Introducción al entorno R

Paula Elosua

ARGITALPEN ZERBITZUA
SERVICIO EDITORIAL

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea

www.argitalpenak.ehu.es
ISBN: 978-84-9860-497-9

INTRODUCCIÓN AL ENTORNO R

INTRODUCCIÓN AL ENTORNO R

Paula Elosua

eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

ARGITALPEN ZERBITZUA
SERVICIO EDITORIAL

Este trabajo ha sido parcialmente financiado por
la Universidad del País Vasco/Euskal Herriko Unibertsitatea - **GIU09/22**

ISBN: 978-84-9860-497-9
L.G./D.L.: BI-163-2011

Bilbao, enero, 2011

www.argitalpenak.ehu.es

INTRODUCCIÓN AL ENTORNO R

R puede entenderse como un lenguaje de programación, como un potente software de análisis estadísticos o incluso como un generador de gráficos. Cualquiera de las tres acepciones es compatible con una definición de R. Estas páginas ofrecen al lector interesado y no iniciado en programación, los conceptos básicos y necesarios de la sintaxis de R que le permitirán trabajar en un entorno pensado para su uso con interfaces de códigos. Si bien es cierto que existen interfaces gráficas (`R Commander`) que facilitan el uso de R como software para el análisis de datos, no es menos cierto que la adquisición de destrezas en el manejo de unas normas de sintaxis básicas permitirán al usuario de R beneficiarse aún más de la potencia y de las ventajas que ofrece este entorno.

Paula Elosua es profesora del Departamento de Psicología Social y Metodología de las Ciencias del Comportamiento de la Universidad del País Vasco. Como usuaria de R, tanto en investigación como en docencia, es autora de los siguientes trabajos:

Elosua, P. (2009). "¿Existe vida más allá del SPSS? Descubre R" Psicothema, 21(4), 652-655.

Elosua, P. (2010). "R gizarte-zientzietarako. Datuen eta eskalen analisia Rcommander-ekin" Bilbao: UPV/EHU (accesible en el sitio: http://www.argitalpenak.ehu.es/p291-content/eu/contenidos/libro/se_ccsspdf/eu_ccsspdf/adjuntos/R%20gizarte.pdf)

Elosua, P. y Etxeberria, J. (2011). "Análisis de datos con R Commander" Madrid: La Muralla.

ÍNDICE

1. ¿QUE ES R?

R es un entorno de programación, análisis estadístico y software gráfico derivado del lenguaje de programación S (Becker, Chambers y Wilks, 1988; Chambers, 1998; Chambers y Hastie, 1992; Venables y Ripley, 2000). La primera versión de R se desarrolló en el Departamento de Estadística de la Universidad de Auckland (Nueva Zelanda) por **R**oss Ihaka y **R**obert Gentleman (Ihaka y Gentleman, 1996). Esta, junto a la fonética de R "`our`" –nuestro–, que enlaza R con el software libre, son las razones del nombre R. R se difundió rápidamente y la expansión es hoy irrefrenable. Desde su creación, R se alimenta y crece con los trabajos de investigadores provenientes de prácticamente todas las ramas del conocimiento. Las aportaciones continuas y desinteresadas de funciones y paquetes de propósito general o específico perfilan a R como un entorno dinámico formado por una comunidad activa y adherida a la filosofía del software libre.

R, en tanto en cuanto software libre, se asienta dentro del proyecto GNU *General Public Licence*. (Licencia Pública General, GNU). Se trata de una licencia creada por *Free Software Foundation* (Fundación para el software libre) organización fundada por Richard Matthew Stallman (rms) en el año 1985. El principal propósito de la licencia GNU es declarar la libertad del uso, modificación y distribución del software y protegerlo de intentos de privatización que puedan de algún modo restringir su uso (el contenido de la licencia puede consultarse en el sitio http://www.gnu.org/copyleft/gpl. html). Dentro de esta licencia se distribuyen un sinfín de programas, muchos de los cuales son versiones libres del software privativo generalista más utilizado. De entre ellos tal vez los más extendidos sean la suite ofimática *OpenOffice*, el navegador *Mozilla*, los artículos de *wikipedia,* el sistema operativo *GNU/Linux*, o el editor de textos *Emacs*.

R integra multitud de paquetes cuya continua incorporación al entorno R incrementan su capacidad y versatilidad. Si bien definiremos más adelante que son los paquetes, podríamos equipararlos con los módulos que ofrece el software privativo y comercial para el análisis de datos. R dispone de funciones básicas relacionadas con los análisis descriptivos de datos, y de los modelos más complejos y actuales concernientes con los últimos avances en el campo de la estadística, la psicometría, la econometría o el análisis de datos en áreas como la psicología, economía, sociología, estadística, biología, enfermería, farmacia, medicina o informática.

Aparte de las capacidades de análisis estadístico, R es un potentísimo generador de gráficos. Permite componer un gráfico simple, definir figuras extremadamente complejas e incluso crear animaciones (Maindonald y Braun, 2007; Murrel, 2005; Sarkar, 2008). Puede disfrutarse de una muestra de gráficos generados con R en el

sitio http://addictedtor.free.fr/graphiques/. Todos ellos evidencian la versatilidad y posibilidades de R.

1.1. Obtener e instalar R

El primer paso en el trabajo con R es obtenerlo e instalarlo. Para ello,

- Visitar la página oficial del R (http://www-r-project.org/), desde donde se descargará el archivo de instalación.

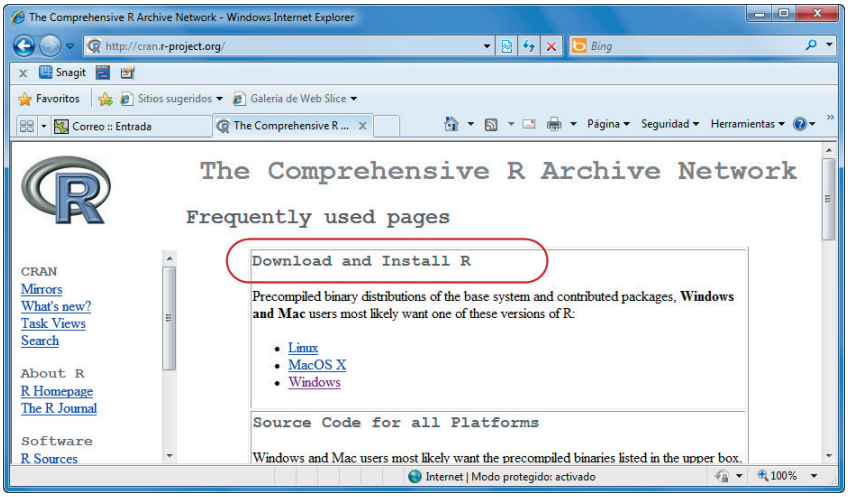

Figura 1.1. Página CRAN

- Tras pulsar sobre la plataforma adecuada (Linux, MacOS X, Windows), se accede a una pantalla en la que se seleccionarse la opción `base`.

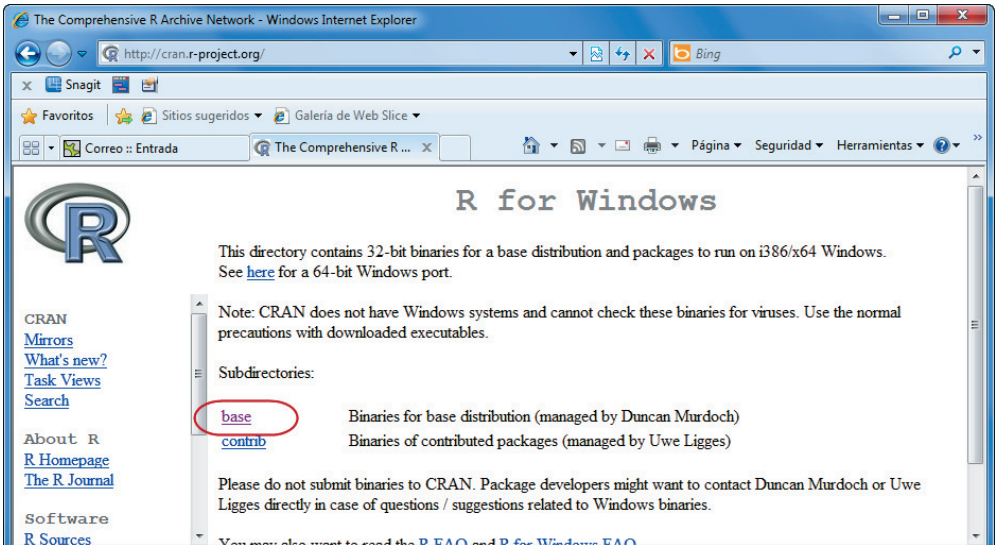

Figura 1.2. R para Windows, página de descarga (1)

- Descargar la última versión de R. (En el momento de redacción de este manual la versión fue R-2 12.0).

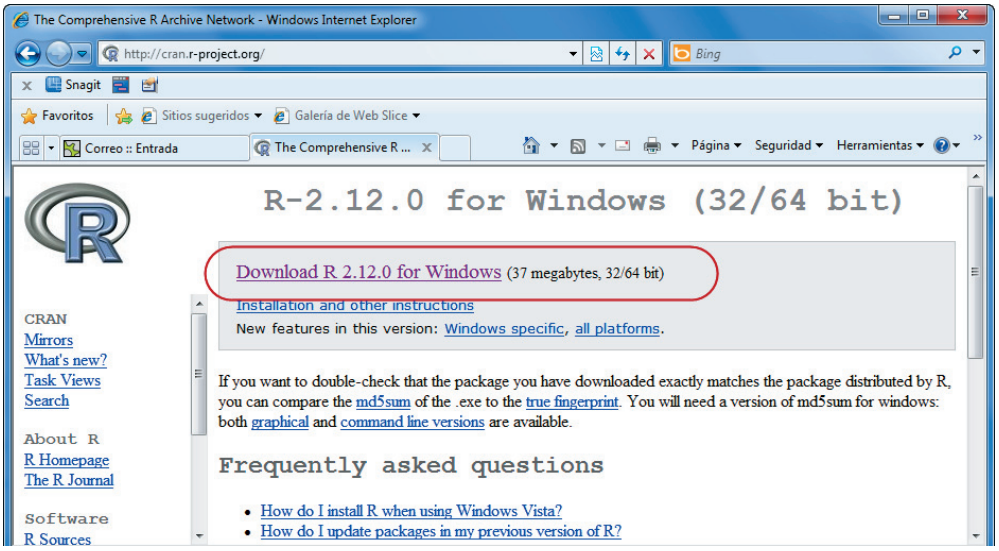

Figura. 1.3. R para Windows, página de descarga (2)

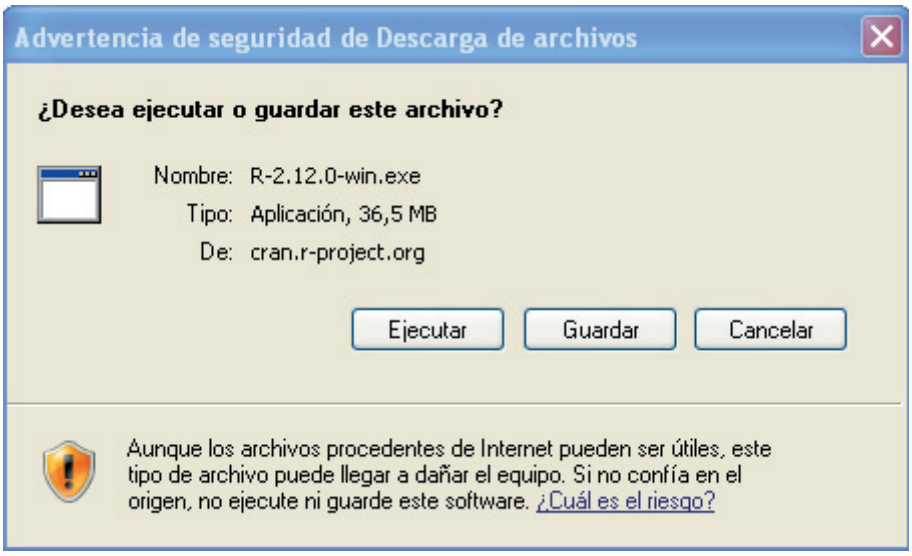

Figura. 1.4. R para Windows, página de descarga (3)

- Instalar R. El proceso de instalación es automático, y similar al de cualquier otro software.
- Una vez instalado R, su icono aparecerá en el escritorio.

1.2. Como se trabaja con R?

Cuando se ejecuta R, la pantalla que emerge recibe el nombre de `consola de R` (*R console*); en ella puede comprobarse la versión instalada (`R version` 2.12.0). La

consola muestra en color rojo el símbolo del sistema o `prompt` (>), que indica que R está listo para recibir comandos.

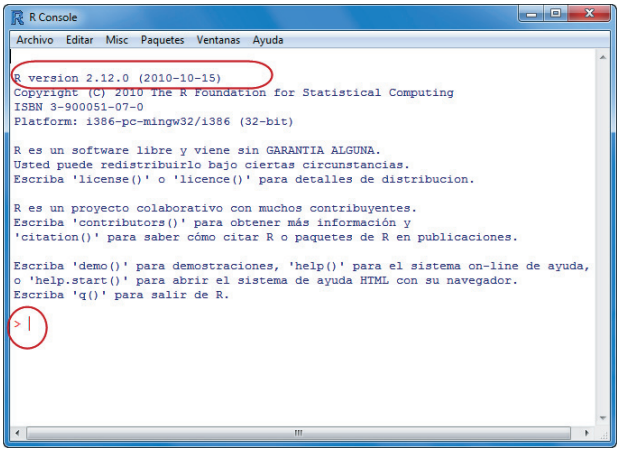

Figura 1.5. Consola R

La primera impresión que causa R al descubrir las escasas opciones de su interfaz estándar es de sorpresa,. Las funciones a las que se accede a través de la barra de menús son rudimentarias, y no dispone de menús desplegables para el análisis de datos, o para la generación de gráficos. El entorno R está configurado en principio, sobre una interfaz de comandos donde el usuario ha de escribir las instrucciones y comandos que desea ejecutar. Esta forma de trabajo se contrapone con el modo de operar construido sobre interfaces gráficas compuestas por sistemas de ventanas y menús desplegables. No hay duda de que en muchos contextos, las interfaces gráficas son útiles y convenientes. Sin embargo, las posibilidades que ofrecen las interfaces de comandos son mayores y no están limitadas más que por la habilidad del programador. El uso de comandos exige una curva de aprendizaje mayor que el requerido por las interfaces gráficas, pero las ganancias en términos de independencia, creatividad y control no son comparables. Escribir un código supone una comprensión más profunda de aquello que se desea aplicar.

1.2.1. *Interfaces para programación*

Son varias las posibilidades que ofrece R para la redacción de códigos:

- Trabajar directamente sobre la consola de R (`R console`), tecleando los comandos en la línea de "símbolo del sistema" (>). Es un modo de trabajo interactivo en el que sólo puede ejecutarse una línea de comandos al mismo tiempo; por lo tanto, solamente resulta válida para ejecutar acciones simples. Las flechas de dirección facilitan el trabajo en la consola. La flecha hacia arriba (↑) permite acceder a comandos previos, y la flecha hacia abajo (↓) da acceso a comandos posteriores al actual. Las flechas hacia la izquierda (←) y hacia la derecha (→) mueven el cursor en ambos sentidos dentro de una línea. Los comandos pueden copiarse y pegarse, como en cualquier editor de textos, con las funciones `Ctrl+C` y `Ctrl+V`.

- Abrir y trabajar en una ventana de escritura (`Script window`) a la que se accede seleccionando la opción `Nuevo Script` de la barra de menús, `Archivo > Nuevo Script`. Esta opción permite crear códigos más complejos. Los códigos se teclearán directamente en esta ventana o podrán ser copiados de un archivo ya existente en formato ASCII. Los códigos puede ejecutarse por líneas o por bloques, para ello se debe de marcar la sección a ejecutar y pulsar CTRL+R o en su defecto el icono de ejecución  . Cuando se utiliza la ventana de escritura como entrada para la ejecución de comandos, las salidas se muestran en la consola; por ello, es recomendable mantener simultáneamente abiertas ambas ventanas: consola y ventana de códigos.

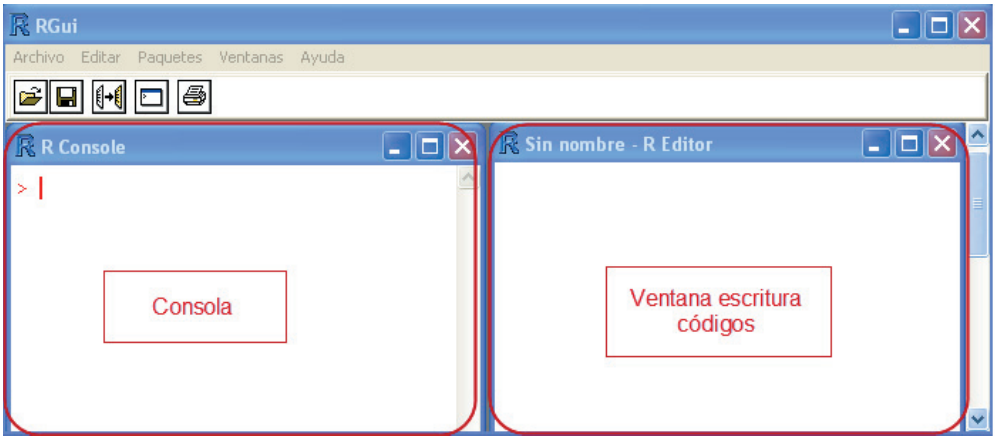

Figura 1.6. R Consola y ventana de escritura de códigos

- Interfaces para la construcción de códigos. Existen programas específicos para la creación/edición de códigos. Es la opción preferida de los usuarios avanzados de R. Entre los más utilizados:
  - `Tinn-R`, (http://www.sciviews.org/Tinn-R/),
  - `WinEdit` (http://www.winedt.com/) o
  - `Emacs` (http://www.gnu.org/software/emacs/), software creado inicialmente para el entorno Unix bajo la filosofía GNU.

La construcción de códigos permite al usuario diseñar y programar sus propias funciones. La programación significa controlar el ordenador por medio de instrucciones que indican que operaciones llevar a cabo sobre los datos, que imprimir, que guardar, que resolver, o como diseñar un gráfico.

1.2.2. *Interfaces gráficas*

Aquellos no familiarizados con una interfaz de códigos y que se sientan más cómodos con interfaces basadas en menús contextuales disponen de varias opciones. Existen varias GUIs (`Graphical User Interfaz`) que facilitan el trabajo con R a personas no conocedoras de este entorno de programación y análisis. Entre las GUI disponibles:

- `R.NET` (http://www.u.arizona.edu/~ryckman/RNet.php)
- `Poor Man's GUI` (http//www.math.csi.cuny.edu/pmg)
- `Rkward` (http://es/wikipedia.org/wiki/RKWard) todavía no disponible para Windows
- `R Commander` (Fox, 2005)
- `RExcel` (http://www.statconn.com) que se distribuye bajo licencia privativa.

De entre ellas la más extendida es `R Commander`. Entre sus cualidades podrían destacarse su adecuación hacia los contenidos metodológicos utilizados en las ciencias sociales, y la simplicidad de uso. Es el intermediario perfecto para acercar al usuario habitual de paquetes comerciales al entorno de programación R, porque permite una transición sencilla hacia este entorno de trabajo (Arriaza y col., 2008; Elosua, 2010; Elosua y Etxeberria, 2011).

2. **PRIMEROS PASOS**

Una vez ejecutado R, la presencia del símbolo del sistema o `prompt,` en color rojo, (>) indica que R está listo para recibir comandos. Por ejemplo, tecleando "2 + 3" (pruebe el lector cualquier otra operación) y pulsando <ENTER>, R devolverá el resultado de la operación aritmética.

```
> 2+3
[1] 5
>
```

El [1] indica el orden de aparición de los resultados. En este caso sólo se ha solicitado un resultado. Si la salida hubiera sido más compleja R reconocería cada uno de los elementos por un número correlativo entre corchetes que señala du posición. Una vez ejecutado el comando, el símbolo de sistema ">" indica que R vuelve a estar listo para recibir otra instrucción.

Además del símbolo del sistema (>), la consola R puede presentar un símbolo "+" (también en color rojo) para indicar que la instrucción dada a R es incompleta, y que no puede ejecutarse sin antes finalizar correctamente la secuencia de comandos. Por ejemplo si se teclea "`3+5+(`" , R devuelve el símbolo " +" con el que advierte que la secuencia está incompleta. Si ante el símbolo "`+`" se teclea algo que pueda concluir la línea (por ejemplo "7*3)") R proporcionará el resultado de la operación, 29. Ante la presencia del símbolo "+", pulsar la tecla "`Esc`" (escape) permitirá volver al símbolo del sistema.

Es posible escribir más de una función en la misma línea de comandos, para ello basta utilizar un carácter delimitador entre ellas" ; ".

```
> 3+5+(
+
> 3+5+(
+ 7*3)
[1] 29
> 2+3;4*7
[1] 5
[1] 28
```

Otro símbolo importante en el entorno de programación R es el símbolo "#", utilizado para añadir comentarios no ejecutables durante la construcción de un código. Ayuda a mantener organizado un código, y facilita su lectura posterior a propios y ajenos.

Para abandonar R tras una sesión de trabajo son varias las opciones disponibles:

- Teclear en la consola `q()`.
- Acceder a la opción `Salir` a través de la barra de menús `Archivo > Salir`.
- Pulsar sobre el icono de salida  .

2.1. **R como calculadora**

El modo de trabajar más simple o primitivo con R sería utilizarlo como una potente calculadora. En este sentido R evalúa y devuelve los resultados de cualquier expresión introducida en la línea de comandos. La tabla siguiente ofrece una breve descripción de las funciones algebraicas más comunes, y cuáles son los comandos que las ejecutan:

Como ejemplo, que puede servir al lector de primer ejercicio con R se muestra la ejecución de algunas de ellas.

```
> 2+3
[1] 5
> sqrt(20)
[1] 4.472136
> 5%/%3 # devuelve la parte entera de la división 5/3
[1] 1
> 5%%3 # devuelve el resto de la división 5/3
[1] 2
```

Función	Operación
`+, -, *, /`	Suma, Resta, Multiplicación, División
`abs`	Valor absoluto
`asin acos atan`	Inversas de las funciones trigonométricas
`exp, log`	Exponencial y logaritmo natural
`round`	Redondeo
`sin con tan`	Funciones trigonométricas
`sqrt (), ^`	Raíz cuadrada, Potencia
`%/%`	División entera
`%%`	Resto de la división

Tabla 2.1. Funciones algebraicas

```
> pi*5^2 # área de un círculo de radio 5
[1] 78.53982
> 1000*(1+0.05)^3-100
[1] 1057.625
> sqrt(c(10,100,1000))# raíz cuadrada de cada uno de los
valores incluidos en la secuencia
[1] 3.162278 10.000000 31.622777
```

2.2. **Donde obtener ayuda**

El entorno R es extraordinariamente amplio y heterogéneo. Para facilitar el tránsito por él, dispone de varias fuentes de ayuda sobre procedimientos, comandos, paquetes o sobre la aplicación de modelos formales. El modo básico de obtener ayuda en R es a través de la opción `Ayuda` de la barra de menús, desde ella se pueden seleccionar varias alternativas

2.2.1. *Información sobre funciones instaladas*

- La opción `Ayuda > Funciones R(texto)` abre una ventana interactiva en la que se tecleará la función sobre el que se busca información.

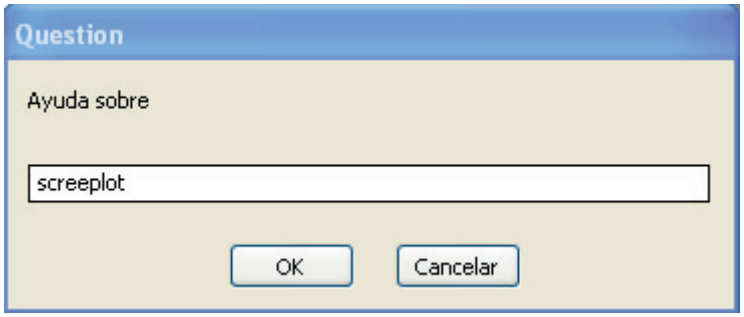

Figura 2.1. Menú de ayuda

Este modo de obtener información sobre una función concreta, es equivalente a teclear `help(screeplot)` sobre la consola R, o incluso `?screeplot`.

```
> help(screeplot)
> ?screeplot
```

- La opción `Ayuda > Ayuda Html`, acciona el explorador definido por defecto, y muestra un pantalla a través de la cual se accede a la información almacenada en la memoria.

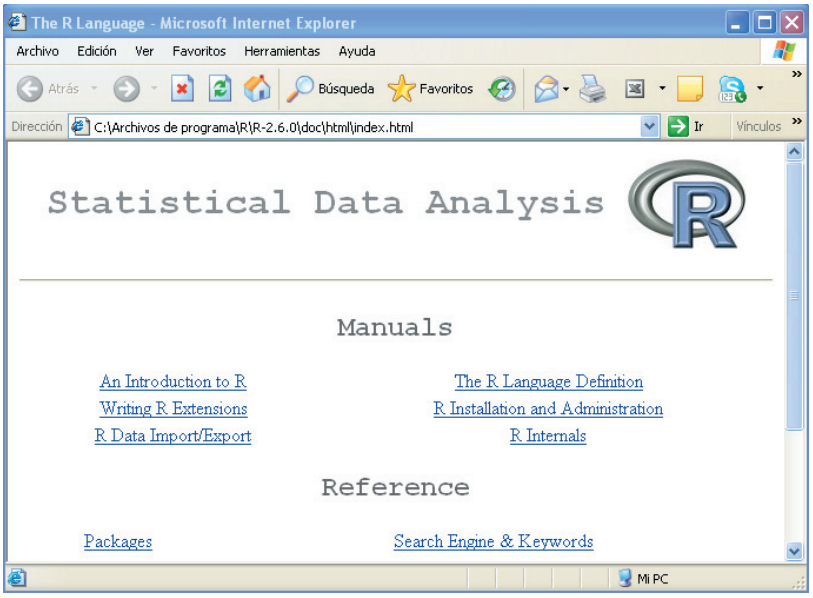

Figura 2.2. Ayuda Html

Podemos acceder a la misma página a través del comando `help.start()` que se tecleará directamente sobre la consola R.

2.2.2. *Información sobre procedimientos o modelos*

- Existen otras dos funciones de búsqueda `help.search()` y `apropos()`, que pueden ser útiles cuando se desea encontrar información sobre un procedimiento o modelo determinado que no se sabe exactamente dentro de qué función localizar. Por ejemplo: supongamos que se quiere obtener información sobre el análisis factorial "*factor analysis*"; A través de la barra de menús se seleccionará la opción `Ayuda > Buscar Ayuda`...que permitirá teclear el texto que describe lo que se desea buscar. Se puede obtener la misma información tecleando directamente sobre la consola lo siguiente:

```
help.search("factor analysis")
```

Este comando ofrece como salida una lista de todas las funciones cuyas páginas de ayuda contienen la palabra situada entre comillas. En este caso "*factor analysis*".

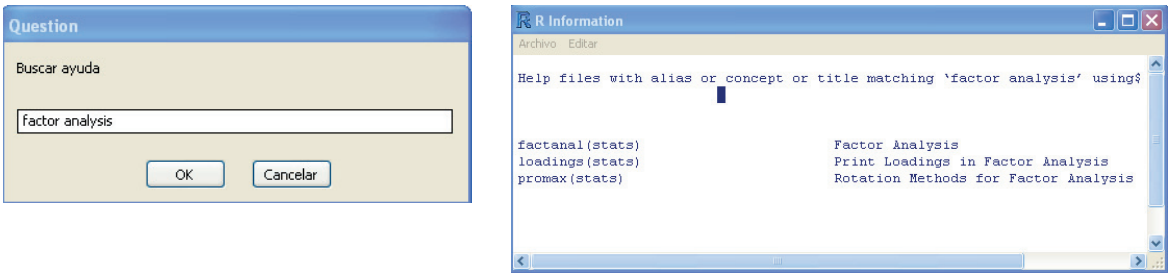

Figura 2.3. Help.search()

La salida indica que existe una función llamada `factanal` en el paquete `stats`, en la que también se encuentran las funciones `loadings` y `promax`. Si se desea acceder al contenido de estas funciones, a su descripción, bastaría con teclear en la consola el nombre de la función, y la librería que la contiene:

```
help(factanal,package = stats)
```

Este comando abriría una página de ayuda sobre la función `factanal` en la que se incluye una descripción de la misma, su uso, el modo de especificar sus argumentos y un conjunto de ejemplos.

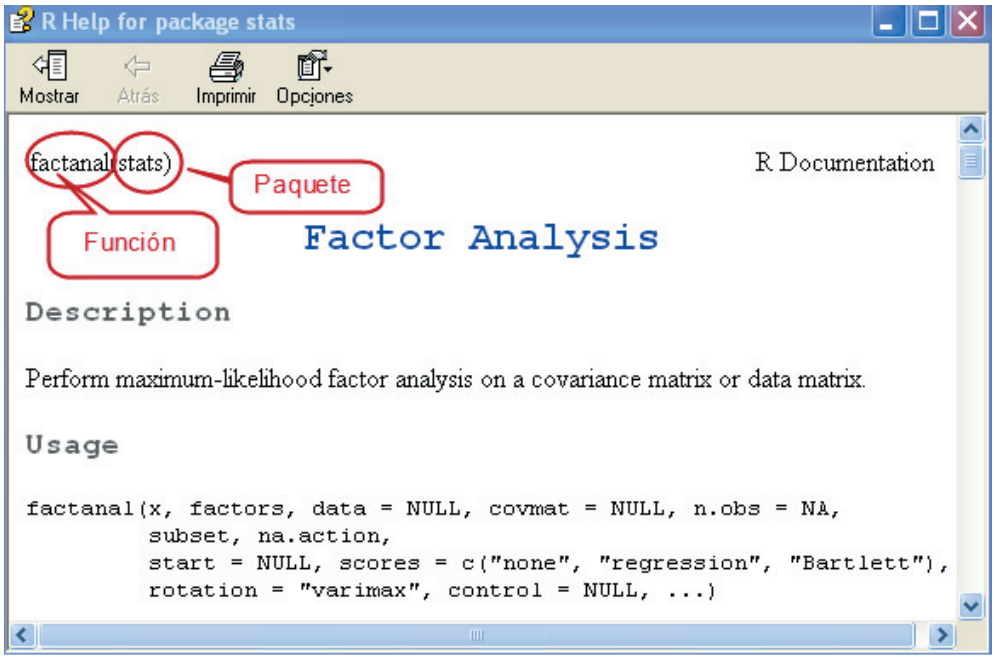

Figura 2.4. Página de información sobre comando

Si se teclea directamente sobre la consola:

```
> apropos("vector")
```

Se accede a una lista de todas las funciones que incluyen el texto entrecomillado; en este caso "`vector`".

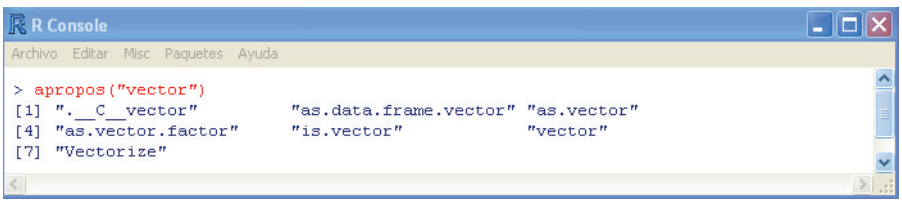

Figura 2.5. `apropos()`

2.2.3. *Ayuda en la red*

- Los comandos anteriores ofrecen información sobre funciones que están incluidas en los paquetes que tengamos instalados. Sin embargo, en muchas ocasiones el interés se centra en saber si R dispone de funciones para ejecutar determinado tipo de análisis, y esta información va más allá de los paquetes instalados. Es posible ampliar la base de la exploración y optar por una búsqueda a en la red. Para ello es suficiente con seleccionar la opción de búsqueda `Ayuda > search.r.project.org`. Otra opción es teclear la función `RSiteSearch()` directamente sobre la consola R.

2.2.4. *Fuentes de información*

- Existen además de las descritas, varias fuentes de información sobre R accesibles en la red; R dispone de un lista de distribución (https://stat.ethz.ch/mailman/listinfo/r-help) y de un buscador específico sobre funciones, comandos o cuestiones relacionadas con R http://www.rseek.org/.

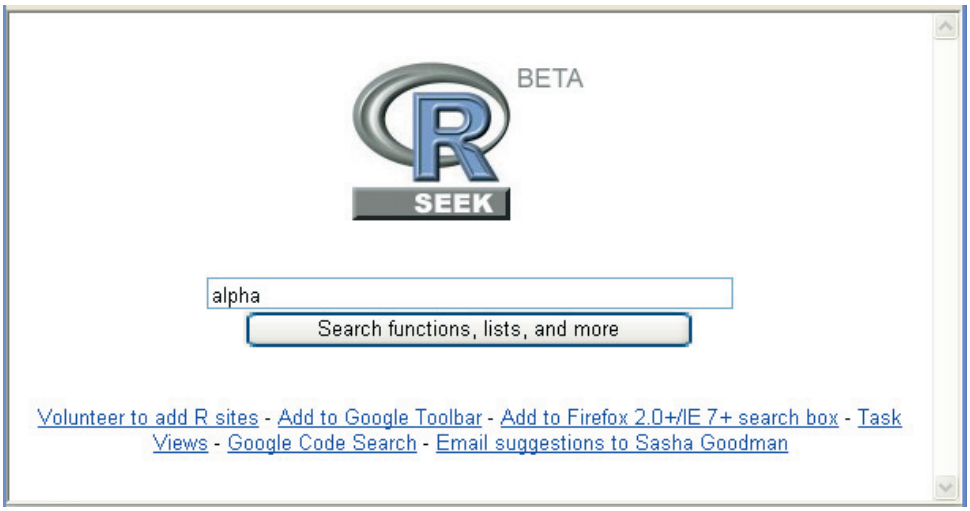

Figura 2.6. Rseek

- La página principal de CRAN (`Comprehensive R Archive Network`; http://cran.r-project.org/) almacena una gran cantidad de información sobre R; incluye manuales de uso en varios idiomas, información sobre paquetes o listas sobre FAQ (*Frequently Asked Questions*) que pueden solucionar más de una duda. Entre ellas resulta especialmente atractiva la *wiki* (http://wiki.r-project.org/rwiki/doku.php).

2.3. **Paquetes**

R es un sistema integrado al que se añaden complementos que reciben el nombre de paquetes (*packages*). Un paquete es un conjunto de funciones que mantienen algún

tipo de relación entre ellas, y que usualmente vienen acompañadas de archivos de ayuda y de ficheros de datos. El número de paquetes que pueden integrarse en R es superior a 2500 (En Noviembre de 2010 el número de paquetes disponibles era 2625). La diversidad de paquetes va pareja a la pluralidad de áreas de conocimiento, aplicaciones y modelos estadísticos. Existen paquetes para analizar microarrays, paquetes para el modelaje de riesgos de crédito, paquetes para las ciencias sociales, paquetes para diseñar y resolver sudokus…. Algunos de ellos están incluidos en la instalación básica de R (`base`), y al resto se accede a través de repositorios públicos. Instalar paquetes en R es una tarea muy sencilla; puede instalarse un paquete utilizando las opciones de la barra de menús, o utilizando la consola:

- Seleccionar de la barra de menús la opción `Paquetes > Instalar Paquetes...`
- Teclear directamente sobre la consola R `install.packages("nombre paquete")`.

Ambas opciones dan paso a una ventana que ofrece un listado de sitios imagen o "`mirror`". Una vez seleccionado uno de ellos, lo habitual es elegir alguno geográficamente cercano, se mostrarán en pantalla todos los paquetes disponibles; los paquetes aparecen ordenados alfabéticamente. Pulsar sobre el paquete que se desea instalar, y presionar el botón OK. R descargará e instalará el paquete.

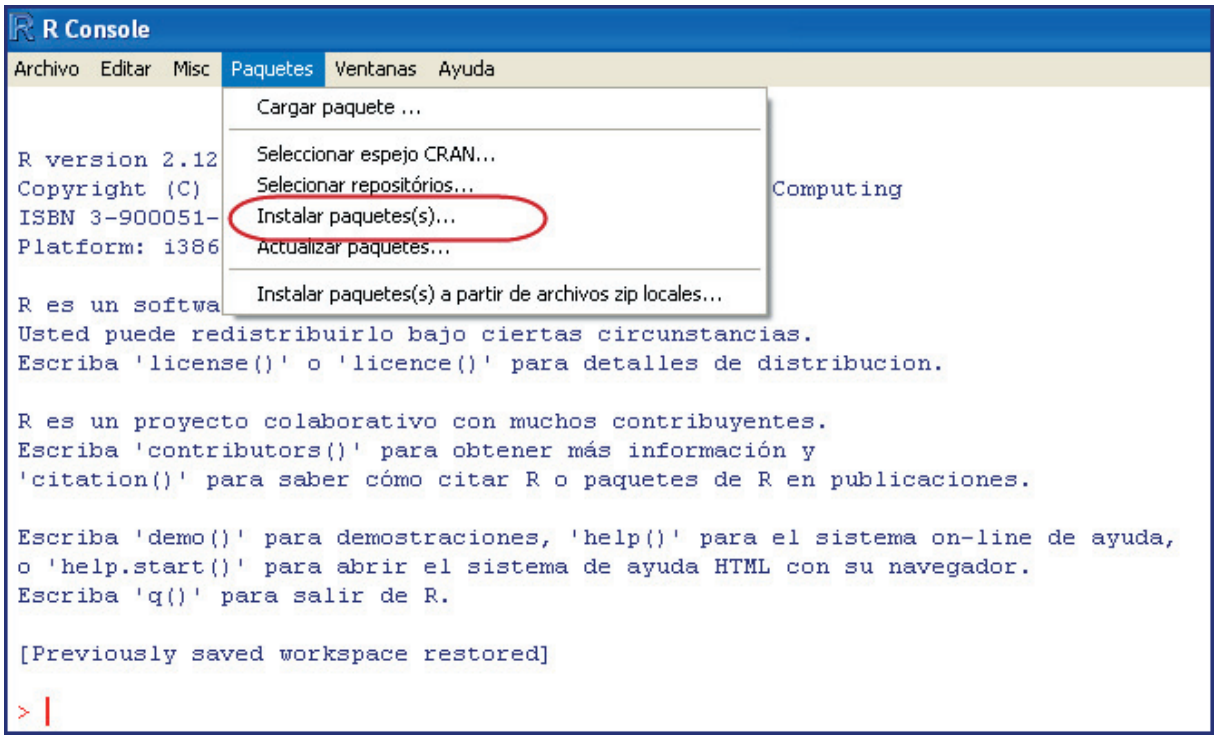

Figura 2.7. Instalar paquete

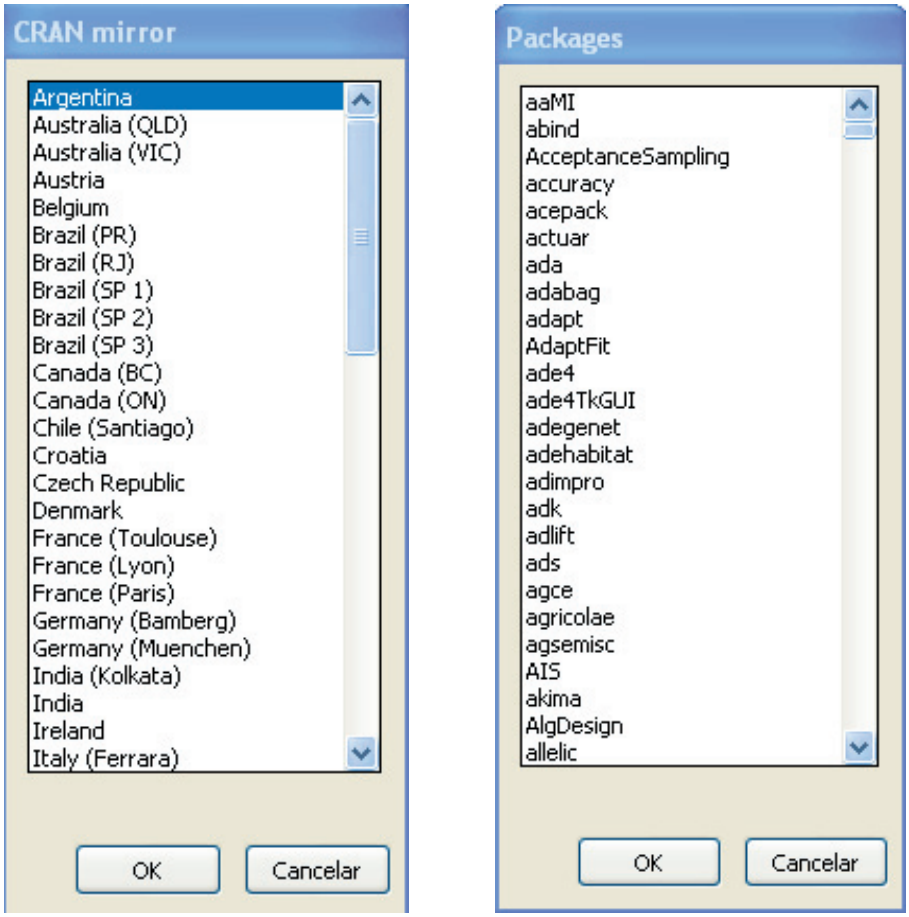

Figura 2.8. Repositorios y listado de paquetes

2.3.1. *Cargar paquete*

Para utilizar las funciones incluidas en un paquete es necesario cumplir dos condiciones: instalarlo y cargarlo. Un paquete se instala una sola vez, pero es necesario cargarlo en cada sesión de trabajo en el que se desee utilizar. Es una tarea sencilla, que puede realizarse a través del menú o a través de la consola:

- Seleccionar de la barra de menús a la opción `Paquetes > Cargar Paquete`
- Teclear `library(nombre del paquete)`
- Teclear `require(nombre del paquete)`.

Si se intenta utilizar una función incluida en un paquete que no ha sido cargado, R devuelve un mensaje de error. R ofrece varios comandos específicos para el manejo de paquetes:

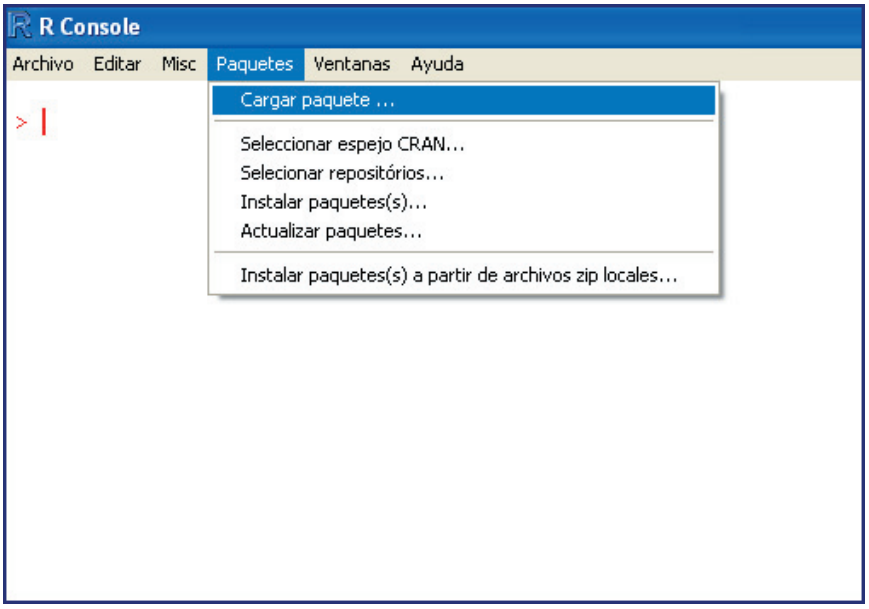

Figura 2.9. Cargar paquete

- Listar todos los paquetes instalados: `installed.packages()# excepto los incluidos en la instalación por defecto`
- Listar los paquetes instalados por defecto: `get()option`
- Listar los paquete cargados: `(.packages())`
- Listar de todos los paquetes instalados: `( packages(all.available = TRUE)), o library()`.

```
> Option("defaultPackages")
 [1] "datasets"  "utils"     "grDevices"  "graphics"  "stats"     "methods"
> (.packages())
 [1] "stats"     "graphics"  "grDevices"  "utils"     "datasets"  "methods"
 [7] "base"
> (.packages(all.available = TRUE))
 [1] "base"          "boot"       "class"      "cluster"     "codetools"
 [6] "datasets"      "foreign"    "graphics"   "grDevices"   "grid"
[11] "KernSmooth"    "lattice"    "MASS"       "Matrix"      "methods"
[16] "mgcv"          "nlme"       "nnet"       "rpart"       "spatial"
[21] "splines"       "stats"      "stats4"     "survival"    "tcltk"
[26] "tools"         "utils"
```

2.3.2. *Organización de paquetes en CRAN*

El rápido incremento en el número de paquetes disponibles para R (más de 2500) ha hecho necesario crear un sistema de clasificación que facilite orientarse entre ellos. Los paquetes han sido catalogados en varios grupos a los que se puede acceder a través de la página CRAN (http://www.cran.r-project.org/web/views/).

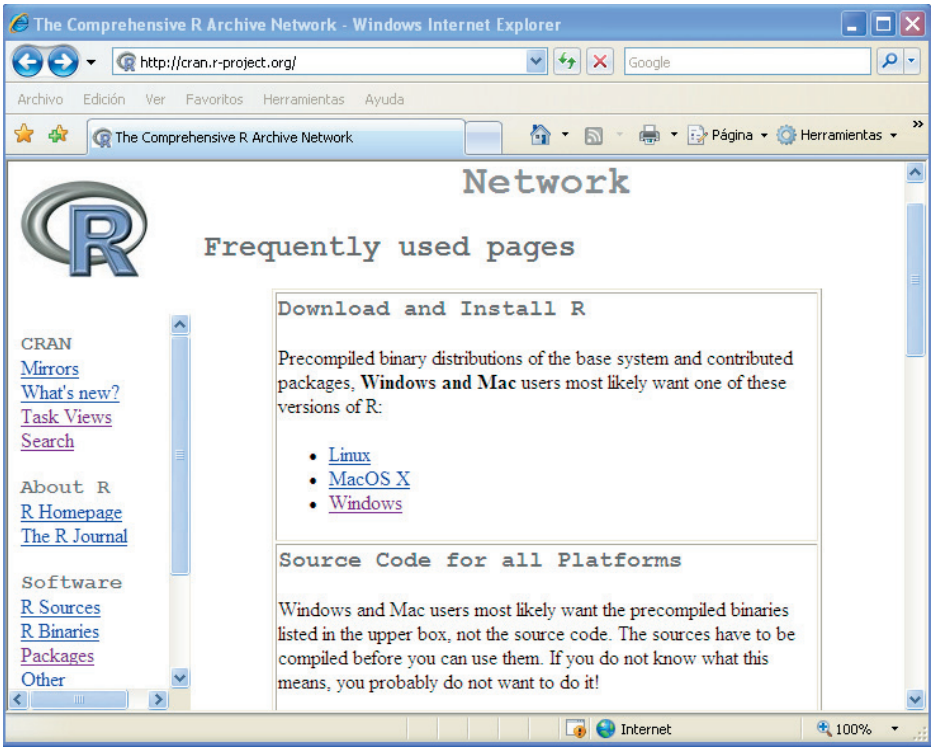

Figura 2.10. Task Views de CRAN

La estructura de la información ha sido reducida a las siguientes categorías:

Clasificación de paquetes en CRAN	
`Bayesian`	Estadística bayesiana
`ChemPhys`	Quimiometría y Física Computacional
`ClinicalTrials`	Diseño,monitorización y análisis de ensayos clínicos
`Cluster`	Análisis de conglomerados y modelos mixtos
`Distributions`	Distribuciones
`Econometrics`	Econometría
`Environmetrics`	Modelos para la Ecología
`ExperimentalDesign`	Diseño de experimentos
`Finance`	Análisis financiero
`Genetics`	Modelos para la genética
`Graphics`	Gráficos
`gR`	Modelos de gRafos
`High Performance Computing`	Computación intensiva
`MachineLearning`	Aprendizaje maquina
`MedicalImaging`	Análisis de imágenes médicas
`Multivariate`	Análisis multivariado
`NaturalLanguageProcessing`	Procesamiento lenguaje natural
`Optimization`	Estadística operativa
`Pharmacokinetics`	Datos farmaquinéticos
`Psychometrics`	Modelos psicométricos
`Robust`	Métodos robustos
`SocialSciences`	Estadística aplicada a las CC.SS.
`Spatial`	Análisis de datos espaciales
`Survival`	Análisis de supervivencia
`TimeSeries`	Análisis de series temporales

Tabla 2.2. Clasificación de paquetes

Cada categoría ofrece una breve descripción de los paquetes diseñados para cubrir determinados menesteres. Por ejemplo en el grupo de paquetes incluidos en el apartado "Ciencias Sociales" se detallan aquellos que incluyen funciones dentro del modelo lineal general, modelo lineal generalizado, análisis de datos categóricos, otros modelos de regresión (regresiones no-lineales).

3. **OBJETOS**

En R prácticamente todo es definido como un objeto; un dato numérico, un vector, una matriz de datos o una función, son objetos, y R opera sobre ellos. Cada objeto tiene un nombre, y el hecho de escribirlo en la ventana de comandos hará que se muestre su contenido. R distingue entre distintos tipos de objetos (no es lo mismo un vector, una matriz o una función) cada uno de los cuales posee características propias, que en el entorno R se conocen como modo y atributos. Los objetos son de un determinado tipo (`mode`) y tienen atributos (`attributes`). Un objeto es en definitiva la forma en la que R almacena la información. Para obtener un listado de los objetos disponibles en el espacio de trabajo pueden utilizarse las funciones `ls()` u `objects()`.

El caso de objeto más simple sería un número o carácter numérico que puede almacenarse en un objeto de nombre `x`. Para almacenar un objeto es necesario utilizar una regla de asignación, es decir, es necesario informar a R de que existe un objeto. La función de asignación " <- " tiene ese cometido (no es necesario dejar espacios en blanco antes y después de la función de asignación; sin embargo, es un uso que está aconsejado y que ayudará a generar códigos claros y limpios).

```
> x <- 5 # el número 5 es asignado al objeto x
> x
[1] 5
```

Se ha creado el objeto "`x`" al que se ha asignado el valor "5". Tecleando el nombre del objeto ("`x`"), R devolverá su valor "5". Si tras está asignación se vuelve a reasignar un nuevo contenido a `x`, éste se almacenará sobre el viejo objeto de modo que la primera asignación desaparecerá.

Para nombrar objetos es posible utilizar letras, números, guiones o el punto, que es especialmente útil porque permite prolongar el nombre del objeto. La única restricción en el manejo de estos símbolos es que los nombres de los objetos no pueden comenzar por un carácter numérico. Otra característica de los nombres utilizados por R es que son caso-sensitivos, de modo que "casa" no es equivalente a "Casa", o el objeto "x" es diferente del objeto "X".

```
> x <- 5
> x
[1] 5
> X # el nombre del objeto es x, tecleando X obtenemos
un mensaje de error
Error: objeto 'X' no encontrado
```

3.1.1. *Tipos de objetos*

Un objeto puede ser almacenado bajo diferentes formas ("`type`") . Los tipos de objetos más comunes son los objetos dobles, enteros, complejos, lógicos, carácter y listas.

- Dobles (`Double`)
  R selecciona por defecto datos de tipo "doble" para representar números. Los datos de este tipo son valores numéricos continuos. Para verificar si un determinado dato es doble la instrucción a emplear sería `is.double()`.

```
> x <- 10
> is.double(x)
[1] TRUE
> y <- "a"
> is.double(y)
[1] FALSE
```

- Enteros (`Integer`)
  Se trata de variables numéricas de naturaleza no continua (p.e. número de hijos). Para definir un valor numérico como entero habría que recurrir a la función `as.integer()`.

```
> x <- 7
> is.integer(x)
[1] FALSE
> x <- as.integer(x)
> is.integer(x)
[1] TRUE
```

- Complejos (`complex`)
  Aunque R reconoce los números complejos este tipo de formato no se utiliza apenas en el análisis de datos.

- Lógicos (`logical`)
  Los datos de tipo lógico sólo contienen dos valores `FALSE` (falso) y `TRUE` (verdadero). Son generados por R tras evaluar expresiones lógicas.

```
> x <- 5
> y <- x>10
> y
[1] FALSE
> y <- (x>1) & (x<20)
> y
[1] TRUE
```

Los operadores lógicos validos en R para construir expresiones lógicas se muestran en la tabla siguiente:

Expresiones lógicas	
<, < = >, > = = = ! =	Menor que; Menor o igual que Mayor que; Mayor o igual que Igual a No igual a
Operadores lógicos	
& | !	"y" "o" "no"

Tabla 3.1. Expresiones lógicas

- Carácter (`Character`)
  Los datos de tipo carácter se representan siempre entre comillas ("")

```
> a <- "hoy"
> b <- "33"
> c <- "g"
> x <- 3
> is.character(a); is.character(b); is.character(c);is.
character(x)
[1] TRUE
[1] TRUE
[1] TRUE
[1] FALSE
```

3.1.2. *Estructuras de objetos*

Las estructuras sobre las que trabaja R son:

1 Vector
2 Factor
3 Matriz
4 Array
5 Marco de Datos
6 Lista
7 Función

3.2. **Vector**

La estructura más simple en R es un vector. Un vector es un objeto unidimensional constituido por elementos del mismo tipo. Un único número es un vector de longitud unidad. Los elementos de un vector pueden ser de tipo numérico, lógico o carácter, por lo tanto, habrá vectores numéricos, lógicos carácter. Son ejemplos de vectores,

```
> a <- c(2,4,6,7,8)  # vector numérico creado por
                     concatenación
> b <- a>5           # vector lógico formado tras evaluar
                     la condición
> c <- c(letters)    # vector carácter formado por
                     concatenación
> a
[1] 2 4 6 7 8
> b
[1] FALSE FALSE TRUE TRUE TRUE
> c
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m"
"n" "o" "p" "q" "r" "s"
[20] "t" "u" "v" "w" "x" "y" "z"
>
```

El tipo de dato determinará el modo del vector, `mode()` que junto con su longitud (`lenght`) serán sus atributos.

```
> a <- c(2,4,6,7,8)
> is.vector(a)
[1] TRUE
> mode(a); length(a)
[1] "numeric"
[1] 5
```

3.2.1. ***Generación de vectores***

3.2.1.a. *Concatenación*

Para construir un vector es posible utilizar la función `c` (concatenar), que agrupan elementos simples o vectores definidos previamente.

```
> x <- c(10,5,4,9)# agrupa en un vector los números
entre paréntesis
> x
[1] 10 5 4 9
> CC.SS <- c("sociología", "psicología", "economía")
> CC.SS
[1] "sociología" "psicología" "economía"
> logico <- c(TRUE, TRUE, FALSE, FALSE, TRUE)
> logico
[1] TRUE TRUE FALSE FALSE TRUE
> y <- c(x, 0.78,x,x)
> y
[1] 10.00 5.00 4.00 9.00 0.78 10.00 5.00 4.00 9.00 10.00
5.00 4.00 9.00
```

3.2.1.b. *Secuencias*

Es posible generar secuencias numéricas que se almacenarán como vectores. El modo más sencillo para hacerlo es la utilización del operador "`:`".

```
> hasta20 <- 1:20 #genera una secuencia de números del 1
al 20
> hasta20
[1]1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Se podría haber obtenido una secuencia descendente utilizando `20:1`. La función `seq()` genera secuencias de un modo más general. Sólo es necesario especificar el valor inicial y el valor final de la secuencia junto al incremento aplicado.

```
> a <- seq (from = -5, to = 5, by = 1.5)
> a
[1] -5.0 -3.5 -2.0 -0.5 1.0 2.5 4.0
> b <- seq(10,3,-2) # no es necesario especificar
"from","to" o "by"
> b
[1] 10 8 6 4
```

La función `rep()` repite un vector dado, independientemente del número de elementos que contenga. El primer argumento de la función es el vector, y el segundo argumento puede ser un número que indica cuantas veces tiene que repetirse el vector.

```
> rep(2,3) # repite el número 2 tres veces
[1] 2 2 2
> rep(1:4,3) # repite la secuencia 1-2-3-4 tres veces
[1] 1 2 3 4 1 2 3 4 1 2 3 4
```

Como segundo argumento es posible utilizar un vector de la misma longitud que el usado en el primer argumento. En este caso, cada elemento en el segundo vector indica cuantas veces se repetirá el elemento en el primer vector.

```
> rep(1:3, c(3,2,3))# repite 1 tres veces, 2 dos veces,
3 tres veces
[1] 1 1 1 2 2 3 3 3
> rep(1:3,1:3) # repite 1 una vez, 2 dos veces, 3 tres
veces
[1] 1 2 2 3 3 3
```

Si el vector estuviera formado por caracteres, la generación de secuencias podría incluir la función `letters` o `LETTERS`. Ambas generan los 26 caracteres del alfabeto romano, bien en minúsculas (`letters`) bien en mayúsculas (`LETTERS`).

```
> A <- c(LETTERS [15:5])
> A
[1] "O" "N" "M" "L" "K" "J" "I" "H" "G" "F" "E"
> a <- c(letters [ 1:3])
> a
[1] "a" "b" "c"
> a1 <- rep(c(letters[1:3]),3)
> a1
[1] "a" "b" "c" "a" "b" "c" "a" "b" "c"
> a2 <- paste (a1, 1:9, sep = "")
> a2
[1] "a1" "b2" "c3" "a4" "b5" "c6" "a7" "b8" "c9"
```

3.2.2. *Vectores. Operaciones*

Los cálculos matemáticos ejecutados sobre vectores se llevan a cabo sobre cada uno de sus elementos. Dado un vector `x`, la operación `x*x` generará un nuevo vector, cuyos elementos están definidos por el cuadrado de cada uno de los elementos de `x`.

```
> x <- c(3,4,5,1)
> x*x
[1] 9 16 25 1
> log(x)
[1] 1.098612 1.386294 1.609438 0.000000
> sqrt(x)+5
[1] 6.732051 7.000000 7.236068 6.000000
```

Si los vectores incluidos en una expresión algebraica no son de la misma longitud, el vector de menor longitud se repetirá hasta que alcance la longitud del vector mayor. El caso más simple podría quedar representado por un vector y un número; el número se repetirá tantas veces como elementos tenga el vector.

```
> x <- c(3,4,5,1)
> y <- 3
> z <- x+y
> z
[1] 6 7 8 4
> x <- c(3,4,5,1)
> y <- c(1:5)
> z <- x+y
Warning message:
In x + y : longer object length is not a multiple of
shorter object length
> z
[1] 4 6 8 5 8
```

En el último ejemplo, que puede ser algo artificial, R genera un mensaje de aviso (`Warning message`). Los avisos pueden ignorarse, pero es aconsejable revisar tanto el procedimiento que lo ha originado como el resultado final del proceso.

3.2.3. *Vectores. Funciones*

Existen varias funciones específicas que trabajan sobre vectores. La tabla siguiente presenta un resumen de las más utilizadas.

Función	Salida
`length()`	Longitud del vector
`sum()`	Suma de los elementos del vector
`prod()`	Producto de los elementos del vector
`max()`	Máximo valor del vector
`min()`	Mínimo valor del vector
`cumsum()`	Vector de sumas acumulada de los elementos
`cummax()`	Vector de máximos acumulados
`cummin()`	Vector de mínimos acumulados
`cumprod()`	Producto acumulado de los elementos
`diff()`	Vector de diferencias entre elementos
`unique()`	Un vector de valores únicos
`duplicated()`	Vector lógico que indica si los elementos están duplicados

Tabla 3.2. Funciones aplicables sobre vectores

```
> x <- c(1,3,5,2,2)
> length(x)
[1] 5
> sum(x)
[1] 13
> prod(x)
[1] 60
> max(x)
[1] 5
> min(x)
[1] 1
```

Además R posee varias funciones estadísticas aplicables directamente sobre vectores:

Función	Resultado
`median ()`	Mediana del vector x
`mean()`	Media del vector
`quantile()`	Cuantiles
`IQR()`	Rango intercuartil
`range (x)`	Rango del vector x
`sd (x)`	Desviación estándar
`var()`	Varianza de los elementos
`summary()`	Resumen descriptivos

Tabla 3.3. Funciones estadísticas

```
> cumsum(x); cumprod(x); mean(x); median(x); var(x);
unique(x); sort(x); rev(x)
[1] 1 4 9 11 13
[1] 1 3 15 30 60
[1] 2.6
[1] 2
[1] 2.3
[1] 1 3 5 2
[1] 1 2 2 3 5
[1] 2 2 5 3 1
diff(x)
[1] 2 2 -3 0
> diff(x, lag = 3)# el argumento lag indica el intervalo
                   para el computo de las diferencias
[1] 1 -1
```

3.2.3.a. *Ordenar un vector*

La función `sort()` ordena los elementos de un vector de forma ascendente.

```
> x
[1] 1 3 5 2 2
> sort(x)
[1] 1 2 2 3 5
```

Para ordenar el vector en modo descendente la función a utilizar sería `rev(sort(x)),`

```
> x
[1] 1 3 5 2 2
> rev(sort(x))
[1] 5 3 2 2 1
```

La función `order()` genera un vector cuyos elementos indican el orden ascendente (o descendente) que ocupan en el vector original sus integrantes. La función `rank(x)` devuelve el orden de los elementos en el vector. Esta última permite tratar las igualdades entre elementos (`first, random, max, min`) y los valores ausentes de varias formas.

```
> x
[1] 1 3 5 2 2
> order(x)
[1] 1 4 5 2 3
> rank(x)
[1] 1.0 4.0 5.0 2.5 2.5
> rank (x, ties.method = "max")
[1] 1 4 5 3 3
> rank (x, ties.method = "min")
[1] 1 4 5 2 2
> rank (x, ties.method = "random")
[1] 1 4 5 3 2
> rank (x, ties.method = "first")
[1] 1 4 5 2 3
```

Las funciones anteriores pueden resultar triviales; sin embargo, en muchos contextos de trabajo el objetivo no es ordenar un vector, sino ordenar un conjunto de variables de acuerdo al valor adoptado por otra variable. Con este propósito es habitual utilizar una construcción que pudiera parecer algo abstracta al principio pero que tiene una gran potencia. Por ejemplo, la interpretación de la aplicación de la función `sort()` al vector

`x` es sencilla. Los valores ordenados del vector `x` ocupan los lugares primero, cuarto, quinto, segundo y tercero.

```
> x
[1] 1 3 5 2 2
> order(x)
[1] 1 4 5 2 3
```

Este vector de posiciones es útil para indizar y ordenar cualquier otra variable con el mismo criterio. La indización con un vector que contiene números del 1 al número de elementos del vector genera un nuevo vector ordenado.

```
> x <- c(1,5,3,2,2)
> x
[1] 1 5 3 2 2
> y <- c(10, 14,23,45,89)
> y
[1] 10 14 23 45 89
> o <- order(x)
> o
[1] 1 4 5 3 2
> y[o] ###ordena el vector y en function de los valores
de x
[1] 10 45 89 23 14
```

El ordenamiento puede extenderse a un criterio múltiple añadiendo argumentos a la función `order()`; por ejemplo `order(z,n)`.

3.2.3.b. *Cálculos cruzados,* `outer()`

Esta función es útil para obtener resultados cruzados (sumas, productos....) entre dos vectores. Se genera una tercera matriz de dimensiones `dim(x)`, `dim(y)` cuyos elementos son el resultado de aplicar una determinada función sobre `x` e `y`.

```
> x <- 1:10; names(x) <- x
> y <- 20:15; names (y) <- y
> outer (x,y, "*") ###producto cruzado
20 19 18 17 16 15
1 20 19 18 17 16 15
2 40 38 36 34 32 30
3 60 57 54 51 48 45
4 80 76 72 68 64 60
5 100 95 90 85 80 75
```

```
6 120 114 108 102 96 90
7 140 133 126 119 112 105
8 160 152 144 136 128 120
9 180 171 162 153 144 135
10 200 190 180 170 160 150
> outer (x,y, "-") # diferencia cruzada
20 19 18 17 16 15
1 -19 -18 -17 -16 -15 -14
2 -18 -17 -16 -15 -14 -13
3 -17 -16 -15 -14 -13 -12
4 -16 -15 -14 -13 -12 -11
5 -15 -14 -13 -12 -11 -10
6 -14 -13 -12 -11 -10 -9
7 -13 -12 -11 -10 -9 -8
8 -12 -11 -10 -9 -8 -7
9 -11 -10 -9 -8 -7 -6
10 -10 -9 -8 -7 -6 -5
```

3.3. **Factor**

El factor es el modo que utiliza R para almacenar variables categóricas. En un factor se guardan el número de niveles de la variable categórica, y el número de elementos existente en cada uno de ellos. Por ejemplo, en un cuestionario de personalidad donde hay 1269 participantes, de los cuales 700 son mujeres y 569 son varones, se podría generar un vector de variables carácter de este modo:

```
> sexo <- c(rep("mujer", 700), rep("varon", 569))
```

Es posible convertir el vector en factor por medio de la expresión,

```
> sexo <- factor(sexo)
```

La utilización de factores reduce el espacio de almacenamiento del vector. Internamente el factor sexo es almacenado como 569 valores de 1, y 700 valores de 2. Estos valores 1, 2, o sus correspondientes etiquetas "mujer"/"varón" son los niveles del factor. Por defecto los niveles se almacenan siguiendo el orden alfanumérico, por lo tanto "mujer" precederá a "varón". El comando `levels()` permite visualizar los niveles del factor, y el orden en el que éstos han sido almacenados.

```
> levels(sexo)
[1] "mujer" "varon"
```

El orden de los niveles del factor es importante porque determinará el orden en que aparecerán en las tablas o gráficos en los que vayan a utilizarse. Para imponer un orden predeterminado, el comando a emplear sería `relevel()`.

```
> sexo <- relevel(sexo, ref = "varon")
> levels(sexo)
[1] "varon" "mujer"
O de modo alternativo
> sexo <- factor(sexo, levels = c("varon", "mujer")
> levels(sexo)
[1] "mujer" "varon"
```

3.3.1. *Combinación de niveles del factor*

En ocasiones se hace necesaria la combinación de varios niveles de un factor en un nuevo nivel; por ejemplo, cuando el número de elementos en cada nivel es excesivamente reducido. Para ello, se podría seguir el siguiente procedimiento

```
> edad <- c(rep("niño", 300), rep("adolescente",
200),rep("adulto", 20),rep("mayor", 3))
> edad <- factor (edad)
> levels (edad)
[1] "adolescente" "adulto" "mayor" "niño"
> levels (edad) <- list (niño = "niño", adolescente =
"adolescente", adulto = c("adulto", "mayor"))
> levels (edad)
[1] "niño" "adolescente" "adulto"
```

Otra opción podría ser asignar la misma etiqueta de nivel a dos categorías diferentes.

```
> levels (edad) <- list ("niño","adolescente", "adulto",
"adulto")
> levels (edad)
[[1]]
[1] "niño"
[[2]]
[1] "adolescente"
[[3]]
[1] "adulto"
[[4]]
[1] "adulto"
```

3.3.2. *Generación de factores*

La categorización de un dato continuo crea un factor.. El comando que posibilita esta acción es `cut()`. Como argumento se especifican los intervalos que definirán cada uno de los niveles; la opción para ello es `breaks`.

```
> x <- 1:15
> x
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
> y <- cut(x, breaks = c(0,5,10,15))
> y
[1] (0,5] (0,5] (0,5] (0,5] (0,5] (5,10] (5,10] (5,10]
(5,10] (5,10] (10,15] (10,15] (10,15] (10,15] (10,15]
Levels: (0,5] (5,10] (10,15]
```

3.3.3. *Factores. Operaciones Construcción de tablas*

Una de las operaciones más comunes con factores es la generación de tablas de frecuencias.

```
> table(y)
y
   (0,5]  (5,10]  (10,15]
       5       5        5
> table(sexo)
    sexo
     mujer varon
      700   569
```

Las funciones `table(), xtabs(), y ftable()` son útiles en la creación de tablas unidimensionales, tablas de contingencia bidimensionales y tablas multidimensionales. De todas ellas, `table()` es la más antigua y básica. `table()` genera un objeto que puede fijarse utilizando una regla de asignación (<-).

```
> table(Datos$sexo) ### tabla para la variable sexo
0   1
468 508
Sexo.ocupa <- table(Datos$sexo, Datos$Ocupacion) # tabla
de contingencia entre sexo y ocupación
    0   1  2  3  4
0 324  55 34 25 30
1 232 120 48 51 56
```

Es posible asignar nombres a las filas y a las columnas de cualquier tabla; para ello las funciones específicas son `colnames()`, y `rownames()`.

```
rownames (sexo.ocupa) <- c ("varón", "mujer")
colnames (sexo.ocupa) <- c("0", "1", "2", "3", "4")
sexo.ocupa
        0   1  2  3  4
varón 324  55 34 25 30
mujer 232 120 48 51 56
```

La función `xtabs()` es similar a la función `table()` pero la ejecuta aplicando un modelo. La función simplifica el modo de nombrar las variables de un marco de datos. Las funciones siguientes son equivalentes:

```
xtabs(~Datos$sexo + Datos$ocupacion)
Datos$ocupacion
Datos$sexo  0   1  2  3  4
        0 324  55 34 25 30
        1 232 120 48 51 56
xtabs (~sexo+ocupacion, data = Datos)
ocupacion
sexo  0   1  2  3  4
  0 324  55 34 25 30
  1 232 120 48 51 56
```

La generación de tablas K dimensionales sigue el mismo procedimiento que para las tablas bidimensionales; `table`(variable1, variable2, variable3) ó `xtabs`

Es posible transponer una tabla por medio de la función `t` .

```
> t(table(Datos$sexo, Datos$ocupacion))
    0   1
0 324 232
1  55 120
2  34  48
3  25  51
4  30  56
```

3.3.3.a. *Frecuencias marginales y frecuencias relativas*

Existen funciones específicas para el manejo de tablas; entre ellas las más útiles `margin.table` y `prop.table`. En el argumento de esas funciones se especificará

el valor de 1 si se desean las frecuencias (o proporciones) por filas, o un 2 en caso de que el objetivo se centre en las columnas.

```
> margin.table(sexo.ocupacion,1)
varón mujer
  468   507
> margin.table (sexo.ocupacion,2)
  0   1  2  3  4
556 175 82 76 86

> prop.table (sexo.ocupacion,1)
               0          1          2          3          4
varón 0.69230769 0.11752137 0.07264957 0.05341880 0.06410256
mujer 0.45759369 0.23668639 0.09467456 0.10059172 0.11045365

> prop.table (sexo.ocupacion,2)
              0         1         2         3         4
varón 0.5827338 0.3142857 0.4146341 0.3289474 0.3488372
mujer 0.4172662 0.6857143 0.5853659 0.6710526 0.6511628
```

3.4. **Matriz**

Una matriz es una clase (*class*) de objeto bidimensional constituido por filas y columnas de elementos del mismo tipo. Las filas y las columnas determinan la dimensión o atributos de la matriz (`dim`). Los elementos de una matriz pueden ser de tipo numérico, lógico o carácter y ellos definen el modo (`mode`) de la matriz. Una matriz es, en definitiva, una generalización bidimensional de un vector.

```
> a
     [,1] [,2] [,3]
[1,]   1    4    7
[2,]   2    5    8
[3,]   3    6    9
> attributes (a)
$dim
[1] 3 3
> class(a)
[1] "matrix"
> mode(a)
[1] "numeric"
```

3.4.1. ***Generación de matrices***

3.4.1.a. *La función dim()*

La aplicación de la función `dim()` sobre un vector existente convierte a éste en una matriz con un número de filas y columnas que se especificará por medio de un vector de 2 elementos, `c(nºfilas, nºcolumnas).`

```
> x <- 1:8
> dim (x) <- c(2,4)# convierte el vector x en una matriz 2×4
> x
[,1] [,2] [,3] [,4]
[1,] 1 3 5 7
[2,] 2 4 6 8
```

3.4.1.b. *La función matrix()*

La función `matrix()` genera matrices a partir de un vector. Bastará para ello definir el número de filas, el número de columnas, y el modo en que se irá completando la matriz, bien por filas (`byrow = T`) bien por columnas (`byrow = F`).

```
> x <- matrix(1:8,2,4,byrow = F)# genera una matriz con
2 filas y 4 columnas que se irá completando por columnas
> x
     [,1] [,2] [,3] [,4]
[1,]   1    3    5    7
[2,]   2    4    6    8

x <- matrix(1:8,2,4,byrow = T)# genera la matriz
completándola por filas
> x
     [,1] [,2] [,3] [,4]
[1,]   1    2    3    4
[2,]   5    6    7    8
> a <- 1:8 # se crea el vector a
> x <- matrix(a,2,4)# se convierte a en matriz;es
equivalente al primer ejemplo
```

3.4.1.c. *Concatenación*

Puede crearse una matriz concatenando (uniendo) varios vectores o matrices ya existentes. La función `cbind()` unirá los vectores o matrices por columnas, y la función `rbind()` concatenará los vectores o matrices por filas.

```
> cbind (c(1,2,3),c(4,5,6))
     [,1] [,2]
[1,]   1    4
[2,]   2    5
[3,]   3    6
> rbind (c(1,2,3),c(4,5,6))
     [,1] [,2] [,3]
[1,]   1    2    3
[2,]   4    5    6
```

3.4.2. *Matrices. Operaciones*

En tanto en cuanto las matrices son generalizaciones de los vectores, las funciones matemáticas aplicables sobre vectores son también validas para las matrices, y al igual que en el caso de los vectores son ejecutadas sobre cada uno de los elementos de la matriz.

```
> x <- matrix(1:9,ncol = 3)
> y <- c(5:7)
> x
     [,1] [,2] [,3]
[1,]   1    4    7
[2,]   2    5    8
[3,]   3    6    9
> y
[1] 5 6 7
> x+5
     [,1] [,2] [,3]
[1,]   6    9   12
[2,]   7   10   13
[3,]   8   11   14
> x+y
     [,1] [,2] [,3]
[1,]   6    9   12
[2,]   8   11   14
[3,]  10   13   16
> x*x
     [,1] [,2] [,3]
[1,]   1   16   49
[2,]   4   25   64
[3,]   9   36   81
```

Como se aprecia en el último ejemplo el producto de x*x ofrece como resultado el cuadrado de cada uno de los elementos de la matriz y no el resultado de una multiplicación de matrices.

3.4.3. *Matrices. Funciones*

R dispone de varias funciones específicas relacionadas con el cálculo matricial. Por ejemplo para multiplicar matrices la función a utilizar es `%*%`.

```
> x%*%x
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
La multiplicación de una matriz por su transpuesta
sería: x%*% t(x)
> x%*% t(x)
     [,1] [,2] [,3]
[1,]   66   78   90
[2,]   78   93  108
[3,]   90  108  126
```

La tabla siguiente ofrece un resumen de las funciones específicas para matrices:

Función	Salida
`chol()`	Descomposición de Cholesky
`col()`	Matriz cuyos elementos son el número de columna
`det()`	Determinante de la matriz
`diag()`	Extrae los elementos de la diagonal de x
`eigen()`	Computa Eigenvalues y Eigenvectores
`ncol()`	Devuelve el número de columnas de la matriz
`nrow()`	Devuelve el número de filas de la matriz
`qr()`	Descomposición qr de una matriz
`row()`	Matriz cuyos elementos son el número de fila
`solve()`	Calcula la inversa de x
`svd()`	Descomposición en valores singulares
`t()`	Transpuesta de la matriz x
`var()`	Estima la matriz de varianzas-covarianzas
`cor()`	Estima la matriz de correlaciones
`%*%`	Producto matricial

Tabla 3.4. Cálculo matricial

La aplicación de las funciones estadísticas `var()` y `cor()` descritas para los vectores genera respectivamente, matrices de varianzas/covarianzas, y la matriz de correlación. El resto de funciones son aplicadas sobre cada una de las columnas de la matriz.

```
> a <- matrix (rnorm(9),ncol=3,nrow=3)#generación de una
matriz
> var(a)# matriz de varianzas/covarianzas
          [,1]      [,2]      [,3]
[1,] 0.8009927 0.5779654 0.3640324
[2,] 0.5779654 1.5157221 0.7706619
[3,] 0.3640324 0.7706619 0.4003197
> cor(a)# matriz de correlaciones
          [,1]      [,2]      [,3]
[1,] 1.0000000 0.5245390 0.6428688
[2,] 0.5245390 1.0000000 0.9893516
[3,] 0.6428688 0.9893516 1.0000000
> mean (a)# media aritmética de todos los elementos de
la matriz
[1] -0.8259644
> sd (a)# desviación típica de cada columna
[1] 0.8949819 1.2311466 0.6327082
> summary (a)# resumen de cada columna
      V1                 V2                 V3
Min.   :-1.4145  Min.   :-1.9223  Min.   :-1.9884
1st Qu.:-1.1673  1st Qu.:-1.0427  1st Qu.:-1.4767
Median :-0.9201  Median :-0.1630  Median :-0.9650
Mean   :-0.6707  Mean   :-0.5453  Mean   :-1.2619
3rd Qu.:-0.2988  3rd Qu.: 0.1431  3rd Qu.:-0.8986
Max.   : 0.3225  Max.   : 0.4493  Max.   :-0.8321
```

Además de las operaciones directamente relacionadas con el cálculo matricial, es posible aplicar varias funciones sobre las filas o columnas de una matriz. Entre ellas podríamos citar,

Función	Resultado
`colnames()`	Nombres de las columnas de la matriz
`colSums ()`	Suma los elementos de las columnas
`dim()`	Dimensiones de la matriz
`length()`	Número de elementos de la matriz
`Rownames`	Nombres de las filas de la matriz
`rowSums ()`	Suma los elementos de las filas de una matriz
`colMeans ()`	Obtiene la media aritmética de cada columna
`rowMeans ()`	Obtiene la media aritmética de cada fila

Tabla 3.5. Funciones sobre filas/columnas

```
> x <- cbind(col1 = 3, x2 = 1:8)
> x

     col1 x2
[1,]    3  1
[2,]    3  2
[3,]    3  3
[4,]    3  4
[5,]    3  5
[6,]    3  6
[7,]    3  7
[8,]    3  8
> rowSums(x); colSums(x)
[1]  4  5  6  7  8  9  10  11
col1   x2
  24   36
```

La función `colSums (rowSums)` puede utilizarse para contar el número de apariciones de un determinado valor o el número de veces que los datos cumplen una condición.

```
> colSums(x = = 3)
col1   x2
   8    1
> colSums (x>4)
col1   x2
   0    4
```

3.5. **Array**

Si las matrices pueden definirse como generalizaciones bidimensionales de los vectores, los arrays son las extensiones multidimensionales de las matrices. Se trata de estructuras compuestas por elementos del mismo tipo (numérico, carácter, lógico) que pueden tener más de 3 niveles o dimensiones. La generación de arrays es similar a la generación de matrices.

En el ejemplo que mostramos a continuación creamos un array de 2´3´2 (2 filas, 3 columnas, 2 niveles) al que llamamos `x`. Todos los elementos son 0. Los atributos de un array vienen definidos por sus dimensiones.

```
> x <- array(0,c(2,3,2))
> x
, , 1
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
, , 2
     [,1] [,2] [,3]
[1,]    0    0    0
[2,]    0    0    0
> attributes (x); mode (x); class(x)
$dim
[1] 2 3 2
[1] "numeric"
[1] "array"
```

3.6. **Marco de datos**

El marco de datos o `data frame` es el tipo de estructura más común en R para el análisis de datos. Puede considerarse una generalización de una matriz con la salvedad de que las columnas pueden contener datos almacenados en diferentes modos, es decir, una columna puede ser un vector carácter mientras que otra columna puede ser numérica. Podría equipararse a las matrices de datos utilizadas por los usuarios de los paquetes estadísticos al uso. Podemos pensar en ellas como conjuntos de datos en las que las líneas representan casos, y las columnas variables. Las variables pueden ser de distinto tipo, pero todos los datos referidos a una misma variable son del mismo modo.

Los marcos de datos incorporan (o pueden incorporar) nombres para cada una de las variables o columnas (`names()`) , y nombres para cada fila (`row.names()`). Para mostrar las diferencias entre las estructuras *matriz y marco de datos* imaginemos que disponemos de tres vectores (x, y, z) cuyos contenidos son respectivamente de tipo numérico, numérico y carácter. Si a partir de ellos se genera una matriz (por medio del comando `cbind()`), el modo de la matriz sería carácter. Para mantener el tipo de dato correspondiente a cada columna la opción es necesariamente, `data.frame()`.

3.6.1. ***Marcos de datos. Generación***

3.6.1.a. *Importar datos*

Esta opción, la más utilizada, la veremos en un apartado posterior.

3.6.1.b. *Función data.frame()*

La función `data.frame()` concatena vectores(matrices o marcos de datos) y crea una nueva estructura que incluye nombres para las variables y nombres para los casos. Las funciones `names()` y `row.names()` devolverán estos nombres.

```
> x <- 1:10; y <- round (rnorm(10, 5,2),0); z <- letters
[10:1]
> z
[1] "j" "i" "h" "g" "f" "e" "d" "c" "b" "a"
> juntos <- cbind (x,y,z)
> juntos
        x   y   z
[1,]   "1" "8" "j"
[2,]   "2" "2" "i"
[3,]   "3" "0" "h"
[4,]   "4" "2" "g"
[5,]   "5" "4" "f"
[6,]   "6" "5" "e"
[7,]   "7" "5" "d"
[8,]   "8" "6" "c"
[9,]   "9" "5" "b"
[10,] "10" "6" "a"
> class(juntos)
[1] "matrix"
> mode(juntos)
[1] "character"
> bien <- data.frame(x,y,z)
> bien
    x y z
 1  1 8 j
 2  2 2 i
 3  3 0 h
 4  4 2 g
 5  5 4 f
 6  6 5 e
 7  7 5 d
 8  8 6 c
 9  9 5 b
10 10 6 a
> attributes(bien)
$names
[1] "x" "y" "z"
$row.names
[1] 1 2 3 4 5 6 7 8 9 10
$class
[1] "data.frame"
```

3.6.1.c. *Función as.data.frame()*

La función `as.data.frame()` transforma cualquier objeto en un marco de datos. Por ejemplo, para convertir una matriz cualquiera en un marco de datos se utilizará la siguiente función:

```
x <- as.data.frame(x)
```

```
> x <- matrix(1:9,3,3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x <- as.data.frame(x)
> x
  V1 V2 V3
1  1  4  7
2  2  5  8
3  3  6  9
```

Como puede verse en el ejemplo anterior R asigna los nombres V1, V2 y V3 a las columnas; utilizando la función `names()` (`names()`, `row.names()`) es posible modificarlas.

```
> names(x) <- c("Sociología", "Psicología", "Derecho")
> row.names(x) <- c("Juan", "Maria", "Pedro")
> x
      Sociología  Psicología  Derecho
Juan           1           4        7
Maria          2           5        8
Pedro          3           6        9
>
```

3.6.2. ***Adición de filas/columnas***

Sería posible añadir una columna a un marco de datos por medio de la función `cbind()`. Si quisiéramos añadir al marco de datos de nombre "ejemplo" una nueva variable que se haya almacenada en un vector de nombre "datos" la sintaxis del comando sería:

```
ejemplo <- cbind(ejemplo, final = datos)
```

El lado izquierdo del símbolo de igualdad especifica el nombre que tendrá la variable incorporada en el nuevo marco de datos (`final`). El lado derecho de la igualdad especifica el nombre del vector que se desea incorporar.

Se pueden añadir filas a un marco de datos provenientes de otro marco de datos con el comando `rbind()`, en el que se especificará el nombre de la nueva estructura creada, y las variables comunes de los dos marcos de datos que se quieren integrar.

```
Nueva <- rbind( data.frame1[, c("","")],data.frame2[,
c("","")])
```

3.6.3. *Contenido*

Para visualizar los primeros casos de un marco de datos, la función a utilizar es `head()` , y para obtener un resumen de las variables que lo integran habría que aplicar la función `summary(),` o la función `str().`

```
> summary (Datos)
     Edad      Sexo        Pesoact        Altura        Pesomax
Min.   :18.00 M:155 Min.   : 41.00 Min.   :150.0 Min.   : 41.00
1st Qu.:19.00 V: 37 1st Qu.: 54.00 1st Qu.:162.0 1st Qu.: 56.75
Median :21.00       Median : 60.00 Median :166.0 Median : 62.50
Mean   :22.27       Mean   : 61.22 Mean   :167.6 Mean   : 64.97
3rd Qu.:22.00       3rd Qu.: 67.00 3rd Qu.:172.2 3rd Qu.: 71.00
Max.   :49.00       Max.   :117.00 Max.   :196.0 Max.   :122.00
   Pesomin        Pesideal        item2           item9
Min.   :38.00  Min.   :40.00  Min.   :1.000  Min.   :1.000
1st Qu.:49.44  1st Qu.:52.00  1st Qu.:3.000  1st Qu.:3.000
Median :55.00  Median :56.00  Median :5.000  Median :4.000
Mean   :55.71  Mean   :58.02  Mean   :4.339  Mean   :3.948
3rd Qu.:60.00  3rd Qu.:62.00  3rd Qu.:6.000  3rd Qu.:6.000
Max.   :80.00  Max.   :84.00  Max.   :6.000  Max.   :6.000
```

```
> str(Datos)
'data.frame': 192 obs. of 30 variables:
$ Edad      : num 29 21 46 24 19 18 21 25 20 18 ...
$ Sexo      : Factor w/ 2 levels "M","V": 2 2 2 2 2 2 2 2 2 2 ...
$ Pesoact   : num 84 117 80 77 80 74 78 82.6 70 85 ...
$ Altura    : num 188 182 173 186 187 185 183 174 182 181 ...
$ Pesomax   : num 88 122 82 78 93 76 83 83 73 88 ...
$ Pesomin   : num 80 80 75 72 75 69 73 67 65 78 ...
$ Pesideal  : num 84 80 80 80 79 79 78 78 77 75 ...
```

3.6.4. *Fusionar marcos de datos*

La utilización de la función `cbind()` puede acarrear problemas cuando los datos en dos estructuras están ordenados de acuerdo a criterios diferentes, o cuando el número de casos no coincide en ambas estructuras. La función `merge()` evita este tipo de situaciones; permite fusionar marcos de datos diferentes en función de algún criterio de emparejamiento, que habitualmente es un conjunto de variables que tienen el mismo nombre en los marcos de datos que se quieren fundir.

```
Merge (datoX, datoY, by = "DNI")
```

Esta función fusiona los marcos de datos `datoX` y `datoY` utilizando como variable de emparejamiento el DNI. El origen de las variables fusionadas puede reconocerse fácilmente porque en el nuevo marco de datos se les añade el sufijo `.x` o `.y`.

```
> a
  Num V1 V2 V3
1   5  1  4  7
2   7  2  5  8
3   9  3  6  9
> b
  Num V1 V2 V3
1   5 10 13 16
2   7 11 14 17
3   9 12 15 18
> m <- merge (a,b, by = "Num")
> m
  Num V1.x V2.x V3.x V1.y V2.y V3.y
1   5    1    4    7   10   13   16
2   7    2    5    8   11   14   17
3   9    3    6    9   12   15   18
```

3.7. **Listas**

Una lista es una colección ordenada de elementos de distinto tipo. Una lista puede contener otra lista, y de este modo puede utilizarse para construir estructuras de datos arbitrarias. Las listas son utilizadas por R como salidas de las funciones estadísticas. A menudo son colecciones de estimaciones de parámetros, residuales, índices de ajuste…

3.7.1. *Generación de listas*

El comando `list()` permite crear listas; los nombres de los componentes de la lista pueden especificarse por medio de los argumentos de la función `list()` utilizando para ello el carácter " = ".

```
> x <- 1:7
> y <- c("Ana", "Marcos", "Juan", "Pedro", "Ramón")
> z <- list(secuencia = x, nombres = y)
> z
$secuencia
[1] 1 2 3 4 5 6 7

$nombres
[1] "Ana" "Marcos" "Juan" "Pedro" "Ramón"
```

La función `names()` permite extraer los nombres de los componentes, y al mismo tiempo cambiar sus etiquetas.

```
> names(z)
[1] "secuencia" "nombres"
> names(z) <- c("números", "personas")
> names(z)
[1] "números" "personas"
>
```

También es posible añadir nuevos elementos a la lista; para ello se puede utilizar el doble corchete [[ ]], o el símbolo $:

```
> z[[3]] <- 20:17
> z$saludos <- "como estáis"
> z
$números
[1] 1 2 3 4 5 6 7
$personas
[1] "Ana" "Marcos" "Juan" "Pedro" "Ramón"
[[3]]
[1] 20 19 18 17
$saludos
[1] "como estáis"
```

4. MANIPULACIÓN DE DATOS

Una vez creada una estructura de datos la flexibilidad que ofrece R para su manipulación es enorme; permite seleccionar cualquier elemento/s de una estructura y operar sobre él. La situación más simple puede representarse como la extracción de un elemento perteneciente a un vector `x`; para ello, sería suficiente especificar entre corchetes el lugar que ocupa el elemento dentro del vector. Por ejemplo `x[3]`.

```
> x <- c(seq(1, 9, by = 2))
> x
[1] 1 3 5 7 9
> x[3]
[1] 5
```

En el caso de un objeto matriz y dado su carácter bidimensional, habría que indicar tanto el subíndice de la fila como el subíndice de la columna. Para extraer el elemento de la segunda fila y primera columna de una matriz `x`, habríamos de especificar, `x[2,1]`

```
> x <- matrix(1:9,3,3)
> x
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> x[2,1]
[1] 2
```

4.1. Vectores

En el manejo de vectores los subíndices de localización pueden utilizarse de 4 formas diferentes:

1. Un vector de números naturales, en el cual se indican los lugares de los elementos a extraer

```
> x
[1] 1 3 5 7 9
> x[1:3]
[1] 1 3 5
> x[c(4,2,4)]
[1] 7 3 7
```

2. Un vector lógico de la misma longitud que el vector original que se genera tras la evaluación de una condición.

```
> x
[1] 1 3 5 7 9
> y <- x>6
> y
[1] FALSE FALSE FALSE TRUE TRUE
> x[y]
[1] 7 9
O directamente
> x[x>6]
[1] 7 9
```

3. Un vector de números naturales negativos. Por medio de esta indexación se seleccionan todos los elementos del vector excepto los indicados con valores negativos.

```
> x
[1] 1 3 5 2 2
> x[-(1:3)]
[1] 2 2
```

4. Extracción de elementos que cumplen determinada condición

Muchas veces resulta conveniente extraer de un vector las posiciones que cumplen determinada condición; la función que utilizaremos para ello es `which(condition)`

```
a <- 1:19
> a
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
> which (a>15)
[1] 16 17 18 19
a <- c (1,3,5,6,7,9,2)
> which (a = = 2)
[1] 7
```

La modificación de elementos que cumplen determinada condición es posible utilizando la regla de asignación del siguiente modo:

```
> x[x>6] <- 2
> x
[1] 1 3 5 2 2
```

4.2. **Matrices**

Para el tipo de objeto matriz, la indización adopta básicamente tres formas:

1. Utilización de un par de subíndices (fila, columna). "Fila" es un vector que indica la/s fila/s que se quieren seleccionar, y "columna" indicaría la/s columna/s. Ambos valores pueden ser un vector de longitud unidad, pueden tener valores negativos o pueden estar ausentes.

```
> x <- matrix(1:36, nrow = 6)
> x
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    7   13   19   25   31
[2,]    2    8   14   20   26   32
[3,]    3    9   15   21   27   33
[4,]    4   10   16   22   28   34
[5,]    5   11   17   23   29   35
[6,]    6   12   18   24   30   36
> x[4,5] # selecciona el elemento que ocupa la fila 4
columna 5
[1] 28
> x[4,] # selecciona todos los elementos de la fila 4
[1] 4 10 16 22 28 34
> x[,5] # selecciona todos los elementos de la columna 5
[1] 25 26 27 28 29 30
x[4, c(3:5)] # selecciona los elementos de las columnas
3-4-5 de la fila 4
[1] 16 22 28
> x[-1,-3] # genera una matriz menor eliminando la fila 1
y columna 3
     [,1] [,2] [,3] [,4] [,5]
[1,]    2    8   20   26   32
[2,]    3    9   21   27   33
[3,]    4   10   22   28   34
[4,]    5   11   23   29   35
[5,]    6   12   24   30   36
>
```

2. Una matriz lógica de la mima dimensión que la matriz original.

```
> y <- x>15
> y
      [,1]  [,2]  [,3] [,4] [,5] [,6]
[1,] FALSE FALSE FALSE TRUE TRUE TRUE
[2,] FALSE FALSE FALSE TRUE TRUE TRUE
[3,] FALSE FALSE FALSE TRUE TRUE TRUE
[4,] FALSE FALSE  TRUE TRUE TRUE TRUE
[5,] FALSE FALSE  TRUE TRUE TRUE TRUE
[6,] FALSE FALSE  TRUE TRUE TRUE TRUE
> x[y]
[1] 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
```

En este caso `x[y]` es un vector.

3. Una matriz `r` con dos columnas. Cada una de las filas de la matriz `r` selecciona un elemento de la matriz original. El resultado es un vector que contiene los elementos seleccionados.

```
> r
     [,1] [,2]
[1,]    1    3
[2,]    2    1
[3,]    3    2
> x[r]
[1] 13 2 9
```

4.3. **Marco de datos**

Dado que un marco de datos puede considerarse la generalización de una matriz, todos los procedimientos de extracción de elementos explicados para las matrices pueden aplicarse directamente sobre los marcos de datos.

Además es posible utilizar el símbolo “$ “ (marco de datos$nombre variable) o bien corchetes, situando entre comillas el nombre de la variable a seleccionar. La utilización de corchetes puede generar vectores o marcos de datos. Si se selecciona una columna utilizando un doble corchete [[ ]], el resultado es un vector. Sin embargo la utilización de un corchete simple [], generará un marco de datos. Con este último procedimiento es posible seleccionar simultáneamente más de una columna.

```
Datos$Edad
Datos[[“Edad”]]
Datos[c(“Edad”, “Altura”)]
```

La selección de un subconjunto de elementos que cumplan determinada condición es posible con la función `subset()`. En el ejemplo siguiente se seleccionan aquellas mujeres que miden más de 175cm y se crea un marco de datos (mujer.alta) que contiene las variables Pesoact (peso actual) y Edad

```
Mujer.alta <- subset (Datos, Sexo = = "M" & Altura>175,
select = c(Pesoact, Edad))
```

4.4. **Carácter**

R dispone de varias funciones para la manipulación de caracteres; entre ellas tal vez las más utilizadas sean `nchar()`, `substring()`, `paste()` y `sep()`. La función `nchar()` devuelve la longitud de un objeto carácter y la función `substring()` extrae o reemplaza subcadenas.

```
hola <- "hola a todos"
> hola
[1] "hola a todos"
> nchar (hola)
[1] 12
> substring(hola, 2,6)
[1] "ola a"
> substring("abcdef", 2,3)
[1] "bc"
> substring("abcdef",1:6,1:6)
[1] "a" "b" "c" "d" "e" "f"
> substring (hola, 11, 11) <- c("as")
> hola
[1] "hola a todas"
```

La función `paste()` es muy útil para concatenar dos o más objetos carácter.

```
> paste("código", 1:5, sep = ".")
[1] "código.1" "código.2" "código.3" "código.4"
"código.5"
```

El argumento `sep` permite especificar el contenido a insertar entre dos objetos carácter. En el caso de que no queramos utilizar ningún separador la opción sería `sep = ""`.

5. LECTURA Y GRABACIÓN DE DATOS

R no es un editor de datos, y el modo de trabajo habitual comienza con la lectura de datos desde un entorno externo.

5.1. Directorio de trabajo

R utiliza por defecto un directorio de trabajo, en el que se guardan las sesiones de trabajo, las salidas o los objetos que se desee almacenar. Puede comprobarse cual es el directorio de trabajo activo con la función `getwd();`este puede modificarse con el comando `setwd()`. La fijación del directorio de trabajo por parte del usuario es una opción recomendable, para ello es importante anotar que en R los *path* se definen a través del símbolo "/" o "\\"en lugar del símbolo habitual bajo Windows "\".

```
> setwd ("C:\\Documents and Settings\\Rmanual") ## CORRECTO
> setwd ("C:/Documents and Settings/Rmanual") ## CORRECTO
> setwd ("C\Documents and Settings\Rmanual") ## NO NO NO CORRECTO
```

5.2. Introducción manual de datos

Si el conjunto de datos con los que se va a operar es pequeño, los datos pueden introducirse a través del editor de R, o utilizando la consola.

5.2.1. *Uso del editor,* ***fix()***

R dispone de varias funciones que cumplen esa finalidad, `edit(), data.entry()` y fix(). En el ejemplo siguiente se muestra la secuencia para la edición de datos; primero es necesario crear un objeto, y después editarlo.

```
> a <- data.frame()
> edit(a) # no guarda los cambios
Data frame with 0 columns and 0 rows
> a <- edit(a)# para almacenar los cambios
> a
  var1 var2
1   12    3
2    4    5
> fix(a)# almacena las modificaciones
> a
  var1 var2
1   12    3
2    4    5
3    3    4 [1] 3
```

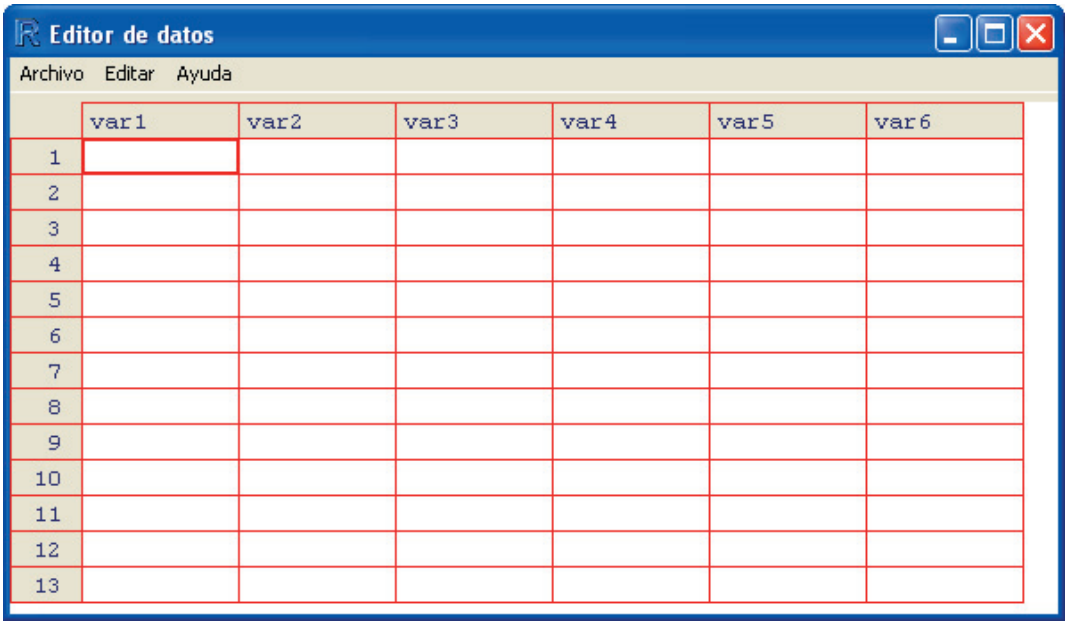

Figura 5.1. Editor de datos

Los cambios producidos en una estructura de datos con la utilización de `edit()` no se almacenan de forma automática. Para salvar los datos introducidos o modificados es necesario asignar los cambios a un objeto. La función que permite salvar los cambios es fix().

5.2.2. *Uso de la consola,* `scan()`

El uso más simple de la función `scan()` está relacionado con la introducción de datos desde la consola de R. Los argumentos por defecto de `scan()` indican que los datos serán introducidos utilizando el teclado. La introducción de datos se da por finalizada tras una línea en blanco o tras la pulsación de Ctrl.+D.

```
> x <- scan()
1: 23 78 76 90
5: 100 67
7: 88
8:
Read 7 items
> x
[1] 23 78 76 90 100 67 88
```

5.3. **Importar datos**

5.3.1. *Archivos de texto delimitados,* ***read.table()***

La función `read.table()` lee archivos externos en el que las líneas han sido definidas como casos y las columnas como variables. El formato básico de la función `read.table()`, incluye el nombre del fichero a leer (con el path completo), un indicativo del punto decimal, `dec = ",",` y la definición del separador entre variables o delimitador de campos. Si la primera línea del fichero contuviera las etiquetas de las variables habría que indicar `header = T`.

La función `read.table()` asume que los caracteres ausentes en el fichero a importar están representados por NA, en el caso de no ser así, existen dos opciones; modificar estos valores antes de importar los datos, o definir los valores ausentes por medio del argumento `na.string.` Si se desean saltar varias líneas (algunas salidas de programas de análisis de datos generan varias líneas de cabeceras) el argumento a utilizar es `skip`. La función `read.table()` es especialmente útil para la importación de ficheros que hayan sido grabados como texto y cuyos separadores han sido bien definidos.

```
datos <- read.table("C:/misdatos.txt", header = TRUE, dec = ",")
   Edad  Sexo  Pesoact  Altura  Pesomax  Pesomin  Pesideal
1  29    V     84,00    188,00  88,00    80,00    84,00
2  21    V     117,00   182,00  122,00   80,00    80,00
3  46    V     80,00    173,00  82,00    75,00    80,00
4  24    V     77,00    186,00  78,00    72,00    80,00
5  19    V     80,00    187,00  93,00    75,00    79,00
……
```

5.3.2. *Archivos de texto con formato fijo,* ***read.fwf()***

Esta función lee ficheros de datos grabados en un formato fijo y los transforma en marcos de datos. La utilización de esta función exige indicar el ancho de las columnas

que contienen las variables; La especificación se lleva a cabo a través del argumento `widths`, que no es más que un vector de valores enteros que indica el número de columnas que ocupa cada variable.

Supongamos que tenemos un fichero cuyas dos primeras líneas son 123456, y 987654. La instrucción para la lectura de este fichero sería la siguiente; en ella se muestra el efecto de la definición de argumentos diferentes en `widths`.

```
> datos <- read.fwf(fichero, widths = c(1,2,3))
  V1 V2  V3
1  1 23 456
2  9 87 654
> datos <- read.fwf(fichero, widths = c(1,-2,3))
  V1  V2
1  1 456
2  9 654
```

5.3.3. *Archivos sin formato fijo,* `scan()`

Esta función, `scan()`, permite la lectura de ficheros de datos externos que no tiene un formato fijo. Suponiendo que se desea leer una matriz de datos 3´4 cuyo nombre es `datos.txt` la instrucción sería:

```
> x <- matrix(scan("datos.txt"), ncol = 4, byrow = T)
Read 12 items
> x
     [,1] [,2] [,3] [,4]
[1,]    1    3    4    7
[2,]    5    9    8   19
[3,]    3    7    8    8
```

Al igual que `read.table()`, `scan()` permite la utilización del argumento `skip` para especificar el número de líneas que no deben de importarse. La diferencia fundamental entre `read.table()` y `scan()` es la mayor flexibilidad de ésta última respecto al tipo de fichero de datos a importar.

5.3.4. *Enlaces con otros programas*

Cuando se quieren exportar datos provenientes de algún otro software para el análisis de datos, una posible vía es transformar el archivo a formato ASCII para luego importarlo a R. Sin embargo, esa no es la única vía. Existe un paquete clasificado como "paquete recomendado" que permite una comunicación fluida entre programas para el análisis de datos; `foreign`. El paquete `foreign` incorpora varias funciones para la

lectura de datos provenientes de SPSS (`.sav` formato), SAS, Epi-Infor (`.rec`), Stata, Systat, Minitab ó S-plus.

Para la lectura de datos almacenados en bases de datos CRAN dispone también de varios paquetes; por ejemplo, `RODBC`; reconoce datos guardados en aplicaciones tan comunes como Excel o Access.

Para una información más completa sobre el tema puede consultarse el pequeño manual que incorpora R-, "`R Data Import/Export`".

5.4. **Almacenar el trabajo realizado**

R permite almacenar todo el espacio de trabajo creado durante una sesión. Bajo Windows la forma más sencilla de salvar el espacio de trabajo es utilizar las opciones de la barra de menú `Archivo > Guardar área de trabajo`; esta opción genera un archivo de extensión `.RData` que se almacena en el directorio de trabajo. Se obtiene el mismo resultado tecleando `save.image(file = "misdatos.RData").` Basta teclear el archivo con extensión `.RData` para recuperar todos los objetos utilizados durante la sesión de trabajo. Si quisiéramos almacenar sólo un objeto o varios objetos, el comando a utilizar sería `save( objeto/s, file = "misdatos.RData")`.

Antes de abandonar una sesión de trabajo R ofrece siempre la posibilidad de guardar una imagen del espacio de trabajo. Dado que habitualmente son muchos los objetos que se pueden ir generando, conviene adoptar el buen hábito de ir eliminando aquellos objetos que no necesitemos; el comando que ejecuta esa opción es `rm()`. En lugar de salvar toda el área de trabajo, nosotros preferimos guardar los códigos utilizados para crearla; accediendo a ello es prácticamente inmediato reproducir el trabajo anterior. Los comandos que se han ido generando se almacenan en un fichero con extensión `.Rhistory` , del cual son fácilmente recuperables desde cualquier editor de texto.

5.5. **Grabar datos**

La grabación externa de un vector o matriz de datos generados por R se realiza con el comando `write()`.

```
write(x,file = "misdatos")
```

En este ejemplo el nombre del vector en R es `x` y entrecomillado se indica el nombre que se asigna al fichero de datos que se va a grabar. Esta función permite especificar el número de columnas por medio del argumento `ncolumns`.

Si el conjunto de datos a grabar es un marco de datos, la función a utilizar sería `write.table()`. Entre los argumentos posibles de esta función están `sep` , que in-

dica el separador a utilizar entre variables; la opción "`\t`" que indica la utilización de tabuladores, y la opción "`,`"para el uso de la coma. También es posible especificar si se desea que se graben los nombres de las variables y los nombres de las filas; las opciones correspondientes son `row.names` y `col.names`.

```
write.table(dataframe, file = "", sep = "",row.names =
FALSE, col.names = FALSE)
```

6. PROGRAMACIÓN BÁSICA

6.1. Control de flujo

R, al igual que cualquier otro lenguaje de programación, permite definir acciones dependientes del cumplimiento de determinada condición, o acciones repetitivas que se ejecutarán hasta que se cumpa cierto criterio; Las funciones que permiten el control de flujo son básicamente `for(), if(), repeat(),` y `while().`

6.1.1. *Bucle* `for()`

El comando `for()` permite especificar el número de veces que se ejecutará una secuencia de comandos. Funciona del siguiente modo: un indizador (`i`) adopta valores de forma secuencial, y para cada uno de ellos se ejecuta una serie de funciones, que finalizarán cuando `i` alcance un valor prefijado. En el siguiente ejemplo se define una variable contador (`i`) cuyo primer valor es 1 (`i in 1`); se ejecuta la función (i^3) y se imprime su resultado (`print`). Tras estas dos operaciones (cálculo e impresión) el contador (indizador) incrementa su valor en 1. Se repiten las funciones para este nuevo valor de i que ahora es igual a 2, (2^3) y se imprime el resultado. Este bucle se repite en 5 ocasiones, para cada una de las cuales los valores del contador son: 1, 2, 3, 4 y 5. En este punto finaliza el proceso. El diagrama de flujo correspondiente a esta simple secuencia sería:

```
> for (i in 1:5) {print (i^3)}
[1] 1
[1] 8
[1] 27
[1] 64
[1] 125
```

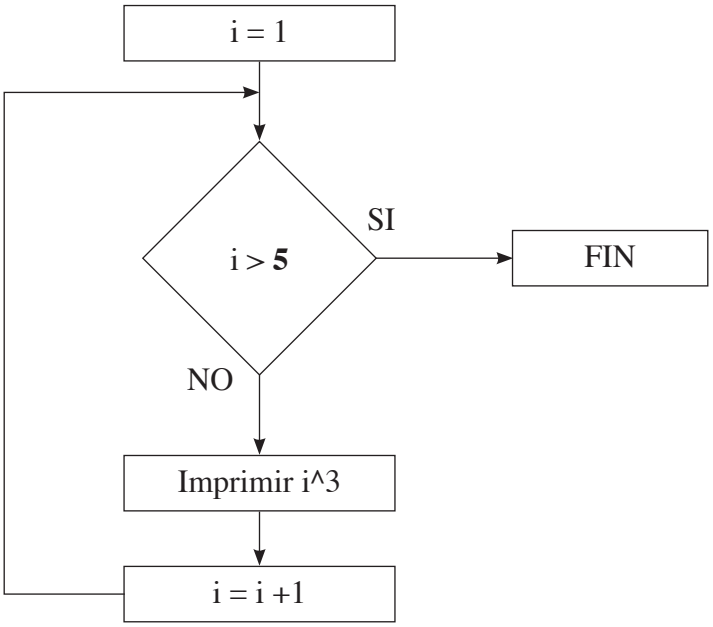

Figura 6.1. Diagrama de flujo.Ejemplo `for()`

```
> numeros <- c(1:12)#genera un vector de 12 números
> secuencia <- numeric(12)# crea dos vectores de
longitud 12
> numeros
[1] 1 2 3 4 5 6 7 8 9 10 11 12
> secuencia[1] <- 1 # fija el primer valor del vector a 1
> for (i in 2:12) {
secuencia[i] <- secuencia [i-1]+numeros[i]} # comenzando
con el valor 2 suma el contenido del vector números en
la posición "i" con el contenido del vector secuencia en
la posición "i-1"
> secuencia
[1] 1 3 6 10 15 21 28 36 45 55 66 78
```

6.1.2. *Comando* `if()`

El comando `if()` permite controlar las ejecuciones que estarán condicionadas al cumplimiento de determinada condición

```
> x <- 5
> if (x >2 ) y <- 3*x else y <- x^3
> y
[1] 15
```

La función `if(),` se utiliza a menudo dentro de funciones definidas por el usuario. Por ejemplo,

```
corgraf <- function (x,y, graf) {
      if (graf = = TRUE) plot(x,y)
      cor (x,y)
                  }
corgraf (c(2,3,4,4,5,6,6), c(4,6,7,3,4,5,6), FALSE)#
solo se imprime el valor del coeficiente de correlación
[1] 0.1575522
corgraf (c(2,3,4,4,5,6,6), c(4,6,7,3,4,5,6), TRUE)# se
representan gráficamente los valores de los vectores
```

6.1.3. *Comando* `while()`

El comando `while()` se utiliza cuando se quieren repetir determinadas funciones pero el número de veces en que han de repetirse no está prefijado.

```
a <- 3
while (a < 5 ) {
      t <- a^3+6
      a <- a+1
      print (t); print (a)
            }
[1] 33
[1] 4
[1] 70
[1] 5
```

6.1.4. *Comando* `repeat`

El comando `repeat` repite una sucesión de comandos hasta que se ejecuta la función `break`

```
x <- 3
repeat {
     a <- x^2+7
     if (a > 1999) break
     print (a)
     x <- x+9
  }
[1] 16
[1] 151
[1] 448
[1] 907
[1] 1528
```

Construcción	Descripción
`for (vector) {expesión}`	Repite la expresión el número de veces especificado
`if (condicion) {expresión}`	Evalúa la condición y si es verdadera ejecuta la expresión
`if (condicion) {expresión}` `else {expresión cuando FALSE}`	Evalúa la condición y si es verdadera ejecuta la expresión, en casa de que sea falsa ejecuta conexp
`while (expresión) {expression}`	Se repite la expresión hasta que se cumpla determinada condición
`repeat {expresión}`	Normalmente incluye break-tambien puede utilizarse en for() and while()

Tabla 6.1. Control de flujo

6.2. **Cálculos eficientes**

6.2.1. ***Cálculos vectorizados***

Una de las características más relevantes de R es que permite efectuar cálculos vectorizados sobre todos los elementos de una estructura de datos; el objetivo de los cálculos vectorizados es evitar el uso de bucles, y de ese modo optimizar el procedimiento de ejecución. Por ejemplo, suponiendo que disponemos de un vector de tipo numérico en el que se desea reemplazar los valores mayores que 5, se podría utilizar un código que comparara de forma secuencial cada uno de los elementos del vector con el número 5 y proceder en un segundo paso a la sustitución de ese elemento. El código podría ser algo similar a:

```
> x
 [1]  1  2  3  4  5  6  7  8  9 10
> for (i in 1:length(x))
{if (x[i] > 5) { x[i] <- 88}}
> x
 [1]  1  2  3  4  5 88 88 88 88 88
```

Sin embargo el cálculo sería más eficiente utilizando,

```
> x[x>5] <- 88
> x
 [1]  1  2  3  4  5 88 88 88 88 88
> x <- ifelse(x>5, 88, 99)
> x
 [1] 99 99 99 99 99 88 88 88 88 88
```

6.2.1.a. *Familia de funciones apply()*

`apply` () permite la ejecución de funciones sobre partes de una matriz (o array); sólo será necesario indicar la función a ejecutar (suma, producto, media, varianza…) y si ésta se aplicará sobre las filas de la matriz (1), o sobre las columnas (2).

```
> x <- matrix (1:8, ncol = 2)
> x
     [,1] [,2]
[1,]   1    5
[2,]   2    6
[3,]   3    7
[4,]   4    8
> apply (x, 1, sum)# suma los valores de las filas de x
[1] 6  8 10 12
> apply (x, 2, prod)# multiplica los valores de las
columnas
[1] 24 1680
> apply (x,1, mean) # calcula la media de cada fila
[1] 3 4 5 6
> apply (x, 2, mean)# calcula la media de cada columna
[1] 2.5 6.5
```

Además de `apply()` existen funciones como `lapply()` o `sapply()` que se aplican sobre los elementos de una lista. El resultado de `lapply()` es siempre una lista, y el resultado de `sapply()` es un vector o array.

```
> x <- list (a = rnorm(10), b = rnorm(100), c = rnorm(25))
> x
$a
  [1]  1.69317368 -0.57942986  0.27186422 -0.20549261 -0.02490787 1.14610459
  [7] -0.46420521 -0.27506592 -0.76638539 -0.63043261
$b
  [1] -1.04088067 -0.67542481  0.93125504 -0.03588596 -0.96772352 1.53851422
  [7] -1.15750319  0.77031582  0.45507832  0.62870478 -1.40918274 0.02362100
..
 [85]  0.76023098  1.12208219  0.08822311  0.83834441 -0.36147960 -1.67444295
 [91] -0.97444326 -0.50934257  1.59798473  0.52551713  0.64930165 -0.55317999
 [97] -0.48262064  0.73437382  1.08350851 -0.28369426
$c
  [1]  0.64005752  1.14757901  0.02566498 -0.95942991  2.20059902  1.38534784
  [7]  2.29494228  0.99401506  0.50010636  2.27580507  2.26455848 -0.69742313
 [13] -3.05054457 -1.93938677  1.12395646 -0.95100759 -0.16679956 -1.67270147
 [19]  0.92047540 -1.22516083 -0.35445521 -1.71672928  2.06621460 -0.03262274
 [25] -1.23337723
> sapply (x, mean)
          a           b           c
 0.01652230  -0.08685703  0.15358735
> lapply(x, mean)
$a
[1] 0.01652230
$b
[1] -0.08685703
$c
[1] 0.1535874
```

Si el objetivo es realizar cálculos condicionados sobre una variable en función de los niveles de un factor la función a utilizar es `tapply()`. Por ejemplo, calcular las medias de la variable altura en función del sexo:

```
> sexo <- c("mujer", "varon", "varon", "mujer", "mujer",
"varon", "varon", "varon")
> altura <- c(1.60, 1.87, 1.70, 1.69, 1.58, 1.4, 1.80,
1.79)
> sexo.factor <- as.factor (sexo)
> table(sexo)
sexo
hombre  varon
     5      3
> tapply(altura, sexo, mean)# medias en altura por sexo
  varon     mujer
1.712000  1.623333
```

6.3. **Valores faltantes**

En el análisis de datos es importante el tratamiento que se haga de los valores faltantes. R asigna el código "`NA`" (`Non available`) a los valores ausentes de cualquier estructura de datos. Prácticamente todas las operaciones ejecutadas sobre estructuras que contienen valores ausentes generan como resultado un valor faltante.

```
> x <- c(1,2,3,NA,8,9)
> x
[1]  1  2  3 NA  8  9
> sum(x)
[1] NA
> mean(x)
[1] NA
> var(x)
[1] NA
```

Para obtener información sobre los valores faltantes presentes en un objeto la función a utilizar es `is.na()`.

```
> z <- c(1:3, NA)
> z
[1]  1  2  3 NA
> ind <- is.na(z)
> ind
[1] FALSE FALSE FALSE TRUE
> x <- c(NA,3:6)
> sum(x)
[1] NA
> mv <- is.na(x)
> mv
[1] TRUE FALSE FALSE FALSE FALSE
```

Para excluir los valores ausentes de las operaciones que se vayan a ejecutar se debe de especificar el argumento `na.rm = TRUE`

```
> x <- c(1,2,3,NA,8,9)
> sum (x,na.rm = TRUE)
[1] 23
> mean(x,na.rm = TRUE)
[1] 4.6
```

Otro modo de proceder podría ser,

```
> sum(x[!is.na(x)])
[1] 23
> mean(x[!is.na(x)])
[1] 4.6
```

Si se desea asignar a los valores ausentes un determinado valor,

```
> x[is.na(x)] <- 99
> x
[1] 99  3  4  5  6
```

Para evitar problemas es importante tener en cuenta que `is.na(x)` no es equivalente a "`x = = NA`"; este último indicaría que x es una variable cuyo valor es "`NA`".

```
> x <- c(rep(NA,3), 5:3)
> x
[1] NA NA NA 5 4 3
> x.sinNA <- na.omit(x)
```

Si quisiéramos eliminar todos los valores ausentes en un vector, matriz, marco de datos, la opción sería `na.omit()`.

```
> x <- c(rep(NA,3), 5:3)
> x
[1] NA NA NA  5  4  3
> x.NAgabe <- na.omit(x)
> x.NAgabe
[1] 5 4 3
attr(,"na.action")
[1] 1 2 3
attr(,"class")
[1] "omit"
> sum(x.NAgabe)
[1] 12
```

Además de los valores ausentes (`NA`), R puede generar valores `NaN` (`not a number`) cuando se encuentra con expresiones como 0/0, o `Inf-Inf`.

```
> 0/0
[1] NaN
> Inf-Inf
[1] NaN
> 4/0
[1] Inf
```

6.4. **Funciones**

R ejecuta sus acciones por medio de funciones. El formato de una función en R es fijo, y viene determinado por el nombre de la función y sus argumentos. Por ejemplo, en la siguiente función "`apply(x,1,sum)`" "`apply`" es el nombre de la función y entre paréntesis se indican sus argumentos; en este caso señalan el objeto sobre el que se aplicará la función (matriz `x`), si esta se ejecutará sobre sus filas o sobre sus columnas (`1` es el indicativo de fila) y que función va a ser aplicada (en este caso la suma, `sum`). Para comprobar los argumentos formales de cualquier función, así como sus valores por defecto, la opción es `args`() .

```
> args (apply)
function (X, MARGIN, FUN, ...)
NULL
> args (rnorm)
function (n, mean = 0, sd = 1)
```

Hasta ahora se han descrito varias funciones pertenecientes a R para el trabajo con estructuras de datos. Sin embargo, cada usuario puede definir funciones según sus necesidades; para ello ha de respetar el formato general de una función:

```
Nombre <- function (argumento1, argumento2,…) { cuerpo
de la función}
```

Los argumentos hacen referencia a los datos de entrada de la función; aunque es cierto que no todas las funciones necesitan datos de entrada. El cuerpo de la función queda definido por los comandos e instrucciones que se desean ejecutar sobre los datos de entrada. La función devolverá un sólo valor con formato de vector, matriz u otra estructura. En el ejemplo siguiente se ha definido una función de nombre `potencia` que calcula el cuadrado del dato de entrada independientemente de si éste es un escalar, un vector o una matriz.

```
potencia <- function (x) { x^2}
x <-  5
potencia (x)
[1] 25
x <- c(2,3,4)
potencia (x)
[1]  4  9 16
> x <- matrix( 1:9, ncol = 3)
> x
     [,1] [,2] [,3]
[1,]   1    4    7
[2,]   2    5    8
[3,]   3    6    9
> potencia (x)
     [,1] [,2] [,3]
[1,]   1   16   49
[2,]   4   25   64
[3,]   9   36   81
```

La siguiente función calcula la media aritmética:

```
> media function(x){sum(x)/length(x)}
> media (rnorm (10000))# se generan 10000 valores
pseudoaletoarios que siguen una distribución normal y se
estima su media
[1] -0.01130342
> media (1:40)# se calcula la media de la secuencia 1:40
[1] 20.5
```

6.4.1. *Argumentos de una función*

Cuando se define una función es opcional especificar valores a sus argumentos. En la función anterior (`potencia`) por ejemplo, el valor por defecto es 2, y no necesita ser especificado; pero si se desea calcular potencias diferentes a 2, en los datos de entrada debería de especificarse el valor de la potencia:

```
> potencia <- function (x, k = 2) { x^k}
>
> x <- 5
> potencia (x)
[1] 25
> potencia (x, 3)
[1] 125
> x <- c(2,3,4)
> potencia(x, 3)
[1]  8 27 64
```

Una función ejecuta determinadas operaciones sobre los datos; ahora bien, si el objetivo es conservar los resultados generados, es necesario definir la asignación del resultado a un objeto; en caso contrario se perderían.

```
resultado.potencia <- potencia(x)
```

6.4.2. *Modificar funciones* **`fix()`**

La función `fix()` permite crear una función o editar una función ya existente, y reemplazarla por una nueva versión. Si se teclea `fix(potencia),` se abre una ventana de edición en la que aparece la función especificada lista para ser editada. Una vez modificada, los cambios introducidos quedan automáticamente almacenados en la función "`potencia`".

```
function (x, k = 2) { x^k+6}
> potencia (2,3)
[1] 14
```

Para crear una función de nombre “factorial” se teclea `fix(factorial),` que abrirá una ventana de edición de funciones.

```
fix(factorial)
function (n)
{prod(n:1)
}
> factor.n(3)
[1] 6
> factor.n(6)
[1] 720
```

6.5. **Depuración y actualización de códigos**

El proceso de depuración y corrección de errores (`bug`) que se realizan durante la programación se conoce como “`debugging`”. Para localizar un error en un código, el primer paso es leer los mensajes de error que devuelve el programa y por supuesto, entenderlos. R ofrece una herramienta de uso sencillo que ayuda a detectar el punto del programa en el que se produce el error. La función `traceback()` aporta información sobre el punto en el que se genera el mensaje de error.

```
saludo <- function (x)   {
  if (x>0) print (“Hola”)
  else
  print (“Hasta Luego”)
  }
saludo(3)
saludo(-4
> saludo(3)
[1] “Hola”
> saludo(-4)
[1] “Hasta Luego”

> x <- log(-1)
Aviso en log(-1): Se han producido NaNs
> saludo(x)
Error en if (x > 0) print(“Hola”) else print(“Hasta Luego”) :
  valor ausente donde TRUE/FALSE es necesario
```

El mensaje de aviso que aparece en `x <- log (-1),` se convierte en un error al trasladarlo a la función `saludo`. La función incluye una comparación del valor de x con 0, sin embargo x es NaN y R no lo puede resolver.

6.6. **Generación pseudoaletaoria de datos**

Uno de los métodos más utilizados como aproximación a las propiedades de las variables aleatorias es el método Monte Carlo. Por ejemplo, para aproximar la media $\mu = E(X)$ por medio de simulación Monte Carlo, se generan `m` valores independientes e idénticamente distribuidos de `X, X1,…`X_m, y se utiliza la media muestral $\overline{X} = (1/m)\Sigma Xi$ como estimador de E(X). Según la ley de los grandes números para valores altos de m, $\overline{X}$ es una buena aproximación a E(X). Es más, según el teorema central del límite si `m` es alto, la distribución de la media muestral $\overline{X}$ puede aproximarse por una distribución normal con media μ y varianza σ^2/m. Donde σ^2 es la varianza V(X) que puede aproximarse por la varianza muestral $s^2 = [1/(m-1)] \Sigma(X_i-\overline{X})^2$.

El mecanismo básico de generación de números pseusoaletaorios lo describimos ejemplificándolo con la simulación de variables aleatorias independientes y uniformes en el intervalo [0,1]. Un generador de números aleatorios produce una secuencia de números pseudoaleatorios $u_0,u_1,u_2,\ldots$ que simulan una variable aleatoria uniforme en el intervalo [0,1]. Sea *m* un número entero y sea *b* otro número entero menor que *m*. El valor de *b* suele ser cercano a la raíz cuadrada de *m*. Diferentes valores de *m* y *b* producen generadores de números pseudoaleatorios de diferente calidad. Existen varios criterios para seleccionar buenos parámetros para *m* y *b*, pero la mejor regla para contrastar la calidad de la generación es analizar los resultados. Para comenzar, se elige un numero entero,x_0, entre los valores 1 y m. x_0 es la semilla de la generación. Una vez fijada la semilla, el generador actúa del modo siguiente:

```
X1 = b x0 (resto m)
U1 = x1/m
```

U_1 es el primer número psuedoaleatorio que toma algún valor entre 0 y 1. El siguiente número pseudoaleatorio se obtiene del mismo modo:

```
X2 = b x1 (resto m)
U2 = x/m
```

U_2 es otro número pseudoaleatorio. Si *m* y *b* han sido bien seleccionados será difícil predecir el valor de U_2, dado U_1. Es decir, U_2 es aproximadamente independiente de U_1. El método continúa de acuerdo a las siguientes formulas:

$$x_n = bx_{n-1}$$
$$u_n = x_n / m$$

Este procedimiento genera números que son deterministas, pero para un observador que no conoce la formula generadora de los números, la secuencia producida aparece como aleatoria e impredecible. En el siguiente ejemplo, tomado de Braun y Murdoch (2007) se muestra su aplicación:

```
M = 7; b = 3; x0 = 3
x1 = 3 ×2 (resto 7) = 6; u1 = 0,857
x2 = 3 ×6 (resto 7) = 4; u2 = 0,571
x3 = 3 ×4 (resto 7) = 5; u3 = 0,714
x4 = 3 ×5 (resto 7) = 1; u4 = 0,143
x5 = 3 ×1 (resto 7) = 3; u5 = 0,429
x6 = 3 ×3 (resto 7) = 2; u6 = 0,286
```

En este ejemplo, el séptimo numero de la secuencia (x_7) es igual al primero (x_1); es decir, la secuencia es cíclica. Es difícil predecir U_2 a partir de U_1, pero como$U_{i+6} = U_i$ para todo $i > 0$ los valores son pronosticables. Como consecuencia, se deduce fácilmente que los ciclos deben de ser lo suficientemente largos como para que los valores no se repitan. Dado que la longitud del ciclo no puede ser mayor que `m`, conviene dotar a `m` de un valor alto.

6.6.1. *Distribuciones de variables aleatorias*

R genera muestras pseudoaleatorias de datos a partir de una gran variedad de funciones de probabilidad, algunas de las cuales se muestran en la siguiente tabla.

R dispone de funciones específicas para cada una de las distribuciones, que devuelven la densidad de probabilidad, la función de distribución, los valores cuantiles o genera valores pseudoaleatorios para cada una de ellas.

Distribución	Nombre R
Beta	`beta`
Binomial	`binom`
Cauchy	`cauchy`
Chi cuadrado	`chisq`
Exponencial	`exp`
F	`F`
Gamma	`gamma`
Geometrica	`geom.`
Hipergeométrica	`hyper`
Log-normal	`lnorm`
Logistica	`logis`
Binomial negativa	`nbinom`
Normal	`norm`
Poisson	`pois`
T	`T`
Uniform	`unif`

Tabla 6.2. Distribuciones de probabilidad

6.6.1.a. *Variable aleatoria de distribución normal*

Las distribuciones estadísticas con más interés desde el punto de vista estadístico-matemático son las distribuciones normales. No sólo porque existen muchos fenómenos de la vida cotidiana que se distribuyen siguiendo el modelo normal, sino también porque la aplicación de muchas de las técnicas estadísticas más utilizadas, se fundamenta en que las variables implicadas siguen este modelo.

Una variable aleatoria normal X tiene la siguiente función de densidad:

$$f(x) = \frac{1}{\sigma_X \sqrt{2\Pi}} e^{-\frac{(x-\mu)^2}{2\sigma_X{}^2}}$$

Las funciones relacionadas con la distribución normal son:

- *Densidad* (f(x) = P(X = x)). Refleja la probabilidad para un valor dado (densidad) . `dnorm(x, mean = 0, sd = 1).` Si no se especificaran valores para la media aritmética (`mean`) y la desviación estándar (`sd`), los valores asignados por defecto son (0,1).
- *Distribución acumulada*. (F(x) = Px(X≤x)). Ofrece al área por debajo del valor especificado. `pnorm(q, mean = 0, sd = 1, lower.tail = TRUE). ).` Si no se especificaran valores para la media aritmética (`mean`) y la desviación estándar (`sd`) los valores asignados por defecto son (0,1). Si se desea obtener el área sobre el valor habrá que indicarlo, "`lower.tail = FALSE`".
- *Cuantiles*. (P(X< = x_alpha) = alpha). Es la opuesta a la función acumulada. `qnorm(p, mean = 0, sd = 1, lower.tail = TRUE).`
- Generación de valores psuedoaleatorios `rnorm(n, mean = 0, sd = 1).` Una vez definido el número de valores a generar (`n`), se especificaran la media aritmética (`mean`) y la desviación estándar (`sd`) de la distribución de origen. Si no se especificaran valores para la media aritmética (`mean`) y la desviación estándar (`sd`), los valores asignados por defecto son (0,1).

```
> pnorm (1.96)
[1] 0.9750021
> pnorm (-1.96)
[1] 0.02499790
> qnorm (0.975)
[1] 1.959964
> rnorm (10, 0,1)#genera 10 elementos de la distribución
N(0,1)
[1] 0.59730804 0.58565221 0.83386072 1.57668353 -0.47701595
-0.32683712
[7] -1.66099073 -0.44461203 0.02053587 -1.81053407
> a <- rnorm (5, 10, 3)# 5 elementos N(10,3) asignados al
vector a
> a
[1] 7.534002 11.755060 10.332604 7.217336 9.362176
```

Si se desea generar una serie aleatoria de números que se distribuyan según la ley normal, y después representar su distribución por medio de un histograma el comando a emplear sería el siguiente (más adelante se explicarán ésta y otras funciones gráficas).

```
hist(rnorm(100), col=3, xlim=c(-3,3),main="Ejemplo
histograma N=100",xlab="", ylab="frecuencia")
```

Se ha ejecutado este comando en tres ocasiones diferentes variando el número de elementos generados. La sucesión de histogramas muestra la aproximación a la distribución normal de la muestra a medida que aumenta el número de elementos generados (N=100, N=1000; N=10000).

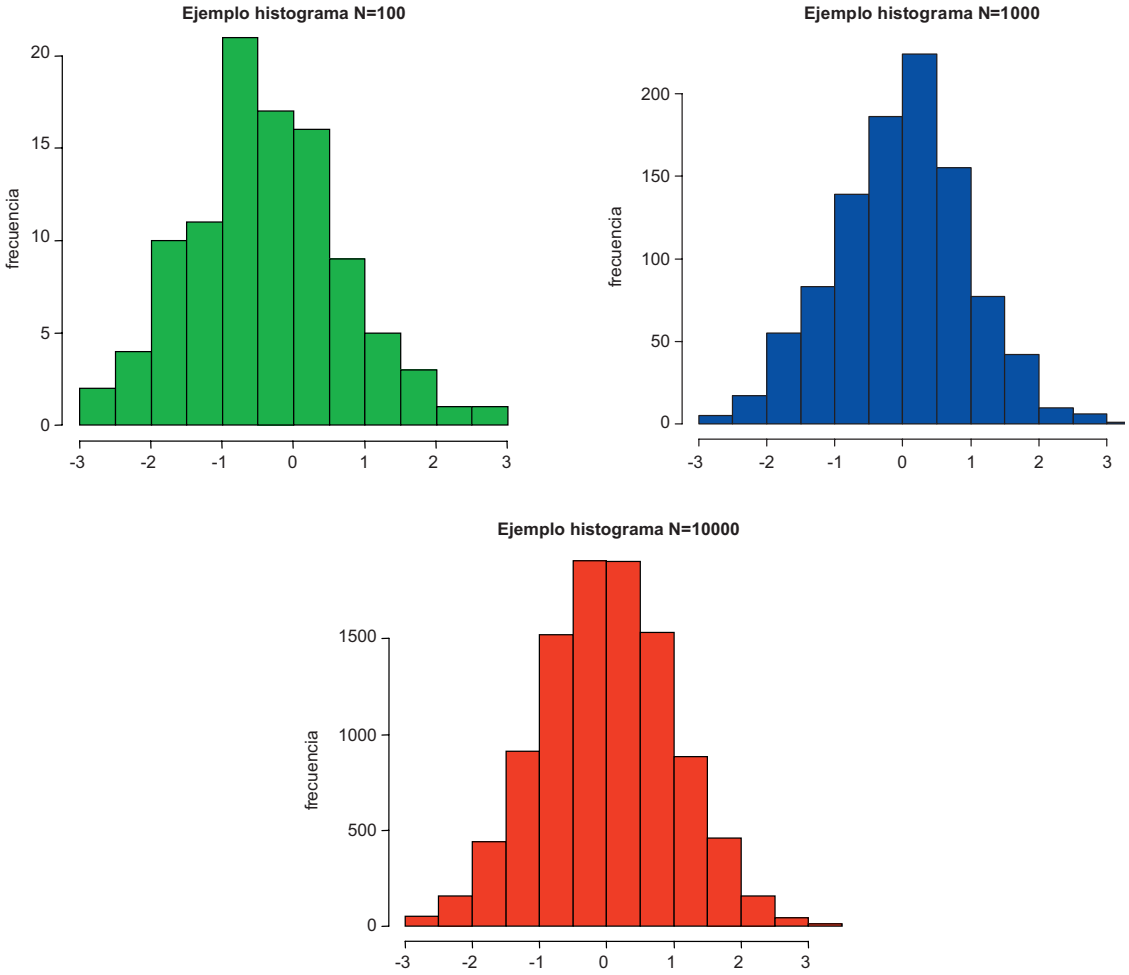

Figura 6.2. Generación y representación de variables

6.6.1.b. *Variable aleatoria de distribución uniforme*

La función `runif()`, genera `n` números pseudoaleatorios que siguen una distribución uniforme en el intervalo [a,b]. Los valores por defecto son a = 0, b = 1. La semilla se elige internamente.

```
runif(n = 5) ##generación a partir de una distribución
uniforme
[1] 0.6840801 0.4979612 0.7414479 0.1982571 0.6043425
> b <- runif(6,4,5)# el resultado se almacena en b
> b
[1] 4.438509 4.901914 4.949303 4.905108 4.724932
4.496114
```

6.6.1.c. *Variables de Bernoulli*

Un experimento de Bernoulli es aquel en el que solo son posibles dos resultados. Cada resultado tiene una probabilidad dada, y la suma de ambas ha de ser 1.

```
> azar <- runif (20)
> correctas <- (azar < 0.6)
> correctas
[1] FALSE TRUE TRUE TRUE TRUE FALSE TRUE TRUE TRUE FALSE
FALSE FALSE
[13] TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
```

6.6.1.d. *Variables aleatorias binomiales*

Sea `X` la suma de `m` variables de Bernoulli independientes, cada una de ellas con probabilidad `p`. `X` es una variables aleatoria binomial, y representa el número de éxitos en un experimento de Bernoulli. Una variable aleatoria binomial puede tomar valores en el intervalo (0,1,2,…m). La probabilidad de que una variable binomial adopte un valor x viene dada por:

$$P(X=x)=\binom{m}{x}p^{x}(1-p)^{m-x},x=0,1,2,..m$$

Estas probabilidades pueden computarse con la función `dbinom()`.

```
> rbinom(25,10,0.3)
[1] 1 2 2 0 2 3 3 4 3 1 4 5 5 2 3 3 2 2 3 2 2 2 1 3 2
> pbinom (4,6,0.4)
[1] 0.95904
> dbinom (x=4, size=8, prob=0.3)
```

6.6.2. *Extracciones con* `sample()`

La extracción de muestras a partir de un vector `x` se ejecuta con la función `sample()`, que genera una secuencia aleatoria con o sin reemplazamiento de una longitud

determinada. Sin más especificaciones, `sample` produce una permutación aleatoria de los elementos de `x.` Por defecto el muestreo se lleva a cabo sin reemplazamiento y el tamaño de la muestra es igual al tamaño del objeto del que ésta es extraída.

```
> x <- 1:10
> sample(x)
[1] 10 9 8 3 7 2 6 1 5 4
> sample(x, 3) #extrae una muestra de tamaño 3
[1] 8 1 9
> sample (x,4, replace = TRUE)
[1] 4 7 10 7
>sample(a, 2*length(x), replace = TRUE)
[1] 3 1 8 9 10 9 9 3 1 5 6 10 9 8 2 3 1 1 6 3
```

7. GRÁFICOS

La capacidad gráfica de R convierte a este entorno en uno de los software gráficos por excelencia. La versatilidad de R en la producción de gráficos puede comprobarse tecleando en la consola la orden "`demo(graphics)`" o "`demo(persp)`" . R dispone de varios sistemas gráficos; el más antiguo de los cuales, conocido como gráficos base (`base graphics`) sería equiparablele al sistema `S graphics`. Se trata de gráficos "estándar" que se construyen incorporando a una función general o de alto nivel, parámetros gráficos y elementos (puntos, texto, ecuación, color..) que mejoran la apariencia del mismo y lo adaptan a contextos particulares. R incorpora también un nuevo sistema gráfico recogido en el paquete `grid`, que se considera una alternativa al paquete base. La forma de construcción gráfica es diferente; el usuario define regiones rectangulares arbitrarias (`viewports`) sobre el dispositivo gráfico y delimita un número de sistemas de coordenadas para cada región. Por último los paquetes `lattice` y `gglplot` (`ggplot2`) permiten diseñar gráficos de panel (`grid graphics`) especialmente útiles en el trabajo de modelos multivariables.

7.1. Dispositivos gráficos

Los gráficos, a diferencia del resto de estructuras que R utiliza, no son almacenados como objetos. Son enviados a dispositivos gráficos (`graphical devices`) que define el usuario; pueden actuar de dispositivos gráficos la pantalla o un archivo de formato predefinido.

Cuando se ejecuta un comando gráfico, R por defecto abre una nueva ventana en la cual se visualizará el resultado. Esta ventana se constituye por defecto en el dispositivo gráfico. Es posible mantener abiertos varios dispositivos gráficos simultáneamente, de los cuales el último dispositivo abierto se convierte en el dispositivo en el que se almacenarán los gráficos. Para obtener una lista de los dispositivos abiertos se puede utilizar la función `dev.list`(). La función `dev.cur`() devuelve el dispositivo activo, y si se desea fijar un dispositivo concreto el comando es `dev.set`(). La función `dev.off`() cierra un dispositivo.

Funciones	Formatos gráficos
Dispositivos pantalla	

`x11()` `edo X11`
`windows()`
`quartz()`
	Ventana x de Windows
	Microsoft Windows
	Mac OS X Quartz

Archivos	

`postscript()`
`pdf()`
`pictex()`
`xfig()`
`bitmap()`
`png()`
`jpeg()`
	Adobe postsript
	Adobe PDF
	Latex Pictex
	XFIG
	GhostScript
	PNG
	JPEG

Sólo para Windows	
`win.metafile()` `bmp()`	Windows Metafie Windows BMF

Tabla 7.1. Dispositivos gráficos.

Cuando se utiliza la pantalla como dispositivo gráfico, se puede acceder a una ventana que permite gestionar de alguna manera los gráficos generados. Por ejemplo si se crea un histograma y se pulsa sobre él, se accede a una ventana con una barra de herramientas en la cual existen varias opciones,

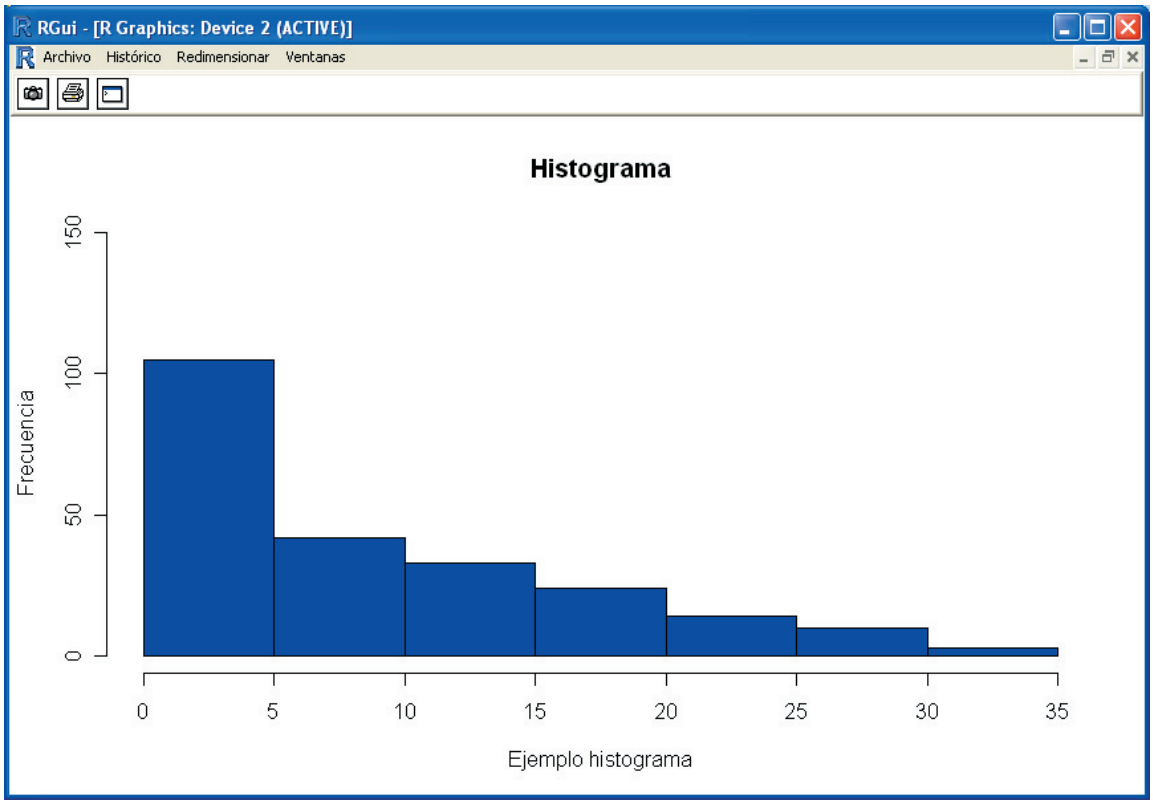

Figura 7.1. Gestión básica de gráficos

La opción `Archivo` permite guardar, copiar o imprimir la imagen. Dentro de la opción `Guardar` es posible seleccionar entre varios formatos gráficos, Metafile, Postscript, PDF, Pnd, BMP y JEPG. Las opciones que aparecen en la barra de herramientas bajo el nombre genérico de `Histórico` facilitan el almacenamiento secuencial de los gráficos.

7.2. **Estructura básica**

El procedimiento de construcción de gráficos combina dos tipos de funciones gráficas; las funciones gráficas de alto nivel, y las funciones gráficas de bajo nivel. Las primeras son funciones generales que construyen gráficos, y las segundas son funciones que permiten añadir elementos a los gráficos generados.

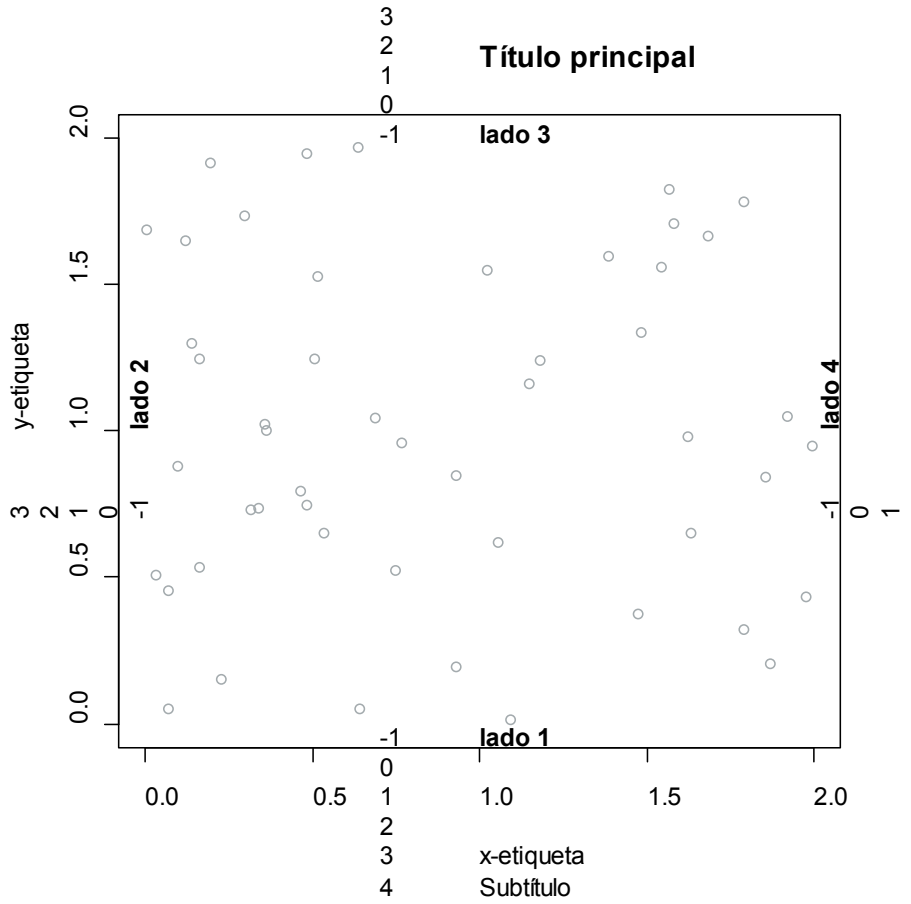

Figura 7.2. Estructura básica de un gráfico

Los gráficos construidos por R están divididos en regiones. La región en la que se dibuja la figura gráfica es la región principal, y a su alrededor quedan definidos los márgenes del gráfico. Las coordenadas de la región principal se especifican en las mimas unidades de medida que el gráfico (las definidas en los ejes), y las coordenadas de los márgenes se especifican en líneas de texto cuando se refieren a puntos perpendiculares a la región principal, y en unidades de medida de los datos si el movimiento es en el sentido del área principal.

La región principal es el núcleo del gráfico y puede modificarse añadiendo elementos tales como puntos, líneas, texto, polígonos…En un gráfico estándar las coordenada X e Y está acompañadas de títulos que se escriben en los márgenes de la imagen. Es sencillo modificarlos o adaptarlos.

7.3. **Funciones gráficas**

El objetivo de las funciones gráficas es la generación de gráficos completos a partir de datos que forman parte del argumento de la función. Entre las funciones gráficas genéricas las más utilizada es `plot()`. Basta especificar los datos, que podrán ser factores o vectores, para generar por defecto, gráficas como las mostradas en la siguiente ilustración.

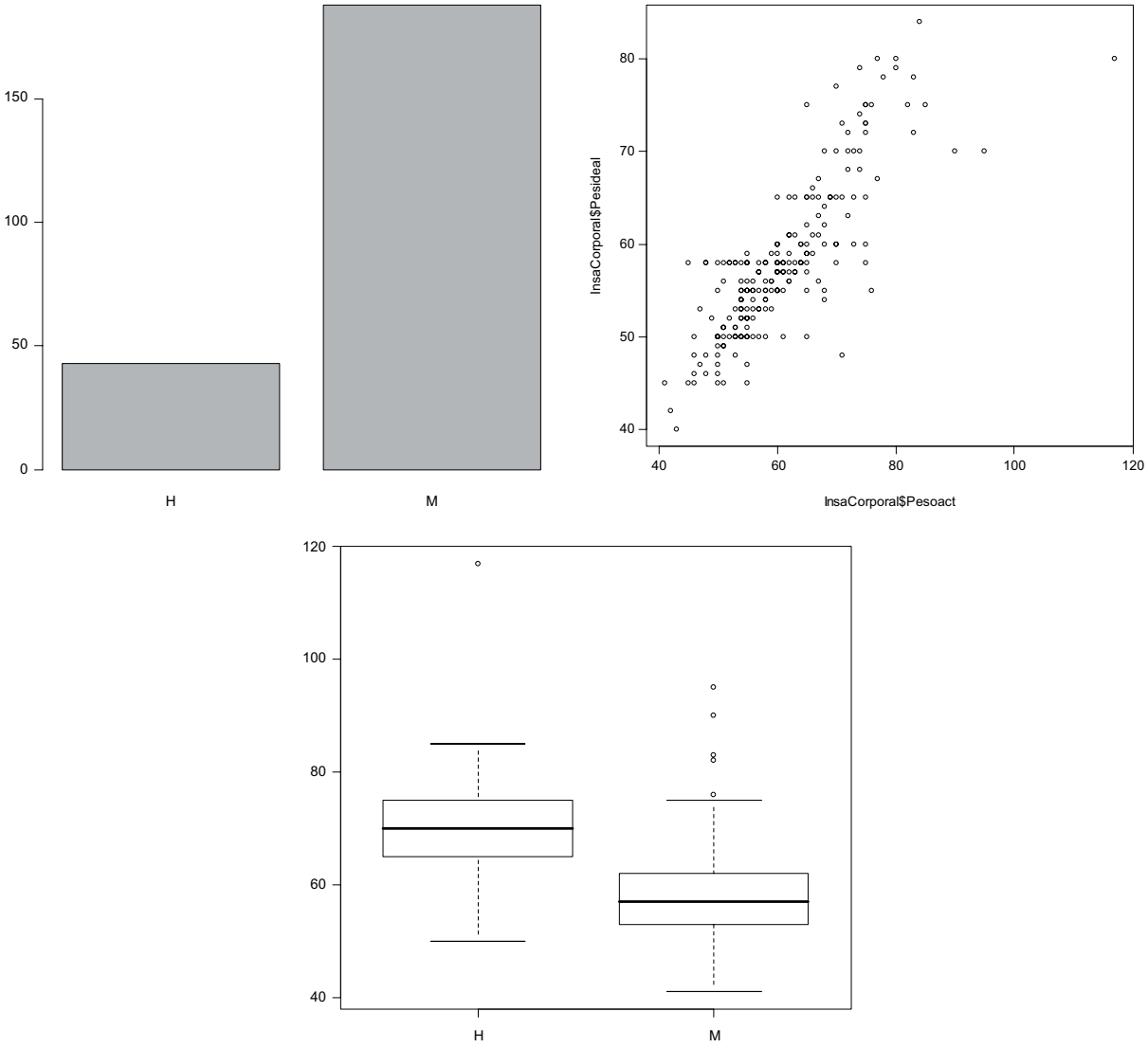

Figura 7.3. Ejemplos simples de la función plot

Los tres gráficos han sido generadas con `plot`, definiendo como argumento en cada uno de los casos, una variable discreta o factor (diagrama de barras), dos variables continuas (gráfico de dispersión), y una variable categórica/una variable continua (grá-

fico de cajas). Además de `plot()`, R ofrece varias funciones genéricas, algunas de las cuales se muestran en la siguiente tabla:

Función	Tipo de gráfico
`plot()`	Barras, dispersión, cajas
`hist()`	Histograma
`stem.leaf()`	Tallo y hojas
`boxplot()`	Gráfico de cajas
`qqplot()`	Gráfico QQ de cuantlies
`scatterplot`	Diagrama de dispersión
`pie()`	Gráfico de sectores
`dotchart(x)`	Gráfico Cleveland

Tabla 7.2. Funciones gráficas de alto nivel

Las funciones gráficas de alto nivel fijan por defecto los valores correspondiente al formato de los ejes, las escalas, los títulos del gráfico o los símbolos gráficos a utilizar. Esas características pueden modificarse por medio de los argumentos gráficos de las funciones. La mayoría de las funciones gráficas disponen de los siguientes argumentos cuyos efectos pueden leerse en la tabla:

Argumentos gráficos	Efecto
`type` = ""	Especifica el tipo de gráfico: "p" puntos "l" líneas "b" puntos y líneas "n" No gráfico "o" puntos y líneas "h" líneas verticales
`axes` =	F elimina los ejes T dibuja los ejes (defecto)
`xlab` = "texto"	Añade texto al eje
`ylab` = "texto"	
`main` = "texto"	Título del gráfico
`sub` = "texto"	Subtítulo bajo el eje X
`xlim` = c(mínmo, máximo)	Escalas de los ejes X e Y
`ylim` = c(mínimo, máximo)	

Tabla 7.3. Argumentos de las funciones gráficas

Además de las funciones genéricas, R aporta una serie de funciones conocidas como "funciones gráficas de bajo nivel" que añaden información al gráfico además de controlar su formato. Entre las funciones de bajo nivel, tal vez las más utilizadas sean:

Funciones de bajo nivel	Efecto
`points (x,y,...)`	Añade puntos a la función gráfica (plot) inmediatamente anterior. El tipo de punto se fija con pch = y su tamaño por cex = o mkh =
`lines(x,y,...)`	Añade líneas a la función gráfica (plot) inmediatemetne anterior. El tipo de linea se fija con lty = y su tamaño por lwd =
`text(x,y,labels,...)`	Añade texto a un gráfico en los puntos de coordenadas x,y
`abline(a,b,h = ,v = ,..)`	Añade una recta con intercepto (a) y pendiente (b). h = fija el valor de y para las líneas horizontales. v = fija el valor de X para las líneas verticales
`legend (x,y, texto...)`	Añade al gráfico una leyenda en la posición especificada. Por defecto las leyendas se imprimen dentro de un rectángulo.
`title(main = "texto", sub = "texto", xlab = "texto", ylab = "texto", line = NA, outer = FALSE, ...)`	Añade texto a un gráfico. Un título principal que por defecto aparecerá en la parte superior, un subtitulo (bajo el eje X), y etiquetas para los ejes X e Y.
`axis (side,labels, tick...`	side = 1 eje inferior; 2 = eje izquierdo; 3 = eje superior; 4 = eje derecho. Sus argumentos permiten controlar las marcas de división (tick) y las etiquetas de división (lab)
`segments(x0,y0,x1,y1)`	Dibuja segemenos en las coordenadas dadas
`polygon(x,y…)`	Añade polígonos cerrados
`arrows(x0,y0,x1,y1)`	Añade flechas
`symbols (x,y,…)`	Dibuja círculos, cuadrados,….

Tabla 7.4. Funciones gráficas de bajo nivel

7.4. **Parámetros gráficos**

Al mismo tiempo que se especifican las características de cada uno de los gráficos, es posible definir un conjunto de parámetros que controlarán su apariencia. Los parámetros gráficos pueden definirse bien de modo permanente, o bien de modo temporal. Para conocer los valores asumidos por un dispositivo gráfico basta teclear la función `par()`. Los parámetros pueden incluirse (la mayoría de ellos) como argumentos de las funciones gráficas.

La función `par()` puede parecer al principio un poco confusa, y hace falta algo de práctica para familiarizarse con ella. "`par`" permite fijar el ancho, y el tipo de línea, el tipo y tamaño de los carácteres gráficos, el estilo de los ejes, el tamaño de la áreas gráficas, los márgenes…-

En el siguiente ejemplo mostramos una aplicación de "par" en la determinación de los márgenes de un gráfico. Las áreas comprendidas entre la caja azul y la caja roja son los márgenes del gráfico, que vienen controladas por `mar.` Para comprobar los márgenes que afectan a un gráfico la función es `par()$mar`,

```
> par()$mar
[1] 5.1 4.1 4.1 2.1
```

En el gráfico siguiente el margen inferior ocupa aproximadamente 5 líneas, y el margen izquierdo 4 líneas, igual que el superior. En el margen derecho se han especificado 2 líneas. Para definir los márgenes externos de las figuras gráficas (`outer margin area`),el parámetro a utilizar es `oma`

R permite la generación múltiple de figuras en una misma ventana o en su defecto, en un mismo dispositivo gráfico. Los parámetros que permiten tal función se definen del siguiente modo

```
par(mfrow = c(2,3))
par(mfcol = c(2,3))
```

Ambos dividen la pantalla en 6 regiones; el primero define 2 filas por 3 columnas, mientras que el segundo (`mfcol`) define 2 columnas por 3 filas. Los gráficos generados se van añadiendo a los anteriores bien por filas (`mfrow`) bien por columnas (`mfcol`).

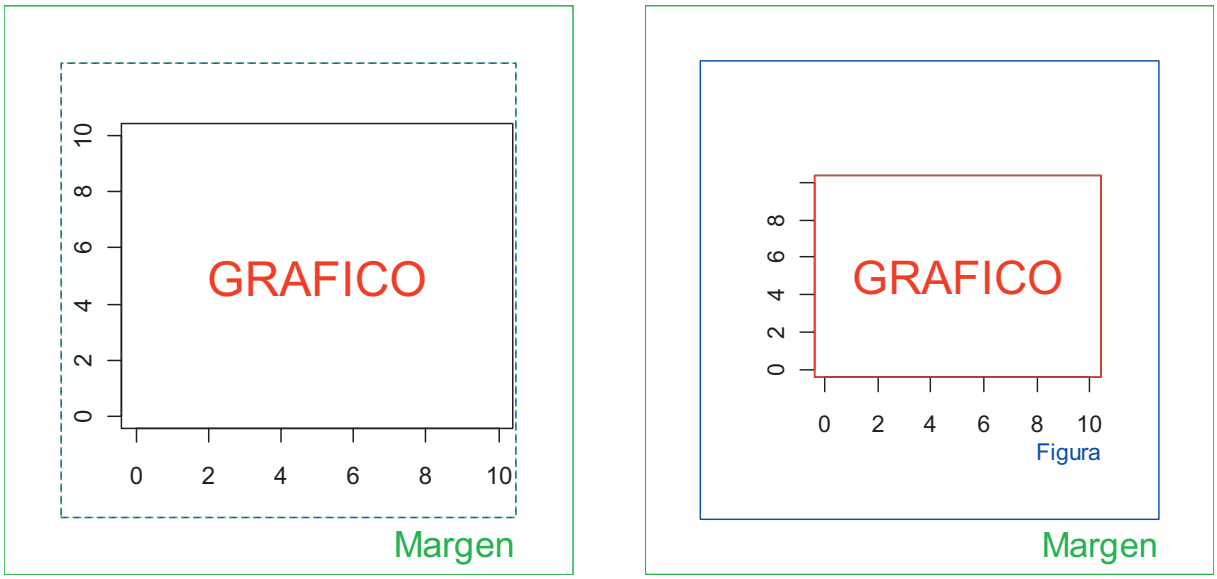

Figura 7.4. Parámetros gráficos `mar` y `oma`

Parámetros gráficos	Efecto
`mfrow = c(m,n)`	Permite dibujar `m` filas y `n` columnas de gráficos
`mfg = c(i,j)`	Especifica las coordenadas en las que se dibujará el gráfico
`ask = TRUE`	Se pregunta al usuario antes de borrar un gráfico para dibujar uno nuevo
`mar = c(s1,s2,s3,s4)`	Fija los márgenes del `plot`
`oma = c(s1,s2,s3,s4)`	Fija los márgenes externos

Tabla 7.5. Parámetros gráficos

7.4.1. *Caracteres y elementos gráficos*

Los puntos, líneas y texto que se quieren incluir en un gráfico pueden controlarse por medio de parámetros que afectan a un conjunto de figuras, o bien, pueden incluirse como argumentos de la funciones gráficas.

Elementos gráficos	Efecto
`pch = " "`	Define el tipo de carácter. El usuario puede seleccionar entre 19 caracteres diferentes a cada uno de los cuales le corresponde un indicador numérico entre 0 y 18
`cex =`	Especifica el tamaño de los caracteres. Recibe el nombre de carácter de expansión.
`mkh =`	Especifica el tamaño de las marcas de los ejes
`lth =`	Tipo de linea, entre 1 (linea continua) y 6.
`lwd =`	Define la anchura de la línea y afecta tanto a las lineas de los ejes como a las lineas creadas con lines.
`font =`	Controla la fuente (1, texto normal, 2 negrilla, 3 cursiva, 4 = cursiva y negrilla.

Tabla 7.6. Parámetros para los elementos gráficos

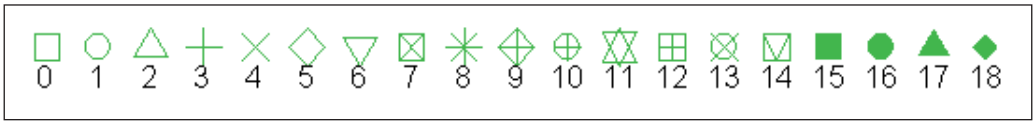

Figura 7.5. Caracteres gráficos

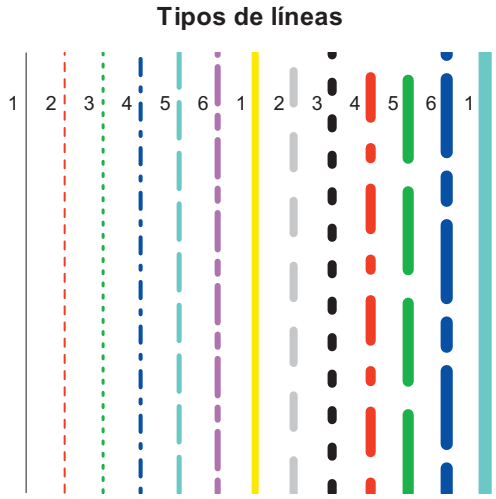

Figura 7.6. Tipos de líneas

```
plot(x <- c(1:13), y <- c(1:13), type = "n", xlab = "", ylab
= "", axes = FALSE)
for (i in 1:13) (abline (v = i, lty = i, lwd = i, col = i) )
text (x,12, labels = rep(1:6,2), pos = 2)
title (main = "Tipos de lineas")
```

7.4.2. *Uso del color*

R presenta una amplia paleta de colores, cuyos valores vienen controlados por el parámetro `col =` . Pueden definirse de forma numérica (cada color tiene asignado un número) o de forma alfanumérica (por su nombre). Por ejemplo `colors()` genera los nombre de los 657 colores disponibles en R, si se extrae una muestra aleatoria de 10 colores se obtendría algo similar a lo siguiente.

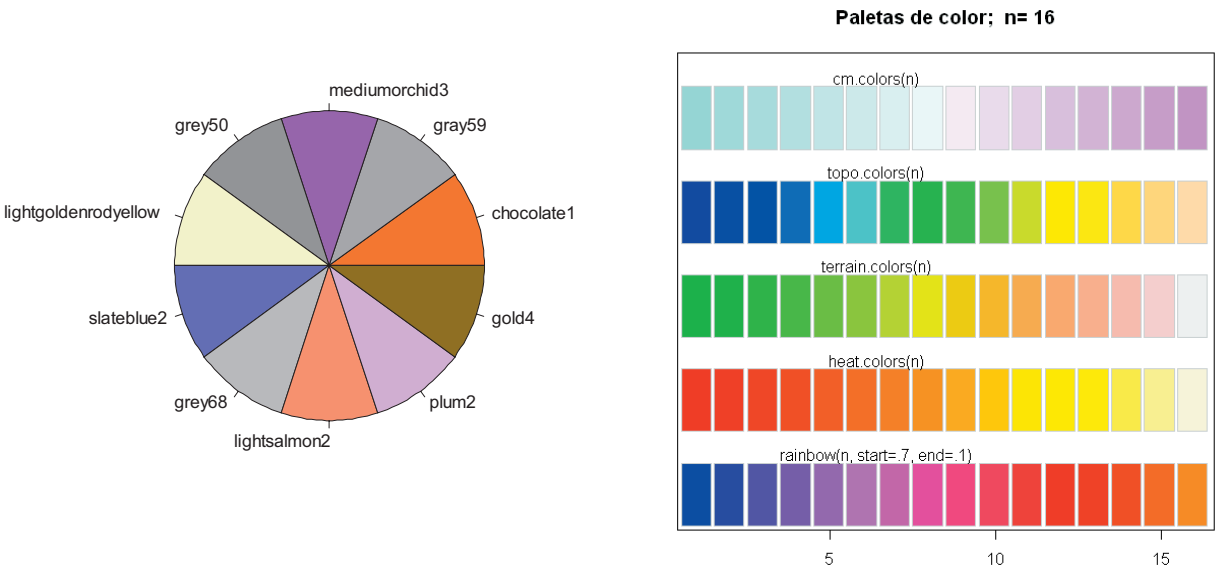

Figura 7.7. Paleta de colores

```
a <- sample(colors(), 10)
> pie (rep(1,10), col = a, labels = a)"
```

Los tonos grises se controlan con la función `gray(level)` crea un vector de colores en la gama de grises. Por ejemplo,

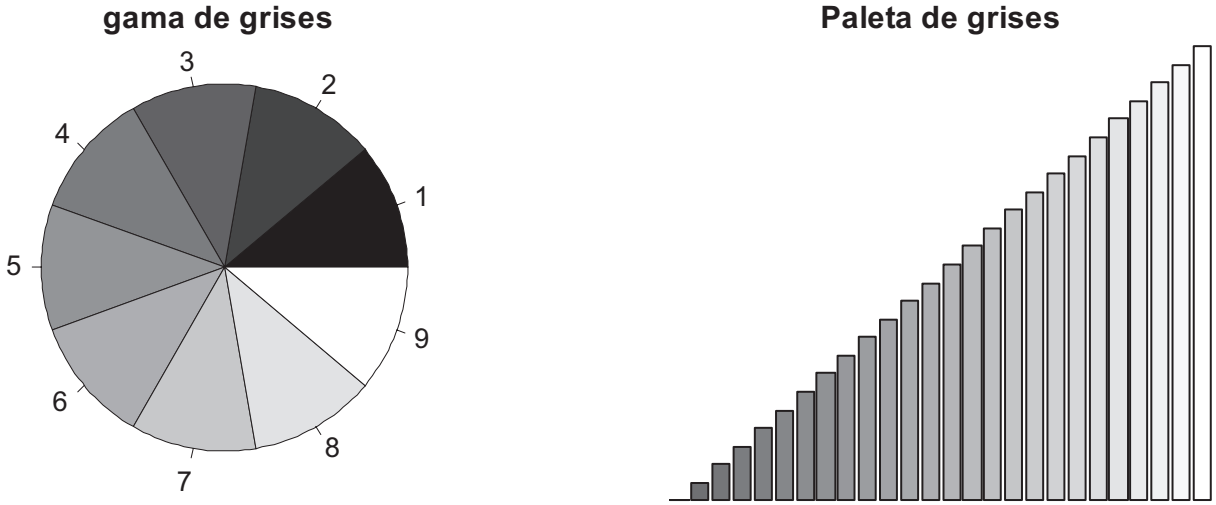

Figura 7.8. Gama de grises

```
a <- gray (0:8/8)
> pie(rep (1,9), col = a)
```

7.5. **Ejemplos. Creando gráficos simples**

En este apartado intentaremos exponer las pautas a seguir en la construcción de gráficos sencillos; se muestra el gráfico en su formato más básico, y tras haber incluido algunas modificaciones en él. Si quisiéramos representar gráficamente una variable en función de tres rangos de edad, una de las opciones más adecuadas sería el gráfico de cajas, porque estaríamos representando una variable continua y una variable discreta. . Utilizando la función `boxplot()` y los argumentos definidos por defecto para ella, se obtendría lo siguiente:

```
boxplot(Datos~Edad, ylab = "IC", xlab = "EDAD_RE", data
= Data)
```

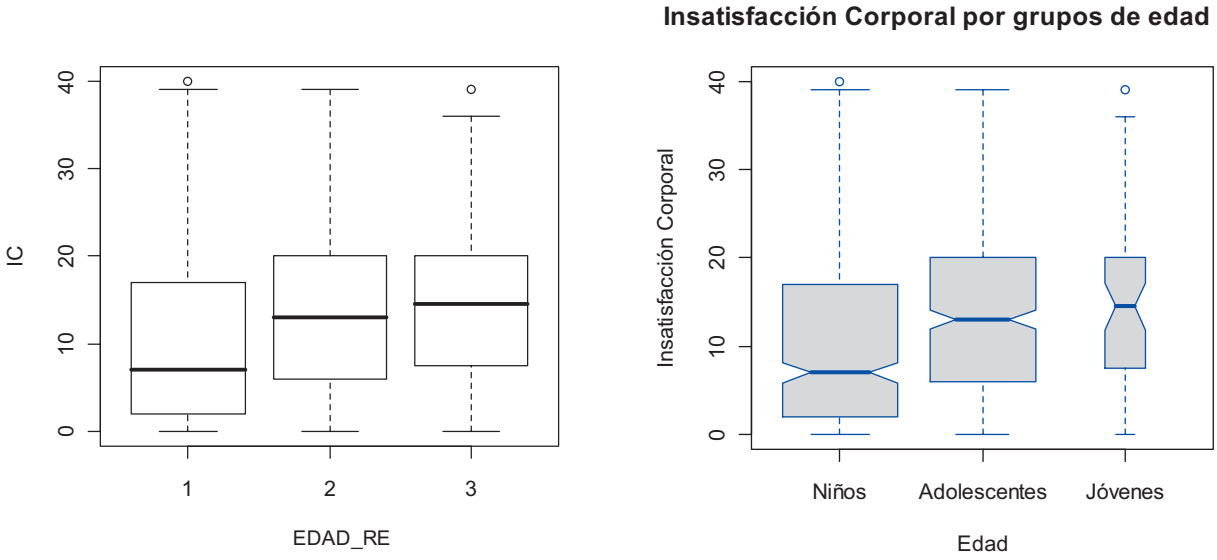

Figura 7.9. Diagrama de cajas

Este gráfico condensa mucha información, pero su apariencia no es elegante. Podría mejorarse incluyendo varios elementos gráficos; por ejemplo,

- Renombrando los ejes
- Añadiendo color
- Añadiendo bisagras en el punto de la mediana
- Adecuando el tamaño de la caja al tamaño de los grupos de edad

```
boxplot(DatosEdad, ylab = "Insatisfacción Corporal",
     names = c("Niños", "Adolescentes", "Jóvenes"),xlab
= "Edad", data = Data, varwidth = TRUE,boxfill =
"lightgrey",
     border = "blue",notch = TRUE, font = 4)
title(main = "Insatisfacción Corporal por grupos de
edad")
```

En el siguiente ejemplo mostramos la construcción de un diagrama a partir de los datos de una tabla. Se desea representar la distribución de las puntuaciones en una variable que tiene 5 posibles valores (por ejemplo, una respuesta en un cuestionario), en función del Sexo. Se trata de la representación de dos variables categóricas. El gráfico representará las respuestas a cada una de las categorías en una misma área gráfica. La figura de la izquierda surge de la aplicación directa de las opciones por defecto incluidas en la función `barplot()`. Se dibujan dos barras y en cada una de ellas se representan de modo apilado las frecuencias de respuesta en cada una de las opciones. La segunda imagen, construida con la misma función de alto nivel, es más clara. Cada una de las opciones de respuesta se representa por una barra, y se han añadido elementos gráficos como etiquetas, títulos y algo de color.

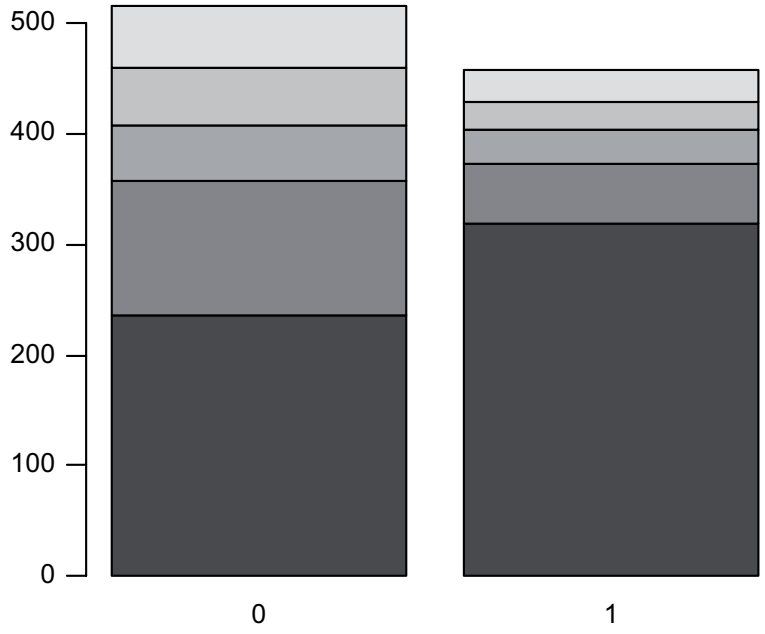

Figura 7.10. Diagrama de barras

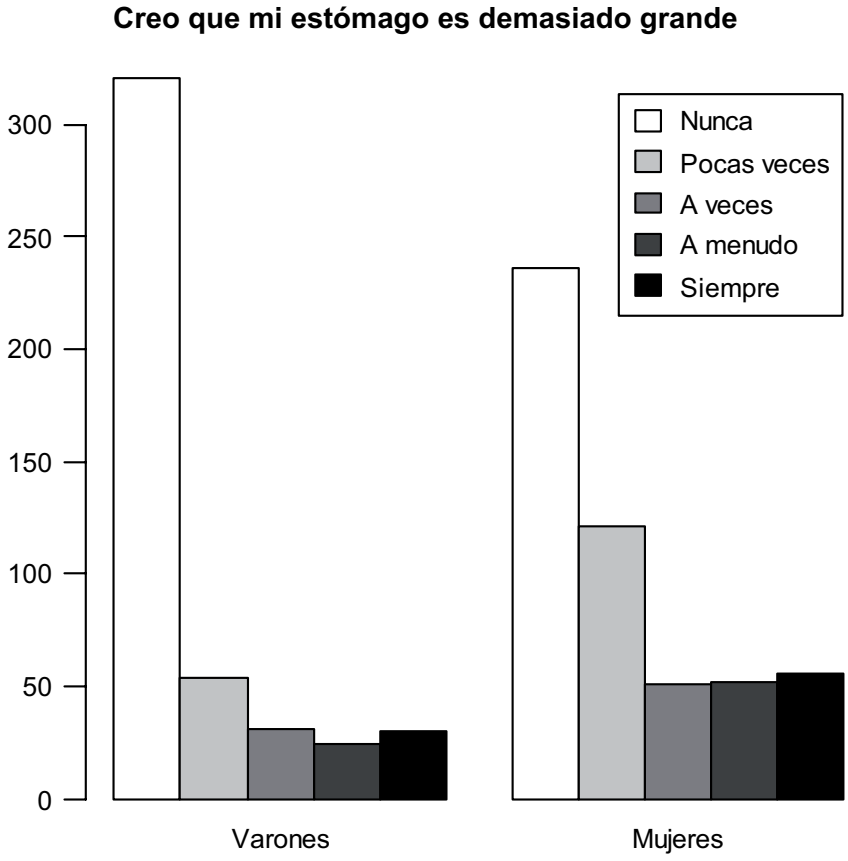

Figura 7.11. Diagrama de barras

Como tercer ejemplo representamos una variable continua. Si el objetivo fuera dibujar un histograma se podría utilizar la función de alto nivel `Hist()`:

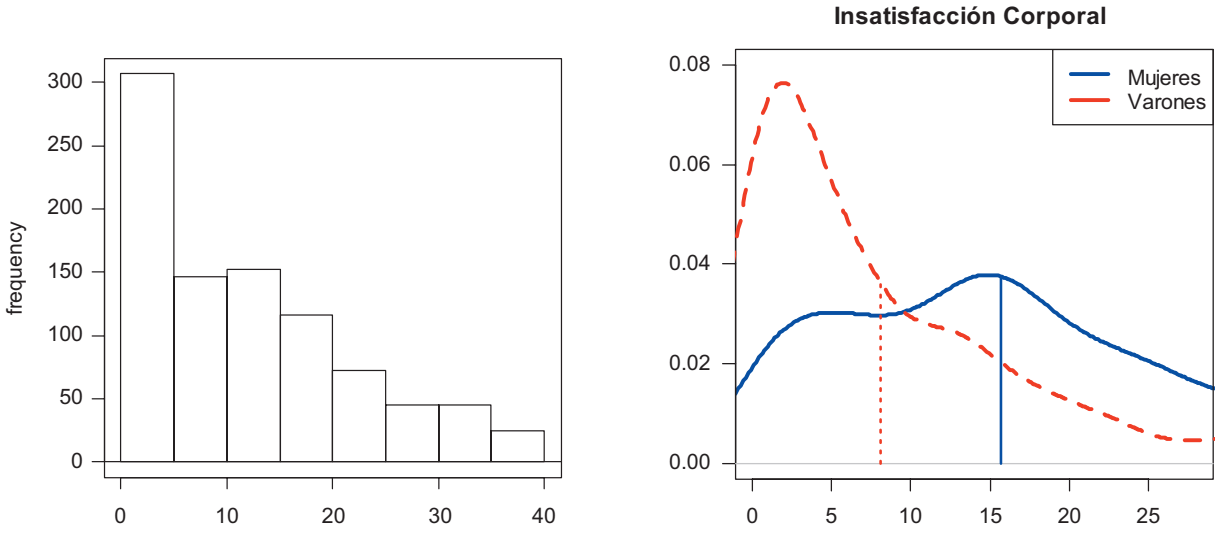

Figura 7.12. Histograma y gráfico de densidad

```
Hist(Datos$IC, scale = "density", breaks = "Sturges",
col = "darkgray")
```

El histograma es tal vez uno de los gráficos más utilizados en la investigación aplicada para la representación de variables continuas; sin embargo, a pesar de su extendido uso, presenta problemas relacionados sobre todo con el efecto de la selección de distintos intervalos de puntuación en el perfil gráfico. Este hecho hace que la interpretación de un histograma pueda resultar "engañosa". Un modo de evitar los problemas que aquejan a la utilización de histogramas para la representación de variables continuas, es la utilización de gráficos de densidad. En este ejemplo se ha representado el gráfico de densidad de la misma variable representada por el histograma. Dado el carácter diferencial de esta variable en función del sexo, se ha optado por representar los gráficos correspondientes a los dos grupos (varones, mujeres) en el mismo área, y en cada una de ellas se ha representado el punto correspondiente a la media aritmética de cada distribución. La información que aporta la visión conjunta de los gráficos de densidad es más intuitiva y fácil de aprehender, así como sencilla en su interpretación y en su comunicación, que una tabla de valores descriptivos (media aritmética, desviación estándar, asimetría) y los referidos a las comparaciones entre ellas (diferencia de medias, test de significación, igualdad de varianzas) . El gráfico de densidad incluye información sobre la forma y la desviación de la distribución así como sobre las medidas de tendencia central y de las diferencias entre las distribuciones.

7.6. **Gráficos de panel**

Una de las formas más elegantes para la representación gráfica multivariable es el formato gráfico *trellis* desarrollado en el lenguaje S. Un gráfico de rejilla, panel o trellis representa relaciones multivariadas entre variables. En R los gráficos de este tipo vienen implementados en los paquetes `grid` y `lattice`.

Los gráficos trellis no son gráficos estándar; utilizan parámetros gráficos diferentes y funciones adaptadas de los gráficos base. Pueden generar representaciones unidimensionales, bidimensionales o tridimensionales, y son un elemento importante en la evaluación de relaciones de dependencia entre variables. Permiten la representación condicional multivariada lo cual les confiere una gran potencia en el análisis de las relaciones entre variables, y en la exploración del ajuste de modelos formales a los datos.

Algunas de las funciones que incorpora el paquete `lattice` se resumen en la siguiente tabla:

Funciones `Lattice`	
`Xyplot`	Gráfico bidimensional
`Bwplot`	Diagrama de cajas
`Stripplot`	Gráfico unidimesnional para una variable continua y un factor
`Dotplot`	Gráfico de puntos
`Histogram`	Histograma
`Densityplot`	Gráfico de densidad
`Barchart`	Gráfico de barras
`Piechart`	Gráfico de sectores
`Splom`	Matrices gráficas

Tabla 7.7. Funciones `Lattice`

En la figura siguiente se muestra un gráfico realizado con la función `bwplot`.

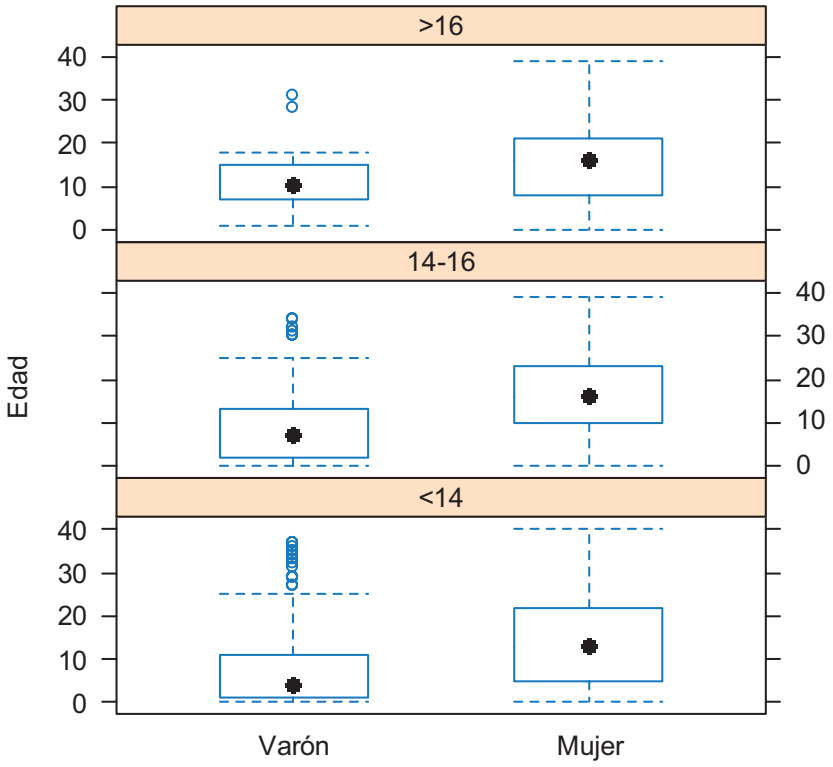

Figura 7.13. Gráfico Trellis

8. **REFERENCIAS BIBLIOGRÁFICAS**

ARRIAZA, A.J., FERNÁNDEZ, F., LÓPEZ, M.A., MUÑOZ, M., PÉREZ, S. y SÁNCHEZ, A. (2008). Estadística básica con R y R-Commander. Servicio de publicaciones de la Universidad de Cádiz.

BECKER, R. A., CHAMBERS, J. M. y WILKS, A. R. (1988). *The new S language: A programming environment for data analysis and graphics*. Pacific Grove, CA: Wadsworth.

CHAMBERS, J.M. (1998). *Programming with data: a guide to the S language*.

——— (2007). *Software for data analysis: Programming with R*. New York: Springer.

CHAMBERS, J. y HASTIE, T. (1992). *Statistical models in S*. Pacific Grove, CA: Wadsworth y Brooks/Cole.

CRAWLEY, M. J. (2008). *The R book*. Chichester: John Wiley y Sons.

DELGAARD, P. (2002). *Introductory statistics with R*. New York: Springer.

ELOSUA, P. (2009). ¿Existe vida más allá del SPSS? Descubre R. *Psicothema*, 21(4), 652-655.

——— (2010). Eskalen eta datuen analisia R eta Rcommnader-ekin. Bilbao: UPV.

ELOSUA, P. y ETXEBERRIA, J. (2011). Análisis de datos con R Commander. Madrid: Muralla.

FOX, J. (2002). *An R and S-plus companion to applied regression*. Thousand Oaks, CA: Sage.

——— (2005). The `Rcommander`: A Basic-Statistics Graphical User Interface to R. *Journal of Statistical Software*, 14 (9), 1-42.

——— (2007). *Getting Started With the R Commander*. Dirección URL: http://socserv.mcmaster.ca/jfox/Courses/soc3h6/Getting-Started-with-the-Rcmdr.pdf

——— (2008). Editorial. *RNews*, 8,2,1-2.

IHAKA, R. y GENTLEMAN, R. (1996). R: A Language for Data Analysis and Graphics. *Journal of Computational and Graphical Statistics*, 5, 299–314.

JENSEN, A. R. (1980). *Bias in mental testing*. New York: Free Press.

JENNRICH, R. I. y SAMPSON, P. F. (1966). Rotation for simple loading. *Psychometrika, 31*, 313-323.

KAISER, H. F. (1958). The varimax criterion for analytic rotation in factor analysis. *Psychometrica, 23*, 187-200.

——— (1960). The application of electronic computers to factor analysis. *Educational and Psychological Measurement, 20*, 141-151.

KERNS, G. J. y CHANG, G.A. (2010) *Introduction to Probability and Statistics Using R (IPSUR* Taylor & Francis Publishing.

KERLINGER, F. N. y PEDHAZUR, E. J. (1973). *Multiple Regression in Behavioral Research*. Nueva York.

LAWLEY, D. N. (1940). The estimation of factor loadings by the method of maximum likelihood. *Proceedings of the Royal Society of Edinburgh, 60*, 64-82.

——— (1943). On problems connectes with item selectgion and test construction. *Proceedings of the Royal Society of Edinburg, 61*, 273-287.

LE, S., JOSSE, J. y HUSSON, F. (2008). FactoMineR: An R Package for Multivariate Analysis. Journal of Statistical Software, 25(1). [http://www.jstatsoft.org/v25/i01/paper]

LEEW, J. y MAIR, P. (2007). An introduction to the special volumen on "Psychometrics in R". *Journal of Statistical Software*, 20 (1), 1-5.

MUENCHEN, R. A. (2009). *R for SAS and SPSS users*. New York: Springer.

MURREL, P. (2005). *R Graphics*. Boca Ratón, Florida: Chapman & Hall.

PARADIS, E. (2005). *R for Beginners*. Institut des Sciences de l'Evolution: Montpellier, France.

VENABLES, W. N. y RIPLEY, B. D. (2000). *S programming*, Springer: New York.

——— (2002). *Modern Applied Statistics with S*, Fourth Edition. New York: Springer.

VENABLES, W. N., SMITH, D. M. y THE R DEVELOPMENT CORE TEAM (2007). *An introduction to R*. Vienna, Austria: R foundation for Statistical Computing.