#removing everything in your environment:

rm(list=ls())

The stop() and stopifnot() functions will stop a function if it has a dodgy value

in it

The browser function, when inserted into another function, allows you to explore the workspace inside the function as it is being run.

reading in data:

read.table(file = "/Users/jojo/Documents/R ucd/3/row.txt", header = FALSE, sep = ",")

# when current folder has been set as working directory (from More settings in the Files tab – bottom right)

mydata <- read.table(file = "row3.txt", header = TRUE, sep = ",", row.names = 1)

read.csv

read\_excel (readxl package)

summary()

str()

getwd()

setwd()

mode()

typeof() # for use in code

runif(x, min = 0, max = 1, log = FALSE)

sum()

help()

help.search('standard deviation')

eample()

demo()

mean()

sd(x, na.rm = FALSE)

t()

complete.cases()

plot(x, y)

hist()

vector(mode = "logical", length = 0)

seq(from = 1, to = 1, by = ((to - from)/(length.out - 1)), length.out = NULL, along.with = NULL, ...)

rep()

length()

colnames()

rownames()

names() # gives names of components of an object

all(..., na.rm = FALSE)

any(..., na.rm = FALSE)

ifelse(test, yes, no)

sqrt()

subset(x, ...)

which(x, arr.ind = FALSE, useNames = TRUE) #gives indexes of values!

matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)

array(data = NA, dim = length(data), dimnames = NULL)

list()

apply(m, dimcode, f, fargs)

lapply(x, function) #for lists

sapply(x, function)

cumsum(x) # cumulative sum

diag()

det()

solve(a, b, ...)

eigen(x, symmetric, only.values = FALSE, EISPACK = FALSE)

functions to define a vector:

y <- vector(length = 2)

y2 <- vector("numeric", length = 2)

y3 <- numeric(2)

data.frame(..., row.names = NULL, check.rows = FALSE, check.names = TRUE, fix.empty.names = TRUE, stringsAsFactors = default.stringsAsFactors())

kids <- **c**('Jack', 'Jill')

ages <- **c**(12, 10)

d <- data.frame(kids, ages)

names() #gets names of list elements

merge(x, y, by.x=, by.y=) # this will be inner join

merge(x = d1, y = d2, by = "colname", all = TRUE) # this will be outer join

strsplit(x, 'split ') # splits the elements of a character vector x into substrings according to ‘split’

order()

factor(x = character(), levels, labels = levels, exclude = NA, ordered = is.ordered(x), nmax = NA) # turns vector elements into factors, defines levels

tapply(v, g, f) # v is a num vector, g is a vector of factors, f is a function

by(data, INDICES, FUN, ..., simplify = TRUE) # applies function to a data frame split by factors (works the same as tapply but with an object, not a vector

split(v, g) # v is a num vector, g is a vector of factors, gives a list of vectors by group

table() # creates a table

addmargins() # gives marginal sums for a frequency table

aggregate() # calls tapply for each variable in a group, example:

aggregate(birthwt$age, list(birthwt$race), mean)

cut(x) # converts numeric to factor; function for generating a factor from a list of bins.

get(x) # returns a value of a named object

# Next 4 functions form row and column sums and means for numeric arrays and data frames

colSums (x, na.rm = FALSE, dims = 1)

rowSums (x, na.rm = FALSE, dims = 1)

colMeans(x, na.rm = FALSE, dims = 1)

rowMeans(x, na.rm = FALSE, dims = 1)

# Some examples of various maths functions

exp(1) # The value of e

log(10) # Natural log

log10(10) # Log to base 10

log(10, base = 2) # Choosing the base

sqrt(64) # Square root

abs(c(-5, 2)) # Absolute value

sin(pi/2) # Sin, cos, etc

cos(pi/2)

asin(0)

min(1:5) # min and max (pmin and pmax later)

max(1:5)

sum(1:10) # Sum and product

prod(1:10)

cumsum(1:10) # Cumulative sum and product

cumprod(1:10)

round(3.21) # Round to nearest integer

floor(3.21) # Round to lowest nearest integer

ceiling(3.21) # Round to highest nearest integer

signif(12332, digits = 3) # Rounds to specified number of significant digits

pmin and pmax find parallel minima/maxima across multiple vectors or matrices.

D(expression(exp(x^2)), 'x') # derivative

D finds the derivative of a mathematical expression (created by the function expression) with respect to the chosen variable.

integrate(function(x) x^2, 0, 1)

The integrate function will calculate numerical integrals over the provided range.

R contains a large range of functions for evaluating probability distributions. These all follow a set pattern:

– d for density (or pdf) #this gives the graph of the function

– p for cumulative density (or cdf)

– q for quantiles

– r for random number generation

# p above is for probability that you have a value less or equal to a value you specify

Distributions include:

norm (Normal),

binom (Binomial),

chisq (Chi-squared),

f (F-distribution),

pois (Poisson),

t -distribution

#normal distribution

rnorm(100, mean=100, sd=1) #create random numbers

# finding probabilities (as in table 4)

pnorm(-2) # area under curve to the left of -2

qnorm(0.02275013) # when we know the probability and want to find the related z value

# finding z values for percentiles

qnorm(0.90) # gives 90th percentile z value

dnorm(0) # takes values from x axis and gives the corresponding values on the y axis

dnorm(1)

dnorm(-1)

sort(x) # takes a vector and puts it in order

order(x) # gives order of the vector elements (by index)

x <- c(5, 13, 12, 5)

rank(x) # ranks elements, for example here the second element

# 13 is the fourth from the bottom so it was marked as 4

• If you have a set of items stored in a vector, R can compare them using set operations:

– union(x, y): Creates the union set of x and y

– intersect(x, y): Creates the intersection of x and y

– setdiff(x, y): Creates the set of all objects in x that are not in y

– setequal(x, y): Determines if x and y are equal

– c %in% y: Tests whether c is an element of y

• We have already met vector operations and matrix multiplication (%\*%). • There are other useful linear algebra functions:

– solve(): finds a matrix equation solution or matrix inverse

– t(): matrix transpose

– chol(), qr(), eigen(): Cholesky, QR and eigen decomposition

– det(): Determinant

– diag(): Find or set the matrix diagonal

– sweep(): Removes summary statistics from matrices.

3/3 matrix, we add 1 to row 1, 4 to row 2, 7 to row 3:

m <- matrix(1:9, 3, 3, byrow= TRUE)

**sweep**(m, 1, c(1,4,7), '+')

sweep is typically used when you operate a matrix by row or by column, and the other input of the operation is a different value for each row / column. Whether you operate by row or column is defined by MARGIN, as for apply. The values used for what I called "the other input" is defined by STATS. So, for each row (or column), you will take a value from STATS and use in the operation defined by FUN.

curve(expr, from = NULL, to = NULL, n = 101, add = FALSE,

type = "l", xname = "x", xlab = xname, ylab = NULL,

log = NULL, xlim = NULL, ...)

density(x, bw = "nrd0", adjust = 1,

kernel = c("gaussian", "epanechnikov", "rectangular",

"triangular", "biweight",

"cosine", "optcosine"),

weights = NULL, window = kernel, width,

give.Rkern = FALSE,

n = 512, from, to, cut = 3, na.rm = FALSE, ...)

example of use with Old faithful data:

xx <- density(faithful$eruptions)

plot(xx)

# Use of diag

M <- matrix(1:9, 3, 3)

diag(M)

diag(M) <- 7

M <- diag(c(3, 1))

sample(x, size, replace = FALSE, prob = NULL) #draws random sample from x

factor(x = character(), levels, labels = levels,

exclude = NA, ordered = is.ordered(x), nmax = NA)

# adding trend line to graph

abline(a = NULL, b = NULL, h = NULL, v = NULL, reg = NULL,

coef = NULL, untf = FALSE, ...)

a, b

the intercept and slope, single values.

abline(h=mean(Nile)) # adds line at the level of mean

pnorm(1.70) # cum probability that x < 1.70

pnorm(1.70, lower.tail = FALSE) # cum probability that x > 1.70

qnorm(0.98956) # you input propability, and get back quantile (using table in reverse)

curve(expr, from = NULL, to = NULL, n = 101, add = FALSE,

type = "l", xname = "x", xlab = xname, ylab = NULL,

log = NULL, xlim = NULL, ...)

curve(dnorm, from = -4, to = 4)

curve(pnorm, from = -4, to = 4)

curve(qnorm, from = 0, to = 1) #range is probabilties

v <- c(1,1,1,2,4,56,7,4,5,9,11,34,55,7)

quantile(v, 0.20)

w <- c(1,2,3,4,5)

to **standarise** the vector subtract the mean, and divide by sd

(w - mean(w)) / sd(w)

a matrix of scatterplots: can explore relationship between different sets of variables, the position of names of variables telss which axis they are displayed on

pairs(~ arr\_delay + dep\_delay)

pairs(~ distance + dep\_delay + arr\_delay + air\_time)

dim(x) <- value

#Retrieve or set the dimension of an object.

**derivatives and curves of functions and derivatives**

function(x) x^2 \* sin(x)

D(expression(x^2 \* sin(x)), 'x') # derivative

x <- 5 # we want to substitute 5 for x, to evaluate at this value, use eval()

eval(D(expression(x^2 \* sin(x)), 'x'))

curve(x^2 \* sin(x), from= 0, to= 10) # graph of function

curve(eval(D(expression(x^2 \* sin(x)), 'x')), from=0, to= 10, add=T)

# graph the derivative

**integrating pnorm function:**

integrate(dnorm, -1.96, 1.96, mean=2, sd=1)

integrate(dnorm, -Inf, 2, mean=0, sd=1)

**ordering a matrix**

d <- data.frame(kids = c('Jack', 'Jill', 'Billy'), ages = c(12, 10, 13))

d[order(d$kids),]

d[order(d$ages),]

append(x, values, after = length(x))

append(1:10, 99, after=5)

append(1:10, -99, after=0) # appends at the start

use of paste():

plot(Nile, main = 'Nile River Annual Flow', xlab = 'Year', ylab = 'Flow')

abline(h=mean(Nile))

text(x = 1930, y= 1200, adj =0, **paste**('Average flow:', mean(Nile)))

#scans data from files such as text files. file extension must be given

scan(file = "", what = double(), sep = "")

scan('z3.txt', what = 'character') # if mix of char and numeric in the file

scan('z3.txt', what = '', sep = '\n') # keeps lines together

reading a line from console with a prompt

inits <- readline('Type your initials: ')

inits

concatenation function:

cat(... , file = "", sep = " ", fill = FALSE, labels = NULL,

append = FALSE)

cat('The values of x are', x, '\n')

cat('The values of x are', x, '\n', sep = '')

• Note that cat requires a new line character '\n' so that R starts writing on the next line.

• By default cat introduces spaces between the different elements, use sep = '' to stop this happening.

read.table(file, header = FALSE, sep = "", quote = "\"'",

dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),

row.names, col.names, as.is = !stringsAsFactors,

na.strings = "NA", colClasses = NA, nrows = -1,

skip = 0, check.names = TRUE, fill = !blank.lines.skip,

strip.white = FALSE, blank.lines.skip = TRUE,

comment.char = "#",

allowEscapes = FALSE, flush = FALSE,

stringsAsFactors = default.stringsAsFactors(),

fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file. Also:

read.delim(file, header = TRUE, sep = "\t", quote = "\"",

dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",

dec = ",", fill = TRUE, comment.char = "", ...)

All files read in this way need to have the same number of columns in each row but can be of mixed mode in each column.

There are lots of useful extra arguments to the read.XXXX functions, including:

– header which if TRUE sets the column names to be the top row.

– sep which changes the delimiter type (useful in read.table).

– skip which skips out some of the top lines.

– nrows which will read in only a set number of rows.

– colClasses which will set modes for each different column.

– fill which will over-ride the need to have equal numbers of columns for each row.

– stringsAsFactors which will read in character columns as factors (though is over-ridden by colClasses which is more powerful).

using a connection to read a file and read in line by line

y <- file('z5.txt','r') # ‘r’ means read, ‘w’ would be for write

readLines(y, n = 1)

readLines(y, n = 1)

readLines(y, n = 1)

readLines(y, n = 1)

after getting to end we can rewind with seek or close file:

# rewind

seek(con = y, where = 0)

readLines(y, n = 1)

close(y)

# Use of write.table

kids <- c('Jack', 'Jill')

ages <- c(12, 10)

d <- data.frame(kids, ages, stringsAsFactors = FALSE)

write.table(d, file = 'd1.txt')

write.table(d, file = 'd2.txt', quote = FALSE, row.names = FALSE)

# it creates a file in you WD and writes a table in it

similar result as above with cat:

# Use of cat

x <- 2:4

cat('abc\n ', x, file = 'd3.txt', sep = '')

cat(x, 'de\n', x, file = 'd4.txt', append = TRUE)

# Load in some data from the web

prostate.url <-

'https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data'

prostate <- read.table(prostate.url, header = TRUE, sep = '\t', row.names = 1)

head(prostate)

# Use Rcurl to download webpages

library(bitops)

library(RCurl)

# Download the webpage

oct17 <- getURL("https://stat.ethz.ch/pipermail/r-help/2017-October/date.html")

# Sort into lines

webpage <- strsplit(oct17,"\n")[[1]]

webpage[1:5]

finds the specified pattern inside the vector x

grep(pattern, x, ignore.case = FALSE, perl = FALSE, value = FALSE,

fixed = FALSE, useBytes = FALSE, invert = FALSE)

gsub(pattern, replacement, x, ignore.case = FALSE, perl = FALSE,

fixed = FALSE, useBytes = FALSE)

x <- 'South Pole'

gsub('South', 'North', x)

finds the length of the string x. Note that it includes spaces (and has some funny behaviour for non character variables)

nchar('South Pole')

nchar('SouthPole')

extracts the part of the string from value start to value stop

substr(x, start, stop)

substr('Equator', 3, 5)

[1] "Eq"

splits the current string into a list of substrings by the character(s) in split

strsplit('October-17', split = '-')

paste (…) will concatenate several strings together, with the result returned as one long string. The extra argument sep gives the character by which to join them (by default just a space)

paste('North', 'Pole')

paste('North', 'Pole', sep = '')

x <- 'and'

paste('North', x, 'South', 'Pole')

concatenating strings:

sprintf(...) assembles strings from parts in a formatted manner. It is very similar to C functions used to print out strings.

i <- 8

s <- sprintf('the square of %d is %d', i, i^2)

s

finding patterns:

• regexp(pattern, text) gives the character position of the first occurrence of pattern in text, as well as the length of the pattern.

• gregexp(pattern, text) gives all the instances of pattern in text (and also the length of the pattern).

• gsub(pattern, replacement, text) will replace the values in replacement with pattern in object

text

# Example regular expressions

# Find 'a' or 'u' in a vector

grep('[au]', c('Equator', 'North Pole', 'South Pole'))

# Find places where 'o' is followed by a character and then an 'e'

grep('o.e', c('Equator', 'North Pole', 'South Pole'))

# Find places where 'N' is separated by two characters from 't'

grep('N..t', c('Equator', 'North Pole', 'South Pole'))

# What if you want to search for a .?

grep('.', c('abc', 'de', 'f.g'))

# The way it's done

grep('\\.', c('abc', 'de', 'f.g'))

# More examples - states ending in 'ana'

state.name[grep('ana$', state.name)]

# States beginning with 'South'

state.name[grep('^South', state.name)]

# More on regular expressions

?regexp

# It doesn't have to be single characters

words <- c('cat','bat','dog','rabbit','rat')

grep('bat|cat', words)

# bat or cat

words[grep('(b|c)at', words)]

# At least zero of b or c

words[grep('(b\*|c\*)at', words)]

# converting a factor to a numeric (z is a factor):

x <- as.numeric(as.character(z)) #!!!!!!!!!!

# creates a data frame from a list containing vectors as opposed to data.frame() which takes vectors directly

as.data.frame(x, row.names = NULL, optional = FALSE, ...)

# example: collating row based observations into a data frame

do.call(what, args, quote = FALSE, envir = parent.frame())

example:

do.call(rbind,obs)

# appending rows to data frame

newRow <- data.frame(city='West Dundee', county = 'Kane', state = 'IL', pop =

5428)

suburbs <- rbind(suburbs, newRow)

# merging two data frames

merge(x, y, by = intersect(names(x), names(y)),

by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,

sort = TRUE, suffixes = c(".x",".y"), no.dups = TRUE,

incomparables = NULL, ...)

selecting columns in a data frame

fin[[2]] # returns column as a vector

fin[2] # returns column as a data frame with one column