```
##
##  Descriptive Statistics with R
##
##  Author:   Stefan Evert
##  Modified: Sun May  8 17:44:41 2005 (evert)
##

## How to use this tutorial:
##   Lines beginning with a # character are comments, and should be read. :o)
##   All other text lines are R commands. Use cut & paste to copy them into the
##   R command window. If you are running R from Emacs/ESS, first start an R
##   process (M-x R RET). Then you can execute the current line in this file
##   by pressing C-c C-n. (Note to Emacs novices: C-c means "press the 'c' key
##   while holding down the Ctrl key"; M-x refers to the "meta" key, which is
##   often mapped to the left Alt key.)


#########################################################################
##
##  Working with tables of observations (data frames)
##

## We will use data from a small non-representative survey conducted at the
## IMS Stuttgart in spring 2001. The table of observations is stored in the
## file "people.tbl" in the form of nicely formatted ASCII text.

## The best way of editing data tables is to load them into a spreadsheet
## program (such as MS Excel or the OpenOffice.org spreadsheet component).
## In order for this to work reliably, it is better to use a format where
## columns are separated by a single character (e.g. "," or TAB) rather
## than lined up with blanks (see "people.csv").

## Remember that there are also a number of built-in data sets available,
## which are useful for experimentation with R's plotting and data manipulation
## functions.
data()
?mtcars
data(mtcars)

## load table of observations into data frame
people <- read.table("people.tbl")

## read.table() has many useful options. If row names are missing from the .tbl
## file, ``header=T'' tells R to interpret the first line as a header (giving
## the names of variables, i.e. columns). If the observations are stored in a
## TAB-separated file (as it is typically used with Perl scripts), use ``sep="\t"''.
## These two settings are the defaults of the read.delim() function. Similarly,
## read.csv() has appropriate defaults for reading data in the comma-separated
## format used by many relational databases and spreadsheet programs.
?read.table

## number of columns == number of observed variables
ncol(people)

## number of rows == number of observations
nrow(people)

## list observed variables
colnames(people)

## anonymised names of the subjects (or labels for the observations)
rownames(people)

## display table of observation, adjusting screen width (characters per line)
```

```
## for wide terminal or emacs windows
print(people)
options(width=105)
print(people)

## view / edit data (caution: this will modify the data frame; no undo available!)
fix(people)

## it's safer to store the modified data frame in a different variable
p2 <- edit(people)

## for viewing the table, use edit() and "hide" the returned data frame
invisible(edit(people))

## show individual observations (by row number of subject name)
people[1:5,]
people[c("P10", "P14", "P15", "P22"),]

## observations for a single variable form a vector (numeric or factor)
people$groesse
people$schuh
people$farbe

## combine vectors of identical length into new data set
p <- data.frame(gr=people$groesse, schuh=people$schuh, sex=people$sex)
print(p[1:10,])

## if all vectors are variables from a single data set, we can use the
## following command to preserve row names
p <- people[, c("groesse", "schuh", "sex")]
print(p[1:10,])

## subset data frame based on observed variables
small.people <- p[(p$gr < 170), ]
print(small.people)

## change row names (useful trick for anonymisation)
rownames(small.people) <- paste("S", 1:nrow(small.people), sep="")
print(small.people)

## write data frame to disk file without quoting strings
## (caution: must set ``quote=TRUE'' if data set contains strings with blanks)
write.table(small.people, file="small-people.tbl", quote=FALSE)

## see help page for further options
?write.table

## manipulating data: age is only shown as two-digit year of birth in the survey
## first compute real year of birth (add 1900 to two-digit versions), then subtract
## from current year to approximate age of subjects
people$jahr <- 1900 + people$jahr
people$alter <- 2005 - people$jahr
print(people[, c("jahr", "alter")])

######################################################################
##
##  Numerical data (interval or ratio scale)
##  Summary statistics: mean, variance, median, quantiles
##

## For numerical data (on an interval or ratio scale), the classical characteristics
## are mean and variance (or s.d.). Sometimes the more robust median value is also given.

## central tendency: the mean
```

```
x <- people$groesse
mu <- mean(x)
print(mu)

## measuring variability:
## average deviation from mean is always zero (almost ...)
n <- length(x)
sum(x - mu) / n
## need to sum absolute values of deviation
sum(abs(x - mu)) / n
## mean squared error = variance is (mathematically) more manageable
sigma2 <- sum((x - mu)^2) / n
print(sigma2)

## sample estimate: empirical variance (unbiased estimator)
## (for a reasonably large population, the difference between the two
##  equations is negligible)
s2 <- var(x)
print(s2)
sum((x - mu)^2) / (n-1)

## standard deviation: square root of variance (scales linearly)
sigma <- sqrt(sigma2)
print(sigma)
s <- sd(x)
print(s)
sqrt(s2)

## another example: shoe size
mean(people$schuh)
sd(people$schuh)

## the median M is a more robust measure if there are outliers (extreme values) in the
data
## (M is chosen such that exactly half of all values in x are < M)
median(x)

## theoretical example using random numbers (imagine values e.g. as yearly income)
income <- round(1000 * rlnorm(100, mean=3))
mean(income)
sd(income)
median(income)

## quantiles are a generalisation of the median:
## - the p-th percentile is the value Q such that p% of all values are < Q
## - first quartile = 25% percentile, second quartile = median, third quartile = 75%
percentile
quantile(x, .25)                        # first quartile
quantile(x, .5)                         # median
quantile(x, .75)                        # third quartile
quantile(x, 0:10 / 10)                  # 0%, 10%, ... , 100% quantiles

## summary for a numerical vectors lists the quartiles (including min and max) and the
mean
summary(x)
summary(people$schuh)

## quantiles (or summary) give better impression of a skewed distribution
quantile(income, 0:10 / 10)
summary(income)

## for a small data set, values can also be printed in sorted order
## (-> empirical distribution function)
sort(people$schuh)
```

```
sort(income)

## mean, median, variance etc. cannot be computed for variables with missing values
(NA's)
mean(people$abi)
## use ``na.rm=TRUE'' option to remove missing values before computation:
## this has to be kept in mind when interpreting the results!
mean(people$abi, na.rm=TRUE)
sd(people$abi, na.rm=TRUE)
median(people$abi, na.rm=TRUE)
## summary() automatically removes (and reports) missing values
summary(people$abi)

## for further computations with numerical variables, missing values must be removed
explicitly
## test whether variable contains missing values (shows number of NA's)
sum(is.na(people$groesse))
sum(is.na(people$abi))
## remove NA's with na.omit() function
abi.no.na <- na.omit(people$abi)

## note: we can also show a summary for the entire table of observations
summary(people)

######################################################################
##
##  Graphical presentation of numerical data
##

## plots for numerical data are usually meaningless (unless observations are ordered)
barplot(people$schuh, ylab="Schuhgroesse", names.arg=rownames(people))

## discrete variables often have duplicate values:
## grouping values yields frequency table with barplot-like histogram display
t <- table(people$schuh)
print(t)
plot(t)

## warning: bar plots may be misleading and should not be used
barplot(table(people$schuh))

## choose reasonable scale on y-axis, especially for low frequencies
plot(t, xlab="Schuhgroesse", ylab="Anzahl", ylim=c(0,15))
plot(table(people$alter), xlab="Alter", ylab="Anzahl", ylim=c(0,15))

## continuous variable will usually have frequency == 1 -> rug plot
plot(table(people$groesse), xlab="Groesse", ylab="Anzahl", ylim=c(0,15))

## the distribution is characterised by the density of lines in the rug plot,
## but this is difficult for the human eye to pick out;
## decimal places seem irrelevant for body heights -> group rounded values
t <- table(round(people$groesse))
print(t)
plot(t, ylim=c(0,15))

## more general approach: histogram groups values into "buckets"
hist(people$groesse, ylim=c(0,15))

## a good choice of bucket sizes and breakpoints is essential:
## typically 10-15 groups with equidistant breakpoints, but fewer for small data sets

## too few ...
hist(people$groesse, breaks=5, ylim=c(0,15))
## too many ...
```

```
hist(people$groesse, breaks=50, ylim=c(0,15))

## it may be best to specify the exact breakpoints rather than just the number:
## for small counts, the y-axis should always be scaled (as above)
hist(people$groesse, breaks=seq(from=157.5, to=202.5, by=5), ylim=c(0,15))

## add rug plot of the original data, and some other useful options
hist(people$groesse, breaks=seq(from=157.5, to=202.5, by=5), xlab="Groesse", ylim=c
(0,15), col="lightblue", labels=T)
rug(people$groesse)

## to compare the histogram to a theoretical distribution (e.g. normal distribution),
## the y-axis must be scaled to densities rather than absolute frequency counts; in
## this scaling, which is activated with the option "freq=FALSE", the area of each
## bar represents the proportion of data points in the corresponding bucket;
## the density() function computes a smoothed continuous density function, which
## can then be added to the histogram
hist(people$groesse, breaks=seq(from=157.5, to=202.5, by=5), freq=F, xlab="Groesse",
ylim=c(0,0.1), col="lightblue", labels=T)
rug(people$groesse)
lines(density(people$groesse), col="red", lwd=2)

## for large data sets, the estimated density curve approximates the histogram plot
## (constructed example with normal distribution)
x <- rnorm(5000, mean=175, sd=8)
x <- x[(x >= 158) & (x <= 202)]
hist(x, breaks=seq(from=157.5, to=202.5, by=1), freq=F, xlab="Groesse", ylim=c(0,0.1),
col="lightblue")
rug(x)
lines(density(x), col="red", lwd=2)

## add mean and standard deviation to histogram plot
hist(people$groesse, breaks=seq(from=157.5, to=202.5, by=5), freq=F, xlab="Groesse",
ylim=c(0,0.1), col="lightblue", labels=T)
rug(people$groesse)
lines(density(people$groesse), col="red", lwd=2)
abline(v=mu, col="blue", lwd=2)
abline(v=(mu-sigma), col="blue", lwd=1)
abline(v=(mu+sigma), col="blue", lwd=1)

## for a skewed distribution, mean and median are different, the
## median being shifted towards the mode (maximum) of the distribution
income <- income / 1000                 # scaled to units of 1000 Euro
hist(income, breaks=seq(0,max(income)+5,5), xlim=c(0,100), freq=FALSE, xlab="income (in
1000 EUR)")
lines(density(income), col="black", lwd=2)
mu.income <- mean(income)
sigma.income <- sd(income)
abline(v=mu.income, col="blue", lwd=2)
abline(v=(mu.income-sigma.income), col="blue", lwd=1)
abline(v=(mu.income+sigma.income), col="blue", lwd=1)
abline(v=median(income), col="red", lwd=2)

## empirical distribution function: for each threshold Q, show the proportion of values
<= Q
x <- people$groesse
n <- length(x)
qqplot(x, 1:n, xlab="Groesse", ylab="Anzahl", type="s")
## (uses qqplot() with the percentages corresponding to i out of n values on the y-axis)
qqplot(x, 100 * (1:n) / n, xlab="Groesse", ylab="Anteil (%)", type="s")

## visualise quantiles (e.g. the median) in the empirical distribution function
q.50 <- median(x)
lines(c(q.50,q.50,0), c(0,50,50), col="blue", lwd=2)
```

```
q.90 <- quantile(x, .90)
lines(c(q.90,q.90,0), c(0,90,90), col="darkgreen", lwd=2)

## the quantile function smoothes data
p <- 0:100
q <- quantile(x, p/100)
lines(q, p, col="red", lwd=2)

#########################################################################
##
##  Comparing distributions and visualising correlations
##


## scatterplot for comparison of two numerical variables (on the same data set)
plot(people$groesse, people$schuh, xlab="Groesse [cm]", ylab="Schuhgroesse")
## note the automatic scaling, which assumes interval scale;
## if both variables are on ratio scale, both axes should start from 0:
plot(people$groesse, people$schuh, xlab="Groesse [cm]", ylab="Schuhgroesse", xlim=c
(0,205), ylim=c(0,50))

## another example, using the built-in mtcars data set
data(mtcars)
colnames(mtcars)
plot(mtcars$disp, mtcars$mpg, xlab="Displacement", ylab="Miles per Gallon")

## in this case we do not expect a correlation between the variables
plot(people$schuh, people$abi, xlab="Schuhgroesse", ylab="Abiturnote (Mathematik)",
ylim=c(0,15))

## side-by-side boxplots for comparison of values of numerical variable
## on different data sets (or subsets of a data sets)
## first, we must divide the observations into two or more subsets
gr.m <- people$groesse[people$sex == "m"]
gr.w <- people$groesse[people$sex == "w"]
## then draw as many side-by-side boxplots as you like (see "?boxplot.stats" for details)
boxplot(gr.w, gr.m, ylab="Groesse", names=c("Girls", "Boys"))
## some options to spruce up the graphics
boxplot(gr.w, gr.m, ylab="Groesse", names=c("Girls", "Boys"), boxwex=0.5, col=c
("pink","lightblue"))
abline(h=mean(gr.w), lwd=2, col="darkred")
abline(h=mean(gr.m), lwd=2, col="darkblue")

## there's a much easier way to do this ...
gr <- split(people$groesse, people$sex)
print(gr)
boxplot(gr, ylab="Groesse", boxwex=0.5)

## testing a popular prejudice ...
boxplot(split(people$abi, people$sex), ylab="Abiturnote (Mathematik)", ylim=c(0,15),
col=c("lightblue","pink"))

## we can have any number of categories (e.g. 6 categories in the chickwts data set)
data(chickwts)
boxplot(split(chickwts$weight, chickwts$feed), ylab="weight after 6 weeks",
col="lightblue")

## plot interaction of two numerical variables for different categories
## coplot() stands for "conditioning plot";
## ``schuh ~ groesse | sex'' is the formula of a statistical model
## (read this as: shoe size depends on height, conditional on sex)
coplot(schuh ~ groesse | sex, data=people)

## we can do something similar by colouring the points
```

```
plot(people$groesse, people$schuh, xlab="Groesse", ylab="Schuhgroesse")
idx.m <- people$sex == "m"
idx.w <- people$sex == "w"
points(people$groesse[idx.m], people$schuh[idx.m], pch=21, bg="blue")
points(people$groesse[idx.w], people$schuh[idx.w], pch=21, bg="red")

######################################################################
##
##  Categorical data (nominal and ordinal scale)
##

## categorical data are described by a frequency table
table(people$sex)
table(people$linux)
table(people$stern)

## summary statistics such as mean and variance are not meaningful for
## categorical data; it is possible to obtain a rough approximation of the
## median for data on an ordinal scale, but R refuses to do so
median(people$kochnote)

## the summary of a factor (i.e. categorical) variable is just a frequency table
summary(people$kochnote)
summary(people$stern)

## plot() automatically displays categorical data as a barplot of the frequency table
plot(people$stern)
## this is actually a combination of table() and barplot()
barplot(table(people$stern))

## some more examples: don't forget to scale the y-axis for small frequency counts
barplot(table(people$stern), ylim=c(0,15))
barplot(table(people$linux))
barplot(table(people$sex))

## note the alphabetical ordering for the signs (people$stern);
## convert factor to ordered factor, making sequence of levels explicit
print(people$stern)
people$stern <- ordered(people$stern, levels=c
("Steinbock","Wassermann","Fische","Widder","Stier","Zwillinge","Krebs","Loewe","Jungfrau
","Waage","Skorpion","Schuetze"))
print(people$stern)
barplot(table(people$stern), ylim=c(0,15))

## if data on ordinal scale are coded as integers, convert explicitly into ordered factor
plot(people$kochen)
print(people$kochen)
people$kochen <- ordered(people$kochen)
print(people$kochen)
table(people$kochen)
barplot(table(people$kochen))

## cross-tabulation for two categorical variables: contingency table
table(people$microsoft, people$linux)
table(people$hasch, people$linux)
table(people$sex, people$kochnote)

######################################################################
##
##  Working with corpus frequency data
##

## Frequency counts are always based on a classification of objects or observations
## (= tokens) into a finite set of (predefined) categories (= types).  The frequency
```

```
## of a type is the number of tokens that were assigned to this category.

## As an example, we will look at the frequencies of the letters A...Z in a short
## English text. First, we have to split the text into tokens (for this purpose,
## each token is a single character) and assign each token to a type (letters
## a...z plus special types 0/SPC and 0/PUNCT).  This is best achieved with an
## external program (such as Perl or Emacs).  Here, we just read in the resulting
## table of tokens (= rows) with their type classification (= cell values).
corpus <- read.delim("the_garden_chars.tbl")
tokens <- corpus$char

## number of tokens and beginning of token/type vector; can you read the text?
length(tokens)
tokens[1:50]
## the vector has automatically been converted to a "factor" (= categorical data
## on nominal scale), which is appropriate for token/type data

## list types ("levels" of the factor) and determine number of types
levels(tokens)
length(levels(tokens))

## obtain frequency counts with the table() function and display as bar graph
table(tokens)
barplot(table(tokens))
## R knows that this is the sensible choice for nominal data:
plot(tokens)

## sort frequency table to find the most frequent types
freqs <- table(tokens)
print(sort(freqs))
print(sort(freqs, decreasing=TRUE))
freqs.sorted <- sort(freqs, decreasing=TRUE)

## plot distribution of all and of most frequent types
barplot(freqs.sorted)
barplot(freqs.sorted[1:10])

## scale to relative frequencies = probabilities
N <- length(tokens)
probs.sorted <- freqs.sorted / N
barplot(100 * probs.sorted, ylim=c(0,25), ylab="relative frequency (%)")
```