
Introducción al uso y programación del sistema estadístico R

Ramón Díaz-Uriarte

`rdiaz@cnio.es`

`http://bioinfo.cnio.es/~rdiaz`

Unidad de Bioinformática

Centro Nacional de Investigaciones Oncológicas (CNIO)

Copyright © 2003 Ramón Díaz-Uriarte

Handouts

- Tríptico R
- A quick introduction to ESS

Programa

- Introducción a R

Programa

- Introducción a R
- Uso de R con XEmacs + ESS

Programa

- Introducción a R
- Uso de R con XEmacs + ESS
- Objetos en R

Programa

- Introducción a R
- Uso de R con XEmacs + ESS
- Objetos en R
- Importación/Exportación de datos

Programa

- Introducción a R
- Uso de R con XEmacs + ESS
- Objetos en R
- Importación/Exportación de datos
- Gráficos en R

Programa

- Introducción a R
- Uso de R con XEmacs + ESS
- Objetos en R
- Importación/Exportación de datos
- Gráficos en R
- Programación en R

Programa

- Introducción a R
- Uso de R con XEmacs + ESS
- Objetos en R
- Importación/Exportación de datos
- Gráficos en R
- Programación en R
- Ejemplos prácticos

"Horario"

- 1^a mañana:
 - Introducción a R
 - Uso de R con XEmacs + ESS
 - Objetos en R

"Horario"

- 1^a mañana:
 - Introducción a R
 - Uso de R con XEmacs + ESS
 - Objetos en R
- 1^a tarde:
 - Objetos en R
 - Gráficos en R

"Horario"

- 1^a mañana:
 - Introducción a R
 - Uso de R con XEmacs + ESS
 - Objetos en R
- 1^a tarde:
 - Objetos en R
 - Gráficos en R
- 2^a mañana:
 - Programación en R
 - Ejemplos prácticos

"Horario"

- 1^a mañana:
 - Introducción a R
 - Uso de R con XEmacs + ESS
 - Objetos en R
- 1^a tarde:
 - Objetos en R
 - Gráficos en R
- 2^a mañana:
 - Programación en R
 - Ejemplos prácticos
- 2^a tarde:
 - Ejemplos prácticos

Introducción

- Qué son R y S

Introducción

- Qué son R y S
- Obtención e instalación de R

Introducción

- Qué son R y S
- Obtención e instalación de R
- Uso de R con ESS y XEmacs

Introducción

- Qué son R y S
- Obtención e instalación de R
- Uso de R con ESS y XEmacs
- Dos ejemplos con tests de la t y correlación

Qué son R y S

- "R, also known as “GNU S”, is a language and environment for statistical computing and graphics. R implements a dialect of the award-winning language S, developed at Bell Laboratories by John Chambers et al. For newcomers it provides easy access to a wide variety of statistical and graphical techniques. Advanced users are offered a full-featured programming language with which to add functionality by defining new functions." (Del folleto que teneis en las manos).

Qué son R y S

- "R, also known as “GNU S”, is a language and environment for statistical computing and graphics. R implements a dialect of the award-winning language S, developed at Bell Laboratories by John Chambers et al. For newcomers it provides easy access to a wide variety of statistical and graphical techniques. Advanced users are offered a full-featured programming language with which to add functionality by defining new functions." (Del folleto que teneis en las manos).
- "[S] has forever altered the way how people analyze, visualize and manipulate data" (Association of Computer Machinery Software System Award 1998 a John Chambers).

Qué son R y S

- " R, also known as “GNU S”, is a language and environment for statistical computing and graphics. R implements a dialect of the award-winning language S, developed at Bell Laboratories by John Chambers et al. For newcomers it provides easy access to a wide variety of statistical and graphical techniques. Advanced users are offered a full-featured programming language with which to add functionality by defining new functions." (Del folleto que teneis en las manos).
- " [S] has forever altered the way how people analyze, visualize and manipulate data" (Association of Computer Machinery Software System Award 1998 a John Chambers).
- Probablemente, S y R son los dos lenguajes más usados en investigación en estadística. Otras virtudes en el folletillo.

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:
 - La capacidad de combinar, sin fisuras, análisis "preempaquetados" (ej., una regresión logística) con análisis ad-hoc, específicos para una situación: capacidad de manipular y modificar datos y funciones.

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:
 - La capacidad de combinar, sin fisuras, análisis "preempaquetados" (ej., una regresión logística) con análisis ad-hoc, específicos para una situación: capacidad de manipular y modificar datos y funciones.
 - Los gráficos de alta calidad (revelaciones de la visualización de datos y producción de gráficas para papers).

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:
 - La capacidad de combinar, sin fisuras, análisis "preempaquetados" (ej., una regresión logística) con análisis ad-hoc, específicos para una situación: capacidad de manipular y modificar datos y funciones.
 - Los gráficos de alta calidad (revelaciones de la visualización de datos y producción de gráficas para papers).
- La comunidad de R es muy dinámica (ej., crecimiento en número de paquetes), integrada por estadísticos de gran renombre (ej., J. Chambers, L. Terney, B. Ripley, D. Bates, etc).

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:
 - La capacidad de combinar, sin fisuras, análisis "preempaquetados" (ej., una regresión logística) con análisis ad-hoc, específicos para una situación: capacidad de manipular y modificar datos y funciones.
 - Los gráficos de alta calidad (revelaciones de la visualización de datos y producción de gráficas para papers).
- La comunidad de R es muy dinámica (ej., crecimiento en número de paquetes), integrada por estadísticos de gran renombre (ej., J. Chambers, L. Terney, B. Ripley, D. Bates, etc.).
- Extensiones específicas a áreas nuevas (bioinformática, geoestadística, modelos gráficos).

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:
 - La capacidad de combinar, sin fisuras, análisis "preempaquetados" (ej., una regresión logística) con análisis ad-hoc, específicos para una situación: capacidad de manipular y modificar datos y funciones.
 - Los gráficos de alta calidad (revelaciones de la visualización de datos y producción de gráficas para papers).
- La comunidad de R es muy dinámica (ej., crecimiento en número de paquetes), integrada por estadísticos de gran renombre (ej., J. Chambers, L. Terney, B. Ripley, D. Bates, etc.).
- Extensiones específicas a áreas nuevas (bioinformática, geoestadística, modelos gráficos).
- Un lenguaje orientado a objetos.

Qué son R y S (II)

- En pocas palabras, los grandes atractivos de R/S son:
 - La capacidad de combinar, sin fisuras, análisis "preempaquetados" (ej., una regresión logística) con análisis ad-hoc, específicos para una situación: capacidad de manipular y modificar datos y funciones.
 - Los gráficos de alta calidad (revelaciones de la visualización de datos y producción de gráficas para papers).
- La comunidad de R es muy dinámica (ej., crecimiento en número de paquetes), integrada por estadísticos de gran renombre (ej., J. Chambers, L. Terney, B. Ripley, D. Bates, etc.).
- Extensiones específicas a áreas nuevas (bioinformática, geoestadística, modelos gráficos).
- Un lenguaje orientado a objetos.
- Muy parecido a Matlab y Octave, y con sintaxis que recuerda a C/C++.

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.
- Filosofía y objetivos del proyecto GNU: www.gnu.org.

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.
- Filosofía y objetivos del proyecto GNU: www.gnu.org.
- Desarrollar un sistema completo y "libre" (donde "free is free as in freedom, not free as in beer").

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.
- Filosofía y objetivos del proyecto GNU: www.gnu.org.
- Desarrollar un sistema completo y "libre" (donde "free is free as in freedom, not free as in beer").
- Algunos "GNUs famosos": Emacs, gcc, GNU/Linux, etc.

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.
- Filosofía y objetivos del proyecto GNU: www.gnu.org.
- Desarrollar un sistema completo y "libre" (donde "free is free as in freedom, not free as in beer").
- Algunos "GNUs famosos": Emacs, gcc, GNU/Linux, etc.
- R se distribuye con licencia GNU GPL o General Public License (ver <http://www.gnu.org/licenses/gpl.html>.)

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.
- Filosofía y objetivos del proyecto GNU: www.gnu.org.
- Desarrollar un sistema completo y "libre" (donde "free is free as in freedom, not free as in beer").
- Algunos "GNUs famosos": Emacs, gcc, GNU/Linux, etc.
- R se distribuye con licencia GNU GPL o General Public License (ver <http://www.gnu.org/licenses/gpl.html>.)
- La GPL no pone ninguna restricción al uso de R. Restringe su distribución (ha de ser GPL).

¿Cuánto cuesta R? R es "GNU S"

- R es la implementación GNU de S.
- Filosofía y objetivos del proyecto GNU: www.gnu.org.
- Desarrollar un sistema completo y "libre" (donde "free is free as in freedom, not free as in beer").
- Algunos "GNUs famosos": Emacs, gcc, GNU/Linux, etc.
- R se distribuye con licencia GNU GPL o General Public License (ver <http://www.gnu.org/licenses/gpl.html>.)
- La GPL no pone ninguna restricción al uso de R. Restringe su distribución (ha de ser GPL).
- R se obtiene por 0 euros en <http://cran.r-project.org>

Obtención e instalación de R

Depende del sistema operativo, pero todo se puede encontrar en
<http://cran.r-project.org/bin>.

- Windows: bajar ("download") el ejecutable desde
<http://cran.r-project.org/bin/windows/base>. (por ejemplo,
<http://cran.r-project.org/bin/windows/base/rw1070.exe>). Ejecutar el fichero. Instalará el sistema base y los paquetes recomendados.

Obtención e instalación de R

Depende del sistema operativo, pero todo se puede encontrar en
<http://cran.r-project.org/bin>.

- Windows: bajar ("download") el ejecutable desde
<http://cran.r-project.org/bin/windows/base>. (por ejemplo,
<http://cran.r-project.org/bin/windows/base/rw1070.exe>). Ejecutar el fichero. Instalará el sistema base y los paquetes recomendados.
- GNU/Linux: dos opciones:

Obtención e instalación de R

Depende del sistema operativo, pero todo se puede encontrar en <http://cran.r-project.org/bin>.

- Windows: bajar ("download") el ejecutable desde <http://cran.r-project.org/bin/windows/base>. (por ejemplo,
<http://cran.r-project.org/bin/windows/base/rw1070.exe>). Ejecutar el fichero. Instalará el sistema base y los paquetes recomendados.
- GNU/Linux: dos opciones:
 - Obtener el R-x.y.z.tar.gz y compilar desde las fuentes, y también bajar los paquetes adicionales e instalar. (Buena forma de comprobar que el sistema tiene development tools).

Obtención e instalación de R

Depende del sistema operativo, pero todo se puede encontrar en
<http://cran.r-project.org/bin>.

- Windows: bajar ("download") el ejecutable desde
<http://cran.r-project.org/bin/windows/base>. (por ejemplo,
<http://cran.r-project.org/bin/windows/base/rw1070.exe>). Ejecutar el fichero. Instalará el sistema base y los paquetes recomendados.
- GNU/Linux: dos opciones:
 - Obtener el R-x.y.z.tar.gz y compilar desde las fuentes, y también bajar los paquetes adicionales e instalar. (Buena forma de comprobar que el sistema tiene development tools).
 - Obtener binarios (ej., *.deb para Debian, *.rpm para RedHat, SuSE, Mandrake).

Paquetes adicionales

R consta de un "sistema base" y de paquetes adicionales que extienden la funcionalidad. Distintos "tipos" de paquetes:

- Los que forman parte del sistema base (ej. `ctest`).

Paquetes adicionales

R consta de un "sistema base" y de paquetes adicionales que extienden la funcionalidad. Distintos "tipos" de paquetes:

- Los que forman parte del sistema base (ej. ctest).
- Los que no son parte del sistema base, pero son "**recommended**" (ej., survival, nlme). En GNU/Linux y Windows ya (desde 1.6.0?) forman parte de la distribución standard.

Paquetes adicionales

R consta de un "sistema base" y de paquetes adicionales que extienden la funcionalidad. Distintos "tipos" de paquetes:

- Los que forman parte del sistema base (ej. ctest).
- Los que no son parte del sistema base, pero son "**recommended**" (ej., survival, nlme). En GNU/Linux y Windows ya (desde 1.6.0?) forman parte de la distribución standard.
- Otros paquetes; ej., car, gregmisc, los paquetes de Bioconductor (como multtest, etc). Estos necesitamos seleccionarlos e instalarlos individualmente. Más adelante veremos como.

Documentación sobre R (I)

Los "manuales" de R, incluidos en todas las instalaciones. Son:

- *An introduction to R.* De lectura requerida.
- *Writing R extensions.*
- *R data import/export.*
- *The R language definition.*
- *R installation and administration.*

Documentación sobre R (II)

Documentación general:

- *A guide for the unwilling S user*, de P. Burns. En http://cran.r-project.org/doc/contrib/Burns-unwilling_S.pdf o <http://www.burns-stat.com/pages/tutorials.html>. ¡Sólo 8 páginas!
- *R para principiantes*, de E. Paradis. En <http://cran.r-project.org/other-docs.html> o http://cran.r-project.org/doc/contrib/rdebut_es.pdf.
- FAQ.
- *S Programming*, de W. Venables y B. Ripley. (Ver también <http://www.stats.ox.ac.uk/pub/MASS3/Sprog>.)

Documentación general:

- *S poetry* de P. Burns. En
<http://www.burns-stat.com/pages/spoetry.html>.
- Otros documentos en la página de J. Fox
(<http://cran.r-project.org/doc/contrib/Fox-Companion/appendix.html>),
ej. sobre Frames, etc).
- El site de Paul Johnson
(<http://lark.cc.ukans.edu/~pauljohn/R/statsRus.html>).
- Los libros azul , marrón y verde de Chambers et al.

Documentación sobre R (III)

Estadística:

- *Introductory statistics with R* de P. Dalgaard.
- *An R and S-PLUS companion to applied regression*, de J. Fox.
- *Modern applied statistics with S, 4th ed.* de W. Venables y B. Ripley. (Ver también
<http://www.stats.ox.ac.uk/pub/MASS4.>)
- *Practical regression and ANOVA using R* de J. Faraway , en
<http://cran.r-project.org/other-docs.html> o
<http://www.stat.lsa.umich.edu/~faraway/book/>.
- Otros documentos en
<http://cran.r-project.org/other-docs.html>.
- *S-PLUS 6.0 for Unix. Guide to statistics.* Vol. I & II. En
<http://www.insightful.com/support/documentation.asp?DID=3>.

Documentación sobre R (IV)

- *Mixed-effects models in S and S-PLUS*, de J. Pinheiro y D. Bates.
- *Regression modelling strategies*, de F. Harrell.
- Site con documentación sobre análisis para datos categóricos (site para libro de A. Agresti *Categorical data analysis*).
<http://www.stat.ufl.edu/~aa/cda/cda.html>.
- *Modeling survival data: extending the Cox model*, de T. M. Therneau y P. M. Grambsch.
- Documentos misceláneos en página de J. Fox.
(<http://cran.r-project.org/doc/contrib/Fox-Companion/appendix.html>.)

Obteniendo ayuda sobre R

- Ayuda incluida con el programa (veremos más adelante).

Obteniendo ayuda sobre R

- Ayuda incluida con el programa (veremos más adelante).
- FAQ.

Obteniendo ayuda sobre R

- Ayuda incluida con el programa (veremos más adelante).
- FAQ.
- Site de Paul Johnson

<http://lark.cc.ukans.edu/~pauljohn/R/statsRus.html>.

Obteniendo ayuda sobre R

- Ayuda incluida con el programa (veremos más adelante).
- FAQ.
- Site de Paul Johnson
<http://lark.cc.ukans.edu/~pauljohn/R/statsRus.html>.
- R-help. Peeero:

Obteniendo ayuda sobre R

- Ayuda incluida con el programa (veremos más adelante).
- FAQ.
- Site de Paul Johnson
<http://lark.cc.ukans.edu/~pauljohn/R/statsRus.html>.
- R-help. Peeero:
- Las email lists son "searchable". Ver
<http://cran.r-project.org/search.html>; y
<http://finzi.psych.upenn.edu/search.html> permite hacer las búsquedas no sólo sobre las listas de email sino también sobre la documentación (incluyendo paquetes).

-
- Antes de hacer preguntas comprobar si ya han sido contestadas.

-
- Antes de hacer preguntas comprobar si ya han sido contestadas.
 - Las listas de ayuda son voluntarias: nadie puede exigir soluciones. (¿Cuánto habeis pagado por R?)

-
- Antes de hacer preguntas comprobar si ya han sido contestadas.
 - Las listas de ayuda son voluntarias: nadie puede exigir soluciones. (¿Cuánto habeis pagado por R?)
 - Pausarse antes de gritar "bug": véase FAQ, sección 9. Pero si hay un bug, por favor reportese.

Instalación de paquetes adicionales

Depende del sistema operativo

- Windows:

Instalación de paquetes adicionales

Depende del sistema operativo

- Windows:
 - Desde la "GUI" o desde la interfaz de XEmacs.

Instalación de paquetes adicionales

Depende del sistema operativo

- Windows:
 - Desde la "GUI" o desde la interfaz de XEmacs.
 - Desde R, con "install.packages()", como en GNU/Linux (ver siguiente).

Instalación de paquetes adicionales

Depende del sistema operativo

- Windows:
 - Desde la "GUI" o desde la interfaz de XEmacs.
 - Desde R, con "install.packages()", como en GNU/Linux (ver siguiente).
- GNU/Linux:

Instalación de paquetes adicionales

Depende del sistema operativo

- Windows:
 - Desde la "GUI" o desde la interfaz de XEmacs.
 - Desde R, con "install.packages()", como en GNU/Linux (ver siguiente).
- GNU/Linux:
 - "R CMD INSTALL paquete-x.y.z.tar.gz". Permite instalar aunque uno no sea root (especificando el directorio).

Instalación de paquetes adicionales

Depende del sistema operativo

- Windows:
 - Desde la "GUI" o desde la interfaz de XEmacs.
 - Desde R, con "install.packages()", como en GNU/Linux (ver siguiente).
- GNU/Linux:
 - "R CMD INSTALL paquete-x.y.z.tar.gz". Permite instalar aunque uno no sea root (especificando el directorio).
 - Más cómodo, desde R, "install.packages()", "update.packages()", etc. También permiten instalar no siendo root (especificar lib.loc).

Inicio de una sesión de R

- GNU/Linux:

Inicio de una sesión de R

- GNU/Linux:
 - Teclear "R" en una shell.

Inicio de una sesión de R

- GNU/Linux:
 - Teclear "R" en una shell.
 - Iniciar R desde (X)Emacs (M-X R).

Inicio de una sesión de R

- GNU/Linux:
 - Teclear "R" en una shell.
 - Iniciar R desde (X)Emacs (M-X R).
- Windows:

Inicio de una sesión de R

- GNU/Linux:
 - Teclear "R" en una shell.
 - Iniciar R desde (X)Emacs (M-X R).
- Windows:
 - Hacer click dos veces en el icono. Se abrirá "Rgui".

Inicio de una sesión de R

- GNU/Linux:
 - Teclear "R" en una shell.
 - Iniciar R desde (X)Emacs (M-X R).
- Windows:
 - Hacer click dos veces en el icono. Se abrirá "Rgui".
 - Desde una "ventana del sistema" ejecutar "Rterm"; parecido a "R" en Unix o Linux.

Inicio de una sesión de R

- GNU/Linux:
 - Teclear "R" en una shell.
 - Iniciar R desde (X)Emacs (M-X R).
- Windows:
 - Hacer click dos veces en el icono. Se abrirá "Rgui".
 - Desde una "ventana del sistema" ejecutar "Rterm"; parecido a "R" en Unix o Linux.
 - Iniciar R desde de XEmacs.

Inicio de una sesión de R

- GNU/Linux:
 - Teclear "R" en una shell.
 - Iniciar R desde (X)Emacs (M-X R).
- Windows:
 - Hacer click dos veces en el icono. Se abrirá "Rgui".
 - Desde una "ventana del sistema" ejecutar "Rterm"; parecido a "R" en Unix o Linux.
 - Iniciar R desde de XEmacs.
- Se puede "customizar" como se inicia una sesión de R (ej., que paquetes se cargan, mensajes, etc). Ver sección 10.8 en *An introduction to R*.

Una primera sesión

```
> rnorm(5) # 5 numeros aleatorios de una distribucion normal (mean= 0, sd = 1)
> ## Hemos dicho que "#" indica el comienzo de un comentario?
>
> ## Los números se producen, y se muestran (print).
>
> x <- rnorm(5) # asignamos esos números a un objeto (un vector) llamado x.
> summary(x) ## mostrar un "summary" de x (un summary "inteligente").
>
> ## o también:
> w <- summary(x)
> w
> print(w) # teclear w y print(y) producen el mismo resultado.
>
> ## summary(x) TAMBIEN es un objeto. (virtually) "everything is an object".
```

Ayuda incluida con el programa



?rnorm

Ayuda incluida con el programa

- `?rnorm`
- `help.start()`

Ayuda incluida con el programa

- `?rnorm`
- `help.start()`
- `?help.search`

Ayuda incluida con el programa

- `?rnorm`
- `help.start()`
- `?help.search`
- `help.search("normal")`

Ayuda incluida con el programa

- `?rnorm`
- `help.start()`
- `?help.search`
- `help.search("normal")`
- `?apropos`

Ayuda incluida con el programa

- `?rnorm`
- `help.start()`
- `?help.search`
- `help.search("normal")`
- `?apropos`
- `apropos("normal")`

Ayuda incluida con el programa

- `?rnorm`
- `help.start()`
- `?help.search`
- `help.search("normal")`
- `?apropos`
- `apropos("normal")`
- `?demo`

Ayuda incluida con el programa

- `?rnorm`
- `help.start()`
- `?help.search`
- `help.search("normal")`
- `?apropos`
- `apropos("normal")`
- `?demo`
- `demo(graphics); demo(persp); demo(lm.glm)`

Usando R con (X)Emacs

¿Por qué usar R con XEmacs?

- Uso de scripts, mantenimiento de código ordenado y comentado. "Buena práctica estadística". (ver también `loadhistory`, `savehistory`).

Usando R con (X)Emacs

¿Por qué usar R con XEmacs?

- Uso de scripts, mantenimiento de código ordenado y comentado. "Buena práctica estadística". (ver también `loadhistory`, `savehistory`).
- Colorea sintaxis, completa paréntesis, etc.

Usando R con (X)Emacs

¿Por qué usar R con XEmacs?

- Uso de scripts, mantenimiento de código ordenado y comentado. "Buena práctica estadística". (ver también `loadhistory`, `savehistory`).
- Colorea sintaxis, completa paréntesis, etc.
- Una interfaz común para R en distintos sistemas operativos.

Usando R con (X)Emacs

¿Por qué usar R con XEmacs?

- Uso de scripts, mantenimiento de código ordenado y comentado. "Buena práctica estadística". (ver también `loadhistory`, `savehistory`).
- Colorea sintaxis, completa paréntesis, etc.
- Una interfaz común para R en distintos sistemas operativos.
- Una interfaz común para otros paquetes estadísticos (ej., SAS, XLispStat, Arc, etc) y numéricos (ej., Octave).

Usando R con (X)Emacs

¿Por qué usar R con XEmacs?

- Uso de scripts, mantenimiento de código ordenado y comentado. "Buena práctica estadística". (ver también `loadhistory`, `savehistory`).
- Colorea sintaxis, completa paréntesis, etc.
- Una interfaz común para R en distintos sistemas operativos.
- Una interfaz común para otros paquetes estadísticos (ej., SAS, XLispStat, Arc, etc) y numéricos (ej., Octave).
- Pero aunque (X)Emacs es MUCHO más que un editor...
 - (X)Emacs con ESS no es "familiar" para los usuarios de Windows (pero no tanto con las modificaciones de J. Fox).
 - Problemas no resueltos en interacción R, XEmacs, Windows.

- Alternativa principal en Windows es WinEdt
(<http://www.winedt.com> y
<http://cran.r-project.org/contrib/extrawinedt>).
Pero:

- Alternativa principal en Windows es WinEdt
(<http://www.winedt.com> y
<http://cran.r-project.org/contrib/extrawinedt>).
Pero:
 - WinEdt no es GPL ni gratuito (es shareware). Problemas de licencia (por ej., para nosotros aquí).

- Alternativa principal en Windows es WinEdt
(<http://www.winedt.com> y
<http://cran.r-project.org/contrib/extrawinedt>).
Pero:
 - WinEdt no es GPL ni gratuito (es shareware). Problemas de licencia (por ej., para nosotros aquí).
 - WinEdt no está disponible para otros sistemas operativos.

- Alternativa principal en Windows es WinEdt
(<http://www.winedt.com> y
<http://cran.r-project.org/contrib/extrawinedt>). Pero:
 - WinEdt no es GPL ni gratuito (es shareware). Problemas de licencia (por ej., para nosotros aquí).
 - WinEdt no está disponible para otros sistemas operativos.
- En GNU/Linux, hay coloreado de sintaxis (y completado de paréntesis?) para Vim y Nedit, pero nada similar a la interacción con un buffer con R.

R, ESS + XEmacs, (a la J. Fox)

- Toda la información está en
<http://www.socsci.mcmaster.ca/jfox/Books/Companion/ESS/>.
- Las últimas páginas de *An Introduction to ESS + XEmacs for Windows Users of R* tiene una **clarísima y detallada** información de como instalar y configurar ESS y XEmacs con las modificaciones de J. Fox. En resumen:
- Descargar XEmacs para Windows (la versión binaria para Windows, no para Cygwin) de
<http://www.xemacs.org/Download/win32>. Si es posible, instalar todos los paquetes seleccionados por defecto (si no, ver documento J. Fox "An introduction to ESS (...)" para paquetes requeridos.)
- Instalar R (si no está ya instalado).

- Añadir c:/Archivos de Programa/R/rwxxxx/bin al search path the windows (o modificar init.el para que XEmacs sepa donde encontrar rterm.exe).
- Descargar el fichero "fox-ess-config.zip" de <http://www.socsci.mcmaster.ca/jfox/Books/Companion/ESS/>. Descomprimir.
- Crear un directorio .xemacs en el "home directory".
- Copiar "init.el" a "./.xemacs".

Modificaciones (*XEmacs + ESS*) (I)

- Muy conveniente mantener cada proyecto separado.
Separamos código, datos, etc. Para eso:

Modificaciones (*XEmacs + ESS*) (I)

- Muy conveniente mantener cada proyecto separado.
Separamos código, datos, etc. Para eso:
 - Abrir en el Explorador de Windows el/los directorios donde queremos trabajar.

Modificaciones (XEmacs + ESS) (I)

- Muy conveniente mantener cada proyecto separado.
Separamos código, datos, etc. Para eso:
 - Abrir en el Explorador de Windows el/los directorios donde queremos trabajar.
 - Right-click en el icono de XEmacs, y arrastrar hasta el directorio.

Modificaciones (XEmacs + ESS) (I)

- Muy conveniente mantener cada proyecto separado.
Separamos código, datos, etc. Para eso:
 - Abrir en el Explorador de Windows el/los directorios donde queremos trabajar.
 - Right-click en el icono de XEmacs, y arrastrar hasta el directorio.
 - Seleccionar "crear iconos de acceso directo aquí".

Modificaciones (XEmacs + ESS) (I)

- Muy conveniente mantener cada proyecto separado.
Separamos código, datos, etc. Para eso:
 - Abrir en el Explorador de Windows el/los directorios donde queremos trabajar.
 - Right-click en el icono de XEmacs, y arrastrar hasta el directorio.
 - Seleccionar "crear iconos de acceso directo aquí".
- El icono de cada directorio inicia R en ese directorio.

Modificaciones (XEmacs + ESS) (I)

- Muy conveniente mantener cada proyecto separado.
Separamos código, datos, etc. Para eso:
 - Abrir en el Explorador de Windows el/los directorios donde queremos trabajar.
 - Right-click en el icono de XEmacs, y arrastrar hasta el directorio.
 - Seleccionar "crear iconos de acceso directo aquí".
- El icono de cada directorio inicia R en ese directorio.
- (Si nos olvidamos, siempre podemos cambiar directorios con "getwd").

Modificaciones (XEmacs + ESS) (I)

- Muy conveniente mantener cada proyecto separado.
Separamos código, datos, etc. Para eso:
 - Abrir en el Explorador de Windows el/los directorios donde queremos trabajar.
 - Right-click en el icono de XEmacs, y arrastrar hasta el directorio.
 - Seleccionar "crear iconos de acceso directo aquí".
- El icono de cada directorio inicia R en ese directorio.
- (Si nos olvidamos, siempre podemos cambiar directorios con "getwd").
- También podemos editar `init.el` y donde pone (`setq ess-ask-for-ess-directory nil`) poner (`setq ess-ask-for-ess-directory t`).

Modificaciones (XEmacs + ESS) (II)

- Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).

Modificaciones (XEmacs + ESS) (II)

- Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).
- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.

Modificaciones (XEmacs + ESS) (II)

- Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).
- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.

Modificaciones (XEmacs + ESS) (II)

- Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).
- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.
- Para lo segundo: ya instalado por defecto en vuestras máquinas.

Modificaciones (XEmacs + ESS) (II)

- Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).
- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.
- Para lo segundo: ya instalado por defecto en vuestras máquinas.
- Mis modificaciones están en mi página web (<http://bioinfo.cnio.es/~rdiaz/#cursosR.>).

Modificaciones (XEmacs + ESS) (III)

- Podríamos querer usar XEmacs y ESS de la forma "tradicional", con el estilo genuino de XEmacs.

Modificaciones (XEmacs + ESS) (III)

- Podríamos querer usar XEmacs y ESS de la forma "tradicional", con el estilo genuino de XEmacs.
- Toda la funcionalidad de XEmacs (que se ve disminuida un poco por los keybindings tipo Windows al no permitir "C-x" como inicio de secuencia de teclas).

Modificaciones (XEmacs + ESS) (III)

- Podríamos querer usar XEmacs y ESS de la forma "tradicional", con el estilo genuino de XEmacs.
- Toda la funcionalidad de XEmacs (que se ve disminuida un poco por los keybindings tipo Windows al no permitir "C-x" como inicio de secuencia de teclas).
- Funciona si nos movemos a otro sistema operativo/otra máquina.

-
- Algunas limitaciones usando "modo windows":

- Algunas limitaciones usando "modo windows":
 - No podemos usar shortcuts para evaluar línea, región o buffer (C-c C-j, C-c C-r, C-c C-b).

- Algunas limitaciones usando "modo windows":
 - No podemos usar shortcuts para evaluar línea, región o buffer (`C-c C-j`, `C-c C-r`, `C-c C-b`).
 - No podemos usar shortcuts para abrir ficheros (`C-x C-f`), para guardarlos (`C-x C-s`), para dividir la pantalla en 2 (`C-x 2`, `C-x 3`), para seleccionar el buffer que estamos editando (`C-x b`), o para cambiar de ventana (`C-x o`).

- Algunas limitaciones usando "modo windows":
 - No podemos usar shortcuts para evaluar línea, región o buffer (C-c C-j, C-c C-r, C-c C-b).
 - No podemos usar shortcuts para abrir ficheros (C-x C-f), para guardarlos (C-x C-s), para dividir la pantalla en 2 (C-x 2, C-x 3), para seleccionar el buffer que estamos editando (C-x b), o para cambiar de ventana (C-x o).
 - Limita un uso completo y sofisticado, tanto de ESS como de (X)Emacs.

- Algunas limitaciones usando "modo windows":
 - No podemos usar shortcuts para evaluar línea, región o buffer (`C-c C-j`, `C-c C-r`, `C-c C-b`).
 - No podemos usar shortcuts para abrir ficheros (`C-x C-f`), para guardarlos (`C-x C-s`), para dividir la pantalla en 2 (`C-x 2`, `C-x 3`), para seleccionar el buffer que estamos editando (`C-x b`), o para cambiar de ventana (`C-x o`).
 - Limita un uso completo y sofisticado, tanto de ESS como de (X)Emacs.
 - ESS es un sistema grande y complejo con muchas opciones. No se aprende en 2 días. (¡(X)Emacs mucho menos!).

Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).

- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.

Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).

- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.

Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).

- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.
- Para lo segundo: ya instalado por defecto en vuestras máquinas.

Si preferimos los keybindings de XEmacs: cambiar la linea "(defconst pc-behaviour-level 2)" a "(defconst pc-behaviour-level 0)" (Y algún otro cambio menor).

- Para que indique en que fila y columna nos encontramos usar el "custom.el" provisto.
- Para lo primero, copiar ramon.init.el o fox.init.el como init.el.
- Para lo segundo: ya instalado por defecto en vuestras máquinas.
- Mis modificaciones están en mi página web (<http://bioinfo.cnio.es/~rdiaz/#cursosR.>).

R, ESS + XEmacs bajo GNU/Linux

- Obtener XEmacs y ESS.

R, ESS + XEmacs bajo GNU/Linux

- Obtener XEmacs y ESS.
- En algunas distribuciones, ESS precompilado (ej. Debian). En otras hay que instalar desde fichero tar.gz
(<http://software.biostat.washington.edu/statsoft/ess/>).

R, ESS + XEmacs bajo GNU/Linux

- Obtener XEmacs y ESS.
- En algunas distribuciones, ESS precompilado (ej. Debian). En otras hay que instalar desde fichero tar.gz (<http://software.biostat.washington.edu/statsoft/ess/>).
- Si se quiere replicar comportamiento bajo windows:
 - Crear .xemacs en home.
 - Obtener xemacs-files.tar.gz de mi página y descomprimir.
 - Copiar RDU.Linux.like.Windows.init.el a
/ .xemacs/init.el.
 - Copiar también custom.el.
 - Poner iconos (xpm) en su sitio (ej.,
/usr/lib/xemacs-x.y.z/etc/toolbar.)

R, ESS + XEmacs bajo GNU/Linux (II)

Si usamos el método "como en Windows", para **mantener los proyectos diferenciados** necesitaremos iniciar xemacs desde directorios distintos.

O editar el fichero init.el, y donde pone (setq ess-ask-for-ess-directory nil) poner (setq ess-ask-for-ess-directory t).

Esto difiere del comportamiento habitual si no usamos el Linux.like.Windows.

Uso básico de ESS + XEmacs (I)

Tomado de *An introduction to ESS + XEmacs for Windows users of R* de J. Fox.

- **Frame:** la ventana de Xemacs.

Uso básico de ESS + XEmacs (I)

Tomado de *An introduction to ESS + XEmacs for Windows users of R* de J. Fox.

- **Frame:** la ventana de Xemacs.
- **Menu bar, toolbar:** pues eso.

Uso básico de ESS + XEmacs (I)

Tomado de *An introduction to ESS + XEmacs for Windows users of R* de J. Fox.

- **Frame:** la ventana de Xemacs.
- **Menu bar, toolbar:** pues eso.
- **Window:** cada una de las ventanas.

Uso básico de ESS + XEmacs (I)

Tomado de *An introduction to ESS + XEmacs for Windows users of R* de J. Fox.

- **Frame:** la ventana de Xemacs.
- **Menu bar, toolbar:** pues eso.
- **Window:** cada una de las ventanas.
- **Minibuffer:** muestra mensajes, se introducen comandos (ej., "C-s", o seleccionar "Edit/FInd" en Menubar).

Uso básico de ESS + XEmacs (I)

Tomado de *An introduction to ESS + XEmacs for Windows users of R* de J. Fox.

- **Frame**: la ventana de Xemacs.
- **Menu bar, toolbar**: pues eso.
- **Window**: cada una de las ventanas.
- **Minibuffer**: muestra mensajes, se introducen comandos (ej., "C-s", o seleccionar "Edit/FInd" en Menubar).
- **Buffer**: lo que Xemacs está editando. Dos importantes:
 - **Inferior R process**: el buffer cuyo **buffer name** es "***R***".
 - **"*scratch buffer"**: donde introduciremos comandos y código para R. Lo renombraremos **lo-que-sea.R**.

Uso básico de ESS + XEmacs (I)

Tomado de *An introduction to ESS + XEmacs for Windows users of R* de J. Fox.

- **Frame**: la ventana de Xemacs.
- **Menu bar, toolbar**: pues eso.
- **Window**: cada una de las ventanas.
- **Minibuffer**: muestra mensajes, se introducen comandos (ej., "C-s", o seleccionar "Edit/FInd" en Menubar).
- **Buffer**: lo que Xemacs está editando. Dos importantes:
 - **Inferior R process**: el buffer cuyo **buffer name** es "***R***".
 - **"*scratch buffer"**: donde introduciremos comandos y código para R. Lo renombraremos **lo-que-sea.R**.
- Podemos tener muchos otros buffers (ej., podemos editar código HTML, o C++, o Python, o Perl, o \LaTeX).

Uso básico de ESS + XEmacs (II)

- **Mode line:** información sobre el buffer que está justo encima. Información sobre **buffer name**, **major mode**, **font**, **column and row**. (Ojo: la numeración de la columna empieza por 0 por defecto).

Uso básico de ESS + XEmacs (II)

- **Mode line:** información sobre el buffer que está justo encima. Información sobre **buffer name**, **major mode**, **font**, **column and row**. (Ojo: la numeración de la columna empieza por 0 por defecto).
- El buffer que estamos editando es lo que está en memoria, no es el fichero mismo. Los cambios sólo se guardan si salvamos el buffer.

Uso básico de ESS + XEmacs (II)

- **Mode line:** información sobre el buffer que está justo encima. Información sobre **buffer name**, **major mode**, **font**, **column and row**. (Ojo: la numeración de la columna empieza por 0 por defecto).
- El buffer que estamos editando es lo que está en memoria, no es el fichero mismo. Los cambios sólo se guardan si salvamos el buffer.
- Se crean ficheros "nombre~" que son copias del anterior estado del buffer. También se puede seleccionar para que Xemacs haga copias periódicas (útil en caso de crash.)

Uso básico de ESS + XEmacs (II)

- **Mode line:** información sobre el buffer que está justo encima. Información sobre **buffer name**, **major mode**, **font**, **column and row**. (Ojo: la numeración de la columna empieza por 0 por defecto).
- El buffer que estamos editando es lo que está en memoria, no es el fichero mismo. Los cambios sólo se guardan si salvamos el buffer.
- Se crean ficheros "nombre~" que son copias del anterior estado del buffer. También se puede seleccionar para que Xemacs haga copias periódicas (útil en caso de crash.)
- La **extensión** de un fichero ayuda a (X)Emacs a elegir el modo de edición apropiado. Para nuestros ficheros con código en R usaremos ".R" (o ".r" o ".s" o ".S"). Pero, ¿qué pasa si abrimos (o creamos) un fichero con extensión Py, cpp, tex, html?

Uso básico de ESS + XEmacs (III)

- Iniciamos XEmacs en el directorio de nuestro interés.

Uso básico de ESS + XEmacs (III)

- Iniciamos XEmacs en el directorio de nuestro interés.
- Abrimos o creamos un fichero con código en R.

Uso básico de ESS + XEmacs (III)

- Iniciamos XEmacs en el directorio de nuestro interés.
- Abrimos o creamos un fichero con código en R.
- El fichero de configuración de J. Fox reserva el buffer inferior para R, así que otros buffers sólo por arriba.

Uso básico de ESS + XEmacs (III)

- Iniciamos XEmacs en el directorio de nuestro interés.
- Abrimos o creamos un fichero con código en R.
- El fichero de configuración de J. Fox reserva el buffer inferior para R, así que otros buffers sólo por arriba.
- Escribimos algo de código en R...

(Cosas a probar)

- Edición de código.

(Cosas a probar)

- Edición de código.
- Envío de una línea.

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.
- Guardar una figura como Metafile y pegarla en Word.

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.
- Guardar una figura como Metafile y pegarla en Word.
- Completar paréntesis.

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.
- Guardar una figura como Metafile y pegarla en Word.
- Completar paréntesis.
- Uso de "STOP" para detener un cálculo.

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.
- Guardar una figura como Metafile y pegarla en Word.
- Completar paréntesis.
- Uso de "STOP" para detener un cálculo.
- Trabajar directamente en *R*, editando comandos, completando comandos, etc.

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.
- Guardar una figura como Metafile y pegarla en Word.
- Completar paréntesis.
- Uso de "STOP" para detener un cálculo.
- Trabajar directamente en *R*, editando comandos, completando comandos, etc.
- Podemos instalar paquetes desde "ESS/R" (tenemos que estar en un buffer con código R).

(Cosas a probar)

- Edición de código.
- Envío de una línea.
- Envío de una región marcada.
- Envío de un buffer entero.
- Guardar una figura como Metafile y pegarla en Word.
- Completar paréntesis.
- Uso de "STOP" para detener un cálculo.
- Trabajar directamente en *R*, editando comandos, completando comandos, etc.
- Podemos instalar paquetes desde "ESS/R" (tenemos que estar en un buffer con código R).
- Salir de R. Es preferible hacer explícito la salida de R via "q()" (en vez de usar el menú de "File/ExitR").

```
> x1 <- rnorm(100)
> x2 <- rnorm(100)
> plot(x1, x2)
> rug(x1)
>
> typo.paciente <- factor(c(rep("e", 50), rep("s", 50)))
> plot(x1, x2, type = "n", xlab = "gen A", ylab = "gen B")
> points(x1, x2, col = c("red", "blue")[typo.paciente])
>
> par(mfrow = c(2,2))
> typo.paciente <-
+     factor(c(rep("Enfermo", 50), rep("Sano", 50)))
>
> plot(x1, x2, type = "n", xlab = "gen A", ylab = "gen B")
> points(x1, x2, col = c("red", "blue")[typo.paciente], pch = 19)
> boxplot(x1 ~ typo.paciente, ylab = "Expresión normalizada",
+           xlab = "Tipo de paciente")
> hist(x1)
> hist(x2, main = "Histograma del gen B")
```

Interludio: guardando gráficas

- Hemos guardado una figura copiándola al portapapeles.

Interludio: guardando gráficas

- Hemos guardado una figura copiándola al portapapeles.
- El menú ofrece otras opciones... pero no funcionan bien.

Interludio: guardando gráficas

- Hemos guardado una figura copiándola al portapapeles.
- El menú ofrece otras opciones... pero no funcionan bien.
- Si queremos hacer uso de estas opciones, más sencillo arrancar Rgui, copiar el código, y guardar la figura como nos parezca.

Interludio: guardando gráficas

- Hemos guardado una figura copiándola al portapapeles.
- El menú ofrece otras opciones... pero no funcionan bien.
- Si queremos hacer uso de estas opciones, más sencillo arrancar Rgui, copiar el código, y guardar la figura como nos parezca.
- También podemos usar comandos como `dev.copy()` , `dev.copy2eps()` , `pdf()` (ver `?dev.copy`).

-
- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:

- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:
 - Incluir en documentos de \LaTeX (en ps o pdf) y en algunos casos de Word (como wmf; ej., si nos hacen usar macros que piden inserción de figuras).

- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:
 - Incluir en documentos de \LaTeX (en ps o pdf) y en algunos casos de Word (como wmf; ej., si nos hacen usar macros que piden inserción de figuras).
 - Para mantener una copia de la figura en formato no Window-céntrico (ej., en formato pdf) que podemos mandar por email, etc.

- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:
 - Incluir en documentos de \LaTeX (en ps o pdf) y en algunos casos de Word (como wmf; ej., si nos hacen usar macros que piden inserción de figuras).
 - Para mantener una copia de la figura en formato no Window-céntrico (ej., en formato pdf) que podemos mandar por email, etc.
 - Para añadir figuras en páginas Web.

- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:
 - Incluir en documentos de \LaTeX (en ps o pdf) y en algunos casos de Word (como wmf; ej., si nos hacen usar macros que piden inserción de figuras).
 - Para mantener una copia de la figura en formato no Window-céntrico (ej., en formato pdf) que podemos mandar por email, etc.
 - Para añadir figuras en páginas Web.
 - ¡Recordad que formatos como pdf, o ps son más "portables" que metafile!

- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:
 - Incluir en documentos de \LaTeX (en ps o pdf) y en algunos casos de Word (como wmf; ej., si nos hacen usar macros que piden inserción de figuras).
 - Para mantener una copia de la figura en formato no Window-céntrico (ej., en formato pdf) que podemos mandar por email, etc.
 - Para añadir figuras en páginas Web.
 - ¡Recordad que formatos como pdf, o ps son más "portables" que metafile!
- Por defecto, yo siempre uso `dev.copy2eps()` y `pdf()`.

- Guardar la figura (en formato pdf o emf o lo que sea) es más útil que copiar al clipboard para:
 - Incluir en documentos de \LaTeX (en ps o pdf) y en algunos casos de Word (como wmf; ej., si nos hacen usar macros que piden inserción de figuras).
 - Para mantener una copia de la figura en formato no Window-céntrico (ej., en formato pdf) que podemos mandar por email, etc.
 - Para añadir figuras en páginas Web.
 - ¡Recordad que formatos como pdf, o ps son más "portables" que metafile!
- Por defecto, yo siempre uso `dev.copy2eps()` y `pdf()`.
- En GNU/Linux no hay un "File/Save As" para las figuras.

Shortcuts útiles en ESS y XEmacs

- Los siguientes funcionan tanto con como sin las modificaciones de J. Fox.

Shortcuts útiles en ESS y XEmacs

- Los siguientes funcionan tanto con como sin las modificaciones de J. Fox.
- "M-x comment-region" ("M" es la "Meta key" que corresponde a "Alt").

Shortcuts útiles en ESS y XEmacs

- Los siguientes funcionan tanto con como sin las modificaciones de J. Fox.
- "M-x comment-region" ("M" es la "Meta key" que corresponde a "Alt").
- "C-s": incremental forward search.

Shortcuts útiles en ESS y XEmacs

- Los siguientes funcionan tanto con como sin las modificaciones de J. Fox.
- "M-x comment-region" ("M" es la "Meta key" que corresponde a "Alt").
- "C-s": incremental forward search.
- "C-r": incremental backward search.

Shortcuts útiles en ESS y XEmacs

- Los siguientes funcionan tanto con como sin las modificaciones de J. Fox.
- "M-x comment-region" ("M" es la "Meta key" que corresponde a "Alt").
- "C-s": incremental forward search.
- "C-r": incremental backward search.
- "M-Shift-5": query replace.

Shortcuts útiles en ESS y XEmacs

- Los siguientes funcionan tanto con como sin las modificaciones de J. Fox.
- "M-x comment-region" ("M" es la "Meta key" que corresponde a "Alt").
- "C-s": incremental forward search.
- "C-r": incremental backward search.
- "M-Shift-5": query replace.
- "M-g": Ir a linea.

Un ejemplo más largo

- Vamos a ordenar un conjunto de datos en función del p-value del estadístico de la t. (Simulamos los datos; sujetos en columnas, "genes" en filas.)

Un ejemplo más largo

- Vamos a ordenar un conjunto de datos en función del p-value del estadístico de la t. (Simulamos los datos; sujetos en columnas, "genes" en filas.)

- ```
> data <- matrix(rnorm(50 * 100), ncol = 50)
> clase <- factor(c(rep("sano", 20),
+ rep("enfermo", 30)))
> tmp <- t.test(data[1,] ~ clase)
> tmp
> attributes(tmp)
> tmp$p.value
> resultado <- apply(data, 1,
+ function(t.test(x ~ clase)$p.value))
> hist(resultado); order(resultado)
> which(resultado < 0.05)
```

## *Segundo ejemplo*

---

- Además de data (ej., medidas de expresión con la técnica A) tenemos dataB (con la técnica B). Queremos seleccionar aquellos genes con alta correlación positiva.

# *Segundo ejemplo*

---

- Además de data (ej., medidas de expresión con la técnica A) tenemos dataB (con la técnica B). Queremos seleccionar aquellos genes con alta correlación positiva.

- ```
> dataB <- matrix(rnorm(50 * 100), ncol = 50)
> correlaciones <- apply(cbind(data,dataB), 1,
+   function(x) cor(x[1:50], x[51:100]))
> order(correlaciones)
> which(correlaciones > 0.7)
> hist(correlaciones)
```

Manejo de datos en R

- Tipos de objetos

Manejo de datos en R

- Tipos de objetos
- Operadores básicos

Manejo de datos en R

- Tipos de objetos
- Operadores básicos
- Generación de secuencias

Manejo de datos en R

- Tipos de objetos
- Operadores básicos
- Generación de secuencias
- Acceso a elementos

Manejo de datos en R

- Tipos de objetos
- Operadores básicos
- Generación de secuencias
- Acceso a elementos
- Ordenación

Manejo de datos en R

- Tipos de objetos
- Operadores básicos
- Generación de secuencias
- Acceso a elementos
- Ordenación
- La familia apply

Tipos de objetos (I)

- (Casi) todo en R es un objeto.

Tipos de objetos (I)

- (Casi) todo en R es un objeto.
- **vector** Colección ordenada elementos del mismo tipo.

```
> x <- c(1, 2, 3); y <- c("a", "b", "Hola")  
> z1 <- c(TRUE, TRUE, FALSE)
```

Tipos de objetos (I)

- (Casi) todo en R es un objeto.
- **vector** Colección ordenada elementos del mismo tipo.

```
> x <- c(1, 2, 3); y <- c("a", "b", "Hola")  
> z1 <- c(TRUE, TRUE, FALSE)
```

- **array** Generalización multidimensional de vector. Elementos del mismo tipo.

Tipos de objetos (I)

- (Casi) todo en R es un objeto.
- **vector** Colección ordenada elementos del mismo tipo.

```
> x <- c(1, 2, 3); y <- c("a", "b", "Hola")  
> z1 <- c(TRUE, TRUE, FALSE)
```

- **array** Generalización multidimensional de vector. Elementos del mismo tipo.
- **data frame** Como array, pero permiten elementos (columnas) de distintos tipos. El objeto más habitual para manejo de datos procedentes de experimentos.

```
> my.data.frame <-  
+   data.frame(ID = c("gen0", "genB", "genZ"),  
+   subj1 = c(10, 25, 33), subj2 = c(NA, 34, 15),  
+   oncogen = c(TRUE, TRUE, FALSE),  
+   loc = c(1,30, 125))
```

Tipos de objetos (II)

- **factor** Un tipo de vector para datos categóricos.

Tipos de objetos (II)

- **factor** Un tipo de vector para datos categóricos.
- **list** Un "vector generalizado". Cada lista está formada por componentes (que pueden ser otras listas), y cada componente puede ser de un tipo distinto. Son unos "contenedores generales".

```
> una.lista <- c(un.vector = 1:10,  
+      una.palabra = "Hola",  
+      una.matriz = matrix(rnorm(20), ncol = 5),  
+      otra.lista = c(a = 5,  
+                      b = factor(c("a", "b"))))
```

Tipos de objetos (II)

- **factor** Un tipo de vector para datos categóricos.
- **list** Un "vector generalizado". Cada lista está formada por componentes (que pueden ser otras listas), y cada componente puede ser de un tipo distinto. Son unos "contenedores generales".

```
> una.lista <- c(un.vector = 1:10,  
+      una.palabra = "Hola",  
+      una.matriz = matrix(rnorm(20), ncol = 5),  
+      otra.lista = c(a = 5,  
+                      b = factor(c("a", "b"))))
```

- **funciones** Código.

Tipos de objetos (II)

- **factor** Un tipo de vector para datos categóricos.
- **list** Un "vector generalizado". Cada lista está formada por componentes (que pueden ser otras listas), y cada componente puede ser de un tipo distinto. Son unos "contenedores generales".

```
> una.lista <- c(un.vector = 1:10,  
+      una.palabra = "Hola",  
+      una.matriz = matrix(rnorm(20), ncol = 5),  
+      otra.lista = c(a = 5,  
+                      b = factor(c("a", "b"))))
```

- **funciones** Código.

Atributos de los objetos

- `x <- 1:15; length(x)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`
- `mode(x); mode(y); z <- c(TRUE, FALSE); mode(z)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`
- `mode(x); mode(y); z <- c(TRUE, FALSE); mode(z)`
- `attributes(y); w <- list(a = 1:3, b = 5); attributes(w)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`
- `mode(x); mode(y); z <- c(TRUE, FALSE); mode(z)`
- `attributes(y); w <- list(a = 1:3, b = 5); attributes(w)`
- `y <- as.data.frame(y); attributes(y)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`
- `mode(x); mode(y); z <- c(TRUE, FALSE); mode(z)`
- `attributes(y); w <- list(a = 1:3, b = 5); attributes(w)`
- `y <- as.data.frame(y); attributes(y)`
- `f1 <- function(x) {return(2 * x)}; attributes(f1); is.function(f1)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`
- `mode(x); mode(y); z <- c(TRUE, FALSE); mode(z)`
- `attributes(y); w <- list(a = 1:3, b = 5); attributes(w)`
- `y <- as.data.frame(y); attributes(y)`
- `f1 <- function(x) {return(2 * x)}; attributes(f1); is.function(f1)`
- `x1 <- 1:5; x2 <- c(1, 2, 3, 4, 5); typeof(x2); typeof(x3)`

Atributos de los objetos

- `x <- 1:15; length(x)`
- `y <- matrix(5, nrow = 3, ncol = 4); dim(y)`
- `is.vector(x); is.vector(y); is.array(x)`
- `mode(x); mode(y); z <- c(TRUE, FALSE); mode(z)`
- `attributes(y); w <- list(a = 1:3, b = 5); attributes(w)`
- `y <- as.data.frame(y); attributes(y)`
- `f1 <- function(x) {return(2 * x)}; attributes(f1); is.function(f1)`
- `x1 <- 1:5; x2 <- c(1, 2, 3, 4, 5); typeof(x2); typeof(x3)`
- Los atributos podemos verlos, pero también podemos cambiarlos.

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.
- R es "case-sensitive". `x != x`.

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.
- R es "case-sensitive". `x != x`.
- Hay nombres reservados ("function", "if", etc).

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.
- R es "case-sensitive". `x != x`.
- Hay nombres reservados ("function", "if", etc).
- Otras consideraciones:

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.
- R es "case-sensitive". `x != x`.
- Hay nombres reservados ("function", "if", etc).
- Otras consideraciones:
 - El uso del `.` es diferente al de C++.

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.
- R es "case-sensitive". `x != x`.
- Hay nombres reservados ("function", "if", etc).
- Otras consideraciones:
 - El uso del `.` es diferente al de C++.
 - Mejor evitar nombres que R usa (ej., `"c"`) (pero no es dramático si nos confundimos: podemos arreglarlo).
`c <- 4; x <- c(3, 8); x; rm(c); c(7, 9)`

Nombres de objetos

- Los nombres válidos para un objeto son combinaciones de letras, números, y el punto (".").
- Los nombres no pueden empezar con un número.
- R es "case-sensitive". `x != x`.
- Hay nombres reservados ("function", "if", etc).
- Otras consideraciones:
 - El uso del "." es diferente al de C++.
 - Mejor evitar nombres que R usa (ej., "c") (pero no es dramático si nos confundimos: podemos arreglarlo).
`c <- 4; x <- c(3, 8); x; rm(c); c(7, 9)`
 - Las asignaciones se hacen con "<-", y es buen estilo el rodear "<-" por un espacio a cada lado.
`x<-1:5 # Mal estilo`
`x <- 1:5 # Mucho mejor.`

- Las "reglas" habituales de nombrar objetos en programación (usar estilo consistente, uso razonable del ".", nombres que tengan sentido —equilibrio entre longitud y frecuencia de uso, etc). Por ej., suelo nombrar data frames con la inicial en mayúscula, pero las variables siempre en minúscula. Si varias funciones hacen cosas parecidas a objetos distintos, separo con "." (más fácil luego usar clases).

- Las "reglas" habituales de nombrar objetos en programación (usar estilo consistente, uso razonable del ".", nombres que tengan sentido —equilibrio entre longitud y frecuencia de uso, etc). Por ej., suelo nombrar data frames con la inicial en mayúscula, pero las variables siempre en minúscula. Si varias funciones hacen cosas parecidas a objetos distintos, separo con "." (más fácil luego usar clases).
- Sobre el uso de ";". No es bueno abusar, porque hace el código *muy difícil* de leer. (Pero en estas transparencias, si no lo usara ocuparíamos muchísimo espacio).

Operaciones aritméticas con vectores (I)

- *FUNDAMENTAL*: R puede operar *sobre vectores enteros de un golpe*.

```
> x <- 1:10  
> y <- x/2; z <- x^2; w <- y + z
```

Operaciones aritméticas con vectores (I)

- **FUNDAMENTAL:** R puede operar *sobre vectores enteros de un golpe*.

```
> x <- 1:10  
> y <- x/2; z <- x^2; w <- y + z
```

- Si un elemento es más corto, se "recicla". Es "intuitivo" si la operación es escalar con vector. Pero también ocurre en operaciones entre vectores.

```
> x + 15  
> x2 <- 1:5  
> x + x2
```

Operaciones aritméticas con vectores (I)

- **FUNDAMENTAL:** R puede operar *sobre vectores enteros de un golpe*.

```
> x <- 1:10  
> y <- x/2; z <- x^2; w <- y + z
```

- Si un elemento es más corto, se "recicla". Es "intuitivo" si la operación es escalar con vector. Pero también ocurre en operaciones entre vectores.

```
> x + 15  
> x2 <- 1:5  
> x + x2
```

- El reciclado cuando (vector, escalar) suele ser lo que queremos. Entre vectores no siempre: cuidado. (R da un warning, de momento –S4 classes dan un error).

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es *MUCHO MEJOR* que usar loops:

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es *MUCHO MEJOR* que usar loops:
 - Mucho más claro:

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es *MUCHO MEJOR* que usar loops:
 - Mucho más claro:
 - Es la forma natural de operar sobre objetos enteros.

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es *MUCHO MEJOR* que usar loops:
 - Mucho más claro:
 - Es la forma natural de operar sobre objetos enteros.
 - Código más fácil de entender.

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es *MUCHO MEJOR* que usar loops:
 - Mucho más claro:
 - Es la forma natural de operar sobre objetos enteros.
 - Código más fácil de entender.
 - Más sencillo de modificar y mantener.

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es **MUCHO MEJOR** que usar loops:
 - Mucho más claro:
 - Es la forma natural de operar sobre objetos enteros.
 - Código más fácil de entender.
 - Más sencillo de modificar y mantener.
 - Más fácil hacer debugging.

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es *MUCHO MEJOR* que usar loops:
 - Mucho más claro:
 - Es la forma natural de operar sobre objetos enteros.
 - Código más fácil de entender.
 - Más sencillo de modificar y mantener.
 - Más fácil hacer debugging.
 - Más rápido de escribir (y no sólo porque muchas menos líneas!).

Operaciones aritméticas con vectores (II)

- Operar sobre vectores enteros es **MUCHO MEJOR** que usar loops:
 - Mucho más claro:
 - Es la forma natural de operar sobre objetos enteros.
 - Código más fácil de entender.
 - Más sencillo de modificar y mantener.
 - Más fácil hacer debugging.
 - Más rápido de escribir (y no sólo porque muchas menos líneas!).
 - Más eficiente (en tiempo y memoria).

Operaciones aritméticas con vectores (III)

- `+ , - , * , / , ^ , %% , % / %.`

Operaciones aritméticas con vectores (III)

- `+`, `-`, `*`, `/`, `^`, `%%`, `%/%.`
- `log`, `log10`, `log2`, `log(x, base)`, `exp`, `sin`, `cos`,
`tan`, `sqrt`

Operaciones aritméticas con vectores (III)

- +, -, *, /, ^, %% , %/ %.
- log, log10, log2, log(x, base), exp, sin, cos, tan, sqrt
- Otras:

Operaciones aritméticas con vectores (III)

- +, -, *, /, ^, %% , %/ %.
- log, log10, log2, log(x, base), exp, sin, cos, tan, sqrt
- Otras:
 - max, min, range, mean, var, sd, sum, prod

Operaciones aritméticas con vectores (III)

- +, -, *, /, ^, %% , %/ %.
- log, log10, log2, log(x, base), exp, sin, cos, tan, sqrt
- Otras:
 - max, min, range, mean, var, sd, sum, prod
 - which.max, which.min.

Operaciones aritméticas con vectores (III)

- +, -, *, /, ^, %% , %/ %.
- log, log10, log2, log(x, base), exp, sin, cos, tan, sqrt
- Otras:
 - max, min, range, mean, var, sd, sum, prod
 - which.max, which.min.
 - pmax, pmin. max and min devuelven el máximo y el mínimo de un conjunto de números; devuelven un solo número aunque les pasemos varios vectores. pmax, pmin devuelven un vector que contiene el elemento max, min en esa posición.

Operaciones aritméticas con vectores (III)

- +, -, *, /, ^, %% , %/ %.
- log, log10, log2, log(x, base), exp, sin, cos, tan, sqrt
- Otras:
 - max, min, range, mean, var, sd, sum, prod
 - which.max, which.min.
 - pmax, pmin. max and min devuelven el máximo y el mínimo de un conjunto de números; devuelven un solo número aunque les pasemos varios vectores. pmax, pmin devuelven un vector que contiene el elemento max, min en esa posición.
 - cumsum, cumprod, diff.

Operadores comparativos y lógicos

- < , > , <= , >= , == , !=

Operadores comparativos y lógicos

- < , > , <= , >= , == , !=
- ! , & , | , xor() y los parecidos && , ||

Operadores comparativos y lógicos

- <, >, <=, >=, ==, !=
- !, &, |, xor() y los parecidos &&, ||
- x <- 5; x < 5; x >= 5; x == 6; x != 5

Operadores comparativos y lógicos

- <, >, <=, >=, ==, !=
- !, &, |, xor() y los parecidos &&, ||
- x <- 5; x < 5; x >= 5; x == 6; x != 5
- y <- c(TRUE, FALSE); !y; z <- c(TRUE, TRUE)
xor(y, z)

Operadores comparativos y lógicos

- <, >, <=, >=, ==, !=
- !, &, |, xor() y los parecidos &&, ||
- x <- 5; x < 5; x >= 5; x == 6; x != 5
- y <- c(TRUE, FALSE); !y; z <- c(TRUE, TRUE)
xor(y, z)
- y & z; y | z

Operadores comparativos y lógicos

- <, >, <=, >=, ==, !=
- !, &, |, xor() y los parecidos &&, ||
- x <- 5; x < 5; x >= 5; x == 6; x != 5
- y <- c(TRUE, FALSE); !y; z <- c(TRUE, TRUE)
xor(y, z)
- y & z; y | z
- Las formas &&, || son "short-circuit operators" que se suelen usar dentro de if statements. Se aplican a vectores de longitud uno y sólo evaluan el segundo argumento si es preciso.

```
if (is.numeric(x) && min(x) > 0) {lo que sea....  
min(x) no tiene sentido si x no es numérico.}
```

Operadores comparativos y lógicos

- <, >, <=, >=, ==, !=
- !, &, |, xor() y los parecidos &&, ||
- x <- 5; x < 5; x >= 5; x == 6; x != 5
- y <- c(TRUE, FALSE); !y; z <- c(TRUE, TRUE)
xor(y, z)
- y & z; y | z
- Las formas &&, || son "short-circuit operators" que se suelen usar dentro de if statements. Se aplican a vectores de longitud uno y sólo evaluan el segundo argumento si es preciso.

```
if (is.numeric(x) && min(x) > 0) {lo que sea....  
min(x) no tiene sentido si x no es numérico.
```
- 0 + y; as.numeric(y); mode(y) <- "numeric"

Operaciones de conjuntos

- `x <- 1:5; y <- c(1, 3, 7:10)`

Operaciones de conjuntos

- `x <- 1:5; y <- c(1, 3, 7:10)`
- `union(x, y)`

Operaciones de conjuntos

- `x <- 1:5; y <- c(1, 3, 7:10)`
- `union(x, y)`
- `intersect(x, y)`

Operaciones de conjuntos

- `x <- 1:5; y <- c(1, 3, 7:10)`
- `union(x, y)`
- `intersect(x, y)`
- `setdiff(y, x)`

Operaciones de conjuntos

- `x <- 1:5; y <- c(1, 3, 7:10)`
- `union(x, y)`
- `intersect(x, y)`
- `setdiff(y, x)`
- `v <- c("bcA1", "bcA2", "blx1")`
`w <- c("bcA2", "xA3")`

Operaciones de conjuntos

- `x <- 1:5; y <- c(1, 3, 7:10)`
- `union(x, y)`
- `intersect(x, y)`
- `setdiff(y, x)`
- `v <- c("bcA1", "bcA2", "blx1")`
`w <- c("bcA2", "xA3")`
- `union(v, w)`
`intersect(v, w)`
`setdiff(w, v)`
`setdiff(v, w)`

Generación de secuencias aleatorias

- Muestrear un vector (imprescindible en bootstrapping y cross-validation):

```
> sample(5)  
  
> sample(5, 3)  
  
> x <- 1:10  
  
> sample(x)  
  
> sample(x, replace = TRUE)  
  
> sample(x, length = 2* length(x), replace = TRUE)  
  
> probs <- x/sum(x)  
  
> sample(x, prob = probs)
```

- Números aleatorios: usar **rFuncionDistribucion** (con sus parámetros).

```
> rnorm(10)  
> rnorm(10, mean = 13, sd = 18)  
> runif(15)  
> runif(8, min = 3, max = 59)
```

- Números aleatorios: usar `rFuncionDistribucion` (con sus parámetros).

```
> rnorm(10)  
> rnorm(10, mean = 13, sd = 18)  
> runif(15)  
> runif(8, min = 3, max = 59)
```
- Muchas funciones de distribución: *lognormal*, *t*, *F*, χ^2 , *exponential*, *gamma*, *beta*, *poisson*, *binomial*, etc. Ver, por ejemplo, sección 8.1 (p. 34) de *An introduction to R* y p. 16 de *R para principiantes*, de E. Paradis. (Con esas funciones podemos también obtener la densidad, quantiles y función cumulativa de todas estas distribuciones).

Generación de secuencias: seq

- `x <- c(1, 2, 3, 4, 5)`

Generación de secuencias: *seq*

- `x <- c(1, 2, 3, 4, 5)`
- `x <- 1:10; y <- -5:3`

Generación de secuencias: *seq*

- `x <- c(1, 2, 3, 4, 5)`
- `x <- 1:10; y <- -5:3`
- `x <- seq(from = 2, to = 18, by = 2)`

Generación de secuencias: *seq*

- `x <- c(1, 2, 3, 4, 5)`
- `x <- 1:10; y <- -5:3`
- `x <- seq(from = 2, to = 18, by = 2)`
- `x <- seq(from = 2, to = 18, length = 30)`

Generación de secuencias: *seq*

- `x <- c(1, 2, 3, 4, 5)`
- `x <- 1:10; y <- -5:3`
- `x <- seq(from = 2, to = 18, by = 2)`
- `x <- seq(from = 2, to = 18, length = 30)`
- `x <- 1:10; y <- seq(along(x))`

Generación de secuencias: *seq*

- `x <- c(1, 2, 3, 4, 5)`
- `x <- 1:10; y <- -5:3`
- `x <- seq(from = 2, to = 18, by = 2)`
- `x <- seq(from = 2, to = 18, length = 30)`
- `x <- 1:10; y <- seq(along(x))`
- `z2 <- c(1:5, 7:10, seq(from = -7, to = 5, by = 2))`

Generación de secuencias: `rep`

- `rep(1, 5)`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`
- `rep(x, length = 8)`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`
- `rep(x, length = 8)`
- `gl(3, 5) # como rep(1:3, rep(5, 3))`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`
- `rep(x, length = 8)`
- `gl(3, 5) # como rep(1:3, rep(5, 3))`
- `gl(4, 1, length = 20) #Ojo: gl genera factores`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`
- `rep(x, length = 8)`
- `gl(3, 5) # como rep(1:3, rep(5, 3))`
- `gl(4, 1, length = 20) #Ojo: gl genera factores`
- `gl(3, 4, label = c("Sano", "Enfermo", "Muerto"))`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`
- `rep(x, length = 8)`
- `gl(3, 5) # como rep(1:3, rep(5, 3))`
- `gl(4, 1, length = 20) #Ojo: gl genera factores`
- `gl(3, 4, label = c("Sano", "Enfermo", "Muerto"))`
- `expand.grid(edad = c(10, 18, 25),
 sexo = c("Macho", "Hembra"), loc = 1:3)`

Generación de secuencias: *rep*

- `rep(1, 5)`
- `x <- 1:3; rep(x, 2)`
- `y <- rep(5, 3); rep(x, y)`
- `rep(1:3, rep(5, 3))`
- `rep(x, x)`
- `rep(x, length = 8)`
- `gl(3, 5) # como rep(1:3, rep(5, 3))`
- `gl(4, 1, length = 20) #Ojo: gl genera factores`
- `gl(3, 4, label = c("Sano", "Enfermo", "Muerto"))`
- `expand.grid(edad = c(10, 18, 25),
 sexo = c("Macho", "Hembra"), loc = 1:3)`
- Podemos combinar: `z5 <- c(1:5, rep(8, 3))`

Indexación de vectores

- Una de las gran virtudes de R es la flexibilidad en el acceso a los elementos (de vectores, arrays, data frames, etc).

Indexación de vectores

- Una de las gran virtudes de R es la flexibilidad en el acceso a los elementos (de vectores, arrays, data frames, etc).
- `x <- 1:5; x[1]; x[3]`

Indexación de vectores

- Una de las gran virtudes de R es la flexibilidad en el acceso a los elementos (de vectores, arrays, data frames, etc).
- ```
x <- 1:5; x[1]; x[3]
```
- ```
x[x > 3]
```



```
x > 3
```



```
y <- x > 3
```



```
x[y]
```

Indexación de vectores

- Una de las gran virtudes de R es la flexibilidad en el acceso a los elementos (de vectores, arrays, data frames, etc).
- ```
x <- 1:5; x[1]; x[3]
```
- ```
x[x > 3]
```



```
x > 3
```



```
y <- x > 3
```



```
x[y]
```
- ```
x[-c(1, 4)]; y <- c(1, 2, 5); x[y]
```

# *Indexación de vectores*

- Una de las gran virtudes de R es la flexibilidad en el acceso a los elementos (de vectores, arrays, data frames, etc).

- `x <- 1:5; x[1]; x[3]`

- `x[x > 3]`

```
x > 3
```

```
y <- x > 3
```

```
x[y]
```

- `x[-c(1, 4)]; y <- c(1, 2, 5); x[y]`

- `names(x) <- c("a", "b", "c", "d", "patata")`

# *Indexación de vectores*

- Una de las gran virtudes de R es la flexibilidad en el acceso a los elementos (de vectores, arrays, data frames, etc).
- ```
x <- 1:5; x[1]; x[3]
```
- ```
x[x > 3]
```

  

```
x > 3
```

  

```
y <- x > 3
```

  

```
x[y]
```
- ```
x[-c(1, 4)]; y <- c(1, 2, 5); x[y]
```
- ```
names(x) <- c("a", "b", "c", "d", "patata")
```
- ```
x[c("b", "patata")]
```

-
- Podemos indexar un vector de cuatro formas:

-
- Podemos indexar un vector de cuatro formas:
 - Un vector lógico.

- Podemos indexar un vector de cuatro formas:
 - Un vector lógico.
 - Un vector de enteros (integers) positivos.

- Podemos indexar un vector de cuatro formas:
 - Un vector lógico.
 - Un vector de enteros (integers) positivos.
 - Un vector de enteros negativos.

- Podemos indexar un vector de cuatro formas:
 - Un vector lógico.
 - Un vector de enteros (integers) positivos.
 - Un vector de enteros negativos.
 - Un vector de cadenas de caracteres (character strings).

- Podemos indexar un vector de cuatro formas:
 - Un vector lógico.
 - Un vector de enteros (integers) positivos.
 - Un vector de enteros negativos.
 - Un vector de cadenas de caracteres (character strings).
- También, por supuesto, podemos indexar un vector usando cualquier expresión o función que resulte una de las cuatro anteriores.

- Podemos indexar un vector de cuatro formas:
 - Un vector lógico.
 - Un vector de enteros (integers) positivos.
 - Un vector de enteros negativos.
 - Un vector de cadenas de caracteres (character strings).
- También, por supuesto, podemos indexar un vector usando cualquier expresión o función que resulte una de las cuatro anteriores.
- No solo podemos acceder (devolver) el/los valores, sino también asignarlos:
`x[c(1, 3)] <- c(25, 79); x[x > 3] <- 97`

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`
- `is.na(v); which(is.na(v))`

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`
- `is.na(v); which(is.na(v))`
- `v == NA` # No funciona! Por qué?

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`
- `is.na(v); which(is.na(v))`
- `v == NA # No funciona! Por qué?`
- Sustituir NA por, p.ej., 0: `v[is.na(v)] <- 0.`

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`
- `is.na(v); which(is.na(v))`
- `v == NA` # No funciona! Por qué?
- Sustituir NA por, p.ej., 0: `v[is.na(v)] <- 0.`
- Infinito y NaN (not a number). Son distintos de NA.

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`
- `is.na(v); which(is.na(v))`
- `v == NA` # No funciona! Por qué?
- Sustituir NA por, p.ej., 0: `v[is.na(v)] <- 0.`
- Infinito y NaN (not a number). Son distintos de NA.
- `5/0; -5/0; 0/0`

Interludio: NA, NaN, Inf

- "NA" es el código de "Not available".
- `v <- c(1:3, NA)`
- `is.na(v); which(is.na(v))`
- `v == NA` # No funciona! Por qué?
- Sustituir NA por, p.ej., 0: `v[is.na(v)] <- 0.`
- Infinito y NaN (not a number). Son distintos de NA.
`5/0; -5/0; 0/0`
- `is.infinite(-5/0); is.nan(0/0); is.na(5/0)`

-
- `xna <- c(1, 2, 3, NA, 4); mean(xna)`
`mean(xna, na.rm = TRUE)`

-
- ```
xna <- c(1, 2, 3, NA, 4); mean(xna)
mean(xna, na.rm = TRUE)
```
  - Lo mismo con otras funciones.

- 
- ```
xna <- c(1, 2, 3, NA, 4); mean(xna)
mean(xna, na.rm = TRUE)
```
 - Lo mismo con otras funciones.
 - Para "modelling functions" (ej., lm) lo mejor es usar "na.omit" o "na.exclude" ("na.exclude" más conveniente para generar predicciones, residuos, etc).

- ```
xna <- c(1, 2, 3, NA, 4); mean(xna)
mean(xna, na.rm = TRUE)
```
- Lo mismo con otras funciones.
- Para "modelling functions" (ej., lm) lo mejor es usar "na.omit" o "na.exclude" ("na.exclude" más conveniente para generar predicciones, residuos, etc).
- Eliminar todos los NA:

```
> XNA <- matrix(c(1, 2, NA, 3, NA, 4), nrow = 3)
> XNA
> X.no.na <- na.omit(XNA)
```

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`
- `rank(x1)`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`
- `rank(x1)`
- `x1[order(x1)]`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`
- `rank(x1)`
- `x1[order(x1)]`
- `x2 <- c(1, 2, 2, 3, 3, 4); rank(x2)`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`
- `rank(x1)`
- `x1[order(x1)]`
- `x2 <- c(1, 2, 2, 3, 3, 4); rank(x2)`
- `min(x1); which.min(x1); which(x1 == min(x1))`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`
- `rank(x1)`
- `x1[order(x1)]`
- `x2 <- c(1, 2, 2, 3, 3, 4); rank(x2)`
- `min(x1); which.min(x1); which(x1 == min(x1))`
- `y <- c(1, 1, 2, 2); order(y, x)`

# *Ordenando vectores*

---

- `x <- c(5, 1, 8, 3)`
- `order(x1)`
- `sort(x1)`
- `rank(x1)`
- `x1[order(x1)]`
- `x2 <- c(1, 2, 2, 3, 3, 4); rank(x2)`
- `min(x1); which.min(x1); which(x1 == min(x1))`
- `y <- c(1, 1, 2, 2); order(y, x)`
- `order` y `sort` admiten "decreasing = TRUE".

# Vectores de caracteres

---

- codigos <- paste(c("A", "B"), 2:3, sep = "")

# Vectores de caracteres

---

- `codigos <- paste(c("A", "B"), 2:3, sep = "")`
- `codigos <- paste(c("A", "B"), 2:3, sep = ".")`

# Vectores de caracteres

---

- `codigos <- paste(c("A", "B"), 2:3, sep = "")`
- `codigos <- paste(c("A", "B"), 2:3, sep = ".")`
- `juntar <-  
paste(c("una", "frase", "tonta", collapse = ""))`

# Vectores de caracteres

---

- `codigos <- paste(c("A", "B"), 2:3, sep = "")`
- `codigos <- paste(c("A", "B"), 2:3, sep = ".")`
- `juntar <-  
 paste(c("una", "frase", "tonta", collapse = ""))`
- `columna.a <- LETTERS[1:5]; columna.b <- 10:15;  
juntar <- paste(columna.a, columna.b, sep = "")`

# Vectores de caracteres

---

- `codigos <- paste(c("A", "B"), 2:3, sep = "")`
- `codigos <- paste(c("A", "B"), 2:3, sep = ".")`
- `juntar <-  
 paste(c("una", "frase", "tonta", collapse = ""))`
- `columna.a <- LETTERS[1:5]; columna.b <- 10:15;  
juntar <- paste(columna.a, columna.b, sep = "")`
- `substr("abcdef", 2, 4)`

# Vectores de caracteres

---

- `codigos <- paste(c("A", "B"), 2:3, sep = "")`
- `codigos <- paste(c("A", "B"), 2:3, sep = ".")`
- `juntar <-  
 paste(c("una", "frase", "tonta", collapse = ""))`
- `columna.a <- LETTERS[1:5]; columna.b <- 10:15;  
juntar <- paste(columna.a, columna.b, sep = "")`
- `substr("abcdef", 2, 4)`
- `x <- paste(LETTERS[1:5], collapse = "")  
substr(x, 3, 5) <- c("uv")`

# Vectores de caracteres

- `codigos <- paste(c("A", "B"), 2:3, sep = "")`
- `codigos <- paste(c("A", "B"), 2:3, sep = ".")`
- `juntar <-  
 paste(c("una", "frase", "tonta", collapse = ""))`
- `columna.a <- LETTERS[1:5]; columna.b <- 10:15;  
juntar <- paste(columna.a, columna.b, sep = "")`
- `substr("abcdef", 2, 4)`
- `x <- paste(LETTERS[1:5], collapse = "")  
substr(x, 3, 5) <- c("uv")`
- Otras muchas funciones de manipulación de caracteres, que pueden hacer poco necesario recurrir a Awk, Python, o Perl.  
Ver grep, pmatch, match, tolower, toupper, sub, gsub, regexpr.

# *Factores*

---

- ```
codigo.postal <- c(28430, 28016, 28034);  
mode(codigo.postal)
```

Factores

- ```
codigo.postal <- c(28430, 28016, 28034);
mode(codigo.postal)
```
- Pero *no deberíamos* usar el código postal en, por ej., un ANOVA, como si fuera un vector numérico. El usar códigos (aparentemente) numéricos en análisis estadísticos es una frecuente fuente de errores.

# Factores

---

- `codigo.postal <- c(28430, 28016, 28034);  
mode(codigo.postal)`
- Pero *no deberíamos* usar el código postal en, por ej., un ANOVA, como si fuera un vector numérico. El usar códigos (aparentemente) numéricos en análisis estadísticos es una frecuente fuente de errores.
- `codigo.postal <- factor(codigo.postal) # mejor.`

# Factores

- `codigo.postal <- c(28430, 28016, 28034);  
mode(codigo.postal)`
- Pero *no deberíamos* usar el código postal en, por ej., un ANOVA, como si fuera un vector numérico. El usar códigos (aparentemente) numéricos en análisis estadísticos es una frecuente fuente de errores.
- `codigo.postal <- factor(codigo.postal) # mejor.`
- Si tenemos un vector con caracteres, y lo queremos usar para un análisis, necesitamos convertirlo en un factor. Si no, R, sabiamente, no se deja.

```
y <- rnorm(10); x <- rep(letters[1:5], 2);
aov(y ~ x) # error!
aov(y ~ factor(x)) # funciona.
```

- A veces, al leer datos, un vector de números se convierte en factor. O queremos convertir un vector factor en numérico.

```
x <- c(34, 89, 1000); y <- factor(x); y
as.numeric(y) # Mal:
los valores han sido recodificados.
as.numeric(as.character(y)) # Bien
```

- A veces, al leer datos, un vector de números se convierte en factor. O queremos convertir un vector factor en numérico.

```
x <- c(34, 89, 1000); y <- factor(x); y
as.numeric(y) # Mal:
los valores han sido recodificados.
as.numeric(as.character(y)) # Bien
```

- Podemos fijar el orden de las etiquetas.

```
ftr1 <- factor(c("alto", "bajo", "medio"))
ftr1
ftr1 <- factor(c("alto", "bajo", "medio"),
+ levels = c("bajo", "medio", "alto"))
```

- R también cuenta con "ordered factors". Sólo difieren de los factors "normales" en algunos análisis (tipo de contrastes usados con modelos lineales —treatment vs. polinomial). No lo veremos.

- R también cuenta con "ordered factors". Sólo difieren de los factors "normales" en algunos análisis (tipo de contrastes usados con modelos lineales —treatment vs. polinomial). No lo veremos.
- Como *medida de "higiene mental y analítica"* convirtamos en factores lo que debe serlo. Eliminará muchos errores y es "buena práctica estadística".

- Para **discretizar** datos: usar `cut`:

```
vv <- c(1, 2, 3, 7, 8, 9)
cut1 <- cut(vv, 3); summary(cut1)
cut2 <- cut(vv, quantile(vv, c(0, 1/3, 2/3, 1)),
+ include.lowest = TRUE)
summary(cut2)
```

- Para **discretizar** datos: usar `cut`:

```
vv <- c(1, 2, 3, 7, 8, 9)
cut1 <- cut(vv, 3); summary(cut1)
cut2 <- cut(vv, quantile(vv, c(0, 1/3, 2/3, 1)),
+ include.lowest = TRUE)
summary(cut2)
```

- Ver también `split(x, f)`: divide datos del vector x usando los grupos definidos en f.

# *Matrices y arrays (I)*

---

- Generalizaciones de vectores. Nos centraremos en arrays de 2 dimensiones, pero podemos usar un número arbitrario. Con dos dimensiones, `array` y `matrix` proveen similar funcionalidad, pero `matrix` es más cómoda.

# *Matrices y arrays (I)*

---

- Generalizaciones de vectores. Nos centraremos en arrays de 2 dimensiones, pero podemos usar un número arbitrario. Con dos dimensiones, `array` y `matrix` proveen similar funcionalidad, pero `matrix` es más cómoda.
- `a1 <- array(9, dim = c(5,4))`

# *Matrices y arrays (I)*

- Generalizaciones de vectores. Nos centraremos en arrays de 2 dimensiones, pero podemos usar un número arbitrario. Con dos dimensiones, `array` y `matrix` proveen similar funcionalidad, pero `matrix` es más cómoda.
- ```
a1 <- array(9, dim = c(5,4))
```
- ```
a2 <- matrix(1:20, nrow = 5) # column-major-order,
como en FORTRAN.
```
- ```
a3 <- matrix(1:20, nrow = 5, byrow = TRUE)
```

Matrices y arrays (I)

- Generalizaciones de vectores. Nos centraremos en arrays de 2 dimensiones, pero podemos usar un número arbitrario. Con dos dimensiones, `array` y `matrix` proveen similar funcionalidad, pero `matrix` es más cómoda.
- ```
a1 <- array(9, dim = c(5,4))
```
- ```
a2 <- matrix(1:20, nrow = 5) # column-major-order,
# como en FORTRAN.
```



```
a3 <- matrix(1:20, nrow = 5, byrow = TRUE)
```
- ```
a4 <- 1:20; dim(a4) <- c(5, 4)
```

# *Matrices y arrays (II)*

---

- Indexado de arrays es similar a vectores.

# *Matrices y arrays (II)*

---

- Indexado de arrays es similar a vectores.
- `a4[1, 4]; a4[1, ]; a4[, 2]; a4[c(1, 3), c(2, 4)]`

# Matrices y arrays (II)

- Indexado de arrays es similar a vectores.
- `a4[1, 4]; a4[1, ]; a4[, 2]; a4[c(1, 3), c(2, 4)]`
- Podemos usar los cuatro métodos para vectores y también **indexar con una matriz**. En la matriz con los índices cada fila extrae un elemento; su fila (primera dimensión) se especifica por el elemento en la primera columna, y su columna por el elemento en la segunda columna.

```
> im <- matrix(c(1, 3, 2, 4), nrow = 2)
> im
> a4[im]
> # Por qué difiere de a4[c(1, 3), c(2, 4)]?
```

# Matrices y arrays (II)

- Indexado de arrays es similar a vectores.
- `a4[1, 4]; a4[1, ]; a4[, 2]; a4[c(1, 3), c(2, 4)]`
- Podemos usar los cuatro métodos para vectores y también **indexar con una matriz**. En la matriz con los índices cada fila extrae un elemento; su fila (primera dimensión) se especifica por el elemento en la primera columna, y su columna por el elemento en la segunda columna.

```
> im <- matrix(c(1, 3, 2, 4), nrow = 2)
> im
> a4[im]
> # Por qué difiere de a4[c(1, 3), c(2, 4)]?
```

- El indexado con matrices se extiende también a arrays de más dos dimensiones.

- Ordenar un array usando una de las columnas:

```
o.array <- matrix(rnorm(20), ncol = 4)
o.array <- o.array[order(o.array[, 1]),]
```

- Ordenar un array usando una de las columnas:

```
o.array <- matrix(rnorm(20), ncol = 4)
o.array <- o.array[order(o.array[, 1]),]
```

- Selección aleatoria de filas:

```
sample.of.rows <-
+ o.array[sample(1:dim(o.array)[1],
+ replace = TRUE),]
```

# *Matrices y arrays (III)*

---

- `a4[1, ]` #es un vector!

# *Matrices y arrays (III)*

---

- `a4[1, ]` #es un vector!
- Para evitar "dropping indices" usar  
`a4[1, , drop = FALSE]`  
`a4[, 3, drop = FALSE].`

# *Matrices y arrays (III)*

---

- `a4[1, ]` #es un vector!
- Para evitar "dropping indices" usar  
`a4[1, , drop = FALSE]`  
`a4[, 3, drop = FALSE].`
- Esto mismo se aplica a arrays de más de dos dimensiones.

# Matrices y arrays (III)

---

- `a4[1, ]` #es un vector!
- Para evitar "dropping indices" usar  
`a4[1, , drop = FALSE]`  
`a4[, 3, drop = FALSE].`
- Esto mismo se aplica a arrays de más de dos dimensiones.
- "drop = FALSE" generalmente debe usarse, de forma defensiva, cuando escribamos funciones (si no, nos encontraremos con errores como "`Error in apply(a6, 2, mean) : dim(x) must have a positive length.`".)

# Matrices y arrays (III)

- `a4[1, ]` #es un vector!
- Para evitar "dropping indices" usar  
`a4[1, , drop = FALSE]`  
`a4[, 3, drop = FALSE].`
- Esto mismo se aplica a arrays de más de dos dimensiones.
- "drop = FALSE" generalmente debe usarse, de forma defensiva, cuando escribamos funciones (si no, nos encontraremos con errores como `Error in apply(a6, 2, mean) : dim(x) must have a positive length`.)
- Operaciones matriciales: muchas disponibles incluyendo factorizaciones como svd y qr, determinantes, rango, solución de sistemas, etc. Sólo mencionamos diag, producto de matrices (%\*%) y traspuesto (t):

```
a6 <- diag(6); diag(a4) <- -17
a4 %*% a2; a2 %*% t(a2)
```

# **Matrices *rbind*, *cbind* y otros**

---

- Para combinar vectores o arrays para obtener arrays, usamos `rbind`, `cbind`.

# **Matrices *rbind*, *cbind* y otros**

---

- Para combinar vectores o arrays para obtener arrays, usamos *rbind*, *cbind*.

- ```
x1 <- 1:10; x2 <- 10:20
```

```
a7 <- cbind(x1, x2); a8 <- cbind(x1, x2)
```

```
a12 <- cbind(a2, a4)
```

```
a9 <- matrix(rnorm(30), nrow = 5)
```

```
cbind(a4, a6) # no funciona
```

```
rbind(a4, a6)
```

Matrices *rbind*, *cbind* y otros

- Para combinar vectores o arrays para obtener arrays, usamos *rbind*, *cbind*.

- ```
x1 <- 1:10; x2 <- 10:20
```

```
a7 <- cbind(x1, x2); a8 <- cbind(x1, x2)
```

```
a12 <- cbind(a2, a4)
```

```
a9 <- matrix(rnorm(30), nrow = 5)
```

```
cbind(a4, a6) # no funciona
```

```
rbind(a4, a6)
```

- Las matrices tiene atributos. En particular:

```
attributes(a4)
```

```
colnames(a4) <- paste("v", 1:4, sep = "")
```

```
rownames(a4) <- paste("id", 1:5, sep = ".")
```

```
a4[, c("v1", "v3")]
```

```
attributes(a4)
```

# ***data.frames***

---

- ¿Y si son de distinto tipo?

```
x3 <- letters[1:10]
a9 <- cbind(x1, x2, x3)
```

# ***data.frames***

---

- ¿Y si son de distinto tipo?

```
x3 <- letters[1:10]
a9 <- cbind(x1, x2, x3)
```

- ¡¡¡Pero yo no quería eso!!!. Quizás quería un **data.frame**  
a10 <- data.frame(x1, x2, x3)

# ***data.frames***

---

- ¿Y si son de distinto tipo?

```
x3 <- letters[1:10]
a9 <- cbind(x1, x2, x3)
```

- ¡¡¡Pero yo no quería eso!!!. Quizás quería un **data.frame**  
a10 <- data.frame(x1, x2, x3)

- El indexado y subsetting de data frames como el de arrays y matrices. (Por supuesto, no hallaremos el determinante de un data frame que tenga factores) Pero si podemos hacer:

```
library(mva) #aquí están func. estad. multiv.
prcomp(a10[, c(1,2)])
prcomp(a10[, c("x1", "x2")])
prcomp(a10[, -3])
```

# ***data.frames***

---

- ¿Y si son de distinto tipo?

```
x3 <- letters[1:10]
a9 <- cbind(x1, x2, x3)
```

- ¡¡¡Pero yo no quería eso!!!. Quizás quería un **data.frame**  
a10 <- data.frame(x1, x2, x3)

- El indexado y subsetting de data frames como el de arrays y matrices. (Por supuesto, no hallaremos el determinante de un data frame que tenga factores) Pero si podemos hacer:

```
library(mva) #aquí están func. estad. multiv.
prcomp(a10[, c(1,2)])
prcomp(a10[, c("x1", "x2")])
prcomp(a10[, -3])
```

- Además, al hacer data.frame, los "character vectors" son convertidos a factors (lo que es una ventaja).

- 
- Podemos convertir matrices a data frames con `as.data.frame()`.

- Podemos convertir matrices a data frames con `as.data.frame()`.
- Por estas razones, las data frames suelen ser objetos más útiles que las arrays. Permiten lo mismo que las arrays, pero se puede guardar/acceder a otra información.

- Podemos convertir matrices a data frames con `as.data.frame()`.
- Por estas razones, las data frames suelen ser objetos más útiles que las arrays. Permiten lo mismo que las arrays, pero se puede guardar/acceder a otra información.
- Las data.frames también tienen `rownames`, `colnames`.  
`attributes(a10) # pero nosotros no habíamos  
# fijado los row.names`

- Podemos convertir matrices a data frames con `as.data.frame()`.
- Por estas razones, las data frames suelen ser objetos más útiles que las arrays. Permiten lo mismo que las arrays, pero se puede guardar/acceder a otra información.
- Las data.frames también tienen `rownames`, `colnames`.  
`attributes(a10) # pero nosotros no habíamos  
# fijado los row.names`
- También `dimnames(a10)`.

- La estructura de las data.frames es *sujetos (casos) en fila, variables en columna*. Esta es (por buenas razones) la organización de los datos usada en estadística. Pero esta estructura es *la traspuesta* de la organización habitual de los *datos de microarrays*: los datos de microarrays generalmente tiene los sujetos (o casos o tumores) en columnas y las variables (genes) en filas.

- La estructura de las data.frames es *sujetos (casos) en fila, variables en columna*. Esta es (por buenas razones) la organización de los datos usada en estadística. Pero esta estructura es *la traspuesta* de la organización habitual de los *datos de microarrays*: los datos de microarrays generalmente tiene los sujetos (o casos o tumores) en columnas y las variables (genes) en filas.
- Esto no representa un problema. Podemos trasponer los datos, o guardar otros datos fenotípicos en otras data.frames. Veremos ejemplos.

# *data.frames: \$ y attach*

- Usar \$ facilita el acceso y creación de nuevas columnas.

```
> set.seed(1) # fijar la semilla
> ## del random number generator
> d1 <- data.frame(g1 = runif(10), g2 = rnorm(10))
> d1$edad <- c(rep(20, 5), rep(40, 5))
>
> set.seed(1)
> d2 <- cbind(g1 = runif(10), g2 = rnorm(10))
> d2[, 3] <- c(rep(20, 5), rep(40, 5)) # error
> d2 <- cbind(d2, edad = c(rep(20, 5), rep(40, 5)))
> #OK
```

- Si usamos mucho los datos de un `data.frame`, podemos acceder a ellos directamente:

```
> attach(d1)
> g1
> edad
> plot(g1, g2) # en vez de plot(d1$g1, d1$g2)
```

# *Precauciones con attach( )*

---

- Con `attach( )` ponemos el data frame nombrado en el **search path** (en la posición 2).

# *Precauciones con attach( )*

---

- Con `attach( )` ponemos el data frame nombrado en el **search path** (en la posición 2).
- Ciento cuidado, por masking.

```
> edad <- 25
> edad # usando la que está antes
```

# *Precauciones con attach( )*

---

- Con `attach( )` ponemos el data frame nombrado en el **search path** (en la posición 2).
- Ciento cuidado, por masking.

```
> edad <- 25
> edad # usando la que está antes
```
- `search() #el search path`

# *Precauciones con attach( )*

- Con `attach()` ponemos el data frame nombrado en el **search path** (en la posición 2).
- Ciento cuidado, por masking.

```
> edad <- 25
> edad # usando la que está antes
```

- `search()` #el search path
- Además, la actualización no es dinámica:

```
> d1$g3 <- "nueva columna"
> d1 # aquí está
> g3 # pero aquí no
> detach() # detach(d1) aquí es equivalente
> attach(d1)
> g3
> edad # y esto qué?
```

# *Precauciones con attach( )*

- Con `attach()` ponemos el data frame nombrado en el **search path** (en la posición 2).
- Ciento cuidado, por masking.

```
> edad <- 25
> edad # usando la que está antes
```

- `search()` #el search path
- Además, la actualización no es dinámica:

```
> d1$g3 <- "nueva columna"
> d1 # aquí está
> g3 # pero aquí no
> detach() # detach(d1) aquí es equivalente
> attach(d1)
> g3
> edad # y esto qué?
```

---

En "entornos confusos" (ej., un análisis que se prolonga 2 semanas) mejor evitar `attach()` y acceder siempre a las variables usando su localización explícita y completa.

# *subset*

---

- ```
d3 <- data.frame(g1 = runif(10), g2 = rnorm(10),  
  id1 = c(rep("a", 3), rep("b", 2), rep("c", 2),  
  rep("d", 3))))
```

subset

- ```
d3 <- data.frame(g1 = runif(10), g2 = rnorm(10),
 id1 = c(rep("a", 3), rep("b", 2), rep("c", 2),
 rep("d", 3))))
```
- Los que no son "a" ni "b"  

```
d3[!(d3$id1 %in% c("a", "b")) ,]
```

# *subset*

---

- ```
d3 <- data.frame(g1 = runif(10), g2 = rnorm(10),  
                   id1 = c(rep("a", 3), rep("b", 2), rep("c", 2),  
                   rep("d", 3))))
```
- Los que no son "a" ni "b"

```
d3[ !(d3$id1 %in% c("a", "b")) , ]
```
- O usar subset

```
subset(d3, !(id1 %in% c("a", "b")) )
```

subset

- ```
d3 <- data.frame(g1 = runif(10), g2 = rnorm(10),
 id1 = c(rep("a", 3), rep("b", 2), rep("c", 2),
 rep("d", 3))))
```
- Los que no son "a" ni "b"  

```
d3[!(d3$id1 %in% c("a", "b")) ,]
```
- O usar subset  

```
subset(d3, !(id1 %in% c("a", "b")))
```
- Por supuesto, podemos usar reglas más complejas (ej., que sea igual a una variable, distinto en otra, y no missing en una tercera).

# *subset*

---

- ```
d3 <- data.frame(g1 = runif(10), g2 = rnorm(10),  
                   id1 = c(rep("a", 3), rep("b", 2), rep("c", 2),  
                   rep("d", 3))))
```
- Los que no son "a" ni "b"

```
d3[ !(d3$id1 %in% c("a", "b")) , ]
```
- O usar subset

```
subset(d3, !(id1 %in% c("a", "b")))
```
- Por supuesto, podemos usar reglas más complejas (ej., que sea igual a una variable, distinto en otra, y no missing en una tercera).
- subset también permite la selección de variables (columnas); el argumento "select".

aggregate, etc

- Escenario: un data frame (datos); varias réplicas del mismo clon (cniOID); queremos "colapsar" por clon. Si es una variable numérica, hallar la mediana; si categórica, sólo debería haber un valor: devolver ese (si hay más valores, señalarlo.)

```
colapsar <- function(x) {  
  +   if(is.numeric(x))  
  +     return(median(x, na.rm = TRUE))  
  +   else {  
  +     tmp <- unique(x)  
  +     if(length(tmp) > 1) return("¡varios!")  
  +     else return(as.character(tmp[1]))  
  +   }  
}
```

- ```
d.collapse.by.cnoid <- aggregate(datos,
 list(id = datos[, "cnioID"]),
 FUN = colapsar)

d.collapse.by.cnoid <- d.collapse.by.cnoid[, -1]

Asegurarse que se ha echo lo que queríamos!!!
```

- ```
d.collapse.by.cnoid <- aggregate(datos,
  list(id = datos[, "cnioID"]),
  FUN = colapsar)

d.collapse.by.cnoid <- d.collapse.by.cnoid[, -1]
# Asegurarse que se ha echo lo que queríamos!!!
```
- Otras funciones para objetivos similares: ver `?reshape`, `?merge`

La familia apply

- ```
ax <- matrix(rnorm(20, ncol = 5)
medias.por.fila <- apply(ax, 1, mean)
por.si.na <- apply(ax, 1, mean, na.rm = TRUE)
mi.f1 <- function(x) { return(2*x - 25)}
mi.f1.por.fila <- apply(ax, 1, mi.f1)
mas.simple <- apply(ax, 1, function(x)
+ {return(2*x -25)})
media.por.columna <- apply(ax, 2, mean)
sample.rows <- apply(ax, 1, sample)
dos.cosas <- function(y)
+ {return(c(mean(y), var(y)))}
apply(ax, 1, dos.cosas)
t(apply(ax, 1, dos.cosas))
```

- 
- Operar con apply generalmente mucho más eficiente que loops (además de más claro, más fácil, etc, etc).

- Operar con `apply` generalmente mucho más eficiente que loops (además de más claro, más fácil, etc, etc).
- `sapply`, `lapply` son como `apply` pero no hay que especificar el "margin"; `sapply` intenta simplificar el resultado a un vector o a una matriz (la "s" es de "simplify"), pero `lapply` siempre devuelve una lista. Ambas pueden aplicarse a vectores, listas, arrays.

# *Un ejemplo inútil de apply*

- Una forma tonta de generar datos de distribuciones normales con una media y desviación típicas dadas:

```
> parameters <- cbind(mean = -5:5, sd = 2:12)
> data <- t(apply(parameters, 1, function(x)
+ rnorm(10000, x[1], x[2])))
> apply(data, 1, mean); apply(data, 1, sd)
```

# *Un ejemplo inútil de apply*

- Una forma tonta de generar datos de distribuciones normales con una media y desviación típicas dadas:

```
> parameters <- cbind(mean = -5:5, sd = 2:12)
> data <- t(apply(parameters, 1, function(x)
+ rnorm(10000, x[1], x[2])))
> apply(data, 1, mean); apply(data, 1, sd)
```

- Este es un ejemplo tonto, por supuesto. Es más sencillo (y matemáticamente equivalente) hacer:

```
> z.data <- matrix(rnorm(10000 * 11), nrow = 11)
> data2 <- (z.data * parameters[, 2]) +
+ parameters[, 1]
>
> apply(data2, 1, mean); apply(data2, 1, sd)
```

# *tapply, table*

---

- > # see MASS, p. 38  
> library(MASS)  
> data(quine) # absentismo escolar  
> attach(quine)  
> tapply(Days, Age, mean)  
> tapply(Days, list(Sex, Age), mean)  
> tapply(Days, list(Sex, Age),  
+ function(y) sqrt( (var(y)/length(y)) ))

# *tapply, table*

---

- > # see MASS, p. 38  
> library(MASS)  
> data(quine) # absentismo escolar  
> attach(quine)  
> tapply(Days, Age, mean)  
> tapply(Days, list(Sex, Age), mean)  
> tapply(Days, list(Sex, Age),  
+ function(y) sqrt( (var(y)/length(y)) ))
- Una tabla  
> table(Sex, Age)

- Algunas funciones directamente hacen un "apply".

```
> m1 <- matrix(1:20, ncol = 5)
> d1 <- as.data.frame(m1)
> mean(x1); mean(d1); sd(x1); sd(d1); median(m1);
median(d1)
```

- Algunas funciones directamente hacen un "apply".

```
> m1 <- matrix(1:20, ncol = 5)
> d1 <- as.data.frame(m1)
> mean(x1); mean(d1); sd(x1); sd(d1); median(m1);
median(d1)
```

- `apply`, `sapply`, `lapply` y `tapply` son funciones *muy útiles* que contribuyen a hacer el código más legible, fácil de entender, y facilitan posteriores modificaciones y aplicaciones.

- Algunas funciones directamente hacen un "apply".

```
> m1 <- matrix(1:20, ncol = 5)
> d1 <- as.data.frame(m1)
> mean(x1); mean(d1); sd(x1); sd(d1); median(m1);
median(d1)
```

- `apply`, `sapply`, `lapply` y `tapply` son funciones *muy útiles* que contribuyen a hacer el código más legible, fácil de entender, y facilitan posteriores modificaciones y aplicaciones.
- Cada vez que vayamos a usar un "loop" explícito, intentemos substituirlo por algún miembro de *la ilustre familia apply*.

# *Miscel. (cov, cor, outer)*

---

- Dos funciones que se aplican directamente sobre matrices:

```
> cov(m1)
> cor(m1) #demasiados dígitos ...
> round(cor(m1), 3)
> # cor.test hace otra cosa
> cor.test(m1[, 1], m1[, 2])
> cor(apply(m1, 2, rank)) # corr. rangos (Spearman)
```

# *Miscel. (cov, cor, outer)*

- Dos funciones que se aplican directamente sobre matrices:

```
> cov(m1)
> cor(m1) #demasiados dígitos ...
> round(cor(m1), 3)
> # cor.test hace otra cosa
> cor.test(m1[, 1], m1[, 2])
> cor(apply(m1, 2, rank)) # corr. rangos (Spearman)
```

- tabla.multiplicación <- outer(11:15, 11:20, "\*")

# **Miscel. (*cov*, *cor*, *outer*)**

- Dos funciones que se aplican directamente sobre matrices:

```
> cov(m1)
> cor(m1) #demasiados dígitos ...
> round(cor(m1), 3)
> # cor.test hace otra cosa
> cor.test(m1[, 1], m1[, 2])
> cor(apply(m1, 2, rank)) # corr. rangos (Spearman)
```

- tabla.multiplicación <- outer(11:15, 11:20, "\*")
- A outer se pueden pasar funciones mucho más complejas; por ej., outer se usa en el ejemplo de persp.  
?persp # y ejecutemos el primero ejemplo  
par(mfrow(1,2)); ?image # y otro ejemplo

# **Miscl. (*cov*, *cor*, *outer*)**

- Dos funciones que se aplican directamente sobre matrices:

```
> cov(m1)
> cor(m1) #demasiados dígitos ...
> round(cor(m1), 3)
> # cor.test hace otra cosa
> cor.test(m1[, 1], m1[, 2])
> cor(apply(m1, 2, rank)) # corr. rangos (Spearman)
```
- `tabla.multiplicación <- outer(11:15, 11:20, "*")`
- A `outer` se pueden pasar funciones mucho más complejas; por ej., `outer` se usa en el ejemplo de `persp`.  
`?persp # y ejecutemos el primero ejemplo
par(mfrow(1,2)); ?image # y otro ejemplo`
- `outer` requiere funciones vectorizadas (`apply` no). Ver FAQ, 7.19 para más detalles.

## *Interludio: listas*

---

- Las data.frames son, en realidad, tipos especiales de listas.

## *Interludio: listas*

---

- Las data.frames son, en realidad, tipos especiales de listas.
- Las listas son contenedores sin estructura determinada.

## *Interludio: listas*

---

- Las data.frames son, en realidad, tipos especiales de listas.
- Las listas son contenedores sin estructura determinada.
- Por tanto muy flexibles, pero sin estructura.

## *Interludio: listas*

---

- Las data.frames son, en realidad, tipos especiales de listas.
- Las listas son contenedores sin estructura determinada.
- Por tanto muy flexibles, pero sin estructura.
- Muchas funciones devuelven listas: devuelven un conjunto de resultados de distinta longitud y distinto tipo.

● > d3 <- data.frame(g1 = runif(10), g2 = rnorm(10),  
+ id1 = c(rep("a", 3), rep("b", 2),  
+ rep("c", 2), rep("d", 3))))  
> my.fun <- function(x) {  
+ las.medias <- mean(x[, -3])  
+ las.vars <- var(x[, -3])  
+ max.total <- max(x[, -3])  
+ tabla.clases <- table(x[, 3])  
+ return(list(row.means = las.medias,  
+ row.vars = las.vars, maximum = max.total,  
+ factor.classes = tabla.clases))  
}  
> my.fun(d3)



```
> una.lista <- my.fun(d3); una.lista
> attributes(una.lista); names(una.lista)
> length(una.lista)
> una.lista[[4]]
> una.lista[4] # por qué sale el nombre?
> una.lista$factor.classes
> una.lista[[3]] <- list(NULL); una.lista
> una.lista[[3]] <- NULL
> una.lista # hemos eliminado el "slot" maximum
> unlist(una.lista)
> otra.lista <- list(cucu = 25, una.lista)
> unlist(otra.lista)
> unlist(otra.lista, drop = FALSE)
> una.lista <- c(una.lista, otro.elemento = "una
frase")
```

# *Resumen (y apología de R)*

---

- Una vez que los datos están en R, su manipulación es *muy* flexible.

# *Resumen (y apología de R)*

---

- Una vez que los datos están en R, su manipulación es *muy* flexible.
- Podemos seleccionar variables, casos, subsecciones de datos, etc, de acuerdo con criterios arbitrarios (que usan, además, condiciones que pueden implicar a un número arbitrario de variables y casos).

# *Resumen (y apología de R)*

---

- Una vez que los datos están en R, su manipulación es *muy* flexible.
- Podemos seleccionar variables, casos, subsecciones de datos, etc, de acuerdo con criterios arbitrarios (que usan, además, condiciones que pueden implicar a un número arbitrario de variables y casos).
- Los data frames y las matrices pueden separarse, juntarse, cambiarse de forma (reshape), etc.

# *Resumen (y apología de R)*

---

- Una vez que los datos están en R, su manipulación es *muy* flexible.
- Podemos seleccionar variables, casos, subsecciones de datos, etc, de acuerdo con criterios arbitrarios (que usan, además, condiciones que pueden implicar a un número arbitrario de variables y casos).
- Los data frames y las matrices pueden separarse, juntarse, cambiarse de forma (reshape), etc.
- El indexado y selección de casos pueden usar números, factores, cadenas de caracteres, etc.

# *Resumen (y apología de R)*

---

- Una vez que los datos están en R, su manipulación es *muy* flexible.
- Podemos seleccionar variables, casos, subsecciones de datos, etc, de acuerdo con criterios arbitrarios (que usan, además, condiciones que pueden implicar a un número arbitrario de variables y casos).
- Los data frames y las matrices pueden separarse, juntarse, cambiarse de forma (reshape), etc.
- El indexado y selección de casos pueden usar números, factores, cadenas de caracteres, etc.
- Podemos preparar código que repita las mismas operaciones con datos semejantes (i.e., podemos automatizar el proceso con sencillez).

- Podemos verificar "on-the-fly" que estas transformaciones hacen lo que queremos que hagan (mirando selectivamente los resultados, o "emulando" el proceso en unos datos artificiales más pequeños).

- Podemos verificar "on-the-fly" que estas transformaciones hacen lo que queremos que hagan (mirando selectivamente los resultados, o "emulando" el proceso en unos datos artificiales más pequeños).
- Por tanto, una vez que los datos están en R, no hay muchas razones para exportarlos y hacer la selección y manipulación con otros lenguajes (ej., Python, Perl) para luego volver a leerlos en R.

# *Importando y exportando datos*

---

- `read.table`

# *Importando y exportando datos*

---

- `read.table`
- `write.table`

# *Importando y exportando datos*

---

- `read.table`
- `write.table`
- `save.image`

# *Importando y exportando datos*

---

- `read.table`
- `write.table`
- `save.image`
- `data`

# *Importando datos*

---

- R es un tanto inflexible en la importación de datos. En ocasiones, puede ser necesario un preprocesado de los datos con otro programa (Python, Perl).

# *Importando datos*

---

- R es un tanto inflexible en la importación de datos. En ocasiones, puede ser necesario un preprocesado de los datos con otro programa (Python, Perl).
- Hay funciones para importar datos en formato binario de SAS, SPSS, Minitab, Stata, S3 (ver package `foreign` (<http://cran.r-project.org/src/contrib/PACKAGES.html#foreign>)). También de bases de datos. Ver *R data import/export*.

# *Importando datos*

---

- R es un tanto inflexible en la importación de datos. En ocasiones, puede ser necesario un preprocesado de los datos con otro programa (Python, Perl).
- Hay funciones para importar datos en formato binario de SAS, SPSS, Minitab, Stata, S3 (ver package `foreign` (<http://cran.r-project.org/src/contrib/PACKAGES.html#foreign>)). También de bases de datos. Ver *R data import/export*.
- No hay, por el momento, formas sencillas de importar directamente datos en formato binario Excel.

# *Importando datos*

---

- R es un tanto inflexible en la importación de datos. En ocasiones, puede ser necesario un preprocesado de los datos con otro programa (Python, Perl).
- Hay funciones para importar datos en formato binario de SAS, SPSS, Minitab, Stata, S3 (ver package `foreign` (<http://cran.r-project.org/src/contrib/PACKAGES.html#foreign>)). También de bases de datos. Ver *R data import/export*.
- No hay, por el momento, formas sencillas de importar directamente datos en formato binario Excel.
- Por tanto, datos en Excel habrán de ser exportados o salvados desde Excel como texto.

# *Importando datos*

---

- R es un tanto inflexible en la importación de datos. En ocasiones, puede ser necesario un preprocesado de los datos con otro programa (Python, Perl).
- Hay funciones para importar datos en formato binario de SAS, SPSS, Minitab, Stata, S3 (ver package `foreign` (<http://cran.r-project.org/src/contrib/PACKAGES.html#foreign>)). También de bases de datos. Ver *R data import/export*.
- No hay, por el momento, formas sencillas de importar directamente datos en formato binario Excel.
- Por tanto, datos en Excel habrán de ser exportados o salvados desde Excel como texto.
- (Para usuarios de GNU/Linux: con ficheros de Excel de tamaño moderado, Gnumeric generalmente es más rápido que OpenOffice. Con ficheros muy grandes, Gnumeric parece atragantarse.)

# *Importando datos: `read.table`*

---

- Queremos leer un fichero simple, separado por tabuladores.

# *Importando datos: read.table*

---

- Queremos leer un fichero simple, separado por tabuladores.
- ```
my.data.frame <- read.table("mi.fichero",
+     header = TRUE, sep = "\t",
+     comment.char = "")
```

Importando datos: read.table

- Queremos leer un fichero simple, separado por tabuladores.
- ```
my.data.frame <- read.table("mi.fichero",
+ header = TRUE, sep = "\t",
+ comment.char = "")
```
- Si el carácter decimal no es un punto sino, por ej., una coma, usar: `dec = ",."`.

# *Importando datos: read.table*

---

- Queremos leer un fichero simple, separado por tabuladores.
- ```
my.data.frame <- read.table("mi.fichero",
+     header = TRUE, sep = "\t",
+     comment.char = "")
```
- Si el carácter decimal no es un punto sino, por ej., una coma, usar: `dec = ",."`.
- Se pueden saltar líneas (`skip`) o leer un número fijo de líneas (`nrows`).

Importando datos: read.table

- Queremos leer un fichero simple, separado por tabuladores.
- ```
my.data.frame <- read.table("mi.fichero",
+ header = TRUE, sep = "\t",
+ comment.char = "")
```
- Si el carácter decimal no es un punto sino, por ej., una coma, usar: `dec = ",."`.
- Se pueden saltar líneas (`skip`) o leer un número fijo de líneas (`nrows`).
- Otro separador habitual de columnas es `sep = " "`.

# **Importando datos: `read.table`**

---

- Queremos leer un fichero simple, separado por tabuladores.
- ```
my.data.frame <- read.table("mi.fichero",
+     header = TRUE, sep = "\t",
+     comment.char = "")
```
- Si el carácter decimal no es un punto sino, por ej., una coma, usar: `dec = ",."`.
- Se pueden saltar líneas (`skip`) o leer un número fijo de líneas (`nrows`).
- Otro separador habitual de columnas es `sep = " "`.
- Hay funciones especializadas para otros ficheros (ej., `read.csv`) pero son casos específicos de `read.table`.

-
- Ojo con `comment.char`. El "default" es "#" lo que puede causar problemas si ese carácter se usa en nombres (como en muchos ficheros de arrays).

- Ojo con `comment.char`. El "default" es "#" lo que puede causar problemas si ese carácter se usa en nombres (como en muchos ficheros de arrays).
- Podemos especificar el código para "missing data" (por defecto usa NA y el que no haya valor —columna vacía).

- Ojo con `comment.char`. El "default" es "#" lo que puede causar problemas si ese carácter se usa en nombres (como en muchos ficheros de arrays).
- Podemos especificar el código para "missing data" (por defecto usa NA y el que no haya valor —columna vacía).
- Muchos errores de lectura relacionados con distinto número de columnas. Usar `count.fields` para examinar donde está el problema.

- Ojo con `comment.char`. El "default" es "#" lo que puede causar problemas si ese carácter se usa en nombres (como en muchos ficheros de arrays).
- Podemos especificar el código para "missing data" (por defecto usa `NA` y el que no haya valor —columna vacía).
- Muchos errores de lectura relacionados con distinto número de columnas. Usar `count.fields` para examinar donde está el problema.
- `scan` es una función más general, pero de uso más complicado; sólo necesaria en ocasiones especiales o cuando escasos de memoria.

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.
- Ojo con las últimas columnas y missing data (Excel elimina los "trailing tabs"). Dos formas de minimizar problemas:

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.
- Ojo con las últimas columnas y missing data (Excel elimina los "trailing tabs"). Dos formas de minimizar problemas:
 - Usar "NA" para missing.

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.
- Ojo con las últimas columnas y missing data (Excel elimina los "trailing tabs"). Dos formas de minimizar problemas:
 - Usar "NA" para missing.
 - Poner una última columna con datos arbitrarios (ej., una columna llena de 2s).

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.
- Ojo con las últimas columnas y missing data (Excel elimina los "trailing tabs"). Dos formas de minimizar problemas:
 - Usar "NA" para missing.
 - Poner una última columna con datos arbitrarios (ej., una columna llena de 2s).
- Cuidado también con líneas extra al final del fichero.

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.
- Ojo con las últimas columnas y missing data (Excel elimina los "trailing tabs"). Dos formas de minimizar problemas:
 - Usar "NA" para missing.
 - Poner una última columna con datos arbitrarios (ej., una columna llena de 2s).
- Cuidado también con líneas extra al final del fichero.
- Salvamos como texto (sólo salvamos una de las hojas).

Importando de Excel

- Lo mejor es exportar desde Excel como fichero de texto separado por tabuladores.
- Ojo con las últimas columnas y missing data (Excel elimina los "trailing tabs"). Dos formas de minimizar problemas:
 - Usar "NA" para missing.
 - Poner una última columna con datos arbitrarios (ej., una columna llena de 2s).
- Cuidado también con líneas extra al final del fichero.
- Salvamos como texto (sólo salvamos una de las hojas).
- Importamos en R con `read.table`.

Exportando datos

- Lo más sencillo es exportar una tabla.

Exportando datos

- Lo más sencillo es exportar una tabla.
- `write.table(my.data.frame, file = "mi.output.txt",
+ sep = "", row.names = FALSE,
+ col.names = TRUE)`

Exportando datos

- Lo más sencillo es exportar una tabla.
- `write.table(my.data.frame, file = "mi.output.txt",
+ sep = "", row.names = FALSE,
+ col.names = TRUE)`
- ¿Queremos `row.names` y `col.names` como TRUE o como FALSE?

Guardando datos

- Guardar datos, funciones, etc, para ser usados en otras sesiones de R.

Guardando datos

- Guardar datos, funciones, etc, para ser usados en otras sesiones de R.
- Datos pueden compartirse entre sesiones de R en distintos sistemas operativos.

Guardando datos

- Guardar datos, funciones, etc, para ser usados en otras sesiones de R.
- Datos pueden compartirse entre sesiones de R en distintos sistemas operativos.
- ```
> a1 <- rnorm(10)
> a2 <- 1:10
> a3 <- letters[10:20]
> save("unos.datos.guardados.RData", a1, a2, a3)
```

# **Guardando datos**

---

- Guardar datos, funciones, etc, para ser usados en otras sesiones de R.
- Datos pueden compartirse entre sesiones de R en distintos sistemas operativos.
- ```
> a1 <- rnorm(10)
> a2 <- 1:10
> a3 <- letters[10:20]
> save("unos.datos.guardados.RData", a1, a2, a3)
```
- Los leemos con

```
> load("unos.datos.salvados.RData")
```

Guardando datos

- Guardar datos, funciones, etc, para ser usados en otras sesiones de R.
- Datos pueden compartirse entre sesiones de R en distintos sistemas operativos.
- ```
> a1 <- rnorm(10)
> a2 <- 1:10
> a3 <- letters[10:20]
> save("unos.datos.guardados.RData", a1, a2, a3)
```
- Los leemos con  

```
> load("unos.datos.salvados.RData")
```
- Podemos salvar todos los objetos con  

```
> save.image() # salvado como ".RData"
> save.image(file = "un.nombre.RData")
```

# **Guardando datos**

---

- Guardar datos, funciones, etc, para ser usados en otras sesiones de R.
- Datos pueden compartirse entre sesiones de R en distintos sistemas operativos.
- ```
> a1 <- rnorm(10)
> a2 <- 1:10
> a3 <- letters[10:20]
> save("unos.datos.guardados.RData", a1, a2, a3)
```
- Los leemos con

```
> load("unos.datos.salvados.RData")
```
- Podemos salvar todos los objetos con

```
> save.image() # salvado como ".RData"
> save.image(file = "un.nombre.RData")
```
- El fichero ".RData" es cargado al iniciarse R.

Cargando built-in data

- R, y muchos paquetes, incorporan ficheros con datos.

Cargando built-in data

- R, y muchos paquetes, incorporan ficheros con datos.
- Se cargan con `load(nombre.fichero)`.

Cargando built-in data

- R, y muchos paquetes, incorporan ficheros con datos.
- Se cargan con `load(nombre.fichero)`.

- `rm(list = ls()) # borrar todo del workspace`

`## para ver efectos de siguientes comandos`

```
library(multtest)
```

```
data(golub)
```

```
# o data(golub, package = "multtest") si no
```

```
# hemos attached la librería
```

```
search()
```

```
?golub
```

Gráficos

- plot

Gráficos

- plot
- Identificación datos

Gráficos

- plot
- Identificación datos
- Datos multivariantes

Gráficos

- plot
- Identificación datos
- Datos multivariantes
- Jittering

Gráficos

- plot
- Identificación datos
- Datos multivariantes
- Jittering
- Adición rectas regresión

Gráficos

- plot
- Identificación datos
- Datos multivariantes
- Jittering
- Adición rectas regresión
- Boxplots

Gráficos

- plot
- Identificación datos
- Datos multivariantes
- Jittering
- Adición rectas regresión
- Boxplots
- Otros gráficos

Gráficos

- plot
- Identificación datos
- Datos multivariantes
- Jittering
- Adición rectas regresión
- Boxplots
- Otros gráficos
- Guardando gráficos

Introducción a los gráficos

- R incluye *muchas y variadas* funciones para hacer gráficos.

Introducción a los gráficos

- R incluye *muchas y variadas* funciones para hacer gráficos.
- El sistema permite desde simples plots a figuras de calidad para incluir en artículos y libros.

Introducción a los gráficos

- R incluye *muchas y variadas* funciones para hacer gráficos.
- El sistema permite desde simples plots a figuras de calidad para incluir en artículos y libros.
- Sólo examinaremos la superficie. Más detalles en el capítulo 4 de *Modern applied statistics with S*; los capítulos 3 y 7 de *An R and S-PLUS companion to applied regression*; el capítulo 4 de *R para principiantes*; el capítulo 12 de *An introduction to R*.

Introducción a los gráficos

- R incluye *muchas y variadas* funciones para hacer gráficos.
- El sistema permite desde simples plots a figuras de calidad para incluir en artículos y libros.
- Sólo examinaremos la superficie. Más detalles en el capítulo 4 de *Modern applied statistics with S*; los capítulos 3 y 7 de *An R and S-PLUS companion to applied regression*; el capítulo 4 de *R para principiantes*; el capítulo 12 de *An introduction to R*.
- También `demo(graphics)`.

plot()

- `plot()` función gráfica básica.

plot()

- `plot()` función gráfica básica.
- ```
Modificado de "Introductory statistics with R"
x <- runif(50, 0, 4); y <- runif(50, 0, 4)
plot(x, y, main = "Título principal",
 sub = "subtítulo", xlab = "x label",
 ylab = "y label", xlim = c(-1, 5),
 ylim = c(1, 5))
abline(h = 0, lty = 1); abline(v = 0, lty = 2)
text(1, 4, "Algo de texto")
mtext("mtext", side = 1)
mtext("mtext en side 2", side = 2, line = -3,
 cex = 2)
```

## ● Variaciones de plot **plot(x)**

```
z <- cbind(x,y)
plot(z)
plot(y ~ x)
plot(log(y + 1) ~ x) # transformacion de y
plot(x, y, type = "p")
plot(x, y, type = "l")
plot(x, y, type = "b")
plot(c(1,5), c(1,5))
legend(1, 4, c("uno", "dos", "tres"), lty = 1:3,
+ col = c("red", "blue", "green"),
+ pch = 15:17, cex = 2)
```

- Con `text` podemos representar caracteres de texto directamente:

```
sexo <- c(rep("v", 20), rep("m", 30))
plot(x, y, type = "n")
text(x, y, labels = sexo)
```

## *plot: pch, col, lty*

---

- `plot(x, y, type = "n")  
points(x, y, pch = 3, col = "red")`

## ***plot: pch, col, lty***

---

- `plot(x, y, type = "n")`

```
points(x, y, pch = 3, col = "red")
```

- Tipos de puntos:

```
plot(c(1, 10), c(1, 3), type = "n", axes = FALSE,
+ xlab = "", ylab="")
```

```
points(1:10, rep(1, 10), pch = 1:10, cex = 2,
+ col = "blue")
```

```
points(1:10, rep(2, 10), pch = 11:20, cex = 2,
+ col = "red")
```

```
points(1:10, rep(3, 10), pch = 21:30, cex = 2,
+ col = "blue", bg = "yellow")
```

- Tipos de líneas:

```
plot(c(0, 10), c(0, 10), type = "n", xlab = "",
+ ylab = "")

for(i in 1:10)
+ abline(0, i/5, lty = i, lwd = 2)
```

- Tipos de líneas:

```
plot(c(0, 10), c(0, 10), type = "n", xlab = "",
+ ylab = "")

for(i in 1:10)
+ abline(0, i/5, lty = i, lwd = 2)
```

- `lty` permite especificaciones más complejas (longitud de los segmentos que son alternativamente dibujados y no dibujados).

- Tipos de líneas:

```
plot(c(0, 10), c(0, 10), type = "n", xlab = "",
+ ylab = "")

for(i in 1:10)
+ abline(0, i/5, lty = i, lwd = 2)
```

- `lty` permite especificaciones más complejas (longitud de los segmentos que son alternativamente dibujados y no dibujados).
- `par` controla muchos parámetros gráficos. Por ejemplo, `cex` puede referirse a los "labels" (`cex.lab`), otro, `cex.axis`, a la anotación de los ejes, etc.

- Tipos de líneas:

```
plot(c(0, 10), c(0, 10), type = "n", xlab = "",
+ ylab = "")

for(i in 1:10)
+ abline(0, i/5, lty = i, lwd = 2)
```

- `lty` permite especificaciones más complejas (longitud de los segmentos que son alternativamente dibujados y no dibujados).
- `par` controla muchos parámetros gráficos. Por ejemplo, `cex` puede referirse a los "labels" (`cex.lab`), otro, `cex.axis`, a la anotación de los ejes, etc.
- Hay muchos más colores. Ver `palette`, `colors`.

# *Identificación interactiva de datos*

---



```
> x <- 1:10
> y <- sample(1:10)
> nombres <- paste("punto", x, ".", y, sep = "")
>
> plot(x, y)
> identify(x, y, labels = nombres)
```

# *Identificación interactiva de datos*

---

- > x <- 1:10  
> y <- sample(1:10)  
> nombres <- paste("punto", x, ".", y, sep = "")  
>  
> plot(x, y)  
> identify(x, y, labels = nombres)
- locator devuelve la posición de los puntos.  
> plot(x, y)  
> locator()  
> text(locator(1), "el marcado", adj = 0)

# Múltiples gráficos por ventana

---

- Podemos mostrar muchos gráficos en el mismo dispositivo gráfico. La función más flexible y sofisticada es `split.screen`, bien explicada en *R para principiantes*, secc. 4.1.2 (p. 30).

# Múltiples gráficos por ventana

---

- Podemos mostrar muchos gráficos en el mismo dispositivo gráfico. La función más flexible y sofisticada es `split.screen`, bien explicada en *R para principiantes*, secc. 4.1.2 (p. 30).
- Mucho más sencillo es `par(mfrow=c(filas, columnas))`

# Múltiples gráficos por ventana

---

- Podemos mostrar muchos gráficos en el mismo dispositivo gráfico. La función más flexible y sofisticada es `split.screen`, bien explicada en *R para principiantes*, secc. 4.1.2 (p. 30).
- Mucho más sencillo es `par(mfrow=c(filas, columnas))`
- ```
par(mfrow = c(2, 2))
plot(rnorm(10))
plot(runif(5), rnorm(5))
plot(runif(10))
plot(rnorm(10), rnorm(10))
```

Datos multivariantes

- Una "pairwise scatterplot matrix":

```
> x <- matrix(rnorm(1000), ncol = 5)
> colnames(x) <- c("a", "id", "edad", "loc",
+      "weight")
> pairs(x)
```

Datos multivariantes

- Una "pairwise scatterplot matrix":

```
> X <- matrix(rnorm(1000), ncol = 5)  
> colnames(X) <- c("a", "id", "edad", "loc",  
+      "weight")  
> pairs(X)
```

- "Conditioning plots" (revelan, entre otros, interacciones):

```
> Y <- as.data.frame(X)  
> Y$sexo <- as.factor(c(rep("Macho", 80),  
+      rep("Hembra", 120)))  
> coplot(weight ~ edad | sexo, data = Y)  
> coplot(weight ~ edad | loc, data = Y)  
> coplot(weight ~ edad | loc + sexo, data = Y)
```

Datos multivariantes

- Una "pairwise scatterplot matrix":

```
> X <- matrix(rnorm(1000), ncol = 5)  
> colnames(X) <- c("a", "id", "edad", "loc",  
+      "weight")  
> pairs(X)
```

- "Conditioning plots" (revelan, entre otros, interacciones):

```
> Y <- as.data.frame(X)  
> Y$sexo <- as.factor(c(rep("Macho", 80),  
+      rep("Hembra", 120)))  
> coplot(weight ~ edad | sexo, data = Y)  
> coplot(weight ~ edad | loc, data = Y)  
> coplot(weight ~ edad | loc + sexo, data = Y)
```

- La librería **lattice** permite lo mismo, y mucho más, que coplot. Ver secc. 4.6 de *R para principiantes*.

Boxplots

- Muy útiles para ver rápidamente las características de una variable, o comparar entre variables.

Boxplots

- Muy útiles para ver rápidamente las características de una variable, o comparar entre variables.
- `attach(Y)`

```
boxplot(weight)
plot(sexo, weight)
detach()
boxplot(weight ~ sexo, data = Y,
+       col = c("red", "blue"))
```

Boxplots

- Muy útiles para ver rápidamente las características de una variable, o comparar entre variables.

- `attach(Y)`

```
boxplot(weight)  
plot(sexo, weight)  
detach()  
boxplot(weight ~ sexo, data = Y,  
+       col = c("red", "blue"))
```

- `boxplot` tiene muchas opciones; se puede modificar el aspecto, mostrarlos horizontalmente, en una matriz de boxplots, etc. Vease la ayuda (`?boxplot`).

Jittering en scatterplots

- Los datos cuantitativos discretos pueden ser difíciles de ver bien:

```
dc1 <- sample(1:5, 500, replace = TRUE)  
dc2 <- dc1 + sample(-2:2, 500, replace = TRUE,  
+      prob = c(1, 2, 3, 2, 1)/9)  
plot(dc1, dc2)  
plot(jitter(dc1), jitter(dc2))
```

Jittering en scatterplots

- Los datos cuantitativos discretos pueden ser difíciles de ver bien:

```
dc1 <- sample(1:5, 500, replace = TRUE)  
dc2 <- dc1 + sample(-2:2, 500, replace = TRUE,  
+      prob = c(1, 2, 3, 2, 1)/9)  
plot(dc1, dc2)  
plot(jitter(dc1), jitter(dc2))
```

- También útil si sólo una de las variables está en pocas categorías.

Adición de rectas de regresión

- Podemos añadir muchos elementos a un gráfico, además de leyendas y líneas rectas:

Adición de rectas de regresión

- Podemos añadir muchos elementos a un gráfico, además de leyendas y líneas rectas:

- ```
> x <- rnorm(50)
> y <- rnorm(50)
> plot(x, y)
> lines(lowess(x, y), lty = 2)
> plot(x, y)
> abline(lm(y ~ x), lty = 3)
```

# *Adición de rectas de regresión*

---

- Podemos añadir muchos elementos a un gráfico, además de leyendas y líneas rectas:

```
> x <- rnorm(50)
> y <- rnorm(50)
> plot(x, y)
> lines(lowess(x, y), lty = 2)
> plot(x, y)
> abline(lm(y ~ x), lty = 3)
```

- Podemos añadir otros elementos con "panel functions" en otras funciones (como pairs, lattice, etc).

- Lo más sencillo es usar `panel.car` y `scatterplot.matrix`, en el paquete "car".

```
> library(car)
> coplot(a ~ edad | loc, panel = panel.car,
+ data = x)
> scatterplot.matrix(x, diagonal = "density")
```

## Otros

---

- Podemos modificar márgenes exteriores de figuras y entre figuras (vease `?par` y búsqunse `oma`, `omi`, `mar`, `mai`; ejemplos en *An introduction to R*, secc. 12.5.3 y 12.5.4.

# Otros

---

- Podemos modificar márgenes exteriores de figuras y entre figuras (vease `?par` y búsqunse `oma`, `omi`, `mar`, `mai`; ejemplos en *An introduction to R*, secc. 12.5.3 y 12.5.4).
- También **gráficos 3D**: `persp`, `image`, `contour`; **histogramas**: `hist`; **gráficos de barras**: `barplot`; **gráficos de comparación de cuantiles**, usados para **comparar la distribución** de dos variables, o la distribución de unos datos frente a un estándard (ej., distribución normal): `qqplot`, `qgnorm` y, en paquete "car", `qq.plot`.

# Otros

---

- Podemos modificar márgenes exteriores de figuras y entre figuras (vease `?par` y búsqunse `oma`, `omi`, `mar`, `mai`; ejemplos en *An introduction to R*, secc. 12.5.3 y 12.5.4).
- También **gráficos 3D**: `persp`, `image`, `contour`; **histogramas**: `hist`; **gráficos de barras**: `barplot`; **gráficos de comparación de cuantiles**, usados para **comparar la distribución** de dos variables, o la distribución de unos datos frente a un estándard (ej., distribución normal): `qqplot`, `qgnorm` y, en paquete "car", `qq.plot`.
- Notación matemática (`plotmath`) y expresiones de texto arbitrariamente complejas.

# Otros

---

- Podemos modificar márgenes exteriores de figuras y entre figuras (vease `?par` y búsqunse `oma`, `omi`, `mar`, `mai`; ejemplos en *An introduction to R*, secc. 12.5.3 y 12.5.4).
- También **gráficos 3D**: `persp`, `image`, `contour`; **histogramas**: `hist`; **gráficos de barras**: `barplot`; **gráficos de comparación de cuantiles**, usados para **comparar la distribución** de dos variables, o la distribución de unos datos frente a un estándard (ej., distribución normal): `qqplot`, `qgnorm` y, en paquete "car", `qq.plot`.
- Notación matemática (`plotmath`) y expresiones de texto arbitrariamente complejas.
- Gráficos tridimensionales dinámicos con XGobi y GGobi. (Ver <http://cran.r-project.org/src/contrib/PACKAGES.html#xgobi>, <http://www.ggobi.org>, <http://www.stat.auckland.ac.nz/~kwan022/pub/gguide.pdf>).

# *Guardando los gráficos*

---

- En Windows, podemos usar los menús y guardar con distintos formatos.

# *Guardando los gráficos*

---

- En Windows, podemos usar los menús y guardar con distintos formatos.
- Tambien podemos especificar donde queremos guardar el gráfico:

```
> pdf(file = "f1.pdf", width = 8, height = 10)
> plot(rnorm(10))
> dev.off()
```

# *Guardando los gráficos*

---

- En Windows, podemos usar los menús y guardar con distintos formatos.
- Tambien podemos especificar donde queremos guardar el gráfico:

```
> pdf(file = "f1.pdf", width = 8, height = 10)
> plot(rnorm(10))
> dev.off()
```

- O bien, podemos copiar una figura a un fichero: >  
`plot(runif(50))`  
`> dev.copy2eps()`

# *Programación en R*

---

- Definición de funciones

# *Programación en R*

---

- Definición de funciones
- Argumentos

# *Programación en R*

---

- Definición de funciones
- Argumentos
- Control de ejecución: condicionales, loops

# *Programación en R*

---

- Definición de funciones
- Argumentos
- Control de ejecución: condicionales, loops
- Cuando algo va mal: traceback, browser, debug

# *Programación en R*

---

- Definición de funciones
- Argumentos
- Control de ejecución: condicionales, loops
- Cuando algo va mal: traceback, browser, debug
- unix.time, Rprof

# *Definición de funciones*

---

- Ya hemos definido varias funciones. Aquí una más:

```
my.f2 <- function(x, y) {
 + z <- rnorm(10)
 + y2 <- z * y
 + y3 <- z * y * x
 + return(y3 + 25)
}
```

# *Definición de funciones*

---

- Ya hemos definido varias funciones. Aquí una más:

```
my.f2 <- function(x, y) {
 + z <- rnorm(10)
 + y2 <- z * y
 + y3 <- z * y * x
 + return(y3 + 25)
}
```

- Lo que una función devuelve puede ser un simple número o vector, o puede producir una gráfica, o devolver una lista o un mensaje.

# Argumentos

---

- `otra.f <- function(a, b, c = 4, d = FALSE) {  
+ x1 <- a * z ...}`

# Argumentos

- ```
otra.f <- function(a, b, c = 4, d = FALSE) {  
+     x1 <- a * z ...}
```
- Los argumentos "a" y "b" tienen que darse en el orden debido o, si los nombramos, podemos darlos en cualquier orden:
`otra.f(4, 5)`
`otra.f(b = 5, a = 4)`

Argumentos

- ```
otra.f <- function(a, b, c = 4, d = FALSE) {
+ x1 <- a * z ...}
```
- Los argumentos "a" y "b" tienen que darse en el orden debido o, si los nombramos, podemos darlos en cualquier orden:  
`otra.f(4, 5)`  
`otra.f(b = 5, a = 4)`
- Pero los argumentos con nombre siempre se tienen que dar despues de los posicionales  
`otra.f(c = 25, 4, 5) # error`

# Argumentos

- ```
otra.f <- function(a, b, c = 4, d = FALSE) {  
+     x1 <- a * z ...}
```
- Los argumentos "a" y "b" tienen que darse en el orden debido o, si los nombramos, podemos darlos en cualquier orden:
`otra.f(4, 5)`
`otra.f(b = 5, a = 4)`
- Pero los argumentos con nombre siempre se tienen que dar despues de los posicionales
`otra.f(c = 25, 4, 5) # error`
- Los argumentos "c" y "d" tienen "default values". Podemos especificarlos nosotros, o no especificarlos (i.e., usar los valores por defecto).

Argumentos

- `otra.f <- function(a, b, c = 4, d = FALSE) {
+ x1 <- a * z ...}`
- Los argumentos "a" y "b" tienen que darse en el orden debido o, si los nombramos, podemos darlos en cualquier orden:
`otra.f(4, 5)`
`otra.f(b = 5, a = 4)`
- Pero los argumentos con nombre siempre se tienen que dar despues de los posicionales
`otra.f(c = 25, 4, 5) # error`
- Los argumentos "c" y "d" tienen "default values". Podemos especificarlos nosotros, o no especificarlos (i.e., usar los valores por defecto).
- `args(nombre.funcion)` nos muestra los argumentos (de cualquier función).

- "z" es una "free variable": ¿cómo se especifica su valor?

Lexical scoping. Ver documento *Frames, environments, and scope in R and S-PLUS* de J. Fox (en

<http://cran.r-project.org/doc/contrib/Fox-Companion/appendix.html>) y
sección 10.7 en *An introduction to R*. También
demo(scoping).

- "z" es una "free variable": ¿cómo se especifica su valor?

Lexical scoping. Ver documento *Frames, environments, and scope in R and S-PLUS* de J. Fox (en

<http://cran.r-project.org/doc/contrib/Fox-Companion/appendix.html>) y sección 10.7 en *An introduction to R*. También demo(scoping).

- "... " permite pasar argumentos a otra función:

```
> f3 <- function(x, y, label = "la x", ...) {  
+   plot(x, y, xlab = label, ...)  
}  
>  
> f3(1:5, 1:5)  
> f3(1:5, 1:5, col = "red")
```

Control de ejecución: if

- Condicional: if

Control de ejecución: *if*

- Condicional: *if*
- *if (condicion.logica) instrucion* donde "*instrucion*" es cualquier expresión válida (incluida una entre {}).

Control de ejecución: *if*

- Condicional: *if*
- *if (condicion.logica) instrucion* donde "*instrucion*" es cualquier expresión válida (incluida una entre {}).
- *if (condicion.logica) instrucion else instrucion.alternativa.*

Control de ejecución: *if*

- Condicional: *if*
- *if (condicion.logica) instrucion* donde "*instrucion*" es cualquier expresión válida (incluida una entre {}).
- *if (condicion.logica) instrucion else instrucion.alternativa.*
- ```
> f4 <- function(x) {
+ if(x > 5) print("x > 5")
+ else {
+ y <- runif(1)
+ print(paste("y is ", y))
+ }
+ }
```

- `ifelse` es una versión vectorizada:

```
> #from Thomas Unternährer, R-help, 2003-04-17
> odd.even <- function(x) {
+ ifelse(x %% 2 == 1, "Odd", "Even")
+ }
```

- `ifelse` es una versión vectorizada:

```
> #from Thomas Unternährer, R-help, 2003-04-17
```

```
> odd.even <- function(x) {
+ ifelse(x %% 2 == 1, "Odd", "Even")
+ }
```

- ```
mtf <- matrix(c(TRUE, FALSE, TRUE, TRUE),  
+               nrow = 2)  
ifelse(mtf, 0, 1)
```

Control de ejecución: *while*, *for*

- `while (condicion.logica) instrucion`

Control de ejecución: *while*, *for*

- `while (condicion.logica) instrucion`
- `for (variable.loop in valores) instrucion`

Control de ejecución: *while*, *for*

- `while (condicion.logica) instrucion`
- `for (variable.loop in valores) instrucion`
- `for(i in 1:10) cat("el valor de i es", i, "\n")`

Control de ejecución: *while*, *for*

- `while (condicion.logica) instrucion`
- `for (variable.loop in valores) instrucion`
- `for(i in 1:10) cat("el valor de i es", i, "\n")`
- `continue.loop <- TRUE`
`x <- 0`
`while(continue.loop) {`
 - + `x <- x + 1`
 - + `print(x)`
 - + `if(x > 10) continue.loop <- FALSE``}`

Control de ejecución: *while*, *for*

- `while (condicion.logica) instrucion`
- `for (variable.loop in valores) instrucion`
- `for(i in 1:10) cat("el valor de i es", i, "\n")`
- `continue.loop <- TRUE`
`x <- 0`
`while(continue.loop) {`
 - + `x <- x + 1`
 - + `print(x)`
 - + `if(x > 10) continue.loop <- FALSE``}`
- `repeat`, `switch` también están disponibles.

Control de ejecución: *while*, *for*

- `while (condicion.logica) instrucion`
- `for (variable.loop in valores) instrucion`
- `for(i in 1:10) cat("el valor de i es", i, "\n")`
- `continue.loop <- TRUE`
`x <- 0`
`while(continue.loop) {`
 - + `x <- x + 1`
 - + `print(x)`
 - + `if(x > 10) continue.loop <- FALSE``}`
- `repeat`, `switch` también están disponibles.
- `break`, para salir de un loop.

Cuando algo va mal

- Cuando se produce un **error**, `traceback()` nos informa de la secuencia de llamadas antes del crash de nuestra función. Util cuando se produce mensajes de error incomprensibles.

Cuando algo va mal

- Cuando se produce un **error**, traceback() nos informa de la secuencia de llamadas antes del crash de nuestra función. Util cuando se produce mensajes de error incomprensibles.
- Cuando se producen **errores** o la función da **resultados incorrectos o warnings indebidos** podemos seguir la ejecución de la función.

Cuando algo va mal

- Cuando se produce un **error**, traceback() nos informa de la secuencia de llamadas antes del crash de nuestra función. Util cuando se produce mensajes de error incomprensibles.
- Cuando se producen **errores** o la función da **resultados incorrectos o warnings indebidos** podemos seguir la ejecución de la función.
- browser interrumpe la ejecución a partir de ese punto y permite seguir la ejecución o examinar el entorno; con "n" paso a paso, si otra tecla sigue ejecución normal. "Q" para salir.

Cuando algo va mal

- Cuando se produce un **error**, traceback() nos informa de la secuencia de llamadas antes del crash de nuestra función. Util cuando se produce mensajes de error incomprensibles.
- Cuando se producen **errores** o la función da **resultados incorrectos o warnings indebidos** podemos seguir la ejecución de la función.
- browser interrumpe la ejecución a partir de ese punto y permite seguir la ejecución o examinar el entorno; con "n" paso a paso, si otra tecla sigue ejecución normal. "Q" para salir.
- debug es como poner un "browser" al principio de la función, y se ejecuta la función paso a paso. Se sale con "Q".

```
> debug(my.buggy.function)  
> ...  
> undebug(my.buggy.function)
```

- my.f2 <- function(x, y) {
+ z <- rnorm(10)
+ y2 <- z * y
+ y3 <- z * y * x
+ return(y3 + 25)
}

- ```
my.f2 <- function(x, y) {
+ z <- rnorm(10)
+ y2 <- z * y
+ y3 <- z * y * x
+ return(y3 + 25)
}
```
- ```
my.f2(runif(3), 1:4)  
debug(my.f2)  
my.f2(runif(3), 1:4)  
undebug(my.f2)  
# insertar un browser() y correr de nuevo
```

unix.time y Rprof

- Nuestro objetivo aquí no es producir las funciones más eficientes, sino funciones que hagan lo que deben.

unix.time y Rprof

- Nuestro objetivo aquí no es producir las funciones más eficientes, sino funciones que hagan lo que deben.
- Las administraciones habituales sobre no optimizar y jamás optimizar antes de tiempo.

unix.time y *Rprof*

- Nuestro objetivo aquí no es producir las funciones más eficientes, sino funciones que hagan lo que deben.
- Las administraciones habituales sobre no optimizar y jamás optimizar antes de tiempo.
- Pero a veces útil saber cuanto dura la ejecución de una función: `unix.time(my.f2(runif(10000), rnorm(1000)))`.

unix.time y *Rprof*

- Nuestro objetivo aquí no es producir las funciones más eficientes, sino funciones que hagan lo que deben.
- Las administraciones habituales sobre no optimizar y jamás optimizar antes de tiempo.
- Pero a veces útil saber cuanto dura la ejecución de una función: `unix.time(my.f2(runif(10000), rnorm(1000)))`.
- Rprof: un profiler para ver cuantas veces es llamada cada función y cuanto tiempo se usa en esa función. Ver la ayuda.

unix.time y *Rprof*

- Nuestro objetivo aquí no es producir las funciones más eficientes, sino funciones que hagan lo que deben.
- Las administraciones habituales sobre no optimizar y jamás optimizar antes de tiempo.
- Pero a veces útil saber cuanto dura la ejecución de una función: `unix.time(my.f2(runif(10000), rnorm(1000)))`.
- Rprof: un profiler para ver cuantas veces es llamada cada función y cuanto tiempo se usa en esa función. Ver la ayuda.
- Garbage collection: desde R 1.2.0 hay garbage collection. Se puede ver el status con `gc()`, que sirve también para despejar las cosas después de operaciones con manejo de grandes objetos.

source y *BATCH*

- Para la ejecución no interactiva de código.

source y *BATCH*

- Para la ejecución no interactiva de código.
- Con `source` abrimos una sesión de R y hacemos > `source("mi.fichero.con.codigo.R")`.

source y *BATCH*

- Para la ejecución no interactiva de código.
- Con *source* abrimos una sesión de R y hacemos > `source("mi.fichero.con.codigo.R").`
- Con *BATCH*: % R CMD BATCH mi.fichero.con.codigo.R.

source y *BATCH*

- Para la ejecución no interactiva de código.
- Con *source* abrimos una sesión de R y hacemos > `source("mi.fichero.con.codigo.R")`.
- Con *BATCH*: % R CMD BATCH mi.fichero.con.codigo.R.
- *source* es en ocasiones más útil porque informa inmediatamente de errores en el código. *BATCH* no informa, pero no requiere tener abierta una sesión (se puede correr en el background).

source y *BATCH*

- Para la ejecución no interactiva de código.
- Con *source* abrimos una sesión de R y hacemos > `source("mi.fichero.con.codigo.R")`.
- Con *BATCH*: % R CMD BATCH mi.fichero.con.codigo.R.
- *source* es en ocasiones más útil porque informa inmediatamente de errores en el código. *BATCH* no informa, pero no requiere tener abierta una sesión (se puede correr en el background).
- Ver la ayuda: R CMD BATCH -help.

source y *BATCH*

- Para la ejecución no interactiva de código.
- Con `source` abrimos una sesión de R y hacemos > `source("mi.fichero.con.codigo.R")`.
- Con BATCH: % R CMD BATCH mi.fichero.con.codigo.R.
- `source` es en ocasiones más útil porque informa inmediatamente de errores en el código. BATCH no informa, pero no requiere tener abierta una sesión (se puede correr en el background).
- Ver la ayuda: R CMD BATCH -help.
- Puede que necesitemos explícitos `print` statements o hacer `source(my.file.R, echo = TRUE)`.

source y *BATCH*

- Para la ejecución no interactiva de código.
- Con *source* abrimos una sesión de R y hacemos > `source("mi.fichero.con.codigo.R")`.
- Con *BATCH*: % R CMD BATCH mi.fichero.con.codigo.R.
- *source* es en ocasiones más útil porque informa inmediatamente de errores en el código. *BATCH* no informa, pero no requiere tener abierta una sesión (se puede correr en el background).
- Ver la ayuda: R CMD BATCH -help.
- Puede que necesitemos explícitos `print` statements o hacer `source(my.file.R, echo = TRUE)`.
- *sink* es el inverso de *source* (manda todo a un fichero).

Otros

- Se pueden crear paquetes, con nuestras funciones, que se comporten igual que los demás paquetes. Ver *Writing R extensions*.

Otros

- Se pueden crear paquetes, con nuestras funciones, que se comporten igual que los demás paquetes. Ver *Writing R extensions*.
- R puede llamar código compilado en C/C++ y FORTRAN. Ver `.C`, `.Call`, `.Fortran`.

Otros

- Se pueden crear paquetes, con nuestras funciones, que se comporten igual que los demás paquetes. Ver *Writing R extensions*.
- R puede llamar código compilado en C/C++ y FORTRAN. Ver `.C`, `.Call`, `.Fortran`.
- Lexical scoping importante en programación más avanzada.

Otros

- Se pueden crear paquetes, con nuestras funciones, que se comporten igual que los demás paquetes. Ver *Writing R extensions*.
- R puede llamar código compilado en C/C++ y FORTRAN. Ver `.C`, `.Call`, `.Fortran`.
- Lexical scoping importante en programación más avanzada.
- No hemos mencionado el "computing on the language" (ej., `do.call`, `eval`, etc).

Otros

- Se pueden crear paquetes, con nuestras funciones, que se comporten igual que los demás paquetes. Ver *Writing R extensions*.
- R puede llamar código compilado en C/C++ y FORTRAN. Ver `.C`, `.Call`, `.Fortran`.
- Lexical scoping importante en programación más avanzada.
- No hemos mencionado el "computing on the language" (ej., `do.call`, `eval`, etc).
- R es un verdadero "object-oriented language". Dos implementaciones, las S3 classes y las S4 classes.

Ejemplos

- t-test-then-cluster fallacy

Ejemplos

- t-test-then-cluster fallacy
- Validación cruzada de un predictor
(con y sin "selection bias")

Ejemplos

- t-test-then-cluster fallacy
- Validación cruzada de un predictor
(con y sin "selection bias")
- Ejercicio: método de van't Veer et al.

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.
- Lo que no está claro es como de fiables son esos clusters.

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.
- Lo que no está claro es como de fiables son esos clusters.
- Si hacemos clustering con genes seleccionados con un test de la t, siempre nos salen magníficos resultados aún con datos aleatorios. Veámoslo en acción.

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.
- Lo que no está claro es como de fiables son esos clusters.
- Si hacemos clustering con genes seleccionados con un test de la t, siempre nos salen magníficos resultados aún con datos aleatorios. Veámoslo en acción.
- Objetivo: usando datos aleatorios, replicar el proceso de:

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.
- Lo que no está claro es como de fiables son esos clusters.
- Si hacemos clustering con genes seleccionados con un test de la t, siempre nos salen magníficos resultados aún con datos aleatorios. Veámoslo en acción.
- Objetivo: usando datos aleatorios, replicar el proceso de:
 - Seleccionar genes.

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.
- Lo que no está claro es como de fiables son esos clusters.
- Si hacemos clustering con genes seleccionados con un test de la t, siempre nos salen magníficos resultados aún con datos aleatorios. Veámoslo en acción.
- Objetivo: usando datos aleatorios, replicar el proceso de:
 - Seleccionar genes.
 - Hacer clustering con esos genes.

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.
- Lo que no está claro es como de fiables son esos clusters.
- Si hacemos clustering con genes seleccionados con un test de la t, siempre nos salen magníficos resultados aún con datos aleatorios. Veámoslo en acción.
- Objetivo: usando datos aleatorios, replicar el proceso de:
 - Seleccionar genes.
 - Hacer clustering con esos genes.
 - Representar el cluster

t-test-then-cluster fallacy

- Truismo: un algoritmo de clustering *siempre* nos devuelve clusters.
- Lo que no está claro es como de fiables son esos clusters.
- Si hacemos clustering con genes seleccionados con un test de la t, siempre nos salen magníficos resultados aún con datos aleatorios. Veámoslo en acción.
- Objetivo: usando datos aleatorios, replicar el proceso de:
 - Seleccionar genes.
 - Hacer clustering con esos genes.
 - Representar el cluster
- Por sencillez, suponemos que tenemos sólo dos clases de sujetos.

clustering: algoritmo

- Buscamos funciones de clustering:

```
> help.search("cluster")  
> help.search("hierar cluster")  
> args(hclust) # una primera idea  
> ?hclust # cómo funciona hclust?
```

clustering: algoritmo

- Buscamos funciones de clustering:

```
> help.search("cluster")  
> help.search("hierar cluster")  
> args(hclust) # una primera idea  
> ?hclust # cómo funciona hclust?
```

- Usaremos hclust (aunque podríamos usar cualquier otra).

Selección de genes

- (En el siguiente ejemplo veremos otra forma de hacerlo).

Selección de genes

- (En el siguiente ejemplo veremos otra forma de hacerlo).
- Queremos seleccionar aquellos genes para los que el p-valor de un test de la t sea "significativo".

Selección de genes

- (En el siguiente ejemplo veremos otra forma de hacerlo).
- Queremos seleccionar aquellos genes para los que el p-valor de un test de la t sea "significativo".
- ```
gene.select <-function(x, class, threshold) {
 + p.values <- apply(x, 1, function(x) {
 + t.test(x ~ class)$p.value})
 + return(which(p.values < threshold))
}
```

# Selección de genes

- La probamos:

```
> grupo1 <- 10
> grupo2 <- 10
> genes <- 5000
> xmat <- matrix(rnorm(genes * (grupo1 + grupo2)),
+ nrow = genes)
> labels <- c(rep(0, grupo1), rep(1, grupo2))
> coco <- gene.select(xmat, labels, 0.05)
> length(coco)
> unix.time(gene.select(xmat, labels, 0.05))
> # muy lenta
```

# Selección de genes

- La probamos:

```
> grupo1 <- 10
> grupo2 <- 10
> genes <- 5000
> xmat <- matrix(rnorm(genes * (grupo1 + grupo2)),
+ nrow = genes)
> labels <- c(rep(0, grupo1), rep(1, grupo2))
> coco <- gene.select(xmat, labels, 0.05)
> length(coco)
> unix.time(gene.select(xmat, labels, 0.05))
> # muy lenta
```

- Fijaos que usamos el formato de genes en filas y sujetos en columnas.

# *La función*

```
● t.test.cluster <- function(data, labels, threshold,
+ hang = -1) {
+ gene.select <- function(x, class, threshold)
+ { ... }
+ genes.selected <- gene.select(data, labels,
+ threshold)
+ data.reduced <- data[genes.selected, ,
+ drop = FALSE]
+ mi.cluster <- hclust(dist(t(data.reduced))^2,
+ method = "average")
+ plot(mi.cluster, hang = hang)
}
```

# *La función*

- ```
t.test.cluster <- function(data, labels, threshold,
+      hang = -1) {
+      gene.select <- function(x, class, threshold)
+      { ... }
+      genes.selected <- gene.select(data, labels,
+          threshold)
+      data.reduced <- data[genes.selected, ,
+          drop = FALSE]
+      mi.cluster <- hclust(dist(t(data.reduced))^2,
+          method = "average")
+      plot(mi.cluster, hang = hang)
}
```

- A correr:

```
t.test.cluster(xmat, labels, 0.05, 0)
```

t-test-then-... : Ejercicios

- Usar selección de genes basáandonos en adjusted p-values.
(Usar librería multtest).

t-test-then-... : Ejercicios

- Usar selección de genes basáandonos en adjusted p-values.
(Usar librería multtest).
- Usar otros algoritmos de clustering.

t-test-then-... : Ejercicios

- Usar selección de genes basáandonos en adjusted p-values.
(Usar librería multtest).
- Usar otros algoritmos de clustering.
- Usar tres clases en vez de dos (usar como estadístico una F o como test un ANOVA).

Cross-validation de un predictor

- Objetivos:

- Usar validación cruzada (cross-validation) para evaluar funcionamiento (error de predicción) de un predictor.
- Comparar los resultados de validación cruzada teniendo en cuenta y sin tener en cuenta el selection bias (resultado de un prefiltrado de los genes).
- (Referencias: Ambroise & McLachlan, 2002 en <http://www.pnas.org/cgi/content/abstract/99/10/656> y Simon et al., 2003, J. Nat. Cancer Institute, 95 : 14–18.)

Cross-validation de un predictor

- Objetivos:
 - Usar validación cruzada (cross-validation) para evaluar funcionamiento (error de predicción) de un predictor.
 - Comparar los resultados de validación cruzada teniendo en cuenta y sin tener en cuenta el selection bias (resultado de un prefiltrado de los genes).
 - (Referencias: Ambroise & McLachlan, 2002 en <http://www.pnas.org/cgi/content/abstract/99/10/656> y Simon et al., 2003, J. Nat. Cancer Institute, 95 : 14–18.)
- Usaremos dos predictores: knn (k-nearest neighbor) y svm (support vector machines), porque funcionan bien, y son populares (y están implementados en R).

Cross-validation de un predictor

- Objetivos:
 - Usar validación cruzada (cross-validation) para evaluar funcionamiento (error de predicción) de un predictor.
 - Comparar los resultados de validación cruzada teniendo en cuenta y sin tener en cuenta el selection bias (resultado de un prefiltrado de los genes).
 - (Referencias: Ambroise & McLachlan, 2002 en <http://www.pnas.org/cgi/content/abstract/99/10/656> y Simon et al., 2003, J. Nat. Cancer Institute, 95 : 14–18.)
- Usaremos dos predictores: knn (k-nearest neighbor) y svm (support vector machines), porque funcionan bien, y son populares (y están implementados en R).
- Por simplicidad, supondremos que tenemos una situación con dos clases (llamadas 0 y 1).

Cross-validation: pasos

- ¿Dónde están las funciones para knn y svm?

Cross-validation: pasos

- ¿Dónde están las funciones para knn y svm?

- `help.search("knn")`

- `help.search("nearest neighbor") # UK o USA?`

- `help.search("vector machine")`

Cross-validation: pasos

- ¿Dónde están las funciones para knn y svm?

- `help.search("knn")`

- `help.search("nearest neighbor") # UK o USA?`

- `help.search("vector machine")`

- He "hecho trampa", porque e1071 no es un paquete que se instale por defecto. Por eso, si no la tuviera instalada, no la hubiera encontrado. Solución: ir a CRAN, y buscar entre los packages (y en R-help y

- <http://finzi.psych.upenn.edu/search.html>)

Cross-validation: pasos

- ¿Dónde están las funciones para knn y svm?
 - `help.search("knn")`
`help.search("nearest neighbor") # UK o USA?`
`help.search("vector machine")`
 - He "hecho trampa", porque e1071 no es un paquete que se instale por defecto. Por eso, si no la tuviera instalada, no la hubiera encontrado. Solución: ir a CRAN, y buscar entre los packages (y en R-help y
<http://finzi.psych.upenn.edu/search.html>)
- Código para selección de genes.

Cross-validation: pasos

- ¿Dónde están las funciones para knn y svm?
 - `help.search("knn")`
`help.search("nearest neighbor") # UK o USA?`
`help.search("vector machine")`
 - He "hecho trampa", porque e1071 no es un paquete que se instale por defecto. Por eso, si no la tuviera instalada, no la hubiera encontrado. Solución: ir a CRAN, y buscar entre los packages (y en R-help y
<http://finzi.psych.upenn.edu/search.html>)
- Código para selección de genes.
- Código para cross-validation.

CV: svm

- Con svm, necesitamos las predicciones del modelo. Vamos a ?svm y encontramos predict.svm.

CV: svm

- Con svm, necesitamos las predicciones del modelo. Vamos a ?svm y encontramos predict.svm.
- Las predicciones de clase para unos datos de test (test.data) con un predictor entrenado con unos datos de training (training.data, cuyas clases son cl.train.data) se obtienen entonces como:

```
fitted.svm <- svm(training.data, cl.train.data,  
+      kernel = "linear")  
predict(fitted.svm, test.data)
```

CV: svm

- Con svm, necesitamos las predicciones del modelo. Vamos a ?svm y encontramos predict.svm.
- Las predicciones de clase para unos datos de test (test.data) con un predictor entrenado con unos datos de training (training.data, cuyas clases son cl.train.data) se obtienen entonces como:

```
fitted.svm <- svm(training.data, cl.train.data,  
+       kernel = "linear")  
  
predict(fitted.svm, test.data)
```

- Pero eso se puede simplificar, porque nosotros no necesitamos para nada "fitted.svm":

```
predict(svm(training.data, cl.train.data,  
+       kernel = "linear"), test.data)
```

-
- Ojo, esta es la estructura habitual de sujetos en filas y variables en columnas.

- Ojo, esta es la estructura habitual de sujetos en filas y variables en columnas.
- ¿Estamos seguros de predict.svm? Lo probamos o jugamos con algunos ejemplos de la ayuda (no mostrado).

CV:*Error rates*

- Con predicciones, fácil obtener error rates. Error: todo aquel caso para el que predicho != real. Por si acaso hay desviaciones sistemáticas, desglosamos la situación entre predichos de clase 0 y observados de clase 1, y predichos de clase 1 y observados de clase 0.

CV:Error rates

- Con predicciones, fácil obtener error rates. Error: todo aquel caso para el que predicho != real. Por si acaso hay desviaciones sistemáticas, desglosamos la situación entre predichos de clase 0 y observados de clase 1, y predichos de clase 1 y observados de clase 0.
- Para cada conjunto de test tres columnas: predicho 0 y observado 1, predicho 1 y observado 0, y la suma de las dos anteriores. Una cuarta columna donde anotamos el número de casos en esa partición. Llaremos a esta matriz "cv.errors".

CV:Error rates

- Con predicciones, fácil obtener error rates. Error: todo aquel caso para el que predicho != real. Por si acaso hay desviaciones sistemáticas, desglosamos la situación entre predichos de clase 0 y observados de clase 1, y predichos de clase 1 y observados de clase 0.
- Para cada conjunto de test tres columnas: predicho 0 y observado 1, predicho 1 y observado 0, y la suma de las dos anteriores. Una cuarta columna donde anotamos el número de casos en esa partición. Llaremos a esta matriz "cv.errors".
- Despues de examinar todas las particiones, el error de predicción será:

```
sum(cv.errors[, 3])/sum(cv.errors[, 4]).
```

CV:Error rates

- Con predicciones, fácil obtener error rates. Error: todo aquel caso para el que predicho != real. Por si acaso hay desviaciones sistemáticas, desglosamos la situación entre predichos de clase 0 y observados de clase 1, y predichos de clase 1 y observados de clase 0.
- Para cada conjunto de test tres columnas: predicho 0 y observado 1, predicho 1 y observado 0, y la suma de las dos anteriores. Una cuarta columna donde anotamos el número de casos en esa partición. Llaremos a esta matriz "cv.errors".
- Despues de examinar todas las particiones, el error de predicción será:
`sum(cv.errors[, 3])/sum(cv.errors[, 4]).`
- (Por eficiencia, retraso el nombrar las columnas hasta el final del código).

CV: Selección de genes

- Coger k ($k = 200?$) genes con expresión más distinta.

CV: Selección de genes

- Coger k ($k = 200?$) genes con expresión más distinta.
- Juzgamos "expresión más distinta" con test de la t.

CV: Selección de genes

- Coger k ($k = 200?$) genes con expresión más distinta.
- Juzgamos "expresión más distinta" con test de la t.
- Podríamos hacer:

```
t.statistic <- apply(train.data, 2,  
+ function(x) {  
+   abs(t.test(x ~ cl.train.data)$statistic)})
```

CV: Selección de genes

- Coger k ($k = 200?$) genes con expresión más distinta.
- Juzgamos "expresión más distinta" con test de la t.
- Podríamos hacer:

```
t.statistic <- apply(train.data, 2,  
+ function(x) {  
+ abs(t.test(x ~ cl.train.data)$statistic)})
```

- Pero no es muy rápido. `x <- matrix(rnorm(1000*50), nrow = 50)`

```
clases <- c(rep(0, 25), rep(1, 25))
```

```
unix.time(
```

```
+ t.stat <- apply(x, 2, function(x) {  
+ abs(t.test(x ~ clases)$statistic)})  
+ )
```

-
- Podríamos vivir con ello si fueramos a usar nuestra función sólo unas pocas veces. Pero yo la uso mucho en simulaciones. Necesitamos algo más rápido.

- Podríamos vivir con ello si fueramos a usar nuestra función sólo unas pocas veces. Pero yo la uso mucho en simulaciones. Necesitamos algo más rápido.
- Usaremos "mt.maxT", del paquete "multtest" (implementa el test de la t directamente en C, por lo que es mucho más rápida.)

- Podríamos vivir con ello si fueramos a usar nuestra función sólo unas pocas veces. Pero yo la uso mucho en simulaciones. Necesitamos algo más rápido.
- Usaremos "mt.maxT", del paquete "multtest" (implementa el test de la t directamente en C, por lo que es mucho más rápida.)
- mt.maxT sirve para obtener p-values ajustados para multiple testing. Pero nosotros no estamos aquí interesados en el p-value, sino en el estadístico, que usamos sólo como una forma de ordenar los genes para su posterior selección.

- Podríamos vivir con ello si fueramos a usar nuestra función sólo unas pocas veces. Pero yo la uso mucho en simulaciones. Necesitamos algo más rápido.
- Usaremos "mt.maxT", del paquete "multtest" (implementa el test de la t directamente en C, por lo que es mucho más rápida.)
- mt.maxT sirve para obtener p-values ajustados para multiple testing. Pero nosotros no estamos aquí interesados en el p-value, sino en el estadístico, que usamos sólo como una forma de ordenar los genes para su posterior selección.
- Por eso, podemos usar mt.maxT sin permutaciones.

- Podríamos vivir con ello si fueramos a usar nuestra función sólo unas pocas veces. Pero yo la uso mucho en simulaciones. Necesitamos algo más rápido.
- Usaremos "mt.maxT", del paquete "multtest" (implementa el test de la t directamente en C, por lo que es mucho más rápida.)
- mt.maxT sirve para obtener p-values ajustados para multiple testing. Pero nosotros no estamos aquí interesados en el p-value, sino en el estadístico, que usamos sólo como una forma de ordenar los genes para su posterior selección.
- Por eso, podemos usar mt.maxT sin permutaciones.
- mt.maxT recibe los datos en la forma habitual de las microarrays (sujetos en columnas, genes en filas). Habrá que trasponerlos.

- Cargamos el paquete librería, y vemos la ayuda para saber qué nos interesa:

```
> library(multtest)  
> ?mt.maxT  
> # un ejemplo corto, para ver con  
> ## que nos quedamos:  
> data(golub)  
> tmp <- mt.maxT(golub[1:20, ], golub.cl, B = 1)  
> tmp # nos basta la primera columna o la 1a y 2a.
```

- Nuestra función es un simple "wrapper" a `mt.maxT`, incluyendo el número de genes con el que nos queremos quedar (`size`) (`threshold` no lo usamos aquí):

```
> gene.select <-  
+   function(data, class,  
+   size, threshold = NULL) {  
+     ## t.stat <- apply(data, 2, function(x)  
+     ## {abs(t.test(x ~ class)$statistic)}) slow  
+     tmp <- mt.maxT(t(data),class,B= 1)[,c(1, 2)]  
+     selected <- tmp[seq(1:size), 1]  
+     return(selected)  
}
```

-
- Traspongo los datos al llamar a `mt.maxT`.

-
- Traspongo los datos al llamar a `mt.maxT`.
 - He dejado el antiguo código que usa la función `t.stat` directamente, por si algún día me hace falta. Y he puesto comentarios sobre lo que ocurre.

CV:Código para cv

- Si usamos, por ej., 10-fold cross-validation, lo que queremos son 10 fracciones de los datos; entrenamos con las 9 primeras y testamos en la 10^a restante; repetimos con las demás. (Llamemos a 10 el "knumber").

CV:Código para cv

- Si usamos, por ej., 10-fold cross-validation, lo que queremos son 10 fracciones de los datos; entrenamos con las 9 primeras y testamos en la 10^a restante; repetimos con las demás. (Llamemos a 10 el "knumber").
- Código de cv. sencillo si asociamos cada sujeto a un número, de 1 a 10. Y ahora iteramos de 1 a 10.

CV:Código para cv

- Si usamos, por ej., 10-fold cross-validation, lo que queremos son 10 fracciones de los datos; entrenamos con las 9 primeras y testamos en la 10^a restante; repetimos con las demás. (Llamemos a 10 el "knumber").
- Código de cv. sencillo si asociamos cada sujeto a un número, de 1 a 10. Y ahora iteramos de 1 a 10.
- Cuando el iterador es, por ej., 3, los sujetos con el número 3 son del test set y todos los demás del training set.

CV:Código para cv

- Si usamos, por ej., 10-fold cross-validation, lo que queremos son 10 fracciones de los datos; entrenamos con las 9 primeras y testamos en la 10^a restante; repetimos con las demás. (Llamemos a 10 el "knumber").
- Código de cv. sencillo si asociamos cada sujeto a un número, de 1 a 10. Y ahora iteramos de 1 a 10.
- Cuando el iterador es, por ej., 3, los sujetos con el número 3 son del test set y todos los demás del training set.
- Queremos que el tamaño sea balanceado.

- El siguiente código está sacado de Venables & Ripley, *S Programming*, p. 175.

```
> los.indices <- rep(1:knumber, length = N)
> # now, reshuffle
> sample(los.indices, N, replace = FALSE)
> # or sample(los.indices)
```

- El siguiente código está sacado de Venables & Ripley, *S Programming*, p. 175.

```
> los.indices <- rep(1:knumber, length = N)
> # now, reshuffle
> sample(los.indices, N, replace = FALSE)
> # or sample(los.indices)
```

- "los.indices" genera secuencias de 1 a knumber, y las repite hasta que tenemos un total de números igual al número de sujetos. Por tanto, bastante balanceado. Y por último, reshuffle aleatoriamente.

- El siguiente código está sacado de Venables & Ripley, *S Programming*, p. 175.

```
> los.indices <- rep(1:knumber, length = N)
> # now, reshuffle
> sample(los.indices, N, replace = FALSE)
> # or sample(los.indices)
```

- "los.indices" genera secuencias de 1 a knumber, y las repite hasta que tenemos un total de números igual al número de sujetos. Por tanto, bastante balanceado. Y por último, reshuffle aleatoriamente.
- Lo podemos simplificar:

```
index.select <-
  +
  sample(rep(1:knumber, length = N),
  N, replace = FALSE)
```

- Por último facilitamos las cosas haciendo que leave-one-out no requiera que recordemos el número de sujetos:

```
if(is.null(knumber)) {  
  knumber <- length(y)  
}  
  
N <- length(y)  
  
index.select <-  
  sample(rep(1:knumber, length = N),  
  N, replace = FALSE)
```

CV: poniendo todo junto

- Ya tenemos todas las piezas. El esquema es:

CV: poniendo todo junto

- Ya tenemos todas las piezas. El esquema es:
 - Generar lista de sujetos de cada partición.

CV: poniendo todo junto

- Ya tenemos todas las piezas. El esquema es:
 - Generar lista de sujetos de cada partición.
 - Para cada training set, hacer:

CV: poniendo todo junto

- Ya tenemos todas las piezas. El esquema es:
 - Generar lista de sujetos de cada partición.
 - Para cada training set, hacer:
 - Selección de genes

CV: poniendo todo junto

- Ya tenemos todas las piezas. El esquema es:
 - Generar lista de sujetos de cada partición.
 - Para cada training set, hacer:
 - Selección de genes
 - Predicción de los sujetos en el testing set usando los genes seleccionados.

CV: poniendo todo junto

- Ya tenemos todas las piezas. El esquema es:
 - Generar lista de sujetos de cada partición.
 - Para cada training set, hacer:
 - Selección de genes
 - Predicción de los sujetos en el testing set usando los genes seleccionados.
 - Media de errores de predicción sobre el total de los testing sets.

Selection bias

- Para no corregir el selection bias, la selección de genes se hace antes de la partición de sujetos en training y testing sets.

Selection bias

- Para no corregir el selection bias, la selección de genes se hace antes de la partición de sujetos en training y testing sets.
- De esa forma, la selección de genes se hace con el total de los sujetos.

Selection bias

- Para no corregir el selection bias, la selección de genes se hace antes de la partición de sujetos en training y testing sets.
- De esa forma, la selección de genes se hace con el total de los sujetos.
- Por tanto, no se replica con honestidad el proceso de construcción del predictor.

Selection bias

- Para no corregir el selection bias, la selección de genes se hace antes de la partición de sujetos en training y testing sets.
- De esa forma, la selección de genes se hace con el total de los sujetos.
- Por tanto, no se replica con honestidad el proceso de construcción del predictor.
- Miremos el código.

Cross-validation: KNN

- Todo es similar a svm, excepto porque usamos KNN.

Cross-validation: KNN

- Todo es similar a svm, excepto porque usamos KNN.
- Con KNN tenemos que decidir cuantos neighbors usamos.

Cross-validation: KNN

- Todo es similar a svm, excepto porque usamos KNN.
- Con KNN tenemos que decidir cuantos neighbors usamos.
- La función `knn.cv` devuelve las leave-one-out predictions para cada sujeto. Podemos utilizar esta función para seleccionar el número de neighbors.

Cross-validation: KNN

- Todo es similar a svm, excepto porque usamos KNN.
- Con KNN tenemos que decidir cuantos neighbors usamos.
- La función `knn.cv` devuelve las leave-one-out predictions para cada sujeto. Podemos utilizar esta función para seleccionar el número de neighbors.
- Para un rango de tamaño de vecino, examinamos cual produce el menor leave-one-out error, y usamos ese valor como el número óptimo de vecinos.

Cross-validation: KNN

- Todo es similar a svm, excepto porque usamos KNN.
- Con KNN tenemos que decidir cuantos neighbors usamos.
- La función `knn.cv` devuelve las leave-one-out predictions para cada sujeto. Podemos utilizar esta función para seleccionar el número de neighbors.
- Para un rango de tamaño de vecino, examinamos cual produce el menor leave-one-out error, y usamos ese valor como el número óptimo de vecinos.
- Ese número óptimo se usa luego para predecir la clase del testing set (i.e., unos datos que NO han sido usados en la elección del número de vecinos).

Cross-validation: KNN

- Todo es similar a svm, excepto porque usamos KNN.
- Con KNN tenemos que decidir cuantos neighbors usamos.
- La función `knn.cv` devuelve las leave-one-out predictions para cada sujeto. Podemos utilizar esta función para seleccionar el número de neighbors.
- Para un rango de tamaño de vecino, examinamos cual produce el menor leave-one-out error, y usamos ese valor como el número óptimo de vecinos.
- Ese número óptimo se usa luego para predecir la clase del testing set (i.e., unos datos que NO han sido usados en la elección del número de vecinos).
- Por tanto, tenemos un leave-one-out dentro de una cross-validation.

KNN: número óptimo de vecinos

- ```
select.k <- function(data, class, max.k = 20) {
 error.k <- rep(-9, max.k)
 for (i in 1:max.k) {
 error.k[i]
 <- length(which(class !=
 knn.cv(data, class, k = i)))
 }
 return(which.min(error.k)) }
```

# **CV: ¿podemos simplificar?**

---

- Las cuatro funciones son muy similares: casi todas las líneas de código son idénticas.

# **CV: ¿podemos simplificar?**

---

- Las cuatro funciones son muy similares: casi todas las líneas de código son idénticas.
- Podríamos usar un sólo esqueleto básico, y especificar si la selección de genes es intra cross-validation.

# **CV: ¿podemos simplificar?**

---

- Las cuatro funciones son muy similares: casi todas las líneas de código son idénticas.
- Podríamos usar un sólo esqueleto básico, y especificar si la selección de genes es intra cross-validation.
- Además, pasar la función de predicción (svm o knn) como un argumento.

# **CV: ¿podemos simplificar?**

---

- Las cuatro funciones son muy similares: casi todas las líneas de código son idénticas.
- Podríamos usar un sólo esqueleto básico, y especificar si la selección de genes es intra cross-validation.
- Además, pasar la función de predicción (svm o knn) como un argumento.
- Lo último es lo que se hace en el paquete "ipred".

# **CV: ¿podemos simplificar?**

---

- Las cuatro funciones son muy similares: casi todas las líneas de código son idénticas.
- Podríamos usar un sólo esqueleto básico, y especificar si la selección de genes es intra cross-validation.
- Además, pasar la función de predicción (svm o knn) como un argumento.
- Lo último es lo que se hace en el paquete "ipred".
- En nuestro caso no merece demasiado la pena, porque probablemente usemos las funciones sólo unas pocas veces.

# *Ejercicio: van't Veer et al.*

---

- van't Veer et al. (2002, Nature, 415:530–536) describen un método supervisado en tres etapas.

# *Ejercicio: van't Veer et al.*

---

- van't Veer et al. (2002, Nature, 415:530–536) describen un método supervisado en tres etapas.
  - Preseleccionar genes (ej., basado en un test de la t). La preselección deja un gran número de genes (en su caso, 5000).

# **Ejercicio: van't Veer et al.**

---

- van't Veer et al. (2002, Nature, 415:530–536) describen un método supervisado en tres etapas.
  - Preseleccionar genes (ej., basado en un test de la t). La preselección deja un gran número de genes (en su caso, 5000).
  - Calcular correlación entre outcome (clase) y expresión; seleccionamos sólo aquellos con  $\text{corr.coeff.} > 0.3$  o  $< -0.3$ .

# **Ejercicio: van't Veer et al.**

---

- van't Veer et al. (2002, Nature, 415:530–536) describen un método supervisado en tres etapas.
  - Preseleccionar genes (ej., basado en un test de la t). La preselección deja un gran número de genes (en su caso, 5000).
  - Calcular correlación entre outcome (clase) y expresión; seleccionamos sólo aquellos con  $\text{corr.coeff.} > 0.3$  o  $< -0.3$ .
  - Ordenar genes en función del (valor absoluto?) del coeficiente de correlación.

# **Ejercicio: van't Veer et al.**

---

- van't Veer et al. (2002, Nature, 415:530–536) describen un método supervisado en tres etapas.
  - Preseleccionar genes (ej., basado en un test de la t). La preselección deja un gran número de genes (en su caso, 5000).
  - Calcular correlación entre outcome (clase) y expresión; seleccionamos sólo aquellos con  $\text{corr.coeff.} > 0.3$  o  $< -0.3$ .
  - Ordenar genes en función del (valor absoluto?) del coeficiente de correlación.
  - El clasificador se optimiza añadiendo secuencialmente conjuntos de 5 genes (los 5 con rango superior). Se evalúa el clasificador usando leave-one-out.

# **Ejercicio: van't Veer et al.**

- van't Veer et al. (2002, Nature, 415:530–536) describen un método supervisado en tres etapas.
  - Preseleccionar genes (ej., basado en un test de la t). La preselección deja un gran número de genes (en su caso, 5000).
  - Calcular correlación entre outcome (clase) y expresión; seleccionamos sólo aquellos con  $\text{corr.coeff.} > 0.3$  o  $< -0.3$ .
  - Ordenar genes en función del (valor absoluto?) del coeficiente de correlación.
  - El clasificador se optimiza añadiendo secuencialmente conjuntos de 5 genes (los 5 con rango superior). Se evalúa el clasificador usando leave-one-out.
- La clasificación se hace basándose en la correlación del perfil de la left-out-sample con "the mean expression levels of the remaining samples from the good and poor prognosis".

- Pero... ¿es la correlación con el perfil de expresión medio, o la media de las correlaciones con los perfiles de expresión? Es la primera (su Fig.2b).

- Pero... ¿es la correlación con el perfil de expresión medio, o la media de las correlaciones con los perfiles de expresión? Es la primera (su Fig.2b).
- ¿Y qué se hace si un gen tiene correlación positiva con la prognosis y otro negativa? ¿Cambiaremos los signos?

# *Fin*

---