An R Programming Quick Reference

Created by Ross Ihaka
for your edification and pleasure

"Share and Enjoy"

Basic Data Representation

`TRUE`, `FALSE`	logical true and false
`1`, `2.5`, `117.333`	simple numbers
`1.23e20`	scientific notation, 1.23×10^{20}.
`3+4i`	complex numbers
`"hello, world"`	a character string
`NA`	missing value (in any type of vector)
`NULL`	missing value indicator in lists
`NaN`	not a number
`Inf`	positive infinity
`-Inf`	negative infinity
‘*var*‘	quotation for special variable name (e.g. `+`, `%*%`, etc.)

Creating Vectors

`c(`$a_1, \ldots, a_n$`)`	combine into a vector
`logical(n)`	logical vector of length `n` (containing falses)
`numeric(n)`	numeric vector of length `n` (containing zeros)
`complex(n)`	complex vector of length `n` (containing zeros)
`character(n)`	character vector of length `n` (containing empty strings)

Creating Lists

`list(`$e_1, \ldots, e_k$`)`	combine as a list
`vector(k, "list")`	create a list of length `k` (the elements are all `NULL`)

Basic Vector and List Properties

`length(x)`	the number of elements in `x`
`mode(x)`	the mode or type of `x`

Tests for Types

`is.logical(x)`	true for logical vectors
`is.numeric(x)`	true for numeric vectors
`is.complex(x)`	true for complex vectors
`is.character(x)`	true for character vectors
`is.list(x)`	true for lists
`is.vector(x)`	true for both lists and vectors

Tests for Special Values

`is.na(x)`	true for elements which are `NA` or `NaN`
`is.nan(x)`	true for elements which are `NaN`
`is.null(x)`	tests whether `x` is `NULL`
`is.finite(x)`	true for finite elements (i.e. not `NA`, `NaN`, `Inf` or `-Inf`)
`is.infinite(x)`	true for elements equal to `Inf` or `-Inf`

Explicit Type Coercion

`as.logical(x)`	coerces to a logical vector
`as.numeric(x)`	coerces to a numeric vector
`as.complex(x)`	coerces to a complex vector
`as.character(x)`	coerces to a character vector
`as.list(x)`	coerces to a list
`as.vector(x)`	coerces to a vector (lists remain lists)
`unlist(x)`	converts a list to a vector

Vector and List Names

`c(`n_1`=`e_1`,...,`n_k`=`e_k`)`	combine as a named vector
`list(`n_1`=`e_1`,...,`n_k`=`e_k`)`	combine as a named list
`names(x)`	extract the names of `x`
`names(x) = v`	(re)set the names of `x` to `v`
`names(x) = NULL`	remove the names from `x`

Vector Subsetting

`x[1:5]`	select elements by index
`x[-(1:5)]`	exclude elements by index
`x[c(TRUE, FALSE)]`	select elements corresponding to `TRUE`
`x[c("a", "b")]`	select elements by name

List Subsetting

`x[1:5]`	extract a *sublist* of the list `x`
`x[-(1:5)]`	extract a *sublist* by excluding elements
`x[c(TRUE, FALSE)]`	extract a *sublist* with logical subscripts
`x[c("a", "b")]`	extract a *sublist* by name

Extracting Elements from Lists

`x[[2]]`	extract an *element* of the list `x`
`x[["a"]]`	extract the *element* with name `"a"` from `x`
`x$a`	extract the *element* with name name `"a"` from `x`

Logical Selection

`ifelse(cond, yes, no)`	conditionally select elements from `yes` and `no`
`which(v)`	returns the indices of `TRUE` values in `v`

Sequences and Repetition

`a:b`	sequence from `a` to `b` in steps of size 1
`seq(n)`	same as `1:n`
`seq(a,b)`	same as `a:b`
`seq(a,b,by=s)`	`a` to `b` in steps of size `s`
`seq(a,b,length=n)`	sequence of length `n` from `a` to `b`
`seq(along=x)`	like `1:length(n)`, but works when `x` has zero length
`rep(x,n)`	`x`, repeated `n` times
`rep(x,v)`	elements of `x` with `x[i]` repeated `v[i]` times
`rep(x,each=n)`	elements of `x`, each repreated `n` times

Sorting and Ordering

`sort(x)`	sort into ascending order
`sort(x, decreasing=TRUE)`	sort into descending order
`rev(x)`	reverse the elements in x
`order(x)`	get the ordering permutation for `x`

Basic Arithmetic Operations

`x + y`	addition, "x plus y"
`x - y`	subtraction, "x minus y"
`x * y`	multiplication, "x times y"
`x / y`	division, "x divided by y"
`x ^ y`	exponentiation, "x raised to power y"
`x %% y`	remainder, "x modulo y"
`x %/% y`	integer division, "x divided by y, discard fractional part"

Rounding

`round(x)`	round to nearest integer
`round(x,d)`	round `x` to `d` decimal places
`signif(x,d)`	round `x` to `d` significant digits
`floor(x)`	round down to next lowest integer
`ceiling(x)`	round up to next highest integer

Common Mathematical Functions

`abs(x)`	absolute values
`sqrt(x)`	square root
`exp(x)`	exponential functiopn
`log(x)`	natural logarithms (base e)
`log10(x)`	common logarithms (base 10)
`log2(x)`	base 2 logarithms
`log(x,base=b)`	base `b` logarithms

Trigonometric and Hyperbolic Functions

`sin(x)`, `cos(x)`, `tan(x)`	trigonometric functions
`asin(x)`, `acos(x)`, `atan(x)`	inverse trigonometric functions
`atan2(x,y)`	arc tangent with two arguments
`sinh(x)`, `cosh(x)`, `tanh(x)`	hyperbolic functions
`asinh(x)`, `acosh(x)`, `atanh(x)`	inverse hyperbolic functions

Combinatorics

`choose(n, k)`	binomial coefficients
`lchoose(n, k)`	log binomial coefficients
`factorial(x)`	factorials
`lfactorial(x)`	log factorials

Special Mathematical Functions

`beta(x,y)`	the beta function
`lbeta(x,y)`	the log beta function
`gamma(x)`	the gamma function
`lgamma(x)`	the log gamma function
`psigamma(x,deriv=0)`	the psigamma function
`digamma(x)`	the digamma function
`trigamma(x)`	the trigamma function

Bessel Functions

`besselI(x,nu)`	Bessel Functions of the first kind
`besselK(x,nu)`	Bessel Functions of the second kind
`besselJ(x,nu)`	modified Bessel Functions of the first kind
`besselY(x,nu)`	modified Bessel Functions of the third kind

Special Floating-Pointl Values

`.Machine$double.xmax`	largest floating point value (1.797693×10^{308})
`.Machine$double.xmin`	smallest floating point value ($2.225074 \times 10^{-308}$)
`.Machine$double.eps`	machine epsilon (2.220446×10^{-16})

Basic Summaries

`sum(`x_1`,`x_2`,...)`	sum of values in arguments
`prod(`x_1`,`x_2`,...)`	product of values in arguments
`min(`x_1`,`x_2`,...)`	minimum of values in arguments
`max(`x_1`,`x_2`,...)`	maximum of values in arguments
`range(`x_1`,`x_2`,...)`	range (minimum and maximum)

Cumulative Summaries

`cumsum(x)`	cumulative sum
`cumprod(x)`	cumulative product
`cummin(x)`	cumulative minimum
`cummax(x)`	cumulative maximum

Parallel Summaries

`pmin(`x_1`,`x_2`,...)`	parallel minimum
`pmax(`x_1`,`x_2`,...)`	parallel maximum

Statistical Summaries

`mean(x)`	mean of elements
`sd(x)`	standard deviation of elements
`var(x)`	variance of elements
`median(x)`	variance of elements
`quantile(x)`	median, quartiles and extremes
`quantile(x, p)`	specified quantiles

Uniform Distribution

`runif(n)`	vector of `n` Uniform[0,1] random numbers
`runif(n,a,b)`	vector of `n` Uniform[`a`,`b`] random numbers
`punif(x,a,b)`	distribution function of Uniform[`a`,`b`]
`qunif(x,a,b)`	inverse distribution function of Uniform[`a`,`b`]
`dunif(x,a,b)`	density function of Uniform[`a`,`b`]

Binomial Distribution

`rbinom(n,size,prob)`	a vector of `n` Binomial(`size`,`prob`) random numbers
`pbinom(x,size,prob)`	Binomial(`size`,`prob`) distribution function
`pbinom(x,size,prob)`	Binomial(`size`,`prob`) inverse distribution function
`pbinom(x,size,prob)`	Binomial(`size`,`prob`) density function

Normal Distribution

`rnorm(n)`	a vector of `n` $N(0,1)$ random numbers
`pnorm(x)`	$N(0,1)$ distribution function
`qnorm(x)`	$N(0,1)$ inverse distribution function
`dnorm(x)`	$N(0,1)$ density function
`rnorm(n,mean,sd)`	a vector of `n` normal random numbers with given mean and s.d.
`pnorm(x,mean,sd)`	normal distribution function with given mean and s.d.
`qnorm(x,mean,sd)`	normal inverse distribution function with given mean and s.d.
`dnorm(x,mean,sd)`	normal density function with given mean and s.d.

Chi-Squared Distribution

`rchisq(n,df)`	a vector of `n` χ^2 random numbers with degrees of freedom `df`
`pchisq(x,df)`	χ^2 distribution function with degrees of freedom `df`
`qchisq(x,df)`	χ^2 inverse distribution function with degrees of freedom `df`
`dchisq(x,df)`	χ^2 density function with degrees of freedom `df`

t Distribution

`rt(n,df)`	a vector of `n` t random numbers with degrees of freedom `df`
`pt(x,df)`	t distribution function with degrees of freedom `df`
`qt(x,df)`	t inverse distribution function with degrees of freedom `df`
`dt(x,df)`	t density function with degrees of freedom `df`

F Distribution

`rf(n,df1,df2)`	a vector of `n` F random numbers with degrees of freedom `df1` & `df2`
`pf(x,df1,df2)`	F distribution function with degrees of freedom `df1` & `df2`
`qf(x,df1,df2)`	F inverse distribution function with degrees of freedom `df1` & `df2`
`df(x,df1,df2)`	F density function with degrees of freedom `df1` & `df2`

Matrices

`matrix(x, nr=`r`, nc=`c`)`	create a matrix from `x` (column major order)
`matrix(x, nr=`r`, nc=`c`, byrow=TRUE)`	create a matrix from `x` (row major order)

Matrix Dimensions

`nrow(x)`	number of rows in `x`
`ncol(x)`	number of columns in `x`
`dim(x)`	vector coltaining `nrow(x)` and `ncol(x)`

Row and Column Indices

`row(x)`	matrix of row indices for matrix `x`
`col(x)`	matrix of column indices for matrix `x`

Naming Rows and Columns

`rownames(x)`	get the row names of `x`
`rownames(x) = v`	set the row names of `x` to `v`
`colnames(x)`	get the column names of `x`
`colnames(x) = v`	set the column names of `x` to `v`
`dimnames(x)`	get both row and column names (in a list)
`dimnames(x) = list(`rn`,`cn`)`	set both row and column names

Binding Rows and Columns

`rbind(`v_1`,`v_2`,...)`	assemble a matrix from rows
`cbind(`v_1`,`v_2`,...)`	assemble a matrix from columns
`rbind(`n_1`=`v_1`,`n_2`=`v_2`,...)`	assemble by rows, specifying row names
`cbind(`n_2`=`v_1`,`n_2`=`v_2`,...)`	assemble by columns, specifying column names

Matrix Subsets

`x[i,j]`	submatrix, rows and columns specified by `i` and `j`
`x[i,j] = v`	reset a submatrix, rows and columns specified by `i` and `j`
`x[i,]`	submatrix, contains just the rows a specified by `i`
`x[i,] = v`	reset specified rows of a matrix
`x[,j]`	submatrix, contains just the columns specified by `j`
`x[,j] = v`	reset specified columns of a matrix
`x[i]`	subset as a vector
`x[i] = v`	reset elements (treated as a vector operation)

Matrix Diagonals

`diag(A)`	extract the diagonal of the matrix `A`
`diag(v)`	diagonal matrix with elements in the vector `v`
`diag(n)`	the `n`×`n` identity matrix

Applying Summaries over Rows and Columns

`apply(X,1,fun)`	apply `fun` to the rows of `X`
`apply(X,2,fun)`	apply `fun` to the columns of `X`

Basic Matrix Manipulation

`t(A)`	matrix transpose
`A %*% B`	matrix product
`outer(u, v)`	outer product of vectors
`outer(u, v, f)`	generalised outer product

Linear Equations

`solve(A, b)`	solve a system of linear equations
`solve(A, B)`	same, with multiple right-hand sides
`solve(A)`	invert the square matrix `A`

Matrix Decompositions

`chol(A)`	the Choleski decomposition
`qr(A)`	the QR decomposition
`svd(A)`	the singular-value decomposition
`eigen(A)`	eigenvalues and eigenvectors

Least-Squares Fitting

`lsfit(X,y)`	least-squares fit with carriers `X` and response `y`

Factors and Ordered Factors

`factor(x)`	create a factor from the values in `x`
`factor(x,levels=l)`	create a factor with the given level set
`ordered(x)`	create an ordered factor with the given level set
`is.factor(x)`	true for factors and ordered factors
`is.ordered(x)`	true for ordered factors
`levels(x)`	the levels of a factor or ordered factor
`levels(x) = v`	reset the levels of a factor or ordered factor

Tabulation and Cross-Tabulation

`table(x)`	tabulate the values in `x`
`table(`f_1`,`f_2`,...)`	cross tabulation of factors

Summary over Factor Levels

`tapply(x,f,fun)`	apply summary `fun` to `x` broken down by `f`
`tapply(x,list(`f_1`,`f_2`,...),fun)`	apply summary `fun` to `x` broken down by several factors

Data Frames

`data.frame(`n_1`=`x_1`,`n_2`=`x_2`,...)`	create a data frame
`row.names(df)`	extract the observation names from a data frame
`row.names(df) = v`	(re)set the observation names of a data frame
`names(df)`	extract the variable names from a data frame
`names(df)` = v	(re)set the variable names of a data frame

Subsetting and Transforming Data Frames

`df[i,j]`	matrix subsetting of a data frame
`df[i,j] = dfv`	reset a subset of a data frame
`subset(df,subset=i)`	subset of the cases of a data frame
`subset(df,select=i)`	subset of the variables of a data frame
`subset(df,subset=i,select=j)`	subset of the cases and variables of a data frame
`transform(df,`n_1`=`e_1`,`n_2`=`e_2`,...)`	transform variables in a data frame
`merge(df1,df2,...)`	merge data frames based on common variables

Compound Expressions

`{` $expr_1$`, ...,` $expr_n$`}`	compound expressions

Alternation

`if (`*cond*`)` $expr_1$ `else` $expr_1$	conditional execution
`if (`*cond*`)` *expr*	conditional execution, no alternative

Iteration

`for (`*var* `in` *vector*`)` *expr*	`for` loops
`while (`*cond*`)` *expr*	`while` loops
`repeat` *expr*	infinite repetition
`continue`	jump to end of enclosing loop
`break`	break out of enclosing loop

Function Definition

`function(`*args*`)` *expr*	function definition
var	function argument with no default
var`=`*expr*	function argument with default value
`return(`*expr*`)`	return the given value from a function
`missing(a)`	true if argument `a` was not supplied

Error Handling

`stop(`*message*`)`	terminate a computation with an error message
`warning(`*message*`)`	issue a warning message
`on.exit(`*expr*`)`	save an expression for execution on function return

Language Computation

`quote(`*expr*`)`	returns the expression *expr* unevaluated
`substitute(`*arg*`)`	returns the expression passed as argument *arg*
`substitute(`*expr*`,`*subs*`)`	make the specified substitutions in the given expression

Interpolation

`approx(x, y, xout)`	linear interpolation at `xout` using `x` and `y`
`spline(x, y, xout)`	spline interpolation at `xout` using `x` and `y`
`approxfun(x, y, xout)`	interpolating linear function for `x` and `y`
`splinefun(x, y, xout)`	interpolating spline for `x` and `y`

Root-Finding and Optimisation

`polyroot(coef)`	roots of polynomial with coefficients in `coef`
`uniroot(f,interval)`	find a root of the function `f` in the given interval
`optimize(f,interval)`	find an extreme of the function `f` in the given interval
`optim(x,f)`	find an extreme of the function `f` starting at the point `x`
`nlm(f,x)`	and alternative to `optim`
`nlminb(x,f)`	optimization subject to constraints

Integration

`integrate(x,lower,upper)`	integrate the function `f` from `lower` to `upper`