Introduction to the R software

Chapter 2. Graphical Exploration of Data

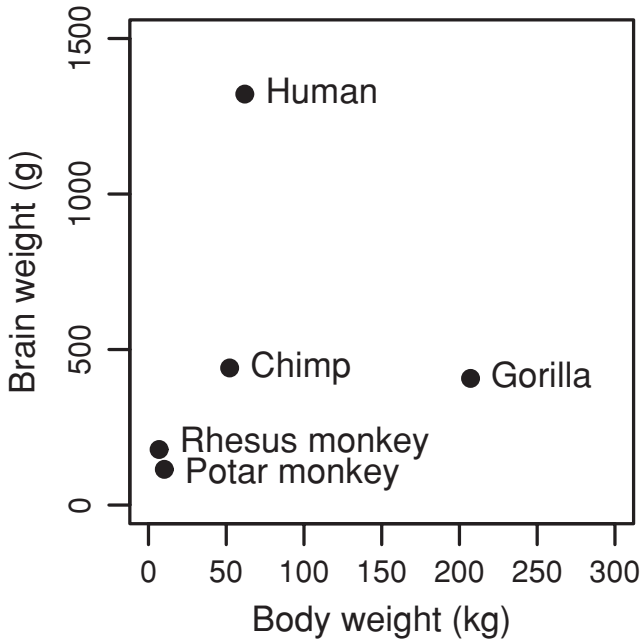
Peng Zhang

School of Mathematical Science
Zhejiang University

Outline

The function plot()

```
plot(Brainwt ~ Bodywt, data=primates) # plot(y~x) syntax
with(primates, plot(Bodywt, Brainwt)) # plot(x, y) syntax
```



	Bodywt	Brainwt
Potar monkey	10.0	115
Gorilla	207.0	406
Human	62.0	1320
Rhesus monkey	6.8	179
Chimp	52.2	440

Adding points, lines, text, and axis annotation

- Put labels on points.

```
## Place labels on points
plot(Brainwt ~ Bodywt, xlim=c(0, 300), data=primates)
# Specify xlim so that there is room for the labels
with(primates,
text(Brainwt ~ Bodywt, labels=row.names(primates), pos=4))

# pos=4 places text to the right of the points. Other
# possibilities are: 1: below; 2: to the left; 3: above
```

- Use `points()` to add points to a plot.
- Use `lines()` to add lines.

Parameter settings

- The function `mtext(text, side, line, ...)` adds text in the margin of the current plot. The sides are numbered 1 (x-axis), 2 (y-axis), 3 (top), and 4 (right vertical axis). By default, `adj=0.5`, which centers the text at the axis midpoint. Specify `adj=0` to position the left extreme of the text at the left margin, and `adj=1` to position its right extreme at the right margin.
- The `axis()` function gives fine control over axis ticks and labels. To use for the x-axis, plot the initial graph with `xaxt="n"`. Then call `axis()` with the argument `side=1`, and with other arguments as required. See `help(axis)` for details.
- Plotting symbols: `pch` (choice of symbol); `cex` ("character expansion"); `col` (color). Thus `par(cex=1.2)` increases the plot symbol size 20% above the default.
- Lines: `lty` (line type); `lwd` (line width); `col` (color).

Parameter settings

- Axis limits: `xlim; ylim`. (Assuming `xaxs="r"`, x-axis limits are by default extended by 4% relative to the data limits. Specify `xaxs="i"` to make the default an exact fit to the data limits. For the y-axis, replace `xaxs` by `yaxs`.)
- Axis annotation and labels: `cex.axis` (character expansion for axis annotation, independently of `cex`); `cex.labels` (size of the axis labels); `mgp` (margin line for the axis title, axis labels, and axis line; default is `mgp=c(3, 1, 0)`).
- Graph margins: `mar` (inner margins, clockwise from the bottom; the out-of-the-box default is `mar=c(5.1, 4.1, 4.1, 2.1)`, in lines out from the axis); `oma` (outer margins, relevant when there are multiple graphs on the one graphics page).
- Plot shape: `pty="s"` gives a square plot (must be set using `par()`).
- Multiple graphs on the one graphics page: Specify `par(mfrow=c(m,n))` to get an m rows by n columns layout of graphs on a page.

Expressions and mathematical symbols

```
 symbols(x=1.5, y=0, circles=1.2, xlim=c(0,3),
    ylim=c(-1.5,1.5), bg="gray", inches=FALSE)
    # inches=FALSE ensures that radius is in x-axis units
 text(1.5, 0.5, expression("Area" ==
pi*phantom("'")*italic(r)^2))
    # Use '==' to insert '='.
    # Text or symbols that appear either side of '*' are
juxtaposed.
    # Notice the use of phantom("'") to insert a small
space.
```

Identification and location on the figure region

Following the drawing of the initial graph, the two functions that may be used are:

- `identify()` labels points;
- `locator()` prints the co-ordinates of points.
- For `identify()` the default setting is the number of data points, while for `locator()` the default is 500.

```
plot(Brainwt ~ Bodywt, data=primates)
with(primates, identify(Brainwt ~ Bodywt,
      labels=row.names(primates), n=2))
# Now click near 2 plotted points
```

Goals of today lecture

Describing the instructions for

- manipulating graphics windows;
- drawing simple plots;
- dealing with colors;
- adding text on graphs;
- improving titles, axes and captions;
- interacting with plots;
- fine-tuning graphical parameters.

Basic stuff

To open a new graphical window, use the instructions `windows()` or `win.graph()` (for MacOS, use `quartz()`).

To close some graphical device, use: `dev.off(`*device-number*`)`.

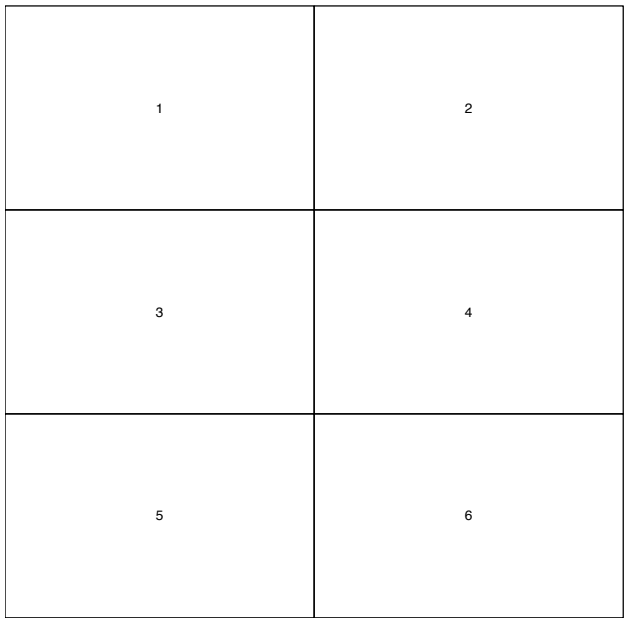
To save a plot that has already been drawn, use: `savePlot(filename="Rplot",type="pdf")`.

You can also proceed as follows:

```
> pdf(file="mygraph.pdf")
> hist(runif(100))
> dev.off()
```

Splitting the Graphics Window

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

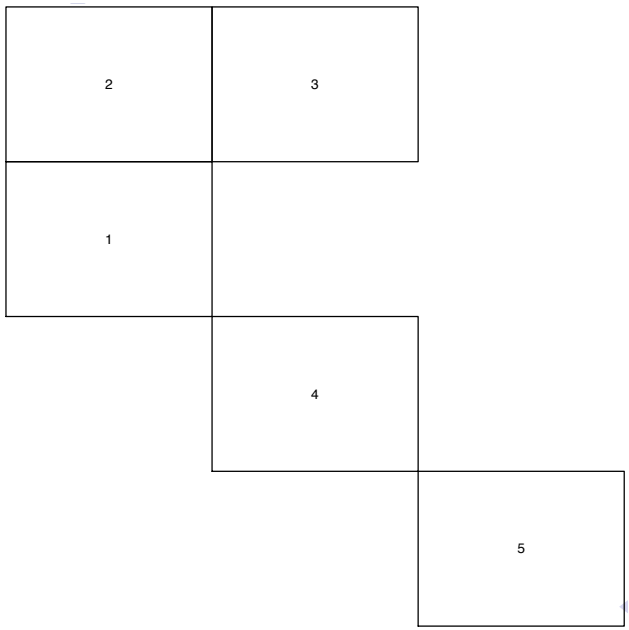



```
> par(mfcol=c(3,2)) # by columns.
```

Splitting the Graphics Window

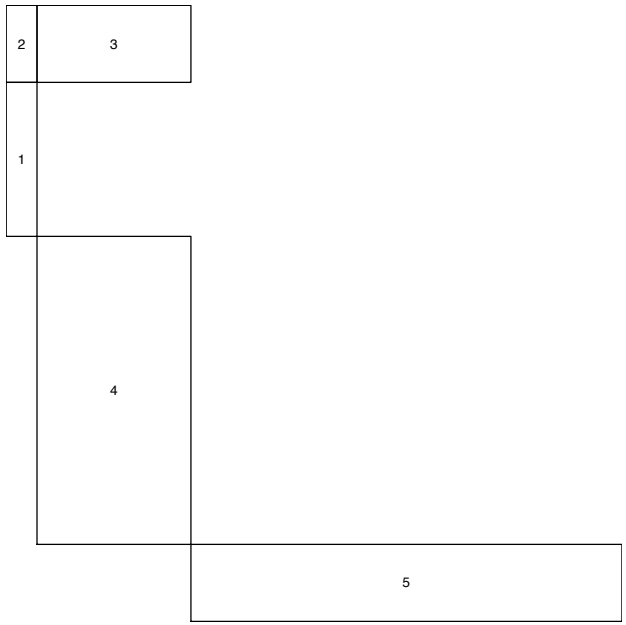
```
>(mat <- matrix(c(2,3,0,1,0,0,0,4,0,0,0,5),4,3,byrow=T))
     [,1] [,2] [,3]
[1,]    2    3    0
[2,]    1    0    0
[3,]    0    4    0
[4,]    0    0    5
```

```
> layout(mat) ; layout.show(5)
```



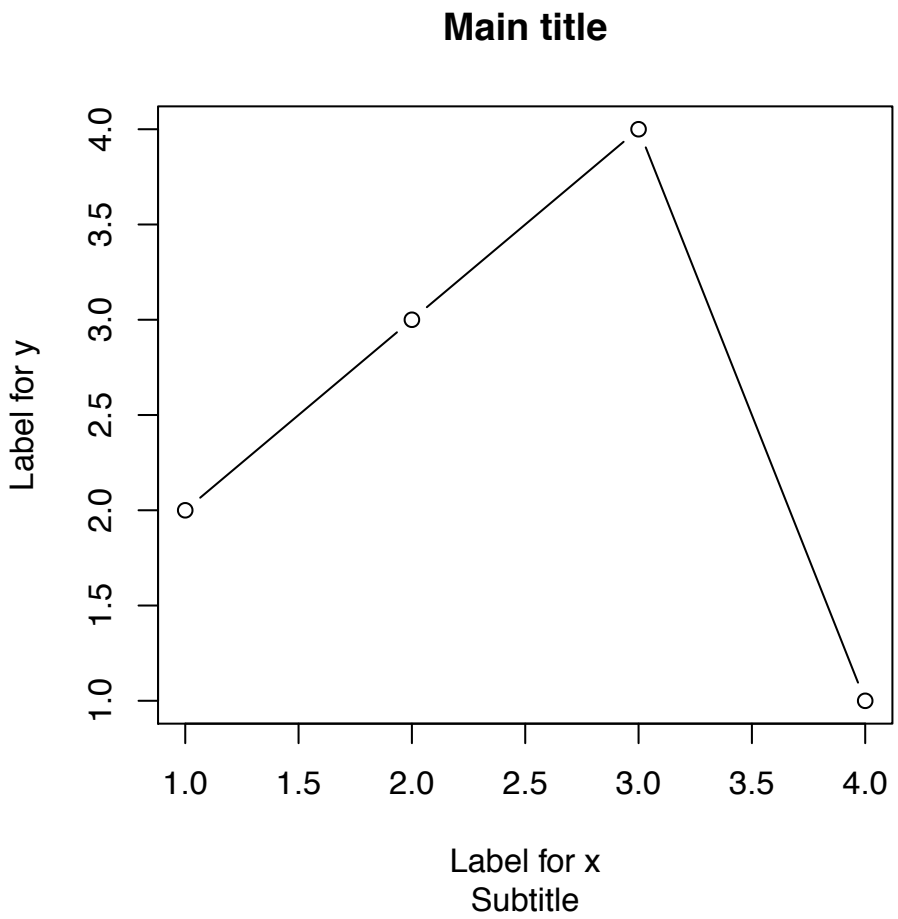
Splitting the Graphics Window

```
> layout(mat,widths=c(1,5,14),heights=c(1,2,4,1))
> layout.show(5)
```



The functions plot() and points()

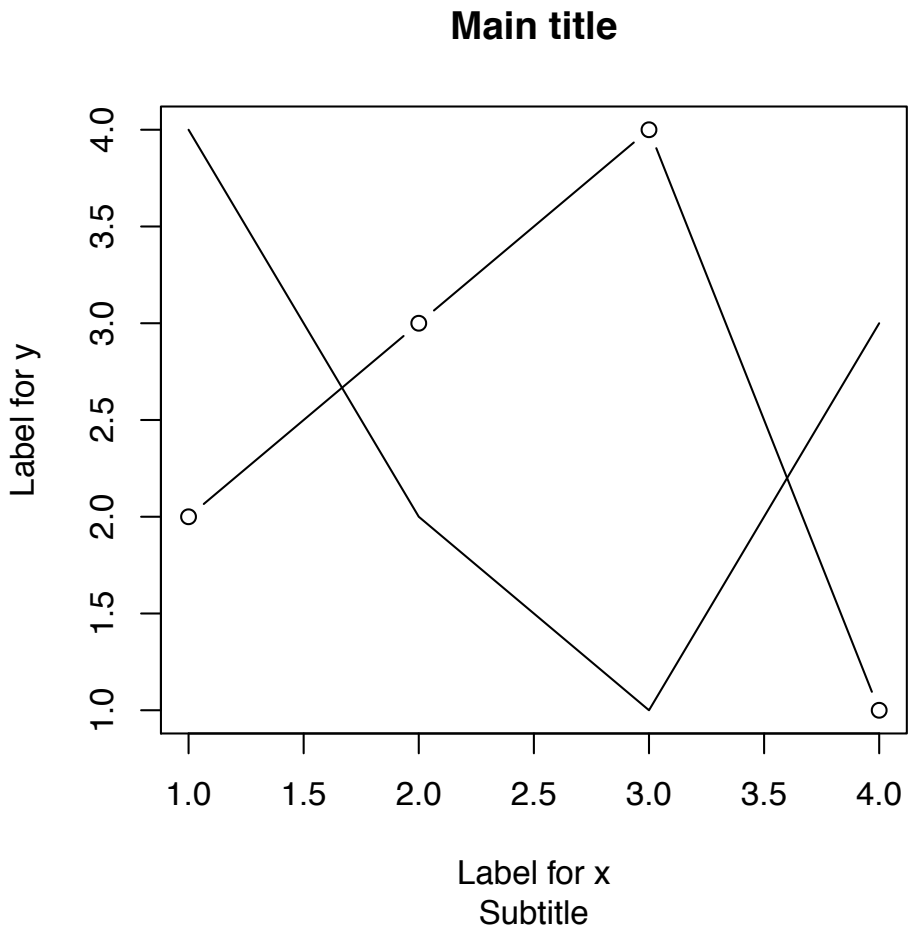
```
> plot(1:4,c(2,3,4,1),type="b",main="Main title",
+   sub="Subtitle",xlab="Label for x",ylab="Label for y")
```



Argument	Description
`x`	Vector of x coordinates of points to draw.
`y`	Vector of y coordinates of points to draw.
`type`	Specify the type of plotting: `"p"` for points, `"l"` for lines, `"b"` for both, `"c"` for empty points joined by lines, `"o"` for overplotted points and lines, `"h"` for vertical lines, `"s"` for stair steps and `"n"` to plot nothing (but to display the window, with axes).
`main`	Specify the main title.
`sub`	Specify the subtitle.
`xlab`	Specify the label of the x axis.
`ylab`	Specify the label of the y axis.
`xlim`	Vector of length 2. Specify the lower and upper bound for the x axis.
`ylim`	Vector of length 2. Specify the lower and upper bound for the y axis.
`log`	Character string which contains `"x"` (respectively `"y"`, `"xy"` or `"yx"`) if the x axis (respectively the y axis, both) is to be logarithmic.

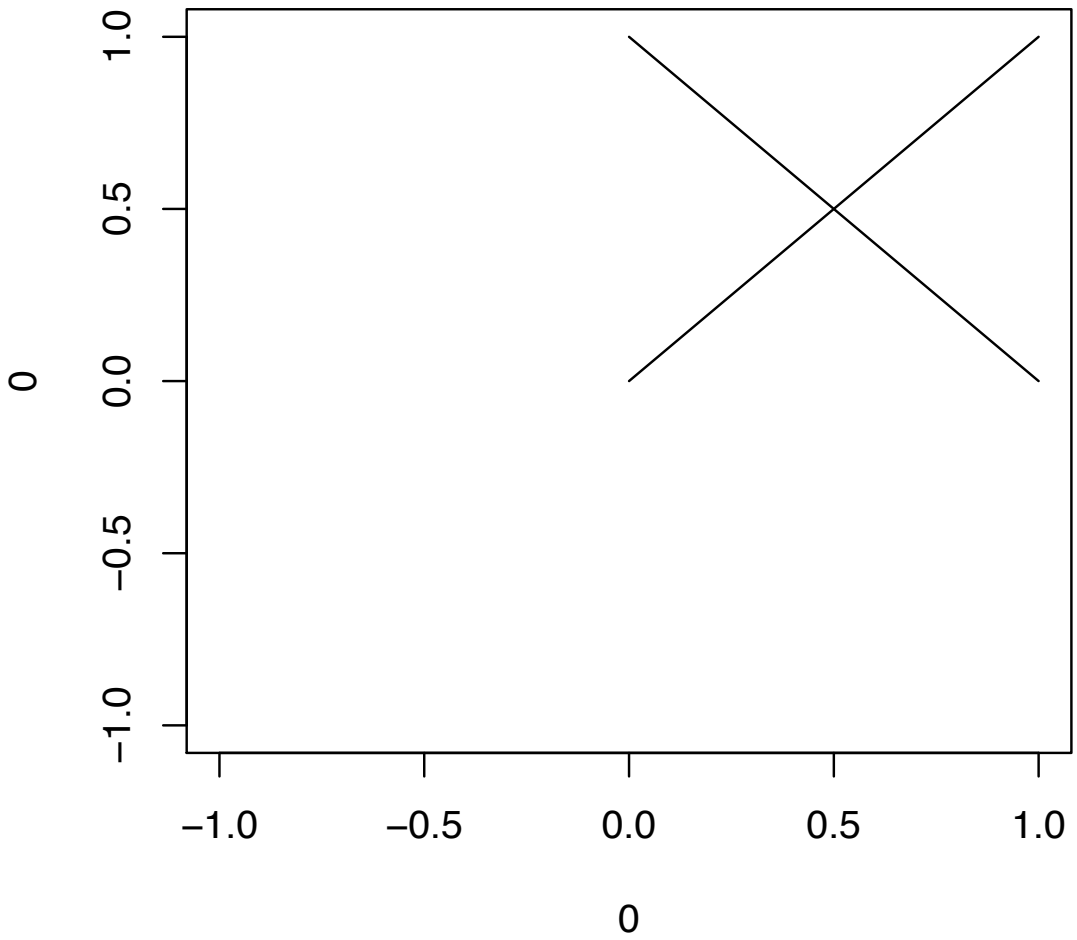
Note that successive calls of the function `plot()` create a new plot every time, which replaces the previous one (unless the graphics window has been split, as explained above). The function `points()` can remediate this issue by overlaying the new plot on top of the old one. It takes the same arguments as `plot()`.

```
> points(1:4,c(4,2,1,3),type="l")
```



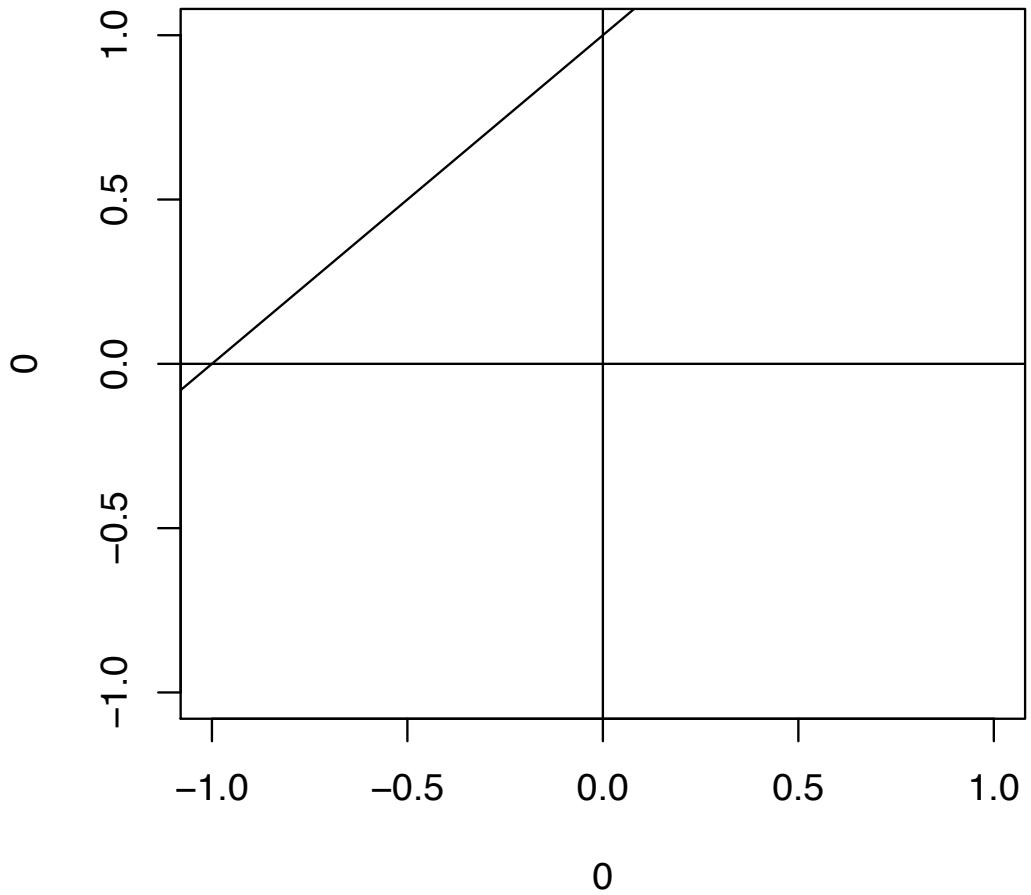
The functions segments(), lines() and abline()

```
> plot(0,0,"n")
> segments(x0=0,y0=0,x1=1,y1=1)
> lines(x=c(1,0),y=c(0,1))
```



The function `abline()` is used to draw a straight line of equation $y = a + bx$ (specified by the arguments `a` and `b`), or a horizontal (argument `h`) or vertical (argument `v`) line.

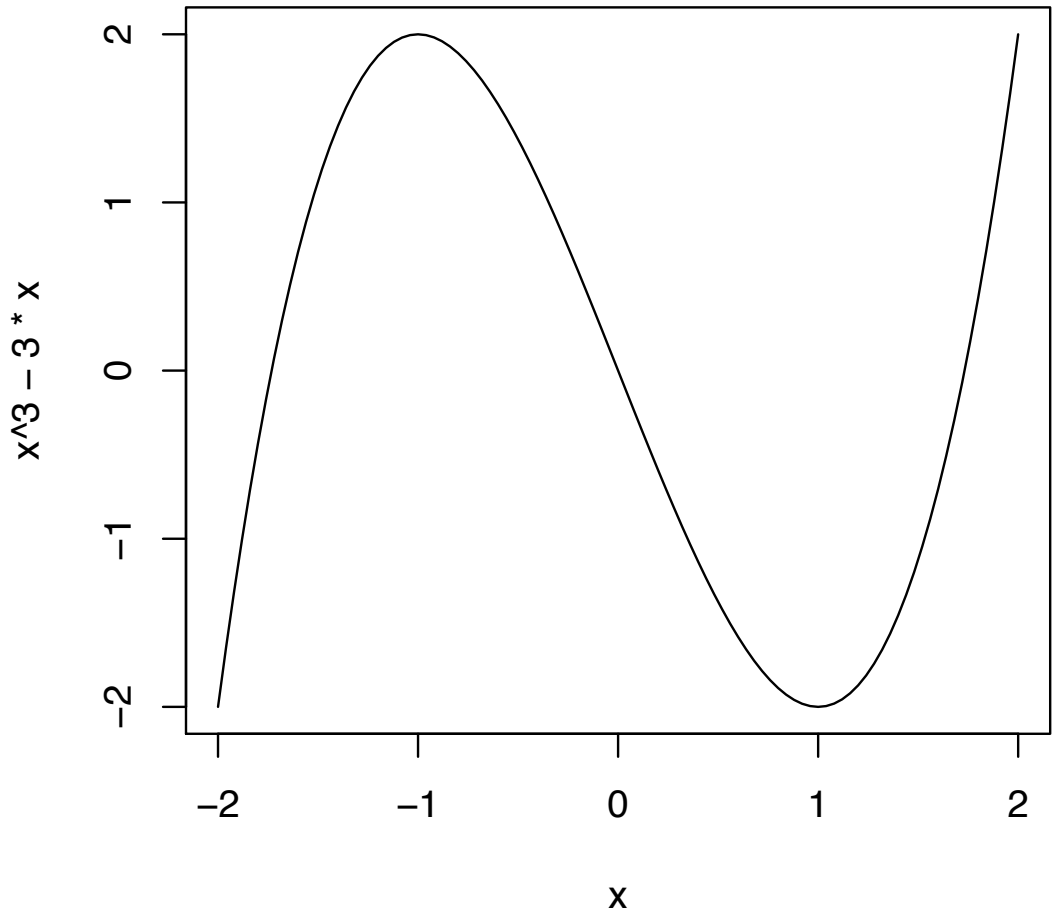
```
> plot(0,0,"n");abline(h=0,v=0);abline(a=1,b=1)
```



The function curve()

This function is used to draw a curve in a Cartesian coordinate system, on the interval specified by the bounds `from` and `to`.

```
> curve(x^3-3*x,from=-2,to=2)
```



The function colors()

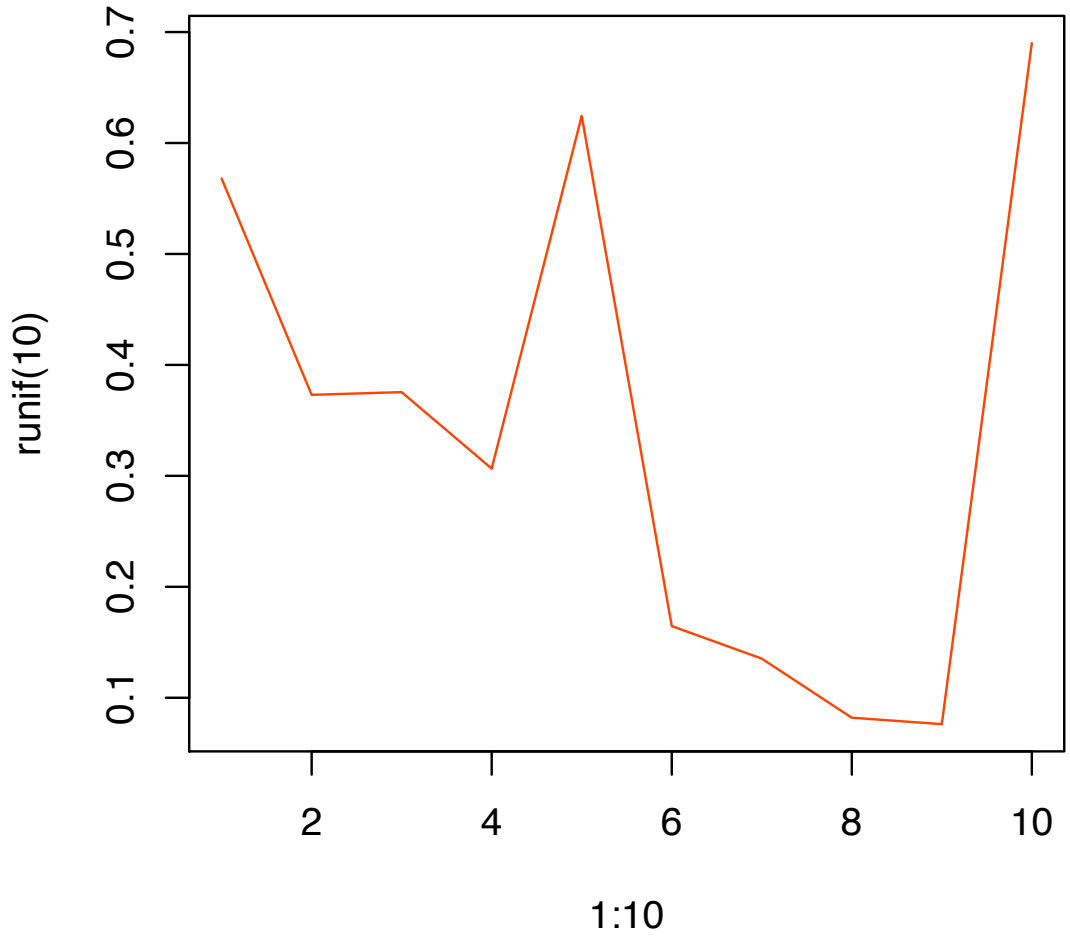
This function returns the names of the 657 colours known to **R**.

If you want to get the different shades of orange, you can use the instruction:

```
> colors()[grep("orange",colors())]
 [1] "darkorange"  "darkorange1" "darkorange2" "darkorange3"
 [5] "darkorange4" "orange"      "orange1"     "orange2"
 [9] "orange3"     "orange4"     "orangered"   "orangered1"
[13] "orangered2"  "orangered3"  "orangered4"
```

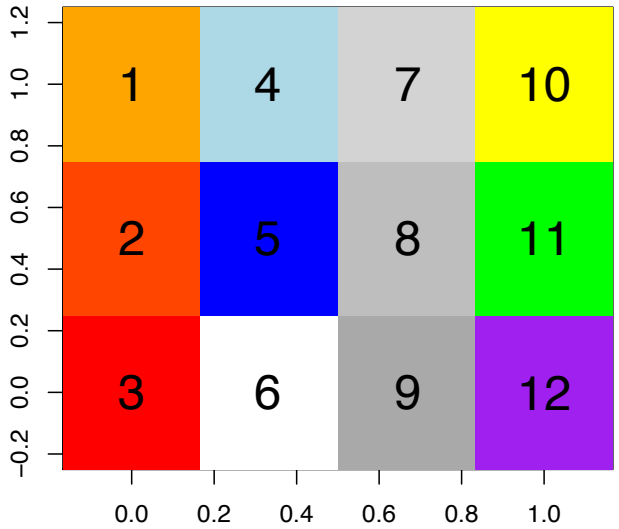
These colours can be used in your plots, for example with the argument `col` of the function `plot()`.

```
> plot(1:10,runif(10),type="l",col="orangered")
```



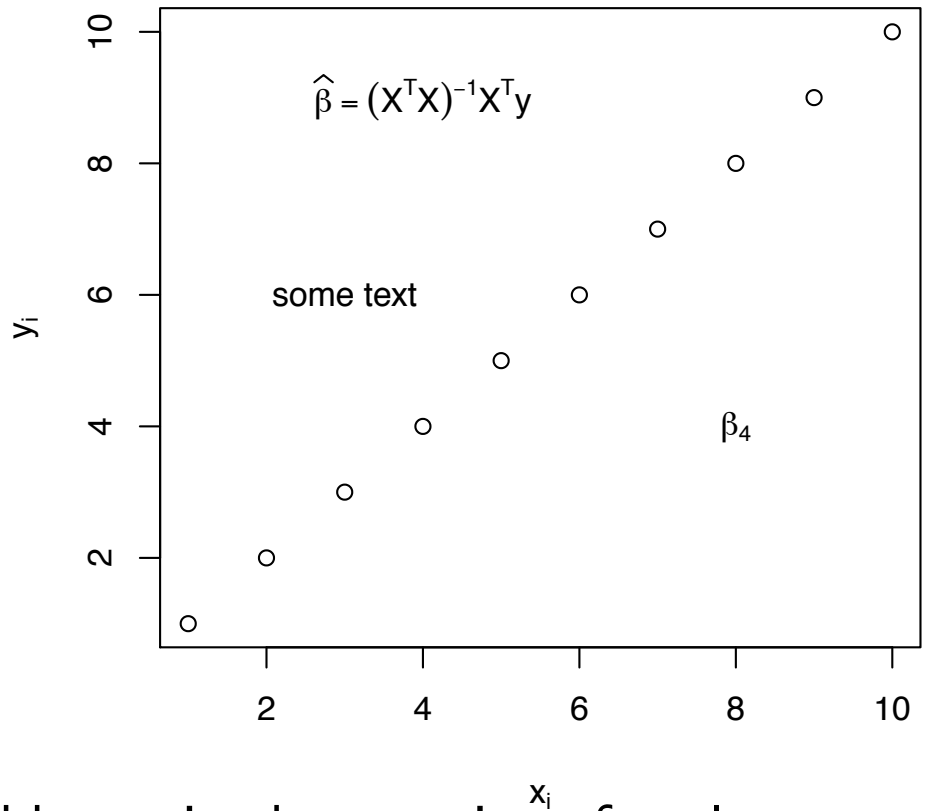
The function image()

```
> (X <- matrix(1:12,nrow=3))
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> colours <- c("orange","orangered","red","lightblue",
+                    "blue", "white","lightgrey","grey",
+                    "darkgrey","yellow","green","purple")
> image(as.matrix(rev(as.data.frame(t(X)))),col=colours)
> text(rep(c(0,0.33,0.67,1),each=3),rep(c(1,0.5,0),4),1:12,cex=2)
```



The function text()

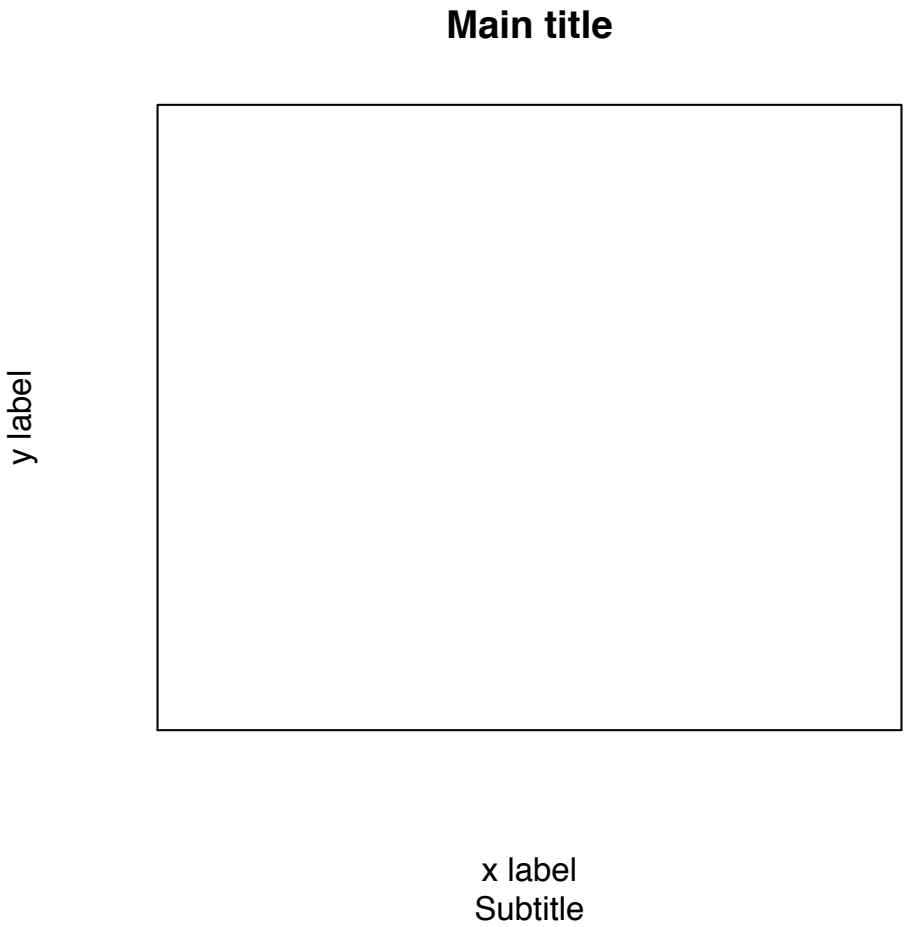
```
> plot(1:10,1:10,xlab=bquote(x[i]),ylab=bquote(y[i]))
> text(3,6,"some text")
> text(4,9,expression(widehat(beta) == (X^T * X)^{-1} * X^T * y))
> p <- 4; text(8,4,bquote(beta[.(p)])) # Combining "math" and
                                       # numerical variables.
```



Use `mtext()` to add text in the margins of a plot.

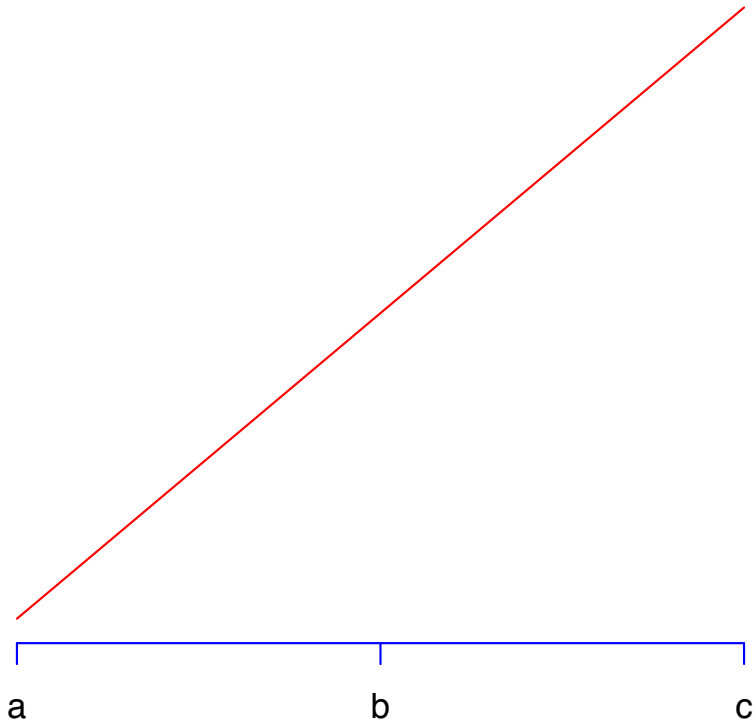
The function `title()`

```
> plot.new()
> box()
> title(main = "Main title", sub = "Subtitle",
+           xlab = "x label", ylab = "y label")
```



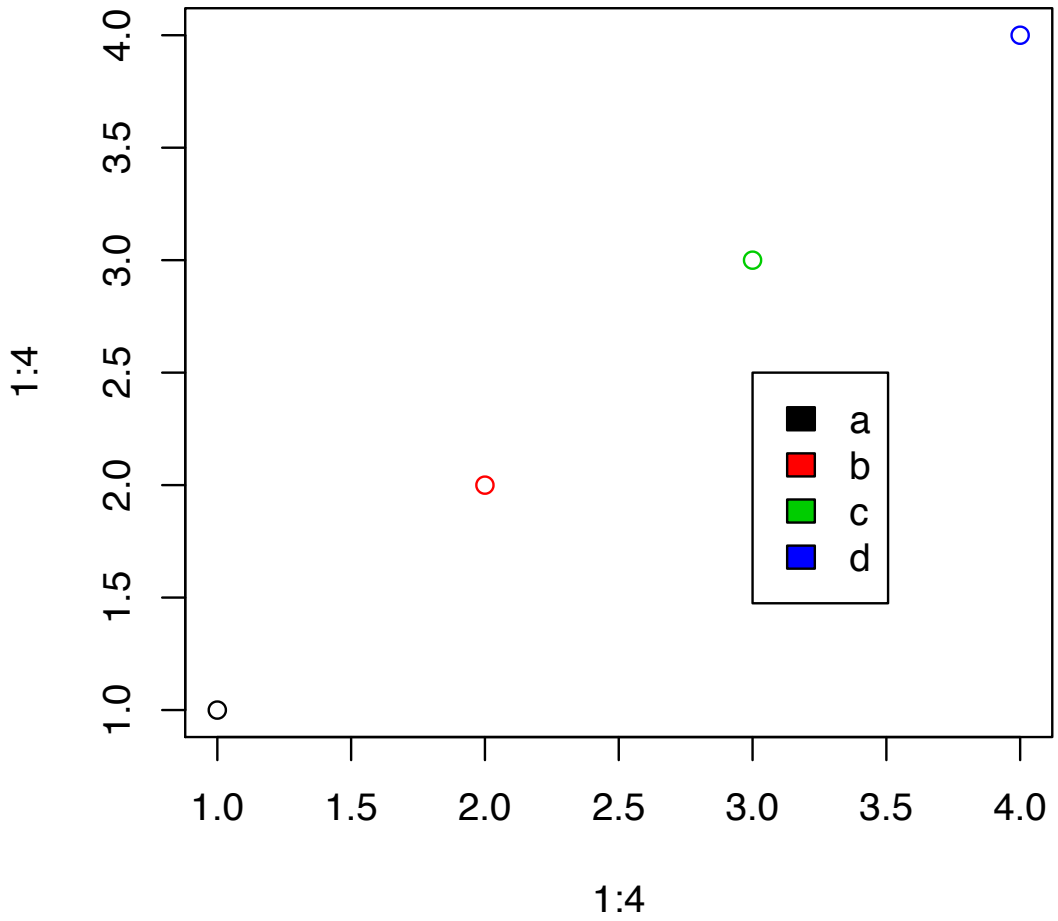
The function axis()

```
> plot.new()
> lines(x=c(0,1),y=c(0,1),col="red")
> axis(side=1,at=c(0,0.5,1),labels=c("a","b","c"),
+          col="blue")
```

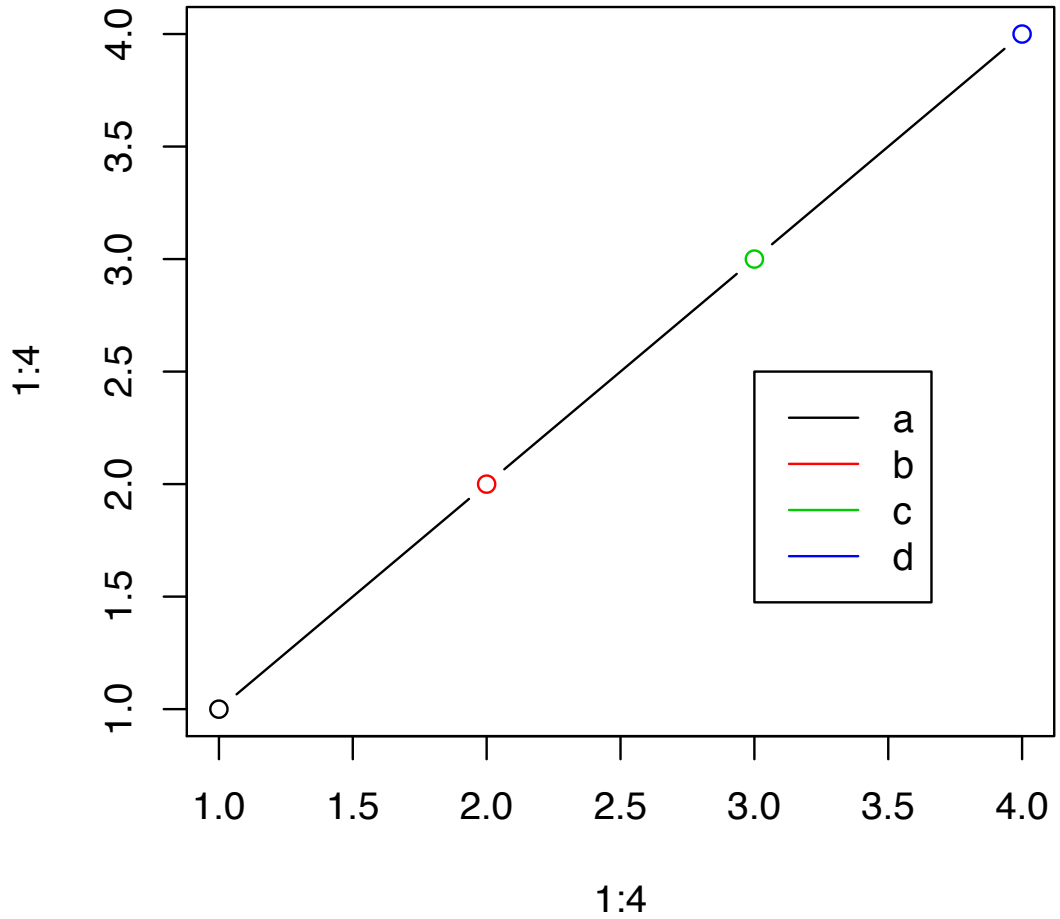


The function legend()

```
> plot(1:4,1:4,col=1:4)
> legend(x=3,y=2.5,legend=c("a","b","c","d"),fill=1:4)
```



```
> plot(1:4,1:4,col=1:4,type="b")
> legend(x=3,y=2.5,legend=c("a","b","c","d"),col=1:4,
+        lty=1)
```



The function `locator()`

It is used to place a point on a plot, or to get its coordinates with a click of the mouse. It can also be useful to add text (or a caption) at a specific location, thanks to the mouse.

Enter the following instructions, then click anywhere on the plot you get.

```
plot(1,1)
text(locator(1),labels="Here") # Click on the window.
```

The function identify()

It is used to identify and mark points already present on a plot.

Enter the following instructions, then click next to points on the plot. Use a right-click to exit the interactive mode.

```
> plot(swiss[,1:2])
> x <- identify(swiss[,1:2],labels=rownames(swiss))
> x
```

The `par()` function

The function `par()` takes many arguments to fine-tune your plots. Use this function to set (or query) general graphical parameters.
Here is how to use this instruction:

- `par(`*arg-name*`)` outputs the default value of the parameter *arg-name* of the function `par()`;
- `par(`*arg-name*`=val)` changes the value of the parameter *arg-name* to the value `val`;
- `par()` returns the list of all graphical parameters currently in use, as well as the current values.

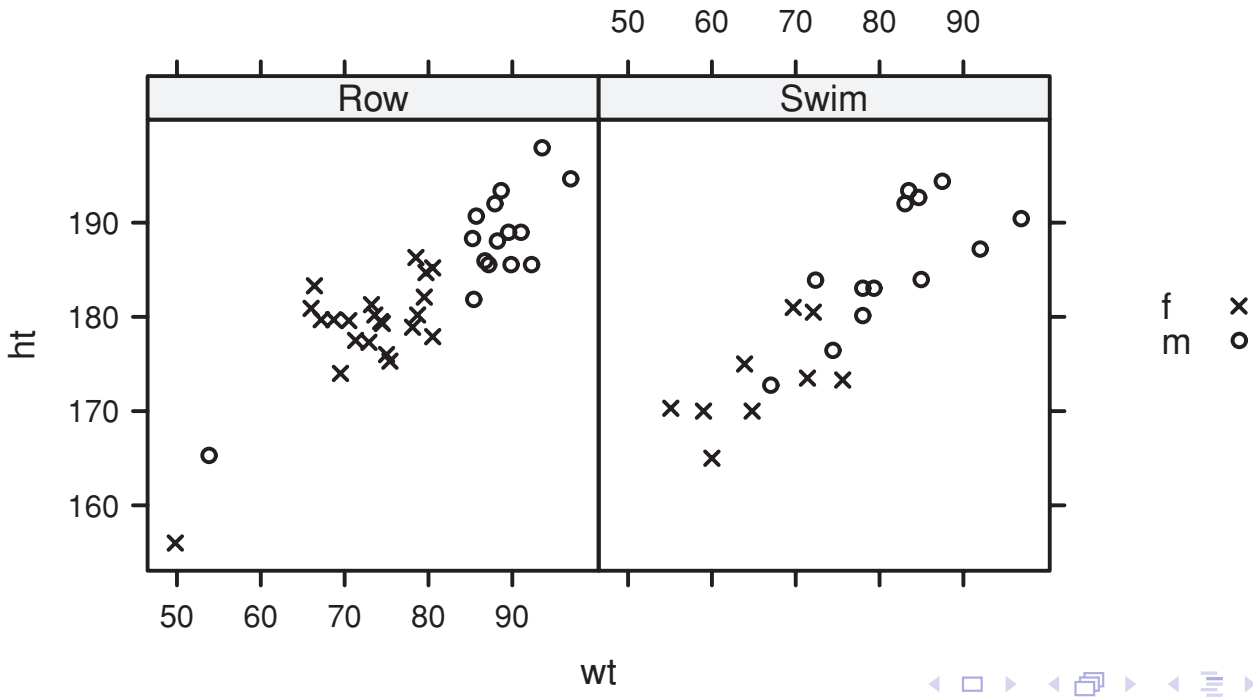
Before changing the values of parameters of the function `par()`, you should save the old values. That way, you can restore them later if needed.

```
# Save the default values of par().
save.par <- par(no.readonly = TRUE)
# Now we can change some parameters.
par(bg="red")
# Then restore the old values.
par(save.par)
```

There are a lot (really!) different argument for the function `par()`. See `help("par")` or the book.

Lattice (trellis) graphics

```
trellis.device(color=FALSE)
xyplot(ht~wt|sport,groups=sex,pch=c(4,1),aspect=1,data=ais,
auto.key=list(columns=2),subset=sport%in%c("Row","Swim"))
dev.off() # Close device
trellis.device() # Start new device, by default with color
```



Explanation

- In the graphics formula `ht ~ wt | sport`, the vertical bar indicates that what follows, in this case `sport`, is a conditioning variable or factor. The graphical information is broken down according to the factor levels or distinct values.
- The parameter `aspect` controls the ratio of dimensions in the y and x directions.
- The setting `auto.key=list(columns=2)` generates a simple key, with the two key items side by side in two columns rather than one under another in a single column as happens with the default setting `columns=1`.

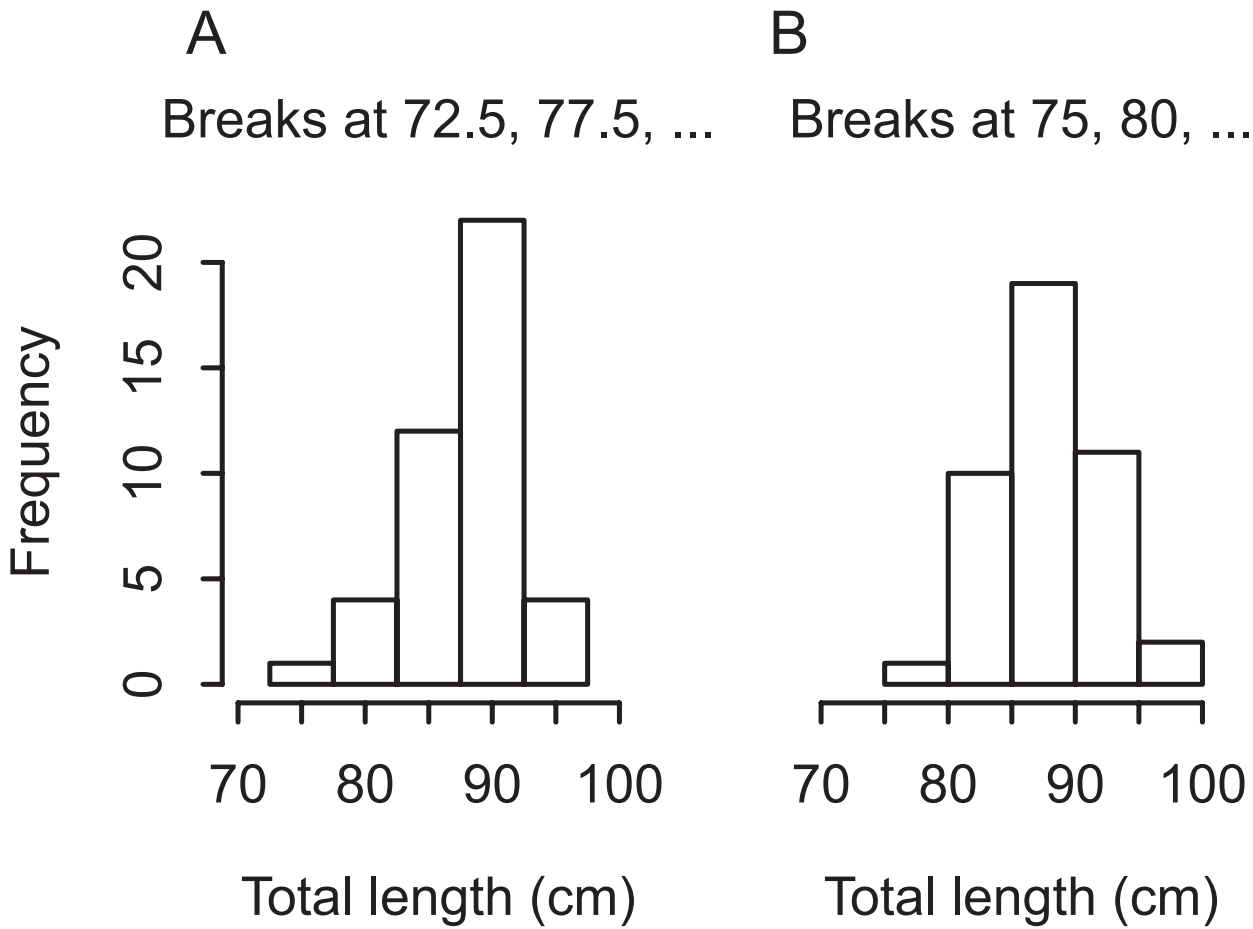
Selected lattice functions

```
dotplot(factor ~ numeric,..)          # 1-dim. Display
stripplot(factor ~ numeric,..)        # 1-dim. Display
barchart(character ~ numeric,..)
histogram( ~ numeric,..)
densityplot( ~ numeric,..)            # Density plot
bwplot(factor ~ numeric,..)           # Box and whisker plot
qqmath(factor ~ numeric,..)           # normal probability plots
splom( ~ dataframe,..)                # Scatterplot matrix
parallel( ~ dataframe,..)             # Parallel coordinate plot
cloud(numeric ~ numeric * numeric, ...) # 3D surface
wireframe(numeric ~ numeric * numeric, ...) # 3D
scatterplot
```

Histograms and density plots

```
library(DAAG) # Ensure that the DAAG package is attached
## Form the subset of possum that holds data on females
fossum <- subset(possum, sex=="f")
attach(fossum)
hist(totlngth, breaks = 72.5 + (0:5)*5, ylim = c(0, 22),
xlab="Total length (cm)",main="A: Breaks at 72.5,77.5,
...")
hist(totlngth, breaks = 75 + (0:5) * 5, ylim = c(0, 22),
xlab="Total length (cm)", main="B: Breaks at 75, 80, ...")
```

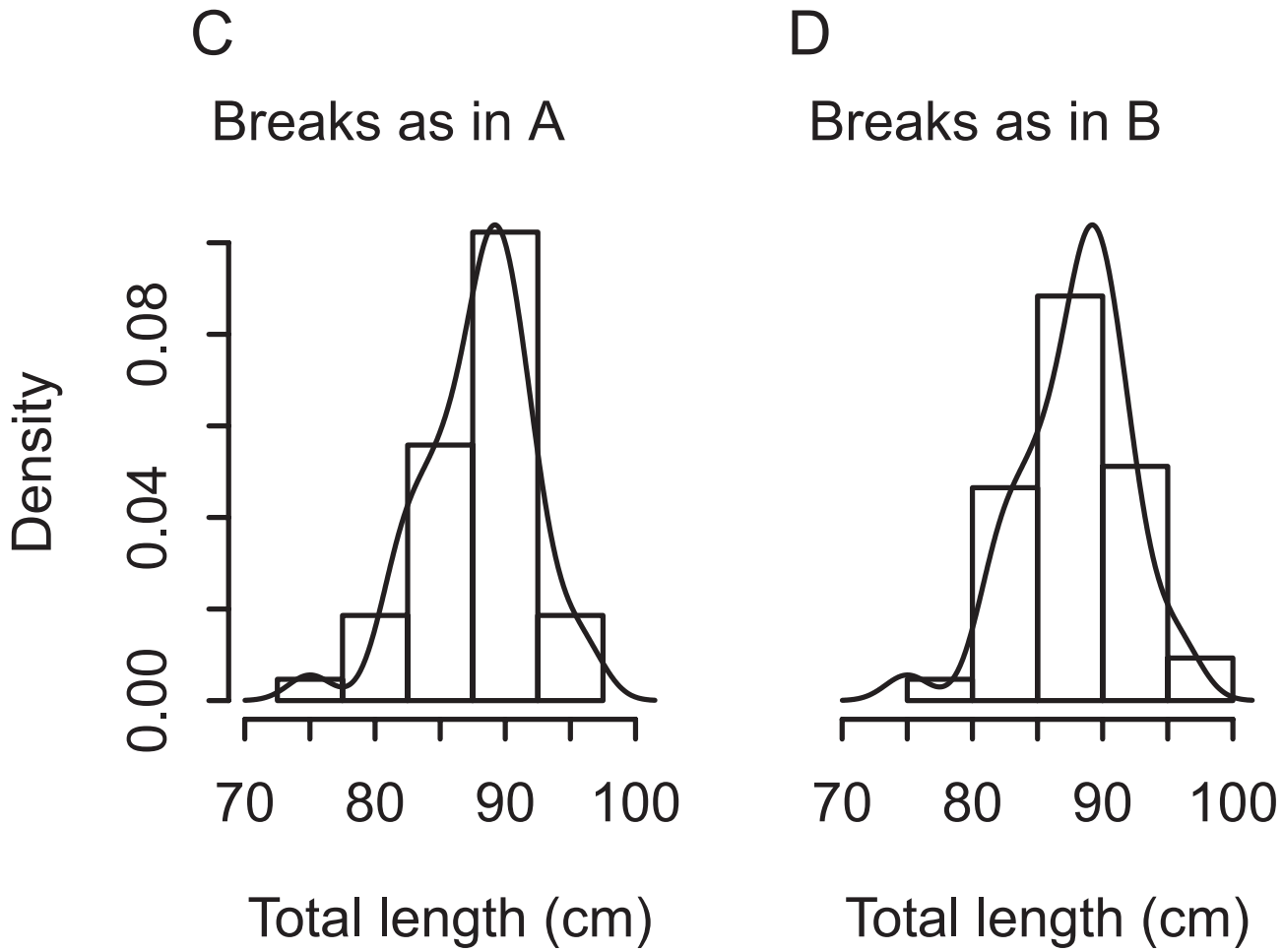
Graph


A
Breaks at 72.5, 77.5, ...
Frequency
0
5
10
15
20
70
80
90
100
Total length (cm)
B
Breaks at 75, 80, ...
70
80
90
100
Total length (cm)

Histograms and density plots

```
dens <- density(totlngth)
xlim <- range(dens$x)
ylim <- range(dens$y)
hist(totlngth, breaks = 72.5+(0:5)*5, probability = T,
xlim=xlim, ylim=ylim, xlab="Total length (cm)", main=" ")
lines(dens)
hist(totlngth, breaks = 75 + (0:5) * 5, probability = T,
xlim=xlim, ylim=ylim, xlab="Total length (cm)", main=" ")
lines(dens)
par(mfrow=c(1,1))
detach(fossum)
```

Graph

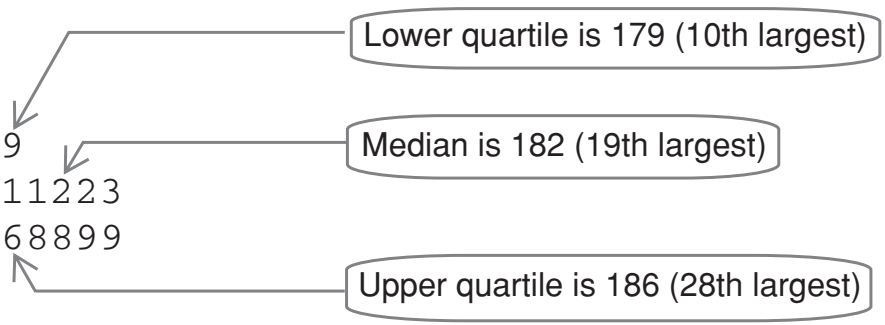

C
Breaks as in A
D
Breaks as in B
Density
0.00
0.04
0.08
70
80
90
100
Total length (cm)
70
80
90
100
Total length (cm)

The stem-and-leaf display

```
with(ais, stem(ht[sport=="Row"]))
```

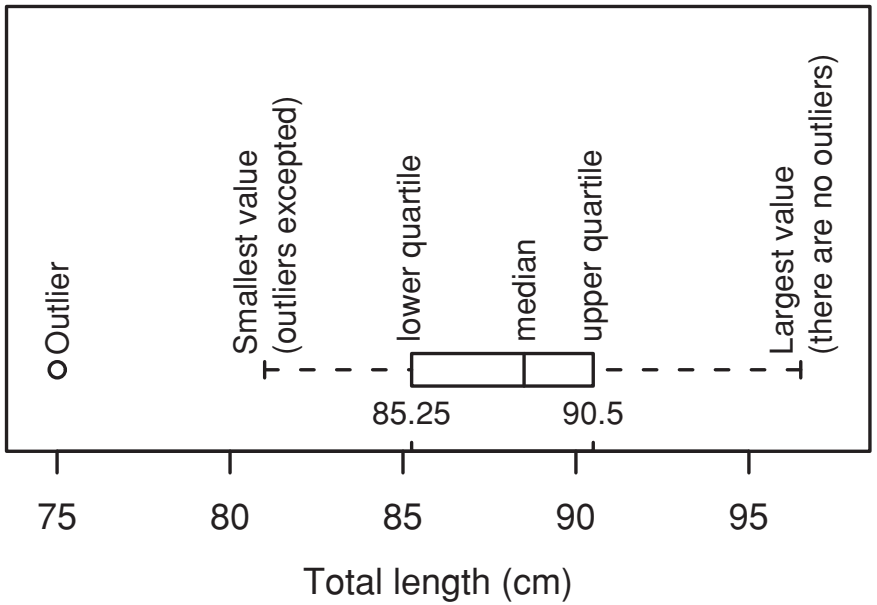
```
The decimal point is 1 digit(s) to the right of the |

15 | 6
16 |
16 | 5
17 | 4
17 | 5678899
18 | 00000011223
18 | 55666668899
19 | 123
19 | 58
```



Boxplots

```
## Base graphics boxplot function
with(fossum, boxplot(totlngth, horiz=TRUE))
## Alternative: lattice graphics bwplot function
bwplot(~ totlngth, data=fossum)
```



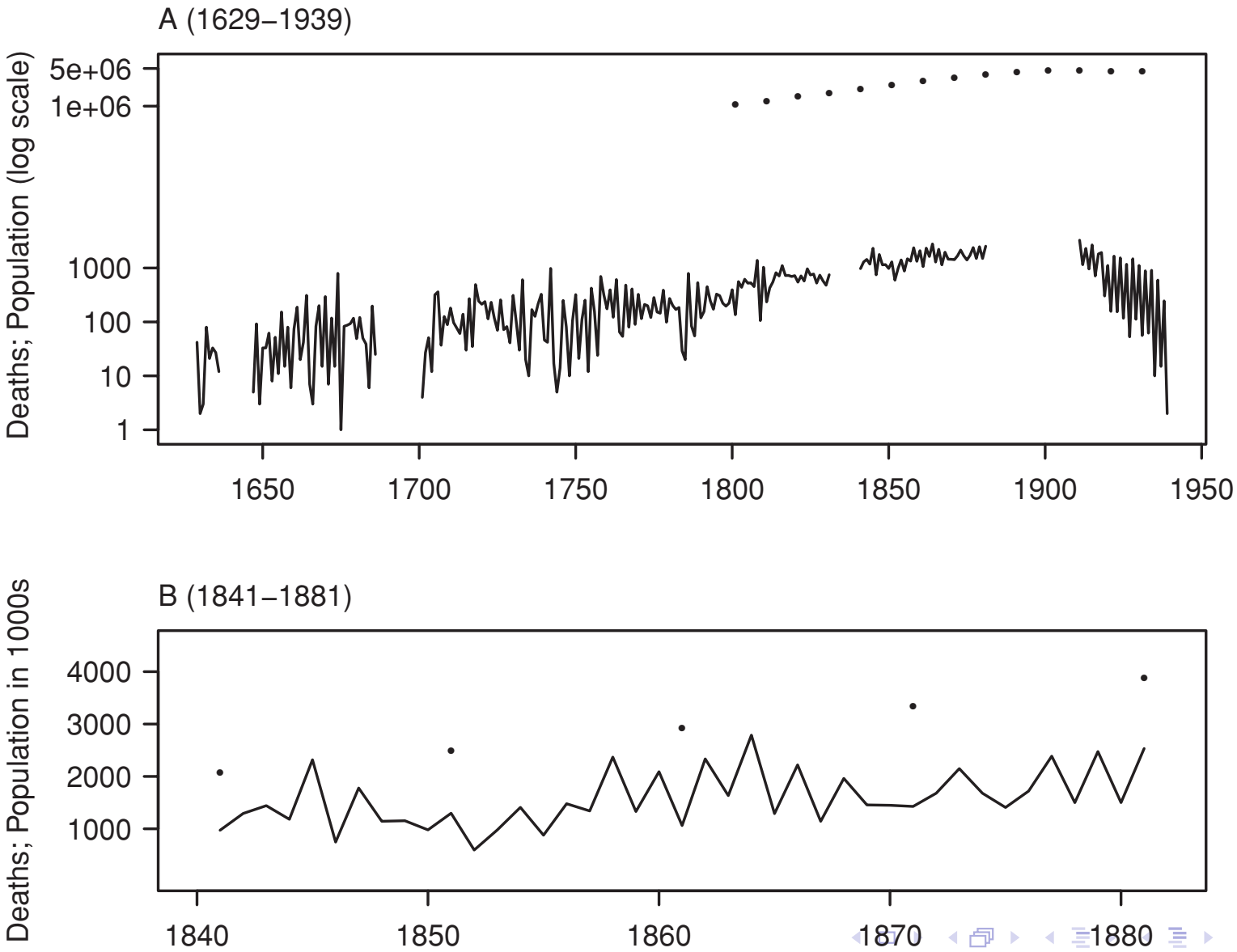
Inter–quartile range
= 90.5 – 85.25
= 5.25

Compare
0.75 x Inter–quartile range
= 3.9
with standard deviation
= 4.2

Univariate time series

```
## Panel A
plot(log10(measles),xlab="",ylim=log10(c(1,5000*1000)),
ylab="Deaths; Population (log scale)", yaxt="n")
ytiks <- c(1, 10, 100, 1000, 1000000, 5000000)
## London population in thousands
londonpop <-
ts(c(1088,1258,1504,1778,2073,2491,2921,3336,3881,4266,
4563,4541,4498,4408), start=1801, end=1931, deltat=10)
points(log10(londonpop*1000), pch=16, cex=.5)
axis(2, at=log10(ytiks), labels=paste(ytiks), las=2)
## Panel B
plot(window(measles, start=1840, end=1882),
ylim=c(0, 4600), yaxt="n")
axis(2, at=(0:4)*1000, labels=paste(0:4), las=2)
```

Graph

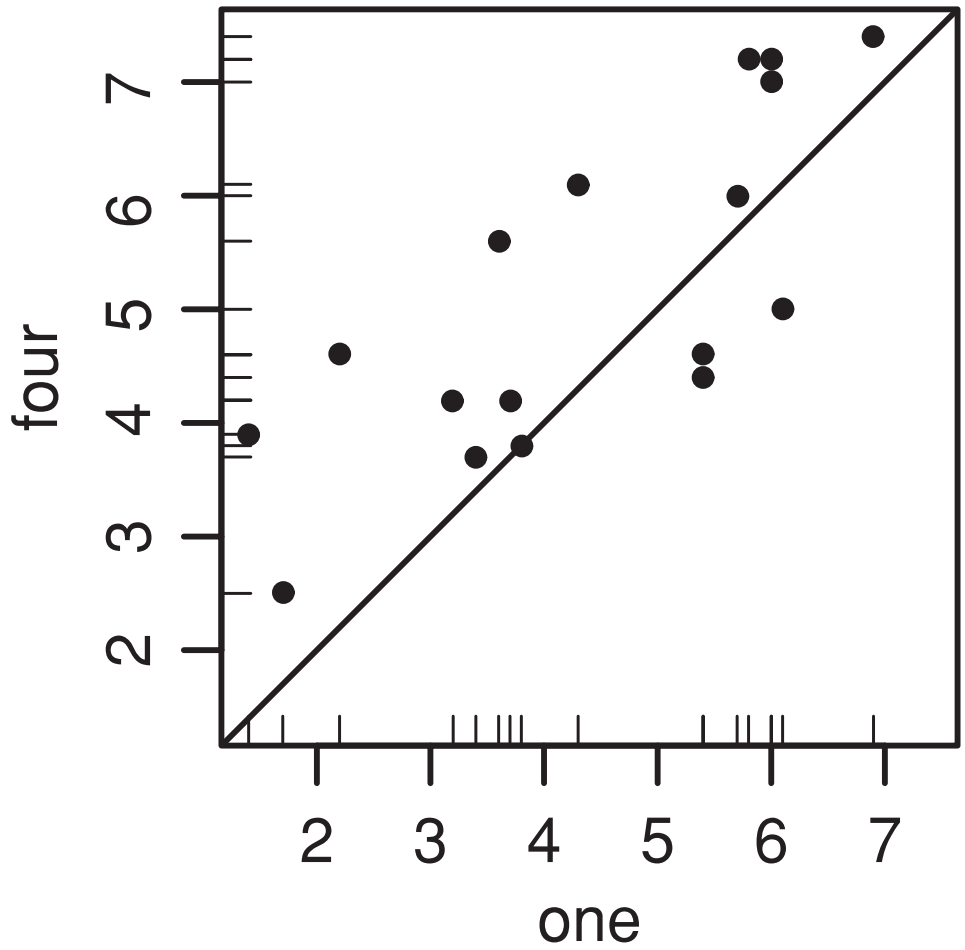

A (1629–1939)
Deaths; Population (log scale)
5e+06
1e+06
1000
100
10
1
1650
1700
1750
1800
1850
1900
1950
B (1841–1881)
Deaths; Population in 1000s
4000
3000
2000
1000
1840
1850
1860
1870
1880

Patterns in bivariate data

- Data from a tasting session where each of 17 panelists assessed the sweetness of each of two milk samples, one with four units of additive, and the other with one unit of additive.
- The line $y = x$ has been added. The function *rug*() adds a "rug".

```
## Plot four vs one: data frame milk (DAAG)
xyrange <- range(milk)
plot(four ~ one, data=milk, xlim=xyrange, ylim=xyrange,
pch = 16, pty="s") # pty="s": square plotting region
rug(milk$one)      # x-axis rug (default is side=1)
rug(milk$four, side = 2) # y-axis rug
abline(0, 1)
```

Graph

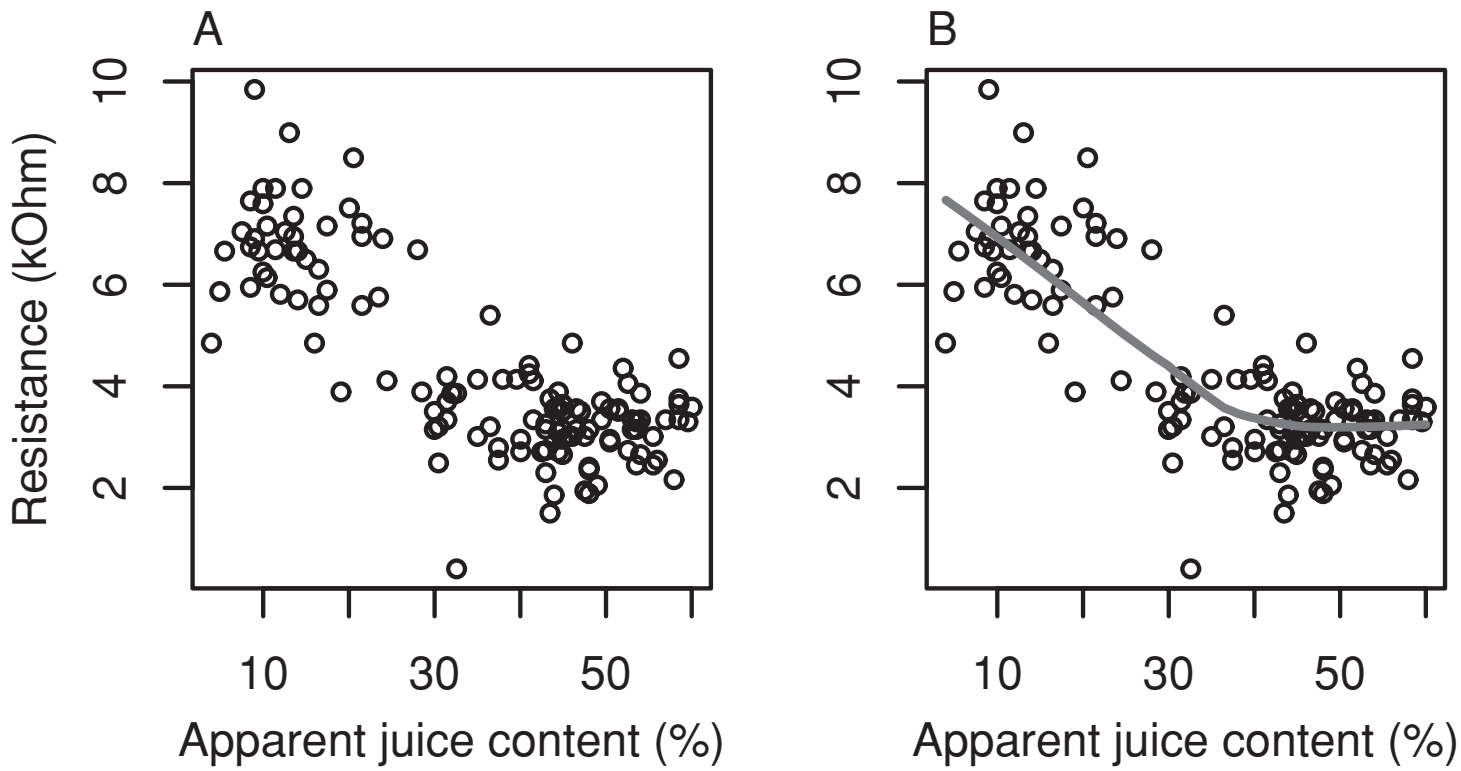


The fitting of a smooth trend curve

- Data from a study that measured both electrical resistance and apparent juice content for a number of slabs of kiwifruit.
- The curve in panel B, obtained using the *lowess* method, estimates the relationship between electrical resistance and apparent juice content.

```
## Plot ohms vs juice: data frame fruitohms (DAAG)
plot(ohms ~ juice, xlab="Apparent juice content (%)",
ylab="Resistance (ohms)", data=fruitohms)
## Add a smooth curve, as in Panel B
with(fruitohms, lines(lowess(juice, ohms), lwd=2))
# With lwd=2, the curve is twice the default thickness
```

Graph

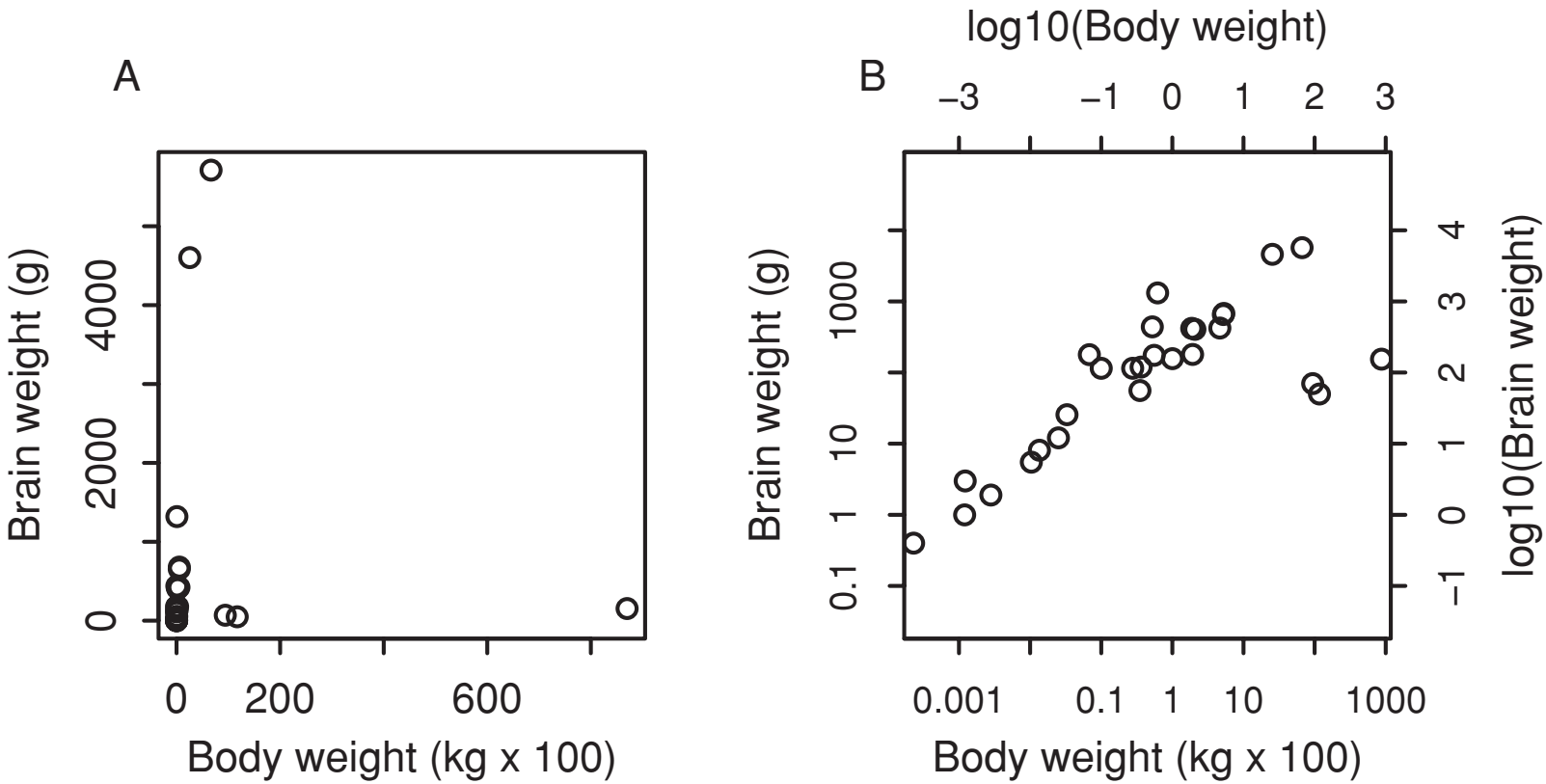

A
B
Resistance (kOhm)
2
4
6
8
10
10
30
50
Apparent juice content (%)
Apparent juice content (%)

Transformation of data

A and B plot brain weight (g) against body weight (kg), for a number of different animals:

```
## The following omits the labeling information
oldpar <- par(mfrow = c(1,2), pty="s")
## Plot brain vs body: data frame Animals (MASS package)
library(MASS)
plot(brain ~ body, data=Animals) # Panel A
plot(log(brain) ~ log(body), data=Animals) # Panel B
par(oldpar)
```

Graph

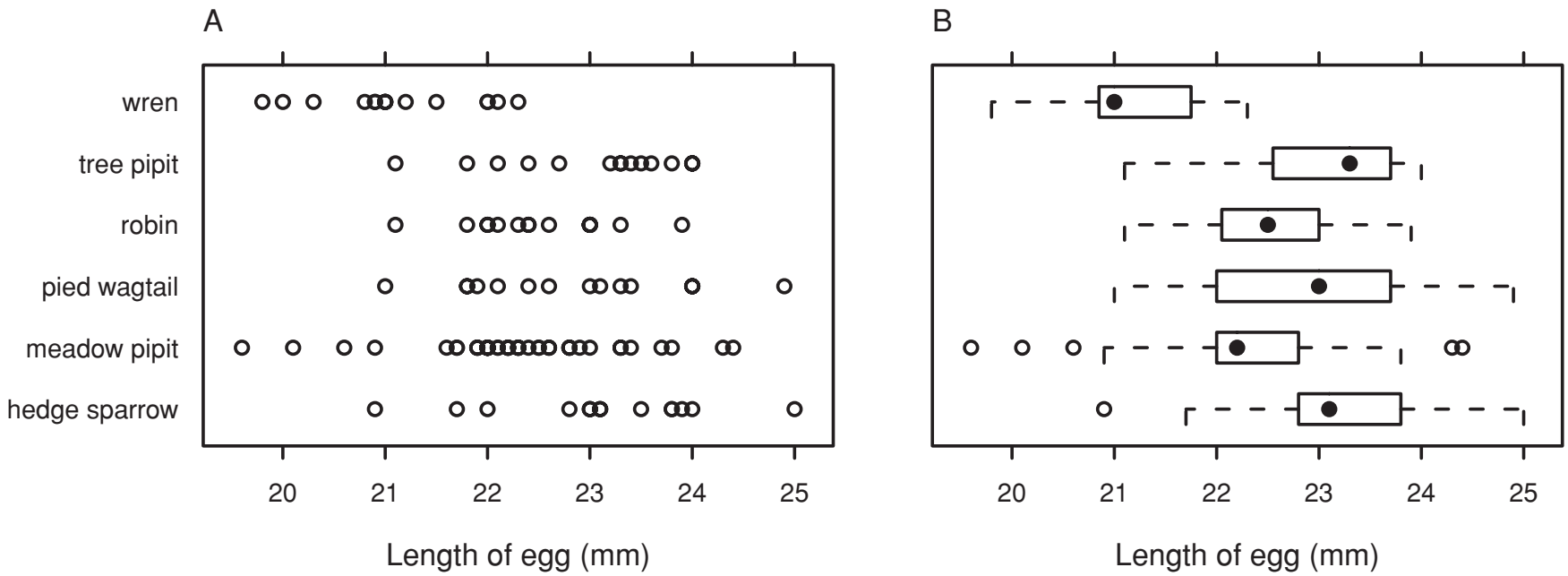

A
Brain weight (g)
0
2000
4000
0
200
600
Body weight (kg x 100)
B
log10(Body weight)
−3
−1
0
1
2
3
Brain weight (g)
0.1
1
10
1000
log10(Brain weight)
−1
0
1
2
3
4
0.001
0.1
1
10
1000
Body weight (kg x 100)

Patterns in grouped data

- Cuckoos lay eggs in the nests of other birds. The eggs are then unwittingly adopted and hatched by the host birds.
- The egg lengths are grouped by the species of the host bird, using both a *stripplot* display (panel A) and *boxplot* summaries (panel B).

```
 ## Compare stripplot() with bwplot(), both from lattice
package
 stripplot(species ~ length, xlab="Length of egg (mm)",
data=cuckoos)
 bwplot(species ~ length, xlab="Length of egg (mm)",
data=cuckoos,
 scales=list(y=list(alternating=0)))
 # alternating=0; omit y-axis labels
```

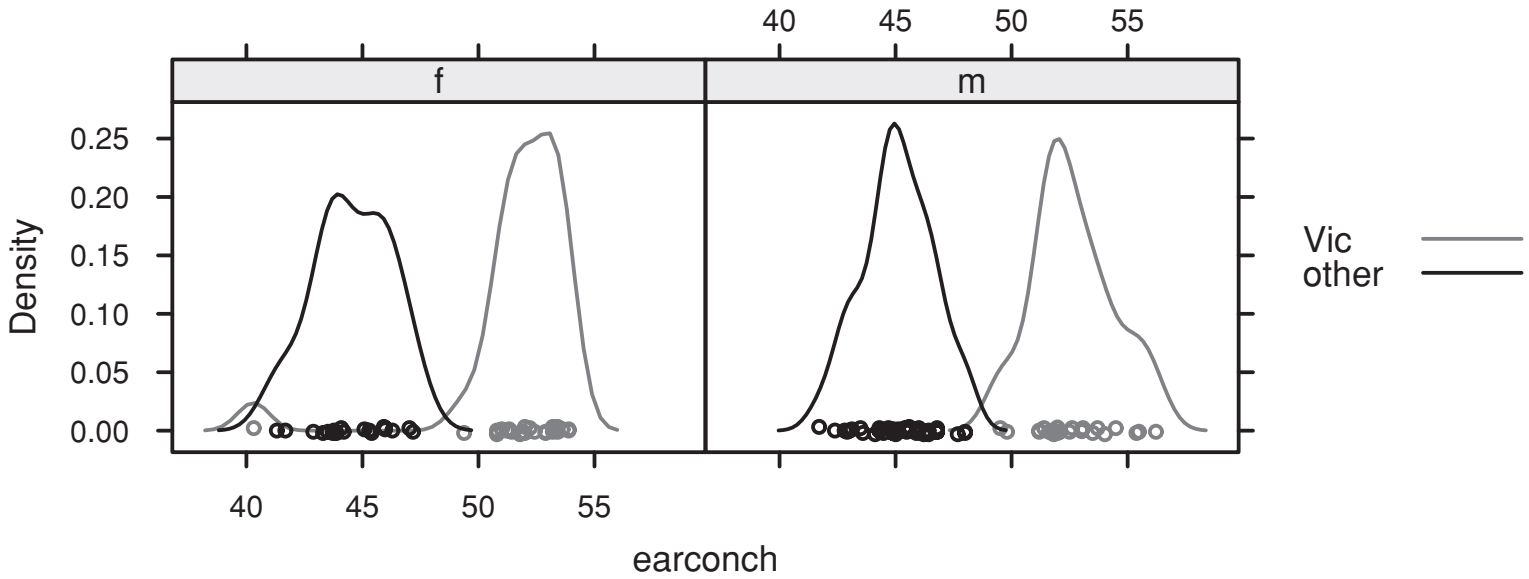
Graph


A
wren
tree pipit
robin
pied wagtail
meadow pipit
hedge sparrow
20
21
22
23
24
25
Length of egg (mm)
B
20
21
22
23
24
25
Length of egg (mm)

Comparing densities between groups

Lattice-style density plots can be useful for getting an indication of how distributions may differ across different groups of data.

```
## Density plot for earconch: data possum (DAAG package)
library(lattice)
densityplot(~earconch | sex, groups=Pop, data=possum,
auto.key=list(space="right"))
```



What to look for in plots

- Outliers
  - Outliers are points that are isolated from the main body of the data.
  - Boxplots, and the normal probability plot are useful for highlighting outliers in one dimension.
  - Scatterplots may highlight outliers in two dimensions.
  - Some outliers will be apparent only in three or more dimensions.
- Asymmetry of the distribution
  - Most asymmetric distributions can be characterized as either positively skewed or negatively skewed.
  - Severe skewness is typically a more serious problem for the validity of results than other types of non-normality.
  - Provided that all values are greater than zero, a logarithmic transformation typically makes such a distribution more symmetric.
- Changes in variability
  - Boxplots and histograms readily convey an impression of the extent of variability or scatter in the data.
  - When variability increases as data values increase, the logarithmic transformation will often help.

What to look for in plots

- Clustering
  - Clusters in scatterplots may suggest structure in the data that may or may not have been expected.
  - When we proceed to a formal analysis, this structure must be taken into account.
- Non-linearity
  - We should not fit a linear model to data where relationships are demonstrably non-linear.
  - Often it is possible to transform variables so that terms enter into the model in a manner that is closer to linear.
  - If there is a theory that suggests the form of model, then this is a good starting point.

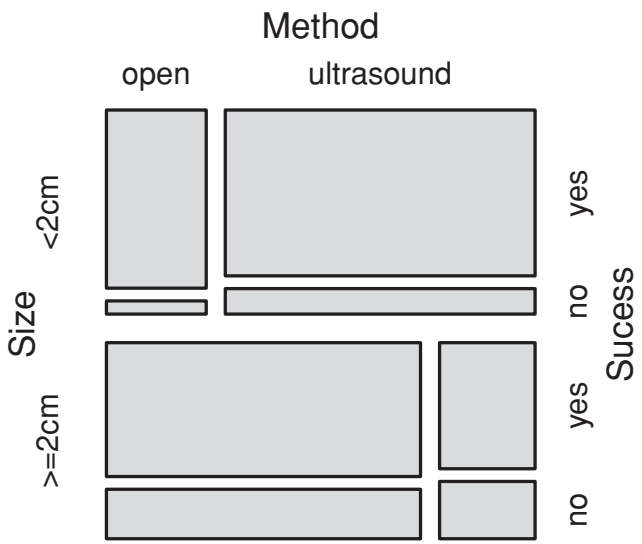
table() function

Data in the data frame nswpsid1 (DAAG) compare two groups of individuals with a history of unemployment problems ne an "untreated" control group and the other a "treatment" group whose members were exposed to a labor training program.

```
> ## Table of counts example: data frame nswpsid1 (DAAG)
> tab <- with(nswpsid1, table(trt, nodeg, useNA="ifany"))
> dimnames(tab) <- list(trt=c("none", "training"),
+                       educ = c("completed", "dropout"))
> tab
                    educ
                completed  dropout
trt  none            1730      760
     training          80      217
```

Three-way tables

```
stones <- array(c(81,6,234,36,192,71,55,25), dim=c(2,2,2),
           dimnames=list(Success=c("yes","no"),
Method=c("open","ultrasound"), Size=c("<2cm", ">=2cm")))
# NB: The margins are 1:Success, 2:Method, 3:Size
library(vcd)
mosaic(stones, sort=3:1) # c.f. mosaicplot() in base
```



		Success		
		yes	no	%Yes
Method	Size			
open	<2cm	81	6	93.1
	>=2cm	192	71	73.0
ultrasound	<2cm	234	36	86.7
	>=2cm	55	25	68.8
Add over Size				
open		273	77	78.0
ultrasound		289	61	82.6

Tabulation that accounts for frequencies or weights

Data description:

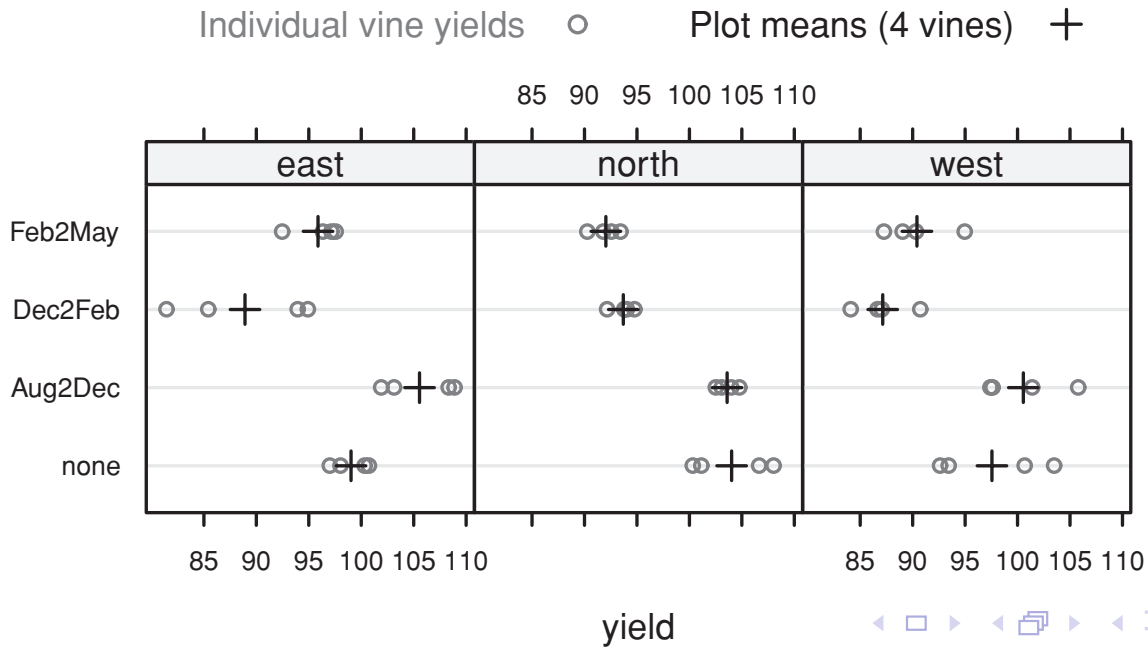
- The data in nassCDS (DAAG) are police-reported crashes in which there is a harmful event, and from which at least one vehicle is towed.
- Factors whose effect warrant investigation include, as a minimum: A : *airbag* (was an airbag fitted?), S : *seatbelt* (was a seatbelt used?), and *dvcat* (F: a force of impact measure). The letters A, S, and F will be used as abbreviations when tables are generated.
- The column *weight* (national inflation factor) holds the inverses of the sampling fraction estimates. The weight is designed to be the amount by which the contribution for the relevant row should be multiplied when tables of numbers of deaths and numbers of accidents are created. The following uses *xtabs*() to estimate numbers of front-seat passengers alive and dead, classified by airbag use:

Code

```
> library(DAAG)
> ## The parentheses generate an implicit print(abtab)
> (Atab <- xtabs(weight ~ airbag + dead, data=nassCDS))
airbag           dead
none    5445245.90  39676.02
airbag  6622690.98  25919.11
> roundpc2 <- function(x) round(100*x[2]/sum(x), 2)
> addmargins(Atab, margin=2, FUN=c("%Dead"=roundpc2))
airbag           dead
none    5445245.90  39676.02  0.72
airbag  6622690.98  25919.11  0.39
```

Summary as a prelude to analysis

- The data frame *kiwishade* has yield measurements from 48 vines. Plots, made up of four vines each, were the experimental units.
- The 12 plots were divided into three blocks of four plots each. One block was north-facing, a second west-facing, and a third east-facing.
- Shading treatments were applied to whole plot. The shading treatments were applied either from August to December, December to February, February to May, or not at all.



Code

```
> ## mean yield by block by shade: data frame kiwishade
> kiwimeans <- with(kiwishade,
+ aggregate(yield, by=list(block, shade), mean))
> names(kiwimeans) <- c("block","shade","meanyield")
## Individual vine means, by block and treatment
library(lattice)
## Panel function calls panel.dotplot(), then panel.average()
dotplot(shade ~ yield | block, data=kiwishade, aspect=1,
panel=function(x,y,...){panel.dotplot(x,y,pch=1,col="gray40")
panel.average(x, y, type="p", col="black", pch=3, cex=1.25)},
key=list(space="top", columns=2, col=c("gray40", "black"),
text=list(c("Individual vine yields", "Plot means (4 vines)")),
points=list(pch=c(1,3), cex=c(1,1.25))), layout=c(3,1))
# Note that parameter settings were given both in the calls
# to the panel functions and in the list supplied to key.
```

Standard deviation and inter-quartile range

- In R, use the function *sd*() to calculate the standard deviation, or *var*() to calculate the variance.
- The inter-quartile range H is the difference between the first and third quartiles.
- For data that are approximately normally distributed, note the approximate relationship

$$s \approx 0.75H.$$

```
## SD of length, by species: data frame cuckoos
sapply(split(cuckoos$length, cuckoos$species), sd)
```

Hedge sparrow	Meadow pipit	Pied wagtail	Robin	Tree pipit	Wren
1.049	0.920	1.072	0.682	0.880	0.754

Correlation

- The usual Pearson or product-moment correlation is a summary measure of linear relationship.
- If the relationship is monotonic, but is not linear and/or has asymmetric marginal distributions, it may be appropriate to use a Spearman rank correlation.

```
> ## Correlation between body and brain: Animals (MASS)
> ## Pearson correlation
> with(Animals, cor(body, brain))
[1] -0.005341
> ## Pearson correlation, after log transformation
> with(log(Animals), cor(body, brain))
[1] 0.7795
> ## Spearman rank correlation
> with(Animals, cor(body, brain, method="spearman"))
[1] 0.7163
```