R Reference Card

by Tom Short, EPRI PEAC, tshort@epri-peac.com 2004-10-21

Granted to the public domain. See www.Rpad.org for the source and latest version. Includes material from R for Beginners by Emmanuel Paradis (with permission).

Getting help
Most R functions have online documentation. help(topic) documentation on topic ?topic id. help.search("topic") search the help system apropos("topic") the names of all objects in the search list matching the regular expression "topic"
help.start() start the HTML version of help

Input and output
load() load the datasets written with save data(x) loads specified data sets library(x) load add-on packages
read.table(file) reads a file in table format and creates a data frame from it; the default separator sep="" is any whitespace; use header=TRUE to read the first line as a header of column names; use as.is=TRUE to prevent character vectors from being converted to factors; use comment.char="" to prevent # from being interpreted as a comment; use skip=n to skip n lines before reading data; see the help for options on row naming, NA treatment, and others
read.csv("filename", header=TRUE) id. but with defaults set for reading comma-delimited files
read.delim("filename", header=TRUE) id. but with defaults set for reading tab-delimited files
read.fwf(file, widths, header=FALSE, sep="", as.is=FALSE) read a table of fixed width formatted data into a 'data.frame'; widths is an integer vector, giving the widths of the fixed-width fields
save(file, ...) saves the specified objects (...) in the XDR platform-independent binary format
save.image(file) saves all objects

cat(..., file="", sep="") prints the arguments after coercing to character; sep is the character separator between arguments
print(a,...) prints its arguments; generic, meaning it can have different methods for different objects
format(x,...) format an R object for pretty printing
write.table(x, file="", row.names=TRUE, col.names=TRUE, sep="") prints x after converting to a data frame; if quote is TRUE, character or factor columns are surrounded by quotes ("); sep is the field separator; eol is the end-of-line separator; na is the string for missing values; use col.names=NA to add a blank column header to get the column headers aligned correctly for spreadsheet input

sink(file) output to file, until sink() Most of the I/O functions have a file argument. This can often be a character string naming a file or a connection. file="" means the standard input or output. Connections can include files, pipes, zipped files, and R variables. On windows, the file connection can also be used with description -"clipboard". To read a table copied from Excel, use x <- read.delim("clipboard") To write a table to the clipboard for Excel, use write.table(x,"clipboard",sep="",col.names=NA)

ls() show objects in the search path; specify pat="pat" to search on a pattern
ls.str() str() for each variable in the search path
dir() show files in the current directory
methods(a) shows S3 methods of class a
ls(x)

Slicing and extracting data
Indexing vectors
x[n] n\th element
all but the nth element
first n elements
elements from n+1 to the end
specific elements
element named "name"
all elements greater than 3
all elements between 3 and 5
all elements in the given set

Variable conversion
as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information
is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)

Variable selection
x[1:n] first n elements
x[1] first element
x[1:n] first n elements
x[1:4] first 4 elements
x[1:3] first 3 elements
x="name" element named "name"
x$name element

Data creation
Add vectors
c(...) generic function to combine arguments with the default forming a vector; with recursive=TRUE descends through lists combining all elements into one vector
from:to generates a sequence: "2:4" has operator priority; 1:4+1 is "2,3,4,5"
seq(from,to) generates a sequence by specifying increment; length=specifies desired length
seq(along=x) generates 1, 2, ..., length(along); useful for for loops
rep(x, times) replicate x times; use each= to repeat "each" element of x times each; rep(c(1,2,3),2) is 1 2 3 1 2 3; rep(c(1,2,3),each=2) is 1 1 2 2 3 3
data.frame(...) create a data frame of the named or unnamed arguments: data.frame(v=1:4,ch=c("a","B","c","d"),n=10); shorter vectors are recycled to the length of the longest
list(...) create a list of the named or unnamed arguments; list(a=c(1,2),b="hi",c=31);
array(x,dim) array with data x; specify dimensions like dim=c(3,2); elements of x recycle if x is not long enough
matrix(x,nrow=ncol) matrix; elements of x recycle if x is not long enough

Variable conversion
as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information
is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)

Variable selection
x[1:n] first n elements
x[1] first element
x[1:n] first n elements
x[1:4] first 4 elements
x[1:3] first 3 elements
x="name" element named "name"
x$name element

Indexing lists
x[n] list with elements n
x[n] n\th element of the list
x["name"] element of the list named "name"
x$name id.

Data creation
Add vectors
c(...) generic function to combine arguments with the default forming a vector; with recursive=TRUE descends through lists combining all elements into one vector
from:to generates a sequence: "2:4" has operator priority; 1:4+1 is "2,3,4,5"
seq(from,to) generates a sequence by specifying increment; length=specifies desired length
seq(along=x) generates 1, 2, ..., length(along); useful for for loops
rep(x, times) replicate x times; use each= to repeat "each" element of x times each; rep(c(1,2,3),2) is 1 2 3 1 2 3; rep(c(1,2,3),each=2) is 1 1 2 2 3 3
data.frame(...) create a data frame of the named or unnamed arguments: data.frame(v=1:4,ch=c("a","B","c","d"),n=10); shorter vectors are recycled to the length of the longest
list(...) create a list of the named or unnamed arguments; list(a=c(1,2),b="hi",c=31);
array(x,dim) array with data x; specify dimensions like dim=c(3,2); elements of x recycle if x is not long enough
matrix(x,nrow=ncol) matrix; elements of x recycle if x is not long enough

Variable conversion
as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information
is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)

Variable selection
x[1:n] first n elements
x[1] first element
x[1:n] first n elements
x[1:4] first 4 elements
x[1:3] first 3 elements
x="name" element named "name"
x$name element

Indexing lists
x[n] list with elements n
x[n] n\th element of the list
x["name"] element of the list named "name"
x$name id.

Indexing matrices
x[i,j] element at row i, column j
x[i,] row i
x[,j] column j
x[,c(1,3)] columns 1 and 3
x["name",] row named "name"

Indexing data frames (matrix indexing plus the following)
x["name"] column named "name"
x$name id.

Variable conversion
as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information
is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)

Variable selection
x[1:n] first n elements
x[1] first element
x[1:n] first n elements
x[1:4] first 4 elements
x[1:3] first 3 elements
x="name" element named "name"
x$name element

Indexing lists
x[n] list with elements n
x[n] n\th element of the list
x["name"] element of the list named "name"
x$name id.

Indexing matrices
x[i,j] element at row i, column j
x[i,] row i
x[,j] column j
x[,c(1,3)] columns 1 and 3
x["name",] row named "name"

Indexing data frames (matrix indexing plus the following)
x["name"] column named "name"
x$name id.

Variable conversion
as.array(x), as.data.frame(x), as.numeric(x), as.logical(x), as.complex(x), as.character(x), ... convert type; for a complete list, use methods(as)

Variable information
is.na(x), is.null(x), is.array(x), is.data.frame(x), is.numeric(x), is.complex(x), is.character(x), ... test for type; for a complete list, use methods(is)

Variable selection
x[1:n] first n elements
x[1] first element
x[1:n] first n elements
x[1:4] first 4 elements
x[1:3] first 3 elements
x="name" element named "name"
x$name element

Indexing lists
x[n] list with elements n
x[n] n\th element of the list
x["name"] element of the list named "name"
x$name id.

Indexing matrices
x[i,j] element at row i, column j
x[i,] row i
x[,j] column j
x[,c(1,3)] columns 1 and 3
x["name",] row named "name"

Indexing data frames (matrix indexing plus the following)
x["name"] column named "name"
x$name id.
bivariate plot of the first column of \( x \) vs. the first one of \( y \), the second one of \( x \) vs. the second one of \( y \), etc. 

`fourfoldplot(x)` visualizes, with quarters of circles, the association between two dichotomous variables for different populations \( (x \) must be an array with `dim=c(2, 2, k)`, or a matrix with `dim=c(2, 2)` if \( k = 1 \) 

`assocplot(x)` Cohen–Friendly graph showing the deviations from independence of rows and columns in a two dimensional contingency table.

`mosaicplot(x)` ‘mosaic’ graph of the residuals from a log-linear regression of a contingency table.

`pairs(x)` if \( x \) is a matrix or a data frame, draws all possible bivariate plots between the columns of \( x \)

`plot.ts(x)` if \( x \) is an object of class "ts", plot of \( x \) with respect to time, \( x \) may be multivariate but the series must have the same frequency and dates.

`ts.plot(x)` id. but the series must have the same frequency and dates

`qqnorm(x)` quantiles of \( x \) with respect to the values expected under a normal law

`qqplot(x, y)` quantiles of \( x \) with respect to the quantiles of \( y \)

`contour(x, y, z)` contour plot (data are interpolated to draw the curves), \( x \) and \( y \) must be vectors and \( z \) must be a matrix so that `dim(z)=c(length(x), length(y))` (\( x \) and \( y \) may be omitted).

`filled.contour(x, y, z)` adds the drawn elements and the blanks, for example `lty="44"` will have the same effect than `lty=2`

`lwd` a numeric which controls the width of lines, default 1

`mar` a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`

`mfcol` a vector of the form `c(nz, nc)` which partitions the graphical window as a matrix of \( nr \) lines and \( nc \) columns, the plots are then drawn in columns

`mfrow` id. but the plots are drawn row by row

`pch` controls the type of symbol, either an integer between 1 and 25, or any single character within ""'

`ps` an integer which controls the size in points of texts and symbols

`pty` a character which specifies the type of the plotting region, "s": square, "n": maximal

`tck` a value which specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if `tck=-1` a grid is drawn

`tcl` a value which specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default `tcl=0.5`)

`xaxt` if `xaxt="n"` the \( x \)-axis is set but not drawn (useful in conjunction with `axis(side=1, ...)`) 

`yaxt` if `yaxt="n"` the \( y \)-axis is set but not drawn (useful in conjunction with `axis(side=2, ...)`) 

**Lattice (Trellis) graphics**

`xyplot(y ~ x)` bivariate plots (with many functionalities)

`barchart(y ~ x)` histogram of the values of \( y \) with respect to those of \( x \)

`dotplot(y ~ x)` Cleveland dot plot (stacked plots line-by-line and column-by-column)

`densityplot(y ~ x)` density functions plot

`histogram(x ~ x)` histogram of the frequencies of \( x \)

`bwplot(y ~ x)` "box-and-whiskers" plot

`qqmath(x ~ x)` quantiles of \( x \) with respect to the values expected under a theoretical distribution

`sstripplot(y ~ x)` single dimension plot, \( x \) must be numeric, \( y \) may be a factor

`qq(y ~ x)` quantiles to compare two distributions, \( x \) must be numeric, \( y \) may be numeric, character, or factor but must have two 'levels'

`splom(x ~ x)` matrix of bivariate plots

`parallel(x ~ x)` parallel coordinates plot

`levelplot(z ~ x*y | g1*g2)` coloured plot of the values of \( z \) at the coordinates given by \( x \) and \( y \); \( x, y, z \) are all of the same length

`wireframe(z ~ x*y | g1*g2)` 3d surface plot

`cloud(z ~ x*y | g1*g2)` 3d scatter plot
In the normal Lattice formula, $y \times g1 \times g2$ has combinations of optional conditioning variables $g1$ and $g2$ plotted on separate panels. Lattice functions take many of the same arguments as base graphics plus also `data=` the data frame for the formula variables and `subset=` for subsetting. Use `panel=` to define a custom panel function (see `apropos("panel")` and `?llines`). Lattice functions return an object of class trellis and have to be `print`-ed to produce the graph. Use `print(xplot(...))` inside functions where automatic printing doesn’t work. Use `lattice.theme` and `lset` to change Lattice defaults.

**Optimization and model fitting**

`optim(par, fn, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN")` general-purpose optimization; `par` is initial values, `fn` is function to optimize (normally minimize) `nlm(f,p)` minimize function $f$ using a Newton-type algorithm with starting values $p$

`lm(formula)` fit linear models; formula is typically of the form $response \sim termA + termB + ...$; use $I(x^2) + I(x'2)$ for terms made of nonlinear components

`glm(formula, family=)` fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution; `family` is a description of the error distribution and link function to be used in the model; see `?family`

`nls(formula)` nonlinear least-squares estimates of the nonlinear model parameters

`approx(x,y=)` linearly interpolate given data points; $x$ can be an xy plotting structure

`spline(x,y=`) cubic spline interpolation

`loess(formula)` fit a polynomial surface using local fitting

Many of the formula-based modeling functions have several common arguments: `data=` the data frame for the formula variables, `subset=` a subset of variables used in the fit, `na.action=` action for missing values: "na.fail", "na.omit", or a function. The following generics often apply to model fitting functions:

- `predict(fit,....)` predictions from `fit` based on input data
- `df.residual(fit)` returns the number of residual degrees of freedom
- `coef(fit)` returns the estimated coefficients (sometimes with their standard-errors)
- `residuals(fit)` returns the residuals
- `deviance(fit)` returns the deviance
- `fitted(fit)` returns the fitted values
- `logLik(fit)` computes the loglikelihood of the likelihood and the number of parameters
- `AIC(fit)` computes the Akaike information criterion or AIC

**Statistics**

`aov(formula)` analysis of variance model

`anova(fit,...)` analysis of variance (or deviance) tables for one or more fitted model objects

`density(x)` kernel density estimates of $x$

`binom.test()`, `pairwise.t.test()`, `power.t.test()`, `prop.test()`, `t.test()`, ... use `help.search("test")`

**Distributions**

- `rnorm(n, mean=0, sd=1)` Gaussian (normal)
- `rexp(n, rate=1)` exponential
- `rgamma(n, shape, scale=1)` gamma
- `rpois(n, lambda)` Poisson
- `rweibull(n, shape, scale=1)` Weibull
- `rccauchy(n, location=0, scale=1)` Cauchy
- `rbeta(n, shape1, shape2)` beta
- `rt(n, df) 'Student' (t)`
- `rf(n, df1, df2)` Fisher–Snedecor (F) ($\chi^2$)
- `rchisq(n, df)` Pearson
- `rbinom(n, size, prob)` binomial
- `rgeom(n, prob)` geometric
- `rhyper(nn, m, n, k)` hypergeometric
- `rlogis(n, location=0, scale=1)` logistic
- `rlnorm(n, meanlog=0, sdlog=1)` lognormal
- `rnbinom(n, size, prob)` negative binomial
- `runif(n, min=0, max=1)` uniform
- `rwilcox(nn, m, n), rsignrank(nn, n)` Wilcoxon’s statistics

All these functions can be used by replacing the letter `r` with `d`, `p` or `q` to get, respectively, the probability density ($dfunc(x, ...)$), the cumulative probability density ($pfunc(x, ...)$), and the value of quantile ($qfunc(p, ...)$), with $0 < p < 1$.

**Programming**

`function( arglist ) expr` function definition

`return(value)`

`if(cond) expr` or

`if(cond) cons.expr else alt.expr`

`for(var in seq) expr` or

`while(cond) expr` or

`repeat expr` or

`break` or

`next`

Use braces `{}` around statements

`ifelse(test, yes, no)` a value with the same shape as `test` filled with elements from either `yes` or `no`

`do.call(funnname, args)` executes a function call from the name of the function and a list of arguments to be passed to it.