Introducción al Entorno Estadístico

Angelo Santana

INTRODUCCIÓN AL ENTONO ESTADÍSTICO R

© Angelo Santana

Departamento de Matemáticas, ULPGC.

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Departamento de Matemáticas

FECHA: 14 DE FEBRERO DE 2012

Índice general

Capítulo 1

El Entorno Estadístico R

1. ¿Qué es R?[1]

Podemos definir R como un *entorno en el que se aplican los métodos estadísticos de análisis de datos*. En este entorno, tal como se indica en la web del programa, www.r-project.org, podemos encontrar los siguientes componentes:

- Herramientas de lectura, importación, gestión y almacenamiento de datos.
- Funciones y operadores diseñados para actuar directamente sobre vectores o matrices.
- Una gran colección, coherente e integrada, de herramientas para el análisis de datos.
- Procedimientos gráficos para el análisis y visualización de datos, ya sea en pantalla o en papel.
- Un completo y bien desarrollado lenguaje de programación, simple y eficiente, que incluye gran cantidad de funciones predefinidas, instrucciones condicionales, bucles, posibilidad de definir funciones recursivas y procedimientos avanzados de entrada y salida, con la posibilidad de importar o exportar datos a un gran número de aplicaciones.
- Un *intérprete de comandos* para la aplicación efectiva de todas estas herramientas. A este intérprete es posible acceder de varias formas:

[1] La página web http://procomun.wordpress.com/2011/02/23/que-es-r/ contiene enlaces muy interesantes para hacerse una idea de qué es R .

- De modo interactivo a través de una *consola*.
- Lanzando un *script* (que podemos traducir como *guión)* que consiste básicamente en un archivo en el que el usuario ha introducido todos los comandos que desea ejecutar.
- A través de diversas interfaces gráficas de usuario (GUI).

Como R está diseñado en torno a un auténtico lenguaje de programación, los usuarios pueden, sin demasiado esfuerzo, añadir funcionalidad adicional mediante la definición de nuevas funciones. Gran parte del sistema está en sí mismo escrito en el lenguaje R , lo que facilita a los usuarios interesados el seguimiento y desarrollo del código. Para las tareas de cómputo intensivo, R puede enlazar en tiempo de ejecución con código C, C++ y Fortran. Los usuarios avanzados pueden incluso escribir código C para manipular objetos R directamente. No obstante, y para no asustar a aquellos usuarios que se acercan por primera vez a este programa, conviene señalar que no es necesario saber programar para utilizar R , si bien es cierto que unos mínimos conocimientos de programación permiten hacer un uso más eficiente de sus recursos.

Las capacidades de R pueden ampliarse fácilmente a través de la incorporación de *paquetes* o *librerías*. En el momento de la redacción de estas notas (enero 2012), R cuenta con 3523 paquetes disponibles[2], que cubren una gama muy amplia de la estadística moderna. En su mayor parte, han sido los propios usuarios de R los que, utilizando las capacidades de programación que éste ofrece, han desarrollado estas librerías que incluyen algoritmos avanzados para el análisis de datos, el desarrollo de nuevas técnicas de data mining, o la realización de gráficos de alta calidad -desde los gráficos estadísticos más simples, al tratamiento de mapas o de imágenes de resonancia magnética- demandados por publicaciones científicas, por la empresa o por la administración.

Sin duda, el hecho de que R sea software libre, que puede ser utilizado por el usuario sin coste alguno, unido a la facilidad de programación, han sido las claves para que este programa se haya convertido en la actualidad en el estándar *de facto* para el desarrollo e intercambio de nuevo software estadístico en la comunidad científico-técnica.

2. Ventajas e inconvenientes del uso de R.

Ventajas:

- Es software libre y por tanto su coste es nulo.

[2]Señalemos de paso que hace un año (enero 2011) el número de paquetes era de 2782, lo que significa que R ha crecido en el último año a la nada despreciable velocidad de aproximadamente 2 paquetes diarios.

- Es multiplataforma: existen versiones para Linux, Mac y Windows. Los procedimientos y análisis desarrollados en una plataforma son inmediatamente ejecutables en otra[3].
- Implementa una enorme cantidad de métodos estadísticos, desde los más clásicos a los más modernos. Los métodos se organizan en librerías cuyo número se encuentra en constante crecimiento.
- Dispone de una enorme capacidad para combinar, de manera simple, métodos de análisis estándar (regresión, análisis cluster, análisis de series temporales) con análisis desarrollados ad hoc para una situación específica.
- Capacidad para acceder a datos en múltiples formatos. Dispone de librerías para leer datos desde SPSS, SAS, Access, MySQL, Excel,... Asimismo permite también la generación de informes de resultados en diversos formatos.
- Enorme capacidad para manipular y /o modificar datos y funciones.
- Generación de gráficos de alta calidad.
- Existencia de una comunidad de usuarios muy activa, en la que participan estadísticos de renombre.
- Amplia disponibilidad de documentación, tanto en internet como en libros publicados por editoriales de prestigio (Springer, Wiley)[4].
- Facilidad de integración con actividades de formación en técnicas y métodos estadísticos en todos los ámbitos del conocimiento. Su uso es cada vez más generalizado en las universidades, lo que implica que las nuevas generaciones de profesionales ya salen al mercado laboral con formación específica en el manejo de este programa.
- En particular, su uso en la docencia tiene la ventaja de que no es necesario que el estudiante adquiera licencias para su uso, por lo que cualquier alumno puede instalar R en su ordenador personal y desarrollar tareas, trabajos, etc. utilizando este programa. Asimismo, una vez que el estudiante se gradúe y abandone la universidad, podrá seguir utilizando R en cualquier ámbito profesional o de desarrollo.
- Existencia de extensiones específicas para nuevas áreas como bioinformática, geoestadística, modelos gráficos o análisis de mercados financieros.

[3] Debemos puntualizar, no obstante, que hay algunas librerías –muy pocas– que funcionan sólo en linux o sólo en windows.

[4] La editorial Springer tiene una colección –UseR!– dedicada exclusivamente a R .

- Todos los algoritmos implementados en R pueden ser vistos e interpretados por cualquier usuario, por lo que éste puede saber exactamente qué es lo que hace el ordenador cuando ejecuta un comando.

Inconvenientes:

- Suele señalarse como principal desventaja de R el hecho de que el paquete base no dispone de una interfaz amigable para el usuario; no existe un menú principal en el que el usuario pueda acceder mediante el ratón a submenús para la lectura de datos, la ejecución de procedimientos estadísticos o la generación de gráficos. Estas tareas se llevan a cabo mediante un lenguaje de comandos que puede resultar duro de aprender para el usuario común. No obstante se han desarrollado algunas GUIs (Graphical User Interfaces) que facilitan enormemente esta tarea. En particular la interfaz R-Commander desarrollada por John M. Fox en la McMaster University de Canadá presenta un menú para el acceso a los comandos más habituales que, además, muestra el código que emplea R de forma que permite al usuario familiarizarse con el lenguaje.

- El código R es *interpretado,* no *compilado*, por lo que algún algoritmo puede resultar de ejecución lenta, en particular si se realizan tareas de simulación intensiva. Esto no constituye mayor problema para un uso ordinario del programa. En cualquier caso, a partir de la versión 2.14, todas las funciones y librerías de R se encuentran *precompiladas,* lo que acelera su ejecución de manera notable.

- No dispone de un sistema de base de datos propio, aunque sí cuenta con un formato para el almacenamiento e intercambio de datos. En cualquier caso se han desarrollado paquetes para conectar y acceder a múltiples sistemas de bases de datos (las propias de SAS o SPSS, Access, dBase, Excel, MySQL, ...). Tiene algunas limitaciones en cuanto al uso de la memoria, que dificultan el análisis de bases de datos masivas[5]. No obstante estas limitaciones han ido desapareciendo a medida que se han ido desarrollando ordenadores con mayor capacidad (procesadores de 64 bits, más disponibilidad de memoria y de direccionamiento de la misma). En cualquier caso, salvo que el usuario deba acceder a millones de registros simultáneamente, es difícil que llegue a notar problemas con la memoria.

- En algún caso las nuevas librerías que se incorporan a R pueden tener errores o fallos de implementación. Estos fallos, no obstante, suelen ser detectados por los usuarios, informados a los desarrolladores de las librerías y corregidos en tiempo récord. Debe señalarse, no obstante, que ningún programa (incluso los comerciales) está exento de fallos y el proceso de revisión y corrección de fallos en programas comerciales mediante parches o actualizaciones suele ser notablemente más lento.

[5] Ello se debe a que *todos* los datos con los que se trabaja deben permanecer simultáneamente en memoria. No obstante es posible derivar parte de la carga de trabajo del procesamiento de datos al propio motor de la base de datos que se utiliza, mediante comandos SQL.

- No hay nadie a quien reclamar si algo falla, ni hay un departamento de atención al cliente que nos diga qué podemos hacer si algo va mal, si algún procedimiento nos da un error, o simplemente si no sabemos qué sintaxis utilizar. Pero a cambio existe una comunidad de usuarios organizada en foros y dispuesta a colaborar desinteresadamente en la resolución de problemas.

A todos los puntos anteriores podemos añadir el siguiente, que será considerado por unos una ventaja y por otros un inconveniente:

- Para hacer un buen uso de R hay que tener un buen conocimiento de los métodos estadísticos.

En realidad esta afirmación es cierta no sólo para R, sino para cualquier paquete estadístico. Sin embargo en la práctica programas como SPSS, Statistica o SYSTAT permiten, a través de sus menús, que el usuario pueda aplicar casi cualquier procedimiento estadístico –sea o no adecuado para sus datos o su problema– sin apenas esfuerzo y obtenga páginas de resultados que muchas veces es incapaz de interpretar. R es bastante más comedido en sus salidas de resultados y, cuando se han de aplicar modelos de cierta complejidad, la mayoría de las veces el usuario se verá obligado a especificar exactamente qué es lo que quiere hacer, lo que implica buen nivel de conocimiento de los problemas abordados.

3. Fuentes de información sobre R.

En los últimos años se han publicado cientos de libros de estadística que utilizan R como soporte informático para la puesta en práctica de los conocimientos adquiridos. Algunos de esos libros están disponibles para su descarga en la web. Asimismo múltiples universidades y centros de enseñanza imparten cursos de R y tiene su material también disponible en la web. Como sitios interesantes, la propia página web de R (`www.r-project.org`), en el enlace `Manuals` situado a la izquierda, y a partir de ahí en el enlace `contributed documentation`) dispone de una sección de documentación muy amplia en la que podemos encontrar los siguientes manuales en castellano:

- *"R para Principiantes"* (versión española de "*R for Beginners*", traducido por Jorge A. Ahumada)
- *"Introducción a R"* (versión española de *"An Introduction to R"* por Andrés González and Silvia González)
- *"Gráficos Estadísticos con R"* por Juan Carlos Correa and Nelfi González.
- *"Cartas sobre Estadística de la Revista Argentina de Bioingeniería"* por Marcelo R. Risk.
- *"Introducción al uso y programación del sistema estadístico R"* por Ramón Díaz-Uriarte (transparencias preparadas para un curso de 16 horas sobre R, dirigido principalmente a biólogos y bioinformáticos).

- *"Metodos Estadisticos con R y R Commander"* por Antonio Jose Saez Castillo.

Fuera de la web de R es posible encontrar manuales en español en muchos lugares. Por ejemplo:

- http://knuth.uca.es/moodle/course/view.php?id=37: Libro "libre": "*Estadística Básica con R y R-commander*" de A. J. Arriaza Gómez y otros profesores de la Universidad de Cádiz.

- eio.usc.es/pub/pateiro/files/PubDocentePracticasEstadistica.pdf: manual de estadística con R (muy orientado a la programación) de Manuel Febrero y otros, de la Universidad de Santiago de Compostela.

- http://www.argitalpenak.ehu.es/p291-content/eu/contenidos/libro/se_ccsspdf/eu_ccsspdf/adjuntos/Introduccion%20al%20entorno%20R.pdf: Introducción a R, de la profesora Paula Elosúa de la Universidad del País Vasco.

También en castellano, las siguientes webs resultan de bastante interés para la formación en R:

- http://personales.unican.es/gonzaleof/R/index.html: página web de Francisco J. González, profesor de la Universidad de Cantabria.

- http://www.ub.edu/stat/docencia/EADB/Curso%20basico%20de%20R.htm: curso básico de R, ofrecido por el profesor F. Carmona de la Universidad de Barcelona.

- http://ocw.uc3m.es/estadistica/aprendizaje-del-software-estadistico-r-un- curso *on-line* de R ofrecido por el profesor Alberto Muñoz de la Universidad Carlos III de Madrid.

4. Descarga e instalación del programa

El procedimiento de instalación de R en Windows es muy simple[6]. Basta acceder a la web www.r-project.org (ver figura 1) y en el recuadro central (Getting started) picar en el enlace download R. Este enlace nos conducirá a una lista de repositorios distribuidos por todo el mundo desde los que es posible descargar el programa. En nuestro caso podemos elegir el repositorio situado en España. A partir de ahí le indicaremos que deseamos descargar la versión para Windows, a continuación seleccionamos el paquete base y por último seleccionamos Download R. La web nos preguntará si deseamos guardar el archivo R-xx.yy.z-win.exe (donde *xx.yy.z* es el número de la última versión disponible de R, la 2.12.1 en el momento de escribir estas líneas). Elegimos la opción

[6]En esta sección nos referiremos exclusivamente a la instalación en Windows, por ser el sistema operativo más usado (por ahora). No obstante, la instalación para Mac es prácticamente idéntica (en este caso nos descargaríamos un archivo .pkg). Para Linux bastará con añadir la dirección de algún *mirror* de R (la lista de mirrors está en la web de R) a la lista de repositorios que utiliza el sistema, y proceder a instalar el programa mediante los comandos *sudo apt-get update* y *sudo apt-get install r-base* (en Ubuntu se pueden usar también *synaptic* o el nuevo *Centro de Software*).

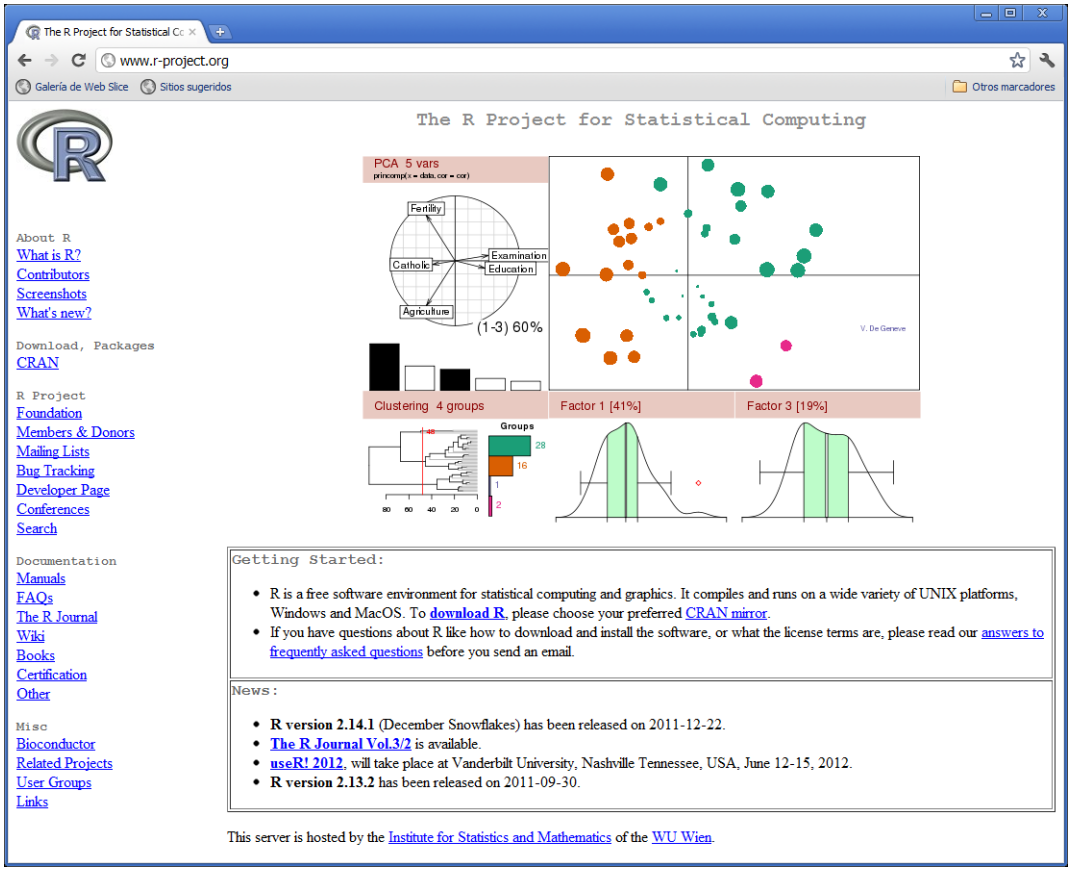


Figura 1: Pantalla principal de la página web de R-project

Guardar archivo y una vez que haya terminado de descargarse lo ejecutamos. Arranca entonces el *Asistente de instalación* que nos hará las preguntas típicas relativas al idioma de instalación, directorio donde guardar el programa, etc. Podemos elegir todas las opciones que nos ofrece por defecto (pinchando en Siguiente> en las sucesivas ventanas que van apareciendo). Si todo ha ido bien al finalizar el proceso tendremos el icono de R en el escritorio de Windows. Al picar dos veces sobre este icono se iniciará el programa tal como se muestra en la figura 2.

Este entorno dista mucho de los habituales en otros programas estadísticos. No vemos iconos ni múltiples botones que nos permitan hacer cosas. Si recorremos con el ratón los items del menú de la parte superior de la ventana, veremos como se nos van desplegando diversos submenús, en ninguno de los cuales encontramos opciones específicamente orientadas al análisis de datos. ¿Cómo funciona R entonces? ¿Cómo leemos nuestros datos, los procesamos, obtenemos resultados, imprimimos informes, gráficos,...?

La respuesta es que en realidad para hacer todas estas cosas R dispone de su propio lenguaje. Este lenguaje es el que dota a R de la tremenda versatilidad y potencia que posee. Pero también es lo que asusta y aleja a muchos usuarios potenciales del programa, que ven la sintaxis de este lenguaje como un obstáculo insalvable. Sin embargo, al igual que sucede con el aprendizaje de cualquier idioma, una vez que se conocen algunas palabras y las reglas gramaticales básicas, ya es posible mantener

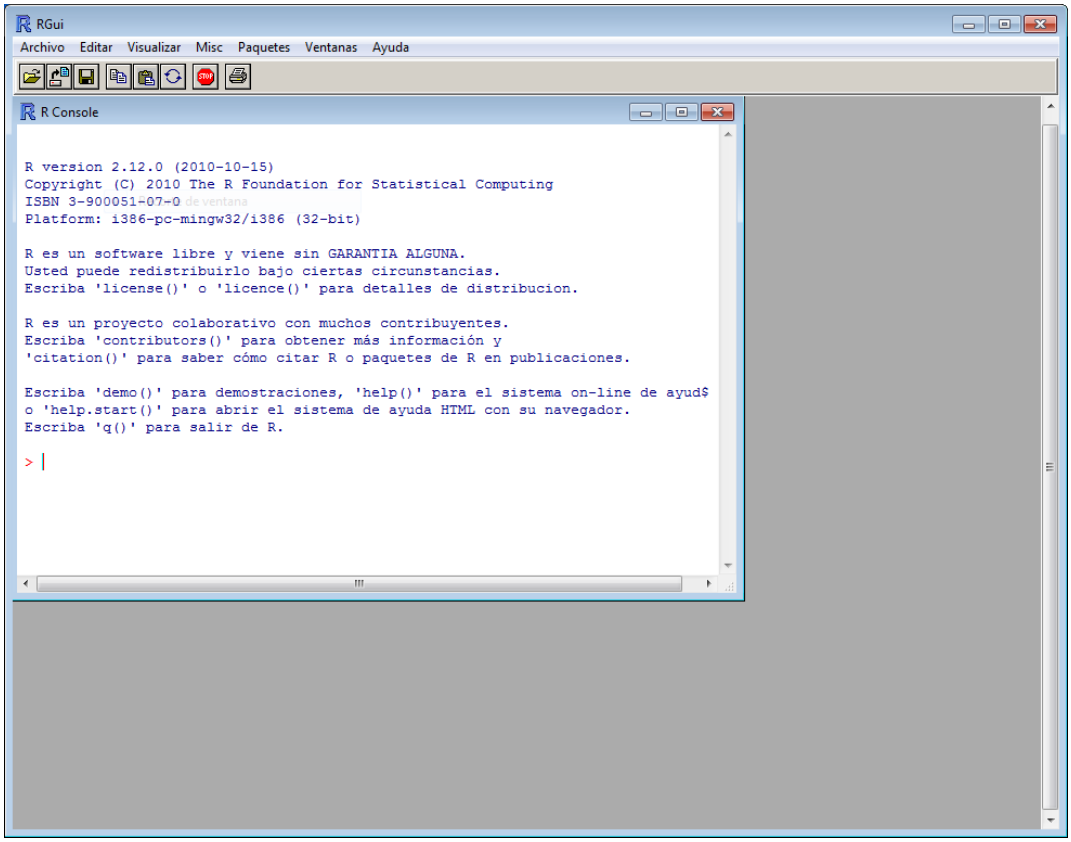


Figura 2: Entorno R por defecto en Windows

conversaciones sencillas. A partir de ahí, con la práctica y un poquito de empeño poco a poco podremos ir desenvolviéndonos cada vez con mayor soltura.

Para hacernos una idea de la sintaxis y posibilidades de R, comencemos nuestra primera sesión con el programa.

Capítulo 2

Primeros pasos con R.

1. Primera sesión con R.

Se ha seleccionado una muestra aleatoria de diez personas. A cada una de ellas se le ha preguntado su edad y se le ha medido el número de minutos que ha utilizado el móvil durante un día, entre lunes y viernes, de la semana en que se realizó el estudio. Para cada persona se anotó también su sexo (H: Hombre, M: Mujer). Los resultados se muestran en la tabla 2.1.

Edad	22	34	29	25	30	33	31	27	25	25
Tiempo	14.21	10.36	11.89	13.81	12.03	10.99	12.48	13.37	12.29	11.92
Sexo	M	H	M	M	M	H	M	M	H	H

Tabla 2.1: Tiempo diario de uso del móvil en una muestra de 10 personas

Vamos a utilizar R para hacer una estadística descriptiva sencilla de estas variables. Una vez arrancado el programa (figura 2) lo primero que hemos de hacer es introducir los datos. Para ello usamos la función *concatenar,* `c()`, escribiendo en nuestra ventana de R:

```
> edad <- c(22, 34, 29, 25, 30, 33, 31, 27, 25, 25)
> tiempo <- c(14.21, 10.36, 11.89, 13.81, 12.03, 10.99, 12.48,
     13.37, 12.29, 11.92)
```

La variable sexo podemos introducirla como texto, en cuyo caso hemos de utilizar comillas:

```
> sexo <- c("M", "H", "M", "M", "M", "H", "M", "M", "H", "H")
```

Podemos asignar *etiquetas* a los valores de esta variable, de forma que en las salidas de resultados quede más claro el significado de cada valor. Para ello redefinimos *sexo* como `factor` y especificamos las etiquetas (`labels`) correspondientes a cada valor (`level`) del siguiente modo:

```
> sexo <- factor(sexo, levels = c("H", "M"), labels = c("Hombre",
     "Mujer"))
```

Calculemos las medias de las variables edad y tiempo de uso del móvil, y contemos cuantos hombres y mujeres tenemos en la muestra:

```
> mean(edad)
[1] 28.1
> mean(tiempo)
[1] 12.34
> table(sexo)
sexo
Hombre  Mujer
     4      6
```

Dibujemos ahora un histograma (`hist()`) del tiempo de uso del móvil y un diagrama de barras (`barplot()`) de la distribución de sexos:

```
> par(mfrow = c(1, 2))
> hist(tiempo, main = "Tiempo de uso del movil", xlab = "Tiempo (minutos)")
> barplot(table(sexo), main = "Distribución de sexos")
```

El comando `par(mfrow=c(1,2))` tiene como única función configurar la salida de gráficos en 1 fila y 2 columnas, de tal modo que nos muestra los dos gráficos solicitados uno junto a otro. La figura 1 muestra el resultado de estos comandos.

Por último calculamos la recta de regresión (`lm()`, *linear model*) para predecir el tiempo de uso del móvil en función de la edad del sujeto, y la representamos gráficamente (figura 2).

```
> regresion <- lm(tiempo ~ edad)
> regresion
Call:
lm(formula = tiempo ~ edad)

Coefficients:
(Intercept)         edad
     19.320       -0.249
> par(mfrow = c(1, 1))
> plot(edad, tiempo)
> abline(regresion, col = "red")
```

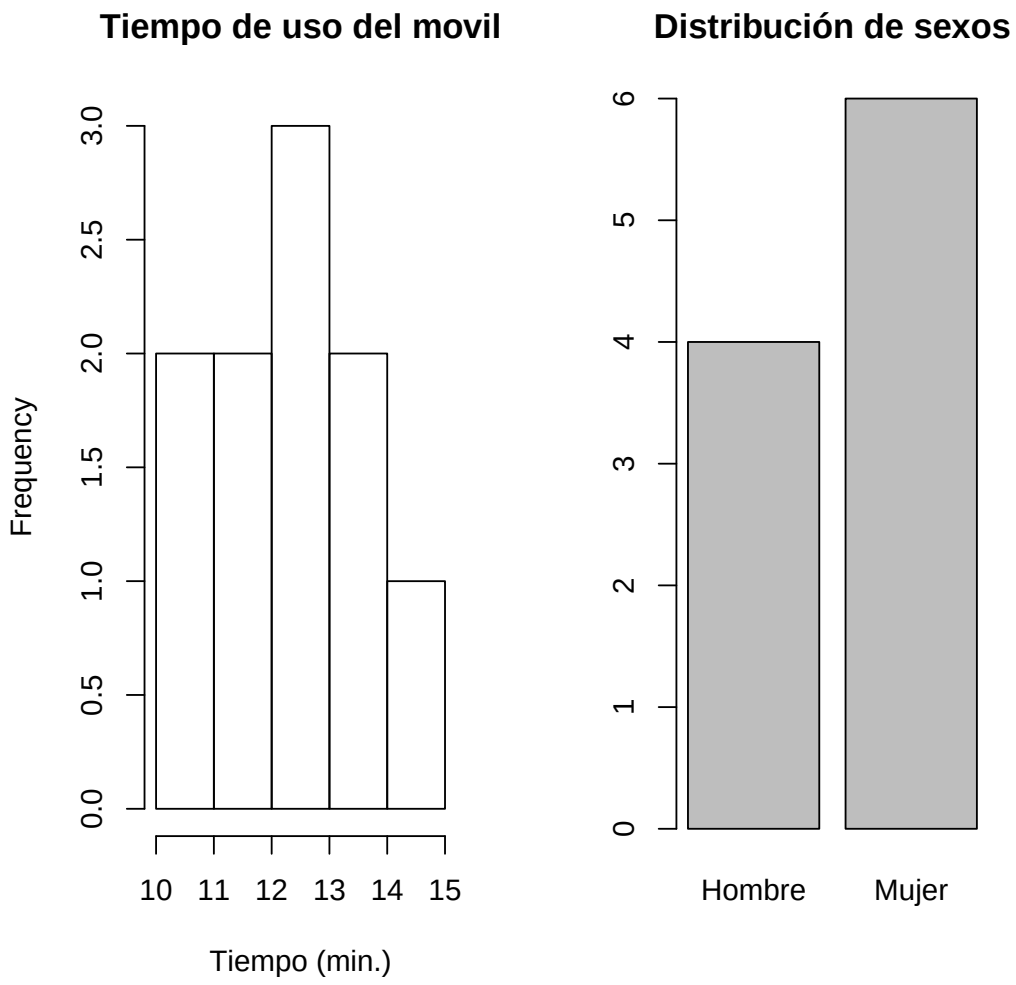


Figura 1: Ejemplos de histograma y diagrama de barras

Nótese que para especificar el modelo lineal que se desea estimar es preciso utilizar el símbolo ~. Este símbolo se obtiene:

- En Windows pulsando simultáneamente las teclas Alt-4 seguidas de la barra espaciadora.
- En Mac pulsando simultáneamente Alt-ñ.
- En Linux también se obtiene con Alt-4 (no es necesaria la barra espaciadora).

2. Elementos básicos de R.

Observemos los comandos (instrucciones) que hemos empleado en esta primera sesión con R. Para R todo lo que se maneja en su lenguaje son *objetos*. Los objetos pueden ser de muchas *clases*

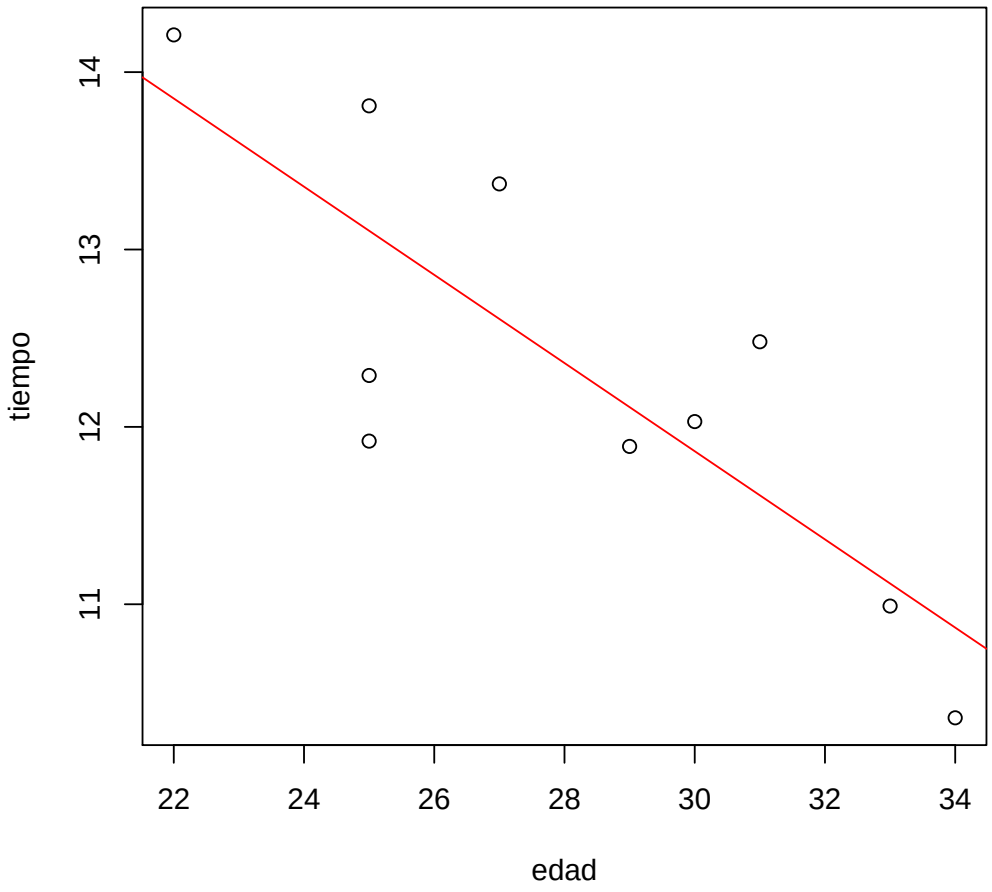


Figura 2: Gráfico del tiempo de uso del móvil frente a la edad del usuario para los datos de la tabla 2.1

diferentes. Así, en esta sesión hemos utilizado:

- **Vectores**: un vector es una colección de valores observados en una variable. En nuestro ejemplo, `tiempo` es un vector de *longitud* 10, que contiene los valores `(14.21, 10.36, 11.89, 13.81, 12.03, 10.99, 12.48, 13.37, 12.29, 11.92)`. Del mismo modo, `sexo` es también un vector de longitud 10 con los valores `("M", "H", "H", "M", "M", "H", "M", "M", "H", "H")`. El vector `tiempo` es numérico (nótese de paso que el separador decimal en R es el punto), y `sexo` es *alfanumérico* (sus valores pueden ser letras). Los valores de las variables alfanuméricas deben ir entrecomillados.

- **Funciones:** una función puede entenderse como un algoritmo o programa encargado de realizar alguna tarea sobre los valores que se le pasen como *argumento*. En nuestro ejemplo hemos usado las funciones:

  - `c()`: función *concatenación*. Se encarga de agrupar en un único vector todos los valores

que figuran en el paréntesis.

- `mean()`: calcula la media de los valores que se le pasen como argumento. Si su argumento es un vector, calcula la media de todos los valores que forman el vector (tal como ha hecho en nuestro ejemplo, calculando las medias de `edad` y `tiempo`.
- `table()`: calcula la tabla de frecuencias absolutas (número de veces que se observa cada valor) de la variable que se encuentra en el paréntesis.
- `hist()`: Dibuja un histograma de la variable que se le pase como argumento.
- `barplot()`: Dibuja un diagrama de barras correspondiente a la *tabla* que se le pasa como argumento.
- `lm(y ~ x)`: calcula la recta de regresión de `y` sobre `x`.
- `plot(x,y)`: dibuja la nube de puntos `(x,y)`.
- `abline()`: dibuja la recta que se especifique en el paréntesis.

- **Operadores:** El símbolo "`<-`" es el operador de asignación. De manera equivalente[1] podíamos haber utilizado también el signo "`=`". El comando `edad <- c(22, 34,..., 25)` *asigna* al vector `edad` los valores concatenados por la función `c()`. Asimismo el comando `regresion <- lm(tiempo ~ edad)` *asigna* al objeto `regresion` el resultado de calcular la recta de regresión del `tiempo` en función de la `edad`.

- **Listas:** una *lista* en R es una agrupación de vectores de distinta longitud y tipo. En nuestro ejemplo, `regresion` es una lista que contiene prácticamente todo el análisis estadístico de la regresión. Si bien antes cuando le pedimos a R que nos mostrase el valor de `regresion` se limitó a mostrarnos los valores de la pendiente y la ordenada en el origen, en realidad `regresion` contiene muchas más cosas. Si utilizamos la función `str()` podemos ver la estructura detallada de este objeto:

```
> str(regresion)
List of 12
 $ coefficients : Named num [1:2] 19.32 -0.249
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "edad"
 $ residuals    : Named num [1:10] 0.359 -0.508 -0.221 0.704 0.167 ...
  ..- attr(*, "names")= chr [1:10] "1" "2" "3" "4" ...
 $ effects      : Named num [1:10] -39.007 2.929 -0.234 0.568 0.186 ...
```

[1] En realidad esta equivalencia es sólo cierta en el *entorno –environment–* principal de trabajo, el espacio de memoria que utiliza R por defecto para almacenar los datos, funciones, variables y resultados que se generan en una sesión de trabajo. No obstante, en R es posible definir otros entornos, que podemos entender como espacios de memoria reservados, en los que la asignación debe realizarse necesariamente con el símbolo "<-".

```
 ..- attr(*, "names")= chr [1:10] "(Intercept)" "edad" "" "" ...
$ rank         : int 2
$ fitted.values: Named num [1:10] 13.9 10.9 12.1 13.1 11.9 ...
 ..- attr(*, "names")= chr [1:10] "1" "2" "3" "4" ...
$ assign       : int [1:2] 0 1
$ qr           :List of 5
 ..$ qr   : num [1:10, 1:2] -3.162 0.316 0.316 0.316 0.316 ...
 .. ..- attr(*, "dimnames")=List of 2
 .. .. ..$ : chr [1:10] "1" "2" "3" "4" ...
 .. .. ..$ : chr [1:2] "(Intercept)" "edad"
 .. ..- attr(*, "assign")= int [1:2] 0 1
 ..$ qraux: num [1:2] 1.32 1.62
 ..$ pivot: int [1:2] 1 2
 ..$ tol  : num 1e-07
 ..$ rank : int 2
 ..- attr(*, "class")= chr "qr"
$ df.residual  : int 8
$ xlevels      : Named list()
$ call         : language lm(formula = tiempo ~ edad)
$ terms        :Classes 'terms', 'formula' length 3 tiempo ~ edad
 .. ..- attr(*, "variables")= language list(tiempo, edad)
 .. ..- attr(*, "factors")= int [1:2, 1] 0 1
 .. .. ..- attr(*, "dimnames")=List of 2
 .. .. .. ..$ : chr [1:2] "tiempo" "edad"
 .. .. .. ..$ : chr "edad"
 .. ..- attr(*, "term.labels")= chr "edad"
 .. ..- attr(*, "order")= int 1
 .. ..- attr(*, "intercept")= int 1
 .. ..- attr(*, "response")= int 1
 .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
 .. ..- attr(*, "predvars")= language list(tiempo, edad)
 .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
 .. .. ..- attr(*, "names")= chr [1:2] "tiempo" "edad"
$ model        :'data.frame':        10 obs. of  2 variables:
 ..$ tiempo: num [1:10] 14.2 10.4 11.9 13.8 12 ...
 ..$ edad  : num [1:10] 22 34 29 25 30 33 31 27 25 25
 ..- attr(*, "terms")=Classes 'terms', 'formula' length 3 tiempo ~ edad
 .. .. ..- attr(*, "variables")= language list(tiempo, edad)
```

```
  .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
  .. .. .. ..- attr(*, "dimnames")=List of 2
  .. .. .. .. ..$ : chr [1:2] "tiempo" "edad"
  .. .. .. .. ..$ : chr "edad"
  .. .. ..- attr(*, "term.labels")= chr "edad"
  .. .. ..- attr(*, "order")= int 1
  .. .. ..- attr(*, "intercept")= int 1
  .. .. ..- attr(*, "response")= int 1
  .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
  .. .. ..- attr(*, "predvars")= language list(tiempo, edad)
  .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  .. .. .. ..- attr(*, "names")= chr [1:2] "tiempo" "edad"
 - attr(*, "class")= chr "lm"
```

No entraremos aquí a explicar qué es cada uno de los términos que componen el objeto `regresion`. Baste decir que contienen información suficiente para dibujar la recta (es lo que hemos hecho aplicando al objeto `regresion` la función `abline`), hacer predicciones, estimar intervalos de confianza para los coeficientes, etc. R cuenta con funciones encargadas de extraer dicha información en un formato mucho más amigable. A modo de ejemplo podemos citar las funciones:

- `summary()`: nos presenta un resumen de la inferencia para la regresión.
- `confint()`: nos da intervalos de confianza para los coeficientes de la recta.
- `residuals()`: nos muestra los residuos de la regresión.

Invitamos al lector a que ejecute estos comandos en su sesión de R:

```
> summary(regresion)
> confint(regresion)
> residuals(regresion)
```

3. Sistemas de ayuda en el entorno R

Hasta ahora hemos sido capaces de ejecutar algunas instrucciones en R y obtener resultados sin mucha dificultad. No obstante, muchas veces nos encontraremos con comandos de los que no sabemos (o no recordamos) cómo es su sintaxis, qué opciones ofrecen, etc. Para ello R ofrece varias posibilidades de ayuda:

- `help()`: es el comando de ayuda por excelencia. Por ejemplo, para obtener ayuda sobre la función `mean` basta con teclear:

```
> help(mean)
```

De manera equivalente podríamos usar la sintaxis `?mean`.

- `apropos()`: si queremos utilizar una función en cuyo nombre aparecía el término `norm` pero no recordamos exactamente qué función era, `apropos("norm")` nos devuelve un listado de funciones que contienen dicho témino en su definición.

- `help.search()`: si, en cambio, quisiéramos saber, entre todos los paquetes que R tiene instalados en nuestro sistema, cuáles contienen funciones relacionadas, aunque sea vagamente, con el término `norm` podemos utilizar `help.search("norm")` o, de manera equivalente `??norm`.

- `help.start()`: nos arranca una página html como la que se muestra en la figura 3 en la que podemos acceder a los manuales originales del programa (en inglés), documentación de referencia sobre los distintos paquetes instalados en nuestro sistema (con información sobre cada una de las funciones que lo componen) y diverso material misceláneo.

- `help(package=nombre-libreria)` Arranca una página html con ayuda sobre el paquete o librería especificada.

- `demo()`: nos hace demostraciones de las posibilidades de uso de distintos paquetes. Por ejemplo: `demo(graphics); demo(persp); demo(lm.glm)`. `demo()` sin argumentos muestra una lista de los paquetes disponibles en el sistema que incluyen demostraciones.

- `vignette(tema,package,...)` Muestra un documento pdf sobre el tema especificado, asociado al paquete que se indica. No todos los paquetes contienen documentos pdf de esta clase. Puede obtenerse una lista de dichos documentos mediante `browseVignettes()`.

Por último, si dentro de nuestro entorno R no encontramos ayuda para el problema que nos ocupa, siempre podemos recurrir a la página web de R (fig 1) y en el menú de la izquierda elegir la opción search. Por último, otro sitio interesante donde obtener ayuda es R-Wiki (http://rwiki.sciviews.org/doku.php).

Figura 3: Página html de ayuda que se despliega al ejecutar el comando `help.start()`

Capítulo 3

Scripts (Archivos de Sintaxis) en R.

1. Archivos de Sintaxis en el entorno por defecto de R.

Como hemos podido ver en nuestra primera sesión con R, el uso de este programa requiere el conocimiento de su sintaxis y de las funciones predefinidas. Obviamente ello sólo es posible consultando manuales y aprendiendo qué función es la que hace el análisis estadístico que nos interesa en cada momento. El uso continuado del programa permite que vayamos memorizando estas funciones y las usemos con soltura. Ahora bien, en muchos entornos profesionales -medicina, ciencias sociales, ingeniería,...- la estadística es una herramienta que, si bien resulta indispensable, tampoco se usa todos los días. Ello da lugar a que, enfrentado a un problema de análisis de datos, sea éste nuevo o rutinario, el usuario deba hacer a veces un gran esfuerzo para recordar como arranca el programa, como se leen los datos, como se calcula una media...

Dos son las aproximaciones más usadas para enfrentarse a este problema:

1. **Documentar y archivar los procedimientos empleados para cada análisis** en archivos de comandos, llamados *scripts* (guiones). Estos *scripts* pueden reutilizarse con nuevos conjuntos de datos y, en muchas ocasiones, nos sirven de guía para el desarrollo de nuevos procedimientos. Por ello es recomendable en las sesiones de trabajo con R (salvo que sea algo extremadamente simple) no introducir nuestras instrucciones directamente en la linea de comandos, (la que empieza con el símbolo “>”, y en la que hemos ejecutado el ejemplo visto en la sección anterior), sino abrir un archivo de script e incluir en éste todas las acciones a realizar durante nuestra sesión de trabajo. En dicho archivo podenos incluir lineas de comentario, que deben empezar por el símbolo #, informando de qué es lo que se hace y por qué se hace. Pensemos que aunque ahora sepamos muy bien lo que estamos haciendo, si vamos a volver a empler este código dentro de unos meses (¡a lo mejor sólo dentro de unas semanas!) es muy probable que para ese entonces no recordemos cuál era su función. Es por ello que el guión para nuestra sesión de ejemplo debería quedar, una vez editado, tal como se muestra en la tabla 3.1; como vemos, se han insertado lineas de comentario (precedidas por el símbolo #) en las que se explica brevemente qué hacen

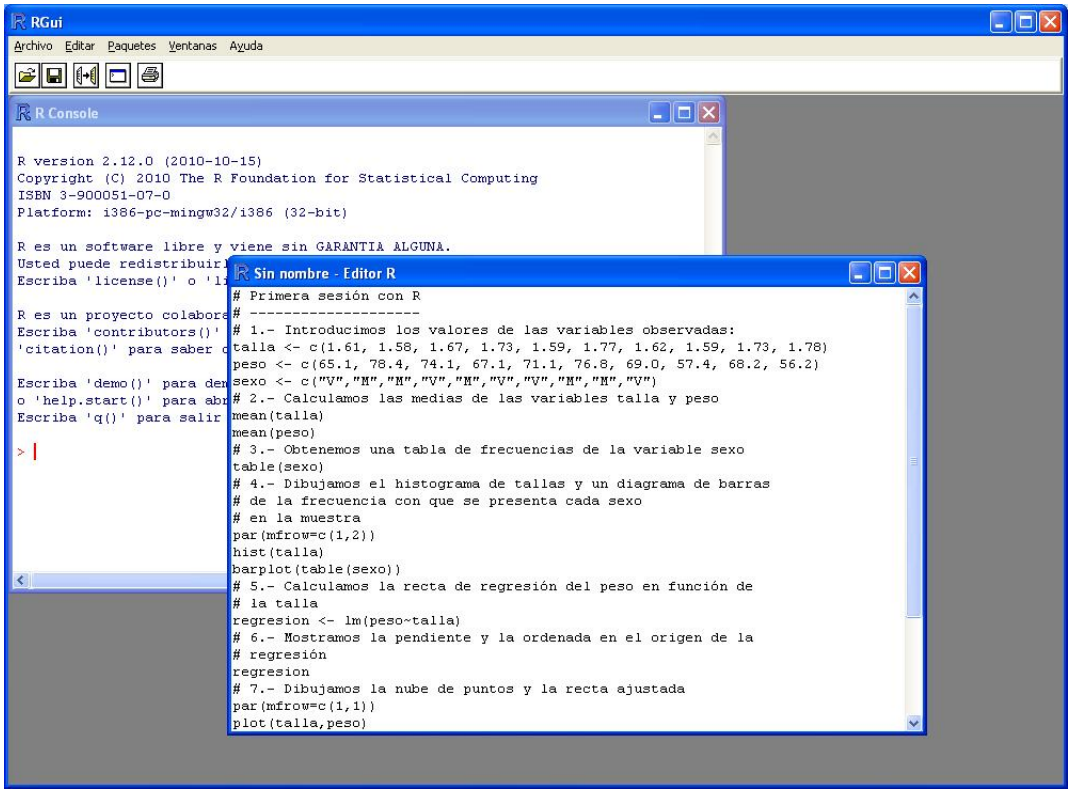


Figura 1: Ventana de edición de scripts de código de R

las distintas partes del código.

```
# --------------------
# Primera sesión con R
# --------------------
# 1.- Introducimos los valores de las variables
observadas:
edad <- c(22, 34, 29, 25, 30, 33, 31, 27, 25, 25)
tiempo <- c(14.21, 10.36, 11.89, 13.81, 12.03, 10.99,
12.48, 13.37, 12.29, 11.92)
sexo <- c("M", "H", "H", "M", "M", "H", "M", "M", "H",
"H")
# 2.- Calculamos las medias de las variables edad y
tiempo
mean(edad)
mean(tiempo)
# 3.- Obtenemos una tabla de frecuencias de la variable
sexo
table(sexo)
# 4.- Dibujamos el histograma del tiempo de uso del
móvil y un
# diagrama de barras de la frecuencia con que se
presenta cada
# sexo en la muestra
par(mfrow=c(1,2))
hist(tiempo)
barplot(table(sexo))
# 5.- Calculamos la recta de regresión del tiempo en
función de
# la edad
regresion <- lm(tiempo~edad)
# 6.- Mostramos la pendiente y la ordenada en el origen
de la
# regresión
regresion
# 7.- Dibujamos la nube de puntos y la recta ajustada
par(mfrow=c(1,1))
plot(edad, tiempo)
abline(regresion,col="red")
# 8.- Vemos el resumen de la inferencia sobre la
regresión
summary(regresion)
# 9.- Intervalos de confianza para los coeficientes
confint(regresion)
# 10.- Obtenemos los residuos de la regresión
residuals(regresion)
```

Tabla 3.1: Script de código R para el ejemplo de la sección 1

Para generar un archivo de script, en el menú de R elegimos la opción *Archivo* → *Nuevo Script.* Se nos abre una ventana de edición en la que podemos escribir nuestras líneas de código, tal como se muestra en la figura 1. Una vez que hayamos terminado, guardamos el archivo. Conviene añadirle la extensión *".R"* para que nuestro archivo de código R quede perfectamente identificado para el sistema operativo.
Una vez que se dispone de un *script* éste puede ejecutarse linea a linea simplemente situando el cursor sobre la linea a ejecutar y pulsando CTRL-R, o pulsando el tercer icono de la parte superior izquierda de la pantalla (un icono que muestra dos pantallas con una flecha apuntando de una a otra). Si se desea, se puede ejecutar también todo el código de una vez marcándolo y pulsando CTRL-R, o pulsando en el mismo icono anterior. El comando o comandos que se hayan indicado se ejecutarán y mostrarán su resultado en la ventana principal de R.

2. **Utilizar un entorno de R que disponga de un menú de comandos** al estilo de otros programas estadísticos. R dispone de una librería llamada *R-commander* que facilita un entorno con estas características: el usuario elige el procedimiento en un menú y R-commander se encarga de escribir y ejecutar el código R asociado. Un poco más adelante aprenderemos a instalar y utilizar este entorno.

2. Editores de Código alternativos.

Tal como hemos visto en la sección anterior, para editar código R podemos utilizar el editor de scripts implementado en el propio programa. Este editor en general va muy bien, pero resulta poco práctico si desarrollamos un código muy largo que incluya, por ejemplo, llamadas a múltiples funciones o generación de nuevas variables u objetos. En estos casos los principales problemas a que se enfrenta el usuario derivan de errores muy simples: paréntesis que se han abierto pero no se han cerrado, comillas mal colocadas, comentarios mal declarados,... En el editor de scripts que incluye R por defecto estos errores resultan difíciles de detectar. Existen otros editores que utilizan diferentes colores para las distintas clases de objetos de R, que autocompletan el código y que incluyen algunos menús de ayuda, facilitándonos mucho la tarea de desarrollar nuestros programas en R y detectar los posibles errores de sintaxis. Describimos a continuación dos de estos editores, que destacan por sus prestaciones y facilidad de uso.

2.1. RStudio

RStudio es un IDE (*Integrated Development Environment*, o Entorno de Desarrollo Integrado) de código abierto para R, lanzado recientemente (febrero 2011). Reúne todas las características que acabamos de citar, a las que se une una muy interesante organización del escritorio de trabajo, que nos muestra un entorno muy compacto con una ventana de edición, otra de resultados, otra para gráficos

y una cuarta para la gestión de variables en memoria. Asimismo los menús de ayuda y la descarga de librerías se organizan en pestañas fácilmente accesibles. Cuenta además con la ventaja de ser multi-plataforma, lo que significa que existen versiones para Windows, Linux y Mac. En la figura podemos ver el aspecto de RStudio. Este programa puede descargarse libremente de la web http://www.rstudio.org/. La organización del escritorio y la excelente integración con R hacen MUY RECOMENDABLE el uso de este programa.

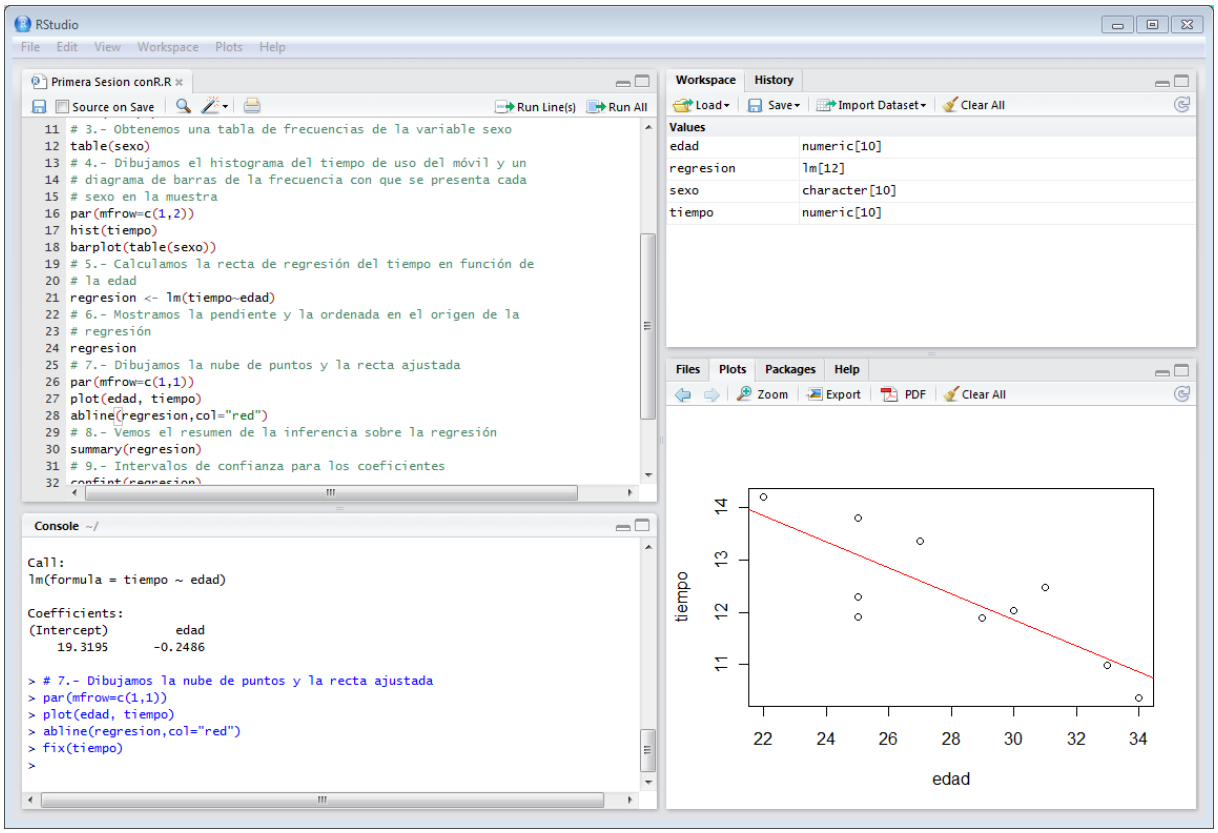


Figura 2: Entorno Integrado de Desarrollo para R que ofrece el programa RStudio. Pueden verse la ventana de edición, la ventana de resultados, la ventana de inspección de variables y la ventana de gráficos.

2.2. Tinn-R

En Windows, un editor muy completo (si bien no tan cómodo como `RStudio`) con todas las características citadas es Tinn-R[1], que puede descargarse gratuitamente de la dirección `http://sourceforge.net/projects/tinn-r/`. En la figura 3 podemos ver un código de ejemplo en

[1]Si utilizamos Linux, existe un editor similar, el `Rkward`. Se puede obtener información sobre su instalación en `http://rkward.sourceforge.net/`. Para Ubuntu en particular Rkward puede instalarse desde los repositorios oficiales mediante `sudo apt-get install rkward`, o a través de *synaptic* o el *Centro de Software*. Si utilizamos Mac OS X, el editor por defecto que equipa R es suficiente para un uso normal del programa, si bien el `RStudio` es una alternativa más cómoda.

la ventana de Tinn-R. Puede apreciarse el coloreado de la sintaxis así como la presencia de múltiples iconos que facilitan la edición y ejecución del código R.

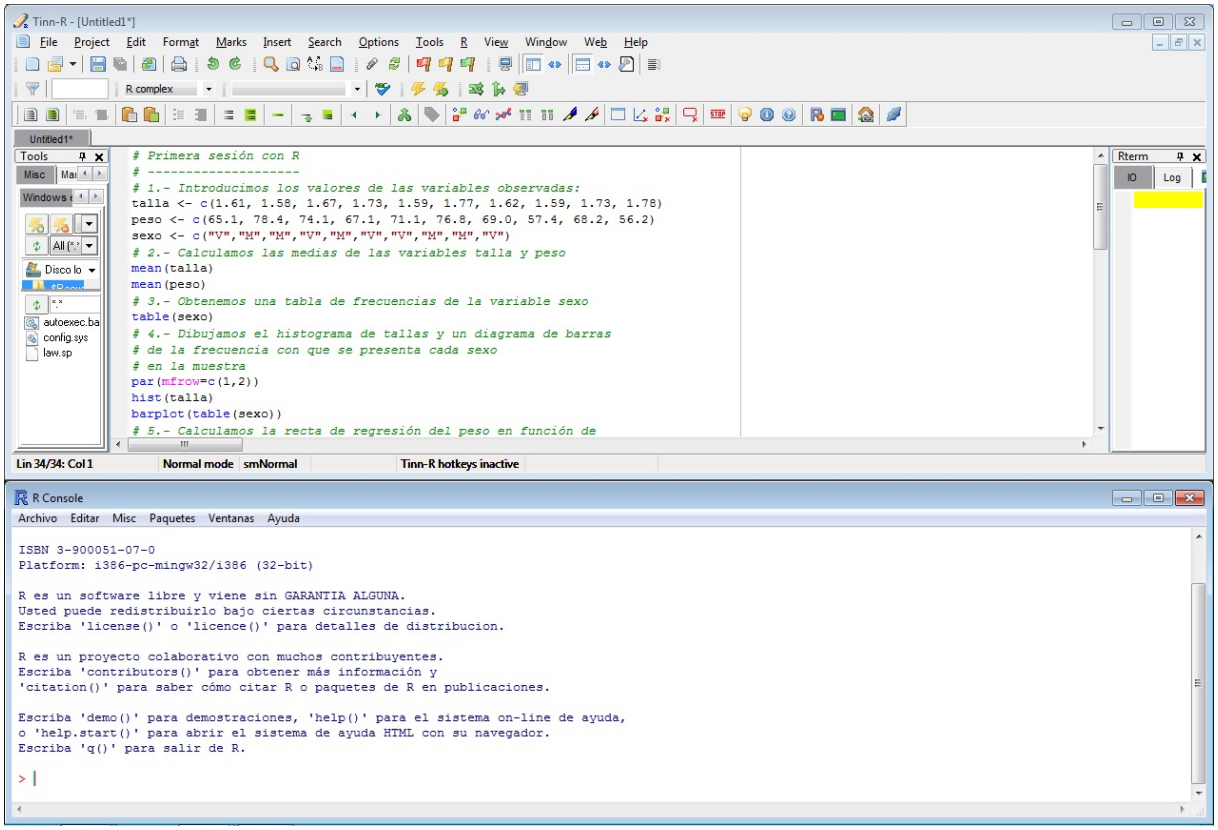


Figura 3: Ventana de edición de Tinn-R (parte superior). La ventana inferior es el entorno de ejecución de R por defecto en Windows. El propio programa Tinn-R al arrancar se encarga de ubicar ambas ventanas de esta manera, de tal forma que en la ventana superior escribimos nuestro código y en la inferior obtenemos el resultado que proporciona.

Capítulo 4

Instalación de librerías adicionales en R

1. Instalación de librerías directamente desde Internet.

Como ya hemos señalado, una de las grandes ventajas de R respecto a otros programas de análisis de datos es su capacidad de crecer mediante la incorporación de nuevas librerías que añaden nuevas funciones a las ya existentes en el programa. En la actualidad R cuenta con librerías que contienen funciones específicas para econometría, climatología, medicina, genética, química, demografía, mapas, análisis de redes, control de calidad,... Y el número de librerías aumenta continuamente.

El uso de una librería en R requiere que ésta haya sido instalada previamente en nuestro ordenador. Para ello, simplemente arrancamos R, y en el menú superior elegimos la opción[1] *Paquetes* $\rightarrow$*Instalar Paquete(s)*, tal como se muestra en la figura 1.

R nos pregunta entonces desde qué repositorio[2] deseamos descargar la librería. En principio, aunque todos los repositorios funcionan como *espejos* (todos tienen el mismo contenido), algunas librerías cuentan con ciertas adaptaciones locales en lo referido al idioma, por lo que resulta preferible realizar las descargas desde algún repositorio español. Tras elegir repositorio, se nos muestra un nuevo menú con el listado completo de librerías disponibles, ordenadas alfabéticamente[3], en el que seleccionamos el paquete que deseamos instalar[4]. Una vez que pulsemos OK comienza la descarga. Si todo va bien, ésta concluirá con un mensaje similar al siguiente:

[1] Conviene distinguir entre los conceptos *"Cargar paquete"* e *"Instalar paquete",* que aparecen agrupados en el mismo menú. *"Instalar paquete"* significa que deseamos descargarnos una librería desde el sitio de R en internet e instalarla físicamente en nuestro ordenador. Es lo que debemos hacer la primera vez que queremos usar una nueva librería. *"Cargar paquete",* en cambio, presupone que la librería ya ha sido instalada con anterioridad y que ahora pretendemos cargarla en memoria para que R pueda utilizarla durante esta sesión.

[2] Un *repositorio* es un lugar en el que se encuentran almacenados todos los programas que componen R, tanto el sistema base, como las librerías, como el código fuente. Ahora mismo existen múltiples repositorios repartidos por todo el planeta, que se mantienen constantemente actualizados.

[3] En algunos sistemas, dependiendo de su configuración, es posible que en la lista de librerías aparezcan primero aquellas cuyo nombre empieza con letra mayúscula y después las que empiezan con minúscula.

[4] Si deseamos instalar varios a la vez, podemos seleccionarlos manteniendo pulsada la tecla `Ctrl` mientras marcamos los nombres de los paquetes.

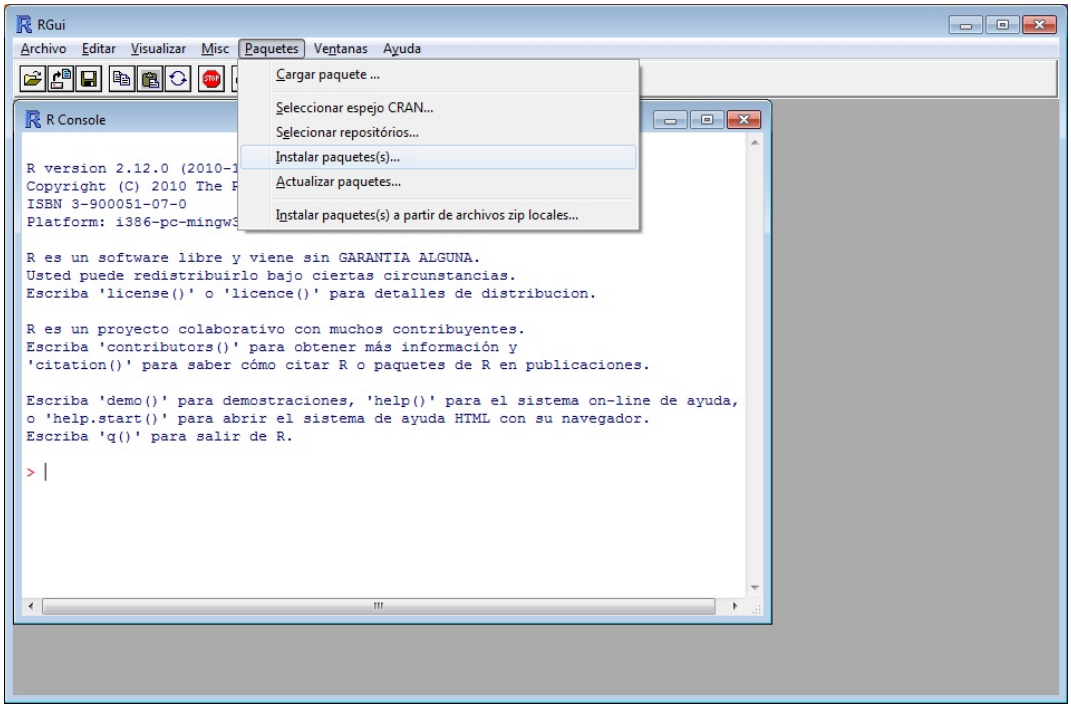


Figura 1: Instalación de paquetes en R

```
XYZT successfully unpacked and MD5 sums checked
The downloaded packages are in C:\Users\...\Rtmprp5FkF\downloaded_packages
```

donde XYZT es el nombre de la librería que nos hemos descargado. Este mensaje nos informa de que dicha librería se ha descomprimido y se encuentra instalada en nuestro ordenador. Una cuestión importante a tener en cuenta es que *tener una librería físicamente instalada en nuestro ordenador no significa que esté automáticamente disponible para su uso con R*. Para ello en nuestra sesión de trabajo, normalmente al principio y en cualquier caso *antes de comenzar a utilizar las funciones que contiene la librería*, debemos indicar a R que la cargue en memoria mediante el comando:

```
> library(nombre_de_la_librería)
```

Para conocer las funciones que contiene la librería y una pequeña descripción de las mismas podemos ejecutar:

```
> library(help = nombre_de_la_librería)
```

Para obtener información individualizada sobre cada función, suponiendo que ya se ha cargado la librería mediante `library()`, bastará con utilizar:

```
> help(nombre_de_la_funcion)
```

En lugar del comando `library()` puede utilizarse también el comando `require()`; tiene el mismo efecto que `library()` con la única diferencia de que si ya la librería estuviera cargada en memoria no vuelve a cargarla.

A modo de ejemplo podemos instalar la librería `sudoku` (¡En R hay incluso librerías para algunos juegos!). Vamos a *Paquetes* →*Instalar Paquete(s)* →*Spain (Madrid)* →*sudoku.*

Una vez que la librería se haya descargado e instalado, ejecutamos:

```
> library(help = sudoku)
```

y vemos las funciones que contiene. En particular vemos que la función `playSudoku()` permite hacer un sudoku en la pantalla del ordenador. Por tanto, para comenzar nuestro sudoku bastará con ejecutar:

```
> library(sudoku)
> playSudoku()
```

2. Instalación de paquetes desde archivos zip locales.

En ocasiones necesitamos instalar librerías de R sin estar conectados a internet. En tal caso, si disponemos del paquete que contiene la librería en formato comprimido zip (lo que presupone que nosotros mismos hemos descargado dicho paquete previamente de internet, o que nos ha sido facilitado de alguna otra manera, por ejemplo en un CD), podemos proceder muy sencillamente a su instalación simplemente eligiendo la opción:

Paquetes → *Instalar paquete(s) a partir de archivos zip locales...*

del menú principal de R. En la ventana que nos muestra sólo tendremos que elegir el directorio y el paquete que deseamos instalar. Si deseamos instalar varios podemos seleccionarlos todos a la vez.[5]

3. La librería R-Commander

Ya mencionamos en una sección anterior que una alternativa cómoda para utilizar R sin necesidad de conocer su lenguaje es emplear un sistema de menús al estilo de otros paquetes estadísticos, de

[5]Con la instalación de paquetes a partir de archivos zip locales nos encontramos a veces con problemas derivados de *dependencias* entre paquetes. Muchos paquetes de R dependen de funciones que se encuentran en otros paquetes. Si el paquete que estamos instalando a partir de un archivo local depende de algún otro paquete del que no disponemos, la instalación de nuestro paquete se completará, pero algunas funciones de las que contiene (y en algún caso puede que todas) no funcionarán adecuadamente.

tal forma que a través de dichos menús sea posible importar o exportar datos, elegir el tipo de análisis a realizar, elegir las representaciones gráficas, etc. En estos momentos R cuenta con la librería *R-Commander*, que crea un entorno de trabajo con un menú de estas características. La gran ventaja de este entorno, aparte de facilitar el acceso a los procedimientos estadísticos, es que cuando el usuario elige un procedimiento del menú, en la pantalla se muestra el código R que lleva a cabo dicho procedimiento, de tal forma que aún sin querer vamos aprendiendo el lenguaje R. Además el código generado puede guardarse en un fichero *script* para poder ser reutilizado en otra sesión de trabajo.

Para instalar R-Commander (al igual que para instalar cualquier otra librería), simplemente arrancamos R, y en el menú superior elegimos la opción[6] *Paquetes* $\rightarrow$*Instalar Paquete(s)*. En la lista de repositorios conviene elegir el respositorio *Spain (Madrid),* ya que la versión de R-Commander de dicho repositorio viene con el sistema de menús casi completamente traducido al castellano. A continuación, cuando se despliegue la lista de paquetes disponibles, elegimos `Rcmdr`.

Una vez que el paquete haya terminado de descargarse, para comenzar a utilizarlo en nuestra sesión de trabajo[7] ejecutaremos:

```
> library(Rcmdr)
```

La primera vez que ejecutemos R-Commander nos informará de que le faltan algunos paquetes para su correcto funcionamiento[8] y nos pedirá permiso para instalarlos[9]. Le decimos que sí y esperamos unos minutos mientras realiza la descarga e instalación de esos paquetes. Al finalizar arrancará el entorno R-Commander tal como se muestra en la figura 2.

En Windows el paquete Rcmdr funciona mejor con la interfaz SDI (Single Document Interface). Por defecto, R se instala con la interfaz MDI (Multiple Document Interface), que permite tener simultáneamente varias ventanas abiertas dentro de R. Sin embargo, como efecto secundario de esta interfaz, las ventanas de R-Commander en las que nos pide información asociada al procedimiento que hayamos puesto en marcha, se quedan ocultas bajo la ventana principal de R, lo que dificulta su manejo. La manera más sencilla de evitar este efecto es la siguiente:

1. Hacer una copia del icono R que se encuentra en el escritorio (picar con el botón derecho del ratón sobre el icono, elegir *copiar*, y pegar en otro lugar del escritorio). Al nuevo icono podemos ponerle el nombre R-SDI, por ejemplo.

[6] Conviene distinguir entre los conceptos *"Cargar paquete"* e *"Instalar paquete",* que aparecen agrupados en el mismo menú. *"Instalar paquete"* significa que deseamos descargarnos una librería desde el sitio de R en internet e instalarla físicamente en nuestro ordenador. Es lo que debemos hacer la primera vez que queremos usar una nueva librería. *"Cargar paquete",* en cambio, presupone que la librería ya ha sido instalada con anterioridad y que ahora pretendemos cargarla en memoria para que R pueda utilizarla durante esta sesión.

[7] Recuérdese: el paquete se instala desde internet una única vez; pero siempre que queramos utilizarlo deberemos usar el comando `library(Rcmdr)` al comienzo de nuestra sesión de trabajo con R.

[8] R-Commander ha ido incorporando a su sistema de menús opciones que se encuentran en otras librerías. Dependiendo de la configuración del sistema de cada usuario, al instalar Rcmdr, el programa detecta qué librerías están ya instaladas en el ordenador del usuario y descarga el resto.

[9] Conviene aclarar, no obstante, que la mayoría de las librerías de R son autocontenidas, y no requieren de la instalación de paquetes adicionales, como ocurre con R-commander.

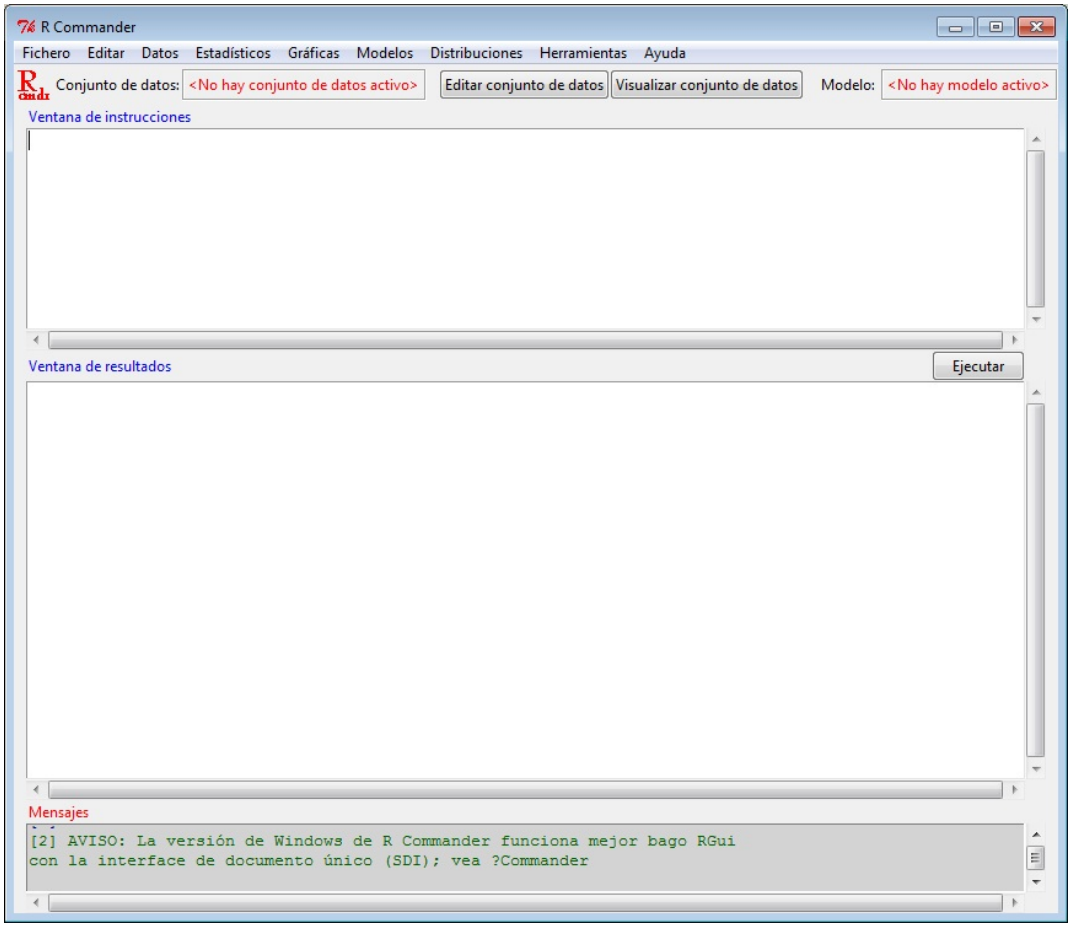


Figura 2: Entorno R-Commander

2. Picar con el botón derecho del ratón sobre el nuevo icono y seleccionar *Propiedades.* Veremos que en la cajita *Destino* dice algo así como *"C:\Program Files\R\R-2.12.0\bin\i386\Rgui.exe".* Añadir - *-sdi* a continuación, de tal manera que quede de la forma: *"C:\Program Files\R\R-2.12.0\bin\i386\Rgui.exe" - -sdi.*

3. Ahora cuando arranquemos R desde el icono R-SDI, y ejecutemos `library(Rcmdr)`, las distintas ventanas de R-Commander se presentarán correctamente.

Capítulo 5

Objetos en R: factores, vectores, matrices, data.frames, listas.

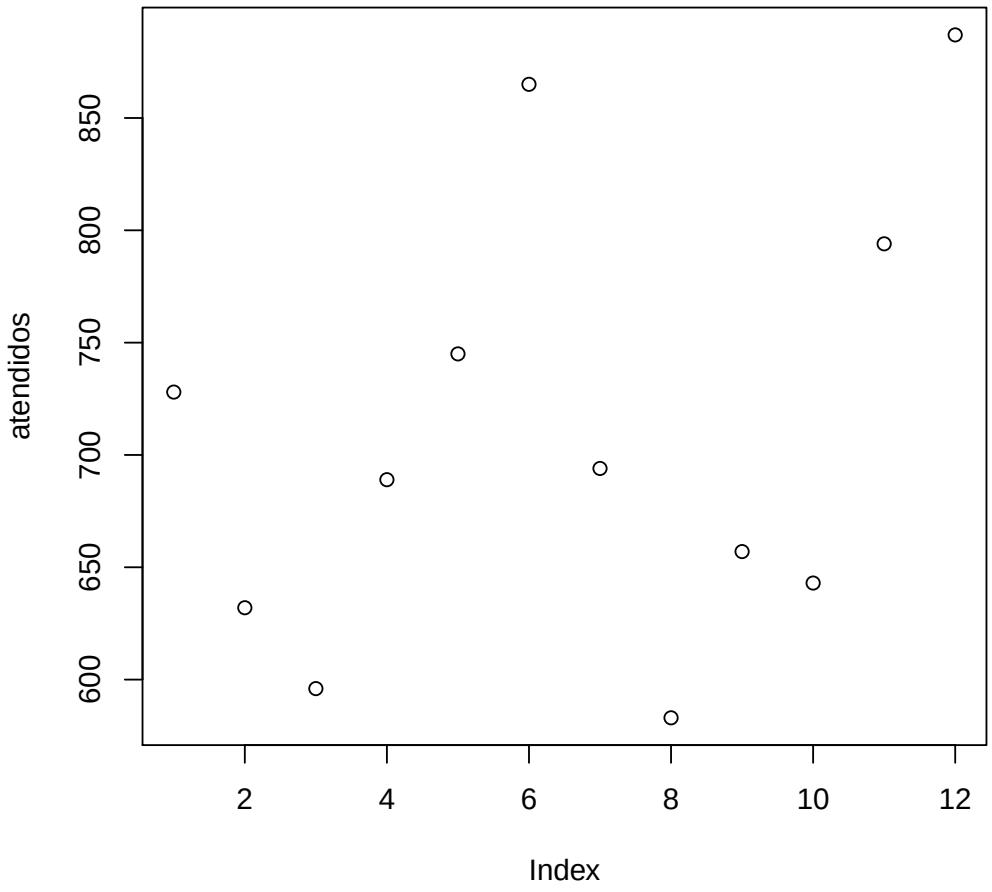
1. Programación orientada a objetos en R.

Para realizar un uso eficiente del programa R es preciso entender y aprender a manipular bien las clases de los distintos objetos que maneja, ya que R utiliza *programación orientada a objetos*. Esto significa que *una misma función hace cosas distintas según la clase del objeto que recibe como argumento*, pudiendo incluso no hacer nada (o dar error) si se le pasan argumentos de una clase inadecuada. A modo de ejemplo, veamos como la función `plot()` puede mostrar distintos gráficos según la clase del objeto a representar. Para ello, supongamos que el siguiente vector representa el número de personas atendidas mensualmente en el servicio de urgencias de un centro de salud durante el pasado año (datos de enero a diciembre):

```
> atendidos <- c(728, 632, 596, 689, 745, 865, 694, 583, 657,
     643, 794, 887)
> class(atendidos)
[1] "numeric"
```

La función `class()` nos devuelve la *clase* del objeto `atendidos`, que como vemos es un vector numérico. Podemos obtener una representación gráfica de este vector simplemente mediante:

```
> plot(atendidos)
```

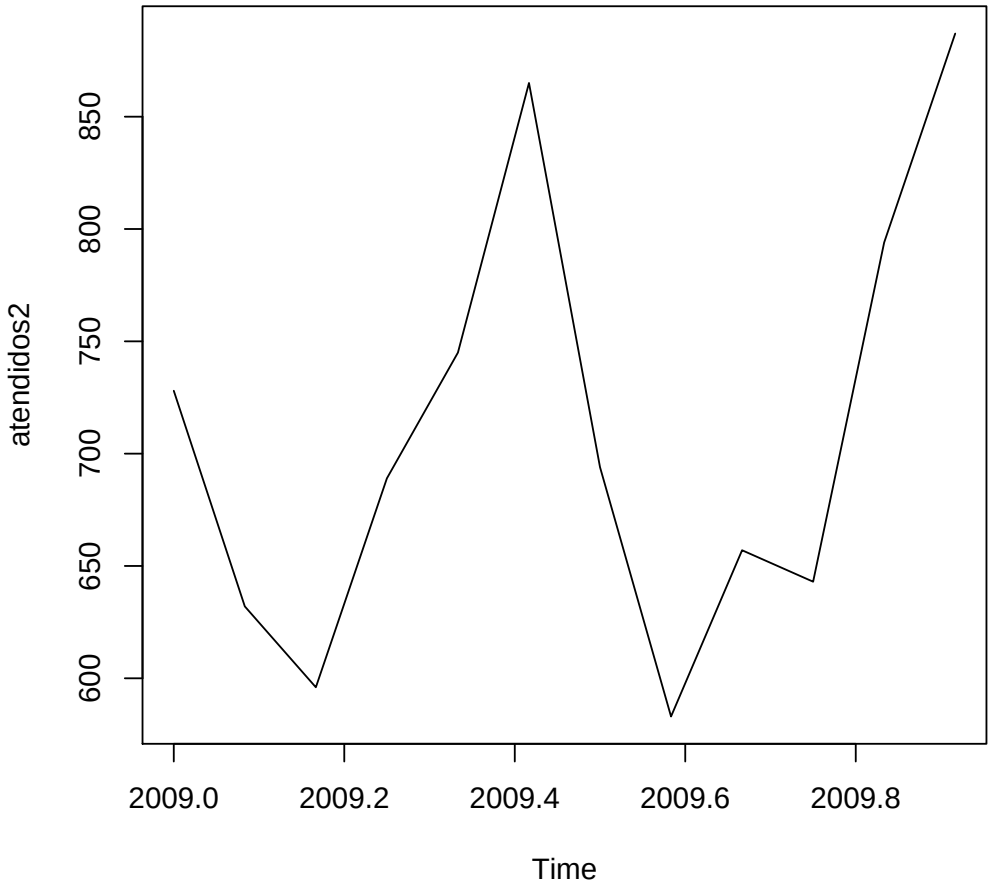


Ahora convertimos estos datos en *serie temporal* mediante la función `ts()`, indicando que esta serie comienza en enero del año 2009 y tiene una frecuencia de 12 observaciones por año (esto es, una por mes):

```
> atendidos2 <- ts(atendidos, frequency = 12, start = c(2009,
    1))
> atendidos2
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
2009 728 632 596 689 745 865 694 583 657 643 794 887
> class(atendidos2)
[1] "ts"
```

Como podemos ver, la clase del objeto `atendidos2` es `ts` (*time series*). Podemos comprobar que si aplicamos la misma función `plot()` a `atendidos2`, el gráfico obtenido es distinto que cuando se aplica a `atendidos`, aún cuando los datos sean exactamente los mismos:

```
> plot(atendidos2)
```



2. Factores

Un *factor* es una variable categórica. En R los factores se utilizan habitualmente para realizar clasificaciones de los datos, estableciendo su pertenencia a los grupos o categorías determinados por los niveles (valores) del factor. Un factor puede ser de tipo numérico o de tipo carácter[1]. A modo de ejemplo, la variable `sexo` que ya utilizamos en nuestra primera sesión con R :

```
> sexo <- c("M", "H", "M", "M", "M", "H", "M", "M", "H", "H")
```

puede ser considerada un factor, ya que establece para cada sujeto su pertenencia a una de las dos categorías *Hombre* o *Mujer*. Para que R reconozca al sexo como factor, una vez introducidos los datos utilizamos la función:

[1] Independientemente de que el factor sea numérico o carácter, sus valores son siempre almacenados internamente por R como números enteros, con lo que se consigue economizar memoria.

```
> sexo <- factor(sexo)
> sexo
 [1] M H M M M H M M H H
Levels: H M
```

con lo que hemos convertido `sexo` en un factor con dos niveles `M` y `H`. En muchos casos, los niveles del factor son poco ilustrativos de su significado. La siguiente sintaxis especifica explícitamente los niveles del factor (*levels*) y asigna etiquetas (*labels*) a cada uno de ellos:

```
> sexo <- factor(sexo, levels = c("H", "M"), labels = c("Hombre",
     "Mujer"))
```

Estas etiquetas aparecerán en los resultados de los procedimientos estadísticos donde apareza el factor, aclarando su significado. Por ejemplo, si pedimos a R que nos construya la tabla de frecuencias de sexos, en lugar de `H` o `M` nos mostrará los términos *Hombre* o *Mujer:*

```
> table(sexo)
sexo
Hombre  Mujer
     4      6
```

3. Vectores. Asignación y selección de los valores de un vector.

Ya hemos visto en nuestra primera sesión (ver tabla 3.1) que el operador para asignar valores a un vector es `<-`. Este operador puede utilizarse de varias formas equivalentes:

```
> edad <- c(22, 34, 29, 25, 30, 33, 31, 27, 25, 25)
```

También puede utilizarse indistintamente el símbolo "=":

```
> edad = c(22, 34, 29, 25, 30, 33, 31, 27, 25, 25)
```

Si deseamos asignar a una variable una sucesión de valores consecutivos podemos utilizar el operador `":"`. Así para asignar al vector x los valores de 1 a 10 procederíamos del siguiente modo:

```
> x = 1:10
> x
 [1]  1  2  3  4  5  6  7  8  9 10
```

Podemos construir secuencias de números más complejas mediante la función `seq()`:

- Sucesión de valores de 1 a 20 de 2 en 2:

```
> x = seq(1, 20, by = 2)
> x
 [1]  1  3  5  7  9 11 13 15 17 19
```

- Sucesión de 8 valores equiespaciados entre 1 y 20:

```
> y = seq(1, 20, length = 8)
> y
[1]  1.000  3.714  6.429  9.143 11.857 14.571 17.286 20.000
```

Es posible acceder al valor que ocupa la posición k dentro de un vector x refiriéndonos a él como `x[k]`:

```
> x[3]
[1] 5
> y[5]
[1] 11.86
```

Podemos acceder también simultáneamente a varios valores dentro de un vector. Por ejemplo, si deseamos ver del segundo al quinto de los valores observados en la variable `edad`:

```
> edad[2:5]
[1] 34 29 25 30
```

Y si quisiéramos ver sólo los valores primero, tercero y séptimo:

```
> edad[c(1, 3, 7)]
[1] 22 29 31
```

4. Selección condicionada.

La función `which()` nos da las posiciones, dentro de un vector, de los valores que cumplen cierta condición. Por ejemplo:

```
> which(edad > 25)
[1] 2 3 5 6 7 8
```

nos indica que los valores del vector `edad` mayores que 25 son los que ocupan las posiciones 2, 3, 5, 6, 7 y 8. Podemos ver cuáles son concretamente esos valores mediante:

```
> edad[which(edad > 25)]
[1] 34 29 30 33 31 27
```

Esta expresión puede simplificarse; si no utilizamos `which()` obtenemos exactamente el mismo resultado:

```
> edad[edad > 25]
[1] 34 29 30 33 31 27
```

Se puede realizar también la selección de valores de un vector condicionando por los valores de otro vector. Por ejemplo, si las diez edades del ejemplo anterior corresponden a personas cuyo sexo viene dado por:

```
> sexo <- c("M", "H", "H", "M", "M", "H", "M", "M", "H", "H")
```

podríamos seleccionar la edad de las mujeres simplemente mediante:

```
> edad[sexo == "M"]
[1] 22 25 30 31 27
```

o podríamos seleccionar el sexo de aquellos que han hablado más de 13 minutos diarios mediante:

```
> sexo[tiempo > 13]
[1] "M" "M" "M"
```

5. Aritmética vectorial.

R utiliza los operadores aritméticos elementales: suma (+), resta (-), multiplicación (*) y división (/). Cuenta asimismo con un catálogo muy completo de funciones matemáticas. Por citar unas pocas: logaritmo neperiano (*log*), exponencial (*exp*), seno (*sin*), coseno (*cos*), valor absoluto (*abs*), parte entera (*floor*), redondeo (*round*).

Una característica importante de R es su capacidad para la *aritmética vectorial*. Esto significa que cuando una función o un operador se aplica a un vector, dicha operación o función se ejecuta sobre todos los componentes del vector. Veamos algunos ejemplos:

- Necesitamos transformar la variable `tiempo` a escala logarítmica y guardar el resultado en una variable llamada `logtime`. En R esto es tan simple como hacer lo siguiente:

```
> options(width = 60)
```

```
> logtime = log(tiempo)
> logtime
 [1] 2.654 2.338 2.476 2.625 2.487 2.397 2.524 2.593 2.509
[10] 2.478
```

- Queremos *tipificar* los valores de la variable tiempo (restarles su media y dividir por su desviación típica). La variable tiempo tipificada se obtiene fácilmente en R mediante:

```
> ztiempo = (tiempo - mean(tiempo))/sd(tiempo)
> ztiempo
 [1]  1.5626 -1.6459 -0.3709  1.2292 -0.2542 -1.1209  0.1208
 [8]  0.8626 -0.0375 -0.3459
```

- Queremos pasar la variable *tiempo* que está en minutos, a segundos. Si hacemos:

```
> tiempo_seg = 60 * tiempo
> tiempo_seg
 [1] 852.6 621.6 713.4 828.6 721.8 659.4 748.8 802.2 737.4
[10] 715.2
```

comprobamos que se han multiplicado por 60 todos los valores de tiempo.

6. Matrices

Hay varias maneras de definir una matriz en R. Si es una matriz pequeña podemos utilizar la siguiente sintaxis:

```
> A = matrix(nrow = 3, ncol = 3, c(1, 2, 3, 4, 5,
     6, 7, 8, 9), byrow = TRUE)
```

Con el argumento `nrow` hemos indicado el número de filas de nuestra matriz, con `ncol` el número de columnas; a continuación hemos puesto los valores que forman la matriz (los valores del 1 al 9), y le hemos pedido a R que use esos valores para rellenar la matriz A *por filas* (`byrow=TRUE`). La matriz A así construida es:

```
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

Si disponemos de varios vectores de la misma longitud que queremos utilizar como filas (o columnas) de una matriz, podemos utilizar la función `rbind()` para *unirlos por filas* o la función `cbind()` para *unirlos por columnas*, como vemos en el siguiente ejemplo:

```
> vector1 = c(1,2,3,4)
> vector2 = c(5,6,7,8)
> vector3 = c(9,10,11,12)
> M1 = cbind(vector1,vector2,vector3) # Unimos por columnas
> M1
     vector1 vector2 vector3
[1,]       1       5       9
[2,]       2       6      10
[3,]       3       7      11
[4,]       4       8      12
> M2 = rbind(vector1,vector2,vector3) # Unimos por filas
> M2
        [,1] [,2] [,3] [,4]
vector1    1    2    3    4
vector2    5    6    7    8
vector3    9   10   11   12
```

Se pueden seleccionar partes de una matriz utilizando los índices de posición $[fila, columna]$ entre corchetes. El siguiente ejemplo ilustra la forma de hacerlo:

```
> A[2,3]   # Se selecciona el valor de la fila 2, columna 3
[1] 6
> A[2,]    # Se selecciona la fila 2 completa
[1] 4 5 6
> A[,3]    # Se selecciona la columna 3 completa
[1] 3 6 9
> A[1,2:3] # Se seleccionan el segundo y tercer valor de la fila 1
[1] 2 3
```

6.1. Operaciones con matrices

La función `diag()` extrae la diagonal principal de una matriz:

```
> diag(A)
[1] 1 5 9
```

También permite crear matrices diagonales:

```
> diag(c(1, 2, 3, 4))
     [,1] [,2] [,3] [,4]
[1,]    1    0    0    0
[2,]    0    2    0    0
[3,]    0    0    3    0
[4,]    0    0    0    4
```

Hay que tener cierto cuidado con los operadores aritméticos básicos. Si se suman una matriz y una constante, el efecto es que dicha constante se suma a todos los elementos de la matriz. Lo mismo ocurre con la diferencia, la multiplicación y la división:

```
> M = matrix(nrow = 2, c(1, 2, 5, 7), byrow = F)
> M
     [,1] [,2]
[1,]    1    5
[2,]    2    7
```

```
> M + 2
     [,1] [,2]
[1,]    3    7
[2,]    4    9
> M - 2
     [,1] [,2]
[1,]   -1    3
[2,]    0    5
> M * 2
     [,1] [,2]
[1,]    2   10
[2,]    4   14
> M/2
     [,1] [,2]
[1,]  0.5  2.5
[2,]  1.0  3.5
```

Asimismo, si a una matriz se le suma un vector cuya longitud sea igual al número de filas de la matriz, se obtiene como resultado una nueva matriz cuyas columnas son la suma de las columnas de la matriz original más dicho vector. Lo mismo ocurre con la diferencia, la multiplicación y la división:

```
> v = c(3, 4)
> M + v
     [,1] [,2]
[1,]    4    8
[2,]    6   11
> M - v
     [,1] [,2]
[1,]   -2    2
[2,]   -2    3
> M * v
     [,1] [,2]
[1,]    3   15
[2,]    8   28
> M/v
```

```
        [,1]  [,2]
[1,] 0.3333 1.667
[2,] 0.5000 1.750
```

La suma o resta de matrices de la misma dimensión se realiza con los operadores + y -; el producto de matrices (siempre que sean compatibles) se realiza con el símbolo `%*%`:

```
> M + M
     [,1] [,2]
[1,]    2   10
[2,]    4   14
> M - M
     [,1] [,2]
[1,]    0    0
[2,]    0    0
> M %*% M
     [,1] [,2]
[1,]   11   40
[2,]   16   59
```

Una fuente de posibles errores en el cálculo matricial, cuando se utilizan matrices de la misma dimensión, es utilizar los operadores `*` y `/` ya que no realizan el producto matricial, sino que multiplican las matrices término a término:

```
> M * M
     [,1] [,2]
[1,]    1   25
[2,]    4   49
> M/M
     [,1] [,2]
[1,]    1    1
[2,]    1    1
```

6.2. Potencia de una matriz.

R no dispone en su paquete base de una función para calcular la potencia n-ésima de una matriz. No obstante el paquete expm define el operador potencia `%^%` para matrices. Su uso es análogo al cálculo de la potencia de un número: si queremos calcular la matriz A^n bastará con utilizar el comando `A%^%n`. Debemos recordar cargar primero el paquete expm:

```
> library(expm)
> A = matrix(1:4, nrow = 2)
> A
     [,1] [,2]
[1,]    1    3
[2,]    2    4
> A %*% A
     [,1] [,2]
[1,]    7   15
[2,]   10   22
> A %^% 2
     [,1] [,2]
[1,]    7   15
[2,]   10   22
> A %^% 5
     [,1] [,2]
[1,] 1069 2337
[2,] 1558 3406
```

6.3. Sistemas de ecuaciones lineales. Matrices inversas.

R dispone de la función `solve()` para resolver sistemas lineales de ecuaciones. En particular, si el sistema a resolver es de la forma $Ax = b$ (donde b podría ser también una matriz), su solución es $x = A^{-1}b$, que en R se obtiene mediante `solve(A,b)`. En particular, si b es la matriz identidad, la función anterior devuelve la matriz inversa de A:

```
> A=matrix(ncol=3,c(2,4,3,5,7,1,2,2,3),byrow=TRUE)
> A
```

```
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    5    7    1
[3,]    2    2    3
```

```
> I=diag(1,nrow=3) # Matriz diagonal de dimension 3
> solve(A,I) # Matriz inversa de A
```

```
        [,1]    [,2]    [,3]
[1,] -0.7308  0.2308  0.6538
[2,]  0.5000  0.0000 -0.5000
[3,]  0.1538 -0.1538  0.2308
```

Podemos obviar la referencia a la matriz identidad `I`. Si utilizamos solamente `solve(A)` obtenemos también la inversa de `A`:

```
> solve(A)
```

```
        [,1]    [,2]    [,3]
[1,] -0.7308  0.2308  0.6538
[2,]  0.5000  0.0000 -0.5000
[3,]  0.1538 -0.1538  0.2308
```

6.4. Autovalores y autovectores.

El comando `eigen()` permite calcular de manera sencilla los autovalores y autovectores de una matriz:

```
> eigen(A)
```

```
$values
[1] 10.531  2.469 -1.000

$vectors
       [,1]    [,2]    [,3]
[1,] 0.4950  0.3348  0.8452
[2,] 0.7982 -0.5399 -0.5071
[3,] 0.3434  0.7722 -0.1690
```

7. Data.frames

El término *data.frame* es difícil de traducir al castellano. Podría traducirse como *Hoja de datos* o *Marco de datos.* Los *data.frames* son una clase de objetos especial en R. Normalmente, cuando se realiza un estudio estadístico sobre los sujetos u objetos de una muestra, la información se organiza precisamente en un *data.frame:* una hoja de datos, en los que cada fila corresponde a un sujeto y cada columna a una variable. Así, el conjunto de datos que ya vimos en la tabla 2.1 de la sección 1 constituye un data.frame tal como se muestra en la tabla 5.1.

edad	tiempo	sexo
22	14.21	M
34	10.36	H
29	11.89	H
25	13.81	M
30	12.03	M
33	10.99	H
31	12.48	M
27	13.37	M
25	12.29	H
25	11.92	H

Tabla 5.1: Datos de la tabla 2.1 organizados en forma de *data.frame*

La estructura de un *data.frame* es muy similar a la de una matriz. La diferencia es que una matriz sólo admite valores numéricos, mientras que en un *data.frame* podemos incluir también datos alfanuméricos. El siguiente ejemplo nos muestra como crear un *data.frame* con las variables de la tabla 5.1:

```
> edad <- c(22, 34, 29, 25, 30, 33, 31, 27, 25,
     25)
> tiempo <- c(14.21, 10.36, 11.89, 13.81, 12.03,
     10.99, 12.48, 13.37, 12.29, 11.92)
> sexo <- c("M", "H", "H", "M", "M", "H", "M", "M",
     "H", "H")
> misdatos <- data.frame(edad, tiempo, sexo)
> misdatos
   edad tiempo sexo
1    22  14.21    M
2    34  10.36    H
3    29  11.89    H
4    25  13.81    M
```

```
5    30  12.03    M
6    33  10.99    H
7    31  12.48    M
8    27  13.37    M
9    25  12.29    H
10   25  11.92    H
```

```
> str(misdatos)
```

```
'data.frame':        10 obs. of  3 variables:
 $ edad  : num  22 34 29 25 30 33 31 27 25 25
 $ tiempo: num  14.2 10.4 11.9 13.8 12 ...
 $ sexo  : Factor w/ 2 levels "H","M": 2 1 1 2 2 1 2 2 1 1
```

```
> names(misdatos)
```

```
[1] "edad"   "tiempo" "sexo"
```

En este ejemplo hemos creado un *data.frame* llamado `misdatos` que contiene a las tres variables `edad`, `tiempo` y `sexo`. La función `str()` nos muestra la estructura de este objeto, confirmándonos que es un *data.frame* de tres variables con 10 observaciones cada una. Nos informa además de que las dos primeras variables son numéricas y la tercera, el sexo, es un *factor* con dos valores, *"V"* y *"M"*. La función `names()` por su parte, nos devuelve los nombres de las variables contenidas en *misdatos.*

Como veremos en la siguiente sección, cuando desde R leemos datos situados en un fichero externo (un fichero de texto, una hoja excel, un archivo de datos de SPSS,...), estos datos se importan en un *data.frame*, sobre el cual podemos realizar nuestro estudio estadístico.

El acceso a los datos que se encuentran en un *data.frame* es muy similar al acceso a los datos de una matriz que ya vimos en la sección anterior. Sin embargo, para los *data.frames* R dispone de algunas funciones que facilitan la tarea de seleccionar o filtrar datos. Así por ejemplo, si queremos ver sólo los datos de los sujetos 3 a 6, escribiríamos:

```
> misdatos[3:6, ]
```

```
  edad tiempo sexo
3   29  11.89    H
4   25  13.81    M
5   30  12.03    M
6   33  10.99    H
```

Si queremos seleccionar los datos de edad (primera columna), podemos tratar a `misdatos` igual que si fuese una matriz:

```
> misdatos[, 1]
```

```
 [1] 22 34 29 25 30 33 31 27 25 25
```

Aunque también podemos referirnos a la columna por su nombre:

```
> misdatos$edad
 [1] 22 34 29 25 30 33 31 27 25 25
```

Nótese que en este caso hemos de utilizar el nombre del *data.frame* (en este caso `misdatos`) seguido del símbolo `$` y del nombre de la variable que nos interesa (`edad`).

La función `subset()` nos permite seleccionar una parte del *data.frame*. Por ejemplo, si deseamos quedarnos sólo con los hombres utilizaríamos:

```
> hombres = subset(misdatos, sexo == "H")
> hombres
   edad tiempo sexo
2    34  10.36    H
3    29  11.89    H
6    33  10.99    H
9    25  12.29    H
10   25  11.92    H
```

Podemos elaborar selecciones más complejas; por ejemplo:

- Sujetos que sean hombres y tengan más de 30 años (la condición "y" se especifica mediante el símbolo "&"):

```
> mayores = subset(misdatos, sexo == "H" & edad >
    30)
> mayores
  edad tiempo sexo
2   34  10.36    H
6   33  10.99    H
```

- Hombres que tengan menos de 25 años y hayan hablado por el móvil más de 12 minutos diarios:

```
> jov_habladores = subset(misdatos, sexo == "H" &
    edad < 20 & tiempo > 12)
> jov_habladores
```

```
[1] edad   tiempo sexo
<0 rows> (or 0-length row.names)
```

- Sujetos que tengan menos de 25 o más 30 años (la condición "o" se expresa mediante la línea vertical "|"):

```
> extremos = subset(misdatos, edad < 25 | edad >
    30)
> extremos
  edad tiempo sexo
1   22  14.21    M
2   34  10.36    H
6   33  10.99    H
7   31  12.48    M
```

Podemos seleccionar además un subconjunto de variables del *data.frame.* Por ejemplo, si nos interesan solo la edad y el tiempo de uso del móvil de los hombres de la muestra:

```
> hombres = subset(misdatos, sexo == "H", select = c(edad,
    tiempo))
> hombres
   edad tiempo
2    34  10.36
3    29  11.89
6    33  10.99
9    25  12.29
10   25  11.92
```

8. Listas

La *lista* es la estructura de datos más compleja que maneja R. Podemos entender una lista como un contenedor de objetos que pueden ser de cualquier clase: números, vectores, matrices, funciones, data.frames, incluso otras listas. Una lista puede contener a la vez varios de estos objetos, que pueden ser además de distintas dimensiones.

Ya hemos visto una lista en la sección 2 cuando pedimos a R que nos mostrase la estructura del objeto `regresion`. Podemos crear ahora una lista que contenga el *data.frame* `misdatos`, la matriz `A`, el vector `x=c(1,2,3,4)` y la constante `e=exp(1)`:

```
> MiLista <- list(misdatos, A, M = M, x = c(1, 2,
     3, 4), e = exp(1))
> MiLista
[[1]]
   edad tiempo sexo
1    22  14.21    M
2    34  10.36    H
3    29  11.89    H
4    25  13.81    M
5    30  12.03    M
6    33  10.99    H
7    31  12.48    M
8    27  13.37    M
9    25  12.29    H
10   25  11.92    H

[[2]]
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    5    7    1
[3,]    2    2    3

$M
     [,1] [,2]
[1,]    1    5
[2,]    2    7

$x
[1] 1 2 3 4

$e
[1] 2.718
> MiLista$misdatos
NULL
> MiLista[[1]]
   edad tiempo sexo
1    22  14.21    M
```

```
2    34  10.36    H
3    29  11.89    H
4    25  13.81    M
5    30  12.03    M
6    33  10.99    H
7    31  12.48    M
8    27  13.37    M
9    25  12.29    H
10   25  11.92    H
```

```
> MiLista$M
```

```
     [,1] [,2]
[1,]    1    5
[2,]    2    7
```

```
> MiLista$x
```

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

```
> MiLista$e
```

```
[1] 2.718
```

Como vemos, para acceder a los objetos que forman parte de una lista, basta con añadir su nombre a continuación del de la lista, separados por el símbolo `$`, o bien con el índice de posición dentro de la lista con doble corchete `[[]]`. Nótese que los objetos `misdatos` y `A` no tienen nombre dentro de la lista, por lo que hemos de referirnos a ellos como `MiLista[[1]]` o `MiLista[[2]]`. Sin embargo, el objeto `M` sí que tiene nombre. Para que un objeto dentro de una lista tenga nombre, éste debe declararse explícitamente en la construcción de la lista, tal como se hizo con `M`, `x` o `e`.

R utiliza las listas, sobre, todo como salida de los distintos procedimientos estadísticos. Así, por ejemplo, al realizar un contraste de las medias de dos poblaciones, R nos calcula, entre otras cosas, la diferencia de medias muestrales, el valor del estadístico de contraste, el p-valor del test y el intervalo de confianza para la diferencia observada. Todos estos términos forman parte de una lista. La sintaxis para comparar, por ejemplo, el tiempo medio de uso del móvil entre hombres y mujeres a partir de nuestros datos sería:

```
> t.test(tiempo ~ sexo, data = misdatos)
```

```
        Welch Two Sample t-test

data:  tiempo by sexo
t = -3.133, df = 7.854, p-value = 0.01427
```

```
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -2.9378 -0.4422
sample estimates:
mean in group H mean in group M
          11.49           13.18
```

Si guardamos el resultado de este contraste en el objeto `contraste`, podemos observar que tiene estructura de lista:

```
> contraste = t.test(tiempo ~ sexo, data = misdatos)
> str(contraste)
List of 9
 $ statistic  : Named num -3.13
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 7.85
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 0.0143
 $ conf.int   : atomic [1:2] -2.938 -0.442
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num [1:2] 11.5 13.2
  ..- attr(*, "names")= chr [1:2] "mean in group H" "mean in group M"
 $ null.value : Named num 0
  ..- attr(*, "names")= chr "difference in means"
 $ alternative: chr "two.sided"
 $ method     : chr "Welch Two Sample t-test"
 $ data.name  : chr "tiempo by sexo"
 - attr(*, "class")= chr "htest"
```

Capítulo 6

Acceso a datos almacenados en archivos externos.

En las secciones anteriores hemos visto como introducir datos en R utilizando la función de concatenación `c()`. Obviamente esta forma de crear conjntos de datos sólo es eficaz si el número de valores a introducir es pequeño. Si hemos de manejar bases de datos muy grandes resulta más conveniente utilizar programas específicamente diseñados para ello: gestores de hojas de cálculo como Excel u OpenOffice Calc, bases de datos como MySQL o Access, o incluso los editores de datos que proporcionan SPSS o SAS. Para acceder a estos datos desde R existen fundamentalmente dos maneras:

- Que el programa que hemos utilizado para crear la base de datos la exporte a un formato *"libre"*, normalmente texto ASCII plano, que puede ser leido directamente por R.
- Que carguemos en R alguna librería que sea capaz de entender el formato en que se han guardado los datos. Como veremos en esta sección, R dispone de librerías que le permiten cargar directamente datos guardados en los formatos específicos de los programas citados más arriba.

1. Establecimiento del directorio de trabajo en R

Cuando vamos a utilizar R para leer datos de ficheros externos, es conveniente decirle a R desde el principio cuál es el directorio donde están los datos. Habitualmente, éste será el *directorio de trabajo*: no sólo contiene los datos, sino también guardaremos en él los resultados de nuestro trabajo, los gráficos que podamos generar, etc. Si nuestro directorio de trabajo es, por ejemplo, `c:\usuario\Mis Documentos\Mis Trabajos R`, utilizaremos la función `setwd()` (*set working directory*) para indicar a R que es en ese directorio donde se realizarán las operaciones de entrada/salida de datos/-resultados durante nuestra sesión de trabajo:

```
> setwd("c:/usuario/Mis Documentos/Mis Trabajos R")
```

Es importante tener en cuenta que:

- Las barras que separan los distintos subdirectorios que componen la ruta son de la forma "/" y no la habitual "\" de Windows[1]. Alternativamente, se puede utilizar también doble barra "\\".
- El nombre del directorio debe especificarse entre comillas.

El directorio de trabajo puede modificarse en cualquier momento de nuestra sesión. Basta con ejecutar de nuevo la función `setwd()` indicando un nuevo directorio.

2. Lectura y escritura de datos en formato .csv

El formato `csv` (siglas de *Comma Separated Values*) es un formato estándar para el almacenamiento de datos. Básicamente un archivo.*csv* es un archivo ASCII (texto plano) en el que cada fila corresponde a un caso, esto es, a un sujeto u objeto sobre el que se han observado varias variables; y en cada fila, los valores de las distintas variables se introducen separados por *punto y coma*[2]. Asimismo los valores de variables alfanuméricas deben escribirse entre comillas. Usualmente, en la primera fila de un archivo.csv se escriben los nombres de las variables que contiene, entre comillas y separados también por punto y coma.

La figura 1 muestra el contenido de un archivo.csv que contiene los cinco primeros datos de edad, tiempo de uso del móvil y sexo ya vistos en la tabla 2.1.

```
"edad"; "tiempo"; "sexo"
22; 14.21; "M"
34; 10.36; "H"
29; 11.89; "H"
25; 13.81; "M"
30; 12.03; "M"
```

Figura 1: Contenido de un archivo en formato.csv

Si tuviésemos que utilizar un editor de texto (el *Notepad* de Windows, por ejemplo, o *gedit* en Linux) para escribir datos en formato *csv,* es obvio que no habríamos avanzado mucho respecto a introducir directamente los datos en la consola de R con la función de concatenación. Sin embargo muchos programas tipo hoja de cálculo -en particular *Excel* y *OpenOffice Calc*[3]- permiten guardar

[1] Esto es herencia del origen de R en sistemas Unix/Linux, en los que se usa la barra de la forma "/" para separar los directorios de una ruta. Obviamente, si hemos instalado R en Linux (o Mac), esta manera de escribir la ruta de un directorio es la natural.

[2] En realidad en la definición original de los ficheros csv, los valores de las variables se separan por comas. Ahora bien, en España y otros países la coma se reserva para separar los decimales por lo que, para evitar confusiones, en los ficheros csv "españoles" las variables deben separarse por punto y coma.

[3] Señalemos que estos dos programas, si bien son los más populares, no son los únicos que manejan hojas de cálculo y son capaces de generar ficheros csv. Podemos citar también *Gnumeric*, integrado en Gnome Office de Linux, *Numbers*, integrado en iWork de Apple, *Lotus 1-2-3,* integrado en Lotus SmartSuite, *Corel Quattro Pro,* integrado en WordPerfect, o *KSpread,* integrado en KOffice, también de Linux.

los datos en este formato, sin que tengamos que preocuparnos de escribir los puntos y comas o las comillas, ya que el programa lo hace por nosotros; asimismo si con uno de estos programas abrimos un archivo.csv, dicho archivo se abre como hoja de cálculo, con lo que su manipulación (introducción de datos, correcciones,...) resulta muy sencilla.

Para leer con R un archivo *csv* podemos utilizar la función `read.table()`. Si el fichero anterior se llama `usomovil.csv`, (y ya nos hemos situado en el directorio de trabajo mediante `setwd()` la sintaxis para su lectura[4] es:

```
> movildata <- read.table(file = "usomovil.csv",
     header = TRUE, sep = ";", dec = ",")
```

De esta forma, los datos del archivo se asignan a un *data.frame* con el nombre `movildata`. En esta función hemos utilizado cuatro argumentos:

- `file`: es el nombre del archivo. Debe escribirse entre comillas.
- `header`: es TRUE si los nombres de las variables están escritos en la cabecera (primera linea) del archivo. En caso contrario, se declara como FALSE.
- `sep`: es el símbolo que se usa para separar las variables. En nuestro caso es el punto y coma (";"). Debe ir entre comillas.
- `dec`: es el símbolo que se usa para separar los decimales. En nuestro caso es la coma (","). También va entre comillas.

Si como resultado de nuestro trabajo se ha generado un conjunto de datos llamado `results` y queremos guardarlo en el fichero `resultado.csv`, bastará con utilizar la sintaxis:

```
> write.table(results, file = "resultado.csv", sep = ";",
     dec = ",", row.names = FALSE, col.names = TRUE)
```

Hemos especificado `col.names=TRUE` para que se escriban los nombres de las variables en la cabecera de las columnas, y `row.names=FALSE` para que *no* se escriban los nombres o números de fila.

3. Las funciones `attach()` y `detach()`

Una vez que los datos de un archivo han sido leidos y convertidos en un *data.frame*, su manipulación puede realizarse tal como hemos visto en la sección 7. En particular, para acceder a los datos de una

[4] Señalemos que a veces podremos encontrarnos con ficheros *csv* al estilo anglosajón, con los decimales separados por puntos y las variables por comas. En tal caso en nuestra función `read.table()` deberemos especificar `dec="."` y `sep=","`.

variable, su nombre ha de ir precedido por el nombre del *data.frame* y separado de éste por el símbolo $. Esto puede resultar tedioso si vamos a llevar a cabo un cierto número de procedimientos estadísticos utilizando estas variables y constantemente hemos de referirnos a ellas de esta forma. Para resolver este problema podemos usar la función `attach()`, que activa el *accceso directo* a las variables por su nombre.

Supongamos que, como se indicó en la sección anterior, hemos leido el archivo `usomovil.csv` y hemos asignado su contenido al *data.frame* `movildata`. Si ahora ejecutamos:

```
> attach(movildata)
```

podremos acceder a las variables utilizando directamente su nombre `edad`, `tiempo` o `sexo` sin que sea necesario utilizar `movildata$edad`, `movildata$tiempo` o `movildata$sexo`. Una vez que hayamos terminado de analizar estos datos, podemos desactivar el acceso directo mediante:

```
> detach(movildata)
```

Debe tenerse en cuenta que si estamos trabajando con varios *data.frames* que contienen variables de igual nombre, las variables a las que se tiene acceso directo son las que corresponden al último *data.frame* sobre el que se haya realizado el `attach()`. Así, por ejemplo, si tenemos un *data.frame* `mvdata1` con edad, tiempo y sexo de una muestra de sujetos, y otro *data.frame* `mvdata2` con datos de las mismas variables medidos en una segunda muestra, la sintaxis:

```
> attach(mvdata1)
> attach(mvdata2)
> mean(tiempo)
```

mostraría el tiempo medio de uso del móvil por parte de los sujetos de la segunda muestra. Si ahora escribimos:

```
> detach(mvdata2)
> mean(tiempo)
```

la linea `detach(mvdata2)` desactiva el acceso directo a las variables de `mvdata2`; como no se ha realizado el `detach()` de `mvdata1`, la linea `mean(tiempo)` nos mostrará el tiempo medio de uso del móvil de los sujetos de la muestra 1 que es la que ahora está "activa".

4. Lectura de datos en formato.sav de SPSS

Para leer archivos de datos generados por SPSS (usualmente con la extensión.*sav*), hemos de cargar la librería[5] `foreign`:

[5] Si no tenemos esta librería deberemos instalarla tal como hemos visto en la sección4.

```
> library(foreign)
```

y a continuación proceder a la lectura del archivo en cuestión. Si el archivo que queremos leer se llama `traffic.sav` escribiremos:

```
> trafico <- read.spss(file = "traffic.sav", to.data.frame = TRUE)
```

De esta forma los datos del archivo se almacenan en `trafico`. Es recomendable utilizar el argumento `to.data.frame=TRUE` para que el objeto `trafico` en el que hemos puesto los datos tenga estructura de *data.frame.* En caso contrario su estructura sería la de una *lista*, lo que puede dar lugar a algunos efectos colaterales indeseados.

5. Lectura/Escritura de datos en formato .xls y .xlsx de Excel

Como ya hemos señalado en la sección 2, Excel (en cualquiera de sus versiones) permite guardar las hojas de cálculo en formato csv, que R lee directamente a través de la función `read.table()`. Ahora bien, al guardar en este formato, Excel sólo guarda la hoja activa, con lo que si tenemos un Libro Excel con muchas hojas de cálculo, tendríamos que guardarlas por separado en múltiples archivos csv, lo que puede resultar trabajoso (amén de lo confuso que podría llegar a resultar un directorio lleno de archivos csv). Por ello en estos casos resulta ventajoso utilizar alguna librería que permita, desde dentro de R, acceder al Libro Excel y seleccionar directamente cada hoja en la que estamos interesados.

Hay tres librerías que permiten que R lea directamente datos en los formatos que usa Excel. Aclaremos en primer lugar que a partir de Excel 2007 el formato por defecto de los archivos generados por este programa es.xlsx. En las versiones anteriores, el formato es.xls. Para las versiones anteriores a la 2007, las librerías que permiten acceder a los archivos.xls son `xlsReadWrite` y `RODBC`. Para acceder a archivos.xlsx de Excel 2007 y posteriores, puede utilizarse la librería `XLConnect` que permite leer y guardar archivos Excel de manera muy cómoda[6].

5.1. La librería XLConnect.

Leer datos Excel con esta librería es muy simple. En primer lugar la cargamos en memoria:

```
> library(XLConnect)
```

Si queremos leer la primera hoja de cálculo del libro "prueba.xlsx" podemos hacerlo mediante:

```
> datos <- readWorksheetFromFile("prueba.xlsx",
     sheet = 1)
```

[6]Esta librería funciona a través de Java, por lo que en algunos casos puede dar problemas si Java no se encuentra correctamente instalado en el ordenador

Si queremos leer una hoja concreta, llamada "hojaDatos" especificamos su nombre:

```
> datos <- readWorksheetFromFile("prueba.xlsx",
     sheet = "hojaDatos")
```

Si vamos a leer varias hojas de un mismo libro resulta más conveniente utilizar la siguiente sintaxis, en la que se "lee" el libro Excel de una vez, y a continuación se van construyendo sucesivos data.frames con las hojas de interés:

```
> xlsfile = loadWorkbook("prueba.xlsx")
> datos1 = xlsfile["hojaDatos1"]
> datos2 = xlsfile["hojaDatos2"]
> datos3 = xlsfile["hojaDatos3"]
```

Guardar datos en un archivo Excel con esta librería es también un proceso muy sencillo. Veámoslo con un ejemplo. Creamos un data.frame que contiene tres variables `x`, `y` y `z` cuyos valores se generan aleatoriamente entre 0 y 1:

```
> x = runif(100)
> y = runif(100)
> z = runif(100)
> misDatos = data.frame(x, y, z)
```

Para guardar el data.frame `misDatos` en el archivo `prueba2.xlsx` ejecutamos el siguiente comando (es obligatorio especificar nombre de hoja):

```
> writeWorksheetToFile("prueba2.xlsx", data = misDatos,
     sheet = "hojaDatos")
```

Si como extensión de archivo elegimos `.xls` en lugar de `.xlsx` entonces el archivo se guarda con el formato de Excel 2007.

En caso de que se deseen guardar distintos data.frames en varias hojas del mismo libro Excel resulta más conveniente el siguiente procedimiento:

```
> xlsfile = loadWorkbook("prueba2.xlsx", create = TRUE)
> xlsfile["hojaDatos1"] = misDatos
> xlsfile["hojaDatos2"] = data.frame(x,y)
> xlsfile["hojaDatos3"] = data.frame(z)
> saveWorkbook(xlsfile)
```

En primer lugar se ha creado el archivo prueba2.xlsx, añadiendo la opción `create=TRUE` a la función `loadWorkbook`; a continuación hemos guardado: el data.frame `misDatos` completo en la hoja `hojaDatos1`, un nuevo data.frame formado por las variables `x` e `y` en la hora `hojaDatos2`, y un data.frame que contiene sólo a la variable `z` en la hoja `hojaDatos3`; por último hemos cerrado y escrito a disco el archivo mediante el comando `saveWorkbook(xlsfile)`. **Es importante señalar que mientras no se ejecute este último comando el archivo no se guarda efectivamente en el disco.**

5.2. La librería RODBC

Esta librería hace uso de ODBC (*Open DataBase Connectivity*), un controlador estándar de acceso a bases de datos desarrollado por Microsoft, que permite que otras aplicaciones accedan a datos almacenados en Access o Excel[7]. Si bien el uso de esta librería permite realizar incluso consultas complejas a bases de datos, aquí nos limitaremos a señalar como se utiliza para leer y guardar datos en Excel.

Supongamos que en el directorio `c:\usuario\Desktop` tenemos un archivo llamado `pruebaExcel.xls` que contiene los datos de cinco variables en la primera hoja de cálculo[8]. Cada variable ocupa una columna y cada caso una fila. En la primera fila del archivo están los nombres de las variables. Para leer este archivo desde R utilizaremos la siguiente sintaxis:

```
> library(RODBC)
> setwd("c:/usuario/Desktop")
> conex.in <- odbcConnectExcel(xls.file = "pruebaExcel.xls")
> sqlTables(conex.in)
> misdatos <- sqlFetch(conex.in, "Hoja1$")
> odbcClose(conex.in)
```

Las dos primeras lineas simplemente especifican que cargamos la librería `RODBC` en memoria, y nos situamos en el directorio de trabajo `c:\usuario\Desktop` donde se encuentra el archivo que vamos a leer. La linea:

```
> conex.in <- odbcConnectExcel(xls.file = "pruebaExcel.xls")
```

indica que creamos una "*conexión*" con el archivo `pruebaExcel.xls`, y que dicha conexión se llama `conex.in` (podemos usar cualquier otro nombre). Crear una *conexión* significa simplemente

[7] En realidad ODBC tiene un uso más amplio ya que en general permite la conexión con sistemas de bases de datos que usen SQL (*Structured Query Language,* o *Lenguaje de Consulta Estructurado*), tales como MySQL, PostgreSQL, DB2, Oracle o SQLite.

[8] Cuando se guarda un archivo en Excel, por defecto se crea un "libro" con tres hojas de cálculo llamadas "Hoja1", "Hoja2" y "Hoja3". Obviamente el usuario puede cambiar estos nombres y eliminar o añadir hojas de cálculo a su libro. En este ejemplo asumimos que el archivo contiene tres hojas con sus nombres por defecto.

abrir un canal de comunicación entre R y el contenido del archivo Excel. La función `sqlTables(conex.in)` que encontramos en la linea siguiente, usa dicha conexión para mostrar la lista de hojas de cálculo que contiene el libro `pruebaExcel.xls`. La siguiente linea:

```
> misdatos <- sqlFetch(conex.in, "Hoja1$")
```

"lee" la hoja 1 (la función `sqlTables()` utilizada en la linea anterior nos ha mostrado que el nombre de dicha hoja es `Hoja1$`) a través de la conexión `conex.in` y guarda los datos en el *data.frame* `misdatos`. Por último, si no vamos a leer nada más en el archivo Excel cerramos la conexión mediante la función `odbcClose(conex.in)`.

De manera similar, si deseamos guardar el *data.frame* `results` en el archivo Excel `resultados.xls`, deberemos abrir una conexión, guardar los datos y cerrar la conexión. Para ello utilizaremos la sintaxis:

```
> conex.out <- odbcConnectExcel(xls.file = "OutExcel.xls",
     readOnly = FALSE)
> sqlSave(conex.out, results, rownames = F)
> odbcClose(conex.out)
```

Nótese que en este caso, al abrir la conexión mediante `odbcConnectExcel` hemos especificado la opción `readOnly=FALSE`. Esto indica a R que la conexión *no* es de sólo lectura, ya que vamos a escribir en el archivo.

5.3. La librería xlsReadWrite

Esta librería proporciona las funciones `read.xls()` y `write.xls()` que facilitan notablemente la lectura y escritura de archivos Excel. Parte de esta librería usa software que no es de código abierto, por lo que en realidad se compone de dos paquetes: uno que contiene el código abierto, y que R instala normalmente a través del menú *Paquetes* $\rightarrow$ *Instalar Paquetes,* y otro que se descarga[9] e instala automáticamente al ejecutar por primera vez la función:

```
> library(xlsReadWrite)
```

Si queremos leer los datos que están en la primera hoja del archivo `pruebaExcel.xls` y asignarlos al *data.frame* `misdatos` la sintaxis a utilizar es simplemente:

```
> misdatos <- read.xls(file = "pruebaExcel.xls",
     sheet = "Hoja1")
```

Asimismo, si queremos guardar el contenido del *data.frame* `misdatos` en el archivo Excel `resultado.xls` la sintaxis es:

```
> write.xls(misdatos, file = "resultado.xls")
```

[9] directamente desde la web del autor de la librería.

6. Lectura/Escritura de datos en formato .Rdata de R

Como muchos otros programas para el análisis estadístico, R dispone de su propio formato compacto para el almacenamiento de datos. Si, a lo largo de una sesión de trabajo, hemos generado los objetos `misdatos`, `regre`, `x` e `y` (pueden ser objetos de cualquier clase*: data.frames, listas, vectores,...*) podemos guardarlos en el archivo `SesionDeTrabajo.Rdata` mediante la sintaxis:

```
> save(misdatos, regre, x, y, file = "SesionDeTrabajo.Rdata")
```

Este formato de archivo sólo resulta legible desde R. Si deseamos leer este fichero para trabajar nuevamente con los datos que contiene, la sintaxis a emplear es:

```
> load(file = "SesionDeTrabajo.Rdata")
```

Las variables que contiene este archivo se cargan directamente en el actual "entorno" o "ambiente" (*environment*) de trabajo, con lo cual ya podemos acceder a ellas por su nombre. Nótese la diferencia con la lectura de datos desde archivos .csv,.xls. o.sav: cuando se emplean `read.table()`, `read.spss()` o `read.xls()`, los datos normalmente se asignan a un *data.frame* que debemos *activar* mediante `attach()`. Sin embargo, cuando se usa `load(nombre-de-archivo.Rdata)` los *data.frames* que se guardaron mediante `save(nombre-de-archivo.Rdata)` se cargan directamente en memoria.

Conviene aclarar algo más este concepto; cuando arrancamos R, automáticamente se asigna un espacio en la memoria del ordenador en el que se irán almacenando los objetos que creemos durante nuestra sesión de trabajo; este espacio de memoria recibe el nombre de "*parent enviroment*" o *entorno padre*. R permite asignar otros espacios de memoria, distintos del entorno padre, en los que se pueden crear nuevos objetos. Esto tiene interés en aquellas ocasiones (normalmente vinculadas a la programación) en las que es necesario trabajar con dos variables *distintas* que compartan el *mismo nombre*. Así, si creamos un entorno llamado `nuevoEspacio` podríamos tener datos de una variable `x` en el *entorno padre* y datos (distintos) de otra variable también llamada `x` en el entorno `nuevoEspacio` sin que ambas se confundan.

Imaginemos ahora que hemos iniciado una nueva sesión de trabajo y que durante el curso de la misma hemos creado dos nuevas variables `x` e `y`. Si ahora mediante la función `load(file="SesionDeTrabajo.` cargamos el archivo `SesionDeTrabajo.Rdata` creado anteriormente (y que, recordemos, contiene unas variables `x` e `y` distintas de las que acabamos de crear), nuestras nuevas `x` e `y` son borradas y sustituidas por las contenidas en `SesionDeTrabajo.Rdata`. Para evitar este efecto, es recomendable (salvo que estemos seguros de que no vamos a borrar nada) ejecutar el código siguiente:

```
> nuevoEspacio = new.env()
> load(file = "SesionDeTrabajo.Rdata", NuevoEspacio)
> misObjetos = as.list(NuevoEspacio)
> names(misObjetos)
```

En la primera linea, mediante la función `new.env()` hemos creado un nuevo *entorno de trabajo* llamado `NuevoEspacio` en memoria; la segunda linea lee los objetos que están en `SesionDeTrabajo.Rdata` y los coloca en el *entorno* `nuevoEspacio`; a continuación, la función `as.list()` convierte el contenido de este entorno en una *lista* que hemos llamado `MisObjetos`. Por último, la función `names()` nos muestra los nombres de los objetos contenidos en dicha lista. Para acceder a esos objetos, como siempre, habremos de preceder su nombre por el nombre de la lista y el símbolo `$` (`MisObjetos$x`, `MisObjetos$y`, etc).

Capítulo 7

Creación de funciones por parte del usuario: programación elemental en R.

En las secciones anteriores hemos podido ver como prácticamente todos los procedimientos en R se ejecutan mediante funciones. La gran versatilidad de R deriva del hecho de que cualquier usuario puede programar nuevas funciones, incrementando así paulatinamente las posibilidades del programa.

De una manera muy sucinta podemos definir una función como un fragmento de código que se encarga de realizar una o varias acciones. Estas acciones pueden ser de naturaleza muy variopinta: realizar cálculos, elaborar gráficos, mostrar mensajes, leer o guardar datos, interactuar con el sistema operativo,...

La declaración (definición o construcción) de una función de R consta de cuatro partes:

- **Nombre:** La única condición que debe cumplir el nombre de una función para estar bien definido es empezar por una letra y no contener espacios ni símbolos reservados (",+,-, etc.). En el nombre de una función sí que pueden utilizarse el punto (.) y el guión bajo (_). A continuación del nombre debe especificarse "`<- function`" (o de modo alternativo "`= function`"), para informar a R de que estamos definiendo una función.

- **Argumentos:** son los valores que recibe la función y con los que operará para obtener algún resultado. Una función puede carecer de argumentos, si la acción a realizar no depende de ningún valor que tenga que pasarle el usuario. Los argumentos deben especificarse entre paréntesis.

- **Código:** es el conjunto de instrucciones de R necesarias para que la función cumpla su cometido.

- **Resultado:** En el caso de que el objetivo de la función sea realizar algún tipo de cálculo, la función debe terminar devolviendo el resultado. El resultado se especifica entre paréntesis a continuación del comando `return`. No todas las funciones tienen por qué devolver un resultado (por ejemplo, las que se encargan sólo de imprimir un mensaje)

Ejemplo 1: Como ejemplo veamos como declarar una función que nos devuelva simplemente la suma de dos valores:

```
> suma <- function(a, b) {
      total = a + b
      return(total)
  }
```

Podemos identificar fácilmente en esta función las cuatro partes de las que hablábamos más arriba: el nombre de la función (`suma`) y su declaración como una función (`<-function`), seguido de sus argumentos, `(a,b)`. A continuación, entre llaves `{}`, está el *cuerpo* de la función que contiene el código a ejecutar (en este caso se crea una variable llamada `total` en la que se guarda el valor de `a+b`), y se devuelve el resultado mediante el comando `return(total)`.

Para ejecutar una función basta con escribir su nombre seguido de los valores de sus argumentos entre paréntesis. Por ejemplo, para sumar 3 y 5 con nuestra función bastaría con escribir:

```
> suma(3, 5)
[1] 8
```

Ejemplo 2: Vamos a construir ahora una función que nos permita leer fácilmente archivos excel utilizando la librería RODBC. Recordemos que leer archivos excel con esta librería resultaba algo complejo, puesto que es necesario "abrir" una conexión con el libro excel, leer los datos en la hoja adecuada y finalmente "cerrar" la conexión. Podemos incluir todos estos pasos en una función a la que sólo le pasemos como argumentos el nombre del libro excel y el nombre de la hoja donde están los datos:

```
> read.xls <- function(libro, hoja) {
      require(RODBC)
      conex.in <- odbcConnectExcel(xls.file = libro)
      misdatos <- sqlFetch(conex.in, paste(hoja,
          "$"))
      odbcClose(conex.in)
      return(misdatos)
  }
```

De esta forma, para leer la hoja 1 del archivo `prueba.xls` simplemente escribiríamos:

```
> nuevosDatos <- read.xls("prueba.xls", "Hoja1")
```

Por cierto, notemos el uso de la función `require()`. Esta función es muy similar a `library()`. Recordemos que `library(nombre-del-paquete)` carga directamente en memoria el paquete cuyo nombre hemos especificado. La función `require(nombre-del-paquete)`, en cambio, sólo carga el paquete en caso de que no haya sido cargado previamente.

Ejemplo 3: La siguiente función calcula la superficie de un rectángulo conocidas su base y altura:

```
> superfRect = function(base, altura) {
      S = base * altura
      return(S)
  }
```

Podemos probar a ejecutar varias veces esta función y comprobar como, cada vez que cambiemos los valores de la base y la altura, obtenemos la superficie correcta:

```
> superfRect(2, 4)
[1] 8
> superfRect(5, 3)
[1] 15
```

De manera equivalente podemos llamar a la función especificando también los *nombres* de sus argumentos (que, lógicamente, deben coincidir con los nombres especificados en la definición de la función). Cuando se especifican los nombres de los argumentos, éstos pueden colocarse en cualquier orden, no necesariamente el que se utilizó en la definición de la función):

```
> superfRect(base = 3, altura = 8)
[1] 24
> superfRect(altura = 5, base = 6)
[1] 30
```

Así pues, nuestro programa para el cálculo de la superficie de un rectángulo se reduce simplemente a la función que acabamos de definir. Podemos guardar este programa para volver a utilizarlo más adelante mediante **File**→**Save As**, indicando a continuación el nombre del archivo donde queremos guardar el programa. Resulta recomendable que los archivos que contienen código R tengan la extensión .R, de tal forma que podamos identificarlos fácilmente en los directorios de nuestro ordenador.

1. Estructura básica de una función en R .

A modo de resumen de lo que hemos visto en la sección anterior, podemos sintetizar la estructura de una función en R en el siguiente cuadro:

```
Nombre_función =
function(argumentos){
    Cuerpo de la función
    return(salida)
}
```

Observamos aquí la presencia de dos *palabras reservadas* (palabras propias del lenguaje R), `function` y `return`. Como vemos, para definir una función, primero ha de especificarse su nombre seguido del signo "=" (puede utilizarse también el símbolo "<–"), la palabra `function`, y a continuación entre paréntesis los argumentos que recibe la función; el cuerpo de la función (secuencia de instrucciones que debe realizar) se escribe seguidamente entre llaves {}, y normalmente termina con el comando[1] `return` que especifica entre paréntesis qué resultado debe mostrar la función como salida.

1.1. Un programa con entrada/salida más elaborada[2].

En ocasiones es necesario escribir programas que soliciten explícitamente al usuario la introducción de algunos valores. Supongamos, por ejemplo, que en nuestro programa para el cálculo de la superficie del rectángulo no queremos llamar a la función con unos valores concretos de base y altura, sino que nos interesa que sea la propia función la que los pida al usuario. Podemos conseguir este objetivo definiendo la función de la forma siguiente:

```
> superfRect = function() {
     cat("Longitud de la base: ")
     base = scan(n = 1, quiet = TRUE)
     cat("Longitud de la altura: ")
     altura = scan(n = 1, quiet = TRUE)
     S = base * altura
     cat("Superficie del rectángulo: ", S)
 }
```

Observemos que ahora la función no recibe argumentos, por lo que los paréntesis tras la palabra `function` deben quedar en blanco, (). A continuación hemos utilizado dos nuevos comandos:

- El comando `cat()` que, de manera muy simple, nos permite presentar texto en la consola.
- El comando `scan()`, que permite leer valores desde el teclado y asignarlos, respectivamente, a las variables base y altura; con este comando hemos utilizado además las opciones `n=1`, que

[1] Utilizaremos indistintamente los términos *comando* o *instrucción* para referirnos a aquellas palabras propias del lenguaje R que espefican la ejecución de una acción concreta.

[2] En este epígrafe y los siguientes supondremos que se utiliza RStudio como entorno para trabajar con R .

indica que se va a leer un único valor, y `quiet=TRUE` que indica a `scan` que, una vez leído ese valor, no informe por pantalla de cuántos valores ha leído (ese sería su comportamiento por defecto y no nos interesa en este caso).

Nuevamente, tras introducir esta función en la ventana de edición de RStudio, marcarla con el ratón, y pinchar en el icono 

, si no hay errores de sintaxis la función se reproduce en la consola. Para ejecutarla, la invocamos en la consola mediante:

```
> superfRect()
```

Si queremos calcular la superficie de un rectángulo de base 8 y altura 5, introduciríamos dichos valores a medida que el programa los solicite, obteniendo finalmente el resultado:

```
Longitud de la base:
1: 8
Longitud de la altura:
1: 5
Superficie del rectángulo: 40
```

En la sección 5 mostramos como se puede utilizar la librería `tcltk` para construir menús de entrada/salida más convenientes. En esta librería se apoyan múltiples GUIs (Interfaces Gráficas de Usuario) diseñadas para R , en particular, la interfaz de `Rcommander`.

2. Ejecución condicional de comandos.

El comando `if` permite la ejecución de un comando condicionado al valor de una variable. Así, por ejemplo, la siguiente función permite determinar el menor de dos números:

```
> menor = function(a, b) {
      if (a < b)
          menor = a
      else menor = b
      return(menor)
  }
```

En esta función la condición es muy sencilla, por lo que la instrucción condicional `if` ocupa una sola línea. En general, la sintaxis del comando `if` es la siguiente:

```
if (condición){
          expresión
      } else {
          expresión alternativa
      }
```

El comando `else` puede omitirse si no se realiza ninguna acción alternativa; asimismo pueden encadenarse varios comandos `else if`.

Ejemplo: la siguiente función nos dice si un vector es de dimensión 1, 2 o mayor.

```
> dimVector = function(x) {
     if (length(x) == 1)
         print("El vector es de dimensión 1")
     else if (length(x) == 2)
         print("El vector es de dimensión 2")
     else print("El vector es de dimensión mayor que 2")
 }
```

Obsérvese en este código que:

- length(x) nos devuelve la dimensión del vector.
- La comparación se realiza con el doble signo '=='
- else if se escribe separado.

Cuando hay un encadenamiento de alternativas `else if` resulta más conveniente utilizar la función `switch` (ver `?switch`).

3. Bucles.

En R hay varios comandos que permiten implementar bucles: `for`, `while` y `repeat`. En esencia, un bucle es una secuencia de comandos que se repite hasta que se cumple determinada condición.

3.1. For.

El bucle más sencillo es el bucle for. Así, por ejemplo, para calcular la suma de los cuadrados de los números enteros de 1 a n podríamos utilizar la siguiente función:

```
> sumaCuadrados = function(n) {
      suma = 0
      for (i in 1:n) {
          suma = suma + i^2
      }
      return(suma)
 }
```

En general, la sintaxis de un bucle `for` es la siguiente:

```
for (variable in secuencia){
          expresión
      }
```

3.2. While.

Otra forma de realizar la misma tarea es mediante la utilización de un bucle while: mientras se cumpla la condición que se especifica se va repitiendo el contenido del bucle:

```
> otraSumaCuadrados = function(n) {
      suma = 0
      i = 1
      while (i <= n) {
          suma = suma + i^2
          i = i + 1
      }
      return(suma)
 }
```

La sintaxis del bucle `while` es de la forma:

```
while (condición){
          expresión
      }
```

3.3. Repeat

Por último, R también dispone del comando `repeat` que repite una expresión indefinidamente hasta que encuentra el comando `break`. Así, nuestro ejemplo de la suma de cuadrados podría implementarse del siguiente modo:

```
> otraSumaCuadradosMas = function(n) {
      suma = 0
      i = 0
      repeat {
          i = i + 1
          suma = suma + i^2
          if (i == n)
              break
      }
      return(suma)
 }
```

La sintaxis de un bucle `repeat` es, pues, de la forma:

```
repeat {
            expresión
            if (condición) break
       }
```

Dependiendo del problema a resolver será más sencilla la implementación de uno u otro tipo de bucle.

En cualquiera de los bucles anteriores es posible introducir el comando `next`, que produce un salto inmediato a la siguiente iteración. Asimismo, en los bucles `for` y `while` es posible utilizar `break` para producir la salida inmediata del bucle.

4. Vectorización de los cálculos.

Al ser un programa interpretado, la ejecución de bucles en R suele ser muy lenta (aunque se han producido algunas innovaciones a partir de la versión 2.14). Por ello conviene evitar los bucles tanto como sea posible. Las funciones `apply()`, `sapply()`, `lapply()` y `tapply()` resultan extremadamente útiles para este fin. Así, por ejemplo, la función suma de cuadrados que hemos utilizado como ejemplo en la implementación de bucles puede programarse de manera mucho más eficiente del siguiente modo:

```
> n = 10
> cuadrado = function(x) x^2
> n2 = sapply(1:n, cuadrado)
> sum(n2)
[1] 385
```

Como vemos, hemos definido primero la función `cuadrado()` que calcula simplemente el cuadrado de un número; a continuación hemos calculado `n2` aplicando directamente, mediante `sapply`, esta función a todos los números de la secuencia de 1 hasta `n` en lugar de construir un bucle. Por último utilizamos la función `sum()` para obtener la suma de dichos valores.

Muchas funciones de R operan directamente en forma vectorial. Así, el cálculo anterior podíamos haberlo llevado a cabo directamente mediante:

```
> n = 10
> sum((1:n)^2)
[1] 385
```

El comando `sapply()` aplica una función a todos los elementos de un vector; `apply` funciona de manera similar aplicando una función a todas las filas (o columnas) de una matriz. Así, si definimos la matriz:

```
> A = matrix(1:9, ncol = 3, byrow = T)
> A
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
```

podemos obtener la suma de cuadrados de sus valores por filas mediante:

```
> sumCuad = function(x) {
     sum(x^2)
 }
> apply(A, 1, sumCuad)
[1]  14  77 194
```

y la suma de cuadrados de sus valores por columnas mediante:

```
> apply(A, 2, sumCuad)
```

```
[1]  66  93 126
```

El comando `lapply()` por su parte permite aplicar una función a todos los miembros de una lista, y `tapply(x,fac,fn)` aplica la función `fn` a los valores del vector `x` agrupados según los niveles del factor `fac`.

Señalemos por último que existe también una versión vectorial de `if`, la función `ifelse()` (ver `help(ifelse)`.sites

5. Construcción de Interfaces Gráficas de Usuario (GUI) con R : la librería `tcltk`.

Una de las facetas más importantes en el desarrollo de un programa es el diseño de la *interfaz de usuario.* La interfaz es la "fachada" que el programa presenta al usuario, por lo que el programador debe esforzarse en que sea clara y cómoda de utilizar. Cuando la interfaz presenta elementos que permiten la interacción (botones, menús desplegables, etc), se conoce como *interfaz gráfica* (conocida por sus siglas en inglés *GUI, Graphical User Interface*). Si bien en principio R carece de herramientas propias para el desarrollo de GUIs, la librería `tcltk` permite que desde R sea posible utilizar otro programa diseñado específicamente para ello, el programa Tk (*Tool Kit*). Este programa cuenta con varias ventajas que facilitan su integración en R : es multiplataforma (esto es, funciona en todos los sistemas operativos), es de código abierto, usa un lenguaje interpretado (el `Tcl`, *Tool Command Language*), y su sintaxis es muy sencilla.

Tanto `Tk` como `Tcl` fueron creados por John Ousterhout, profesor de Ciencias de la Computación en la Universidad de Stanford, para el desarrollo rápido de prototipos e interfaces gráficas. La librería `tcltk` desarrollada para R por Peter Dalgaard, profesor del Departamento de Bioestadística de la Universidad de Copenhague, actúa como puente entre R y `Tcl/Tk`: el usuario desarrolla el código utilizando la sintaxis y comandos típicos de R ; la librería `tcltk` se encarga de traducir estos comandos a `Tcl` que a su vez invoca a `Tk` para construir la interfaz gráfica.

No es el objetivo de este texto incluir un curso sobre la programación con `tcltk` (se requeriría probablemente un libro solo para ello), pero sí mostramos a continuación un ejemplo muy elemental, en el que puede comprobarse que no es difícil desarrollar widgets (aplicaciones gráficas de escritorio) utilizando R y `tcltk`, y citamos algunas fuentes de internet en las que el lector interesado puede documentarse para elaborar elegantes GUIs para sus programas.

Ejemplo: Utilizaremos la librería `tcltk` para construir un pequeño *widget* que calcula la superficie de un rectángulo (el mismo ejemplo que ya vimos en la sección 1.1). El *widget* consistirá simplemente en una pequeña ventana que nos pida la base y la altura de nuestro rectángulo, y que tenga dos botones: uno para realizar el cálculo y otro para cerrar la aplicación. La figura 1 muestra el aspecto final del *widget.*

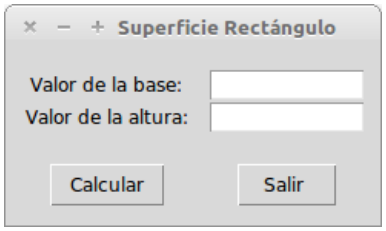


Figura 1: *Widget* para el cálculo de la superficie de un rectángulo.

El desarrollo del programa que construye el widget y lleva a cabo la operación de cálculo de la superficie del rectángulo es el siguiente:

1. En primer lugar comprobamos si la librería `tcltk` está cargada en memoria, y la cargamos en caso de no estarlo; la función require() se encarga de este cometido:

   require(tcltk)

2. Mediante el comando tktoplevel() creamos la ventana que va a contener al widget. Además asignamos dicha ventana a un objeto (que llamaremos tt en este ejemplo) en R , de tal forma que las futuras referencias a la ventana se hagan a través del nombre de este objeto:

   tt<-tktoplevel()

3. Le ponemos un título al widget, que aparecerá en la parte superior de la ventana que lo contiene:

   tkwm.title(tt, "Superficie Rectángulo")

4. El widget utilizará sus propias variables para “capturar” los valores de base y altura, variables que serán manejadas por el lenguaje `Tcl/Tk` (que es el que en realidad codifica la estructura del widget). Los siguientes comandos declaran estas variables de tal forma que R pueda acceder posteriormente a su contenido. Las declaramos como de tipo carácter, y las inicializamos con el valor nulo “”:

   base.tcl <- tclVar("")

   altura.tcl <- tclVar("")

5. Declaramos las “caja de texto” donde el usuario deberá introducir los valores de la base y la altura del rectángulo. Para cada caja se declara su anchura, y el nombre de la variable `Tcl` que almacenará el valor que se introduzca.

   entry.base <-tkentry(tt, width="12", textvariable=base.tcl)

   entry.altura <-tkentry(tt, width="12", textvariable=altura.tcl)

6. A continuación definimos las acciones que han de realizarse al pulsar cada uno de los botones del widget (“Calcular Superficie” y “Salir”). Cada acción se codifica en una función:

a) FUNCIÓN PARA LANZAR EL CÁLCULO DE LA SUPERFICIE DEL RECTÁNGULO. Esta función utiliza el comando tclvalue() para que R acceda a los valores contenidos en las variables base.tcl y altura.tcl del widget; como tales valores son de tipo texto, se convierten en numéricos mediante el comando as.numeric(). Seguidamente se calcula la superficie del rectángulo, se combina con un texto informativo mediante el comando paste() y se muestra su valor en una nueva ventana mediante el comando tkmessageBox():

```
OnCalc <- function() {
        base <- as.numeric(tclvalue(base.tcl))
        altura <- as.numeric(tclvalue(altura.tcl))
        S=base*altura
        msg <- paste("Superficie del rectángulo =",S)
        tkmessageBox(message=msg)
}
```

b) FUNCIÓN PARA SALIR DEL PROGRAMA Y CERRAR EL WIDGET: para ello se utiliza el comando tkdestroy():

```
OnExit <- function() {
        tkdestroy(tt)
}
```

7. Definimos ahora los botones del widget y sus acciones asociadas:

   a) Botón "*Calcular":*

   ```
   OK.but <-tkbutton(tt, text="Calcular", command=OnOK)
   ```

   b) Botón "*Salir*":

   ```
   Exit.but <-tkbutton(tt, text=" Salir ", command=OnExit)
   ```

8. Se alinean y colocan los componentes del widget: para ello utilizamos el comando tklabel(), que se encarga de escribir un texto dentro del widget, y el comando tkgrid(), que es el encargado de colocar los objetos (texto, cajas para introducir texto, botones, etc) en el widget. Cada uso de tkgrid() genera una nueva linea en el widget con los objetos que se especifiquen. Nótese que para dejar una linea en blanco tenemos que utilizar explícitamente la instrucción tkgrid(tklabel(tt,text=" ")):

   ```
   tkgrid(tklabel(tt,text=" "))
   tkgrid(tklabel(tt,text="Valor de la base:"), entry.base)
   tkgrid(tklabel(tt,text="Valor de la altura:"), entry.altura)
   tkgrid(tklabel(tt,text=" "))
   ```

tkgrid(OK.but, Exit.but)

9. Por último se activa la ventana que contiene el widget, y se la mantiene en espera hasta que el usuario realice alguna acción:

   tkfocus(tt)

   tkwait.window(tt)

Todos los comandos anteriores pueden integrarse en una única función que queda construída definitivamente del siguiente modo:

```
> supRect = function() {
      require(tcltk)
      tt <- tktoplevel()
      tkwm.title(tt, "Superficie Rectángulo")
      base.tcl <- tclVar("")
      altura.tcl <- tclVar("")
      entry.base <- tkentry(tt, width = "12", textvariable = base.tcl)
      entry.altura <- tkentry(tt, width = "12",
          textvariable = altura.tcl)
      OnOK <- function() {
          base <- as.numeric(tclvalue(base.tcl))
          altura <- as.numeric(tclvalue(altura.tcl))
          S = base * altura
          msg <- paste("Superficie del rectángulo =",
              S)
          tkmessageBox(message = msg)
      }
      OnExit <- function() {
          tkdestroy(tt)
      }
      OK.but <- tkbutton(tt, text = "Calcular",
          command = OnOK)
      Exit.but <- tkbutton(tt, text = " Salir  ",
          command = OnExit)
      tkgrid(tklabel(tt, text = "                                    "))
      tkgrid(tklabel(tt, text = "Valor de la base:"),
          entry.base)
      tkgrid(tklabel(tt, text = "Valor de la altura:"),
```

```
        entry.altura)
    tkgrid(tklabel(tt, text = "                                    "))
    tkgrid(OK.but, Exit.but)
    tkfocus(tt)
    tkwait.window(tt)
}
```

Bastará ahora ejecutar esta función (si usamos RStudio, marcamos el código con el ratón y pinchamos en Run), y escribir en la consola:

```
> supRect()
```

para que R lance el widget. La figura 2 muestra el resultado de ejecutar este programa:

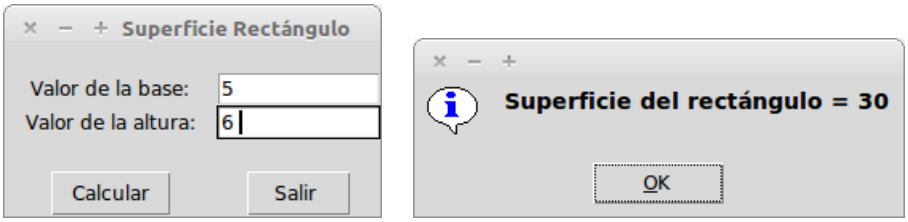


Figura 2: Cálculo de la superficie de un rectángulo mediante un programa que utiliza comandos de la librería `tcltk`.

Ejecución del widget desde la linea de comandos o mediante un acceso directo.

Supongamos que hemos guardado el programa anterior en el archivo supRectGUI.R. Resulta particularmente interesante la posibilidad de lanzar el widget sin necesidad de arrancar RStudio, ni el entorno estándar de R . En Windows ello puede hacerse de varias formas:

1. DIRECTAMENTE DESDE UNA TERMINAL. Para ello abrimos una terminal (Inicio→Todos los programas→Acceso del sistema o también Inicio→ejecutar→cmd), nos situamos en el directorio que contiene el programa y escribimos (incluyendo las comillas):

   "c:\Program Files\R\R-2.12.0\bin\i386\Rscript" supRectGUI.R

   (NOTA: la dirección que se ha puesto aquí es la correspondiente a la versión 2.12 de R . Si tenemos otra versión habrá que modificar esta dirección de manera adecuada).

2. CREANDO UN ACCESO DIRECTO. Para ello nos situamos en el escritorio (o en la carpeta en la que queremos crear el acceso directo) pulsamos el botón derecho del ratón y elegimos (Nuevo→Acceso directo). En la ventana que se abre pinchamos en “Examinar” y buscamos y seleccionamos el archivo supRectGUI.R. Pinchamos en “Siguiente”, damos un nombre al acceso directo y seleccionamos “Finalizar”. A continuación nos situamos sobre el icono con el acceso directo que

acabamos de crear y con el botón derecho del ratón seleccionamos “Propiedades”. En la casilla “Destino” aparece la dirección de nuestro archivo con el programa; justo delante, y separado por un espacio, escribimos (incluyendo las comillas):

"c:\Program Files\R\R-2.12.0\bin\i386\Rscript"

(si tenemos otra versión de R hay que modificar esta ruta adecuadamente). Por último, en la casilla “Ejecutar” seleccionamos la opción “Minimizada”, y pinchamos en “Aceptar”. Al pinchar con el ratón sobre el acceso directo que hemos creado de esta forma, nuestro widget se ejecutará de manera inmediata sin tener que arrancar R .

Información adicional sobre Tcl/Tk en Internet.

1. Paquete `tcltk` en R :
   - Artículo original de P. Dalgaard: The R-Tcl/Tk interface
   - Ejemplos (James Wettenhall): R TclTk Examples
   - Más ejemplos (Wettenhall/Grosjean): R TclTk coding examples
2. Paquete `widgetTools` en R : simplifica el acceso a determinadas funciones de `Tcl/Tk` desde R .
   - WidgetTools: widgetTools
3. Paquete `rpanel`: contiene diversas funciones para generar widgets `Tcl/Tk` que permiten interaccionar con los gráficos generados por R .
   - rpanel: The ‘rpanel’ package
4. Lenguaje `Tcl/Tk`: muchas de las capacidades de `Tcl/Tk` a las que se puede acceder desde R no están documentadas en la ayuda del paquete `tcltk`. Sin embargo sí que se encuentran en la documentación original de `Tcl` y `Tk`, y normalmente pueden adaptarse a su versión en R de forma muy sencilla.
   - Web oficial de Tcl/Tk: Tcl/Tk
   - En español: Tutorial de Tk en español