

R quick reference card

Session management

- > `q()` Quitting R (see page 8)
- > `ls()` List the objects in the current environment (see page 7)
- > `rm(...)` Remove objects from the current environment (see page 7)
- > `setwd(dir)` Set the current working directory (see page 7)
- > `getwd()` Get the current working directory (see page 7)

Getting help

- > `?function` Getting help on a function (see page 8)
- > `help(function)` Getting help on a function (see page 8)
- > `example(function)` Run the examples associated with the manual page for the function (see page 8)
- > `demo(topic)` Run an installed demonstration script (see page 8)
- > `apropos("topic")` Return names of all objects in search list that match “topic” (see page 9)
- > `help.search("topic")` Getting help about a concept (see page 9)
- > `help.start()` Launch R HTML documentation (see page 9)

Built in constants

- > `LETTERS` the 26 upper-case letters of the English alphabet (see page 17)
- > `letters` the 26 lower-case letters of the English alphabet (see page 17)

- > `month.name` English names of the 12 months of the year
- > `month.abb` Abbreviated English names of the 12 months of the year
- > `pi` π — the ratio of a circles circumference to diameter (see page 105)

Packages

- > `installed.packages()` List of all currently installed packages (see page 44)
- > `update.packages()` Update installed packages (see page 44)
- > `install.packages(pkgs)` Install package(s) (pkgs) from CRAN mirror (see page 45)
- R CMD INSTALL package** Install an add-on package (see page 43)
- > `library(package)` Loading an add-on package (see page 45)
- > `data(name)` Load a data set or structure inbuilt into R or a loaded package.

Importing/Exporting

- > `source("file")` Input, parse and sequentially evaluate the file (see page 45)
- > `sink("file")` Redirect non-graphical output to file
- > `read.table("file", header=T, sep=)` Read data in table format and create a data frame, with variables in columns (see page 51)
- > `read.table("clipboard", header=T, sep=)` Read data left on the clipboard in table format and create a data frame, with variables in columns (see page 51)
- > `read.systat("file.sysd", to.data.frame=T)` Read SYSTAT data file and create a data frame (see page 52)

- > `read.spss("file.sav", to.data.frame=T)` Read SPSS data file and create a data frame (see page 52)
- > `as.data.frame(read.mtp("file.mtp"))` Read Minitab Portable Worksheet data file and create a data frame (see page 52)
- > `read.xport("file")` Read SAS XPORT data file and create a data frame (see page 52)
- > `write.table(dataframe, "file", row.names=F, quote=F, sep=)` Write the contents of a dataframe to file in table format (see page 53)
- > `save(object, file="file.RData")` Write the contents of the object to file (see page 53)
- > `load(file="file.RData")` Load the contents of a file (see page 53)
- > `dump(object, file="file")` Save the contents of an object to a file (see page 53)

Generating Vectors

- > `c(...)` Concatenate objects (see page 6)
- > `seq(from, to, by=, length=)` Generate a sequence (see page 12)
- > `rep(x, times, each)` Replicate each of the values of `x` (see page 13)

Character vectors

- > `paste(..., sep=)` Combine multiple vectors together after converting them into character vectors (see page 13)
- > `substr(x, start, stop)` Extract substrings from a character vector (see page 14)

Factors

- > `factor(x)` Convert the vector (`x`) into a factor (see page 15)

- > **factor(x, levels=c())** Convert the vector (x) into a factor and define the order of levels (see page 15)
- > **gl(levels, reps, length, labels=)** Generate a factor vector by specifying the pattern of levels (see page 15)
- > **levels(factor)** Lists the levels (in order) of a factor (see page 54)
- > **levels(factor) <-** Sets the names of the levels of a factor (see page 54)

Matrices

- > **matrix(x, nrow, ncol, byrow=F)** Create a matrix with nrow and/or ncol dimensions out of a vector (x) (see page 16)
- > **cbind(...)** Create a matrix (or data frame) by combining the sequence of vectors, matrices or data frames by columns (see page 16)
- > **rbind(...)** Create a matrix (or data frame) by combining the sequence of vectors, matrices or data frames by rows (see page 16)
- > **rownames(x)** Read (or set with <-) the row names of the matrix (x) (see page 17)
- > **colnames(x)** Read (or set with <-) the column names of the matrix (x) (see page 17)

Lists

- > **list(...)** Generate a list of named (for arguments in the form name=x) and/or unnamed (for arguments in the form (x) components from the sequence of objects (see page 17)

Data frames

- > **data.frame(...)** Convert a set of vectors into a data frame (see page 49)

- > **row.names(dataframe)** Read (or set with <-) the row names of the data frame (see page 49)
- > **fix(dataframe)** View and edit a dataframe in a spreadsheet (see page 49)

Indexing

Vectors

- > **x[i]** Select the *i*th element (see page 21)
- > **x[i:j]** Select the *i*th through *j*th elements inclusive (see page 21)
- > **x[c(1,5,6,9)]** Select specific elements (see page 21)
- > **x[-i]** Select all except the *i*th element (see page 21)
- > **x["name"]** Select the element called "name" (see page 21)
- > **x[x > 10]** Select all elements greater than 10 (see page 22)
- > **x[x > 10 & x < 20]** Select all elements between 10 and 20 (both conditions must be satisfied) (see page 22)
- > **x[y == "value"]** Select all elements of x according to which y elements are equal to "value" (see page 22)
- > **x[x > 10 | y == "value"]** Select all elements which satisfy either condition (see page 22)

Matrices

- > **x[i,j]** Select element in row *i*, column *j* (see page 23)
- > **x[i,]** Select all elements in row *i* (see page 23)
- > **x[,j]** Select all elements in column *j* (see page 23)
- > **x[-i,]** Select all elements in each row other than the *i*th row (see page 23)
- > **x["name",1:2]** Select columns 1 through to 2 for the row named "name" (see page 23)
- > **x[x[,i] > 4,]** Select all rows for which the value of the column named "var1" is greater than 4 (see page 23)

- > **x[,x[, "var1"] == "value"]** Select all columns for which the value of the column named "var1" is equal to "value"
- Lists*
- > **x[[i]]** Select the *i*th object of the list (see page 24)
- > **x[["value"]]** Select the object named "value" from the list (see page 24)
- > **x[["value"]][1:3]** Select the first three elements of the object named "value" from the list (see page 24)
- Data frames*
- > **xc(i,j)** Select rows *i* and *j* for each column of the data frame (see page 56)
- > **x[, "name"]** Select each row of the column named "name" (see page 56)
- > **x[["name"]]** Select the column named "name"
- > **x\$name** Refer to a vector named "name" within the data frame (x) (see page 53)

Object information

- > **length(x)** number of elements in x (see page 34)
- > **class(x)** get the class of object x (see page 18)
- > **class(x) <-** set the class of object x (see page 18)
- > **attributes(x)** get (or set) the attributes of object x (see page 19)
- > **attr(x, which)** get (or set) the *which* attribute of object x (see page 19)
- > **is.na(x)**, **is.numeric(x)**, **is.character(x)**, **is.factor(x)**, ... methods used to assess the type of object x (methods (is) provides full list) (see page 18)

Object conversion

- > **as.null(x)**, **as.numeric(x)**, **as.character(x)**, **as.factor(x)**, ... methods used to covert x to the

specified type (methods `is`) provides full list) (see page 20)

Data manipulations

- > **subset(x, subset=, select=)** Subset a vector or data frame according to a set of conditions (see page 56)
- > **sample(x, size)** Randomly resample size number of elements from the `x` vector without replacement. Use the option `replace=TRUE` to sample with replacement. (see page 76)
- > **apply(x, INDEX, FUN)** Apply the function (`FUN`) to the margins (`INDEX=1` is rows, `INDEX=2` is columns, `INDEX=c(1, 2)` is both) of a vector, array or list (`x`) (see page 29)
- > **apply(x, factorlist, FUN)** Apply the function (`FUN`) to the vector (`x`) separately for each combination of the list of factors (see page 30)
- > **lapply(x, FUN)** Apply the function (`FUN`) to each element of the list `x` (see page 30)
- > **replicate(n, EXP)** Re-evaluate the expression (`EXP`) `n` times. Differs from `rep` function which repeats the result of a single evaluation (see page 28)
- > **aggregate(x, by, FUN)** Splits data according to a combination of factors and calculates summary statistics on each set (see page 58)
- > **sort(x, decreasing=)** Sorts a vector in increasing or decreasing (default) order (see page 26)
- > **order(x, decreasing=)** Returns a list of indices reflecting the vector sorted in ascending or descending order (see page 26)
- > **rank(x, ties.method=)** Returns the ranks of the values in the vector, tied values averaged by default (see page 27)
- > **which.min(x)** Index of minimum element in `x`
- > **which.max(x)** Index of maximum element in `x`

- > **rev(x)** Reverse the order of entries in the vector (`x`) (see page 27)
- > **unique(x)** Removes duplicate values (see page 337)
- > **t(x)** Transpose the matrix or data frame (`x`) (see page 387)
- > **cut(x, breaks)** Creates a factor out of a vector by slicing the vector `x` up into chunks. The option `breaks` is either a number indicating the number of cuts or else a vector of cut values (see page 111)
- > **which(x == a)** Each of the elements of `x` is compared to the value of `a` and a vector of indices for which the logical comparison is true is returned
- > **match(x,y)** A vector of the same length as `x` with the indices of the first occurrence of each element of `x` within `y`
- > **choose(n,k)** Computes the number of unique combinations in which `k` events can be arranged in a sequence of `n`
- > **combn(x,k)** List all the unique combinations in which the elements of `x` can be arranged when taken `k` elements at a time
- > **with(x, EXP)** Evaluate an expression (`EXP`) (typically a function) in an environment defined by `x` (see page 59)

Search and replace

- > **grep(pattern, x, ...)** Searches a character vector (`x`) for entries that match the pattern (`pattern`) (see page 24)
- > **regexpr(pattern, x, ...)** Returns the position and length of identified pattern (`pattern`) within the character vector (`x`) (see page 25)
- > **gsub(pattern, replacement, x, ...)** Replaces ALL occurrences of the pattern (`pattern`) within the character vector (`x`) with replacement (`replacement`) (see page 26)

- > **sub(pattern, replacement, x, ...)** Replaces THE FIRST occurrence of the pattern (`pattern`) within the character vector (`x`) with replacement (`replacement`) (see page 26)

Formatting data

- > **ceiling(x)** Rounds vector entries up to the nearest integer that is no smaller than the original vector entry (see page 27)
- > **floor(x)** Rounds vector entries up to the nearest integer that is no smaller than the original vector entry (see page 27)
- > **trunc(x)** Rounds vector entries to the nearest integer towards 0 (zero) (see page 27)
- > **round(x, digits=)** rounds vector entries to the nearest numeric with the specified number of decimal places (`digits=`). Digits of 5 are rounded off to the nearest even digit (see page 27)
- > **formatC(x, format=, digits=, ...)** Format vector entries according to a set of specifications (see page 28)

Math functions

Summary statistics

- > **mean(x)** Mean of elements of `x` (see page 70)
- > **var(x)** Variance of elements of `x` (see page 70)
- > **sd(x)** Standard deviation of elements of `x` (see page 70)
- > **length(x)** Number of elements of `x` (see page 34)
- > **sd(x)/sqrt(length(x))** Standard error of elements of `x` (see page 70)
- > **quantile(x, probs=)** Quantiles of `x` corresponding to probabilities (default: `0, 0.25, 0.5, 0.75, 1`)
- > **median(x)** Median of elements of `x` (see page 70)
- > **min(x)** Minimum of elements of `x` (see page 70)



- > **max(x)** Maximum of elements of x (see page 70)
 - > **range(x)** Same as $c(\min(x), \max(x))$ (see page 111)
 - > **sum(x)** Sum of elements of x (see page 106)
 - > **csum(x)** A vector the same length as x and whose i^{th} element is the sum of all elements up to and including i
 - > **prod(x)** Product of elements of x
 - > **cumprod(x)** A vector the same length as x and whose i^{th} element is the product of all elements up to and including i
 - > **cumin(x)** A vector the same length as x and whose i^{th} element is the minimum value of all elements up to and including i
 - > **cummax(x)** A vector the same length as x and whose i^{th} element is the maximum value of all elements up to and including i
 - > **var(x,y)** variance between x and y (matrix if x and y are matrices of data frames)
 - > **cov(x,y)** covariance between x and y (matrix if x and y are matrices of data frames)
 - > **cor(x,y)** linear correlation between x and y (matrix if x and y are matrices of data frames) (see page 226)
- Scale transformations*
- > **exp(x)** Transform values to exponentials (see page 212)
 - > **log(x)** Transform values to \log_e (see page 69)
 - > **log(x, 10)** Transform values to \log_{10} (see page 69)
 - > **log10(x)** Transform values to \log_{10} (see page 69)
 - > **sqrt(x)** Square root transform values of x (see page 69)
 - > **asin(sqrt(x))** Arcsin transform values of x (which must be proportions) (see page 69)
 - > **rank(x)** Transform values of x to ranks (see page 27)
 - > **scale(x, center=, scale=)** Scales (mean of 0 and sd of 1) values of x to ranks. To only center

- data, use `scale=FALSE`, to only reduce data use `center=FALSE` (see page 220)
- ## Distributions
- The following are used for the following list of distribution functions*
- x = a vector of quantiles
 - q = a vector of quantiles
 - p = a vector of probabilities
 - n = the number of observations
 - > **dnorm(x, mean, sd)**, **pnorm(q, mean, sd)**, **qnorm(p, mean, sd)**, **rnorm(n, mean, sd)**
Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd (see page 63)
 - > **dlnorm(x, meanlog, sdlog)**, **plnorm(q, meanlog, sdlog)**, **qlnorm(p, meanlog, sdlog)**, **rlnorm(n, meanlog, sdlog)**
Density, distribution function, quantile function and random generation for the log normal distribution whose logarithm has a mean equal to meanlog and standard deviation equal to sdlog (see page 63)
 - > **dunif(x, min, max)**, **runif(n, min, max)**, **qunif(p, min, max)**, **runif(n, min, max)**
Density, distribution function, quantile function and random generation for the uniform distribution with a minimum equal to min and maximum equal to max (see page 63)
 - > **dt(x, df)**, **pt(q, df)**, **qt(p, df)**, **rt(n, df)**
Density, distribution function, quantile function and random generation for the t distribution with df degrees of freedom
 - > **df(x, df1, df2)**, **pf(q, df1, df2)**, **qf(p, df1, df2)**, **rf(n, df1, df2)**
Density, distribution function, quantile function and random generation for the F distribution with df1 and df2 degrees of freedom

- > **dchisq(x, df)**, **pchisq(q, df)**, **qchisq(p, df)**, **rchisq(n, df)**
Density, distribution function, quantile function and random generation for the chi-squared distribution with df degrees of freedom (see page 499)
- > **dbinom(x, size, prob)**, **pbinom(q, size, prob)**, **qbinom(p, size, prob)**, **rbinom(n, size, prob)**
Density, distribution function, quantile function and random generation for the binomial distribution with parameters size and prob (see page 63)
- > **dnbinom(x, size, mu)**, **pnbinom(q, size, mu)**, **qnbinom(p, size, mu)**, **rnbinom(n, size, mu)**
Density, distribution function, quantile function and random generation for the negative binomial distribution with parameters size and mu (see page 63)
- > **dpois(x, lambda)**, **ppois(q, lambda)**, **qpois(p, lambda)**, **rpois(n, lambda)**
Density, distribution function, quantile function and random generation for the Poisson distribution with parameter lambda (see page 63)

Spatial procedures

- sp package
- > **Polygon(xy)** Convert a 2-column numeric matrix (xy) with coordinates into a object of class Polygon. Note the first point (row) must be equal to the last coordinates (row) (see page 79)
- > **Polygons(Plygnr, ID)** Combine one or more Polygon objects (Plygnr) together into an object of class Polygons. (see page 80)
- > **SpatialPolygons(xy)** A list of one or more Polygons. (see page 80)
- > **spsample(x, n, type=)** Generate approximately n points on or within a SpatialPolygons object (x). The



option `type=` indicates the type of sampling ("random", "regular", "stratified" or "non-aligned") (see page 80)

Plotting

- > **hist(x, breaks)** Histogram of the frequencies of vector `x`. The option `breaks` specifies how the bins are constructed and is typically either a number (number of bins), a vector of breakpoints (see page 116)
- > **plot(x)** Plot the values of `x` (on `y`-axis) ordered on `x`-axis (see page 85)
- > **plot(x, y)** Scatterplot of `y` (on `y`-axis) against `x` (`x`-axis) (see page 37)
- > **plot(formula)** If all vectors numeric - Scatterplot of `lhs` (on `y`-axis) against `rhs` (`x`-axis), otherwise a "box-and-whisker" plot with a separate box for each combination of `rhs` categories (see page 37)
- > **boxplot(x)** "Box-and-whiskers" plot for vector or formula `x` (see page 119)
- > **pairs(x)** Scatterplot matrices for multiple numeric vectors or formula `x` (see page 122)
- > **Mbargraph(dv, iv)** Bargraph (*biology package*) of mean `dv` against categorical `iv` with error bars (see page 268)
- > **interaction.plot(x.fact, trace.fact, response)** Plots the mean (or other summary) of the response (`response`) for two-way combinations of factors (`x`-axis factor: `x.fact` and trace factor: `trace.fact`), thereby illustrating possible interactions (see page 126)
- > **scatterplot(x)** (*car package*) Fancy scatterplot for a pair of numeric vectors or formula `x`. Includes boxplots on margins and regression line (see page 121)
- > **scatterplot.matrix(x)** (*car package*) Fancy scatterplot matrices for multiple numeric vectors or

formula `x`. Includes univariate displays in diagonals (see page 122)

Low-level plotting commands

- > **points(x, y)** Adds points with coordinates `x`, `y`. Option `type=` can be used (see page 99)
- > **lines(x, y)** Adds lines with coordinates `x`, `y`. Option `type=` can be used (see page 109)
- > **abline(fit)** Adds a regression line from the linear model `fit` (see page 109)
- > **abline(a, b)** Adds a regression line with a `y`-intercept of `a` and a slope of `b`
- > **axis(text, at, labels, ...)** Adds an axis to the bottom (`side=1`), left (`side=2`), top (`side=3`) or right (`side=4`) plot margin. Options `at` and `labels` can be used to specify where to draw tick marks and what labels to put at each tick mark (see page 107)
- > **box(which=, bty=, ...)** Draws a box around the plot (which="plot"), figure (which="figure"), inner (which="inner") or outer (which="outer") region of the current plot. Option `bty` specifies the type of box to draw ("o", "l", "7", "c", "u" or "j" result in boxes that resembles the corresponding upper case letter) (see page 127)
- > **mtext(text, side, line=0, ...)** Adds text (`text`) to the plot margin specified by `side` (see `axis()` above). Option `line` specifies the distance (in lines) away from the axis to put the text (see page 101)
- > **matlines(x, y, ...)** Adds confidence or prediction (`y`) limits along a sequence (`x`) to the plot (see page 113)
- > **data.ellipse(x, y, levels, ...)** Adds data ellipses from vectors (`x, y`) to the plot (see page 184)
- > **confidence.ellipse(x, y, levels, ...)**, **confidence.ellipse(model, ...)** Adds

confidence ellipses to the plot for linear models from vectors (`x, y`) or fitted model

Model fitting

- > **contrasts(x)** View the contrasts associated with the factor `x` (see section 7.3.1)
- > **contrasts(x) <- value** Set the contrasts associated with the factor `x`. The `value` parameter can either be a numeric matrix of coefficients or else a quoted name of a function that computes the matrix. (see section 7.3.1)
- > **lm(formula)** Fit linear model from formula of format `response ~ predictor1 + predictor2 + ... use I(x*y) + I(x^2)` to include nonlinear terms (see chapters 8&10)
- > **lm.ii(formula)** (*biology package*) Fit linear model II regression from formula of format `response~predictor`. (see chapter 8)
- > **rlm(formula)** (*MASS package*) Fit Maximum likelihood estimator linear model from formula of format `response~predictor`. (see chapter 8)
- > **mblm(formula)** (*mblm package*) Fit nonparametric regression model from formula of format `response~predictor`. (see chapter 8)
- > **glm(formula, family)** Fit generalized linear model from formula. Error distribution and link function are specified by `family` - see `family()` (see chapter 17)
- > **aov(formula)** Fit an anova model by making a call to `lm` for each stratum within formula (see chapters 10-15)
- > **nls(formula, start)** Determine the nonlinear least-squares estimates of the parameters of a nonlinear model formula. Starting estimates are provided as a named list or numeric vector (`start`) (see chapter 9)
- > **lme(fixed, random, correlation, ...)** (*nlme package*) Fit linear mixed effects models from

- > a specification of the fixed-effects formula (`fixed`) and random-effects formula (`random`) and correlation structure (`correlation`) (see chapters 11-14)
- > **lmer(formula, ...)** (*lme4 package*) Fit (generalized) linear mixed effects models from a specification of a formula (`formula`) (see chapters 11-14)
- > **gam(formula, family=, ...)** (*gam package*) Fit generalized additive models from the formula (`formula`). Error distribution and link function are specified by `family` - see `family()` (see chapter 17)
- > **pvals.fnc(.lmer, nsim, withMCMC, ...)** (*languageR package*) Calculate p-values from lmer models (`.lmer`) via Markov Chain Monte Carlo sampling. (see chapters 11-14)
- > **VarCorr(fit)** (*nlme package*) Calculate variance components from a linear mixed effects model (`fit`). (see chapters 11-14)
- Fit diagnostics** *The following generic functions can be applied to some of the above fitted model objects*
- > **plot(fit)** Diagnostic plots for a fitted model `fit` (see chapters 8-15)
- > **av.plots(fit)** Added-variable (partial-regression) plots for a fitted model `fit` (see chapter 9)
- > **residuals(fit)** Residuals from a fitted model `fit` (see chapters 8-15)
- > **deviance(fit)** Deviance of a fitted model `fit` (see chapter 17)
- > **influence.measures(fit)** Regression diagnostics for a fitted model `fit` (see chapters 8-15, 17)
- > **vif(fit)** Calculate variance-inflation factor for a fitted model `fit` (see chapters 9, 17)
- > **1/vif(fit)** Calculate tolerance for each term in a fitted model `fit` (see chapters 9, 17)
- > **predict(fit, data.frame)** Predicted responses from a fitted model `fit` given a set of predictor values `data.frame` (see chapters 8-15, 17)

- > **confint(fit)** Parameter confidence intervals from a fitted model `fit` (see chapters 8-15, 17)
- > **replications(formula)** Determine the number of replicates of each term in `formula` (see chapters 11-15)
- > **is.balanced(formula)** (*biology package*) Determine whether the design specified by the `formula` is balanced (see chapters 11-15)
- > **tukey.nonadd.test(fit)** (*ahr3 package*) Perform Tukey's test for nonadditivity from a model (`fit`) fitted via `lm()` (see chapters 13-15)

Measures of model fit

- > **extractAIC(fit, ...)** Compute AIC for parametric model (`fit`). Equivalent BIC using `k=log(nrow(dataset))` argument. (see chapters 9 & 17)
- > **AIC(fit, ...)** Compute AIC for any model (`fit`). Equivalent BIC using `k=log(nrow(dataset))` argument. (see chapters 9 & 17)
- > **AICc(fit)** (*MuMIn package*) Compute AIC corrected for small sample sizes for a fitted model (`fit`). (see chapters 9 & 17)
- > **QAIC(fit)** (*MuMIn package*) Compute quasi-AIC corrected for overdispersion for a fitted model (`fit`). (see chapters 9 & 17)
- > **QAICc(fit)** (*biology package*) Compute quasi-AIC corrected for overdispersion and small sample sizes for a fitted model (`fit`). (see chapters 9 & 17)
- > **deviance(fit)** Compute deviance for a fitted model (`fit`). (see chapters 9 & 17)
- > **Model.selection(fit)** (*biology package*) Generate various model fit estimates and perform model averaging for all possible combinations of predictor variables in a supplied model `fit`. (see chapters 9 & 17)

- > **dredge(fit)** (*MuMIn package*) Select most parsimonious model from all possible combinations of predictor variables in a supplied model `fit` based on information criteria (`rank`= either "AICc", "QAIC" or "BIC"). (see chapters 9 & 17)
- > **model.avg(ml)** (*MuMIn package*) Perform model averaging from a supplied fitted model object `ml` returned from model dredging. (see chapters 9 & 17)

Post-hoc analyses

- > **mainEffects(fit, at)** (*biology package*) Perform main effects tests from the fitted model (`fit`) (see chapters 12-15, 17)
- > **glt(fit, linct=mcp(FACTOR=type))** (*multcomp package*) Post-hoc, pairwise comparisons of factor (`FACTOR`). Option `type` specifies what type of post-hoc test to perform ("Dunnett", "Tukey", "sequen", "AVE", "Changeoint", "Williams", "Marcus", "McDermott") (see chapter 10)
- > **npmc(dataset, ...)** (*npmc package*) Non-parametric post-hoc, pairwise comparisons on a specifically constructed dataset (`dataset`). (see chapter 10)
- > **mt.rawp2adjp(pvalues, proc)** (*multtest package*) Multiple pairwise comparison p-value (`pvalues`) adjustments. (see chapter 10)
- > **p.adjust(pvalues, method)** Multiple pairwise comparison p-value (`pvalues`) adjustments. (see chapter 10)

Statistics and summaries

- > **t.test(x, y, t.test(formula))** One and two sample t-tests on vectors (`x`, `y`) or formula `formula`. Option `var.equal` indicates whether pooled or separate variance t-test and option `paired` indicates whether independent or paired t-test (see chapter 6)

- > **cor.test(x, y)**, **cor.test(formula)** Correlation between sample pairs from separate vectors (*x*, *y*, ...) or formula *formula*, ... Option method indicates the form of correlation ('pearson', kendall or spearman') (see chapter 8)
- > **hier.part(y, data, gof)** (*hier-part package*) Hierarchical partitioning given a vector of dependent variables *y* and a data frame *data*. Option *gof*= used to specify assessment of fit (root mean square prediction error: "RMSPE", Log-Likelihood: "LogLik" or R-squared: "Rsq") (see chapter 9)
- > **anova(fit, ...)** Compute analysis of variance table for a fitted model *fit* or models (see chapters 8-15, 17)
- > **summary(fit)** Summarize parameter estimates for a fitted model *fit* (see chapters 8-15, 17)
- > **AnovaM(fit, ...)** (*biology package*) Compute analysis of variance table for a fitted model *fit* accommodating unbalanced hierarchical designs (see chapters 11-15)
- > **wilcox.JN(fit)** (*biology package*) Perform Wilcoxon modified Johnson-Neyman procedure on fitted ANCOVA model (*fit*) (see chapter 15)
- > **tree(formula, ...)** (*rree package*) Perform binary recursive partitioning (regression tree) from response and predictors specified in *formula*. (see chapter 9)

Robust statistics

- > **wilcox.test(x, y)**, **t.test(formula)** One and two sample ("Mann-Whitney") Wilcoxon testson

- vectors (*x*, *y*) or formula *formula*. Option indicates whether independent or paired Wilcoxon-test (see chapter 6)
- > **oneway.test(formula, ...)** Perform Welch's test comparing the means of two or more groups specified by formula *formula*, ... (see chapter 10)
- > **kruskal.test(formula, ...)** Perform Kruskal-Wallis rank sum test, specified by formula *formula*, ... (see chapter 10)
- > **friedman.test(formula, ...)** Perform Friedman rank sum test with unreplicated blocked data, specified by formula *formula*, ... (see chapter 13)
- > **friedmanmc(DV, FACTOR, BLOCK)** (*pgirmess library*) Multiple pairwise comparison test following Friedman's test. (see chapter 13)

Frequency analysis

- > **chisq.test(x)** Performs chi-squared goodness-of-fit tests and contingency table tests. (see chapter 16)
- > **fisher.test(x)** Performs fishers exact test goodness-of-fit tests and contingency table tests. (see chapter 16)
- > **ks.test(x)** Performs Kolmogorov-Smirnov tests. (see chapter 16)
- > **g.test(x)** (*biology package*) Performs G-test for goodness-of-fit tests and contingency table tests. (see chapter 16)
- > **oddsratios(xtab)** (*biology package*) Calculate pairwise odds ratios from a contingency table (*xtab*). (see chapter 16-17)

Bootstrapping

- > **boot(data, stat, R, sim, rand.gen)** (*boot package*) Generates R bootstrap replicates from a statistical function (*stat*) incorporating a particular simulation (*sim*=one of "parametric", "balanced", "permutation" or "arithmetic"). Function *rand.gen* defines how randomization occurs (see page 149)

Power analysis

- > **power.t.test(n, delta, sd, power)** Calculate one of; sample size (*n*), true difference in means (*delta*), standard deviation (*sd*) or power (*power*) of t-test. The option *type* indicates the type of t-test ("two.sample", "one.sample", "paired")
- > **pwr.x.test(n, x, power)** (*pwr package*) Calculate one of; sample size (*n*), correlation coefficient (*x*) or power (*power*) of t-test. > **power.anova.test(groups, n, between.var, within.var, power)** Calculate one of; number of groups (*groups*), sample size (*n*), between group variance (*between.var*), within group variation (*within.var*) or power (*power*) of ANOVA.
- > **pwr.chisq.test(w, N, df, power)** (*pwr package*) Calculate one of; effect size (*w*), total number of observations (*N*), degrees of freedom (*df*) or power (*power*) of chi-square test.