

# R Reference Card

by Tom Short, EPRI PEAC, tshort@epri-peac.com 2004-11-07

Granted to the public domain. See [www.Rpad.org](http://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

**str(a)** display the internal \*str\*ucture of an R object

**summary(a)** gives a "summary" of *a*, usually a statistical summary but it is *generic* meaning it has different operations for different classes of *a*

**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 *a*

**methods(class=class(a))** lists all the methods to handle objects of class *a*

## 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="^\t",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 from Excel, use

```
write.table(x,"clipboard",sep="\t",col.names=NA)
```

For database interaction, see packages RODBC, DBI, RMySQL, RPgSQL, and ROracle. See packages XML, hdf5, netCDF for reading other file formats.

## Data creation

**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; ":" has operator priority; 1:4 + 1 is "2,3,4,5"

**seq(from,to)** generates a sequence by= specifies 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* each times; 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=3i);

**array(x,dim=)** array with data *x*; specify dimensions like dim=c(3,4,2); elements of *x* recycle if *x* is not long enough

**matrix(x,nrow=,ncol=)** matrix; elements of *x* recycle

**factor(x,levels=)** encodes a vector *x* as a factor

**gl(n,k,length=n\*k,labels=1:n)** generate levels (factors) by specifying the pattern of their levels; *k* is the number of levels, and *n* is the number of replications

**expand.grid()** a data frame from all combinations of the supplied vectors or factors

**rbind(...)** combine arguments by rows for matrices, data frames, and others

**cbind(...)** id. by columns

## Slicing and extracting data

Indexing vectors

```
x[n]          nth element
x[-n]         all but the nth element
x[1:n]        first n elements
x[-(1:n)]     elements from n+1 to the end
x[c(1,4,2)]   specific elements
x["name"]     element named "name"
x[x > 3]      all elements greater than 3
x[x > 3 & x < 5] all elements between 3 and 5
x[x %in% c("a","and","the")] elements in the given set
```

Indexing lists

```
x[n]          list with elements n
x[[n]]        nth 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)

**length(x)** number of elements in *x*

**dim(x)** Retrieve or set the dimension of an object; dim(x) <- c(3,2)

**dimnames(x)** Retrieve or set the dimension names of an object

**nrow(x)** number of rows; NROW(x) is the same but treats a vector as a one-row matrix

**ncol(x)** and NCOL(x) id. for columns

**class(x)** get or set the class of *x*; class(x) <- "myclass"

**unclass(x)** remove the class attribute of *x*

**attr(x,which)** get or set the attribute *which* of *x*

**attributes(obj)** get or set the list of attributes of *obj*

## Data selection and manipulation

**which.max(x)** returns the index of the greatest element of *x*

**which.min(x)** returns the index of the smallest element of *x*

**rev(x)** reverses the elements of *x*

**sort(x)** sorts the elements of *x* in increasing order; to sort in decreasing order: rev(sort(x))

**cut(x,breaks)** divides *x* into intervals (factors); *breaks* is the number of cut intervals or a vector of cut points

**match(x,y)** returns a vector of the same length than *x* with the elements of *x* which are in *y* (NA otherwise)

**which(x == a)** returns a vector of the indices of *x* if the comparison operation is true (TRUE), in this example the values of *i* for which x[i] == *a* (the argument of this function must be a variable of mode logical)

**choose(n,k)** computes the combinations of *k* events among *n* repetitions =  $n! / [(n-k)!k!]$

**na.omit(x)** suppresses the observations with missing data (NA) (suppresses the corresponding line if *x* is a matrix or a data frame)

**na.fail(x)** returns an error message if *x* contains at least one NA

**unique(x)** if *x* is a vector or a data frame, returns a similar object but with the duplicate elements suppressed

**table(x)** returns a table with the numbers of the different values of *x* (typically for integers or factors)

**subset(x, ...)** returns a selection of *x* with respect to criteria (...), typically comparisons: `x$V1 < 10`; if *x* is a data frame, the option `select` gives the variables to be kept or dropped using a minus sign

**sample(x, size)** resample randomly and without replacement `size` elements in the vector *x*, the option `replace = TRUE` allows to resample with replacement

**prop.table(x, margin=)** table entries as fraction of marginal table

## Math

**sin, cos, tan, asin, acos, atan, atan2, log, log10, exp**

**max(x)** maximum of the elements of *x*

**min(x)** minimum of the elements of *x*

**range(x)** id. then `c(min(x), max(x))`

**sum(x)** sum of the elements of *x*

**diff(x)** lagged and iterated differences of vector *x*

**prod(x)** product of the elements of *x*

**mean(x)** mean of the elements of *x*

**median(x)** median of the elements of *x*

**quantile(x, probs=)** sample quantiles corresponding to the given probabilities (defaults to 0, .25, .5, .75, 1)

**weighted.mean(x, w)** mean of *x* with weights *w*

**rank(x)** ranks of the elements of *x*

**var(x)** or `cov(x)` variance of the elements of *x* (calculated on  $n - 1$ ); if *x* is a matrix or a data frame, the variance-covariance matrix is calculated

**sd(x)** standard deviation of *x*

**cor(x)** correlation matrix of *x* if it is a matrix or a data frame (1 if *x* is a vector)

**var(x, y)** or `cov(x, y)` covariance between *x* and *y*, or between the columns of *x* and those of *y* if they are matrices or data frames

**cor(x, y)** linear correlation between *x* and *y*, or correlation matrix if they are matrices or data frames

**round(x, n)** rounds the elements of *x* to *n* decimals

**log(x, base)** computes the logarithm of *x* with base *base*

**scale(x)** if *x* is a matrix, centers and reduces the data; to center only use the option `center=FALSE`, to reduce only `scale=FALSE` (by default `center=TRUE`, `scale=TRUE`)

**pmin(x, y, ...)** a vector which *i*th element is the minimum of `x[i]`, `y[i]`, ...

**pmax(x, y, ...)** id. for the maximum

**cumsum(x)** a vector which *i*th element is the sum from `x[1]` to `x[i]`

**cumprod(x)** id. for the product

**cummin(x)** id. for the minimum

**cummax(x)** id. for the maximum

**union(x, y), intersect(x, y), setdiff(x, y), setequal(x, y), is.element(e1, set)** “set” functions

**Re(x)** real part of a complex number

**Im(x)** imaginary part

**Mod(x)** modulus; `abs(x)` is the same

**Arg(x)** angle in radians of the complex number

**Conj(x)** complex conjugate

**convolve(x, y)** compute the several kinds of convolutions of two sequences

**fft(x)** Fast Fourier Transform of an array

**mvfft(x)** FFT of each column of a matrix

**filter(x, filter)** applies linear filtering to a univariate time series or to each series separately of a multivariate time series

Many math functions have a logical parameter `na.rm=FALSE` to specify missing data (NA) removal.

## Matrices

**t(x)** transpose

**diag(x)** diagonal

**%%%** matrix multiplication

**solve(a, b)** solves a `%%%` `x = b` for *x*

**solve(a)** matrix inverse of *a*

**rowsum(x)** sum of rows for a matrix-like object; **rowSums(x)** is a faster version

**colsum(x), colSums(x)** id. for columns

**rowMeans(x)** fast version of row means

**colMeans(x)** id. for columns

## Advanced data processing

**apply(X, INDEX, FUN=)** a vector or array or list of values obtained by applying a function *FUN* to margins (*INDEX*) of *X*

**lapply(X, FUN)** apply *FUN* to each element of the list *X*

**tapply(X, INDEX, FUN=)** apply *FUN* to each cell of a ragged array given by *X* with indexes *INDEX*

**by(data, INDEX, FUN)** apply *FUN* to data frame *data* subsetted by *INDEX*

**merge(a, b)** merge two data frames by common columns or row names

**xtabs(a, b, data=x)** a contingency table from cross-classifying factors

**aggregate(x, by, FUN)** splits the data frame *x* into subsets, computes summary statistics for each, and returns the result in a convenient form; *by* is a list of grouping elements, each as long as the variables in *x*

**stack(x, ...)** transform data available as separate columns in a data frame or list into a single column

**unstack(x, ...)** inverse of `stack()`

**reshape(x, ...)** reshapes a data frame between ‘wide’ format with repeated measurements in separate columns of the same record and ‘long’ format with the repeated measurements in separate records; use (`direction=“wide”`) or (`direction=“long”`)

## Strings

**paste(...)** concatenate vectors after converting to character; `sep=` is the string to separate terms (a single space is the default); `collapse=` is an optional string to separate “collapsed” results

**substr(x, start, stop)** substrings in a character vector; can also assign, as `substr(x, start, stop) <- value`

**strsplit(x, split)** split *x* according to the substring *split*

**grep(pattern, x)** searches for matches to *pattern* within *x*; see `?regex`

**gsub(pattern, replacement, x)** replacement of matches determined by regular expression matching `sub()` is the same but only replaces the first occurrence.

**tolower(x)** convert to lowercase

**toupper(x)** convert to uppercase

**match(x, table)** a vector of the positions of first matches for the elements of *x* among *table*

**x %in% table** id. but returns a logical vector

**pmatch(x, table)** partial matches for the elements of *x* among *table*

**nchar(x)** number of characters

## Dates and Times

The class `Date` has dates without times. `POSIXct` has dates and times, including time zones. Comparisons (e.g. `>`), `seq()`, and `difftime()` are useful. `Date` also allows `+` and `-`. `?DateTimeClasses` gives more information. See also package `chron`.

**as.Date(s)** and **as.POSIXct(s)** convert to the respective class; `format(dt)` converts to a string representation. The default string format is “2001-02-21”. These accept a second argument to specify a format for conversion. Some common formats are:

`%a, %A` Abbreviated and full weekday name.

`%b, %B` Abbreviated and full month name.

`%d` Day of the month (01–31).

`%H` Hours (00–23).

`%I` Hours (01–12).

`%j` Day of year (001–366).

`%m` Month (01–12).

`%M` Minute (00–59).

`%p` AM/PM indicator.

`%S` Second as decimal number (00–61).

`%U` Week (00–53); the first Sunday as day 1 of week 1.

`%w` Weekday (0–6, Sunday is 0).

`%W` Week (00–53); the first Monday as day 1 of week 1.

`%y` Year without century (00–99). Don’t use.

`%Y` Year with century.

`%z` (output only.) Offset from Greenwich; `-0800` is 8 hours west of.

`%Z` (output only.) Time zone as a character string (empty if not available).

Where leading zeros are shown they will be used on output but are optional on input. See `?strftime`.

## Plotting

**plot(x)** plot of the values of *x* (on the *y*-axis) ordered on the *x*-axis

**plot(x, y)** bivariate plot of *x* (on the *x*-axis) and *y* (on the *y*-axis)

**hist(x)** histogram of the frequencies of *x*

**barplot(x)** histogram of the values of *x*; use `horiz=FALSE` for horizontal bars

**dotchart(x)** if *x* is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column)

**pie(x)** circular pie-chart

**boxplot(x)** “box-and-whiskers” plot

**sunflowerplot(x, y)** id. than `plot()` but the points with similar coordinates are drawn as flowers which petal number represents the number of points

**stripplot(x)** plot of the values of *x* on a line (an alternative to `boxplot()` for small sample sizes)

**coplot(x~y | z)** bivariate plot of *x* and *y* for each value or interval of values of *z*

**interaction.plot(f1, f2, y)** if *f1* and *f2* are factors, plots the means of *y* (on the *y*-axis) with respect to the values of *f1* (on the *x*-axis) and of *f2* (different curves); the option `fun` allows to choose the summary statistic of *y* (by default `fun=mean`)