Entorno Estadístico R

Una nueva generación de software estadístico gratuito

Manuel Febrero Bande / Beatriz Pateiro López

¿Qué es R?

- Lenguaje derivado del S (Bell Labs.) con implementaciones para manipulación de datos, cálculo matricial y análisis gráfico ➔ Lenguaje estadístico y matemático.
- Entorno para el desarrollo de nuevos métodos de análisis de datos gratuito (⇏baja calidad) y dinámico (⇏inestable)
- Pág. principal: http://www.r-project.org

Un poco de historia

1976
- Lenguaje S (Bell Laboratories)

1988
- Nuevo lenguaje S (John Chambers)
- "Turn ideas into software, quickly and faithfully"

1994
- S-PLUS 3.2

1997
- R 0.16 (Ross Ihaka & Robert Gentleman)

S-PLUS 8.0

R 2.8.1

Necesidades de R

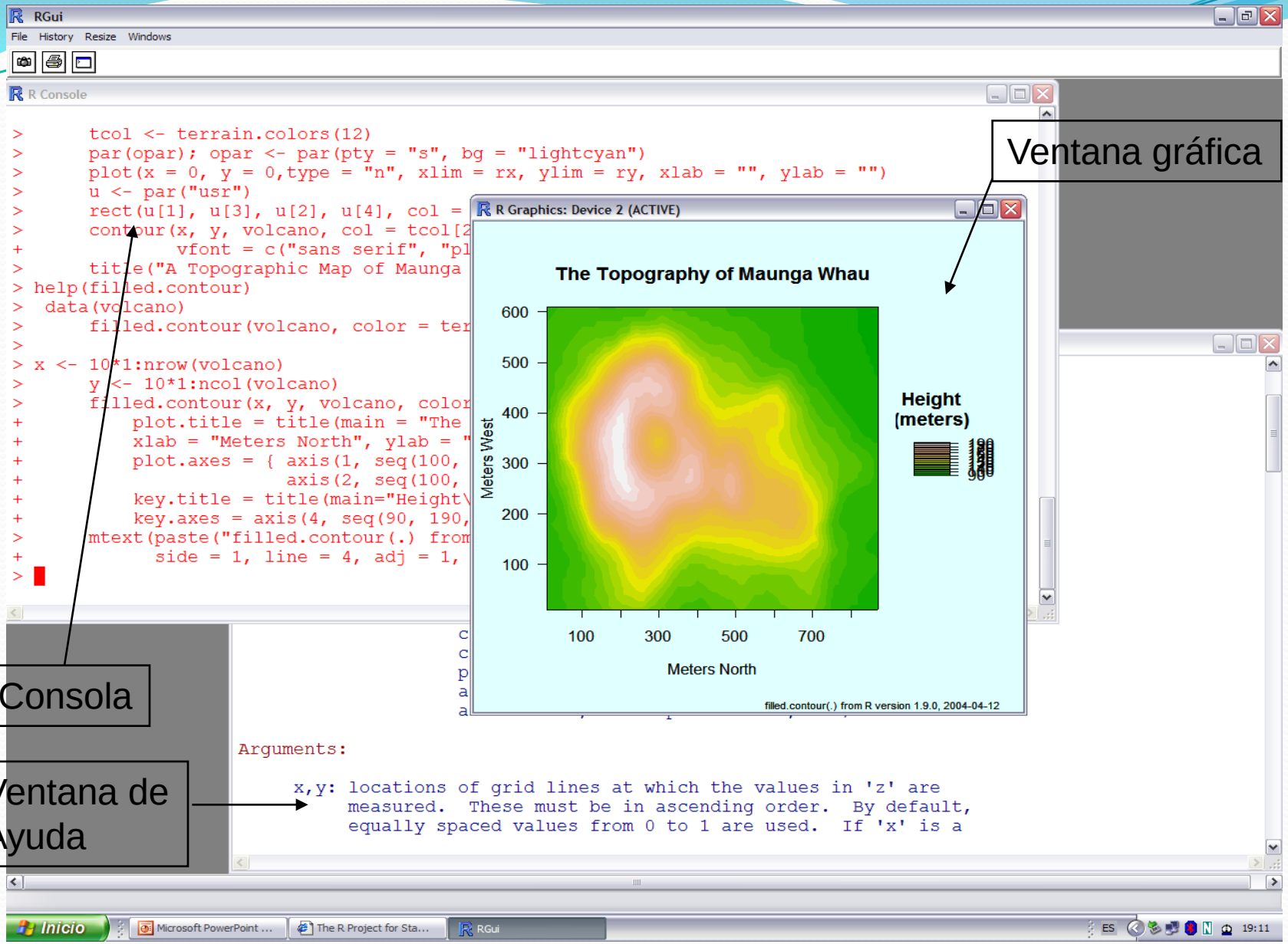
- Cualquier sistema operativo con o sin interfaz gráfica (Linux, MacOS, Windows, AIX, etc) sin importar la potencia del equipo.
- Deseable conexión a Internet para instalación, actualización y búsqueda de nuevos procedimientos.
- Fichero instalación: 34 MB
  Instalación básica: 50 MB
  Nº de paquetes totales: 1719
  Instalación con todos los paquetes: 2.21 GB.

Lo más importante: Ayuda

- http://cran.es.r-project.org/
  - Search – Menu CRAN
  - FAQ –Menu Documentation
  - Manuals –Menu Documentation
  - Contributed –Menu Documentation
  - Newsletter –Menu Documentation
- Dentro de R
  - `help.search("quiero esto")`
  - `help(function); ?function`
  - `Ayuda en HTML`

Antes de empezar con R

- Lenguaje interpretado orientado a objetos
- Es “case sensitive”
- Los comandos en una línea se separan con ;
- Comentarios: empiezan por #
- Si un comando no está completo el carácter de continuación mostrado es: +
- Se agrupan comandos con {}
- Asignación con: `<-`, `<<-`, `=`, `->`, `->>`


RGui
File History Resize Windows
R Console
> tcol <- terrain.colors(12)
> par(opar); opar <- par(pty = "s", bg = "lightcyan")
> plot(x = 0, y = 0,type = "n", xlim = rx, ylim = ry, xlab = "", ylab = "")
> u <- par("usr")
> rect(u[1], u[3], u[2], u[4], col =
> contour(x, y, volcano, col = tcol[2
+ vfont = c("sans serif", "pl
> title("A Topographic Map of Maunga
> help(filled.contour)
> data(volcano)
> filled.contour(volcano, color = ter
>
> x <- 10*1:nrow(volcano)
> y <- 10*1:ncol(volcano)
> filled.contour(x, y, volcano, color
+ plot.title = title(main = "The
+ xlab = "Meters North", ylab = "
+ plot.axes = { axis(1, seq(100,
+ axis(2, seq(100,
+ key.title = title(main="Height\
+ key.axes = axis(4, seq(90, 190,
> mtext(paste("filled.contour(.) from
+ side = 1, line = 4, adj = 1,
>
R Graphics: Device 2 (ACTIVE)
The Topography of Maunga Whau
Meters West
600
500
400
300
200
100
100
300
500
700
Meters North
Height
(meters)
filled.contour(.) from R version 1.9.0, 2004-04-12
Ventana gráfica
Consola
Ventana de Ayuda
Arguments:
x,y: locations of grid lines at which the values in 'z' are
measured. These must be in ascending order. By default,
equally spaced values from 0 to 1 are used. If 'x' is a
Inicio
Microsoft PowerPoint ...
The R Project for Sta...
RGui
ES
19:11

Empezando a trabajar con R

`help(solve), ?solve, help.search("solve")`	Busca ayuda sobre la función `solve`.
`example(mean)`	Ejemplo de cómo usar la función `mean`
`source("prog.R")`	Ejecuta los comandos de *prog.R*
`sink("sal.lis")`	Guarda la salida de texto en el fichero *sal.lis*
`objects(),ls()`	Lista los objetos de la memoria de usuario

Objetos Básicos

- Similar a cualquier otro lenguaje de programación
  - `logical (TRUE, FALSE, T, F)`
  - `integer`,
  - `numeric (real, single, double)`
  - `complex`
  - `character`
- Se definen al ser asignados y no es necesario definirlos con antelación

```
x<-5;class(x);mode(x);str(x)
x<-5+5i;class(x);mode(x);str(x)
z<-5/x;class(x);mode(x);str(z)
```

- A partir de estos se definen nuevas estructuras: vectores, matrices, listas, data.frames, arrays, time series, …
- Se guardan en memoria y se eliminan con `rm()` o `remove()`

```
remove(list=ls(pat="patron*"))
```

Operadores objetos

- Conversión entre objetos: `as.integer(x), as.character(x), as.tipo(x)`
- Comprobación de objetos: `is.tipo(x)`
- Operadores aritméticos: `+, –, *, /, ^, %%, %/%, ?Arith`
- Funciones aritméticas: `log, log10, log2, log(x, base), exp, sin, cos, tan, sqrt, beta, gamma, choose, factorial, ceiling, floor, signif, round, Arg, Conj, Im, Mod, Re, …`
- Operadores lógicos: `&, &&, |, ||, xor, !, ?Logic`
- Operadores lógicos: `==, <=, >=, >, <, =, !=`
- Especiales: `NA, NaN, Inf, -Inf,…`
- Funciones especiales: `is.na, is.nan, is.finite, is.infinite, na.action, na.omit, na.fail, complete.cases,…`

Vectores

- R es vectorial en sus operaciones y eficiente cuando se usa vectorialmente
- La memoria se reserva al asignar y no es necesaria asignación previa.
- Para asignar específicamente n posiciones de memoria: x<-*tipo*(n).
- Creación básica de vectores:

```
x<-c(1.0,4.5,-9)#combinar en un vector
x<-rep(a,b)     # repetir a, b veces
x<-seq(c,d,by=0.1)# secuencia de c a d.       si
el paso es 1 se abrevia c:d
```

Vectores

- También se obtienen vectores dinámicamente (menos eficiente).
  `x=5;x=c(x,3)`
- Vectores con componentes de distintos tipo se vuelven del tipo más complejo.
- Pueden llevar nombres:
  `x=c(e=exp(1),pi=pi,dospi=2*pi)`
- Y cambiarse cuando sea necesario:
  `names(x)<-c("e","pi","dpi")`
- Todas las operaciones matemáticas funcionan sobre vectores.
- Las funciones lógicas sobre vectores devuelven vectores lógicos (salvo && y ||)

Funciones de vectores

- Funciones básicas:

```
length(x): Longitud del vector
union(x,y) : Union sin repetir
intersect(x,y): Comunes sin repetir
setdiff(x,y): Cuáles de x no están en y.
subset(x,cond_log): Subconjunto del vector que
cumple una condición
any(cond_log): ¿Alguno cumple condición?
all(cond_log): ¿Todos cumplen condición?
```

- Funciones numéricas básicas:

```
sum(): Suma de todo el vector ->número
prod(): Producto ->número
max(): Máximo ->número, carácter
min(): Mínimo ->número, carácter
cumsum(): Vector con suma acumulada ->vector
cumprod(): Vector con producto acumulado
```

Funciones de vectores

- Funciones estadística básicas:

```
mean(x): Media ->número
var(x): Cuasivarianza -> número
sd(x): cuasidesviación típica -> número
median(x): Mediana  -> número
range(x): Vector de dim 2 con mín y máx.
quantile(x): Cuantiles  -> vector
mad(x): Desviación Absoluta Mediana -> número
IQR(x): Rango intercuartilico
cor(x,y): Correlación -> número
cov(x,y): Covarianza -> número
ecdf(x): Función de distribución -> Función
```

Vectores. Indexado

- `x[i]` hace referencia al elemento i-ésimo
- `x[1:5]` subvector de x desde 1 a 5
- `x[-1]` subvector de x sin el primer elemento
- `x[-c(1,3)]` subvector de x sin el 1 y el 3
- `x[c(F,T)]` subvector de elementos pares
- `x[y<0]` subvector de x tal que y es negativo
- `x & y` devuelve vector lógico
- `x && y` devuelve un solo valor
- `factor`. Vector de enteros para agrupar niveles.
- `cut`. Función para dividir en niveles

Vectores. Indexado. Ejemplos

```
x<-1:20; y<-sin(pi*x/10);lo<-rep(c(F,T),10)
sum(x) # suma de 1 a 20 = 210
sum(x[lo]) # suma de pares.
      sum(x[x%%2==0])
x[-c(1,20)] # de 2 a 19 = 2:19
x[y>0] #de 1:9
f<-cut(y,breaks=4,labels=letters[1:4]) #Crea un
  factor con 4 niveles llamados a, b, c y d.
y[f=="a"] # valores del factor a
as.numeric(f) # enteros asociados a niveles
split(x,f) # divide el vector x en grupos definidos
  por el factor
```

Texto

- character: Para escribir texto se usa la función `print()` o `cat()`
- Varios vectores predefinidos:
  `LETTERS, letters` 26 letras alfabeto americano.
- Para formatear texto se usan los caracteres especiales de ESC (como en C):
  `\n: nueva linea, \t: tabulador, \b: Espacio atrás, \": Comillas, \\: Barra invertida, \a: Alerta`
- `paste() #` Concatena texto y variables
- `substr()` # Devuelve subcadenas de una dada
- `strsplit()` # Separa cadenas mediante un carácter
- `format()` # Permite escribir valores numéricos con formato.
- `pmatch()` # Compara una cadena con un vector de caracteres.
- Importante! Consultar documentación sobre regexp cuando se busque texto (por ej. `ls()`)
- En gráficos se pueden escribir formulas matemáticas con `expression()` y siguiendo el pseudolenguaje `plotmath`.
  Probar `demo(plotmath)` y `demo(Hershey)`

Matrices y Arrays.

- Vector con atributos de dimensión.
  `x<-1:20;dim(x)<-c(10,2)# por columnas`
- `matrix()`: 2 dimensiones, `nrow()`,`ncol()`,`rownames()`, `colnames()`, `dimnames()`(nº filas y columnas, nombres).
- `array()` Crea arrays (más de 2 dimensiones)
- `x[,1]` primera columna de x.
- `x[,c(1,3)]` columnas 1 y 3
- `x[c(F,T),]` sólo filas pares
- `%*%` producto matricial
- %o% producto exterior, alias de `outer(a,b,"*")`
- %x% producto de Kronecker

Matrices y Arrays.

- `t()`: Transponer
- `aperm()`: Permutar dimensiones
- `det(A)`: Determinante de A
- `diag(A)`: Extrae la diagonal de A
- `diag(n)`: Matriz identidad de dim n
- `lower.tri(A)`: Triangular inferior de A
- `upper.tri(A)`: Triangular superior de A
- `cbind(A,B)`: Crear matriz uniendo columnas
- `rbind(A,B)`: Crear matriz uniendo filas
- `crossprod(X,Y)`: Producto cruzado X^tY
- `kappa(A)`: $||A||||A^{-1}||$

Matrices y Arrays.

- `solve(A,B):` AX=B o solve(A)=Inversa
- `svd(A):` Singular Value Decomposition de A=UDV
- `eigen(A):` Autovalores y autovectores de A
- `chol(A):` Descomposición de Choleski
- `apply(A,MARGIN,FUN):` Aplica FUN sobre el margen MARGIN

```
x=1:20;y=5*x; solve(crossprod(x))%*%t(x)%*%y
```

¡OJO! . Ejemplo: Recentrar una matriz

```
X=matrix(1:40,nrow=10,ncol=4);mX=apply(X,2,mean)
  Xcen=X-mX # Mal
unos=rep(1,nrow(X))
Xcen=X-outer(unos,mX) # Versión clásica
Xcen=sweep(X,MARGIN=2,STATS=mean,FUN="-") # fácil
```

Factores

- Factor: Vector de datos cualitativos.
  `f=factor(rep(c("Enfermo","Sano"),c(3,4)))`
- Internamente los niveles se guardan como enteros:
  `as.numeric(f);table(f)`
- Factor Ordenado: Vector de datos cualitativos con niveles ordenados.
  `f=ordered(rep(c("Enfermo","Sano"),c(3,4)),levels=c("Sano","Enfermo"))`
- `gl(n,k)`: Genera n factores con k replicas.
- `levels(f)`: Determina o cambia niveles
- `nlevels(f)`: Número de niveles

Listas y data frames

- Lista: Colección ordenada de otros objetos de cualquier clase.
  `Lst<-list("x"=x, "y"=y)`
- Accesos a las componentes mediante:
  `Lst$x, Lst[[1]], Lst[["x"]]`
- `names(Lst) #` Nombre de las componentes
- La mayoría de funciones de R devuelven listas donde se incluyen todos los resultados.
- Las nuevas clases o estructuras suelen ser listas con un formato determinado.

Data Frames

- Data frame: Lista “matriciada” de variables al estilo hoja de calculo. Columnas=Variables, Filas=Casos. Estructura básica de un conjunto de datos.
- `x<-1:20;y<-21:40;`
  `df<-data.frame(v1=x,v2=y)`
- El comando `edit` permite editar el data frame.
  `df=edit(df)`
- El comando `summary` resume la información del data frame apropiadamente.
- El comando `data` incorpora en memoria un data frame.
  `data(iris);df=iris`

Vuelta al indexado.

- El operador "[" extrae componentes con el mismo tipo de donde fueron extraidos
```
dfnew=df[df[,1]>0,] # data frame
lista=Lst[1]  # 1º de la lista (lista)
lista=Lst[[1]]# 1º de la lista (original)
lista=Lst$x   # componente x
```
- La combinación del operador "[" con data.frames obtiene subconjuntos y es de los procedimientos más usados.
- Siempre debe testearse el tipo de los datos devueltos. (fuente de errores).
- `mean(iris);mean(as.integer(iris[[5]]))`.
- El comando `subset` es otra opción muy recomendable.
- `subset(iris,Species %in% c("setosa", "versicolor")`.
- ¿Qué produce esta línea?
```
c("Yes", "No")[rep(1:2, 5)]
```

Funciones de objetos S3

- `summary()` # resume información del objeto
- `print()` # escribe el objeto – `methods(print)`
- `plot()` # dibuja el objeto – `methods(plot)`
- `coef()` # devuelve los coeficientes de un objeto (si tiene sentido).
- `attributes(),attr()` # Atributos del objeto.
- Familia `apply`. Aplican funciones a objetos
  `apply(x,dim,func)` # a matrices `lapply(x,func)` # a una lista
  `sapply(x,func)` # versión simple de lapply
  `tapply(x,f,func)` # por grupos de factores
  `mapply()` # version multivariante de sapply

Formulas

- También son objetos las formulas que se escriben en lenguaje simbólico.
- `y~model` # significa y depende de model.

  Modelo

  `x+z` # modelo aditivo de x y z

  `x*z` # modelo aditivo + termino cruzado de x y z

  `(x+z)^2` # Todos los términos hasta grado 2. Idem

  `x-1` # el signo menos quita el elemento independiente.

  `I(x-log(z))` # identidad. Para crear términos tal cual.

  `x:z` # término cruzado de x y z.

  `x %in% z` # x anidado en z.

  `poly(x,3)` # polinomios ortogonales hasta grado 3

  `x|z` # significa x condicionado en z

  También se pueden incluir funciones numéricas.

Leyendo datos formato texto.

- El comando básico para leer ficheros es `read.table("u:\\dir\\fich", sep="", header=TRUE)`
- Parámetros: `sep, header, quote, dec, row.names, col.names, skip.`
- Alias: `read.csv(), read.delim(), read.csv2(), read.delim2(), read.fwf() # formato fijo`
- Devuelve un data.frame con nombres de variables en función de `header` y `col.names`.

Leyendo datos de otras fuentes

- El paquete `foreign` permite leer y escribir en formatos de Arff, DBF, SPSS, Minitab, Stata y Epi Info.
- ```
  library(foreign)
  read.spss("U:\\path\\dir\\fichero.sav", use.value.labels=FALSE, to.data.frame=FALSE)
  ```
- Devuelve data.frame o lista de variables según el valor de `to.data.frame`.
- Hay paquetes para la lectura de datos de fuentes especificas → NetCDF,…
  ```

Importando datos de Excel

- El paquete `xlsReadWrite` permite leer y escribir en formato de Excel nativo.
- `read.xls( file, colNames = TRUE, sheet = 1, type = "data.frame", from = 1, rowNames = NA, colClasses = NA, checkNames = TRUE, dateTimeAs = "numeric", stringsAsFactors = default.stringsAsFactors())`
- El formato CSV sigue siendo el más sencillo para importar los datos de Excel.

Importando datos de Excel (o cualquier otra base de datos)

- El paquete RODBC permite leer, escribir y cambiar estructuras de bases de datos ODBC.
- ```
  conn<-odbcConnectExcel("Enerxia.xls")
  sqlTables(conn) #Tablas en fichero
  sqlColumns(conn,"Hoja2") # Columnas
  datos<-sqlFetch(conn,"Hoja2") #datos
  odbcClose(conn) # Fin uso fichero
  ```
- Otras bases de datos admitidas: Access, Dbase, MySQL, mSQL, etc. Cualquiera que admita el lenguaje SQL para establecer consultas.
  `odbcDriverConnect("")`
  ```

Exportando o salvando datos

- Se suelen exportar data.frames
- `write.table()` escribe texto plano o fijo
- `write.spss()` del paquete `foreign` escribe ficheros SPSS
- Los objetos en memoria de datos de R se guardan con la función `save.image()` o `save()`. Escribe un fichero `.RData` con los objetos. La función `load.image()` hace la función contraria.
- Con el paquete `xlsReadWrite` se pueden escribir ficheros Excel sencillos.
- Con el paquete RODBC se pueden escribir en bases de datos con soporte SQL.
- `getwd()` y `setwd()` # Devuelven o establecen el directorio de trabajo.

Análisis descriptivo

- Variables cuantitativas: véanse las funciones básicas de vectores.
- Variables cualitativas: Tablas de contingencia:
  ```
  t<-table(Month, cut(Wind, quantile(Wind)))
  ftable(Titanic,row.vars=1:3)
  ```
- Todas las funciones de vectores pueden ser aplicadas ahora.
  ```
  fr1<-apply(t,1,sum)/sum(t) # Frec. por filas
  fr2<-apply(t,2,sum)/sum(t) # Frec. por columnas
  ```
- Aunque esto se hace más fácilmente:
  ```
  t2=prop.table(t);fr1=margin.table(t2,1);
  fr2=margin.table(t2,2)
  ```
- `chisq.test(t)` # Test chi-cuadrado de asociación
- `plot(t)` # Dibujo de mosaico de la tabla

Gráficos

- R dispone de muchas funciones gráficas
  `demo(graphics)` # ejemplos de varios gráficos
- Las funciones se dividen en:
  de 1º nivel: Crean ventanas y establecen coordenadas en el gráfico.
  de 2º nivel: Crean elementos gráficos en ventanas gráficas ya abiertas y con coordenadas establecidas.
  Funciones trellis (lattice): Crean funciones multipanel y no se pueden modificar.
- Los modificadores de opciones gráficas suelen tener el mismo nombre en todas las funciones gráficas y la función `par()` permite establecer valores por defecto. Las opciones `mfrow()` y `mfcol()` permiten dividir la ventana en varios gráficos.
  `split.screen()` o `layout()` son otras alternativas.
- Las gráficas se pueden copiar en distintos formatos desde menú o desde línea de comandos (más opciones).

Gráficos

- Muchos dispositivos nos permiten crear gráficas en distintos formatos. –`help(Devices)`—
- Gráficos en pantalla en cada sistema: `windows(), X11(), macintosh()`
- Ficheros postscript: `postscript()`
- Gráfico en PDF: pdf()
- Otros dispositivos gráficos: `jpeg(), bmp(), wmf(), png(), xfig(), pictex()`
- `dev.cur(), dev.list(), dev.next(which = dev.cur()), dev.prev(which = dev.cur()), dev.off(which = dev.cur()), dev.set(which = dev.next()), graphics.off()`

Guardando gráficas.

- Desde la propia ventana del gráfico en Windows se puede guardar el gráfico como metafile o postscript o copiar al portapapeles como metafile o bitmap.
- `pdf(file = "f1.pdf", width = 8, height = 10) #` Crea fichero pdf para rutinas gráficas
- `dev.off()` # Cierra el fichero abierto
- `dev.copy2eps()` # Copia la ventana gráfica a un eps (guardado como Rplot.eps en directorio de R)
- `dev.copy()` # Copia gráficos entre "devices"
- **`devices`**: `bmp(), jpg(), png(), pdf(), xfig(), pictex(), postscript(), windows(), bitmap()#` Consultar `devices` en paquete `grDevices`

par()

- `par()` devuelve los valores configurables en los gráficos y permite establecer nuevos valores .
- `adj` : [0,1] Establece alineación. 0=izq. 1=dcha, 0.5=centro.
- `bty` cambia tipo de borde `"o","l","7","c","u","]","n"`.
- `col.axis, col.lab, col.main, col.sub` cambia el color de ejes, etiquetas, principal y sub-principal.
- `lty` cambia el tipo de línea y `xlim, ylim` establecen la escala.
- `pch` establece el tipo de símbolo y `xlog, ylog` establecen escala logarítmica en los ejes.
- Los títulos se añaden con `main`, `sub`, `xlab` e `ylab`.

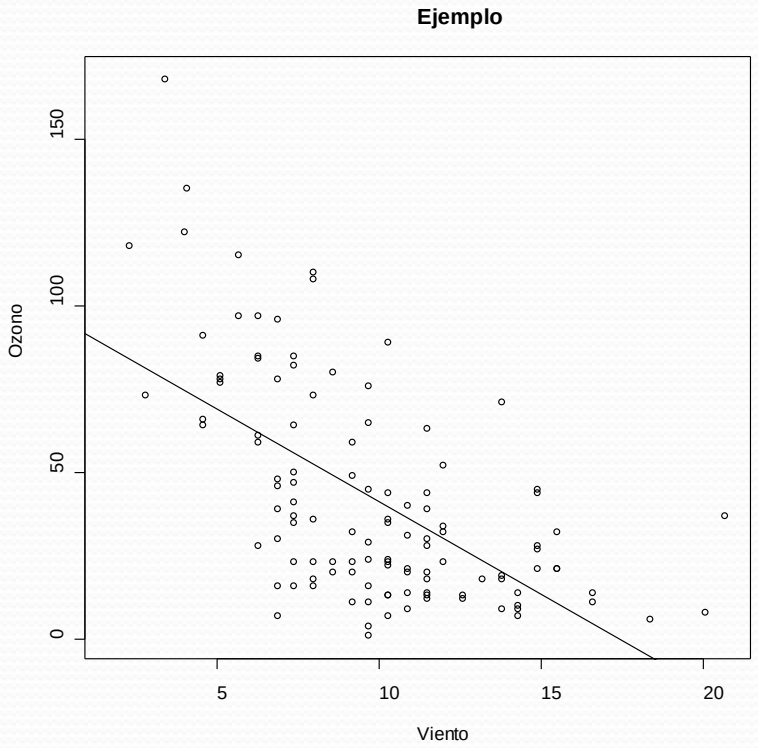
plot()

- `plot()` es la función genérica de dibujo de 1º nivel. En su versión más simple dibuja un gráfico de puntos, líneas o dispersión.

- Modificadores
- `type` cambia como se dibuja. `"l"=línea, "p"=puntos, "b"=puntos y línea, "n"=no dibuja, "h"=líneas verticales, "s"=escalera`
- `col` cambia el color y `lwd` cambia el tamaño de la línea.
- `lty` cambia el tipo de línea y `xlim, ylim` establecen la escala.
- `pch` establece el tipo de simbolo y `xlog, ylog` establecen escala logarítmica en los ejes.
- Los títulos se añaden con `main`, `sub`, `xlab` e `ylab`.

Ejemplo de plot

```
data(airquality)
attach(airquality)
plot(Wind,Ozone,
  type="p",
  main="Ejemplo",
  xlab="Viento",
  ylab="Ozono")
abline(a=96.873,b=-
  5.551) # Función de
  2º nivel que añade
  lineas
```

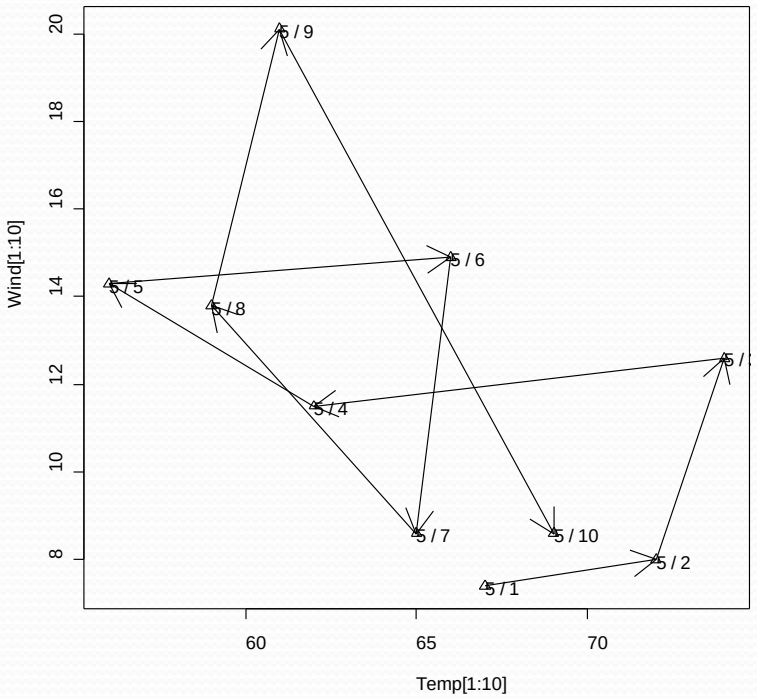


Añadir elementos gráficos

- `points()` # Añade puntos
- `lines()` # Añade líneas
- `abline()` # Añade una línea paramétrica
- `legend()` # Añade una leyenda
- `mtext()` # Escribe texto en los márgenes
- `text()` # Escribe texto dentro ventana en posiciones x,y.
- `arrows(), segments(), polygon()` # Se dibujan flechas, segmentos y polígonos
- `title()` # Añade títulos
- `identify()` y `locator()` # Permiten identificar puntos del gráfico o coordenadas

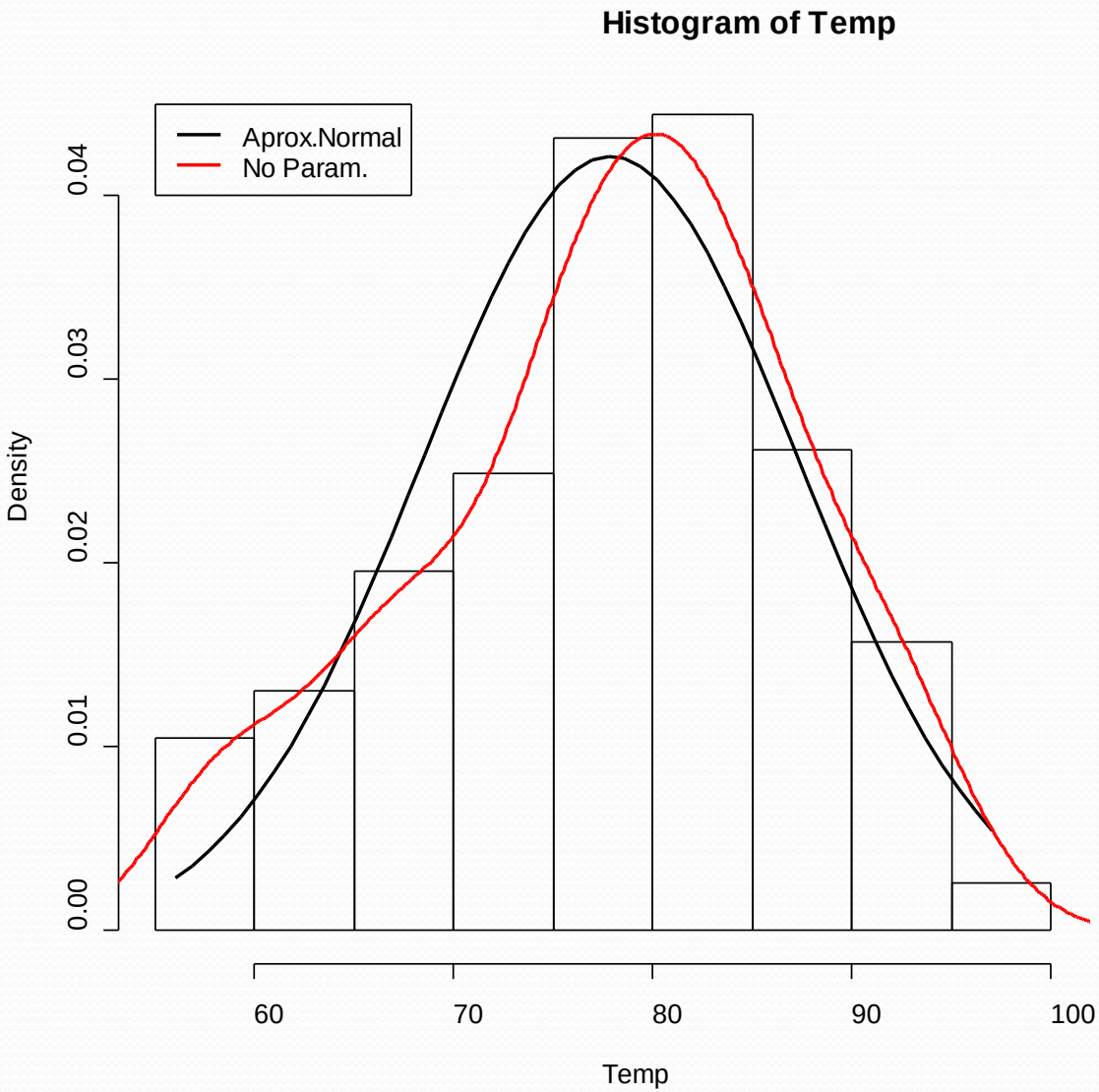
Ejemplo de otros elementos

```
plot(Temp[1:10],
  Wind[1:10],pch=2)
arrows(Temp[1:9],
  Wind[1:9],
  Temp[2:10],
  Wind[2:10])
text(Temp[1:10],
  Wind[1:10],
  paste(Month[1:10],
  "/",Day[1:10]))
```

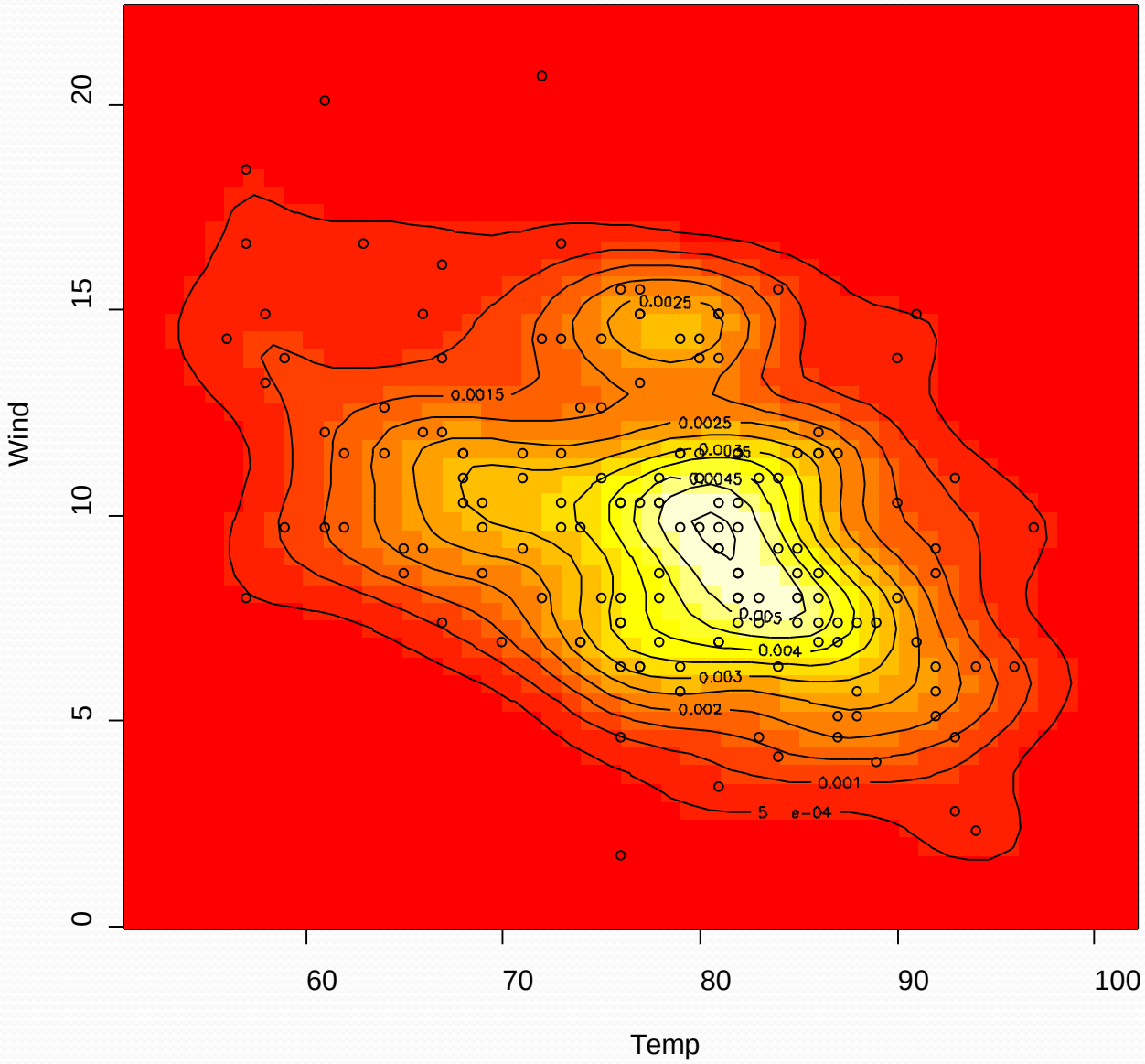


Más funciones gráficas

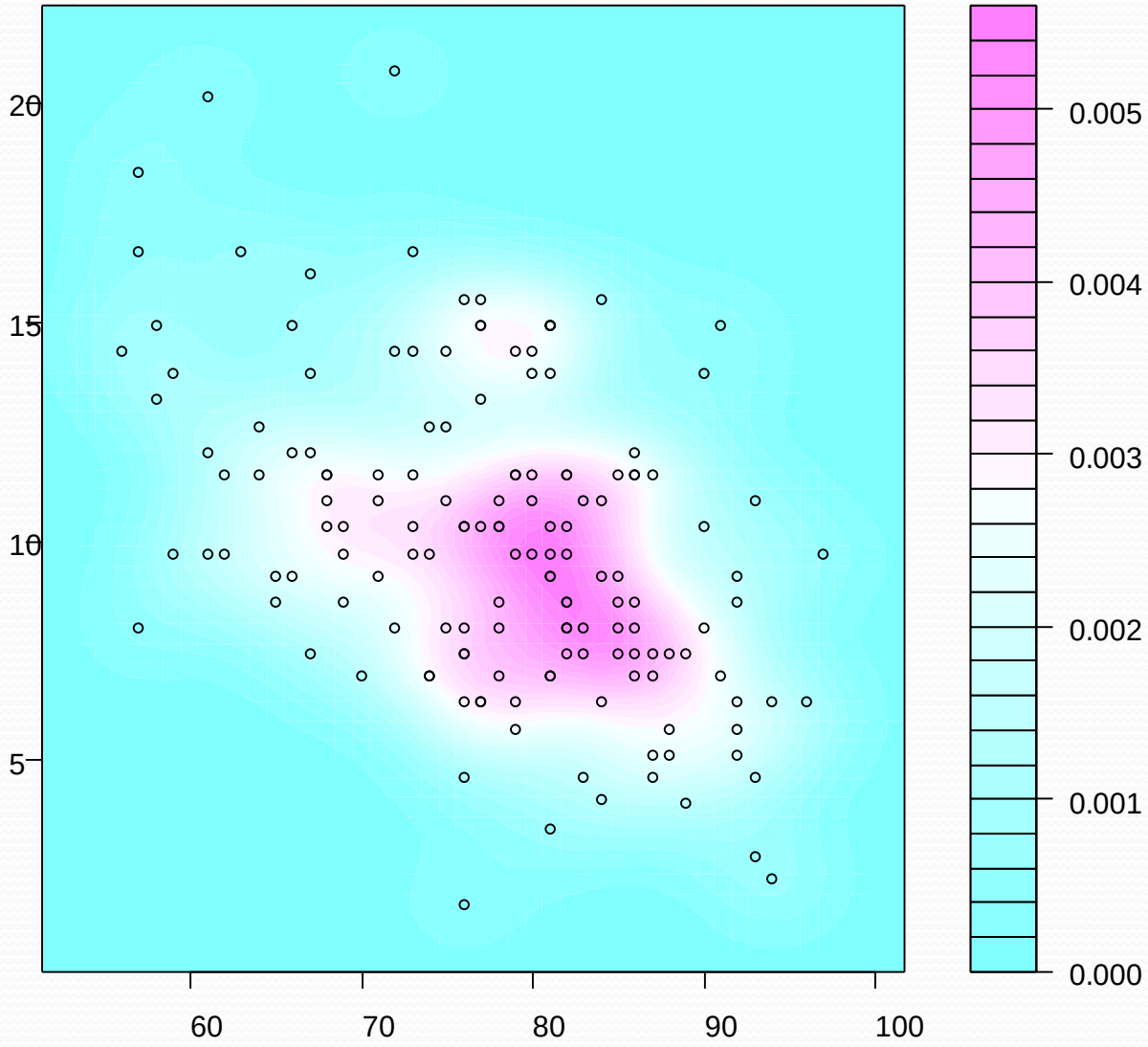
- `symbols()` # Dibuja variables con simbolos (circulos(1), rectangulos(2), estrellas(3+), boxplots(5), termometros(3/4),…)
- `pie()` # Clásico gráfico de tarta
- `boxplot()` # Su nombre lo dice todo
- `hist()` # Histogramas
- `pairs()` # Graficos x-y multivariantes
- `matplot()` # Gráfico de x con una matriz de y's
- `coplot()` # Gráficos x-y condicionados
- `plot(density(),type="l")` # Estimación no paramétrica de la densidad.
- `barplot()` # Clásico gráfico de barras
- `image()` # Gráfico de niveles de una matriz
- `contour()` y `filled.contour()` # líneas de contorno
- `persp()` # Gráfico 3-D



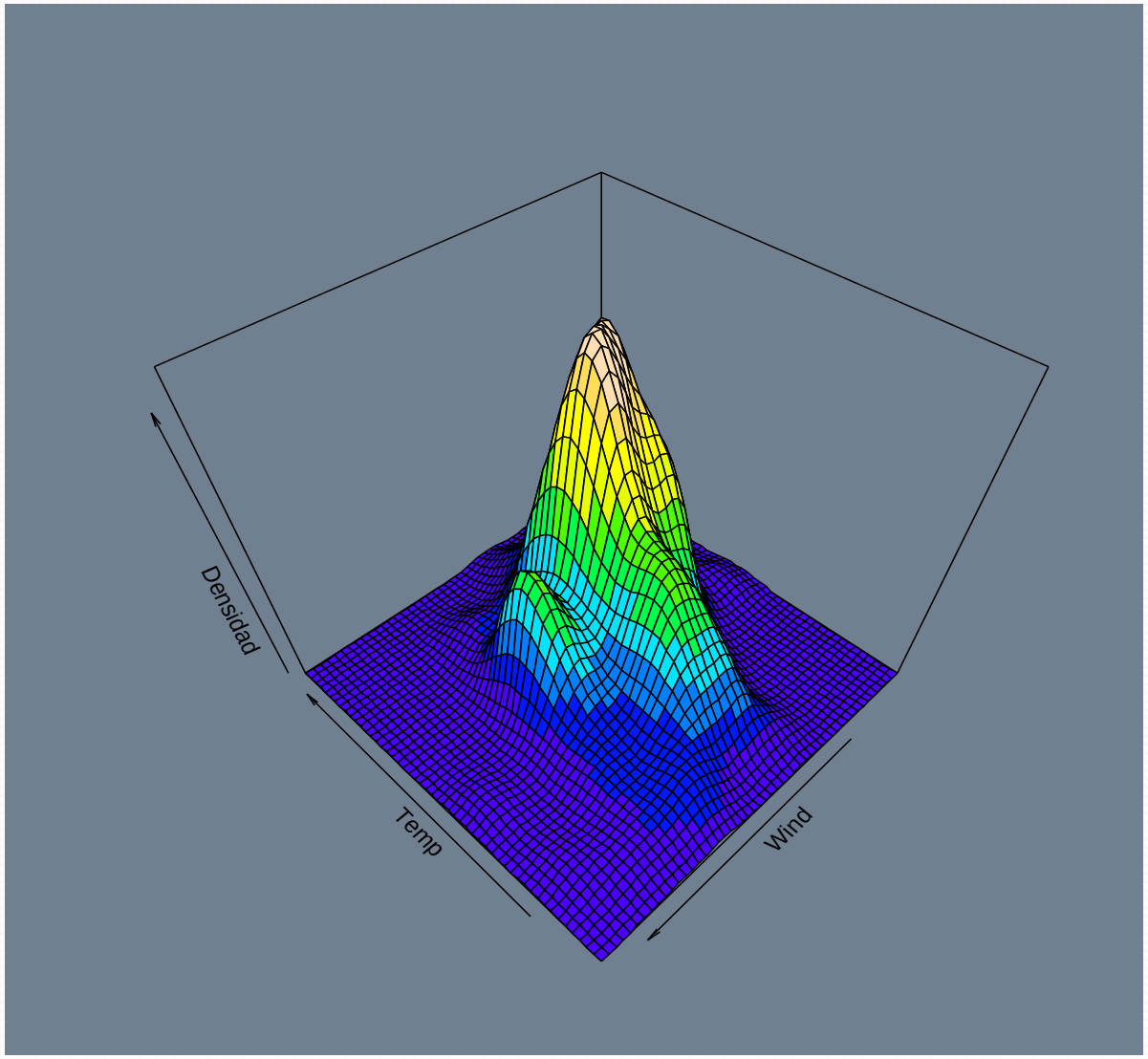

```
hist(Temp,freq=FALSE); x<-seq(min(Temp),max(Temp),length=50)
lines(x,dnorm(x,mean=mean(Temp),sd=sd(Temp)),lwd=2)
lines(density(Temp),lwd=2,col=2)
legend(55,0.045,legend=c("Aprox.Normal","No Param."),lwd=2,col=1:2)
```



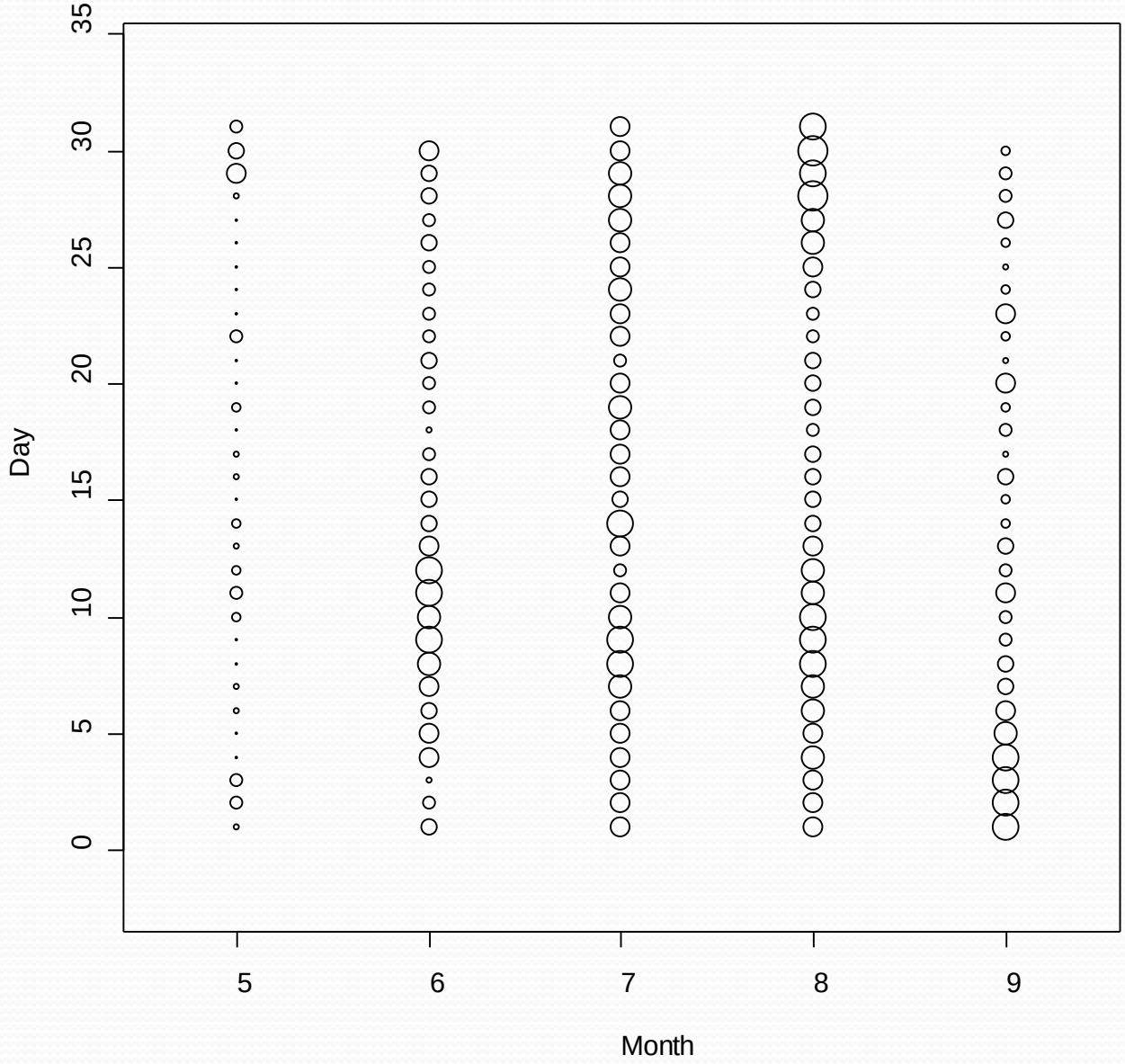

```
est<-bkde2D(cbind(Temp,Wind),c(bw.nrd0(Temp),bw.nrd0(Wind)))
image(est$x1,est$x2,est$fhat,xlab="Temp",ylab="Wind")
points(Temp,Wind);contour(x=est$x1,y=est$x2,est$fhat,add=T)
```



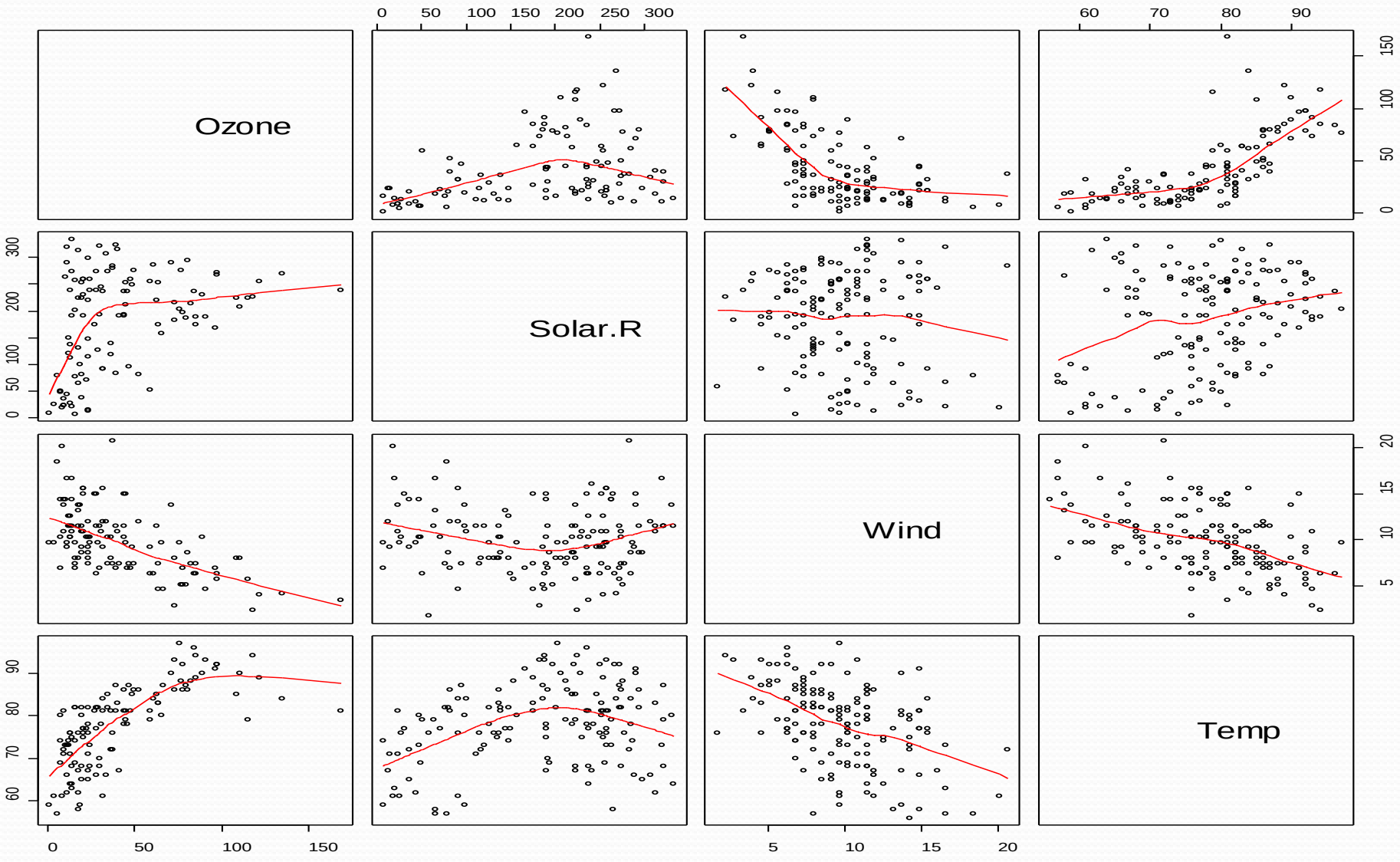

```
filled.contour(x=est$x1,y=est$x2,est$fhat,
       plot.axes={axis(1);axis(2);points(Temp,Wind)})
```


Densidad
Temp
Wind

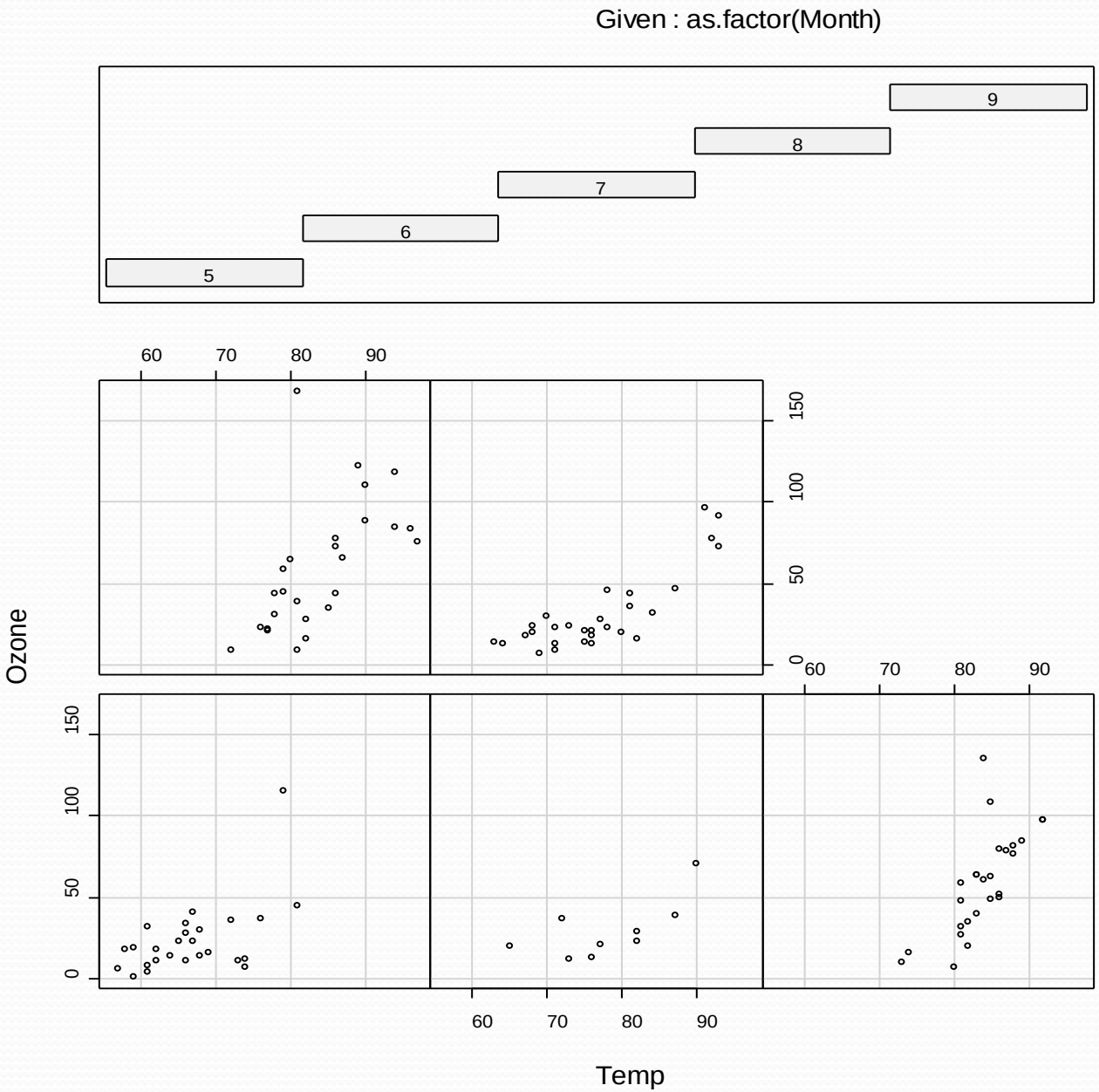
```
boxplot(Temp~Month,xlab="Mes",ylab="Temperatura")
```




```
symbols(Month,Day,circle=(Temp-min(Temp))/500,inches=FALSE)
```




```
pairs(airquality[,1:4],panel=panel.smooth)
```




```
coplot(Ozone~Temp|as.factor(Month))
```

Funciones gráficas Lattice

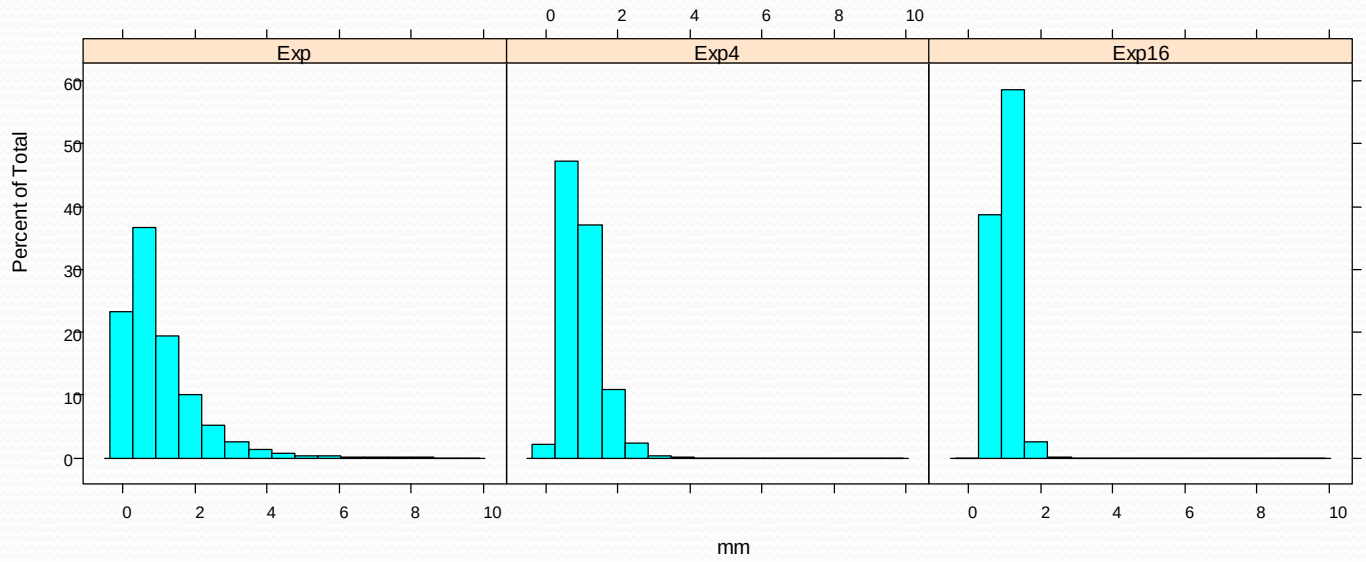
- El paquete `lattice` tiene muchas otras funciones gráficas que permite condicionamientos estilo panel.
- Las opciones gráficas de este paquete son, en general, menos intuitivas pero más potentes.
- Con este paquete se pueden actualizar los objetos gráficos cambiando alguna opción sin necesidad de re-ejecutar el código. Update(trellis.last.object(),aspect="iso")
- En general la llamada a las funciones del paquete lattice se hacen en formato fórmula `y~x|f*g`
- El paquete `latticeExtra` tiene alguna función más.

Funciones gráficas Lattice

- `densityplot()` # Estimación de la densidad
- `xyplot()` # Gráfico x-y |z
- `dotplot()` # Gráfico x-y |z
- `barchart()` # Como `barplot`
- `bwplot()` # Como `boxplot`
- `levelplot()` # Gráfico de niveles similar a `image()`
- `wireframe()` # Gráfico 3-D similar a `persp()`
- `cloud()` # Puntos en 3-D
- `splom()` # Como `pairs()`
- `contourplot()` # Como `contour()`
- `mapplot()` # Mapa, como `map()`

Ejemplo Lattice

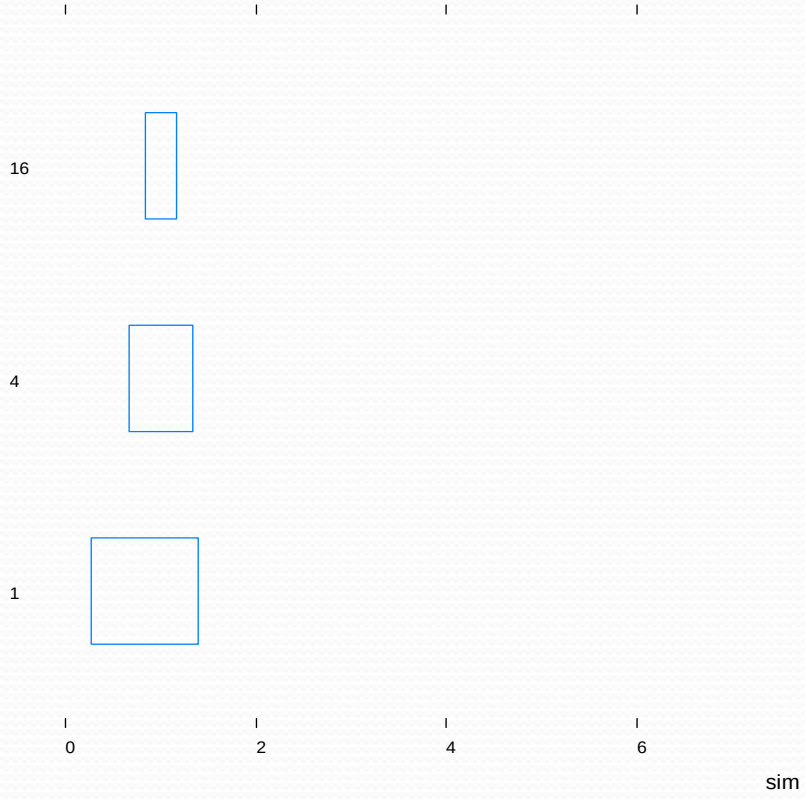
```
datos=matrix(rexp(10000*16),ncol=16)
mns=cbind(mns=datos[,1],m4=apply(datos[,1:4],1,mean),m1
  6=apply(datos,1,mean))
library(lattice)
histogram(~mm|sz,data=data.frame(mm=c(mns),sz=gl(3,1000
  0,labels=c("Exp","Exp4","Exp16"))),layout=c(3,1))
```



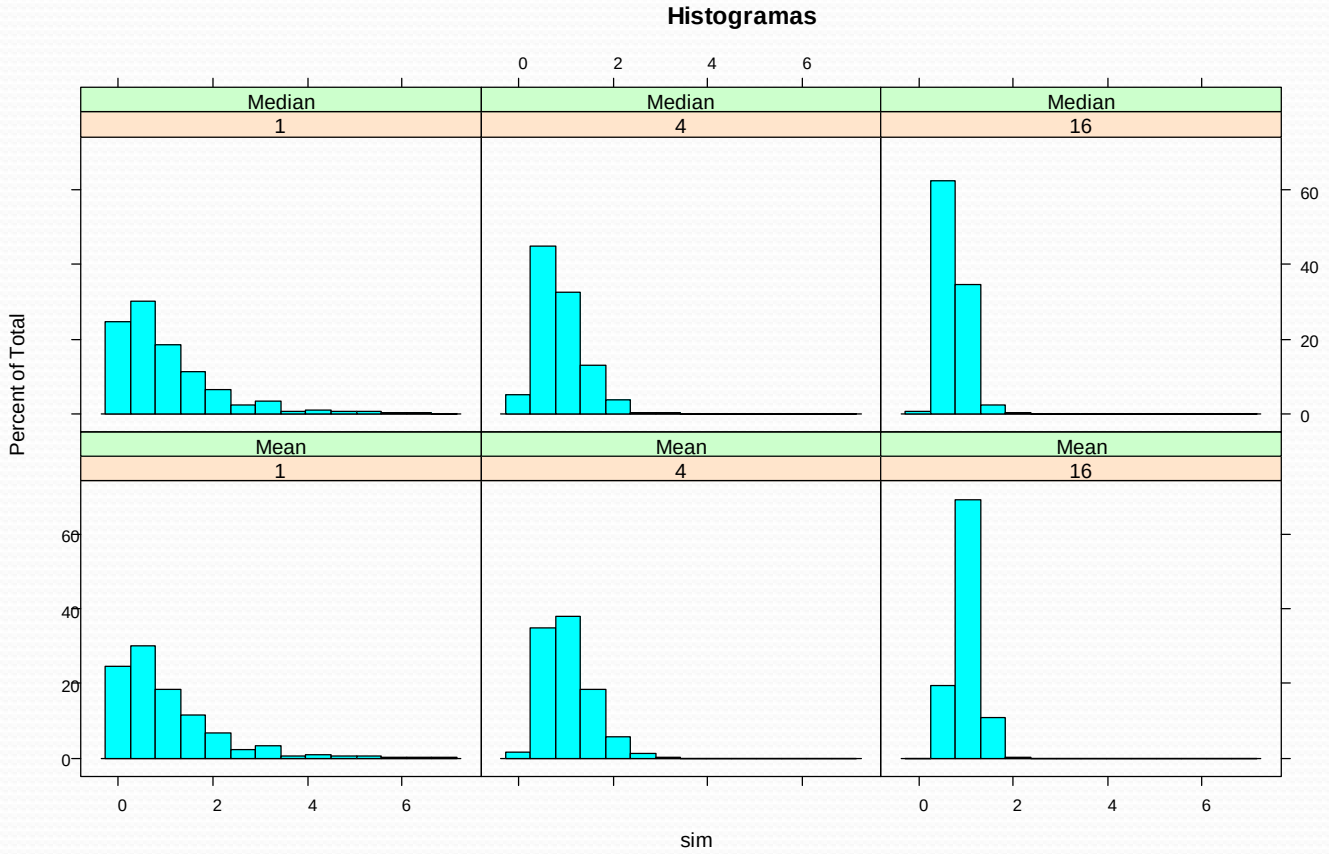
Ejemplo Lattice II

```
datos=matrix(rexp(1000*16),ncol=16)
mns=cbind(mns=datos[,1],m4=apply(datos[,1:4],1,mean),m1
  6=apply(datos,1,mean))
med=cbind(me=datos[,1],me4=apply(datos[,1:4],1,median),
  me16=apply(datos,1,median))
library(lattice)
todo <- data.frame(sim = c(mns, med),sz = gl(3, 1000,
  len = 6000, labels = c("1", "4", "16")),type = gl(2,
  3000, labels = c("Mean", "Median")))
bwplot(sz~sim|type,data=todo)
histogram(~ sim | sz * type, data = todo, layout = c(3,
  2), main = "Histogramas")
```

Ejemplo Lattice II

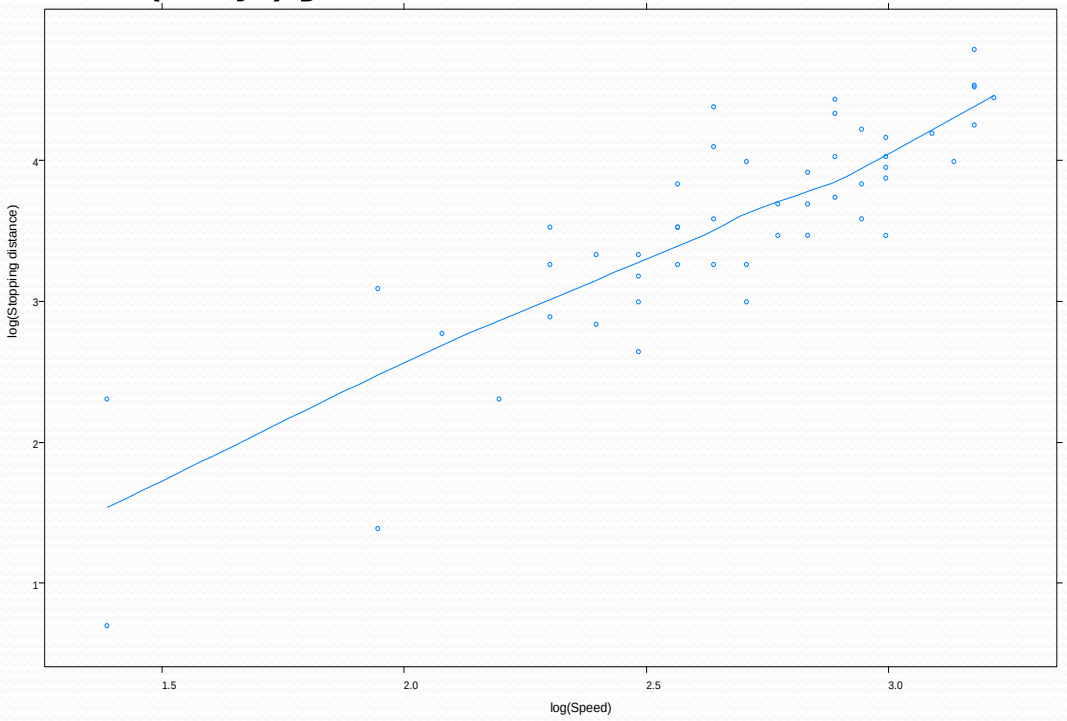


Ejemplo Lattice II



Ejemplo Lattice

```
xyplot(log(dist)~log(speed), data=cars,
  xlab="log(Velocidad)",ylab="log(Dist. frenado)",
  panel=function(x,y){panel.xyplot(x,y);
  panel.loess(x,y)})
```



Otras librerías gráficas

- library(plotrix)
- library(rgl)
- library(vcd)
- library(misc3d)
- library(klaR)
- library(maps)
- … Task Views (Graphics)

Otras librerías gráficas

```
library(rgl);data(volcano);x=1:nrow(volcano)
y=1:ncol(volcano);open3d();zlim=range(volcano)
zl=(zlim[2]-zlim[1])+1;
clut=terrain.colors(zl)
col=clut[(volcano-zlim[1])+1]
surface3d(x,y,volcano,col=col)
x=seq(-3,3,len=51);y=seq(-3,3,len=51)
z=outer(x,y,function(x,y){dnorm(x)*dnorm(y)})
persp3d(x,y,z,col="green")
lines3d(x=0,y,z=dnorm(0)*dnorm(y),col="blue")
rgl.postscript("prueba",fmt="ps")
```