



Métodos Estadísticos con R y R Commander

Versión 3.1, febrero de 2012

Prof. Dr. Antonio José Sáez Castillo

Dpto de Estadística e Investigación

Operativa

Universidad de Jaén

Esta obra está bajo una licencia Reconocimiento-No comercial-Sin obras derivadas 3.0 España de Creative Commons. Para ver una copia de esta licencia, visite <http://creativecommons.org/licenses/by-nc-nd/3.0/es/> o envíe una carta a Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.



Métodos Estadísticos con R y R Commander by Antonio José Sáez-Castillo is licensed under [Creative Commons Reconocimiento-No comercial-Sin obras derivadas 3.0 España License](http://creativecommons.org/licenses/by-nc-nd/3.0/es/).

Métodos Estadísticos
con R y R Commander
Versión 3.1

Prof. Dr. Antonio José Sáez Castillo
Departamento de Estadística e Investigación Operativa
Universidad de Jaén

Febrero de 2012

Prólogo

El presente documento se creó originalmente como *Guión de Prácticas* para la asignatura *Métodos Estadísticos de la Ingeniería*, impartida en distintas especialidades de la entonces Ingeniería Técnica Industrial de la Universidad de Jaén. Sin embargo, dada la escasez de documentación en español acerca del uso de R, he querido redactar un documento más general que sirva como guía introductoria para usuarios principiantes de R.

Es sabido que existen en el mercado distintos paquetes de software estadístico propietario, como SPSS, Statgraphics, Minitab, Statistica, SAS, S-Plus, etc, que cubren todas las necesidades de cualquier usuario de técnicas estadísticas, básicas o avanzadas. Frente a estas opciones, en los últimos años R ha surgido con fuerza como alternativa de software libre en muy distintos ambientes.

En este sentido, antes de tomar la decisión de utilizar R como programa bajo el que desarrollar mis prácticas, he discutido ampliamente (en el buen sentido de la expresión) con compañeros y alumnos al respecto. Como conclusión de esos debates (a veces encendidos), quisiera plasmar dos aspectos que, a mi juicio, avalan la decisión:

- Después de algún tiempo, cada vez que se me cuestiona acerca de *¿por qué usar R?*, he desarrollado una respuesta rápida: *¿y por qué no hacerlo?* Facilita todo lo necesario, luego, como punto de partida, es una solución factible.
- Es software libre, lo que, en primer lugar, es una ventaja desde el punto de vista económico para la Universidad. Además, esto es una ventaja desde el punto de vista económico para nuestro alumnado y para cualquier usuario en general. Supone la posibilidad de acceder al programa desde sus propios equipos fuera del horario de clase y en cualquier momento de su futuro ejercicio profesional. Finalmente, este hecho, el que sea software libre tiene, a nivel personal, unas connotaciones importantes sobre la manera de entender el papel que juega la Universidad en la sociedad. La construcción del conocimiento hoy en día me parece muy difícil de entender sin el acceso libre por parte de toda la comunidad a las herramientas fundamentales que lo favorecen. Con ello no quiero decir que esté en contra de que determinadas empresas

comercialicen paquetes de software estadístico, sino que me tranquiliza que existan alternativas a estos paquetes que permitan decidir en libertad si quiero comprarlo o no.

Probablemente alguien pensará que R debe tener alguna desventaja en comparación con los programas estadísticos comerciales. Mi experiencia me hace resaltar las siguientes:

1. Su entorno no es tan *amigable* como el de los programas comerciales. Esta cuestión está parcialmente resuelta con la utilización de la interfaz R Commander, cuya apariencia es muy similar al de estos programas.
2. Tampoco facilita las tablas de resultados de forma tan directa como otros paquete comerciales, donde exportar estas tablas a otros paquetes ofimáticos para el tratamiento de textos es tan sencillo como *copiar-pegar*.
3. Hemos comprobado que algunas opciones de ventana del paquete R Commander hacen que el programa se cuelgue con frecuencia, obligándonos a reiniciarlo. Mi experiencia es que éste es un problema muy infrecuente en el uso de R desde la consola.

El documento está pensado para que quien lo lea pueda elegir si desea manejar R utilizando su código o bien sólo hacerlo a través de los menús desplegables tipo ventana que facilita R Commander. Así se especifica en cada apartado. Tan sólo hay dos capítulos, el de estimación puntual y el de contrastes no paramétricos, que no pueden abordarse íntegramente con los menús de R Commander y otro, el de el manejo de distribuciones, en el que, pudiendo utilizar Commander, sugiero que se use el código.

Esta posibilidad de elegir entre código y ventanas del Commander podría hacer pensar en usuarios *avanzados* (los que opten por el código) o *no avanzados* (los que opten por R Commander). No me parece una distinción afortunada. Mi consejo es que se utilice el código, pero quienes estén acostumbrados a manejar programas estadísticos con licencia comercial puede que prefieran empezar con un entorno, el de Commander, que les resulte más familiar. Yo, sin embargo, les animo a que den el paso y disfruten de todas las ventajas que el manejo de un lenguaje de computación como R proporciona. También hay que reconocer que el propio diseño de R Commander, mostrando en su ventana de instrucciones el código correspondiente a las ventanas desplegables, puede ser una excelente manera de ir aprendiendo poco a poco sobre el lenguaje de R.

Para finalizar, quiero agradecer especialmente a mi compañero el Prof. Dr. Jesús Navarro Moreno, su ayuda en la depuración del documento y en la redacción de algunos de sus apartados.

Índice general

1. Instalación e introducción a R y R Commander	10
1.1. Introducción	10
1.2. Instalación de R	11
1.3. La consola y el editor de R	12
1.4. Instalación e introducción a R Commander	14
2. Preliminares sobre el lenguaje de R	18
2.1. Introducción	18
2.2. Algunos tipos de objetos de R	18
2.2.1. Vectores	19
2.2.2. Factores	20
2.2.3. Matrices	20
2.2.4. Hojas de datos	21
2.3. Funciones más comunes en R	23
2.4. Operaciones lógicas	23
2.5. La ayuda de R	24
3. Manejo de datos	26
3.1. Introducción de datos nuevos	26
3.1.1. La hoja de datos	26
3.1.2. Introducción de la hoja de datos mediante código	27
3.1.3. Introducción de una hoja de datos en R Commander	29
3.1.4. Almacenamiento de un conjunto de datos mediante código. Las funciones <i>save</i> y <i>load</i>	30

3.1.5.	Almacenamiento de un conjunto de datos en R Commander	32
3.1.6.	Datos faltantes	33
3.2.	Importar datos	33
3.2.1.	Importar datos de tipo texto	34
3.2.1.1.	Mediante código. La función <i>read.table</i>	34
3.2.1.2.	Mediante R Commander	36
3.2.2.	Importar archivos de tipo Excel	37
3.2.2.1.	Mediante código. El paquete <i>xlsReadWrite</i>	37
3.2.2.2.	Mediante R Commander	37
3.3.	Exportar datos	38
3.3.1.	Mediante código. Función <i>write.table</i>	38
3.3.2.	Mediante R Commander	38
3.4.	Recodificación de variables	39
3.4.1.	Recodificación de una variable numérica	39
3.4.1.1.	Mediante código. Función <i>recode</i> del paquete <i>car</i>	39
3.4.1.2.	Mediante R Commander	40
3.4.2.	Recodificación de una variable tipo carácter	41
3.5.	Cálculo de nuevas variables	41
3.5.1.	Mediante código	42
3.5.2.	Mediante R Commander	43
3.6.	Filtrado de datos	44
3.6.1.	Mediante código	44
3.6.2.	Mediante R Commander	45
3.7.	Almacenamiento de instrucciones y resultados	45
3.7.1.	Mediante código. El <i>script</i> y la <i>sesión de trabajo</i>	45
3.7.2.	Mediante R Commander	47
4.	Estadística descriptiva	48
4.1.	Cálculo de medidas de posición, dispersión y forma	48
4.1.1.	Mediante R Commander	48

4.1.2. Mediante código	49
4.1.3. Resúmenes por grupos	50
4.2. Distribuciones de frecuencias	51
4.2.1. Mediante R Commander	51
4.2.2. Mediante código	52
4.3. Diagrama de barras y diagrama de sectores	52
4.3.1. Diagrama de barras para variables cualitativas	52
4.3.2. Diagrama de sectores para variables cualitativas	54
4.4. Histograma para variables continuas y discretas	55
4.4.1. Histograma para variables continuas	55
4.4.2. Histograma para variables discretas	56
4.5. Detección de valores atípicos. Diagrama de caja	58
4.5.1. Mediante R Commander	58
4.5.2. Mediante código. La función <code>boxplot()</code>	59
5. Manejo de distribuciones de probabilidad	61
5.1. Introducción	61
5.2. Cálculo de probabilidades	62
5.2.1. Distribuciones discretas	62
5.2.2. Distribuciones continuas	63
5.3. Cálculo de cuantiles	64
5.3.1. Distribuciones discretas	65
5.3.2. Distribuciones continuas	65
5.4. Representaciones gráficas de distribuciones de probabilidad. La función <code>plot()</code> . . .	65
5.5. Simulación de muestras	67
5.5.1. Muestra procedente de una distribución normal	67
5.5.2. Muestra procedente de una distribución de Poisson	69
6. Estimación puntual y por intervalos de confianza	70
6.1. Introducción	70
6.2. Estimación máximo verosímil	70

6.2.1.	Para la distribución de Poisson	72
6.2.1.1.	Estimación del parámetro	72
6.2.1.2.	Comparación del modelo estimado con la distribución de frecuencias	73
6.2.2.	Para la distribución geométrica	74
6.2.2.1.	Estimación del parámetro	75
6.2.2.2.	Comparación del modelo estimado con la distribución de frecuencias	75
6.2.3.	Para la distribución binomial negativa	76
6.2.3.1.	Estimación de los parámetros	76
6.2.3.2.	Comparación del modelo ajustado con la distribución de frecuencias	77
6.2.4.	Para la distribución exponencial	78
6.2.4.1.	Estimación de los parámetros	78
6.2.4.2.	Comparación del modelo ajustado con los datos muestrales	79
6.2.5.	Para la distribución Gamma	79
6.2.5.1.	Estimación de los parámetros	79
6.2.5.2.	Comparación del modelo ajustado con los datos muestrales	81
6.2.6.	Para la distribución normal	81
6.2.6.1.	Estimación de los parámetros	82
6.2.6.2.	Comparación del modelo ajustado con los datos muestrales	82
6.3.	Estimación por intervalos de confianza	83
6.3.1.	De la media de una distribución normal con varianza desconocida	84
6.3.2.	De la media de una distribución cualquiera, con muestras grandes	84
6.3.3.	De una proporción	85
6.3.4.	De la varianza de una distribución normal	86
7.	Contraste de hipótesis paramétricas	87
7.1.	Introducción	87
7.2.	Contrastes sobre medias	87
7.2.1.	Contraste sobre la media de una población	87
7.2.1.1.	Resolución del problema mediante R Commander	88
7.2.1.2.	Consideraciones finales	90

7.2.2.	Contraste para la diferencia de medias de poblaciones independientes	90
7.2.2.1.	Resolución mediante R Commander	91
7.2.2.2.	Consideraciones finales	94
7.2.3.	Contraste para la diferencia de medias de poblaciones apareadas	94
7.2.3.1.	Resolución del problema mediante R Commander	95
7.2.3.2.	Comentarios finales	96
7.2.4.	Contrastes para medias mediante código. Función <code>t.test()</code>	96
7.3.	Contraste para la proporción en una población	98
7.3.1.	Enunciado y planteamiento del problema	99
7.3.2.	Preparación de los datos y resolución del problema mediante R Commander	99
7.3.3.	Resolución mediante código. La función <code>prop.test()</code>	101
7.4.	Contraste para la diferencia de proporciones	102
7.4.1.	Enunciado y planteamiento del problema	102
7.4.2.	Resolución del problema mediante R Commander	103
7.4.3.	Resolución mediante código	104
7.5.	Contraste para la comparación de varianzas	105
7.5.1.	Enunciado y planteamiento del problema	105
7.5.2.	Resolución del ejercicio mediante R Commander	106
7.5.3.	Resolución mediante código. La función <code>var.test</code>	107
7.6.	ANOVA de un factor	108
7.6.1.	Enunciado y planteamiento del problema	108
7.6.2.	Resolución mediante R Commander	109
7.6.3.	Resolución mediante código. La función <code>avov</code>	111
8.	Contrastes de hipótesis no paramétricos	113
8.1.	Contrastes de bondad de ajuste	113
8.1.1.	Contraste χ^2 de bondad de ajuste	113
8.1.1.1.	Para la distribución de Poisson	115
8.1.1.2.	Para la distribución geométrica	116
8.1.1.3.	Para la distribución binomial negativa	118

8.1.1.4. Para la distribución binomial	119
8.1.2. Contraste de Kolmogorov-Smirnoff	120
8.2. Contraste χ^2 de independencia	122
8.2.1. Mediante R Commander	122
8.2.2. Mediante código	123
9. Regresión lineal simple	124
9.1. Correlación	124
9.1.1. Mediante R Commander	124
9.1.2. Mediante código. Las funciones <code>pairs()</code> , <code>cors()</code> y <code>cor.test()</code>	128
9.2. Ajuste de la recta de regresión	128
9.2.1. Mediante R Commander	128
9.2.2. Mediante código. La función <code>lm()</code>	130
9.3. Predicciones, estimaciones e intervalos de confianza para ellas	131
9.3.1. Mediante R Commander	131
9.3.2. Mediante código. La función <code>predict()</code>	134
9.4. Algo sobre diagnosis del modelo	134
9.4.1. Mediante R Commander	135
9.4.2. Mediante código	135

Capítulo 1

Instalación e introducción a R y R Commander

Objetivos:

1. Instalar R y R Commander.
2. Familiarizarnos con la consola y el editor de R.
3. Familiarizarnos con la interfaz de R Commander.

1.1. Introducción

R¹ es un lenguaje de programación especialmente indicado para el análisis estadístico. A diferencia de la mayoría de los programas que solemos utilizar en nuestros ordenadores, que tienen interfaces tipo ventana, R es manejado a través de una consola en la que se introduce código propio de su lenguaje para obtener los resultados deseados.

R fue inicialmente diseñado por Robert Gentleman y Ross Ihaka, miembros del Departamento de Estadística de la Universidad de Auckland, en Nueva Zelanda. Sin embargo, una de las grandes ventajas de R es que hoy en día es, en realidad, fruto del esfuerzo de miles de personas en todo el mundo que colaboran en su desarrollo.

Por otra parte, R se considera la versión libre de otro programa propietario, llamado S o S-Plus, desarrollado por los Laboratorios Bell. Aunque las diferencias entre R y S son importantes, la mayoría del código escrito para S funciona en R sin modificaciones.

¹R Development Core Team (2011). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL <http://www.R-project.org/>.

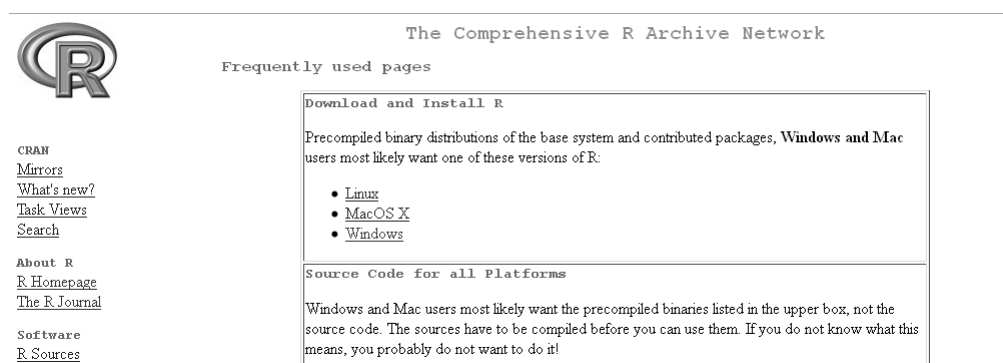


Figura 1.1: Instalación de R (I de III). Clicamos en *Windows*.

El código de R está disponible como software libre bajo las condiciones de la licencia GNU-GPL, y puede ser instalado tanto en sistemas operativos tipo Windows como en Linux o MacOS X.

La página principal desde la que se puede acceder tanto a los archivos necesarios para su instalación como al resto de recursos del proyecto R es

<http://www.r-project.org>.

1.2. Instalación de R

Vamos a explicar aquí como se realiza la instalación en Windows.

La descarga del archivo de instalación se realiza desde

<http://cran.es.r-project.org/>.

En dicha página debemos elegir la instalación en Windows, posteriormente la descarga de *base* y, finalmente, el archivo de instalación (ver Figuras 1.1, 1.2, 1.3). En el momento del inicio de la edición de esta guía la última versión era la 2.14.1. Es recomendable elegir siempre la última versión disponible. Con respecto a posibles problemas de incompatibilidades entre versiones, nosotros venimos trabajando con R desde la versión 1.3 y no hemos encontrado ninguna incidencia de esa índole.

La instalación en sí con el archivo ejecutable es trivial. El único paso que es importante en ella es el momento de decidir si deseamos la instalación en arquitecturas de 32 o 64 bits. Una vez concluida la instalación, podemos ejecutar el programa desde cualquiera de los iconos que nos genera.

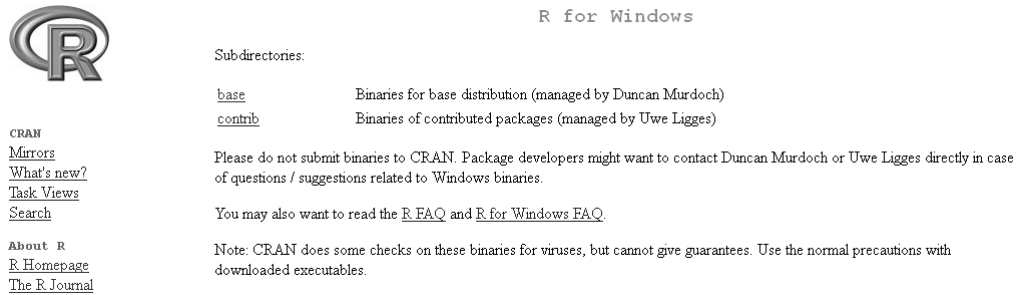


Figura 1.2: Instalación de R (II de III). Clicamos en *base*.

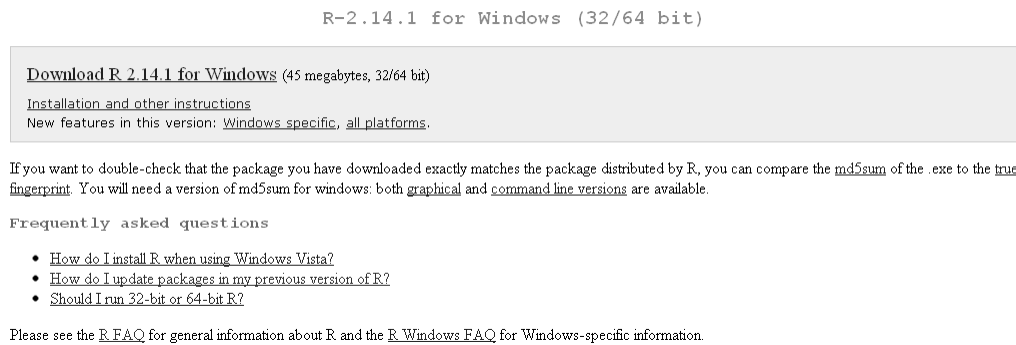


Figura 1.3: Instalación de R (III de III).

1.3. La consola y el editor de R

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

Sin embargo, esta no es la manera más eficiente de trabajar en R. A poco que estemos realizando un trabajo de mediana complejidad, será muy útil manejar todas las entradas de código que solicitemos a R en un entorno donde podamos corregirlas, retocarlas, repetirlas, guardarlas para continuar el trabajo en otro momento, etc. Esta es la función del editor de R.

Podemos ver en la Figura 1.4 cómo acceder a un documento en blanco del editor, llamado *script*. Una vez que hayamos seleccionado un nuevo script, para mayor comodidad, elegiremos la opción *Divida horizontalmente* del menú *Ventana* de la consola, después de lo cuál, el aspecto será el que aparece en la Figura 1.5.

En esa misma figura ya hemos introducido algunas líneas en el editor. Puede verse que es posible incluir comentarios que R no leerá si utilizamos líneas que comiencen con el carácter `#`. Por el contrario, si escribimos cualquier orden no antecedida de `#` y queremos solicitar la respuesta a R, podemos hacerlo mediante la tecla `F5` situándonos en cualquier posición de esa línea (no necesaria-

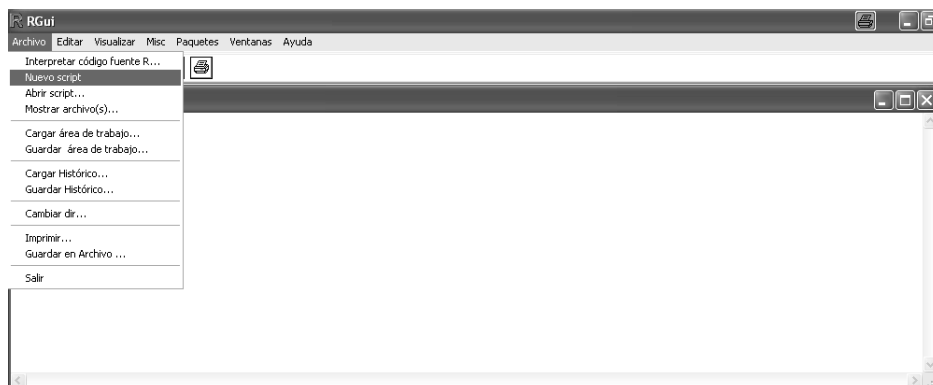


Figura 1.4: Consola de R y forma de acceder al editor

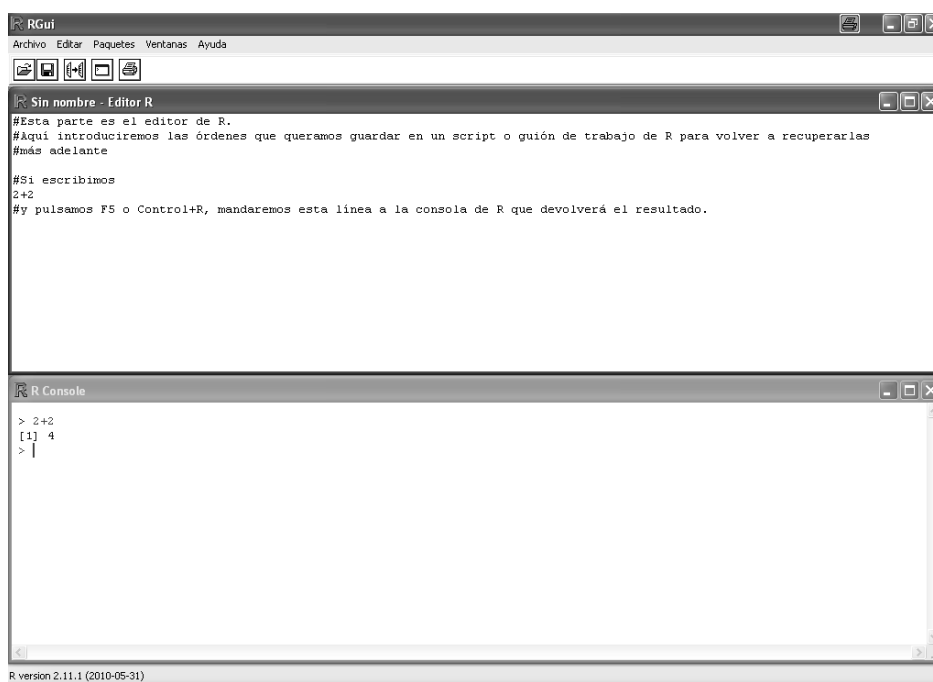


Figura 1.5: Consola y editor de R

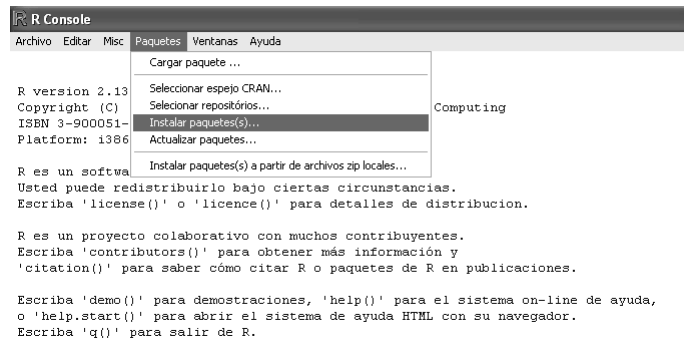


Figura 1.6: Instalación de R Commander (I de III)

mente en el final) o la combinación de teclas **Control+R**. Asimismo, si seleccionamos con el ratón más de una línea, éstas pueden ser ejecutadas simultáneamente también con **F5** o **Control+R**.

La utilidad de un script o guión de trabajo radica en que podemos modificar nuestras líneas de código con comodidad y guardarlas para el futuro. Para ello, utilizaremos la opción *Guardar* o *Guardar como* del menú *Archivo* de la consola. Evidentemente, después podremos recuperar el script previamente guardado mediante la opción *Abrir script* del mismo menú.

1.4. Instalación e introducción a R Commander

R Commander² es una interfaz tipo ventana que cubre la mayor parte de los análisis estadísticos más habituales en unos menús desplegables a los que estamos bastante acostumbrados, ya que la mayoría de los programas que utilizamos en cualquier sistema operativo son de este tipo. Podemos decir que es una manera de manejar R sin necesidad de aprender su código o casi nada de él, lo cual lo hace bastante práctico cuando se está aprendiendo a usarlo.

Además, una de las funcionalidades que podríamos destacar como más afortunadas de R Commander es que, a pesar de que permite estos *atajos* mediante sus menús para no utilizar el código de R, escribe el código de las operaciones realizadas en una ventana de sintaxis o *ventana de instrucciones*, de manera que siempre lo veremos en la pantalla y podremos, poco a poco, ir aprendiéndolo, casi sin darnos cuenta.

La instalación de R Commander se realiza en 4 sencillos pasos:

1. En la consola de R seleccionamos *Paquetes* → *Instalar paquete(s)* (ver Figura 1.6).
2. Es posible que aparezca una ventana solicitando un *mirror* desde el que descargar los paquetes, de entre los cuales elegimos, por ejemplo, *Spain* (ver Figura 1.7). Hay que decir que

²Fox, J. (2005). The R Commander: A Basic Statistics Graphical User Interface to R. Journal of Statistical Software, 14(9): 1–42.

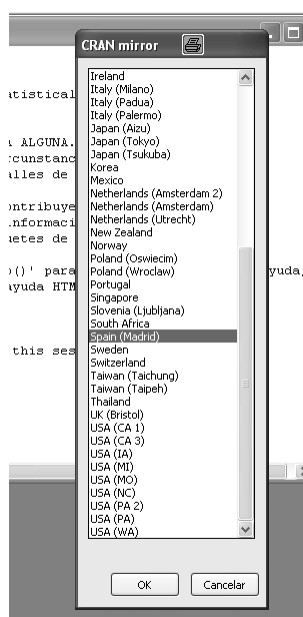


Figura 1.7: Instalación de R Commander (II de III)

ocasionalmente el *mirror* español *se cae*. Observaremos que eso ocurre si no aparece ninguna ventana de paquetes o si vemos que los paquetes que aparecen son muy pocos: en ese caso, podemos elegir otro cualquiera de los *mirror* disponibles.

3. Se abrirá una ventana donde aparecen todos los paquetes disponibles para R. Seleccionaríamos, en principio, el paquete *Rcmdr* (ver Figura 1.8). También podemos instalar algunos de los *plugins* que añaden algunas funcionalidades a R Commander. De entre ellos yo he usado con frecuencia dos: *RcmdrPlugin.HH*³ y *RcmdrPlugin.IPSUR*⁴.
4. A continuación, cargamos R Commander, introduciendo el siguiente código en la consola de R: `library(Rcmdr)`⁵. Esta primera vez que cargamos R Commander nos pedirá la instalación de otros paquetes necesarios: debemos autorizarlo, eligiendo la opción, que aparece por defecto, de descarga desde *CRAN*. No debemos extrañarnos si tarda unos minutos en descargar e instalar estos otros paquetes.

Una vez cargado R Commander⁶ veremos una ventana como la de la Figura 1.9. En ella podemos distinguir 4 partes:

³Richard M. Heiberger and with contributions from Burt Holland. (2008). *RcmdrPlugin.HH*: Rcmdr support for the HH package. R package version 1.1-21.

⁴G. Jay Kerns with contributions by Theophilus Boye, Tyler Drombosky and adapted from the work of John Fox et al. (2008). *RcmdrPlugin.IPSUR*: Introduction to Probability and Statistics Using R. R package version 0.1-5. <http://www.r-project.org>, <http://ipsur.r-forge.r-project.org/>

⁵También podemos utilizar la opción *Cargar paquete* del menú *Paquetes* y elegir en la lista nuestro paquete *Rcmdr*.

⁶En ocasiones podemos, por error o por necesidad, cerrar la ventana de R Commander. En ese caso, para volver a abrirla sin tener que reiniciar R, ejecutaremos en la consola `Commander()`.

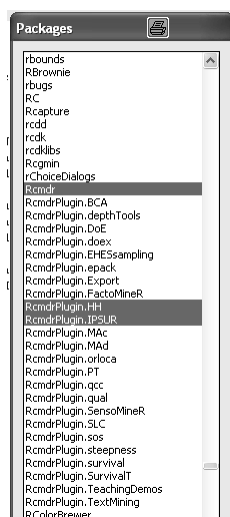


Figura 1.8: Instalación de R Commander (III de III)

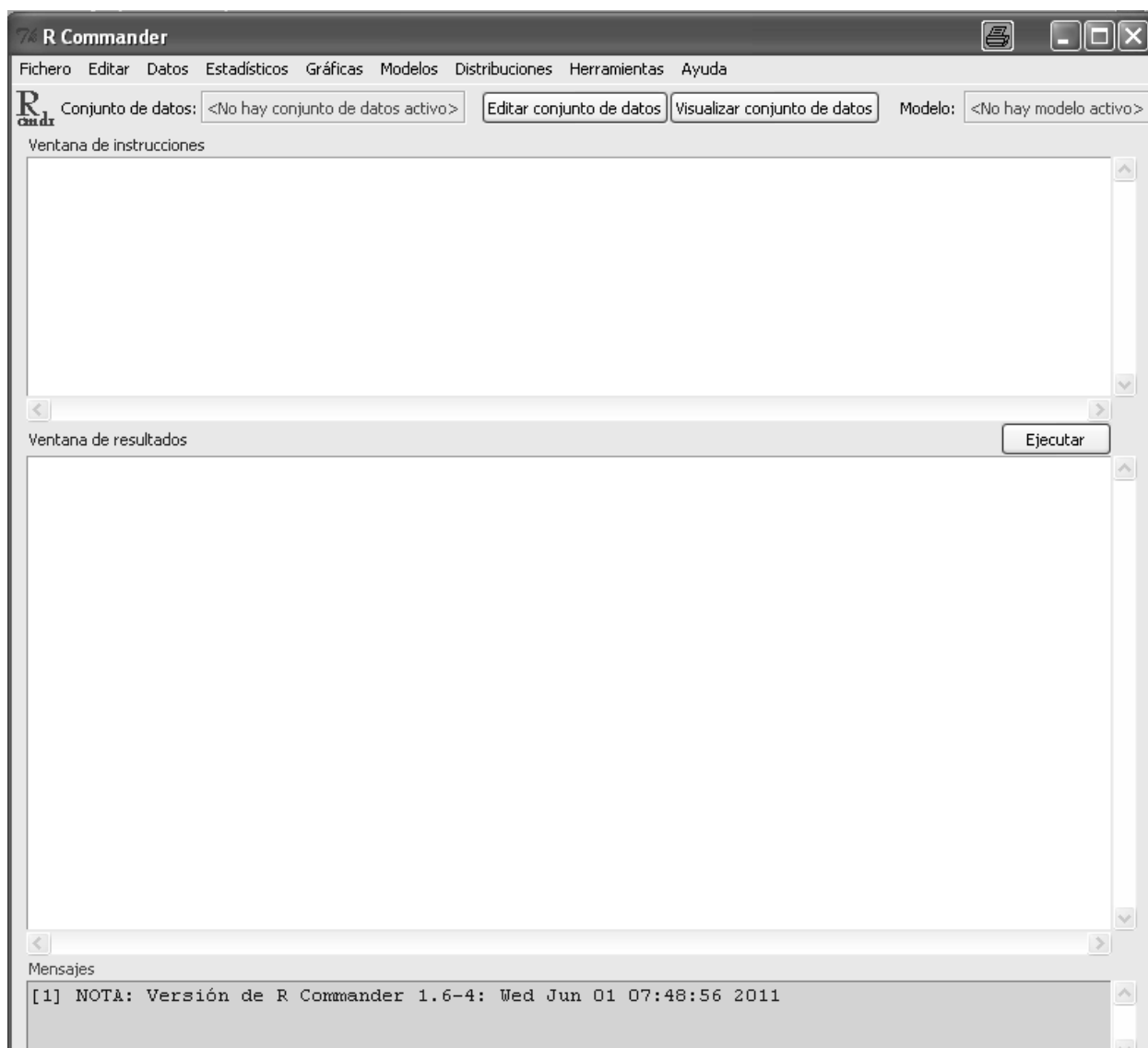


Figura 1.9: Interfaz de R Commander

1. El menú de ventanas desplegable, con las opciones *Fichero, Editar, Datos, ...*

Es un menú de ventanas con entradas bastante intuitivas, que no requieren conocimientos de R, pero sí de Estadística.

2. La ventana de instrucciones.

Cada vez que ejecutemos alguna acción del menú, R Commander traducirá dicha acción a código de R y lo escribirá en esta ventana. Como decíamos, eso permite ir aprendiendo este código y, además, facilita la posibilidad de volver a ejecutar la misma acción o una ligera variante de la misma retocando el código, sin tener que volver a utilizar el menú.

Por otra parte, esta ventana de instrucciones es equivalente al editor de R. Por ejemplo, podemos escribir $2+2$, clicar en el botón de *ejecutar* (equivalente a **F5** o **Control+R**) obteniendo el resultado.

3. La ventana de resultados.

Si hemos realizado ese sencillo ejemplo en la ventana de instrucciones, habremos visto que el resultado aparece en esta ventana. En general, cualquier resultado de R Commander será mostrado aquí.

4. La ventana de mensajes.

Es la más inferior de todas y aparece ligeramente sombreada. Sirve para que R Commander nos informe de cualquier aspecto, especialmente de errores cometidos.

Para finalizar esta breve introducción a R Commander, queremos comentar que los paquetes adicionales (*plugins*) que podemos instalar junto con R Commander son complementos que diversos autores de contenidos de R han ido poniendo a disposición de la comunidad de usuarios de R. Para cargar estos *plugins* se debe elegir en el menú de R Commander *Herramientas* → *Cargar plugins de R Commander* y seleccionarlos. Se pedirá reinicializar R Commander, tras lo cual todos ellos están disponibles.

Capítulo 2

Preliminares sobre el lenguaje de R

Objetivos:

1. Introducir los tipos de objetos más comunes de R.
2. Introducir las funciones más sencillas del lenguaje de R.
3. Describir las operaciones lógicas en R.
4. Describir el funcionamiento de la ayuda de R.

2.1. Introducción

Este capítulo pretende ofrecer una descripción de algunas cuestiones relativas al lenguaje de R que resultarán de interés en el resto del documento. Podrían parecer a un lector que se inicie en R desde aquí *demasiado complejas*, y a un lector que ya haya manejado R previamente, *demasiado simples*. Por mi parte, creo necesario dejar plasmadas estas cuestiones, a pesar de todo, porque constituyen una base que después puede afianzarse conforme necesitemos un uso más avanzado del programa. También describimos el uso de la ayuda, lo que facilita información útil sobre las funciones y el lenguaje de R.

2.2. Algunos tipos de objetos de R

En el lenguaje de R, los elementos u *objetos* que se vayan definiendo, bien por nosotros mismos, bien como resultado del uso del programa, pueden y deben ser distinguidos para su uso correcto. Por ejemplo, una matriz, por su propia definición, es una colección de números configurados en filas y columnas, todas ellas de la misma longitud. Sin embargo, en ocasiones es necesario reunir números

en vectores y estos en algún objeto, cuando no todos ellos tienen la misma dimensión: esto es posible en un tipo especial de objeto llamado *lista*. Desde luego, una lista no es una matriz, luego, aunque nos sirva para meter en ella vectores de dimensiones distintas, no admite las operaciones matriciales habituales, por ejemplo.

No pretendo ser exhaustivo en la descripción de los tipos de objeto de R. Tan sólo voy a describir los que creo que son más utilizados en el contexto de un manual como éste. Concretamente, vamos a hablar de:

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

Insisto en que otros tipos de objetos, como las listas, las variables indexadas (arrays), las funciones o los modelos, son también muy importantes, pero o bien no los vamos a utilizar aquí o su uso puntual no requiere que sea necesario describirlo.

2.2.1. Vectores

Un vector en R puede contener una colección de números o de caracteres no numéricos.

Para definir un vector, por ejemplo, el vector $x = (1, 3, 5)$, usaríamos la orden

```
x<-c(1,3,5)
```

Así podremos llamar al vector x en el futuro. Observemos que se utiliza el operador asignación `<-` y no el operador `=` que se reserva para otro tipo de definiciones. Observemos también que es la función de concatenación `c()` la que construye el vector.

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

```
1:5
```

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

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

y

proporciona

```
[1] -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0
```

Obsérvese que al ejecutar `y<-seq(-3,3,0.5)` no ha aparecido el valor de `y` en la consola: eso es porque hemos hecho una definición, pero no hemos pedido al programa que *imprima* dicha definición; es por eso que después hemos ejecutado una simple llamada al vector `y` para ver su valor.

También es útil la función `rep()` para definir vectores como repetición de otros vectores. Por ejemplo, `rep(0,100)` devolvería un vector de 100 ceros. O también, `rep(1:3,3)` devolvería

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

Si queremos saber la longitud de un vector, usaremos `length()`. Por ejemplo, `length(y)` nos devolvería el valor 13.

Decir, por último, que no hay problema en que un vector, en vez de incluir números, incluya caracteres, siempre que éstos estén entre comillas. Por ejemplo, podríamos definir el vector

```
genero<-c("Mujer","Hombre")
```

2.2.2. Factores

Los factores son un tipo especial de vectores¹ que permiten analizar un conjunto de datos clasificados según las características que definen el factor.

Por ejemplo, si deseamos analizar datos de una muestra de 5 municipios andaluces según su provincia, podríamos definir el factor *provincia* de la siguiente manera:

```
provincia<-factor(c("Huelva","Huelva","Jaén","Sevilla","Almería"),  
levels=c("Huelva","Cádiz","Málaga","Granada","Almería","Jaén","Córdoba","Sevilla"))
```

2.2.3. Matrices

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

Por ejemplo, para definir la matriz

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

lo haríamos usando

```
matriz<-matrix(c(1,2,3,4,5,6,7,8,9),3,3)
```

¹Formalmente, no son vectores.

Las dimensiones (nº de filas y columnas) de la matriz pueden obtenerse mediante la función `dim()`. Por ejemplo, `dim(matriz)` proporcionaría el valor 3 3.

Si queremos llegar a elementos concretos de una matriz lo haremos utilizando corchetes para indicar filas y columnas. Por ejemplo, `matriz[2,3]` devolvería el valor 8, `matriz[1:2,2:3]` devolvería

$$\begin{pmatrix} 4 & 6 \\ 5 & 7 \end{pmatrix}$$

y `matriz[,c(1,3)]` devolvería

$$\begin{pmatrix} 1 & 7 \\ 2 & 8 \\ 3 & 9 \end{pmatrix}.$$

Esa misma sintaxis permite manejar los elementos de un vector.

Por otra parte, tanto para vectores como para matrices funcionan las operaciones suma y diferencia sin más complicaciones.

En el caso del producto, sin embargo, hay que clarificar que, por ejemplo, `matriz*matriz` devuelve la multiplicación elemento a elemento, es decir,

$$\begin{pmatrix} 1 & 16 & 49 \\ 4 & 25 & 64 \\ 9 & 36 & 81 \end{pmatrix},$$

mientras que `matriz**matriz` sí devuelve el producto matricial, esto es,

$$\begin{pmatrix} 30 & 66 & 102 \\ 36 & 81 & 126 \\ 42 & 96 & 150 \end{pmatrix}.$$

2.2.4. Hojas de datos

Las hojas de datos (*data frames* en inglés) constituyen la manera más eficiente en que R puede analizar un conjunto estadístico de datos. Habitualmente se configuran de tal manera que cada fila se refiere a un elemento de la muestra que se analiza, mientras que cada columna hace referencia a las distintas variables analizadas. Con una nomenclatura estadística, diríamos que las filas son los casos y las columnas son las variables. Esa configuración hace que, visualmente, una hoja de datos parezca una matriz. Sin embargo, como objetos de R, son cosas distintas.

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

Empezaríamos definiendo el color de los ojos:

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

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

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

Así, tendremos tres variables, llamadas *Color.ojos*, *Peso* y *Altura*.

Podemos forzar a que una matriz se convierta en una hoja de datos mediante la función `as.matrix`. Por ejemplo,

```
datos2<-as.data.frame(matriz)
```

convertiría `matriz` en una hoja de datos. Si ponemos `names(datos2)` veremos los nombres que para las variables ha elegido por defecto R:

```
[1] "V1" "V2" "V3"
```

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

```
names(datos2)<-c("Variable 1","Variable 2","Variable 3")
```

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

1. Podemos usar el mismo método que para las matrices.
2. Podemos usar el operador `$`, de la siguiente manera. Para obtener los datos de la variable *Color.ojos*, por ejemplo, escribiríamos `datos$Color.ojos`.

Para saber el número de filas y de columnas de una hoja de datos utilizaremos las funciones `nrow()` y `ncol()`. Por ejemplo, `ncol(datos)` es 3.

¿Qué ocurre si no estamos seguros de que un objeto de R, que sabemos que contiene datos, sea una hoja de datos o una matriz? Existen funciones que nos informan sobre el carácter de los objetos. En el caso que nos ocupa, `is.vector()`, `is.matrix()` e `is.data.frame()`. Así, por ejemplo, `is.data.frame(matriz)` devolverá `FALSE`, mientras que `is.data.frame(datos2)` devolverá `TRUE`.

2.3. Funciones más comunes en R

En este apartado quiero destacar algunas funciones de R que usaremos en el resto del documento. Por ejemplo:

- `sum()` proporciona la suma de los elementos del argumento. Así, `sum(x)` daría el valor 9.
- `cumsum()` proporciona un vector con la suma acumulada del vector argumento. Por ejemplo, teniendo en cuenta que $x = (1, 3, 5)$, `cumsum(x)` daría

```
[1] 1 4 9
```
- `rowSums()` y `colSums()` suman, por filas y por columnas, respectivamente, los datos de una hoja de datos.
- `prod()` y `cumprod()` son el equivalente a `sum()` y `cumsum()` para el producto.
- `sqrt()` es la función raíz cuadrada.
- `log()` es la función logaritmo natural o neperiano.
- `log10()` es el logaritmo en base 10.
- `exp()` es la función exponencial.
- `max()` y `min()` proporcionan el máximo y el mínimo del argumento (habitualmente, un vector).
- `sort()` proporciona la ordenación de un vector de menor a mayor.

2.4. Operaciones lógicas

Voy a enumerar a continuación los operadores lógicos más utilizados, considerando un ejemplo en cada caso que involucre a los objetos `x`, `y`, `matriz` y `datos` que ya hemos definido:

- `<`, `>`, `<=` y `>=` son los operadores *menor*, *mayor*, *menor que* y *mayor que*, respectivamente. Por ejemplo, `x>=1.5` devolvería

```
[1] FALSE TRUE TRUE
```
- `==` es el operador de igualdad. Por ejemplo, `datos$Color.ojos=="Marrones"` devuelve

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

- `&y` | son los operadores *y* y *o*, respectivamente. Por ejemplo, `(matriz>2)&(matriz<4)` devuelve
`[,1] [,2] [,3]`
`[1,] FALSE FALSE FALSE`
`[2,] FALSE FALSE FALSE`
`[3,] TRUE FALSE FALSE`

2.5. La ayuda de R

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

Por ejemplo, `?sort` abrirá una ventana de nuestro explorador² con todos los detalles sobre el uso de esa función, incluyendo interesantes ejemplos. Personalmente, creo que las ayudas suelen ser muy eficaces, aunque el alumnado suele desanimarse un poco cuando comprueban que están en inglés.

Pero, ¿qué ocurre si queremos ayuda sobre un aspecto del que desconocemos qué función nos lo facilita? Supongamos, por ejemplo, que deseamos saber cómo se realiza la descomposición de Choleski de una matriz. En ese caso, si no sabemos qué función facilita esa descomposición, pondremos `??choleski`. Eso abrirá una ventana de R con todas las funciones que incluyen la palabra *Choleski* en su ayuda. El resultado concreto podemos verlo en la Figura 2.1.

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

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

Un último paso en la búsqueda de ayuda. ¿Qué ocurre si necesitamos ayuda sobre algo que está en una función de un paquete que nosotros no tenemos instalado? Tengamos en cuenta que, al instalar R tan sólo incorporamos una mínima parte de los paquetes que el proyecto CRAN tiene, gracias a la colaboración de los miles de desarrolladores de R³, así que, si no encontramos ayuda en los

²La ayuda, por defecto, se facilita en formato HTML.

³Humildemente he de decir que, junto con otros investigadores de nuestra universidad, he participado en el desarrollo de uno de esos paquetes, llamado *GWRM*, donde se analiza un curioso conjunto de datos. Animo al lector a que curioseé sobre él.

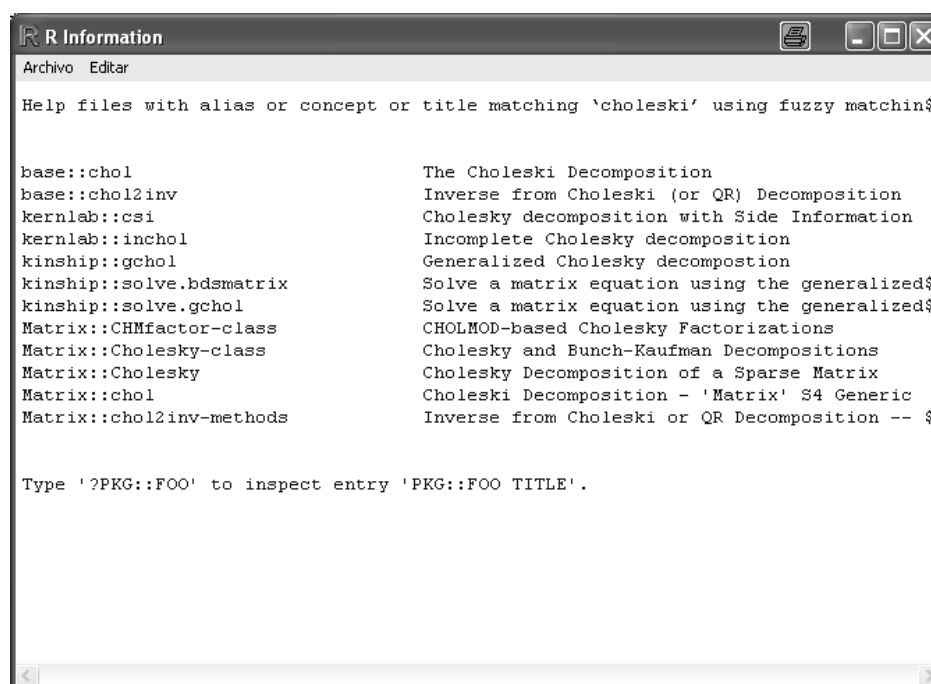


Figura 2.1: Información que da la ayuda de R a propósito de la entrada `choleski`

paquetes instalados por defecto, puede que aún así, exista un paquete en CRAN donde haya algo al respecto. Mi experiencia es que, dada la amplísima difusión de R, si en algún momento buscamos algo sobre un tema concreto, podemos tener fundadas esperanzas de que alguien antes que nosotros se habrá hecho la misma pregunta y probablemente ha encontrado la respuesta por sí mismo o con la ayuda de otros usuarios de R. La cuestión es, ¿cómo acceder a esa información? En este punto, la ayuda de R nos permite acceder, por un lado, a la ayuda de todos los paquetes alojados en CRAN y, por otro, a todos los mensajes con preguntas y respuestas que diariamente fluyen a través de las listas de correo de los usuarios y desarrolladores de R. Y es bien sencillo. Tan sólo tenemos que teclear `RSiteSearch("football")` para, por ejemplo, buscar información sobre funciones que, en algún lugar de la ayuda, incluyan la palabra `football`. Eso abrirá una ventana de nuestro navegador donde podemos elegir el ámbito de nuestra consulta.

Capítulo 3

Manejo de datos

Objetivos:

1. Introducir y almacenar un nuevo conjunto de datos.
2. Importar datos ya almacenados en distintos formatos.
3. Exportar datos en algún formato estandar.
4. Recodificar variables.
5. Calcular nuevas variables a partir de otras ya existentes.
6. Filtrar casos.

3.1. Introducción de datos nuevos

3.1.1. La hoja de datos

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

Por ejemplo, consideremos que tenemos la puntuación en una prueba escrita (x) y en una prueba

oral (y), de una muestra de 10 personas. Su matriz de datos es la siguiente:

161	159
203	206
235	241
176	163
201	197
188	193
228	209
211	189
191	169
178	201

En esta matriz la primera columna corresponde a la variable x y la segunda a la variable y , mientras que, por ejemplo, la fila 4ª corresponde a la persona nº 4 de la muestra. En resumen, no lo olvidemos, los individuos están en las filas y las variables en las columnas.

3.1.2. Introducción de la hoja de datos mediante código

De esta cuestión ya hemos hablado en la sección 2.2.4. Empezaríamos introduciendo los datos de las variables x e y en forma de vector. Además, incluimos el género de cada persona que hace la prueba:

```
x<-c(161,203,235,176,201,188,228,211,191,178)
y<-c(159,206,241,163,197,193,209,189,169,201)
genero<-factor(c("Hombre","Mujer","Hombre","Hombre","Hombre","Mujer",
,"Mujer","Mujer","Hombre","Hombre"))
```

Ahora definiríamos la hoja de datos:

```
Datos.Pruebas<-data.frame(Prueba.escrita=x,Prueba.oral=y,Genero=genero)
```

Así, hemos llamado a la hoja de datos *Datos.Pruebas*. Por su parte, a la primera variable la hemos llamado *Prueba.escrita* y a la segunda *Prueba.oral*. Después, si escribimos en la consola *Datos.Pruebas*, visualizaremos el resultado.

Tenemos, por tanto, la hoja de datos que se llama *Datos.Pruebas*, que, a su vez, contiene dos variables, *Prueba.escrita* y *Prueba.oral*. Recordemos que si queremos ahora trabajar con alguna de esas dos variables tenemos dos opciones:

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

```
Datos.Pruebas$Prueba.escrita
```

```
Datos.Pruebas$Prueba.oral
```

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

```
attach(Datos.Pruebas)
```

```
Prueba.escrita
```

```
Prueba.oral
```

```
detach(Datos.Pruebas)
```

Observemos que al hacer `attach(Datos.Pruebas)` ya no tenemos que escribir `Datos.Pruebas$` y a continuación el nombre de las variables. Eso es porque ahora `Datos.Pruebas` es la hoja de datos activa. Podremos referirnos a las variables así hasta que hagamos `detach(Datos.Pruebas)`, momento en el que dejará de ser la hoja de datos activa.

Si queremos referirnos a un elemento concreto de una hoja de datos, ya sabemos que también podemos identificarlo por su posición dentro de la matriz que constituye la hoja de datos. Por ejemplo, supongamos que queremos saber el resultado de la prueba escrita (1ª variable) del 5º individuo de la muestra. En ese caso pondríamos en la consola `Datos.Pruebas$Prueba.escrita[5]` o también, `Datos.Pruebas[5,1]`.

Supongamos que queremos los resultados de la prueba oral (2ª variable) de los individuos 5º, 8º y 10º. Los lograríamos poniendo `Datos.Pruebas$Prueba.oral[c(5,8,10)]` o bien `Datos.Pruebas[c(5,8,10),2]`.

O, por ejemplo, si deseamos los resultados de ambas pruebas de los individuos del 3º al 8º pondríamos `Datos.Pruebas[3:8,1:2]` o simplemente `Datos.Pruebas[3:8,]`.

Vamos a introducir, también a modo de ejemplo, el conjunto de datos que se resumen como una distribución de frecuencias en el Cuadro 3.1. Si lo hacemos con la función `c()` tendríamos que meter 109 ceros, 65 unos, ... En su lugar, podemos introducir la repetición de 109 ceros como `rep(0,109)`, 65 unos como `rep(1,65)`... de manera que los datos quedarían como sigue:

```
muertes<-c(rep(0,109),rep(1,65),rep(2,22),rep(3,3),4).
```

Ahora tenemos que convertirlos en una hoja de datos mediante

```
Datos.muertes<-data.frame(Muertes<-muertes)
```


Muertes/año	Frecuencia
0	109
1	65
2	22
3	3
4	1

Cuadro 3.1: Distribución de frecuencias

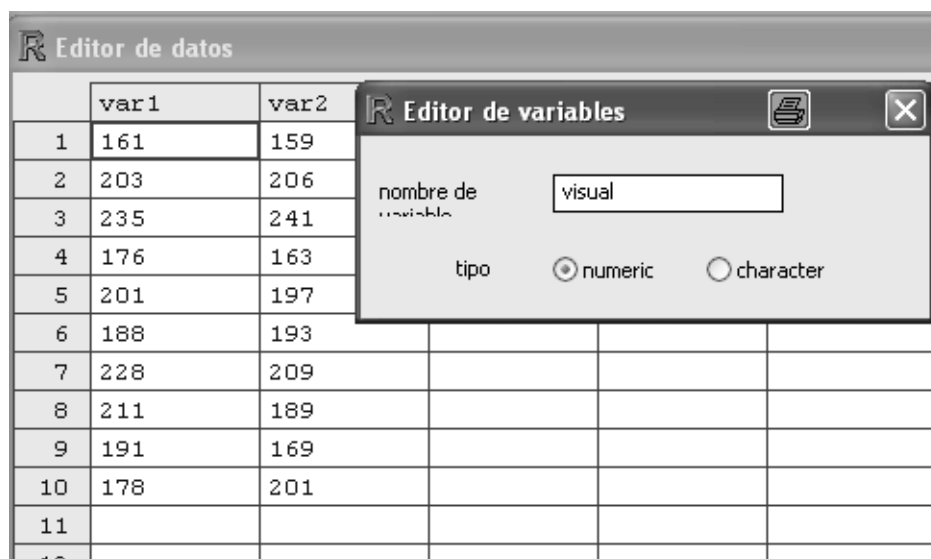


Figura 3.1: Introduciendo datos y renombrando variables

3.1.3. Introducción de una hoja de datos en R Commander

Para introducir los datos de las dos pruebas en R Commander elegimos *Nuevo conjunto de datos* del menú *Datos*. Eso abre el *editor de datos* que, en primer lugar, nos pedirá un nombre para la matriz de datos (ahora hemos elegido *Pruebas*) y a continuación abrirá una ventana con casillas parecida a una hoja de cálculo de Excel. En esta hoja debemos introducir los datos con la misma estructura que tiene la matriz de datos que acabamos de escribir, con los individuos en las filas y las dos variables en dos columnas.

Una vez introducidos los datos, debemos nombrar las variables, es decir, las columnas, con nombres sencillos que nos recuerden a qué variable corresponde cada columna. Para ello clicamos con el ratón sobre la parte superior de cada columna, donde R Commander nombra por defecto las variables como *var1*, *var2*, etc. y escribimos otros nombres más acordes con nuestros datos¹. En este caso he nombrado las variables como *escrita* y *oral* (Figura 3.1).

Por último, fijémonos que en la ventana donde R Commander nos permite cambiar el nombre de la variable aparece señalada la opción *numeric*. Eso es porque al haber introducido datos numéricos, el

¹No deben introducirse nombres que incluyan espacios, ni caracteres extraños, ni muy largos.

programa ya ha asimilado que es una variable numérica. Si hubiéramos introducido datos cualitativos mediante caracteres no numéricos, aparecería activada la opción *character*. En ese caso, además, R Commander considerará que una variable tipo *character* es un factor.

Para terminar, cerramos la ventana del editor de datos. En ese momento, R habrá almacenado los datos introducidos convirtiéndolos en lo que R Commander llama el *conjunto de datos activo* y que antes, con el código, hemos logrado con la función *attach()*. Observad que justo encima de la ventana de instrucciones aparece ahora una pestaña informativa que pone *Conjunto de datos: Pruebas*. Esta ventana especifica que, en efecto, el conjunto de datos activo en este momento es el que nosotros hemos llamado *Pruebas*.

Finalmente, podemos retocar estos datos pulsando la pestaña *Editar conjunto de datos* que hay justo sobre la ventana de instrucciones o simplemente visualizarlos pulsando la pestaña *Visualizar conjunto de datos*.

Como comentario final, debemos advertir que por problemas con el lenguaje de programación en el que está diseñado R y R Commander, es frecuente que al abrir y cerrar el editor de datos, el programa se cuelgue, por lo que recomendamos hacerlo sólo cuando sea imprescindible. Además, también es recomendable cerrar la ventana del editor, evitando dejarla minimizada.

3.1.4. Almacenamiento de un conjunto de datos mediante código. Las funciones *save* y *load*

Debemos tener presente que el conjunto de datos que hemos introducido está sólo almacenado temporalmente, y que si cerramos R serán eliminados. Para que esto no ocurra podemos guardarlos y cargarlos con posterioridad.

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

```
setwd("D:/Asignaturas/Curso R")  
save(Datos.Pruebas,file="Pruebas.RData")
```

1. La función `setwd()` permite cambiar el directorio donde estamos trabajando (*working directory*). Hay que tener en cuenta que los datos se guardarán en ese directorio.

2. Después hemos ejecutado la función `save()` para guardar *Datos.Pruebas* en el fichero *Pruebas.RData*.

Si reiniciamos el programa o borramos todos los objetos mediante `rm(list=ls(all=TRUE))`², al poner *Datos.Pruebas* daría error, porque tal objeto ya no existe. Sin embargo, al cargar el archivo *Pruebas.RData* mediante la función `load()`, recuperamos la hoja de datos:

```
load("Pruebas.RData")
```

Sobre el directorio de trabajo En todo el manejo de archivos, habitualmente de datos, el usuario que comienza con R debe tener en cuenta con atención cuál es el directorio de trabajo (o *working directory*). Suelen ser muchos los problemas que se encuentran cuando no se tiene este concepto claro, así que vamos a centrarnos un poco en él.

En primer lugar, como hemos dicho, lo más importante es situar a R en el directorio de trabajo de nuestro ordenador, es decir, el sitio donde localizaremos todos los datos, los resultados, los gráficos, etc., de nuestro análisis. Mi sugerencia es que cada trabajo distinto se sitúe en un directorio distinto. Podemos situarnos en un directorio concreto de dos formas:

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

```
setwd("D:/Asignaturas/Curso R")
```

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

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

¿Y cómo podemos saber en un momento dado si estamos en el directorio correcto? Mediante la función `getwd()`. Si, por ejemplo, yo arranco R y escribo `getwd()`, el resultado es

```
[1] "D:/Datos de Usuario/UJA/Mis documentos"
```

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

```
[1] "D:/Asignaturas/Curso R"
```

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

²Elegimos en el menú de la consola de R *Misc*→*Remove todos los objetos*.

1. Arranquemos R y, rápidamente busquemos nuestro directorio de trabajo mediante la opción *Cambiar dir...* del menú *Archivo* de la consola de R.
2. Hagamos `getwd()` y copiemos el resultado que sale entre comillas.
3. Escribamos la primera línea de nuestro script con `setwd()` incluyendo como argumento el resultado anterior que hemos copiado.

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

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

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

Ahora podemos tener otro problema: al cargar un fichero de datos de R, este fichero incorpora a nuestro entorno de trabajo de R uno o varios objetos, habitualmente una o varias hojas de datos. ¿Cómo podemos saber cómo se llaman? Mediante la función `objects()`. Simplemente, ejecutando `objects()` obtendremos todos los nombres de todos los objetos que en ese momento están definidos en nuestra sesión de R. Por ejemplo, en el momento actual, mi sesión de trabajo incluye como objetos

```
[1] "Datos" "genero" "muertes" "x" "y"
```

3.1.5. Almacenamiento de un conjunto de datos en R Commander

Para guardar una hoja de datos en R Commander, seleccionamos en el menú *Datos* la opción *Conjunto de datos activo* y, dentro de ésta, *Guardar el conjunto de datos activo* (Figura 3.2). A

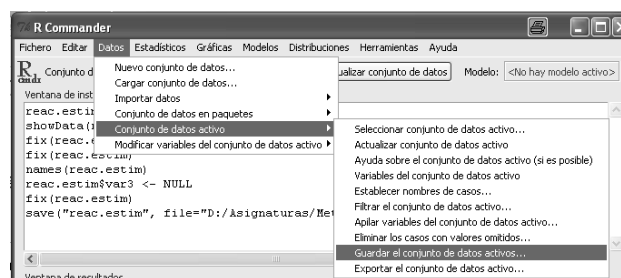


Figura 3.2: Guardando un conjunto de datos activo

continuación nos pedirá un nombre y un directorio donde almacenar el fichero, cuya extensión por defecto será *.rda*³.

Si posteriormente queremos cargar estos datos, no tenemos más que usar la opción del menú *Datos* → *Cargar conjunto de datos* y buscar el archivo correspondiente mediante la ventana del explorador que se abre.

3.1.6. Datos faltantes

¿Qué ocurre si, por alguna razón, nos falta el dato de una variable referida a un individuo concreto? Eso se conoce en Estadística como *dato faltante* o *missing data*.

Para R es muy fácil identificar esos datos cuando los introducimos *a mano* con R Commander: basta con dejar esa casilla vacía, en cuyo caso el editor de datos escribirá en ella NA, acrónimo de *Not Available*.

Si estamos trabajando con código, el carácter para un dato faltante es también *NA*. Por ejemplo, si tengo un vector de 5 datos y el 3º de ellos es un dato faltante, debería escribir, por ejemplo, `c(3,2,NA,2,8)`.

3.2. Importar datos

Hay que decir que introducir datos a mano puede convertirse en una tarea muy pesada a poco que el número de casos o de variables sea medianamente alto. Hoy en día, por otra parte, es bastante común tener los datos almacenados en algún tipo de formato electrónico y la clave del éxito para aprovechar estos recursos y no tener que introducir los datos manualmente radica en hacer que nuestro programa estadístico, en este caso R, *lea* estos datos.

³La extensión que por defecto asigna R a sus archivos de datos es *RData*, mientras que R Commander utiliza por defecto *rda*. Sin embargo, no hay ningún problema con que desde R se guarden o se lean archivos con extensión *rda* ni para que R Commander guarde o lea archivos con extensión *RData*.

Los formatos de archivo más habituales en los que nos podemos encontrar unos datos son, en primer lugar, los archivos tipo texto (con extensión *.txt*) y, en segundo lugar, los archivos de Microsoft Excel (con extensión *.xls*). Existen otros muchos formatos, pero casi siempre son convertibles a estos dos tipos. De hecho, el propio Excel o su análogo en OpenOffice, Calc, permiten transformar archivos de texto en archivos *.xls* y viceversa.

3.2.1. Importar datos de tipo texto

Los archivos de tipo texto que contienen datos suelen tener una estructura en la que los individuos están en filas distintas y las variables están separadas por algún tipo de carácter, tales como comas, tabulaciones, espacios u otros. Si no es así, y esto es muy importante, no deberíamos tratar de importarlos.

Además, es posible que la primera fila contenga los nombres de las variables. Y, por último, también es necesario fijarse en cómo están especificados los decimales, si los hay⁴.

Estas tres cuestiones,

- el hecho de que el archivo incluya los nombres de las variables,
- el carácter que separa las variables y
- el carácter que distingue los decimales,

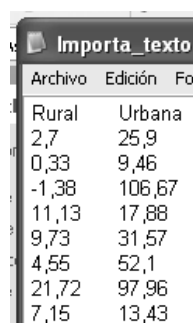
son las necesarias para importar los datos.

Vamos a ver cómo se hace mediante un ejemplo. En el archivo *Importa_texto.txt* aparecen datos relativos al grado de contaminación, en *ppm* de unas muestras de terreno recopiladas en el ámbito rural y urbano, respectivamente. Si abrimos este fichero con el bloc de notas, tiene el aspecto que aparece en la Figura 3.3. En ella podemos ver que, en efecto, se incluye el nombre de las variables y que éstas están separadas por tabulaciones. Además, los decimales están separados por comas, no por puntos.

3.2.1.1. Mediante código. La función *read.table*

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

⁴En español, el carácter que separa los decimales es la coma, mientras que en el mundo anglosajón es el punto.



Rural	Urbana
2,7	25,9
0,33	9,46
-1,38	106,67
11,13	17,88
9,73	31,57
4,55	52,1
21,72	97,96
7,15	13,43

Figura 3.3: Archivo de texto. Variables separadas por tabulaciones y nombres de variables incluidos en la primera línea

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

En esta línea:

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

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

```
Datos<-read.table("Importa_texto.txt",header=TRUE,sep="\t",dec=",")
```

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

Podemos hacer lo mismo si abrimos con el bloc de notas el archivo *Importa_texto.txt*, seleccionamos y copiamos todos los datos (seleccionamos con el ratón y pulsamos **Control+C**) y, una vez que estos datos están en el portapapeles, ejecutamos

```
Datos<-read.table(textConnection(readClipboard()),header=TRUE,sep="\t",dec=",")
```



Figura 3.4: Importando archivos tipo .txt

3.2.1.2. Mediante R Commander

Nos vamos a la opción del menú *Datos* → *Importar datos* → *desde archivo de texto o portapapeles...*. Se abre una ventana como la de la Figura 3.4, en la que debemos elegir las opciones del archivo *Importa_texto.txt*:

- Nombre: Por ejemplo, *Datos*.
- Nombre de las variables en el fichero: activado.
- Indicador de valores ausentes: lo dejamos en blanco.
- Separador de campos: tabuladores.
- Carácter decimal: coma.

Como vemos, se puede escoger entre buscar los datos dentro de un archivo de nuestro disco duro (sistema de archivo local) o bien desde el portapapeles. En el primer caso, se abre una ventana del explorador para que encontremos el archivo y lo seleccionamos. Ahora el conjunto de datos activo es *Datos*. Si lo deseamos, podemos guardar este conjunto de datos activo con formato *.rda* o para que la próxima vez no tengamos que importarlo de nuevo.

3.2.2. Importar archivos de tipo Excel

3.2.2.1. Mediante código. El paquete *xlsReadWrite*

Para importar archivos de tipo Excel⁵, debemos previamente instalar el paquete *xlsReadWrite*⁶, mediante el menú de instalación de paquetes de R⁷.

Vamos a ilustrar el procedimiento de instalación de un archivo llamado *Concentraciones.xls*. Este archivo tiene una única hoja de cálculo, llamada *Hoja1*, que recoge la concentración de ésteres totales de 16 muestras de vino, en *mg/L*. Recordemos que un archivo de Excel puede contener varias hojas de cálculo.

Las sentencias a ejecutar serían las siguientes:

```
library(xlsReadWrite)

Datos<-read.xls("Concentraciones.xls")
```

Si la hoja de cálculo no fuera la primera del archivo de Excel, podemos importarla especificando cuál es mediante la opción *sheet = X*, donde *X* debe sustituirse por el entero correspondiente. Por ejemplo, si los datos no estuvieran en la primera hoja de *Concentraciones.xls* sino en la segunda, sería `read.xls("Concentraciones.xls",sheet=2)`. Existen otras opciones muy útiles de la función `read.xls()`, que recomendamos revisar mediante la ayuda de dicha función.

En ese caso, *Datos* pasa a ser una hoja de datos manejable por R. Esencialmente, debemos tener en cuenta el nombre del fichero de Excel y el de la hoja que contiene los datos.

3.2.2.2. Mediante R Commander

En el caso de los archivos tipo Excel, R Commander no necesita que le digamos nada, ya que detecta automáticamente los nombres de las variables si están presentes. No obstante, éstos no deben incluir caracteres extraños, y deben estar todos los nombres de todas las variables o ninguno; en cualquier otro caso, la importación podría ser inválida. De hecho, si la hoja de Excel no es propiamente una hoja de datos (casos en fila, variables en columnas), la importación fracasará.

Tan sólo tenemos que utilizar la opción del menú *Datos → Importar datos → desde conjunto de datos Excel, Access o dBase...*, eligiendo después el archivo a través de la ventana del explorador.

⁵Nosotros trabajamos con versiones de Excel hasta la 2003. Hemos visto en la ayuda que también es posible importar de la versión de Excel 2007, pero no lo hemos probado.

⁶Hans-Peter Suter (2011). *xlsReadWrite: Read and write Excel files (.xls)*. R package version 1.5.4. <http://CRAN.R-project.org/package=xlsReadWrite>.

⁷Dependiendo de la configuración del equipo, es posible que sea necesario ejecutar la línea de comando `xls.getshlib()` para completar la instalación: en todo caso, esto será advertido como un mensaje en la consola de R.

3.3. Exportar datos

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

3.3.1. Mediante código. Función *write.table*

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

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

Vamos a comentar los detalles de cada argumento:

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

Por ejemplo, teníamos una hoja de datos con los resultados de las pruebas de 10 alumnos (se llamaba *Datos.Pruebas*). Imaginemos que queremos exportarlos en un fichero llamado *Pruebas.txt*, con los datos separados por comas y con los nombres de las variables. El código sería:

```
write.table(Datos.Pruebas,file="Pruebas.txt",sep=",",row.names=FALSE,col.names=TRUE)
```

3.3.2. Mediante R Commander

Utilizaremos la opción del menú *Datos → Conjunto de datos activo → Exportar el conjunto de datos activo*. Las opciones pueden ser modificadas, pero las que aparecen por defecto son las más recomendables (ver Figura 3.5).

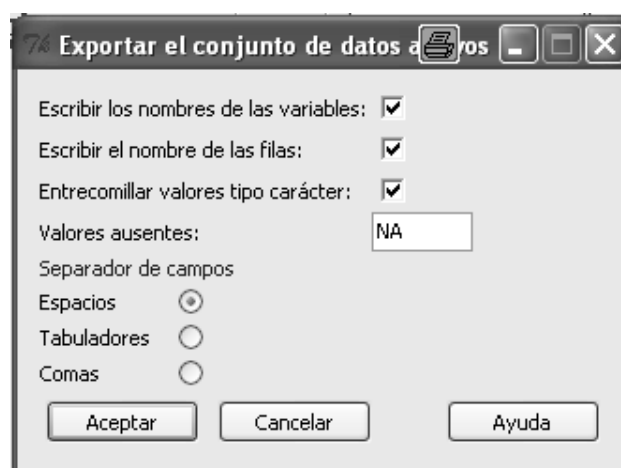


Figura 3.5: Exportar datos

3.4. Recodificación de variables

Recodificar una variable consiste en construir una nueva variable mediante la transformación de los valores de otra variable. Esta recodificación es bastante útil en muchas aplicaciones. En este apartado vamos a ver un par de ejemplos que nos ayudarán a ver cuál es el procedimiento a seguir.

3.4.1. Recodificación de una variable numérica

En el archivo *Jaencordoba.xls* aparece la población de los municipios de Jaén y Córdoba junto con el código que el INE le asigna a cada uno de ellos, código que coincide con parte del código postal. Si queremos analizar por separado los municipios de ambas provincias, ¿cómo podemos distinguir los que son de Córdoba de los que son de Jaén? El código del INE nos puede ayudar, ya que todos los códigos de Jaén empiezan por 23 y todos los de Córdoba por 14. Por tanto, la condición que identifica a un municipio de Jaén mediante su código postal es que éste es mayor que 23000 y menor que 24000, mientras que en Córdoba esta condición es ser mayor que 14000 y menor que 15000.

Tenemos así que podremos distinguir los municipios de las dos provincias si construimos una nueva variable a partir del código postal con esas dos condiciones.

3.4.1.1. Mediante código. Función *recode* del paquete *car*

En la sintaxis de la función `recode()`, incluida en el paquete `car` debemos especificar cuál es la variable a recodificar, cuáles son los criterios de recodificación y si deseamos que la nueva variable resultante de la recodificación sea un factor o no. Analicemos la forma de hacerlo sobre el ejemplo. Vamos a suponer que el nombre de la hoja de datos es *Datos* y que la variable que contiene los

códigos del INE se llama *ine*. La sintaxis sería la siguiente:

```
library(car)#Para cargar el paquete
Datos$Provincia<-recode(Datos$ine,
"14000:14999='Córdoba' ; 23000:23999='Jaén'", as.factor.result=TRUE)
```

- En primer lugar hemos escrito el nombre de la nueva variable, *Provincia*. Como queremos que sea una nueva variable de la hoja de datos *Datos*, hemos escrito *Datos\$Provincia*.
- Como primer argumento de `recode()` hemos escrito la variable que vamos a recodificar, *Datos\$ine*.
- A continuación hemos especificado los criterios de recodificación. Deben estar siempre entre comillas⁸. Si hay más de uno, que es lo más habitual, deben estar separados por puntos y comas. Finalmente, los nombres de los niveles de las variables recodificadas deben estar entre comillas simples⁹.

En el ejemplo queremos que todos los códigos entre 14000 y 15000 sean asignados a Córdoba. Escribir `14000:14999` es una forma rápida de generar el vector con todos los enteros entre 14000 y 14999. Así, el primer criterio de recodificación es `14000:14999='Córdoba'`.

- Finalmente, `as.factor.result=TRUE` indica que la nueva variable *Provincia* será un factor. Eso es conveniente para futuros análisis estadísticos.

3.4.1.2. Mediante R Commander

Vamos a ver cómo se hace mediante R commander.

Importamos en primer lugar el archivo. Seleccionamos la opción *Datos → Modificar variables del conjunto de datos activo → Recodificar variables*. Nos aparece la ventana de la Figura 3.6. En ella ya están incluidas las entradas necesarias para nuestra recodificación:

1. La variable a recodificar: *ine*.
2. El nombre de la nueva variable: *provincia*.
3. Las condiciones que determinan la recodificación. Observad que, como comentábamos, para especificar todos los números entre un valor *a* y otro valor *b* se debe poner *a : b*. Por otra parte, como queremos que los nuevos valores sean caracteres (Jaén, Córdoba), deben escribirse entre comillas.

⁸Las situadas en el 2 de nuestros teclados.

⁹Las situadas justo a la derecha del 0 de nuestros teclados.



Figura 3.6: Recodificando una variable numérica

4. La opción *Convertir cada nueva variable en factor* se deja activada para que la variable recodificada sea considerada como un factor.

3.4.2. Recodificación de una variable tipo carácter

La única diferencia con la recodificación que hemos descrito de una variable numérica es que los valores de la variable cualitativa o de tipo carácter deben escribirse entre comillas en las instrucciones para la recodificación.

Por ejemplo, si deseamos convertir la variable género, que es cualitativa, con valores “Hombre” y “Mujer”, en una variable numérica tipo factor con valores 0 (para los hombres) y 1 (para las mujeres), deberíamos poner "Hombre"=0 y "Mujer"=1 en los criterios de recodificación de `recode()` o en la ventana *Introducir directrices para la recodificación* de R Commander.

3.5. Cálculo de nuevas variables

En ocasiones es necesario realizar cálculos sobre las variables del conjunto de datos durante la realización de su análisis estadístico.

Por ejemplo, el archivo *JaenUsosSuelo.xls* contiene la distribución del suelo de los 96 municipios de la provincia de Jaén según su cobertura vegetal. Concretamente, para cada municipio se tienen 13 variables, que consisten en la extensión, en hectáreas, de suelo del municipio cubierto de: barbecho y otras tierras, cultivos herbáceos, cultivos leñosos, prados naturales, pastizales, monte maderable, monte abierto, monte leñoso, erial a pastos, espartizales, terreno improductivo, superficie no agrícola

y ríos y lagos¹⁰.

Lo que pretendemos es analizar el porcentaje de suelo de los municipios de la provincia de Jaén cubierto de monte. Fijémonos bien que decimos *porcentaje*, ya que debemos de tener en cuenta que Andújar, uno de los municipios más extensos de la provincia, puede tener una mayor extensión de montes que, por ejemplo, Campillo de Arenas, pero habrá que considerar, no la extensión en términos absolutos de los montes, sino la proporción (o porcentaje) de éstos sobre la extensión total.

Es decir, lo que nos interesa analizar no está explícitamente en los datos de la matriz original de datos, pero podemos calcularlo a partir de ella. Concretamente, lo que nos interesa analizar es la variable

$$\% \text{ monte} = \frac{\text{Monte maderable} + \text{Monte abierto} + \text{Monte leñoso}}{\text{Extensión total}} \times 100 \%,$$

expresada en porcentaje, donde *Extension.total* se obtiene, a su vez, como suma de todas las variables presentes en la matriz de datos.

3.5.1. Mediante código

Hemos importado la hoja de datos desde el archivo de Excel, llamándola *Datos*. Para ver el nombre de las variables, escribimos en la consola *names(Datos)*. El resultado es el siguiente:

```
> names(Datos)

[1] "Municipio" "Barbecho y otras tierras" "Cultivos herbáceos" "Cultivos leñosos"
[5] "Prados naturales" "Pastizales" "Monte maderable" "Monte abierto"
[9] "Monte leñoso" "Erial a pastos" "Espartizales" "Terreno improductivo"
[13] "Superficie no agrícola" "Ríos y lagos"
```

Como podemos ver, los nombres de las variables que se han importado desde el archivo de Excel incluyen caracteres como espacios y tildes, por lo que habrá que escribirlos entre comillas, tal y como aparecen en la consola de R.

Lo que queremos hacer en primer lugar es calcular la extensión total de cada municipio sumando todas las variables referidas a la extensión según los distintos tipos de uso del suelo. Recordemos que para referirnos a una variable de una hoja de datos debemos escribir primero el nombre de la hoja de datos, seguido del símbolo *\$* y del nombre de la variable. Vamos a llamar a la variable *Extension.total*. Se calcula de la siguiente forma:

¹⁰Fuente: Instituto de Estadística y Cartografía de Andalucía.

```
Datos$Extension.total<-Datos$"Barbecho y otras tierras"  
+Datos$"Cultivos herbáceos"+Datos$"Cultivos leñosos"  
+Datos$"Prados naturales"+Datos$"Pastizales"  
+Datos$"Monte maderable"+Datos$"Monte abierto"  
+Datos$"Monte leñoso"+Datos$"Erial a pastos"  
+Datos$"Espartizales"+Datos$"Terreno improductivo"  
+Datos$"Superficie no agrícola"+Datos$"Ríos y lagos"
```

Lo que R hace es sumar elemento a elemento los datos correspondientes a cada municipio de la provincia de Jaén. Desde luego, hay una forma mucho más fácil:

```
Datos$Extension.total<-rowSums(Datos[,2:14])
```

Lo que se hace es sumar por filas, todas las variables, de la 2ª a la 14ª, que se refieren a la extensión del municipio.

Una vez calculada la extensión total, vamos a calcular el porcentaje de terreno de cada municipio cubierta de monte, de la siguiente forma:

```
Datos$prop.monte<-100*(Datos$"Monte maderable"  
+Datos$"Monte abierto"+Datos$"Monte leñoso")  
/Datos$Extension.total
```

En general, podemos realizar las operaciones aritméticas habituales (suma, diferencia, multiplicación, división, potencias, etc.), con la notación matemática propia de estas operaciones, sin olvidar que tales operaciones se realizan elemento a elemento sobre todos y cada uno de los elementos que constituyen cada variable de la hoja de datos.

3.5.2. Mediante R Commander

Una vez importados los datos, utilizamos la opción del menú *Datos → Modificar variables del conjunto de datos activo → Calcular una nueva variable*. Esto nos abrirá una ventana que funciona básicamente como una calculadora. Lo que vamos a hacer es calcular en un primer paso la extensión total, *Extension.total*, y en un segundo paso la variable *prop.montes*. La Figura 3.7 muestra ambas entradas. En el cálculo de *prop.montes* no se visualiza todo el contenido de la ventana, pero en el interior aparece lo siguiente: `100*(Monte.abierto+ Monte.leñoso+ Monte.maderable)/Extension.total`.

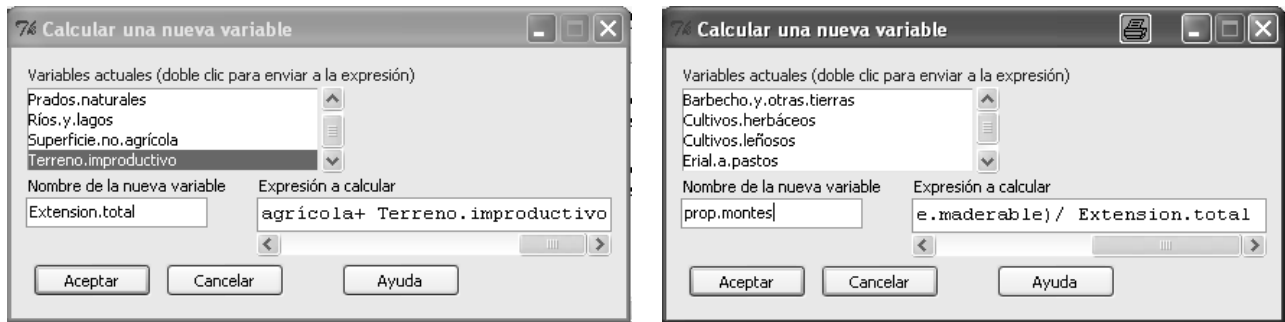


Figura 3.7: Cálculo de la extensión total del municipio (izquierda) y del porcentaje cubierto de monte (derecha)

3.6. Filtrado de datos

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

Por ejemplo, supongamos que en el archivo de datos contenido en *JaenCordoba.xls* deseamos analizar sólo los datos de los municipios de Jaén.

3.6.1. Mediante código

No olvidemos que la forma de referirnos a los elementos de una variable que conocemos hasta ahora es poniendo la posición que ocupan entre corchetes. Ahora vamos a hacer algo parecido, pero escribiendo entre los corchetes la condición que determina el filtro. Por ejemplo, en el caso que nos ocupa, deseamos quedarnos sólo con los datos relativos a la provincia de Jaén, es decir, deseamos sólo las filas de la provincia de Jaén, y todas las columnas (variables) de la hoja de datos. En ese caso, utilizando la variable *Provincia* que ya construimos, podemos hacerlo de la siguiente forma:

```
Datos.Jaen<-Datos[Datos$Provincia=="Jaén",]
```

o también, utilizando el código INE directamente,

```
Datos.Jaen<-Datos[(Datos$ine>23000)&(Datos$ine<24000),].
```

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

En resumen, el filtrado se logra identificando las filas que deseamos conservar mediante una expresión lógica.

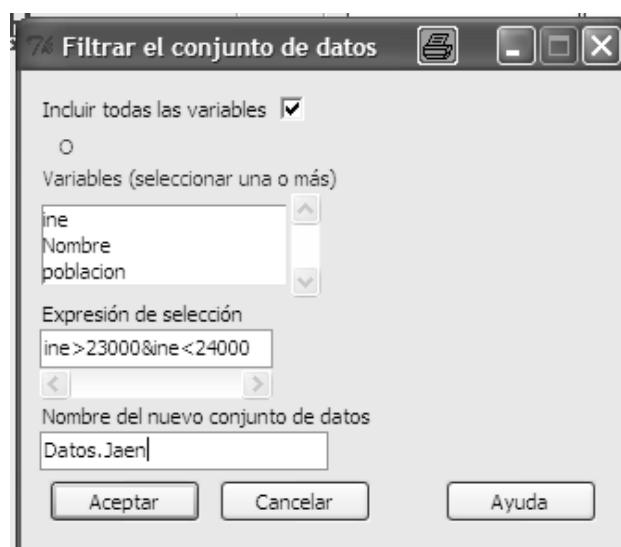


Figura 3.8: Filtrado de casos

3.6.2. Mediante R Commander

Lo haremos de la siguiente forma:

1. Seleccionando en el menú *Datos* → *Conjunto de datos activo* → *Filtrar el conjunto de datos activo*.
2. En la ventana emergente (Figura 3.8) podemos seleccionar si deseamos quedarnos con todas las variables o elegir sólo algunas.
3. La casilla más importante es la de *Expresión de selección*: ahí debemos escribir la expresión lógica que determine nuestro filtro. Por ejemplo, en nuestro caso podría ser `ine>23000&ine<24000`, es decir, código INE mayor que 23000 y menor que 24000. Si tuviéramos la variable Provincia que recodificamos anteriormente en este conjunto de datos, la expresión sería `Provincia=="Jaén"`.
4. Finalmente, es recomendable ponerle un nombre al nuevo conjunto de datos filtrado distinto del original, para evitar que lo sobrescriba. En nuestro caso lo he llamado `Datos.Jaen`.

3.7. Almacenamiento de instrucciones y resultados

3.7.1. Mediante código. El *script* y la *sesión de trabajo*

Ya hemos comentado (ver sección 1.3) que la forma más eficiente de trabajar con R es mediante un script del editor, en el que debemos ir introduciendo todos los pasos de nuestro trabajo. Sin

Figura 3.9: Ejemplo de *script*

embargo, puede que hasta ahora no hayamos tenido un ejemplo que ponga de manifiesto cómo esto, en efecto, facilita nuestro trabajo.

Vamos a retomar todo lo que hemos hecho hasta ahora con el archivo *JaenCordoba.xls*:

1. Lo importamos.
2. Obtuvimos una nueva variable, *Provincia*, recodificando la variable *ine*.
3. Nos quedamos sólo con los datos de Jaén, mediante un filtro.

Un *script* que recogiera todo ese trabajo sería el que aparece en la Figura 3.9.

Observemos que aparecen algunas líneas con comentarios aclaratorios, anteceditas del símbolo *#*, para que R las imprima en la consola, pero no las ejecute. Podemos observar en la parte superior de la ventana del *script* que lo hemos guardado llamándolo *JaenCordoba.R*. Ésta, *.R* (o *.r*), es la extensión propia de los script de R. De esta forma, en el futuro podríamos seguir trabajando con este ejemplo sin necesidad de enpezar todo de nuevo.

De la misma forma que podemos estar interesados en guardar el *script* con todas las instrucciones de un análisis, también podríamos estar interesados en guardar los objetos que se van generando en dichos análisis. Por ejemplo, los conjuntos de datos, pero también cualquier resultado que obtengamos y que pueda ser almacenado poniéndole un nombre. Todos esos objetos que se van generando constituyen lo que en R se conoce como la *sesión de trabajo*. Por ejemplo, tras ejecutar el *script* que acabamos de comentar, los únicos objetos que hay en la sesión de trabajo son *conexion*, *Datos* y *Datos.Jaen*.

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

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

3.7.2. Mediante R Commander

Vamos a ejemplificarlo con el ejemplo que antes hemos tratado sobre el archivo *JaenUsosSuelo.xls*. Debemos importarlo de nuevo y crear las variables *Extension.total* y *prop.montes*.

A continuación seleccionamos en el menú *Fichero* → *Guardar las instrucciones*. Nos pedirá el nombre y la ruta donde guardar el fichero de instrucciones, que tendrá extensión *.R*. Una buena idea para nombrar los ficheros de instrucciones es ponerles como nombre la fecha del día, por ejemplo, *21_02_12*. No es necesario escribir la extensión (pero tampoco la borremos): lo hará el propio programa. Podemos y debemos seguir guardando las instrucciones con posterioridad, eligiendo de nuevo *Guardar las instrucciones*, pero ya no nos pedirá de nuevo un nombre, a no ser que elijamos *Guardar las instrucciones como*.

Ahora vamos a reiniciar R Commander y volvemos a cargar el fichero *JaenUsosSuelo.xls*. A continuación elegimos en el menú *Fichero* → *Abrir fichero de instrucciones* y seleccionamos el fichero de instrucciones que antes hemos guardado. Como podemos ver, aparecen las dos líneas con las que hemos creado las variables *Extension.total* y *prop.montes*. Podemos ejecutar estas líneas directamente desde la ventana de instrucciones sin tener que utilizar el menú.

De igual forma, podemos guardar y recuperar con posterioridad todo lo que aparece en la ventana de resultados, mediante la opción *Fichero* → *Guardar los resultados*. Los ficheros de resultados que R Commander crea son ficheros de texto, con extensión *.txt*.

Capítulo 4

Estadística descriptiva

Objetivos:

1. Obtener medidas de posición, dispersión y forma de un conjunto de datos.
2. Obtener representaciones gráficas que resuman desde el punto de vista estadístico un conjunto de datos.
3. Detectar valores fuera de rango en un conjunto de datos.

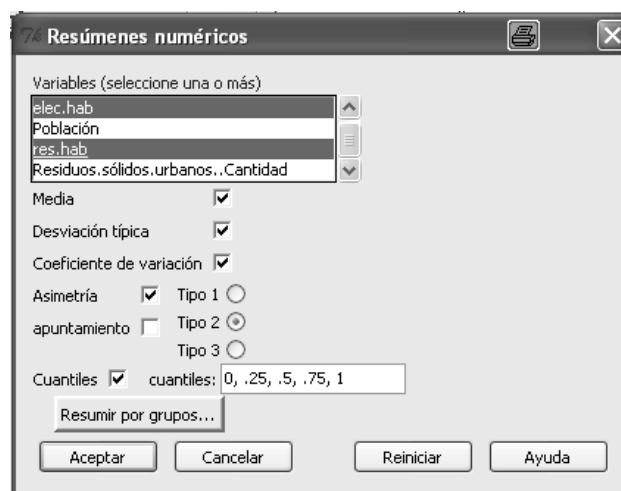
- A lo largo del capítulo vamos a utilizar el conjunto de datos *JaenIndicadores.rda* para todos los ejemplos y explicaciones.

4.1. Cálculo de medidas de posición, dispersión y forma

4.1.1. Mediante R Commander

Las medidas de posición, dispersión y forma más comunes, media, mediana, percentiles, desviación típica y coeficiente de asimetría, se hallan en la opción del menú *Estadísticos* → *Resúmenes* → *Resúmenes numéricos*. A modo de ejemplo, vamos a obtener estas medidas para los promedios *elec.hab*, *agua.hab* y *res.hab*. Estas variables se refieren al consumo de agua por habitante ($m^3/día$), al consumo eléctrico por habitante (megawatios/hora) y a la cantidad de residuos sólidos urbanos (toneladas) por habitante en los municipios de la provincia de Jaén en un año. En la Figura 4.1 aparecen las entradas de esa opción del menú. En ella es posible elegir varias variables a la vez, pulsando la tecla *Control* mientras se clican en las variables deseadas.

En el Cuadro 4.1 se muestra la tabla de resultados que aparece. En ésta, *mean* se refiere a la media, *sd* a la raíz de la cuasi-varianza, *skewness* es el coeficiente de asimetría, el percentil 0 es el valor

Figura 4.1: Opción *Resúmenes numéricos* del menú

mínimo de la variable, el percentil 50, como ya sabemos, es la mediana y el percentil 100 es el valor máximo de la variable.

	mean	sd	CV	skewness	0 %	25 %	50 %	75 %	100 %
agua.hab	0.53	0.14	0.26	1.22	0.14	0.46	0.51	0.56	1.09
elec.hab	2.66	1.40	0.53	2.05	0.94	1.75	2.20	3.10	9.19
res.hab	0.23	0.04	0.16	1.39	0.18	0.21	0.23	0.24	0.35

Cuadro 4.1: Descriptivos básicos de *elec.hab*, *agua.hab* y *res.hab*

4.1.2. Mediante código

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

```
mean(Datos$agua.hab,na.rm=TRUE)
```

```
mean(Datos$elec.hab,na.rm=TRUE)
```

```
mean(Datos$res.hab,na.rm=TRUE)
```

O bien, las tres medias juntas mediante

```
mean(Datos[,c("agua.hab","elec.hab","res.hab")],na.rm=TRUE)
```

El argumento *na.rm = TRUE* indica que los valores faltantes *NA* se eliminan para realizar los cálculos. Si no se incluye esta opción entonces la media resultante sería *NA*.

De igual forma, la desviación típica se lograría mediante

```
sd(Datos$agua.hab,na.rm=TRUE)
```

```
sd(Datos$elec.hab, na.rm=TRUE)
sd(Datos$res.hab, na.rm=TRUE)
```

O bien,

```
sd(Datos[,c("agua.hab", "elec.hab", "res.hab")], na.rm=TRUE).
```

El coeficiente de variación se obtendría como el cociente entre la desviación típica y la media.

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

```
quantile(Datos$agua.hab, na.rm = TRUE, probs=c(0.05, 0.95))
quantile(Datos$elec.hab, na.rm = TRUE, probs=c(0.05, 0.95))
quantile(Datos$res.hab, na.rm = TRUE, probs=c(0.05, 0.95))
```

El coeficiente de asimetría no viene definido en el paquete *base* que se carga por defecto con R, de manera que debemos buscarlo. Si ponemos `??skewness` podremos comprobar cómo hay un paquete llamado `e1071` que tiene una función con ese nombre. Si lo cargamos y vemos su ayuda (`?skewness`), comprobaremos que, en efecto, lo que hace esa función es calcular el coeficiente de asimetría. Por tanto,

```
library(e1071)
skewness(Datos$agua.hab, na.rm=TRUE)
skewness(Datos$elec.hab, na.rm=TRUE)
skewness(Datos$res.hab, na.rm=TRUE)
```

4.1.3. Resúmenes por grupos

¿Y si deseamos hacer el mismo análisis pero en cada uno de los grupos que determina la variable *Tipo*¹? Tengamos en cuenta que esta variable es un factor, así que podemos utilizarla para ello.

Veámoslo en primer lugar mediante R Commander. Utilizamos la misma opción *Estadísticos* → *Resúmenes* → *Resúmenes numéricos*, pero ahora clicamos en la pestaña *Resumir por grupos*. Esta opción abre una ventana (Figura 4.2) donde aparecen como posibilidades todas aquellas variables que pueden dividir al conjunto de datos en grupo. En nuestro caso elegimos la variable *Tipo*. La ventana de resultados mostrará los mismos estadísticos, pero separando cada uno de los tres grupos, para cada variable. Por ejemplo, para la variable *agua.hab* tenemos

¹Esta variable toma el valor *Pequeño* si el municipio es de menos de 2000 habitantes, *Mediano* si el municipio tiene entre 2000 y 4000 habitantes y *Grande* si tiene más de 4000 habitantes.

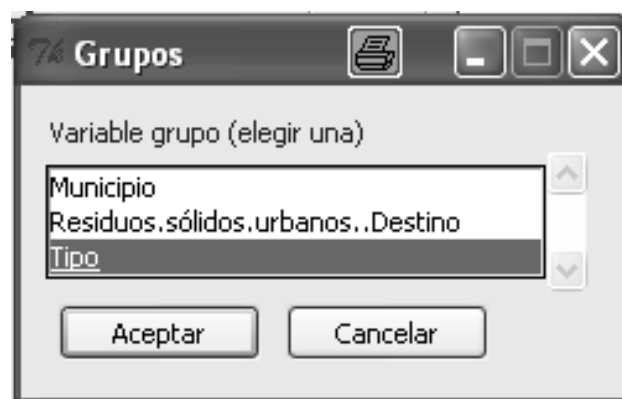


Figura 4.2: Resúmenes por grupos

	mean	sd	CV	skewness	0 %	25 %	50 %	75 %	100 %
Grande	0.51	0.12	0.18	1.45	0.18	0.47	0.51	0.54	1.01
Mediano	0.51	0.07	0.10	0.23	0.41	0.45	0.51	0.57	0.65
Pequeño	0.55	0.19	0.11	0.70	0.14	0.46	0.49	0.66	1.09

Cuadro 4.2: Descriptivos básicos de *agua.hab* en función de *Tipo*

Mediante código, los anteriores resultados se pueden calcular a través de la orden `tapply()`. Por ejemplo, la media, la desviación típica y los percentiles 5 y 95 de la variable *agua.hab* en función de los grupos de la variable *Tipo* se obtendría de la forma siguiente:

```
tapply(Datos$agua.hab,Datos$Tipo,mean, na.rm=TRUE)
```

```
tapply(Datos$agua.hab,Datos$Tipo,sd, na.rm=TRUE)
```

```
tapply(Datos$agua.hab,Datos$Tipo,quantile,probs=c(0.05,0.95),na.rm=TRUE)
```

Obsérvese que en la última línea hemos tenido que especificar las probabilidades requeridas a la función *quantile*.

4.2. Distribuciones de frecuencias

Las variables *Tipo* y *Residuos.sólidos.urbanos..Destino* son de tipo cualitativo, por lo que no pueden ser resumidas mediante medidas numéricas. Para este tipo de variables el resumen más conveniente es, simplemente, su distribución de frecuencias.

4.2.1. Mediante R Commander

Para obtener la distribución de frecuencias de una o varias variables de un conjunto de datos mediante R Commander elegimos la opción *Estadísticos* → *Resúmenes* → *Distribución de frecuencias*. En la ventana emergente elegimos las variables que queremos analizar y la tabla aparece en la

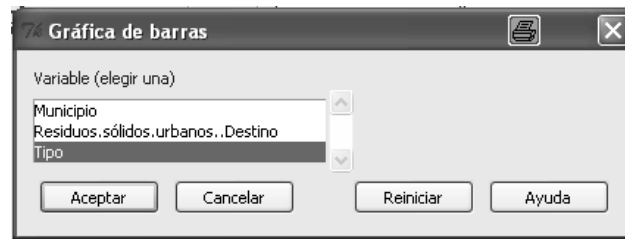


Figura 4.3: Entradas para la construcción de un diagrama de barras

ventana de resultados, incluyendo las frecuencias absolutas y relativas.

4.2.2. Mediante código

Las funciones de código correspondientes son `table()` y `prop.table()`. Así, para la obtención de las distribuciones de frecuencias (absolutas y relativas) de la variable *Tipo* escribimos

```
Tabla <- table(Datos$Tipo)
Tabla # frecuencias absolutas
prop.table(Tabla)# frecuencias relativas
```

4.3. Diagrama de barras y diagrama de sectores

No obstante, asumiendo el dicho *una imagen vale más que mil palabras*, sabemos que existen dos formas de plasmar en un gráfico la distribución de frecuencias de una variable cualitativa o discreta con pocos valores: el diagrama de barras y el diagrama de sectores.

4.3.1. Diagrama de barras para variables cualitativas

En R Commander este tipo de gráficos están en la opción *Gráficas* → *Gráfica de barras*. La ventana de entradas aparece en la Figura 4.3. En esta ventana hemos solicitado un análisis de la variable *Tipo*. Es muy importante tener en cuenta que sólo pueden representarse variables cualitativas de tipo factor.

La función `barplot()` nos permite obtener la distribución de barras mediante código. Aquí tenemos posibilidad de controlar más cosas. Por ejemplo, es interesante añadir las frecuencias exactas al gráfico o retocar los títulos de los ejes y del gráfico. La sintaxis para obtener la Figura 4.4 es la siguiente:

```
diagrama<-barplot(Tabla,col=rainbow(3)
,xlab="Municipios según su Tipo",ylab="Frecuencias absolutas")
```

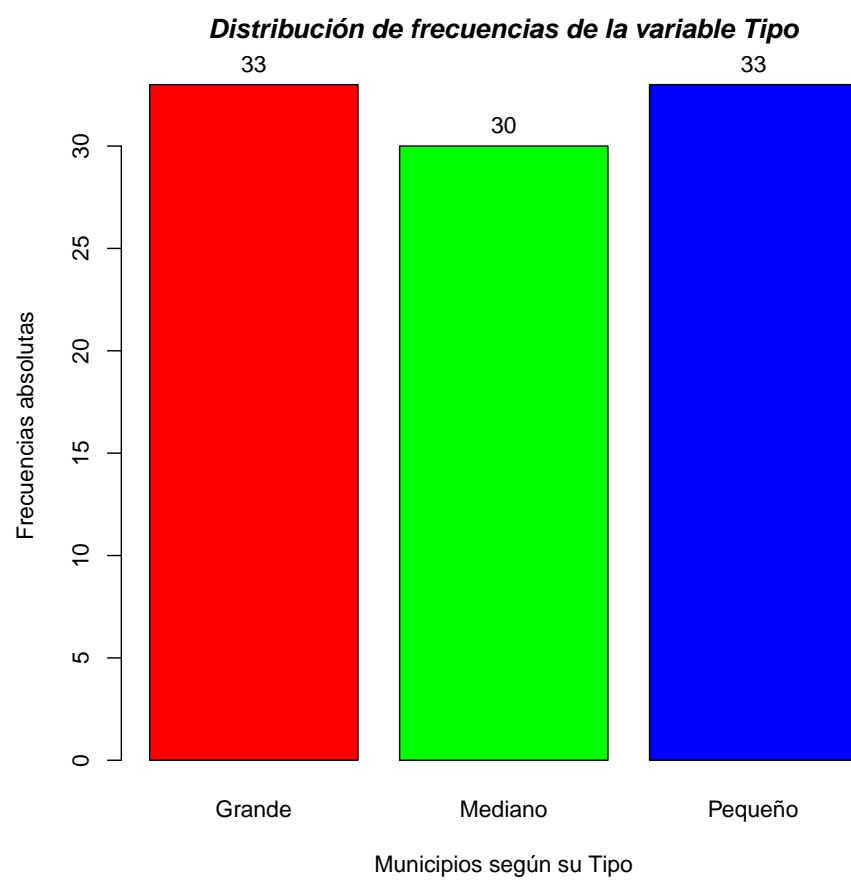



Figura 4.4: Diagrama de barras según el tipo de municipio del destino de los residuos sólidos

Distribución de porcentajes de la variable Destino de los residuos sólidos urbano

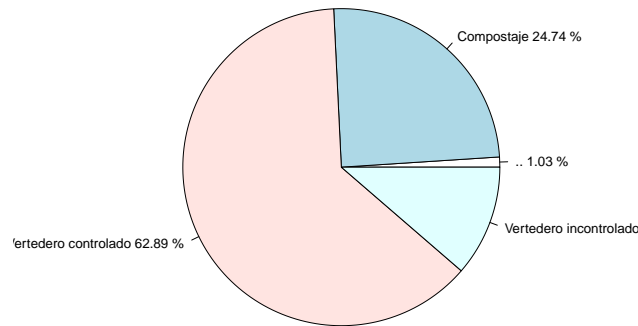


Figura 4.5: Diagrama de sectores del destino de los residuos sólidos urbanos en los municipios de la provincia de Jaén

```
text(diagrama,Tabla + 1,labels=Tabla, xpd = TRUE)
```

```
title(main = "Distribución de frecuencias de la variable Tipo", font.main = 4)
```

Existen otras muchísimas posibilidades en la construcción del gráfico mediante código: aquí sólo mostramos aspectos de su sintaxis más básica.

4.3.2. Diagrama de sectores para variables cualitativas

Para realizar un diagrama de sectores mediante R Commander elegiremos la opción *Gráficas* → *Diagrama de sectores*. La ventana emergente sólo permite elegir una variable cualitativa. De nuevo es muy importante tener en cuenta que sólo pueden representarse variables cualitativas de tipo factor. El diagrama correspondiente al destino de los residuos sólidos de los municipios aparece en la Figura 4.5, aunque se ha retocado mediante código, como se describe más a continuación.

La opción mediante código en su versión más básica es

```
pie(table(Datos$Residuos.sólidos.urbanos..Destino))
```

pero podemos mejorar ese gráfico añadiendo los porcentajes y un título ilustrativo, de la siguiente forma:

```
tabla.destino<-prop.table(table(Datos$Residuos.sólidos.urbanos..Destino))
```

```
tabla.destino<-round(100*tabla.destino,2)
```

```
sectores.destino<-pie(tabla.destino,
```

```
labels=paste(names(tabla.destino),tabla.destino,"%"),
```

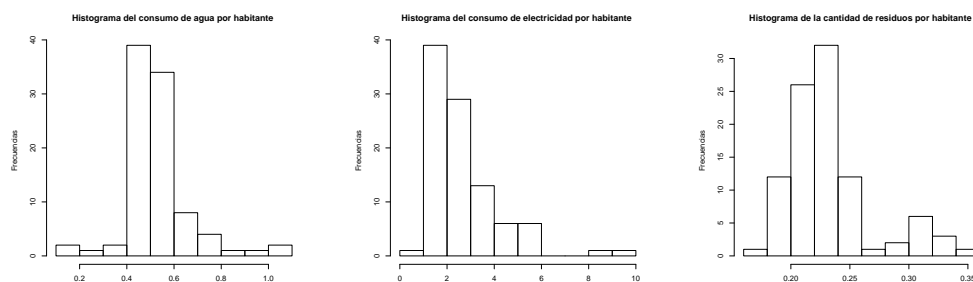


Figura 4.6: De izquierda a derecha, histogramas de *agua.hab*, *elec.hab* y *res.hab*

```
main="Distribución de porcentajes de la variable
```

```
Destino de los residuos sólidos urbanos")
```

Observemos que la función `paste()` nos ha ayudado a *pegar* las etiquetas de la variable con el porcentaje y el símbolo “%”, que luego hemos añadido a los sectores del gráfico.

4.4. Histograma para variables continuas y discretas

4.4.1. Histograma para variables continuas

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

Para realizar un histograma con R Commander elegimos *Gráficas* → *Histograma*. La ventana de entrada permite elegir sólo una variable para cada análisis, el número de intervalos del histograma y la escala de éste (frecuencias absolutas, porcentajes y densidades).

En el caso de las variables *agua.hab*, *elec.hab* y *res.hab* hemos seleccionado histogramas con escala en porcentajes. Los resultados aparecen en la Figura 4.6.

La función `hist()` nos permite representar el histograma mediante código. La sintaxis básica de esta función es la siguiente:

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

- *x* es el vector de datos.
- *breaks* puede especificar el número de intervalos que deseamos o los extremos de los intervalos que deseamos considerar, mediante un vector. Por defecto, asigna el número de intervalos por el conocido como método de Sturges.

- *freq* especifica si la escala del histograma es tal que el área de las barras es igual a la proporción de datos en cada intervalo (escala de densidad, con valor `freq = FALSE`) o su altura es simplemente el recuento de las frecuencias (escala de frecuencias, con valor `freq = TRUE`).
- *main* especifica el título del gráfico, mientras que *xlab* e *ylab* especifican el de los ejes.
- *labels* añade una etiqueta a cada barra con el valor de las frecuencias.

Por ejemplo, para calcular el histograma de frecuencias de la variable *agua.hab* debemos escribir

```
hist(Datos$agua.hab, freq = TRUE,  
main = "Histograma del consumo de agua por habitante ",  
xlab="", ylab="Frecuencias")
```

4.4.2. Histograma para variables discretas

Parece contradictorio hablar de un histograma para variables discretas, ya que una variable discreta debe representarse con un diagrama de barras. Sin embargo, para los ordenadores y los programas estadísticos, la división entre variables discretas y continuas suele no ser habitual. Se entiende que ambas son variables numéricas, al contrario de las variables cualitativas, que son caracteres, y para variables numéricas se ofrece el histograma como representación gráfica.

Lo que podemos hacer es utilizar la función que realiza el histograma para realizar un diagrama de barras de una variable discreta. La idea es muy simple: le pediremos a R a través de Commander o de la consola el histograma de la variable discreta de manera que las barras del histograma representen las frecuencias de los valores de la variable discreta.

Vamos a verlo con los datos que están contenidos en el archivo *DatosMuertesCoces.rda*. Estos datos, de los que hablaremos más adelante se refieren al número de muertos por coces de caballos en el ejército prusiano en una serie de años y secciones a finales del siglo XIX. Es evidente que es una variable discreta. Para realizar un diagrama de barras de estos datos, haremos lo siguiente:

1. Le pediremos a R Commander un histograma de la variable muertes, eligiendo como escala el recuento de frecuencias. No hace falta retocar el número de intervalos porque lo vamos a hacer a mano a continuación.
2. En la ventana de instrucciones R Commander ha escrito lo siguiente:

```
Hist(Datos.muertes$Muertes,scale="frequency",breaks="Sturges",col="darkgray")
```

Además, veremos como resultado el histograma que aparece en la Figura 4.7 a la izquierda.

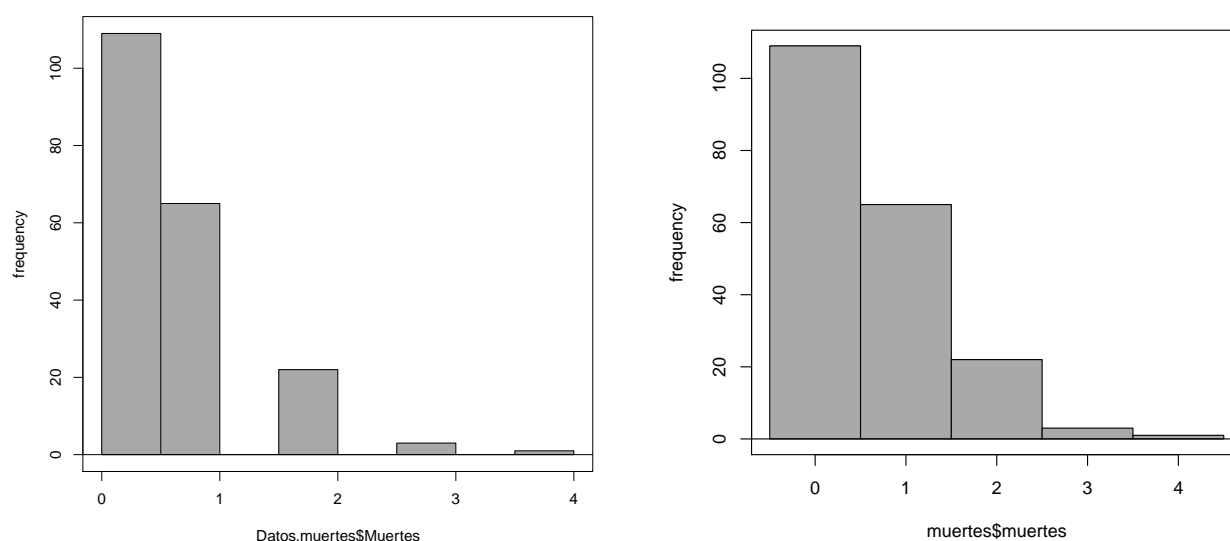


Figura 4.7: Histogramas de una variable discreta

3. Cerraremos ese histograma y retocaremos el código de la siguiente forma. Como deseamos que las barras del histograma cuenten las frecuencias de los valores de la variable, nos fijaremos en que éstos van de 0 a 4, y en la opción `breaks` escribiremos `-0.5:4.5`: de esa forma, los intervalos del histograma contarán las frecuencias en los valores 0, 1, 2, 3 y 4. En resumen, el nuevo código será

```
Hist(Datos.muertes$Muertes,scale="frequency",breaks=-0.5:4.5,col="darkgray")
```

El resultado aparece en la Figura 4.7 a la derecha.

Mediante código es igual de sencillo. Vamos a verlo con otro ejemplo.

Consideremos los datos al azar que se crean con

```
datos<-rpois(100,2.5)
```

Vamos a representarlos adecuadamente mediante un histograma. Para ello, la clave radica en indicarle a la función `hist()` que considere los puntos de corte para los intervalos tales que, al final, lo que haga sea contar los valores enteros de la variable. Es bien sencillo: basta considerar

```
cortes<-(min(datos)-0.5):(max(datos)+0.5)
```

En mi caso, el valor de `cortes` fue de

```
[1] -0.5 0.5 1.5 2.5 3.5 4.5 5.5 6.5
```

porque el mínimo de los datos fue 0 y el máximo 6. De esta forma, los cortes para los intervalos del histograma determinarán que, en realidad, se cuenten los 0, los 1, ..., así hasta el máximo. Para

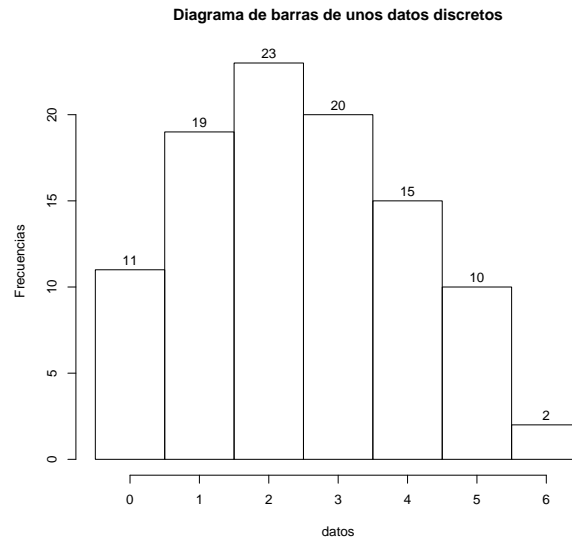


Figura 4.8: Histograma para representar unos datos discretos

ello,

```
hist(datos,breaks=cortes,freq=TRUE,labels=TRUE,ylab="Frecuencias",
main="Diagrama de barras de unos datos discretos")
```

El resultado es el que aparece en la Figura 4.8.

4.5. Detección de valores atípicos. Diagrama de caja

Un método común para la detección de valores atípicos es el llamado diagrama de caja o *boxplot*. Este método es válido para cualquier conjunto de datos, independientemente de la forma de su distribución de frecuencias.

4.5.1. Mediante R Commander

Vamos a obtener el diagrama de caja de las variables *elec.hab* y *res.hab* e identificar los municipios atípicos en cuanto al consumo de energía eléctrica y de los residuos generados por habitante. Para ello, elegimos la opción *Gráficas* → *Diagrama de caja*. En la ventana de entrada tenemos que elegir la variable que queremos analizar y existen dos opciones muy interesantes:

1. Podemos elegir un análisis por grupos, sin más que clicar en la pestaña *Resúmenes por grupos*. En esta ocasión no es necesario, pero podríamos hacerlo con los grupos generados por la variable *Tipo*.

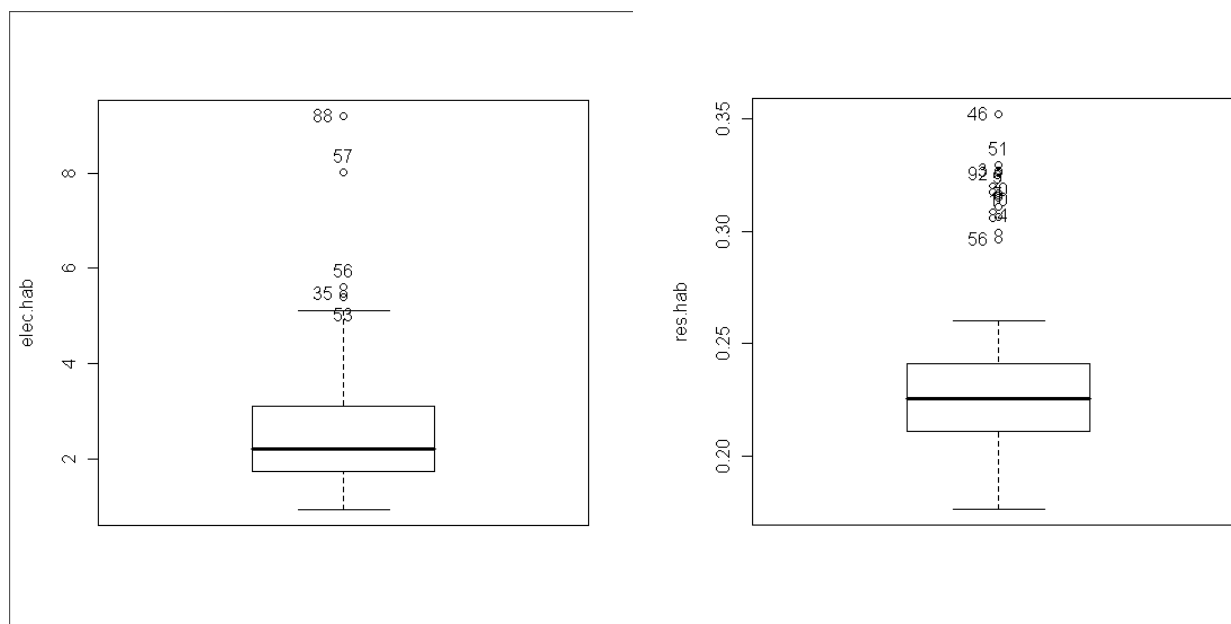


Figura 4.9: Diagramas de caja de las variables *elec.hab* (izquierda) y *res.hab* (derecha)

- Podemos elegir la opción *Identificar atípicos con el ratón*, que es la forma más fácil de señalar los municipios atípicos². Recordemos que en un diagrama de caja los municipios atípicos se señalan con círculos, y los atípicos extremos con asteriscos.

En la Figura 4.9, a la izquierda aparece el diagrama de caja de la variable *elec.hab*. En ella podemos ver gráficamente la asimetría a la derecha de la distribución, ya que el lado a la derecha de la mediana, que separa la caja, es más grande. Por su parte, también podemos ver que hemos detectado como municipios atípicos a la derecha de la distribución, es decir, que destacan por su fuerte consumo por habitante, a los municipios 35, 53, 56, 57 y 88 (estos dos últimos de forma muy clara), que corresponden con Guarromán, Lupión, Martos, Mengíbar y Villanueva de la Reina. Una forma eficiente de ver sus nombres es introducir el siguiente código en la ventana de instrucciones: `Datos$Municipio[c(35,53,56,57,88)]`.

4.5.2. Mediante código. La función `boxplot()`

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

```
boxplot(Datos$elec.hab)
```

También es posible añadir un título al gráfico y a los ejes, como en el caso de `hist()`.

²Sólo hay que tener un poquito de buen pulso cuando los valores están muy próximos entre sí.

Por su parte, si no cerramos el gráfico, también podemos especificar los atípicos mediante la función `identify()`. Vamos a verlo en el ejemplo:

```
identify(rep(1, length(Datos$elec.hab)),  
Datos$elec.hab, rownames(Datos))
```

El primer argumento, `rep(1, length(Datos$elec.hab))`, está especificando que la coordenada x de todos los puntos es 1 (lugar donde se sitúan todos los valores). `Datos$elec.hab` especifica la coordenada y de los puntos (ya que el diagrama es vertical), mientras que `rownames(Datos)` especifica cómo identificar los puntos, en este caso con el número de fila. Podríamos cambiar esta opción por `Datos$Municipio` y al clicar nos daría directamente el nombre del municipio (aunque si señalamos varios, probablemente se solaparán).

Capítulo 5

Manejo de distribuciones de probabilidad

Objetivos:

1. Calcular probabilidades asociadas a las distribuciones de probabilidad más habituales.
2. Calcular percentiles de las distribuciones de probabilidad más habituales.
3. Obtener representaciones gráficas de las distribuciones de probabilidad más habituales.
4. Simular muestras procedentes de las distribuciones de probabilidad más habituales.

5.1. Introducción

R proporciona una excelente manera de obtener probabilidades asociadas a las distribuciones de probabilidad más comunes. Debemos recordar que, incluso los modelos más sencillos, como el binomial o el de Poisson, presentan dificultades en cuanto a lo tedioso que resulta realizar los cálculos. En otras distribuciones, como la Gamma o la normal, el problema es que es imposible *hacer las cuentas a mano*, siendo absolutamente necesario la utilización de algún software matemático para obtener aproximaciones precisas de las probabilidades.

Vamos a ver a lo largo del capítulo cómo utilizar R como calculadora para todas estas cuestiones. Tengo que empezar diciendo que mi experiencia con R Commander en este apartado me hace sugerir que no se utilicen los menús, porque hacen cualquier paso bastante pesado. Además, estos menús suelen ser poco versátiles, sobre todo a la hora de calcular probabilidades. Por ello, todo el capítulo está explicado mediante código. Los usuarios que prefieran utilizar Commander pueden ejecutar este código desde la ventana de instrucciones.

Antes de comenzar, debemos tener claro que vamos a manejar 4 tipos de funciones:

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

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

- *binom* para la binomial.
- *pois* para la Poisson.
- *geom* para la geométrica.
- *nbinom* para la binomial negativa.
- *exp* para la exponencial.
- *gamma* para la Gamma.
- *norm* para la normal.

Evidentemente, son muchísimas más las distribuciones disponibles, si bien nos vamos a centrar en estas, que son las que habitualmente se consideran en un curso básico de Estadística.

Finalmente, los argumentos de cada una de las funciones tendrán que especificar, en cada caso, los parámetros concretos de la distribución, y determinar qué probabilidad o qué cuantil queremos, o incluso cuántos datos simulados necesitamos. El Cuadro 5.1 desarrolla todas las posibilidades y explicita cuáles son los argumentos de cada función.

5.2. Cálculo de probabilidades

5.2.1. Distribuciones discretas

En el caso de las distribuciones discretas, tenemos, básicamente, dos tipos de probabilidades:

1. Las probabilidades que proporciona la función masa, que podríamos llamar *probabilidades simples*, del tipo $P[X = x]$.

	Densidad o masa	Función de distribución	Función cuantil	Muestras aleatorias
Binomial(n, p)	<code>dbinom(x, n, p)</code>	<code>pbinom(x, n, p)</code>	<code>qbinom($prob, n, p$)</code>	<code>rbinom($muestras, n, p$)</code>
Poisson(λ)	<code>dpois(x, λ)</code>	<code>ppois(x, λ)</code>	<code>qpois($prob, \lambda$)</code>	<code>rpois($muestras, \lambda$)</code>
Geométrica(p)	<code>dgeom(x, p)</code>	<code>pgeom(x, p)</code>	<code>qgeom($prob, p$)</code>	<code>rpois($muestras, p$)</code>
Binomial negativa(a, p)	<code>dnbinom(x, a, p)</code>	<code>pnbinom(x, a, p)</code>	<code>qnbinom($prob, a, p$)</code>	<code>rnbinom($muestras, a, p$)</code>
Exponencial(λ)	<code>dexp(x, λ)</code>	<code>pexp(x, λ)</code>	<code>qexp($prob, \lambda$)</code>	<code>rexp($muestras, \lambda$)</code>
Gamma(a, λ)	<code>dgamma(x, a, λ)</code>	<code>pgamma(x, a, λ)</code>	<code>qgamma($prob, a, \lambda$)</code>	<code>rgamma($muestras, a, \lambda$)</code>
Normal(μ, σ)	<code>dnorm(x, μ, σ)</code>	<code>pnorm(x, μ, σ)</code>	<code>qnorm($prob, \mu, \sigma$)</code>	<code>rnorm($muestras, \mu, \sigma$)</code>

Cuadro 5.1: Resumen de las funciones asociadas a las distribuciones y sus argumentos

Distribución	Probabilidad	Código	Resultado
<i>Poisson</i> (5.2)	$X = 3, 6, 9$	<code>sum(dpois(c(3,6,9),5.2))</code>	0.323
<i>Geo</i> (0.75)	$1 < X < 6$	<code>sum(dgeom(2:5,0.75))</code>	0.062
<i>BN</i> (1.2, 0.66)	$1 \leq X \leq 6$	<code>sum(dnbinom(1:5,1.2,0.66))</code>	0.390
<i>B</i> (12, 0.15)	$2 \leq X < 5$	<code>sum(dbinom(2:4,12,0.15))</code>	0.533
<i>Poisson</i> (5.5)	$X > 8$	<code>1-sum(dpois(0:8,5.5))</code>	0.106

Cuadro 5.2: Ejemplo de cálculo de probabilidades de distribuciones discretas

2. *Probabilidades acumuladas* (dadas en términos de la función de distribución), del tipo $P[X \leq x]$.

Es evidente que las probabilidades acumuladas se pueden calcular a partir de las probabilidades simples, sin más que tener en cuenta que $P[X \leq x] = \sum_{x_i \leq x} P[X = x_i]$. Por ello, de cara a simplificar las explicaciones, vamos a utilizar siempre la función masa para calcular probabilidades asociadas a variables discretas.

A modo de ejemplo, supongamos que tenemos una distribución binomial de parámetros $n = 10$ y $p = 0.25$ y deseamos calcular $P[X < 4]$. Entonces, dado que

$$P[X > 4] = P[X = 0, 1, 2, 3] = \sum_{x=0}^3 P[X = x],$$

sólo tenemos que calcular la probabilidad de 0, 1, 2 y 3 y sumarlas. Para ello ejecutamos

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

El Cuadro 5.2 recoge otros ejemplos.

5.2.2. Distribuciones continuas

En el caso de las distribuciones de tipo continuo sabemos que los valores concretos de la variable tienen probabilidad cero o, dicho de otra forma, no tienen masa de probabilidad, sino densidad de probabilidad. En estas variables no tiene sentido preguntarse por probabilidades del tipo $P[X = x]$

Distribución	Probabilidad	Código	Resultado
$exp(1.2)$	$X < 2.5$	<code>pexp(2.5, 1.2)</code>	0.950213
$Gamma(1.2, 3.8)$	$X = 1.5$	¿Para qué?	0
$Gamma(1.2, 3.8)$	$1.05 < X < 5$	<code>pgamma(5, 1.2, 3.8) - pgamma(1.05, 1.2, 3.8)</code>	0.028
$N(2.5, 1.1)$	$2.2 \leq X \leq 3$	<code>pnorm(3, 2.5, 1.1) - pnorm(2.2, 2.5, 1.1)</code>	0.283
$N(2.2, 0.5)$	$2.2 < X < 3$	<code>pnorm(3, 2.2, 0.5) - pnorm(2.2, 2.2, 0.5)</code>	0.371

Cuadro 5.3: Ejemplos de cálculo de probabilidades de distribuciones continuas

porque todas son cero. En su lugar, lo que nos preguntamos es por las probabilidades de que las variables proporcionen valores en intervalos, es decir, probabilidades del tipo $P[a < X < b]$, y donde las desigualdades pueden ser estrictas o no, ya que el resultado final no varía.

Siguiendo con este recordatorio, sabemos que las probabilidades del tipo $P[a < X < b]$ se calculan como

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

donde $f(x)$ es la función de densidad de la variable y $F(x) = \int_{-\infty}^x f(t) dt$ es una primitiva suya, conocida como función de distribución. En resumen, podremos calcular probabilidades del tipo $P[a < X < b]$ siempre que podamos obtener los valores de la función de distribución $F(x)$. Y recordemos que estas funciones de distribución en R son las que empiezan por la letra *p*.

Por ejemplo, si tuviéramos una distribución normal de media 5 y desviación típica 2, y quisiéramos calcular $P[2 < X < 7.6]$ lo haríamos teniendo en cuenta que

$$P[2 < X < 7.6] = \int_2^{7.6} f(x) dx = F(7.6) - F(2)$$

mediante

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

En el Cuadro 5.3 incluimos también algunos ejemplos con el código que los resuelve y el resultado.

5.3. Cálculo de cuantiles

Recordemos que los cuantiles son medidas de posición relativas. El cuantil p (siendo p un nº entre 0 y 1), Q_p , de una distribución de probabilidad es aquél que deja por debajo de sí una probabilidad p ; si, hipotéticamente, tuviéramos 100 datos, el cuantil p es el que, ordenados todos los valores de menor a mayor, ocuparía la posición $100p$. Eso es lo que permite interpretarlos como medidas de

posición relativas, ya que permite analizar si un dato es *alto, medio o bajo en su distribución*.

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

5.3.1. Distribuciones discretas

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

Por ejemplo:

- Si $X \rightarrow B(15, 0.65)$, $Q_{0.05} = 7$, lo que en R se consigue mediante `qbinom(0.05, 15, 0.65)`.
- Si $X \rightarrow Poisson(5.8)$, $Q_{50} = 6$, dado en R por `qpois(0.50, 5.8)`.

5.3.2. Distribuciones continuas

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

A modo de ejemplos:

- Si $X \rightarrow Gamma(1.2, 2.5)$, $Q_{0.05} = 0.037$, dado por `qgamma(0.05, 1.2, 2.5)`.
- Si $X \rightarrow N(0, 4.5)$, $Q_{0.95} = 7.402$, dado en R por `qnorm(0.95, 0, 4.5)`.

5.4. Representaciones gráficas de distribuciones de probabilidad.

La función `plot()`

Vamos a aprovechar la necesidad de comentar el código que permite representar gráficamente una distribución de probabilidad para describir una de las funciones más utilizadas en representaciones gráficas con R: la función `plot()`.

La función `plot()` ejecuta una simple representación de un conjunto de puntos caracterizados mediante coordenadas cartesianas. Su sintaxis básica es la siguiente:

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

En esta expresión,

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

Por tanto, la representación de una distribución de probabilidad mediante `plot()` tan sólo requerirá que le digamos *qué* queremos representar (en el eje Y; probabilidades simples o densidades, probabilidades acumuladas o cuantiles) y *dónde* (en el eje X). Las demás opciones sólo añadirán información al gráfico.

Por ejemplo, supongamos que deseamos representar la función de densidad de una distribución normal $N(\mu = 100, \sigma = 10)$:

1. Empecemos preguntándonos *dónde* queremos la representación. Dado que el 99 % de los valores de una normal están en un intervalo de su media menos/más tres desviaciones típicas, vamos a representarlo entre 70 y 130. Por su parte, vamos a elegir un número razonable de puntos, por ejemplo, 20. Con esto, podemos especificar que las coordenadas en el eje X de los puntos que queremos representar son

```
x<-seq(70,130,length.out=20).
```

2. Queremos representar la función de densidad, luego las coordenadas en el eje Y deben ser

```
y<-dnorm(x,100,10).
```

3. Finalmente, el código sería el siguiente, donde hemos añadido etiquetas y títulos:

```
x<-seq(70,130,length.out=20)
y<-dnorm(x,100,10)
plot(x,y,type="l",main="Densidad de una N(100,10)",
xlab="Valores de la variable",ylab="Densidad")
```

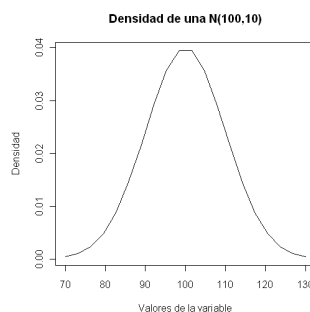


Figura 5.1: Densidad de una $N(100, 10)$

El resultado aparece en la Figura 5.1.

5.5. Simulación de muestras

Una de las aplicaciones más comunes de la Estadística es la de proporcionar un marco teórico para poder realizar simulaciones de procesos más o menos complejos. En esas simulaciones existe la necesidad de contar con *datos inventados*, pero *inventados* según un modelo proporcionado por una distribución de probabilidad que se suponga adecuado para el fenómeno que estamos simulando.

Es por ello que la mayoría de los paquetes de software estadístico, entre ellos R, facilitan la posibilidad de obtener muestras aleatorias simples de las distribuciones más usuales.

Lo que vamos a hacer a continuación es proporcionar dos ejemplos, uno de una distribución continua y otro de una discreta, de cómo se obtienen muestras aleatorias. Además, vamos a realizar una comparación gráfica para comprobar que, en efecto, esos datos simulados en las muestras corresponden con los modelos que supuestamente los generan.

5.5.1. Muestra procedente de una distribución normal

La simulación viene dada por la función `rnorm()`. La sintaxis es muy sencilla: tan sólo tenemos que especificarle el número de valores que queremos simular y los parámetros de la distribución. Por ejemplo, el ejemplo que hemos realizado mediante R Commander se escribiría simplemente como `rnorm(250, 100, 10)`.

Vamos a comprobar que, en efecto, esta muestra procede de una $N(1.2, 2.3)$. Para ello podemos comparar el histograma de la muestra con la función de densidad de esa $N(1.2, 2.3)$. Si, como debería ocurrir, los datos simulados proceden de esa normal, el histograma y la función de densidad deberían parecerse bastante. El resultado aparece en la Figura 5.2. Hay que decir que el parecido es notable, aunque sería aún más notable si hubiéramos elegido aún más datos (más de 250).

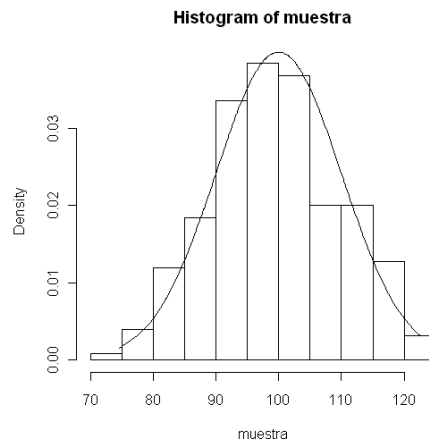


Figura 5.2: Histograma de una muestra junto con la función de densidad del modelo del que procede

El código es el siguiente:

1. `muestra<-sort(rnorm(250,100,10))` genera la muestra de tamaño 250 de la $N(100, 10)$ y la ordena mediante la función `sort()`¹.
2. `hist(muestra,freq=FALSE)` grafica el histograma de los datos de la muestra en escala de densidad.
3. Finalmente `lines(muestra,dnorm(muestra,100,10))` añade al histograma la gráfica de la función de densidad evaluada en los valores de la muestra.

Hemos de hacer un par de comentarios a este código a propósito de la función `lines()`. Se trata de una función muy similar a `plot()` en su estructura y utilidad, cuya principal diferencia radica en que `plot()` es lo que en R se conoce como una función gráfica de primer nivel y `lines()` lo es de segundo nivel. Que una función gráfica sea de primer nivel quiere decir que por sí misma abrirá una ventana gráfica y mostrará su resultado. Que una función sea de segundo nivel implica que por sí misma no tiene autoridad para abrir una ventana de gráficos; tan sólo puede añadirse a una ventana ya abierta.

`hist()` es una función gráfica de primer nivel, por lo que realiza el histograma y lo muestra en una ventana de gráfico. A esa ventana se añade la función de densidad graficada mediante `lines()`. ¿Qué hubiera ocurrido si lo hubiéramos hecho mediante `plot()`? Que al ser esta una función de primer nivel, habría anulado el histograma creando una nueva ventana sólo para ella.

¹¿Qué ocurre si no ordenamos los datos? Realizar el gráfico sin hacerlo y explicar porqué ocurre lo que ocurre.

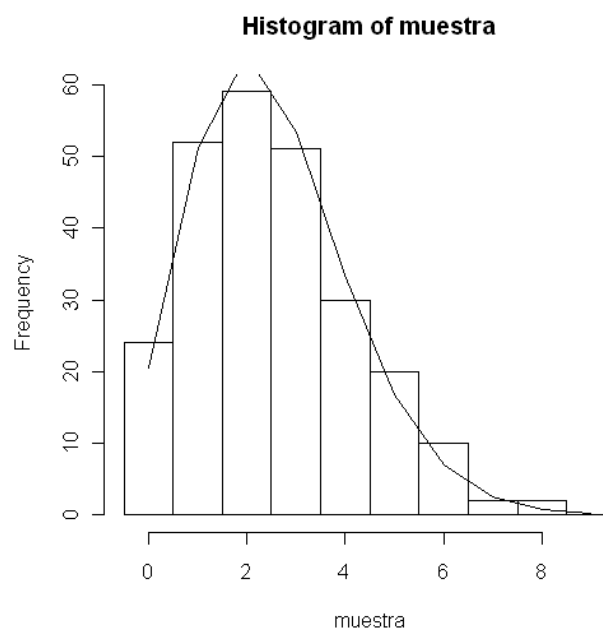


Figura 5.3: Comparación del diagrama de barras de una muestra generada según una *Poisson* (2.5) con las frecuencias esperadas según ese modelo.

5.5.2. Muestra procedente de una distribución de Poisson

Se realiza mediante la función `rpois()`, cuya sintaxis simplemente requiere el número de muestras y el parámetro de la distribución.

De nuevo vamos a comentar el código para plasmar cómo los datos simulados obedecen a la distribución que los generó:

1. Empezamos generando una muestra de 250 valores de una *Poisson* (2.5) mediante `muestra<-rpois(250,2.5)`.
2. Representamos un diagrama de barras para esos datos valiéndonos de la función `hist()`. Como el valor máximo que ha resultado en la muestra es 9, introducimos `hist(muestra,breaks=-0.5:9.5)`.
3. Añadimos a esa gráfica las frecuencias esperadas según la función masa de una *Poisson* (2.5) para los valores de 0 a 9 en una muestra de 250 datos mediante `lines(0:9,dpois(0:9,2.5)*250)`.

El resultado aparece en la Figura 5.3.

Capítulo 6

Estimación puntual y por intervalos de confianza

Objetivos:

1. Estimar mediante el método de máxima verosimilitud los parámetros que ajustan a un conjunto de datos las distribuciones que hemos manejado hasta ahora.
2. Obtener una representación gráfica que permita valorar, al menos visualmente, la bondad del ajuste logrado mediante el ajuste de una distribución a unos datos.
3. Obtener intervalos de confianza para medias, proporciones y varianzas.

6.1. Introducción

Como ya comentamos en el prólogo, en este capítulo se describe una manera de realizar estimaciones puntuales y por intervalos de confianza de algunos parámetros a través del código de R.

Commander, por su parte, no tiene la implementación de ningún método de estimación puntual, mientras que la estimación por intervalos de confianza la realiza simultáneamente con los correspondientes contrastes de hipótesis sobre dichos parámetros.

6.2. Estimación máximo verosímil

Como punto de partida a lo largo de todo el capítulo, vamos a suponer que tenemos una muestra aleatoria simple de datos de una variable. En esta sección lo que vamos a aprender es a obtener una estimación de los parámetros que ajustan las distribuciones que venimos manejando a estos datos.

El método de estimación que vamos a utilizar, quizá el más utilizado a todos los niveles¹, es el método de máxima verosimilitud. Recordemos que éste se basa en la optimización de la llamada función de verosimilitud, que depende de los parámetros de la distribución. R es una magnífica herramienta para programar esta optimización mediante el empleo de métodos numéricos desarrollados por los matemáticos, cuando esto es necesario.

No obstante, nosotros no vamos a programar el método de máxima verosimilitud, sino que vamos a utilizar un paquete de R, *MASS*², que ya trae incorporadas las estimaciones de los parámetros de las distribuciones más usuales. Para ello, recordemos primero cómo se instala y se carga el paquete:

1. Iniciamos R.
2. Elegimos la opción del menú *Paquetes* → *Instalar paquete(s)*.
3. Dependiendo de la versión de R, puede que se abra una ventana para que elijamos un *mirror* desde el que descargar el paquete. Por cercanía, elegimos *Spain*.
4. A continuación se abrirá una ventana titulada *Packages* en la que aparecen todos los paquetes que podemos instalar de R. Elegimos *MASS (VR)* y esperamos a la instalación de paquete.
5. Finalmente, elegimos en R la opción *Paquetes* → *Cargar paquete*. Aparecen todos los paquetes que tenemos incorporados en nuestra instalación de R. De entre ellos, elegimos *MASS*. Opcionalmente, este último paso es equivalente a ejecutar `library(MASS)`.

El paquete *MASS* contiene una función llamada `fitdistr()` que proporciona las estimaciones máximo verosímiles y los errores estándares asociados a esas estimaciones de las distribuciones más usuales dados unos datos. La sintaxis de esta función es muy simple; sólo necesita que especifiquemos:

- Los datos de la muestra en forma de vector. Sólo debemos tener en cuenta que estos datos no pueden incorporar datos faltantes (que aparecen como *NA*).
- La distribución de la cuál queremos los estimadores de sus parámetros. Las opciones que a nosotros nos interesan son las siguientes: "*Poisson*", "*geometric*", "*negative binomial*", "*exponential*", "*gamma*" y "*normal*". No aparece el estimador del parámetro *p* de una distribución binomial, probablemente porque es tan simple que los autores del paquete creyeron que no merecía la pena hacerlo.

¹Que sea o no el más adecuado ya es más controvertido.

²Venables, W. N. & Ripley, B. D. (2002) *Modern Applied Statistics with S*. Fourth Edition. Springer, New York. ISBN 0-387-95457-0

Es importante comentar que la función `fitdistr()` devuelve el valor exacto de los estimadores para las distribuciones de Poisson, geométrica, exponencial y normal, mientras que para las distribuciones binomial negativa y Gamma utiliza métodos numéricos para proporcionar valores aproximados de las estimaciones, ya que no existen fórmulas explícitas de los estimadores para esas distribuciones. Vamos a ir viendo en una serie de ejemplos cómo se concreta esta sintaxis para cada uno de los modelos que conocemos.

6.2.1. Para la distribución de Poisson

6.2.1.1. Estimación del parámetro

Uno de los ejemplos más clásicos sobre el uso de la distribución de Poisson es el del ajuste del número de muertes por coces de caballos en el ejército prusiano. Von Borkiewicz publicó y analizó en 1898 el número de muertes al año por coces de caballo que se habían producido en 10 secciones del ejército de Prusia durante 20 años. La distribución de frecuencias de estos datos aparece en el Cuadro 6.1³

Muertes/año	Frecuencia
0	109
1	65
2	22
3	3
4	1

Cuadro 6.1: Distribución de frecuencias en el ejemplo del ejército prusiano

Lo que se planteaban en aquella época es si este tipo de accidentes mortales se producían por puro azar en las distintas secciones o si había alguna causa que implicara un mayor número de muertes en algunos casos concretos. Esa cuestión del *puro azar* equivale a plantearse si la distribución de Poisson es adecuada para la variable considerada.

Los datos están en el fichero *DatosMuertesCoces.rda*. Al cargarlo tenemos un conjunto de datos activo (que en mi caso se llama *Datos.muertes*), dentro del cual hay una variable que se refiere al nº de muertes al año en cada una de las 20 secciones y los 10 años (la variable en mi conjunto de casos activo se llama *Muertes*). Lo que vamos a hacer es estimar el parámetro de la distribución de Poisson. Es tan simple como introducir en la consola el código siguiente:

```
ajuste.poisson<-fitdistr(Datos.muertes$Muertes,"Poisson")
lambda<-ajuste.poisson$estimate
```

³Recordemos que los datos se pueden introducir con el siguiente código:

```
Datos.muertes<-data.frame(Muertes=c(rep(0,109),rep(1,65),rep(2,22),rep(3,3),4) )
```

El resultado de `fitdistr()` es una lista con varios aspectos sobre el ajuste. Si queremos ver el resultado completo, ejecutamos `ajuste.poisson`:

```
lambda
0.6100000
(0.0552268)
```

Sin embargo, del ajuste lo que más nos interesa es el valor estimado de λ , que es una de las variables de la lista, llamada `estimate`. Por eso hemos definido `lambda<-ajuste.poisson$estimate`.

En resumen, el parámetro λ de la distribución de Poisson, que es su media, es estimado en un valor $\hat{\lambda} = 0.61$, y el error estándar de esa estimación es de 0.052268.

6.2.1.2. Comparación del modelo estimado con la distribución de frecuencias

Así pues, proponemos ajustar los datos cuya distribución de frecuencias aparece en el Cuadro 6.1 mediante una distribución *Poisson* (0.61). Pero, ¿hasta qué punto ese modelo Poisson es realmente bueno para esos datos?

Lo que vamos a hacer nosotros para poder decir algo al respecto es representar en una misma figura la distribución de frecuencias de la muestra, junto con las frecuencias que determina una distribución *Poisson* (0.61) para 200 datos, que son los que tiene la muestra.

En primer lugar, para ver estas frecuencias, asociadas a una muestra de 200 datos, según la distribución de Poisson ajustada, no tenemos más que introducir el código `200*dpois(0:4,lambda)`, lo que nos devuelve los siguientes resultados:

```
108.6701738 66.2888060 20.2180858 4.1110108 0.6269291
```

Ahora, por tanto, lo que queremos es un diagrama de barras que represente la distribución de frecuencias y, además, dibujar estas frecuencias que determina la distribución de Poisson ajustada a los datos:

- `hist(Datos.muertes$Muertes,breaks=-0.5:4.5)` dibuja la distribución de frecuencias. Recordad que no debéis cerrar ese gráfico.
- `lines(0:4,200*dpois(0:4,lambda))` añade al gráfico anterior las frecuencias según el modelo ajustado.

El gráfico resultante lo tenemos en la Figura 6.1. Resulta interesante ver cómo el modelo ajustado casi *clava* la distribución de frecuencias. Eso quiere decir que parece que la distribución de Poisson

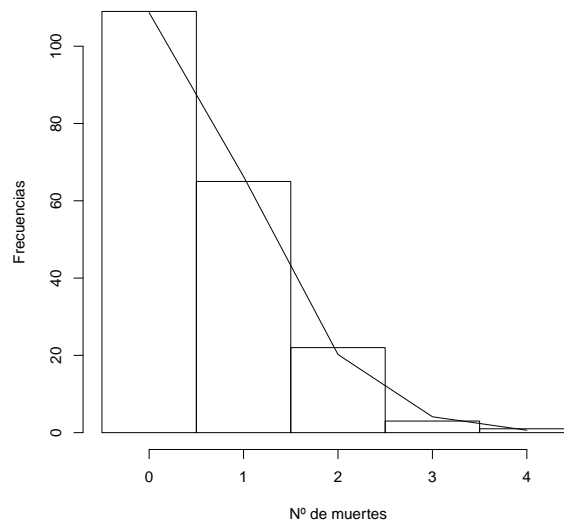


Figura 6.1: Diagrama de barras y frecuencias del modelo de Poisson ajustado para los datos del número de muertes por coces en el ejército prusiano

sí es adecuada para esos datos⁴.

6.2.2. Para la distribución geométrica

En tráfico de telecomunicaciones los datos se transmiten en unas unidades de información llamadas *paquetes*. En un protocolo de transmisión cualquiera el tiempo que pasa entre un paquete de información y el siguiente es variable, mejor dicho, es aleatorio, ya que depende de multitud de factores difícilmente controlables, la mayoría de los cuales se refieren a las condiciones de la red en el momento de la transmisión. Los datos que vamos a manejar se refieren a los tiempos (en *ms*) que transcurren entre una muestra aleatoria de 250 paquetes en una determinada transmisión. Están en el fichero *trafico.telem.rda*. Puede resultar extraño que tratándose de tiempos los analicemos como una variable discreta en vez de continua, pero a este nivel (es decir, al nivel de los milisegundos) los ordenadores suelen trabajar con el tiempo como si fuera una variable discreta.

Una de las aplicaciones más usuales de la Estadística en este campo se refiere a la posibilidad de proporcionar muestras simuladas. Para ello, previamente se necesita un modelo determinado por una distribución de probabilidad. En esta cuestión es donde vamos a considerar nuestro objetivo: queremos proporcionar una distribución de probabilidad adecuada para esos datos.

Uno de los más importantes organismos internacionales de Telecomunicaciones, la ETSI, recomienda que para datos como estos se utilice la distribución geométrica. Vamos, por ahora, a hacerle caso a

⁴Pero no podemos por ahora afirmar nada categóricamente al respecto. Veremos esto con rigor más adelante.

la ETSI y ajustar una distribución geométrica a estos datos estimando su parámetro por máxima verosimilitud.

6.2.2.1. Estimación del parámetro

Para ello, cargamos en primer lugar el conjunto de datos, que se encuentra en el fichero *trafico.telem.rda*, y cuyo nombre es también *Datos.trafico*. Para ver el nombre de la variable podemos ejecutar `names(Datos.trafico)` y veremos que el nombre es *tiempos.entre.paquetes*. Finalmente, el código para obtener la estimación es

```
ajuste.geom<-fitdistr(Datos.trafico$tiempos.entre.paquetes,"geometric")
p<-ajuste.geom$estimate
```

El resultado del ajuste se resume así:

```
prob
0.117591722
(0.006986208)
```

Por lo tanto, el ajuste por el método de máxima verosimilitud de los datos mediante una distribución geométrica lo es con un parámetro $p = 0.1176$, con un error típico para esta estimación de casi 0.007.

6.2.2.2. Comparación del modelo estimado con la distribución de frecuencias

Como en el anterior ejemplo, vamos ahora a obtener una representación gráfica simultánea de la distribución de frecuencias y de las frecuencias que se esperan según una distribución *Geo*(0.1176) para 250 datos.

Los valores que se observan en la muestra van de 0 a 32. Las probabilidades según una *Geo*(0.1176) de los valores de 0 a 32 los proporciona el código `dgeom(0:32,p)`. Debemos multiplicar estas probabilidades por 250, que es el número de datos de la muestra, para obtener las frecuencias esperadas según el modelo. Finalmente, el código para la representación gráfica es el siguiente:

```
■ hist(Datos.trafico$tiempos.entre.paquetes,breaks=-0.5:32.5
,freq=TRUE,main="",xlab="Tiempos entre paquetes"
,ylab="Frecuencias")
```

dibuja la distribución de frecuencias. No debeis cerrar ese gráfico.

```
■ lines(0:32,250*dgeom(0:32,p))
```

añade al gráfico anterior las frecuencias según el modelo ajustado.

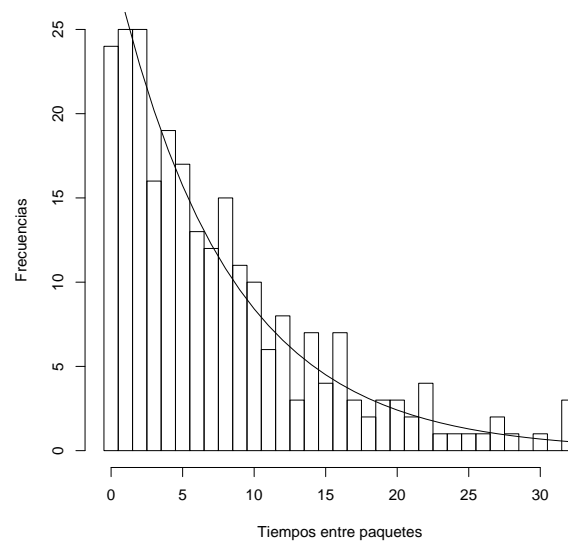


Figura 6.2: Ajuste de los datos de tiempos entre paquetes mediante una distribución geométrica

El resultado aparece en la Figura 6.2. Ahora no es tan fácil valorar si el ajuste es bueno o malo, pero si nos fijamos bien, las frecuencias de los primeros valores (0 y 1 sobre todo) están bastante mal ajustadas, y hay algunas otras frecuencias intermedias que tampoco están bien ajustadas. No podemos decir que el ajuste sea malo, pero tampoco que es excelente. Más adelante aprenderemos a ser más objetivos y precisos en esta cuestión.

6.2.3. Para la distribución binomial negativa

Vamos a continuar con los datos de los tiempos entre paquetes. Como no estamos muy convencidos del ajuste mediante la distribución geométrica y puesto que sabemos que la distribución binomial negativa es una generalización de ésta, vamos a proporcionar un ajuste mediante una binomial negativa.

6.2.3.1. Estimación de los parámetros

Simplemente ejecutamos el siguiente código:

```
ajuste.binoneg<-fitdistr(trafico.telem$tiempos.entre.paquetes,
"negative binomial")
```

Si ejecutamos `ajuste.binoneg` veremos que el resultado es el siguiente:

```
size mu
1.1903584 7.5039917
```


(0.1265281) (0.4682259)

Vamos a interpretarlo:

- El parámetro *size* es el parámetro que nosotros hemos llamado a en una $BN(a, p)$, luego $\hat{a} = 1.19$, con un error típico de 0.1265.
- El parámetro μ es la media. Nosotros sabemos que la media es $\mu = a \frac{1-p}{p}$ luego, despejando, tenemos que

$$p = \frac{a}{a + \mu}.$$

Si aplicamos esto a nuestras estimaciones,

$$\hat{p} = \frac{\hat{a}}{\hat{a} + \hat{\mu}} = \frac{1.1903584}{1.1903584 + 7.5039917} = 0.1369117.$$

Por lo tanto, el ajuste es una distribución $BN(1.19, 0.137)$.

Para plasmar esto con código podemos escribir y ejecutar lo siguiente:

```
a<-ajuste.binoneg$estimate[1]
mu<-ajuste.binoneg$estimate[2]
p<-a/(a+mu)
```

6.2.3.2. Comparación del modelo ajustado con la distribución de frecuencias

Para construir el gráfico que nos permite realizar la comparación del modelo ajustado con la distribución de frecuencias, consideramos el siguiente código:

- `hist(Datos.trafico$tiempos.entre.paquetes,breaks=-0.5:32.5, freq=TRUE,main="",xlab="Tiempos entre paquetes", ylab="Frecuencias")`
dibuja la distribución de frecuencias. No debeis cerrar ese gráfico.
- `lines(0:32,250*dnbinom(0:32,a,p))` añade al gráfico anterior las frecuencias según el modelo ajustado.

El resultado aparece en la Figura 6.3. Podemos ver que el ajuste es mejor que el proporcionado por la geométrica. Fijémonos sobre todo en las frecuencias del 0 y del 1, que ahora están mucho mejor ajustadas.

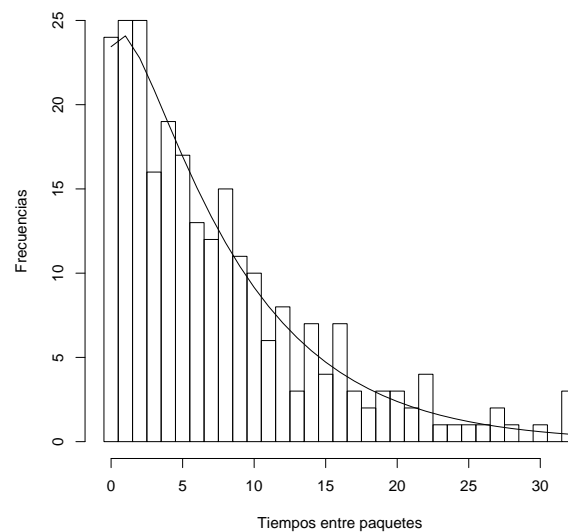


Figura 6.3: Ajuste de los datos de tiempos entre paquetes mediante una distribución binomial negativa

6.2.4. Para la distribución exponencial

Como conjunto de datos para ejemplificar el procedimiento vamos a considerar 300 datos correspondientes al tiempo hasta el fallo (en años) de unas determinadas componentes electrónicas usadas en un proceso industrial. En principio, el histograma se asemeja bastante al de una distribución exponencial, así que no parece descabellado tratar de obtener un ajuste mediante esta distribución para estos datos.

6.2.4.1. Estimación de los parámetros

Los datos se encuentran en el fichero *tiempos.fallo.rda*, en una hoja llamada *Datos.tiempos.fallo*. Al cargarlos, podemos ver que la variable llama *Años*. Con esta información, el código para obtener el estimador máximo-verosímil del parámetro λ de una distribución exponencial que ajuste los datos es el siguiente:

```
ajuste.exp<-fitdistr(Datos.tiempos.fallo$Años,"exponential")
lambda<-ajuste.exp$estimate
```

El resultado de `ajuste.exp` es el siguiente:

```
rate
0.49185399
(0.02839720)
```

Tenemos, por tanto, que el ajuste por el método de máxima verosimilitud de la distribución exponencial de estos datos arroja como modelo el de una $\exp(0.492)$. El error estándar estimado de la estimación de λ es 0.028.

6.2.4.2. Comparación del modelo ajustado con los datos muestrales

Como en los ajustes anteriores, vamos ahora a tratar de obtener una representación gráfica de cómo de preciso es el ajuste. Esta vez se trata de una variable continua, por lo que ya no podemos utilizar frecuencias absolutas ni relativas, sino densidades. Lo que vamos a hacer es comparar el histograma, utilizando una escala de densidad, con la función de densidad de la distribución $\exp(0.492)$:

1. Para obtener ese histograma utilizamos

```
hist(Datos.tiempos.fallo$Años,freq=FALSE,  
main="",xlab="Años",ylab="Densidad").
```

Observad que he elegido `freq=FALSE` para que la escala del histograma sea la de una densidad. Esta gráfica, como siempre, no debeis cerrarla.

2. Ahora vamos a añadirle el gráfico de la función de densidad de la distribución $\exp(0.492)$. Vamos a dibujar esta densidad en el mismo dominio que tiene el histograma, de 0 a 10. Vamos a dibujarla seleccionando, por ejemplo, 100 puntos entre 0 y 10, con la misma distancia entre ellos. Una forma de hacerlo es mediante la función `seq()` (por *sequence*, secuencia), concretamente con `seq(0,10,length.out=100)`. El código es, por tanto,

```
x<-seq(0,10,length.out=100)  
lines(x,dexp(x,lambda))
```

El gráfico resultante aparece en la Figura 6.4. Podemos ver a simple vista bastante parecido entre el histograma y la función de densidad del modelo ajustado.

6.2.5. Para la distribución Gamma

Vamos a tratar de ajustar ahora los mismos datos, relativos a los tiempos hasta el fallo de 300 componentes electrónicas, mediante una distribución Gamma.

6.2.5.1. Estimación de los parámetros

El código ahora es

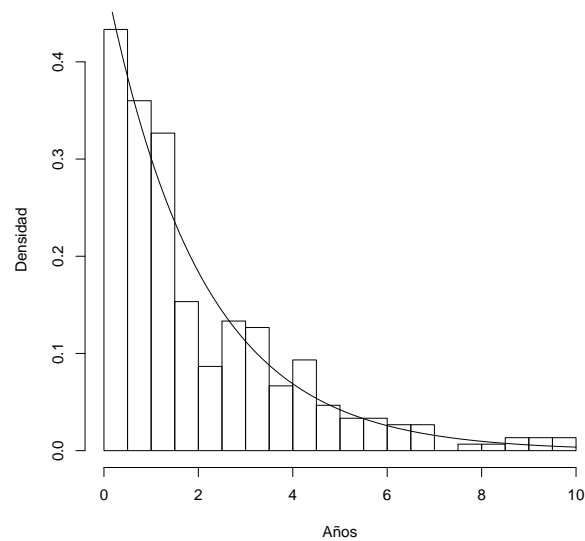


Figura 6.4: Comparación del histograma de los tiempos de fallo con la densidad exponencial ajustada

```
ajuste.gamma<-fitdistr(Datos.tiempos.fallo$Años,"gamma")
```

```
ajuste.gamma
```

```
a<-ajuste.gamma$estimate[1]
```

```
lambda<-ajuste.gamma$estimate[2]
```

El resultado que aparece es el siguiente:

```
shape rate
```

```
0.96639591 0.47532653
```

```
(0.06926486) (0.04404461)
```

Podemos comentar algo importante en estos parámetros en relación al ajuste anterior, el que realizamos con la distribución exponencial: fijémonos en que el parámetro a o parámetro *de forma* (*shape*), se estima en 0.966. Recordemos que una distribución Gamma con $a = 1$ es, en realidad, una distribución exponencial. Lo que nos está diciendo esta estimación es que la distribución Gamma resultante es casi una distribución exponencial. De hecho, fijémonos también en que la estimación del parámetro de razón, λ , es 0.475, cuando en el caso de la exponencial era 0.492. Son valores muy parecidos. Estas estimaciones van a provocar que la densidad exponencial antes ajustada y la de la *Gamma* (0.966, 0.475) sean muy parecidas.

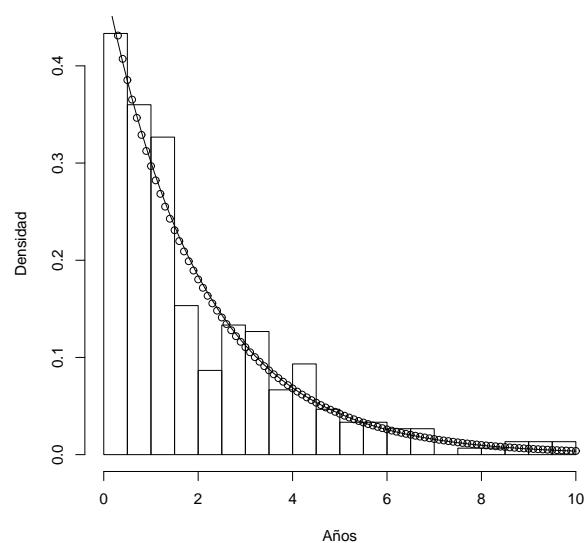


Figura 6.5: Comparación del histograma de los tiempos de fallo con las densidades ajustadas exponencial y Gamma

6.2.5.2. Comparación del modelo ajustado con los datos muestrales

Para poner esto de manifiesto, vamos a añadir al gráfico anterior, el que aparece en la Figura 6.4, la representación de la densidad de la *Gamma*(0.966,0.475). Para ello introducimos el siguiente código:

```
x<-seq(0,10,length.out=100)
lines(x,dgamma(x,a,lambda),"p")
```

Hemos añadido "p" a la función `lines()` para que dibuje la densidad Gamma con puntos ya que, de otra forma es casi imposible distinguirla de la densidad exponencial que antes habíamos dibujado. El resultado aparece en la Figura 6.5.

6.2.6. Para la distribución normal

Como ejemplo con el que vamos a aprender a realizar la estimación, consideremos los datos relativos al tiempo en segundos que tarda cada uno de los operarios cualificados de un colectivo de 158 operarios en ejecutar una tarea sencilla dentro de un proceso industrial. Los datos aparecerán resumidos en su histograma en la Figura 6.6, junto al ajuste mediante una distribución normal. Se encuentran en el fichero *operarios.tarea.rda*; el nombre del conjunto de datos es *muestra* y el nombre de la variable es *tiempos*. Lo primero que debemos hacer, obviamente, es cargar el conjunto de datos.

6.2.6.1. Estimación de los parámetros

Para obtener la estimación de los parámetros μ y σ de la distribución normal por máxima verosimilitud podemos usar el siguiente código:

```
ajuste.normal<-fitdistr(muestra$tiempo,"normal")
```

El resultado es el siguiente:

```
mean sd
42.3531760 3.6633968
(0.2914442) (0.2060822)
```

Es decir, el ajuste mediante el método de máxima verosimilitud viene dado por una $N(42.353, 3.663)$. Podemos ver, además, las estimaciones de los errores estándares de estas estimaciones, 0.291 y 0.206.

Para quedarnos con los valores estimados de μ y σ ,

```
mu<-ajuste.normal$estimate[1]
sigma<-ajuste.normal$estimate[2]
```

6.2.6.2. Comparación del modelo ajustado con los datos muestrales

De nuevo vamos a representar en una misma gráfica el histograma asociado a los datos y la densidad del modelo ajustado. Eso nos permitirá valorar, al menos de forma visual, la precisión del ajuste:

1. En primer lugar, obtenemos con el histograma con la escala de densidad:

```
hist(muestra$tiempos,freq=FALSE,main="",xlab="Tiempos",ylab="Densidad")
```

2. Sin cerrar el gráfico del histograma, y observando que el eje X va aproximadamente de 35 a 55, ejecutamos el siguiente código⁵:

```
x<-seq(35,55,length.out=100)
lines(x,dnorm(x,mu,sigma))
```

El resultado es el que aparece en la Figura 6.6. Podemos estar tentados a decir que el ajuste es muy bueno, pero observemos que el histograma es ligeramente asimétrico a la derecha, mientras que la densidad sólo puede ser totalmente simétrica. Por eso tenemos la densidad ligeramente por encima del histograma en la cola de la izquierda y ligeramente por debajo a la derecha. En la práctica, eso quiere decir que el modelo puede estar sobrevalorando la cola de la izquierda e infravalorando la de

⁵En él, `350:550` proporciona todos los valores enteros entre 350 y 550. Al dividir éstos por 10 obtenemos valores entre 35 y 55 de 0.1 en 0.1.

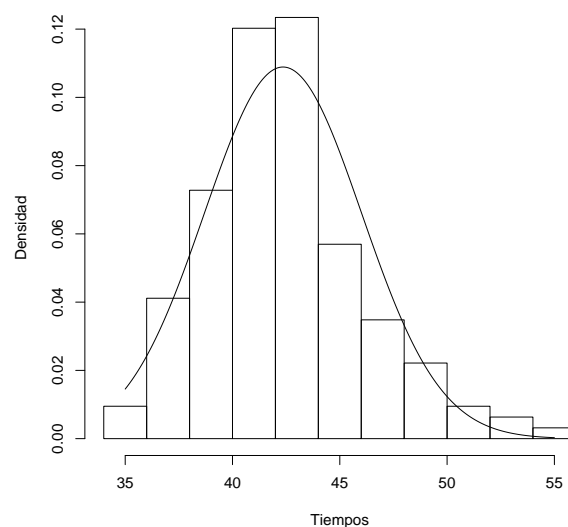


Figura 6.6: Comparación del histograma de los tiempos de realización de la tarea con la densidad normal ajustada

la derecha. No olvidemos que la cola de la derecha corresponde con los operarios que más tardan en ejecutar la tarea.

6.3. Estimación por intervalos de confianza

Los intervalos de confianza guardan una relación biunívoca con los contrastes de hipótesis estadísticas, que estudiaremos en el siguiente capítulo. Ese es el motivo por el que R Commander no facilita un menú específico para la construcción de intervalos de confianza, sino que éstos son proporcionados como parte de los resultados vinculados a los contrastes de hipótesis. Por este motivo, no vamos a comentar aquí cómo se realizan los intervalos de confianza mediante R Commander, sino en el capítulo siguiente.

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

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

- `mean(datos)` devuelve la media muestral de `datos`.
- `sd(datos)` devuelve la desviación típica muestral de `datos`.

- `sqr(x)` devuelve \sqrt{x} .
- `qnorm(α)` devuelve z_α .
- `qt(α, v)` devuelve $t_{\alpha, v}$.
- `qchisq(α, v)` devuelve $\chi^2_{\alpha, v}$.

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

Recordemos que si notamos x_1, \dots, x_N a una muestra de una distribución $N(\mu, \sigma)$, ambas desconocidas, un intervalo de confianza con nivel de significación α para μ viene dado por

$$\left(\bar{x} \mp t_{1-\frac{\alpha}{2}; n-1} s_{n-1} / \sqrt{N} \right).$$

Vamos a considerar de nuevo el ejemplo del tiempo de los tiempos de ejecución de una tarea por parte de trabajadores cualificados, suponiendo que esta variable tenga una distribución normal. Teníamos una muestra de 158 operarios y sus tiempos. El archivo es *operarios.tarea.rda*, el nombre del conjunto de datos es *muestra* y el nombre de la variable es *tiempos*. Queremos un intervalo de confianza para el tiempo medio de ejecución de la tarea, con $\alpha = 0.05$. El código es el siguiente:

- `ci<-mean(muestra$tiempos)-qt(0.975,157)*sd(muestra$tiempos)/sqrt(158)`. Esto calcula la cota inferior del intervalo, llamándola *ci*.
- `cs<-mean(muestra$tiempos)+qt(0.975,157)*sd(muestra$tiempos)/sqrt(158)`. Esto calcula la cota superior del intervalo, llamándola *cs*.
- `c(ci,cs)`. Eso aglutina en un vector la cota inferior y la cota superior, haciéndolas aparecer en la ventana de resultados.

El resultado es 41.77569 42.93066. Es decir, la probabilidad de que el intervalo (41.776, 42.931) contenga a la media de los tiempos de ejecución de la tarea es del 95 %.

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

Antes hemos tratado de obtener un modelo para la variable tiempos hasta el fallo de 300 componentes electrónicas. Probamos la distribución exponencial y la Gamma.

Ahora nos da igual si estos modelos son o no adecuados; lo que queremos hacer es obtener un intervalo de confianza para la media μ desconocida, de la variable.

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

$$\left(\bar{x} \mp z_{1-\frac{\alpha}{2}} s_{n-1} / \sqrt{N} \right),$$

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

- `ci<-mean(Datos.tiempos.fallo$Años)`
`-qnorm(0.975)*sd(Datos.tiempos.fallo$Años)/sqrt(300)`. Esto calcula la cota inferior del intervalo, llamándola *ci*.
- `cs<-mean(Datos.tiempos.fallo$Años)`
`+qnorm(0.975)*sd(Datos.tiempos.fallo$Años)/sqrt(300)`. Esto calcula la cota superior del intervalo, llamándola *cs*.
- `c(ci,cs)`. Eso aglutina en un vector la cota inferior y la cota superior, haciéndolas aparecer en la ventana de resultados.

El resultado es 1.806714 2.259533. Es decir, la probabilidad de que el intervalo (1.807, 2.260) contenga a la media del tiempo de fallo de las componentes electrónicas es del 95 %.

6.3.3. De una proporción

Supongamos que una empresa envasadora de nueces comprueba que en una muestra de 300 nueces, 21 están vacías. La marca quiere proporcionar un intervalo de confianza al 95 % para el porcentaje de nueces vacías en las bolsas que saca al mercado. Vamos a calcularlo.

El intervalo, para un nivel α viene dado por

$$\left(\hat{p} \mp z_{1-\frac{\alpha}{2}} \sqrt{\frac{\hat{p}(1-\hat{p})}{N}} \right)$$

donde \hat{p} es la proporción muestral (en nuestro caso, $\frac{21}{300}$) y N es el tamaño de la muestra (en nuestro caso, 300). El código para obtener el intervalo es el siguiente:

- `ci<-21/300-qnorm(0.975)*sqrt((21/300)*(1-21/300)/300)`. Para la cota inferior.
- `cs<-21/300+qnorm(0.975)*sqrt((21/300)*(1-21/300)/300)`. Para la cota superior.

- `c(ci,cs)`. Para que aparezcan en la ventana de resultados.

El resultado es 0.04112793 0.09887207, luego un intervalo de confianza al 95 % para el porcentaje de nueces vacías de la marca es (4.11 %, 9.89 %).

6.3.4. De la varianza de una distribución normal

Finalmente, vamos a obtener un intervalo de confianza para la varianza de la variable de tiempo de ejecución de la tarea por parte de operarios cualificados.

En clase hemos visto que dicho intervalo viene dado por

$$\left(\frac{\sum_{i=1}^N (x_i - \bar{x})^2}{\chi_{1-\frac{\alpha}{2}; N-1}^2}, \frac{\sum_{i=1}^N (x_i - \bar{x})^2}{\chi_{\frac{\alpha}{2}; N-1}^2} \right).$$

Teniendo en cuenta que $s_{N-1}^2 = \frac{\sum_{i=1}^N (x_i - \bar{x})^2}{N-1}$, otra forma de expresarlo es

$$\left(\frac{(N-1) s_{N-1}^2}{\chi_{1-\frac{\alpha}{2}; N-1}^2}, \frac{(N-1) s_{N-1}^2}{\chi_{\frac{\alpha}{2}; N-1}^2} \right).$$

El código es el siguiente:

- `ci<-157*var(muestra$tiempos)/qchisq(0.975,157)`. Para la cota inferior.
- `cs<-157*var(muestra$tiempos)/qchisq(0.025,157)`. Para la cota superior.
- `c(ci,cs)`. Para que aparezcan en la ventana de resultados.

El resultado es 2.980518 4.645565, lo que quiere decir que la probabilidad de que el intervalo (2.981, 4.646) contenga a la varianza del tiempo de ejecución de la tarea es del 95 %.

Capítulo 7

Contraste de hipótesis paramétricas

Objetivos:

- Realizar contrastes de hipótesis sobre la media de una y dos poblaciones.
- Realizar contrastes de hipótesis sobre la proporción en una y dos poblaciones.
- Realizar contrastes de hipótesis sobre varianzas en dos poblaciones.
- Realizar contrastes sobre la media en más de dos poblaciones (ANOVA).
- Obtener intervalos de confianza mediante R Commander.

7.1. Introducción

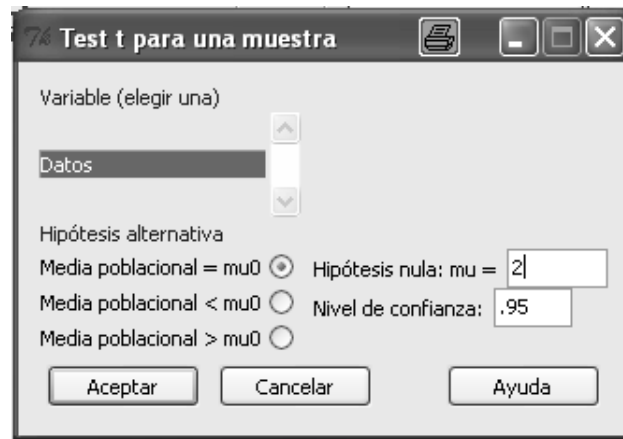
A lo largo de este tema vamos a abordar la realización de contrastes de hipótesis paramétricas a través de diversos ejemplos. Como siempre, vamos a comentar las posibilidades de ejecutar dichos tests directamente a través de la consola de R y mediante las opciones de R Commander.

7.2. Contrastes sobre medias

7.2.1. Contraste sobre la media de una población

Consideremos el siguiente enunciado:

Un ingeniero industrial ha diseñado una máquina que envasa bolsas de cebollas de dos kilos. Sin embargo, debido a diversas razones, como los diferentes pesos de las cebollas, problemas en el llenado, etc. es consciente de que el peso final de la bolsa de cebollas

Figura 7.1: Test t para una muestra

no será exactamente de dos kilos, sino que se producirán variaciones aleatorias con respecto a esta cantidad. Para comprobar si la máquina está bien calibrada, toma una muestra de 45 bolsas llenas de cebollas y contabiliza su peso. Con esta información, ¿tiene razones el ingeniero para pensar que la máquina está mal calibrada? (Utilícese un nivel de significación del 5%).

Los datos se encuentran en el fichero *cebollas.txt*. Vamos a empezar planteando el problema y posteriormente veremos cómo resolverlo.

Fijémonos que nos piden claramente que confirmemos una afirmación: *la máquina está mal calibrada*. Eso obliga a plasmar dicha afirmación en la hipótesis alternativa, que es la única a la que podemos asignar la carga del nivel de confianza. Por lo tanto, si notamos μ al peso medio de las bolsas, tenemos que queremos contrastar $H_0 : \mu = 2$ frente a $H_1 : \mu \neq 2$.

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

7.2.1.1. Resolución del problema mediante R Commander

Debemos importar los datos desde el fichero *cebollas.txt* para manejar la variable en cuestión.

A continuación elegimos la opción del menú *Estadísticos* \rightarrow *Medias* \rightarrow *Test t para una muestra*. Esta opción abrirá la ventana que aparece en la Figura 7.1. Fijémonos con detalle en ella:

- Nos pide en primer lugar que elijamos una (sólo una) variable, que debe ser aquella cuya media estemos analizando.

- Nos pide que indiquemos cuál es la hipótesis alternativa. En nuestro caso hemos elegido la opción de un test bilateral.
- Nos pide que especifiquemos el valor del valor hipotético con el que estamos comparando la media, en nuestro caso, 2.
- Nos pide, por último, que especifiquemos un nivel de confianza. En realidad este nivel de confianza no lo es para el contraste, que se resolverá a través del p-valor, sino para el intervalo de confianza asociado al problema. El enunciado no dice nada, por lo que ponemos la opción habitual del 95 %.

El resultado es el siguiente:

```
One Sample t-test
```

```
data: Datos$Datos
```

```
t = -1.8415, df = 44, p-value = 0.0723
```

```
alternative hypothesis: true mean is not equal to 2
```

```
95 percent confidence interval:
```

```
1.994974 2.000227
```

```
sample estimates:
```

```
mean of x
```

```
1.9976
```

Analicemos el resultado con detalle:

- En primer lugar, nos recuerda que estamos analizando la variable `Datos$Datos`.
- A continuación nos informa del valor del estadístico de contraste (`t = -1.8415`), de los grados de libertad (`df = 44`) y del p-valor (`p-value = 0.0723`). Ya podemos, por tanto, concluir:

Dado que el p-valor no es inferior al 5 %, no tenemos suficientes evidencias en los datos para rechazar la hipótesis nula ($\mu = 2$) en favor de la alternativa ($\mu \neq 2$), es decir, con los datos de la muestra no tenemos suficientes evidencias de que el peso medio de las bolsas sea distinto de 2.
- Nos recuerda cuál era la hipótesis nula que habíamos planteado: `alternative hypothesis: true mean is not equal to 2`.

- A continuación proporciona un intervalo de confianza unilateral a la derecha, con un nivel de confianza del 95 %, para la media de la distribución normal que se le supone a los datos: `95 percent confidence interval: 1.994974 2.000227`. Lo que quiere decir el resultado es que

$$P[\mu \in (1.994974, 2.000227)] = 0.95.$$

La relación que guarda el intervalo de confianza con el contraste de hipótesis es la siguiente: fijémonos que el valor hipotético que hemos considerado para la media, 2, está dentro de este intervalo, luego éste es un valor *de confianza* para μ . Es otra forma de concluir que no hay datos que avalen que la media de la variable es significativamente distinta de 2, ya que éste es un valor bastante plausible para esta media. Si los datos fueran tales que el intervalo de confianza para μ dejara fuera al valor 2, tendríamos razones para pensar que el valor de μ es significativamente distinto de 2, pero no es el caso.

- Finalmente, proporciona los estadísticos muestrales utilizados, en este caso, la media muestral: `sample estimates: mean of x 1.9976`.

7.2.1.2. Consideraciones finales

Por lo tanto, y a modo de conclusión, podemos decir que no hay evidencias de que el peso medio de las bolsas de cebollas sea distinto de dos, lo que implicaría que la máquina está mal calibrada.

7.2.2. Contraste para la diferencia de medias de poblaciones independientes

Nos vamos a centrar ahora en el siguiente enunciado:

Una ingeniera industrial ha sintetizado en el laboratorio una feromona con la que pretende luchar contra una plaga de insectos. La feromona se aplica en trampas donde caen los insectos masivamente. Hasta ahora se trabajaba introduciendo otro producto que se supone que atraía al insecto, por lo que la ingeniera desearía demostrar que su feromona sintetizada es más efectiva que dicho producto. Para probar si esto ocurre, prepara 100 trampas con el producto tradicional y 100 con su feromona y las distribuye, contabilizando el número de insectos atrapados en cada una de las 200 trampas. Con esos datos, ¿puede concluir la ingeniera que su feromona es más efectiva que el producto tradicional? (Utilícese un nivel de significación del 5 %).

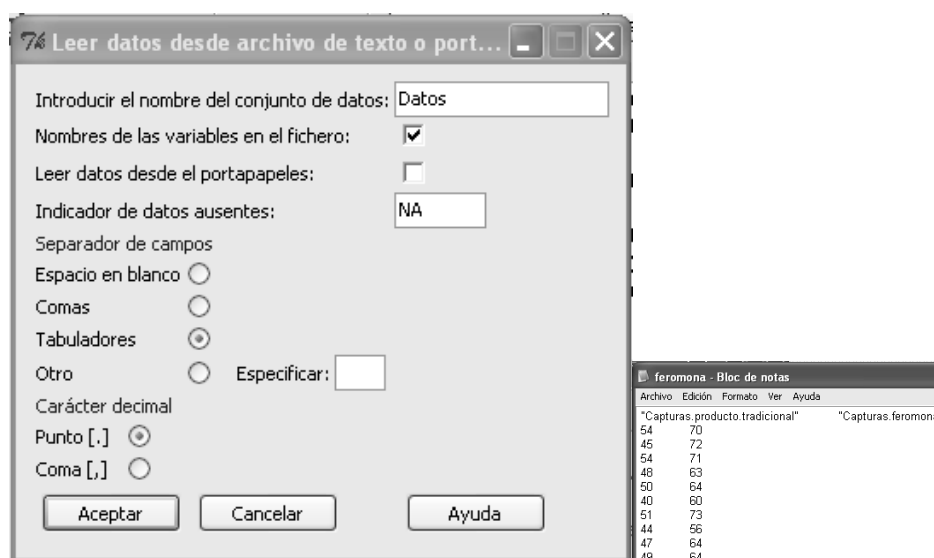


Figura 7.2: Importando los datos de los diámetros de los cojinetes

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

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

7.2.2.1. Resolución mediante R Commander

Lo primero que tenemos que hacer para importar los datos, que se encuentran en un fichero de tipo texto es ver cómo están almacenados: si lo abrimos, por ejemplo, con el bloc de notas, vemos que están separados por tabulaciones y que los nombres de las variables están en la primera fila. La Figura 7.2 a la izquierda muestra la ventana con la que importamos los datos, mientras que la parte de la derecha muestra cómo se ven con el bloc de notas.

Fijémonos que los datos de las dos muestras aparecen en dos columnas paralelas. Esa es una forma no demasiado correcta de especificarlas, ya que parece que cada dato de una de las muestras está



Figura 7.3: Apilando los datos de las dos muestras

relacionado con otro dato de la otra muestra y, en realidad, las muestras son independientes (de hecho podrían tener distinto tamaño muestral).

Por este motivo, tenemos que preparar los datos para que R Commander entienda que se trata de dos muestras independientes. Lo que tenemos que hacer es juntar o apilar las dos muestras en una sola columna, indicando en una segunda columna si el dato es de una muestra u otra.

Esta operación se realiza mediante la opción *Datos* → *Conjunto de datos activo* → *Apilar variables del conjunto de datos activo*. Lo que tenemos que indicar en esta ventana (Figura 7.3 a la derecha) es cuáles son las variables que queremos apilar, el nombre del nuevo conjunto de datos, el nombre de la nueva variable y el nombre del factor que separa los datos de las dos muestras. En la parte de la derecha de la Figura 7.3 aparecen los datos tal y como quedan tras apilarlos.

Observemos que, en efecto, ha creado una primera columna con todos los datos y una segunda columna con un factor que especifica cuáles de ellos son del producto viejo y cuáles del nuevo.

Hay un último aspecto muy importante: los datos son ordenados según el factor, y éste ordena sus categorías por orden alfabético. Eso en nuestro ejemplo hace que aparezcan primero los datos según el nuevo producto y después según el viejo.

Ahora ya tenemos los datos preparados para ser analizados. Elegimos la opción *Estadísticos* → *Medias* → *Test t para muestras independientes*. Esta opción abre la ventana de entradas que aparece en la Figura 7.4. En ella podeis ver que tenemos que especificar el factor que separa las dos muestras, la variable que estamos analizando, la hipótesis alternativa, el nivel de confianza requerido y si podemos suponer varianzas iguales. En nuestro caso:

- El factor es el único que aparece, que hemos llamado *Producto*.
- La variable es la única numérica del conjunto de datos, que hemos llamado *Capturas*.



Figura 7.4: Test t para muestras independientes. Ventana de entradas

- Es un test unilateral: ¿cómo sabemos cuál es la muestra 1 y cuál la muestra 2? Lo sabemos porque el factor se ordena alfabéticamente, luego las capturas con la feromona es la muestra 1 y con el viejo producto la muestra 2. Como queremos contrastar $H_1 : \mu_N > \mu_V$, señalamos *Diferencia* > 0 .
- El nivel de confianza exigido es del 95 %, que va a ser el que consideremos para el intervalo de confianza asociado.
- No tenemos razones que avalen que las varianzas deban ser consideradas iguales.

El resultado que aparece en la ventana de resultados es el siguiente:

Welch Two Sample t-test

data: Capturas by Producto

t = 20.4367, df = 197.952, p-value < 2.2e-16

alternative hypothesis: true difference in means

is greater than 0

95 percent confidence interval:

14.25580 Inf

sample estimates:

mean in group Capturas.feromona

64.94

mean in group Capturas.producto.tradicional

49.43

Vamos a analizarlo punto por punto:

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

7.2.2.2. Consideraciones finales

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

7.2.3. Contraste para la diferencia de medias de poblaciones apareadas

El problema que vamos a utilizar a modo de ejemplo es el siguiente:

El encargado de formación de una empresa ha diseñado un curso de especialización con el que pretende mejorar el rendimiento de los trabajadores que realizan una determinada tarea. La mejora consistiría en que tardaran menos tiempo en realizar la tarea. Para comprobar la eficacia del curso antes de implantarlo en todos los trabajadores que realizan esta tarea, elige al azar una muestra de 35 trabajadores y para cada uno de ellos contabiliza el tiempo medio (en segundos) que tarda en ejecutar la tarea¹ en un turno, antes y después de recibir el curso de especialización.

Basándose en los datos de esa muestra, ¿puede concluir el encargado de formación que el curso es efectivo? (Utilícese un nivel de significación del 5 %).

Los datos se encuentran en el fichero *curso.txt*.

¹ Un mismo trabajador realiza la tarea muchas veces a lo largo de su turno, por lo que contabiliza el tiempo medio de ejecución a lo largo de un turno.

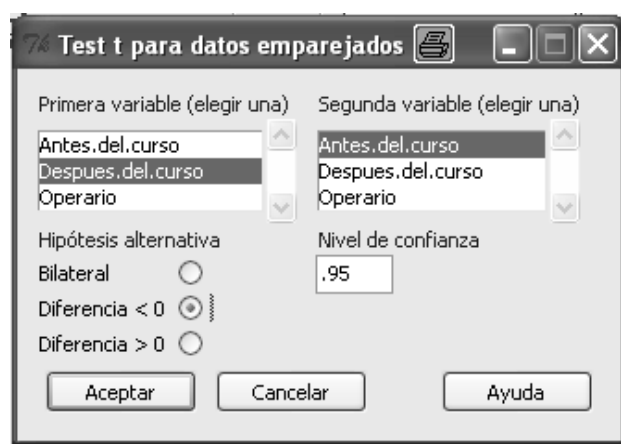


Figura 7.5: Prueba de igualdad de medias en poblaciones apareadas

Vamos a notar por μ_{DC} al promedio de los tiempos medios que tardan los trabajadores en realizar la tarea después del curso y por μ_{AC} al mismo promedio antes del curso. Nos piden contrastar $H_0 : \mu_{DC} = \mu_{AC}$ frente a $H_1 : \mu_{DC} < \mu_{AC}$.

7.2.3.1. Resolución del problema mediante R Commander

En primer lugar debemos cargar los datos. Debemos realizar un contraste de igualdad de medias en poblaciones apareadas. En este caso el conjunto de datos tiene exactamente la forma que R Commander necesita para ello, ya que tenemos dos variables, correspondientes a los tiempos medios antes y después del curso.

Elegimos, por tanto, la opción *Estadísticos* \rightarrow *Medias* \rightarrow *Test t para datos relacionados*, lo cual abrirá una ventana de entradas como la de la Figura 7.5. En ella ya he marcado las opciones requeridas por nuestro análisis. El resultado es el siguiente:

Paired t-test

data: Datos\$Despues.del.curso and Datos\$Antes.del.curso

t = -0.1221, df = 34, p-value = 0.4518

alternative hypothesis: true difference in means is less than 0

95 percent confidence interval:

-Inf 0.002225738

sample estimates:

mean of the differences

-0.0001731764

Vamos a analizar los resultados con detalle:

- En las dos primeras líneas se nos informa que estamos realizando un test t para muestras apareadas sobre los datos relativos a las variables `Despues.del.curso` y `Antes.del.curso` del conjunto de datos `Datos`. A propósito de ello, ¡no hemos dicho nada acerca de la hipótesis requerida en dicho test! Dado que la muestra es suficientemente amplia, podemos considerar que la diferencia media entre las dos variables sigue una distribución normal.
- En la siguiente línea aparece el valor del estadístico de contraste ($t = -0.1221$), de los grados de libertad ($df = 34$) y el p-valor ($p\text{-value} = 0.4518$). Visto el valor de éste podemos concluir que no existen evidencias en los datos de la muestra de que el curso disminuya el promedio que tardan los operarios en realizar la tarea. Observad que el p-valor es muy alto, luego las diferencias detectadas son mínimas, en absoluto significativas.
- A continuación aparece el tipo de hipótesis alternativa que hemos elegido.
- Posteriormente aparece un intervalo de confianza al 95 % para la diferencia de las medias. El hecho de que el cero sea un valor posible de dicho intervalo es otra forma de ver que no podemos rechazar la hipótesis nula en favor de la alternativa.
- Finalmente aparece el valor muestral de la diferencia de las medias.

7.2.3.2. Comentarios finales

En resumen, los datos no muestran el más mínimo indicio de que el curso disminuya el promedio que tardan los operarios en realizar la tarea.

7.2.4. Contrastes para medias mediante código. Función `t.test()`

En los apartados anteriores hemos visto cómo resolver problemas que involucran a la media de una población comparándola con un valor hipotético o a la media de dos poblaciones (independientes o apareadas), comparándolas entre sí. Lo que tienen en común estas pruebas es que todas ellas se basan en un estadístico de contraste que sigue una distribución t de Student.

Por esta razón, la resolución de ese tipo de problemas mediante código en la consola de R se realiza mediante la misma función, la función `t.test()`. Su sintaxis básica es la siguiente:

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

- x es un vector de datos correspondiente a una de las muestras o a la única muestra del problema. Si estamos haciendo un test sobre la media de una población, x contendrá la única

muestra; si estamos realizando un test de comparación de medias, x será la primera de las dos muestras.

- y correspondería con la segunda muestra en un test de comparación de medias. Si no es el caso y estamos en un test sobre una sola población, simplemente no se incluye.
- *alternative* especifica la dirección de la hipótesis alternativa. Como puede verse, tiene 3 posibles valores, "two.sided" (bilateral), "less" (unilateral a la izquierda) y "greater" (unilateral a la derecha).
- μ es el valor hipotético con el que se compara la media o la diferencia de medias en el contraste.
- *paired* especifica si las dos muestras x e y , en caso de que aparezcan, son apareadas o no.
- En el caso en el que aparecen dos muestras, *var.equal* especifica si podemos suponer varianzas iguales o no.
- *conf.level* es el nivel de confianza de los intervalos que se mostrarán asociados al test.

Contraste sobre una media En el caso del problema descrito en el apartado 7.2.1, debemos importar los datos, que se encuentran en el fichero *cebollas.txt*², y especificar cómo especificaríamos el test. Los resultados son exactamente los mismos que comentamos como salidas de R Commander:

```
Datos.cebollas<-read.table("cebollas.txt",header=TRUE,dec=",")
t.test(Datos.cebollas$Kilos,alternative="two.sided",mu=2)
```

Por lo tanto, viendo el p-valor (0.0723) podemos decir que no hay evidencias de que el peso medio de las bolsas de cebollas sea distinto de dos, el valor esperado.

Contraste sobre dos medias de poblaciones independientes De igual forma, para el problema que aparece en el apartado 7.2.2, veamos el código que proporciona las mismas salidas que R Commander:

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

²Es inmediato comprobar que el nombre de la variable es *Kilos*.

Ahora el p-valor sí indica que la media de capturas con la feromona es claramente superior a la media con el producto tradicional (p-valor inferior a 2.2×10^{-16}).

Si los datos estuvieran apilados, habría que indicarle que analice las capturas en función del tipo de trampa, de la siguiente manera:

```
t.test(Capturas~Trampa,data=Datos.feromona.apilados,  
alternative="greater",mu=0)
```

Contraste sobre las medias de dos poblaciones pareadas Finalmente, el problema del apartado 7.2.3 es de comparación de medias en poblaciones apareadas, por lo que el código sería el siguiente:

```
Datos.curso<-read.table("curso.txt",header=TRUE,dec=",")  
t.test(x=Datos.curso$Despues.del.curso,  
y=Datos.curso$Antes.del.curso,  
alternative="less",mu=0,paired=TRUE)
```

Vemos que el p-valor (0.4518) indica que no tenemos evidencias para afirmar que la media después del curso sea inferior a la media antes del curso.

7.3. Contraste para la proporción en una población

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

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

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

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

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

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

7.3.1. Enunciado y planteamiento del problema

El artículo “*Refinement of Gravimetric Geoid Using GPS and Leveling Data*” (W. Thurston, en Journal of Surveying Engineering, 2000:27-56) presenta un método para medir las alturas ortométricas por encima del nivel del mar. Para una muestra de 1225 puntos de partida, 926 dieron resultados que están dentro del espíritu de la clase C nivelando los límites de tolerancia. ¿Se puede concluir que este método produce resultados dentro de los límites de tolerancia más del 75 % de las veces?

Si notamos p a la proporción de resultados dentro de los límites de tolerancia, se nos está pidiendo que contrastemos $H_0 : p = 0.75$ frente a $H_1 : p > 0.75$.

7.3.2. Preparación de los datos y resolución del problema mediante R Commander

Antes de comenzar, conviene que carguemos todos los plugins de R Commander.

Tenemos dos formas de proporcionar los datos para realizar el test y dos formas de realizarlo:

1. Si tenemos los datos en un archivo, los cargamos y seleccionamos la opción del menú *Estadísticos* \rightarrow *Proporciones* \rightarrow *Test de proporciones para una muestra*.

Por ejemplo, los datos del problema están en el fichero *prop.rda*. Los cargamos y seleccionamos la opción del menú. La ventana de entrada emergente aparece en la Figura 7.6 a la izquierda. En ella tenemos que especificar:

- a) El valor hipotético en la hipótesis nula. En nuestro caso 0.75.
 - b) El sentido de la hipótesis alternativa. En nuestro caso unilateral a la derecha.
 - c) Un nivel de confianza para el intervalo de confianza resultante.
 - d) El tipo de prueba. Podemos elegir entre la prueba χ^2 con o sin corrección por continuidad o la prueba binomial exacta.
2. Si no tenemos los datos en un archivo basta con que sepamos cuál es el número de éxitos y fracasos en la muestra. En nuestro caso, 926 éxitos y 299 fracasos. En este caso necesitamos

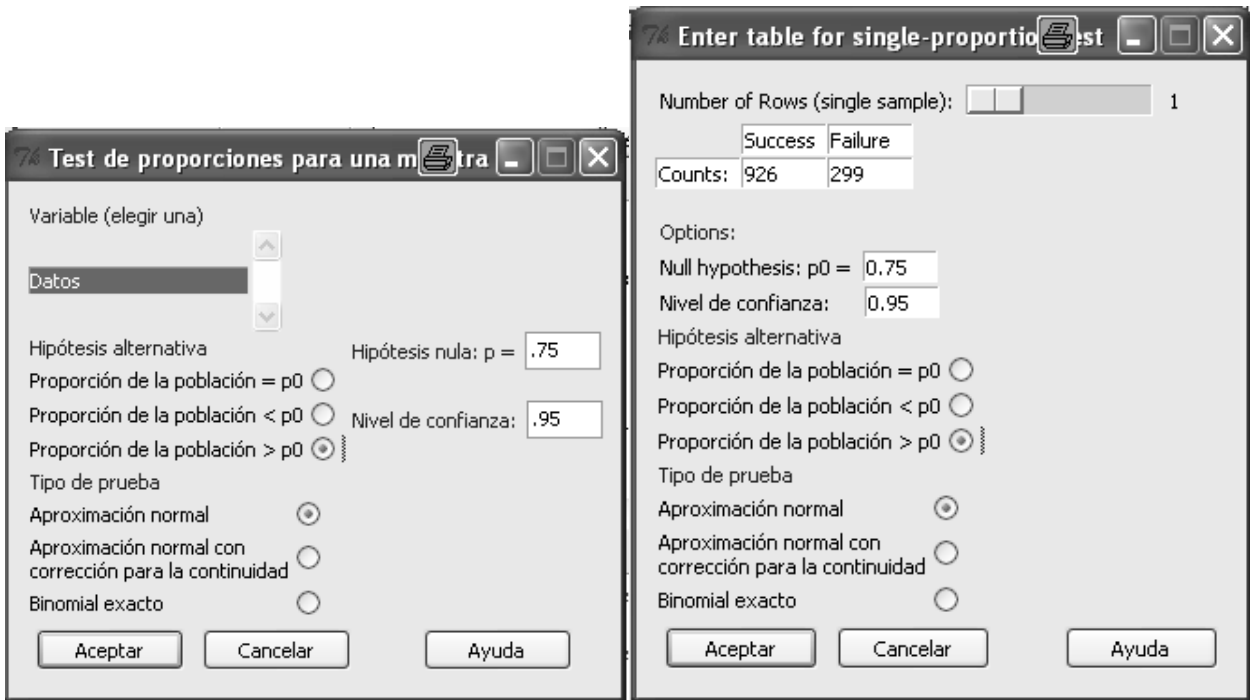


Figura 7.6: Test para una proporción

cargar el plugin *RcmdrPlugin.IPSUR*. Seleccionamos la opción del menú *Estadísticos* → *Proporciones* → *IP SUR Enter table for single-sample*. En la ventana emergente (Figura 7.6 a la derecha) tenemos que especificar:

- El número de éxitos (926) y fracasos (299).
- El valor hipotético en la hipótesis nula. En nuestro caso 0.75.
- El sentido de la hipótesis alternativa. En nuestro caso unilateral a la derecha.
- Un nivel de confianza para el intervalo de confianza resultante.
- El tipo de prueba. Dejamos la opción por defecto.

En ambos casos la ventana de resultados muestra lo mismo:

```
1-sample proportions test without continuity correction
```

```
data: rbind(.Table), null probability 0.75
```

```
X-squared = 0.2288, df = 1, p-value = 0.3162
```

```
alternative hypothesis: true p is greater than 0.75
```

```
95 percent confidence interval:
```

```
0.7351821 1.0000000
```

```
sample estimates:
```


p

0.7559184

Vamos a analizarlos:

- Especifica en primer lugar que se trata de un test para la proporción en una muestra.
- Especifica a continuación el valor hipotético en la hipótesis nula.
- Proporciona el valor del estadístico de contraste y, lo que más nos interesa, el p-valor.

En este caso el p-valor es bastante superior a 0.05, luego a la luz de estos datos no podemos rechazar la hipótesis nula en favor de la alternativa, es decir, no podemos concluir que el porcentaje de resultados dentro de los límites de tolerancia esté por encima del 75 %.

- A continuación recuerda la hipótesis alternativa.
- Facilita un intervalo de confianza (en este caso unilateral a la derecha) para la proporción.
- Finalmente, muestra la proporción muestral.

7.3.3. Resolución mediante código. La función `prop.test()`

Vamos a partir del hecho de que conocemos el número de éxitos y fracasos en la muestra. Si no fuera así, sino que tenemos los datos en una hoja de datos, podemos rápidamente tabularla mediante la función `table()` a la que sólo hay que especificarle la hoja de datos a tabular y, si ésta tuviera más de una variable, cuál de ellas queremos tabular.

La sintaxis de la función `prop.test` es la siguiente. Dicha sintaxis también nos servirá para los contrastes de comparación de dos proporciones:

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

Vamos a comentar con detalle cada uno de los argumentos de la función:

- *x* puede especificar dos cosas. O bien simplemente el número de éxitos, o bien, mediante una matriz de dos columnas, el número de éxitos y de fracasos en cada muestra.
- *n* especifica el número de datos de la muestra en el caso en que *x* sea el número de éxitos, y es ignorado en el caso en que *x* proporcione también el número de fracasos.

- p es el vector de probabilidades de éxito bajo la hipótesis nula. Debe ser un vector de la misma dimensión que el número de elementos especificado en x .
- *alternative* especifica la dirección de la hipótesis alternativa, tomando los valores *"two.sided"*, *"greater"* o *"less"*.
- *conf.level* es el nivel de confianza de los intervalos que se muestran entre los resultados.
- *correct* especifica si se usa la corrección por continuidad de Yates. Obsérvese que la opción por defecto es que sí se use esta corrección.

El ejemplo anterior podemos ejecutarlo rápidamente de dos formas:

1. `prop.test(x=926,n=1225,p=0.75,alternative="greater",correct=FALSE)` o bien
2. `prop.test(x=cbind(926,299),p=0.75,alternative="greater",correct=FALSE)`. En esta línea, la función *cbind()* sirve para concatenar los valores que aparecen entre paréntesis como columnas de una matriz.

Comentar finalmente que la prueba binomial exacta puede realizarse mediante la función *binom.test*, cuya sintaxis básica es similar a la de *prop.test*:

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

7.4. Contraste para la diferencia de proporciones

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

7.4.1. Enunciado y planteamiento del problema

Vamos a trabajar sobre el siguiente enunciado:

En el artículo "Strategic Management in Engineering Organizations" (Journal of Management in Engineering, 2001:60-68) se comparaba el número de compañías privadas (133) que contestaron completamente a una encuesta, de un total de 400 compañías privadas a las que se les mandó la encuesta con el número de compañías públicas (50) de

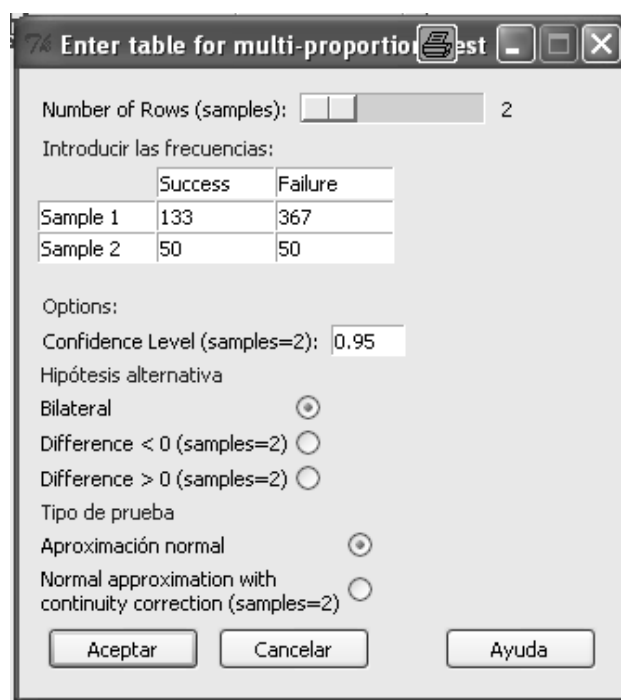


Figura 7.7: Test para la comparación de proporciones

un total de 100 compañías públicas a las que se les envió la encuesta. ¿Podemos concluir que la tasa de respuesta es diferente entre compañías públicas y privadas? (5 % de significación)

Tenemos, por tanto, que contrastar $H_0 : p_{privadas} = p_{publicas}$ frente a $H_1 : p_{privadas} \neq p_{publicas}$, donde p_{XXX} son las probabilidades de que una empresa, pública o privada, responda a la encuesta.

7.4.2. Resolución del problema mediante R Commander

Antes de comenzar hay que decir que, al igual que en el ejemplo anterior, podríamos tener los datos en un archivo y tratarlos directamente desde ahí³, mediante la opción *Estadísticos* → *Proporciones* → *Test de proporciones para dos muestras*. Sin embargo, tan sólo es necesario conocer el número de éxitos y fracasos en cada una de las dos muestras, utilizando la opción *Estadísticos* → *Proporciones* → *IPSUR Enter table for independent samples*. En la ventana emergente (Figura 7.7) tenemos que especificar:

- Número de éxitos y fracasos en la primera muestra. En nuestro caso, 133 y 367.
- Número de éxitos y fracasos en la segunda muestra. En nuestro caso, 50 y 50.

³Tendrían que estar en una única columna, con una segunda variable tipo factor que distinga a qué muestra pertenece cada dato. Recordemos la opción *Apilar* que ya hemos trabajado para eso.

- Un nivel de significación para el intervalo de confianza.
- El sentido de la hipótesis alternativa. En nuestro caso, bilateral.
- El tipo de test. En este caso dejamos la opción por defecto, pero podemos elegir corrección por continuidad o prueba exacta de Fisher.

Los resultados son los siguientes:

```
2-sample test for equality of proportions without continuity
correction
data: .Table
X-squared = 21.5261, df = 1, p-value = 3.49e-06
alternative hypothesis: two.sided
95 percent confidence interval:
-0.3393741 -0.1286259
sample estimates:
prop 1 prop 2
0.266 0.500
```

Lo que realmente nos interesa es el p-valor, que aparece en la tercera línea, $3.49\text{e-}06$, y que indica que se puede rechazar la hipótesis nula en favor de la alternativa, es decir, podemos, con los datos existentes, asegurar con un 95 % de confianza que la tasa de respuesta es diferente entre compañías públicas y privadas.

La prueba exacta de Fisher se utiliza para contrastar la relación entre las dos variables cualitativas que constituyen la tabla de contingencia, pero no admite la posibilidad de contrastar si una proporción en una población es mayor que en otra. Para realizar una prueba de este tipo con R Commander necesitamos el plugin *RcmdrPlugin.HH*. Tras cargarlo, elegimos *Estadísticos* → *Tablas de contingencia* → *Enter and analyze two-way table...* (*HH*). En la ventana emergente introducimos las frecuencias de la tabla y elegimos el tipo de prueba.

7.4.3. Resolución mediante código

El contraste se realiza de nuevo mediante la función `prop.test()`, pero previamente debemos comentar algo acerca de cómo introducir los datos.

Los éxitos y los fracasos de cada muestra deben ir en dos filas de una matriz de dos columnas, es decir, con una estructura como la siguiente:

Nº de éxitos de la muestra 1 (n_{11})	Nº de fracasos de la muestra 1 (n_{12})
Nº de éxitos de la muestra 2 (n_{21})	Nº de fracasos de la muestra 2 (n_{22})

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

- `matrix(c(n_{11} , n_{12} , n_{21} , n_{22}), 2, 2, byrow=TRUE)` o bien
- `matrix(c(n_{11} , n_{21} , n_{12} , n_{22}), 2, 2).`

En el ejemplo, sería de la siguiente forma:

```
tabla<-matrix(c(133,367,50,50),2,2,byrow=TRUE)
```

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

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

Decir por último que la prueba exacta de Fisher se realiza con la función *fisher.test*, que tiene una sintaxis parecida a la de *prop.test*, aunque admite muchos más argumentos en función de las características de la tabla de contingencia a analizar. Para usuarios interesados, recomiendo el uso de la ayuda mediante *?fisher.test*.

7.5. Contraste para la comparación de varianzas

7.5.1. Enunciado y planteamiento del problema

En una comercializadora de aceite de oliva se envasan botellas de 1 litro pero, como ocurre en cualquier proceso industrial, el volumen de llenado de cada botella no es exactamente de un litro, sino que se producen variaciones aleatorias en el volumen real con el que se llenan. El ingeniero responsable del control de calidad es consciente del problema que supone esta variabilidad en el volumen de llenado.

De cara a tratar de reducirla, la empresa se plantea cambiar la máquina de llenado, pero previamente quiere probarla para garantizarse que la inversión merece la pena. Para ello calibra la máquina para llenados de 1 litro y recopila los datos de llenado de 100 botellas. Un análisis exploratorio inicial no le hace pensar que estos datos se salgan de la hipótesis de normalidad. También realiza un muestreo de otras 100 botellas con la vieja máquina, en las mismas condiciones (horario, operarios que extraen las muestras, etc.) que las

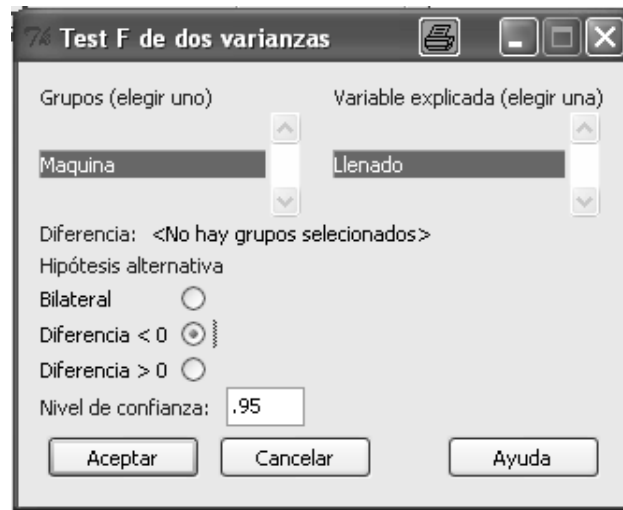


Figura 7.8: Test de igualdad de varianzas

100 botellas muestreadas con la nueva máquina. De nuevo no hay evidencias de que los datos se separen de la normalidad.

A la luz de esos datos, ¿puede concluir el ingeniero de control de calidad que la nueva máquina de llenado disminuye la variabilidad del volumen con el que se llenan las botellas? (Utilícese un nivel de significación del 5%).

Los datos se encuentran en el fichero *aceite.txt*.

Lo que vamos a plantear para resolver el problema es un contraste de igualdad de varianzas (o de desviaciones típicas). Si notamos σ_V a la desviación típica del volumen de llenado bajo el viejo proceso y σ_N a la desviación típica del volumen de llenado bajo el nuevo proceso, se trata de contrastar $H_0 : \sigma_V = \sigma_N$ frente a $H_1 : \sigma_V > \sigma_N$.

De nuevo vamos a asumir desde el principio que los datos proceden de distribuciones normales.

7.5.2. Resolución del ejercicio mediante R Commander

Al igual que hicimos allí, necesitamos que los datos se encuentren apilados en una única variable y que una segunda variable tipo factor distinga de qué muestra procede cada dato. Eso ya lo hicimos anteriormente.

Elegimos la opción *Estadísticos* \rightarrow *Varianzas* \rightarrow *Test F para dos varianzas*. En la ventana de entradas hay que especificar:

- La variable que define los grupos (*Maquina* en nuestro caso) y la variable explicada (*Llenado*).

- La hipótesis alternativa. Aquí es importante que recordemos que el factor (*Maquina*) se ordena alfabéticamente, luego la primera muestra es la del proceso nuevo y la segunda la del viejo. Por tanto, dado que queremos la alternativa $H_1 : \sigma_V > \sigma_N$, tenemos que elegir la opción *Diferencias* < 0. Eso es porque *Diferencias* se refiere a la diferencia entre el primer grupo (nueva) y el segundo grupo (vieja).
- El nivel de confianza para el intervalo de confianza.

Los resultados son los siguientes:

```
F test to compare two variances
```

```
data: Llenado by Maquina
```

```
F = 1.1016, num df = 99, denom df = 99, p-value = 0.6844
```

```
alternative hypothesis: true ratio of variances is less than 1
```

```
95 percent confidence interval:
```

```
0.000000 1.535685
```

```
sample estimates:
```

```
ratio of variances
```

```
1.101590
```

En la tercera línea podemos ver que aparece el valor del estadístico F , los grados de libertad en el numerador y el denominador y el p-valor. Ese $p = 0.6844$ indica que no hay suficientes evidencias en los datos para concluir que el proceso nuevo disminuya significativamente la varianza de los diámetros. De hecho, observemos que las propias varianzas muestrales indican que la varianza de la nueva máquina es mayor que la de la vieja máquina.

7.5.3. Resolución mediante código. La función *var.test*

Esta función tiene una sintaxis muy parecida a las anteriores:

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

- x corresponde al vector de datos de la primera muestra.
- y es el vector de datos de la segunda muestra.

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

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

```
Datos.aceite<-read.table("aceite.txt",header=TRUE,sep="\t",dec=",")  
  
var.test(Datos.aceite$Nueva.maquina,Datos.aceite$Vieja.maquina,  
alternative="less")
```

Si los datos estuvieran apilados, por ejemplo, en la hoja `Datos.aceite.apilados`, cuyas variables fueran `Litros` para el volumen de llenado de las botellas y `Tipo` para el tipo de máquina, la sintaxis sería

```
var.test(Datos.aceite.apilados$Litros~Datos.aceite.apilados$Tipo,  
alternative="less")
```

7.6. ANOVA de un factor

7.6.1. Enunciado y planteamiento del problema

Una compañía química recoge información sobre las concentraciones máximas por hora (en $\mu\text{g}/\text{m}^3$) de SO_2 para cuatro de sus plantas de energía. ¿Los resultados permiten concluir a la compañía que hay diferencias entre las concentraciones máximas por hora entre las cuatro plantas? (Utilícese un nivel de significación del 5 %).

Los datos se encuentran en el fichero *SO2.txt*. Se refieren a 4 lugares, y se nos pide valorar si se detectan diferencias (significativas) entre ellos. Asumiendo que se trate de datos procedentes de distribuciones normales con varianzas iguales, vamos a abordar el problema mediante una prueba ANOVA.

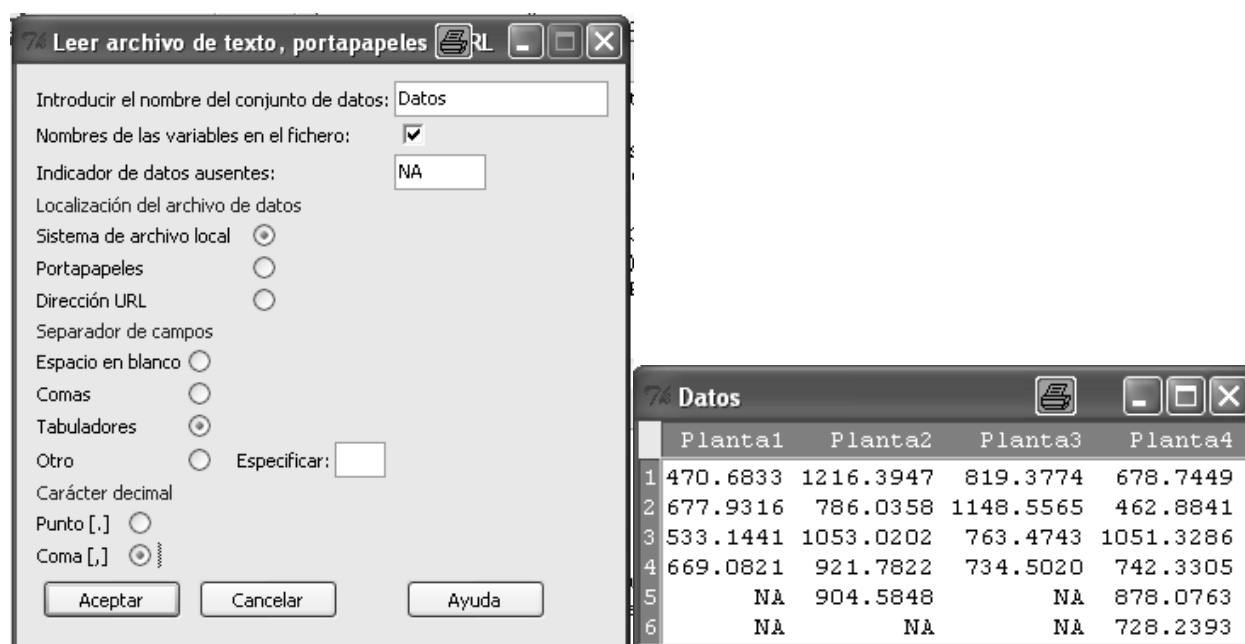


Figura 7.9: Importando los datos del contenido de TiO_2 del carbón

7.6.2. Resolución mediante R Commander

En primer lugar, como es obvio, debemos importar los datos. Si abrimos el fichero *SO2.txt* con el bloc de notas veremos que los nombres de los lugares aparecen en la primera fila y que los datos están separados por tabulaciones. Esta es toda la información que necesitamos para importar los datos (Figura 7.9 a la izquierda). El aspecto que tiene el conjunto de datos tras la importación aparece en la Figura 7.9, a la derecha.

Observemos que algunos de los valores aparecen como *NA*, es decir, como no disponibles, como datos faltantes. Eso es debido a que la muestra no es del mismo tamaño en los 4 lugares donde se muestrea, por lo que el editor tiene que *rellenar* con valores *NA* las casillas donde no hay valor muestral.

Eso nos hace ver que aquí tenemos el mismo problema que con el contraste de muestras independientes: hay que transformar los datos para que R Commander realice el análisis. Lo que tenemos que hacer es lo mismo que hicimos allí, apilar los datos para que aparezcan todos ellos en una misma columna y con una nueva variable que indique el lugar de donde procede el valor muestral. En la Figura 7.10 a la izquierda aparece la ventana de entradas de la opción *Datos* → *Conjunto de datos activo* → *Apilar variables del conjunto de datos*. A la derecha aparecen los datos apilados. Si nos fijamos, hay algunos valores *NA*. No tenemos que hacerles caso, ya que son aquellos que han sido apilados de los originales, pero R los ignorará.

Los datos ya están dispuestos para ser analizados. Seleccionamos la opción *Estadísticos* → *Medias*

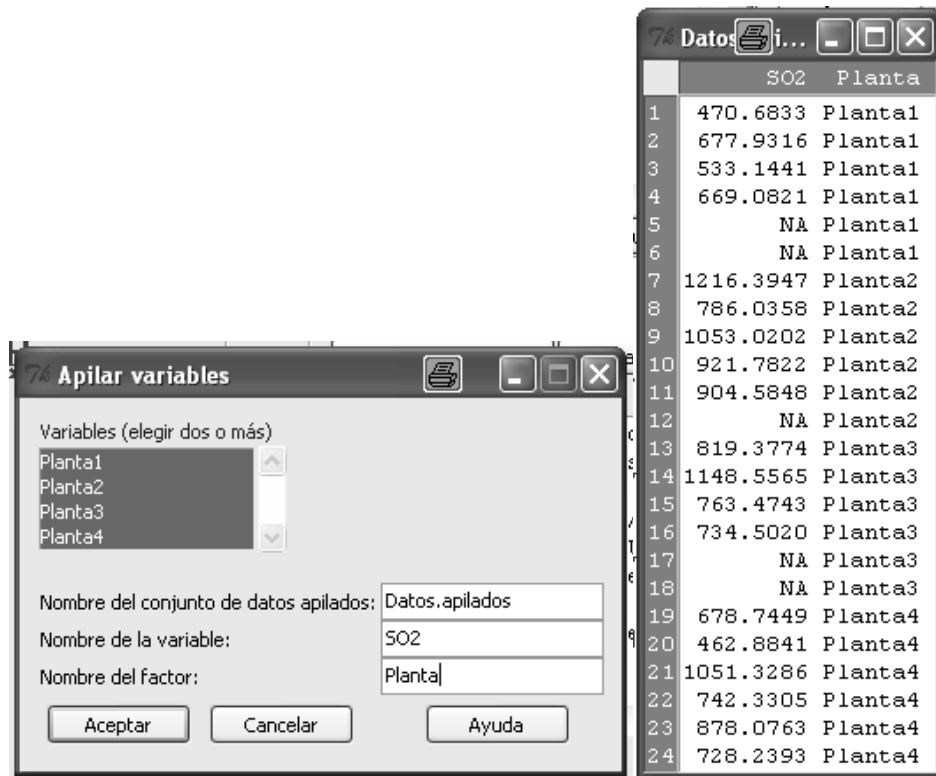
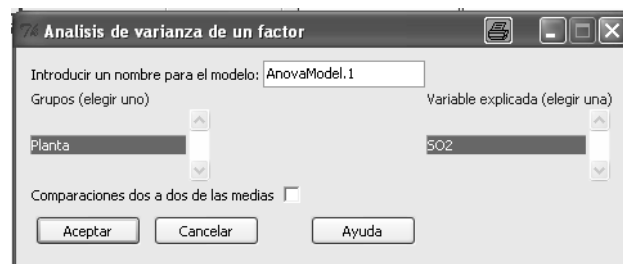
Figura 7.10: Apilando los datos del contenido de TiO_2 del carbón

Figura 7.11: Contraste de igualdad de medias en poblaciones apareadas

→ *ANOVA de un factor* y se abre una ventana de entradas como la de la Figura 7.11. En ella hemos de seleccionar la variable que estamos analizando así como el factor que distingue los grupos, en nuestro caso, los lugares donde se tomaron las muestras.

Los resultados aparecen en la Figura 7.12:

- La variable de respuesta, es decir, la variable que estamos analizando como posiblemente afectada por el factor, es *SO2*.
- A continuación aparecen los grados de libertad, la suma de los cuadrados y la media de los cuadrados entre grupos y dentro de los grupos.
- Aparece en la penúltima columna el valor del estadístico de contraste F , 4.1043.

```

      Df Sum Sq Mean Sq F value    Pr(>F)
Planta    3 364508   121503    4.1043 0.02595 *
Residuals 15 444055    29604
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
5 observations deleted due to missingness

> numSummary(Datos.apilados$SO2 , groups=Datos.apilados$Planta,
+   statistics=c("mean", "sd"))
      mean      sd n NA
Planta1 587.7103 102.3622 4  2
Planta2 976.3635 164.2005 5  1
Planta3 866.4776 191.3236 4  2
Planta4 756.9340 197.4635 6  0

```

Figura 7.12: Resultados del ANOVA

- En la última columna aparece el p-valor del contraste, 0.02595.

Visto este p-valor, podemos concluir que con los datos muestrales aportados se detectan diferencias significativas entre los niveles medios de *SO2* medidos en los 4 lugares.

- Aparecen al final los valores muestrales de las medias, las desviaciones típicas y el número de datos de cada muestra.

7.6.3. Resolución mediante código. La función *aov*

La sintaxis de esta función varía ligeramente de las anteriores porque obliga a presentar los datos en forma de modelo, a través de una fórmula. Concretamente, necesitamos, como en R Commander, que todos los datos de todas las muestras estén contenidos en una única variable, y que una segunda variable de tipo factor especifique a qué grupo pertenece cada dato.

Imaginemos que x_1 es un vector de dimensión n_1 conteniendo la primera muestra, x_2 es un vector de dimensión n_2 que contiene la segunda muestra y así sucesivamente hasta los k grupos del problema. Una forma muy simple de generar los datos necesarios para aplicar *aov* y que es equivalente a la opción *Apilar datos* de R Commander es

```

Datos<-data.frame(Variable=c(x1,x2,...,xk),
Grupo=factor(c(rep(1,n1),rep(2,n2),...,rep(k,nk))))

```

Una vez que tenemos preparados los datos, veamos cómo es la sintaxis básica de *aov*:

```

aov(Variable~Grupo,data=Datos))

```

Como podemos ver, sólo tenemos que especificar el nombre de la variable que contiene los datos (*Variable*), el nombre del factor que distingue a qué grupo pertenece cada dato (*Grupo*) y el nombre de la hoja de datos (*Datos*). La expresión *Variable~Grupo* se conoce en Estadística como la *fórmula del modelo*. Lo que viene a decir es que se trata de explicar la variabilidad de la variable *Variable*

mediante el conocimiento de la variable *Grupo*⁴.

Finalmente, hay que decir que al ejecutar la función *aov* sólo se muestra en pantalla la partición de la varianza que resulta del ANOVA, pero no la tabla de éste que permite concluir el contraste. Para obtener la tabla de ANOVA es necesario aplicar la función *summary()* al resultado de *aov()*. Coadunadamente, habría que ejecutar

```
ANOVA<-aov(Variable~Grupo,data=Datos))
```

```
summary(ANOVA)
```

o simplemente

```
summary(aov(Variable~Grupo,data=Datos)))
```

Con todo, la resolución del ejemplo se lograría de la siguiente forma:

```
Datos<-read.table("S02.txt2",header=TRUE,sep="\t",dec=",")
```

```
x1<-Datos$Planta1[is.na(Datos$Planta1)==0]
```

```
n1<-length(x1)
```

```
x2<-Datos$Planta2[is.na(Datos$Planta2)==0]
```

```
n2<-length(x2)
```

```
x3<-Datos$Planta3[is.na(Datos$Planta3)==0]
```

```
n3<-length(x3)
```

```
x4<-Datos$Planta4[is.na(Datos$Planta4)==0]
```

```
n4<-length(x4)
```

```
Datos<-data.frame(Variable=c(x1,x2,x3,x4),
```

```
Grupo=factor(c(rep(1,n1),rep(2,n2),rep(3,n3),rep(4,n4))))
```

```
summary(aov(Variable~Grupo,data=Datos))
```

Recordemos que al importar los datos, se han colado algunos valores ausentes, ya que no todas las muestras tienen el mismo número de datos. Evitamos cualquier confusión cuando definimos los vectores *x1*, *x2*, *x3*, *x4* con sólo los datos que no son *NA*, a través de la función *is.na()*. Esta función es un operador lógico que será 1 o TRUE si el argumento es *NA* y será 0 o FALSE si no lo es.

⁴El símbolo ~ se obtiene mediante la combinación de teclas ALT+126 o ALT GR+4.

Capítulo 8

Contrastes de hipótesis no paramétricos

Objetivos:

1. Realizar contrastes χ^2 de bondad de ajuste para evaluar los ajustes mediante distribuciones discretas.
2. Realizar contrastes de Kolmogorov-Smirnoff de bondad de ajuste para evaluar los ajustes mediante distribuciones continuas.
3. Realizar contrastes χ^2 de independencia para valorar la relación existente entre variables cualitativas.

8.1. Contrastes de bondad de ajuste

R Commander no dispone de opciones de menú para realizar los contrastes de bondad de ajuste que hemos estudiado. Es por ello que debemos utilizar el código de R directamente para dichos contrastes.

8.1.1. Contraste χ^2 de bondad de ajuste

En el caso del test χ^2 de bondad de ajuste debemos instalar y cargar un paquete adicional de R llamado *vcd*¹. Recordemos que para ello debemos hacer lo siguiente:

1. Ejecutar la opción del menú de R *Paquetes* \rightarrow *Instalar paquete(s)*. Nos pedirá un *mirror* desde el que descargar el paquete, eligiendo *Spain*. Después buscamos el paquete *vcd* en la lista emergente.

¹David Meyer, Achim Zeileis, and Kurt Hornik (2009). *vcd*: Visualizing Categorical Data. R package version 1.2-4.

2. Ejecutar la opción del menú de R *Paquetes* \rightarrow *Cargar paquete*, eligiendo *vcd*.

Este paquete permite ajustar los parámetros de las distribuciones mediante un método distinto al de máxima verosimilitud, llamado método de la mínima χ^2 , y realizar el test χ^2 de bondad de ajuste para las distribuciones de Poisson, binomial y binomial negativa. Indirectamente también permite la distribución geométrica, considerando que ésta es un caso particular de la binomial negativa.

La función para ello es `goodfit()`, cuya sintaxis básica es:

```
ajuste<-goodfit(datos, type="distribución", par, method="MinChisq")
```

Fijaros que le hemos puesto nombre al resultado de la función (*ajuste*). Eso es para poder sacar de él varios resultados, como veremos enseguida. Vamos a especificar los argumentos con detalle:

1. **datos** es el vector de datos de la muestra.
2. **type** es la distribución que consideramos en el ajuste. Los valores posibles son *"poisson"*, *"binomial"* y *"nbinomial"*.
3. **par** es la lista de los parámetros de la distribución del ajuste. Sólo lo usaremos para el caso de la geométrica. Teniendo en cuenta que una $Geo(p)$ es una $BN(1, p)$, si queremos el contraste para un ajuste mediante una distribución $Geo(p)$, tenemos que poner `type="nbinomial"` y `par=list(size=1,prob=p)`.
4. `method="MinChisq"` es la opción para que realice el test χ^2 . Debemos dejarlo fijo siempre.

Los resultados que `goodfit()` proporciona son los siguientes:

1. `summary(ajuste)` dará el valor del estadístico χ^2 , los grados de libertad y el p-valor del test. Hay que decir que no agrupa casillas con frecuencias esperadas inferiores a 5. Si hay alguna casilla así, aparecerá un mensaje de alarma.
2. `plot(ajuste)` dibuja la función masa junto con el diagrama de barras del ajuste, en escala de raíz cuadrada. Las barras del diagrama aparecen alineadas con la función masa, de manera que la falta de ajuste se aprecia en la base de las barras.
3. `ajuste` devuelve la tabla de frecuencias observadas y esperadas.
4. `ajuste$par` devuelve la estimación de los parámetros por el método de la mínima χ^2 .

Vamos a volver sobre los mismo ejemplos que utilizamos en la práctica sobre estimación por máxima verosimilitud. Recordad que allí calculamos las estimaciones y dibujamos las funciones masa junto con los diagramas de barras, pero comentamos que aún no podíamos decidir si los ajustes eran *buenos* o *malos*. Ahora es el momento de terminar con esa cuestión.

8.1.1.1. Para la distribución de Poisson

En el Cuadro 8.1 aparecen de nuevo los datos sobre el número de muertes en las 20 compañías del ejército prusiano debidas a coces de caballos. Esos datos están en el fichero *DatosMuertesCoces.rda*. En su momento ajustamos para esos datos una distribución *Poisson*(0.61). Si ese ajuste fuese *bueno*, podríamos concluir que las muertes se producen por puro azar en las distintas compañías.

Muertes/año	Frecuencia
0	109
1	65
2	22
3	3
4	1

Cuadro 8.1: Distribución de frecuencias en el ejemplo del ejército prusiano

Carguemos los datos del fichero. El conjunto de datos y la variable del conjunto de datos se llaman ambos *muertes*. Para realizar el test χ^2 de bondad de ajuste debemos ejecutar el siguiente código:

1. `ajuste.poisson <- goodfit(Datos.muertes$Muertes, type = "poisson", method="MinChisq")`.

Esto generará el objeto llamado `ajuste.poisson` que contiene los resultados del test χ^2 .

2. `ajuste.poisson$par` devuelve lo siguiente:

```
$lambda
```

```
[1] 0.6139969
```

Por lo tanto, el ajuste por el método de la mínima χ^2 es una *Poisson*(0.614). La diferencia entre la estimación máximo-verosímil y la de la mínima χ^2 es de menos de 4 milésimas.

3. `summary(ajuste.poisson)`. Devuelve el siguiente resultado:

```
Goodness-of-fit test for poisson distribution
```

```
X^2 df P(> X^2)
```

```
Pearson 0.594649 3 0.8976563
```

Tenemos, por tanto, que el valor del estadístico de contraste es 0.594, los grados de libertad son 3 y el p-valor es 0.8976563. Ese p-valor nos permite aceptar la hipótesis nula, es decir, concluir que no existen evidencias en los datos en contra de que estos se ajusten a una *Poisson*(0.614). De hecho, el p-valor es altísimo, lo que indica que el ajuste es excelente.

4. `plot(ajuste.poisson)`. Devuelve la Figura 8.1. Pueden verse sólo ligeros desajustes en las bases de las barras.

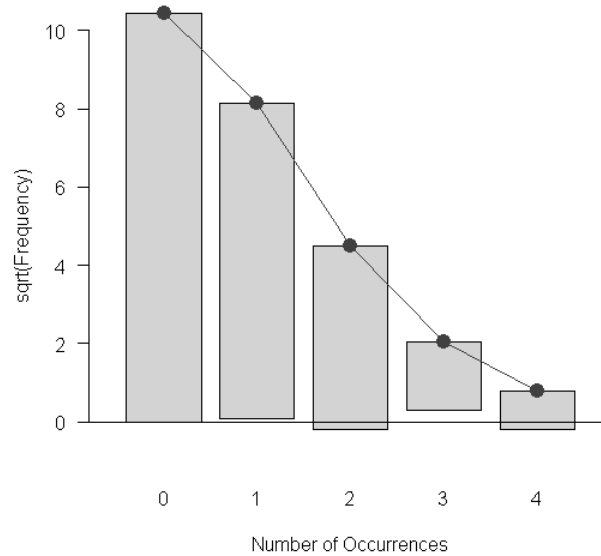


Figura 8.1: Diagrama de barras y ajuste de Poisson en el ejemplo de las muertes por coces de caballos

5. `ajuste.poisson` devuelve la siguiente tabla:

```
Observed and fitted values for poisson distribution
with parameters estimated by 'MinChisq'

count observed fitted
0 109 108.2366935
1 65 66.4569976
2 22 20.4021963
3 3 4.1756286
4 1 0.6409558
```

Podemos ver que las dos últimas frecuencias esperadas son inferiores a 5. Por ello, en la ventana de mensajes aparece lo siguiente:

```
AVISO: Warning in summary.goodfit(ajuste.poisson) : Chi-squared approximation may
be incorrect
```

8.1.1.2. Para la distribución geométrica

Los datos que manejamos se referían a los tiempos (en *ms*) que transcurren entre una muestra aleatoria de 250 paquetes en una determinada transmisión telemática, datos que se encuentran en el fichero *trafico.telem.rda*.

Cargamos los datos. El conjunto de datos se llama *Datos.trafico* y la variable *tiempos.entre.paquetes*. Para realizar el ajuste ejecutamos el siguiente código:

```
1. ajuste.geom <- goodfit(Datos.trafico$tiempos.entre.paquetes, type = "nbinomial",  
  method="MinChisq", par=list(size=1)).
```

```
2. ajuste.geom$par devuelve lo siguiente:
```

```
$size
```

```
[1] 1
```

```
$prob
```

```
[1] 0.1129637
```

Por lo tanto, el ajuste por el método de la mínima χ^2 es una *Geo* (0.113). En su momento vimos que la estimación por máxima verosimilitud de p era 0.117591722. De nuevo la diferencia entre ambas estimaciones es del orden de milésimas.

```
3. summary(ajuste.geom). Devuelve el siguiente resultado:
```

```
Goodness-of-fit test for nbinomial distribution
```

```
X^2 df P(> X^2)
```

```
Pearson 14.66274 29 0.9874704
```

Tenemos, por tanto, que el valor del estadístico de contraste es 0.594, los grados de libertad son 3 y el p-valor es 0.987. Ese p-valor nos permite aceptar la hipótesis nula, es decir, concluir que no existen evidencias en los datos en contra de que estos se ajusten a una *Poisson* (0.614). De hecho, el p-valor es altísimo, lo que viene a decir que el ajuste es excelente.

```
4. plot(ajuste.geom). Devuelve la Figura 8.2. A pesar de que el p-valor indica que el ajuste es excelente, se observan bastantes desajustes, incluso en los valores iniciales que, al tener frecuencias mayores, deberían estar mejor ajustados.
```

```
5. ajuste.geom devuelve la tabla de frecuencias observadas y esperadas. Es demasiado larga para ponerla aquí, pero podeis ver que hay multitud de casillas que deberían ser agrupadas. Eso puede estar provocando que el p-valor no corresponda con la realidad de la distribución del estadístico de contraste.
```

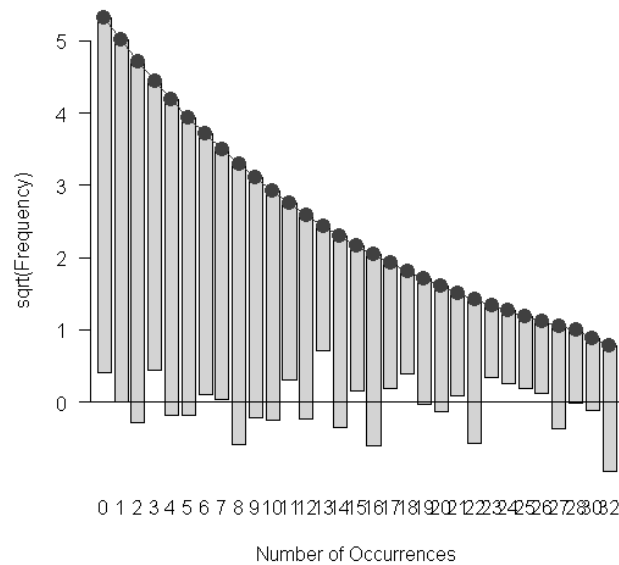


Figura 8.2: Diagrama de barras y ajuste geométrico de los datos de tráfico telemático

8.1.1.3. Para la distribución binomial negativa

Con los mismos datos tratamos de ajustar una distribución binomial negativa. Vamos a hacerlo de nuevo aquí:

```
1. ajuste.nbinom <- goodfit(Datos.trafico$tiempos.entre.paquetes,
  type = "nbinomial",method="MinChisq")
```

```
2. ajuste.nbinom$par devuelve
```

```
$size
```

```
[1] 1.165172
```

```
$prob
```

```
[1] 0.1301135
```

Si recordamos las estimaciones por máxima verosimilitud, vemos de nuevo que no son muy distintas de éstas.

```
3. summary(ajuste.nbinom) devuelve
```

```
Goodness-of-fit test for nbinomial distribution
```

```
X^2 df P(> X^2)
```

```
Pearson 12.65186 28 0.9943238
```

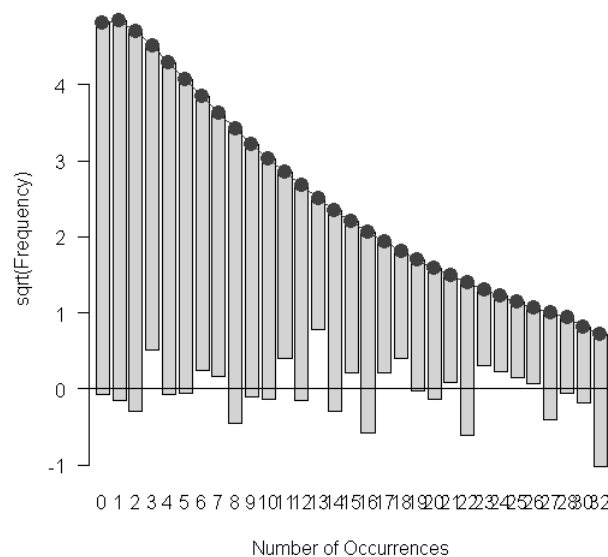


Figura 8.3: Diagrama de barras y ajuste geométrico de los datos de tráfico telemático

Tampoco hay evidencias en contra del ajuste mediante una $BN(1.165, 0.130)$. Además, el p-valor es superior aún al del ajuste mediante la distribución geométrica, luego éste es mejor que aquel. No obstante, al igual que antes, también hay muchas casillas con frecuencias esperadas inferiores a 5.

4. `plot(ajuste.nbinom)` devuelve la Figura 8.3. Los desajustes son ligeramente inferiores.

8.1.1.4. Para la distribución binomial

Para finalizar, consideremos el siguiente ejemplo. Se analizan camadas de ratas que dan a luz a 20 congéneres. Se les suministra una droga para analizar su toxicidad y se contabilizan el número de crías que sobreviven hasta los dos meses. Los datos, referentes a 150 camadas de 20 crías, se encuentran en el fichero `crias.camadas.rda`, donde el conjunto de datos activo tiene el mismo nombre, `crias.camadas` y la variable se llama `crias.supervivientes`. Vamos a ver si una distribución binomial es adecuada para ajustar esos datos, en cuyo caso podríamos inferir que el comportamiento de las camadas es independiente y que la probabilidad de supervivencia al fármaco es constante. Querríamos, además, una estimación de esta probabilidad de supervivencia.

Cargamos el fichero con los datos. El código a ejecutar es el siguiente:

1. `ajuste.binom <- goodfit(crias.camadas$crias.supervivientes, type = "binomial", method="ML", par=list(size=20))`. Aquí es importante incluir el parámetro $n = 20$, ya que, de no hacerlo,

R consideraría como valor de n el máximo valor de la variable, que podría no ser 20 (de hecho no lo es).

2. `ajuste.binom$par` devuelve

```
$size
```

```
[1] 20
```

```
$prob
```

```
[1] 0.1607786
```

Es decir, la estimación por el método de la mínima χ^2 del parámetro p es $\hat{p} = 0.161$. La estimación por el método de máxima verosimilitud es $\frac{\bar{x}}{n} = 0.158$. En nuestro caso, el ajuste de los datos viene dado por una distribución $B(20, 0.161)$.

3. `summary(ajuste.binom)` devuelve

```
Goodness-of-fit test for binomial distribution
```

```
X^2 df P(> X^2)
```

```
Pearson 11.40937 7 0.1217325
```

El p-valor superior a 0.05 indica que no hay evidencias en contra del ajuste de los datos mediante la distribución $B(20, 0.161)$. No obstante, de nuevo hay casillas con frecuencias esperadas inferiores a 5.

4. `plot(ajuste.binom)` devuelve la Figura 8.4. Debemos destacar, al menos, dos cuestiones:

- a) Hay un desajuste entre lo observado y lo esperado bastante importante en la frecuencia que constituye la moda, el valor 3.
- b) La gráfica termina en el valor 8 porque no se observan valores en la muestra superiores a 8. Según el modelo dado por la distribución $B(20, 0.161)$, la probabilidad de que se den valores superiores a 8 es 0.002.

8.1.2. Contraste de Kolmogorov-Smirnoff

Para el ejemplo con el que vamos a describir el contraste, consideremos los datos del fichero *tiempos.fallo.rda*, que en su momento tratamos de ajustar mediante una distribución exponencial. La hoja de datos se llamaba *Datos.tiempos.fallo* y la variable *Años*.

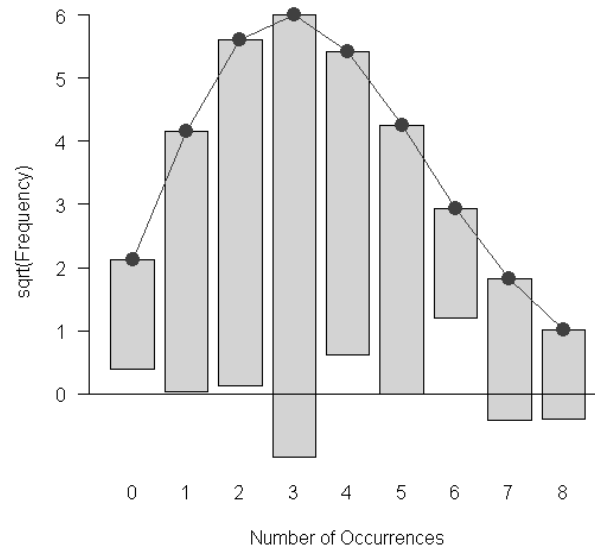


Figura 8.4: Ajuste de los datos de las crías supervivientes al fármaco mediante una distribución binomial

La distribución resultante del ajuste fue $\exp(0.492)$. Vamos a contrastar ahora la bondad de este ajuste. El código es el siguiente:

```
ks.test(Datos.tiempos.fallo$Años,"pexp",0.492)
```

El resultado es

One-sample Kolmogorov-Smirnov test

data: Datos.tiempos.fallo\$Años

D = 0.0485, p-value = 0.4805

alternative hypothesis: two-sided

Por lo tanto, no hay evidencias en los datos en contra del ajuste considerado mediante la distribución exponencial.

Fijaros que en el código para usar la función `ks.test()` sólo hay que introducir lo siguiente:

1. Los datos (`Datos.tiempos.fallo$Años`).
2. La función de distribución del modelo del ajuste, entre comillas. Por ejemplo, `"pexp"`, `"pgamma"`, o `"pnorm"` (en nuestro caso, la distribución exponencial).
3. Los parámetros de la distribución ajustada (en nuestro caso, $\lambda = 0.492$).

	Raza			
Grupo	Blancos	Negros	Mestizos	Total
A	510	78	175	763
B	116	48	62	226
AB	37	10	18	65
O	578	141	314	1033
Total	1241	277	569	2087

Cuadro 8.2: Distribución de frecuencias observadas en función de la raza y el grupo sanguíneo

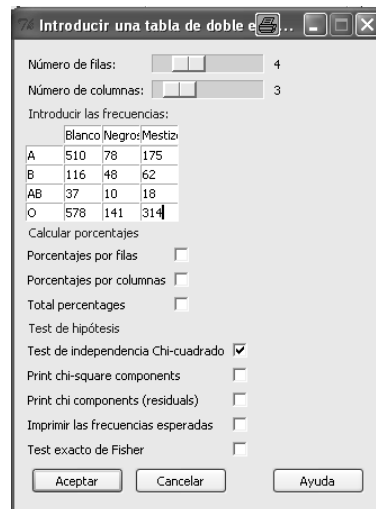


Figura 8.5: Ventana de entradas para tablas de contingencia

8.2. Contraste χ^2 de independencia

8.2.1. Mediante R Commander

Recordemos que este contraste sirve para valorar si existe relación significativa entre un par de variables cualitativas.

A modo de ejemplo, en la Revista Cubana de Hematología, Inmunología y Medicina Transfusional (1997, Vol. 13(2), pp. 122-31), el artículo *Frecuencia de los grupos sanguíneos A1, A2, Aint, Ael, B y O en donantes de sangre* se plantea si el grupo sanguíneo está o no relacionado con la raza. Para ello analizan una muestra de individuos donantes de sangre en relación a su grupo sanguíneo (clasificado en nuestro caso en A, B, AB y O) y su raza: descendientes de europoides (blancos), descendientes de africanoides (negros) y mestizos (mezcla entre ambos)

Para realizar el contraste con R Commander, elegimos la opción *Estadísticos* \rightarrow *Tablas de contingencia* \rightarrow *Introducir y analizar una tabla de doble entrada*. Aparece una ventana como la de la Figura 8.5. En ella tenemos que especificar lo siguiente:

1. El número de filas y de columnas, mediante los *scrolls* que aparecen en la parte superior y

que podemos mover en un sentido o en otro según queramos más o menos filas o columnas.

2. Los valores en sí de la tabla.
3. Si lo deseamos, podemos pedirle que saque en la ventana de resultados la tabla en porcentajes, calculados por filas, por columnas o sobre el total de la tabla.
4. El tipo de test que queremos. Podemos elegir entre tests χ^2 con y sin corrección de Yates y la prueba exacta de Fisher. En nuestro caso hemos elegido una prueba χ^2 sin corrección por continuidad.

El resultado es el siguiente:

Pearson's Chi-squared test

data: .Table

X-squared = 37.1616, df = 6, p-value = 1.638e-06

Vemos el valor del estadístico de contraste (37.1616), los grados de libertad (6) y el p-valor, que permite concluir que con esos datos podemos confirmar que existe relación altamente significativa entre el grupo sanguíneo y la raza.

8.2.2. Mediante código

De nuevo utilizamos la función *chisq.test*, proporcionando simplemente la matriz de frecuencias observadas:

```
Tabla<-matrix(c(510,78,175,116,48,62,37,10,18,578,141,314),4,3,byrow=TRUE)
chisq.test(Tabla,correct=FALSE)
```

En este caso, también se podría haber realizado una prueba exacta de Fisher mediante la función *fisher.test*.

Capítulo 9

Regresión lineal simple

Objetivos:

1. Contrastar si la relación entre pares de variables es estadísticamente significativa.
2. Ajustar la recta de regresión de una variable dependiente dada una variable independiente.
3. Realizar predicciones y estimaciones a partir de la recta de regresión mediante valores puntuales y por intervalos de confianza.
4. Realizar una diagnosis básica del modelo lineal.

9.1. Correlación

A lo largo de esta sección vamos a considerar datos que se refieren al porcentaje de suelo en los municipios andaluces según litología y según su erosión. Aparece también la extensión total y la pendiente media¹. Nos planteamos si existe alguna relación de tipo lineal entre la erosión, la pendiente media y el tipo de suelo según su litología. Se encuentran en el fichero *suelos.rda*.

Debemos comenzar recordando que la manera que tenemos de identificar el grado de relación lineal entre pares de variables es analizando el coeficiente de correlación lineal entre dichos pares de variables.

9.1.1. Mediante R Commander

En el menú de R Commander elegimos la opción *Estadísticos* → *Resúmenes* → *Matriz de correlaciones*. En la ventana emergente debemos seleccionar las variables para las cuales queremos obtener

¹Fuente. Instituto de Estadística y Cartografía.

	Sedimentarias	Metamórficas	Intrusivas	Pdte.media	Erosion.alta
Sedimentarias	1.00	-0.93	-0.47	-0.58	-0.06
Metamórficas	-0.93	1.00	0.11	0.65	0.13
Intrusivas	-0.47	0.11	1.00	0.03	-0.14
Pdte.media	-0.58	0.65	0.03	1.00	0.48
Erosion.alta	-0.06	0.13	-0.14	0.48	1.00

Cuadro 9.1: Coeficientes de correlación lineal entre el porcentaje de roca sedimentaria, metamórfica o intrusiva, la pendiente media y el porcentaje de terreno con erosión elevada o muy elevada.

los coeficientes. El resto de las opciones las dejamos en sus valores predeterminados. El resultado es la matriz que aparece en el Cuadro 9.1.

Vamos a comentar con detalle los resultados:

- En la diagonal, obviamente, aparecen unos, ya que el coeficiente de correlación lineal de una variable consigo misma es uno.
- La matriz es simétrica, ya que el coeficiente de correlación lineal también lo es, es decir, el coeficiente de la variable X con la variable Y es idéntico al de la variable Y con la variable X .
- El coeficiente entre la variable relativa al porcentaje de roca sedimentaria con la erosión es prácticamente cero, indicando que el grado de relación lineal es prácticamente nulo.
- El coeficiente entre la variable relativa al porcentaje de roca sedimentaria con la pendiente media es claramente negativo, lo que indica una tendencia a que municipios con mucha roca sedimentaria tengan poca pendiente.
- El coeficiente entre la pendiente y la erosión es claramente positivo, indicando una tendencia a que los municipios con más pendiente presenten también una alta erosión.

De todas formas, hasta ahora sólo hemos analizado estos coeficientes de una forma descriptiva, cuando lo más interesante es responder con claridad a la pregunta de si existe o no una relación **estadísticamente significativa** entre esos indicadores. Dicho de otra forma, refiriéndonos, por ejemplo, al coeficiente 0.13 (roca metamórfica con erosión), ¿es suficientemente grande como para que indique una relación estadísticamente significativa o su valor podría deberse al azar?

Lo que implícitamente estamos haciendo con estos comentarios es plantear la necesidad de contrastar si estos valores muestrales de los coeficientes son capaces o no de demostrar que los coeficientes de correlación poblacionales son significativamente distintos de cero.

Tenemos que hacerlo de dos en dos. Elegimos la opción *Estadísticos* → *Resúmenes* → *Test de correlación* y en la ventana emergente seleccionamos dos de las variables. Tenemos, además, la

opción de elegir si el test es bilateral o unilateral. En nuestro caso la hipótesis alternativa es que es distinto de cero, luego elegimos bilateral.

Para los datos relativos al porcentaje de roca sedimentaria con la erosión los resultados son los siguientes:

```
Pearson's product-moment correlation  
data: suelos$Erosion.elevada.muyelevada and suelos$Sedimentarias  
t = -1.7882, df = 765, p-value = 0.07415  
alternative hypothesis: true correlation is not equal to 0  
95 percent confidence interval:  
-0.134691484 0.006302951  
sample estimates:  
cor  
-0.06451624
```

Fijémonos que aparecen las variables que hemos elegido, el valor del estadístico t , los grados de libertad y, lo que es más importante, el p-valor. En este caso, es superior a 0.05, así que no podemos rechazar la hipótesis nula (el coeficiente es cero) en favor de la alternativa (el coeficiente es distinto de cero) o, dicho de otra forma, no hemos encontrado indicios de que exista una relación estadísticamente significativa entre el porcentaje de roca sedimentaria y la erosión.

Aparece además, un intervalo de confianza al 95 % para el coeficiente de correlación lineal (que contiene al cero, como cabía esperar) y el valor muestral del coeficiente.

Para finalizar, hemos cometido un pequeño desliz que aún estamos a tiempo de corregir, ya que directamente hemos analizado las posibles relaciones entre las variables como relaciones de tipo lineal. Es muy recomendable realizar un diagrama de dispersión o nube de puntos para detectar otro tipo de relaciones.

En R Commander esto podemos hacerlo mediante la opción *Gráficas* → *Matriz de diagramas de dispersión*. Este análisis es bastante rico, pero nosotros no hemos visto la mayoría de las opciones que permite, por lo que seleccionamos sólo el análisis más simple (ver Figura 9.1 arriba). El resultado aparece en la Figura 9.1. No aparecen patrones que hagan pensar en ningún otro tipo de relación, pero es evidente que deberíamos haber realizado un filtrado previo de datos para analizarlos más detalladamente.

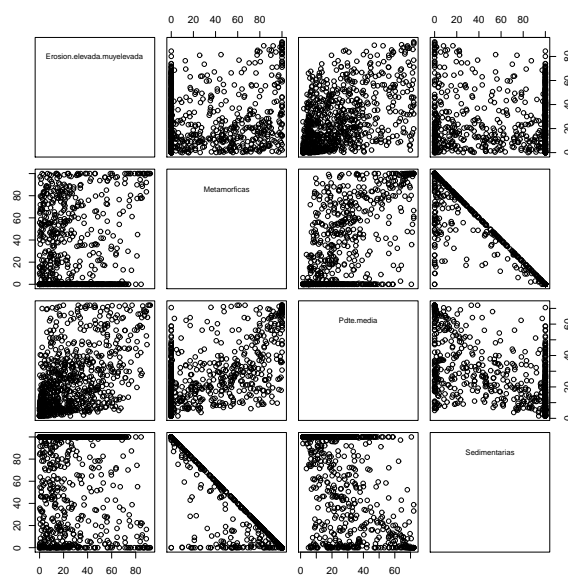
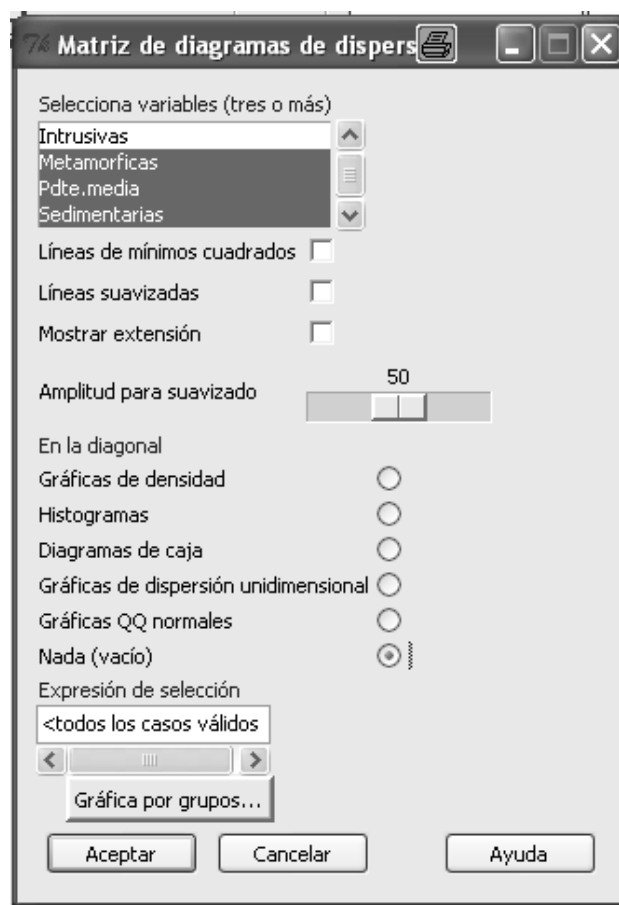


Figura 9.1: Ventana de entradas para la obtención de los diagramas de dispersión

9.1.2. Mediante código. Las funciones `pairs()`, `cors()` y `cor.test()`

La matriz con los diagramas de dispersión puede realizarse trivialmente mediante la función `pairs()` en cuya sintaxis básica tan sólo hay que facilitarle como entrada la matriz que constituyen las muestras de las variables cuyas relaciones estamos analizando:

```
pairs(suelos[,c(2,3,4,6,8)])
```

La matriz de coeficientes de correlación la facilita la función `cor()`, en cuya sintaxis básica de nuevo hay que facilitarle como entrada la matriz de datos y especificar, si es necesario, qué hacer con las observaciones faltantes. Por ejemplo, en el caso que nos ocupa,

```
cor(suelos[,c(2,3,4,6,8)])
```

En ocasiones puede haber datos faltantes. En ese caso, hay que añadir la opción `use="complete.obs"`: en otro caso, los coeficientes de correlación no serán calculados. Lo que especifica ese argumento es que utilice sólo los casos que tengan todos los valores de las variables disponibles.

Finalmente, el test de correlación se realiza mediante la función `cor.test()`. Su sintaxis básica es la siguiente:

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

Por ejemplo, para realizar algunos de los tests de nuestro ejemplo tendríamos

```
cor.test(suelos$Erosion.baja.moderada,suelos$Sedimentarias)  
cor.test(suelos$Pdte.media,suelos$Erosion.elevada.muyelevada)
```

9.2. Ajuste de la recta de regresión

Vamos a trabajar con unos datos que se refieren al contenido de oxígeno (en partes por mil) y a la dureza, medida en ksi, de 29 soldaduras de titanio, y que aparecen en el artículo "Advances in Oxygen Equivalence Equations for Predicting the Properties of Titanium Welds" (The Welding Journal, 2001:126-136). El objetivo es tratar de estimar la dureza de la soldadura, dado su contenido de oxígeno.

9.2.1. Mediante R Commander

Para obtener el ajuste de la recta de regresión entre las dos variables, elegimos la opción *Estadísticos* → *Ajuste de modelos* → *Regresión lineal*. La ventana emergente (Figura 9.2) nos pide que



Figura 9.2: Ventana emergente para el ajuste de la recta de regresión

especifiquemos la variable explicada (o variable dependiente) y la variable explicativa (o variable dependiente)². Adicionalmente, nos permite ponerle un nombre al modelo resultante.

Los resultados que aparecen en la ventana son los siguientes:

Call:

```
lm(formula = Fuerza.ksi ~ Oxig.partes.mil, data = Datos)
```

Residuals:

Min 1Q Median 3Q Max

-10.0185 -3.6639 -0.1139 4.4977 12.6515

Coefficients:

Estimate Std. Error t value Pr(>|t|)

(Intercept) 49.780 7.751 6.423 7e-07 ***

Oxig.partes.mil 16.923 5.050 3.351 0.00239 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 5.841 on 27 degrees of freedom

Multiple R-squared: 0.2937, Adjusted R-squared: 0.2676

F-statistic: 11.23 on 1 and 27 DF, p-value: 0.002391

Algunos de estos resultados requerirían conocimientos avanzados de regresión, pero busquemos los que nos interesan:

²En realidad, se pueden especificar varias variables explicativas, pero en ese caso ya no sería un modelo de regresión lineal simple, sino un modelo de regresión lineal múltiple, que excede los contenidos de este curso.

1. La estimación del valor del parámetro β_0 (*intercept*) es 49.780. Hipotéticamente, se interpretaría como la fuerza estimada de una soldadura o el promedio de fuerza de soldaduras con oxígeno 0.

A continuación lo que aparece es: el error estandar de esa estimación (7.751), el valor del estadístico de contraste (6.423) y el p-valor del contraste de $H_0 : \beta_0 = 0$ frente a $H_1 : \beta_0 \neq 0$ (7e-07).

2. La estimación de β_1 es 16.923, con un error estandar de la estimación de 5.050. El contraste de $H_0 : \beta_1 = 0$ frente a $H_1 : \beta_1 \neq 0$ arroja un valor del estadístico de 3.351 y un p-valor 0.00239.

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

$$\text{Fuerza.ajustada} = 49.780 + 16.923 \times \text{Origeno}.$$

3. El error estandar del ajuste tiene un valor de 5.841.
4. El coeficiente R^2 , aunque es obvio de calcular, aparece en la penúltima línea. Su valor, 0.2937, indica que el 29.37 % de toda la variabilidad que tiene el fenómeno relativo a la fuerza de la soldadura puede ser explicado por la cantidad de oxígeno con el que se realizó. Sin ser un porcentaje muy elevado, resulta interesante.

Aparte del propio ajuste de los parámetros β_0 y β_1 , podemos también obtener intervalos de confianza al nivel que deseemos de los mismos, sin más que clicar en *Modelos* \rightarrow *Intervalos de confianza*. En la ventana emergente sólo tenemos que elegir el nivel de confianza deseado. En nuestro caso, el resultado que aparece es el siguiente:

```
Estimate 2.5% 97.5%  
(Intercept) 49.77962 33.876505 65.68274  
Oxig.partes.mil 16.92287 6.560892 27.28485
```

9.2.2. Mediante código. La función `lm()`

La sintaxis básica de la función `lm()` es la siguiente:

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

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

Por ejemplo, en el caso que nos ocupa tendríamos

```
ajuste<-lm(Fuerza.ksi~Oxig.partes.mil, data = soldaduras)
summary(ajuste)
```

Eso devolvería los mismos resultados que R Commander.

9.3. Predicciones, estimaciones e intervalos de confianza para ellas

Recordemos que los valores que proporciona la recta de regresión para un valor dado de la variable independiente pueden interpretarse como predicciones del valor de la variable o como estimaciones de su promedio.

A modo de ejemplo, imaginemos que queremos estimar el valor de la fuerza de una soldadura realizada con un contenido de oxígeno de 1.5. En ese caso, podemos *predecirla* con la recta de regresión.

De igual forma, pensemos ahora en que queremos estimar la media de la fuerza que tendrán las soldaduras que se realicen con un contenido de oxígeno de 1.5. También podemos *estimarla* con la recta de regresión.

Finalmente, tanto para estas *predicciones* como para estas *estimaciones*, podemos proporcionar intervalos de confianza al nivel que se considere apropiado, normalmente al 95 %.

9.3.1. Mediante R Commander

En primer lugar, necesitamos cargar el plugin *RcmdrPlugin.HH*.

Para hacer todo esto con R Commander primero seleccionamos la opción del menú *Modelos* \rightarrow *Selecciona el modelo activo*. Eso cargará el modelo de la recta de regresión que hemos calculado antes y lo utilizará para todos los cálculos posteriores, hasta que construyamos otro modelo distinto.

A continuación, seleccionamos *Modelos* \rightarrow *Prediction intervals (HH)*. La ventana emergente (Figura 9.3) permite calcular el valor de la recta de regresión (*point estimate only*), el intervalo de confianza

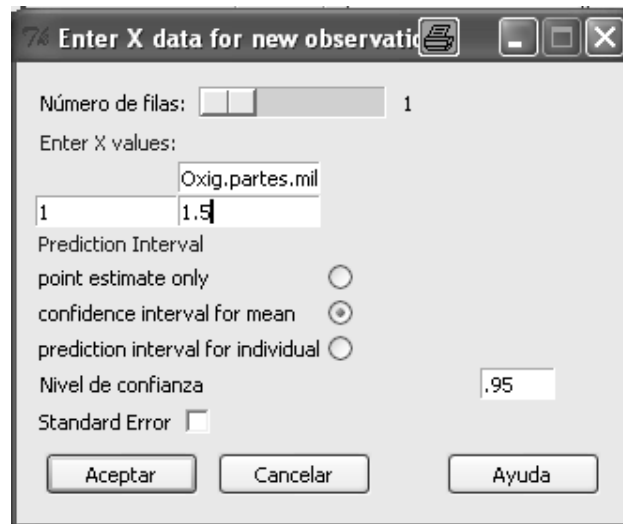


Figura 9.3: Obteniendo predicciones, estimaciones y sus intervalos de confianza

para el promedio dado un valor de la variable independiente (*confidence interval for mean*) o el intervalo de predicción para el valor dado de la variable independiente (*prediction interval for individual*). Estos análisis pueden realizarse para uno o varios valores, utilizando el *scroll* de la parte superior de la ventana.

En nuestro caso, para oxígeno 1.5, el valor predicho o el valor promedio de la fuerza de la soldadura es de 75.16393.

Si consideramos este valor como valor de predicción de la fuerza de una soldadura con ese oxígeno, un intervalo de predicción que con un 95 % contiene a la verdadera fuerza de la soldadura es (62.97281, 87.35505). ¡Es amplísimo! Proporciona poca información porque el objetivo de incluir al verdadero valor de la tasa con un 95 % de probabilidad es demasiado ambicioso, dadas las características de este ajuste. Necesitaríamos un R^2 superior para que el intervalo fuera más estrecho.

Finalmente, el intervalo de confianza al 95 % para el promedio de la fuerza de la soldadura con oxígeno 1.5 es (72.92916, 77.3987).

Hay una forma visual muy ilustrativa de observar las predicciones-estimaciones de la recta de regresión y los intervalos de predicción-confianza. Para verlo, elijamos la opción *Modelos* → *Confidence intervals plot*. En la ventana emergente debemos elegir de nuevo la variable dependiente y la variable independiente. El resultado es el que aparece en la Figura 9.4.

En el gráfico aparecen:

- Los valores observados en la muestra, en forma de un diagrama de dispersión.
- Los valores ajustados según el modelo, es decir, la recta de regresión.

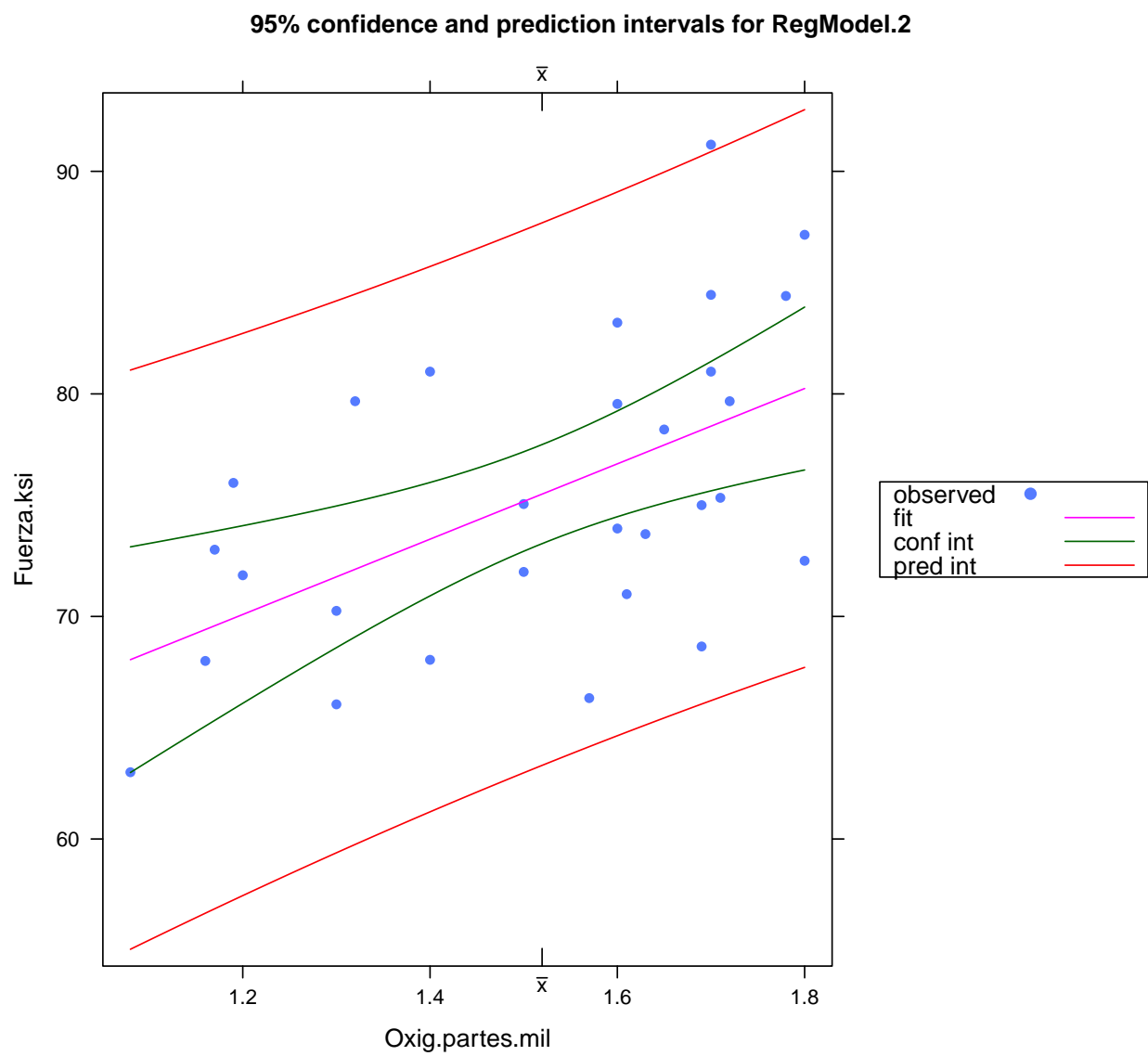


Figura 9.4: Gráfico de los intervalos de predicción y de confianza

- Los intervalos de confianza (al 95 %) para la fuerza media de la soldadura dado cada valor del oxígeno. En la leyenda se muestra que el color es verde, pero no se percibe en blanco y negro. Sin embargo, recordemos que estos intervalos son más estrechos que los de predicción, así que podemos deducir que son los más próximos a la recta de regresión.
- Los intervalos de predicción (al 95 %) para cada valor de la fuerza media. Los extremos de estos intervalos configuran las curvas más exteriores.

9.3.2. Mediante código. La función `predict()`

La sintaxis básica de esta función es la siguiente:

```
predict(modelo, newdata, interval, level = 0.95)
```

- *modelo* se refiere al resultado devuelto por la función *lm*.
- *newdata* debe ser una hoja de datos que especifique los valores de la variable dependiente para los que queremos las estimaciones o predicciones.
- *interval* especifica si queremos intervalos de confianza (para el promedio estimado) o de predicción (para el valor predicho), mediante las opciones "confidence" o "prediction".
- *level* es el nivel de significación de estos intervalos.

Por ejemplo, en nuestro caso sería

```
predict(ajuste, data.frame(Oxig.partes.mil=1.5),  
interval="prediction")
```

o

```
predict(ajuste, data.frame(Oxig.partes.mil=1.5),  
interval="confidence")
```

A través de las salidas de `predict` se puede construir mediante código una gráfica como la que facilita R Commander en la Figura 9.4. Lo dejamos como ejercicio al lector interesado.

9.4. Algo sobre diagnóstico del modelo

Para finalizar, resulta necesario concluir un capítulo dedicado a la regresión hablando, al menos de una forma introductoria, sobre la cuestión de cómo evaluar las hipótesis básicas que requiere el modelo para ser válido. No vamos a detenernos con detalle sobre ello, pero sí podemos comentar algo al respecto, al menos brevemente.

9.4.1. Mediante R Commander

Las opciones más básicas para ello están en *Modelos* → *Gráficas* → *Gráficas básicas de diagnóstico*. De las 4 gráficas que aparecen (ver Figura 9.5), sólo quiero comentar aquí la de la esquina superior izquierda, la **gráfica de residuos frente a valores ajustados**. Se trata de una representación donde el eje X incluye los valores ajustados de la recta para cada municipio y el eje Y los residuos de dichos ajustes. Por ejemplo, el 5º caso corresponde a una soldadura realizada con oxígeno a 1.8 y con fuerza 72.5. Por tanto, el valor ajustado de la recta para ella es $49.780 + 16.923 \times 1.8 = 80.2414$, por lo que el residuo es $72.5 - 80.2414 = -7.7414$; entonces, este punto aparecerá en las coordenadas $(80.2414, -7.7414)$.

En el gráfico además, aparecen los valores atípicos del modelo y una curva, consistente, básicamente, en las medias muestrales de los residuos para pequeños grupos de valores individuales de los datos. ¿Qué sabemos que debe ocurrir con el modelo de regresión lineal simple?

- Las medias de los residuos deben ser cero para cualquier valor de la variable independiente. En nuestro gráfico vemos que la curva de medias no coincide exactamente con el eje X, que es el valor cero de los residuos, lo que supone una pequeña violación de la hipótesis requerida, pero probablemente se debe a lo reducido del número de datos.
- Debe haber una tendencia lineal. En el diagrama de dispersión no se observa ningún patrón extraño no lineal en nuestro caso.
- La varianza de los residuos debía ser constante para todos los valores de la variable independiente. En el caso que analizamos no parece haber diferencias importantes en la varianza según los distintos valores ajustados (eje X).

Finalmente, la hipótesis de que los residuos sigan aproximadamente una distribución normal puede verificarse mediante un gráfico de normalidad de los residuos, que es el que aparece arriba a la derecha. Si fueran completamente normales, estarían todos los puntos sobre la línea recta. En la medida en que se alejen de ella, se alejan de la normalidad. En este caso no habría problema en aceptar la normalidad de los residuos.

9.4.2. Mediante código

La función *lm* que proporciona el ajuste permite obtener las representaciones ligadas a la diagnosis del modelo de forma trivial mediante la función *plot*. Por ejemplo, en nuestro caso sería de la siguiente manera:

lm(Fuerza.ksi ~ Oxig.partes.mil)

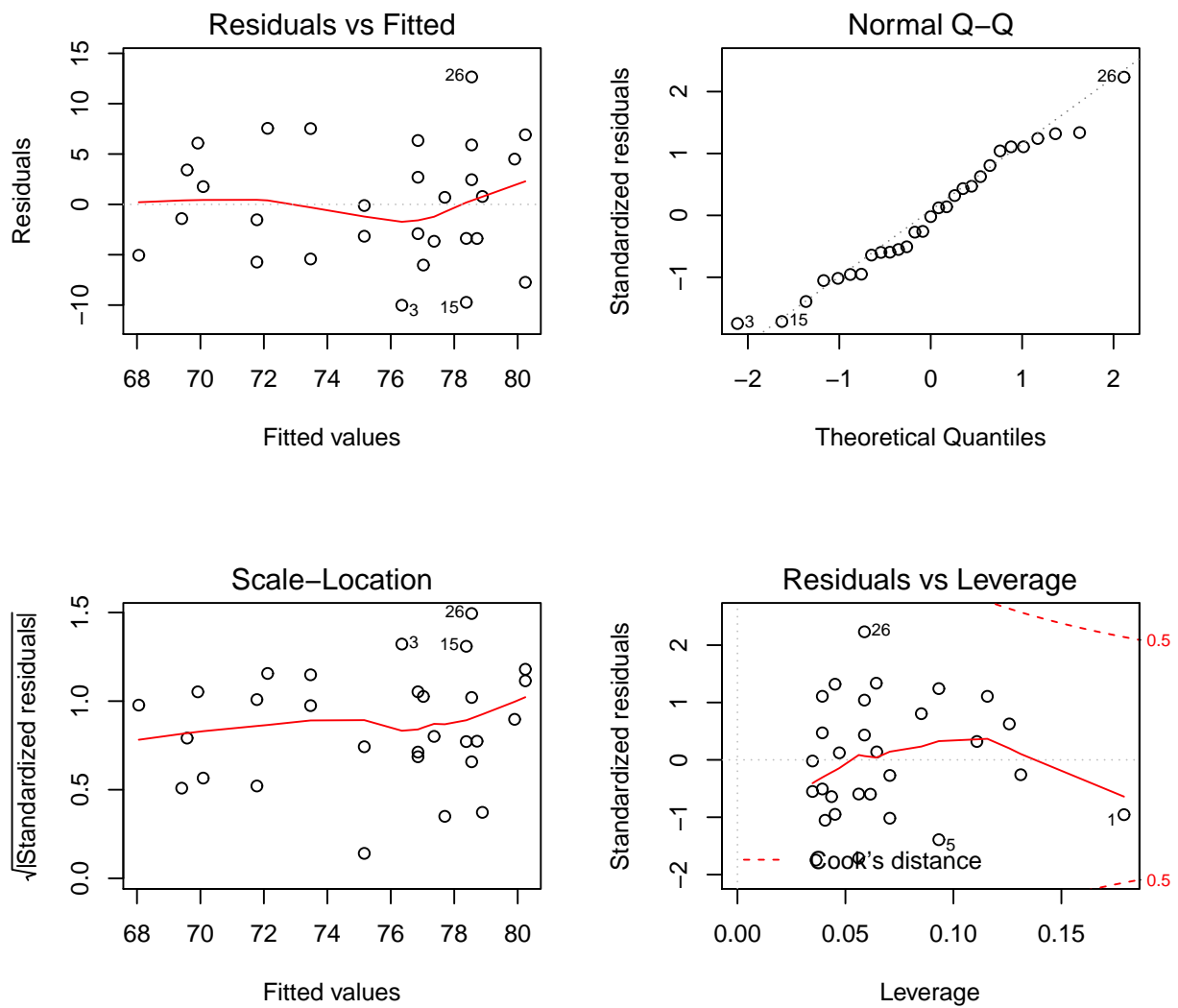


Figura 9.5: Diagn sis gr fico del modelo de regresi n lineal simple

```
ajuste<-lm(Fuerza.ksi~Oxig.partes.mil, data = soldaduras)
plot(ajuste)
```

Aparecerá una pantalla de gráficos. Clicando el ratón irán apareciendo los 4 gráficos.

Si lo que deseamos es que aparezcan los 4 gráficos juntos en una sola ventana, tendríamos que ejecutar las siguientes líneas:

```
ajuste<-lm(Fuerza.ksi~Oxig.partes.mil, data = soldaduras)
par(mfrow=c(2,2))
plot(ajuste)
```

La línea `par(mfrow=c(2,2))` determina como parámetros gráficos la generación de una matriz de gráficos definidos por filas de dos filas y dos columnas.

Bibliografía

- [1] Crawley, M. J. (2007). The R Book. Wiley.
- [2] Devore, J. L. (2005). Probabilidad y Estadística para Ingeniería y Ciencias (6^a edición). Thomson.
- [3] Everitt, B. S. & Hothorn, T. (2006). A Handbook of Statistical Analyses Using R. Chapman & Hall/CRC.
- [4] Maindonald, J. & Braun, J. (2007). Data Analysis and Graphics Using R. Cambridge University Press.
- [5] Mendenhal, W & Sincich, T. (1997). Probabilidad y Estadística para Ingeniería y Ciencias (4^a edición). Prentice Hall.
- [6] Navidi, W. (2006). Estadística para ingenieros y científicos. McGraw-Hill.