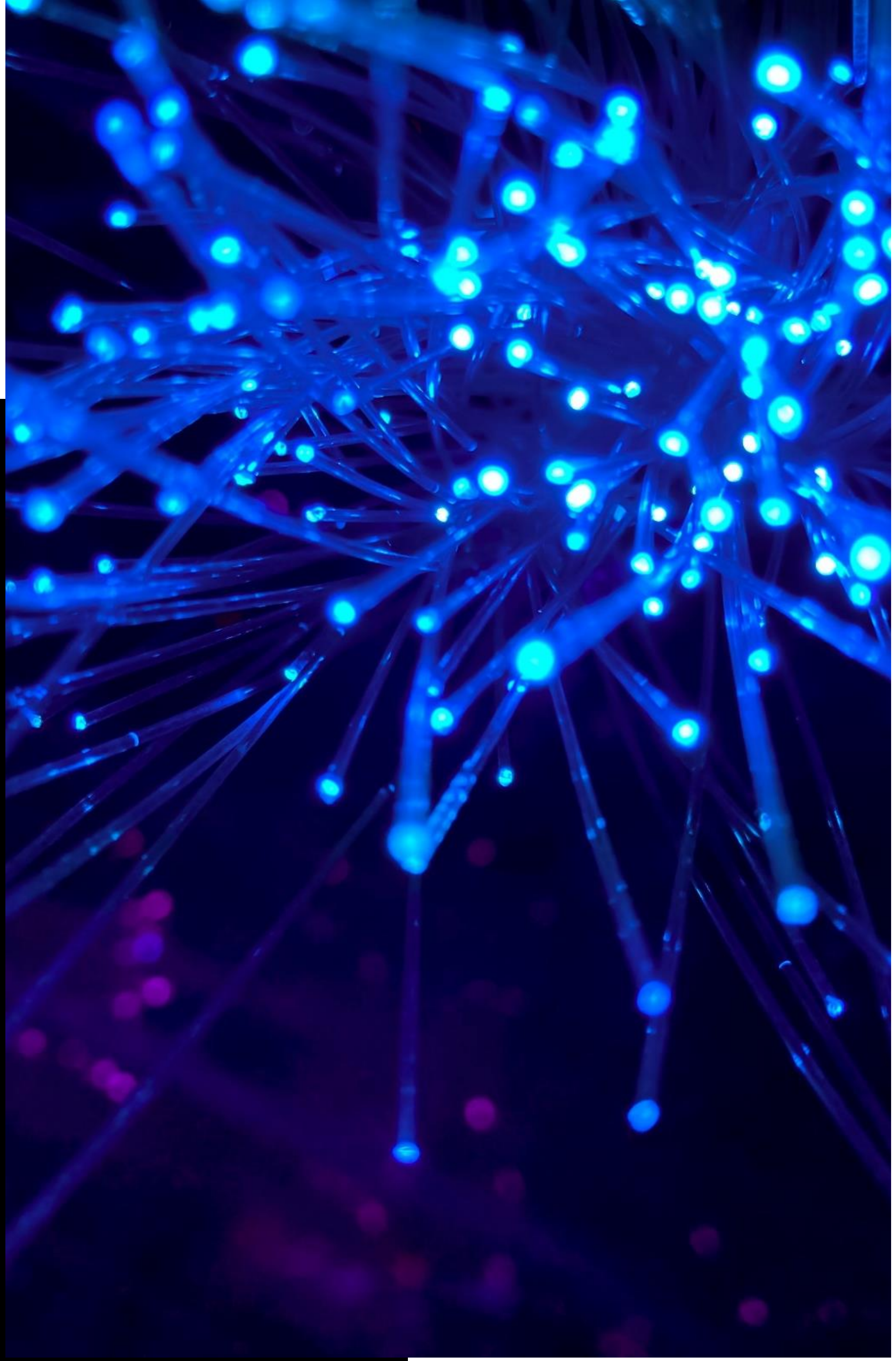
Programación con R

Bioinformática 24-25

Grado en Biomedicina

idelhgar@uax.es



00.- Introducción

¿Qué es R?

- R es un lenguaje de programación de código abierto que cuenta con una alta variedad de técnicas estadísticas y gráficas y admite gran variedad de datos.
- **Es GNU (General Public License).**
- Creado en 1993 por Robert Gentleman y Ross Ihaka. Similar al **entorno S** creado por Bell Laboratories (AT&T, ahora Lucent Technologies) a finales de los 70.
- Modelos de regresión lineal, logísticos, análisis de series de tiempo, pruebas estadísticas clásicas, agrupamientos, clustering, clasificaciones etc etc.
- **Investigación científica, manipulación de datos, análisis estadístico, IA, Machine Learning, técnicas gráficas, modelado y predicciones, matemáticas financieras, bioinformática, investigación biomédica, cálculo numérico….**
- Equivalente en cálculo numérico a MATLAB o a GNU Octave.

Referencias y Manuales de R

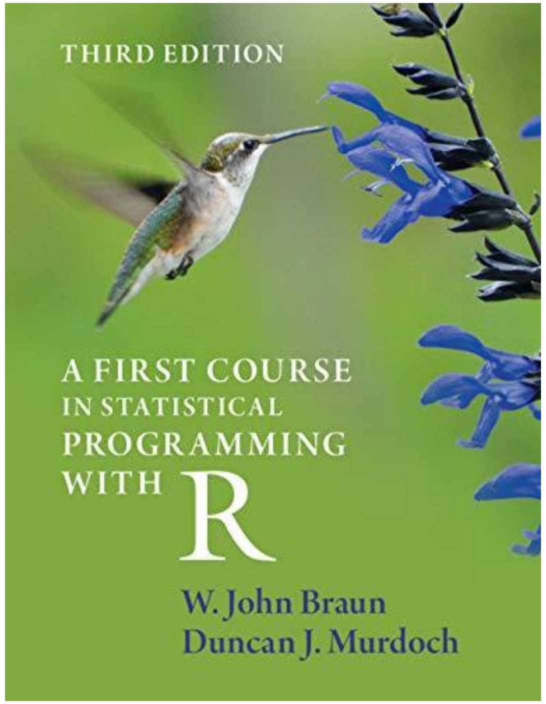
- **W. John Braun y Duncan J. Murdoch. A First Course in Statistical Programming with R. Cambridge University Press, 20 de mayo de 2021. 280 págs. isbn: 1108995144.**

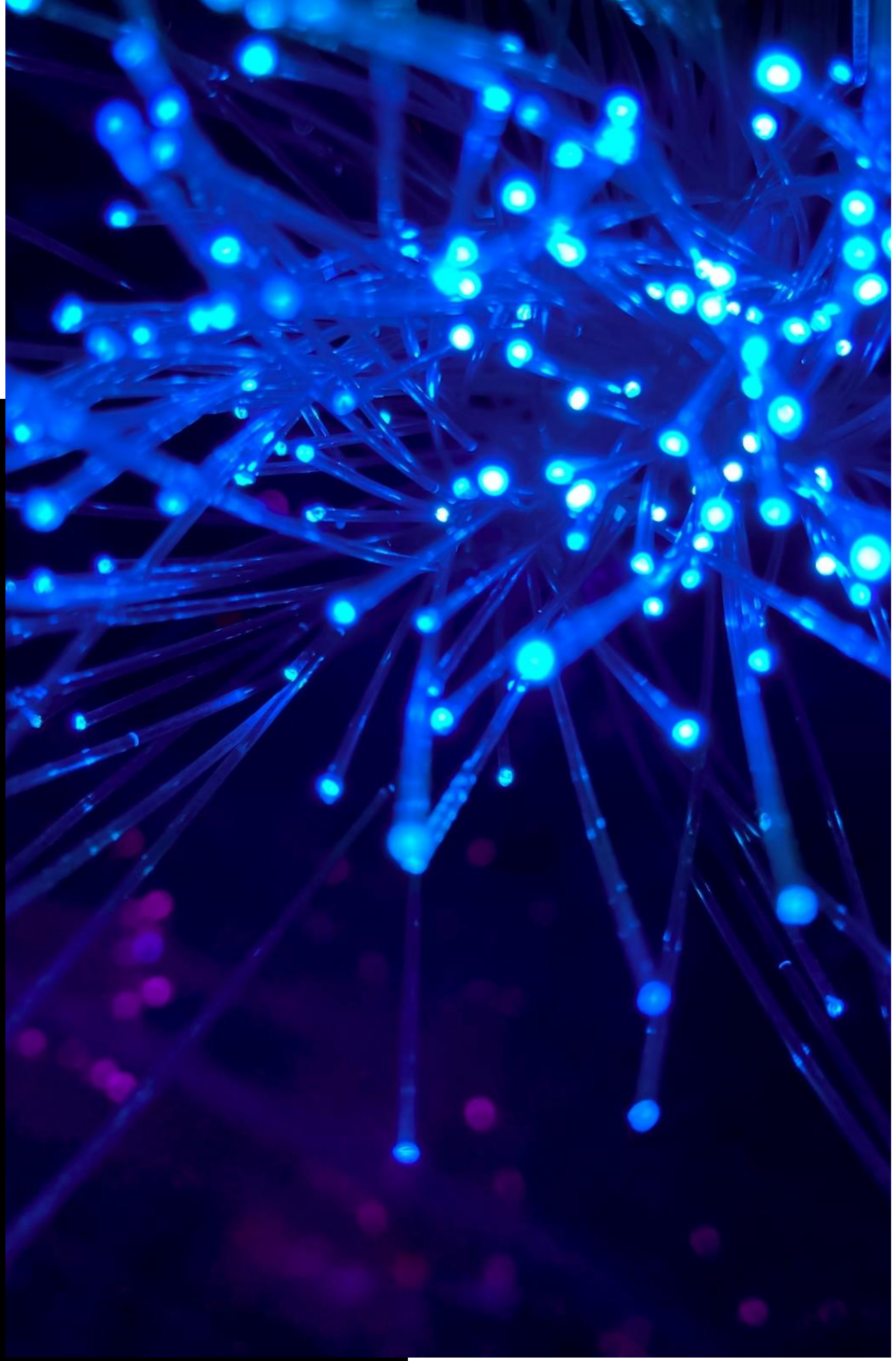
- https://fhernanb.github.io/Manual-de-R/

- https://cran.r-project.org/doc/contrib/rdebuts_es.pdf

- https://cran.r-project.org/doc/contrib/Short-refcard.pdf

- **Guías y cursos de Coursera, Youtube, UDEMY ….**





01.- Instalación

Instrucciones

https://cran.r-project.org/

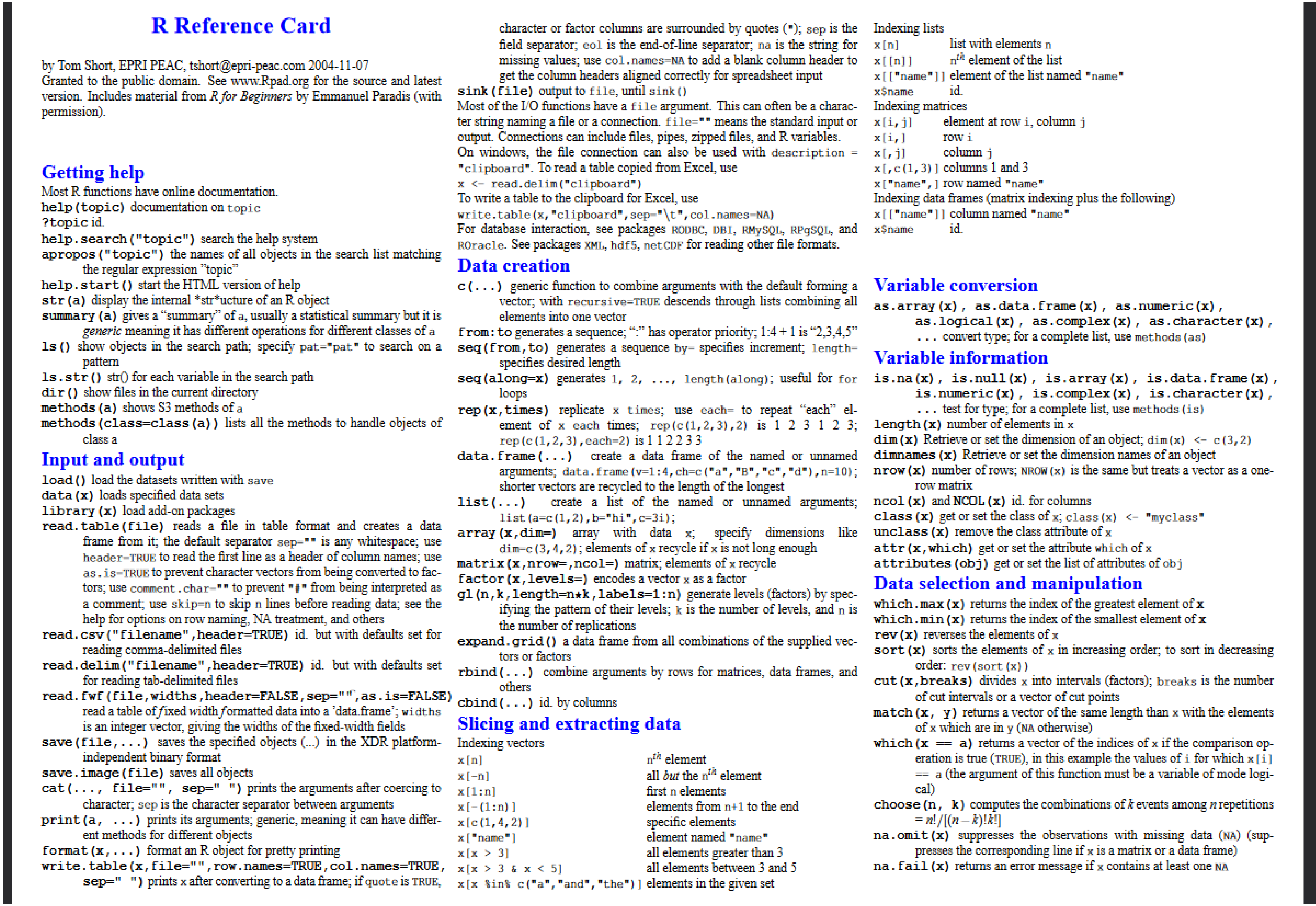
https://cran.r-project.org/bin/windows/base/

https://cran.r-project.org/bin/macosx/
https://datacritica.org/2021/03/19/instalacion-de-r-y-rstudio-en-macos/

1. Conceder acceso directo al escritorio. Por lo demás todo por defecto.

2. Ejecutamos

3. (opcional) Instalar Rstudio
   - **Todo por defecto**
   - **https://posit.co/download/rstudio-desktop/**



R Reference Card

by Tom Short, EPRI PEAC, tshort@epri-peac.com 2004-11-07
Granted to the public domain. See 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="",as.is=FALSE)` read a table of *f*ixed width *f*ormatted 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 for 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]`	n^{th} element
`x[-n]`	all *but* the n^{th} 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]]`	n^{th} 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)

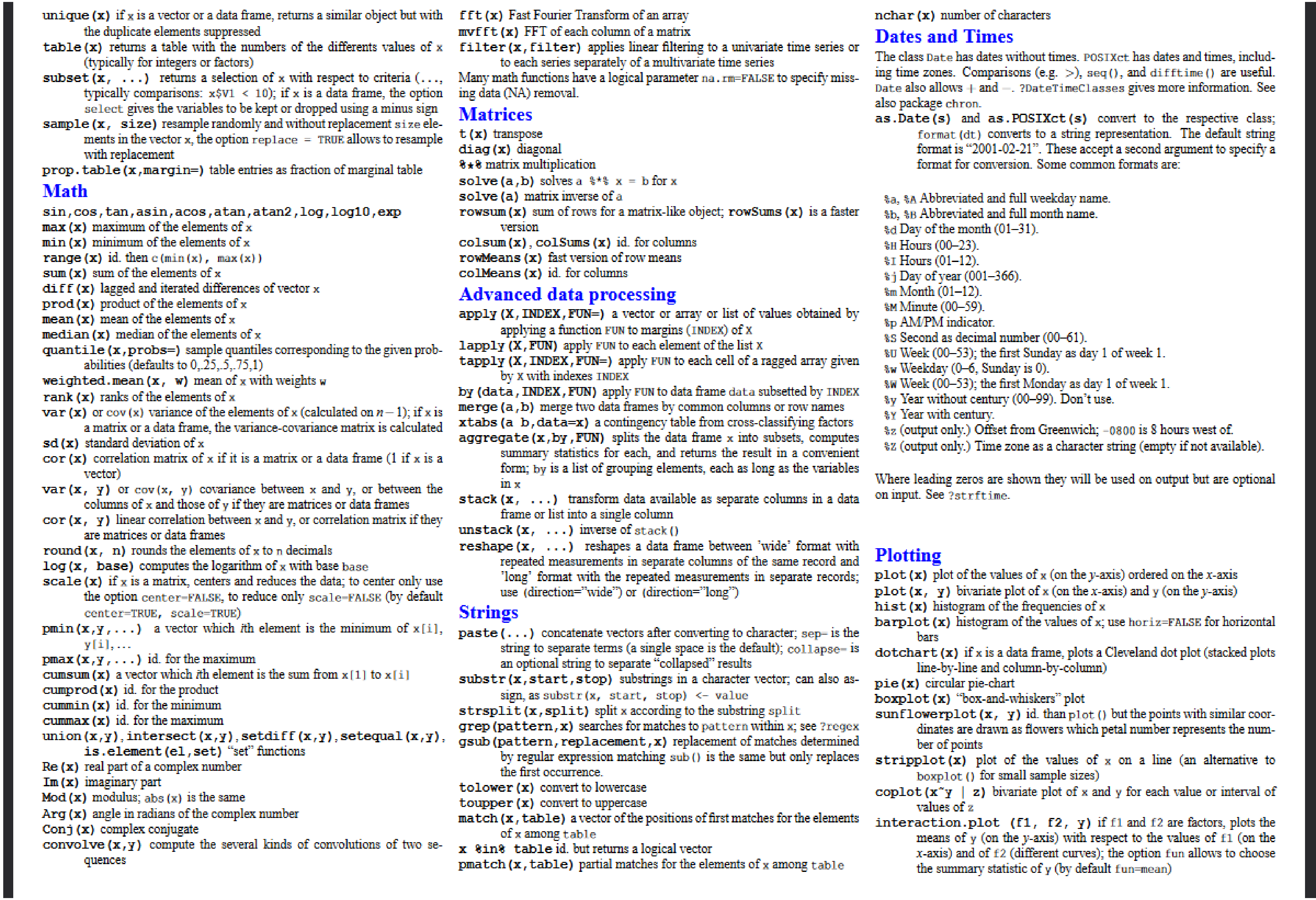
`which(x == a)` returns a vector of the indices of `x` if the comparison operation is true (`TRUE`), in this example the values of `i` for which `x[i] == a` (the argument of this function must be a variable of mode logical)

`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`

UAX Universidad Alfonso X el Sabio



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 differents 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(el,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) `matplot(x,y)` bivariate plot of the first column of x *vs.* the first one of y, the second one of x *vs.* the second one of y, etc.

`fourfoldplot(x)` visualizes, with quarters of circles, the association between two dichotomous variables for different populations (x must be an array with `dim=c(2, 2, k)`, or a matrix with `dim=c(2, 2)` if $k = 1$)

`assocplot(x)` Cohen–Friendly graph showing the deviations from independence of rows and columns in a two dimensional contingency table

`mosaicplot(x)` 'mosaic' graph of the residuals from a log-linear regression of a contingency table

`pairs(x)` if x is a matrix or a data frame, draws all possible bivariate plots between the columns of x

`plot.ts(x)` if x is an object of class `"ts"`, plot of x with respect to time, x may be multivariate but the series must have the same frequency and dates

`ts.plot(x)` id. but if x is multivariate the series may have different dates and must have the same frequency

`qqnorm(x)` quantiles of x with respect to the values expected under a normal law

`qqplot(x, y)` quantiles of y with respect to the quantiles of x

`contour(x, y, z)` contour plot (data are interpolated to draw the curves), x and y must be vectors and z must be a matrix so that `dim(z)=c(length(x), length(y))` (x and y may be omitted)

`filled.contour(x, y, z)` id. but the areas between the contours are coloured, and a legend of the colours is drawn as well

`image(x, y, z)` id. but with colours (actual data are plotted)

`persp(x, y, z)` id. but in perspective (actual data are plotted)

`stars(x)` if x is a matrix or a data frame, draws a graph with segments or a star where each row of x is represented by a star and the columns are the lengths of the segments

`symbols(x, y, ...)` draws, at the coordinates given by x and y, symbols (circles, squares, rectangles, stars, thermometres or "boxplots") which sizes, colours ... are specified by supplementary arguments

`termplot(mod.obj)` plot of the (partial) effects of a regression model (`mod.obj`)

The following parameters are common to many plotting functions:

`add=FALSE` if `TRUE` superposes the plot on the previous one (if it exists)

`axes=TRUE` if `FALSE` does not draw the axes and the box

`type="p"` specifies the type of plot, `"p"`: points, `"l"`: lines, `"b"`: points connected by lines, `"o"`: id. but the lines are over the points, `"h"`: vertical lines, `"s"`: steps, the data are represented by the top of the vertical lines, `"S"`: id. but the data are represented by the bottom of the vertical lines

`xlim=, ylim=` specifies the lower and upper limits of the axes, for example with `xlim=c(1, 10)` or `xlim=range(x)`

`xlab=, ylab=` annotates the axes, must be variables of mode character

`main=` main title, must be a variable of mode character

`sub=` sub-title (written in a smaller font)

Low-level plotting commands

`points(x, y)` adds points (the option `type=` can be used)

`lines(x, y)` id. but with lines

`text(x, y, labels, ...)` adds text given by `labels` at coordinates (x,y); a typical use is: `plot(x, y, type="n"); text(x, y, names)`

`mtext(text, side=3, line=0, ...)` adds text given by `text` in the margin specified by `side` (see `axis()` below); `line` specifies the line from the plotting area

`segments(x0, y0, x1, y1)` draws lines from points (x0,y0) to points (x1,y1)

`arrows(x0, y0, x1, y1, angle= 30, code=2)` id. with arrows at points (x0,y0) if `code=2`, at points (x1,y1) if `code=1`, or both if `code=3`; `angle` controls the angle from the shaft of the arrow to the edge of the arrow head

`abline(a,b)` draws a line of slope b and intercept a

`abline(h=y)` draws a horizontal line at ordinate y

`abline(v=x)` draws a vertical line at abcissa x

`abline(lm.obj)` draws the regression line given by `lm.obj`

`rect(x1, y1, x2, y2)` draws a rectangle which left, right, bottom, and top limits are x1, x2, y1, and y2, respectively

`polygon(x, y)` draws a polygon linking the points with coordinates given by x and y

`legend(x, y, legend)` adds the legend at the point (x,y) with the symbols given by `legend`

`title()` adds a title and optionally a sub-title

`axis(side, vect)` adds an axis at the bottom (`side=1`), on the left (2), at the top (3), or on the right (4); `vect` (optional) gives the abcissa (or ordinates) where tick-marks are drawn

`rug(x)` draws the data x on the *x*-axis as small vertical lines

`locator(n, type="n", ...)` returns the coordinates (*x,y*) after the user has clicked n times on the plot with the mouse; also draws symbols (`type="p"`) or lines (`type="l"`) with respect to optional graphic parameters (`...`); by default nothing is drawn (`type="n"`)

Graphical parameters

These can be set globally with `par(...)`; many can be passed as parameters to plotting commands.

`adj` controls text justification (0 left-justified, 0.5 centred, 1 right-justified)

`bg` specifies the colour of the background (ex. : `bg="red"`, `bg="blue"`, ... the list of the 657 available colours is displayed with `colors()`)

`bty` controls the type of box drawn around the plot, allowed values are: `"o"`, `"l"`, `"7"`, `"c"`, `"u"` ou `"]"` (the box looks like the corresponding character); if `bty="n"` the box is not drawn

`cex` a value controlling the size of texts and symbols with respect to the default; the following parameters have the same control for numbers on the axes, `cex.axis`, the axis labels, `cex.lab`, the title, `cex.main`, and the sub-title, `cex.sub`

`col` controls the color of symbols and lines; use color names: `"red"`, `"blue"` see `colors()` or as `"#RRGGBB"`; see `rgb()`, `hsv()`, `gray()`, and `rainbow()`; as for `cex` there are: `col.axis`, `col.lab`, `col.main`, `col.sub`

`font` an integer which controls the style of text (1: normal, 2: italics, 3: bold, 4: bold italics); as for `cex` there are: `font.axis`, `font.lab`, `font.main`, `font.sub`

`las` an integer which controls the orientation of the axis labels (0: parallel to the axes, 1: horizontal, 2: perpendicular to the axes, 3: vertical)

`lty` controls the type of lines, can be an integer or string (1: `"solid"`, 2: `"dashed"`, 3: `"dotted"`, 4: `"dotdash"`, 5: `"longdash"`, 6: `"twodash"`, or a string of up to eight characters (between `"0"` and `"9"`) which specifies alternatively the length, in points or pixels, of the drawn elements and the blanks, for example `lty="44"` will have the same effect than `lty=2`

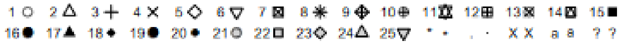
`lwd` a numeric which controls the width of lines, default 1

`mar` a vector of 4 numeric values which control the space between the axes and the border of the graph of the form `c(bottom, left, top, right)`, the default values are `c(5.1, 4.1, 4.1, 2.1)`

`mfcol` a vector of the form `c(nr,nc)` which partitions the graphic window as a matrix of `nr` lines and `nc` columns, the plots are then drawn in columns

`mfrow` id. but the plots are drawn by row

`pch` controls the type of symbol, either an integer between 1 and 25, or any single character within `""`



`ps` an integer which controls the size in points of texts and symbols

`pty` a character which specifies the type of the plotting region, `"s"`: square, `"m"`: maximal

`tck` a value which specifies the length of tick-marks on the axes as a fraction of the smallest of the width or height of the plot; if `tck=1` a grid is drawn

`tcl` a value which specifies the length of tick-marks on the axes as a fraction of the height of a line of text (by default `tcl=-0.5`)

`xaxt` if `xaxt="n"` the *x*-axis is set but not drawn (useful in conjonction with `axis(side=1, ...)`)

`yaxt` if `yaxt="n"` the *y*-axis is set but not drawn (useful in conjonction with `axis(side=2, ...)`)

Lattice (Trellis) graphics

`xyplot(y~x)` bivariate plots (with many functionalities)

`barchart(y~x)` histogram of the values of y with respect to those of x

`dotplot(y~x)` Cleveland dot plot (stacked plots line-by-line and column-by-column)

`densityplot(~x)` density functions plot

`histogram(~x)` histogram of the frequencies of x

`bwplot(y~x)` "box-and-whiskers" plot

`qqmath(~x)` quantiles of x with respect to the values expected under a theoretical distribution

`stripplot(y~x)` single dimension plot, x must be numeric, y may be a factor

`qq(y~x)` quantiles to compare two distributions, x must be numeric, y may be numeric, character, or factor but must have two 'levels'

`splom(~x)` matrix of bivariate plots

`parallel(~x)` parallel coordinates plot

`levelplot(z~x*y|g1*g2)` coloured plot of the values of z at the coordinates given by x and y (x, y and z are all of the same length)

`wireframe(z~x*y|g1*g2)` 3d surface plot

`cloud(z~x*y|g1*g2)` 3d scatter plot

In the normal Lattice formula, `y  x|g1*g2` has combinations of optional conditioning variables `g1` and `g2` plotted on separate panels. Lattice functions take many of the same arguments as base graphics plus also `data=` the data frame for the formula variables and `subset=` for subsetting. Use `panel=` to define a custom panel function (see `apropos("panel")` and `?llines`). Lattice functions return an object of class trellis and have to be `print`-ed to produce the graph. Use `print(xyplot(...))` inside functions where automatic printing doesn't work. Use `lattice.theme` and `lset` to change Lattice defaults.

Optimization and model fitting

`optim(par, fn, method = c("Nelder-Mead", "BFGS", "CG", "L-BFGS-B", "SANN")` general-purpose optimization; `par` is initial values, `fn` is function to optimize (normally minimize)

`nlm(f,p)` minimize function `f` using a Newton-type algorithm with starting values `p`

`lm(formula)` fit linear models; `formula` is typically of the form `response termA + termB + ...`; use `I(x*y) + I(x^2)` for terms made of nonlinear components

`glm(formula,family=)` fit generalized linear models, specified by giving a symbolic description of the linear predictor and a description of the error distribution; `family` is a description of the error distribution and link function to be used in the model; see `?family`

`nls(formula)` nonlinear least-squares estimates of the nonlinear model parameters

`approx(x,y=)` linearly interpolate given data points; `x` can be an xy plotting structure

`spline(x,y=)` cubic spline interpolation

`loess(formula)` fit a polynomial surface using local fitting

Many of the formula-based modeling functions have several common arguments: `data=` the data frame for the formula variables, `subset=` a subset of variables used in the fit, `na.action=` action for missing values: `"na.fail"`, `"na.omit"`, or a function. The following generics often apply to model fitting functions:

`predict(fit,...)` predictions from `fit` based on input data

`df.residual(fit)` returns the number of residual degrees of freedom

`coef(fit)` returns the estimated coefficients (sometimes with their standard-errors)

`residuals(fit)` returns the residuals

`deviance(fit)` returns the deviance

`fitted(fit)` returns the fitted values

`logLik(fit)` computes the logarithm of the likelihood and the number of parameters

`AIC(fit)` computes the Akaike information criterion or AIC

Statistics

`aov(formula)` analysis of variance model

`anova(fit,...)` analysis of variance (or deviance) tables for one or more fitted model objects

`density(x)` kernel density estimates of `x`

`binom.test()`, `pairwise.t.test()`, `power.t.test()`, `prop.test()`, `t.test()`, ... use `help.search("test")`

Distributions

`rnorm(n, mean=0, sd=1)` Gaussian (normal)
`rexp(n, rate=1)` exponential
`rgamma(n, shape, scale=1)` gamma
`rpois(n, lambda)` Poisson
`rweibull(n, shape, scale=1)` Weibull
`rcauchy(n, location=0, scale=1)` Cauchy
`rbeta(n, shape1, shape2)` beta
`rt(n, df)` 'Student' (*t*)
`rf(n, df1, df2)` Fisher–Snedecor (*F*) (χ^2)
`rchisq(n, df)` Pearson
`rbinom(n, size, prob)` binomial
`rgeom(n, prob)` geometric
`rhyper(nn, m, n, k)` hypergeometric
`rlogis(n, location=0, scale=1)` logistic
`rlnorm(n, meanlog=0, sdlog=1)` lognormal
`rnbinom(n, size, prob)` negative binomial
`runif(n, min=0, max=1)` uniform
`rwilcox(nn, m, n)`, `rsignrank(nn, n)` Wilcoxon's statistics

All these functions can be used by replacing the letter `r` with `d`, `p` or `q` to get, respectively, the probability density (`d`*`func`*`(x, ...)`), the cumulative probability density (`p`*`func`*`(x, ...)`), and the value of quantile (`q`*`func`*`(p, ...)`, with $0 < p < 1$).

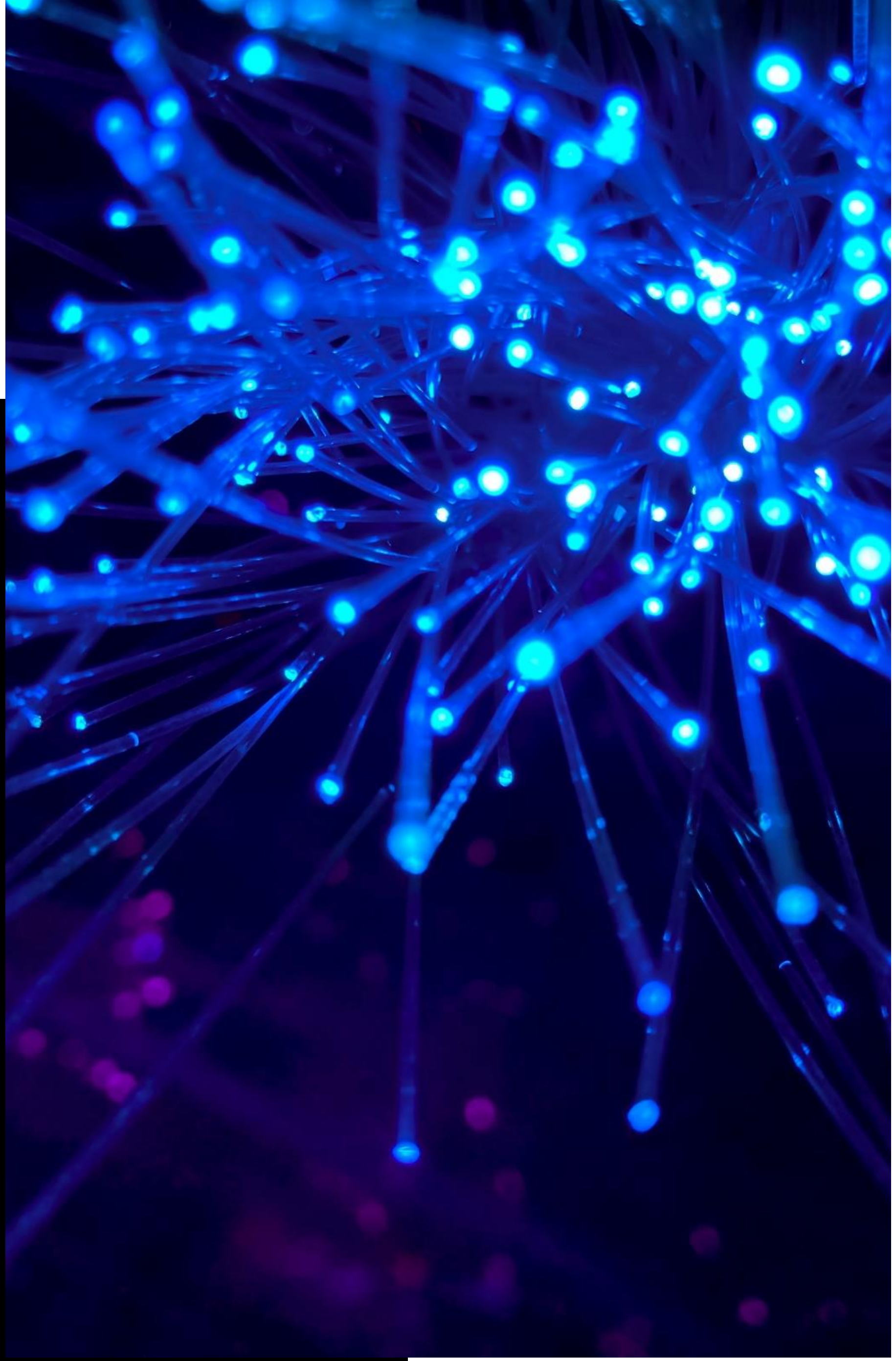
Programming

`function( arglist ) expr` function definition
`return(value)`
`if(cond) expr`
`if(cond) cons.expr else alt.expr`
`for(var in seq) expr`
`while(cond) expr`
`repeat expr`
`break`
`next`

Use braces {} around statements

`ifelse(test, yes, no)` a value with the same shape as `test` filled with elements from either `yes` or `no`

`do.call(funname, args)` executes a function call from the name of the function and a list of arguments to be passed to it

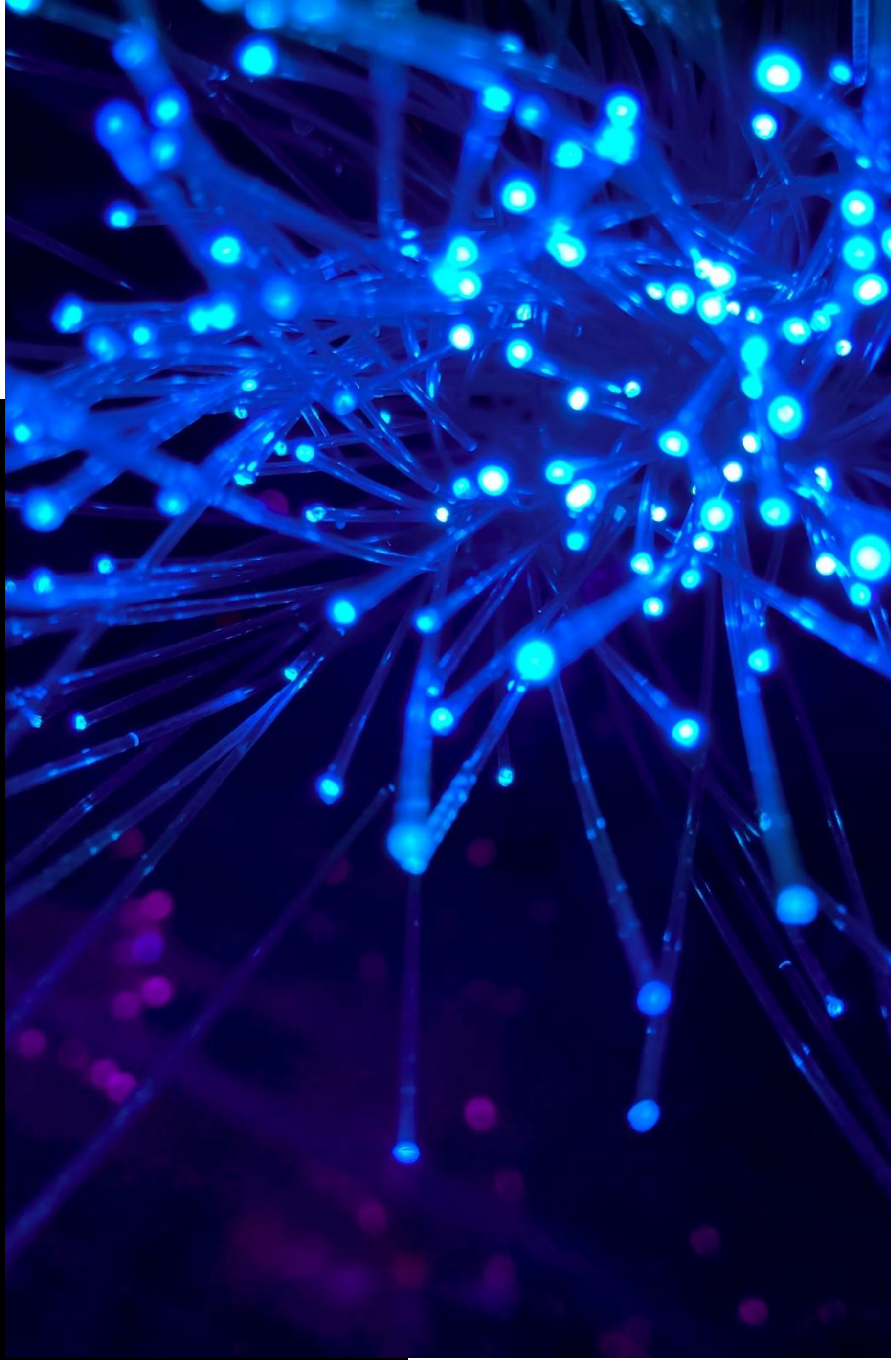


02.- Primer Script

Primer Script

Script de trabajo -> Introduccion.R
Archivo de entrada -> bp.txt

1. Abrir R
2. Cambiar Dir
3. Abrir script Introduccion.R
4. Ejecutar comandos (Ctrl+R / Cmd + R / Botón Ejecutar)



03.- Estadística

Estadística básica

- R tiene una serie de funciones para realizar multitud de cálculos y operaciones estadísticas:
- **sum(), mean(), median(), max(), min(), sd(), var(), length(), quantile()…**
- También cuenta con un montón de paquetes o librerías que realizan análisis estadísticos muy concretos.
- **Nosotros nos vamos a centrar en los que vienen por defecto en R::base**
- **Spahiro-Wilk, Kolgomorov-Smirnov**
- **Fisher**
- **T-test y ANOVA**
- **Wilcox y Kruskal Wallis**
- **Chi Square**
- **Correlación**
- **FDR**

Saphiro-Wilk test

> shapiro.test()

- Se emplea para determinar la **normalidad** de una distribución (3-5000 observaciones).
- ***Para muestras mayores se emplea un quantile-quantile plot

- Si la distribución **es normal -> test paramétricos**
- Si la distribución **no es normal -> test no paramétricos**

Kolmogórov-Smirnov

> ks.test()

- Se emplea para evaluar si dos muestras corresponden a la misma población.

- **Ho:** ambas muestras pertenecen a la misma población
- **H1:** las muestras pertenecen a distintas poblaciones

T-Student y ANOVA

> t.test()

- **Método paramétrico** para comparar la media de máximo 2 muestras de distribución normal.

> aov()

- **Método paramétrico** para comparar la media de 3 o más muestras de distribución normal.
- **Análisis de varianzas y muestras independientes (complejo).**

Wilcoxon / Kruskal-Wallis

> u.test() / wilcox.test()

- **Método no paramétrico** para comparar la mediana de máximo 2 muestras de distribución no normal.

> kruskal.test()

- **Método no paramétrico** para comparar la media de 3 o más muestras de distribución normal.

Fisher’s F-test

> var.test() / fisher.test()

varianzas
evaluo ambas y si salen parecidas = tienen la misma varianza o parecida
si v=
var=TRUE

- Evalúa si dos muestras tienen la misma varianza.
- Puede analizar tablas de contingencia si los datos son pequeños.
- Parecido a fligner.test() y bartlett.test().

Chi-Square

> chisq.test()

- **Test de asociación**. Identifica diferencias significativas entre grupos categóricos en una tabla de contingencia.

- Si la tabla es demasiado pequeña (valores observados inferiores a 5), se recomienda usar el test de Fisher.

How to tell if x, y are independent?

There are two ways to tell if they are independent:

1.By looking at the p-Value: If the p-Value is less that 0.05, we fail to reject the null hypothesis that the x and y are independent. So for the example output above, (p-Value=2.954e-07), we reject the null hypothesis and conclude that x and y are not independent.

2.From Chi.sq value: For 2 x 2 contingency tables with 2 degrees of freedom (d.o.f), if the Chi-Squared calculated is greater than 3.841 (critical value), we reject the null hypothesis that the variables are independent. To find the critical value of larger d.o.f contingency tables, use qchisq(0.95, n-1), where n is the number of variables.

Correlación

> cor.test()

- **Test de correlación** entre dos variables.

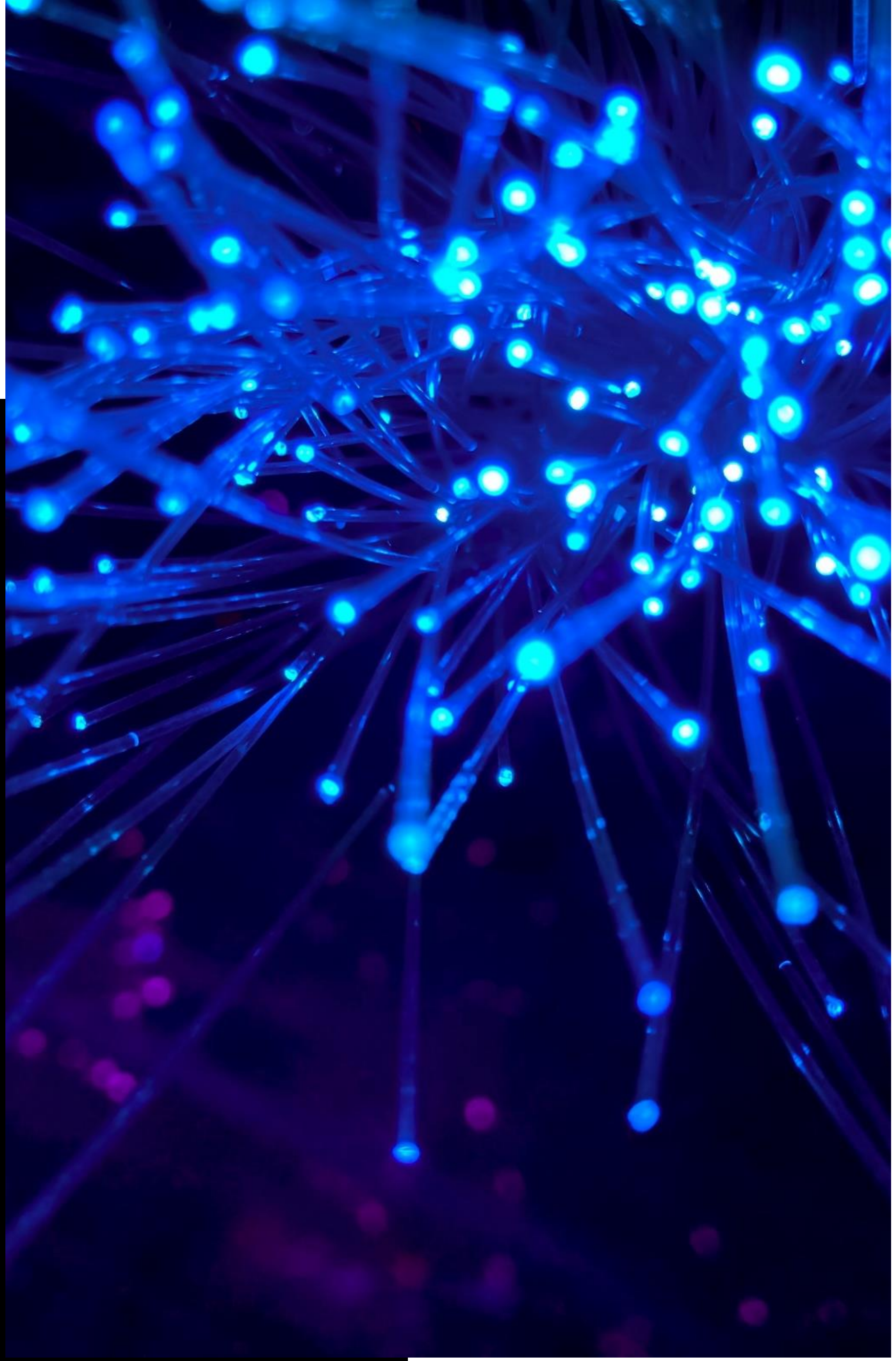
- **Ho:** no existe correlación (independientes)
- **H1:** existe correlación (dependientes)

False Discovery Rate (FDR)

- Las correcciones de p-value cuando hacemos un gran número de test estadísticos (por ejemplo, T-Student) **son imprescindibles** para asegurar que nuestro proyecto sea publicado y tenga una calidad decente.

- En algunos casos se baja el nivel de significancia (ej. 0.01) y en otros se aplica algunas correcciones como la Bonferroni o la de Benjamini_Hochberg

- https://rpubs.com/Joaquin_AR/236898



04.- Ejercicios de estadística

Ejercicios estadística

Script de trabajo -> RStatistics.R
Archivo de entrada -> anova-datos.txt

1. Abrir R
2. Cambiar Dir
3. Abrir script RStatistics.R
4. Ejecutar comandos (Ctrl+R / Cmd + R / Botón Ejecutar)