Ciclo de clases en bioinformática: Principios de R

Leonardo Collado Torres
lcollado@ibt.unam.mx
Licenciado en Ciencias Genómicas
www.lcg.unam.mx/~lcollado/

Instituto de Biotecnología (IBT) de la UNAM y Winter Genomics (WG)

Octubre - Noviembre, 2009

Graficas avanzadas con R

Graficas avanzadas con R

8 Sigue . . .

Mientras. . .

- Por favor instalen los siguientes paquetes si no lo han hecho todavía.

```
> install.packages(c("lattice", "plotrix",
+     "car"))
```

- Estos otros paquetes son útiles para ejemplos:

```
> install.packages(c("mlmRev", "DAAG"))
```

- Usen los espejos de USA (WA) o USA (CA 1).

Ejercicios

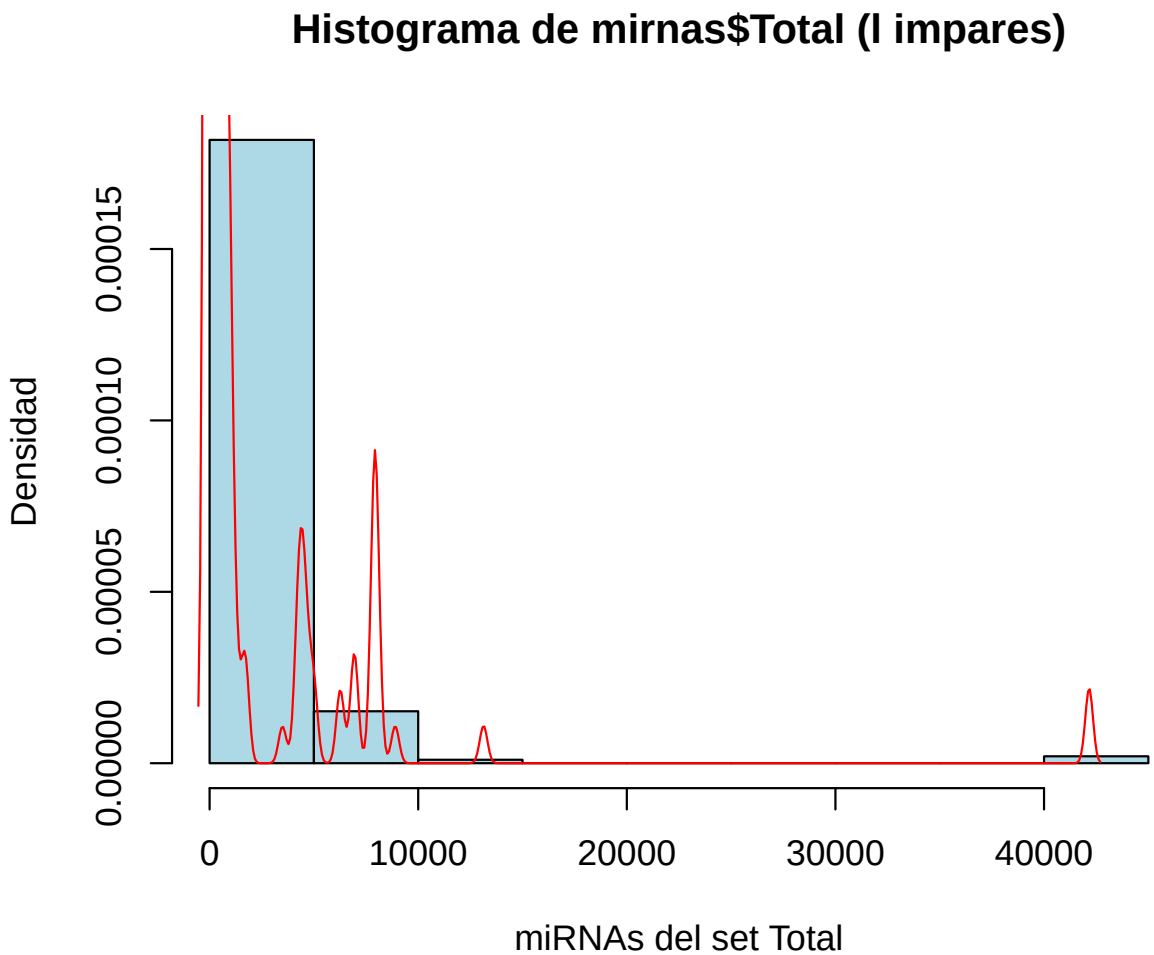
- Usando el objeto `mirnas`, guarden en un objeto[1] los valores de la columna Total de las líneas impares.
- Usando ese objeto, hagan un histograma con la línea de densidad. Póngale un título y nombres a los ejes.
- Con el mismo objeto, hagan un boxplot. Ponganle un título y nombres a los ejes.
  ¿Qué pueden concluir?
- Exploren gráficamente si esos datos se distribuyen como normal usando . . .

[1] Si no quieren está bien, es solo para evitar escribir más código.

Histograma con densidad

```
> mirnas <- read.csv("http://www.lcg.unam.mx/~lcollado/E/data/mirnas.csv"
+     header = T)
> datos <- mirnas$Total[rep(c(TRUE,
+     FALSE), nrow(mirnas)/2)]
> hist(datos, col = "light blue",
+     main = "Histograma de mirnas$Total (1 impares)",
+     ylab = "Densidad", xlab = "miRNAs del set Total",
+     prob = T)
> lines(density(datos), col = "red")
```

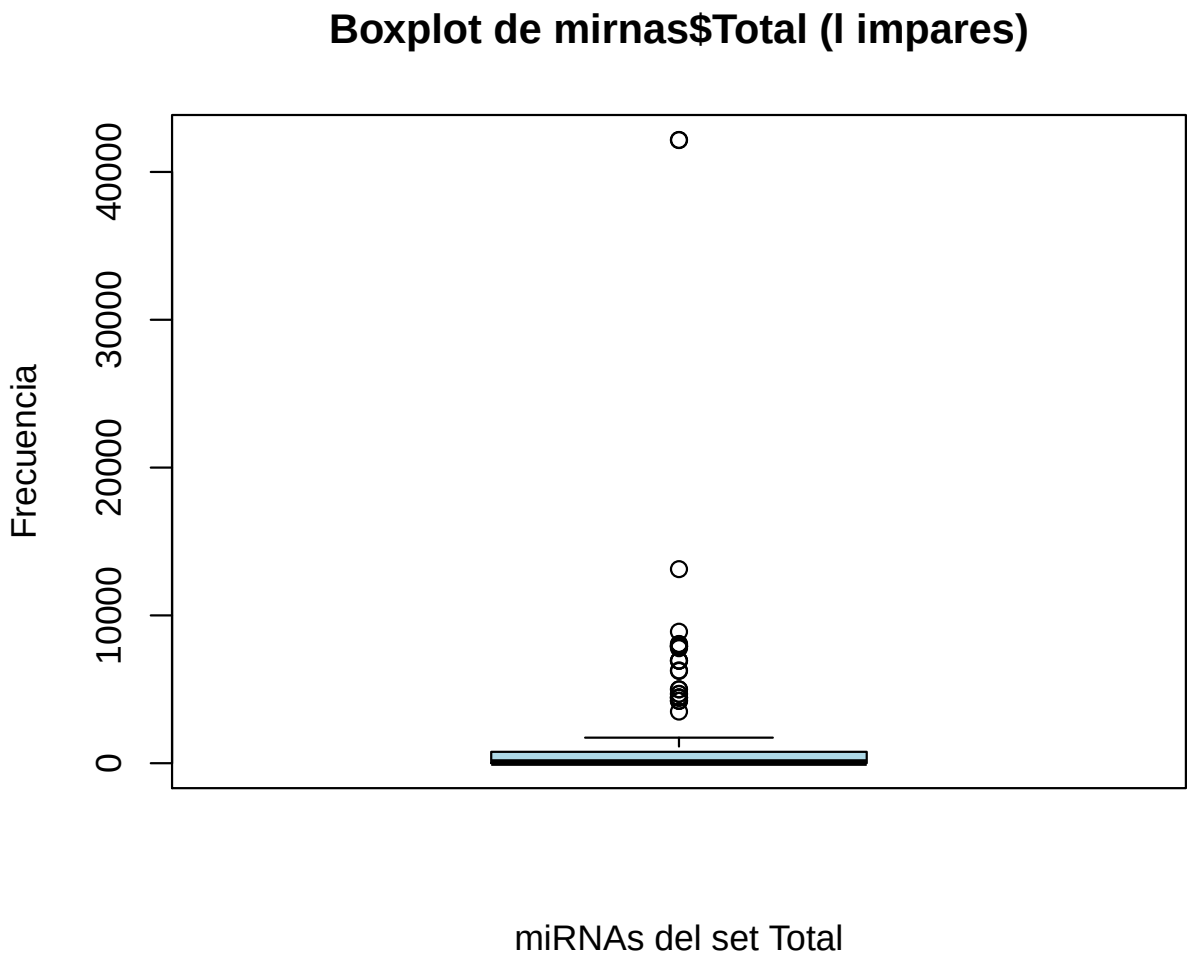
Histograma con densidad



Boxplot

```
> boxplot(datos, main = "Boxplot de mirnas$Total (1 impares)",
+     xlab = "miRNAs del set Total",
+     ylab = "Frecuencia", col = "light blue")
```

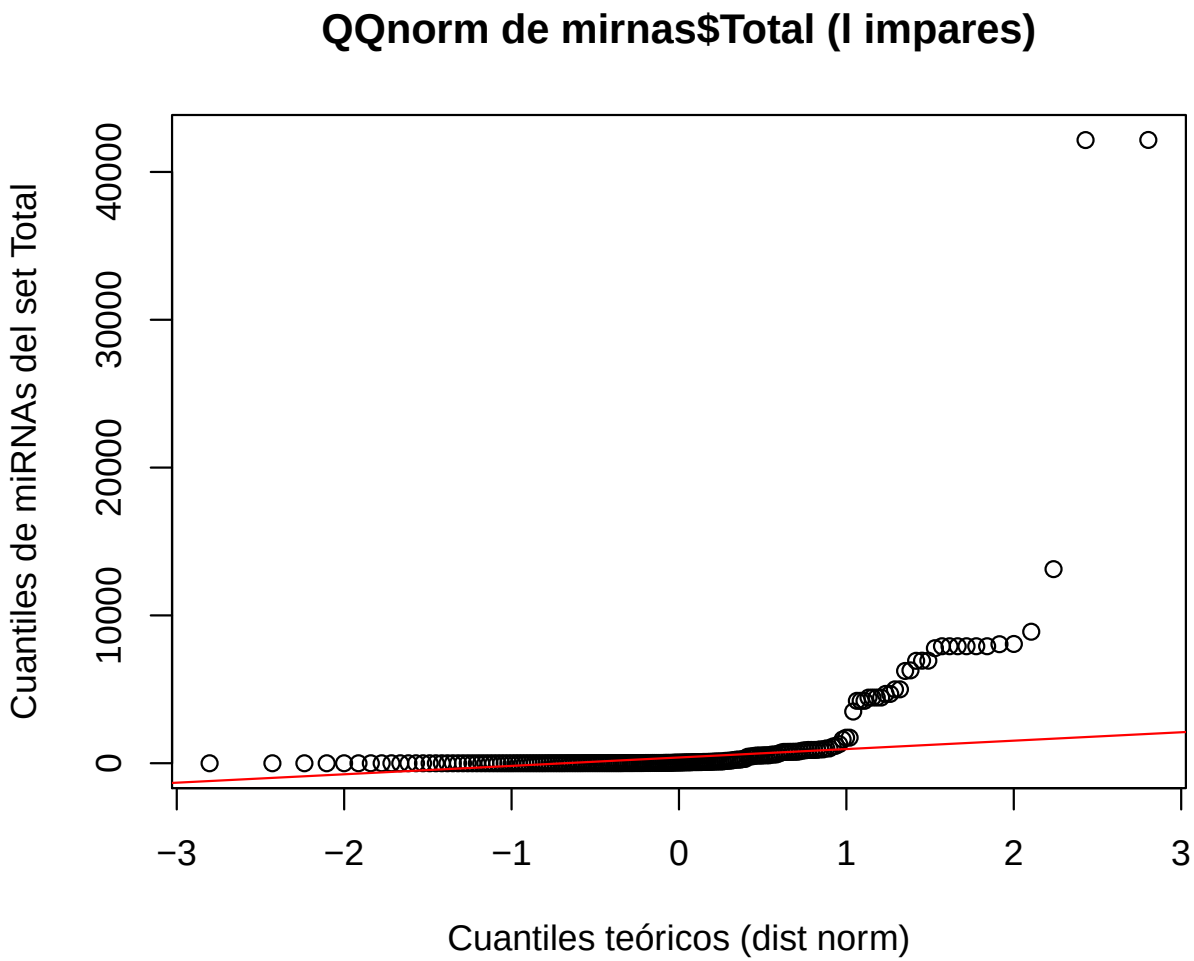
Boxplot


Boxplot de mirnas$Total (I impares)
Frecuencia
0
10000
20000
30000
40000
miRNAs del set Total

QQnorm

```
> qqnorm(datos, main = "QQnorm de mirnas$Total (l impares)",
+     xlab = "Cuantiles teóricos (dist norm)",
+     ylab = "Cuantiles de miRNAs del set Total")
> qqline(datos, col = "red")
```

QQnorm


QQnorm de mirnas$Total (l impares)
Cuantiles de miRNAs del set Total
0
10000
20000
30000
40000
−3
−2
−1
0
1
2
3
Cuantiles teóricos (dist norm)

Conclusiones

- Con el histograma vemos claramente que la gran mayoría de los miRNAs del experimento *Total* aparecen menos de 10 mil veces, con algunos pocos arriba de 40 mil. La línea de densidad nos ayuda a ver que hay varios picos entre 0 y 15 mil.
- El *boxplot* nos reafirma que nuestros datos están sesgados a la derecha. Además, el 75 % de los miRNAs de nuestro set tienen una frecuencia menor a mil por donde está la caja en el diagrama. También notamos que tenemos puntos extremos en la parte derecha (o superior) siendo el máximo aprox 42 mil.

Conclusiones

- Con la gráfica de *QQnorm* vemos claramente que nuestros datos no tienen nada que ver con la distribución normal. Si quisieramos aplicar alguna prueba estadística tendría que ser no paramétrica: no asumen una dist. normal de los datos.

Intro

- Equivalen a las librerías en C, aunque con menos problemas de dependencias.
- Son lo que lo han hecho a R tan funcional y popular.
- Existen tres grandes repositorios:
  1. CRAN: de donde bajaron R, hay más de 2 mil!
  2. Bioconductor: con estándares más altos, son para bioinformática.
  3. OmegaHat

Instalación

- Como ya vimos, la instalación de un paquete de CRAN se hace con la función install.packages:

```
> install.packages("NombrePqt")
```

- En este curso no vamos a ver Bioconductor, pero si quieren instalar un paquete hay que usar un *script* especial llamado biocLite:

```
> source("http://bioconductor.org/biocLite.R")
> biocLite()
> biocLite("NombrePqt")
```

Ayuda de un paquete

- Es muy útil checar la ayuda de un paquete!
- Allí viene la lista de funciones del paquete más una pequeña descripción.

  ```
  > help.start()
  > help(package = NombrePaquete)
  ```

- ¿Cúal es el reemplazo para la función hist en el paquete lattice?

Encontrar paquetes

- Para encontrar paquetes en CRAN que sean de su interés les recomiendo mucho los CRAN Task Views.
- Por ejemplo, el de gráficas es: `http://cran.r-project.org/web/views/Graphics.html`
- Para paquetes de Bioconductor les recomiendo:
- `http://bioconductor.org/download`
- `http://bioconductor.org/packages/release/BiocViews.html`

Cargando el pqt

- El primer paquete que veremos hoy es lattice.
- Carguenlo en su sesión con:

  ```
  > library(lattice)
  ```

- Si alguien está interesado en saber más del paquete chequen:

  ```
  > `?`(Lattice)
  ```

Datos: Chem97

- Vamos a usar el siguiente set de datos para ejemplos:

  ```
  > data(Chem97, package = "mlmRev")
  ```

- ¿Qué tipo de objeto es *Chem97*?
- ¿Cúantos datos tenemos?

Chem97

- Rápidamente:

```
> class(Chem97)

[1] "data.frame"

> dim(Chem97)

[1] 31022     8

> Chem97[1, ]

  lea school student score gender age
1   1      1       1     4      F   3
  gcsescore   gcsecnt
1     6.625 0.3393157
```

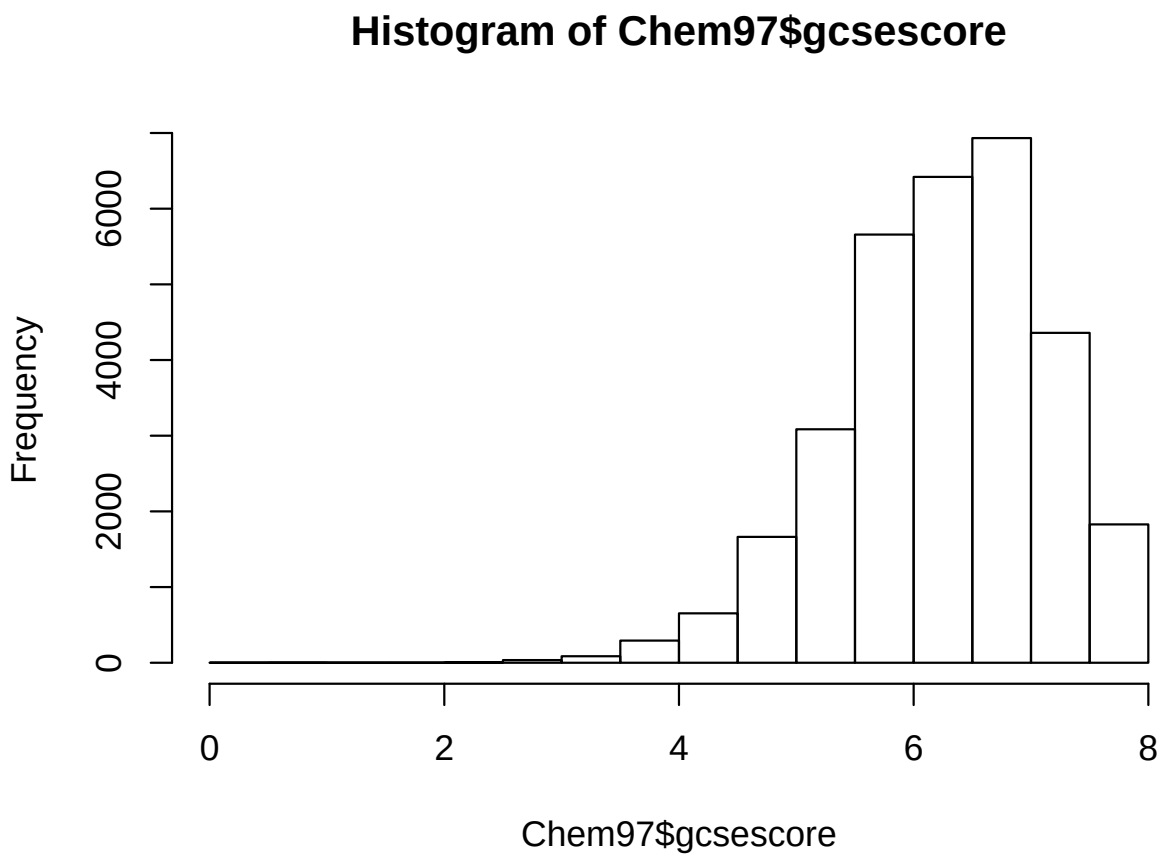
- Son unos datos de exámenes en USA.

Sintáxis de fórmula

- lattice usa la sintáxis de fórmula.
- Básicamente es $y \sim x|g1 * g2$ donde x es la variable con datos numéricos y $g1$ es de tipo *factor*.
- ¿Qué era un *factor*?
- Ahora comparemos dos histogramas:

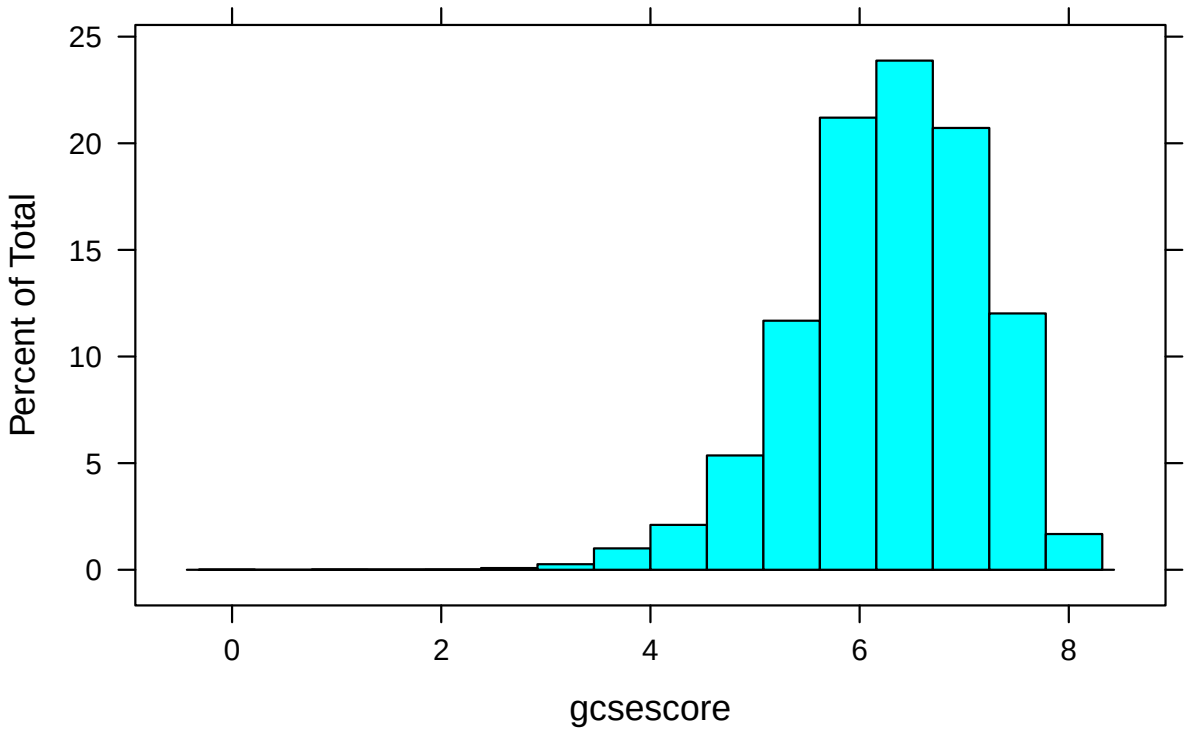
Opción normal

```
> hist(Chem97$gcsescore)
```



Opción lattice

```
> print(histogram(~gcsescore, data = Chem97))
```



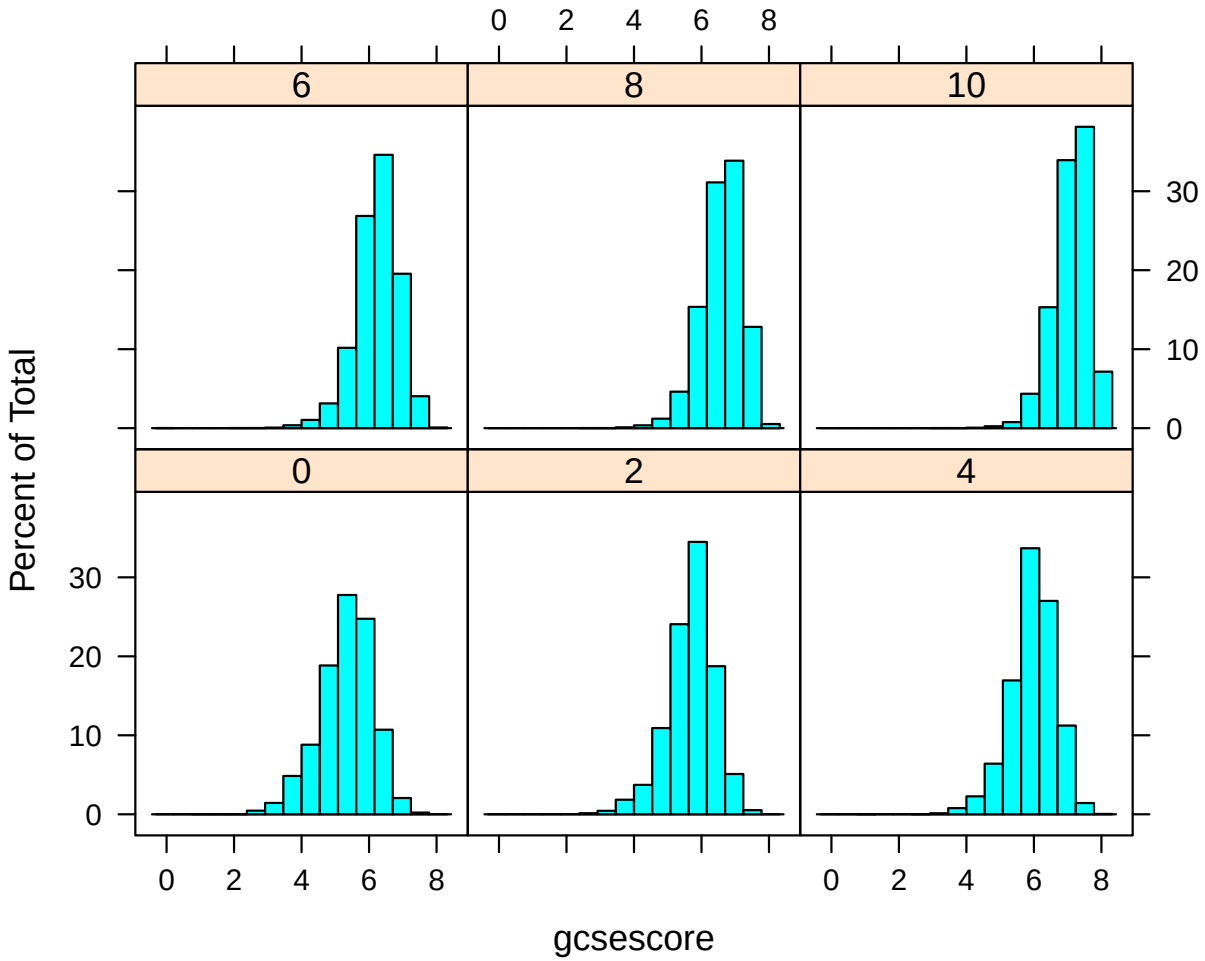
Print

- ¿Por qué usé la función print? ¿Alguna idea?
- La respuesta será clara cuando intenten guardar su imagen usando alguna función como pdf.
- ¿Por qué es útil usar el argumento de *data*?
- ¿Son iguales nuestros histogramas?
- En realidad los dos histogramas se parecen mucho, pero la gran ventaja de usar lattice es cuando agrupamos nuestros datos de una variable por las categorías de otra.

gcsescore dado score

```
> print(histogram(~gcsescore | factor(score),
+     data = Chem97))
```

gcsescore dado score



Ahora con 3 variables

- ¿Por qué usamos la función factor?

```
> class(Chem97$score)
```

```
[1] "numeric"
```

- En la anterior gráfica tenemos un panel para cada categoría de la variable *score*.
- Pero bueno, el chiste no era visualizar dos variables, sino 3! Por ejemplo, el género:

```
> class(Chem97$gender)
```

```
[1] "factor"
```

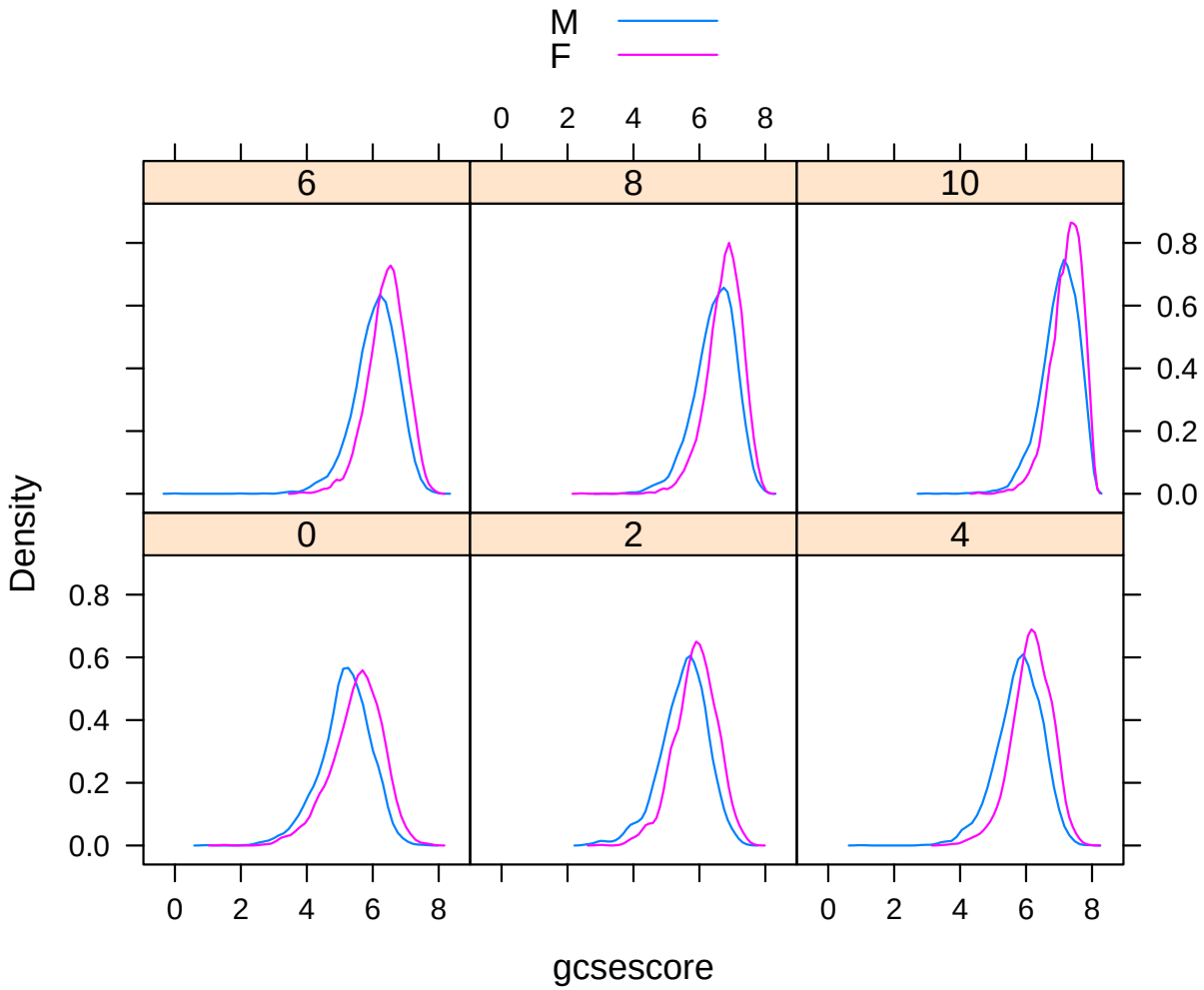
- ¿Se podrán hacer histogramas con estas tres variables?
- ¿O se les ocurre alguna alternativa?

Densityplot

En realidad está dificil con histogramas, pero si se puede con líneas de densidad :)

```
> print(densityplot(~gcsescore |
+     factor(score), Chem97, groups = gender,
+     plot.points = FALSE, auto.key = TRUE))
```

Densityplot



Otros argumentos

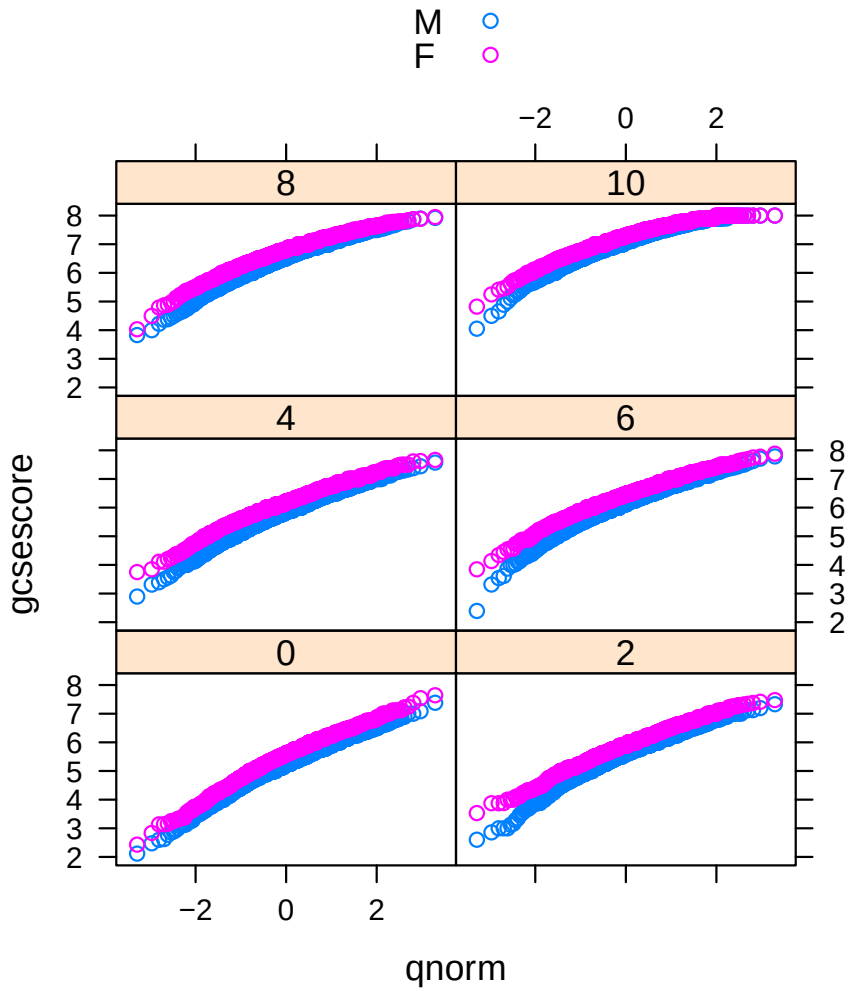
- ¿Cómo usé nuestra tercera variable?
- Uso el argumento *plot.points* para no saturar nuestra gráfica.
- ¿Qué creen que hace el argumento auto.key?

qqmath

También podemos usar qqmath que es muy similar a *qqplot*

```
> print(qqmath(~gcsescore | factor(score),
+      Chem97, groups = gender, auto.key = TRUE,
+      aspect = "xy", f.value = ppoints(1000)))
```

qqmath

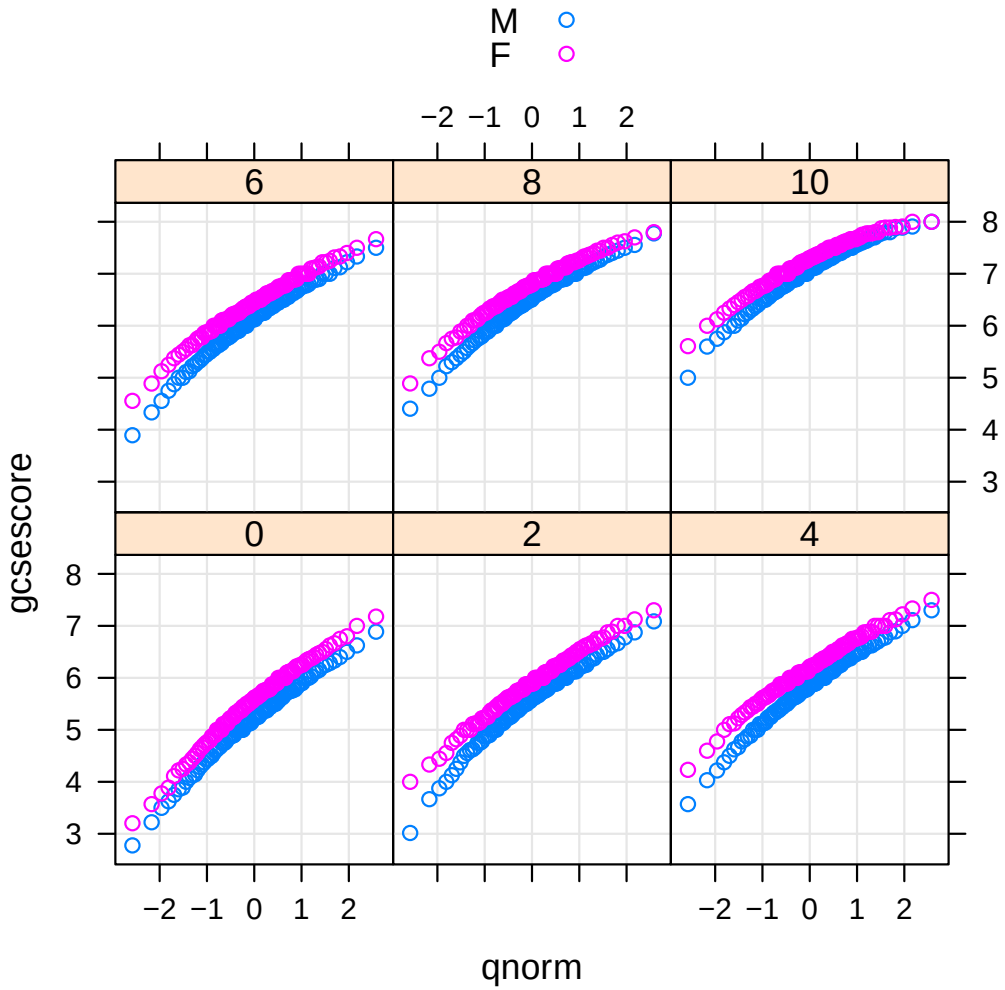


Otra opción

¿Qué notan de diferente en esta gráfica?

```
> print(qqmath(~gcsescore | factor(score),
+      Chem97, groups = gender, auto.key = TRUE,
+      aspect = "xy", f.value = ppoints(100),
+      type = c("p", "g")))
```

Otra opción

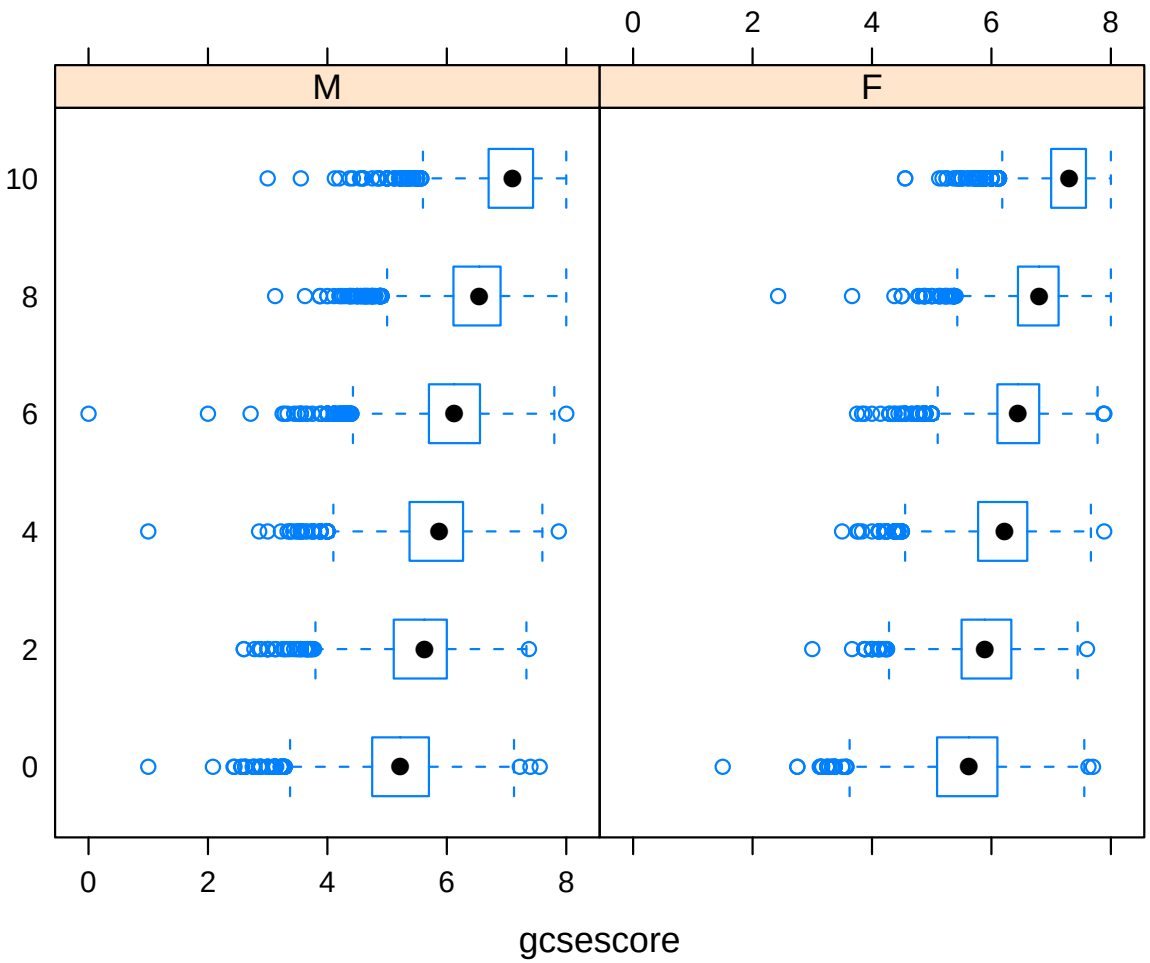


Boxplots

También podemos hacer diagramas de caja y brazos (boxplots) usando bwplot:

```
> print(bwplot(factor(score) ~ gcsescore |
+     gender, Chem97))
```

Boxplots

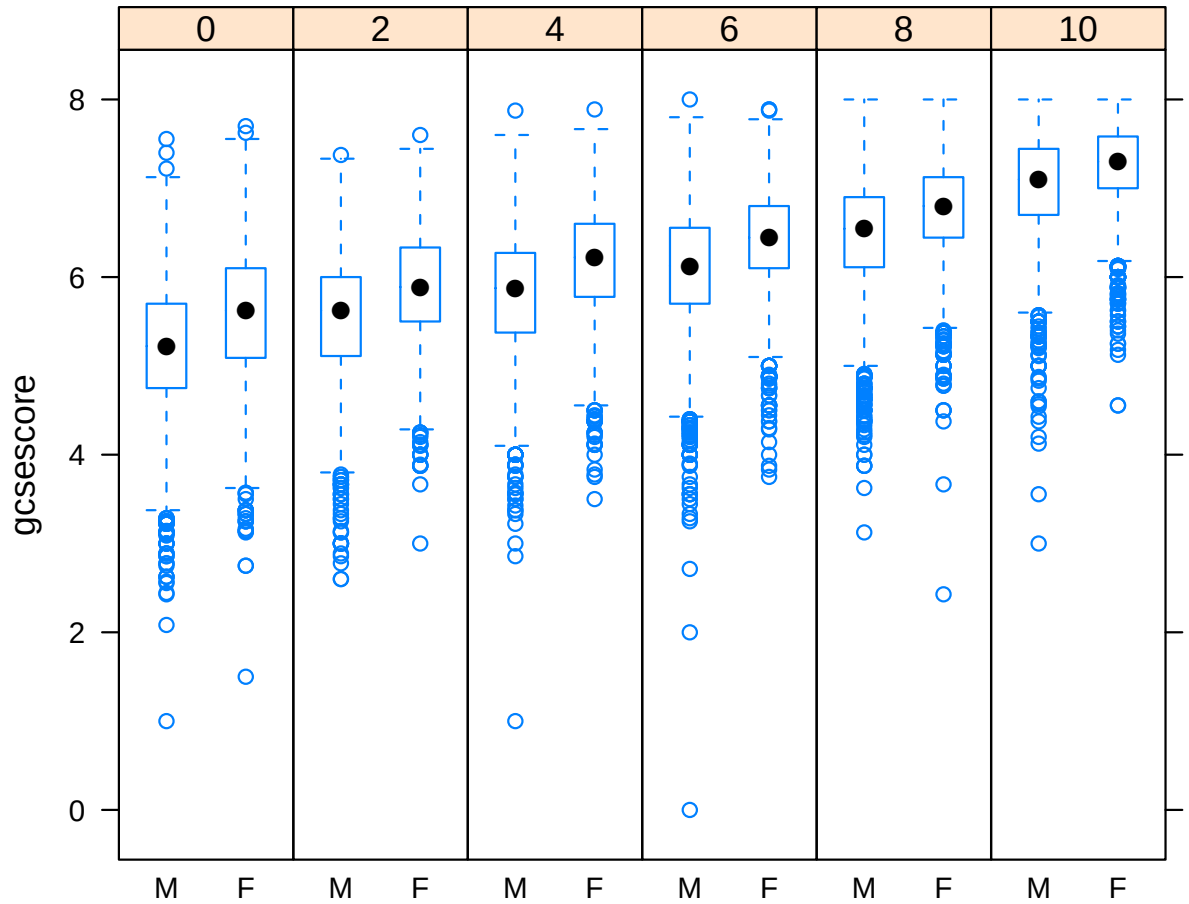


Boxplots II

¿Qué cambia en esta segunda llamada a bwplot?
¿Cúal es más informativa?

```
> print(bwplot(gcsescore ~ gender |
+     factor(score), Chem97, layout = c(6,
+     1)))
```

Boxplots II



Stripplot

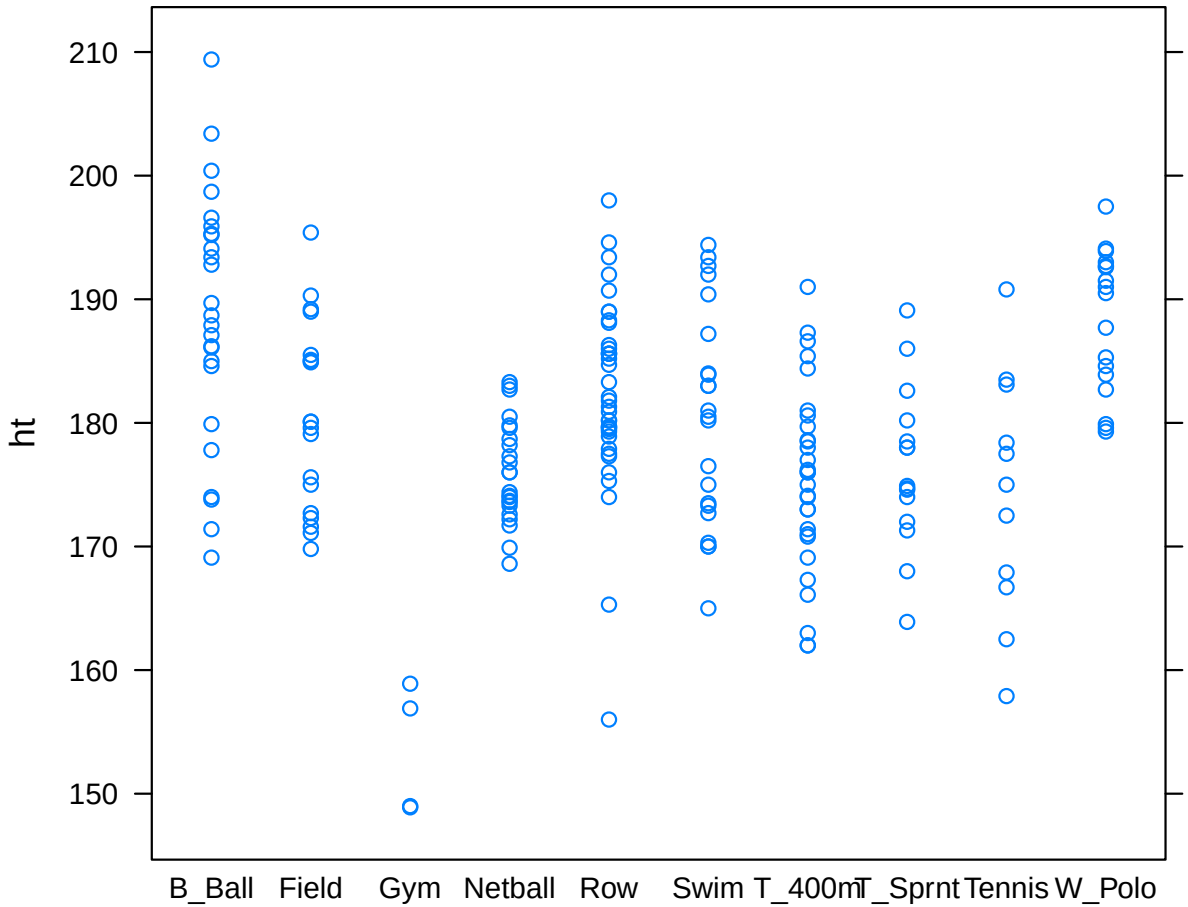
Otra función gráfica útil es la de stripplot.
Es muy útil para explorar rápidamente unos datos.

```
> library(DAAG)
> ais[1, ]
```

```
   rcc wcc   hc   hg ferr   bmi   ssf
1 3.96 7.5 37.5 12.3   60 20.56 109.1
  pcBfat   lbm    ht   wt sex  sport
1  19.75 63.32 195.9 78.9   f B_Ball
```

```
> print(stripplot(ht ~ factor(sport),
+     data = ais))
```

Stripplot

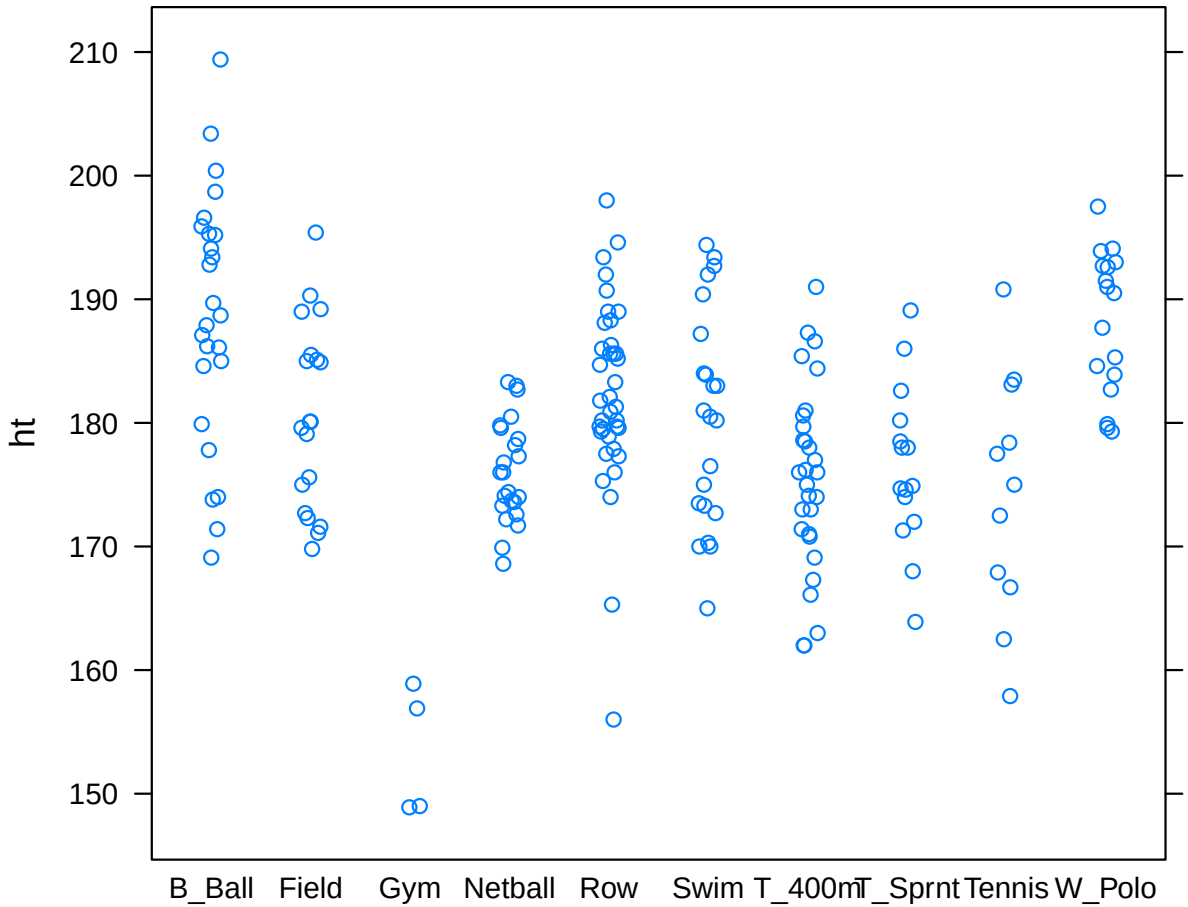

ht
210
200
190
180
170
160
150
B_Ball
Field
Gym
Netball
Row
Swim
T_400m
T_Sprnt
Tennis
W_Polo

Stripplot II

- Además, es excelente para ejemplificar el argumento jitter :)
- ¿Qué hace este argumento?

```
> print(stripplot(ht ~ factor(sport),
+     data = ais, jitter = T))
```

Stripplot II

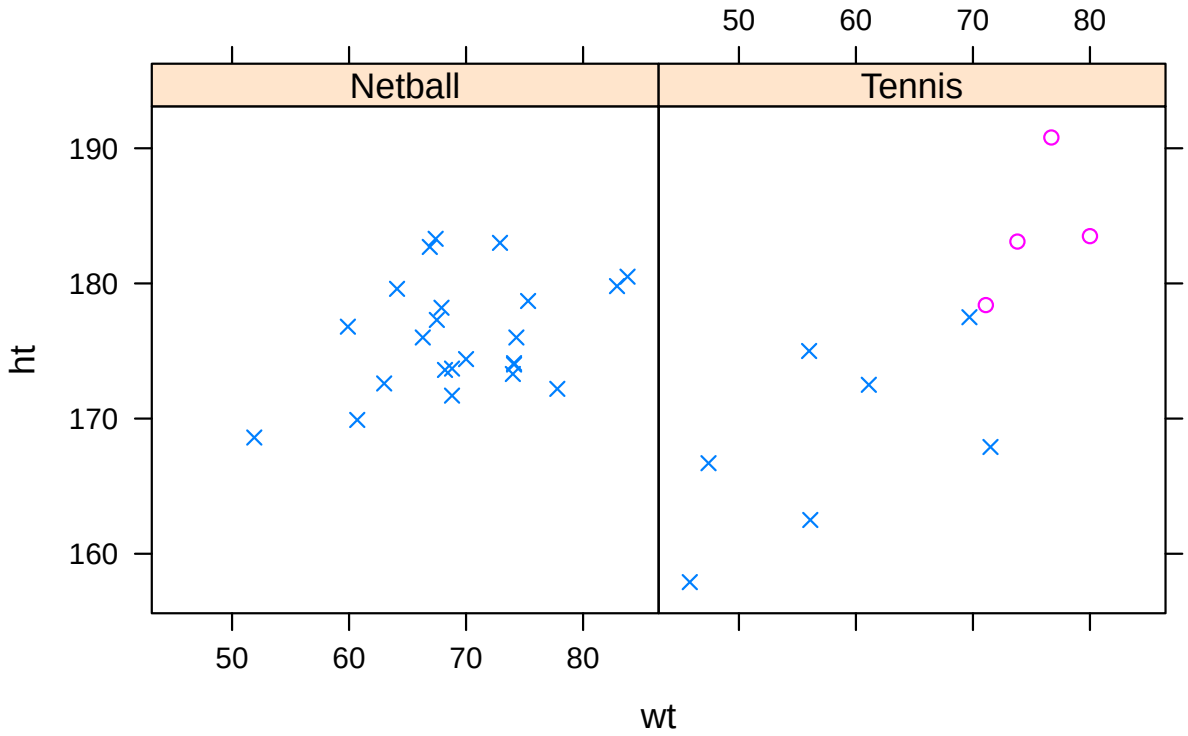


xyplot

- En lattice, la función madre[2] es xyplot.
- Primero obtenemos un subconjunto de datos y luego usamos `xyplot`[3]:

```
> subset <- ais$sport %in% c("Netball",
+     "Tennis")
> print(xyplot(ht ~ wt | sport, groups = sex,
+     pch = c(4, 1), aspect = 1,
+     subset = subset, data = ais))
```

xyplot



[2] Es muy similar a *plot*.

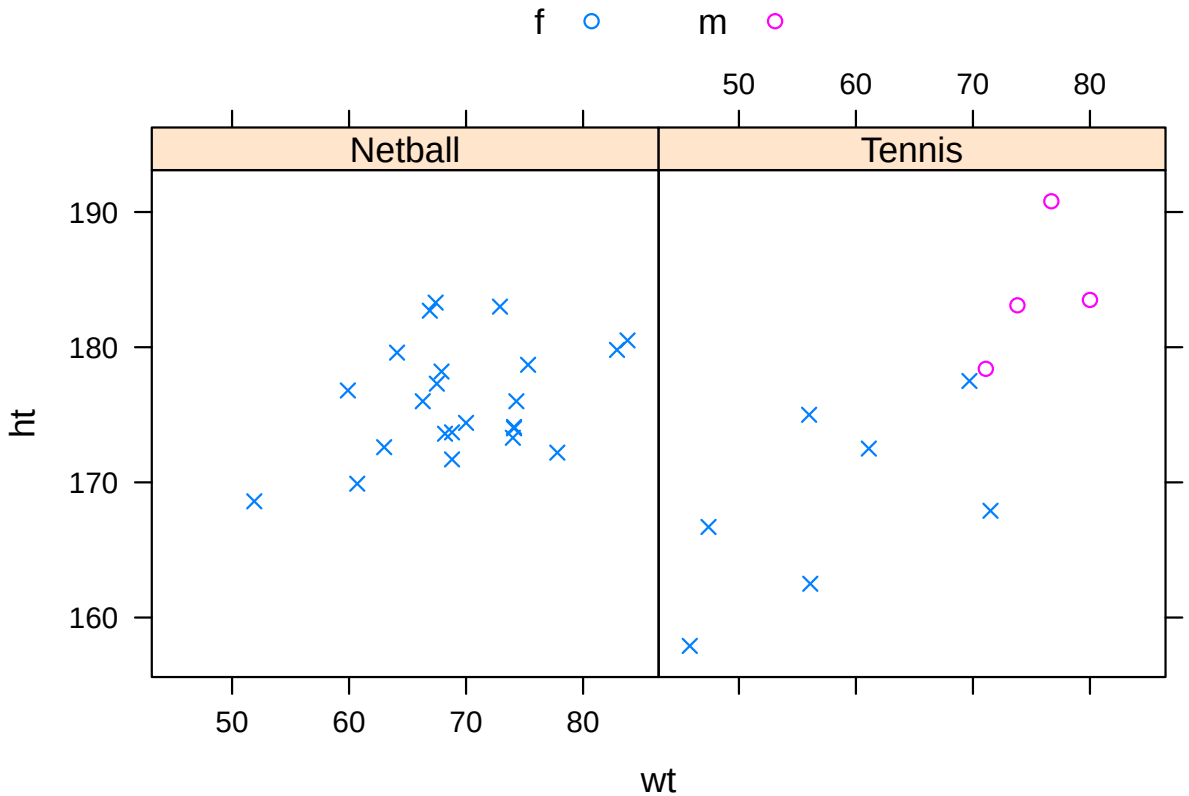
[3] Noten que usamos el argumento `subset`.

xyplot II

¿Cúantas variables estamos viendo en la siguiente gráfica?

```
> print(xyplot(ht ~ wt | sport, groups = sex,
+      pch = c(4, 1), aspect = 1,
+      auto.key = list(columns = 2),
+      subset = subset, data = ais))
```

xyplot II

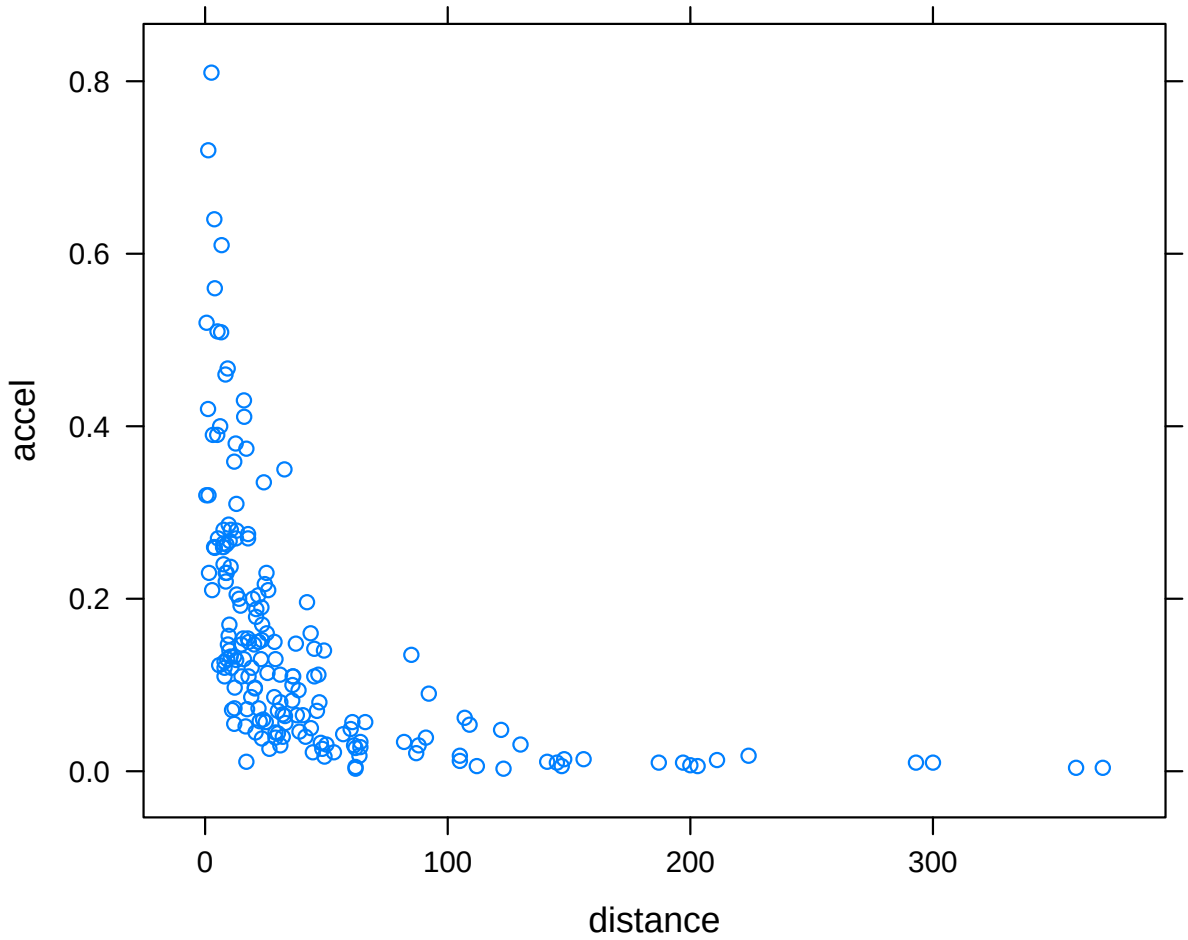


xyplot B

Este es otro ejemplo con datos de temblores:

```
> data(Earthquake, package = "nlme")
> print(xyplot(accel ~ distance,
+      data = Earthquake))
```

xyplot B



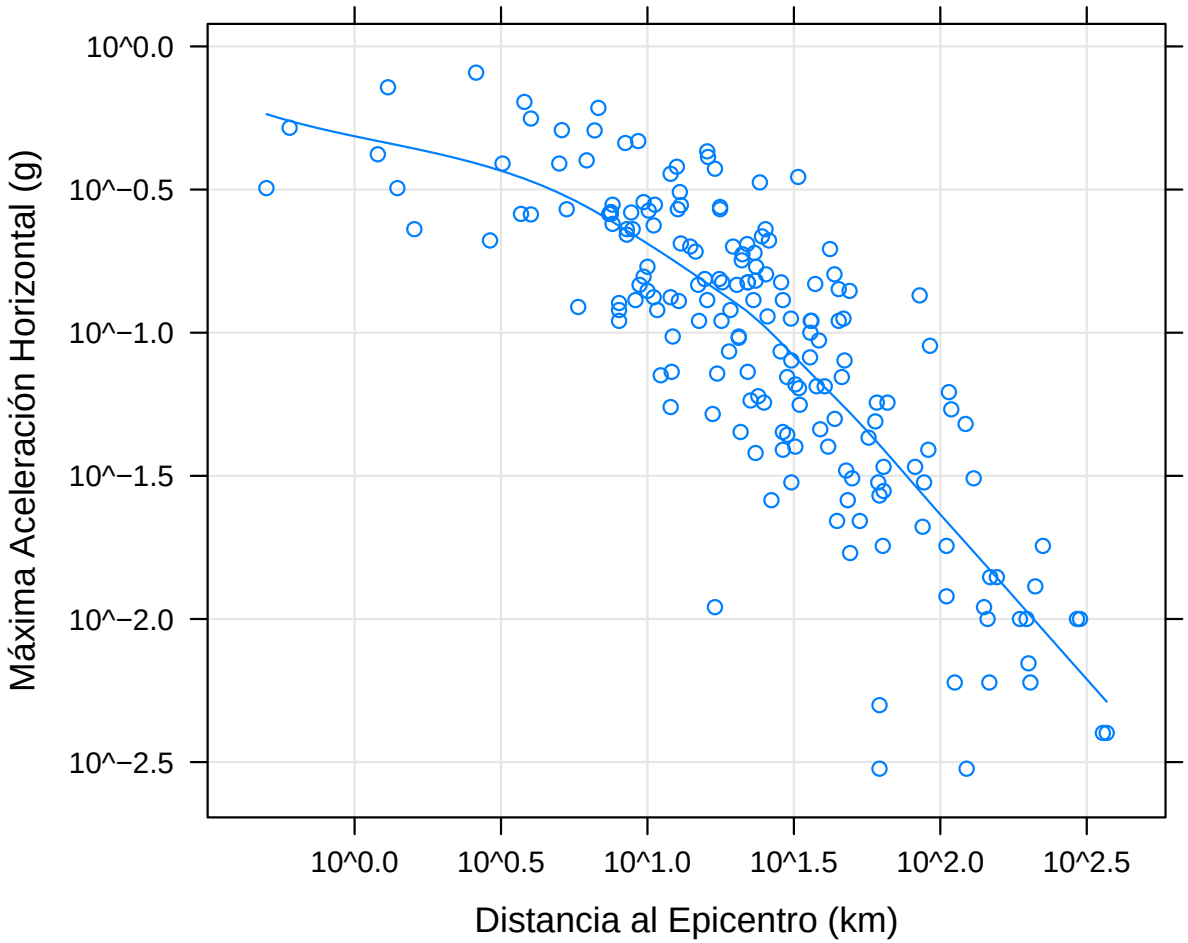
xyplot B II

Esta segunda gráfica, con los mismos datos, es más complicada porque estoy usando más argumentos.

- ¿Qué controla `scales`?
- ¿Qué hace `smooth` en el argumento `type`?

```
> print(xyplot(accel ~ distance,
+      data = Earthquake, scales = list(log = TRUE),
+      type = c("p", "g", "smooth"),
+      xlab = "Distancia al Epicentro (km)",
+      ylab = "Máxima Aceleración Horizontal (g)"))
```

xyplot B II

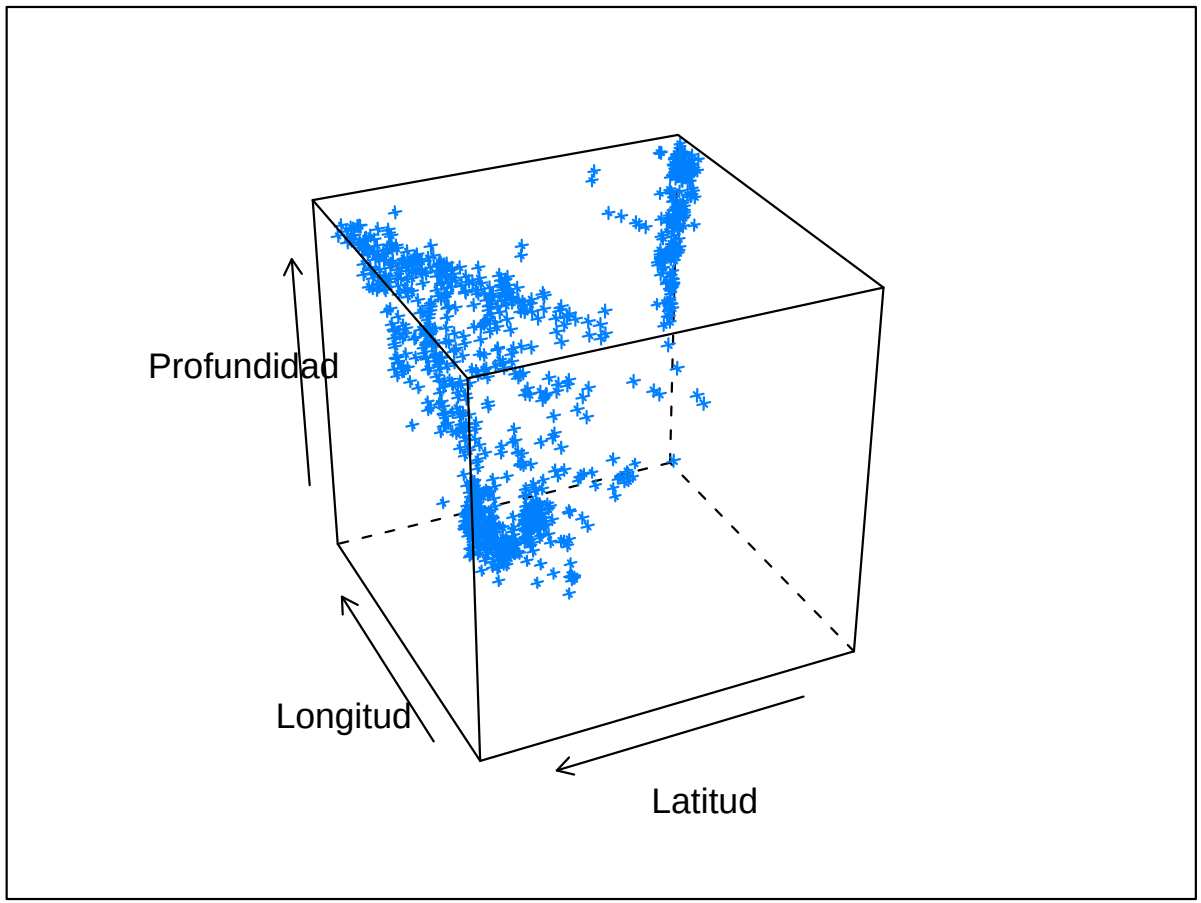

Máxima Aceleración Horizontal (g)
10^0.0
10^−0.5
10^−1.0
10^−1.5
10^−2.0
10^−2.5
10^0.0
10^0.5
10^1.0
10^1.5
10^2.0
10^2.5
Distancia al Epicentro (km)

Ahora en 3D!

- Lattice nos permite hacer gráficas en 3D con la función cloud.
- Al igual que el resto de las gráficas de R no es interactiva.
- Pero pueden hacer diferentes cambiando los valores de x, y y z.

```
> print(cloud(depth ~ lat * long,
+       data = quakes, zlim = rev(range(quakes$depth)),
+       screen = list(z = 115, x = -60),
+       panel.aspect = 0.75, xlab = "Longitud",
+       ylab = "Latitud", zlab = "Profundidad"))
```

Ahora en 3D!



Fin Lattice

- Eso es todo lo que veremos de lattice, aunque hay más funciones como barchart y dotplot.
- Una herramienta excelente para ver todas las gráficas que se pueden hacer con un paquete es con la función testInstalledPackage:

```
> library(tools)
> testInstalledPackage(NombrePqt)
```

- Exploren paquetes como latticeExtra!

Intro

- Muchas funciones gráficas!
- Excelente candidato para que usen la función que acabamos de ver :)

Barras y tabla

- Primero creamos el objeto `df` con datos :)
- Luego usamos barp para crear las barras.
- Finalmente agregamos la tabla de datos usando addtable2plot!

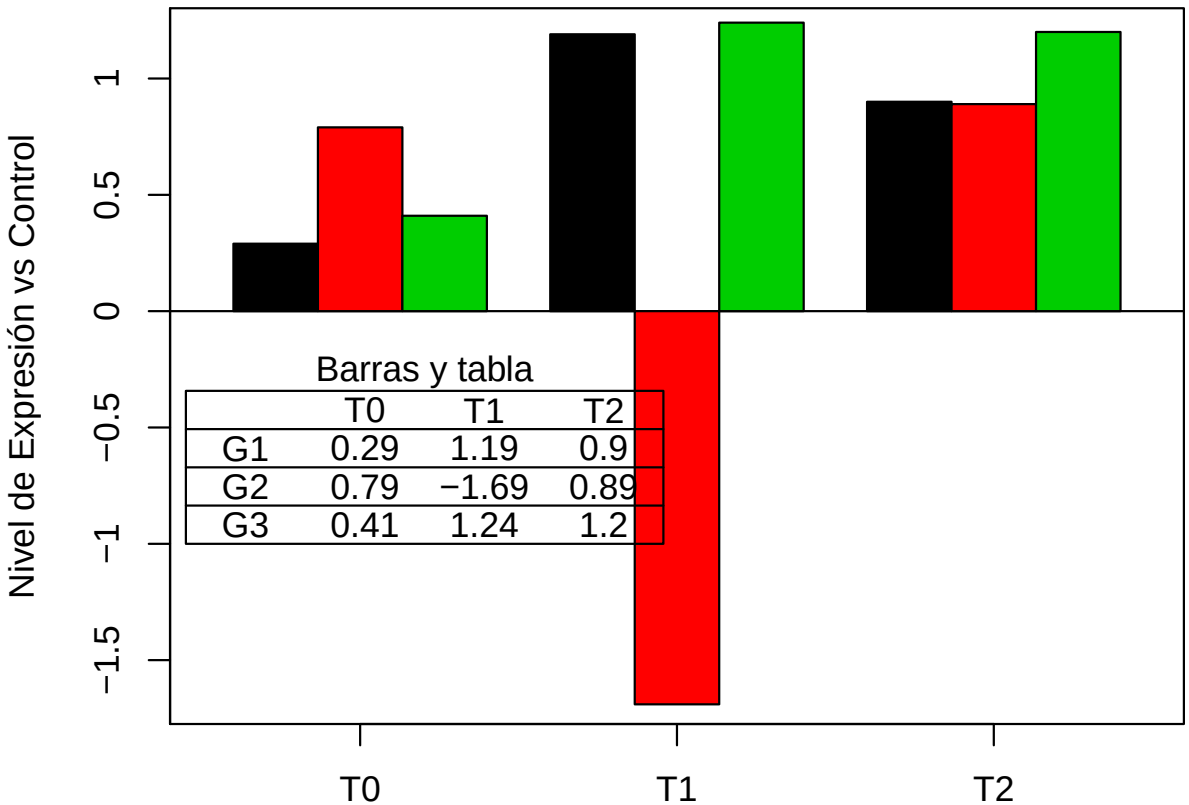
```
> set.seed(123)
> df <- data.frame(T0 = runif(3),
+     T1 = rnorm(3), T2 = rlnorm(3))
> df <- round(df, digits = 2)
> rownames(df) <- c("G1", "G2", "G3")
> df
```

```
     T0    T1   T2
G1 0.29  1.19 0.90
G2 0.79 -1.69 0.89
G3 0.41  1.24 1.20
```

Barras y tabla

```
> library(plotrix)
> barp(df, ylab = "Nivel de Expresión vs Control",
+     names.arg = colnames(df), col = 1:3)
> addtable2plot(0.45, -1, df, bty = "o",
+     display.rownames = TRUE, hlines = TRUE,
+     title = "Barras y tabla")
```

Barras y tabla

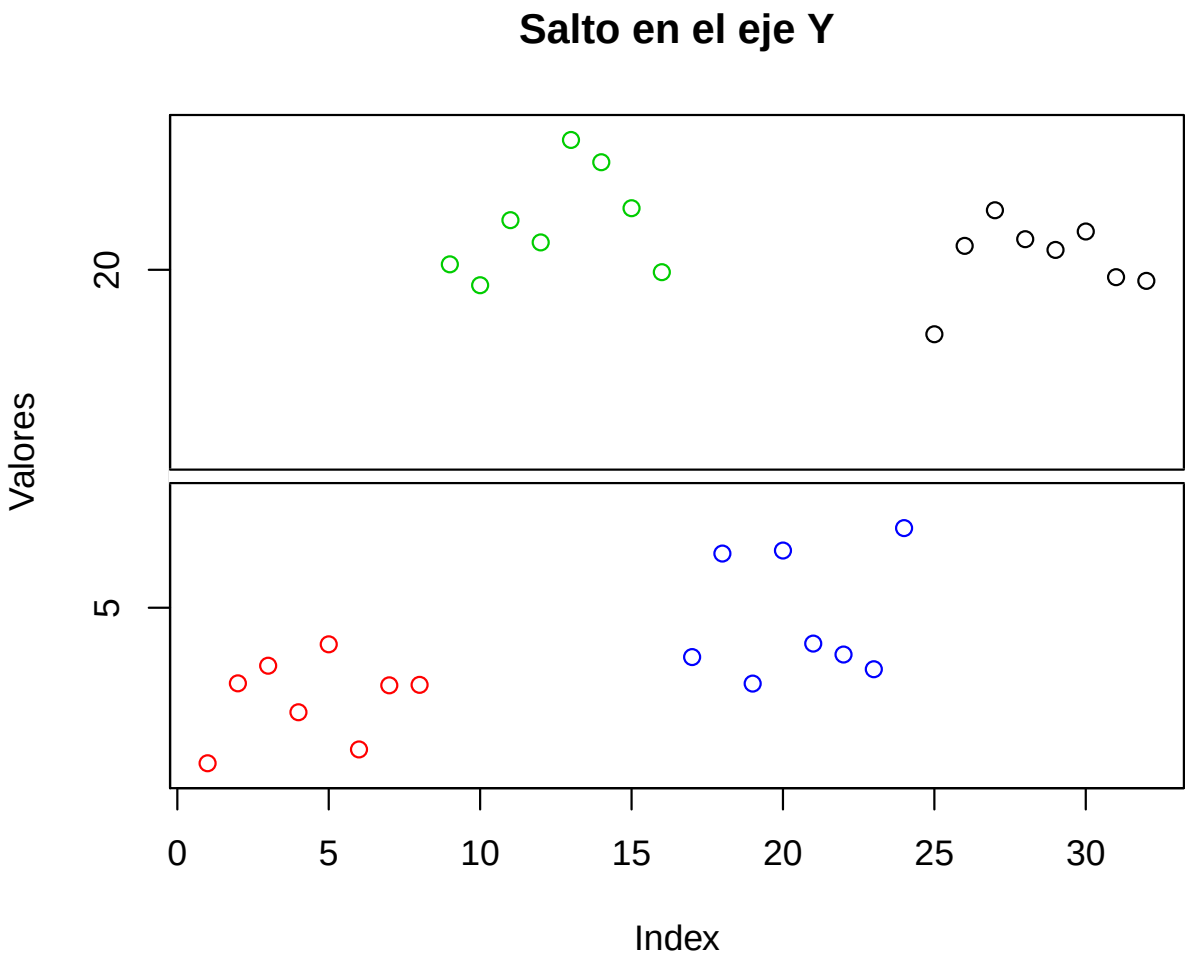


Brincos

- Con Plotrix podemos hacer gráficas que se saltan (brincan) valores en un eje.
- Por ejemplo, con un brinco en el eje Y:

```
> data <- c(rnorm(8) + 3, rnorm(8) +
+     21, rnorm(8) + 4.5, rnorm(8) +
+     20)
> color <- c(rep(2, 8), rep(3, 8),
+     rep(4, 8), rep(1, 8))
> gap.plot(data, gap = c(8, 16),
+     xlab = "Index", ylab = "Valores",
+     main = "Salto en el eje Y",
+     col = color)
```

Brincos

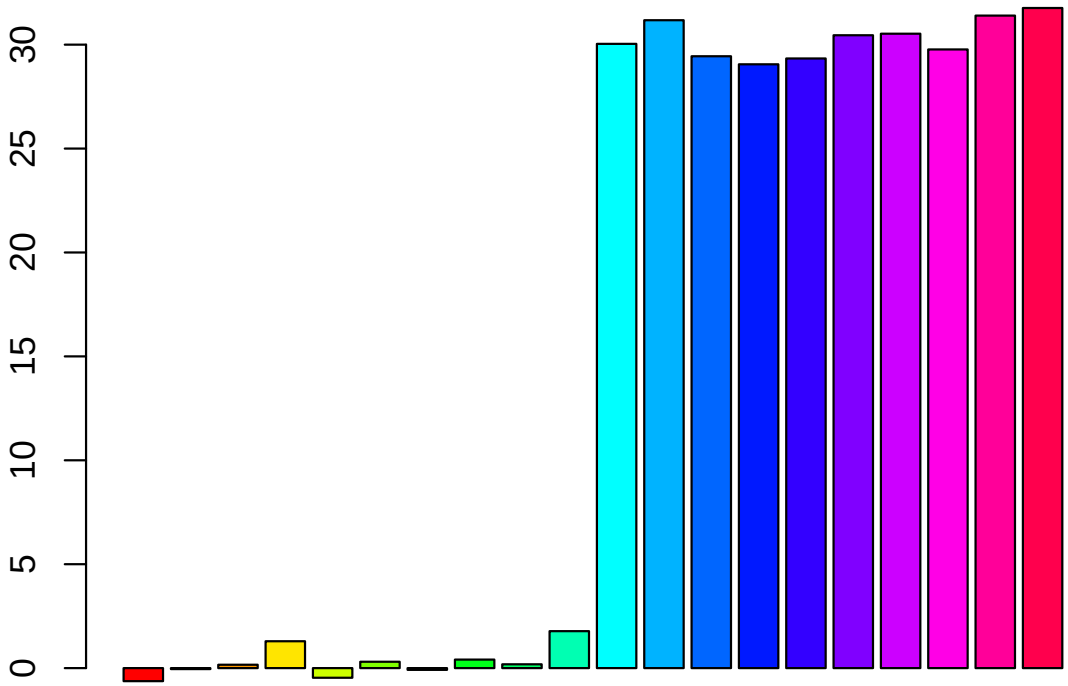


Un problema

¿Qué problema tenemos con gráficas como la siguiente?

```
> data <- c(rnorm(10), rnorm(10) +
+     30)
> barplot(data, col = rainbow(20))
```

Un problema

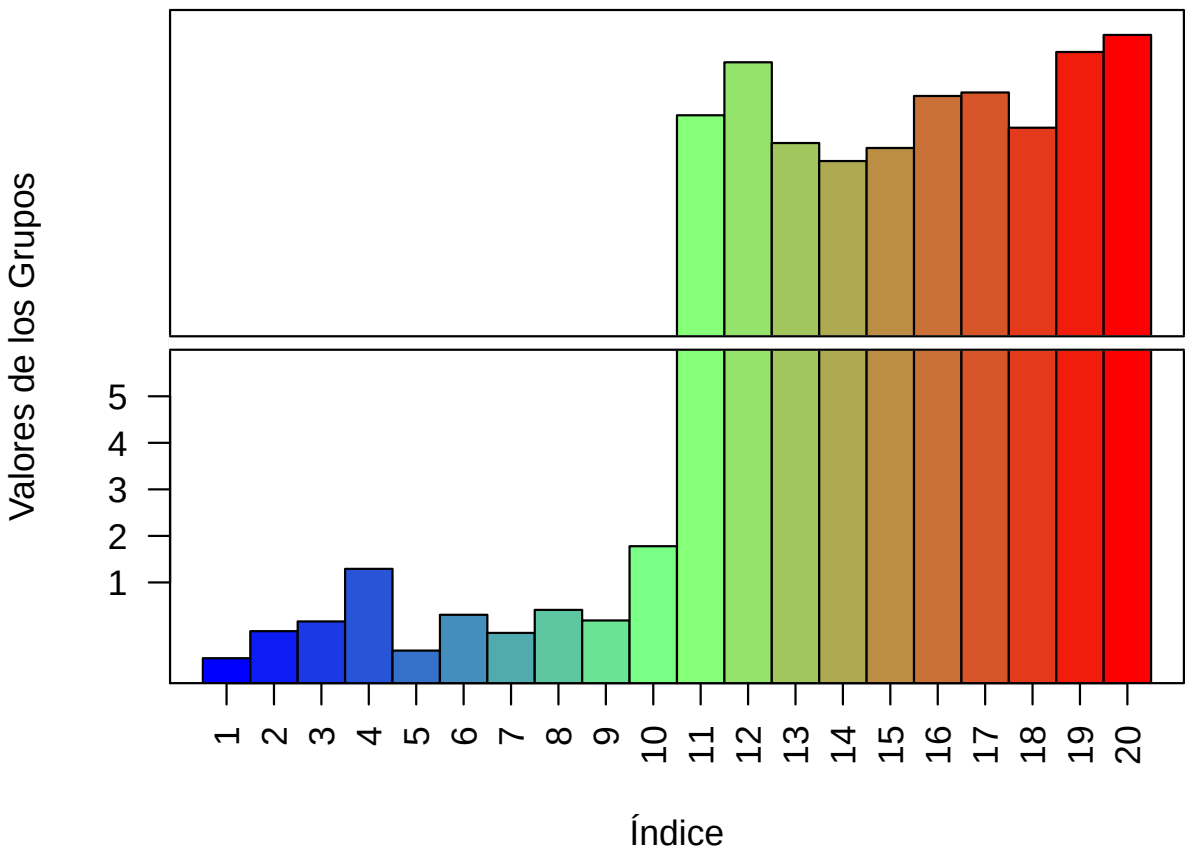


Saltando barras :)

- También podemos tener brincos con barras!
- Si sus datos están partidos en unos muy chicos y otros muy grandes, este tipo de gráfica es muy útil para visualizar todos!
- Hay un problema con los nombres en el eje Y así que tengan cuidado.

```
> gap.barplot(data, gap = c(6, 25),
+     xlab = "Índice", ytics = c(1:30),
+     ylab = "Valores de los Grupos",
+     las = 2)
```

Saltando barras :)

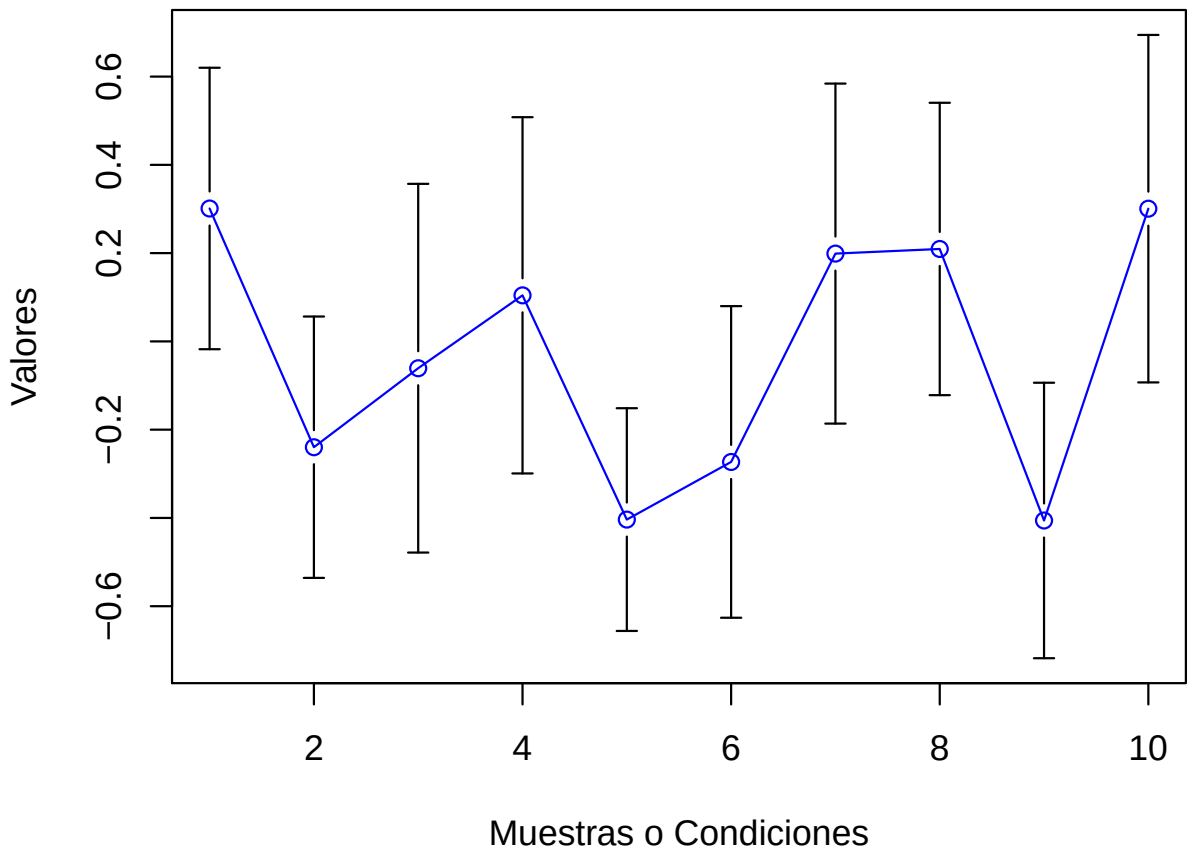


Con barras para los errores

- Como muchos experimentos se hacen con réplicas, seguido vemos gráficas con las barras para los *errores*.
- Usando dispersion podemos hacer gráficas que muestren el error estándar:

```
> data <- matrix(rnorm(100), 10,
+     10)
> a <- colMeans(data)
> b <- std.error(data)
> plot(a, ylim = c(min(a - b), max(a +
+     b)), xlab = "Muestras o Condiciones",
+     ylab = "Valores", col = 4,
+     type = "o")
> dispersion(1:10, colMeans(data),
+     b)
```

Con barras para los errores



Ahora on datos reales

- Ahora vamos a usar datos de este artículo. Es el de la secuencia de un coreano.
- Leamos unas tablas en formato `csv` que ya había preparado.

```
> t1 <- read.csv("http://www.lcg.unam.mx/~lcollado/B/data/SuppTable(
+     header = T)
> t2 <- read.csv("http://www.lcg.unam.mx/~lcollado/B/data/SuppTable(
+     header = T)
```

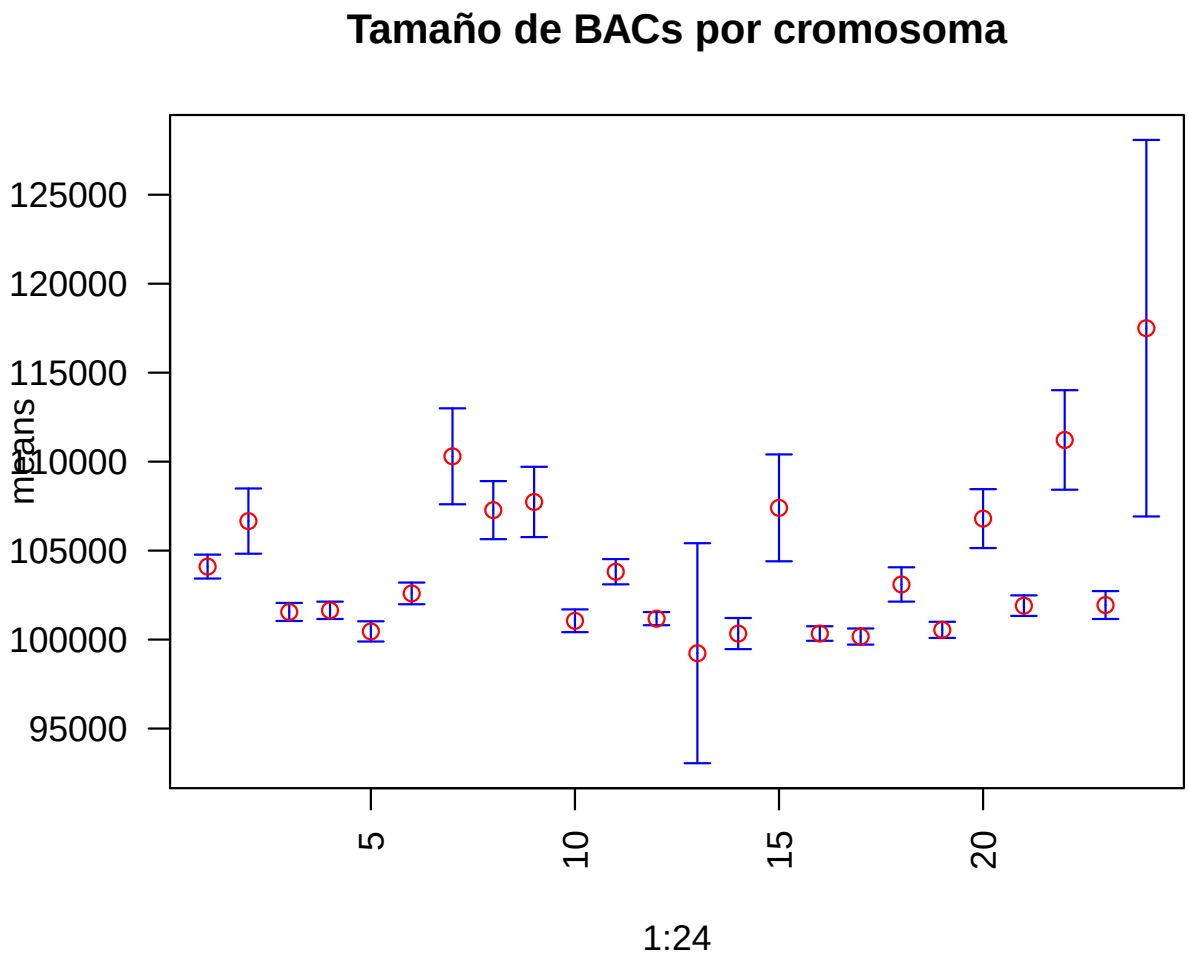
- Usen head, dim, class para aprender más de los datos.

plotCI

- Además de `dispersion` podemos usar la función plotCI para graficar los puntos con los errores.
- Vamos a usar `tapply` para encontrar la media del tamaño de los BACs por cromosoma y lo mismo para el error estándar.

```
> means <- tapply(t1$bac_size, t1$chrNo,
+     mean)
> err <- tapply(t1$bac_size, t1$chrNo,
+     std.error)
> plotCI(1:24, means, err, col = "red",
+     scol = "blue", las = 2, main = "Tamaño de BACs por cromosoma")
```

plotCI

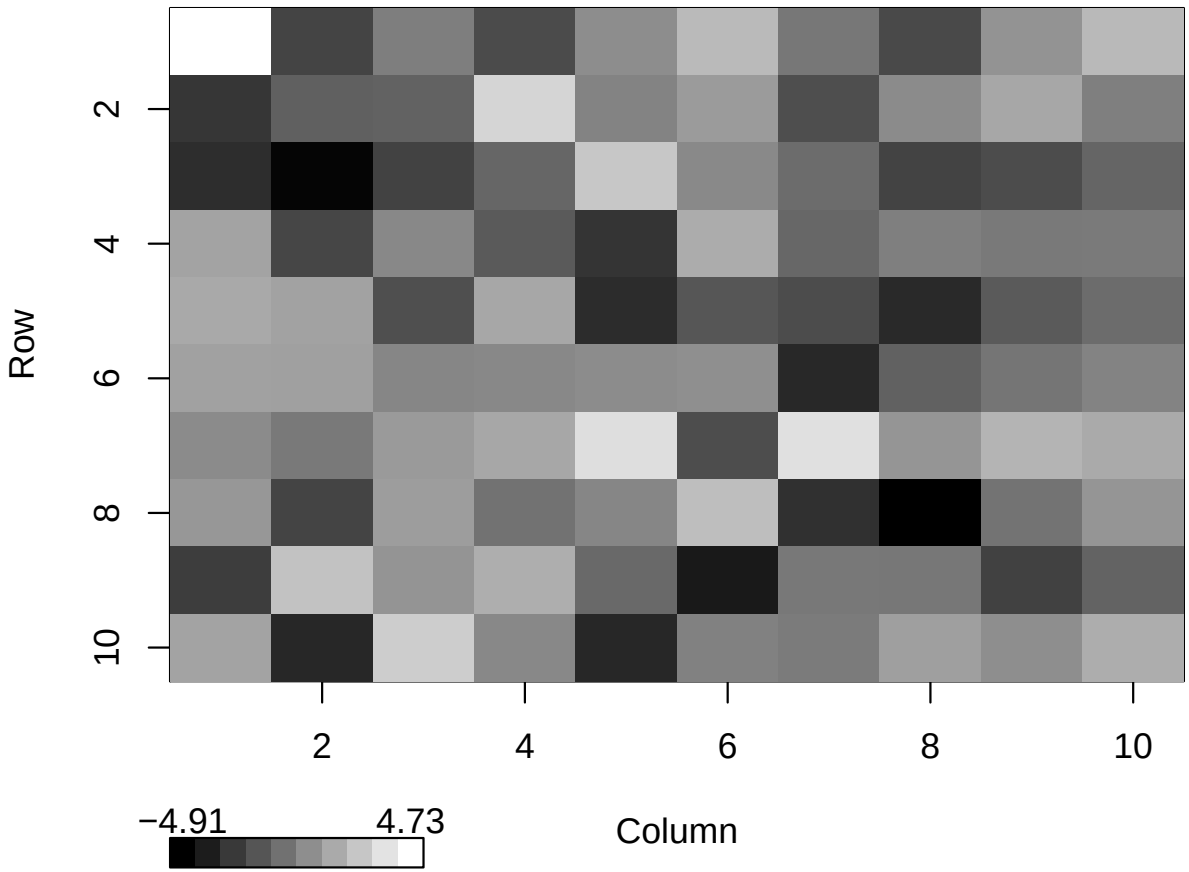

Tamaño de BACs por cromosoma
125000
120000
115000
110000
105000
100000
95000
means
5
10
15
20
1:24

Para visualizar matrices

- Con color2D.matplot podemos visualizar una matriz[4]

```
> mat <- matrix(rnorm(100, 0, 2),
+     10, 10)
> color2D.matplot(mat, show.legend = T)
```

Para visualizar matrices



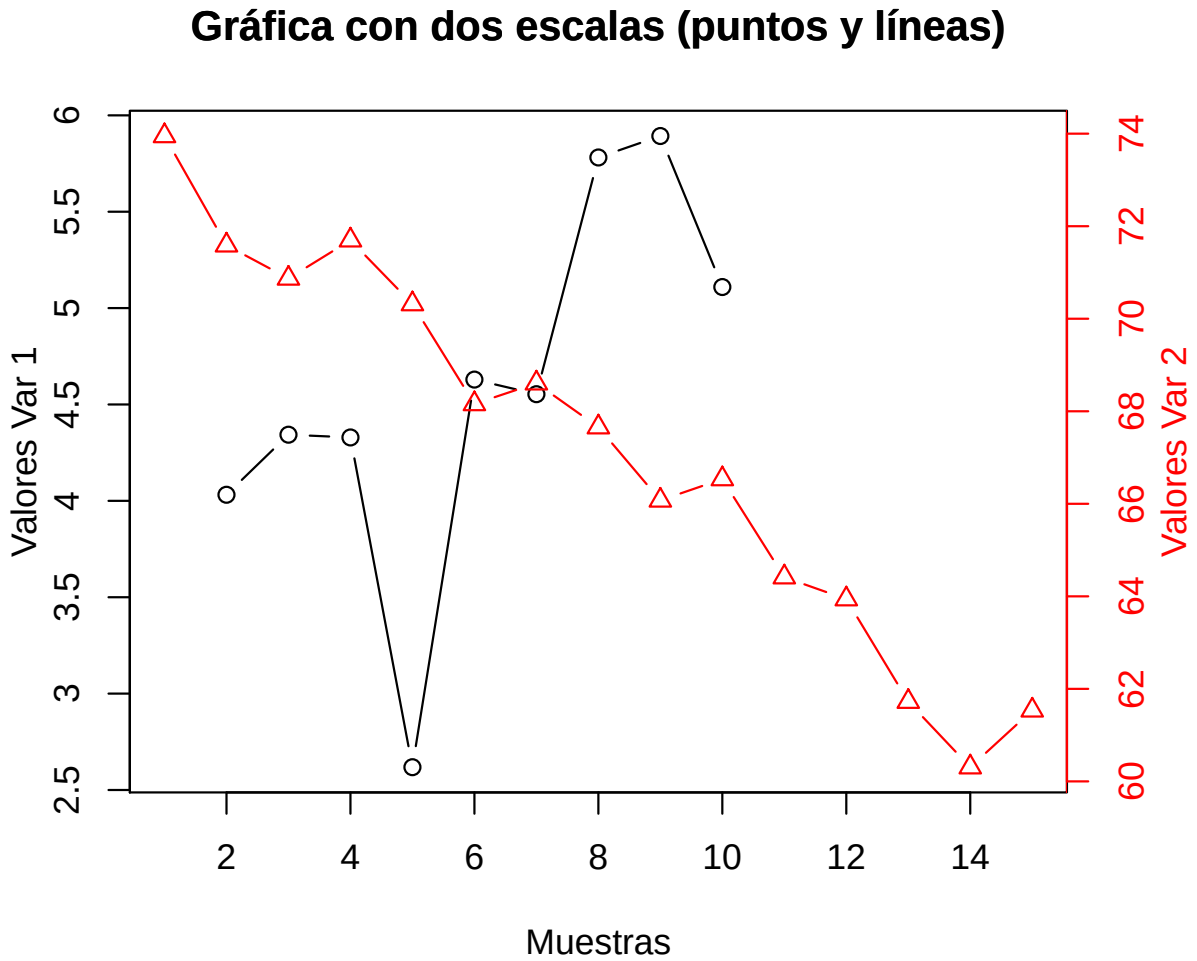
[4] Una función similar de R básico es image.

Dos escalas

- Algunas veces estás midiendo dos variables para un conjunto de muestras. Estas variables pueden tener escalas muy diferentes.
- Por eso usamos twoord.plot para visualizar las dos variables :)

```
> twoord.plot(2:10, seq(3, 7, by = 0.5) +
+     rnorm(9), 1:15, rev(60:74) +
+     rnorm(15), xlab = "Muestras",
+     ylab = "Valores Var 1", rylab = "Valores Var 2",
+     main = "Gráfica con dos escalas (puntos y líneas)")
```

Dos escalas

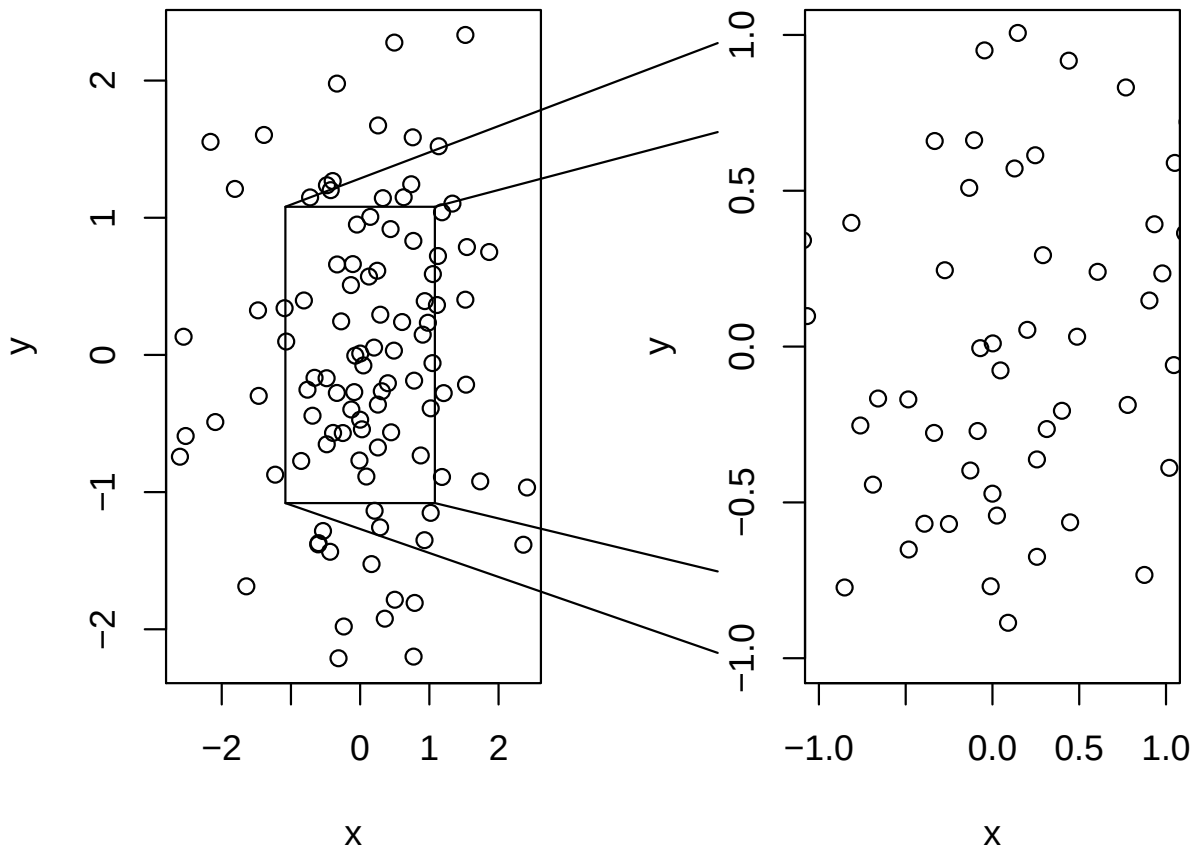

Gráfica con dos escalas (puntos y líneas)
Valores Var 1
Valores Var 2
Muestras
2.5
3
3.5
4
4.5
5
5.5
6
60
62
64
66
68
70
72
74
2
4
6
8
10
12
14

Zoom

- Otra función gráfica que me gustó es zoomInPlot para visualizar un conjunto de datos y un subconjunto interesante.

```
> zoomInPlot(rnorm(100), rnorm(100),
+     rxlim = c(-1, 1), rylim = c(-1,
+         1))
```

Zoom

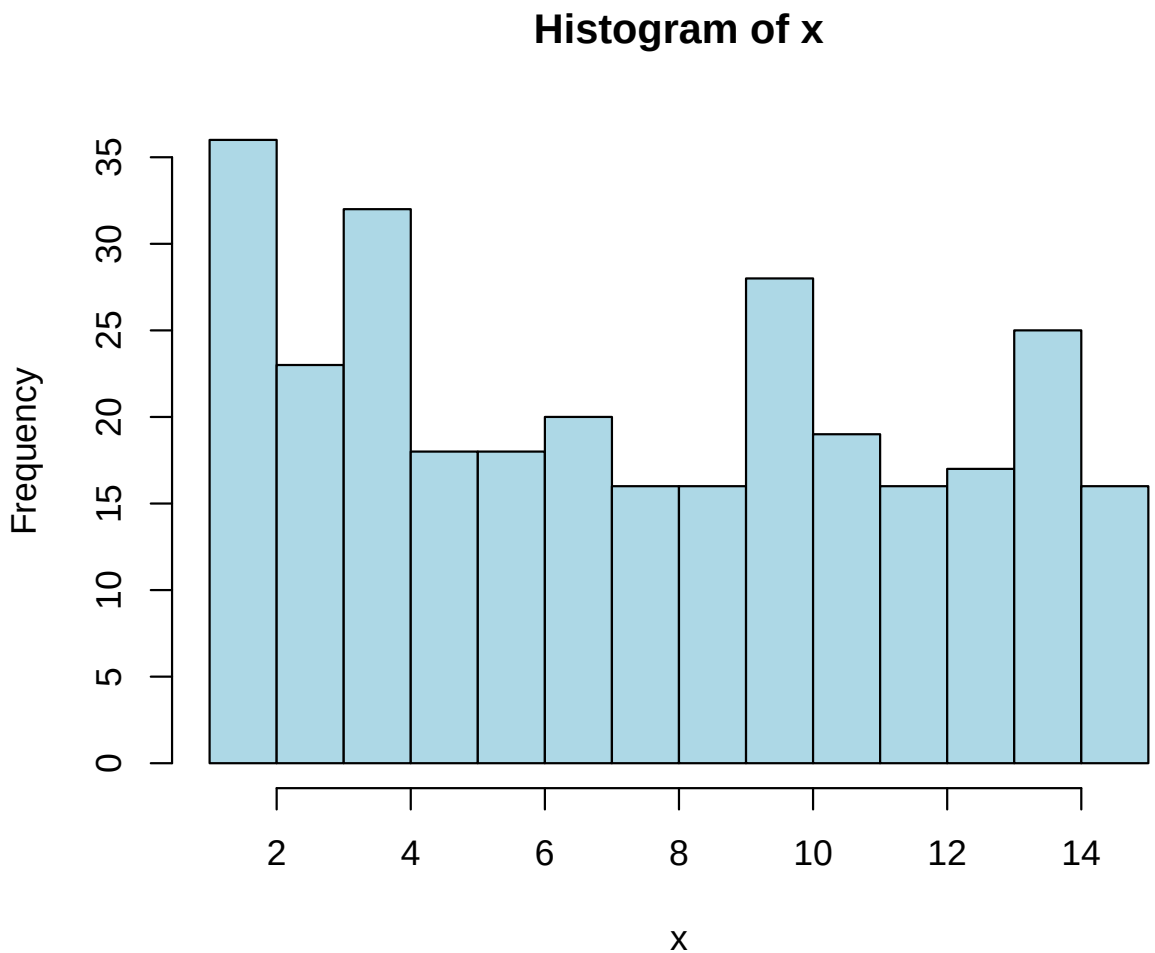


Histograma calibrado

- Para terminar con Plotrix, usaremos weighted.hist.
- Es útil para *calibrar* una variable con otra. A esta segunda var se le llama el peso.
- Primero hagamos un histograma de nuestra variable x:

```
> x <- sample(1:15, 300, TRUE)
> hist(x, col = "light blue")
```

Histograma calibrado

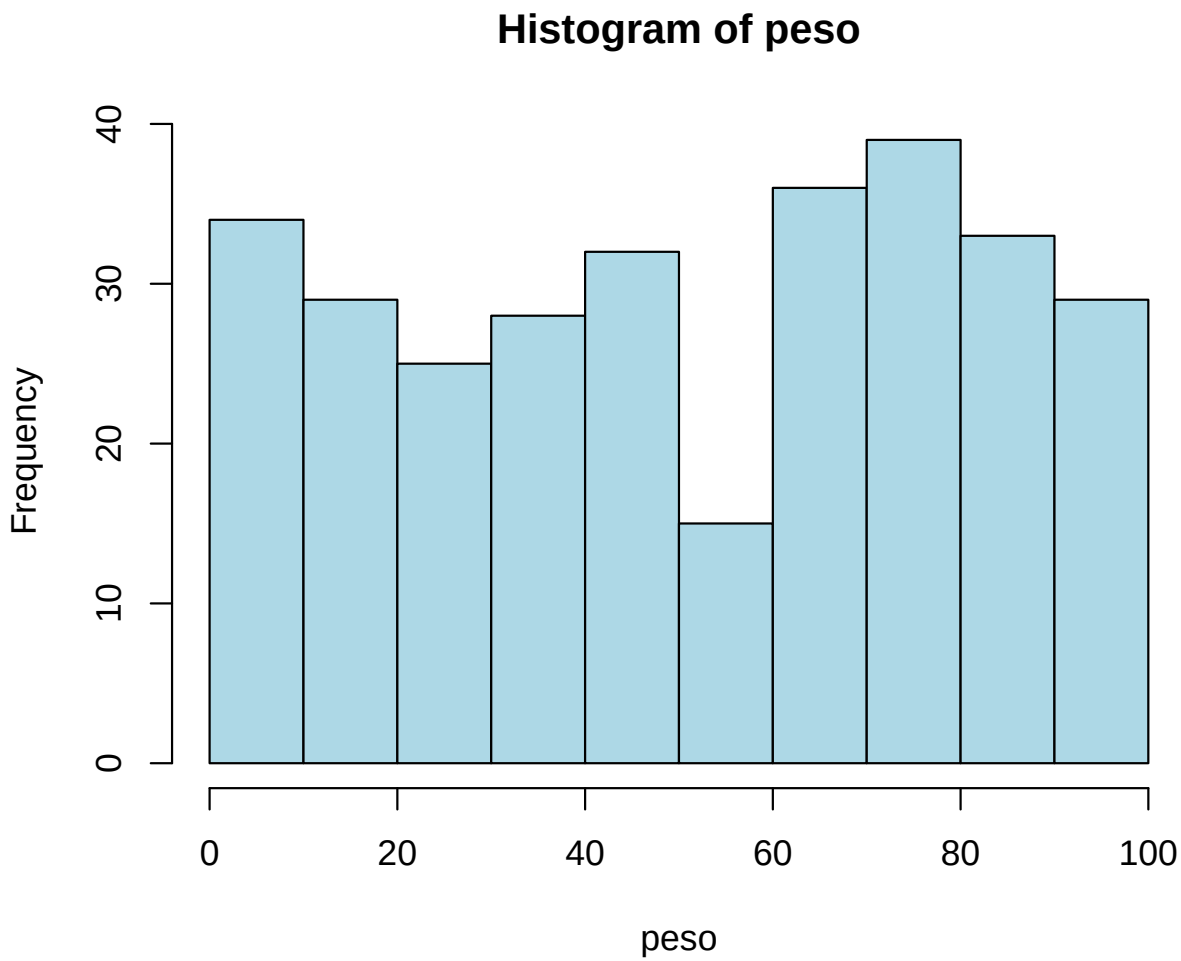


Histograma calibrado II

Ahora veamos el histograma de la variable peso.

```
> peso <- sample(1:100, 300, TRUE)
> hist(peso, col = "light blue")
```

Histograma calibrado II

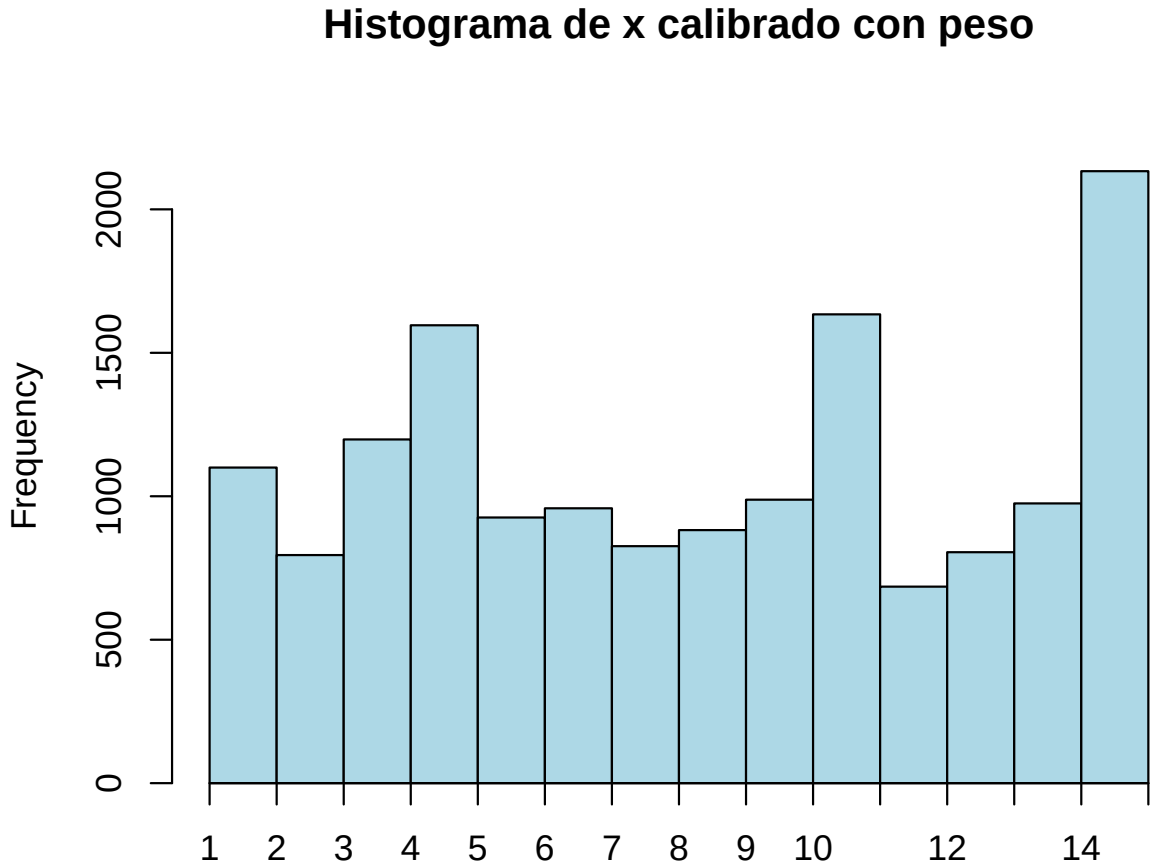


Histograma calibrado III

Juntemos las variables usando un histograma calibrado:

```
> weighted.hist(x, peso, breaks = 1:15,
+      main = "Histograma de x calibrado con peso",
+      col = "light blue")
```

Histograma calibrado III



Intro

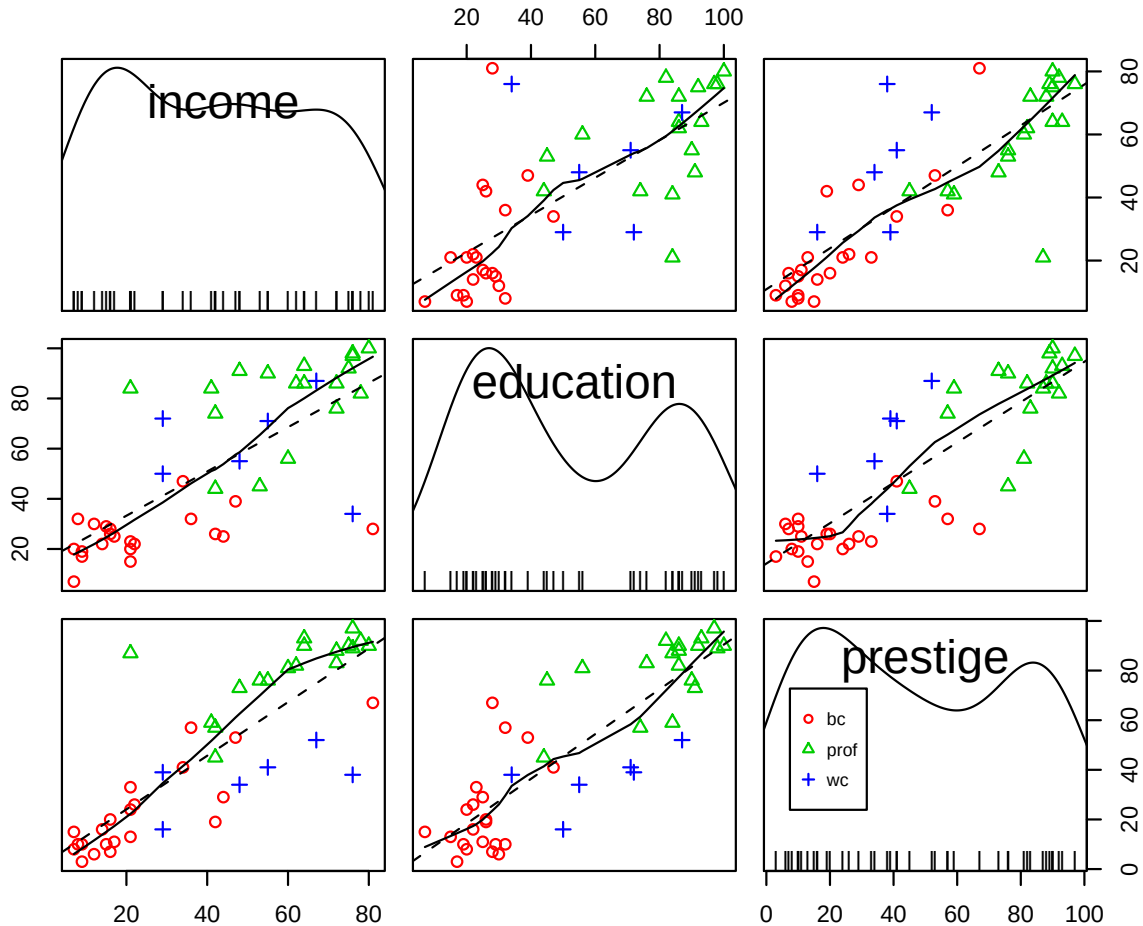
- Este es el último paquete gráfico que veremos.
- Acuérdense de explorar todos los ejemplos usando . . . ¿Cómo se llama la función?

scatterplot.matrix

- Cuando tienes tres[5] variables, la forma tradicional es hacer una gráfica con un cuadro con var1 vs var2, otro de var1 vs var3 y otro de var2 vs var3. A este tipo de gráfica se le llama un scatterplot.
- Para aprender la forma tradicional les recomiendo la siguiente práctica. Está excelente!
- Con plotrix podemos hacer un *scatterplot* aún más completo usando scatterplot.matrix:

```
> library(car)
> scatterplot.matrix(~income + education +
+     prestige | type, data = Duncan)
```

scatterplot.matrix



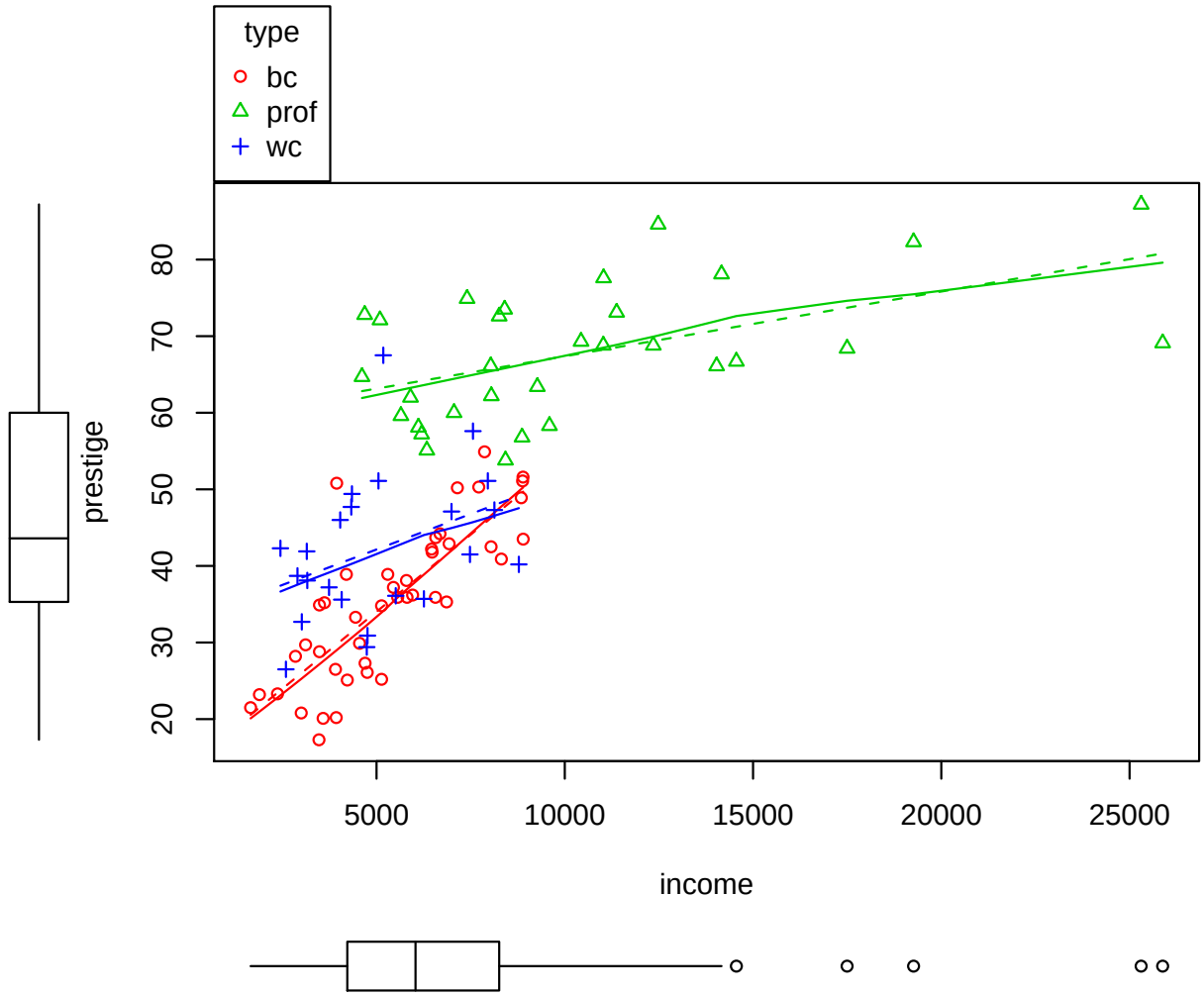
[5] O más.

scatterplot

- Chequen la ayuda de esa función porque hay varias opciones a parte de gráficas de densidad para los cuadros de var1 vs var1.
- Sin embargo, una función gráfica que me encantó es scatterplot.
- Con esta podemos agregar boxplots en los ejes! :D
- También nos agrega regresiones lineales para cada grupo de nuestros datos :)

```
> scatterplot(prestige ~ income |
+     type, data = Prestige, span = 1)
```

scatterplot



Ejercicios

1. Usando el objeto t2, hagan una gráfica de densidad de la variable *position* para cada cromosoma.
2. Usando los datos de los cromosomas X y Y[6] del objeto t2, hagan una gráfica de densidad para la variable *position* donde agrupen los datos por el variable *reference.allele*. Además, para cada cromosoma separen los datos por la variable *AK1.allele*. Van a tener que checar bien la ayuda de densityplot[7]. Su gráfica al final debe tener 8 paneles.
3. Usando el objeto t2 hagan una gráfica donde veamos la media de la variable *position* y su error estándar por cromosoma. En total van a ser 24 puntos con sus barras de error. Les mostré un ejemplo muy similar en la clase con el objeto t1.

[6] El argumento subset va a ser útil para esto.

[7] La clave está en la sintáxis de fórmula.

Si quieren más

- Los cursos que he dado en la LCG están en línea:
  1. `www.lcg.unam.mx/~lcollado/E/`
  2. `www.lcg.unam.mx/~lcollado/R/`
  3. `www.lcg.unam.mx/~lcollado/B/`
- Hay varios libros y sitios de interés :)
  Están en las secciones de material de apoyo.
- Fin de Principios de R![8]
- Vero continuará con UNIX :)

[8] Las soluciones a los ejercicios se las doy el próximo miércoles. Así que tienen tarea ;)

sessionInfo

Información de mi sesión:

```
> sessionInfo()

R version 2.10.0 (2009-10-26)
i686-pc-linux-gnu

locale:
 [1] LC_CTYPE=en_US.UTF-8
 [2] LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8
 [4] LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=C
 [6] LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8
 [8] LC_NAME=C
 [9] LC_ADDRESS=C
[10] LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8
[12] LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices
```

sessionInfo

```
[4] utils     datasets  methods
[7] base

other attached packages:
[1] car_1.2-16
[2] plotrix_2.7-2
[3] DAAG_1.00
[4] randomForest_4.5-33
[5] rpart_3.1-45
[6] MASS_7.3-3
[7] lattice_0.17-26

loaded via a namespace (and not attached):
[1] grid_2.10.0  tools_2.10.0
```