Annexe B

DictionnaiRe

Aide en ligne

documentation : `help(topic)` ou `?topic`

html : `help.start()`

rechercher dans l'aide : `help.search(pattern)` ou `??pattern`

rechercher dans les titres : `apropos(what)`

rechercher sur le site de R : `RSiteSearch(string)`

Comparaisons

différent : `x != y`

égal : `x == y`

équivalence : `identical(x, y)`

plus grand : `x > y`

plus grand ou égal : `x >= y`

plus petit : `x < y`

plus petit ou égal : `x <= y`

quasi-égalité et différences : `all.equal(target, current)`

Complexes

argument : `Arg(x)`

conjugé : `Conj(x)`

module : `Mod(x)`, `abs(x)`

partie réelle : `Re(x)`
partie imaginaire : `Im(x)`

Constantes

faux : `FALSE`
lettres minuscules : `letters`
lettres majuscules : `LETTERS`
pi : `pi`
valeur indéfinie : `NaN`
valeur infinie : `Inf`
valeur manquante : `NA`
vrai : `TRUE`

Chaines de caractères

abréger : `abbreviate()`
concatener : `paste(...)`
diviser : `strsplit(x, split)`
expressions régulières (regex) :
(alternative, "ou") `|`
(caractères spéciaux et échappés) `\\`
(classe des caractères "chiffres") `[:digit:]`
(classe des caractères "espaces") `[:space:]`
(classe des caractères majuscules) `[:upper:]`
(classe des caractères minuscules) `[:lower:]`
(classe des caractères de ponctuation) `[:punct:]`
(début d'une chaîne) `^`
(liste de caractères à égaler) `[ ]`
(liste de caractères à ne pas égaler) `[^ ]`
(fin d'une chaîne) `$`
(intervalle de caractères) `-`
(regroupement) `()`
(répéter 0 fois ou plus l'expression précédente) `*`
(répéter 1 fois ou plus l'expression précédente) `+`
(répéter 0 ou 1 fois l'expression précédente) `?`
(répéter n fois exactement l'expression précédente) `{n}`
(répéter de n à m fois l'expression précédente) `{n,m}`

nombre de caractères : `nchar(x)`
passer en minuscules : `tolower(x)`
passer en majuscules : `toupper(x)`
sous-chaine : `substr(x, start, stop)`
recherche approximative : `agrep(pattern, x)`
rechercher : `grep(pattern, x)`
remplacer caractères : `chartr(old, new, x)`
remplacer partout : `gsub(pattern, replacement, x)`
remplacer première occurrence :
`sub(pattern, replacement, x)`

Dates et Heures

conversions : `as.Date()`, `as.POSIXct()`, `as.POSIXlt()`, `as.double()`, `strptime()`
date actuelle : `Sys.Date()`, `date()`
date et heure actuelles : `Sys.time()`.
formatage : `format()`
opérations et comparaisons : `+`, `-`, `==`, `!=`, `<`, `<=`, `>`, `>=`
paramètres de formatage :
(année sur 2 chiffres) `%y`,
(année sur 4 chiffres) `%Y`,
(date `%y/%m/%d`) `%x`,
(jour de l'année de 1 à 366) `%j`,
(jour de la semaine, dimanche = 0) `%w`,
(jour du mois de 1 à 31) `%d`,
(heure de 0 à 23) `%H`,
(minute de 0 à 59) `%M`,
(mois de 1 à 12) `%m`,
(nom abrégé du jour) `%a`,
(nom abrégé du mois) `%b`,
(nom complet du jour) `%A`,
(nom complet du mois) `%B`,
(secondes) `%S`,
(semaine de 1 à 53) `%U`,
(temps) `%x`,
(temps `%H:%M:%S`) `%T`, `%X`
séquences : `seq.Date()`, `seq.POSIXt()`

Ensembles d'index

différence : `setdiff(x, y)`

égalité : `setequal(x, y)`

est éléments de : `is.element(el, set)`

intersection : `intersect(x, y)`

union : `union(x, y)`

Entrées - Sorties

afficher : `print(a, ...)`

connections :

```
file(description = "", open = "", blocking = TRUE,
           encoding = getOption("encoding"), raw = FALSE),
url(description, open = "", blocking = TRUE,
           encoding = getOption("encoding")),
open(con, open = "r", blocking = TRUE),
close(con, type = "rw"),
```

dialogue de sélection de fichier : `file.choose()`

écrire : `cat(..., file="", sep="")`

écrire fichier binaire R : `save(..., file)`

écrire fichier structuré :

```
write.table(x,file="", row.names=TRUE, col.names=TRUE, sep="")
```

écrire lignes : `writeLines(text, con = stdout(), sep = "n"`

formatage de l'affichage : `format(x,...)`

lire fichier csv : `read.csv(file, header=TRUE)`

lire fichier structuré : `read.table(file)`

lire fichier tabulé : `read.delim(file, header=TRUE)`

lire fichier binaire R : `load(file)`

lire lignes :

```
readLines(con = stdin(), n = -1L, ok = TRUE,
           warn = TRUE, encoding = "unknown")
```

Graphiques

boîte à moustaches : `boxplot(x)`

camenbert : `pie(x)`

diagramme en barres : `barplot(x)`

graphe de Cleveland : `dotchart(x)`

histogramme : `hist(x)`

nuage de points : `plot(x, y)`

primitives graphiques :

`points(x, y)`, `lines(x, y)`,
`text(x, y, labels, ...)`, `mtext(text, side=3, line=0, ...)`,
`segments(x0, y0, x1, y1)`,
`arrows(x0, y0, x1, y1, angle= 30, code=2)`,
`abline(a,b)`, `abline(h=y)`, `abline(v=x)`,
`rect(x1, y1, x2, y2)`, `polygon(x, y)`,
`legend(x, y, legend`, `title()`

tableau des graphes bivariés : `pairs(x)`

paramètres :

`add=`, `type=`, `axes=`, `xlim=`, `ylim=`,
`xlab=`, `ylab=`, `par(...)`

représentation graphique : `plot(x)`

Indexation

éléments d'index `n` **:** `x[n]`

éléments où `u` **est vrai :** `x[u]`

éléments non indexés dans `n` **:** `x[-n]`

élément de nom `name` **:** `x["name"]`

listes : `x[["name"]]`, `x$name`, `x[[n]]`

matrices et tableaux de données :

`x[i, j]`, `x[i, ]`, `x[ ,j]`,
`x[-i, -j]`, `x[-i, ]`, `x[, -j]`,
`x["name", ]`, `x[, "name"]`

position des valeurs de `x` **dans** `y` **:** `match(x, y)`

Logique

appartient à : `x %in% table`

au moins une valeur vraie ? : `any(..., na.rm = FALSE)`

et vectoriel : `&`

et scalaire : `&&`

négation : `!`

ou vectoriel : `|`

ou scalaire : `||`

toutes les valeurs vraies ? : `all(..., na.rm = FALSE)`

index des valeurs vraies : `which()`

Mathèmatiques

arrondi : `floor(x)`, `ceiling(x)`, `trunc(x)`, `round(x)`, `signif(x)`

division entière : `x %/% y`

est fini ? : `is.finite(x)`

est infini ? : `is.infinite(x)`

est indéfini ? : `is.nan(x)`

fonctions trigonométriques :
`cos(x)`, `sin(x)`, `tan(x)`, `acos(x)`, `asin(x)`, `atan(x)`

logarithmes et exponentielles :
`log(x, base = exp(1))`, `log10(x)`, `log2(x)`, `exp(x)`

modulo : `x %% y`

opérations : `x + y`, `x - y`, `x * y`, `x / y`, `x ^ y`

racine carrée : `sqrt(x)`

valeur absolue : `abs(x)`

signe : `sign(x)`

Matrices

diagonales : `diag(x)`

inverse : `solve(a)`

moyennes par ligne : `rowMeans(x)`

produit matriciel : `%*%`

solution du système linéaire : `solve(a,b)`

sommes par ligne : `rowsum(x)`, `rowSums(x)`

sommes par colonne : `colsum(x)`, `colSums(x)`

transposée : `t(x)`

Objets

accéder à une fonction ou un objet d'un package :
`::` ou `:::`

assignation d'une valeur : `<-` ou `assign(x , value)`

attributs d'un objet : `attributes(obj)`

combiner : `c(...)`

combiner les colonnes d'objets : `cbind(...)`

combiner les lignes d'objets : `rbind(...)`

conversion d'objets :
(vers logique) `as.logical(x)`,
(vers entier) `as.integer(x)`,
(vers réel) `as.numeric(x)`,

création d'objets :
(facteur) `factor(x,levels=)`,
(liste) `list(...)`,
(matrice) `matrix(x,nrow=,ncol=)`,
(tableau) `array(x,dim=)`,
(tableau de données) `data.frame(...)`,
(vecteur entier)`integer(length)`,
(vecteur logique) `logical(length)`,
(vecteur réel) `numeric(length)`

dimension d'un objet : `dim(x)`

est ? :
(caractère) `is.character(x)`,
(complexe) `is.complex(x))`,
(na) `is.na(x)`,
(null) `is.null(x)`,
(numérique) `is.numeric(x)`,
(tableau) `is.array(x)`,
(tableau de données) `is.data.frame(x)`,

extraction d'une partie / composante / slot d'un objet :
`$` ou `@`

jointure / fusion de deux tableaux de données :
(selon les valeurs de colonnes communes)
`merge(x, y, ...)`

lister les objets : `ls(name)` ou `objects(name)`

longueur : `length(x)`

noms des dimensions d'un objet : `dimnames(x)`

structure d'un objet : `str(object)`

supprimer : `rm(...)`

type : `typeof(x)`

Probabilités

coefficient binomial : `choose(n, k)`

densités :

(loi binomiale)
`dbinom(x, size, prob, log = FALSE)`,
(loi normale)
`dnorm(x, mean = 0, sd = 1, log = FALSE)`,
(loi de poisson)
`dpois(x, lambda, log = FALSE)`,
(loi uniforme)
`dunif(x, min=0, max=1, log = FALSE)`

distributions :

(loi binomiale)
`pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)`,
(loi normale)
`pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`,
(loi de poisson)
`ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)`,
(loi uniforme)
`punif(q, min=0, max=1, lower.tail = TRUE, log.p = FALSE)`

factorielle : `factorial(x)`

liste des combinaisons : `combn(x, m)`

log de la factorielle : `lfactorial(x)`

log du coefficient binomial : `lchoose(n, k)`

quantiles :

(loi binomiale)
`qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)`,
(loi normale)
`qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)`,
(loi de poisson)
`qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)`,

(loi uniforme)
`qunif(p, min=0, max=1, lower.tail = TRUE, log.p = FALSE)`

Recoder / transformer

diviser une variable numérique en classes :
`cut(x, ...)` ou `library(Hmisc); cut2(x, ...)`

modifier l'ordre des catégories : `relevel(x, ref)`

numéroter : `seq_along(along.with=x)`, `seq(along.with=x)`

ordre des valeurs : `order(x)`

reporter la dernière valeur non manquante :
`library(zoo); na.locf(object, na.rm = TRUE, ...)`

restructurer les observations d'un tableau de données :
`library(reshape); cast(...)`,
`library(reshape); melt(...)`,
`library(reshape); recast(...)`

trier : `sort(x)`

Séquences

entiers de `from` à `to` : `from:to`, `seq.int(from, to, by, length.out)`

réels de `from` à `to` : `seq(from, to, by, length)`

réplications : `rep(x, times, each)`

Séquences aléatoires

échantillonage : `sample(x, size)`

loi binomiale : `rbinom(n, size, prob)`

loi normale : `rnorm(n, mean=0, sd=1)`

loi de poisson : `rpois(n, lambda)`

loi uniforme : `runif(n, min=0, max=1)`

Statistiques élémentaires

centrer-réduire : `scale(x)`

coefficient de corrélation : `cor(x)` ou `cor(x, y)`

écart-type : `sd(x)`

maximum : `max(x)`
médiane : `median(x)`
minimum : `min(x)`
moyenne : `mean(x)`
position du maximum : `which.max(x)`
position du minimum : `which.min(x)`
produit : `prod(x)`
quantiles : `quantile(x, probs)`
somme : `sum(x)`
résumé : `summary(u)`
tableau de fréquences : `table(x)`
variance : `var(x)`

Structures de contrôle

accéder aux fonctions d'un package `library(x)`
bloc d'instructions `{}`
boucle tant que `while (cond) expr`
itérateur `for (var in seq) expr`
fonction `function(arglist) expr`
si sinon vectoriel `ifelse (test, yes, no)`
test si `if (cond) expr`
test si sinon `if (cond)cons.expr else alt.expr`
valeur à retourner `return(value)`