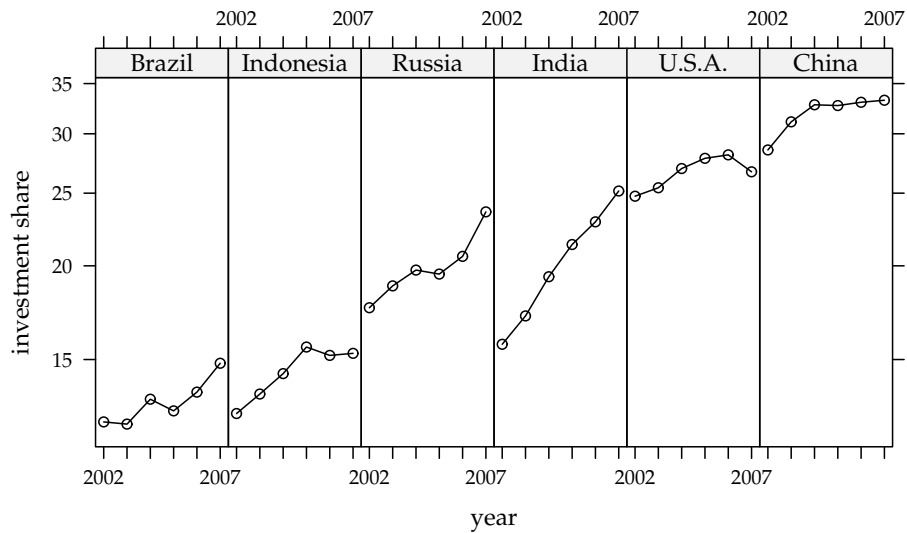


Using Graphs and Visualising Data



Oliver Kirchkamp

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 3 |
| 1.1 | Literature | 3 |
| 1.2 | Examples | 3 |
| 2 | Properties of good graphs | 6 |
| 2.1 | How to present | 7 |
| 2.1.1 | Axes | 7 |
| 2.1.2 | Points | 14 |
| 2.1.3 | Points and timeseries | 17 |
| 2.1.4 | Error Bars | 18 |
| 2.2 | Legends | 18 |
| 2.3 | Clutter | 19 |
| 2.4 | Unnecessary 3D | 21 |
| 2.5 | Aspect ratio | 21 |
| 2.6 | What to present | 25 |
| 2.6.1 | Structuring content | 25 |
| 2.6.2 | Don't discard parts of your data | 26 |
| 2.6.3 | Projecting data | 27 |
| 2.6.4 | Differences | 29 |
| 3 | Presenting nominal data | 30 |
| 3.1 | Nominal univariate | 30 |
| 3.2 | Nominal bivariate | 31 |
| 3.3 | Nominal multivariate | 33 |
| 4 | Presenting continuous data | 35 |
| 4.1 | Diagnostic plots for continuous variables | 35 |
| 4.2 | One continuous plus one nominal | 35 |
| 4.2.1 | Histograms | 36 |
| 4.2.2 | Densities | 37 |
| 4.2.3 | Conditional density plots | 38 |
| 4.2.4 | Boxplots | 39 |
| 4.2.5 | Barplot of means | 39 |
| 4.2.6 | Means and standard deviation | 40 |
| 4.2.7 | Empirical cumulative densities | 41 |
| 4.2.8 | QQ-Plots | 41 |
| 4.2.9 | Dot-Plots | 42 |

| | | |
|----------|--|-----------|
| 4.2.10 | Summary | 44 |
| 4.3 | Two continuous variables | 45 |
| 4.3.1 | Scatterplot | 45 |
| 4.3.2 | Scatterplot with confidence ellipses | 46 |
| 4.3.3 | Bagplot | 47 |
| 4.3.4 | Kernel densities | 48 |
| 4.3.5 | Scatterplot plus loess and regression line | 49 |
| 4.3.6 | Confidence bands for smooth lines | 49 |
| 4.3.7 | Which loess? | 51 |
| 4.4 | Paired data | 53 |
| 4.5 | Three-dimensional simplex | 54 |
| 4.6 | Stars | 55 |
| 5 | Using R | 56 |
| 5.1 | Installation of R | 56 |
| 5.2 | Types and assignments | 56 |
| 5.3 | Functions | 60 |
| 5.4 | Random numbers | 61 |
| 5.5 | Example Datasets | 62 |
| 5.6 | Graphs | 65 |
| 5.7 | Basic Graphs | 65 |
| 5.7.1 | Plotting functions | 67 |
| 5.7.2 | Empty plots | 67 |
| 5.7.3 | Line type | 68 |
| 5.7.4 | Points | 68 |
| 5.7.5 | Legends | 69 |
| 5.7.6 | Auxiliary lines | 69 |
| 5.7.7 | Axes | 70 |
| 5.8 | Fancy math | 71 |
| 5.8.1 | Several diagrams | 72 |
| 5.9 | Tables | 73 |
| 5.10 | Regressions | 74 |
| 5.11 | Starting and stopping R | 75 |
| 6 | Lattice | 75 |
| 6.1 | Multiway xyplots | 75 |
| 6.2 | Syntax | 77 |
| 6.3 | Multiway continued | 79 |

| | | |
|--------|--|----|
| 6.4 | Densityplots | 82 |
| 6.5 | Histograms | 83 |
| 6.6 | Empirical cumulative densities | 83 |
| 6.7 | QQ-plots | 84 |
| 6.8 | Sample QQ-plots | 85 |
| 6.9 | Boxplots | 86 |
| 6.10 | Barcharts | 86 |
| 6.11 | Coplots | 87 |
| 6.12 | Parameters | 88 |
| 6.12.1 | Types | 88 |
| 6.12.2 | Axes | 89 |

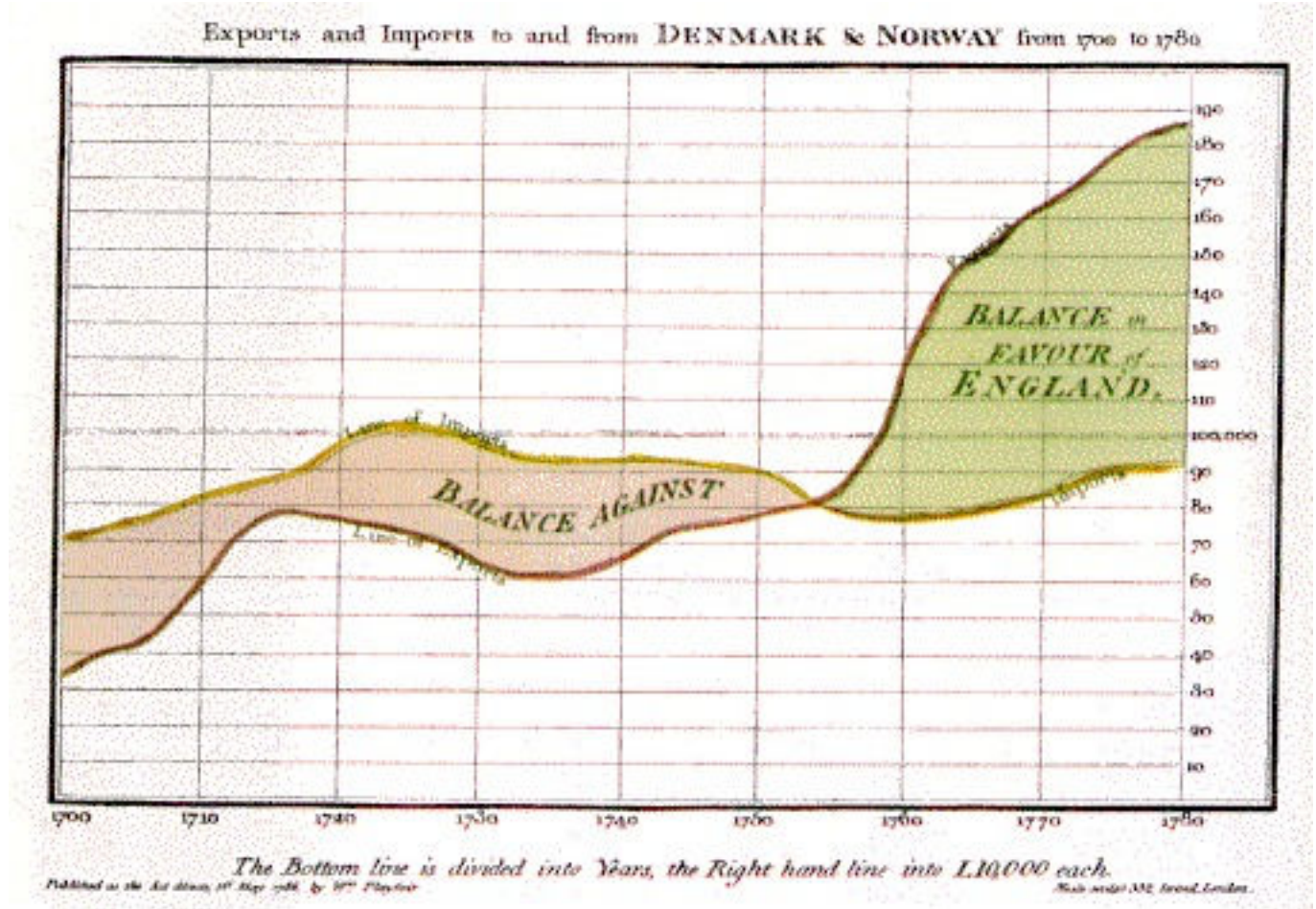
1 Introduction

1.1 Literature

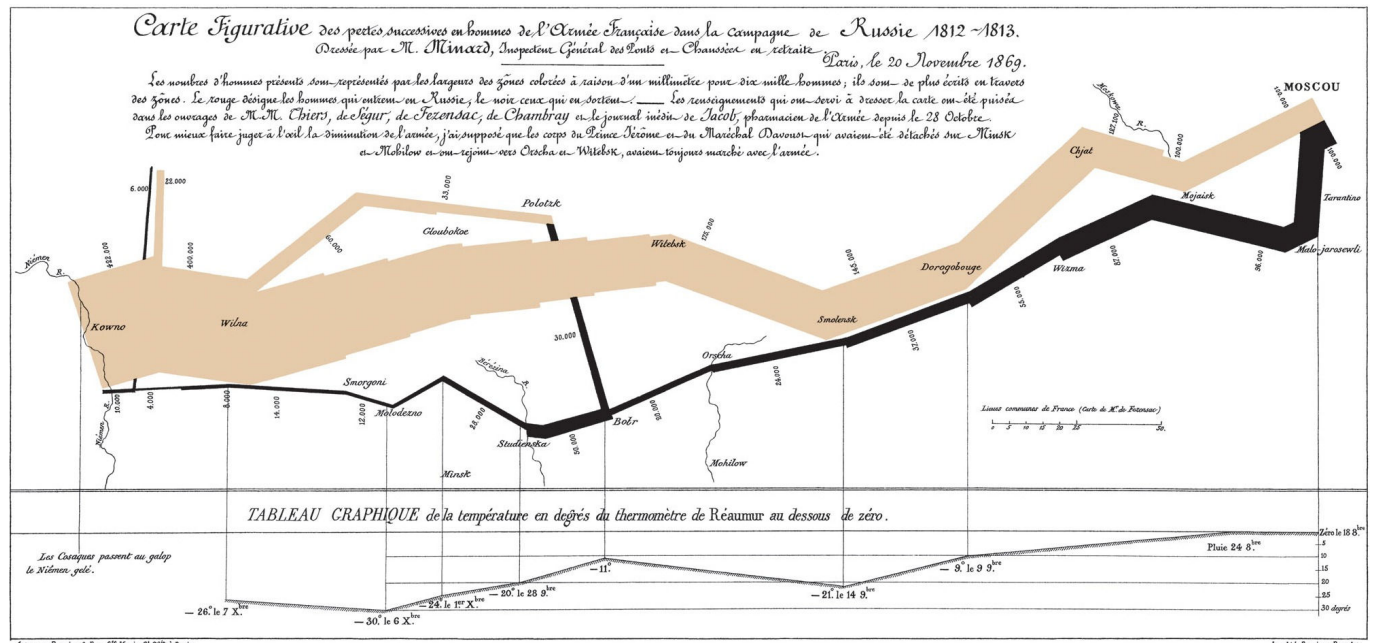
- The Elements of Graphing Data (Revised Edition). W. S. Cleveland (1994). Hobart Press, Summit, New Jersey, U.S.A.
- Lattice—Multivariate Data Visualization with R. Deepayan Sarkar (2008). Springer, New-York.

1.2 Examples

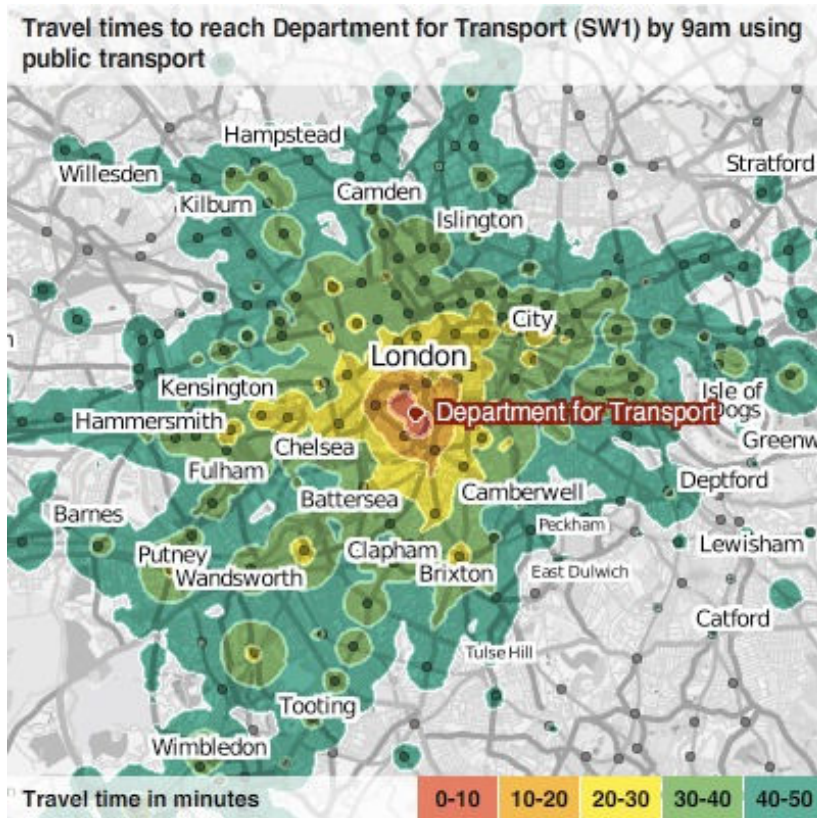
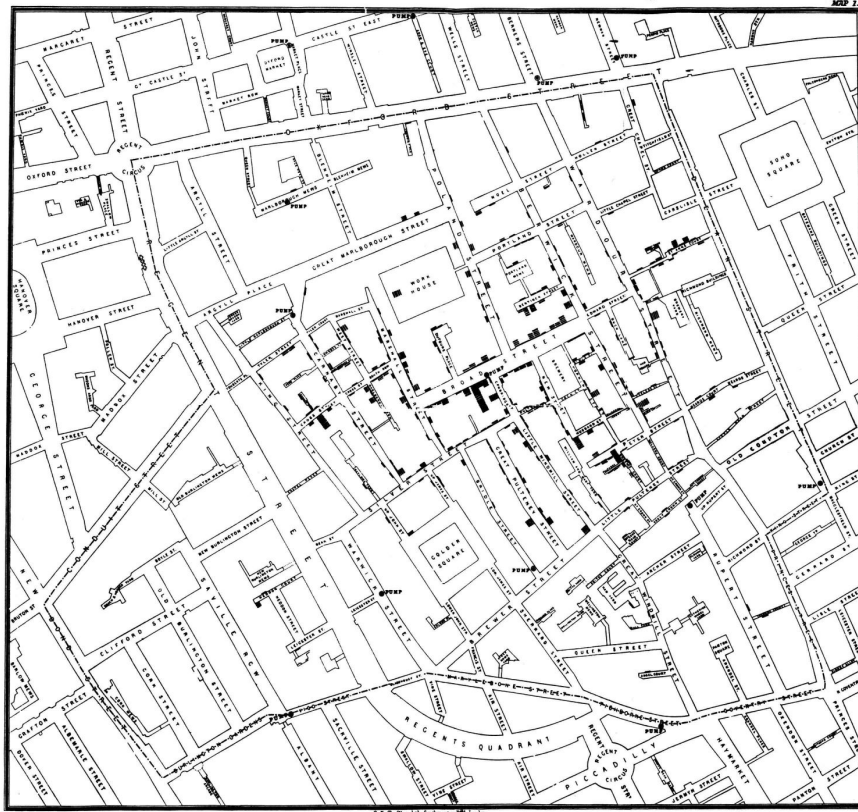
William Playfair's trade-balance time-series chart, 1786:



Charles Minard's 1869 chart of Napoleon's 1812 Russian campaign:



John Snow, 1854 Broad Street cholera outbreak:



Aims:

- Note: good graphs are self-explanatory!
→ The key to understand a graphs should not be hidden somewhere in the text!
- Often, the optimal presentation of data is not “standard”.
- There are no “recipes” how to present data.
- We have to use our own imagination.
- Still, some examples might help.

What can be achieved with a good graph?

A graph can. . .

- . . . make the reader familiar with the structure of the data (create trust),
- . . . motivate a research question,
- . . . summarise conclusions of the paper.

A graph must be very good:

- Some readers look only or mainly at the figures and the graphs.
- Each graph should tell a story and should be self-explanatory.
- Among the many ways to present our data and our results, we have to chose the best way.

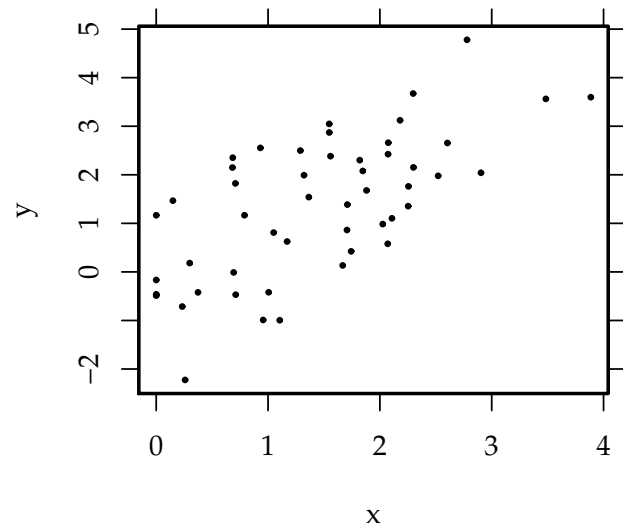
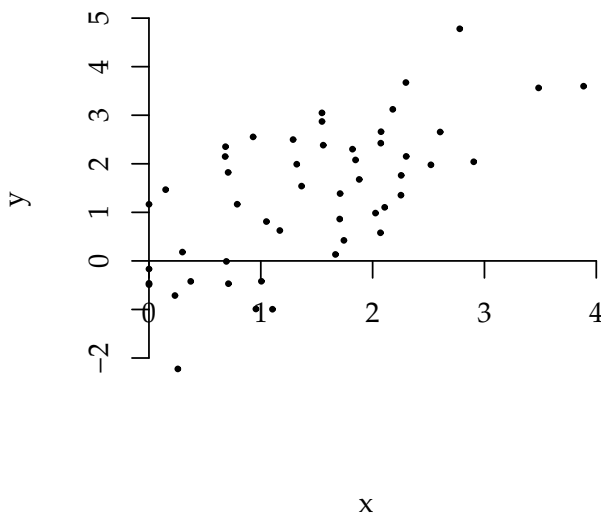
2 Properties of good graphs

- Essential items are shown clearly.
- Superfluous items are not shown.
- All elements of the graph are explained in the figure (not only in the text).

2.1 How to present

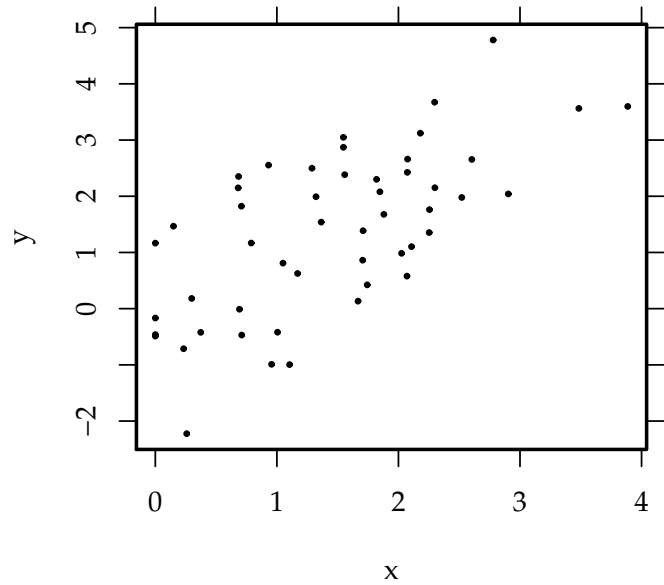
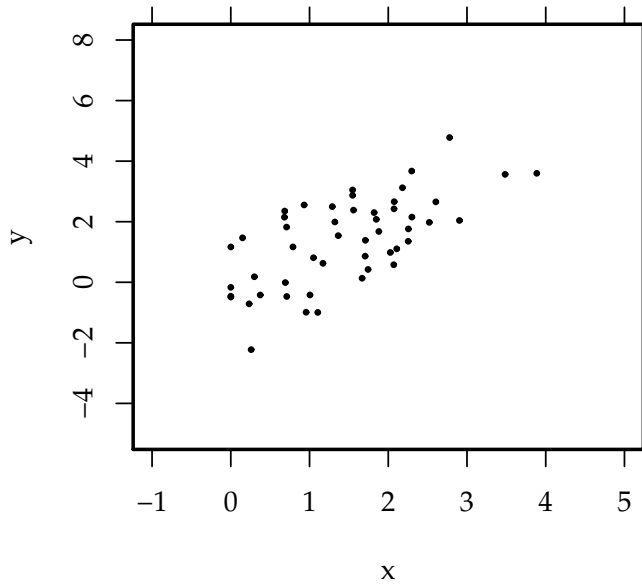
2.1.1 Axes

Frames: The following two graphs show the same data. However, in the graph on the left the axes and the data points are superimposed. In the graph on the right axes and data points are separate items. A frame around the plot region makes it even more clear where the reader should expect data.



- Labels and tick marks are separated from the data.
- Ticks on the opposite axis can help.

Ranges: In the following example both graphs show the same data. The only difference is the range that is covered by the axes.



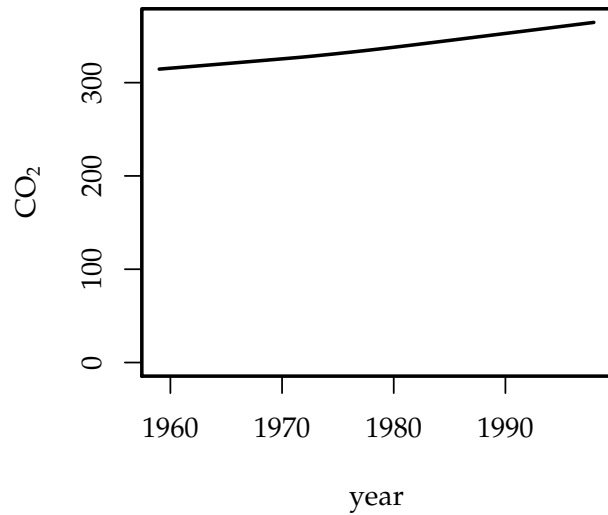
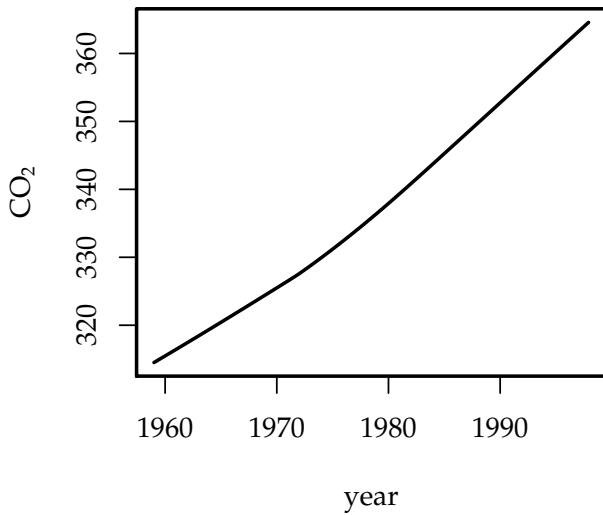
Ranges are chosen such that...

- ... all data is included,
- ... space is used in an efficient way.

Ranges and correlations We tend to perceive more correlation if the data occupies less space.

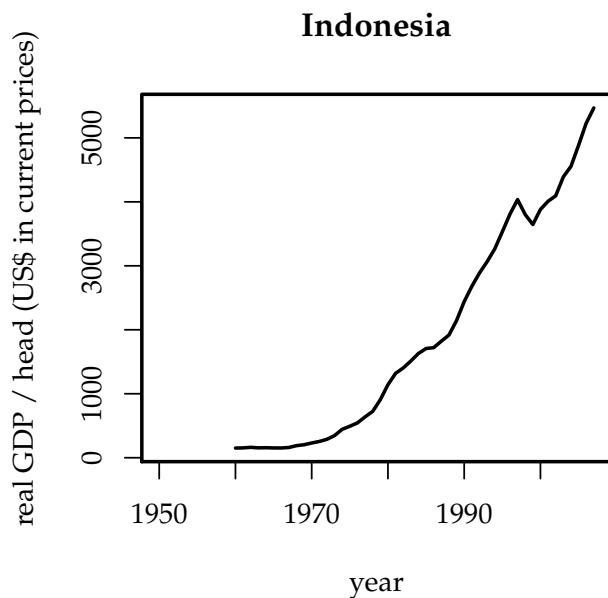
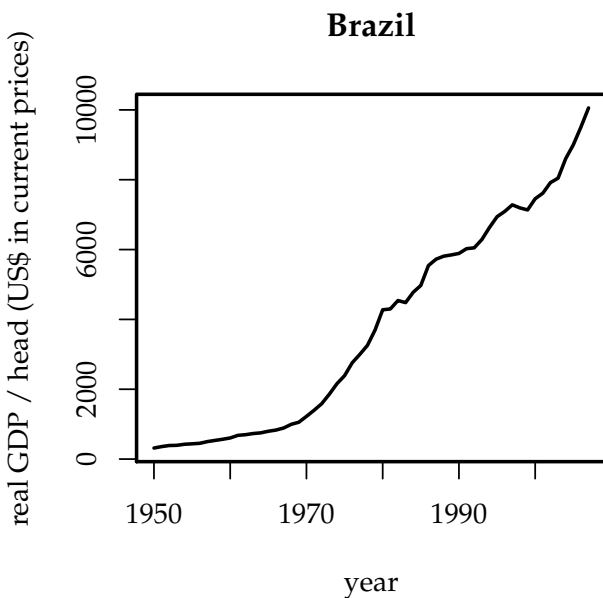
- The amount of white space around the data should be similar in all graphs.

Ranges that include zeroes:



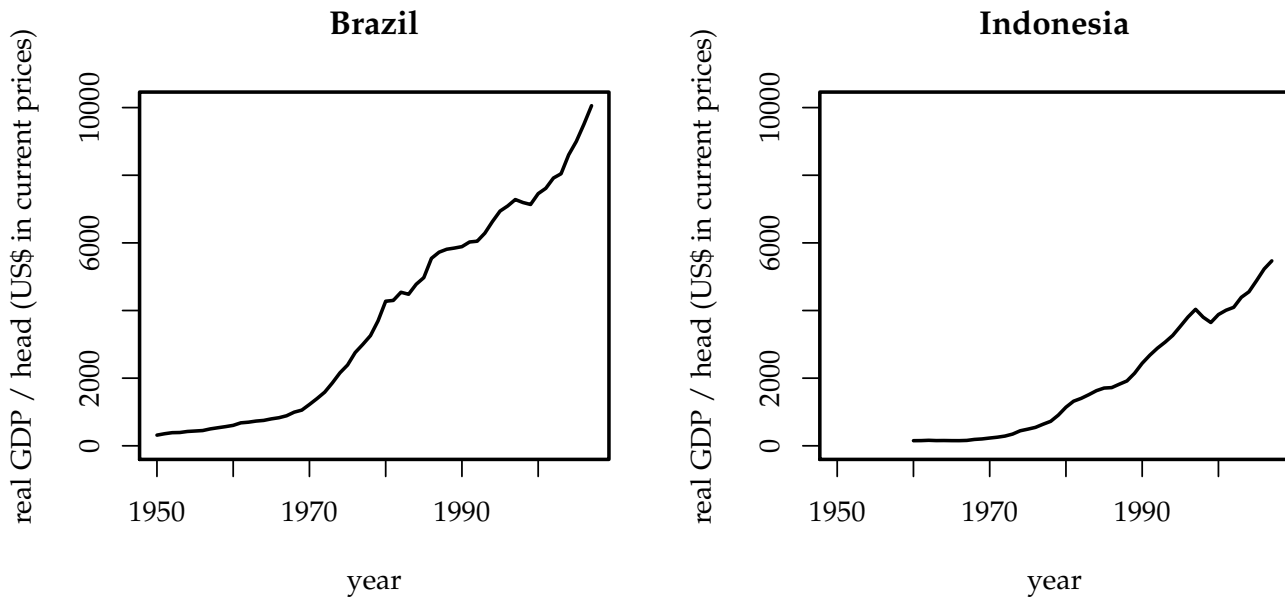
It is often helpful to include zero, but since this might waste space it is not an absolute necessity.

Comparable scales: In the following example we use different scales for the two diagrams. This makes them difficult to compare (although space is used efficiently).



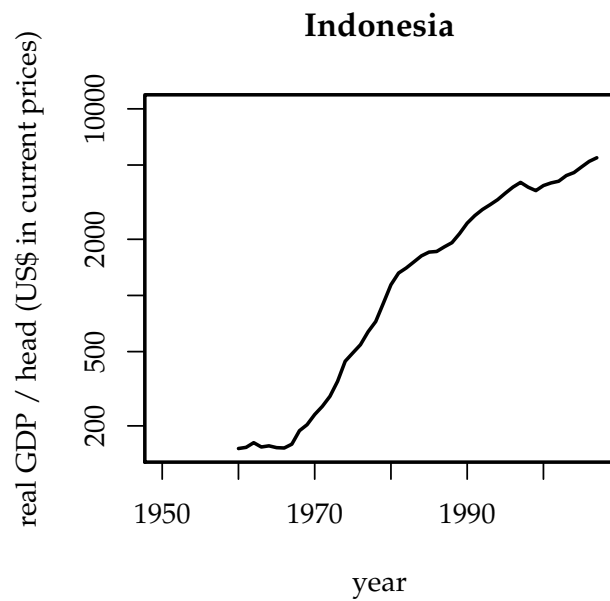
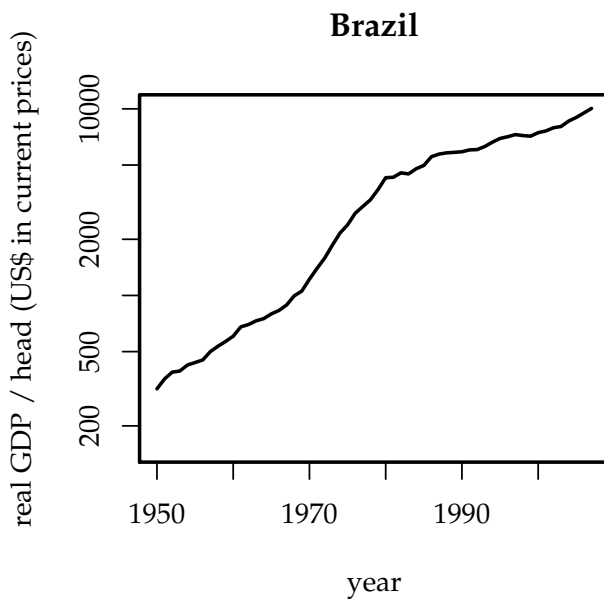
The figure shows real gross domestic product (GDP) per capita (US dollars in current prices). The data is taken from Penn World Table Version 6.3.

In the next figure we use the same scale for the two diagrams. Now we see immediately that GDP is larger in Brazil and smaller in Indonesia. Of course, presenting both lines in one diagram might be preferable, here.

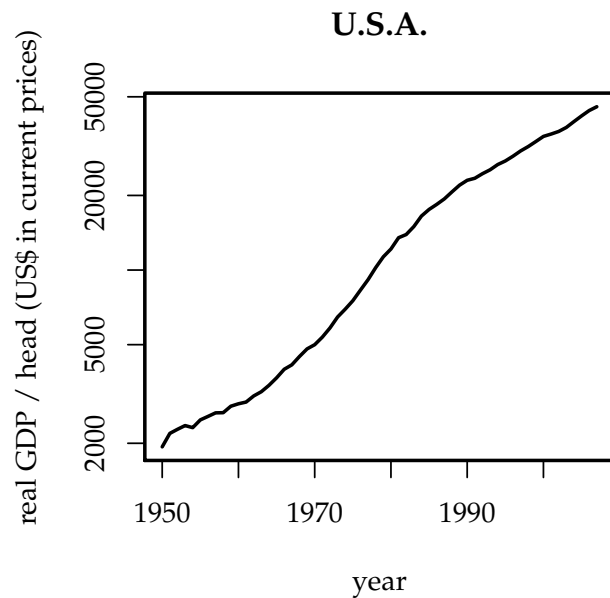
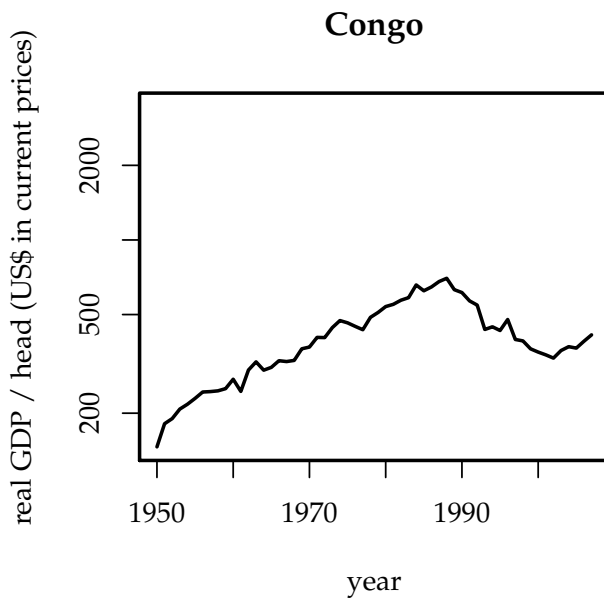


The figure shows real gross domestic product (GDP) per capita (US dollars in current prices). The data is taken from Penn World Table Version 6.3.

Comparability does not always require to use the same axes for several diagrams. Sometimes it might be better to use the same scale with a different origin.

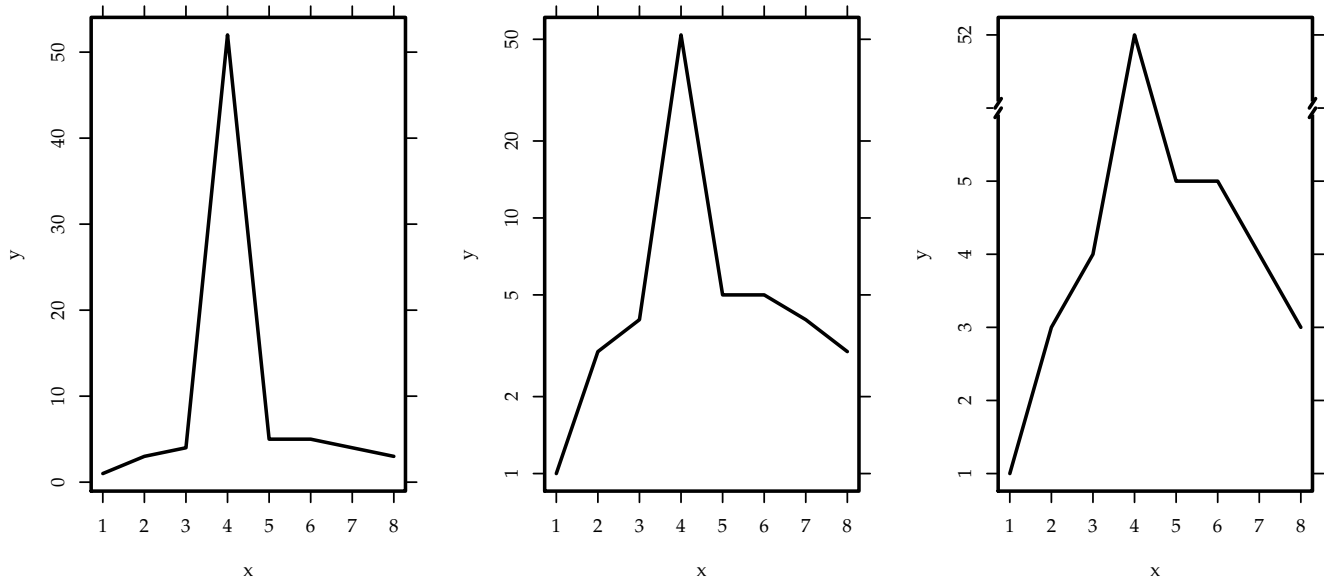


Using a logarithmic scale allows to compare relative growth.

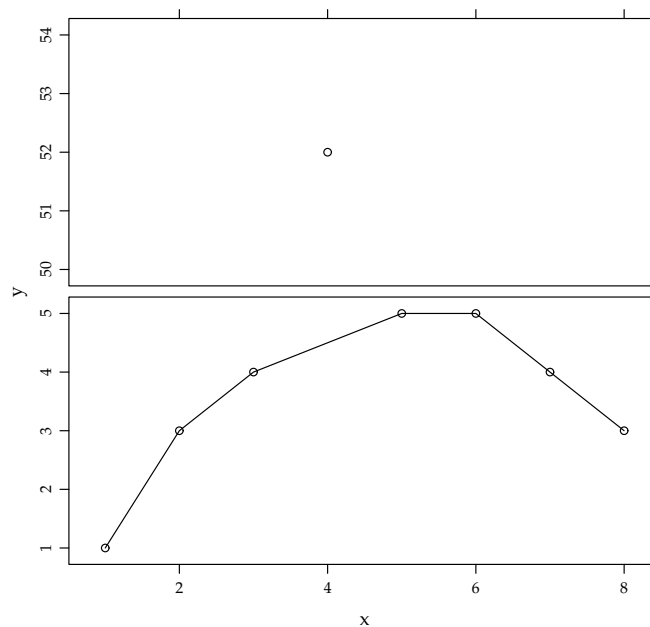


Comparability does not always require to use the same axes for several diagrams. Sometimes it might be better to use the same scale with a different origin.

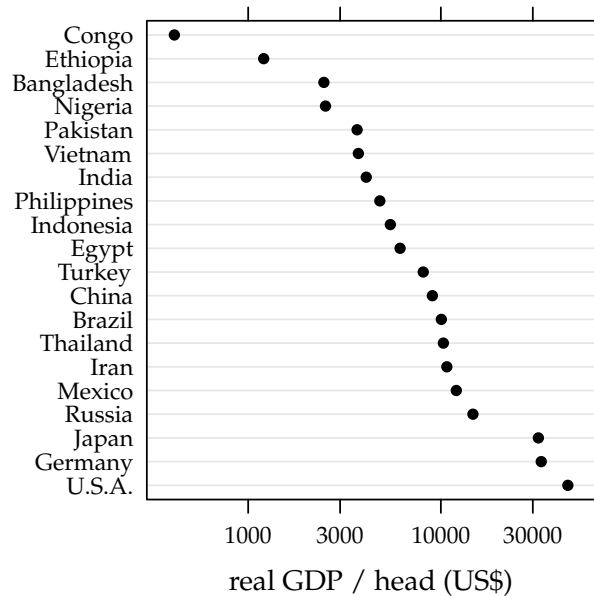
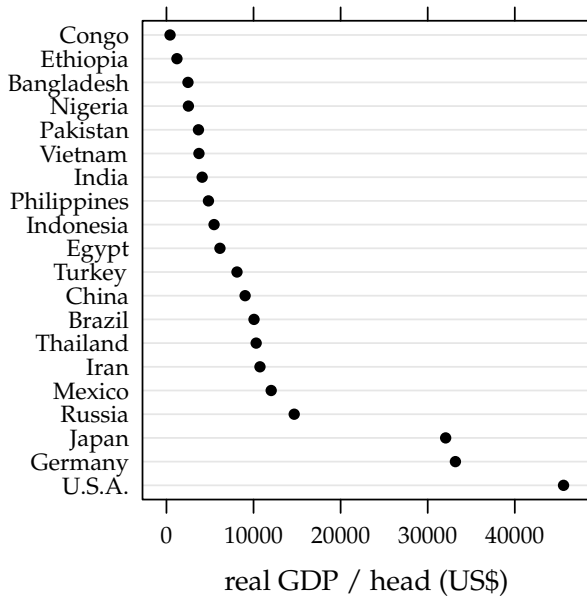
Breaks:



All three graphs try to show the same data. The graph on the left uses a linear scale. Here the outlier is clearly visible. The graph in the middle uses a logarithmic scale. This can be reasonable if ratios of the variable are interesting. The graph on the right tries to save space by “breaking” the axis. Your reader might not notice the gap. If gaps can not be avoided, dividing the graph into different panels might be preferable:

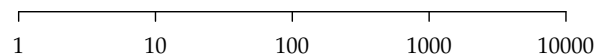


Logarithmic scales: The graph on the left shows GDP on a linear scale, the one on the right uses a logarithmic scale. On the left countries like Congo and Ethiopia seem to be quite similar. The graph on the right makes it easier to compare ratios. We see that, in relative terms, Germany is perhaps closer to the United States of America than Congo is to Ethiopia.

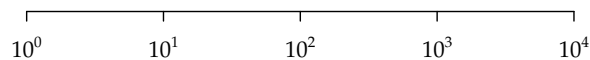


Which logarithm?

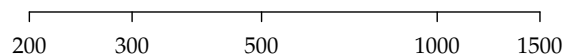
When the ratio of the largest to the smallest value is really large, then a logarithmic scale is obvious.



As an alternative we could also write the powers of 10:



Things become tricky with a less extreme scale:

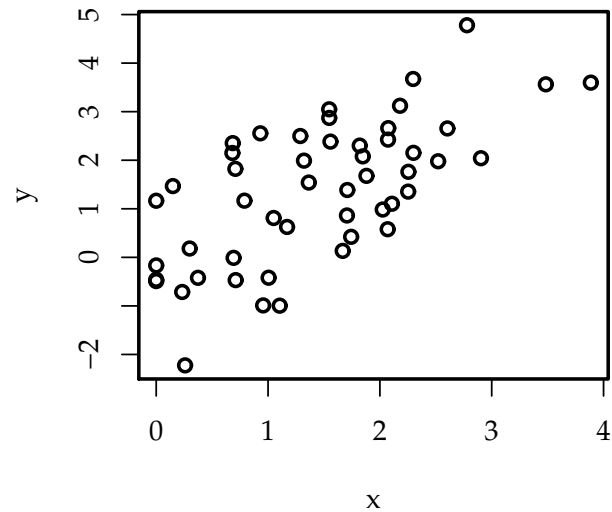
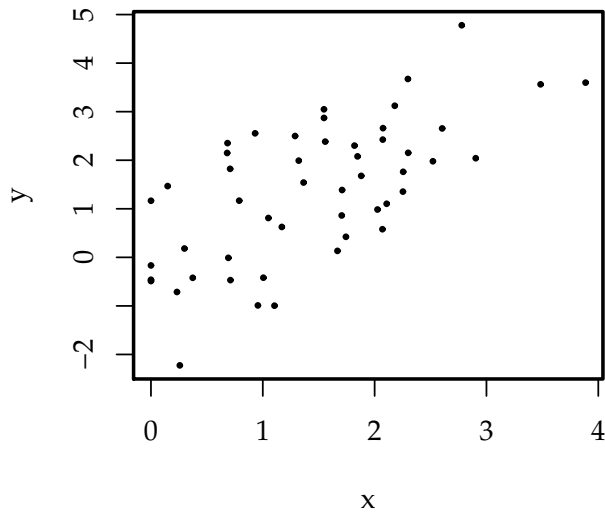


Here using a logarithm with base two can help:

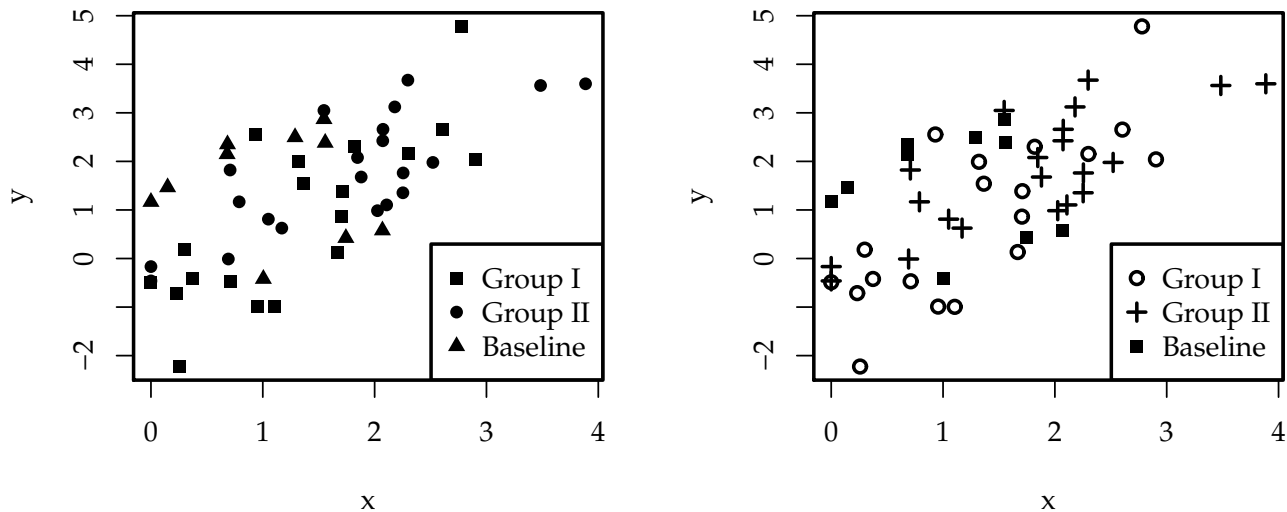


2.1.2 Points

In the following example the graph on the left uses fairly small points, the one on the right uses larger points to display the data.



Plotting symbols should be easy to distinguish. Compare the graph on the left with the graph on the right:



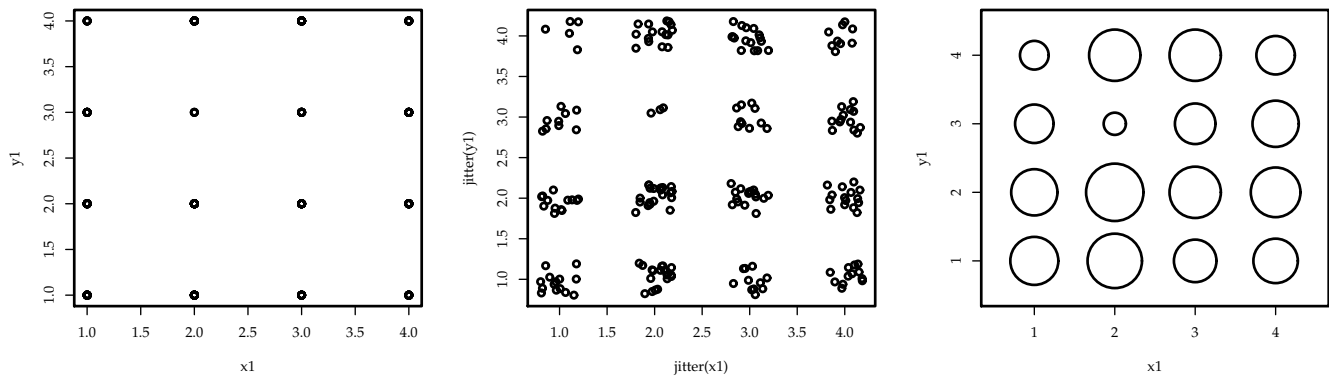
- When points do not tend to overlap we can use both heavy (●) and light (○) plotting symbols. Their contrasts helps us to distinguish different types of points.
- When points tend to overlap, heavy symbols are difficult to disentangle. In this situation we should only use light symbols.

Kröse's experiment Participants see patterns of symbols for 80 ms. They have to identify presence or absence of a given symbol.

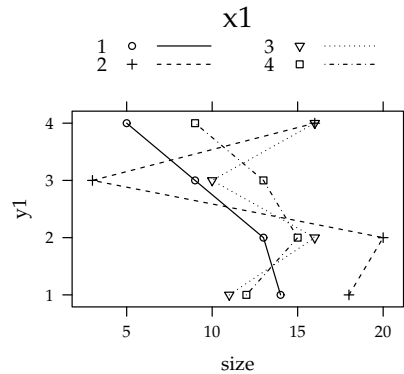
| symbols | % recognised |
|---------|--------------|
| +○ | 100.0 |
| +□ | 88.1 |
| L+ | 68.6 |
| Δ ↓ | 52.3 |
| +T | 37.6 |
| +X | 30.3 |
| TL | 30.6 |

(B. J. A. Kröse, Local structure analyzers as determinants of preattentive pattern discrimination. *Biological Cybernetics*, Volume 55, Number 5, 289-298.)

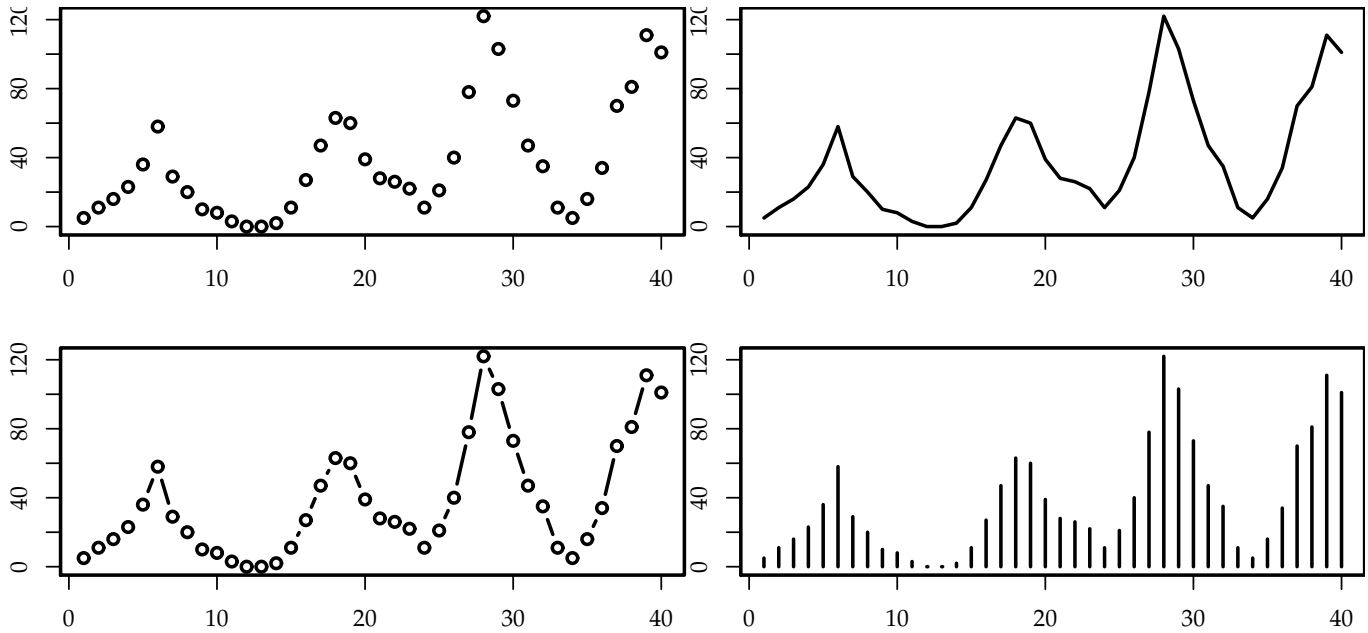
Nominal data and points Sometimes, in particular with nominal data, we want to show the same observation several times. In the diagram on the left multiple dots are simply printed on top of each other. One can not see how frequent an observation is. In the middle we add `jitter` to each observation, small noise, which allows us to distinguish the single observations. The left graph shows frequencies as size of the symbol.



If we have a small number of categories (at least in one dimension), a dotplot might be better:



2.1.3 Points and timeseries

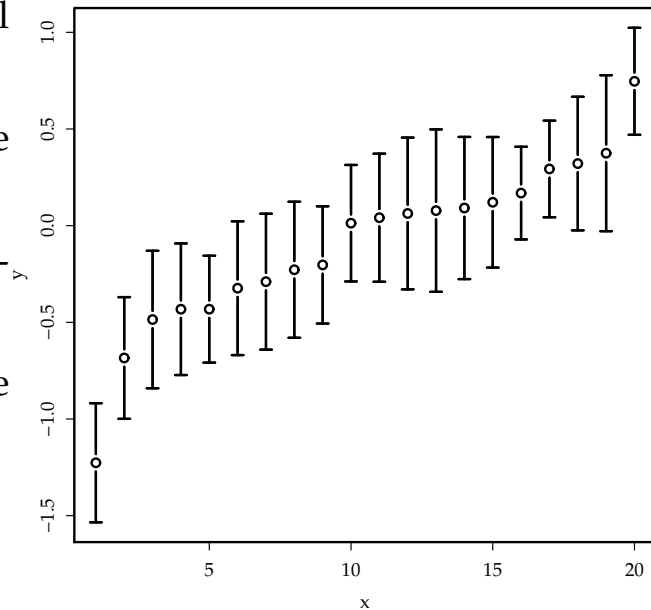


- The development over time is easier to see with lines.
- Lines alone make it impossible to find out when the measurements were taken.
- Points may reveal the long-term trend in the data but they make it difficult to grasp the short term behaviour.

2.1.4 Error Bars

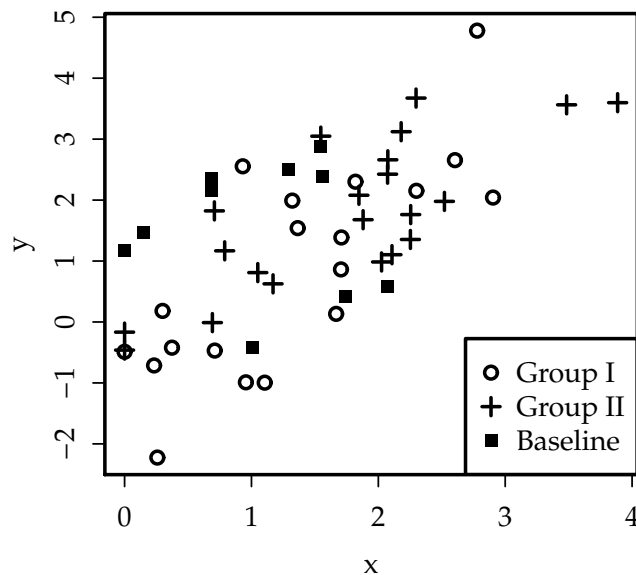
Error bars can refer to several quantities:

- Standard deviation of the sample
- Standard deviation of the estimated mean
- 95% confidence intervals of the estimated mean
- ⋮



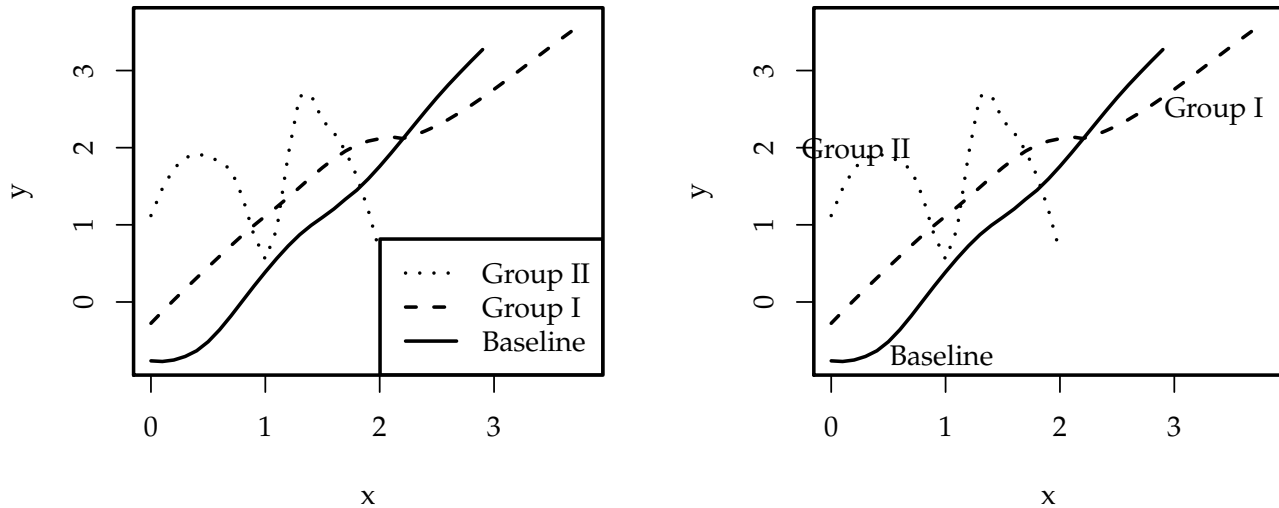
We have to explain clearly in the figure what quantity is shown. Boxplots are often more informative than error bars.

2.2 Legends



When you use more than one type of points, you need a legend. I like putting

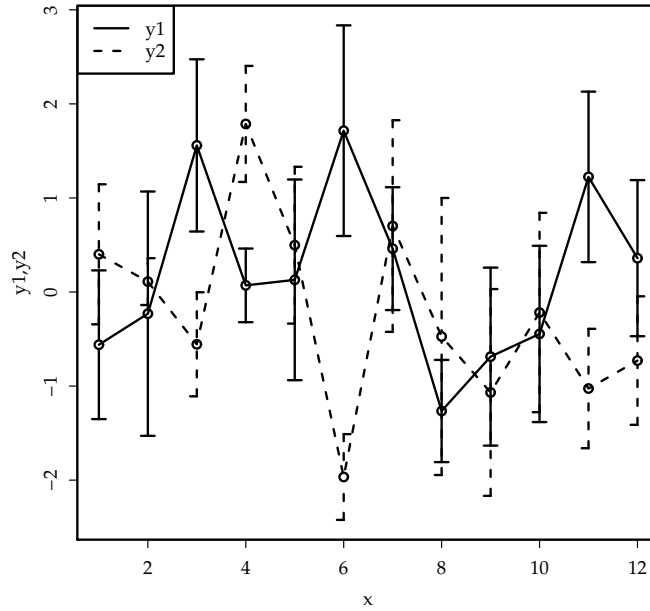
the legend inside the graph since this saves space. However, a legend outside the graph produces less clutter (see also 2.3).



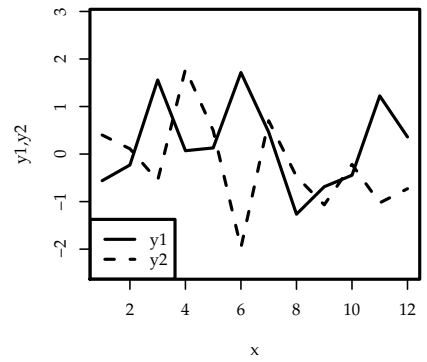
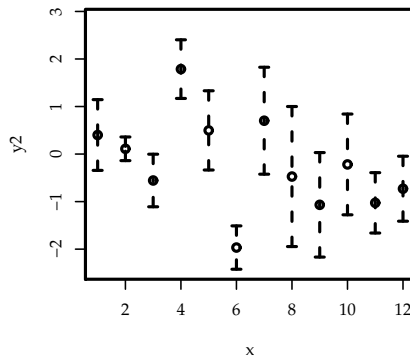
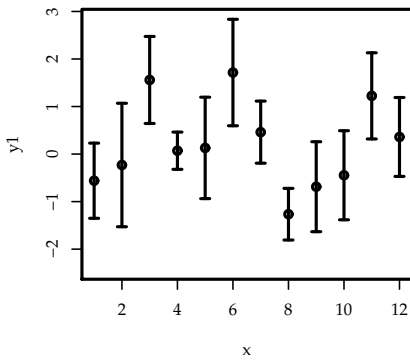
With lines, one may need a legend too. It helps, if the labels follow the same order as the lines (see graph on the left). Often the graph is easier to understand if you label the curves directly (graph on the right).

2.3 Clutter

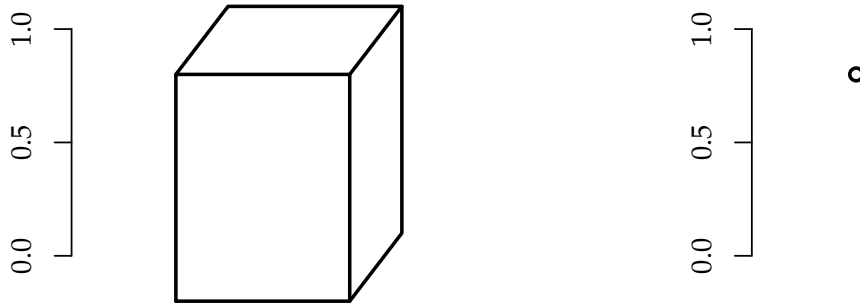
The following graph presents too many things in one graph.



Perhaps splitting the information into several graphs can help:



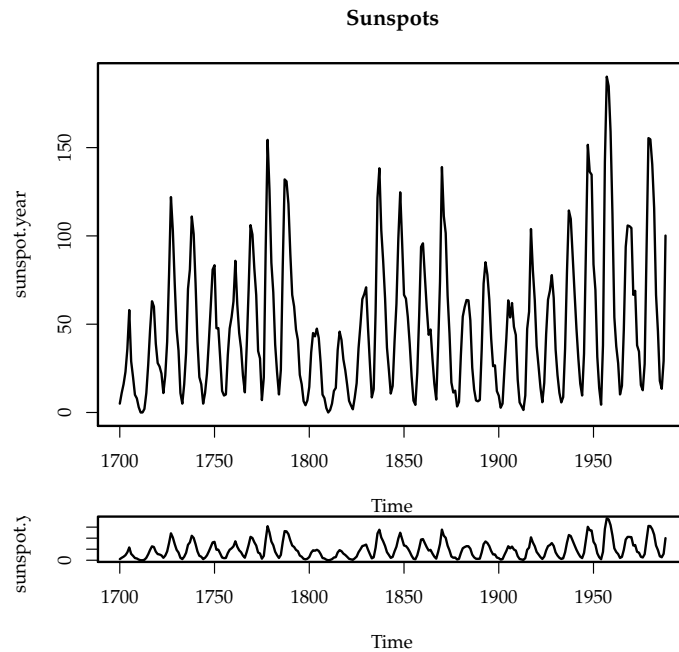
2.4 Unnecessary 3D



Unnecessary 3D effects distract from the content of your graph. Is the bar in the graph on the left larger or smaller than 1.0? Of course, one can work it out, but a simple bar without 3D (on the right) is much easier to understand.

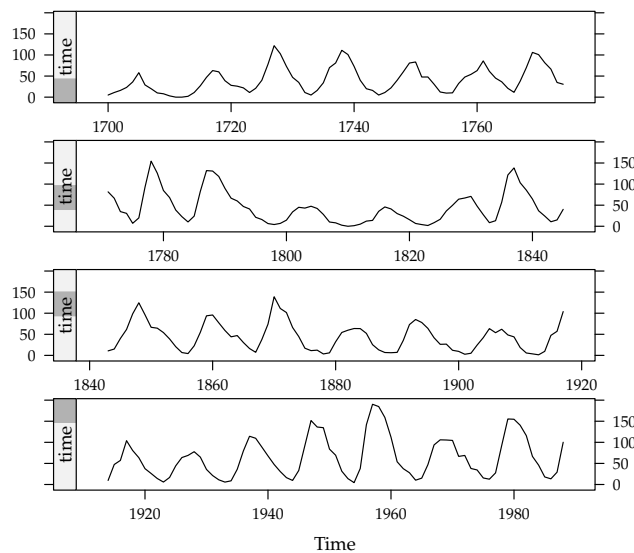
2.5 Aspect ratio

Less can be more — 45°

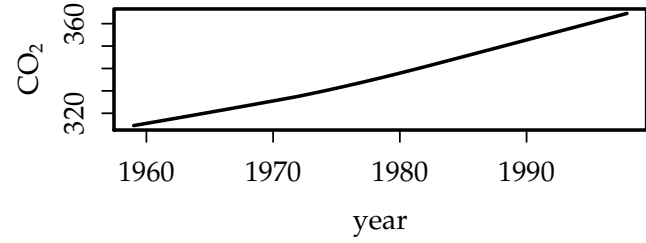
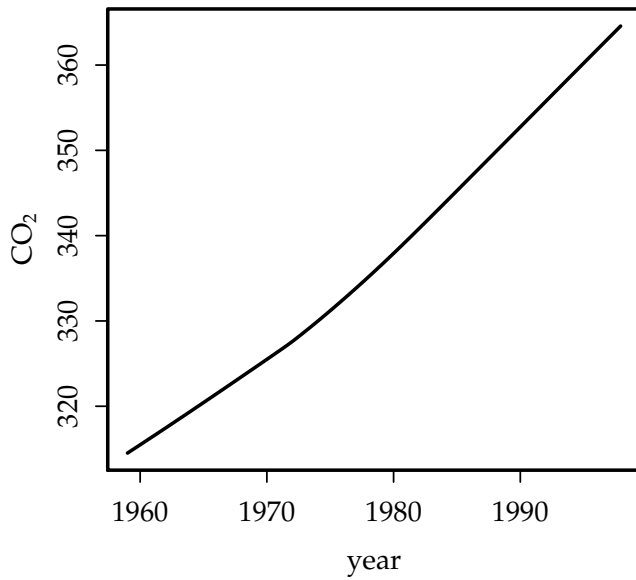


The lower graph might be more informative than the upper one. We can see that activity increases more quickly than it decreases. This is less obvious in the upper graph.

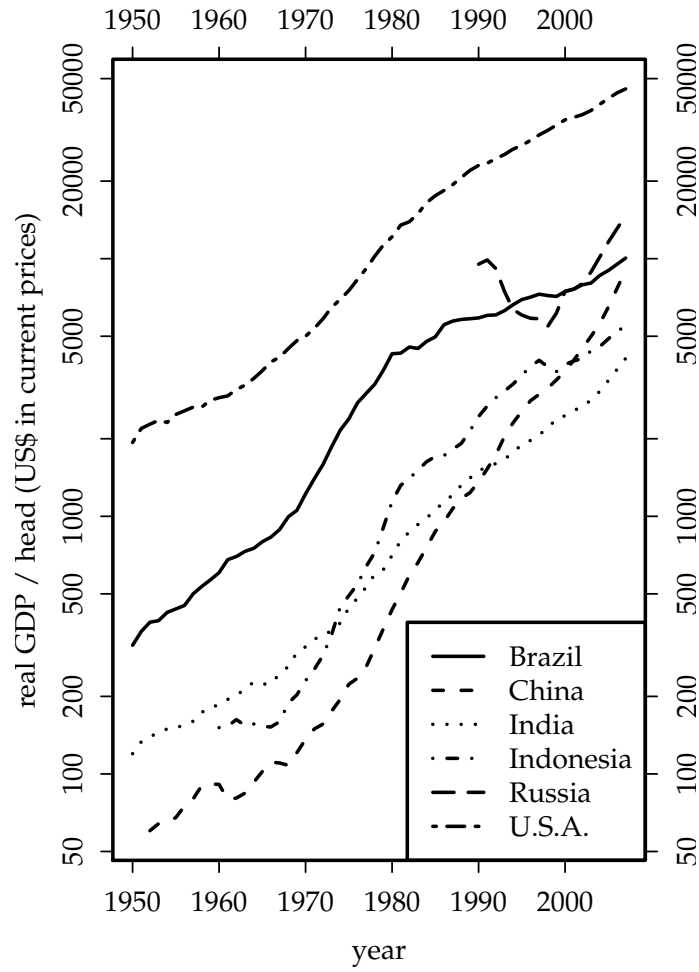
If we feel that the lower graph is too flat then we can 'cut-and-stack' it as follows:



In the following example the graph on the left has a slope of about 45° . This makes it easier to see the convexity of the curve.

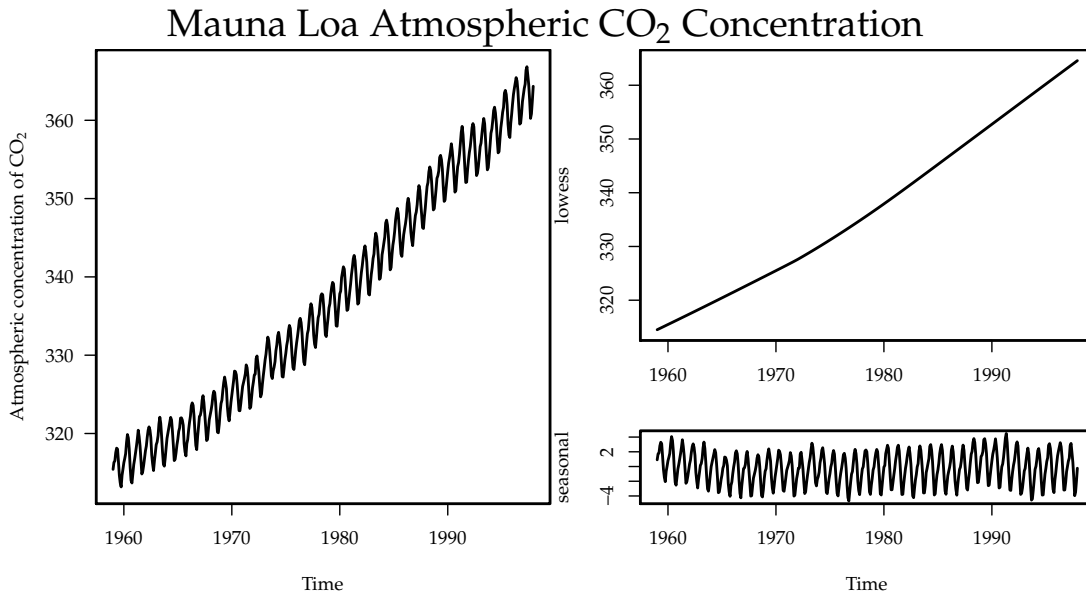


Also in the following graph lines have a slope of about 45° . This makes it easier to compare the different slopes.

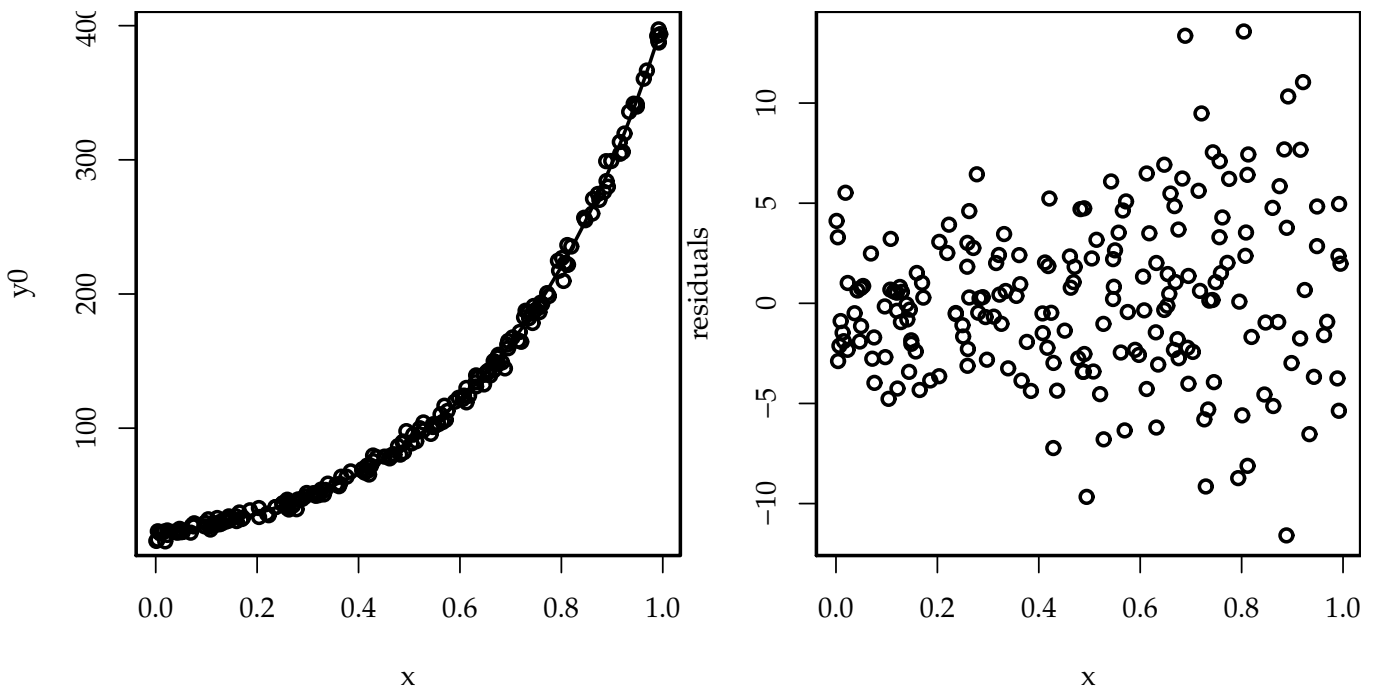


2.6 What to present

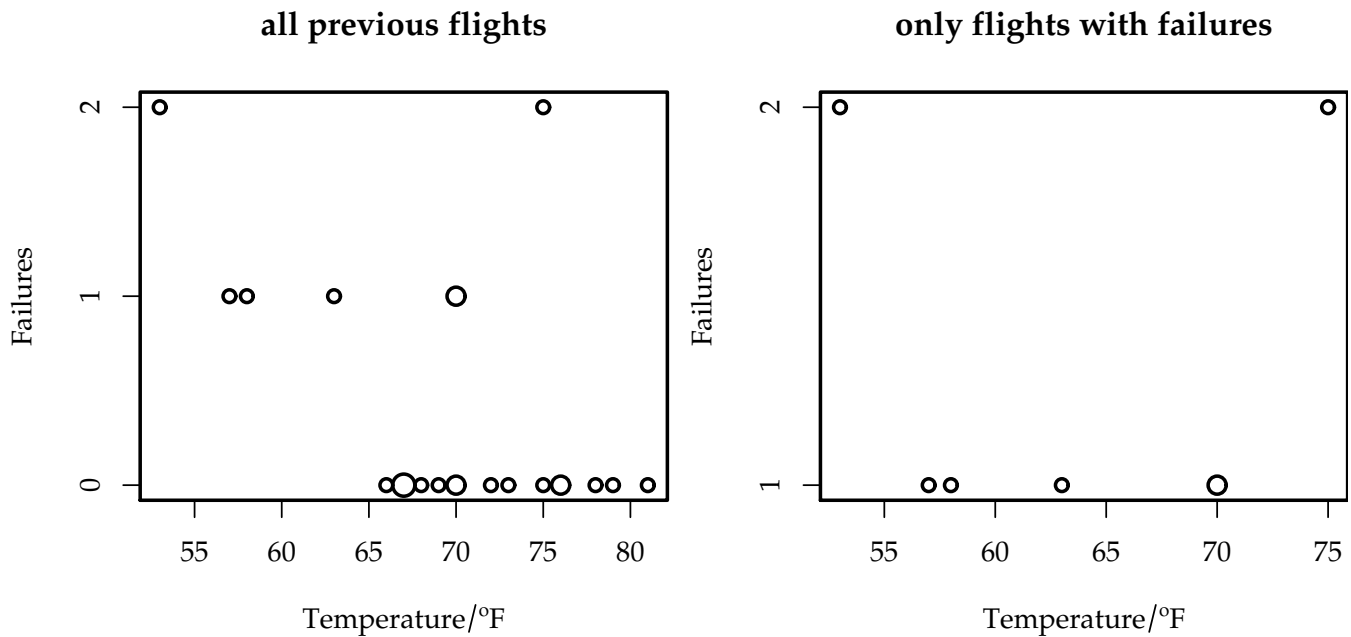
2.6.1 Structuring content



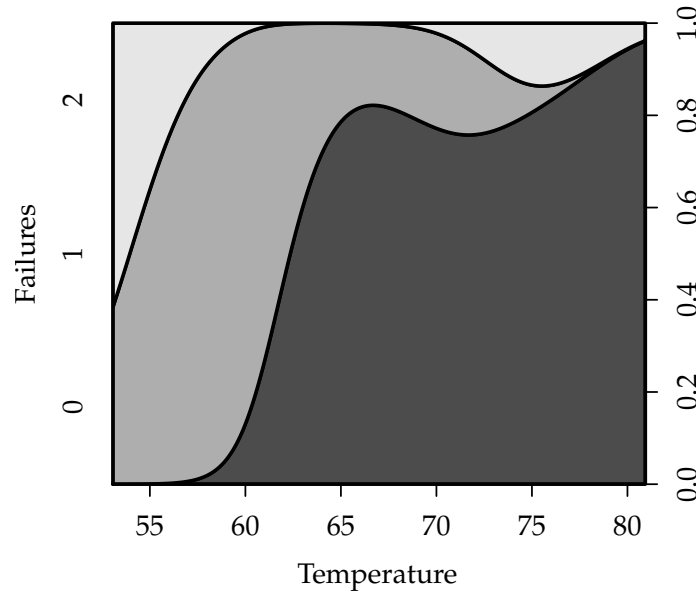
Sometimes it helps to show residuals in a separate graph. In the following example only the graph on the right shows that noise increases for large values of x .



2.6.2 Don't discard parts of your data

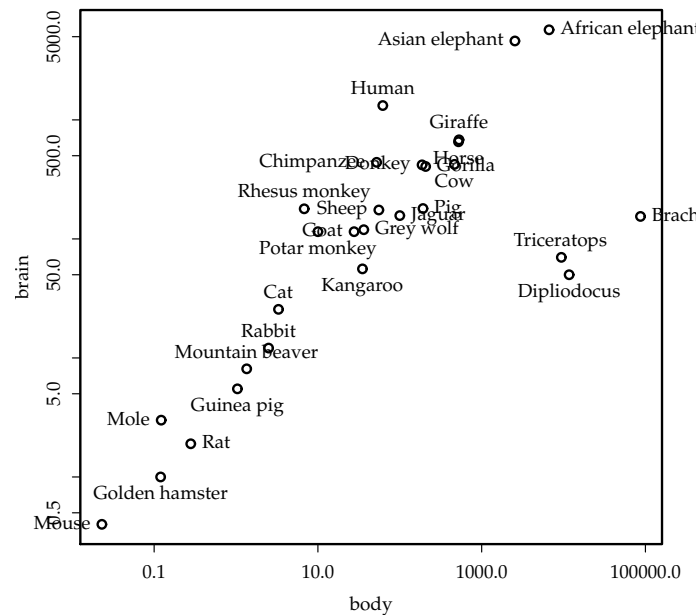


This example illustrates the dire consequences of discarding “irrelevant” data: Previous to the crash of the space shuttle Challenger on 28 January 1986 engineers noticed that the temperature was much lower (31°F) than at other launches before (53°F to 81°F). NASA managers had a look at only the failures of O-rings from previous flights (diagram on the right), discarding the non-failures. They did not see any pattern in the failures and continued the countdown. Sizes of the symbols are proportional to the number of observations. An alternative way to present this data is a conditional density plot:



2.6.3 Projecting data

Carl Sagan¹ argues that intelligence has something to do with the weight of the brain and the weight of the body. We are supposed to see this from a graph which is similar to the following:

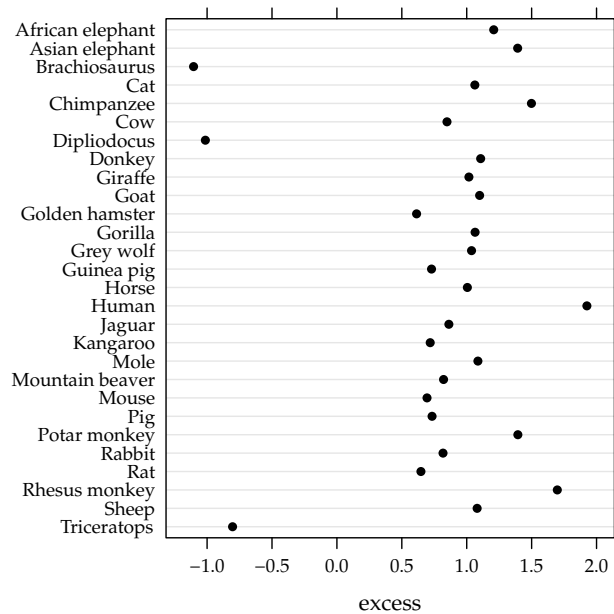
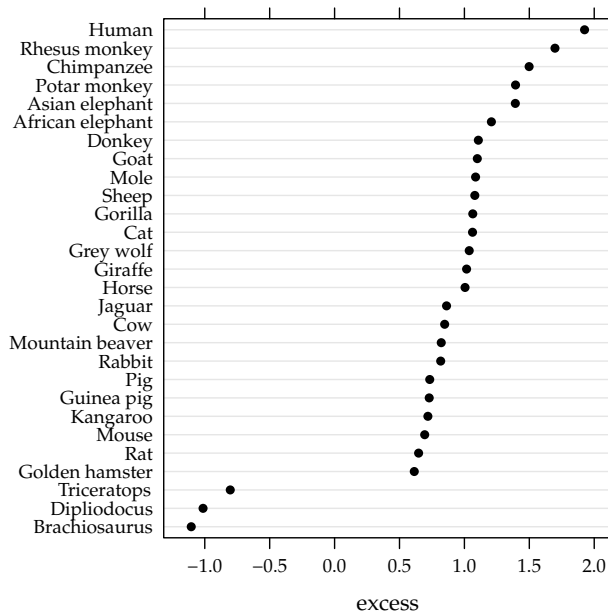


¹The Dragons of Eden: Speculations on the Evolution of Human Intelligence. Random House, New York, 1977.

This is too complicated. Stephen Jay Gould² argues that, a priori, brain size should be proportional to the surface of the body. If we assume that the surface grows quadratically with height, and volume (and weight) cubically, then we should expect that the weight of the brain is proportional to the weight of the body raised to the power of $2/3$. Hence, a reasonable statistic for “excess” brain is $\log(\text{brain mass}) - \frac{2}{3} \log(\text{body mass})$.

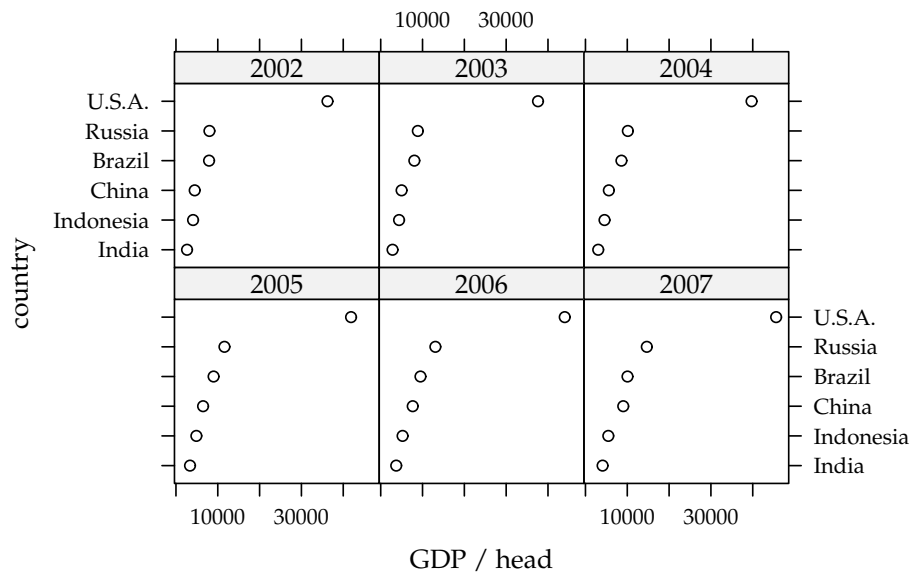
To make it easier to interpret this difference of logs we use logarithms with base 10.

The left graph is ordered by the quantity of “excess brain”, the right one is ordered alphabetically. Often dotplots are easier to understand when they are sorted by the quantity.

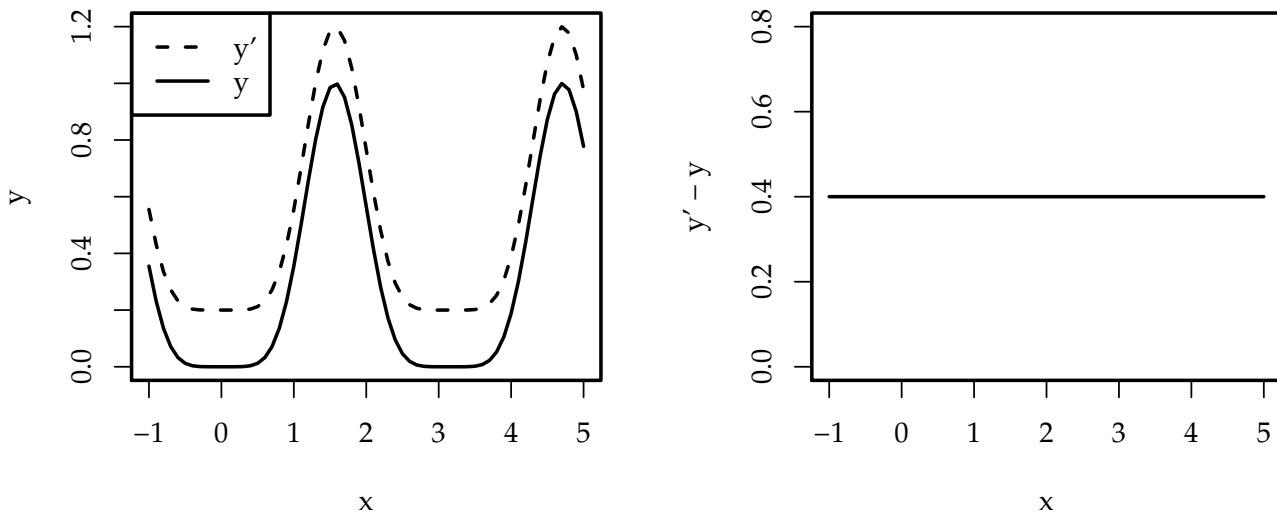


Also for a multiway dotplot ordering by quantity helps. In the following example we use medians of the different categories.

²Ever Since Darwin: Reflections in Natural History. Norton, New York, 1977.



2.6.4 Differences



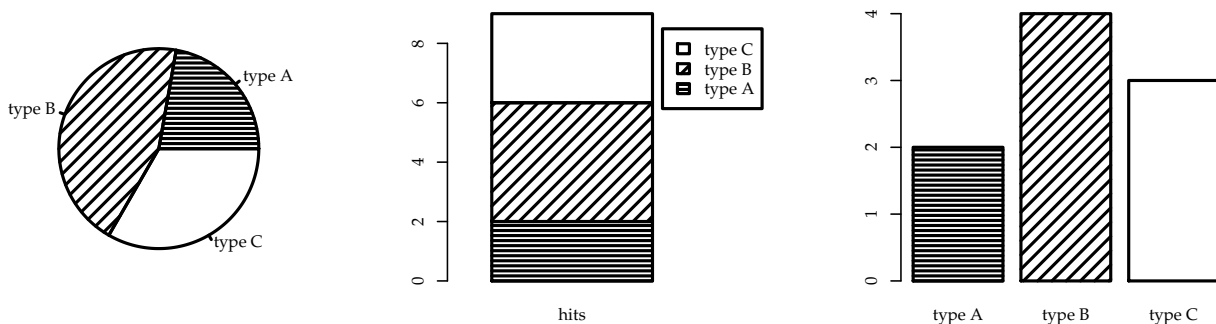
In the graph on the left it is difficult to judge the difference between the two curves. If it is the difference that is interesting, then the graph should show the difference (graph on the right).

3 Presenting nominal data

The case of purely nominal data is rare, although we might want present a simplified version (where only nominal categories matter) of the data in the description.

3.1 Nominal univariate

```
par(mfrow = c(1, 3))
set.seed(123)
dd <- c(30, 15, 0)
aa <- c(0, 45, -45)
hits <- rbinom(3, 10, 0.3)
names(hits) <- c("type A", "type B", "type C")
pie(hits, density = dd, angle = aa)
barplot(cbind(hits), density = dd, angle = aa, col = "black",
        besides = FALSE, legend.text = names(hits), xlim = c(0,
        1.9))
barplot(hits, density = dd, angle = aa, col = "black")
```



Pie chart – The eye is not good at comparing angles. Avoid pie charts.

Stacked bar charts + not much space

- + total size easy to assess
- we need a legend to identify categories

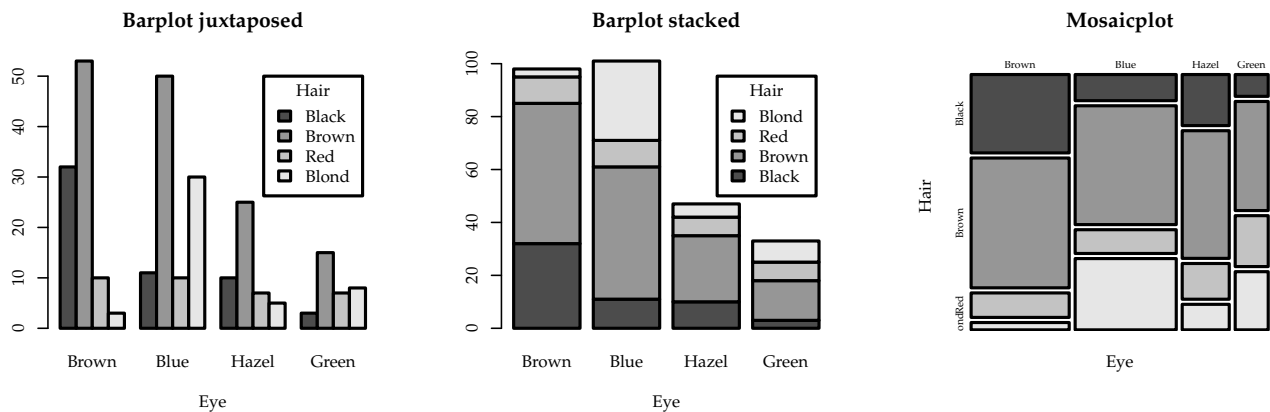
Juxtaposed bar charts + even easier comparison of categories

- + size of sub categories easy to assess

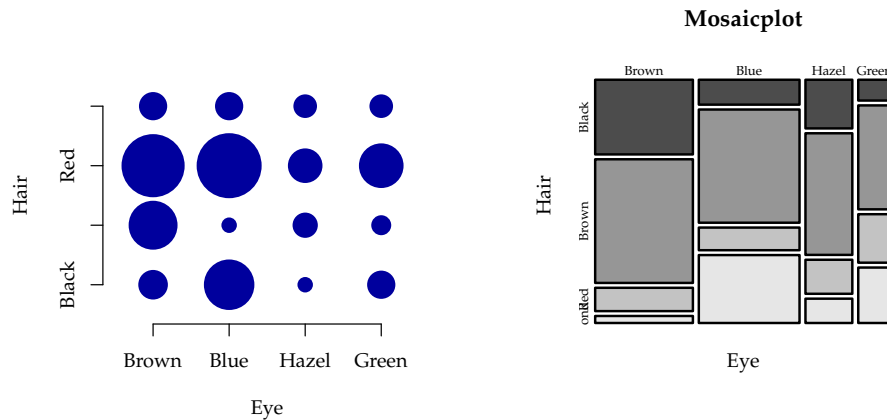
3.2 Nominal bivariate

- Barplots
- Bubbleplots
- Mosaicplots (Hartigan and Kleiner, 1984)
- Dotplots

```
par(mfrow = c(1, 3))
data(HairEyeColor)
xx <- HairEyeColor[, , "Male"]
barplot(xx, legend.text = rownames(xx), xlab = "Eye",
        beside = TRUE, args.legend = list(title = "Hair"),
        main = "Barplot juxtaposed")
barplot(xx, legend.text = rownames(xx), xlab = "Eye",
        args.legend = list(title = "Hair"), main = "Barplot stacked")
mosaicplot(t(xx), color = TRUE, main = "Mosaicplot")
```



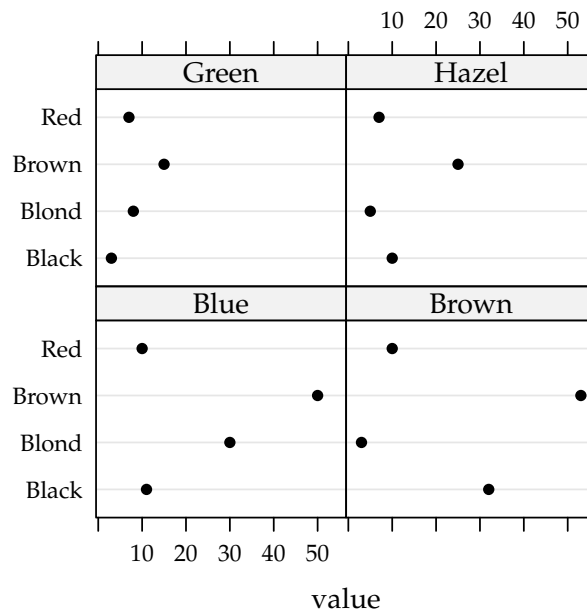
```
library(reshape)
par(mfrow = c(1, 2))
xx <- HairEyeColor[, , "Male"]
zz <- melt(xx)
with(zz, plot(as.numeric(Hair) ~ as.numeric(Eye), pch = 16,
             col = "darkblue", cex = sqrt(value), axes = FALSE,
             xlim = c(0.5, 4.5), ylim = c(0.5, 4.5), xlab = "Eye",
             ylab = "Hair"))
axis(side = 2, labels = rownames(xx), at = 1:4)
axis(side = 1, labels = colnames(xx), at = 1:4)
mosaicplot(t(xx), color = TRUE, main = "Mosaicplot")
```



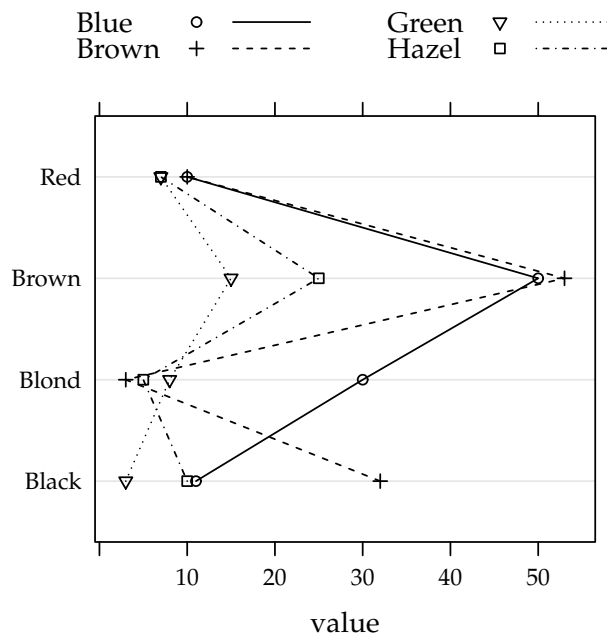
(Spineplots are very similar to mosaicplots)

Multiway dotplots Multiway dotplots are another possibility to present two way count data:

```
| with(zz, print(dotplot(Hair ~ value | Eye)))
```



```
| keys <- list(space = "top", columns = 2, lines = TRUE)
| with(zz, print(dotplot(Hair ~ value, group = Eye, t = c("p",
| "a"), auto.key = keys)))
```



Juxtaposed barplot – hard to see a structure in the sub categories

Stacked barplot + easy to assess sum of observations in each category

Mosaicplot + easy to compare relative frequencies

Bubbleplot + very easy to see structure in categories and sub categories

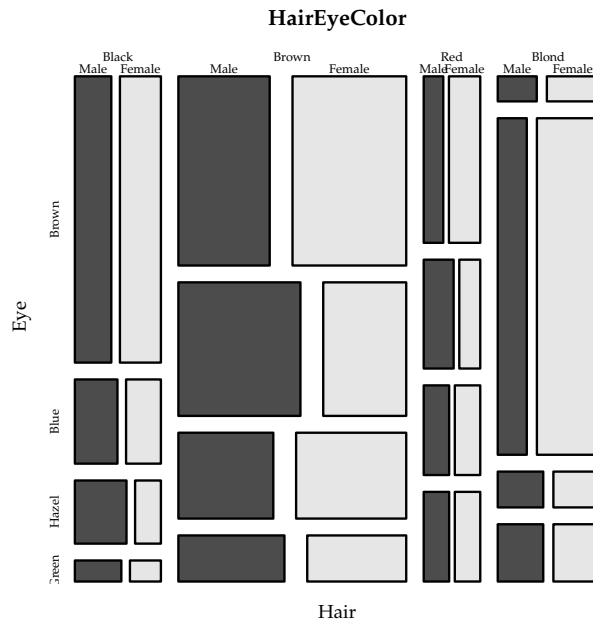
+ good if the number of categories is large (in particular for numeric categories)

Dotplot + easy to look up and to compare absolute frequencies

+ easy to see a pattern

3.3 Nominal multivariate

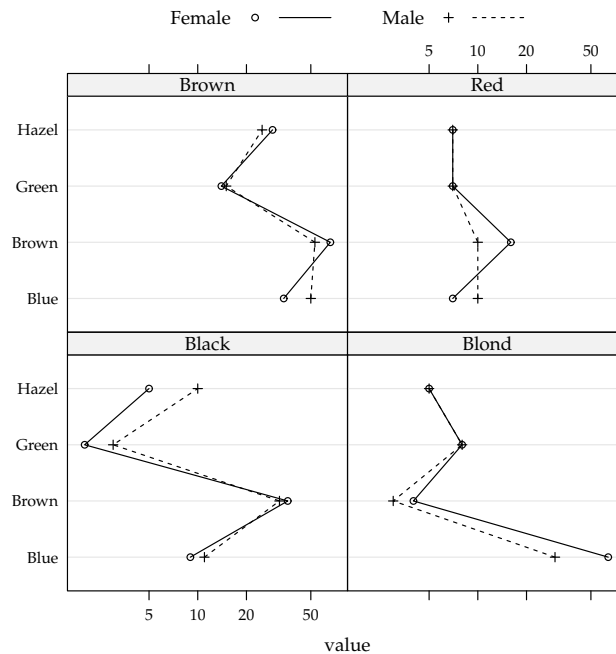
```
data(HairEyeColor)
mosaicplot(HairEyeColor, color = TRUE)
```



```

keys <- list(space = "top", columns = 2, lines = TRUE)
myscale <- list(x = list(log = TRUE, at = c(5, 10, 20,
      50)))
plot(dotplot(Eye ~ value | Hair, group = Sex, t = c("p",
      "a"), scale = myscale, auto.key = keys, data = melt(HairEyeColor)))

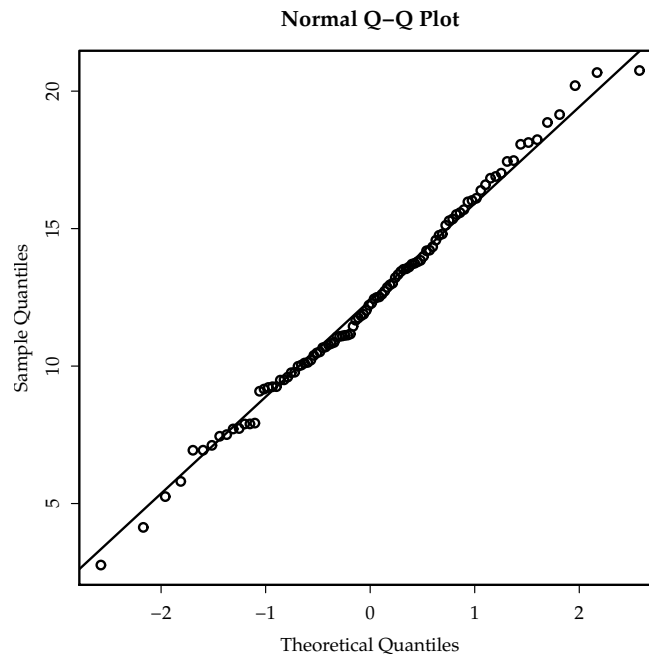
```



4 Presenting continuous data

4.1 Diagnostic plots for continuous variables

```
set.seed(123)
N <- 100
x <- rnorm(N, mean = 12, sd = 4)
qqnorm(x)
qqline(x)
```



`qqnorm` compares with a given (theoretical) distribution. `qqplot` (see 4.2.8) compares with a given empirical distribution.

4.2 One continuous plus one nominal

```
set.seed(123)
N <- 300
t <- rbinom(N, 1, 0.5)
x <- rnorm(N) + 0.6 * t
type <- as.factor(t)
levels(type) <- c("male", "female")
t.test(x ~ type)
```

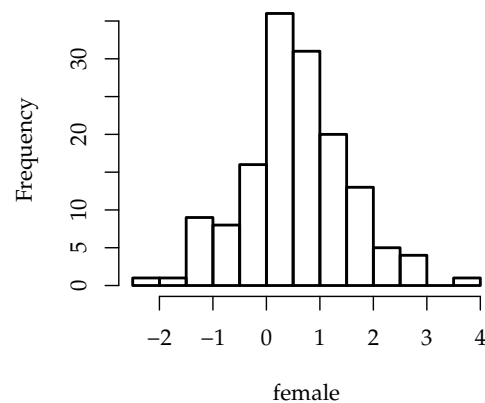
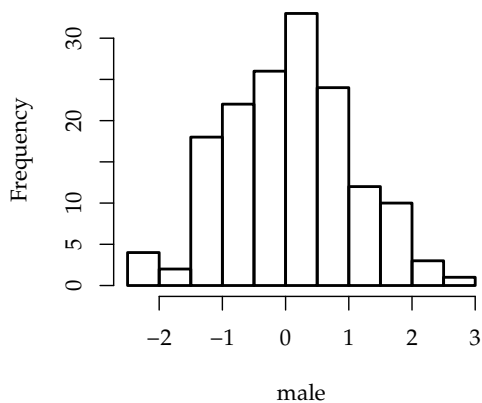
Welch Two Sample t-test

```
data: x by type
t = -4.608, df = 296.914, p-value = 0.000006042
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.7514111 -0.3016662
sample estimates:
 mean in group male mean in group female
      0.05912076      0.58565940
```

4.2.1 Histograms

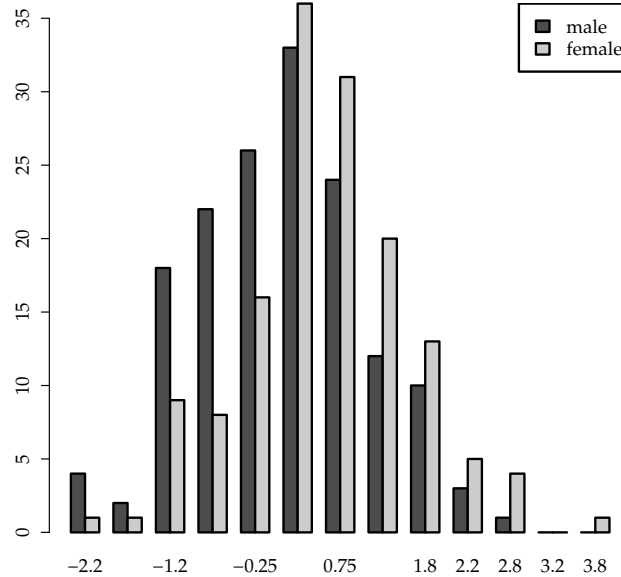
Histograms don't let you see a difference:

```
par(mfrow = c(1, 2))
hist(x[type == "male"], main = "", xlab = "male")
hist(x[type == "female"], main = "", xlab = "female")
```

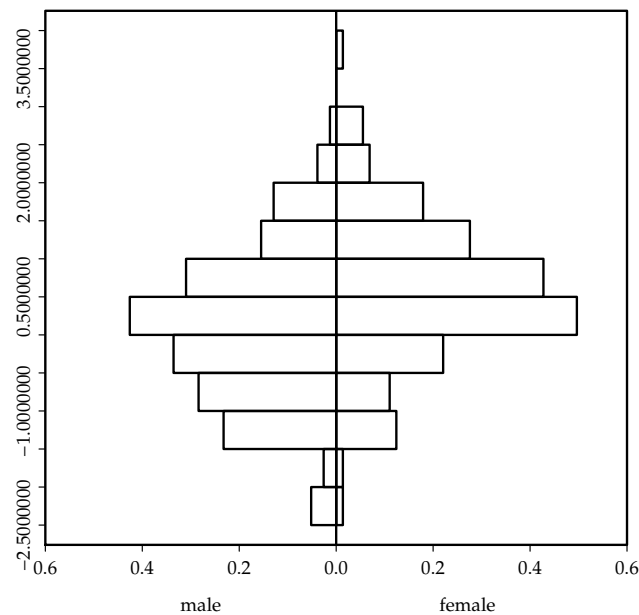


Juxtaposed histograms do not make things much better:

```
library(plotrix)
multhist(list(x[type == "male"], x[type == "female"]),
         col = c(gray(0.3), gray(0.8)))
legend("topright", c("male", "female"), fill = c(gray(0.3),
         gray(0.8)))
```



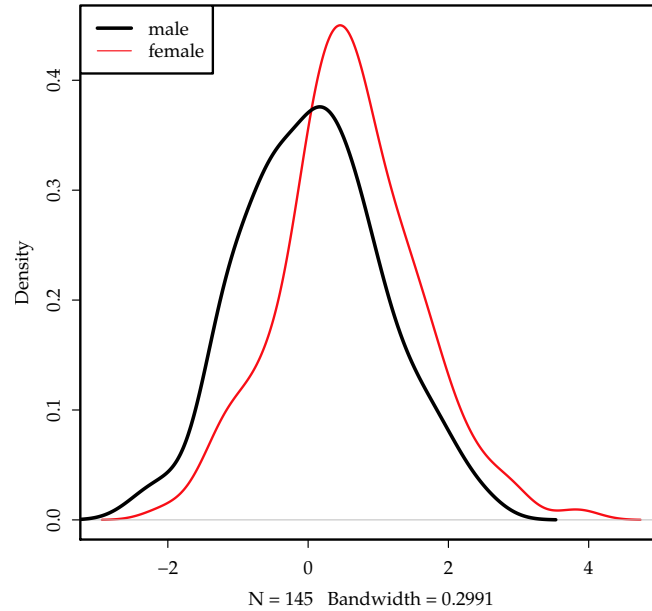
```
library(Hmisc)
histbackback(split(x, type), probability = TRUE, xlab = c("male",
  "female"))
```



4.2.2 Densities

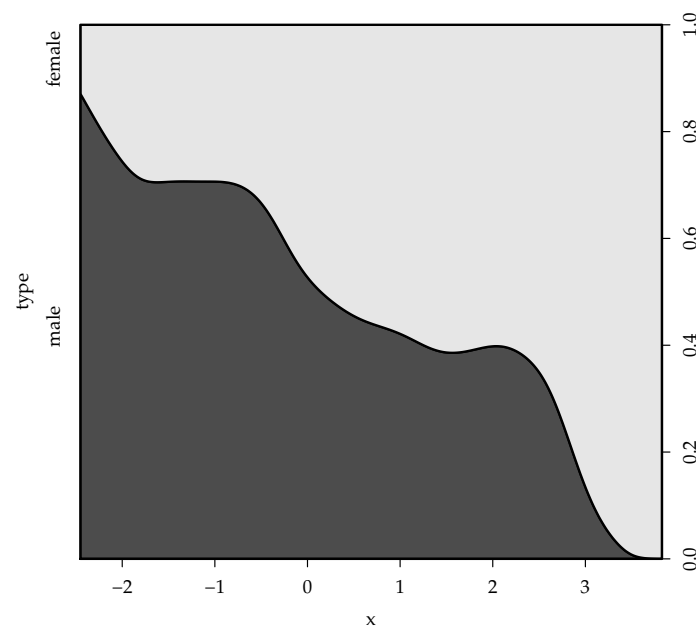
```
plot(density(x[type == "female"]), main = "", col = "red")
lines(density(x[type == "male"]), lwd = 3)
```

```
legend("topleft", c("male", "female"), col = c("black",  
"red"), lwd = c(3, 1), bg = "white")
```



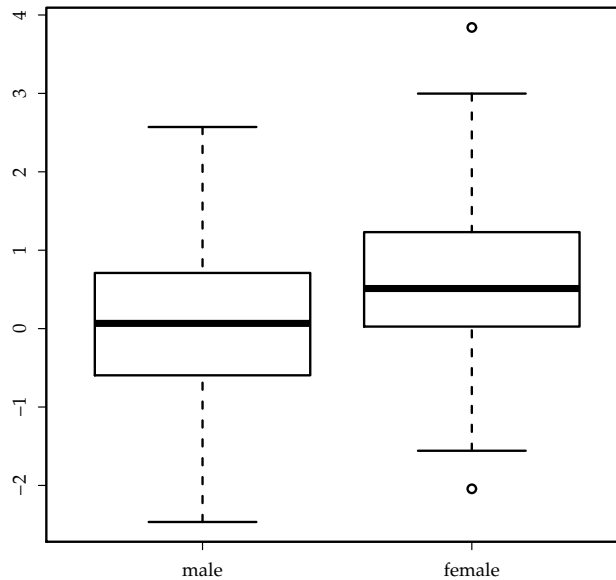
4.2.3 Conditional density plots

```
cdplot(type ~ x)
```



4.2.4 Boxplots

```
| boxplot(x ~ type)
```



Elements of the boxplot

- The median: usually a thick line in the middle
- The “box”, usually from the 25% to the 75% quantile
- The “whiskers”, usually to the most extreme data points which are not more than $1.5 \times$ the interquartile range.

If the data is normally distributed, then the interquartile range covers 1.35 standard deviations. Hence, $1.5 \times$ the interquartile range are 2.02 standard deviations. Outside the whiskers we should, hence, observe 4.3% (or about 5%) of all observations.

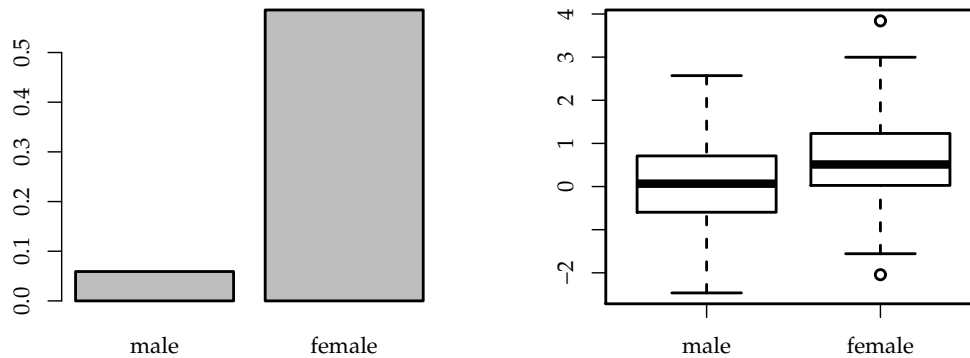
4.2.5 Barplot of means

On the right you see a barplot of means. I show this type only for completeness. Avoid, under all circumstances! You can show much more information on this space. On the right you see, for comparison, a boxplot.

```

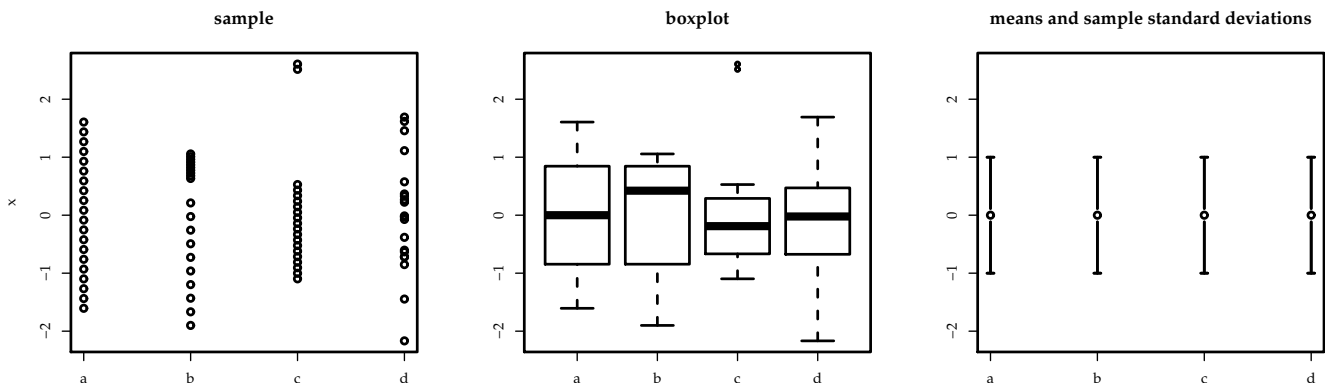
par(mfrow = c(1, 2))
xx <- aggregate(x, list(type), mean)
with(xx, barplot(x, names.arg = Group.1))
boxplot(x ~ type)

```



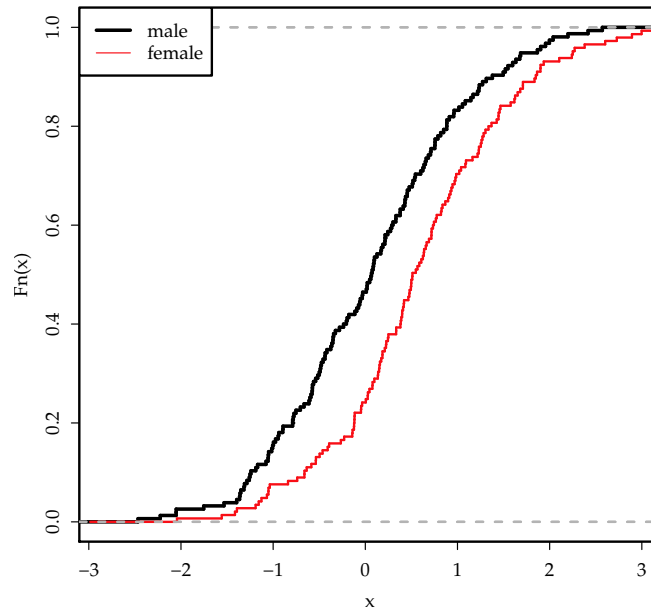
4.2.6 Means and standard deviation

Means and standard deviation are much less informative than boxplots. The following three graphs all show the same four distributions which all have identical sample means and standard deviations (right diagram). Still, scattergrams (left) and boxplots (middle) reveal that the four samples are quite different.



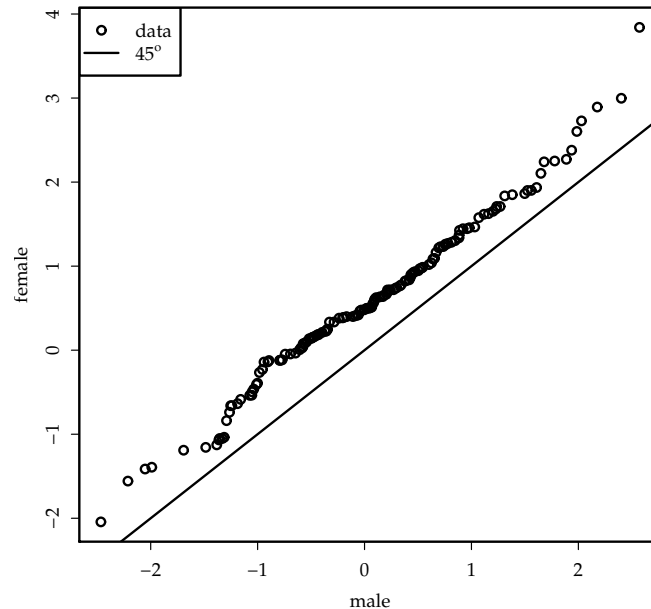
4.2.7 Empirical cumulative densities

```
plot(ecdf(x[type == "male"]), lwd = "3", do.points = FALSE,
     verticals = TRUE, main = "")
lines(ecdf(x[type == "female"]), col = "red", do.points = FALSE,
      verticals = TRUE)
legend("topleft", c("male", "female"), col = c("black",
        "red"), lwd = c(3, 1), bg = "white")
```



4.2.8 QQ-Plots

```
qqplot(x[type == "male"], x[type == "female"], xlab = "male",
       ylab = "female")
abline(a = 0, b = 1)
legend("topleft", c("data", expression(45^o)), pch = c(1,
  NA), lty = c(NA, 1))
```

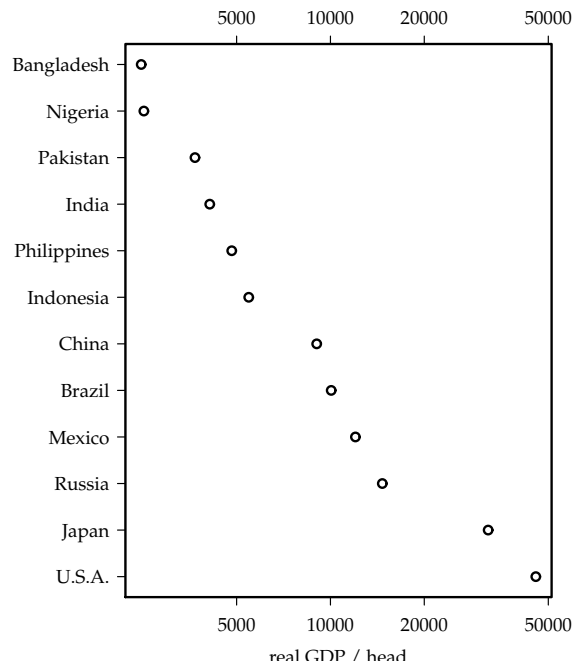


`qqplot` compares with a given empirical distribution. `qqnorm` (see 4.1) compares with a given (theoretical) distribution.

4.2.9 Dot-Plots

We use dot-plots if we have a small number of non-anonymous observations:

```
library(pwt)
par(mar = c(4, 14, 4, 2))
data(pwt6.3)
N <- 12
xx <- subset(pwt6.3, year == 2007 & country != "China Version 2")
levels(xx$country)[grep("China", levels(xx$country))] <- "China"
levels(xx$country)[grep("America", levels(xx$country))] <- "U.S.A."
xx <- subset(xx, pop >= -sort(-xx$pop)[N])
xx <- xx[order(-xx$cgdp), ]
with(xx, plot(cgdp, 1:N, yaxt = "n", ylab = "", log = "x",
             xlab = "real GDP / head"))
axis(2, at = 1:N, labels = xx$country, las = 1)
axis(3)
```

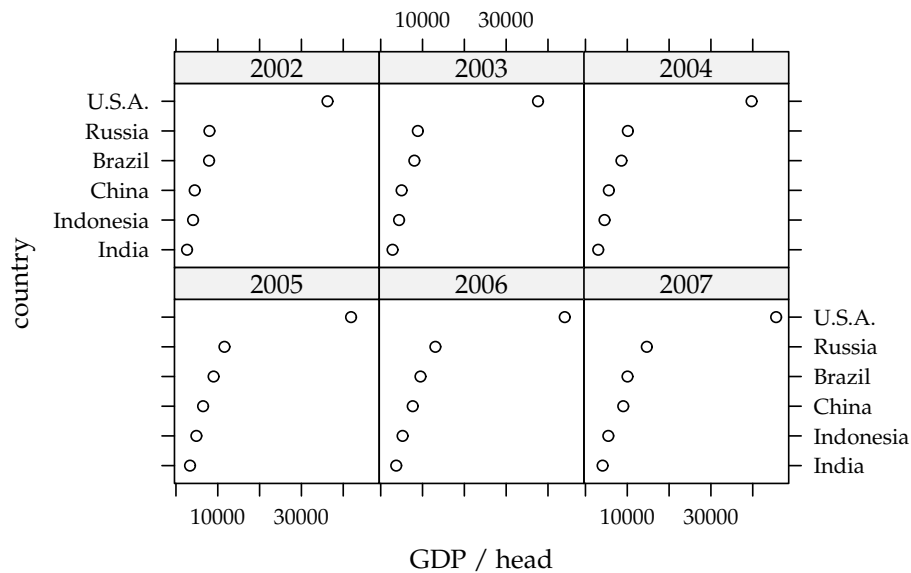


Multiway dot-plots

```

library(pwt)
library(lattice)
lattice.options(default.args = list(as.table = TRUE))
data(pwt6.3)
xx <- with(pwt6.3, aggregate(pop, list(country = country),
  mean))
xx <- subset(xx, country != "China Version 2")
N <- 6
xx <- subset(xx, x >= -sort(-xx$x)[N])
xx <- merge(xx, pwt6.3)
xx <- subset(xx, year > 2001)
levels(xx$country)[grep("United States", levels(xx$country))] <- "U.S.A."
levels(xx$country)[grep("China", levels(xx$country))] <- "China"
xx <- xx[with(xx, order(country, year)), ]
xx <- within(xx, country <- reorder(factor(xx$country),
  xx$cgdp, function(x) median(x, na.rm = TRUE)))
print(xyplot(country ~ cgdp | as.factor(year), data = xx,
  log = "x", xlab = "GDP / head", horizontal = FALSE))

```



4.2.10 Summary

Histograms

- + Everybody understands them
- Don't reveal small differences
- Depend on breaks

Densities

- + Easy to understand
- Need assumptions (must be estimated)
- Depend on bandwidth

Conditional density plots

- + Easy to understand
- + Reveals even small differences between distributions
- Needs assumptions (must be estimated)

Boxplot

- + Shows summary statistics
- Aggregates data

Barplot of means

- Uses a lot of space to show a small amount of information.

ECDF

- + Provides a lot of information
- + Doesn't depend (much) on parameters
- + Reveals even small differences between distributions
- Not so easy to understand

QQ-Plot

- + Provides a lot of information
- + Doesn't depend (much) on parameters
- + Reveals even small differences between distributions
- Only compares two variables

Dot-Plot

- + Provides detailed information
- + Doesn't depend (much) on parameters
- Requires a small number of observations

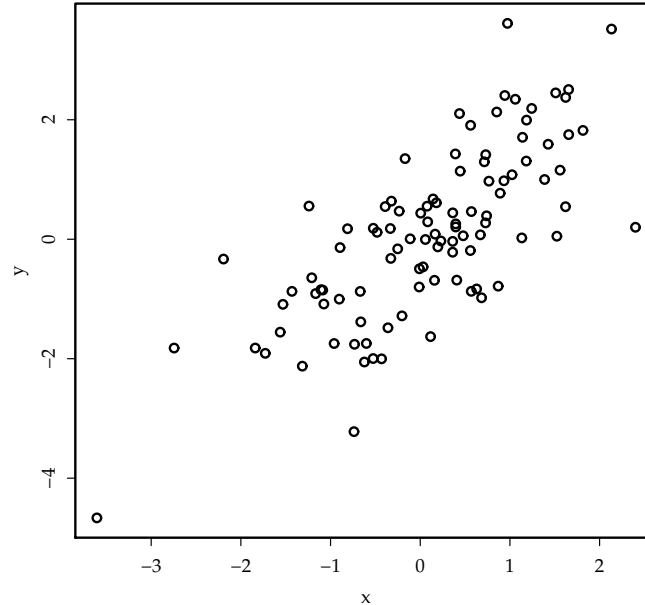
4.3 Two continuous variables

4.3.1 Scatterplot

In this example we create two variables, x and y which are related. Here, there is actually a linear relationship: y is a linear function of x . This is only for the sake of the example and to simplify things.

```
| set.seed(125)  
| N <- 100  
| x <- rnorm(N)  
| y <- x - rnorm(N)
```

```
| plot(x, y)
```

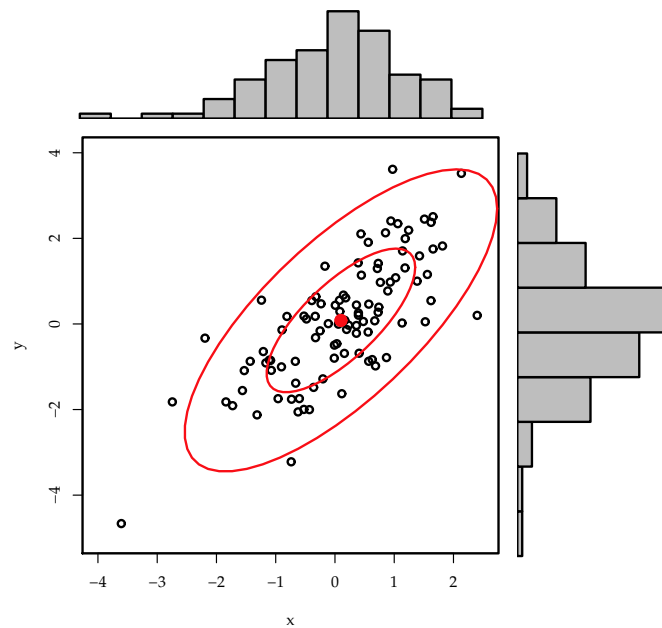


4.3.2 Scatterplot with confidence ellipses

```

library(car)
xhist <- hist(x, breaks = 10, xlim = xrange, plot = FALSE)
yhist <- hist(y, breaks = 10, ylim = yrange, plot = FALSE)
xrange <- range(xhist$breaks)
yrange <- range(yhist$breaks)
top <- max(c(xhist$counts, yhist$counts))
layout(matrix(c(2, 0, 1, 3), 2, 2, byrow = TRUE), c(3,
  1), c(1, 3), TRUE)
par(mar = c(4, 4, 1, 1))
plot(x, y, xlim = xrange, ylim = yrange)
dataEllipse(x, y, levels = c(0.5, 0.95), plot.points = FALSE)
par(mar = c(0, 3, 1, 1))
barplot(xhist$counts, axes = FALSE, ylim = c(0, top),
  space = 0)
par(mar = c(3, 0, 1, 1))
barplot(yhist$counts, axes = FALSE, xlim = c(0, top),
  space = 0, horiz = TRUE)

```

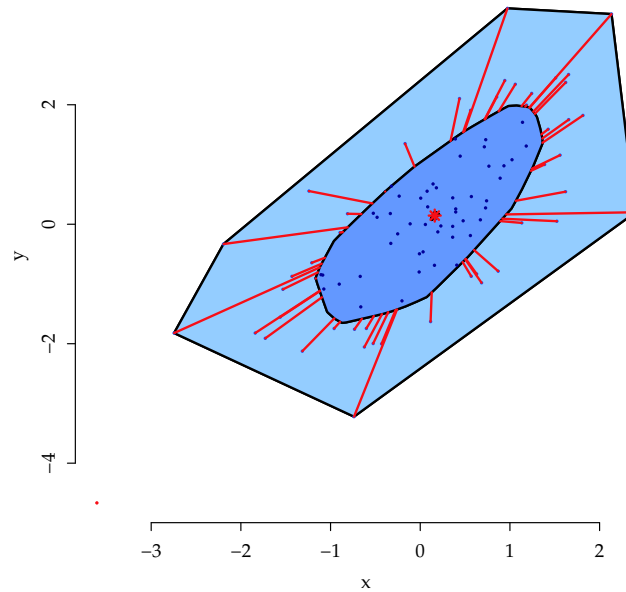


4.3.3 Bagplot

- The dark-blue area: The “bag”. This area contains 50% of all observations.
- The light-blue area: Contains all points which are in the bag 3 times expanded.
- Points outside the light-blue area are considered outliers.

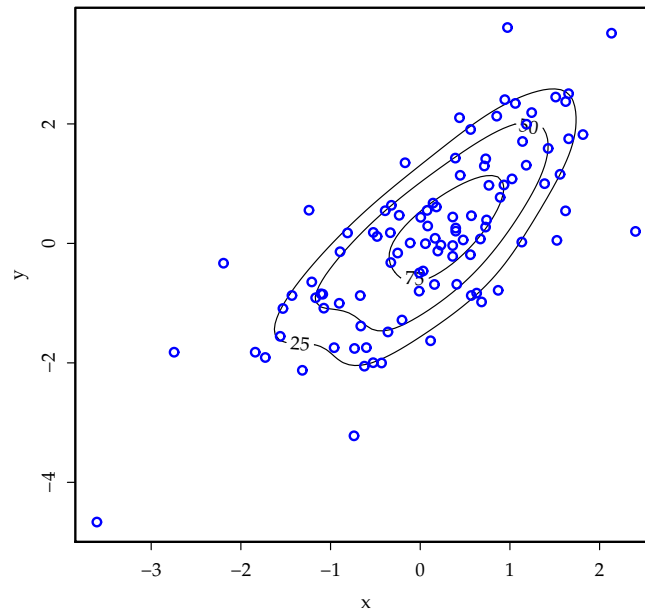
P. J. Rousseeuw, I. Ruts, J. W. Tukey (1999)

```
| library(aplpack)  
| bagplot(x, y)
```



4.3.4 Kernel densities

```
library(ks)
data <- cbind(x, y)
H.scv <- Hscv(data)
fhat <- kde(data, H = H.scv)
plot(fhat, drawpoints = TRUE)
```

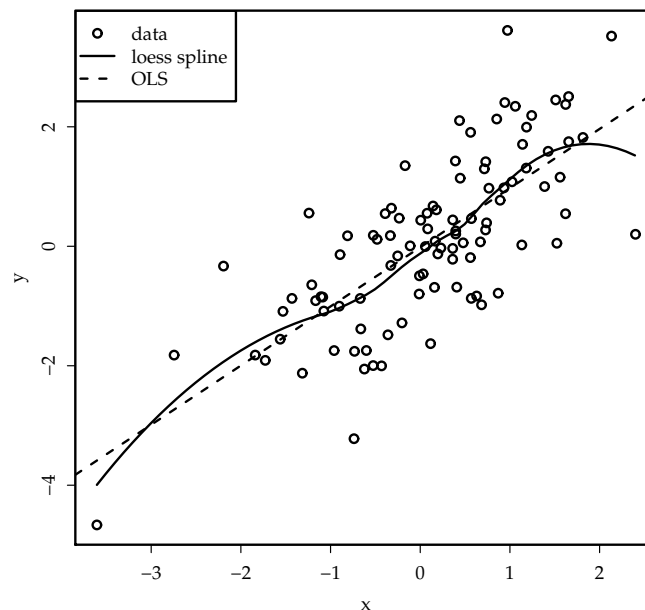


4.3.5 Scatterplot plus loess and regression line

```

plot(x, y)
xy.lo <- loess(y ~ x)
newx <- seq(min(x), max(x), 0.1)
xy.pred <- predict(xy.lo, newdata <- newx)
lines(newx, xy.pred)
abline(lm(y ~ x), lty = 2)
legend("topleft", c("data", "loess spline", "OLS"), pch = c(1,
  NA, NA), lty = c(NA, 1, 2))

```

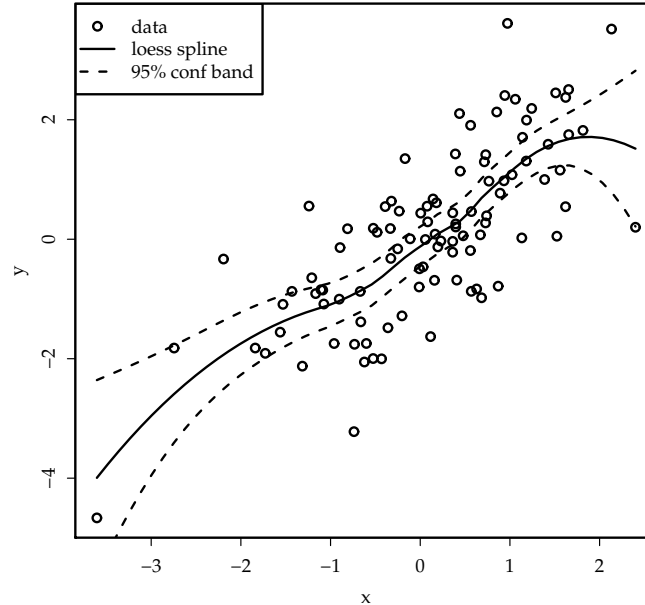


4.3.6 Confidence bands for smooth lines

```

plot(x, y)
xy.lo <- loess(y ~ x)
newx <- seq(min(x), max(x), length.out = 50)
xy.pred <- predict(xy.lo, newdata <- newx, se = TRUE)
lines(newx, xy.pred$fit)
lines(newx, xy.pred$fit + qnorm(0.975) * xy.pred$se,
  lty = 2)
lines(newx, xy.pred$fit + qnorm(0.025) * xy.pred$se,
  lty = 2)
legend("topleft", c("data", "loess spline", "95% conf band"),
  pch = c(1, NA, NA), lty = c(NA, 1, 2))

```

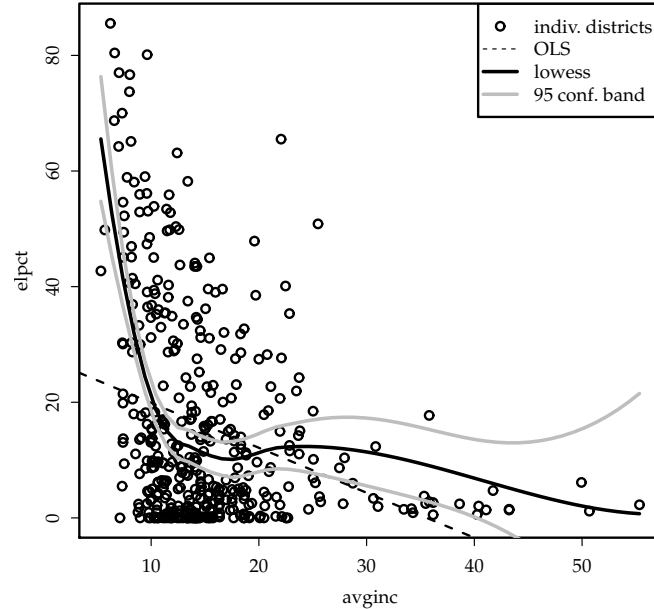


Another example for Loess

```

library(Ecdat)
data(Caschool)
plot(elpct ~ avginc, data = Caschool)
ca.lo <- loess(elpct ~ avginc, data = Caschool)
newinc <- with(Caschool, seq(min(avginc), max(avginc),
  length.out = 50))
abline(lm(elpct ~ avginc, data = Caschool), lty = "dashed")
elpct.pred <- predict(ca.lo, newdata <- newinc, se = TRUE)
lines(newinc, elpct.pred$fit, lwd = 3)
lines(newinc, elpct.pred$fit + qnorm(0.975) * elpct.pred$se,
  col = "grey", lwd = 3)
lines(newinc, elpct.pred$fit + qnorm(0.025) * elpct.pred$se,
  col = "grey", lwd = 3)
legend("topright", c("indiv. districts", "OLS", "lowess",
  "95 conf. band"), lty = c(NA, "dashed", "solid",
  "solid"), pch = c(1, NA, NA, NA), col = c("black",
  "black", "black", "grey"), lwd = c(NA, 1, 3, 3, 3))

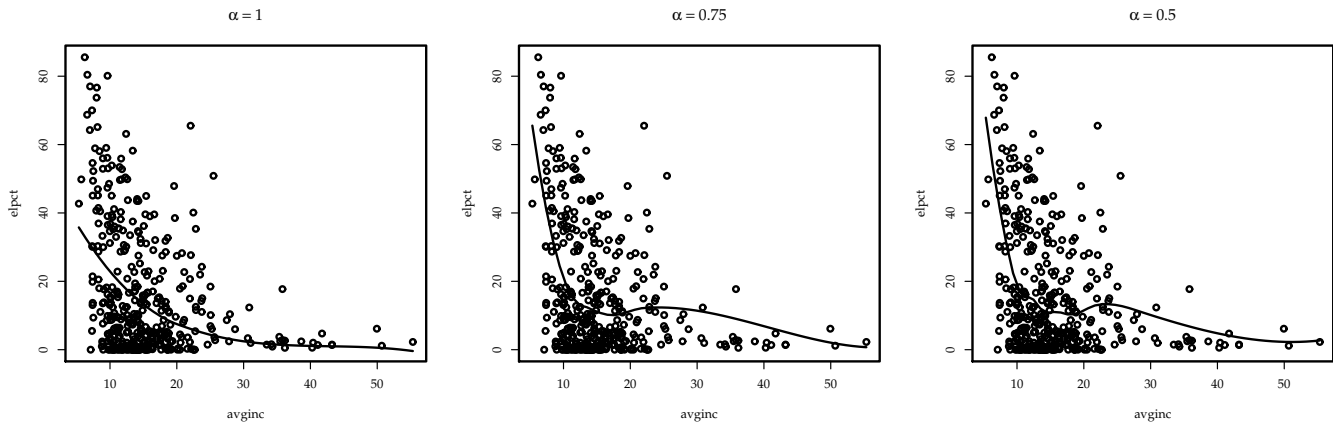
```



4.3.7 Which loess?

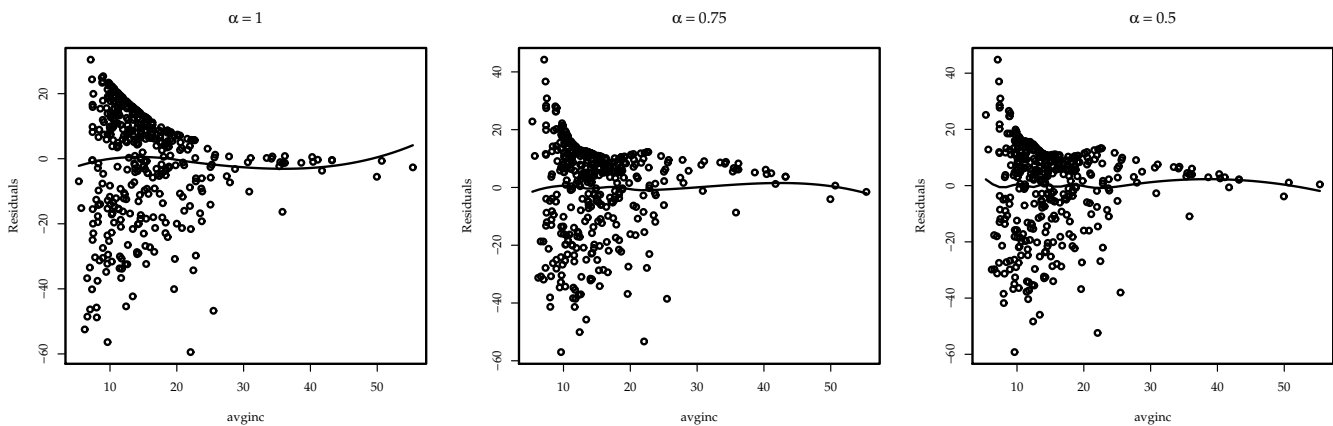
Loess is governed by a parameter α . R uses a default of $\alpha = 0.75$. How can we find out a 'good' value for α ?

```
par(mfrow = c(1, 3))
XX <- function(span) {
  plot(elpct ~ avginc, data = Caschool, main = substitute(list(alpha) ==
    a, list(a = span)))
  ca.lo <- loess(elpct ~ avginc, span = span, data = Caschool)
  ca.pred <- predict(ca.lo, newdata <- newinc)
  lines(newinc, ca.pred, data = Caschool)
}
XX(1)
XX(0.75)
XX(0.5)
```



It can be easier to compare the influence of different values of α if we look only at the residuals:

```
par(mfrow = c(1, 3))
XX <- function(span) {
  ca.lo <- loess(elpct ~ avginc, span = span, data = Caschool)
  resid <- predict(ca.lo) - Caschool$elpct
  plot(resid ~ avginc, ylab = "Residuals", data = Caschool,
       main = substitute(list(alpha) == a, list(a = span)))
  lines(newinc, predict(loess(resid ~ Caschool$avginc,
                             span = span), newdata <- newinc))
}
XX(1)
XX(0.75)
XX(0.5)
```



Scatterplot

+ makes no assumptions

- with a large dataset the graph might be cluttered
- with categorical data points might superimpose

Confidence ellipses

- assumes a linear relationship

Bagplot

- not very well known

Kernel densities

- + easy to understand
- relies on assumptions (must be estimated, depend on bandwidth)

Regression line

- Assumes a linear causal relationship

Loess

- Assumes a causal relationship

4.4 Paired data

Sometimes two-dimensional data comes in pairs where both elements can be compared with each other. One value might be recorded before, the other after a treatment.

If the data are highly correlated then a standard scatterplot (left diagram) wastes a lot of space top left and bottom right from the 45°-line.

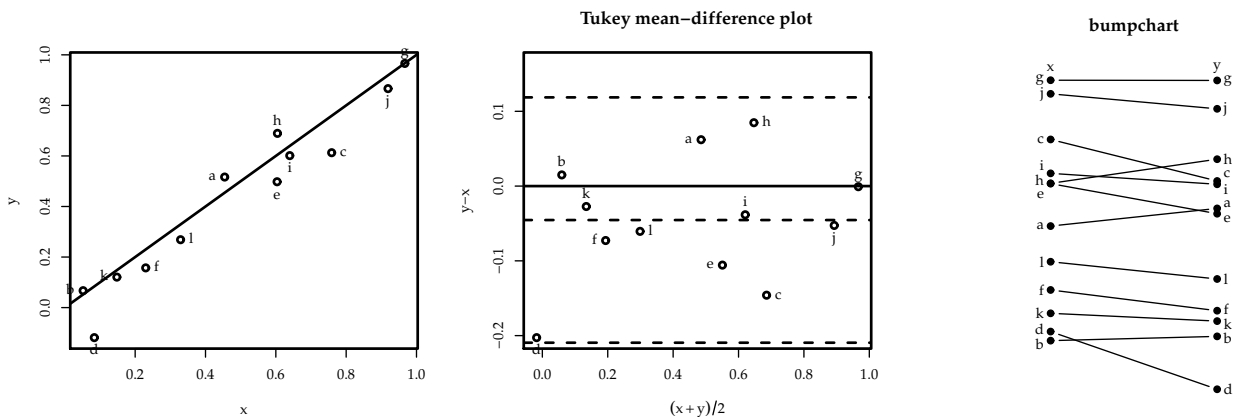
The Tukey mean-difference plot (also known as Bland-Altman plot) basically rotates the diagram by 45° and, thus, can save space. (The Bland-Altman plot aims at showing agreement of the two elements of the pairs and, hence, also shows the mean of the differences \pm two standard deviations.)

The bumpchart presents essentially the same information, but with a focus on the identity of the observations. We would usually not do this for anonymous observations, but, e.g. if observations are for countries or for cities.

```

par(mfrow = c(1, 3))
N <- 12
x <- runif(N)
y <- x + 0.1 * rnorm(N)
plot(x, y)
abline(a = 0, b = 1)
thigmophobe.labels(x, y, letters[1:N])
u <- (x + y)/2
v <- (y - x)
plot(u, v, xlab = expression((x + y)/2), ylab = "y-x",
     ylim = range(c(2 * sd(v), -2 * sd(v), v)), main = "Tukey mean-difference",
     thigmophobe.labels(u, v, letters[1:N])
abline(h = 0)
abline(h = mean(v) + c(-2, 0, 2) * sd(v), lty = 2)
xx <- cbind(x, y)
rownames(xx) <- letters[1:N]
bumpchart(xx, rank = FALSE, main = "bumpchart")

```



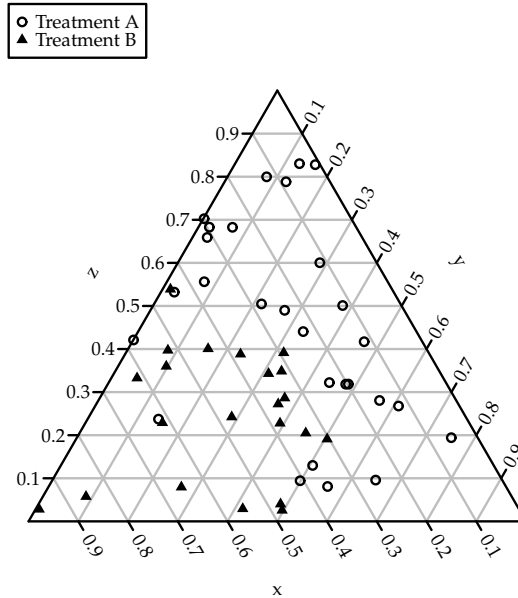
4.5 Three-dimensional simplex

Three dimensional variables are notoriously difficult to present. However, quantities like prices and probabilities can often be conveniently represented in a simplex:

```

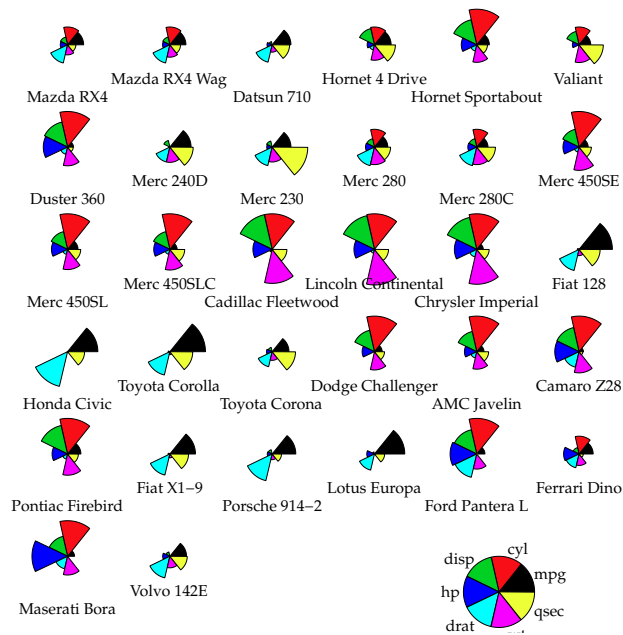
par(mar = c(0, 0, 0, 0))
xx <- matrix(runif(150), ncol = 3)
type <- xx[, 1] > 0.5
xx <- xx/rowSums(xx)
colnames(xx) <- c("x", "y", "z")
triax.plot(xx, show.grid = TRUE, pch = 16 * type + 1)
legend("topleft", c("Treatment A", "Treatment B"), pch = c(1,
  17))

```



4.6 Stars

```
stars(mtcars[, 1:7], len = 0.8, key.loc = c(12, 1.5),
      draw.segments = TRUE)
```



5 Using R

For the purpose of the course we take R as an example for one statistical language. Even if you use other languages for your work, you will find that the concepts are similar.

5.1 Installation of R

On the Homepage of the R Projekt you find in the menu on the left a link Download / CRAN. This link leads to a choice of “mirrors”. If you are in Jena, the GWDG Mirror in Göttingen might be fast. There you also find instructions how to install R on your OS.

Installation of Libraries If the command `library` complains about not being able to find the required library, then the library is most likely not installed. The command

```
| install.packages("Ecdat")
```

installs the library `Ecdat`. Some installations have a menu “Packages” that allows you to install missing libraries. Users of operating systems of Microsoft find support at the FAQ for Packages.

5.2 Types and assignments

R knows about different types of data. We will meet some types in this chapter. To assign a number (or a value, or any object) to a variable, we use the operator `<-`

```
| x <- 4
```

R stores the result of this assignment as `double`

```
| typeof(x)
```

```
[1] "double"
```

Now we can use `x` in our calculations:

```
| 2 * x
```

```
[1] 8
```

```
| sqrt(x)
```

```
[1] 2
```

Often our calculations will not only involve a single number (a scalar) but several which are connected as a vector. Several numbers are connected with `c`

```
| x <- c(21, 22, 23, 24, 25, 16, 17, 18, 19, 20)
|x
```

```
[1] 21 22 23 24 25 16 17 18 19 20
```

When we need a long list of subsequent numbers, we use the operator `:`

```
| 21:30
```

```
[1] 21 22 23 24 25 26 27 28 29 30
```

```
| y <- 21:30
```

Subsets We can access single elements of a variable with `[]`

```
| x[1]
```

```
[1] 21
```

When we want to access several elements at the same time, we simply use several indices (which are connected with `c`). We can use this to change the sequence of values (e.g. to sort).

```
| x[c(3, 2, 1)]
```

```
[1] 23 22 21
```

```
| x[3:1]
```

```
[1] 23 22 21
```

```
| x
```

```
[1] 21 22 23 24 25 16 17 18 19 20
```

(to sort a long vector we would use the function `order`).

```
| order(x)
```

```
[1] 6 7 8 9 10 1 2 3 4 5
```

```
| x[order(x)]
```

```
[1] 16 17 18 19 20 21 22 23 24 25
```

Negative indices drop elements:

```
| x[-1:-3]
```

```
[1] 24 25 16 17 18 19 20
```

Logicals Logicals can be either TRUE or FALSE. When we compare a vector with a number, then all the elements will be compared (this results from the recycling rule, see below):

```
| x < 20
```

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

We can use logicals as indices, too:

```
| x[x < 20]
```

```
[1] 16 17 18 19
```

Characters Not only numbers, also character strings can be assigned to a variable:

```
| x <- "Mary"
```

We can also work with vectors of character strings:

```
| x <- c("John", "Mary", "Jane")
| x[2]
```

```
[1] "Mary"
```

```
| x[3] <- "Lucy"
| x
```

```
[1] "John" "Mary" "Lucy"
```

Factors Often it is clumsy to store a string of characters again and again if this string appears in the dataset several times. We might, e.g., want to store whether an observation belongs to a man or a woman. This can be done in an efficient way by storing 2 for "male", and 1 for "female".

```
| x <- as.factor(c("male", "female", "female", "male"))
| levels(x)
```

```
[1] "female" "male"
```

```
| x[2]
```

```
[1] female
Levels: female male
```

```
| as.numeric(x)
```

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

Usually the first level in a factor is the level that comes first on the alphabet. If we do not want this, we can `relevel` a factor:

```
| x <- relevel(x, "male")
| x
```

```
[1] male   female female male
Levels: male female
```

```
| as.numeric(x)
```

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

Note that the meaning of the values remains unchanged.

Sometimes, when we have more than only two levels, we want to order levels of a factor along a third variable. This is done by `reorder`.

```
| y <- c(12, 7, 8, 11)
| reorder(x, y)
```

```
[1] male   female female male
attr(,"scores")
  male female
  11.5    7.5
Levels: female male
```

5.3 Functions

R knows many built-in functions:

```
| mean(x)
| median(x)
| max(x)
| min(x)
| length(x)
| unique(c(1, 2, 3, 4, 1, 1, 1))
```

When we need more, we can write our own:

```
| square <- function(x) {
|   x * x
| }
```

The last expression in a function (here `x*x`) is the return value. Now we can use the function.

```
| square(7)
```

```
[1] 49
```

When we want to apply a function to many numbers, `sapply` helps:

```
| range <- 1:10
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

```
| sapply(range, square)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

With `sapply` we do not have to define a name for a function:

```
| sapply(range, function(x) x * x)
```

```
[1] 1 4 9 16 25 36 49 64 81 100
```

5.4 Random numbers

Random numbers can be generated for rather different distributions. R calculates pseudo-random numbers, i.e. R picks numbers from a very long list that appears random. Where we start in this long list is determined by `set.seed`:

```
| set.seed(123)
```

10 pseudo-random numbers from a normal distribution can be obtained with

```
| rnorm(10)
```

```
[1] -0.56047565 -0.23017749 1.55870831 0.07050839 0.12928774  
[6] 1.71506499 0.46091621 -1.26506123 -0.68685285 -0.44566197
```

We get the same list when we initialise the list with the same starting value:

```
| set.seed(123)  
| rnorm(10)
```

```
[1] -0.56047565 -0.23017749  1.55870831  0.07050839  0.12928774
[6]  1.71506499  0.46091621 -1.26506123 -0.68685285 -0.44566197
```

This is very useful, when we want to replicate the same “random” results. 10 uniformly distributed random numbers from the interval [100,200] can be obtained with

```
| runif(10, min = 100, max = 200)
```

```
[1] 188.9539 169.2803 164.0507 199.4270 165.5706 170.8530 154.4066
[8] 159.4142 128.9160 114.7114
```

Often we use random numbers when we simulate (stochastic) processes. To replicate a process we use the command `replicate`. E.g.

```
| replicate(10, mean(rnorm(100)))
```

```
[1]  0.016749257 -0.024755975  0.061320514 -0.028205903  0.087712299
[6] -0.025113287 -0.141043824  0.123989920  0.109293109 -0.002743263
```

takes 10 times the mean of each 100 pseudo-normally distributed random numbers.

5.5 Example Datasets

We just saw that the command `c` allows us to describe the elements of a vector. For long datasets this is not very convenient. R contains already a lot of example datasets. These datasets are, similar to statistical functions, organised in libraries. To save space and time R does not load all libraries initially. The command `library` allows us to load a library with a dataset at any time. The library `Ecdat` provides a lot of interesting economic datasets. The library `memisc` gives access to some interesting functions that help us organising our data.

When we need a specific function and we do not know in which library to look for this function we can use the command `RSiteSearch` or the R Site Search Extension for Firefox.

The dataset `BudgetFood` is, e.g., contained in the library `Ecdat`.

```
| data(BudgetFood, package = "Ecdat")
```

To really see the numbers, we can use the command `fix`:

```
| fix(BudgetFood)
```

Usually we do not want to see many numbers. Instead we want to derive (in a structured way) a few numbers (parameters, confidence intervals, p -values,...) The command `help` aids us in finding out the meaning of the numbers of the different columns of a dataset.

```
| help(BudgetFood)
```

An important command to get a summary is `summary`

```
| summary(BudgetFood)
```

How can we access specific columns from our dataset? Since R may have several datasets at the same time in its memory, there are several possibilities. One possibility is to append the name of the dataset `BudgetFood` with a `$` and then the name of the column.

```
| BudgetFood$age
```

```
[1] 43 40 28 60 37 35 40 68  
[ reached getOption("max.print") -- omitted 23964 entries ]]
```

This is helpful when we work with several different datasets at the same time. The example also shows that R does not flood our screen with long lists of numbers. Instead we only see the first few numbers, and then the text “omitted ... entries”.

When we want to use only one dataset, then the command `attach` is helpful.

```
| attach(BudgetFood)  
| age
```

```
[1] 43 40 28 60 37 35 40 68  
[ reached getOption("max.print") -- omitted 23964 entries ]]
```

From now on, all variables will first be searched in the dataset `BudgetFood`. When we no longer want this, then we say

```
| detach(BudgetFood)
```

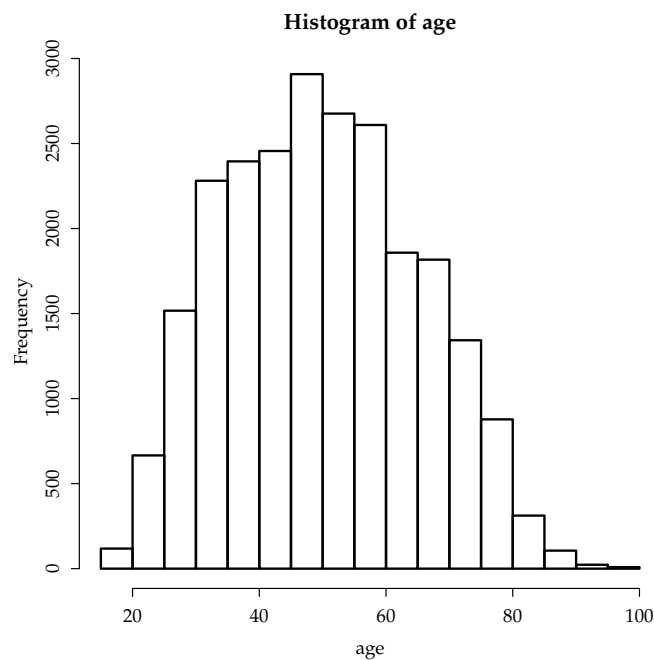
A third possibility is the command `with`:

```
| with(BudgetFood, age)
```

```
[1] 43 40 28 60 37 35 40 68
[ reached getOption("max.print") -- omitted 23964 entries ]]
```

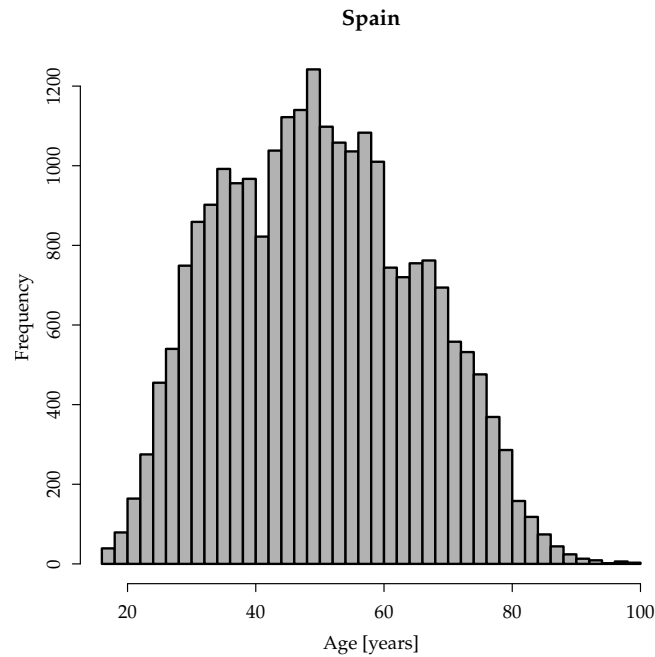
We often use `with` when we use a function and want to refer to a specific dataset in this function. E.g. `hist` shows a histogram:

```
| with(BudgetFood, hist(age))
```



Most commands have several options which allow you to fine-tune the result. Have a look at the help-page for `hist` (you can do this with `help(hist)`). Perhaps you prefer the following graph (where we replaced `width(BudgetFood, ...)` with the option `data=BudgetFood`):

```
| hist(age, data = BudgetFood, breaks = 40, xlab = "Age [years]",
      col = gray(0.7), main = "Spain")
```



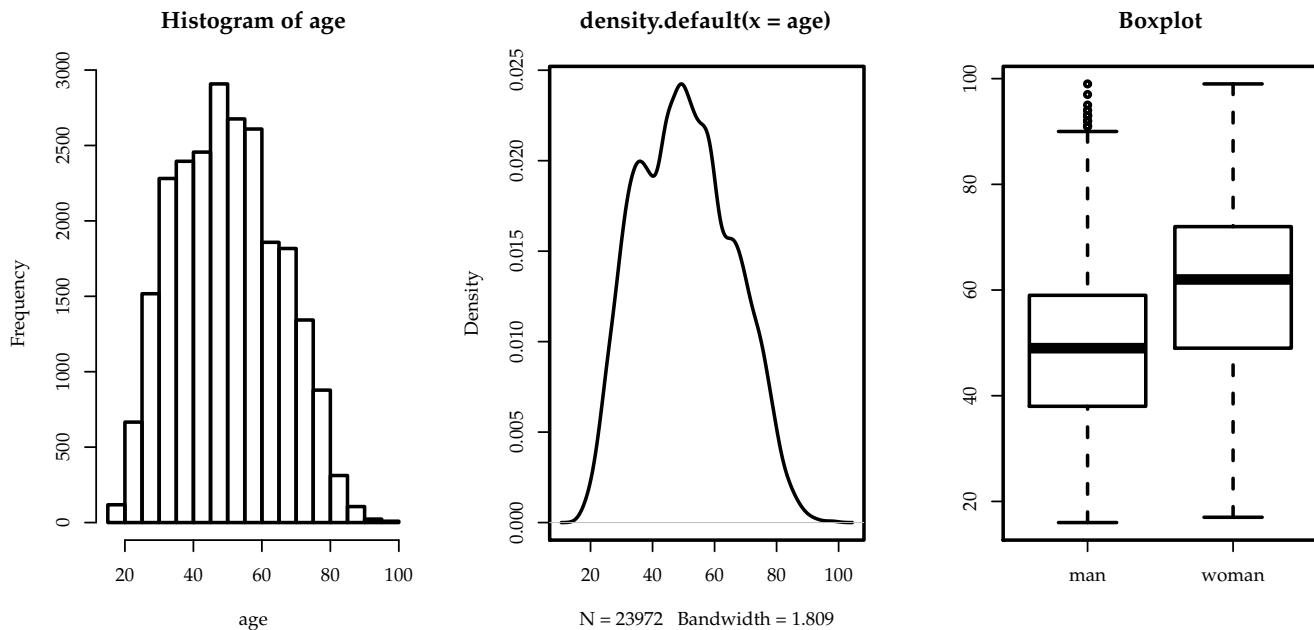
5.6 Graphs

There is more than one way to represent numbers as graphs.

5.7 Basic Graphs

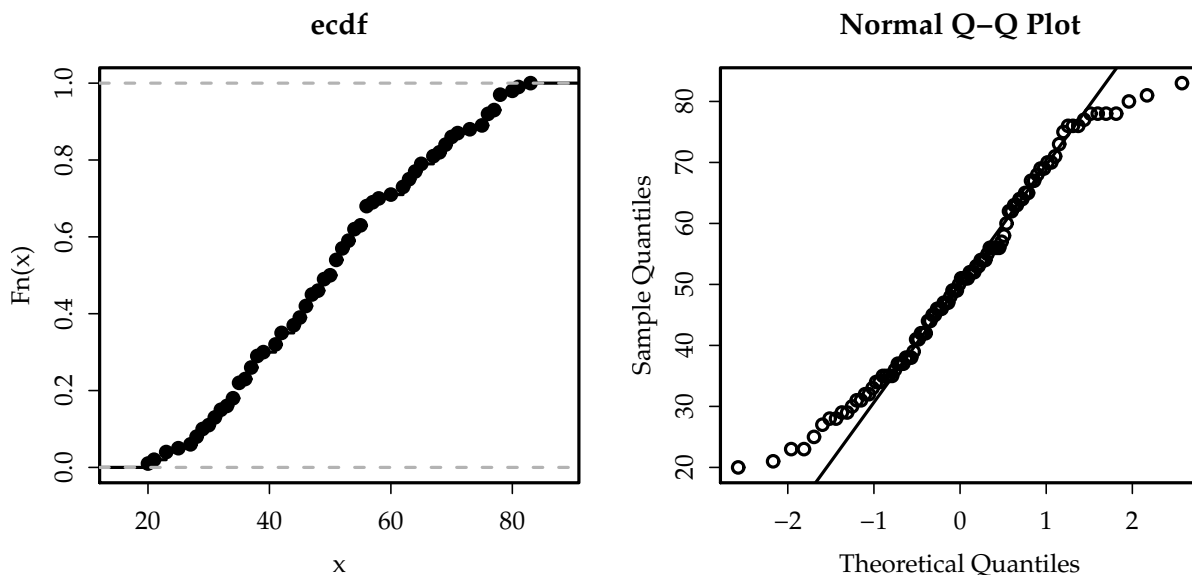
Here are three basic graphs:

```
with(BudgetFood, {  
  hist(age)  
  plot(density(age))  
  boxplot(age ~ sex, main = "Boxplot")  
})
```



Two further helpful plots are `ecdf` and `qqnorm`:

```
x <- sample(BudgetFood$age, 100)
plot(ecdf(x), main = "ecdf")      qqnorm(x)
                                  qqline(x)
```



- Sometimes it is obvious how to prepare our data for these functions. Sometimes it is more complicated. Then other commands help and calculate an object that can be plotted (with `plot`)

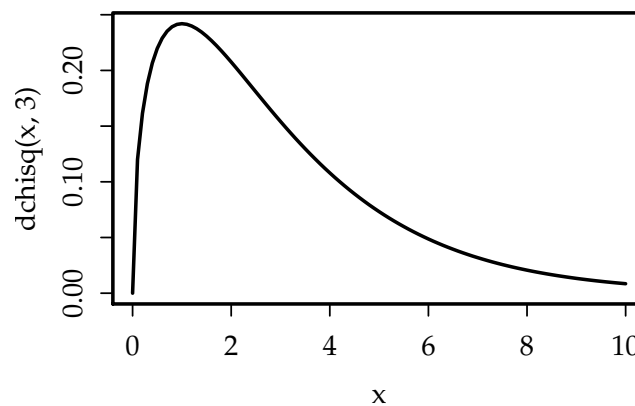
– `density, ecdf, xyplot...`

- Some commands then plot whatever we have prepared:
 - `plot`, `hist`, `boxplot`, `barplot`, `curve`, `mosaicplot`, ...
- Yet other commands add something to an existing plot:
 - `points`, `text`, `lines`, `abline`, `qqline`...

5.7.1 Plotting functions

We can plot functions of x with `curve`.

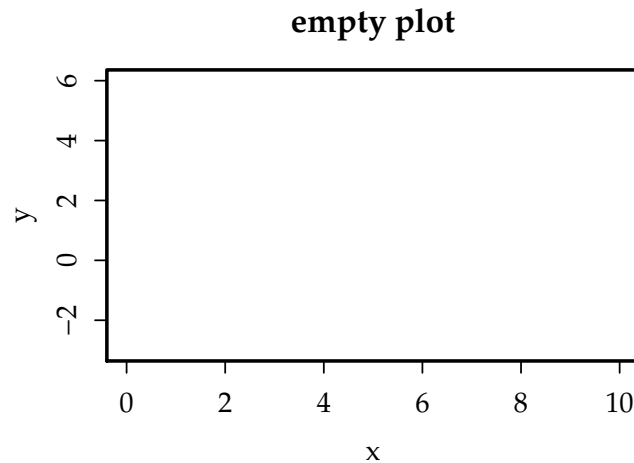
```
| curve(dchisq(x, 3), from = 0, to = 10)
```



5.7.2 Empty plots

Sometimes it is helpful to start with an empty plot. Then we have to help `plot` a little bit. Usually, `plot` can guess from the data the limits and labels of the axes. With an empty plot we have to specify them explicitly.

```
| plot(NULL, xlim = c(0, 10), ylim = c(-3, 6), xlab = "x",  
      ylab = "y", main = "empty plot")
```

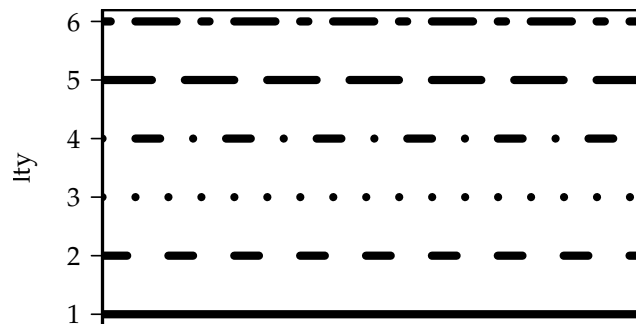


5.7.3 Line type

Almost all commands that draw lines follow the following conventions:

- `lty` linetype ("dashed", "dotted", or simply a number)

```
plot(NULL, ylim = c(1, 6), xlim = c(0, 1), xaxt = "n",
     ylab = "lty", las = 1)
sapply(1:6, function(lty) abline(h = lty, lty = lty,
                               lwd = 5))
```



- `lwd` linewidth (a number)
- `col` colour ("red", "green", `gray(0.5)`)

5.7.4 Points

The character used to draw points is determined with `pch`.

```
range = 1:20
plot(range, range/range, pch = range, frame = FALSE)
text(range, range/range + 0.2, range)
```

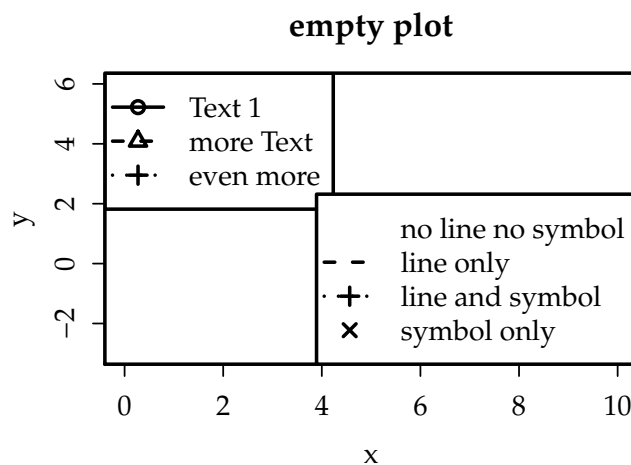
| | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| ○ | △ | + | × | ◇ | ▽ | ■ | * | ◆ | ⊕ | ⊗ | ⊞ | ⊠ | ⊡ | ■ | ● | ▲ | ◆ | ● | ● |

5.7.5 Legends

When we use more than one line or more than one symbol in our plot we have to explain their meaning. This is done in a legend.

Usually legend gets as an option a vector of linetypes `lty` and symbols `pch`. They will be used to construct example lines and symbols next to the actual text of the legend. If the `lty` or `pch` is `NA`, then no line or point is drawn.

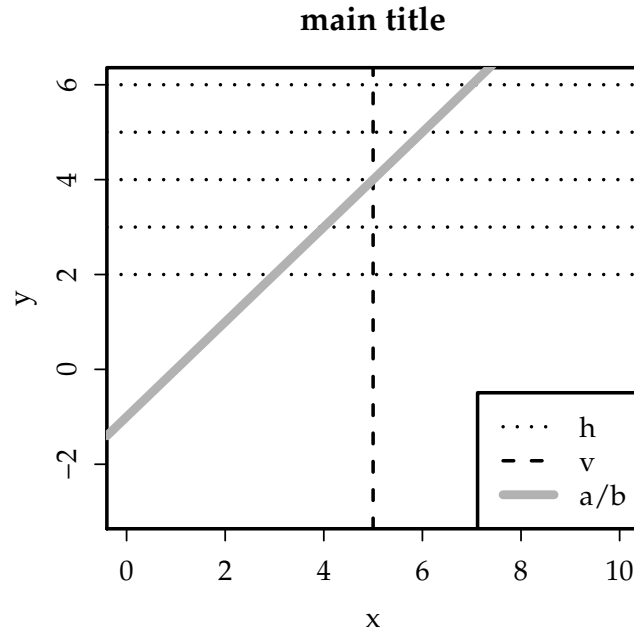
```
plot(NULL, xlim = c(0, 10), ylim = c(-3, 6), xlab = "x",
      ylab = "y", main = "empty plot")
legend("topleft", c("Text 1", "more Text", "even more"),
      lty = 1:3, pch = 1:3)
legend("bottomright", c("no line no symbol", "line only",
                        "line and symbol", "symbol only"), lty = c(NA, 2,
                        3, NA), pch = c(NA, NA, 3, 4), bg = "white")
```



5.7.6 Auxiliary lines

The command `abline` allows us to add auxiliary lines to a plot.

```
plot(NULL, xlim = c(0, 10), ylim = c(-3, 6), xlab = "x",
      ylab = "y", main = "main title")
abline(h = 2:6, lty = "dotted")
abline(v = 5, lty = "dashed")
abline(a = -1, b = 1, lwd = 5, col = grey(0.7))
legend("bottomright", c("h", "v", "a/b"), lty = c("dotted",
  "dashed", "solid"), col = c("black", "black", grey(0.7)),
  lwd = c(2, 2, 5))
```



`abline` knows the following important parameters:

- `h=` for horizontal lines
- `v=` for vertical lines
- `a=...`, `b=...` for lines with intercept `a` and slope `b`

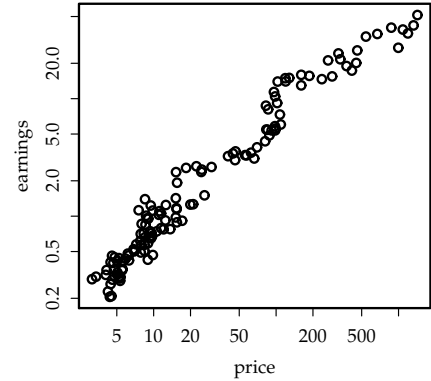
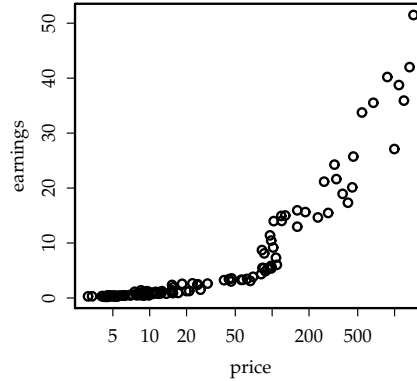
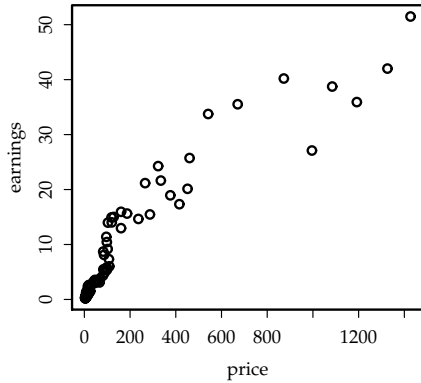
Note, that these arguments can be vectors if we want to draw several lines at the same time.

5.7.7 Axes

The options `log='x'`, `log='y'` or `log='xy'` determine whether which axis is shown in a logarithmic style.

```
data(PE, package = "Ecdat")
xx <- as.data.frame(PE)
attach(xx)
```

```
| plot(price, earnings) | plot(price, earnings, log="x") | plot(price, earnings, log="xy")
```

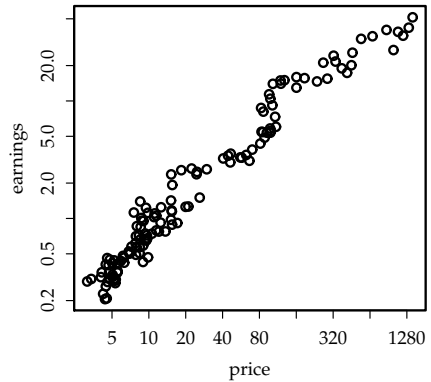
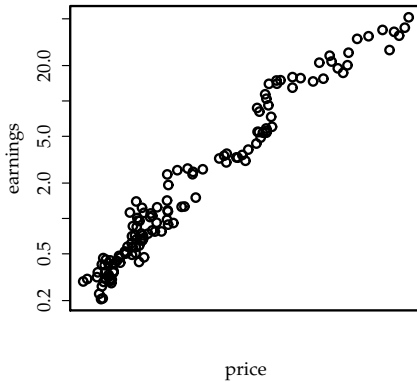
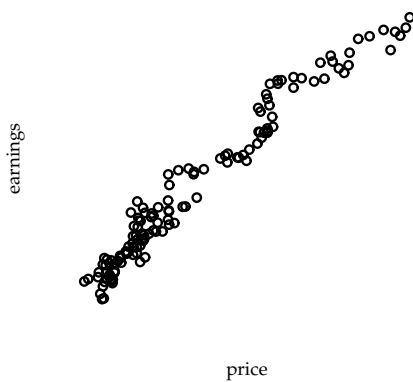


To gain more flexibility `axis` can draw a wide range of axes. Before using `axis` the previous axes can be removed entirely (`axes=FALSE`) or suppressed selectively (`xaxt="n"` or `yaxt="n"`).

```
| plot(price, earnings, log="xy", axes=FALSE)
```

```
| plot(price, earnings, log="xy", yaxt="n")
```

```
| plot(price, earnings, log="xy", yaxt="n")
axis(1, at=c(5, 10, 20, 40, 80, 160, 320, 640, 1280))
```



If we specify a lot of axes labels, as in the example above, `R` does not print them all if they overlap.

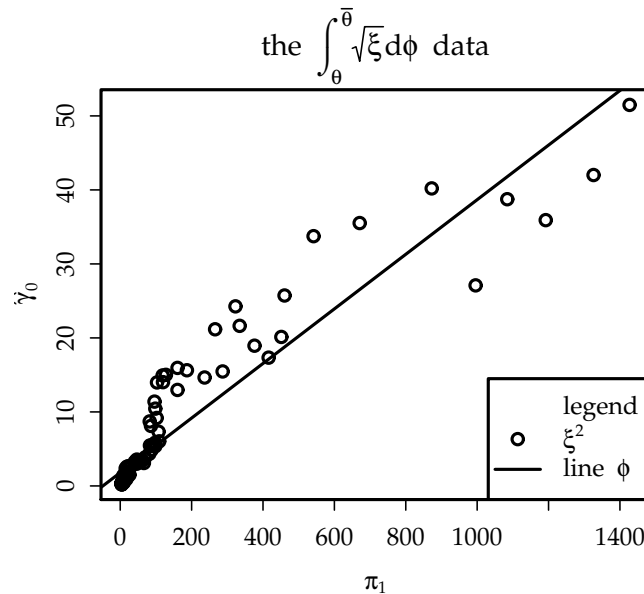
5.8 Fancy math

`R` can also render more than only textual labels with `plotmath`:

```

plot(price, earnings, xlab = expression(pi[1]), ylab = expression(hat(gamma)[0]),
     main = expression(plain(the) ~ ~integral(sqrt(xi) *
       d * phi, theta, bar(theta)) ~ ~plain(data))
abline(lm(earnings ~ price))
legend("bottomright", c("legend", expression(xi^2), expression(plain(line) ~
  ~phi)), pch = c(NA, 1, NA), lty = c(NA, NA, 1))

```



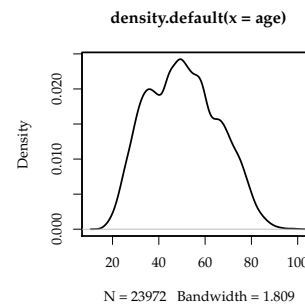
5.8.1 Several diagrams

Diagrams side by side To put several diagrams on one plot side by side we can call `par(mfrow=c(...))` or `layout` or `split.screen`.

```

par(mfrow = c(1, 2))
with(BudgetFood, {
  hist(age)
  plot(density(age))
})

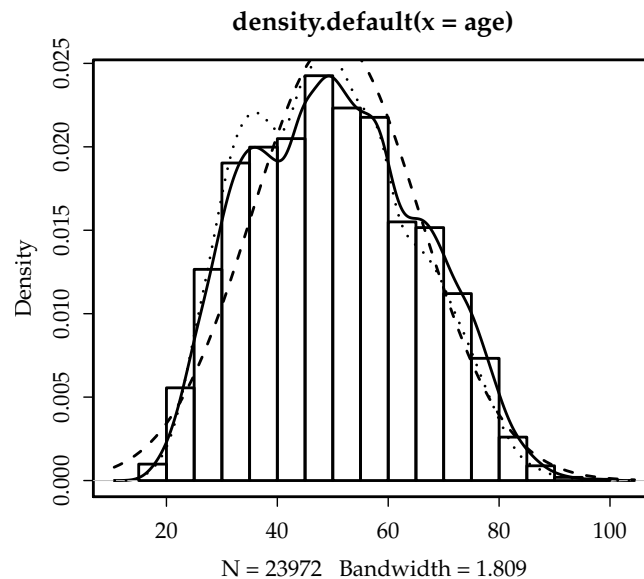
```



Superimposed graphs

- Anything that can create lines or points (like `density` or `ecdf`) can immediately be added to an existing plot.
- Plot-objects that would otherwise create a new figure (like `plot`, `hist`, or `curve`) can be added to an existing plot with the optional parameter `add=TRUE`.

```
with(BudgetFood, {
  plot(density(age), lwd = 2)
  lines(density(age[sex == "man"], na.rm = TRUE), lty = 3,
        lwd = 2)
  hist(age, freq = FALSE, add = TRUE)
  curve(dnorm(x, mean(age), sd(age)), add = TRUE, lty = 2)
})
```



Coplots We will discuss coplots in section 6.

5.9 Tables

Tables of frequencies The command `table` calculates a table of frequencies. Here we show only the first 16 columns:

```
| with(BudgetFood, table(sex, age))[, 1:16]
```

| | age | | | | | | | | | | | | | | | |
|-------|-----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| sex | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 |
| man | 3 | 6 | 21 | 21 | 36 | 37 | 87 | 100 | 132 | 201 | 210 | 248 | 254 | 329 | 367 | 363 |
| woman | 0 | 2 | 7 | 9 | 12 | 21 | 19 | 21 | 22 | 26 | 18 | 28 | 10 | 25 | 28 | 12 |

Other statistics The command `aggregate` groups our data by levels of one or several factors and applies a function to each group. In the following example the factor is `sex`, the function is the mean which is applied to the variable `age`.

```
| with(BudgetFood, aggregate(age ~ sex, FUN = mean))
```

| | sex | age |
|---|-------|----------|
| 1 | man | 49.08985 |
| 2 | woman | 59.47445 |

5.10 Regressions

Simple regressions can be estimated with `lm`. The operator `~` allows us to describe the regression equation. The dependent variable is written on the left side of `~`, the independent variables are written on the right side of `~`.

```
| lm(wfood ~ totexp, data = BudgetFood)
```

```
Call:
lm(formula = wfood ~ totexp, data = BudgetFood)

Coefficients:
(Intercept)          totexp
 0.4950397225  -0.0000001348
```

The result is a bit terse. More details are shown with the command `summary`.

```
| summary(lm(wfood ~ totexp, data = BudgetFood))
```

```
Call:
lm(formula = wfood ~ totexp, data = BudgetFood)

Residuals:
    Min       1Q   Median       3Q      Max
-0.49307 -0.09374 -0.01002  0.08617  1.06182
```

```

Coefficients:
              Estimate      Std. Error t value Pr(>|t|)
(Intercept)  0.495039722500  0.001561819134  316.96  <2e-16 ***
totexp      -0.000000134849  0.000000001459  -92.41  <2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.1422 on 23970 degrees of freedom
Multiple R-squared:  0.2627,      Adjusted R-squared:  0.2626
F-statistic:  8540 on 1 and 23970 DF,  p-value: < 2.2e-16

```

5.11 Starting and stopping R

Whenever we start R, the program attempts to find a file `.Rprofile`, first in the current working directory, then in the home directory. If the file is found, it is “sourced”, i.e. all R commands in this file are executed. This is useful when we want to run the same commands whenever we start R. The following line

```
| options(browser = "/usr/bin/chromium-browser")
```

in `.Rprofile` makes sure that the help system of R always uses chromium.

Also when we quit R with the command `q()`, the application tries to make our life easier.

```
| q()
```

R first asks us

```
Save workspace image? [y/n/c]:
```

Here we have the possibility to save all the data that we currently use (and that are in our workspace) in a file `.Rdata` in the current working directory. When we start R for the next time (from this directory) R automatically reads this file and we can continue our work.

6 Lattice

6.1 Multiway xyplots

Sometimes we want to display one type of diagram separately for different levels of a factor. Here is an example: We want to show how the investment

share (`ci`) develops over time (`year`). This should be done for each `country` separately. We first get a subset of the data (six largest countries, later than 2001) from the Penn World Table:

```
library(pwt)
library(lattice)
lattice.options(default.args = list(as.table = TRUE))
data(pwt6.3)
xx <- with(pwt6.3, aggregate(pop, list(country = country),
  mean))
xx <- subset(xx, country != "China Version 2")
N <- 6
xx <- subset(xx, x >= -sort(-xx$x)[N])
xx <- merge(xx, pwt6.3)
xx <- subset(xx, year > 2001)
```

We also give two countries a shorter name:

```
levels(xx$country)[grep("United States", levels(xx$country))] <- "U.S.A."
levels(xx$country)[grep("China", levels(xx$country))] <- "China"
```

We reorder the countries. The order of the factor is used later in the plots. Here we order according to the median of `ci`:

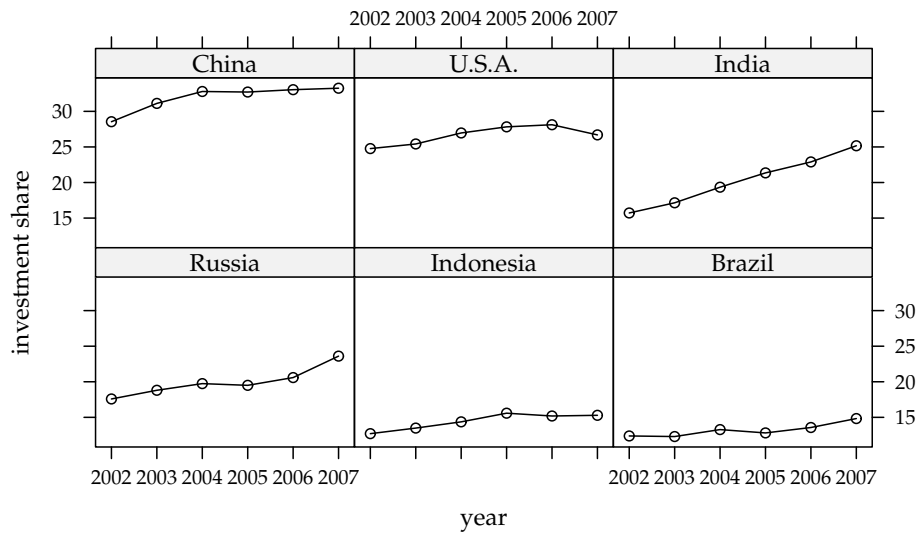
```
xx <- within(xx, country <- reorder(factor(xx$country),
  xx$ci, function(x) -median(x, na.rm = TRUE)))
```

Sorting the data along `year` and `country` makes it easier to draw connect lines later on:

```
xx <- xx[with(xx, order(country, year)), ]
```

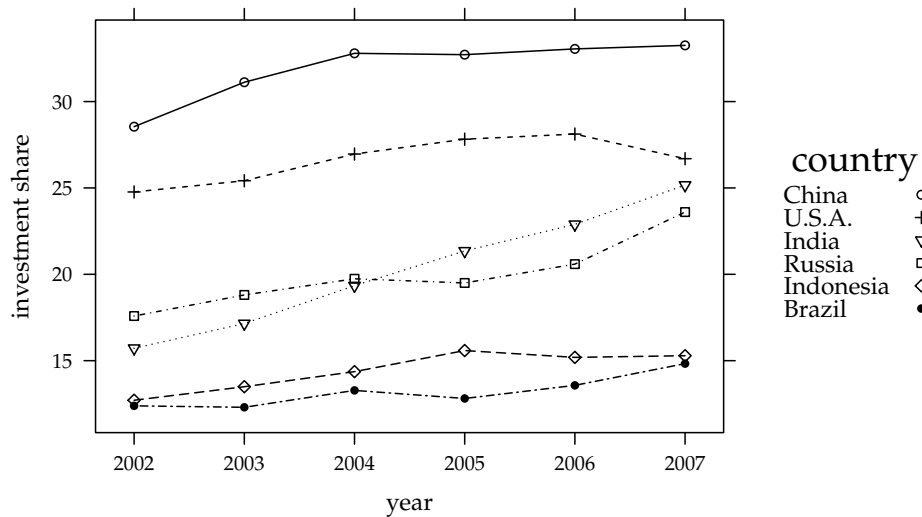
Now let us create a graph with one panel for each country:

```
plot(xyplot(ci ~ year | as.factor(country), data = xx,
  ylab = "investment share", type = "b"))
```



We could also put all lines in one graph:

```
plot(xyplot(ci ~ year, group = country, data = xx, ylab = "investment share",
  type = "b", auto.key = list(space = "right", title = "country")))
```



6.2 Syntax

The data we want to display in our lattice is described with the help of a formula:

Graphs with variables on the vertical and horizontal axis:

- `vertical ~ horizontal` creates only one graph
- `vertical ~ horizontal | conditioning variable` creates for each level of the conditioning variable one panel with one graph.
- `vertical ~ horizontal, group=grouping variable` creates only one panel and superimposes within this panel graphs for each level of the grouping variable.
- `vertical ~ horizontal | conditioning variable , group=grouping variable` creates for each level of the conditioning variable one panel. Within these panels graphs for each level of the grouping variable are superimposed.

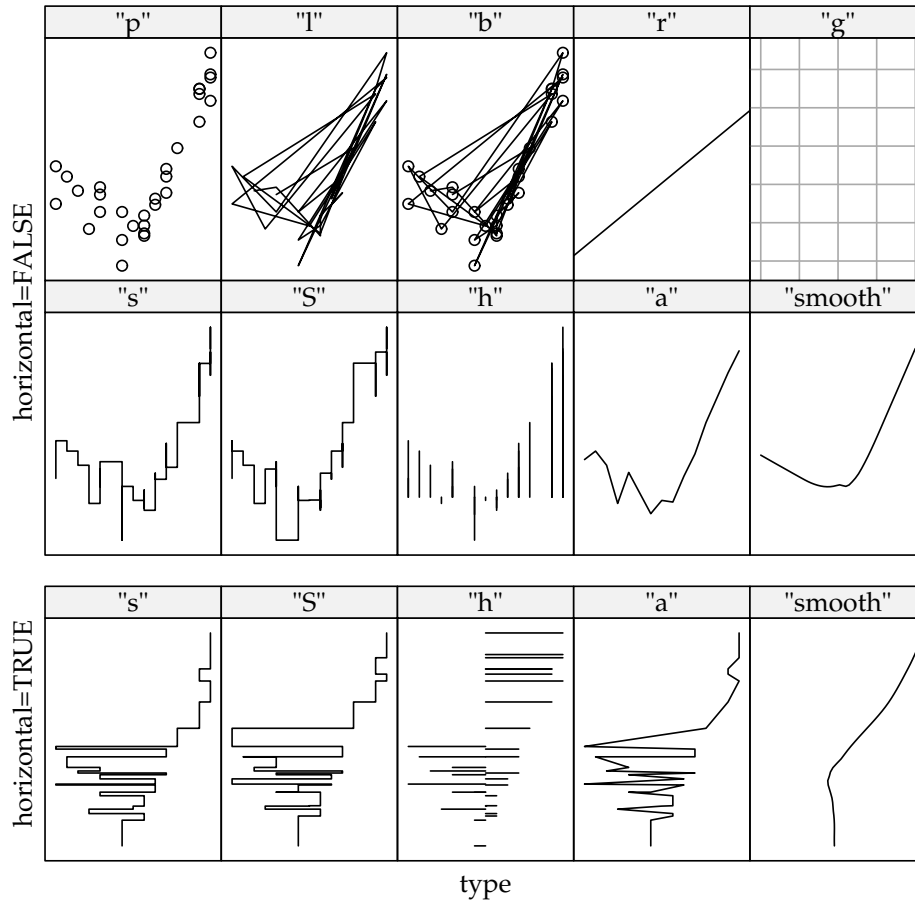
Graphs with variables only on the horizontal axis (examples would be density plots, histograms, etc.):

- `~ horizontal` creates only one graph
- `~ horizontal | conditioning variable` creates for each level of the conditioning variable one panel with one graph.
- `~ horizontal, group=grouping variable` creates only one panel and superimposes within this panel graphs for each level of the grouping variable.
- `~ horizontal | conditioning variable , group=grouping variable` creates for each level of the conditioning variable one panel. Within these panels graphs for each level of the grouping variable are superimposed.

Several variables on the horizontal axis

- `... ~ h1 + h2 ...` shows two variables h1 and h2

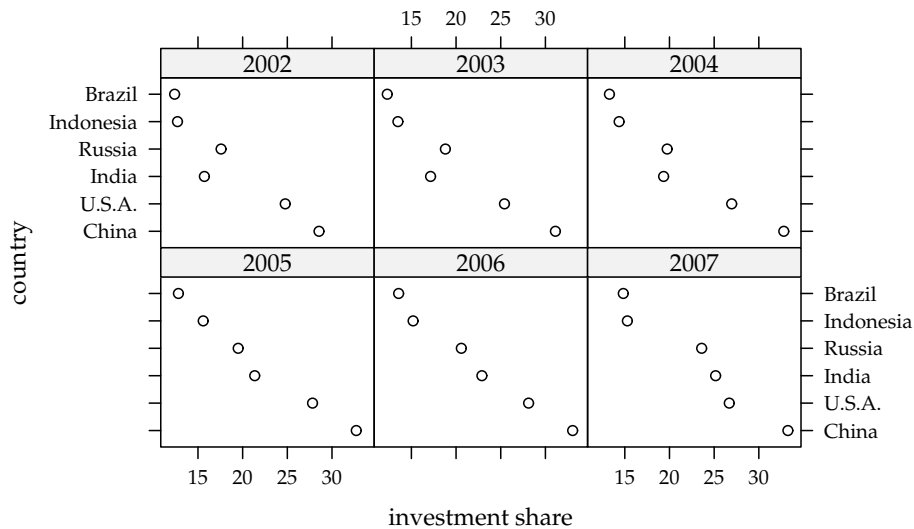
Types of lines The parameter `types` determines how points are displayed:
`type="b"` or `type=c("b", "smooth", "g")`



6.3 Multiway continued

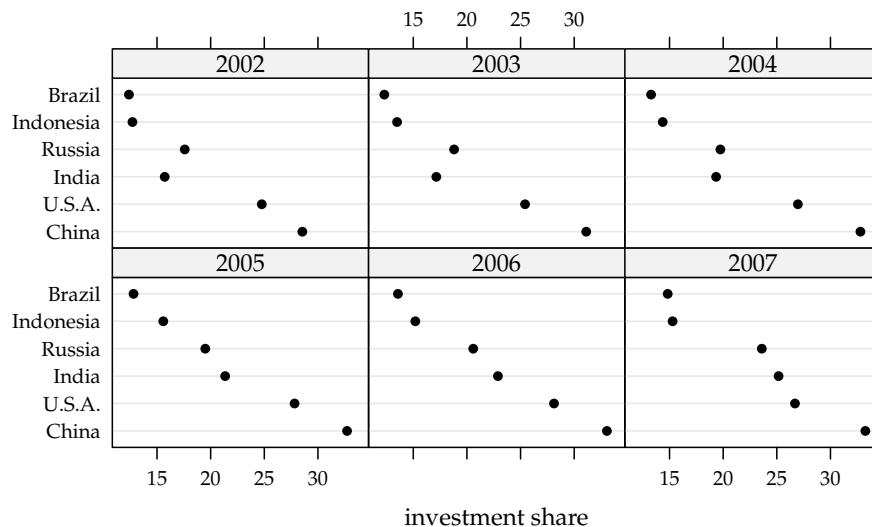
Instead of having different panels for different countries, we could also have different panels for different years:

```
| plot(xyplot(country ~ ci | as.factor(year), data = xx,  
|      xlab = "investment share"))
```



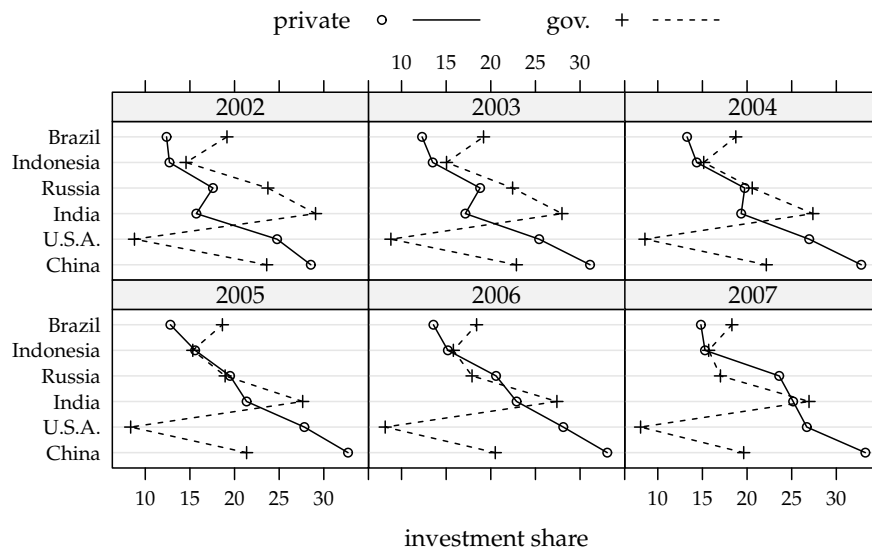
If the vertical variable (`country` in this case) is a factor, then `dotplot` generates even nicer graphs:

```
plot(dotplot(country ~ ci | as.factor(year), data = xx,
            xlab = "investment share", horizontal = TRUE))
```



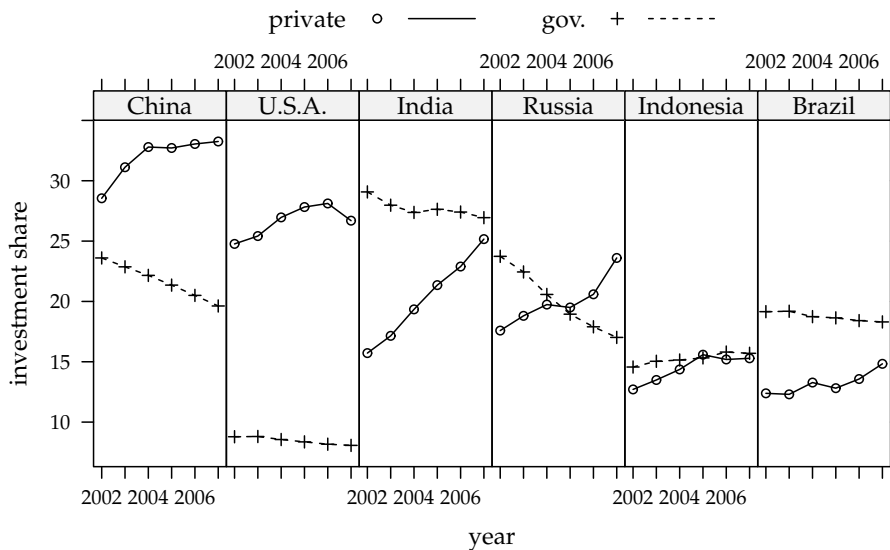
We can, of course, show more than one variable on the horizontal axis:

```
keys <- list(text = c("private", "gov."), space = "top",
            columns = 2, lines = TRUE)
plot(dotplot(country ~ ci + cg | as.factor(year), data = xx,
            xlab = "investment share", horizontal = TRUE, auto.key = keys,
            t = "b"))
```



Certainly, we can also have more than one variable on the vertical axis:

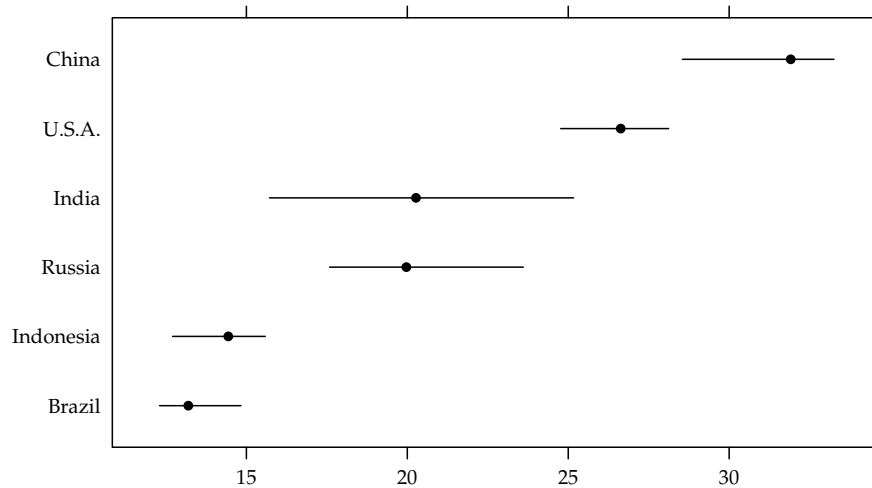
```
plot(xyplot(ci + cg ~ year | as.factor(country), layout = c(6,
  1), data = xx, ylab = "investment share", auto.key = keys,
  t = "b"))
```



Segment plots Sometimes we have to plot segments. Here we plot a range of the minimum investment share to the maximum investment share.

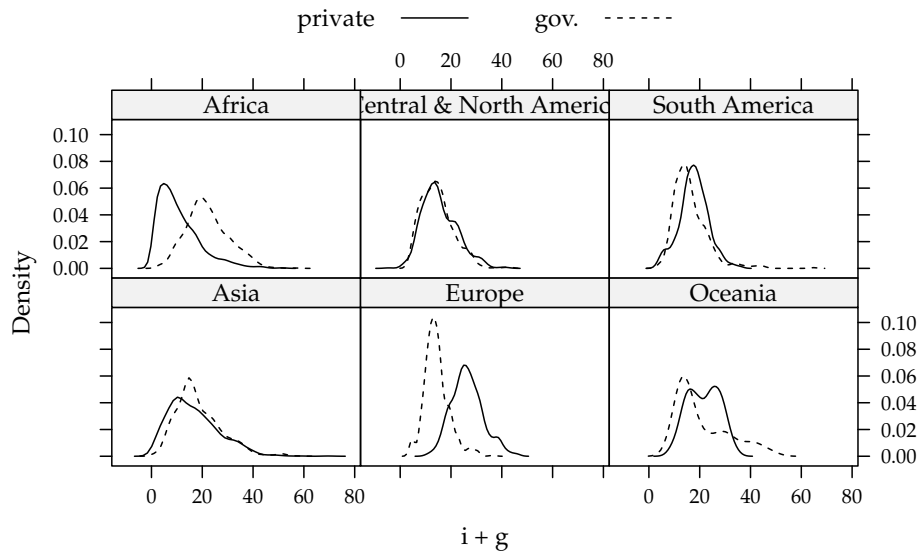
```
library(latticeExtra)
xx2 <- as.data.frame(t(sapply(by(xx, list(xx$country),
  function(x) c(min = min(x$ci), mean = mean(x$ci),
```

```
max = max(x$ci)), c))
xx2 <- within(xx2, {
  country <- as.factor(rownames(xx2))
})
plot(segplot(reorder(factor(country), mean) ~ min + max,
  centers = mean, draw.bands = FALSE, data = xx2))
```



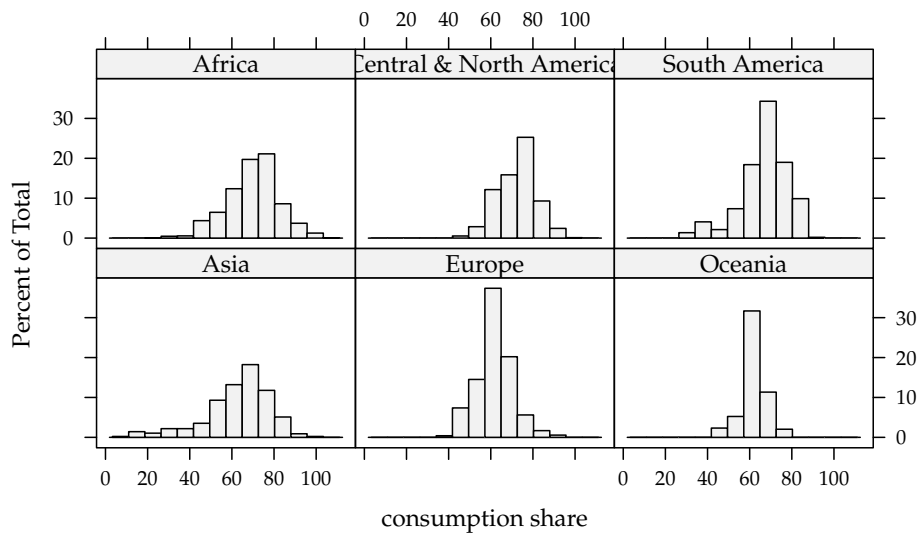
6.4 Densityplots

```
data(pwt5.6)
keys <- list(text = c("private", "gov."), space = "top",
  columns = 2, lines = TRUE)
plot(densityplot(~i + g | continent, data = pwt5.6, plot.points = FALSE,
  auto.key = keys))
```



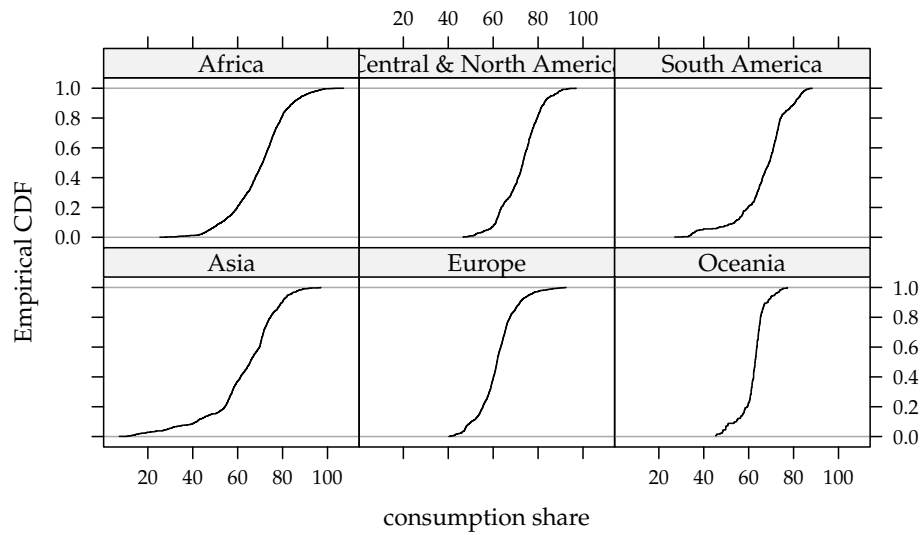
6.5 Histograms

```
plot(histogram(~c | continent, data = pwt5.6, plot.points = FALSE,
              xlab = "consumption share"))
```

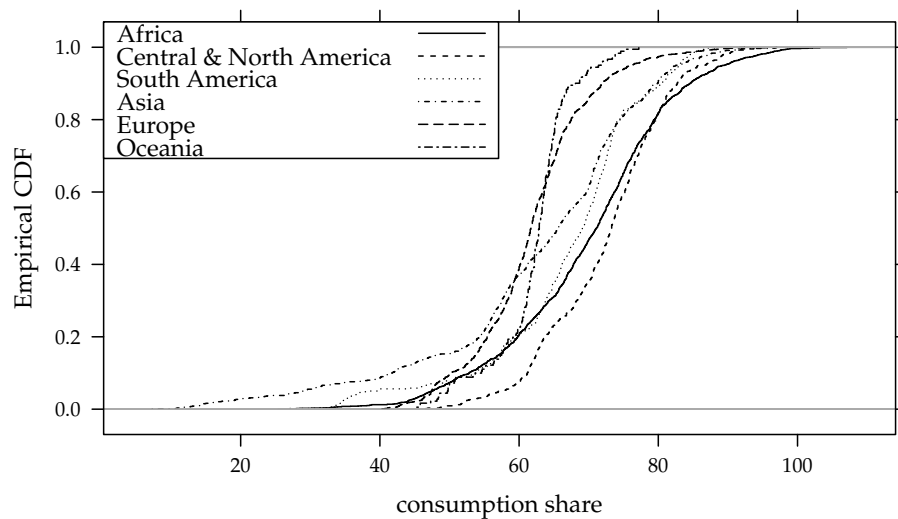


6.6 Empirical cumulative densities

```
library(latticeExtra)
plot(ecdfplot(~c | continent, data = pwt5.6, xlab = "consumption share"))
```

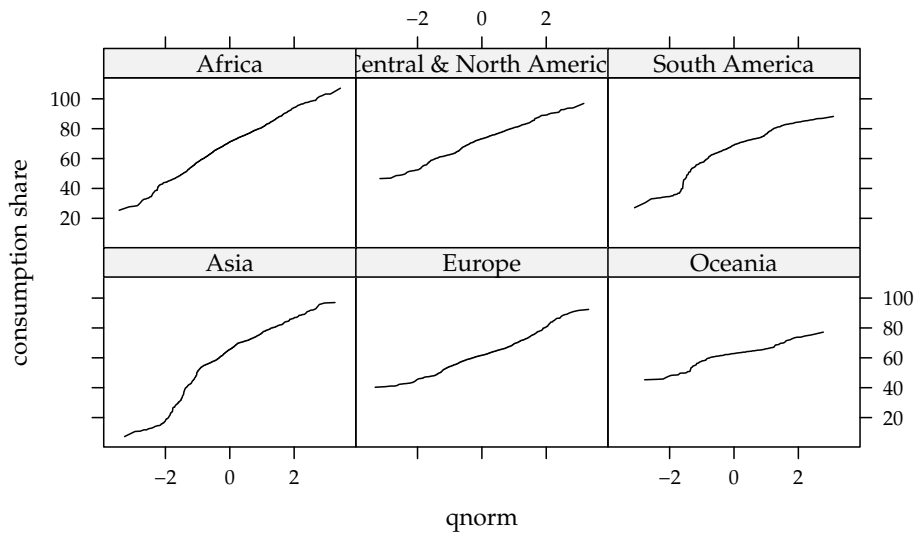


```
plot(ecdfplot(~c, groups = continent, data = pwt5.6,
  auto.key = list(x = 0, y = 1, corner = c(0, 1), background = "white",
  border = TRUE), xlab = "consumption share"))
```

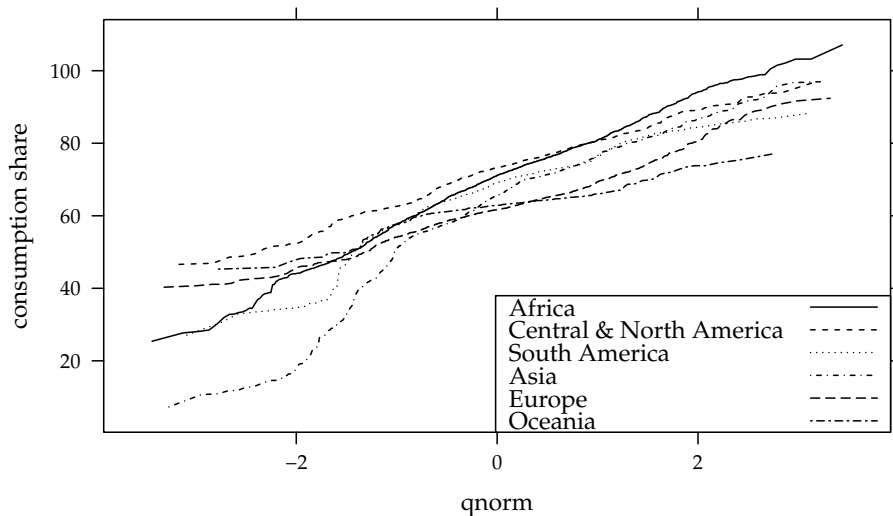


6.7 QQ-plots

```
plot(qqmath(~c | continent, data = pwt5.6, ylab = "consumption share",
  type = "l"))
```

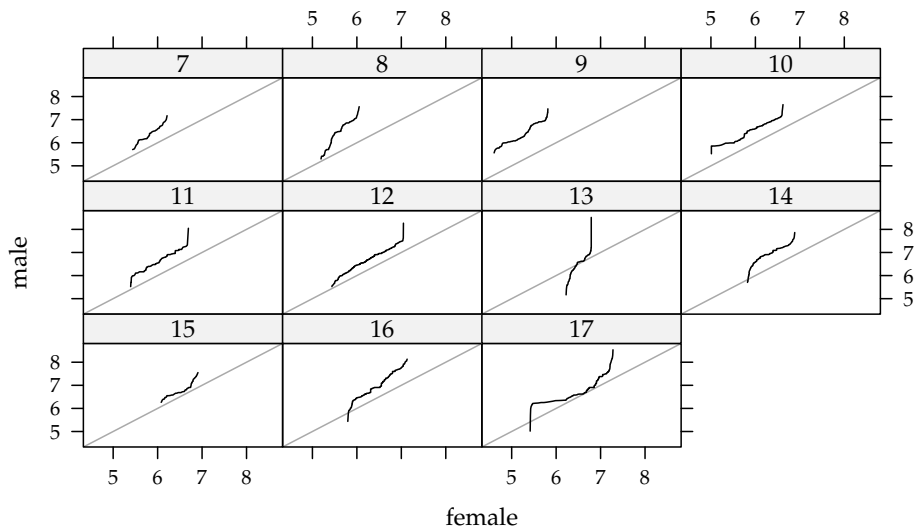


```
plot(qqmath(~c, groups = continent, data = pwt5.6, auto.key = list(x = 1,
  y = 0, corner = c(1, 0), background = "white", border = TRUE,
  points = FALSE, lines = TRUE), ylab = "consumption share",
  type = "l"))
```



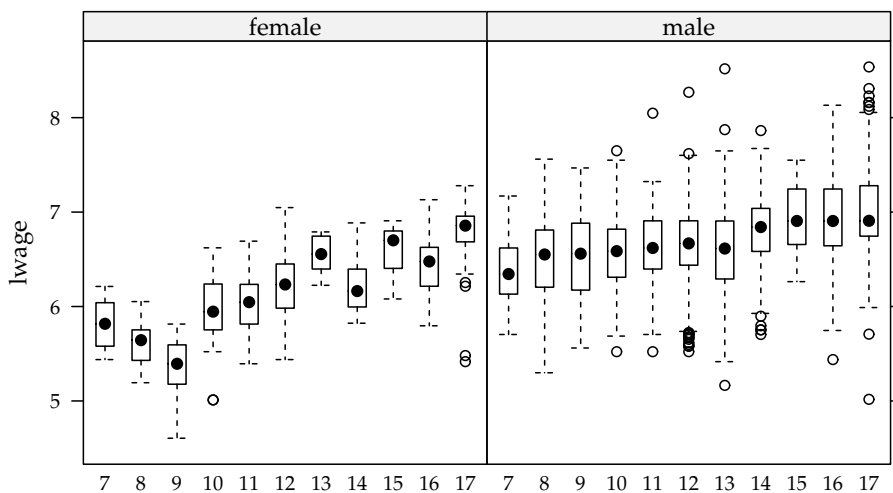
6.8 Sample QQ-plots

```
library(Ecdat)
data(Wages)
plot(qq(sex ~ lwage | as.factor(ed), data = subset(Wages,
  ed >= 7), type = "l"))
```



6.9 Boxplots

```
plot(bwplot(lwage ~ as.factor(ed) | sex, data = subset(Wages,
  ed >= 7)))
```

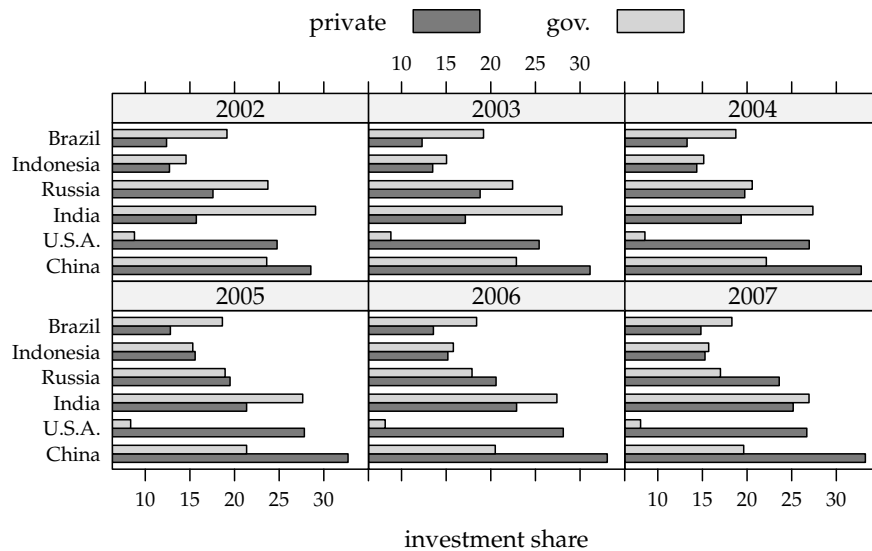


6.10 Barcharts

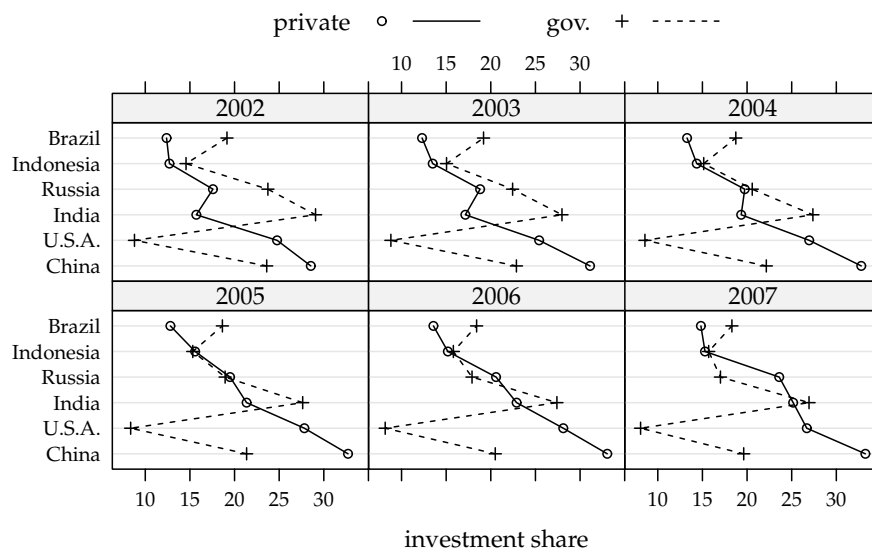
lattice can also do bar charts:

```
keys <- list(text = c("private", "gov."), space = "top",
  columns = 2)
```

```
plot(barchart(country ~ ci + cg | as.factor(year), data = xx,
  xlab = "investment share", horizontal = TRUE, auto.key = keys))
```

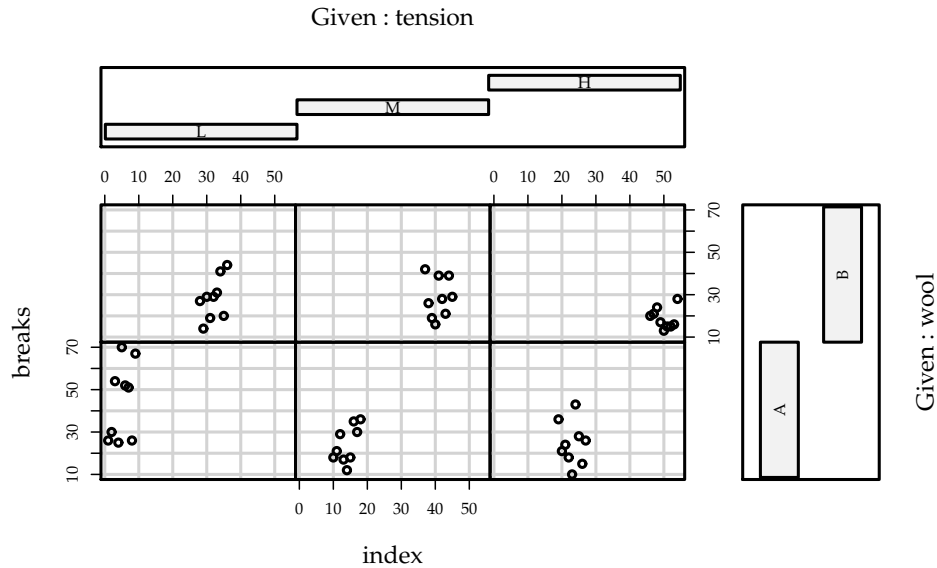


We should note that often a dotplot or xyplot presents the same data in a better way.



6.11 Coplots

```
data(warpbreaks)
coplot(breaks ~ 1:length(breaks) | tension * wool, data = warpbreaks,
  xlab = "index")
```



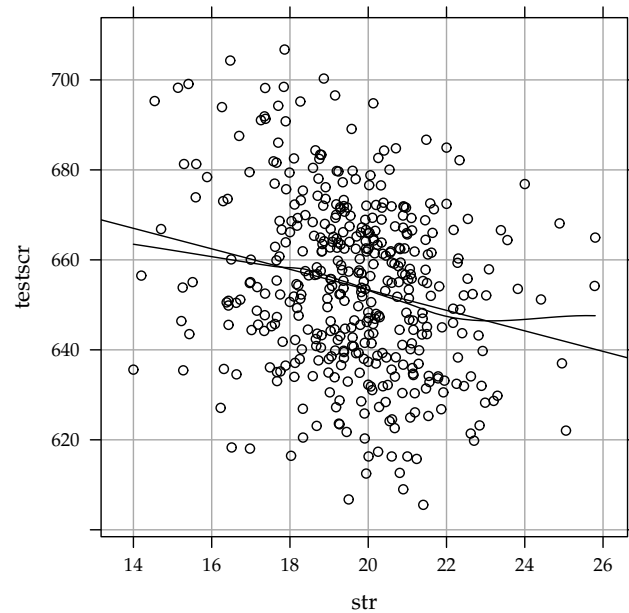
6.12 Parameters

6.12.1 Types

Usually `lattice` renders data as points. The argument `type=(...)` modifies this behaviour. Some useful values are the following:

- `type='p'`: points
- `type='l'`: lines (in the order of the dataset)
- `type='b'`: lines and points
- `type='g'`: a grid
- `type='r'`: a regression line
- `type='smooth'`: a loess smooth line

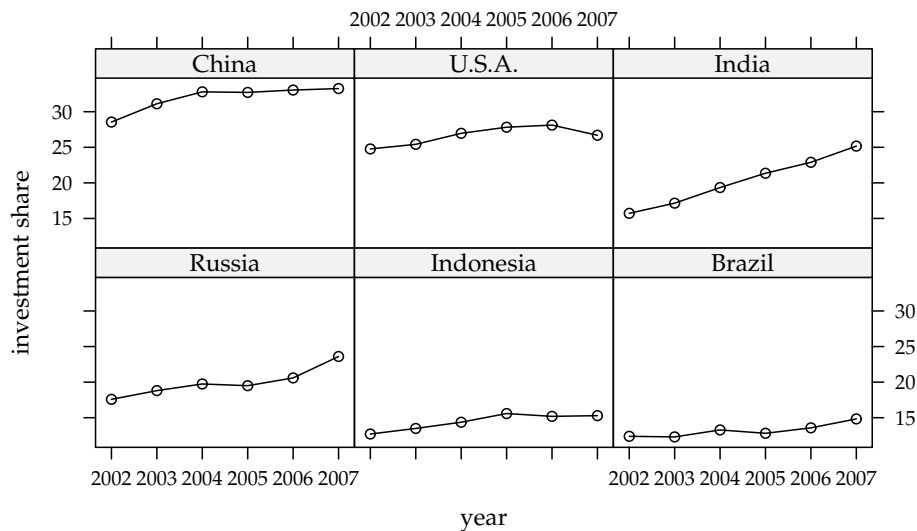
```
data(Caschool, package = "Ecdat")
plot(xyplot(testscr ~ str, data = Caschool, type = c("p",
  "g", "r", "smooth")))
```



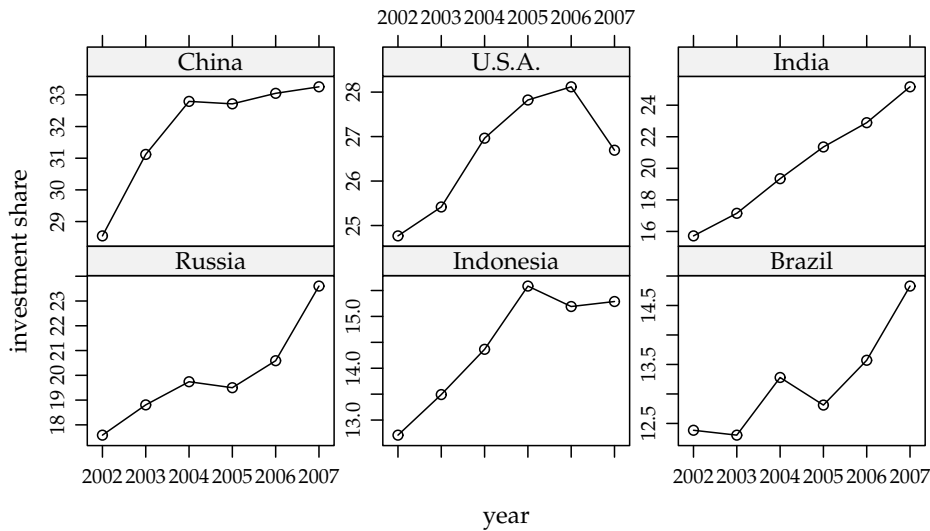
6.12.2 Axes

Different scales for different panels Usually, `lattice` chooses the same scale for all panels in a plot. This can be changed with the help of the parameter `scales`.

```
plot(xyplot(ci ~ year | as.factor(country), data = xx,
           ylab = "investment share", t = "b"))
```

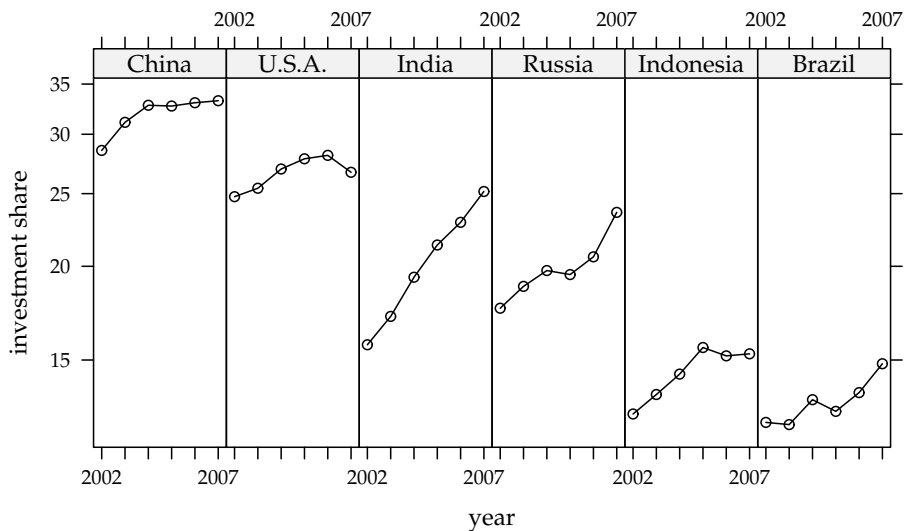


```
plot(xyplot(ci ~ year | as.factor(country), data = xx,
  ylab = "investment share", t = "b", scales = list(x = "same",
  y = "free")))
```



Individual axes

```
myscale <- list(x = list(at = 2002:2007, labels = c(2002,
  "", "", "", "", 2007)), y = list(log = TRUE, at = c(15,
  20, 25, 30, 35)))
plot(xyplot(ci ~ year | as.factor(country), layout = c(6,
  1), scales = myscale, data = xx, ylab = "investment share",
  t = "b"))
```



Themes

```

keys <- list(text = c("consume", "private invest.", "gov."),
             lines = TRUE, space = "top", columns = 3)
mTheme <- custom.theme(symbol = brewer.pal(3, "Set1"),
                       bg = "grey90", fg = "black", pch = 16, lty = 1:3,
                       lwd = 3)
plot(xyplot(cc + ci + cg ~ year | as.factor(country),
           layout = c(6, 1), data = xx, ylab = "", t = "b",
           par.settings = mTheme, auto.key = keys))

```

