 6 2 7 9
```

```
length(x)
```

```
## [1] 13
```

```
unique(x)
```

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

También se pueden realizar fórmulas con los vectores. Las operaciones con vectores se efectúan componente a componente, de modo que si se opera con vectores que poseen longitudes distintas, R devolverá un error (aunque el resultado será un vector con longitud la del más largo). Por ejemplo, un vector `x` que contiene los números 1,2,3,4 sumado a otro que contiene los números 1,2,3,4,5 devuelve el error *'longitud de objeto mayor no es múltiplo de la longitud de uno menor'* y mostraría el resultado 2,4,6,8,6.

A continuación se amplían los ejemplos realizados en el apartado '1.2.4 Fórmulas con variables' y se implementan nuevos.

```
x=c(1,5,6)
y=c(4,6,9)
x+y
```

```
## [1] 5 11 15
```

```
x*y
```

```
## [1] 4 30 54
```

```
seg=c(1740,14,60,180,1460)
min=seg/60
min
```

```
## [1] 29.0000000 0.2333333 1.0000000 3.0000000 24.3333333
```

```
base=c(5,17,7)
altura=c(10,4.5,9)
area=(base*altura)/2
area
```

```
## [1] 25.00 38.25 31.50
```

```
peso=c(90,55,65,80)
altura.cm=c(180,175,170,169)
altura.m=altura.cm/100
imc=peso/(altura.m^2)
imc
```

```
## [1] 27.77778 17.95918 22.49135 28.01022
```

Por último, es posible acceder a un elemento concreto de un vector u obtener un subvector del mismo. Para acceder a cualquiera de los elementos del vector basta con colocar, junto al nombre del vector asignado, la posición que ocupa entre corchetes. Por el contrario, para mostrar varios elementos a la vez (subvectores) habrá que utilizar la función *concatenar* y, dentro de los paréntesis de esta función, indicar las posiciones que se desean mostrar separadas por coma. Otra opción es utilizar `:` para mostrar una secuencia de valores. A continuación se ilustran algunos ejemplos.

```
#Ejemplo de acceso a elementos y subvectores
peso=c(47,45.25,85.10,65.21,50,53.40)
peso[2] #Observación del segundo individuo
```

```
## [1] 45.25
```

```
peso[c(1,2,5)] #Observación del 1º, 2º y 5º individuo
```

```
## [1] 47.00 45.25 50.00
```

```
peso[2:4] #Observación del 2º al 4º individuo
```

```
## [1] 45.25 85.10 65.21
```

```
ciudad=c("Madrid","Barcelona","Almería","Albacete")
ciudad[3]
```

```
## [1] "Almería"
```

```
ciudad[c(1,4)]
```

```
## [1] "Madrid" "Albacete"
```

```
fumador=c(T,T,T,F,F,T,F,F,T)
```

```
fumador[6]
```

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

```
fumador[1:5]
```

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

## 1.3 Archivo de datos en R

Hasta el momento se ha introducido la forma de definir variables y vectores en R que contengan valores o resultados de aplicar fórmulas entre los objetos mencionados. Estos dos objetos representan la manera más sencilla de guardar información en R pero carecen de interés cuando se dispone de una gran cantidad de información. En esta situación conviene organizar la información en estructuras de datos que permitan el manejo y la interpretación de forma más amigable. Las estructuras de datos más utilizadas en R son:

- **Matrices y arrays.** Estos objetos son la extensión de vectores a dos y tres dimensiones respectivamente. Al igual que sucedía con los vectores, las matrices y arrays solo pueden contener datos de un sólo tipo. Para definir una matriz hay que recurrir a la función `matrix()` e introducir los datos en el primer argumento y establecer el número de columnas y filas, es decir, `matrix(data=c(elemento1,elemento2,...),nrow=,ncol=,byrow=FALSE)`. El elemento `byrow` indica si la matriz se irá rellenando por filas (`byrow=TRUE`) o por columnas (`byrow=FALSE`). No se procede a enseñar la estructura de los `arrays` porque es un tipo de estructura que no se suele utilizar con frecuencia y se escapa de los objetivos del presente curso. Si el lector está interesado, se le anima a utilizar el **foro del curso** para solicitar más material.
- **Dataframe.** Esta estructura de datos de dos dimensiones (filas y columnas) es la más utilizada dentro del ámbito de análisis de datos en R. Básicamente, se trata de una tabla de datos en donde las filas son los casos y las columnas las variables. La principal ventaja de los `dataframe` es que permiten la libertad de definir datos de distintos tipos (siempre por columnas), por lo que pueden ser vistos como un tipo de matriz pero más flexibles. En cambio, su ‘única’ desventaja es que todas las columnas deben tener la misma cantidad de elementos. Para definir un `dataframe` hay que recurrir a la función `data.frame()` e introducir las variables separadas por comas, es decir, `data.frame(variable1,variable2,...)`.
- **Listas.** Es la estructura de datos más flexible puesto que permite incluir variables de tipos y longitudes diferentes. En consecuencia, los `dataframes` son un caso particular de las `listas`. Para crear una lista se opera de la misma manera que en el caso de los `dataframes` pero utilizando la función `list()`. En este tipo de estructura tampoco se va a profundizar y se anima al lector a indagar por la red, o pedir material correspondiente para ello. Las `listas`, a diferencia de los `arrays`, sí son utilizadas en R con mucha frecuencia.

### 1.3.1 Almacenamiento de datos en R mediante dataframes

Como se ha comentado, una hoja de datos, cuadro de datos o `dataframe` es una lista de objetos no necesariamente del mismo tipo que se utilizan para el análisis de datos. Es la estructura de datos más importante en R, ya que permite representar la información de manera tabular (cada fila representa la observación de un

sujeto y cada columna una variable) y su manejo es sencillo e intuitivo. Por tanto, debido a que es habitual trabajar con *dataframes* en la práctica, es de suma importancia conocer y entender su funcionamiento.

Con el fin de asentar los conocimientos acerca del almacenamiento y manipulación de una base de datos en R, se va a utilizar la siguiente tabla que alberga la información sobre 7 pacientes de una consulta.

**Nota:** Aunque R permite crear bases de datos y la posterior manipulación de las mismas, es importante tener en cuenta en todo momento que no es su principal virtud. Para estos fines, es recomendable utilizar otros programas como, por ejemplo, Excel. La principal fortaleza de R es la potencia para realizar múltiples análisis estadísticos avanzados.

Nombre	Edad	Peso	Altura	Fumador	Fecha
Carlos	26	64.75	1.80	No	02-03-2020
María	32	55.25	1.65	Sí	05-10-2020
Sara	29	61.20	1.71	Sí	27-09-2020
Laura	26	58.75	1.62	No	16-09-2020
Jose	27	63.30	1.71	Sí	13-07-2020
Pablo	25	59.80	1.83	Sí	01-10-2020
Elena	27	51.30	1.63	No	15-05-2020

El primer paso sería definir las variables que conforman la base de datos y almacenar la información en el correspondiente vector.

```
nombre=c("Carlos","María","Sara","Laura","Jose","Pablo","Elena")
edad=c(26,32,29,26,27,25,27)
peso=c(64.75,55.25,61.20,58.75,63.30,59.80,51.30)
altura=c(1.80,1.65,1.71,1.62,1.71,1.83,1.63)
fumador=c("No","Sí","Sí","No","Sí","Sí","No")
fecha=c("02-03-2020","05-10-2020","27-09-2020","16-09-2020",
        "13-07-2020","01-10-2020","15-05-2020")
```

En segundo lugar se hace uso de la función `data.frame()`. La base de datos se puede guardar en un objeto si se le asigna un nombre, en este caso, *base*.

```
base=data.frame(nombre,edad,peso,altura,fumador,fecha)
base
```

```
##  nombre edad  peso altura fumador      fecha
## 1 Carlos   26 64.75   1.80      No 02-03-2020
## 2 María   32 55.25   1.65      Sí 05-10-2020
## 3 Sara    29 61.20   1.71      Sí 27-09-2020
## 4 Laura   26 58.75   1.62      No 16-09-2020
## 5 Jose    27 63.30   1.71      Sí 13-07-2020
## 6 Pablo   25 59.80   1.83      Sí 01-10-2020
## 7 Elena   27 51.30   1.63      No 15-05-2020
```

Una vez definida la base de datos se puede realizar una serie de manipulaciones y/o operaciones con ella. Una de las principales es nombrar las filas y columnas mediante las funciones `rownames()` y `colnames()` junto a `c()`, que listará los nombres deseados.

```
rownames(base)=c("Paciente1","Paciente2","Paciente3","Paciente4","Paciente5","Paciente6",
                 "Paciente7")
colnames(base)=c("Nombre","Edad","Peso_kg","Altura_m","Fumador","Fecha_Consulta")
base
```



```
##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta
## Paciente1 Carlos   26   64.75    1.80      No      02-03-2020
## Paciente2 María    32   55.25    1.65      Sí      05-10-2020
## Paciente3 Sara     29   61.20    1.71      Sí      27-09-2020
## Paciente4 Laura    26   58.75    1.62      No      16-09-2020
## Paciente5 Jose     27   63.30    1.71      Sí      13-07-2020
## Paciente6 Pablo    25   59.80    1.83      Sí      01-10-2020
## Paciente7 Elena    27   51.30    1.63      No      15-05-2020
```

Si se quiere acceder a la información de una o varias filas (columnas) se actúa de manera similar a cuando se está trabajando con vectores, pero con la salvedad de que en esta ocasión se añade una coma al final (principio) de la función *concatenar*. En lo que sigue figuran algunos ejemplos de selección de filas y columnas de un *dataframe*.

```
pac2=base[2,] #Información del 2º paciente
pac2
```

```
##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta
## Paciente2 María    32   55.25    1.65      Sí      05-10-2020
```

```
hombres=base[c(1,5,6),] #Información de los hombres
hombres
```

```
##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta
## Paciente1 Carlos   26   64.75    1.80      No      02-03-2020
## Paciente5 Jose     27   63.30    1.71      Sí      13-07-2020
## Paciente6 Pablo    25   59.80    1.83      Sí      01-10-2020
```

```
fisio=base[,c(3,4)] #Información de las variables fisiológicas
fisio
```

```
##           Peso_kg Altura_m
## Paciente1   64.75    1.80
## Paciente2   55.25    1.65
## Paciente3   61.20    1.71
## Paciente4   58.75    1.62
## Paciente5   63.30    1.71
## Paciente6   59.80    1.83
## Paciente7   51.30    1.63
```

En uno de los ejemplos anteriores, se seleccionaron a todos los hombres de la base de datos de manera manual. Esta operación puede resultar ser un trabajo tedioso e ineficiente si la base de datos dispone de un número elevado de observaciones. En tales casos, se puede escoger subconjuntos del *dataframe* utilizando la función *subset()* y crear nuevos objetos. En esta función hay que indicar el *dataframe* con el que se está trabajando y la condición que tienen que cumplir los individuos para ser seleccionados. No obstante, en la base de datos considerada no existe ninguna variable que indique el sexo del paciente, por lo que, previamente, habría que definir dicha variable y añadirla a la base de datos. Para indexar una nueva variable al fichero se puede utilizar una de las dos siguientes vertientes:

- A través de la función *cbind()*. Esta función permite combinar vectores, matrices o *dataframes* por columnas. Todos los objetos introducidos en la función deben poseer el mismo número de filas, en caso contrario los elementos de los objetos con menor longitud serán repetidos hasta completar la dimensión del objeto con mayor longitud.

- Introducir la nueva variable mediante la orden `nombre.dataframe$nombre.variable.dataframe=nombre.variable`. El símbolo `$` es otra alternativa disponible en R para seleccionar columnas de un *dataframe*. Va situado entre el nombre del objeto y el nombre de la variable. Por otro lado, la única diferencia entre ‘`nombre.variable.dataframe`’ y ‘`nombre.variable`’ es el nombre que va a aparecer en el fichero de datos y que puede ser diferente al nombre de la variable original creada. Por ejemplo, se puede crear la variable *sexo* todo en minúscula, y querer introducirla en la base de datos con la primera letra en mayúscula (ver el código que figura a continuación) sin necesidad de tener que volver a utilizar la función *colnames*. Con la función *cbind* no hay opción de alterar el nombre directamente; en dicho caso sí habría que recurrir a la función *colnames*.

```
sexo=c("H", "M", "M", "M", "H", "H", "M")
cbind(base, sexo) #Ejemplo del uso de la función cbind
```

```
##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta sexo
## Paciente1 Carlos   26  64.75    1.80      No      02-03-2020    H
## Paciente2 María    32  55.25    1.65     Sí      05-10-2020    M
## Paciente3 Sara     29  61.20    1.71     Sí      27-09-2020    M
## Paciente4 Laura    26  58.75    1.62     No      16-09-2020    M
## Paciente5 Jose     27  63.30    1.71     Sí      13-07-2020    H
## Paciente6 Pablo    25  59.80    1.83     Sí      01-10-2020    H
## Paciente7 Elena    27  51.30    1.63     No      15-05-2020    M
```

```
base$Sexo=sexo
base
```

```
##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta Sexo
## Paciente1 Carlos   26  64.75    1.80      No      02-03-2020    H
## Paciente2 María    32  55.25    1.65     Sí      05-10-2020    M
## Paciente3 Sara     29  61.20    1.71     Sí      27-09-2020    M
## Paciente4 Laura    26  58.75    1.62     No      16-09-2020    M
## Paciente5 Jose     27  63.30    1.71     Sí      13-07-2020    H
## Paciente6 Pablo    25  59.80    1.83     Sí      01-10-2020    H
## Paciente7 Elena    27  51.30    1.63     No      15-05-2020    M
```

Una vez creada la variable *sexo* e indexada en la base de datos (notar que, si se usa la función *cbind*, habría que asignarle un nombre al objeto correspondiente; en el ejemplo **NO** se ha hecho con la función *cbind*), se procedería con la selección. Sin embargo, antes de este paso, es importante saber cómo se “llama” a las variables disponibles en un *dataframe*. Para ello, existen dos posibilidades:

- A través del uso del símbolo `$` entre el nombre del *dataframe* y el nombre de la variable.
- Mediante la función **attach**. Permite referenciar los nombres de las columnas de los *dataframes*, sin necesidad de especificar el nombre del *dataframe* precedido del símbolo `$`, lo que agiliza su manipulación. La función **detach** lo desactiva.

Figura 5 muestra el uso del símbolo `$`. Se puede observar que cuando no se usa este símbolo, R proporciona un mensaje de error. En cambio, si se utiliza la función `attach(nombre.dataframe)`, ya no es necesario indicar previamente el nombre de la base de datos (ver Figura 6).

De aquí en adelante y por su comodidad se utilizará la función **attach**. A continuación, y siguiendo con el objetivo planteado, se selecciona a los hombres con la función *subset*. Asimismo, al fin de ilustrar una opción numérica, también se procede a seleccionar a todos los pacientes cuyo peso sea superior a 60 kg.

```

Console Terminal x R Markdown x Jobs x
C:/PID Medicina/Práctica 1/ ↗
> base$Nombre
[1] "Carlos" "María" "Sara" "Laura" "Jose" "Pablo" "Elena"
> Nombre
Error: objeto 'Nombre' no encontrado

```

Figure 5: Uso del símbolo \$

```

Console Terminal x R Markdown x Jobs x
C:/PID Medicina/Práctica 1/ ↗
> attach(base)
> Nombre
[1] "Carlos" "María" "Sara" "Laura" "Jose" "Pablo" "Elena"
> detach(base)
> Nombre
Error: objeto 'Nombre' no encontrado

```

Figure 6: Uso de las funciones attach y detach

```

attach(base)
hombres=subset(base,Sexo=="H")
hombres

##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta Sexo
## Paciente1 Carlos   26   64.75    1.80      No      02-03-2020    H
## Paciente5  Jose    27   63.30    1.71     Sí      13-07-2020    H
## Paciente6 Pablo    25   59.80    1.83     Sí      01-10-2020    H

peso.sup.60=subset(base,Peso_kg>60)
peso.sup.60

```

```

##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta Sexo
## Paciente1 Carlos   26   64.75    1.80      No      02-03-2020    H
## Paciente3 Sara    29   61.20    1.71     Sí      27-09-2020    M
## Paciente5 Jose    27   63.30    1.71     Sí      13-07-2020    H

```

Se hace hincapié que para hacer referencia al ‘igual’ (se ha utilizado en el ejemplo de los hombres), en R hay que emplear doble signo, es decir, ==. En cambio, para indicar ‘mayor o igual’, o bien, ‘menor o igual’, bastaría con usar >= y <= respectivamente, mientras que el operador ‘diferente’ se aplicaría con !=.

Por otro lado, también puede eliminarse cualquier observación/variable utilizando el número de fila/columna que ocupa. Si se quisiera eliminar al individuo i, bastaría adherir al nombre del dataframe la orden [-i,], mientras que para eliminar la variable j, la orden sería [,~j]. En caso de querer eliminar más de una observación/variable se combina las órdenes anteriores con la función *concatenar*.

```

base[-5,] #Se eliminaría al 5º individuo

##           Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta Sexo
## Paciente1 Carlos   26   64.75    1.80      No      02-03-2020    H

```

```
## Paciente2 María 32 55.25 1.65 Sí 05-10-2020 M
## Paciente3 Sara 29 61.20 1.71 Sí 27-09-2020 M
## Paciente4 Laura 26 58.75 1.62 No 16-09-2020 M
## Paciente6 Pablo 25 59.80 1.83 Sí 01-10-2020 H
## Paciente7 Elena 27 51.30 1.63 No 15-05-2020 M
```

```
base[,-1] #Se eliminaría la 1º variable
```

```
##          Edad Peso_kg Altura_m Fumador Fecha_Consulta Sexo
## Paciente1 26 64.75 1.80 No 02-03-2020 H
## Paciente2 32 55.25 1.65 Sí 05-10-2020 M
## Paciente3 29 61.20 1.71 Sí 27-09-2020 M
## Paciente4 26 58.75 1.62 No 16-09-2020 M
## Paciente5 27 63.30 1.71 Sí 13-07-2020 H
## Paciente6 25 59.80 1.83 Sí 01-10-2020 H
## Paciente7 27 51.30 1.63 No 15-05-2020 M
```

```
base[-c(1,2,3,4,7),] #Se eliminaría al 1º,2º,3º,4º y 7º individuo
```

```
##          Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta Sexo
## Paciente5 Jose 27 63.3 1.71 Sí 13-07-2020 H
## Paciente6 Pablo 25 59.8 1.83 Sí 01-10-2020 H
```

```
base[,-c(5,6)] #Se eliminaría la 5º y 6º variable
```

```
##          Nombre Edad Peso_kg Altura_m Sexo
## Paciente1 Carlos 26 64.75 1.80 H
## Paciente2 María 32 55.25 1.65 M
## Paciente3 Sara 29 61.20 1.71 M
## Paciente4 Laura 26 58.75 1.62 M
## Paciente5 Jose 27 63.30 1.71 H
## Paciente6 Pablo 25 59.80 1.83 H
## Paciente7 Elena 27 51.30 1.63 M
```

Otra opción es chequear el valor del individuo  $i$  en la variable  $j$ . Para esta operación se ejecuta el nombre del *dataframe* seguido de la orden  $[i,j]$ . Asimismo, si una vez definido el *dataframe* el usuario avista que un valor está mal codificado, puede cambiarlo en la definición de las variables al comienzo del *script* o bien, ejecutar el comando `nombre.dataframe[i,j]=valor`. Para ilustrar lo comentado en este párrafo, se puede observar que la edad inicial registrada para el 2º paciente es 32 años, pero en realidad se debe a un error de codificación y resulta que tiene 31 años, por lo que se procede a modificarlo.

```
base[2,2] #Edad del 2º paciente
```

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

```
base[2,2]=31 #Se asigna el nuevo valor
```

```
base
```

```
##          Nombre Edad Peso_kg Altura_m Fumador Fecha_Consulta Sexo
## Paciente1 Carlos 26 64.75 1.80 No 02-03-2020 H
```

```
## Paciente2 María 31 55.25 1.65 Sí 05-10-2020 M
## Paciente3 Sara 29 61.20 1.71 Sí 27-09-2020 M
## Paciente4 Laura 26 58.75 1.62 No 16-09-2020 M
## Paciente5 Jose 27 63.30 1.71 Sí 13-07-2020 H
## Paciente6 Pablo 25 59.80 1.83 Sí 01-10-2020 H
## Paciente7 Elena 27 51.30 1.63 No 15-05-2020 M
```

Finalmente, si se detectara algún error en el orden de la introducción de las variables y/o individuos, se podría modificar la posición de registro de los individuos y/o variables. Por ejemplo, a continuación se va a cambiar la posición del cuarto individuo a la última posición y la variable fumador se va a situar en la segunda posición. Notar que aunque se altere el orden de las filas, el nombre que tiene asignado dichas filas no se modifica. Véase como el cuarto paciente que ha sido desplazado a la última posición mantiene el nombre “Paciente4”. Aquí podría ser interesante renombrar todas las filas de nuevo con el fin de reestablecer el nombre según el orden de aparición.

```
base=base[c(1:3,5:7,4),]
base=base[,c(1,5,2:4,6)]
base
```

```
##      Nombre Fumador Edad Peso_kg Altura_m Fecha_Consulta
## Paciente1 Carlos      No  26  64.75    1.80    02-03-2020
## Paciente2 María      Sí  31  55.25    1.65    05-10-2020
## Paciente3 Sara      Sí  29  61.20    1.71    27-09-2020
## Paciente5 Jose      Sí  27  63.30    1.71    13-07-2020
## Paciente6 Pablo      Sí  25  59.80    1.83    01-10-2020
## Paciente7 Elena      No  27  51.30    1.63    15-05-2020
## Paciente4 Laura      No  26  58.75    1.62    16-09-2020
```

### 1.3.2 Tipos de datos en R

R permite trabajar con datos de distintos tipos. Los más comunes son:

- **Numeric:** Son datos que pueden contener números. A su vez, los datos numéricos se dividen en:
  - *Integer:* datos de tipo entero.
  - *Double:* datos de tipo real. Es el que R considera por defecto, es decir, cuando se define un número en R, éste será de tipo double.
  - *Complejo:* datos de tipo complejo.
- **Character:** Son datos que permiten introducir números y cualquier tipo de carácter.
- **Logical:** Son datos que sólo permite los valores lógicos de verdadero (T) o falso (F).
- **Factor:** Se usan para trabajar con variables categóricas que tienen un conjunto fijo y conocido de modalidades posibles. Son vectores de caracteres con atributos de clase y de nivel que determinan qué cadenas pueden ser introducidas en un vector. Su principal utilidad es que las variables cualitativas definidas como factores (esto no ocurre si se definen como de tipo character) pueden ser utilizadas en análisis posteriores. Sin embargo, no se pueden emplear en operaciones aritméticas ni como funciones en las operaciones entre datos numéricos. Se originan mediante `factor()`, y sus parámetros son:
  - *x:* vector de datos.
  - *levels:* vector de niveles (es opcional); si en el vector de datos aparece un dato no especificado en *levels* se considerará faltante. Los datos faltantes en R son denotados como **NA**.
  - *labels:* vector de etiquetas para los niveles del factor (es opcional).
  - *exclude:* vector de valores a excluir (es opcional); si en el vector de datos existe un dato que también aparece en *exclude* se considerará faltante.

- *ordered*: determina si los niveles del factor están ordenados (*ordered=TRUE*) o no (*ordered=FALSE*). Por defecto R toma *ordered=FALSE*.

```
curso=c("1º","1º","3º","2º","2º")
curso
```

```
## [1] "1º" "1º" "3º" "2º" "2º"
```

```
curso=factor(c("1º","1º","3º","2º","2º"))
curso
```

```
## [1] 1º 1º 3º 2º 2º
## Levels: 1º 2º 3º
```

```
curso=factor(c("1º","1º","3º","2º","2º"),levels=c("1º","2º","3º","4º"))
curso
```

```
## [1] 1º 1º 3º 2º 2º
## Levels: 1º 2º 3º 4º
```

```
curso=factor(c("1º","1º","3º","2º","2º"),levels=c("1º","2º","3º","4º"),labels=c("Primero",
"Segundo","Tercero","Cuarto"))
curso
```

```
## [1] Primero Primero Tercero Segundo Segundo
## Levels: Primero Segundo Tercero Cuarto
```

```
curso=factor(c("1º","1º","3º","2º","2º"),levels=c("1º","2º","3º","4º"),labels=c("Primero",
"Segundo","Tercero","Cuarto"),ordered=TRUE)
curso
```

```
## [1] Primero Primero Tercero Segundo Segundo
## Levels: Primero < Segundo < Tercero < Cuarto
```

- **Fecha:** Son datos que representan fechas de calendario. La mejor opción para tratar con este tipo de datos, es crear las fechas en formato character y posteriormente transformarlo a tipo *fecha* mediante la función **as.Date**. El estándar utilizado por R para las fechas es yyyy-mm-dd, aunque dicho formato puede ser alterado. Los formatos de fechas que es capaz R de leer son los siguientes:

Símbolo	Significado
%d	día (numérico, de 0 a 31)
%a	día de la semana abreviado a tres letras
%A	día de la semana (nombre completo)
%m	mes (numérico de 0 a 12)
%b	mes (nombre abreviado a tres letras)
%B	mes (nombre completo)
%y	año (con dos dígitos)
%Y	año (con cuatro dígitos)

Por ejemplo, en la base de datos creada al comienzo del apartado 1.3.1 Almacenamiento de datos en R mediante dataframes, se introdujo la variable *fecha* de tipo character con formato dd-mm-yyyy. Al ser un formato distinto al predeterminado (no es yyyy-mm-dd), hay que especificar el formato utilizado en la función **as.Date** como sigue:

```
fecha=c("02-03-2020", "05-10-2020", "27-09-2020", "16-09-2020",
        "13-07-2020", "01-10-2020", "15-05-2020")
fecha=as.Date(fecha, format="%d-%m-%Y")
fecha
```

```
## [1] "2020-03-02" "2020-10-05" "2020-09-27" "2020-09-16" "2020-07-13"
## [6] "2020-10-01" "2020-05-15"
```

Llama la atención que R transforma automáticamente el objeto al formato estándar. Si en lugar de haber utilizado el guión como símbolo de separación se hubiera utilizado, por ejemplo, la barra ("/"), habría que especificar dentro de la función el formato con la barra como `format="%d/%m/%Y"`, aunque de nuevo R lo transformará al formato predeterminado.

```
fecha=c("02/03/2020", "05/10/2020", "27/09/2020", "16/09/2020",
        "13/07/2020", "01/10/2020", "15/05/2020")
fecha=as.Date(fecha, format="%d/%m/%Y")
fecha
```

```
## [1] "2020-03-02" "2020-10-05" "2020-09-27" "2020-09-16" "2020-07-13"
## [6] "2020-10-01" "2020-05-15"
```

No obstante, si no se desea trabajar con el formato que establece R por defecto, se puede modificar la forma de la fecha utilizando la función **format** y guardando la salida en un nuevo objeto. La desventaja de esta función es que el objeto resultante no será de tipo fecha. A continuación figuran distintos ejemplos que no serán guardados en un nuevo objeto porque no se van a considerar más tarde.

```
format(fecha, "%d %B %Y")
```

```
## [1] "02 marzo 2020"      "05 octubre 2020"    "27 septiembre 2020"
## [4] "16 septiembre 2020" "13 julio 2020"     "01 octubre 2020"
## [7] "15 mayo 2020"
```

```
format(fecha, "%d %b %y")
```

```
## [1] "02 mar. 20" "05 oct. 20" "27 sep. 20" "16 sep. 20" "13 jul. 20"
## [6] "01 oct. 20" "15 may. 20"
```

```
format(fecha, "%A %b %y")
```

```
## [1] "lunes mar. 20"      "lunes oct. 20"     "domingo sep. 20"
## [4] "miércoles sep. 20" "lunes jul. 20"     "jueves oct. 20"
## [7] "viernes may. 20"
```

Finalmente, otro aspecto muy importante a tener presente es que cuando se crean variables de tipo character y posteriormente se introducen en un *dataframe*, R transforma automáticamente el formato de estas variables a tipo factor dentro del *dataframe*. Por ejemplo, en el apartado anterior donde se definió la variable *Fumador* como de tipo character, ésta perdió su naturaleza dentro del *dataframe* y adoptó la tipología de factor. En consecuencia, la variable *fumador* sí es de tipo character pero *base\$Fumador* es de tipo factor. Por el contrario, esto no ocurre con las variables tipo fecha. Si una variable es originalmente creada como character y posteriormente es transformada a tipo fecha, esta variable mantiene su naturaleza de tipo fecha cuando es introducida al *dataframe*.

Hay situaciones en las que el usuario no sabe con qué tipo de datos está tratando. Para estas situaciones se recurre a la función `class()` que devuelve la naturaleza del objeto que se introduce. A continuación figuran algunos ejemplos, incluidos el comentado en el párrafo anterior.

```
class(5)
```

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

```
class("HOLA")
```

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

```
class(T)
```

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

```
class(base)
```

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

```
class(fumador)
```

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

```
class(base$Fumador)
```

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

```
class(edad)
```

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

```
class(base$Edad)
```

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



### 1.3.3 Importación de un fichero de datos en R

El manejo de datos es uno de los pilares de R siendo relevante la inserción de los mismos. Como se ha visto al comienzo de este tema, una posibilidad de introducir los datos es teclearlos en el mismo programa a través de la creación de vectores, matrices, listas, etc. donde ir almacenándolos. No obstante, si la base de datos es demasiado larga es muy habitual registrar los datos en otro programa (Excel, bloc de notas, SPSS, ...) cuyo interfaz sea más intuitivo y facilite la codificación de los mismos. En tal caso hay que trasladar a R los datos que se han registrado en otro programa, ya sea desde un archivo de texto o desde otro tipo de ficheros (la extensión depende del programa).

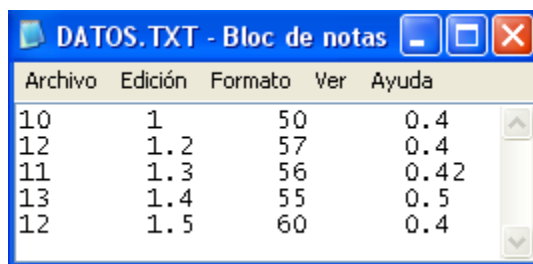
El primer paso sería indicar el directorio de trabajo donde se encuentra el fichero que contiene los datos. Para ello, desde RStudio se cuenta con dos opciones: utilizar la función `setwd()` e indicar la dirección de la carpeta entre comillas, o desde el menú **Session** situado en la **Barra del menú principal** y seguir la ruta **Session** → **Set Working Directory** → **Choose Directory...** y elegir la carpeta donde se encuentre el fichero.

Si se tiene un archivo de texto (formatos `.txt`, `.dat`, `.csv`, etc.) se hará uso de la función `read.csv()`, en la cual puede especificarse entre otras funciones:

- El nombre del fichero entre comillas seguido de la extensión utilizada, es decir, si el fichero es `.txt` se debe indicar `"nombre.fichero.txt"`.
- Si el fichero tiene encabezado (`header=TRUE`) o no (`header=FALSE`).
- El tipo de separador utilizado en el registro de los datos. Aunque hay diversas formas, los más habituales son: `sep=","` si se usaron comas, `sep=" "` si están separados por espacios o `sep="\t"` si se utilizó el tabulador.
- El nombre de las filas (`row.names`) y/o de las columnas (`col.names`).

En lugar de `read.csv()` puede utilizarse `read.table()`; la diferencia estriba en que el primero lleva por defecto leer valores separados por coma (aunque ya se ha indicado que puede cambiarse), lo cual es usual en muchos países donde se utiliza la coma como punto decimal y el punto y coma de separador de campo.

Por ejemplo, si se almacena el fichero DATOS.TXT que aparece en la Figura 7,



Archivo	Edición	Formato	Ver	Ayuda
10	1	50	0.4	
12	1.2	57	0.4	
11	1.3	56	0.42	
13	1.4	55	0.5	
12	1.5	60	0.4	

Figure 7: Fichero de datos separados con tabulaciones y sin nombres en las columnas

podría importarse a R mediante la secuencia:

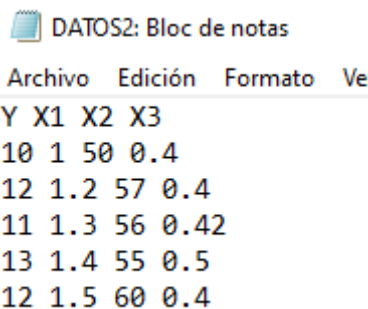
```
ejemplo=read.csv("DATOS.txt",header=FALSE,sep="\t")
ejemplo
```

```
##   V1  V2 V3  V4
## 1 10  1.0 50 0.40
## 2 12  1.2 57 0.40
## 3 11  1.3 56 0.42
## 4 13  1.4 55 0.50
## 5 12  1.5 60 0.40
```

El fichero de la Figura 7 fue registrado mediante tabulaciones y sin asignarle cabecera a las columnas. En cambio si se tiene el fichero DATOS2.TXT, con cabecera y separados los datos por espacios (ver Figura 8), podría ordenarse:

```
ejemplo2=read.csv("DATOS2.txt",header=TRUE,sep=" ")
ejemplo2
```

```
##   Y  X1 X2  X3
## 1 10 1.0 50 0.40
## 2 12 1.2 57 0.40
## 3 11 1.3 56 0.42
## 4 13 1.4 55 0.50
## 5 12 1.5 60 0.40
```



Y	X1	X2	X3
10	1	50	0.4
12	1.2	57	0.4
11	1.3	56	0.42
13	1.4	55	0.5
12	1.5	60	0.4

Figure 8: Fichero de datos separados por espacios y con nombres en las columnas

Si se tienen archivos tipo Excel (formatos .xls, .xlsx) se cargará la **librería readxl** y gracias a la función `read_excel()` se podrá acceder a su contenido. En la siguiente práctica se profundizará sobre el concepto y uso de librerías (también llamados paquetes) en R. De momento, en la presente práctica solo se aborda la instalación y carga de estas librerías:

- **Instalación:** Utilizar la función `install.packages()`, introduciendo el nombre del paquete entre comillas; o bien, seguir la ruta **Packages** (panel 2 de Rstudio) → **botón install** → **escribir nombre del paquete**.
- **Carga:** Ejecutar la función `library()`, introduciendo el nombre del paquete. En este caso no debe escribirse entre comillas.

**Nota:** Una vez instalado el paquete, este permanecerá en el ordenador personal en el que haya sido instalado y no hará falta instalarlo más veces. Sin embargo, el proceso de carga del paquete si será necesario cada vez que se abra el programa RStudio. En cambio, debido al sistema propio de los ordenadores de la Universidad, los paquetes **SÍ** deberán ser instalados todos los días.

A modo de ejemplo se considera el fichero de datos creado en Excel que aparece en la Figura 9.

que puede ser insertado en R como sigue:

```
library(readxl) #Carga la librería
ejemplo3=read_excel("datos3.xlsx")
ejemplo3
```

```
## # A tibble: 6 x 4
```

	A	B	C	D
1	Y	X1	X2	X3
2	5,6	0,8	6,5	2,3
3	4,6	0,9	7,4	2,5
4	4,9	1,2	7,6	3,2
5	5,5	1,1	7,9	3,3
6	6	1	7,7	3
7	5,3	0,75	5,9	2,9
8				

Figure 9: Fichero de datos en excel

```
##      Y    X1    X2    X3
## <dbl> <dbl> <dbl> <dbl>
## 1  5.6  0.8   6.5  2.3
## 2  4.6  0.9   7.4  2.5
## 3  4.9  1.2   7.6  3.2
## 4  5.5  1.1   7.9  3.3
## 5  6    1     7.7  3
## 6  5.3  0.75  5.9  2.9
```

Si se tienen archivos propios del programa SPSS (formato .sav) se cargará la librería **foreign** y gracias a la función **read.spss()** se podrá acceder a su contenido. A modo de ejemplo se importará el fichero de datos **osteo.sav**. Este fichero contiene los datos correspondientes a un estudio acerca de la densidad mineral ósea en pacientes diabéticos insulín-dependientes.

```
library(foreign)
osteo=read.spss("osteo.sav",to.data.frame=TRUE)
head(osteo)      #Muestra la información de los primeros 6 pacientes
```

```
##      num edad grupo_edad  sexo peso talla      imc tevol tabaco alcohol
## 1    1    23      < 25 Hombre 72.3  175 23.60816    2     Sí Moderado
## 2    2    35      > 33 Hombre 82.5  173 27.56524    4     No Moderado
## 3    3    51      > 33 Hombre 61.0  170 21.10727   17     Sí Moderado
## 4    4    44      > 33 Hombre 67.0  167 24.02381   17     Sí Moderado
## 5    5    54      > 33 Mujer 51.0  168 18.06973   30     Sí Moderado
## 6    6    36      > 33 Mujer 61.5  158 24.63548    2     Sí Moderado
##      ingca acfis retin nefro neuro hba1c ca  p  cr pthm pthi bmdcue
## 1  Suficiente  Sí    No    No    No    4.6 9.6 4.2 1.02 2.9 49.0 22.0
## 2  Suficiente  No    No    No    No    6.1 9.5 3.4 1.07 3.1  NA   NA
## 3  Suficiente  Sí    No    No    No    6.7 9.9 4.7 1.35  NA 54.3 23.7
## 4 Insuficiente  Sí  Leve No Grave 8.2 8.3 3.2 0.99  NA  NA   NA
## 5  Suficiente  Sí    No    No    No    9.7 9.5 3.7 0.86 3.1 49.8 35.7
## 6  Suficiente  Sí    No    No    No    8.4 9.9 4.7 0.80 1.1 59.9 38.3
##      szl24 sztri szcue osteo_cue osteo_tri
## 1  1.07  0.55  1.44      No      No
## 2  0.83 -1.62 -0.93      No      No
## 3  0.83 -1.28 -0.07      No      No
## 4  0.72 -1.11 -2.07      Sí      No
## 5  0.56 -1.68 -3.10      Sí      No
## 6  0.92  2.05  0.35      No      No
```

Otra opción disponible para abrir cualquier tipo de fichero es desde el menú **Import Dataset** que aparece en el panel 3. Haciendo click en **Import Dataset** se desplegará un menú en el que habrá que seleccionar el tipo de formato. Posteriormente, se abrirá una nueva ventana en la que habrá que hacer click en el botón **browse** y seleccionar el fichero de datos deseado. Automáticamente R importará la base de datos y la guardará en un objeto. R le asignará al objeto el mismo nombre que tenga el fichero de datos en la carpeta donde se encuentre. Notar que R puede utilizar otros comandos que los comentados anteriormente y puede guardar el fichero en un formato no deseado.