

An R Programming Quick Reference

“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

$c(e_1, \dots, e_n)$	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, \dots, e_k)	combine as a list
vector("list", k)	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

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

Explicit Type Coercion

<code>as.logical(x)</code>	coerces to a logical vector
<code>as.numeric(x)</code>	coerces to a numeric vector
<code>as.complex(x)</code>	coerces to a complex vector
<code>as.character(x)</code>	coerces to a character vector
<code>as.list(x)</code>	coerces to a list
<code>as.vector(x)</code>	coerces to a vector (lists remain lists)
<code>unlist(x)</code>	converts a list to a vector

Vector and List Names

<code>c($n_1=e_1, \dots, n_k=e_k$)</code>	combine as a named vector
<code>list($n_1=e_1, \dots, n_k=e_k$)</code>	combine as a named list
<code>names(x)</code>	extract the names of x
<code>names(x) = v</code>	(re)set the names of x to v
<code>names(x) = NULL</code>	remove the names from x

Vector Subsetting

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

List Subsetting

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

Extracting Elements from Lists

<code>x[[2]]</code>	extract an <i>element</i> of the list x
<code>x[["a"]]</code>	extract the <i>element</i> with name "a" from x
<code>x\$a</code>	extract the <i>element</i> with name "a" from x

Logical Selection

<code>ifelse(cond, yes, no)</code>	conditionally select elements from yes and no
<code>replace(x, list, values)</code>	replace selected values in x with those in values.
<code>which(v)</code>	returns the indices of TRUE values in v

List Manipulation

<code>lapply(X, FUN, ...)</code>	apply FUN to the elements of X
<code>split(x, f)</code>	split x using the factor f

Sequences and Repetition

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

Sorting and Ordering

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

Basic Arithmetic Operations

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

Rounding

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

Common Mathematical Functions

<code>abs(x)</code>	absolute values
<code>sqrt(x)</code>	square root
<code>exp(x)</code>	exponential function
<code>log(x)</code>	natural logarithms (base e)
<code>log10(x)</code>	common logarithms (base 10)
<code>log2(x)</code>	base 2 logarithms
<code>log(x,base=b)</code>	base b logarithms

Trigonometric and Hyperbolic Functions

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

Combinatorics

<code>choose(n, k)</code>	binomial coefficients
<code>lchoose(n, k)</code>	log binomial coefficients
<code>factorial(x)</code>	factorials
<code>lfactorial(x)</code>	log factorials

Special Mathematical Functions

<code>beta(x,y)</code>	the beta function
<code>lbeta(x,y)</code>	the log beta function
<code>gamma(x)</code>	the gamma function
<code>lgamma(x)</code>	the log gamma function
<code>psigamma(x,deriv=0)</code>	the psigamma function
<code>digamma(x)</code>	the digamma function
<code>trigamma(x)</code>	the trigamma function

Bessel Functions

<code>besselI(x,nu)</code>	Bessel Functions of the first kind
<code>besselK(x,nu)</code>	Bessel Functions of the second kind
<code>besselJ(x,nu)</code>	modified Bessel Functions of the first kind
<code>besselY(x,nu)</code>	modified Bessel Functions of the third kind

Special Floating-Point Values

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

Basic Summaries

<code>sum(x_1, x_2, \dots)</code>	sum of values in arguments
<code>prod(x_1, x_2, \dots)</code>	product of values in arguments
<code>min(x_1, x_2, \dots)</code>	minimum of values in arguments
<code>max(x_1, x_2, \dots)</code>	maximum of values in arguments
<code>range(x_1, x_2, \dots)</code>	range (minimum and maximum)

Cumulative Summaries

<code>cumsum(x)</code>	cumulative sum
<code>cumprod(x)</code>	cumulative product
<code>cummin(x)</code>	cumulative minimum
<code>cummax(x)</code>	cumulative maximum

Parallel Summaries

<code>pmin(x_1, x_2, \dots)</code>	parallel minimum
<code>pmax(x_1, x_2, \dots)</code>	parallel maximum

Logical Summaries

<code>all(x)</code>	are all of the elements in <code>x</code> true?
<code>any(x)</code>	are any of the elements in <code>x</code> true?

Statistical Summaries

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

Uniform Distribution

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

Binomial Distribution

<code>rbinom(n,size,prob)</code>	a vector of n Binomial(size,prob) random numbers
<code>pbinom(x,size,prob)</code>	Binomial(size,prob) distribution function
<code>qbinom(x,size,prob)</code>	Binomial(size,prob) inverse distribution function
<code>dbinom(x,size,prob)</code>	Binomial(size,prob) density function

Normal Distribution

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

Chi-Squared Distribution

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

t Distribution

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

F Distribution

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

Matrices

<code>matrix(x, nr=r, nc=c)</code>	create a matrix from <code>x</code> (column major order)
<code>matrix(x, nr=r, nc=c, byrow=TRUE)</code>	create a matrix from <code>x</code> (row major order)

Matrix Dimensions

<code>nrow(x)</code>	number of rows in <code>x</code>
<code>ncol(x)</code>	number of columns in <code>x</code>
<code>dim(x)</code>	vector containing <code>nrow(x)</code> and <code>ncol(x)</code>

Row and Column Indices

<code>row(x)</code>	matrix of row indices for matrix <code>x</code>
<code>col(x)</code>	matrix of column indices for matrix <code>x</code>

Naming Rows and Columns

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

Binding Rows and Columns

<code>rbind(v₁,v₂,...)</code>	assemble a matrix from rows
<code>cbind(v₁,v₂,...)</code>	assemble a matrix from columns
<code>rbind(n₁=v₁,n₂=v₂,...)</code>	assemble by rows, specifying row names
<code>cbind(n₂=v₁,n₂=v₂,...)</code>	assemble by columns, specifying column names

Matrix Subsets

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

Matrix Diagonals

<code>diag(A)</code>	extract the diagonal of the matrix <code>A</code>
<code>diag(v)</code>	diagonal matrix with elements in the vector <code>v</code>
<code>diag(n)</code>	the $n \times n$ identity matrix

Applying Summaries over Rows and Columns

<code>apply(X,1,fun)</code>	apply <code>fun</code> to the rows of <code>X</code>
<code>apply(X,2,fun)</code>	apply <code>fun</code> to the columns of <code>X</code>

Basic Matrix Manipulation

<code>t(A)</code>	matrix transpose
<code>A %*% B</code>	matrix product
<code>outer(u, v)</code>	outer product of vectors
<code>outer(u, v, f)</code>	generalised outer product

Linear Equations

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

Matrix Decompositions

<code>chol(A)</code>	the Choleski decomposition
<code>qr(A)</code>	the QR decomposition
<code>svd(A)</code>	the singular-value decomposition
<code>eigen(A)</code>	eigenvalues and eigenvectors

Least-Squares Fitting

<code>lsfit(X,y)</code>	least-squares fit with carriers X and response y
-------------------------	--

Factors and Ordered Factors

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

Tabulation and Cross-Tabulation

<code>table(x)</code>	tabulate the values in x
<code>table(f₁, f₂, ...)</code>	cross tabulation of factors

Summary over Factor Levels

<code>tapply(x, f, fun)</code>	apply summary fun to x broken down by f
<code>tapply(x, list(f₁, f₂, ...), fun)</code>	apply summary fun to x broken down by several factors

Data Frames

<code>data.frame(n₁=x₁, n₂=x₂, ...)</code>	create a data frame
<code>row.names(df)</code>	extract the observation names from a data frame
<code>row.names(df) = v</code>	(re)set the observation names of a data frame
<code>names(df)</code>	extract the variable names from a data frame
<code>names(df) = v</code>	(re)set the variable names of a data frame

Subsetting and Transforming Data Frames

<code>df[i, j]</code>	matrix subsetting of a data frame
<code>df[i, j] = dfv</code>	reset a subset of a data frame
<code>subset(df, subset=i)</code>	subset of the cases of a data frame
<code>subset(df, select=i)</code>	subset of the variables of a data frame
<code>subset(df, subset=i, select=j)</code>	subset of the cases and variables of a data frame
<code>transform(df, n₁=e₁, n₂=e₂, ...)</code>	transform variables in a data frame
<code>merge(df1, df2, ...)</code>	merge data frames based on common variables

Matching

<code>match(x, table)</code>	find matches for x in table
<code>x %in% table</code>	binary function version of match
<code>pmatch(x, table)</code>	find partial matches for x in table

Set Operations

<code>union(x, y)</code>	union
<code>intersect(x, y)</code>	intersection
<code>setdiff(x, y)</code>	difference
<code>setequal(x, y)</code>	equality

String Handling

<code>nchar(x)</code>	compute string length
<code>paste(..., sep = " ", collapse = NULL)</code>	paste strings together
<code>strsplit(x, split)</code>	split x on pattern <code>split</code>
<code>grep(pattern, x)</code>	return subscripts of matching elements
<code>grep(pattern, x, value = TRUE)</code>	return matching elements
<code>grepl(pattern, x)</code>	do elements match (logical)
<code>sub(pattern, replacement, x)</code>	replace pattern with given replacement
<code>gsub(pattern, replacement, x)</code>	globally replace
<code>substr(x, start, stop)</code>	extract substrings
<code>substring(x, first, last)</code>	extract substrings

Reading Lines

<code>readline(prompt="")</code>	read a line of input
<code>readLines(file, n)</code>	read n lines from the specified file
<code>readLines(file)</code>	read all lines from the specified file

Reading Vectors and Lists

<code>scan(file, what = numeric())</code>	read a vector or list from a file
---	-----------------------------------

Formatting and Printing

<code>format(x)</code>	format a vector in a common format
<code>sprintf(fmt, ...)</code>	formatted printing of R objects
<code>cat(...)</code>	concatenate and print vectors
<code>print(x)</code>	print an R object

Reading Data Frames

<code>read.table(file, header=FALSE)</code>	read a data frame from a file
<code>read.csv(file, header=FALSE)</code>	read a data frame from a csv file

Options for `read.table` and `read.csv`

<code>header=true/false</code>	does first line contain variable names?
<code>row.names=...</code>	row names specification
<code>col.names=...</code>	variable names specification
<code>na.strings="NA"</code>	entries indicating NA values
<code>colClasses=NA</code>	the types associated with columns
<code>nrows=...</code>	the number of rows to be read

Writing Data Frames

<code>write.table(x, file)</code>	write a data frame to a file
<code>write.csv(x, file)</code>	write a data frame to a csv file

Writing Data Frames

<code>write.table(x, file)</code>	write a data frame to a file
<code>write.csv(x, file)</code>	write a data frame to a csv file

High-Level Graphics

<code>plot(x, y)</code>	scatter plot
<code>plot(x, y, type = "l")</code>	line plot
<code>plot(x, y, type = "n")</code>	empty plot

Adding to Plots

<code>abline(a, b)</code>	line in intercept/slope form
<code>abline(h = yvals)</code>	horizontal lines
<code>abline(v = xvals)</code>	vertical lines
<code>points(x, y)</code>	add points
<code>lines(x, y)</code>	add connected polyline
<code>segments(x0, y0, x1, y1)</code>	add disconnected line segments
<code>arrows(x0, y0, x1, y1, code)</code>	add arrows
<code>rect(x0, y0, x1, y1)</code>	add rectangles
<code>polygon(x, y)</code>	a polygon(s)

Low-Level Graphics

<code>plot.new()</code>	start a new plot/figure/panel
<code>plot.window(xlim, ylim, ...)</code>	set up plot coordinates

Options to `plot.window`

<code>xaxs="i"</code>	don't expand x range by 8%
<code>yaxs="i"</code>	don't expand y range by 8%
<code>asp=1</code>	equal-scale x and y axes

Graphical Parameters

<code>par(...)</code>	set/get graphical parameters
-----------------------	------------------------------

Useful Graphical Parameters

<code>mfrow = c(m, n)</code>	set up an m by n array of figures, filled by row
<code>mfcol = c(m, n)</code>	set up an m by n array of figures, filled by column
<code>mar=c(m₁, m₂, m₃, m₄)</code>	set the plot margins (in lines)
<code>mai=c(m₁, m₂, m₃, m₄)</code>	set the plot margins (in inches)
<code>cex=m</code>	set the basic font magnification to m
<code>bg=col</code>	set the device background to col

Layouts

<code>layout(mat, heights, widths)</code>	set up a layout
<code>layout.show(n)</code>	show layout elements (up to n)
<code>lcm(x)</code>	size specification in cm

Compound Expressions

`{ expr1, ..., exprn}` compound expressions

Alternation

`if (cond) expr1 else expr1` 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

<code>approx(x, y, xout)</code>	linear interpolation at <code>xout</code> using <code>x</code> and <code>y</code>
<code>spline(x, y, xout)</code>	spline interpolation at <code>xout</code> using <code>x</code> and <code>y</code>
<code>approxfun(x, y, xout)</code>	interpolating linear function for <code>x</code> and <code>y</code>
<code>splinefun(x, y, xout)</code>	interpolating spline for <code>x</code> and <code>y</code>

Root-Finding and Optimisation

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

Integration

<code>integrate(x, lower, upper)</code>	integrate the function <code>f</code> from <code>lower</code> to <code>upper</code>
---	---