R Cheat Sheet

by Dan Mirman, Drexel University, 2013 (v2.0). For more R Cheat Sheets see http://devcheatsheet.com/tag/r/.

Getting help

`?topic` documentation on `topic`
`??topic` search the help system
`apropos("topic")` the names of all objects in the search list matching the regular expression "`topic`"
`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

Input and Output

`dir()` show files in the current directory
`getwd(), setwd()` get and set working directory
`load()` load the datasets written with **`save`**
`require(x), 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 **`sep="\t"`** for tab-delimited files, **`sep=","`** for comma-delimited, etc.; use **`header=TRUE`** to read the first line as a header of column names; character vectors are converted to factors by default, use **`as.is=TRUE`** to override this; 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(file), read.delim(file)` versions of **`read.table`** with convenient defaults for comma-separated and tab-delimited files.
`save(x, y, z,… file="filename")` saves the objects `x,y,z`… in a R-format file called `filename`
`save.image(file)` saves all objects including loaded packages
`write.table(x, file="", row.names=TRUE, quote=TRUE, sep=" ", eol = "\n", na = "NA", append = FALSE)` prints `x` after converting to a data frame; if `quote`=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; prints row names unless `row.names`=FALSE; will overwrite an existing file unless `append`=TRUE.

Indexing

`x[i]` i-*th* element; can be a vector of element indices; if `i` is a logical vector, select all elements where `i` is TRUE
`x[-i]` all *but* the i-*th* element(s)
`x["name"]` element named "name"
`x$name` "name" column (or variable) in data frame x

Logic

`TRUE, FALSE`
`==, >, <, <=, >=` comparison operators; **==** can be used for strings and factors
`!` negation; use **`!=`** for not-equal-to
`&, |` and, or
`x %in% y` set membership: logical value for each element in `x` evaluating whether it matches any of the elements in set `y`.

Data Creation

`<-` assignment operator; use **`=`** only for arguments in a function call
`c(...)` generic function to combine arguments; default result is a vector
`from:to` generates a sequence; ":" has operator priority so **`1:4 + 1`** is "2,3,4,5"
`seq(from,to)` generates a sequence; use **by=x** to increment by `x`; use **`length.out=x`** to make a sequence of length `x`.
`rep(x,n)` replicate object `x`, `n` times; `rep(c(1,2,3),2)` is `1 2 3 1 2 3`; `rep(c(1:3),each=2)` is `1 1 2 2 3 3`
`data.frame(...)` create a data frame of the named or unnamed arguments; shorter vectors are recycled to the length of the longest; ex: `data.frame(v=1:4, ch=c("a","B","c","d"), n=10)`

Data Examination

`summary(x)` returns a summary of `x`; will return column properties for a `data.frame`, test result summary for statistical tests, etc.
`print(a, ...)` prints its arguments; can have different methods for different objects, including customizing output
`head(x), tail(x)` return the first or last elements in `x`; use **`head(x,n)`** to get the first `n` elements
`str(a)` display the internal structure of an R object
`levels(x)` returns the levels of factor `x`; to rename the levels use **`levels(x) <- c("A","B",...)`**
`length(x)` number of elements in `x`
`dim(x)` retrieve or set the dimension of an object;
`nrow(x), ncol(x)` number of rows or columns; **`NROW(x), NCOL(x)`** are the same but treat the vector as a one row or column matrix
`max(x), min(x)` returns the greatest or smallest element in `x; use which.max(x)` or `which.min(x)` to get the index of the greatest or smallest element of `x`.
`which(x == a)` returns a vector of the indices of **`x`** where the comparison operation is TRUE, e.g., the values of `i` for which `x[i] == a`
`is.na(x), is.null(x), is.data.frame(x)...` test for type; use `methods(is)` for a complete list

Data Manipulation

`rbind(...)` combine arguments by rows, i.e., stack vertically
`cbind(...)` combine arguments by columns, i.e., stack horizontally
`merge(a,b)` merge two data frames by common columns or row names; use `by.x` and `by.y` to specify common columns
`rev(x)` reverses the elements of `x`
`unique(x)` if `x` is a vector or a data frame, returns a similar object but with the duplicate elements excluded
`subset(x, criteria)` returns the subset of `x` where criteria are TRUE; useful for selecting rows (observations), e.g., `subset(x, time > 0)` will return all elements of `x` where `x$time` is greater than `0`; or selecting columns, e.g., `subset(x, select=c(time, value))` will return just the `time` and `value` columns of `x`; `subset(x, select = -junk)` will drop the `junk` column of `x`.
`as.numeric(x), as.factor(x)...` variable coercion/conversion; use `methods(as)` for full list
`t(x)` transpose
`quantile(x, probs=seq(0, 1, by=1/3))` find the break points that divide `x` into the specified quantiles (e.g., tertiles)
`cut(x, breaks=b, labels=c(...))` convert numeric vector `x` into a factor using breakpoints `b` and specified factor level labels
`replace(x, list, y)` replace the `list`ed values in `x` with the values in `y`; remember to assign the result
`sample(x, size)` takes a sample of the specified `size` from the elements of `x`; default is without replacement, use `replace=T` to override

Strings

`paste(...)` concatenate vectors after converting to character; default separator is a single space, to override use `sep=`
`substr(x,start,stop)` substrings in a character vector; can also assign: `substr(x, start, stop) <- value`
`strsplit(x,split)` split `x` according to the substring `split`; ex: `strsplit(x, "-")` will divide a string `x` into multiple strings based on locations of `-`
`grep(pattern,x)` searches for matches to pattern within `x`; for details 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

Math

`sin, cos, tan, asin, acos, atan, atan2, exp`
`max(x), min(x)` maximum and minimum of elements of `x`
`range(x)` same as: `c(min(x), max(x))`
`sum(x)` sum of the elements of `x`
`prod(x)` product of the elements of `x`
`cumsum(x)` returns a vector of same length as `x` with the cumulative sum of the elements of `x`
`mean(x), median(x)` mean and median of elements of `x`
`weighted.mean(x, w)` mean of `x` with weights `w`
`rank(x)` ranks of the elements of `x`
`sd(x)` standard deviation of `x`
`cor(x)` correlation matrix of `x` (matrix or data frame)
`round(x)` rounds the elements of `x`; use **`round(x, n)`** to round to `n` decimals

`log(x)` computes the natural logarithm of `x`; use **`log2(x)`**, **`log10(x)`**, or **`log(x, base)`** to specify other base

`union(x,y)`, `intersect(x,y)`, `setdiff(x,y)`, `setequal(x,y)`, `is.element(el,set)` "set" functions

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

Advanced Data Manipulation

`relevel(x, ref)` set `ref` as the reference levels of factor `x`

`factor(f, levels=c("B","A",...))` where **`B, A, ...`** are the levels of factor **`f`**, will return a factor of the same length as **`f`** with its levels reordered according to **`levels`**.

`reorder(f, dim, fun)` reorder the levels of factor `f` according to their values on dimension `dim`, values are computed by function `fun` (default: mean)

`melt(data, id, measure, variable.name = "variable", value.name = "value")` {reshape} convert data frame `data` from "wide" to "long" format; `id` specifies the variables that should remain in separate columns, can be identified by number: e.g., `c(1:4, 7)`, or name: e.g., `c("A","B",…)`; `measure` specifies the columns that should be collapsed into a single column, with same specification options as `id`; `variable.name` is the name of the new variable column; `value.name` is the name of the new column that contains the values that were in the measure columns

`dcast(data, formula, value.var, fun.aggregate)` {reshape} convert data frame `data` from "long" to "wide" format; in `formula`: variables to the left of the ~ define rows, and variables to the right define columns; `value.var` is the column to use for filling the new columns; can be used to create summary tables by specifying `fun.aggregate` (default is `length`, can also use `mean`, `median`, etc.) and optional `margins` argument

`ddply(data, variables, fun)` {plyr} split data frame `data` into subsets defined by each unique combination of `variables`, apply function `fun`, and return combined results; **`ddply(data, variables, summarize, ...)`** to define specialized summary computations; only summary results and `variables` will be in the output data frame.

Basic Statistics

`cor.test(x, y)` correlation test; default method is pearson, use `method = "spearman"` to specify spearman rank correlation, can also use `"kendall"`; alternative syntax useful for data frames: `cor.test(~ x + y, data = mydata)`

`t.test(y, mu=0)` one-sample t-test with null hypothesis that mean is 0

`t.test(y ~ x, data=mydata)` independent-samples t-test where `y` is the response and `x` is the grouping variable.

`t.test(y1, y2)` independent-samples t-test to compare the means of `y1` and `y2`; use `paired = TRUE` for a paired-samples t-test.

`aov(y ~ A, data = mydata)` one-way ANOVA

`aov(y ~ A + x, data = mydata)` ANCOVA for factor A and covariate x

`aov(y ~ A + B + A:B, data = mydata)` full two-way ANOVA; can also use A*B in formula to specify both main effects and interaction

`aov(y ~ A*B + Error(Subject/(A*B)), data = mydata)` two-way within-subject ANOVA

`aov(y ~ W*B + Error(Subject/W), data = mydata)` mixed ANOVA for within-subject factor W and between-subject factor B

NOTE: `aov(...)` will return a model fit object and print ANOVA diagnostics, to get ANOVA table use **`summary(aov(...))`** or **`anova(aov(...))`**

Regression and Model Fitting

`lm(y ~ x1 + x2 + x3, data=mydata)` basic multiple linear regression

`glm(y ~ x, data=mydata, family="binomial")` basic logistic regression for binary variable `y`; use **`glm(cbind(Y,N) ~ x, data=mydata, family="binomial")`** for logistic regression on counts where Y is the number of "successes" and N is the number of "failures"

`fitted(m)` returns predicted values from model `m`

`summary(m)` prints a useful model summary, including parameter estimates (with SE and t-tests) and R^2 values

`anova(m1, m2)` compare fits of nested models (i.e., stepwise regression test)

`lmer(y ~ x+(1|Item)+(1|Subject),data=mydata)` `{lme4}` multi-level regression with random effects of **`Item`** and **`Subject`**; for multilevel regression with random effect of `Subject` on slope, use **`lmer(y ~ x + (x | Subject), data=mydata)`**

Plotting with ggplot2

The **`ggplot`** command establishes the base "aesthetics" of the plot, then the rest of the plot aspects (type of "geom", axis labels, etc.) are added using the + operator. Examples:

```
ggplot(data, aes(x,y)) + geom_boxplot()
ggplot(data, aes(x,y,color=z)) + geom_line()
ggplot(data, aes(x,y)) +
   stat_summary(fun.y="mean", geom="line") +
   facet_wrap(~z)
```

`ggplot(data, aes(x,y))` set up a plot of data with **x** on the horizontal and **y** on the vertical; to specify mappings for color, shape, linetype, size, etc. use **`color= , shape= , linetype= , size=`** , etc. in `aes(…)`

`geom_line(), geom_point(), geom_bar(), geom_boxplot(), geom_errorbar(), geom_pointrange(), geom_tile()` most useful geoms; each has unique aesthetics that must/can be specified

`stat_summary(fun=, geom=)` Summarize y values at every unique x; `geom` specifies the resulting plot type (line, point, pointrange, etc.); `fun` is the summary function, use `fun.y` for single-element summaries (`mean, median`, etc.) and `fun.data` for range summaries (`mean_se`, etc.; Note: some of these require the `Hmisc` package); can also include additional options such as object size.

`plotmatrix(data)` makes a grid of scatterplots for each pair of columns in **`data`**; the diagonal contains a density plot for each column

`facet_wrap(~ f)` create a wrapped ribbon of panels with subsets of the dataset in different panels; `f` is the subsetting factor; use optional arguments `nrow=` or `ncol=` to specify number of rows or columns

`facet_grid(rows ~ columns)` create a grid of panels with subsets of the dataset in different panels; `rows` is the subsetting factor for rows, `columns` is the subsetting factor for columns

`labs(x=, y=, ...)` set labels for x and y axes; can also be used to set labels for other aesthetics (color, shape, etc.)

`scale_[]_manual(values=c(...))` set mapping for a particular scale (replace [] with name of scale); useful for overriding default scales of, e.g., shape or color; Example: `scale_color_manual(values=c("black", "red", "blue"))`

`theme_bw()` a higher contrast display theme; use `base_size=` to set base font size; in addition to built-in themes, new themes can be defined

`theme(...)` set options/theme elements for a single plot; ex: `+theme(strip.background=element_rect(fill=NA, color="white"))` will remove the grey fill and black outline from the title strip

`ggsave(file, plot)` save `plot`, `default is last_plot()` as an image file; file format is determined by the file extension, such as pdf, tiff, png, etc; use `height`, `width`, and `dpi` options to customize

`pdf()` open a pdf file for graphics output, all subsequent plots will be written to separate pages in the file; use **`file="filename"`** to specify filename (default is Rplots.pdf); use **`dev.off()`** to stop writing to the file.

Programming

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

`ifelse(test, yes, no)` returns a value with the same shape as `test` filled with elements from `yes` where `test` is TRUE and elements from `no` where `test` is FALSE.

`for(var in seq) { expr }`

`while(cond) { expr }`

`do.call(funname, args)` executes function `funname` with arguments `args`

`funname <- function( arglist ) { expr return(value) }` creates a function called `funname` that takes arguments `arglist`, executes `expr`, and returns `value`; arguments can be made optional by specifying default values in `arglist` using `opt.arg1=default.value`

`source("filename")` reads and runs all of the commands in a file