Pourquoi apprendre le langage R ?

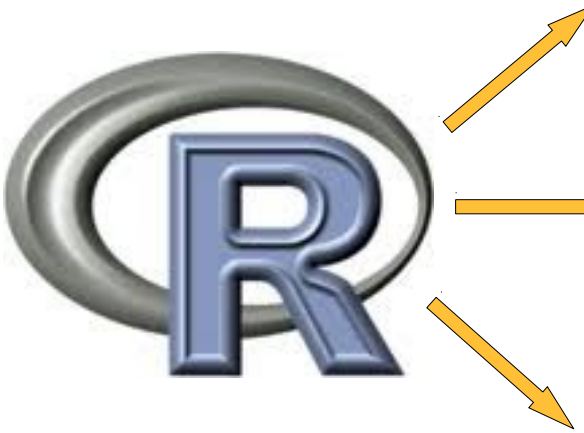
Démonstration d'une infime partie des possibilités

G. San Martin

gilles.sanmartin@gmail.com
Centre Wallon de Recherche Agronomique

cra-w

Domaine d'utilisation



Analyses statistiques

Graphiques

Manipulation de données

+

Langage de script "classique"

Reporting

Etc...

R pour manipuler des données

Quelques exemples

Opérations possibles sur un jeu de données biogéographique classique : espèce, localisation, date, plante hôte, ...

Sélection de colonnes et de lignes

Agrégation de données, tableaux croisés
(nombre d'espèces par carré UTM, table plantes x espèces)

Fusion de tables et vectorisation de calculs
(calcul indices Ellenberg)

Application d'une série d'opérations à un sous ensemble de données
(liste des 10 espèces de plantes les plus fréquentées pour chaque insecte)

Boucles et ré-échantillonnage
(nombre de plantes dans un sous-échantillon aléatoire)

Exemples en « live »

R pour faire des Analyses Statistiques

Pratiquement toutes les méthodes existantes
sont disponibles et en particulier les plus récentes

>7000 “packages” sur le CRAN
http://cran.r-project.org/

“Task views”
http://cran.r-project.org/web/views/

Exceptions :
Certains programmes spécialisés plus avancés
(Structural Equation Modelling, ENFA,...)

Souvent, logiciels commandés depuis R
Pex : BUGS familly : WinBugs, OpenBugs, Jags

Certaines méthodes évitées par choix

R pour faire des Analyses Statistiques

L'analyse de données peut-être vue au sens très large...

Exemple :
Analyse de signaux acoustiques de chauves-souris
Package seewave

Exemples en « live »

R pour faire des Graphiques

Différents systèmes graphiques :

Base
Lattice, ggplot2 (Grid)
RGL, opengl : 3D, manipulables

Tout se règle en ligne de commande !

Possibilité d'exporter en vectoriel et d'éditer dans un logiciel adapté (pex Inkscape)

Exemples en « live »

R comme langage de script

Script :
Succession de commandes que l'on « demande » à l'ordinateur d'effectuer
(sans compilation)

Possibilité de commander d'autres logiciels

Répétition des mêmes tâches
(boucles, fonctions,etc...)

R comme langage de script

Exemple 1 :

Fusion d'un grand nombre de fichiers de données en un seul tableau synthétique + calculs plus ou moins complexes
(ex : fichiers de chromatographie et morphométrie d'ailes de papillons)

Exemple 2 :

Grouper des photos par pile (focus stacks) - appeler d'autres logiciels depuis R pour transformer les photos en tif, les fusionner, écrire des informations dans les métadonnées puis renommer les photos résultantes.

Exemples en « live »

R pour faire des rapports automatiques

Documents textes contenant du code R

Documents odt (Libre Office cfr .doc),
documents Latex, HTML, Markdown, etc...
Packages R : odfweave, Sweave, Knitr, Rmarkdown

Le code est remplacé par le résultat de son exécution (y compris graphiques)
avec de nombreuses options.

Rapports générés directement par du code R

R2HTML : crée des rapports HTML, R2wd : vers MS Word
Généralement peu de texte, surtout des tableaux et des graphiques

Situation intermédiaire

Un script R comprend du texte et des commandes
--> transformé en document Latex, HTML, Markdown,...
Packages Knitr : fonctions stitch, spin

Exemples en « live »

Introduction au langage R

G. San Martin

gilles.sanmartin@gmail.com
Centre Wallon de Recherche Agronomique

cra-w

Introduction au langage R

Contenu

Qu'est-ce qu'un logiciel libre ?
Langage de programmation vs interface graphique
Avantages/désavantages de R
Les conseils de papa...

Premiers contacts :
interface, console interactive, objets, fonctions, packages

Quelques fonctions de base
(+ notions de vectorisation et de recyclage)

Comment utiliser l'aide ?

Notions de base pour les graphiques

Qu'est-ce qu'un logiciel libre ?

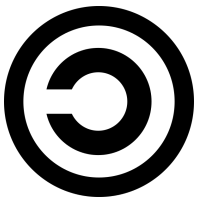
Logiciel Libre $\neq$ Logiciel Gratuit !!

Selon la Free Software Foundation :

0) liberté d'exécuter le programme, pour tous les usages
1) liberté d'étudier le fonctionnement du programme
2) liberté de redistribuer des copies du programme
3) liberté d'améliorer/adapter le programme et de distribuer ces améliorations

--> 1) à 3) : nécessité du code source

Copyleft (licence GPL ou similaire)



Qu'est-ce qu'un logiciel libre ?

Avatanges pratiques

Gratuité (« free as in free beer »)
Licences peu contraignantes pt de vue utilisation
Plus de garantie de pérennité – investissement sur le long terme
Respect des standards, formats ouverts

Avantages Techniques

(selon la communauté “open source”)
Meilleure qualité de logiciel via :
“peer reviewing”, tests et contributions multiples selon les besoins locaux
« La cathédrale et le bazar »

Avantages “philosophiques”

(selon la communauté FSF & co : “free as in free speach”)
Liberté, partage, collaboration
apprentissage, communauté,
participation au bien commun
--> puissante source de motivation !

Langage de programmation vs interface graphique

Mémorisation plus difficile

--> utilisation massive de l'aide, réutiliser de code existant
(investissement, capitalisation !!)

On est "loin" des données

vision plus abstraite des données que dans un tableur
C'est surtout une question d'habitude

Possibilité de répéter les mêmes opérations

Sur sous ensemble des données, sur un nouveau jeu de données, si on corrige une erreur dans le jeu de données d'origine,...

Flexibilité

Pex : dans une analyse toute l'info est stockée dans un objet, et l'utilisateur choisit ce qu'il affiche et peut réutiliser les résultats qu'il veut
Possibilité de créer ses propres analyses/fonctions

Procédure linéaire et auto-documentée (script)

Séparation claire des données et des calculs

Comparer avec un tableur plein de formules

Langage de programmation vs interface graphique

```
> library(fortunes)
> fortune("busses")

When talking about user friendliness of computer software I like the analogy
of cars vs. busses: [...]
Using this analogy programs like SPSS are busses, easy to use for the
standard things, but very frustrating if you want to do something that is not
already preprogrammed.
R is a 4-wheel drive SUV (though environmentally friendly) with a bike on the
back, a kayak on top, good walking and running shoes in the passenger seat,
and mountain climbing and spelunking gear in the back. R can take you
anywhere you want to go if you take time to learn how to use the equipment,
but that is going to take longer than learning where the bus stops are in
SPSS.
   -- Greg Snow
      R-help (May 2006)
```

Langage de programmation vs interface graphique

```
> library(fortunes)
> fortune("Yoda")

Evelyn Hall: I would like to know how (if) I can extract some of the
information from the summary of my nlme.

Simon Blomberg: This is R. There is no if. Only how.

   -- Evelyn Hall and Simon 'Yoda' Blomberg
      R-help (April 2005)
```

Avantages de R ?

Libre

Langage de haut niveau, facile à apprendre, concis (programmation rapide)

Langage interprété (pas compilé) -> + facile (ea pour "débugger")

Parfaitement adapté pour statistiques, manipulation de données, graphiques --> "tout en un"
Probablement un des meilleurs pour les graphiques

Développement très dynamique et en pleine explosion --> très nombreuses applications

Semblable à d'autres langages (Python, Matlab)

Désavantages de R ?

Lent, pas multi-coeur
Mais des (demi-)solutions existent

Toutes les données dans la RAM
--> problèmes avec les (vraiment) très gros jeux de données

????? Fiabilité ?????
N'importe qui peut poster un package sur le CRAN

-->
peer reviewing
“core"/recommended packages
esprit critique
packages liés à une publication

Désavantages de R ?

```
> fortune(223)
I think [R] addresses a niche market for high-end data analysts that want free,
readily available code. [...] We have customers who build engines for aircraft.
I am happy they are not using freeware when I get on a jet.
 -- Anne H. Milley (director of technology product marketing at SAS,
    quoted in Ashlee Vance's article "Data Analysts Captivated by R's Power")
    - The New York Times (January 2009)

> fortune(224)
It's interesting that SAS Institute feels that non-peer-reviewed software
with hidden implementations of analytic methods that cannot be reproduced
by others should be trusted when building aircraft engines.
    -- Frank Harrell  (...) R-help (January 2009)
```

Conseils pour apprendre R

1) investissement sur le long terme
2) similarité avec l'apprentissage d'une langue humaine

Se forcer à utiliser R au début, pratiquer régulièrement

Noter tout + réutiliser du code existant
capitalisation
(→ nécessite rigueur et organisation)

Utiliser l'aide et les ressources internet
correctement et abondamment

Découper les problèmes en petits morceaux
Pour un problème complexe:
Commencer à coder petit puis complexifier

Commenter abondamment son code (#)
Lire les messages d'erreur !

Introduction au langage R

Contenu

Qu'est-ce qu'un logiciel libre ?
Langage de programmation vs interface graphique
Avantages/désavantages de R
Les conseils de papa...

Premiers contacts :
interface, console interactive, objets, fonctions, packages

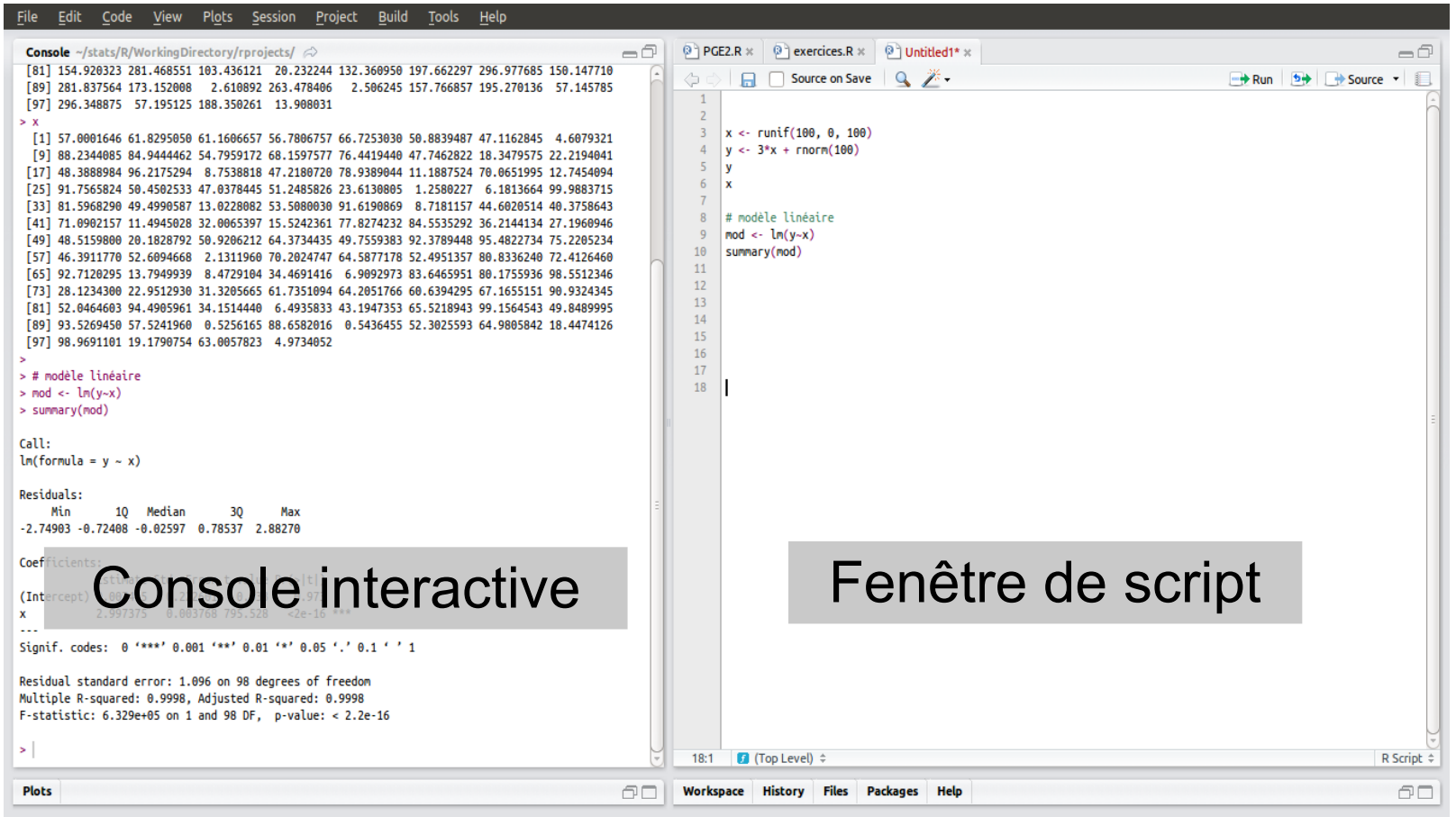
Quelques fonctions de base
(+ notions de vectorisation et de recyclage)

Comment utiliser l'aide ?

Notions de base pour les graphiques

Interface

Dépend des éditeurs...



Console interactive

Fenêtre de script

IDE
R Studio

Invite de commande (« prompt ») : >
Attente d'une suite de commande : +
Historique de commande : haut/bas

Fichier texte (avec extension .R)
Raccourcis claviers pour envoyer commandes vers console

Différents interfaces/éditeurs

Selon les interfaces :

coloration syntaxique, complétion /indentation automatique, aide avancée, mise en évidence des parenthèses, historique des graphiques, facilités d'exportation etc...

Console R et n'importe quel éditeur de texte séparé
+ copier/coller manuels --> pas très efficace

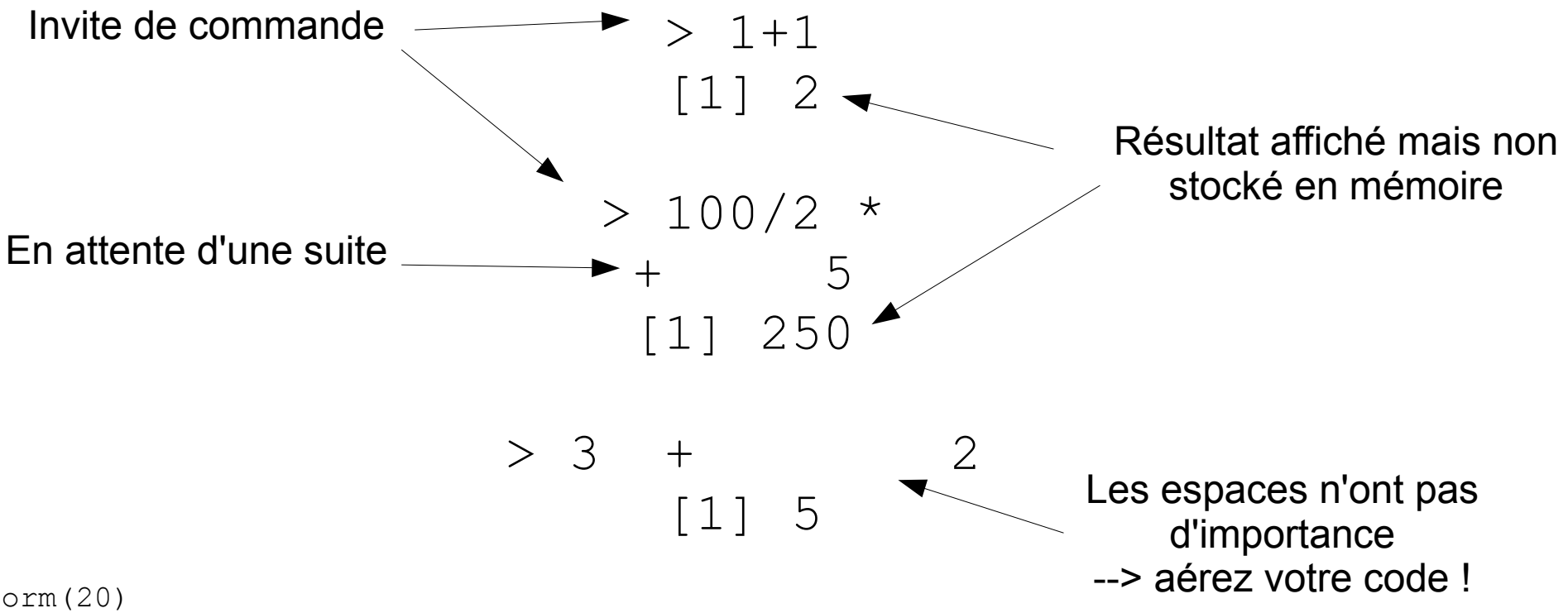
R studio
JGR

Windows : Tinn R / R GUI (défaut : basique)
Linux : Kate + Konsole / Gedit +Rgedit / RKward

Editeurs très avancés, non spécifiques à R
Emacs + ESS
Eclipse

Utilisation interactive

cfr langage Interprété : on tape une commande dans la console puis “enter” et le logiciel retourne la valeur




```
> rnorm(20)
 [1]  1.30848210  0.33161948 -0.40334504  1.75880880 -0.79926102  1.36719364 -2.44453062
 [8] -0.40420299  0.16278240 -0.50425450 -1.46912180  1.42994366  0.85142902 -0.04768551
[15]  0.62383114 -2.34619187 -0.58776281 -1.16118977 -0.26306006 -0.32503095
```

Numéro de l'élément affiché en premier sur la ligne

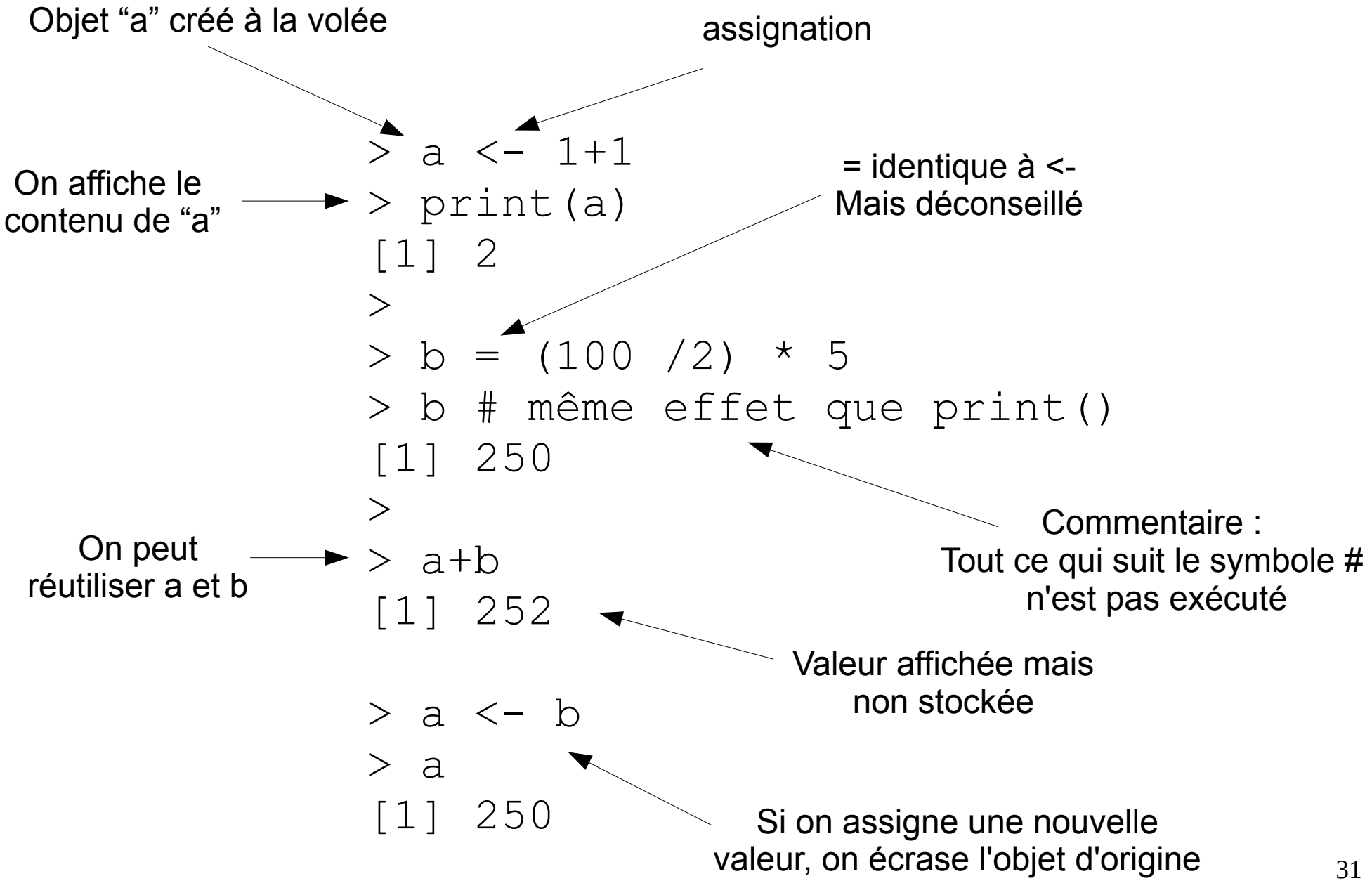
Objets

Tout est stocké dans des objets :
données, fonctions, résultats d'analyses, ...

Objet : unité de stockage désignée par son nom et possédant certaines caractéristiques (classe, mode, dimentions, ...) décrivant le type de contenu et le type de contenant

On crée un objet en lui assignant une valeur avec <-

Objets


Objet “a” créé à la volée
assignation
> a <- 1+1
On affiche le contenu de “a”
> print(a)
[1] 2
>
= identique à <-
Mais déconseillé
> b = (100 /2) * 5
> b # même effet que print()
[1] 250
>
Commentaire :
Tout ce qui suit le symbole #
n'est pas exécuté
On peut réutiliser a et b
> a+b
[1] 252
Valeur affichée mais
non stockée
> a <- b
> a
[1] 250
Si on assigne une nouvelle
valeur, on écrase l'objet d'origine

Objets

!! Noms sensibles à la casse des caractères !!

```
> objet <- 2
> Objet <- 5
> objet + Objet
[1] 7
```

Règles pour les noms d'objets :

doivent commencer par une lettre
peuvent contenir des chiffres et des points

Éviter les noms suivants qui ont déjà une signification :
c, q, s, t, C, D, F, I, T

Personnellement je préfère éviter points et majuscules
(compatibilité avec d'autres logiciels)

Fonctions

Permettent d'agir sur les données
On peut créer ses propres fonctions

Fonction de concaténation
créant un vecteur de nombres

```
> a <- c(1, 23, 6, 4, 7, 90, 32, 7, 3)
> a
[1]  1 23  6  4  7 90 32  7  3
>
> (mymean <- mean(a))
[1] 19.22222
> length(a)
[1] 9
> round(mymean, digits = 2)
[1] 19.22
```

Argument de la fonction

Fonctions

Dans l'aide : `rnorm(n, mean = 0, sd = 1)`

Fonction générant des nombres aléatoires selon loi Normale

Argument obligatoire

Argument avec valeur par défaut --> facultatif

Mean = 0 et sd = 1 par défaut

```
> rnorm(5)
[1] -0.5692024 -1.9194847 -2.5670713 -0.6579548 -0.4690608

> rnorm(mean=5, sd=3)
Erreur dans rnorm(mean = 5, sd = 3) :
  l'argument "n" est manquant, avec aucune valeur par défaut

> rnorm(n = 5, mean = 5, sd = 3)

> rnorm(5, 5, 3)

> rnorm(mean=5, n=5, sd=3)

> rnorm(5, sd=3)
```

n est manquant

Si on respecte l'ordre, pas besoin de nommer les arguments

Fonctions génériques - méthodes

Certaines fonctions dites « génériques » donneront un résultat différent selon le type d'objet qu'elles reçoivent en argument

`summary` = fonction générique

```
> a <- c("rouge", "jaune", "rouge", "noir", "rouge", "noir")
> x <- c(1, 23, 6, 4, 7, 90, 32, 7, 3)
> y <- 2 * x + rnorm(9)
> reg <- lm(y ~ x)

> summary(a)
   Length     Class      Mode
        6 character character

> summary(x)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00    4.00    7.00   19.22   23.00   90.00

> summary(reg)

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.07825    0.46635   0.168    0.871
x            1.99688    0.01412 141.391 2.34e-13 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 1.138 on 7 degrees of freedom
Multiple R-squared: 0.9996,   Adjusted R-squared: 0.9996
F-statistic: 1.999e+04 on 1 and 7 DF,  p-value: 2.335e-13
```

a contient du texte

x contient des nombres

reg contient le résultat d'une régression linéaire

Fonctions génériques - méthodes

Méthodes disponibles pour la fonction “summary”

```
> methods(summary)
 [1] summary.aov             summary.aovlist         summary.aspell*
 [4] summary.connection      summary.corAR1*         summary.corARMA*
 [7] summary.corCAR1*        summary.corCompSymm*    summary.corExp*
[10] summary.corGaus*        summary.corIdent*       summary.corLin*
[13] summary.corNatural*     summary.corRatio*       summary.corSpher*
[16] summary.corStruct*      summary.corSymm*        summary.data.frame
[19] summary.Date            summary.default         summary.ecdf*
[22] summary.factor          summary.glm             summary.gls*
[25] summary.infl            summary.lm              summary.lme*
[28] summary.lmList*         summary.loess*          summary.manova
[31] summary.matrix          summary.mlm             summary.modelStruct*
[34] summary.nls*            summary.nlsList*        summary.packageStatus*
[37] summary.pdBlocked*      summary.pdCompSymm*     summary.pdDiag*
[40] summary.PDF_Dictionary* summary.PDF_Stream*     summary.pdIdent*
[43] summary.pdLogChol*      summary.pdMat*          summary.pdNatural*
[46] summary.pdSymm*         summary.POSIXct         summary.POSIXlt
[49] summary.ppr*            summary.prcomp*         summary.princomp*
[52] summary.reStruct*       summary.shingle*        summary.srcfile
[55] summary.srcref          summary.stepfun         summary.stl*
[58] summary.table           summary.trellis*        summary.tukeysmooth*
[61] summary.varComb*        summary.varConstPower*  summary.varExp*
[64] summary.varFixed*       summary.varFunc*        summary.varIdent*
[67] summary.varPower*
```

Fonctions génériques - méthodes

Fonctions génériques disponibles pour les objets de classe “lm” (modèles linéaires)

```
> methods(class=lm)
 [1] add1.lm*            alias.lm*           anova.lm
 [4] case.names.lm*      confint.lm*         cooks.distance.lm*
 [7] deviance.lm*        dfbeta.lm*          dfbetas.lm*
[10] drop1.lm*           dummy.coef.lm*      effects.lm*
[13] extractAIC.lm*      family.lm*          formula.lm*
[16] hatvalues.lm        influence.lm*       kappa.lm
[19] labels.lm*          logLik.lm*          model.frame.lm
[22] model.matrix.lm     nobs.lm*            plot.lm
[25] predict.lm          print.lm            proj.lm*
[28] qqnorm.lm*          qr.lm*              residuals.lm
[31] rstandard.lm        rstudent.lm         simulate.lm*
[34] summary.lm          variable.names.lm*  vcov.lm*

   Non-visible functions are asterisked
```

Paquets - Packages

R dispose d'une très large bibliothèque d'extensions appelées "packages"

Certains packages sont déjà installés et chargés automatiquement au démarrage de R : stats, graphics,...

Certains packages sont déjà installés avec R mais doivent être "chargés" quand on veut les utiliser avec la fonction library()

(ou require(), qui est quasi synonyme)

```
library(MASS)
library(nlme)
```

Certains packages doivent être installés (une seule fois!) et chargés à chaque fois qu'on en a besoin (fonction library)

```
install.packages(c("lme4", "vegan", "ade4"))
library(lme4)
```

Paquets - Packages

Le package “datasets” est automatiquement chargé au démarrage
Ce package et d'autres packages contiennent des jeux de données qui peuvent être chargés avec la fonction data :

```
> data(iris)
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500

> data(swiss)
> summary(swiss)
   Fertility      Agriculture     Examination     Education       Catholic
 Min.   :35.00   Min.   : 1.20   Min.   : 3.00   Min.   : 1.00   Min.   :  2.150
 1st Qu.:64.70   1st Qu.:35.90   1st Qu.:12.00   1st Qu.: 6.00   1st Qu.:  5.195
 Median :70.40   Median :54.10   Median :16.00   Median : 8.00   Median : 15.140
 Mean   :70.14   Mean   :50.66   Mean   :16.49   Mean   :10.98   Mean   : 41.144
 3rd Qu.:78.45   3rd Qu.:67.65   3rd Qu.:22.00   3rd Qu.:12.00   3rd Qu.: 93.125
 Max.   :92.50   Max.   :89.70   Max.   :37.00   Max.   :53.00   Max.   :100.000
```

Introduction au langage R

Contenu

Qu'est-ce qu'un logiciel libre ?
Langage de programmation vs interface graphique
Avantages/désavantages de R
Les conseils de papa...

Premiers contacts :
interface, console interactive, objets, fonctions, packages

Quelques fonctions de base
(+ notions de vectorisation et de recyclage)

Comment utiliser l'aide ?

Notions de base pour les graphiques

Fonctions de base

Concaténation

```
> a <- c(1, 23, 6, 4, 7)
> b <- c("rouge", "jaune", "rouge", "noir", "rouge")
> c(a, b)
 [1] "1"     "23"    "6"     "4"     "7"     "rouge" "jaune" "rouge" "noir"  "rouge"
```

Imprimer et coller

```
> print(a)
[1]  1 23  6  4  7

> cat("L'objet a contient les nombres suivants : ", a)
L'objet a contient les nombres suivants :  1 23 6 4 7

> paste(a, b, sep="_")
[1] "1_rouge"  "23_jaune" "6_rouge"  "4_noir"   "7_rouge"
```

Caractères spéciaux
tabulation = \t
nouvelle ligne = \n
fin de ligne = \n \r \r\n

Une propriété particulière de R : le recyclage:

```
> paste("couleur :", b)
[1] "couleur : rouge" "couleur : jaune" "couleur : rouge" "couleur : noir"
[5] "couleur : rouge"

> paste(c("bla", "bli"), b) # recyclage !!
[1] "bla rouge" "bli jaune" "bla rouge" "bli noir"  "bla rouge"
```

Longueur = 2

Longueur = 5 → Recyclage du plus petit vecteur !

Fonctions de base

Séquences de nombres

```
> 1:10
 [1]  1  2  3  4  5  6  7  8  9 10

> seq(1, 10, 1)
 [1]  1  2  3  4  5  6  7  8  9 10

> seq(from=1, to=10, by=2)
[1] 1 3 5 7 9

> seq(from=1, to=10, by=0.5)
 [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0
[18]  9.5 10.0
>
> rep(a, 3)
 [1]  1 23  6  4  7  1 23  6  4  7  1 23  6  4  7

> rep(a, each = 3)
 [1]  1  1  1 23 23 23  6  6  6  4  4  4  7  7  7
```

Fonctions de base

```
> (mat <- cbind(a,b, c(1,2)))
     a    b
[1,] "1"  "rouge" "1"
[2,] "23" "jaune" "2"
[3,] "6"  "rouge" "1"
[4,] "4"  "noir"  "2"
[5,] "7"  "rouge" "1"
Message d'avis :
In cbind(a, b, c(1, 2)) :
  number of rows of result is not a multiple of
  vector length (arg 3)
```

cbind() : coller des vecteurs par colonne

Recyclage encore !

```
> rbind(mat, mat)
      a    b
 [1,] "1"  "rouge" "1"
 [2,] "23" "jaune" "2"
 [3,] "6"  "rouge" "1"
 [4,] "4"  "noir"  "2"
 [5,] "7"  "rouge" "1"
 [6,] "1"  "rouge" "1"
 [7,] "23" "jaune" "2"
 [8,] "6"  "rouge" "1"
 [9,] "4"  "noir"  "2"
[10,] "7"  "rouge" "1"
```

rbind() : coller des vecteurs par lignes (row)

```
> t(mat)
  [,1]    [,2]    [,3]    [,4]    [,5]
a "1"     "23"    "6"     "4"     "7"
b "rouge" "jaune" "rouge" "noir" "rouge"
  "1"     "2"     "1"     "2"     "1"
```

t() : transposer une matrice

Fonctions de base

```
> dim(mat)
[1] 5 3

> nrow(mat)
[1] 5
> ncol(mat)
[1] 3
```

dim(), nrow(), ncol() :
nombre de lignes, colonnes, dimensions

```
> colnames(mat)
[1] "a" "b" ""
> rownames(mat)
NULL
```

colnames(), rownames() :
Noms des colonnes et des lignes

Un des rares cas où on assigne une valeur à une fonction:

```
> colnames(mat) <- c("col1", "col2", "blabla")
> rownames(mat) <- paste("row", 1:nrow(mat), sep="_")
> mat
      col1 col2    blabla
row_1 "1"  "rouge" "1"
row_2 "23" "jaune" "2"
row_3 "6"  "rouge" "1"
row_4 "4"  "noir"  "2"
row_5 "7"  "rouge" "1"
```

Fonctions de base

```
> a
[1]  1 23  6  4  7

> length(a)
[1] 5

> sort(a)
[1]  1  4  6  7 23

> sort(b)
[1] "jaune" "noir"  "rouge" "rouge" "rouge"

> rev(sort(a))
[1] 23  7  6  4  1

> sort(a, decreasing=TRUE)
[1] 23  7  6  4  1

> order(a)
[1] 1 4 3 5 2
```

length() : longueur d'un vecteur

sort(), rev(), order() : Trier des valeurs

order() : donne le numéro d'ordre de chaque valeur
--> fonction la plus utilisée

Fonctions de base

```
> b
[1] "rouge" "jaune" "rouge" "noir"  "rouge"

> unique(b)
[1] "rouge" "jaune" "noir"
```

unique() : agrège les valeurs identiques

```
> sample(b, size = 3, replace = FALSE)
[1] "rouge" "jaune" "rouge"
> sample(b, size = 3, replace = FALSE)
[1] "rouge" "jaune" "rouge"
> sample(b, size = 3, replace = FALSE)
[1] "jaune" "rouge" "rouge"
```

sample() : échantillonnage aléatoire. Avec replace=FALSE et size non spécifiée : revient à mélanger les valeurs

Fonctions de base

Opérations algébriques de base :
addition, soustraction, multiplication,
division, exposant

```
> 2+3
[1] 5

> 2-3
[1] -1

> 2*3
[1] 6

> 2/3
[1] 0.6666667

> 2^3
[1] 8
```

Fonctions de base

```
> a
[1]  1 23  6  4  7

> sum(a)
[1] 41

> cumsum(a)
[1]  1 24 30 34 41

> cumprod(a)
[1]    1   23  138  552 3864

> mean(a)
[1] 8.2

> sd(a)
[1] 8.58487

> var(a)
[1] 73.7

> median(a)
[1] 6

> min(a)
[1] 1

> max(a)
[1] 23
```

Diverses fonctions mathématiques

cumsum(), cumprod() : somme et produit cumulés

mean(), sd(), var(): moyenne, écart type (standard deviation), variance

Fonctions de base

Beaucoup de fonctions sont “vectorisées” : s'appliquent à chaque élément d'un vecteur

```
> sqrt(a)
[1] 1.000000 4.795832 2.449490 2.000000 2.645751

> a^0.5
[1] 1.000000 4.795832 2.449490 2.000000 2.645751

> log(a)
[1] 0.000000 3.135494 1.791759 1.386294 1.945910

> log(a, base=10)
[1] 0.0000000 1.3617278 0.7781513 0.6020600 0.8450980

> exp(a)
[1] 2.718282e+00 9.744803e+09 4.034288e+02 5.459815e+01
```

sqrt() : racine carrée, équivalent à a^0.5

log() : par défaut logarithme népérien (base “e”)

exp() : exponentielle = nombre e (= 2.718282) exposant a

```
> abs(c(-1, 1, -2, -3, 5))
[1] 1 1 2 3 5

> cos(a) # sin(a) ; tan(a) ; asin(a) ; etc...
[1]  0.5403023 -0.5328330  0.9601703 -0.6536436  0.7539023
```

abs() : valeur absolue

Trigonométrie...

Fonctions de base

```
> 5/0
[1] Inf
> 0/0
[1] NaN
```

Nombres spéciaux :
Inf = “infini”
NaN = Not a Number

```
> a/3
[1] 0.3333333 7.6666667 2.0000000 1.3333333 2.3333333
```

```
> round(a/3, digits = 2)
[1] 0.33 7.67 2.00 1.33 2.33
```

Arrondis

```
> floor(a/3)
[1] 0 7 2 1 2
```

```
> ceiling(a/3)
[1] 1 8 2 2 3
```

```
> round( c(0.05, 0.15, 0.25, 0.35, 0.45, 0.55), 1)
[1] 0.0 0.2 0.2 0.4 0.4 0.6
```

```
> mean(c(0.05, 0.15, 0.25, 0.35, 0.45, 0.55))
[1] 0.3
> mean(round( c(0.05, 0.15, 0.25, 0.35, 0.45, 0.55), 1))
[1] 0.3
> mean(c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6)) # arrondis comme Excel
[1] 0.35
```

Suit le standard IEC 60559
--> pas la même chose que dans Excel ou Calc ...

Approximation plus exacte !

Introduction au langage R

Contenu

Qu'est-ce qu'un logiciel libre ?
Langage de programmation vs interface graphique
Avantages/désavantages de R
Les conseils de papa...

Premiers contacts :
interface, console interactive, objets, fonctions, packages

Quelques fonctions de base
(+ notions de vectorisation et de recyclage)

Comment utiliser l'aide ?

Notions de base pour les graphiques

Bien utiliser l'aide et les ressources en ligne

!!!! PRIMORDIAL !!!!

Trouver de l'aide sur une fonction dont on connaît le nom :
?nom_de_la_fonction

Dans R Studio : cliquer sur le nom de la fonction puis appuyer sur F1
ou utilisez le champ recherche de l'aide

--> affiche l'aide en HTML

Ou dans la console si on est en mode console (appuyer sur “q” dans ce cas pour quitter l'aide)

Description

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.

Usage

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

read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, comment.char="", ...)

read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",",
          fill = TRUE, comment.char="", ...)

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="", ...)
```

Arguments

file	the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory, getwd(). Tilde-expansion is performed where supported. As from R 2.10.0 this can be a compressed file (see file). Alternatively, file can be a readable text-mode connection (which will be opened for reading if necessary, and if so closed (and hence destroyed) at the end of the function call). (If stdin() is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, Ctrl-D on Unix and Ctrl-Z on Windows. Any pushback on stdin() will be cleared before return.)

Attention aux fonctions génériques :

Comparez l'aide de `plot`, `plot.default` et `plot.lm`

Dans R studio commencez à taper le nom de la fonction dans la recherche de l'aide et R studio vous indique la liste des méthodes disponibles pour cette fonction

Bien utiliser l'aide et les ressources en ligne

Usage : liste des arguments et valeurs par défaut

read.table {utils} R I

Data Input

Description

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.

Usage

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

read.csv(file, header = TRUE, sep = ",", quote="\"", dec=".",
         fill = TRUE, comment.char="", ...)

read.csv2(file, header = TRUE, sep = ";", quote="\"", dec=",",
          fill = TRUE, comment.char="", ...)

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="", ...)
```

Bien utiliser l'aide et les ressources en ligne

On peut aussi utiliser args() si on veut juste la liste des arguments dans la console :

```
> args(read.table)
function (file, header = FALSE, sep = "", quote = "\"'", dec = ".",
    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)
NULL
```

Dans R studio : écrire le nom de la fonction puis appuyer sur tabulation --> liste des arguments et aide

Bien utiliser l'aide et les ressources en ligne

Arguments : détail des arguments, leur signification, les valeurs attendues,...

Arguments

file	the name of the file which the data are to be read from. Each row of the table appears as one line of the file. If it does not contain an *absolute* path, the file name is *relative* to the current working directory, `getwd()`. Tilde-expansion is performed where supported. As from R 2.10.0 this can be a compressed file (see `file`). Alternatively, `file` can be a readable text-mode connection (which will be opened for reading if necessary, and if so `close`d (and hence destroyed) at the end of the function call). (If `stdin()` is used, the prompts for lines may be somewhat confusing. Terminate input with a blank line or an EOF signal, `Ctrl-D` on Unix and `Ctrl-Z` on Windows. Any pushback on `stdin()` will be cleared before return.) `file` can also be a complete URL. (For the supported URL schemes, see the 'URLs' section of the help for `url`.)
header	a logical value indicating whether the file contains the names of the variables as its first line. If missing, the value is determined from the file format: `header` is set to `TRUE` if and only if the first row contains one fewer field than the number of columns.
sep	the field separator character. Values on each line of the file are separated by this character. If `sep = ""` (the default for `read.table`) the separator is 'white space', that is one or more spaces, tabs, newlines or carriage returns.
quote	the set of quoting characters. To disable quoting altogether, use `quote = ""`. See `scan` for the behaviour on quotes embedded in quotes. Quoting is only considered for columns read as character, which is all of them unless `colClasses` is specified.
dec	the character used in the file for decimal points.
row.names	a vector of row names. This can be a vector giving the actual row names, or a single number giving the column of the table which contains the row names, or character string giving the name of the table column containing the row names. If there is a header and the first row contains one fewer field than the number of columns, the first column in the input is used for the row names. Otherwise if `row.names` is missing, the rows are numbered. Using `row.names = NULL` forces row numbering. Missing or `NULL` `row.names` generate row names that are considered to be 'automatic' (and not preserved by `as.matrix`).
col.names	a vector of optional names for the variables. The default is to use `"V"` followed by the column number.

Bien utiliser l'aide et les ressources en ligne

Details : souvent utiles !
Value : le type d'objet et son contenu retournés par la fonction
Divers : ici, pex Memory usage

Details

This function is the principal means of reading tabular data into R.

Unless `colClasses` is specified, all columns are read as character columns and then converted using `type.convert` to logical, integer, numeric, complex or (depending on `as.is`) factor as appropriate. Quotes are (by default) interpreted in all fields, so a column of values like `"42"` will result in an integer column.

A field or line is 'blank' if it contains nothing (except whitespace if no separator is specified) before a comment character or the end of the field or line.

If `row.names` is not specified and the header line has one less entry than the number of columns, the first column is taken to be the row names. This allows data frames to be read in from the format in which they are printed. If `row.names` is specified and does not refer to the first column, that column is discarded from such files.

The number of data columns is determined by looking at the first five lines of input (or the whole file if it has less than five lines), or from the length of `col.names` if it is specified and is longer. This could conceivably be wrong if `fill` or `blank.lines.skip` are true, so specify `col.names` if necessary (as in the 'Examples').

`read.csv` and `read.csv2` are identical to `read.table` except for the defaults. They are intended for reading 'comma separated value' files ('`.csv`') or (`read.csv2`) the variant used in countries that use a comma as decimal point and a semicolon as field separator. Similarly, `read.delim` and `read.delim2` are for reading delimited files, defaulting to the TAB character for the delimiter. Notice that `header = TRUE` and `fill = TRUE` in these variants, and that the comment character is disabled.

The rest of the line after a comment character is skipped; quotes are not processed in comments. Complete comment lines are allowed provided `blank.lines.skip = TRUE`; however, comment lines prior to the header must have the comment character in the first non-blank column.

Quoted fields with embedded newlines are supported except after a comment character.

Value

A data frame (`data.frame`) containing a representation of the data in the file.

Empty input is an error unless `col.names` is specified, when a 0-row data frame is returned: similarly giving just a header line if `header = TRUE` results in a 0-row data frame. Note that in either case the columns will be logical unless `colClasses` was supplied.

Character strings in the result (including factor levels) will have a declared encoding if `encoding` is `"latin1"` or `"UTF-8"`.

Memory usage

These functions can use a surprising amount of memory when reading large files. There is extensive discussion in the 'R Data Import/Export' manual, supplementing the notes here.

Less memory will be used if `colClasses` is specified as one of the six atomic vector classes. This can be particularly so when reading a column that takes many distinct numeric values, as storing each distinct value as a character string can take up to 14 times as much memory as storing it as an integer.

Bien utiliser l'aide et les ressources en ligne

References
See Also : très utile pour trouver d'autres fonctions !
Examples : souvent minimalistes et pas toujours très explicites :-(

Note

The columns referred to in `as.is` and `colClasses` include the column of row names (if any).

Because this function uses `pushBack` it can only handle character strings which can be represented in the current locale. So although `fileEncoding` can be used to specify the encoding of the input file (or a connection can be specified which re-encodes), the implied re-encoding must be possible. This is not a problem in UTF-8 locales, but it can be on Windows — `readLines` or `scan` can be used to avoid this limitation since they have special provisions to convert input to UTF-8.

References

Chambers, J. M. (1992) *Data for models.* Chapter 3 of *Statistical Models in S* eds J. M. Chambers and T. J. Hastie, Wadsworth & Brooks/Cole.

See Also

The 'R Data Import/Export' manual.

`scan`, `type.convert`, `read.fwf` for reading *f*ixed *w*idth *f*ormatted input; `write.table`; `data.frame`.

`count.fields` can be useful to determine problems with reading files which result in reports of incorrect record lengths (see the 'Examples' below).

http://tools.ietf.org/html/rfc4180 for the IANA definition of CSV files (which requires comma as separator and CRLF line endings).

Examples

```
## using count.fields to handle unknown maximum number of fields
## when fill=TRUE
test1 <- c(1:5, "6,7", "8,9,10")
tf <- tempfile()
writeLines(test1, tf)

read.csv(tf, fill = TRUE) # 1 column
ncol <- max(count.fields(tf, sep = ","))
read.csv(tf, fill = TRUE, header = FALSE,
         col.names = paste("V", seq_len(ncol), sep = ""))
unlink(tf)

## "Inline" data set, using text=
## Notice that leading and trailing empty lines are auto-trimmed

read.table(header=TRUE, text="
a b
1 2
3 4
")
```

Bien utiliser l'aide et les ressources en ligne

Aide sur un package :

`help(package="vegan")`

Documentation for package ‘vegan’ version 2.0-5

- DESCRIPTION file.
- Overview of user guides and package vignettes; browse directory.

Help Pages

A B C D E F G H I K L M N O P R S T V W

vegan-package	Community Ecology Package: Ordination, Diversity and Dissimilarities
	-- A --
add1.cca	Add or Drop Single Terms to a Constrained Ordination Model
ade2vegancca	Plot or Extract Results of Constrained Correspondence Analysis or Redundancy Analysis
adipart	Additive Diversity Partitioning and Hierarchical Null Model Testing
adipart.default	Additive Diversity Partitioning and Hierarchical Null Model Testing
adipart.formula	Additive Diversity Partitioning and Hierarchical Null Model Testing
adonis	Permutational Multivariate Analysis of Variance Using Distance Matrices
AIC.radfit	Rank - Abundance or Dominance / Diversity Models
AIC.radfit.frame	Rank - Abundance or Dominance / Diversity Models
alias.cca	Diagnostic Tools for [Constrained] Ordination (CCA, RDA, DCA, CA, PCA)

Liste de toutes les fonctions et datasets + lien vers page d'aide

Vignettes = fichiers pdf décrivant le contenu du package et son utilisation

Souvent un package est aussi associé à une publication décrivant son utilisation

Bien utiliser l'aide et les ressources en ligne

Task views
= Liste des principales ressources dans certains domaines

CRAN Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Computational Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) & Analysis of Experimental Data
Finance	Empirical Finance
Genetics	Statistical Genetics
Graphics	Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning & Statistical Learning
MedicalImaging	Medical Image Analysis

CRAN Task View: Graphic Displays & Dynamic Graphics & Graphic Devices & Visualization

Maintainer: Nicholas Lewin-Koh
Contact: nikko at hailmail.net
Version: 2013-01-29

R is rich with facilities for creating and developing interesting graphics. Base R contains functionality for many plot types including coplots, mosaic plots, biplots, and th postscript, png, jpeg and pdf for outputting graphics as well as device drivers for all platforms running R. lattice and grid are supplied with R's recommended packages lattice is an R implementation of William Cleveland's trellis graphics, while grid defines a much more flexible graphics environment than the base R graphics.

R's base graphics are implemented in the same way as in the S3 system developed by Becker, Chambers, and Wilks. There is a static device, which is treated as a static c through R plotting commands. The device has a set of global parameters such as margins and layouts which can be manipulated by the user using `par()` commands. The visible graphics list, and there is no system of double buffering, so objects cannot be easily edited without redrawing a whole plot. This situation may change in R 2.7.x, buffering for R devices. Even so, the base R graphics can produce many plots with extremely fine graphics in many specialized instances.

One can quickly run into trouble with R's base graphic system if one wants to design complex layouts where scaling is maintained properly on resizing, nested graphs a was designed by Paul Murrell to overcome some of these limitations and as a result packages like lattice, ggplot2, vcd or hexbin (on Bioconductor) use grid for the unde with grid one needs to keep in mind that grid is based on a system of viewports and graphic objects. To add objects one needs to use grid commands, e.g., `grid.polygon()` of viewports from the device and one needs to make sure the desired viewport is at the top of the stack. There is a great deal of explanatory documentation included wi

The graphics packages in R can be organized roughly into the following topics, which range from the more user oriented at the top to the more developer oriented at th exclusive but are for the convenience of presentation:

- *Plotting* : Enhancements for specialized plots can be found in plotrix, for polar plotting, vcd for categorical data, hexbin (on Bioconductor) for hexagon binning, gc plotting enhancements. Some specialized graphs, like Chernoff faces are implemented in aplpack, which also has a nice implementation of Tukey's bag plot. For 3D a selection of plots for different kinds of 3D plotting, scatterplot3d is based on R's base graphics system, while misc3d is based on rgl. The package onion for visuali

Bien utiliser l'aide et les ressources en ligne

Si on ne connaît pas le nom de la fonction :
??”mot-clé”

Par exemple :

```
??"Principal component analysis"
```

The search string was **"Principal component analysis"**

Help pages:

ade4::dudi.fca	Fuzzy Correspondence Analysis and Fuzzy Principal Components Analysis
ade4::dudi.pca	Principal Component Analysis
ade4::pcaiv	Principal component analysis with respect to instrumental variables
ade4::pcaivortho	Principal Component Analysis with respect to orthogonal instrumental variables
ade4::withinpca	Normed within principal component analysis
labdsv::pca	Principal Components Analysis
stats::prcomp	Principal Components Analysis
stats::princomp	Principal Components Analysis
stats::summary.princomp	Summary method for Principal Components Analysis

Fonction “dudi.pca” dans le package “ade4”

Fonction “princomp” dans le package “stats” (package standard chargé au démarrage)

Bien utiliser l'aide et les ressources en ligne

Recherche dans les archives des Forum
--> pratiquement toutes les questions y ont déjà été posées !!

Dans la console :

```
> RSiteSearch("Principal Component analysis")
Une requête de recherche a été soumise à http://search.r-project.org
    La page de résultats devrait s'ouvrir dans votre navigateur
```

R Site Search

Query: Principal Component analysis Search! [How to search]

Display: 20 **Description:** normal **Sort:** by score

Target:
- Functions
- Vignettes
- Task views

For problems WITH THIS PAGE (not with R) contact baron@psych.upenn.edu.

Results:

References:

- **views**: [Principal: 9] [Component: 7] [analysis: 30] [TOTAL: 7]
- **vignettes**: [Principal: 189] [Component: 610] [analysis: 1294] [TOTAL: 125]
- **functions**: [Principal: 1507] [Component: 5712] [analysis (Too many documents hit. Ignored)] [TOTAL: 779]

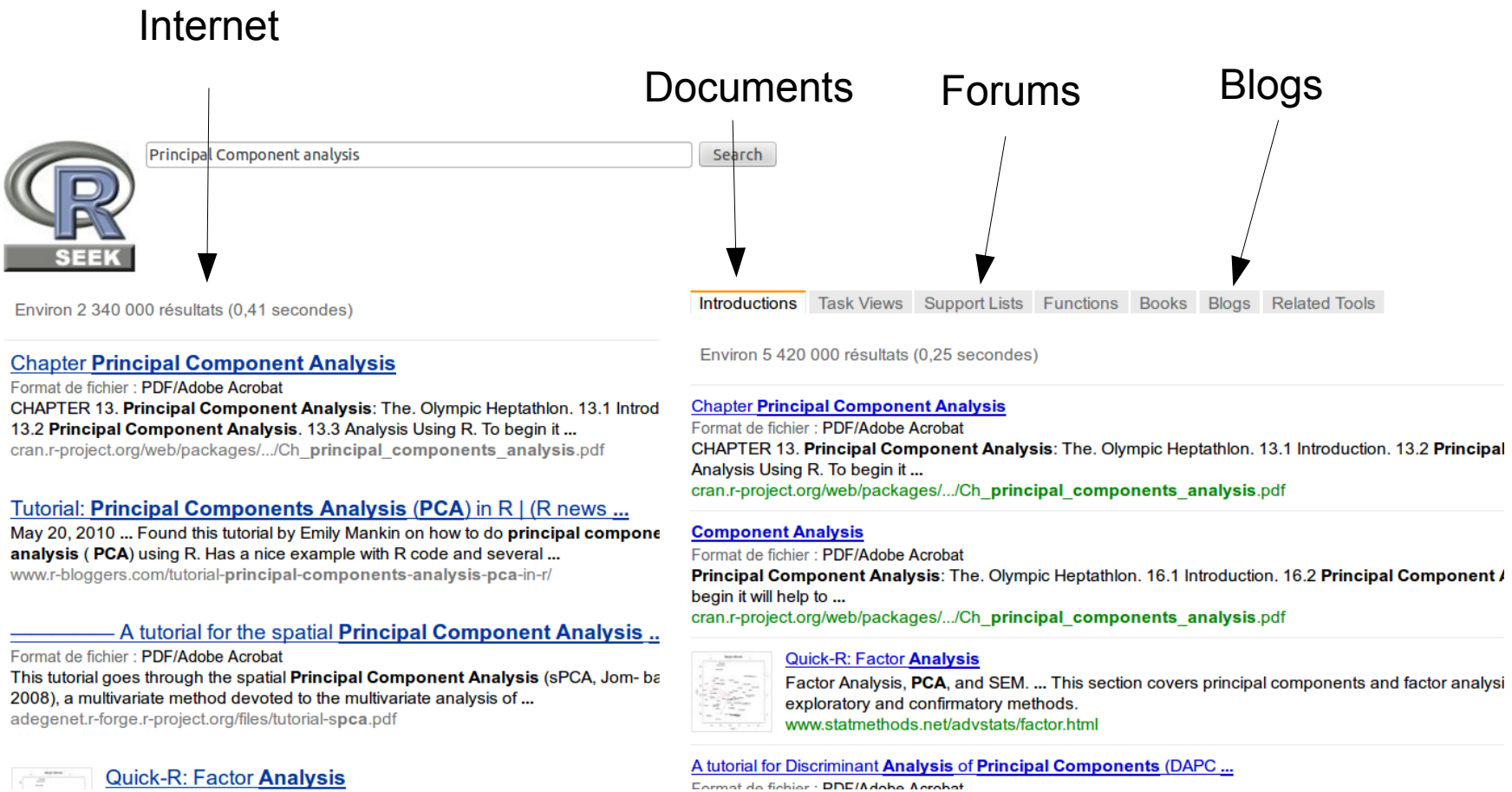
Total 911 documents matching your query.

1. FactoMineR: an R package for multivariate data analysis (score: 38)
   Author: *FranÃ§ois Husson, Julie Josse, JÃ©rÃ´me PagÃ¨s*
   Date: *Tue, 29 Jan 2013 07:28:52 -0500*
   JSS Technical Report â€“ Agrocampus Applied Mathematics Department, September 2010. http://www.agrocampus-ouest.fr/math/ **Principal component** methods - hierarchical
   http://finzi.psych.upenn.edu/R/library/FactoMineR/doc/clustering_and_pricipal_component_methods.pdf (196,236 bytes)

2. R: Plot principal component histograms around a scatter plot (score: 38)
   Author: *unknown*
   Date: *Wed, 22 Feb 2012 19:23:55 -0500*
   Plot **principal component** histograms around a scatter plot Description Usage Arguments Value Note Author(s) See Also Examples page for plotpc {plotpc} plotpc {plotpc} R Docu
   http://finzi.psych.upenn.edu/R/library/plotpc/html/plotpc.html (19,889 bytes)

Bien utiliser l'aide et les ressources en ligne

Recherche dans les archives des Forum et +++
Un google spécifique à R : www.rseek.org



Bien utiliser l'aide et les ressources en ligne

S'y retrouver dans le gestionnaire d'archives de forum...

[R-sig-eco] missing data in PCA

Stéphane Dray dray at biomserv.univ-lyon1.fr
Fri Apr 3 16:34:51 CEST 2009

- Previous message: [R-sig-eco] missing data in PCA
- Next message: [R-sig-eco] missing data in PCA
- **Messages sorted by:** [date] [thread] [subject] [author]

---

ade4 has nipals() which deals with row data.

Cheers.

Jason S wrote:
> *Dear Jari,*
> *>*
> *Thanks for the hints. A package such as mvnmle would be ideal, but it provides only the estimate of the covariance*
> *>*
> *Do you (or anyone else) know a command to do PCA straight from a covariance matrix?*
> *>*
> *best,*
> *>*
> *Jason*
> *>*
> *>*
> *>*
> *>*
> *> _____*
> *From: Jari Oksanen <jari.oksanen at oulu.fi>*
> *>*
> *Cc: Jari Oksanen <jari.oksanen at oulu.fi>; r-sig-ecology at r-project.org*
> *Sent: Friday, April 3, 2009 2:55:20 AM*
> *Subject: Re: [R-sig-eco] missing data in PCA*
> *>*
> *>*
> *On 03/04/2009, at 00:32 AM, Jason S wrote:*
> *>*
> *>*
>> *Dear all,*
>> *>>*
>> *I was wondering if you have good options to deal with missing data on a PCA in R. I guess I could simply delete t*

Pour remonter le fil des messages

Bien utiliser l'aide et les ressources en ligne

S'y retrouver dans le gestionnaire d'archives de forum...

Messages “by thread”

- [R-sig-eco] repeated measures analysis using NP MANOVA (Adonis in vegan package) *Penelope*
- [R-sig-eco] missing data in PCA *Jason S*
  - [R-sig-eco] missing data in PCA *Jari Oksanen*
    - [R-sig-eco] missing data in PCA *Jason S*
      - [R-sig-eco] missing data in PCA *Jari Oksanen*
      - [R-sig-eco] missing data in PCA *Stéphane Dray*
      - [R-sig-eco] missing data in PCA *Farrar.David at epamail.epa.gov*
- [R-sig-eco] Ordination with vegan's metaMDS *Michele Tobias*
  - [R-sig-eco] Ordination with vegan's metaMDS *tyler*
- [R-sig-eco] WG: Re: clustering with random effect *Jana Buerger*
  - [R-sig-eco] WG: Re: clustering with random effect *Jari Oksanen*
- [R-sig-eco] DateTime column required by trip() *Struve, Juliane*
- [R-sig-eco] Subject: Re: missing data in PCA *Nicholas Lewin-Koh*
- [R-sig-eco] About Vegan *mujeeb rahman*
  - [R-sig-eco] About Vegan *stephen sefick*
    - [R-sig-eco] About Vegan *Gavin Simpson*
- [R-sig-eco] about PROTEST *mujeeb rahman*
  - [R-sig-eco] about PROTEST *Stéphane Dray*
- [R-sig-eco] COIA analysis *mujeeb rahman*
- [R-sig-eco] Need an algorithm to identify spatial clusters *Mark Andersen*
  - [R-sig-eco] [R-sig-Geo] Need an algorithm to identify spatial clusters *Miguel Eduardo Gil Biraud*
  - [R-sig-eco] [R-sig-Geo] Need an algorithm to identify spatial clusters *Barry Rowlingson*
  - [R-sig-eco] Need an algorithm to identify spatial clusters *Julian Burgos*
  - [R-sig-eco] [R-sig-Geo] Need an algorithm to identify spatial clusters *van Etten, Jacob (IRRI)*

Bien utiliser l'aide et les ressources en ligne

Autres forum très utiles :

Groupe des Utilisateurs du logiciel R
En français, en général plus “cool” que sur R-help
http://forums.cirad.fr/logiciel-R/viewforum.php?f=3

Probablement une des meilleures bases de connaissances :

http://stackoverflow.com/
Forum de programmeurs, pas spécifique à R mais très actif

CrossValidated
http://stats.stackexchange.com/
Pour poser des questions de statistiques
(pas de programmation)

Bien utiliser l'aide et les ressources en ligne

Si vous ne trouvez vraiment pas, vous pouvez toujours poser votre question sur un forum. Toute une communauté est prête à aider mais il y a quelques règles à respecter :

1) Avoir cherché dans l'aide
(sous peine d'un laconique « RTFM » ou pire...)

2) Avoir cherché dans les forum et sur le web
(sous peine d'un laconique « STFW » ou pire...)

3) ne pas poser de question stats sur un forum dédié à R uniquement

4) Poser une question précise et claire,
Montrer ce que vous avez déjà essayé
Fournir un exemple reproductible
(ie utiliser dput() pour fournir un extrait de vos données)

5) Savoir que le ton est parfois un peu abrupt
Rester poli quand même...

Voir E. Raymond : “How To Ask Questions The Smart Way” : http://www.catb.org/esr/faqs/smart-questions.html

Bien utiliser l'aide et les ressources en ligne

Autres ressources utiles :

http://www.r-bloggers.com/
Agrégateur de Blogs sur R
Très utile pour suivre l'évolution de R et admirer la diversité des applications possibles

http://pbil.univ-lyon1.fr/R/
Statistiques et Biologie – Université de Lyon
Nombreuses ressources pédagogiques

« Aide avancée » : trouver le code d'une fonction

Si on veut savoir comment un calcul est fait, pas besoin de manuel, on va voir dans le code ...
On tape le nom de la fonction sans parenthèses :

```
> lm
function (formula, data, subset, weights, na.action, method = "qr",
    model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,
    contrasts = NULL, offset, ...)
{
    ret.x <- x
    ret.y <- y
    cl <- match.call()
    mf <- match.call(expand.dots = FALSE)
    m <- match(c("formula", "data", "subset", "weights", "na.action",
        "offset"), names(mf), 0L)
    mf <- mf[c(1L, m)]
(...)
```

« Aide avancée » : trouver le code d'une fonction

Avec les fonctions génériques ça ne fonctionne pas :

```
> summary
function (object, ...)
UseMethod("summary")
<bytecode: 0x419ffd0>
<environment: namespace:base>
```

On doit indiquer le nom de la méthode :

```
> methods(summary)
(...)
[28] summary.factor          summary.ggplot*          summary.glht*
[31] summary.glm             summary.gls*             summary.infl
[34] summary.lm              summary.lme*             summary.lmList*
(...)
> summary.lm
function (object, correlation = FALSE, symbolic.cor = FALSE,
    ...)
{
    z <- object
    p <- z$rank
    rdf <- z$df.residual
    if (p == 0) {
(...)
```

Tip :
Dans R studio commencez à taper le nom de la fonction dans la recherche de l'aide et R studio vous indique la liste des méthodes disponibles pour cette fonction

Si ça ne fonctionne pas, indiquer le package où se trouve la méthode de la fonction ou utiliser `getAnywhere` :

```
> stats:::print.summary.lm # ou getAnywhere(print.summary.lm)
> car:::Anova.lm  # ou getAnywhere(Anova.lm)
> lme4:::summary.merMod  # ou getAnywhere(summary.merMod)
>
```

Introduction au langage R

Contenu

Qu'est-ce qu'un logiciel libre ?
Langage de programmation vs interface graphique
Avantages/désavantages de R
Les conseils de papa...

Premiers contacts :
interface, console interactive, objets, fonctions, packages

Quelques fonctions de base
(+ notions de vectorisation et de recyclage)

Comment utiliser l'aide ?

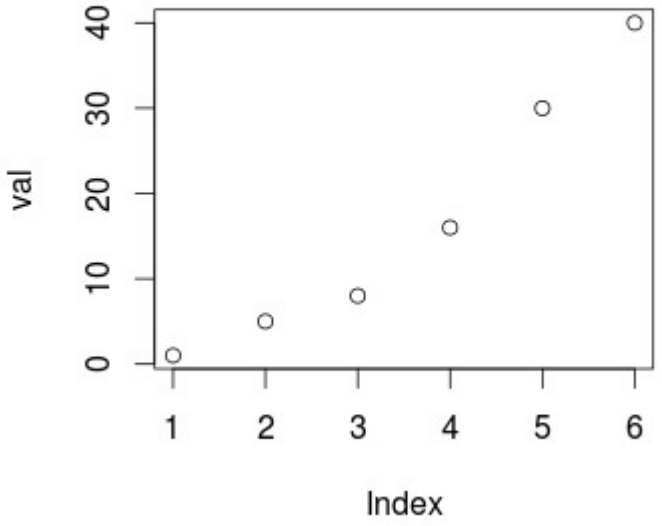
Notions de base pour les graphiques

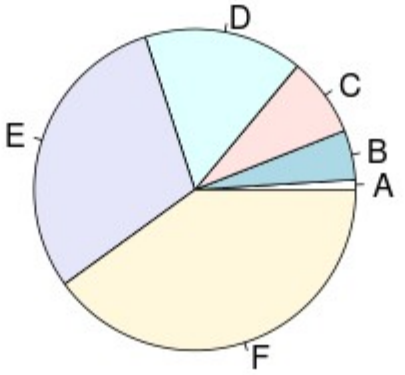
Notions de base sur les graphiques

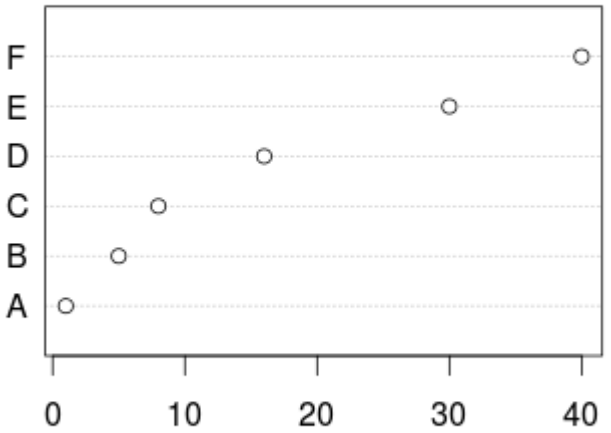
Notions très rapides, approfondies plus tard...

Différentes fonctions graphiques pour différents graphiques :

```
> val <- c(1, 5, 8, 16, 30, 40 )
>
> plot(val)
>
> pie(val, labels = c("A", "B", "C", "D", "E", "F"))
>
> dotchart(val, labels = c("A", "B", "C", "D", "E", "F"))
```





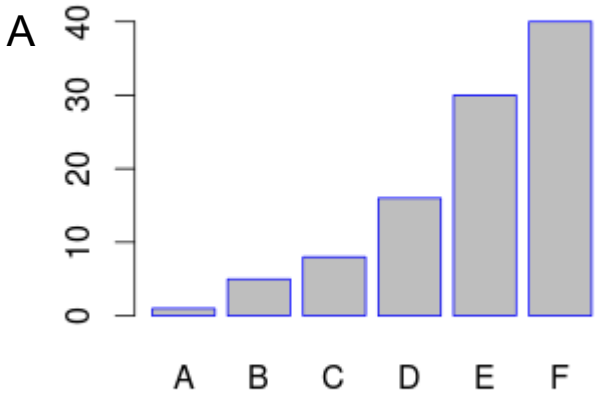


Notions de base sur les graphiques

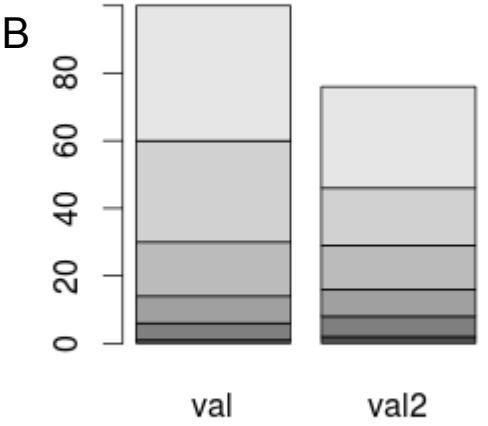
Le type de graphique dépend du type de données ou de leur mise en forme + options

```
> val <- c(1, 5, 8, 16, 30, 40 )
> val2 <- c(2, 6, 8, 13, 17, 30)
A > barplot(val, names.arg= c("A", "B", "C", "D", "E", "F"), border= "blue")
B > barplot(cbind(val, val2))
C > barplot(cbind(val, val2), beside = TRUE)
D > barplot(t(cbind(val, val2)), beside = TRUE)
```

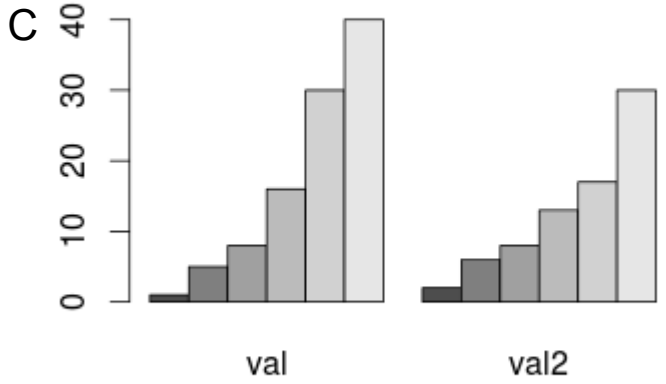
A



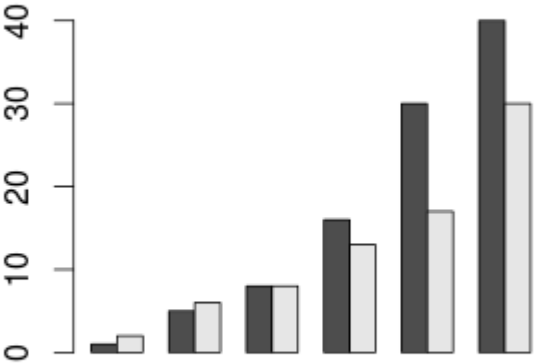
B



C



D



Notions de base sur les graphiques

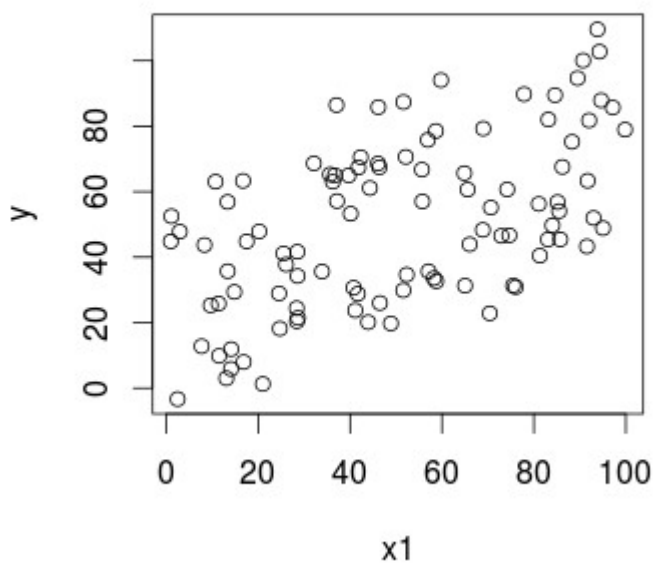
Le type de graphique dépend du type de données ou de leur mise en forme + options

```
> x1 <- runif(100, 0, 100)
> x2 <- factor(rep(c("male", "female"), each = 50))
> y <- 5 + 0.5 * x1 + 40 * (unclass(x2)-1) + rnorm(100, 0, 10)
>
A > plot(y ~ x1)
B > plot(y ~ x2)
```

← Notation en “formule”, identique à `plot(y=y, x=x1)`

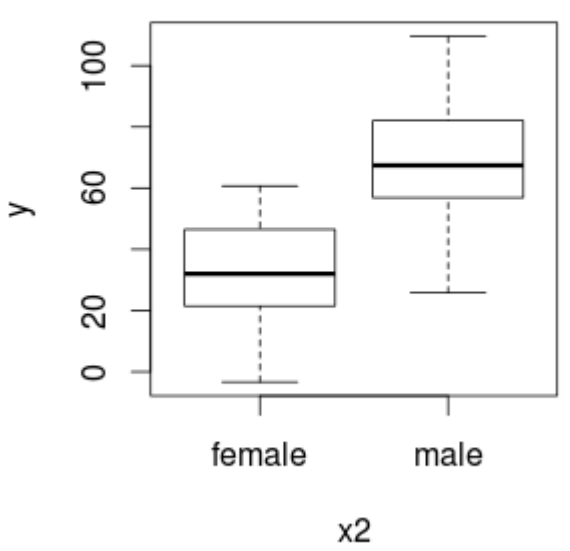
X1 numeric → scatterplot

A



X2 catégoriel → boxplot

B



Notions de base sur les graphiques

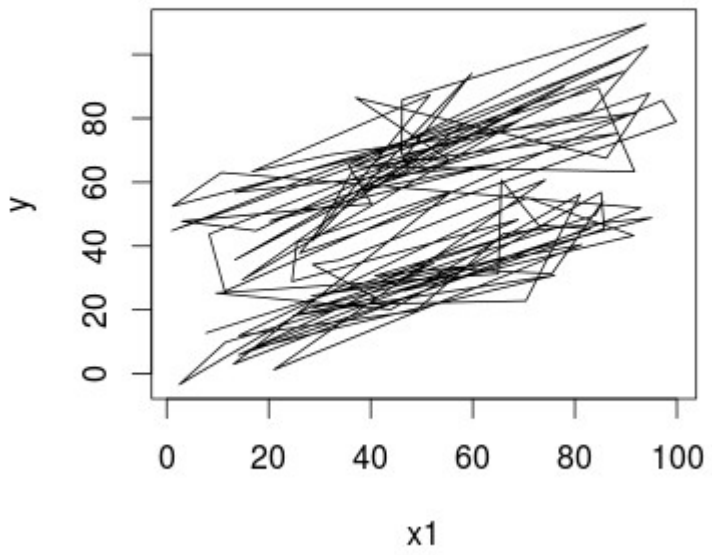
Arguments des fonctions graphiques

L'aide est plus utile que jamais !!

A
```
> plot(y ~ x1, type = "l")
```
B
```
> plot(y ~ x1, ylim = c(-20, 100), ylab = "mes valeurs de y" , pch = 3,
    xlab = "mes valeurs de x", )
```

Un graphique sans intérêt.

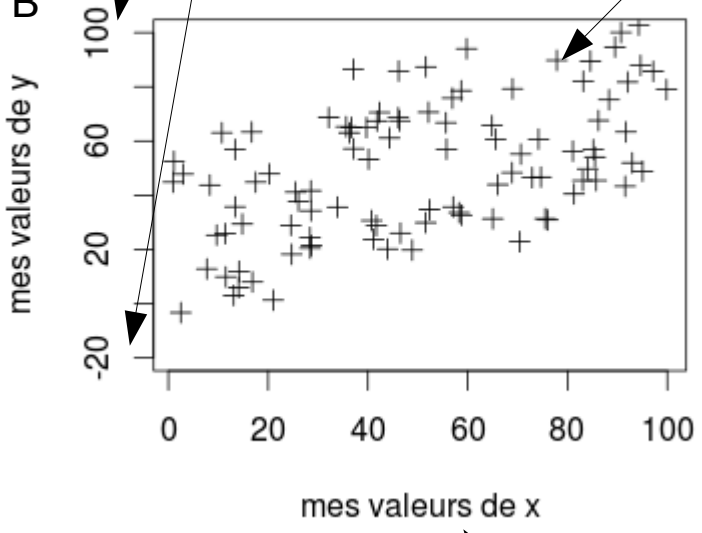
A



ylim= c(-20,100)

Pch = 3 (“point character”)
Voir ?points pour les
différentes valeurs possibles

B



xlab = "mes valeurs de x"

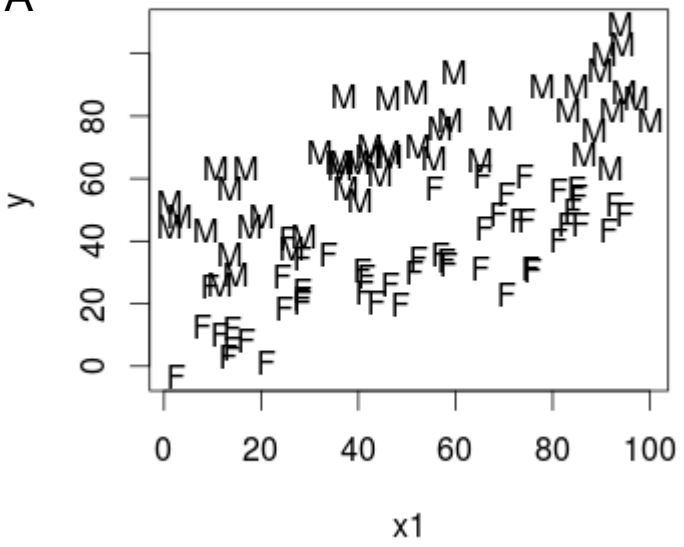
Notions de base sur les graphiques

Fonctions de bas niveau permettant de modifier un graphique existant

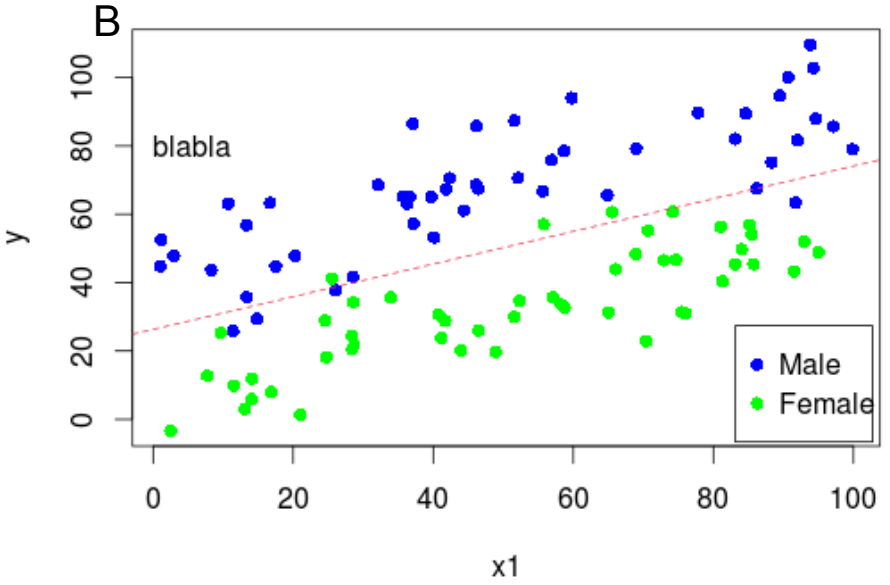
A
```
> plot(y ~ x1, pch = rep(c("M", "F"), each = 50))
```

B
```
> plot(y ~ x1, pch= 16, col = rep(c("blue", "green"), each = 50))
> text(x=0, y = 80, labels= "blabla", adj=0)
> abline(lm(y~x1), col = rgb(234, 65, 87, maxColorValue = 256), lty = 2)
> legend(x= "bottomright", legend = c("Male", "Female"), col = c("blue",
"green"), pch= 16, inset = 0.01 )
```

A



B



Exercices de prise en main !

Manipulation de données avec le langage R

G. San Martin

gilles.sanmartin@gmail.com
Centre Wallon de Recherche Agronomique

cra-w

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de dates et caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

Le langage R

Contenu

Pas très marrant !

Les objets

Importer des données

Pas très marrant non plus ...

Extraire des données (subscripting)

Manipulation de dates et caractères

Expressions régulières : très puissant mais on peut totalement s'en passer

Là ça va commencer à devenir amusant ! Courage !!!

Fusionner des tableaux de données

R comme langage de script

(structures de contrôle : boucles, fonctions, etc...)

Agrégation et « reshaping »

Et là c'est le Nirvana !!

Les graphiques

La création de rapports

Un autre Nirvana (mais la route est longue...)

Le langage R : Les objets

Contenu

Pourquoi commencer par ça ?

Principaux modes :
numeric, character, logical

Gestion des valeurs manquantes (NA)

Principaux types d'objets (~classes) :
~ « vector», matrix, list, data.frame

Classes dérivées
(exemple : objet contenant le résultat d'une régression)

Classes particulières pour gérer les facteurs et les dates

Mais pourquoi commencer par ça ??

Principales sources d'erreur dans R :

1) fautes de frappe, majuscules/minuscules,
oubli de parenthèses ou virgules, etc...
--> facile à corriger

2) les données ne sont pas dans le format attendu par la fonction
--> très important de comprendre les différences entre types d'objets et
de pouvoir passer de l'un à l'autre

--> Partie pas très marrante, un peu abstraite, mais nécessaire...

Essayez de retenir les grands principes - Revenir ici pour les détails
(pex pour les manipulations sur les dates et facteurs)

Objets

Tout est stocké dans des objets :
données, fonctions, résultats d'analyses, ...

Objet :
unité de stockage désignée par son nom et possédant certaines caractéristiques (« attributs ») :
dimensions, noms, classe, mode,...

Classe et mode

Mode = type de contenu (type de données)
Classe = type de contenant (type d'objet)

Certaines classes peuvent contenir des données de plusieurs modes différents, d'autres pas.

Principaux modes :
numeric - character - logical

Principales classes :
vector*, matrix (+array), list, data.frame
+ factor, Date
+ beaucoup d'autres ...

*qui n'est en fait pas une classe mais le type d'objet élémentaire dans R sur lequel sont basés tous les autres. La classe d'un vecteur est égale à son mode

Principaux modes

```
> nbr <- c(1, 3, 2.5, 8.9, 90)
> txt <- c("noir", "jaune", "rouge", "jaune", "noir")
> vf <- c(TRUE, FALSE, FALSE, FALSE, TRUE)

> mode(nbr)
[1] "numeric"

> mode(txt)
[1] "character"

> mode(vf)
[1] "logical"
```

Les "character" sont toujours entourés de guillemets :

```
> mode(c("1", "2", "3.5"))
[1] "character"

> mode(c("TRUE", "FALSE"))
[1] "character"
```

Principaux modes

Les éléments d'un vecteur doivent tous être du même mode

```
> (new <- c(nbr, txt))
 [1] "1"     "3"     "2.5"   "8.9"   "90"    "noir"  "jaune" "rouge"
"jaune" "noir"
> mode(new)
[1] "character"

> (new <- c(vf, txt))
 [1] "TRUE"  "FALSE" "FALSE" "FALSE" "TRUE"  "noir"  "jaune" "rouge"
"jaune" "noir"
> mode(new)
[1] "character"

> (new <- c(nbr, vf))
 [1]  1.0  3.0  2.5  8.9 90.0  1.0  0.0  0.0  0.0  1.0
> mode(new)
[1] "numeric"
```

Les vecteurs logiques sont transformés en nombres

Particularité du mode logique : TRUE = 1, FALSE = 0

```
> sum(vf)
[1] 2
```

Passer d'un mode à l'autre

```
> as.character(nbr)
[1] "1"   "3"   "2.5" "8.9" "90"

> as.numeric(vf)
[1] 1 0 0 0 1

> as.logical(c(1, 0, 0, 1, 0))
[1]  TRUE FALSE FALSE  TRUE FALSE
```

Impossible de transformer un vecteur texte en numérique
--> valeurs manquantes « NA » (« Not Available »)

```
> as.numeric(txt)
[1] NA NA NA NA NA
Message d'avis :
NAs introduits lors de la conversion automatique
```

Mais OK si le texte contient uniquement des chiffres :

```
> (a <- c("3", "8.7", "4.2"))
[1] "3"   "8.7" "4.2"
> as.numeric(a)
[1] 3.0 8.7 4.2
```

Autres modes

Il existe d'autres modes
généralement pas utiles de manière directe à l'utilisateur

```
> mode(mean)
[1] "function"

> mode(y~x)
[1] "call"
```

Les valeurs manquantes NA

NA pour « Not Available »
Leur gestion adéquate est primordiale en analyse de données.
Une valeur manquante n'est pas un 0 !!

```
> a <- c(1, 2, NA, 3, 2/0, 3, 4, 0/0, log(0))
> a
[1]    1    2   NA    3  Inf    3    4  NaN -Inf

> is.na(a)
[1] FALSE FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE
> is.nan(a)
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE
```

Pas de guillemets !

```
> a <- c(1, 2, NA, 3, "NA", 4)
> a
[1] "1"  "2"  NA   "3"  "NA" "4"
```

Les valeurs manquantes NA

Les NA se propagent dans les calculs.

Il existe en général des arguments spécifiques pour la gestion des NA

Voir aussi argument na.action dans les régressions et les `options()`

```
> a <- c(1, 2, NA, 3, NA, 4)
> mean(a)
[1] NA
> sum(a)
[1] NA

> mean(a, na.rm = TRUE)
[1] 2.5
> sum(a, na.rm = TRUE)
[1] 10

> na.omit(a)
[1] 1 2 3 4
attr(,"na.action")
[1] 3 5
attr(,"class")
[1] "omit"
```

Permet d'éliminer les valeurs NA

Classes

Principales classes pour contenir des données :

Vecteur* :

Suite d'éléments unitaires, tous du même mode (1 dimension)

Matrice :

Vecteur à 2 dimensions (un seul mode donc)
(Array = vecteur à n dimensions)

Liste :

Suite d'éléments de classe, de mode et de dimension quelconque

Data.frame :

Liste composée de vecteurs** de même longueur
Le data.frame ressemble donc à une matrice (tableau à 2 dimensions)
mais chaque colonne peut avoir un mode différent

*Comme déjà dit : il n'existe pas réellement de classe "vector", c'est le type d'objet de base
** Presque toujours...

Classes

4 fonctions pour créer ces objets
(avec options par défaut)

```
> (v <- vector(mode = "logical", length = 0))
Logical(0)

> (m <- matrix(data = NA, nrow = 1, ncol = 1,
    byrow = FALSE, dimnames = NULL))
      [,1]
[1,]   NA

> (l <- list())
List()

> (df <- data.frame())
tableau de données (data frame) avec 0 colonnes et 0 lignes
```

Fonctions pour passer d'une classe à l'autre :

`as.vector, as.matrix, as.list, as.data.frame`

Vecteurs

```
> (a <- vector(length=5))
[1] FALSE FALSE FALSE FALSE FALSE

> (b <- c("blanc", "bleu", "rouge"))
[1] "blanc" "bleu"  "rouge"

> (c <- 1:10)
 [1]  1  2  3  4  5  6  7  8  9 10

> names(b)
NULL
> dim(b)
NULL
> mode(b)
[1] "character"
> class(b)
[1] "character"

> names(b) <- c("col1", "col2", "col3")
> b
   col1    col2    col3
"blanc"  "bleu" "rouge"
```

4 propriétés du vecteur « b »

On peut modifier ces propriétés

Matrices

On peut aussi créer des matrices avec cbind et rbind

```
> (mat <- cbind(a,b))
      a       b
[1,] "FALSE" "blanc"
[2,] "FALSE" "bleu"
[3,] "FALSE" "rouge"
[4,] "FALSE" "blanc"
[5,] "FALSE" "bleu"
Message d'avis :
In cbind(a, b) :
  number of rows of result is not a multiple of
   vector length (arg 2)
```

Recyclage !

```
> rbind(mat,mat)
      a       b
 [1,] "FALSE" "blanc"
 [2,] "FALSE" "bleu"
 [3,] "FALSE" "rouge"
 [4,] "FALSE" "blanc"
 [5,] "FALSE" "bleu"
 [6,] "FALSE" "blanc"
 [7,] "FALSE" "bleu"
 [8,] "FALSE" "rouge"
 [9,] "FALSE" "blanc"
[10,] "FALSE" "bleu"
```

Matrices

On peut aussi créer des matrices avec matrix()

```
> matrix(1:10, nrow=2, ncol=5)
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

> (mat <- matrix(1:10, nrow=2, ncol=5, byrow = TRUE))
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10


> names(mat)
NULL
> dim(mat)
[1] 2 5
> mode(mat)
[1] "numeric"
> class(mat)
[1] "matrix"
>
```

4 attributs de la matrice « mat »

Matrices

Modification des noms

```
> mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10

> colnames(mat) <- c("a", "b", "c", "d", "e")
> rownames(mat) <- c("A", "B")

> (n <- paste(rep(rownames(mat), 5), rep(colnames(mat), each=2),
        sep=""))
 [1] "Aa" "Ba" "Ab" "Bb" "Ac" "Bc" "Ad" "Bd" "Ae" "Be"
> names(mat) <- n

> mat
  a b c d  e
A 1 2 3 4  5
B 6 7 8 9 10
attr(,"names")
 [1] "Aa" "Ba" "Ab" "Bb" "Ac" "Bc" "Ad" "Bd" "Ae" "Be"
```

Rarement utilisé pour les matrices

Matrices

Modification des dimensions et transformation en vecteur

```
> dim(mat) <- c(5,2)
> mat
     [,1] [,2]
[1,]    1    8
[2,]    6    4
[3,]    2    9
[4,]    7    5
[5,]    3   10
>
```

Rarement utilisé

Transformation vers d'autres format

```
> as.vector(mat)
 [1]  1  6  2  7  3  8  4  9  5 10
```

Listes

Création d'une liste composée de 3 vecteurs et d'une matrice :

```
> list(a,b,c,mat)

[[1]]
[1] FALSE FALSE FALSE FALSE FALSE

[[2]]
   col1    col2    col3
"blanc"  "bleu" "rouge"

[[3]]
 [1]  1  2  3  4  5  6  7  8  9 10

[[4]]
     [,1] [,2]
[1,] "1"  "8"
[2,] "6"  "4"
[3,] "2"  "9"
[4,] "7"  "5"
[5,] "3"  "10"
```

Listes

Chaque élément peut être nommé

```
> (l <- list(logique = a, couleurs = b, nbr = c, mat = mat))
$logique
[1] FALSE FALSE FALSE FALSE FALSE

$couleurs
   col1    col2    col3
"blanc"  "bleu" "rouge"

$nbr
 [1]  1  2  3  4  5  6  7  8  9 10

$mat
     [,1] [,2]
[1,] "1"  "8"
[2,] "6"  "4"
[3,] "2"  "9"
[4,] "7"  "5"
[5,] "3"  "10"
```

Listes

Liste composée d'un vecteur et d'une liste
elle-même composée de 3 vecteurs et une matrice...

```
> list(a= a, maliste = l)
$a
[1] FALSE FALSE FALSE FALSE FALSE

$maliste
$maliste$logique
[1] FALSE FALSE FALSE FALSE FALSE

$maliste$couleurs
   col1    col2    col3
"blanc"  "bleu" "rouge"

$maliste$nbr
 [1]  1  2  3  4  5  6  7  8  9 10

$maliste$mat
     [,1] [,2]
[1,] "1"  "8"
[2,] "6"  "4"
[3,] "2"  "9"
[4,] "7"  "5"
[5,] "3"  "10"
```

Listes

Liste composée d'un vecteur et d'une liste elle-même composée de 3 vecteurs et une matrice...

```
> dim(l)
NULL
> dim(l$mat)
[1] 5 2
> length(l)
[1] 4

> names(l)
[1] "logique"  "couleurs" "nbr"      "mat"

> mode(l)
[1] "list"
> mode(l$mat)
[1] "character"

> class(l)
[1] "list"
> class(l$mat)
[1] "matrix"
```

N'existe pas pour les listes

L'opérateur $ permet d'extraire un élément de la liste

Listes

Modification des noms

```
> names(l)
[1] "logique"  "couleurs" "nbr"      "mat"
> names(l) <- c("ba", "be", "bi", "bo")
> l
$ba
[1] FALSE FALSE FALSE FALSE FALSE

$be
   col1    col2    col3
"blanc"  "bleu" "rouge"

$bi
 [1]  1  2  3  4  5  6  7  8  9 10

$bo
     [,1] [,2]
[1,] "1"  "8"
[2,] "6"  "4"
[3,] "2"  "9"
[4,] "7"  "5"
[5,] "3"  "10"
```

Listes

Transformation de vecteurs en listes

```
> b
   col1    col2    col3
"blanc"  "bleu" "rouge"

> as.list(b)
$col1
[1] "blanc"

$col2
[1] "bleu"

$col3
[1] "rouge"
```

Listes

Transformation de vecteurs en listes

```
> mat
     [,1] [,2]
[1,] "1"  "8"
[2,] "6"  "4"
[3,] "2"  "9"
[4,] "7"  "5"
[5,] "3"  "10"

> as.list(mat)
[[1]]
[1] "1"

[[2]]
[1] "6"

[[3]]
[1] "2"

[[4]]
[1] "7"

(...)
```

Listes

as.vector() ne fonctionne pas sur une liste
--> on utilise unlist()

```
> l
$ba
[1] FALSE FALSE FALSE FALSE FALSE

$be
   col1    col2    col3
"blanc"  "bleu" "rouge"

$bi
 [1]  1  2  3  4  5  6  7  8  9 10


> unlist(l)

    ba1     ba2     ba3     ba4     ba5 be.col1 be.col2 be.col3     bi1
"FALSE" "FALSE" "FALSE" "FALSE" "FALSE" "blanc"  "bleu" "rouge"     "1"
    bi2     bi3     bi4     bi5     bi6     bi7     bi8     bi9    bi10
    "2"     "3"     "4"     "5"     "6"     "7"     "8"     "9"    "10"
```

Data frames

Liste composée de vecteurs (en général) de même longueur
Le data.frame ressemble donc à une matrice (tableau à 2 dimensions)
mais chaque colonne peut avoir un mode différent

C'est la classe standard pour stocker les jeux de données
Comparaison avec matrices

```
> a <- 1:5
> b <- c(TRUE, FALSE, FALSE, FALSE, TRUE)
> c <- c("a", "b", "c", "d", "e")
>
>
> (df <- data.frame(a=a, b=b, c=c))
  a     b c
1 1  TRUE a
2 2 FALSE b
3 3 FALSE c
4 4 FALSE d
5 5  TRUE e

> (mat <- cbind(a, b, c))
     a   b       c
[1,] "1" "TRUE"  "a"
[2,] "2" "FALSE" "b"
[3,] "3" "FALSE" "c"
[4,] "4" "FALSE" "d"
[5,] "5" "TRUE"  "e"
```

Chaque colonne a un mode différent

Tout est « character »

Data frames

Comparaison avec matrices

```
> sum(df$a)
[1] 15

> sum(mat$a)
Erreur dans mat$a : $ operator is invalid for atomic vectors

> mat[,"a"]
[1] "1" "2" "3" "4" "5"

> sum(mat[,"a"])
Erreur dans sum(mat[, "a"]) : 'type' (character) de l'argument incorrect
```

Opérateur $ uniquement pour liste et donc data.frame

Équivalent du $ pour les matrices

Ne fonctionne pas malgré tout car tout est « character »

Data frames

Transformation d'une matrice en data.frame
Les character sont considérés comme des facteurs par défaut

```
> (df <- as.data.frame(mat))
  a     b c
1 1  TRUE a
2 2 FALSE b
3 3 FALSE c
4 4 FALSE d
5 5  TRUE e

> sum(df$a)
Erreur dans Summary.factor(1:5, na.rm = FALSE) :
  sum ceci n'est pas pertinent pour des variables facteurs

> class(df$a)
[1] "factor"
```

Autres classes

La plupart des résultats d'analyses sont stockés dans des objets de classe particulière similaires à une liste

Exemple avec une régression linéaire :

```
> x <- runif(100, 0, 100)
> y <- 0.2*x + rnorm(100)

> mod <- lm(y ~ x)

> class(mod)
[1] "lm"
```

Autres classes

Pour visualiser la structure de ces résultats : `str()`

Fonction très très utile !!

```
> str(mod)

List of 12
 $ coefficients : Named num [1:2] -0.315 0.206
  ..- attr(*, "names")= chr [1:2] "(Intercept)" "x"
 $ residuals    : Named num [1:100] 2.084 -1.413 -0.619 1.337 0.593 ...
  ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
 $ effects      : Named num [1:100] -99.083 56.534 -0.893 0.881 0.201 ...
  ..- attr(*, "names")= chr [1:100] "(Intercept)" "x" "" "" ...
 $ rank         : int 2
 $ fitted.values: Named num [1:100] 3.39 2.78 12.64 19.82 17.3 ...
  ..- attr(*, "names")= chr [1:100] "1" "2" "3" "4" ...
 $ assign       : int [1:2] 0 1
 $ qr           :List of 5
  ..$ qr   : num [1:100, 1:2] -10 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 0.1 ...
  .. ..- attr(*, "dimnames")=List of 2
(...)
```

Autres classes

On peut alors facilement en extraire les résultats voulus

```
> mod$coefficients
(Intercept)           x
 -0.3146324   0.2056530

> mod$df.residual
[1] 98
```

NB : il vaut mieux utiliser les fonctions génériques d'extraction quand elles existent (plus grande garantie de pérennité):

```
> coef(mod)
(Intercept)           x
 -0.3146324   0.2056530
```

Facteurs

Classe particulière dédiée aux variables catégorielles
(= nominales, discrètes)

Il s'agit d'un vecteur de texte stocké sous forme de nombres
avec un attribut particulier "levels" (niveaux)
décrivant les différentes valeurs possibles du facteur.

Les facteurs sont traités différemment par de
nombreuses fonctions/analyses

Facteurs

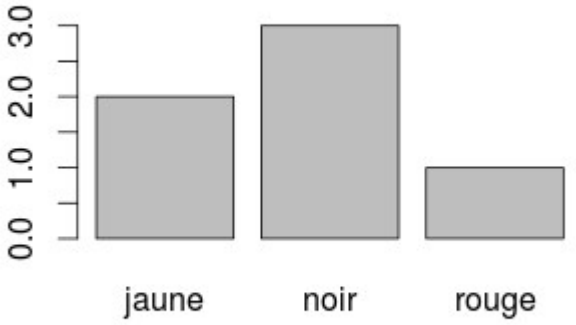
Comparaison d'un facteur et d'un vecteur de texte

```
> txt <- c("noir", "jaune", "rouge", "jaune", "noir", "noir")
> fac <- factor(txt)
>
> txt
[1] "noir"  "jaune" "rouge" "jaune" "noir"  "noir"
> fac
[1] noir  jaune rouge jaune noir  noir
Levels: jaune noir rouge

> summary(txt)
   Length     Class      Mode
        6 character character
> summary(fac)
jaune  noir rouge
    2     3     1
```

Facteurs

Comparaison d'un facteur et d'un vecteur de texte

```
> plot(txt)
Erreur dans plot.window(...) : valeurs finies requises pour 'ylim'
De plus : Messages d'avis :
1: In xy.coords(x, y, xlabel, ylabel, log) :
  NAs introduits lors de la conversion automatique
2: In min(x) : aucun argument trouvé pour min ; Inf est renvoyé
3: In max(x) : aucun argument pour max ; -Inf est renvoyé

> plot(fac)
```



Facteurs

On ne peut pas ajouter de valeurs ne se trouvant pas dans "levels"

```
> a <- c("a", "b", "c")

> c(txt, a)
[1] "noir"  "jaune" "rouge" "jaune" "noir"  "noir"  "a"     "b"     "c"

> c(fac, a)
[1] "2" "1" "3" "1" "2" "2" "a" "b" "c"
```

Le facteur a été converti en nombre puis en texte à la volée
--> ce n'est pas le comportement recherché !

```
> txt[1] <- "bleu"
> txt
[1] "bleu"  "jaune" "rouge" "jaune" "noir"  "noir"

> fac[1] <- "bleu"
Message d'avis :
In `[<-.factor`(`*tmp*`, 1, value = "bleu") :
  invalid factor level, NAs generated

> fac
[1] <NA>  jaune rouge jaune noir  noir
Levels: jaune noir rouge
```

La valeur a été remplacée par une valeur manquante

Facteurs

Pour pouvoir ajouter une nouvelle valeur à un facteur :

```
> levels(fac) <- c(levels(fac), "bleu")
> fac[1] <- "bleu"
> fac
[1] bleu  jaune rouge jaune noir  noir
Levels: jaune noir rouge bleu
```

Autre solution :

```
>
> fac2 <- as.character(fac)
> fac2 <- c(fac2, a)
> fac2 <- factor(fac2)
> fac2
[1] bleu  jaune rouge jaune noir  noir  a     b     c
Levels: a b bleu c jaune noir rouge
```

Facteurs

Modifier les niveaux du facteur

```
> fac
[1] noir  jaune rouge jaune noir  noir
Levels: jaune noir rouge

> levels(fac) <- c("indigo", "jaune paille", "sombre","ocre")

> fac
[1] jaune paille indigo sombre  indigo  jaune paille jaune paille
Levels: indigo jaune paille sombre ocre
```

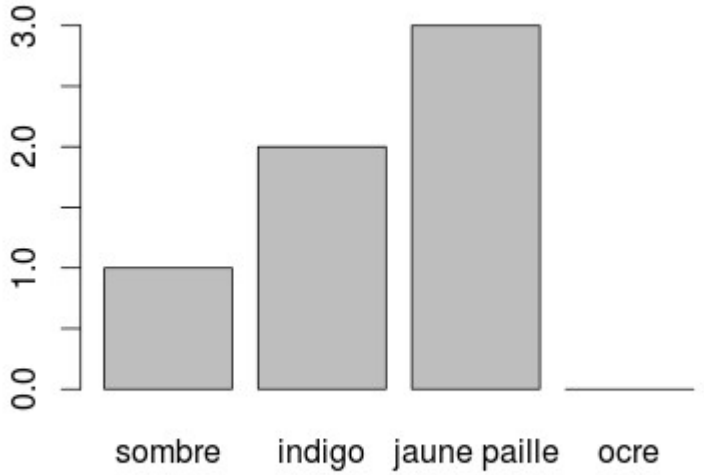
Facteurs

Changer l'ordre des niveaux

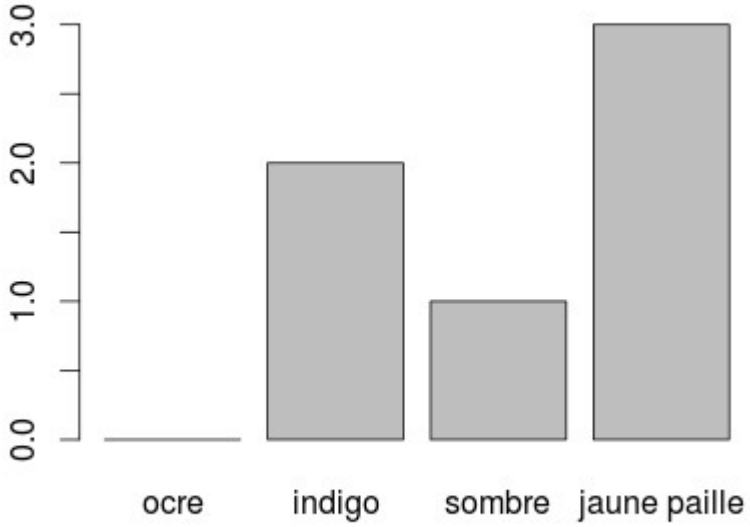
L'ordre des niveaux peut avoir une influence sur les résultats (graphes, analyses etc..)

```
> fac <- relevel(fac, ref="sombre")
> fac
[1] jaune paille indigo sombre indigo jaune paille jaune paille
Levels: sombre indigo jaune paille ocre
A > plot(fac)

> fac <- factor(fac, levels = c("ocre", "indigo", "sombre", "jaune paille"))
> fac
[1] jaune paille indigo sombre indigo jaune paille jaune paille
Levels: ocre indigo sombre jaune paille
B > plot(fac)
```

A



B



Facteurs

Nettoyer les niveaux après suppression de certaines valeurs

```
> fac <- fac[1:2]
> fac
[1] jaune paille indigo
Levels: ocre indigo sombre jaune paille
```

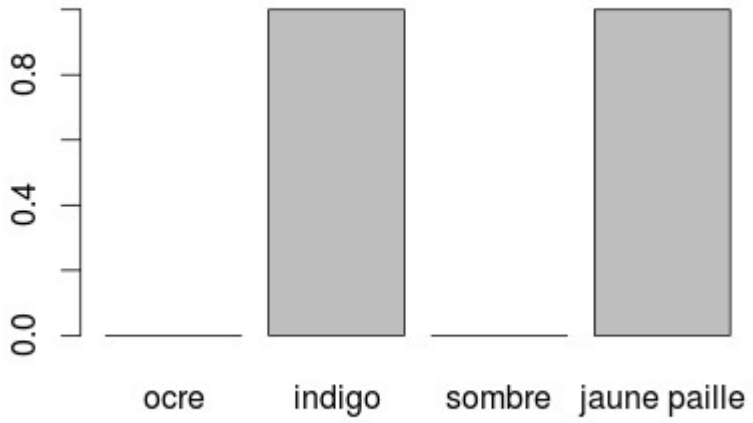
On ne garde que les deux premiers éléments du facteur

Mais tous les niveaux sont toujours présents

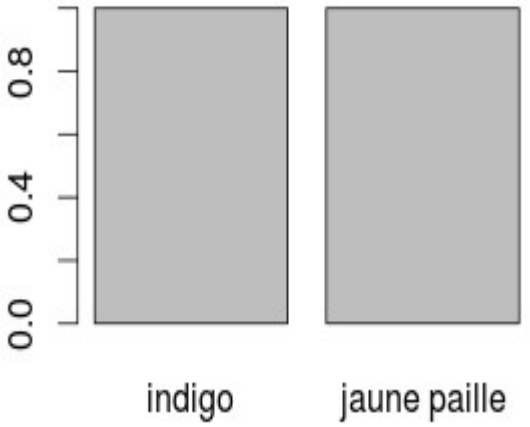
A
```
> plot(fac)
> fac <- factor(fac)
```
« Nettoyage »

B
```
> plot(fac)
```

A



B



Note : on peut aussi utiliser `droplevels()` pour nettoyer tous les facteurs d'un data.frame

Facteurs

Il existe des facteurs « ordonnés » (pour valeurs ordinales)

```
> a <- factor(c("petit", "grand", "grand", "moyen", "petit"),
            levels = c("petit", "moyen", "grand"), ordered = TRUE)
> b <- factor(c( "grand", "moyen", "petit", "petit", "moyen"),
            levels = c("petit", "moyen", "grand"), ordered = TRUE)
> d <- factor(c( "grand", "moyen", "petit", "petit", "moyen"),
            levels = c("petit", "moyen", "grand"), ordered = FALSE)
> a
[1] petit grand grand moyen petit
Levels: petit < moyen < grand
> b
[1] grand moyen petit petit moyen
Levels: petit < moyen < grand
> d
[1] grand moyen petit petit moyen
Levels: petit moyen grand

> a < b
[1]  TRUE FALSE FALSE FALSE  TRUE

> a < d
Erreur dans a < d : comparaison de ces types non implémentée
De plus : Message d'avis :
Méthodes incompatibles ("Ops.ordered", "Ops.factor") pour "<"
```

Paramètre « ordered » = FALSE par défaut

Facteurs

`cut()` : créer des facteurs en "discrétisant" une variable continue

```
> a <- runif(25, 0, 20)
> a
 [1]  8.6203274  1.4543219 16.0480404  6.5055661 15.1457807 11.6854303 14.1767881
 [8]  8.5395153  6.8714540 15.1823997  8.4806042 11.2177451  2.3227155  6.0604360
[15]  9.5760537  6.8966109 12.0142828  1.5216665 19.1198522  0.4441365 16.8342126
[22] 12.6488490  6.2018833 14.8513873 12.7782263

> cut(a, breaks = c(0,5,15,20))

 [1] (5,15]  (0,5]   (15,20] (5,15]  (15,20] (5,15]  (5,15]  (5,15]  (5,15]  (15,20]
[11] (5,15]  (5,15]  (0,5]   (5,15]  (5,15]  (5,15]  (5,15]  (0,5]   (15,20] (0,5]
[21] (15,20] (5,15]  (5,15]  (5,15]  (5,15]
Levels: (0,5] (5,15] (15,20]

> cut(a, breaks = c(0,5,15,20), labels = c("petit", "moyen", "grand"))

 [1] moyen petit grand moyen grand moyen moyen moyen moyen grand moyen moyen petit moyen
[15] moyen moyen moyen petit grand petit grand moyen moyen moyen moyen
Levels: petit moyen grand
```

Dates et temps

Il existe des classes spécifiques pour gérer les dates (Date)
et les données date-temps (POSIXct, POSIXlt)

Pour plus de possibilités voir les packages `chron` et `lubridate`

Les Dates et Date-temps sont de mode numérique :
ils sont stockés comme un nombre de secondes (dates-temps)
ou de jours (date) depuis le 1/1/1970

Objets complexes avec de nombreuses possibilités
--> ne retenez pas les détails
revenez ici en cas de besoin

Date

`as.Date` permet de transformer un "character" en "Date"
Deux notations standards reconnues :

```
> a <- as.Date (c("2012/03/14", "2011/06/30", "2001/12/25"))
> a <- as.Date (c("2012-03-14", "2011-06-30", "2001-12-25"))
> a
[1] "2012-03-14" "2011-06-30" "2001-12-25"
>
```

Si autre notation : utiliser l'argument "format"

```
> b <- as.Date (c("20100314", "20110530", "2001125"),
              format = "%Y%m%d")
> b <- as.Date (c("14 Mars en 2010", "30 Mai en 2011", "25 Avril en 2001"),
              format = "%d %B en %Y")
> b <- as.Date (c("2010, 14 Mar", "2011, 30 Mai", "2001, 25 Avr"),
              format = "%Y, %d %b")
> (b <- as.Date (c("14_03_10", "30_05_11", "25_11_01"),
              format = "%d_%m_%y"))

[1] "2010-03-14" "2011-05-30" "2001-11-25"
```

Voir ?strptime pour les différentes possibilités

Date

Extraction d'une partie de l'info (jours, mois années etc...)

`format()` : transforme un objet Date en character avec la mise en forme voulue

```
> format(b, "%m")
[1] "03" "05" "11"
```

← extraction des mois

```
> format(b, "%Y")
[1] "2010" "2011" "2001"
```

← extraction des années

Certaines fonction permettent d'extraire les jour de la semaine ou le mois tels qu'ils sont reconnus sur votre système

voir ?weekdays pour plus de détails

```
> weekdays(b)
[1] "dimanche" "lundi"    "dimanche"
> months(b)
[1] "mars"     "mai"      "novembre"
```

Date

Opérations sur les dates :

```
> as.numeric(b)
[1] 14682 15124 11651
```

Nbre de jours depuis 1/1/1970

```
> mean(b)
[1] "2007-11-02"
> range(b)
[1] "2001-11-25" "2011-05-30"
```

```
> a-b
Time differences in days
[1] 731  31  30
```

Différence entre dates

```
> b
[1] "2010-03-14" "2011-05-30" "2001-11-25"
> b+4
[1] "2010-03-18" "2011-06-03" "2001-11-29"
```

Ajouter 4 jours

NB : pour ajouter/enlever des mois ou années, utiliser la classe POSIXlt

Séquences

```
# Voir ? seq.Date,?cut.Date
> seq(as.Date("2012-04-24"), as.Date("2012-05-12"), by = "3 days")
[1] "2012-04-24" "2012-04-27" "2012-04-30" "2012-05-03" "2012-05-06"
[6] "2012-05-09" "2012-05-12"
> seq(as.Date("2012-04-24"), as.Date("2012-09-12"), by = "3 weeks")
[1] "2012-04-24" "2012-05-15" "2012-06-05" "2012-06-26" "2012-07-17"
[6] "2012-08-07" "2012-08-28"
```

Date - temps

Classes POSIXct et POSIXlt

POSIXct : vecteur numérique = nbre de secondes depuis 1/1/1970
POSIXlt : liste contenant les secondes, minutes, heures etc...

En pratique :
les 2 classes se comportent presque toujours de la même manière
ea : POSIXlt se comporte en fait pratiquement
toujours comme un vecteur

--> On doit rarement s'en tracasser

Date - temps

Le plus souvent : on crée un vecteur Date-temps
depuis un vecteur "character" avec la fonction `strptime()`

`as.POSIXlt()` et `as.POSIXct()`
permettent de passer d'une classe à l'autre en cas de besoin
mais aussi de convertir un vecteur "numeric" ou "character"
--> on les utilise rarement

`strftime()` permet de passer d'un format POSIXt à un vecteur "character" en changeant le formatage à volonté

Voir aussi `ISOdate()` qui permet de créer des POSIXlt
sur base d'une liste "année", "mois", "jour", etc

+ packages chron & lubridate

Date - temps

`strptime()` : character --> POSIXlt
`strftime()` : POSIXt --> character
en spécifiant le format

```
> a <- strptime(c("23/6/2012, 20h06", "2/3/1999, 7h29"),
        format = "%d/%m/%Y, %Hh%M")
> a
[1] "2012-06-23 20:06:00" "1999-03-02 07:29:00"

> class(a)
[1] "POSIXlt" "POSIXt"

> as.numeric(a)
[1] 1340474760  920356140
```

voir ?strptime pour les différentes options de formatage

On peut reformater complètement la date-heure avec `strftime()` :

```
> strftime(a, format = "%A le %d %B %Y, à %H heures %M minutes")
[1] "samedi le 23 juin 2012, à 20 heures 06 minutes"
[2] "mardi le 02 mars 1999, à 07 heures 29 minutes"
```

Date - temps

Extraction d'une partie de l'info : mois, heures, minutes, etc...

```
> a
[1] "2012-06-23 20:06:00" "1999-03-02 07:29:00"
```

Approche 1 : avec `strftime`

```
> as.numeric(strftime(a, format = "%d")) # extraction du jour du mois
[1] 23  2
> as.numeric(strftime(a, format = "%H")) # extraction des heures
[1] 20  7
```

Approche 2 : uniquement avec objets POSIXlt

```
> attributes(a)
$names
[1] "sec"   "min"   "hour"  "mday"  "mon"   "year"  "wday"  "yday"  "isdst"

> a$hour
[1] 20  7
> a$mday
[1] 23  2
> a$wday
[1] 6 2
> a$yday
[1] 174  60"
```

Date - temps

Extraction d'une partie de l'info : mois, heures, minutes, etc...

Attention aux mauvaises surprises :

les mois sont numérotés de 0 à 11
les jours de la semaine de 0 à 6 (0 = Dimanche)
l'année depuis 1900

```
> b <- strptime(c("6/1/2013 12:33:01", "5/2/2013 12:33:01"),
        format = "%d/%m/%Y %H:%M:%S")
> b$year
[1] 113 113
> b$mon
[1] 0 1
> b$wday
[1] 0 2
> weekdays(b)
[1] "dimanche" "mardi"
```

Date - temps

Opérations sur les dates-temps

```
> a
[1] "2012-06-23 20:06:00" "1999-03-02 07:29:00"

> mean(a)
[1] "2005-10-27 14:17:30 CEST"

> range(a)
[1] "1999-03-02 07:29:00 CET"  "2012-06-23 20:06:00 CEST"

> seq(strptime("2013-03-06 10:23:00", "%Y-%m-%d %H:%M:%S"),
      strptime("2013-03-07 10:23:00", "%Y-%m-%d %H:%M:%S"), by = "4 hours")

[1] "2013-03-06 10:23:00 CET" "2013-03-06 14:23:00 CET"
[3] "2013-03-06 18:23:00 CET" "2013-03-06 22:23:00 CET"
[5] "2013-03-07 02:23:00 CET" "2013-03-07 06:23:00 CET"
[7] "2013-03-07 10:23:00 CET"
```

Date - temps

Ajouter/enlever des jours, heures, etc

Approche 1 ajouter la quantité voulue en secondes :

```
> a
[1] "2012-06-23 20:06:00" "1999-03-02 07:29:00"
```

Ajouter une heure

```
> a + 60*60
[1] "2012-06-23 21:06:00 CEST" "1999-03-02 08:29:00 CET"
```

Enlever un jour

```
> a - 24*60*60
[1] "2012-06-22 20:06:00 CEST" "1999-03-01 07:29:00 CET"
```

Approche 2 : uniquement pour objets POSIXlt

```
> a
[1] "2012-06-23 20:06:00" "1999-03-02 07:29:00"
> a$hour <- a$hour + 1
> a
[1] "2012-06-23 21:06:00" "1999-03-02 08:29:00"
```

Ajouter une heure

```
> a
[1] "2012-06-23 21:06:00" "1999-03-02 08:29:00"
> a$mon <- a$mon - 1
> a
[1] "2012-05-23 21:06:00" "1999-02-02 08:29:00"
```

Enlever un mois

Date - temps

Différence entre 2 date-heures --> objet de classe difftime

```
> a <- strptime(c("23/6/2012, 20h06", "2/3/1999, 7h29"),
        format = "%d/%m/%Y, %Hh%M")
> b <- strptime(c("24/6/2012, 06h06", "1/3/1999, 23h42"),
        format = "%d/%m/%Y, %Hh%M")

> deltat <- a-b
> deltat
Time differences in hours
[1] -10.000000   7.783333
> class(deltat)
[1] "difftime"
```

On peut voir et changer les unités avec la fonction `units()`

```
> units(deltat)
[1] "hours"
> units(deltat) <- "mins"
> deltat
Time differences in mins
[1] -600  467
> as.numeric(deltat)
[1] -600  467
> as.numeric(deltat, units = "secs")
[1] -36000  28020
```

Date - temps

Gestion des fuseaux horaires
Important à prendre en compte si on doit manipuler des temps
dans différents fuseaux
ou en temps universel (GMT ou UTC) et en temps local ("CET" chez nous)

```
> date1 <- "2012-07-01 04:05:52" # CET
> date2 <- "2012-07-01 03:04:52" # UTC = GMT
>
> a <- strptime(date1, format = "%Y-%m-%d %H:%M:%S")
> b <- strptime(date2, format = "%Y-%m-%d %H:%M:%S")
> b-a
Time difference of -1.016667 hours
> b<a
[1] TRUE
>
> a <- strptime(date1, format = "%Y-%m-%d %H:%M:%S", tz="CET")
> b <- strptime(date2, format = "%Y-%m-%d %H:%M:%S", tz = "UTC")
> b-a
Time difference of 59 mins
> b<a
[1] FALSE
Message d'avis :
In check_tzones(e1, e2) : les attributs 'tzone' ne correspondent pas
```

Sans préciser le fuseau
Par défaut = fuseau local
--> résultat incorrect

En précisant le fuseau
--> message d'avis mais réponse correcte

Date - temps

Gestion des fuseaux horaires

Changer de fuseau --> il faut passer par le format POSIXct

```
> a
[1] "2012-07-01 04:05:52 CEST"

> format(as.POSIXct(a), tz="UTC",usetz = TRUE)
[1] "2012-07-01 02:05:52 UTC"

> format(as.POSIXct(a), tz="GMT",usetz = TRUE)
[1] "2012-07-01 02:05:52 GMT"

> format(as.POSIXct(a), tz="WET",usetz = TRUE)
[1] "2012-07-01 03:05:52 WEST"

> format(as.POSIXct(a), tz="nimportequoi",usetz = TRUE)
[1] "2012-07-01 02:05:52 nimportequoi"
>
```

pas correct

!!!!
Si le fuseau donné est inconnu par R
Il utilise l'UTC par défaut
sans message d'avertissement
!!!!

Date - temps

Gestion des fuseaux horaires
Changer de fuseau

Quelques pièges !
Exemple : on veut paser le temps a vers le fuseau WET (West European Time = UK, Portugal etc...))

```
> as.POSIXct(a, tz="WET")
[1] "2012-07-01 04:05:52 WEST"
```

Ne change pas le fuseau mais indique que les heures sont dans le fuseau WEST

Pas correct car tz par défaut pour POSIXlt = UTC

```
> as.POSIXlt(format(as.POSIXct(a), tz="WET",usetz = TRUE))
[1] "2012-07-01 03:05:52"

> as.POSIXlt(format(as.POSIXct(a), tz="WET",usetz = TRUE), tz="WET")
[1] "2012-07-01 03:05:52 WET"
```

correct

Résumé : types d'objets

Types de données :
numeric - character - logical

Factor : Pour stoker des variables qualitatives
Ne peut contenir que les valeurs prédéfinies par l'attribut "levels"
Vecteur numeric avec attribut "levels" donnant la correspondance entre les nombres et le texte

Types d'objets :

	Type de données homogène	**Type de données hétérogène**
1 dimension	vector →	list
2 dimensions	matrix	data.frame
n dimensions	array	

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de dates et caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

Importer un fichier texte dans un data.frame

C'est la manière de travailler la plus fréquente.
On utilise `read.table()` qui dispose de nombreuses options :

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

NB1 : Quelque soit la méthode, les données sont toujours chargées dans la RAM et le fichier d'origine n'est jamais modifié

NB2 : Par défaut `read.table()` importe
les colonnes de nombres comme des vecteurs numériques
et les colonnes de texte comme des facteurs

Importer un fichier texte dans un data.frame

En général on définit d'abord un répertoire de travail avec `setwd()`

Le répertoire de travail est l'endroit sur le disque où R va chercher ou écrire des fichiers

Attention le caractère \ est réservé, on écrira donc :

```
setwd("C:/Documents and Settings/User1/My Documents")
```

Ou

```
setwd("C:\\Documents and Settings\\User1\\My Documents")
```

Mais pas :

```
setwd("C:\Documents and Settings\User1\My Documents")
```

Importer un fichier texte dans un data.frame

Options les plus fréquemment utilisées de `read.table()`

```
read.table(
    file,
    header = FALSE,
    sep = " ",
    dec = ".",
    na.strings = "NA",
    encoding = "unknown")
```

- `file` : Nom du chemin vers le fichier. On peut définir des sous-dossiers dans le répertoire de travail. Pex : `"data/mydataset.csv"`
- `header = FALSE` : Est-ce que la première ligne contient les noms de colonne ?
- `sep = " "` : Le caractère séparant les colonnes souvent : `";"` ou `"\t"` (tabulation)
- `dec = "."` : Le caractère séparant les décimales. En Europe : souvent `","`
- `na.strings = "NA"` : Le caractère indiquant les valeurs Manquantes, parfois : `""` (case vide)
- `encoding = "unknown"` : L'encodage des caractères. A spécifier si les caractères spéciaux (é, è, à,...) sont mal lus. Deux valeurs possibles : `"Latin-1"` ou `"UTF-8"`

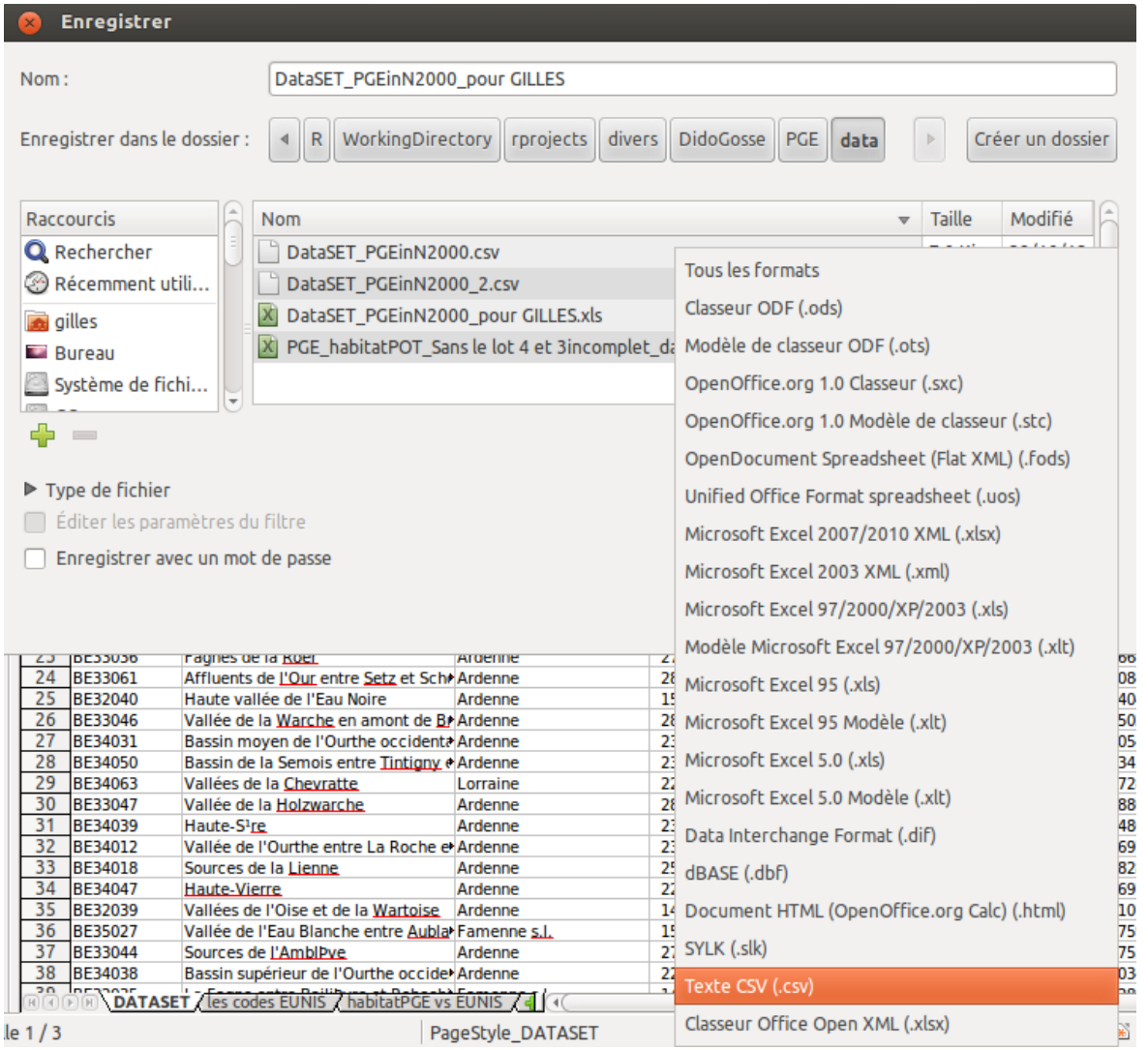
Si les problèmes persistent --> modifier l'encodage du fichier d'origine dans un autre programme

Du tableur vers le fichier texte

Les données sont souvent encodées dans des tableurs (Calc, Excel,...)
L'idéal est des les exporter en format texte délimité(txt (sep = tab), csv,..)

Dans Libre Office Calc :
Fichier/Enregistrer sous

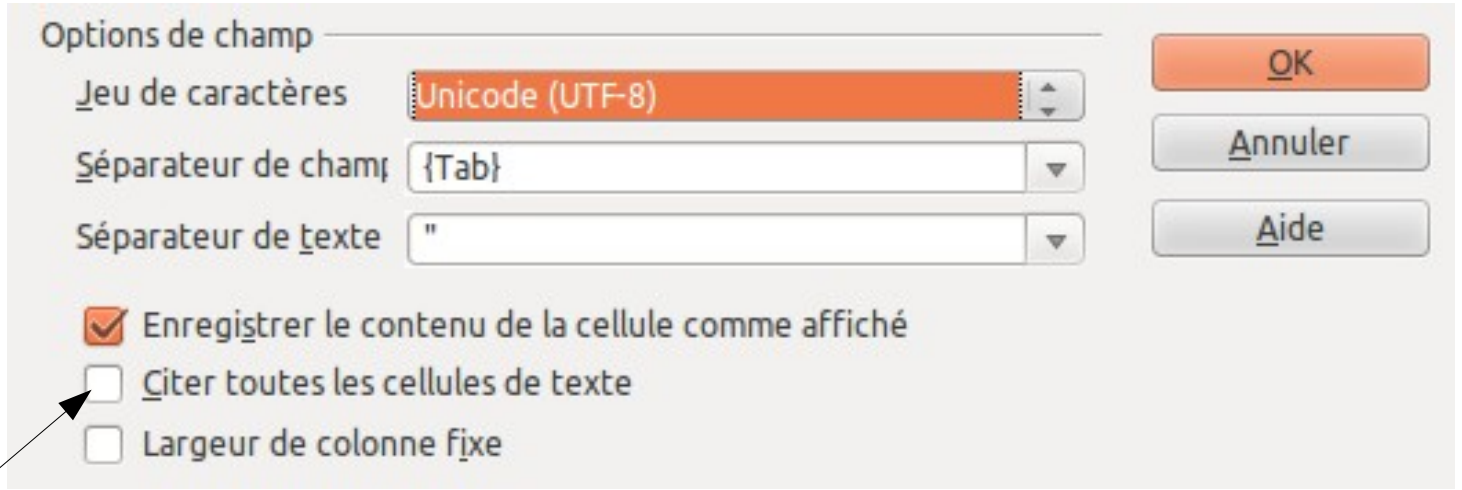
Choisir « Texte CSV »
dans type de fichiers



Du tableur vers le fichier texte

Les données sont souvent encodées dans des tableurs (Calc, Excel,...)
L'idéal est des les exporter en format texte délimité(txt (sep = tab), csv,..)

Options d'enregistrement :



Cochez cette case
pour éviter les problèmes !

Si vous utilisez Excel 2007 et que vous ne vous y retrouvez pas: utilisez Libre Office !!
Version portable (sans installation) ici : http://framakey.org/Portables/LibreOfficePortable

Du tableur vers le fichier texte

Voici quelques conseils pour une exportation vers R
(en passant par un fichier texte) sans (trop de) douleur...

Chaque colonne doit avoir un titre
éviter les caractères spéciaux (y compris espaces) dans les noms de colonne

Pas de colonne vide

Attention aux cases « fantômes » qui n'affichent rien mais contiennent des données
Le plus « safe » : sélectionner la zone de données et la coller dans une feuille vierge

Méfiez-vous du format "date" du tableur
Encodez de préférence les dates en format texte

Choisir de préférence les caractères séparateurs de champ
; ou tabulation

Bien noter le caractère de la décimale (. ou ,)
et la manière dont les valeurs manquantes sont notées

Exporter de préférence en UTF-8 (ou latin-1)
Entourer les champs texte de guillemets

Vérifier les données

Une fois les données importées dans R avec `read.table()` :

!!! TOUJOURS !!! :

1) faire un `summary()` des données

2) vérifier que les données ont été correctement importées

3) vérifier que les données sont dans le bon format

Typiquement certaines valeurs importées comme des nombres doivent être transformées en facteurs, pex site : 1, 2, 3..

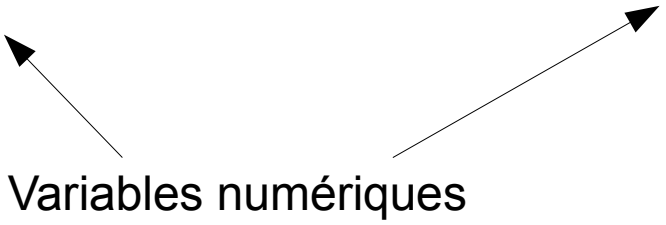
```
mydata$site <- as.factor(mydata$site)
```

4) Souvent utile d'avoir une représentation graphique des données pour repérer les problèmes.
Utiliser par exemple `pairs()`

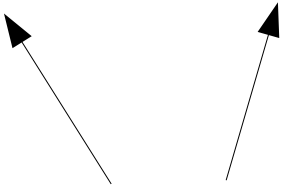
Vérifier les données

Exemple : mesure de taille, poids et détermination du sexe d'oisillons au nid dans plusieurs sites (et commentaires dans le champs memo)

```
> d <- read.table("data/ois.txt", sep= "\t", dec=",", header=TRUE, na.strings=".")

> head(d)
  taille poids site nid   sexe            memo
1    9.4   5.3    3   2   male
2    9.8   5.3    1   1 female
3   11.6   6.3    2   2   <NA>       Mal formé
4   10.1   5.7    4   2 female
5   12.0   3.7    4   1   male peson déréglé ?
6   11.7   6.7    1   1 female

> summary(d)
     taille          poids           site            nid          sexe                    memo
 Min.   : 7.70   Min.   :3.700   Min.   :1.000   Min.   :1.000   female:36                    :72
 1st Qu.: 9.40   1st Qu.:4.900   1st Qu.:2.250   1st Qu.:2.000   femle : 1   Mal formé       : 3
 Median :10.05   Median :5.500   Median :4.000   Median :3.000   male  :38   peson cassé     : 2
 Mean   :10.29   Mean   :5.471   Mean   :3.987   Mean   :2.821   NA's  : 3   peson déréglé ?: 1
 3rd Qu.:10.78   3rd Qu.:6.000   3rd Qu.:6.000   3rd Qu.:4.000
 Max.   :29.00   Max.   :6.900   Max.   :7.000   Max.   :5.000
                 NA's   :2
```



Variables numériques



Variables textes importées comme facteur

Vérifier les données

Les variables site et nid ne sont en fait pas des variables numériques
--> on les transforme en facteur

Le nid 2 du site 1 n'a rien à voir avec le nid 2 du site 3
--> on crée une variable identifiant de manière unique les nids avec `paste()`

```
> d$site <- as.factor(d$site)
> d$nid <- as.factor(d$nid)
> d$IDnid <- paste(d$site, d$nid, sep="_")

> head(d)
  taille poids site nid   sexe             memo IDnid
1    9.4   5.3    3   2   male                    3_2
2    9.8   5.3    1   1 female                    1_1
3   11.6   6.3    2   2   <NA>       Mal formé    2_2
4   10.1   5.7    4   2 female                    4_2
5   12.0   3.7    4   1   male peson déréglé ?    4_1
6   11.7   6.7    1   1 female                    1_1
```

La variable créée « IDnid »
Est considérée comme du texte

```
> summary(d)
     taille          poids       site   nid       sexe                  memo      IDnid
 Min.   : 7.70   Min.   :3.700  1: 8   1:17  female:36                  :72  Length:78
 1st Qu.: 9.40   1st Qu.:4.900  2:12   2:21  femle : 1  Mal formé      : 3  Class :character
 Median :10.05   Median :5.500  3:18   3:13  male  :38  peson cassé    : 2  Mode  :character
 Mean   :10.29   Mean   :5.471  4: 8   4:13  NA's  : 3  peson déréglé ?: 1
 3rd Qu.:10.78   3rd Qu.:6.000  5:10   5:14
 Max.   :29.00   Max.   :6.900  6:11
                 NA's   :2      7:11
```

Vérifier les données

On transforme la variable « IDnid » en facteur

C'est bien un facteur

```
> d$IDnid <- as.factor(d$IDnid)
> summary(d)
     taille          poids          site   nid        sexe                    memo         IDnid
 Min.   : 7.70   Min.   :3.700   1: 8   1:17   female:36                    :72   3_4    : 8
 1st Qu.: 9.40   1st Qu.:4.900   2:12   2:21   femle : 1   Mal formé       : 3   5_2    : 5
 Median :10.05   Median :5.500   3:18   3:13   male  :38   peson cassé     : 2   6_1    : 5
 Mean   :10.29   Mean   :5.471   4: 8   4:13   NA's  : 3   peson déréglé ?: 1   3_2    : 4
 3rd Qu.:10.78   3rd Qu.:6.000   5:10   5:14                                    4_3    : 4
 Max.   :29.00   Max.   :6.900   6:11                                           2_2    : 3
                 NA's   :2       7:11                                           (Other):49
```

Valeur impossible

Faute de frappe

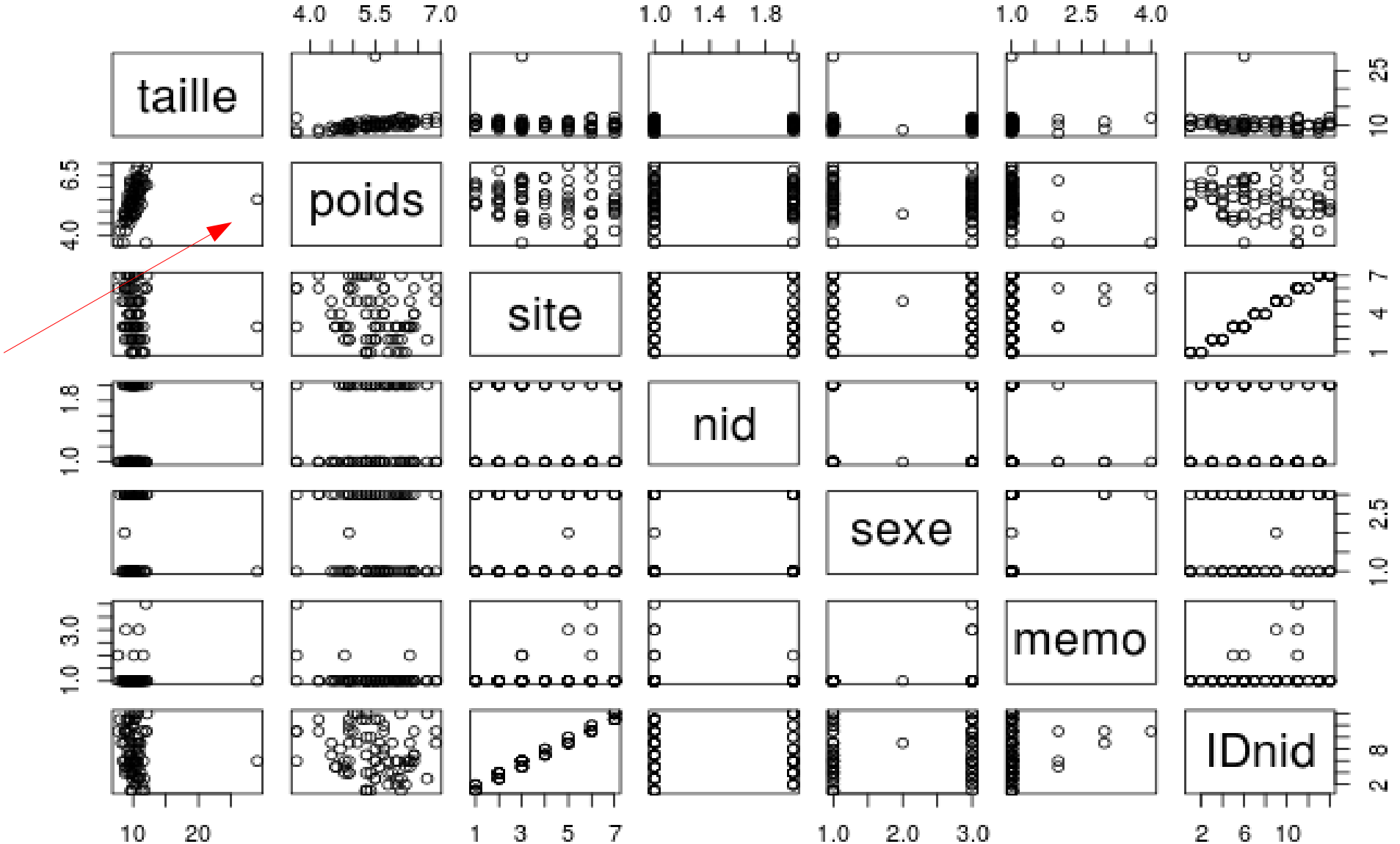
Les valeurs manquantes
Ont été correctement importées

Vérifier les données

On peut aussi repérer les valeurs aberrantes avec des graphiques

`pairs(d)` fait un scatterplot matrix :

une matrice de graphiques de toutes les paires de variables

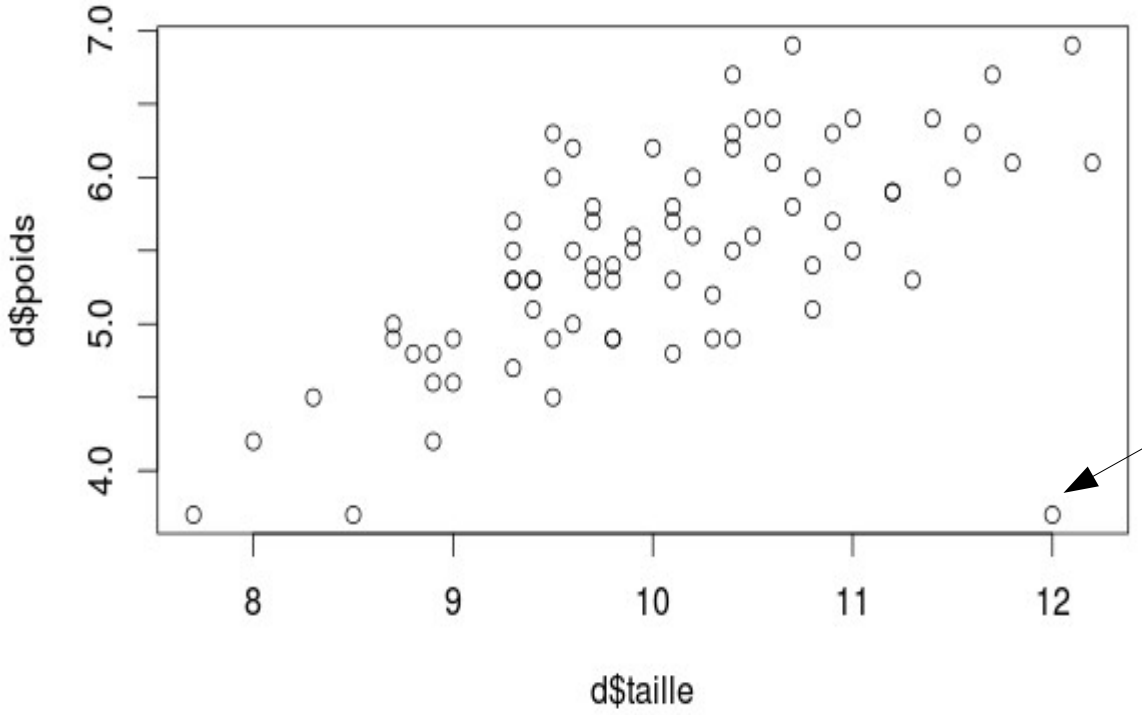


Vérifier les données

On peut corriger les erreurs d'encodage directement dans le fichier d'origine.
On peut aussi les corriger dans R :

```
d[d$taille > 15, "taille"] <- NA
d[d$sexe == "femle" & !is.na(d$sexe), "sexe"] <- "female"
plot(d$poids ~ d$taille)
```

d[*blabla* , *blabla*]
= « subscripting »
--> voir chapitre suivant !!



Il reste une valeur problématique
Le poids et la taille sont plausible.
Mais leur combinaison semble improbable

Vérifier les données

Si on retourne dans les données on voit dans le « memo » qu'il y avait eu pour cette valeur un problème avec l'appareil de mesure de poids
--> on choisi (en son âme et conscience!!)
de remplacer ce poids par une valeur manquante

```
> d[d$poids < 5 & d$taille>11,]

  taille poids site nid sexe             memo IDnid
5     12   3.7    6   1 male peson déréglé ?   6_1

> d[5, "poids"] <- NA

> summary(d)
     taille          poids       site   nid        sexe                    memo         IDnid
 Min.   : 7.70   Min.   :3.700   1: 8   1:17   female:37                    :72   3_4    : 8
 1st Qu.: 9.40   1st Qu.:4.950   2:12   2:21   femle : 0   Mal formé       : 3   5_2    : 5
 Median :10.00   Median :5.500   3:18   3:13   male  :38   peson cassé     : 2   6_1    : 5
 Mean   :10.05   Mean   :5.495   4: 8   4:13   NA's  : 3   peson déréglé ?: 1   3_2    : 4
 3rd Qu.:10.70   3rd Qu.:6.000   5:10   5:14                                     4_3    : 4
 Max.   :12.20   Max.   :6.900   6:11                                            2_2    : 3
 NA's   :1       NA's   :3       7:11                                            (Other):49
```

On peut encore nettoyer
Les niveaux vides de la variable « sexe »

Vérifier les données

On nettoie les niveaux vides du facteur « sexe »

```
> d$sexe <- factor(d$sexe)

> # Alternative :
> d <- droplevels(d)

> summary(d)
     taille          poids          site    nid        sexe                    memo          IDnid
 Min.   : 7.70   Min.   :3.700   1: 8   1:17   female:37                    :72   3_4     : 8
 1st Qu.: 9.40   1st Qu.:4.950   2:12   2:21   male  :38   Mal formé      : 3   5_2     : 5
 Median :10.00   Median :5.500   3:18   3:13   NA's  : 3   peson cassé    : 2   6_1     : 5
 Mean   :10.05   Mean   :5.495   4: 8   4:13               peson déréglé ?: 1   3_2     : 4
 3rd Qu.:10.70   3rd Qu.:6.000   5:10   5:14                                    4_3     : 4
 Max.   :12.20   Max.   :6.900   6:11                                           2_2     : 3
 NA's   :1       NA's   :3       7:11                                           (Other):49
```

OUF ! Le jeu de données semble OK

Autres méthodes d'importation

`read.table()` est de loin la méthode la plus utilisée
mais il existe d'autres possibilités, notamment :

`readLines()`
permet de lire des fichiers textes (pas des tableaux)

`scan()`
plus rapide que read.table (qui est basé sur scan) mais moins facile

`load()`
permet de charger des fichiers binaires .Rdata
sauvés avec la fonction save -> très rapide !

Voir le manuel « R Data Import/Export » pour de nombreux détails
http://cran.r-project.org/manuals.html

Autres types de fichiers

R peut se connecter à de nombreux **Systèmes de Base de Données** (principalement : PostgreSQL, MySQL, SQLite)

Fichiers .xls et .xlsx (MS Excel)

Très déconseillé de stocker et d'accéder à ses données dans ces formats !

On peut néanmoins les lire avec `read.xls()` du package `gdata`

Nécessite Perl ! Et parfois quelques chipotages ! Ne fonctionne pas toujours !

Voir aussi packages `XLConnect`, `xlsx`, `readxl`, `openxlsx`, ...

A priori `readxl` est le plus rapide et sans dépendances.

Mais il ne transforme pas les colonnes de texte en facteur

```
b <- read_excel("myfile.xls", sheet=1, na = "")
b[,sapply(b,is.character)] <-
          lapply(b[,sapply(b,is.character)], as.factor)
```

Fichiers .mdb (MS Access)

Vraiment très très déconseillé de stocker ses données dans ce format !!!

Sous windows uniquement on peut essayer de s'y connecter avec ODBC ou JDBC

On peut essayer `mdb.get()` du package `Hmisc` pour importer les tables

Ça fonctionne parfois !

Exporter des données

`write.table()`

Pour sauver des tableaux dans un fichier

`writeLines()`

Pour sauver du texte dans un fichier

`sink()`

Dérive les sorties de la console vers un fichier texte

Ne pas oublier de terminer par `sink()` sans options pour arrêter la redirection

`cat()`

A un argument « file » qui permet de rediriger le résultat vers un fichier

--> on peut construire entièrement un fichier HTML ou Latex

Mais il existe des méthodes plus efficaces (R2HTML, knitr, Sweave,...)

`save()` et `save.image()`

Sauver un objet ou l'ensemble de la session dans un fichier .Rdata

Particulièrement utile quand on fait des calculs ou des simulations longues

Pour garder une copie des résultats réutilisable dans R

etc ....

Exercices :
Manipulation des types d'objets et importer des données

Le langage R

Contenu

Les objets
Importer des données

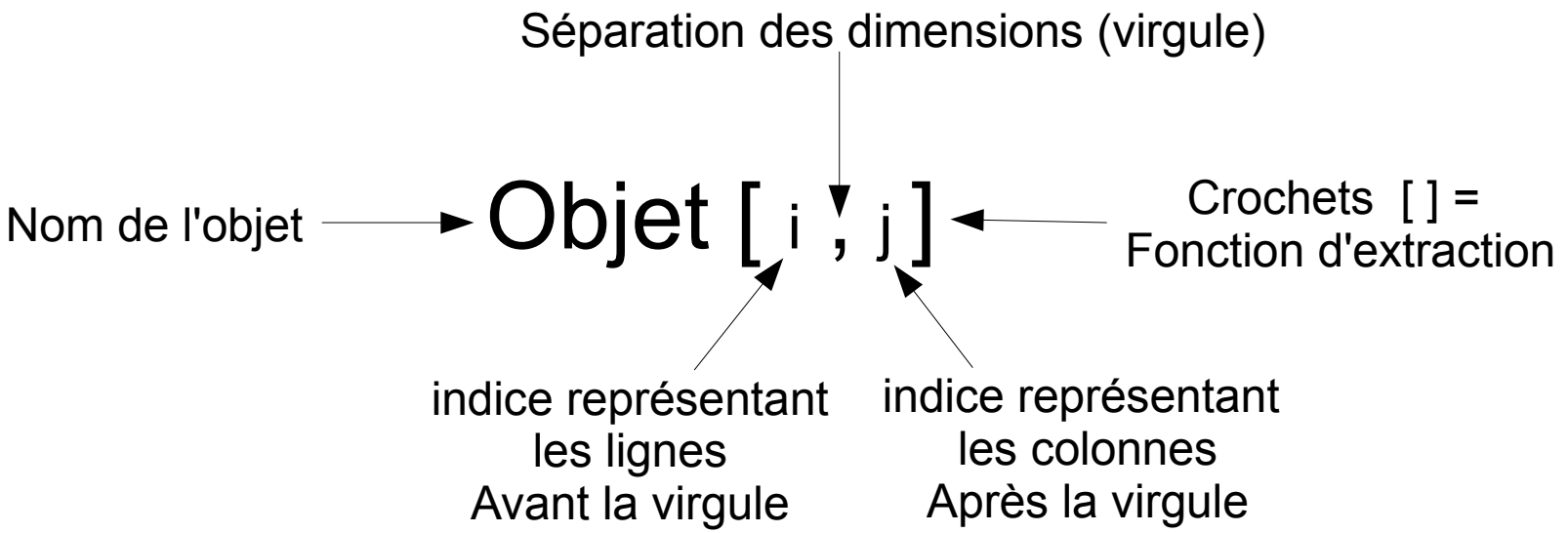
Extraire des données (subscripting)
Manipulation de dates et caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

Principe général

Subscripting = méthode pour extraire des éléments
d'un objet sur base de leur(s) indice(s)
(éléments, lignes, colonnes,...)



Principe général

Quelques exemples :

vecteur (1 dimension)

matrice (2 dimensions)

```
> (vec <- c(23, 76, 9, 2, 6, 267, 8, 8, 52))
[1]  23  76   9   2   6 267   8   8  52
> (mat <- matrix(a, nrow = 3, ncol = 3))
     [,1] [,2] [,3]
[1,]   23    2    8
[2,]   76    6    8
[3,]    9  267   52
```

```
> vec[4]
[1] 2
```

4ème élément du vecteur

```
> vec[c(1, 3, 9)]
[1] 23  9 52
```

éléments 1, 3 et 9 du vecteur

```
> mat[2,1]
[1] 76
```

élément situé sur la deuxième ligne et la première colonne

```
> mat[2,]
[1] 76  6  8
```

éléments de la 2ème ligne (pas d'indication pour la colonne --> toutes les colonnes sont prises)

```
> mat[,c(2,3)]
     [,1] [,2]
[1,]    2    8
[2,]    6    8
[3,]  267   52
```

éléments des colonnes 2 et 3

Principe général

Les « subscripts » peuvent être de 4 types :

1) un vecteur d'entiers positifs
2) un vecteur d'entiers négatifs
3) un vecteur de caractères (texte)
4) un vecteur logique

Dans les 3 premiers cas le vecteur a une longueur quelconque

Dans le dernier cas, il a une dimension égale à la dimension visée (nombre de lignes ou de colonnes ou d'éléments)
Dans le cas contraire les premiers éléments sont recyclés

Subscripts numériques positifs

5 premières lignes du jeu de données

```
> d[1:5,]
  taille poids site nid   sexe   region
1    9.4   5.3    5   5   male Limbourg
2    9.8   5.3    1   2 female Lorraine
3   11.6   6.3    3   4   male Limbourg
4   10.1   5.7    6   4 female  Ardenne
5   10.1   5.4    6   1   male  Ardenne
```

1 ligne sur 3 des 2 premières colonnes

```
> (meslignes <- seq(from = 1, to = nrow(d), by=3 ))
 [1]  1  4  7 10 13 16 19 22 25 28 31 34 37 40 43 46 49 52 55 58 61 64 67 70 73 76

> d[meslignes, 1:2]
   taille poids
1     9.4   5.3
4    10.1   5.7
7    10.5   5.6
10    9.6   5.5
13   10.4   6.2
16   11.8   6.1
19   10.7   6.9
22    9.8   4.9
(...)
```

Subscripts numériques positifs

Sélection aléatoire de 5 lignes pour les 3 dernières colonnes

```
> (meslignes <- sample(1:nrow(d), size = 5))
[1] 71 46 33 72 53

> (mescolonnes <- (ncol(d)-2) : ncol(d))
[1] 4 5 6

> d[meslignes, mescolonnes]

   nid   sexe   region
71   5 female Limbourg
46   4   male Limbourg
33   3   male  Famenne
72   3   male  Ardenne
53   4   male Limbourg
```

Sélection de la troisième colonne

```
> d[,3]
 [1] 5 1 3 6 6 2 3 6 6 3 2 5 1 1 7 1 3 7 5 3 6 6 3 4 5 5 5 7 3 1 4
       2 4 3 7 3 2 5 1 7 4 2 5 7 5 3 3 6 2 2
[51] 7 3 3 6 2 1 3 4 3 3 7 4 4 7 6 2 3 7 5 6 3 6 4 7 2 2 1 2
Levels: 1 2 3 4 5 6 7
```

Subscripts numériques positifs

Les subscripts numériques permettent aussi de trier un tableau grâce à la fonction `order()`

```
> order(d$site)
 [1]  2 13 14 16 30 39 56 77  6 11 32 37 42 49 50 55 66 75 76 78  3
       7 10 17 20 23 29 34 36 46 47 52 53 57 59 60 67 71 24 31 33
       41 58 62 63 73  1 12 19 25 26 27 38 43 45 69  4  5  8  9 21
       22 48 54 65 70 72 15 18 28 35 40 44 51 61 64 68 74
> order(d$site, d$nid, d$sexe)
 [1] 39 30  2 77 14 16 56 13 50 49  6 37 76 75 42 32 78 66 11 55 47
       7 23 59 34 52 29 17 20 57  3 36 46 53 60 71 10 67 58 63 62 31
       73 24 33 41 12 26 69 45 19 25 38 43  1 27  9 70  5  8 54 21 65
       48 72  4 22 64 18 74 28 15 51 61 68 40 35 44
```

```
> d[order(d$site, d$nid, d$sexe),]
   taille poids site nid   sexe        region
39    9.7   5.4    1   1 female      Lorraine
30   11.3   5.3    1   1   male      Lorraine
2     9.8   5.3    1   2 female      Lorraine
77    9.7   5.3    1   2 female      Lorraine
14   10.1   5.8    1   3 female      Lorraine
16   11.8   6.1    1   3 female      Lorraine
56   11.5   6.0    1   4   male      Lorraine
13   10.4   6.2    1   5   male      Lorraine
50    9.9   5.5    2   1 female       Ardenne
49   10.8   6.0    2   1   male       Ardenne
```

Subscripts numériques négatif

On ne prend pas les éléments indiqués par les nombres négatifs
NB : on ne peut pas mélanger nombres positifs et négatifs dans un même vecteur de subscripts

5 premières lignes et toutes les colonnes sauf la 2, 3 et 6

```
> d[1:5, -c(2, 3, 6)]

  taille nid   sexe
1    9.4   5   male
2    9.8   2 female
3   11.6   4   male
4   10.1   4 female
5   10.1   1   male
```

Subscripts de type “character”

Permet d'extraire des éléments, lignes, colonnes en fonction de leur nom

```
> vec <- c(23, 76, 9, 2, 6, 267, 8, 8, 52)
> names(vec) <- letters[1:9]
> vec
  a   b   c   d   e   f   g   h   i
 23  76   9   2   6 267   8   8  52
> vec[c("b", "d", "h")]
 b  d  h
76  2  8

> d[1:5, c("poids", "sexe", "region")]
  poids   sexe   region
1   5.3   male Limbourg
2   5.3 female Lorraine
3   6.3   male Limbourg
4   5.7 female  Ardenne
5   5.4   male  Ardenne

> row.names(d) <- paste("obs", 1:78, sep="_")
> d[ c("obs_3", "obs_26", "obs_32"),]
       taille poids site nid   sexe   region
obs_3    11.6   6.3    3   4   male Limbourg
obs_26    8.3   4.5    5   1 female Limbourg
obs_32    9.7   5.8    2   4 female  Ardenne
```

Subscripts logiques

On extrait les éléments correspondant aux valeurs « TRUE » et pas les « FALSE »

```
> (vec <- c(23, 76, 9, 2, 6, 267, 8, 8, 52))
[1]  23  76   9   2   6 267   8   8  52
> (mat <- matrix(vec, nrow = 3, ncol = 3))
     [,1] [,2] [,3]
[1,]   23    2    8
[2,]   76    6    8
[3,]    9  267   52

> vec > 10
[1]  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
> vec[vec > 10]
[1]  23  76 267  52

> vec == 8
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
> vec[vec == 8]
[1] 8 8

> mat[,1]>10
[1]  TRUE  TRUE FALSE
> mat[mat[,1]>10,]
     [,1] [,2] [,3]
[1,]   23    2    8
[2,]   76    6    8
```

Subscripts logiques

Opérateurs logiques :

<	plus petit
>	plus grand
<=	plus petit ou égal
>=	plus grand ou égal
==	identique
!=	différent
!	négation
%in %	correspondance
&	ET logique
|	OU logique
any()	Au moins un TRUE ?
all()	Uniquement des TRUE ?

Erreur classique :
Confusion entre
== et =

Il existe aussi ||, && et match() qui sont peu utiles en pratique

Opérateurs logiques : Exemples

Identique, différent

```
> (vec <- c(23, 76, 9, 2, 6, 267, 8, 8, 52))
[1]  23  76   9   2   6 267   8   8  52
> vec == 8
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
> vec != 8
[1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE  TRUE
```

Négation : inverse un vecteur logique

```
> a <- c(T, F, T, T, T, F)
> a
[1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE
> !a
[1] FALSE  TRUE FALSE FALSE FALSE  TRUE
```

Opérateurs logiques : Exemples

ET logique

```
> vec
[1]  23  76   9   2   6 267   8   8   52
> vec > 10
[1]  TRUE  TRUE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE
> vec <= 52
[1]  TRUE FALSE  TRUE  TRUE  TRUE FALSE  TRUE  TRUE  TRUE
> vec > 10 & vec <= 52
[1]  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
```

OU logique et %in%

```
> vec == 8
[1] FALSE FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE FALSE
> vec == 76
[1] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> vec == 52
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
> vec == 8 | vec == 76 | vec == 52
[1] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
> vec %in% c(8, 76, 52)
[1] FALSE  TRUE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

Résultat identique

Est ce que les éléments de vec correspondent à 8, 76 ou 52 ?

Opérateurs logiques : Exemples

`any()` et `all()`

```
> a <- c(T, F, F, F, F, F)
> a
[1]  TRUE FALSE FALSE FALSE FALSE FALSE
> any(a)
[1] TRUE
> all(a)
[1] FALSE


> a <- c(T, T, T, T)
> a
[1] TRUE TRUE TRUE TRUE
> any(a)
[1] TRUE
> all(a)
[1] TRUE
```

Subscripts logiques

Les premières lignes du jeu de données pour s'en rappeler la structure :

```
> d[1:5,]
  taille poids site nid   sexe   region
1    9.4   5.3    5   5   male Limbourg
2    9.8   5.3    1   2 female Lorraine
3   11.6   6.3    3   4   male Limbourg
4   10.1   5.7    6   4 female  Ardenne
5   10.1   5.4    6   1   male  Ardenne
>
```

Données avec une taille > 10 et un poids < 4

```
> d[d[,"taille"] > 10 & d[,"poids"] < 5,]
   taille poids site nid   sexe   region
59   10.1   4.8    3   2 female Limbourg
66   10.3   4.9    2   5 female  Ardenne
67   10.4   4.9    3   5   male Limbourg
```

Subscripts logiques

Taille et poids des mâles

```
> d[d$sexe == "male", c("taille", "poids")]
   taille poids
1     9.4   5.3
3    11.6   6.3
5    10.1   5.4
8     8.7   5.0
10    9.6   5.5
11   11.2   5.9
13   10.4   6.2
15    9.4   5.3
18    8.0   4.2
19   10.7   6.9
21    8.9   4.8
24    9.3   5.3
25    9.4   5.1
27   10.8   5.4
(...)
```

Subscripts logiques

Données ne provenant pas du Limbourg

```
> d[d$region != "Limbourg",]

   taille poids site nid   sexe        region
2     9.8   5.3    1   2 female      Lorraine
4    10.1   5.7    6   4 female       Ardenne
5    10.1   5.4    6   1   male       Ardenne
6    11.7   6.7    2   2 female       Ardenne
8     8.7   5.0    6   1   male       Ardenne
9     9.3    NA    6   1 female       Ardenne
11   11.2   5.9    2   5   male       Ardenne
13   10.4   6.2    1   5   male      Lorraine
14   10.1   5.8    1   3 female      Lorraine

(...)
```

Subscripts logiques

Données d'Ardenne ou de Haute Ardenne

```
> d[d$region %in% c("Ardenne", "Haute Ardenne"), ]
   taille poids site nid   sexe         region
4    10.1   5.7    6   4 female        Ardenne
5    10.1   5.4    6   1   male        Ardenne
6    11.7   6.7    2   2 female        Ardenne
8     8.7   5.0    6   1   male        Ardenne
9     9.3    NA    6   1 female        Ardenne
11   11.2   5.9    2   5   male        Ardenne
15    9.4   5.3    7   2   male Haute Ardenne
18    8.0   4.2    7   1   male Haute Ardenne
21    8.9   4.8    6   2   male        Ardenne
(...)
```

Toutes les données sauf celles d' Ardenne ou haute Ardenne

```
> d[! d$region %in% c("Ardenne", "Haute Ardenne"), ]
   taille poids site nid   sexe   region
1     9.4   5.3    5   5   male Limbourg
2     9.8   5.3    1   2 female Lorraine
3    11.6   6.3    3   4   male Limbourg
7    10.5   5.6    3   2 female Limbourg
(...)
```

Subscripts logiques

Poids inférieur à 4.5 mg

Attention les NA sont comptés avec et pour éviter tout problème, les données de la ligne correspondante sont transformées en NA

```
> d[d$poids < 4.5,]
     taille poids site  nid   sexe         region
NA       NA    NA <NA> <NA>   <NA>           <NA>
18      8.0   4.2    7    1   male Haute Ardenne
57      8.5   3.7    3    4 female      Limbourg
NA.1     NA    NA <NA> <NA>   <NA>           <NA>
65      8.9   4.2    6    2   male       Ardenne
72      7.7   3.7    6    3   male       Ardenne
```

Idem sans les NA

```
> d[d$poids < 4.5 & !is.na(d$poids),]
   taille poids site nid   sexe         region
18    8.0   4.2    7   1   male Haute Ardenne
57    8.5   3.7    3   4 female      Limbourg
65    8.9   4.2    6   2   male       Ardenne
72    7.7   3.7    6   3   male       Ardenne
```

Subscripts logiques

On peut aussi assigner des nouvelles valeurs avec des subscripts

Pex : attribuer une valeur NA à toutes les mesures de taille ou de poids des nids 4 ou 5

```
> d2 <- d # copie de d
> d2[d2$nid %in% c(4,5), c("taille", "poids")] <- NA
> d2
   taille poids site nid   sexe        region
1      NA    NA    5   5   male      Limbourg
2     9.8   5.3    1   2 female      Lorraine
3      NA    NA    3   4   male      Limbourg
4      NA    NA    6   4 female       Ardenne
5    10.1   5.4    6   1   male       Ardenne
6    11.7   6.7    2   2 female       Ardenne
7    10.5   5.6    3   2 female      Limbourg
(...)
```

Subscripts logiques

On peut aussi utiliser une seule dimension pour les objets à 2 dimensions

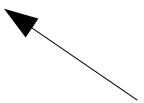
Pex : attribuer la valeur « 1 » à toutes les données d'une matrice > ou = à 1

```
> set.seed(1234)
> (mat <- matrix(floor(runif(32,0,3)), ncol = 8, nrow=4))

     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    2    1    0    0    0    0    2
[2,]    1    1    1    2    0    0    2    0
[3,]    1    0    2    0    0    0    1    1
[4,]    1    0    1    2    0    0    2    0

> mat[mat >= 1] <- 1

> mat
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
[1,]    0    1    1    0    0    0    0    1
[2,]    1    1    1    1    0    0    1    0
[3,]    1    0    1    0    0    0    1    1
[4,]    1    0    1    1    0    0    1    0
```

Cas particulier des listes et data.frames

On peut aussi accéder à un élément d'une liste ou colonne d'un data.frame avec l'opérateur $ suivi de son nom

```
> d$poids
 [1] 5.3 5.3 6.3 5.7 5.4 6.7 5.6 5.0  NA 5.5 5.9 6.3 6.2 5.8 5.3 6.1 6.4
       4.2 6.9 6.0 4.8 4.9 4.6 5.3 5.1 4.5 5.4 5.6 4.6 5.3 5.5 5.8 5.7
    (...)

> set.seed(123)
> mylist <- list(vec1 = c(1,3,4,7,24,78),
+                vec2 = c(1:4),
+                mat = matrix(round(rnorm(9, 5, 1),1), nrow=3, ncol=3))
> mylist
$vec1
[1]  1  3  4  7 24 78

$vec2
[1] 1 2 3 4

$mat
     [,1] [,2] [,3]
[1,]  4.4  5.1  5.5
[2,]  4.8  5.1  3.7
[3,]  6.6  6.7  4.3

> mylist$vec2
[1] 1 2 3 4
```



Création d'une liste exemple

Extraction de vec2 de la liste

Cas particulier des listes et data.frames

$ ne fonctionne pas avec d'autres types d'objets ! (ea matrices)

```
> d2 <- as.matrix(d)

> d2[1:5,]
     taille poids site nid sexe     region
[1,] " 9.4" "5.3" "5"  "5" "male"   "Limbourg"
[2,] " 9.8" "5.3" "1"  "2" "female" "Lorraine"
[3,] "11.6" "6.3" "3"  "4" "male"   "Limbourg"
[4,] "10.1" "5.7" "6"  "4" "female" "Ardenne"
[5,] "10.1" "5.4" "6"  "1" "male"   "Ardenne"

> d2$poids
Erreur dans d2$poids : $ operator is invalid for atomic vectors
```

Cas particulier des listes et data.frames

Pour les listes, l'opérateur [ ] extrait un élément qui sera une liste
Et l'opérateur [[ ]] extrait un élément de classe et mode identique à cet élément --> en général on utilise [[ ]]

```
> mylist[[3]]
     [,1] [,2] [,3]
[1,]  4.4  5.1  5.5
[2,]  4.8  5.1  3.7
[3,]  6.6  6.7  4.3

> mylist[3]
$mat
     [,1] [,2] [,3]
[1,]  4.4  5.1  5.5
[2,]  4.8  5.1  3.7
[3,]  6.6  6.7  4.3

> class(mylist[[3]]) ; mode(mylist[[3]])
[1] "matrix"
[1] "numeric"

> class(mylist[3]) ; mode(mylist[3])
[1] "list"
[1] "list"
```

Cas particulier des listes et data.frames

```
> mylist[[3]]
     [,1] [,2] [,3]
[1,]  4.4  5.1  5.5
[2,]  4.8  5.1  3.7
[3,]  6.6  6.7  4.3

> mylist[[3]] [1,3]
[1] 5.5

> mylist[3] [1,3] # ne fonctionne pas !
Erreur dans mylist[3][1, 3] : nombre de dimensions incorrect
```

Extraction du 3ème élément de la liste
Puis de l'élément de la première ligne et 3ème colonne

Ne fonctionne pas car mylist[3] est une liste et pas une matrice

Cas particulier des listes et data.frames

La plupart des résultats d'analyses statistiques sont stockés dans des objets héritant de la classe liste. On accède donc à leur éléments avec $

```
> set.seed(123)
> x1 <- runif(50, 0, 10) ; x2 <- runif(50, 0, 10)
> y <- 1.5*x1 + 0.5*x2 + rnorm(50, 0, 3)
>
> regr <- summary(lm(y ~ x1 + x2))
> str(regr)
List of 11
 $ call          : language lm(formula = y ~ x1 + x2)
 $ terms         :Classes 'terms', 'formula' length 3 y ~ x1 + x2
  .. ..- attr(*, "variables")= language list(y, x1, x2)
  .. ..- attr(*, "factors")= int [1:3, 1:2] 0 1 0 0 0 1
  .. .. ..- attr(*, "dimnames")=List of 2

(...)

> regr$r.squared
[1] 0.7359259
> regr$coefficients[, c("Estimate", "Std. Error")]
             Estimate Std. Error
(Intercept) 0.7781452  1.0162212
x1          1.4168929  0.1344961
x2          0.5195612  0.1430568
```

Cas particulier des listes et data.frames

Pour certains de ces objets, de type S4 (par opposition à S3), on accède aux éléments avec l'opérateur @

```
> set.seed(123)
> x1 <- runif(50, 0, 10) ; x2 <- as.factor(floor(runif(50, 1, 5)))
> y <- 1.5*x1 + rnorm(50, 0, 3)
>
> library(lme4)
> regr <- summary(lmer(y ~ x1 + (1|x2)))
>
> str(regr)
Formal class 'summary.mer' [package "lme4"] with 42 slots
  ..@ methTitle: chr "Linear mixed model fit by REML"
  ..@ logLik   :Class 'logLik' : -121 (df=4)
  ..@ ngrps    : Named int 4
  .. ..- attr(*, "names")= chr "x2"
  ..@ sigma    : num 2.73
(...)

> regr@coefs[, c("Estimate", "Std. Error")]
             Estimate Std. Error
(Intercept) 0.8639557  0.7910858
x1          1.4183353  0.1327041
```

La fonction `subset()`

Fonction offrant certaines facilités pour extraire des données des objets :
- ne prend pas en compte les NA
- on peut utiliser les noms de variables à la place de numéros de colonnes

Par contre :
- on ne peut pas utiliser de nombres pour sélectionner les lignes
- on ne peut pas attribuer une valeur `subset() <-`

```
> d[d$poids < 4.5,]
     taille poids site  nid   sexe        region
NA       NA    NA <NA> <NA>   <NA>          <NA>
18      8.0   4.2    7    1   male Haute Ardenne
57      8.5   3.7    3    4 female      Limbourg
NA.1     NA    NA <NA> <NA>   <NA>          <NA>
65      8.9   4.2    6    2   male       Ardenne
72      7.7   3.7    6    3   male       Ardenne

> subset(d, poids < 4.5)
   taille poids site nid   sexe        region
18    8.0   4.2    7   1   male Haute Ardenne
57    8.5   3.7    3   4 female      Limbourg
65    8.9   4.2    6   2   male       Ardenne
72    7.7   3.7    6   3   male       Ardenne
```

La fonction subset()

```
> subset(d, subset = taille >11 , select = site:sexe)
   site nid   sexe
3     3   4   male
6     2   2 female
11    2   5   male
16    1   3 female
30    1   1   male
44    7   5   male
45    5   2 female
54    6   1   male
56    1   4   male
70    6   1 femalee
```

équivalent à `d[,3:5]`

La fonction subset()

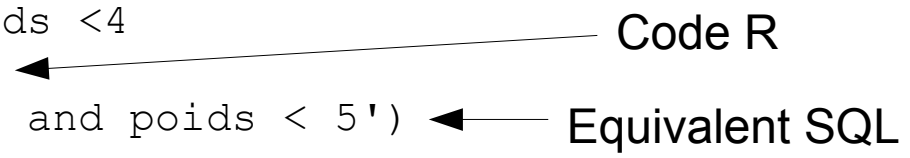
On ne peut pas attribuer une valeur `subset() <-`
Alors qu'on peut le faire avec le subscripting classique

```
> d2 <- d # copie de d

> subset(d2, nid %in% c(4,5), select = c("taille", "poids")) <- NA
Erreur dans subset(d2, nid %in% c(4, 5), select = c("taille", "poids")) <- NA :
  impossible de trouver la fonction "subset<-"

> d2[d2$nid %in% c(4,5), c("taille", "poids")] <- NA

> d2
   taille poids site nid   sexe        region
1      NA    NA    5   5   male      Limbourg
2     9.8   5.3    1   2 female      Lorraine
3      NA    NA    3   4   male      Limbourg
4      NA    NA    6   4 female       Ardenne
5    10.1   5.4    6   1   male       Ardenne
6    11.7   6.7    2   2 female       Ardenne
7    10.5   5.6    3   2 female      Limbourg
(...)
```

Exécuter des requêtes SQL sur un data.frame

C'est possible avec le package `sqldf` et la fonction du même nom

```
library(sqldf)
# données avec une taille > 10 et un poids <4
d[d[,"taille"] > 10 & d[,"poids"] < 5,]
sqldf('select * from d where taille > 10 and poids < 5')

# taille et poids des mâles,
d[d$sexe == "male", c("taille", "poids")]
sqldf('select taille, poids from d where sexe = "male"')

# données ne provenant pas du Limbourg
d[d$region != "Limbourg",]
sqldf('select * from d where region not like "Limbourg"')

# données en Ardenne ou haute ardenne
d[d$region %in% c("Ardenne", "Haute Ardenne"), ]
sqldf('select * from d where region in ("Ardenne", "Haute Ardenne")')

# toutes les données sauf les données d'Ardenne ou haute ardenne
d[! d$region %in% c("Ardenne", "Haute Ardenne"), ]
sqldf('select * from d where region not in ("Ardenne", "Haute Ardenne")')

# poids inférieur à 4.5 mg
d[d$poids < 4.5 & !is.na(d$poids),]
sqldf('select * from d where poids < 4.5')
```



Résumé : subscripting

i, j = subscripts
= vecteur numérique (positif ou négatif), character ou logique

`vector[i]`	élément(s) i
`matrix[i,j]`	ligne(s) i, colonne(s) j
`matrix[i]`	élément i (par colonne, comme un vecteur)
`list[i]`	retourne une liste contenant l(es) élément(s) i
`list[[3]]`	retourne l'élément 3 dans sa classe d'origine
`data.frame[i,j]`	ligne(s) i, colonne(s) j (comme le ferait une matrice)
`data.frame[,3]`	retourne la colonne 3 sous forme de vecteur
`data.frame[3]`	retourne la colonne 3 sous forme de data.frame
`data.frame[,c(2,3)]` `data.frame[c(2,3)]`	retourne les colonnes 2 et 3 sous forme de data.frame

Exercices : subscripting

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

Quelques fonctions simples

Compter le nombre de caractères : `nchar()`
Couper une chaîne de caractères : `strsplit()`

```
> char <- c("Ardenne", "Famenne", "Haute Ardenne", "Ardenne", "Limbourg", "Lorraine")
>
> nchar(char)
[1]  7  7 13  7  8  8
>
> strsplit("Ceci est une phrase", split=" ")
[[1]]
[1] "Ceci"   "est"    "une"    "phrase"

> strsplit("Ceci est une phrase", split="")
[[1]]
 [1] "C" "e" "c" "i" " " " " "e" "s" "t" " " " " "u" "n" "e" " " " " "p" "h" "r" "a" "s" "e"

> strsplit(char, split="")
[[1]]
[1] "A" "r" "d" "e" "n" "n" "e"

[[2]]
[1] "F" "a" "m" "e" "n" "n" "e"

[[3]]
 [1] "H" "a" "u" "t" "e" " " " " "A" "r" "d" "e" "n" "n" "e"

[[4]]
[1] "A" "r" "d" "e" "n" "n" "e"

(...)
```

Quelques fonctions simples

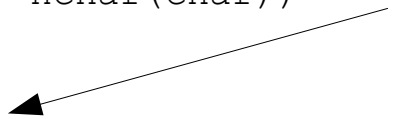
Modifier la casse `toupper() tolower()`

Extraire des caractères `substring()`

```
> toupper(char)
[1] "ARDENNE"       "FAMENNE"       "HAUTE ARDENNE" "ARDENNE"       "LIMBOURG"
"LORRAINE"
> tolower(char)
[1] "ardenne"       "famenne"       "haute ardenne" "ardenne"       "limbourg"
"lorraine"
```

```
> substring(char, first=1, last=4)
[1] "Arde" "Fame" "Haut" "Arde" "Limb" "Lorr"

> substring(char, first = nchar(char)-3, last = nchar(char))
[1] "enne" "enne" "enne" "enne" "ourg" "aine"
```

On peut aussi assigner de nouveaux caractères à une position donnée



```
> substring(char, first = nchar(char)-3, last = nchar(char)) <- "---"
> char
[1] "Ard---e"       "Fam---e"       "Haute Ard---e" "Ard---e"       "Limb---g"
"Lorr---e"
```

Quelques fonctions simples

Remplacer certains caractères par d'autres :

`chartr()`

```
> char <- c("Ardenne", "Famenne", "Haute Ardenne", "Ardenne", "Limbourg", "Lorraine")

> chartr(old = "ae", new= "uo", x= char)
[1] "Ardonno"       "Fumonno"       "Huuto Ardonno" "Ardonno"       "Limbourg"
"Lorruino"

> chartr(old = "aeA", new= "uoU", x= char)
[1] "Urdonno"       "Fumonno"       "Huuto Urdonno" "Urdonno"       "Limbourg"
"Lorruino"
```

Expressions Régulières

Expressions régulières
=
chaîne de caractères décrivant un ensemble de chaînes possibles selon une syntaxe précise

Dans un moteur de recherche classique ou en SQL :
recherche "Coccinell*" pour chercher la chaine de caractère "Coccinell" suivie de n'importe quoi d'autre (ie : Coccinella, Coccinellidae,...)
(attention ce n'est pas la même syntaxe dans les regexpr mais l'idée est la même)

Expressions régulières :
offrent beaucoup plus de possibilités
Mais plus complexes...
Peuvent aussi s'utiliser en dehors de R (tableurs, traitements de texte, tableur,...)

Si ça vous effraye, retenez seulement quelques éléments de syntaxe très simple :

Coccinell ← SQL, Google, etc...

Coccinell ← Expressions régulières

Expressions Régulières

!!! Avertissement !!!

Les expressions régulières vont peut-être vous paraître très compliquées (sans connaissance préalable).

Elles permettent de faire beaucoup de choses mais on peut utiliser R pendant des années sans jamais les utiliser ou presque

Donc si ça vous effraye/vous décourage, retenez seulement quelques éléments de syntaxe très simple et ne vous acharnez pas

Pex : pour chercher tous les éléments qui contiennent "Coccinell" dans un vecteur, on fait :

```
v[grep("Coccinell", v)]
```

Expressions Régulières

Quelques exemples pour montrer les possibilités :

```
> char <- c("Adalia bipunctata", "Adalia 2-punctata", " Adalia 2punctata",
        "Calvia 14-guttata", "Adalia 10-punctata", "Calvia decemguttata",
        "Leptophyes  punctatissima", "Adalia decemguttata", "Cryptocephalus
        bipunctatus", "Anthidium punctatum", "Lasioglossum punctatissimum",
        "Punctagonum bicolor", "Ectemnius guttatus", "Nuctenea umbratica", "999")
```

`grep()` donne la position des éléments contenant "punct"

```
> grep("punct", char)
[1]  1  2  3  5  7  9 10 11
```

Éléments contenant "gutatta"

```
> char[grep("guttata", char)]
[1] "Calvia 14-guttata"   "Calvia decemguttata" "Adalia decemguttata"
```

Éléments contenant 10 ou decem

```
> char[grep("10|decem", char)]
[1] "Adalia 10-punctata"  "Calvia decemguttata" "Adalia decemguttata"
```

Expressions Régulières

Éléments contenant "um" à la fin d'un mot (ie suivi d'une espace) ou à la fin de la séquence

```
> char[grep("um |um$", char)]
[1] "Anthidium punctatum"  "Lasioglossum punctatissimum" "Punctagonum bicolor"
```

Éléments comprenant des chiffres quelconques précédés d'une espace et suivis d'un tiret ou pas

```
> char[grep(" [0-9]-?", char)]
[1] "Adalia 2-punctata"  " Adalia 2punctata"  "Calvia 14-guttata"  "Adalia
10-punctata"
```

Éléments comprenant punct ou guttat mais précédés soit de lettres quelconques soit de chiffres quelconques avec ou sans tiret entre 2

```
> char[grep("([0-9a-z])-?(punct|guttat)", char)]
[1] "Adalia bipunctata"          "Adalia 2-punctata"          " Adalia 2punctata"
[4] "Calvia 14-guttata"          "Adalia 10-punctata"         "Calvia decemguttata"
[7] "Adalia decemguttata"        "Cryptocephalus bipunctatus"
```

Idem mais avec obligatoirement un tiret

```
> char[grep("([0-9a-z])-+(punct|guttat)", char)]
[1] "Adalia 2-punctata"  "Calvia 14-guttata"  "Adalia 10-punctata"
```

Expressions Régulières : syntaxe

^	Marque le début de la chaîne (ligne)
$	Marque la fin de la chaîne (ligne)
.	représente n'importe quel caractère unique
|	sépare des patterns alternatifs ("ou")
( )	groupe des patterns + capture (voir plus loin)
[abc]	classe de caractères : représente tout caractère présent entre les crochets, ici a ou b ou c
[^abc]	représente tout caractère sauf ceux présent entre les crochets, ici tout sauf a ou b ou c
[a-f]	tiret : gamme de caractères
*	0 ou plus occurrences du caractère précédent
?	0 ou 1 occurrence du caractère précédent
+	1 ou plus occurrences du caractère précédent
{n}	n occurrences du caractère précédent
{n,}	minimum n occurrences du caractère précédent
{n,m}	entre n et m occurrences du caractère précédent
\\	échappement pour tous les caractères réservés c'est à dire tous ceux ci-dessus

Particulier à R regexpr en dehors de R : simple \

Voir ?regex pour plus de détails et de possibilités

Expressions Régulières : syntaxe

```
> char <- c("aaa", "bbb", "abc", "abcabcabc", "abcabc", "abd", "def", "abccc", "d f","ab")


> char[grep("abc", char)] # tout ce qui contient "abc"
[1] "abc"       "abcabcabc" "abcabc"    "abccc"

> char[grep("^abc$", char)] # "abc" exactement (entre le début et la fin de la séquence)
[1] "abc"

> char[grep("[abf]", char)] # a ou b ou f
 [1] "aaa"  "bbb"  "abc"  "abcabcabc"  "abcabc"  "abd"  "def"  "abccc"  "d f"   "ab"

> char[grep("(abc){3}", char)]# abc 3 fois
[1] "abcabcabc"

> char[grep("(abc){2,}", char)]# abc au moins 2 fois
[1] "abcabcabc" "abcabc"

> char[grep("abc{3}", char)]# a puis b puis 3 fois c
[1] "abccc"

> char[grep("[a-d].[df]", char)]#a ou b ou c ou d puis un caractère quelconque puis d ou f
[1] "abd" "def" "d f"
```

Expressions Régulières : syntaxe

```
> char <- c("aaa", "bbb", "abc", "abcabcabc", "abcabc", "abd", "def", "abccc", "d f","ab")


> char[grep("abc+$", char)]# ab puis c au moins 1 fois puis fin de séquence
[1] "abc"       "abcabcabc" "abcabc"    "abccc"

> char[grep("abc?$", char)]# ab puis c 0 ou 1 fois puis fin de séquence
[1] "abc"       "abcabcabc" "abcabc"    "ab"

> char[grep("abc*$", char)]# ab puis c au moins 0 fois puis fin de séquence
[1] "abc"       "abcabcabc" "abcabc"    "abccc"     "ab"
```

Expressions Régulières : syntaxe

```
> char <- c("DSC_3567.JPG", "IMG_3564.jpeg", "maphoto.JPG", "IMG_3564.txt",
"img_5678.JPG", "img5678.JPG", "_.jpg")

> # séquence terminant par .jpeg ou .jpg en majuscule ou minuscule.
    NB : le point se note \\.
> char[grep("\\.(JPG|jpg|JPEG|jpeg)$", char)]
[1] "DSC_3567.JPG"  "IMG_3564.jpeg" "maphoto.JPG"   "img_5678.JPG"  "img5678.JPG"
[6] "_.jpg"

> char[grep("\\.(JPG|JPEG)$", char, ignore.case=TRUE)]# idem
[1] "DSC_3567.JPG"  "IMG_3564.jpeg" "maphoto.JPG"   "img_5678.JPG"  "img5678.JPG"
[6] "_.jpg"

> char[grep("\\.JPE?G$", char, ignore.case=TRUE)]# idem mais syntaxe plus simple
[1] "DSC_3567.JPG"  "IMG_3564.jpeg" "maphoto.JPG"   "img_5678.JPG"
[5] "img5678.JPG"   "_.jpg"

> # séquence commençant par 3 lettres majuscules quelconques
> char[grep("^[A-Z]{3}", char)]
[1] "DSC_3567.JPG"  "IMG_3564.jpeg" "IMG_3564.txt"

> # underscore(_) suivi de 0 ou plus caractères quelconques (.*) suivi d'un point (\\.)
suivi de JPG ou JPEG (JPG|JPEG) en fin de chaîne ($)
> char[grep("_.*\\.JPE?G$", char, ignore.case=TRUE)]
[1] "DSC_3567.JPG"  "IMG_3564.jpeg" "img_5678.JPG"  "_.jpg"
```

Fonctions basées sur les Expressions Régulières

`grep()` et `grepl()` :
donnent la position des éléments contenant le pattern
sous forme de vecteur d'entiers ou logique

`grep()` peut aussi retourner les éléments correspondant
plutôt que leur numéro avec l'option value = TRUE

```
> char <- c("DSC__3567.JPG", "I_3564_M2.jpeg", "img5678.JPG", "_.jpg")
>
> grep("_", char)
[1] 1 2 4

> grepl("_", char)
[1]  TRUE  TRUE FALSE  TRUE

> grep("_", char, value = TRUE)
[1] "DSC__3567.JPG"  "I_3564_M2.jpeg" "_.jpg"
```

Fonctions basées sur les Expressions Régulières

`regexpr()` et `gregexpr()` :
donnent la position du pattern (ou -1) dans chaque chaîne de caractères du vecteur.
`regexpr()` donne la première occurrence
`gregexpr()` donne toutes les occurrences sous forme de liste

```
> regexpr("_", char)
[1]  4  2 -1  1
attr(,"match.length")
[1]  1  1 -1  1
attr(,"useBytes")
[1] TRUE

> gregexpr("_", char)
[[1]]
[1] 4 5
attr(,"match.length")
[1] 1 1
attr(,"useBytes")
[1] TRUE

[[2]]
[1] 2 7
attr(,"match.length")
[1] 1 1
attr(,"useBytes")
[1] TRUE
(...)
```

Fonctions basées sur les Expressions Régulières

`sub()` et `gsub()` :
remplacent un pattern par une chaîne de caractères
--> équivalent de "chercher-remplacer"

`sub()` remplace la première occurrence
`gsub()` remplace toutes les occurrences

```
> char <- c("DSC__3567.JPG", "I_3564_M2.jpeg", "img5678.JPG", "_.jpg")

> sub(pattern = "_", replacement = "-", x = char)
[1] "DSC-_3567.JPG"  "I-3564_M2.jpeg" "img5678.JPG"    "-.jpg"

> gsub(pattern = "_", replacement = "-", x = char)
[1] "DSC--3567.JPG"  "I-3564-M2.jpeg" "img5678.JPG"    "-.jpg"
```

Fonctions basées sur les Expressions Régulières

`sub()` et `gsub()` :

On peut aussi "capturer" une partie du pattern pour l'utiliser pour construire le terme de remplacement.

Chaque paire de parenthèses dans le pattern correspond à une capture que l'on pourra réutiliser avec \\1 pour la première parenthèse, \\2 pour la deuxième etc...

Exemple : on veut récupérer le code à 4 chiffres et l'extension pour construire le nouveau nom de type "GSM_1234.jpg"

".*" : N'importe quels caractères au moins 0 fois

Pattern : ".*([0-9]{4}).*(\\..{3-4}$)"

4 chiffres, première capture = \\1

un point suivi de 3 à 4 caractères puis de la fin de la chaîne deuxième capture = \\2

```
> char <- c("DSC__3567.JPG", "I_3564_M2.jpeg", "img5678.JPG", "_6783.tiff",
"DSC8903_nikonD80.jpg")

> gsub(pattern = ".*([0-9]{4}).*(\\..*$)", replacement = "GSM_\\1\\2", x = char)
[1] "GSM_3567.JPG"  "GSM_3564.jpeg" "GSM_5678.JPG"  "GSM_6783.tiff"
"GSM_8903.jpg" )
```

Fonctions basées sur les Expressions Régulières

`sub()` et `gsub()` :
Autre exemple de capture
Remplacer Chorthippus brunneus par C.brunneus

```
> noms <- c("Chorthippus brunneus", "Chorthippus parallelus",
"Chrysochraon dispar", "Euchorthippus declivus")

> gsub("^(.).* (.*)", "\\1\\.\\2", noms)
[1] "C.brunneus"   "C.parallelus" "C.dispar"     "E.declivus"
```

Exercices : manipulation de caractères

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

Fusion de tableaux de données

Génération de jeu de données exemple

```
> obs <- data.frame(
+     site = rep(c("site1", "site2", "site3", "site4", "site5"), c(3,2,1,2,3)),
+     sp = c("sp1", "sp2", "sp3")[c(1:3,1:2,1,1:2,1:3)],
+     abund = rpois(11, 25))

> obs
    site  sp abund
1  site1 sp1    22
2  site1 sp2    21
3  site1 sp3    27
4  site2 sp1    16
5  site2 sp2    35
6  site3 sp1    34
7  site4 sp1    21
8  site4 sp2    28
9  site5 sp1    25
10 site5 sp2    23
11 site5 sp3    21
```

Fusion de tableaux de données

Génération de jeu de données exemple

```
> set.seed(123)
> sites <- data.frame(
+     sitename = c("site1", "site3", "site4", "site5", "site6", "site7"),
+     temp = round(rnorm(6, 15, 5), 2),
+     hum = round(runif(6, 65, 95), 2))

> sites
  sitename  temp   hum
1    site1 12.20 85.33
2    site3 13.85 82.18
3    site4 22.79 68.09
4    site5 15.35 91.99
5    site6 15.65 72.38
6    site7 23.58 66.26
```

Combiner des tableaux de même longueur

`rbind()`, `cbind()` et `data.frame()`

```
> rbind(sites, sites)
   sitename  temp   hum
1     site1 12.20 85.33
2     site3 13.85 82.18
3     site4 22.79 68.09
4     site5 15.35 91.99
5     site6 15.65 72.38
6     site7 23.58 66.26
7     site1 12.20 85.33
8     site3 13.85 82.18
9     site4 22.79 68.09
10    site5 15.35 91.99
11    site6 15.65 72.38
12    site7 23.58 66.26
```

```
> cbind(sites, sites)
  sitename  temp   hum sitename  temp   hum
1    site1 12.20 85.33    site1 12.20 85.33
2    site3 13.85 82.18    site3 13.85 82.18
3    site4 22.79 68.09    site4 22.79 68.09
4    site5 15.35 91.99    site5 15.35 91.99
5    site6 15.65 72.38    site6 15.65 72.38
6    site7 23.58 66.26    site7 23.58 66.26
```

Accepte des noms de colonnes identiques
--> pas idéal !!

```
> data.frame(sites, sites)
  sitename  temp   hum sitename.1 temp.1 hum.1
1    site1 12.20 85.33      site1  12.20 85.33
2    site3 13.85 82.18      site3  13.85 82.18
3    site4 22.79 68.09      site4  22.79 68.09
4    site5 15.35 91.99      site5  15.35 91.99
5    site6 15.65 72.38      site6  15.65 72.38
6    site7 23.58 66.26      site7  23.58 66.26
```

N'accepte pas des noms de colonnes identiques
--> transforme les noms

Combiner des tableaux de même longueur

Quand on colle par ligne des data.frames il faut que les noms de colonnes correspondent

```
> rbind(sites, c("site1", 12, 79))
  sitename  temp   hum
1    site1  12.2 85.33
2    site3 13.85 82.18
3    site4 22.79 68.09
4    site5 15.35 91.99
5    site6 15.65 72.38
6    site7 23.58 66.26
7    site1    12    79

> rbind(sites, data.frame("site1", 12, 79))
Erreur dans match.names(clabs, names(xi)) :
  les noms ne correspondent pas aux noms précédents
```

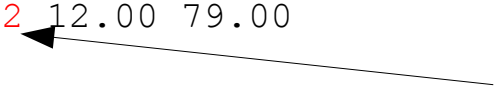
Vecteur sans nom --> ok

data.frame sans nom --> ne fonctionne pas

Combiner des tableaux de même longueur

Si on veut coller des lignes avec des nouveaux niveaux de facteurs, il faut passer par un data.frame

```
> rbind(sites, c("site22", 12, 79))
  sitename  temp   hum
1    site1  12.2 85.33
2    site3 13.85 82.18
3    site4 22.79 68.09
4    site5 15.35 91.99
5    site6 15.65 72.38
6    site7 23.58 66.26
7     <NA>    12    79
Message d'avis :
In `[<-.factor`(`*tmp*`, ri, value = "site22") :
  niveau de facteur incorrect, NAs générés

> rbind(sites, data.frame(sitename="site22", temp=12, hum=79))
  sitename  temp   hum
1    site1 12.20 85.33
2    site3 13.85 82.18
3    site4 22.79 68.09
4    site5 15.35 91.99
5    site6 15.65 72.38
6    site7 23.58 66.26
7   site22 12.00 79.00
```



Les niveaux du facteur ont été mis à jour

Fusionner des tableaux avec `merge()`

Fusionner des tableaux selon 1 ou plusieurs colonnes communes
Équivalent des JOIN en SQL

```
> merge(obs, sites, by.x="site", by.y="sitename")
   site  sp abund  temp   hum
1 site1 sp2    21 12.20 85.33
2 site1 sp3    27 12.20 85.33
3 site1 sp1    22 12.20 85.33
4 site3 sp1    34 13.85 82.18
5 site4 sp1    21 22.79 68.09
6 site4 sp2    28 22.79 68.09
7 site5 sp1    25 15.35 91.99
8 site5 sp2    23 15.35 91.99
9 site5 sp3    21 15.35 91.99
```

On garde uniquement les valeurs communes aux deux tableaux
"Inner Join"

```
> merge(obs, sites, by.x="site", by.y="sitename", all=TRUE)
    site   sp abund  temp   hum
1  site1  sp2    21 12.20 85.33
2  site1  sp3    27 12.20 85.33
3  site1  sp1    22 12.20 85.33
4  site2  sp2    35    NA    NA
5  site2  sp1    16    NA    NA
6  site3  sp1    34 13.85 82.18
7  site4  sp1    21 22.79 68.09
8  site4  sp2    28 22.79 68.09
9  site5  sp1    25 15.35 91.99
10 site5  sp2    23 15.35 91.99
11 site5  sp3    21 15.35 91.99
12 site6 <NA>    NA 15.65 72.38
13 site7 <NA>    NA 23.58 66.26
```

On garde toutes les valeurs
"Outer Join"

Fusionner des tableaux avec `merge()`

```
> merge(obs, sites, by.x="site", by.y="sitename", all.x=TRUE)
    site  sp abund  temp   hum
1  site1 sp2    21 12.20 85.33
2  site1 sp3    27 12.20 85.33
3  site1 sp1    22 12.20 85.33
4  site2 sp2    35    NA    NA
5  site2 sp1    16    NA    NA
6  site3 sp1    34 13.85 82.18
7  site4 sp1    21 22.79 68.09
8  site4 sp2    28 22.79 68.09
9  site5 sp1    25 15.35 91.99
10 site5 sp2    23 15.35 91.99
11 site5 sp3    21 15.35 91.99
```

On garde toutes les valeurs du premier tableau
" Left Join"

```
> merge(obs, sites, by.x="site", by.y="sitename", all.y=TRUE)
    site   sp abund  temp   hum
1  site1  sp2    21 12.20 85.33
2  site1  sp3    27 12.20 85.33
3  site1  sp1    22 12.20 85.33
4  site3  sp1    34 13.85 82.18
5  site4  sp1    21 22.79 68.09
6  site4  sp2    28 22.79 68.09
7  site5  sp1    25 15.35 91.99
8  site5  sp2    23 15.35 91.99
9  site5  sp3    21 15.35 91.99
10 site6 <NA>    NA 15.65 72.38
11 site7 <NA>    NA 23.58 66.26
```

On garde toutes les valeurs du deuxième tableau
" Right Join"

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de dates et caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

Créer ses propres fonctions

Exemple 1 : fonction calculant la moyenne

```
> moyenne <- function(x) {
+     sum(x)/length(x)
+ }

> moyenne(c(8, 12, 14))
[1] 11.33333

> mean(c(8, 12, 14))
[1] 11.33333
```

Créer ses propres fonctions

Exemple 2 : fonction calculant le périmètre et la surface d'un cercle
NB : La fonction retourne la dernière valeur imprimée. Si on veut retourner une autre valeur, utiliser `return()` en fin de fonction

```
> circle <- function(R){
+     perim <- 2*pi*R
+     surf <- pi*R^2
+     result <- cbind(perim, surf)
+     colnames(result) <- c("perimeter", "area")
+ }
> circle(c(12, 8, 9))
```

Ne fonctionne pas car on n'a pas utilisé `return()`

```
> circle <- function(R){
+     perim <- 2*pi*R
+     surf <- pi*R^2
+     result <- cbind(perim, surf)
+     colnames(result) <- c("perimeter", "area")
+     return(result)
+ }

> circle(c(12, 8, 9))
     perimeter     area
[1,]  75.39822 452.3893
[2,]  50.26548 201.0619
[3,]  56.54867 254.4690
```

Créer ses propres fonctions

Exemple 3 : fonction calculant la distance euclidienne au centre (par défaut) ou à un autre série de points

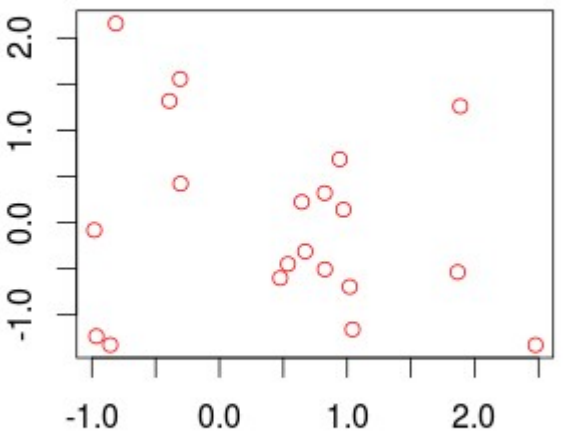
Comme dans les fonctions de R on peut définir plusieurs arguments et des valeurs par défaut

```
> euclid <- function(x1, y1, x2=0, y2=0){
+     base <- x2-x1
+     hauteur <- y2-y1
+     hypothenuse <- sqrt(base^2 + hauteur^2)
+     return(hypothenuse) # facultatif
+ }
>
> coord <- data.frame(x = rnorm(5), y = rnorm(5))
> coord2 <- data.frame(x = rnorm(5), y = rnorm(5))
>
> euclid(x1 = coord$x, y1 = coord$y)
[1] 1.11695863 0.07543967 0.31234416 2.17275407 0.87159508

> euclid(x1 = coord$x, y1 = coord$y, x2 = coord2$x, y2 = coord2$y)
[1] 1.2356738 1.6076108 0.3880114 3.0107558 0.5067120
```

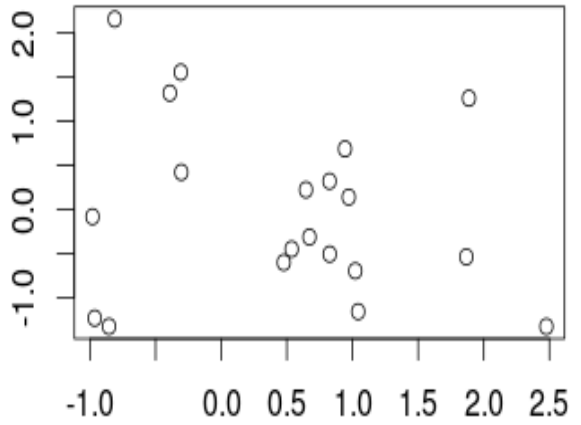
Créer ses propres fonctions

L'argument "..." permet de passer des arguments quelconques à une fonction à l'intérieur de votre fonction

```
> a <- rnorm(20)
> b <- rnorm(20)


> monplot <- function(x, y, ...) {
+     plot(x, y,...)
+ }
> monplot(a, b, col = "red")
```




```
> monplot <- function(x, y) {
+     plot(x, y)
+ }
> monplot(a, b, col = "red")
Erreur dans monplot(a, b, col = "red") :
  argument(s) inutilisé(s) (col = "red")


> monplot <- function(x, y,...) {
+     plot(x, y)
+ }
> monplot(a, b, col = "red")
```



Exécutions conditionnelles

`if (cond) {expr}`

```
> if(0.002 <= 0.05) print("significant")
[1] "significant"
> if(0.67 <= 0.05) print("significant")
```

Un seul TRUE ou un seul FALSE
Pas de vecteur logique de longueur > 1

```
> signif <- function(p) {
+     if(p <= 0.05) {
+         print("significant")
+     }
+ }
>
> signif(0.002)
[1] "significant"

> signif(0.67)
>
```

Accolades pas nécessaires
si il n'y a qu'une seule commande

Exécutions conditionnelles

if (cond) {expr} else {alt.expr}

```
> signif <- function(p) {
+     if(p <= 0.05) {
+         print("significant")
+     }else{
+         print("not significant")
+     }
+ }
> signif(0.002)
[1] "significant"

> signif(0.67)
[1] "not significant"

> signif(c(0.002, 0.67))
[1] "significant"
Message d'avis :
In if (p <= 0.05) { :
  la condition a une longueur > 1 et seul le premier élément est utilisé

> p =10
> if(p <= 0.05) {
+     print("significant")
+ }
> else{
Erreur : 'else' inattendu(e) in "else"
>     print("not significant")
[1] "not significant"
> }
Erreur : '}' inattendu(e) in "}"
```

Opération pas vectorisée !

Toujours mettre else juste après }
pour éviter les problèmes

Exécutions conditionnelles

Pas très important

switch (expr,...)

Choix entre différentes alternatives

```
> signif <- function(p){
+     switch(EXPR = p,
+            "0.10" = "(*)",
+            "0.05" = "*",
+            "0.01" = "**",
+            "0.001" = "***",
+            "les valeurs doivent être exactement 0.10, 0.05, 0.01 ou 0.001")
+ }

> signif("0.01")
[1] "**"

> signif("0.05")
[1] "*"

> signif("0.22")
[1] "les valeurs doivent être exactement 0.10, 0.05, 0.01 ou 0.001"

> signif("0.01", "0.05", "0.22")
Erreur dans signif("0.01", "0.05", "0.22") :
  argument(s) inutilisé(s) ("0.05", "0.22")
```

valeurs retournée si p ne correspond à aucune des valeurs prévues

Opération pas vectorisée !

Exécutions conditionnelles

ifelse (test, yes, no)

```
> # ifelse(test, yes, no)
>
> signif <- function(p){
+     ifelse(p<=0.05, "significant", "not significant")
+ }
> signif(0.002)
[1] "significant"

> signif(0.67)
[1] "not significant"

> signif(c(0.002, 0.67)) # gros avantage : la fonction est vectorisée !!
[1] "significant"     "not significant"
```

ifelse imbriqués :

```
> signif <- function(p) {
+     ifelse(p<0.001,"***",
+            ifelse(p<0.01,"**",
+                   ifelse(p<0.05,"*",
+                          ifelse(p<=0.1,"(*)",""))))
+ }
>
> signif(c(0.002, 0.67, 0.00001, 0.078, 0.008))
[1] "**"   ""     "***" "(*)" "**"
```

Gros avantage :
opération vectorisée !

Boucles explicites

Structures de langage classiques permettant de répéter des opérations

NB : Dans R il est en général recommandé de ne pas utiliser de boucles. On préfère utiliser la vectorisation quand c'est possible et des fonctions spéciales de la famille apply qui permettent de se passer de ces boucles.

Ces solutions propres à R sont en général plus simple à écrire et plus rapides (voir plus loin)

Boucles explicites

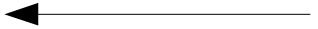
Boucles `for (var in seq) expr`

De loin la plus utilisée

Principe : seq est un vecteur et var représente chaque élément de ce vecteur. On prend d'abord la première valeur de var et on exécute "expr" avec cette valeur. Ensuite on prend la seconde valeur de var et on recommence jusqu'à ce que toutes les valeurs de var aient été utilisées

Exemple : transformation de données en données binaires

```
> mat <- matrix(floor(runif(12,0,4)), nrow=4, ncol=4)
> mat
     [,1] [,2] [,3] [,4]
[1,]    3    2    3    3
[2,]    0    2    0    0
[3,]    3    1    3    3
[4,]    2    0    2    2

> for(i in 1:length(mat)){
+     if(mat[i] > 0){
+         mat[i] <- 1
+     }
+ }
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    0    1    0    0
[3,]    1    1    1    1
[4,]    1    0    1    1
```



On crée une matrice pour l'exemple

Boucles explicites

En pratique on ne fonctionnera jamais comme ça...
On utilisera toujours les puissantes possibilités de vectorisation et subscripting de R quand c'est possible

```
> mat <- matrix(floor(runif(12,0,4)), nrow=4, ncol=4)
> mat
     [,1] [,2] [,3] [,4]
[1,]    0    1    1    0
[2,]    2    3    2    2
[3,]    2    3    3    2
[4,]    3    2    3    3

> ifelse(mat == 0, 0, 1)
     [,1] [,2] [,3] [,4]
[1,]    0    1    1    0
[2,]    1    1    1    1
[3,]    1    1    1    1
[4,]    1    1    1    1
>
```

ou encore plus simple :

```
> mat[mat!=0] <- 1
> mat
     [,1] [,2] [,3] [,4]
[1,]    0    1    1    0
[2,]    1    1    1    1
[3,]    1    1    1    1
[4,]    1    1    1    1
```

Boucles explicites

Pas très important

Autres types de structures beaucoup moins utilisées :
while, repeat, break, next

Boucle `while(cond) expr`

Tant que la condition est vraie, on répète l'expression

```
> mat <- matrix(floor(runif(12,0,4)), nrow=4, ncol=4)
> mat
     [,1] [,2] [,3] [,4]
[1,]    3    3    3    3
[2,]    0    0    1    0
[3,]    3    1    0    3
[4,]    1    1    1    1
>
> i <- 1
> while(i <= length(mat)){
+     if(mat[i] > 0){
+         mat[i] <- 1
+     }
+     i <- i+1
+ }

> mat
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    0    0    1    0
[3,]    1    1    0    1
[4,]    1    1    1    1
```

Boucles explicites

Pas très important

repeat expr

On répète l'expression tant qu'un "break" n'apparaît pas

```
> mat <- matrix(floor(runif(12,0,3)), nrow=4, ncol=4)
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    0    2    1
[2,]    1    1    2    1
[3,]    0    0    0    0
[4,]    1    0    0    1
>
> i <- 1
> repeat {
+     if(mat[i] > 0){
+         mat[i] <- 1
+     }
+     i <- i+1
+     if(i > length(mat)) break
+ }

> mat
     [,1] [,2] [,3] [,4]
[1,]    1    0    1    1
[2,]    1    1    1    1
[3,]    0    0    0    0
[4,]    1    0    0    1
```

Boucles explicites

Pas très important

`next`

Pour "sauter un tour" dans une boucle

```
> mat <- matrix(floor(runif(12,0,4)), nrow=4, ncol=4)
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    3    2    1
[2,]    0    3    1    0
[3,]    2    0    2    2
[4,]    1    0    1    1


> for(i in 1:length(mat)){
+      if(mat[i] == 0) next
+      mat[i] <- 1
+ }

> mat
     [,1] [,2] [,3] [,4]
[1,]    1    1    1    1
[2,]    0    1    1    0
[3,]    1    0    1    1
[4,]    1    0    1    1
```

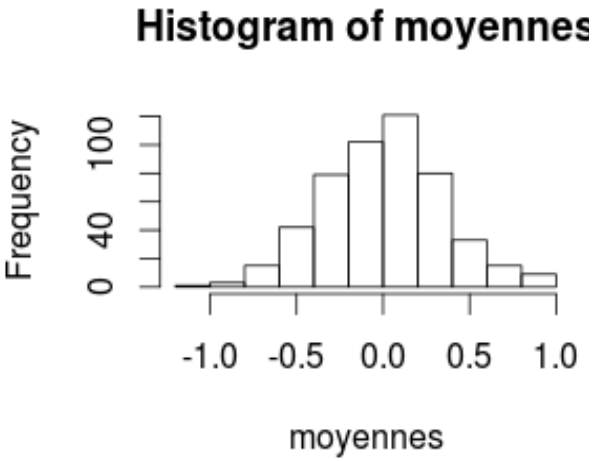
Boucles explicites

`replicate(n, expr)`

Quand on doit répéter une opération à l'identique (sans incrémentation)

Exemple : on prélève un échantillon aléatoire de 10 individus dans une population, on en calcule la moyenne et on recommence 500 fois

```
> pop <- rnorm(10000) # on génère 10000 nombres aléatoires
> n <- 500
>
> moyennes <- replicate(n, mean(sample(pop, 10)))
> hist(moyennes)
```



Avec une boucle for classique on devrait écrire :

```
> moyennes <- vector(length=n) # vecteur vide pour collecter les données

> for(i in 1:n) {
+     moyennes[i] <- mean(sample(pop, 10))
+ }
```

Pourquoi et quand éviter les boucles ?

Les solutions propres à R pour répéter des opérations
(vectorisation, fonctions de la famille apply) sont :

1) toujours plus rapides

2) souvent plus simples à écrire
(mais pas toujours)

Mais les boucles sont plus habituelles et plus faciles à comprendre/lire
--> beaucoup de gens ont tendance à faire des boucles

Pourquoi et quand éviter les boucles ?

Exemple : 4 méthodes pour estimer la racine carrée de n nombres

de la plus compliquée et la plus lente à la plus simple et la plus rapide

```
> n <- 10
> val <- 1:n
```

1) boucle sauvée dans un objet dont la taille n'est pas prédéfinie

```
> a <- NULL
> for (i in 1:n) { a <- c(a, sqrt(val[i]))}
>
```

la taille de l'objet de stockage a est augmentée à chaque itération par concaténation entre l'objet a et la nouvelle valeur

2) boucle sauvée dans un objet dont la taille est prédéfinie

```
> a <- vector(length=n)
> for (i in 1:n) { a[i] <- sqrt(val[i])}
```

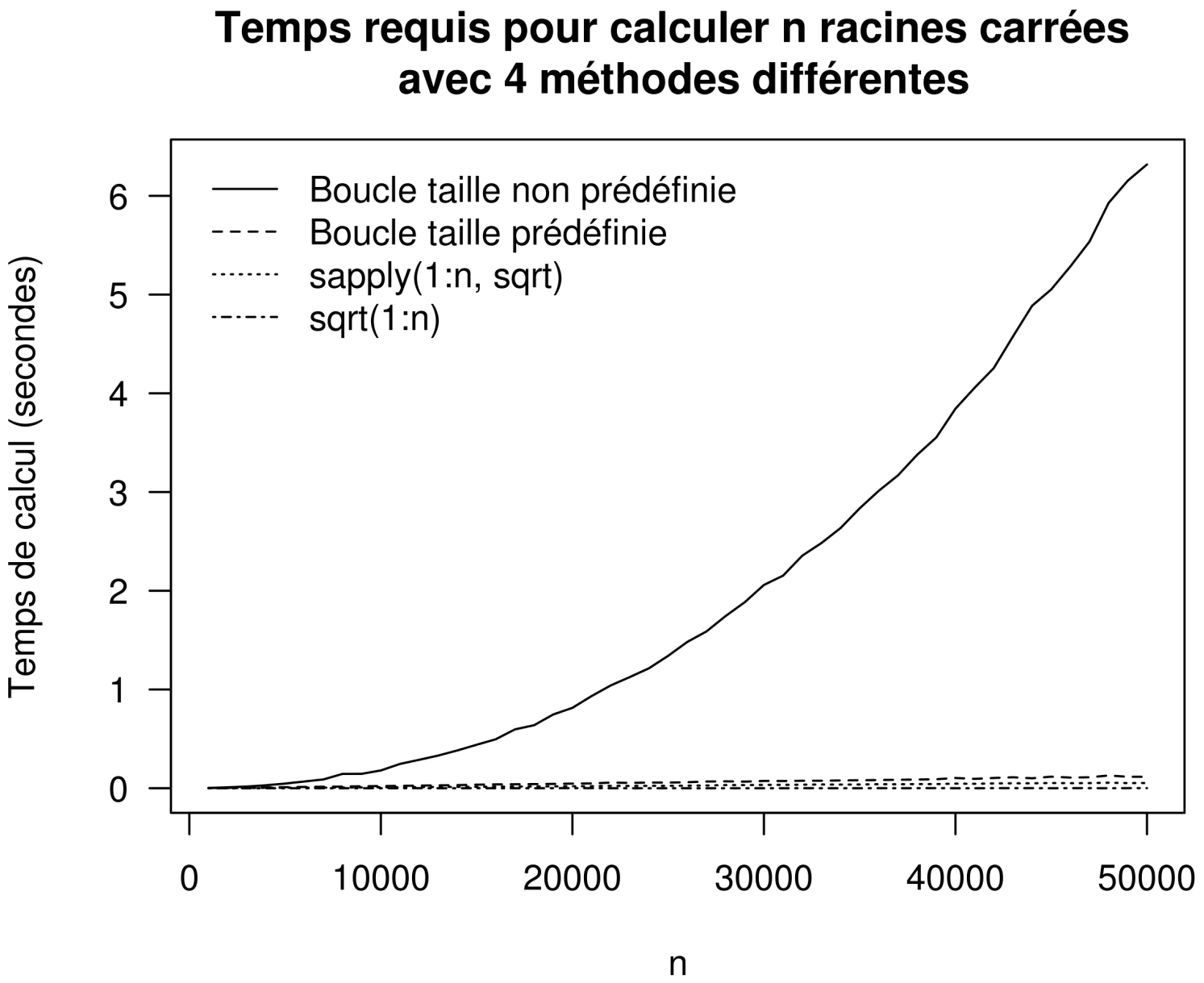
3) utilisation d'une fonction qui applique sqrt à chaque élément du vecteur

```
> sapply(val, sqrt)
```

4) utilisation de la vectorisation implémentée dans R

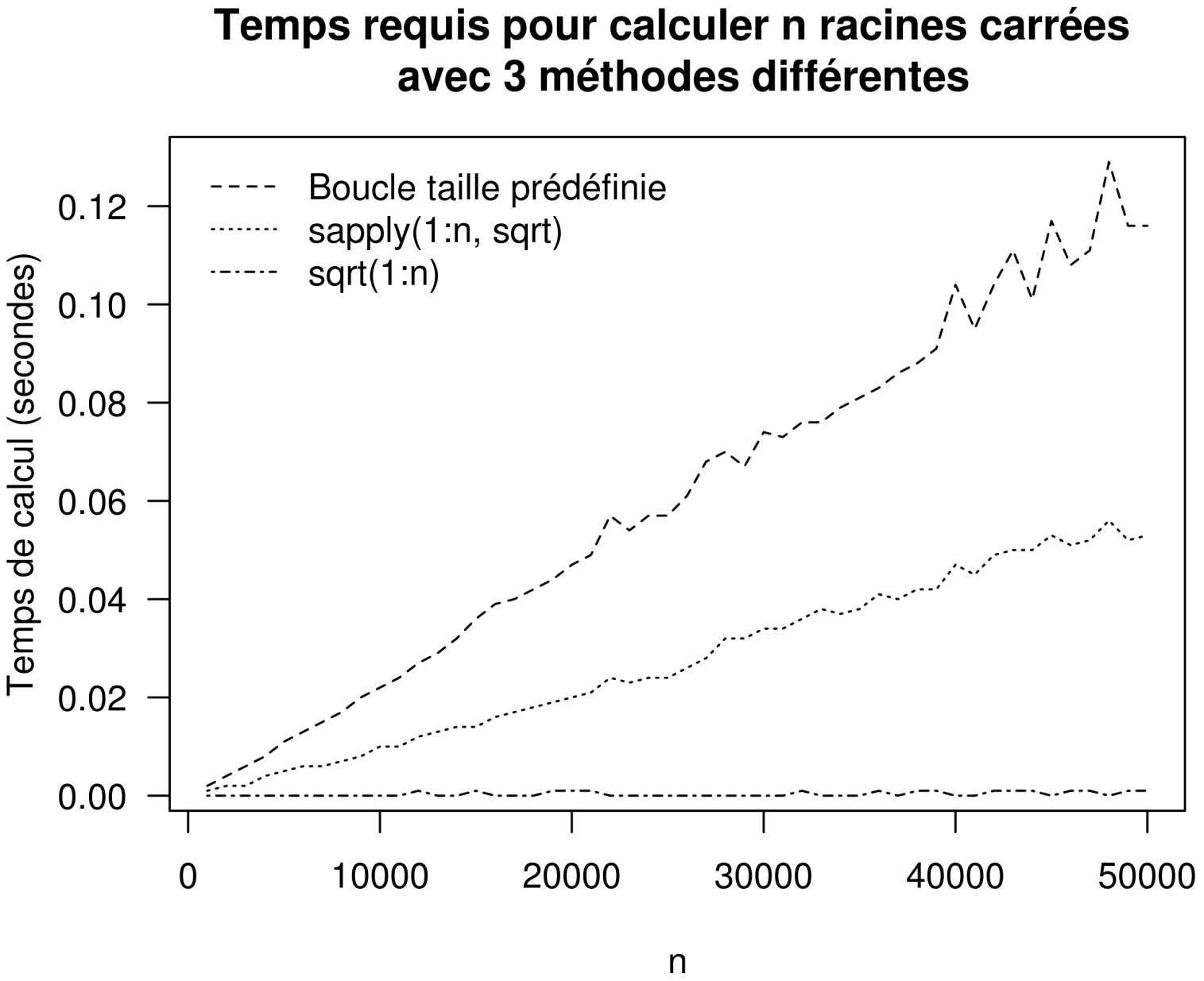
```
> sqrt(val)
```

Pourquoi et quand éviter les boucles ?



Note : la fonction `system.time()` permet d'estimer le temps d'exécution d'une opération

Pourquoi et quand éviter les boucles ?



Pourquoi et quand éviter les boucles ?

Conclusions :

1) si on fait une boucle : toujours prédéfinir la taille de l'objet collectant les résultats.

Si on ne connaît pas cette taille à l'avance : créer un objet trop grand et éliminer les dimensions superflues a posteriori

2) si la vitesse d'exécution est vraiment importante :
ne pas faire de boucle

3) si la vectorisation est disponible :
privilégiez cette solution toujours plus simple et plus rapide

4) sinon choisir l'option qui est la plus facile à coder

Exercices :
boucles, fonctions etc

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de dates et caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

reshaping

reshaping = changer la forme d'un jeu de données
sans le modifier
(réorganiser les lignes, colonnes etc..)

Création d'un jeu de données exemple :

```
> env <- data.frame(
+     management = rep(c("man1", "man2", "man3"), each=6),
+     site = rep(rep(c("site1", "site2", "site3"), each=2), 3),
+     date = rep(c("date1", "date2"), 9)
+ )

> spe <- data.frame(
+     sp1 = c(rpois(6, 20), rpois(6, 10), rpois(6, 2)),
+     sp2 = c(rpois(6, 5), rpois(6, 2), rpois(6, 0)),
+     sp3 = c(rpois(6, 2), rpois(6, 1), rpois(6, 0)),
+     sp4 = rpois(18, 5),
+     sp5 = rpois(18, 2)
+ )

> d <- cbind(env, spe)
```

reshaping

Création d'un jeu de données exemple :

```
> d <- cbind(env, spe)
```

	management	site	date	sp1	sp2	sp3	sp4	sp5
1	man1	site1	date1	17	1	1	8	1
2	man1	site1	date2	25	5	2	4	2
3	man1	site2	date1	12	6	1	6	1
4	man1	site2	date2	20	3	4	2	0
5	man1	site3	date1	27	4	0	4	1
6	man1	site3	date2	22	3	2	4	2
7	man2	site1	date1	5	1	2	7	2
8	man2	site1	date2	4	2	0	5	3
9	man2	site2	date1	13	2	1	7	0
10	man2	site2	date2	11	1	0	7	2
11	man2	site3	date1	11	1	0	7	6
12	man2	site3	date2	10	1	2	4	4
13	man3	site1	date1	1	0	0	6	4
14	man3	site1	date2	1	0	0	6	1
15	man3	site2	date1	5	0	0	6	0
16	man3	site2	date2	4	0	0	0	2
17	man3	site3	date1	3	0	0	5	1
18	man3	site3	date2	3	0	0	3	2

env | spe

reshaping

`t()` : transposition = retourner un jeu de données

```
> spe
   sp1 sp2 sp3 sp4 sp5
1   17   1   1   8   1
2   25   5   2   4   2
3   12   6   1   6   1
4   20   3   4   2   0
5   27   4   0   4   1
6   22   3   2   4   2
(...)

> t(spe)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
sp1   17   25   12   20   27   22    5    4   13    11    11    10     1
sp2    1    5    6    3    4    3    1    2    2     1     1     1     0
sp3    1    2    1    4    0    2    2    0    1     0     0     2     0
sp4    8    4    6    2    4    4    7    5    7     7     7     4     6
sp5    1    2    1    0    1    2    2    3    0     2     6     4     4
     [,14] [,15] [,16] [,17] [,18]
sp1      1      5      4      3      3
sp2      0      0      0      0      0
sp3      0      0      0      0      0
sp4      6      6      0      5      3
sp5      1      0      2      1      2
sp5      4      4      1      0      2
```

reshaping

`stack()` et `unstack()` : passer du format "large" au format "long"

```
> spe[1:5,]
  sp1 sp2 sp3 sp4 sp5
1  17   1   1   8   1
2  25   5   2   4   2
3  12   6   1   6   1
4  20   3   4   2   0
5  27   4   0   4   1

> spe2 <- stack(spe[1:5,])
> spe2
   values ind
1      17 sp1
2      25 sp1
3      12 sp1
4      20 sp1
5      27 sp1
6       1 sp2
7       5 sp2
8       6 sp2
9       3 sp2
10      4 sp2
11      1 sp3
12      2 sp3
(...)
```

```
> spe2 <- unstack(spe2)
> spe2
  sp1 sp2 sp3 sp4 sp5
1  17   1   1   8   1
2  25   5   2   4   2
3  12   6   1   6   1
4  20   3   4   2   0
5  27   4   0   4   1
```

reshaping

`melt()` (package `reshape`) :

passer du format "large" au format "long" pour une partie du tableau

```
> library(reshape)
> d[1:5,]
  management  site  date sp1 sp2 sp3 sp4 sp5
1       man1 site1 date1  17   1   1   8   1
2       man1 site1 date2  25   5   2   4   2
3       man1 site2 date1  12   6   1   6   1
4       man1 site2 date2  20   3   4   2   0
5       man1 site3 date1  27   4   0   4   1

> melt(d[1:5,], id.vars = 1:3)
> melt(d[1:5,], id.vars = c("management", "site", "date"))
> melt(d[1:5,], measure.vars = 4:8)
   management  site  date variable value
1        man1 site1 date1      sp1    17
2        man1 site1 date2      sp1    25
3        man1 site2 date1      sp1    12
4        man1 site2 date2      sp1    20
5        man1 site3 date1      sp1    27
6        man1 site1 date1      sp2     1
7        man1 site1 date2      sp2     5
8        man1 site2 date1      sp2     6
9        man1 site2 date2      sp2     3
10       man1 site3 date1      sp2     4
11       man1 site1 date1      sp3     1
(...)
```

3 syntaxes donnant le même résultat

reshaping

`cast()` (package `reshape`) :
pour réorganiser complètement un tableau déjà au format "long"
NB : permet aussi d'agréger les données (voir plus loin)

```
md <- melt(d, id.var = 1:3, variable_name = "sp")

cast(md, management + site + date ~ sp)
cast(md, management +  sp ~ site + date)
cast(md, management +  sp ~ site | date)
cast(md, management ~ sp  ~ site | date)
cast(md, management ~ sp  | site + date)
```

Tableau "long" sur lequel on va travailler

Différentes syntaxes possibles pour différents résultats

reshaping

```
> md <- melt(d, id.var = 1:3, variable_name = "sp")
> md
   management  site  date  sp value
1        man1 site1 date1 sp1    17
2        man1 site1 date2 sp1    25
3        man1 site2 date1 sp1    12
4        man1 site2 date2 sp1    20
5        man1 site3 date1 sp1    27
6        man1 site3 date2 sp1    22
7        man2 site1 date1 sp1     5
8        man2 site1 date2 sp1     4
9        man2 site2 date1 sp1    13
10       man2 site2 date2 sp1    11
11       man2 site3 date1 sp1    11
12       man2 site3 date2 sp1    10
13       man3 site1 date1 sp1     1
14       man3 site1 date2 sp1     1
15       man3 site2 date1 sp1     5
16       man3 site2 date2 sp1     4
17       man3 site3 date1 sp1     3
18       man3 site3 date2 sp1     3
19       man1 site1 date1 sp2     1
20       man1 site1 date2 sp2     5
21       man1 site2 date1 sp2     6
22       man1 site2 date2 sp2     3
23       man1 site3 date1 sp2     4
(...)
```

Tableau au format "long" sur lequel on va travailler

reshaping

```
> cast(md, management + site + date ~ sp)
```

	management	site	date	sp1	sp2	sp3	sp4	sp5
1	man1	site1	date1	17	1	1	8	1
2	man1	site1	date2	25	5	2	4	2
3	man1	site2	date1	12	6	1	6	1
4	man1	site2	date2	20	3	4	2	0
5	man1	site3	date1	27	4	0	4	1
6	man1	site3	date2	22	3	2	4	2
7	man2	site1	date1	5	1	2	7	2
8	man2	site1	date2	4	2	0	5	3
9	man2	site2	date1	13	2	1	7	0
10	man2	site2	date2	11	1	0	7	2
11	man2	site3	date1	11	1	0	7	6
12	man2	site3	date2	10	1	2	4	4
13	man3	site1	date1	1	0	0	6	4
14	man3	site1	date2	1	0	0	6	1
15	man3	site2	date1	5	0	0	6	0
16	man3	site2	date2	4	0	0	0	2
17	man3	site3	date1	3	0	0	5	1
18	man3	site3	date2	3	0	0	3	2

On retourne au format large d'origine

reshaping

Tableau croisé management, sp x site, date

```
> cast(md, management +  sp ~ site + date)

management  sp site1_date1 site1_date2 site2_date1 site2_date2 site3_date1 site3_date2
      man1 sp1           17           25           12           20           27           22
      man1 sp2            1            5            6            3            4            3
      man1 sp3            1            2            1            4            0            2
      man1 sp4            8            4            6            2            4            4
      man1 sp5            1            2            1            0            1            2
      man2 sp1            5            4           13           11           11           10
      man2 sp2            1            2            2            1            1            1
      man2 sp3            2            0            1            0            0            2
      man2 sp4            7            5            7            7            7            4
      man2 sp5            2            3            0            2            6            4
      man3 sp1            1            1            5            4            3            3
      man3 sp2            0            0            0            0            0            0
      man3 sp3            0            0            0            0            0            0
      man3 sp4            6            6            6            0            5            3
      man3 sp5            4            1            0            2            1            2
```

reshaping

```
> cast(md, management +  sp ~ site | date)
$date1
   management  sp site1 site2 site3
1        man1 sp1    17    12    27
2        man1 sp2     1     6     4
3        man1 sp3     1     1     0
4        man1 sp4     8     6     4
5        man1 sp5     1     1     1
6        man2 sp1     5    13    11
7        man2 sp2     1     2     1
8        man2 sp3     2     1     0
9        man2 sp4     7     7     7
(...)

$date2
   management  sp site1 site2 site3
1        man1 sp1    25    20    22
2        man1 sp2     5     3     3
3        man1 sp3     2     4     2
4        man1 sp4     4     2     4
5        man1 sp5     2     0     2
6        man2 sp1     4    11    10
7        man2 sp2     2     1     1
8        man2 sp3     0     0     2
9        man2 sp4     5     7     4
10       man2 sp5     3     2     4
11       man3 sp1     1     4     3
(...)
```

tableau croisé management, sp x site divisé par date

reshaping

array à 3 dimensions divisé par date

```
> cast(md, management ~ sp  ~ site | date)

$date1
, , site = site1

           sp
management sp1 sp2 sp3 sp4 sp5
      man1  17   1   1   8   1
      man2   5   1   2   7   2
      man3   1   0   0   6   4

, , site = site2

           sp
management sp1 sp2 sp3 sp4 sp5
      man1  12   6   1   6   1
      man2  13   2   1   7   0
      man3   5   0   0   6   0

, , site = site3

           sp
management sp1 sp2 sp3 sp4 sp5
      man1  27   4   0   4   1
      man2  11   1   0   7   6
      man3   3   0   0   5   1

$date2
, , site = site1

           sp
management sp1 sp2 sp3 sp4 sp5
      man1  25   5   2   4   2
      man2   4   2   0   5   3
      man3   1   0   0   6   1

, , site = site2

           sp
management sp1 sp2 sp3 sp4 sp5
      man1  20   3   4   2   0
      man2  11   1   0   7   2
      man3   4   0   0   0   2

, , site = site3

           sp
management sp1 sp2 sp3 sp4 sp5
      man1  22   3   2   4   2
      man2  10   1   2   4   4
      man3   3   0   0   3   2
```

reshaping

```
> cast(md, management ~ sp  | date + site)
```

tableau management x sp
divisé par date et par site

```
$date1

$date1$site1
  management sp1 sp2 sp3 sp4 sp5
1       man1  17   1   1   8   1
2       man2   5   1   2   7   2
3       man3   1   0   0   6   4

$date1$site2
  management sp1 sp2 sp3 sp4 sp5
1       man1  12   6   1   6   1
2       man2  13   2   1   7   0
3       man3   5   0   0   6   0

$date1$site3
  management sp1 sp2 sp3 sp4 sp5
1       man1  27   4   0   4   1
2       man2  11   1   0   7   6
3       man3   3   0   0   5   1
```

```
$date2

$date2$site1
  management sp1 sp2 sp3 sp4 sp5
1       man1  25   5   2   4   2
2       man2   4   2   0   5   3
3       man3   1   0   0   6   1

$date2$site2
  management sp1 sp2 sp3 sp4 sp5
1       man1  20   3   4   2   0
2       man2  11   1   0   7   2
3       man3   4   0   0   0   2

$date2$site3
  management sp1 sp2 sp3 sp4 sp5
1       man1  22   3   2   4   2
2       man2  10   1   2   4   4
3       man3   3   0   0   3   2
```

Fonctions de la famille apply

Ces fonctions sont des boucles implicites.
Elles permettent d'appliquer une fonction à des sous ensemble de données contenues dans un objet.

Elle permettent donc d'"agréger" les données comme en SQL ou dans un tableau croisé et bien plus encore...

Autre manière de voir ces fonctions :
1) diviser les données en sous-groupes
2) Appliquer une fonction sur ces sous-groupes
3) rassembler les résultats dans un objet

Fonctions de la famille apply

Les principales sont :

`apply()` :
applique une fonction aux lignes ou colonnes (ou d'autres dimensions) d'un tableau de données (matrice, data.frame, array)

`lapply(), sapply()` :
appliquent une fonction aux éléments d'un vecteur ou d'une liste

`aggregate(), tapply(), by()` (+ `table(), ftable()`)
divise les données en sous-groupes selon les niveaux de un ou plusieurs facteurs, applique une fonction à ces sous-groupes et retourne un data.frame ou un tableau croisé (classe table) ou une liste

Il y a pas mal de redondance entre ces fonctions et elles divergent en général par la manière dont elles retournent les valeurs et sur le type d'objets sur lesquels elles agissent

apply

On va travailler sur la matrice "spe" :

```
> spe
   sp1 sp2 sp3 sp4 sp5
1   17   1   1   8   1
2   25   5   2   4   2
3   12   6   1   6   1
4   20   3   4   2   0
5   27   4   0   4   1
(...)
```

Calculer la somme pour chaque ligne
(nombre total d'individus par échantillon)

lignes

```
> apply(X = spe, MARGIN = 1, FUN = sum)
 [1] 28 38 26 29 36 33 17 14 23 21 25 21 11  8 11  6  9  8
```

Calculer la médiane pour chaque colonne
(nombre médian d'individus par espèce)

```
> apply(X = spe, MARGIN = 2, FUN = median)
 sp1  sp2  sp3  sp4  sp5
10.5  1.0  0.0  5.5  2.0
```

colonnes

fonction à appliquer sur chaque colonne

apply

NB : si on veut calculer la somme ou la moyenne des colonnes ou lignes, utiliser de préférence `rowSums, colSums, rowMeans, colMeans` qui sont des fonctions vectorisées (donc plus rapides) prévues pour ça...

```
> rowSums(spe)
 [1] 28 38 26 29 36 33 17 14 23 21 25 21 11  8 11  6  9  8

> colMeans(spe)
       sp1         sp2         sp3         sp4         sp5
10.7777778  1.6666667  0.8333333  5.0555556  1.8888889
```

apply

On peut aussi utiliser des fonctions personnelles créées à la volée
Par exemple : calculer pour chaque espèce, le coefficient de variation (écart type divisé par la moyenne)

```
> apply(spe, 2, function(x) {round( sd(x)/mean(x), 2)})

 sp1  sp2  sp3  sp4  sp5
0.78 1.11 1.38 0.40 0.83
```

l'argument x représente dans ce cas-ci une colonne de spe

apply

Si les résultats pour chaque colonne ou ligne est plus long que 1 mais ont tous une même longueur, `apply` essaye de les rassembler dans une matrice, si non `apply` retourne une liste

```
> apply(spe, 2, summary)

          sp1   sp2    sp3   sp4   sp5
Min.     1.00 0.000 0.0000 0.000 0.000
1st Qu.  4.00 0.000 0.0000 4.000 1.000
Median  10.50 1.000 0.0000 5.500 2.000
Mean    10.78 1.667 0.8333 5.056 1.889
3rd Qu. 16.00 2.750 1.7500 6.750 2.000
Max.    27.00 6.000 4.0000 8.000 6.000
```

résultat = matrice

```
> apply(spe, 2, t.test)

$sp1
     One Sample t-test
data:  newX[, i]
t = 5.4582, df = 17, p-value = 4.251e-05
alternative hypothesis: true mean is
 not equal to 0
95 percent confidence interval:
  6.61173 14.94383
sample estimates:
mean of x
 10.77778

$sp2
     One Sample t-test
data:  newX[, i]
t = 3.8282, df = 17, p-value =
0.001346
alternative hypothesis: true mean is
not equal to 0
95 percent confidence interval:
 0.7481273 2.5852060
sample estimates:
mean of x
 1.666667

$sp3
(...)
```

résultat = liste

apply

Grâce à l'argument "..." de `apply` on peut passer des arguments supplémentaires à la fonction spécifiée par FUN

```
> ?quantile
## Default S3 method:
quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE,
         names = TRUE, type = 7, ...)

> apply(spe, 2, quantile)
      sp1  sp2  sp3  sp4 sp5
0%    1.0 0.00 0.00 0.00   0
25%   4.0 0.00 0.00 4.00   1
50%  10.5 1.00 0.00 5.50   2
75%  16.0 2.75 1.75 6.75   2
100% 27.0 6.00 4.00 8.00   6

> apply(spe, 2, quantile, probs = c(0, 0.025, 0.25, 0.5, 0.75, 0.975, 1))
        sp1   sp2  sp3   sp4  sp5
0%     1.00 0.000 0.00 0.000 0.00
2.5%   1.00 0.000 0.00 0.850 0.00
25%    4.00 0.000 0.00 4.000 1.00
50%   10.50 1.000 0.00 5.500 2.00
75%   16.00 2.750 1.75 6.750 2.00
97.5% 26.15 5.575 3.15 7.575 5.15
100%  27.00 6.000 4.00 8.000 6.00
```

options par défaut

option supplémentaire utilisée par la fonction `quantile`

lapply - sapply

`lapply` = "list apply" - `sapply` = "simplified apply"
`lapply` retourne toujours une liste.
`sapply` retourne un vecteur ou une matrice quand c'est possible

```
> l <- list(a = c("Adalia", "Coccinella"), b = 1:5, d = matrix(11:16, 2, 3))

> length(l)
[1] 3
> class(l)
[1] "list"
```

Longueur et classe de la liste l

```
> sapply(l, length)
a b d
2 5 6
> sapply(l, class)
          a           b           d
"character"   "integer"    "matrix"
```

Longueur et classe des éléments de la liste l

```
> lapply(l, class)
$a
[1] "character"
$b
[1] "integer"
$d
[1] "matrix"
```

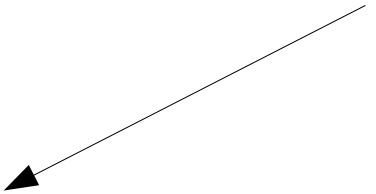
lapply - sapply

Exemple plus réaliste: on effectue un test de t sur chaque espèce, les résultats sont stockés dans une liste
On veut ensuite extraire certaines informations de ces résultats.

NB : du pt de vue statistique cette approche n'est évidemment pas idéale (données non normales à la base)

```
> t.test.results <- apply(spe, 2, t.test)
> str(t.test(spe[,1]))
List of 9
 $ statistic  : Named num 5.46
  ..- attr(*, "names")= chr "t"
 $ parameter  : Named num 17
  ..- attr(*, "names")= chr "df"
 $ p.value    : num 4.25e-05
 $ conf.int   : atomic [1:2] 6.61 14.94
  ..- attr(*, "conf.level")= num 0.95
 $ estimate   : Named num 10.8
  ..- attr(*, "names")= chr "mean of x"
 $ null.value : Named num 0
  ..- attr(*, "names")= chr "mean"
 $ alternative: chr "two.sided"
 $ method     : chr "One Sample t-test"
 $ data.name  : chr "spe[, 1]"
 - attr(*, "class")= chr "htest"
```

Fonction qui extrait les valeurs voulues et les place dans un vecteur



```
> extract.t.test <- function(x){
+     c(x$statistic, x$parameter, p = x$p.value,
+       confint.low = x$conf.int[1], confint.high = x$conf.int[2])
+ }
```

lapply - sapply

```
> extract.t.test <- function(x){
+     c(x$statistic, x$parameter, p = x$p.value,
+       confint.low = x$conf.int[1], confint.high = x$conf.int[2])
+ }
> lapply(t.test.results, extract.t.test)
$sp1
           t           df            p  confint.low confint.high
5.458201e+00 1.700000e+01 4.250734e-05 6.611730e+00 1.494383e+01

$sp2
           t           df            p  confint.low confint.high
 3.828207467 17.000000000  0.001345858  0.748127291  2.585206042

$sp3
           t           df            p  confint.low confint.high
 3.073181486 17.000000000  0.006889031  0.261229259  1.405437407

$sp4
           t           df            p  confint.low confint.high
1.049954e+01 1.700000e+01 7.542480e-09 4.039674e+00 6.071437e+00

$sp5
           t           df            p  confint.low confint.high
5.112043e+00 1.700000e+01 8.673724e-05 1.109316e+00 2.668461e+00

> sapply(t.test.results, extract.t.test) # identique mais résultats "simplifiés" si possible
                    sp1          sp2          sp3          sp4          sp5
t            5.458201e+00  3.828207467  3.073181486 1.049954e+01 5.112043e+00
df           1.700000e+01 17.000000000 17.000000000 1.700000e+01 1.700000e+01
p            4.250734e-05  0.001345858  0.006889031 7.542480e-09 8.673724e-05
confint.low  6.611730e+00  0.748127291  0.261229259 4.039674e+00 1.109316e+00
confint.high 1.494383e+01  2.585206042  1.405437407 6.071437e+00 2.668461e+00
+ }
```

sapply : résultats identiques mais "simplifiés" si possible (en terme de structure)

lapply - sapply

NB : lorsqu'on veut coller ou concaténer des résultats se trouvant dans les éléments d'une liste, la fonction `do.call` est souvent très utile

Par exemple :

```
> a <- lapply(t.test.results, extract.t.test)

> rbind(a[[1]], a[[2]], a[[3]], a[[4]], a[[5]])
             t df            p confint.low confint.high
[1,]  5.458201 17 4.250734e-05   6.6117302    14.943825
[2,]  3.828207 17 1.345858e-03   0.7481273     2.585206
[3,]  3.073181 17 6.889031e-03   0.2612293     1.405437
[4,] 10.499543 17 7.542480e-09   4.0396742     6.071437
[5,]  5.112043 17 8.673724e-05   1.1093165     2.668461
```

Fastidieux et impossible sur des jeux de données plus gros

```
> do.call(rbind, a)
            t df            p confint.low confint.high
sp1  5.458201 17 4.250734e-05   6.6117302    14.943825
sp2  3.828207 17 1.345858e-03   0.7481273     2.585206
sp3  3.073181 17 6.889031e-03   0.2612293     1.405437
sp4 10.499543 17 7.542480e-09   4.0396742     6.071437
sp5  5.112043 17 8.673724e-05   1.1093165     2.668461
>
```

Équivalent mais plus simple

On aurait d'ailleurs pu écrire :

```
> sapply(t.test.results, function(x){do.call(c,x[1:4])})
                      sp1          sp2          sp3          sp4          sp5
statistic.t  5.458201e+00  3.828207467  3.073181486 1.049954e+01 5.112043e+00
parameter.df 1.700000e+01 17.000000000 17.000000000 1.700000e+01 1.700000e+01
p.value      4.250734e-05  0.001345858  0.006889031 7.542480e-09 8.673724e-05
conf.int1    6.611730e+00  0.748127291  0.261229259 4.039674e+00 1.109316e+00
conf.int2    1.494383e+01  2.585206042  1.405437407 6.071437e+00 2.668461e+00
```

aggregate

Divise un ou plusieurs vecteurs selon les nivaux de un ou plusieurs facteurs, applique une fonction à ces sous-ensembles et retourne un data.frame
Équivalent du GROUP BY en SQL

```
> d <- cbind(env, spe)
> summary(d, digits=1)
 management    site       date         sp1            sp2          sp3            sp4
 man1:6     site1:6   date1:9   Min.   : 1   Min.   :0   Min.   :0.0   Min.   :0
 man2:6     site2:6   date2:9   1st Qu.: 4   1st Qu.:0   1st Qu.:0.0   1st Qu.:4
 man3:6     site3:6             Median :10   Median :1   Median :0.0   Median :6
                                Mean   :11   Mean   :2   Mean   :0.8   Mean   :5
                                3rd Qu.:16   3rd Qu.:3   3rd Qu.:1.8   3rd Qu.:7
                                Max.   :27   Max.   :6   Max.   :4.0   Max.   :8
```

```
> aggregate(x = d[,4:8], by = d[, c("management", "site")], FUN = mean)
```

Jeu de données divisé selon les facteurs management et site présentés obligatoirement sous forme de liste ou de data.frame

```
  management  site  sp1 sp2 sp3 sp4 sp5
1       man1 site1 21.0 3.0 1.5 6.0 1.5
2       man2 site1  4.5 1.5 1.0 6.0 2.5
3       man3 site1  1.0 0.0 0.0 6.0 2.5
4       man1 site2 16.0 4.5 2.5 4.0 0.5
5       man2 site2 12.0 1.5 0.5 7.0 1.0
6       man3 site2  4.5 0.0 0.0 3.0 1.0
7       man1 site3 24.5 3.5 1.0 4.0 1.5
8       man2 site3 10.5 1.0 1.0 5.5 5.0
9       man3 site3  3.0 0.0 0.0 4.0 1.5
```

Moyenne par site te par type de management

aggregate

by = liste ou data.frame !

```
> class(d[, c("management", "site")])
[1] "data.frame"
> class(d[, c("management")]) # idem avec class(d$management)
[1] "factor"
> class(d["management"])
[1] "data.frame"

> aggregate(x = d[,4:8], by = d$management, FUN = mean)
Error in aggregate.data.frame(x = d[, 4:8], by = d$management, FUN = mean) :
  'by' doit être une liste

> aggregate(x = d[,4:8], by = list(d$management), FUN = mean)
  Group.1       sp1      sp2       sp3      sp4      sp5
1    man1 20.500000 3.666667 1.6666667 4.666667 1.166667
2    man2  9.000000 1.333333 0.8333333 6.166667 2.833333
3    man3  2.833333 0.000000 0.0000000 4.333333 1.666667

> aggregate(x = d[,4:8], by = list(management = d$management), FUN = mean)
  management       sp1      sp2       sp3      sp4      sp5
1       man1 20.500000 3.666667 1.6666667 4.666667 1.166667
2       man2  9.000000 1.333333 0.8333333 6.166667 2.833333
3       man3  2.833333 0.000000 0.0000000 4.333333 1.666667

> aggregate(x = d[,4:8], by = d["management"], FUN = mean)
  management       sp1      sp2       sp3      sp4      sp5
1       man1 20.500000 3.666667 1.6666667 4.666667 1.166667
2       man2  9.000000 1.333333 0.8333333 6.166667 2.833333
3       man3  2.833333 0.000000 0.0000000 4.333333 1.666667
```

On perd le nom de colonne !

OK mais long et compliqué !

Simples crochets, pas de virgule !
Plus simple !

aggregate

Autre syntaxe : formule

```
> aggregate(cbind(sp1, sp2, sp3, sp4, sp5) ~ management + site, data=d, mean)
  management  site  sp1 sp2 sp3 sp4 sp5
1       man1 site1 21.0 3.0 1.5 6.0 1.5
2       man2 site1  4.5 1.5 1.0 6.0 2.5
3       man3 site1  1.0 0.0 0.0 6.0 2.5
4       man1 site2 16.0 4.5 2.5 4.0 0.5
5       man2 site2 12.0 1.5 0.5 7.0 1.0
6       man3 site2  4.5 0.0 0.0 3.0 1.0
7       man1 site3 24.5 3.5 1.0 4.0 1.5
8       man2 site3 10.5 1.0 1.0 5.5 5.0
9       man3 site3  3.0 0.0 0.0 4.0 1.5
```

doit être une matrice -->
`d[,4:8]` ne fonctionne pas
`as.matrix(d[,4:8])` fonctionne mais on perd les noms de colonne

```
> aggregate(. ~ management + site, data=d, mean)
  management  site date  sp1 sp2 sp3 sp4 sp5
1       man1 site1  1.5 21.0 3.0 1.5 6.0 1.5
2       man2 site1  1.5  4.5 1.5 1.0 6.0 2.5
3       man3 site1  1.5  1.0 0.0 0.0 6.0 2.5
4       man1 site2  1.5 16.0 4.5 2.5 4.0 0.5
5       man2 site2  1.5 12.0 1.5 0.5 7.0 1.0
6       man3 site2  1.5  4.5 0.0 0.0 3.0 1.0
7       man1 site3  1.5 24.5 3.5 1.0 4.0 1.5
8       man2 site3  1.5 10.5 1.0 1.0 5.5 5.0
9       man3 site3  1.5  3.0 0.0 0.0 4.0 1.5
```

le point représente toutes les colonnes restantes du data.frame
Il peut s'utiliser aussi à droite de la formule

aggregate

Attention : la gestion des NA n'est pas identique entre les deux syntaxes !!
La syntaxe par formule peut être très dangereuse !!

```
> d2 <- d
> d2[1:10, "sp1"] <- NA

> aggregate(x = d2[,4:5], by = d2[, c("management", "site")], FUN = mean)
  management  site  sp1 sp2
1       man1 site1   NA 3.0
2       man2 site1   NA 1.5
3       man3 site1  1.0 0.0
4       man1 site2   NA 4.5
5       man2 site2   NA 1.5
6       man3 site2  4.5 0.0
7       man1 site3   NA 3.5
8       man2 site3 10.5 1.0
9       man3 site3  3.0 0.0


> aggregate(cbind(sp1, sp2) ~ management + site, data=d2, mean)
  management  site  sp1 sp2
1       man3 site1  1.0   0
2       man3 site2  4.5   0
3       man2 site3 10.5   1
4       man3 site3  3.0   0
```

Les lignes avec des NA ont été éliminées sans message d'avertissement !!

Utiliser le code suivant pour garder les NA

```
> aggregate(cbind(sp1, sp2) ~ management + site, data=d2, mean,
                    na.action = na.pass)
```

Alternative à aggregate : sqldf

```
> library(sqldf)

> sqldf("SELECT management, site, AVG(sp1) AS sp1 , AVG(sp2) AS sp2, AVG(sp3),
AVG(sp4), AVG(sp5)
+ FROM d GROUP BY management, site")

  management  site  sp1 sp2 AVG(sp3) AVG(sp4) AVG(sp5)
1       man1 site1 21.0 3.0      1.5      6.0      1.5
2       man1 site2 16.0 4.5      2.5      4.0      0.5
3       man1 site3 24.5 3.5      1.0      4.0      1.5
4       man2 site1  4.5 1.5      1.0      6.0      2.5
5       man2 site2 12.0 1.5      0.5      7.0      1.0
6       man2 site3 10.5 1.0      1.0      5.5      5.0
7       man3 site1  1.0 0.0      0.0      6.0      2.5
8       man3 site2  4.5 0.0      0.0      3.0      1.0
9       man3 site3  3.0 0.0      0.0      4.0      1.5
```

Alternative à aggregate : cast

`cast()` (package `reshape`) permet à la fois d'agréger les données et de les mettre en forme différemment (reshaping).
Nombreuses possibilités --> vaut vraiment le coup d'être étudiée en détail
Contrainte : doit travailler sur des données en format "long"

```
> library(reshape)
> md <- melt(d, id = 1:3, variable_name="sp")
> md
   management  site  date  sp value
1        man1 site1 date1 sp1    17
2        man1 site1 date2 sp1    25
3        man1 site2 date1 sp1    12
4        man1 site2 date2 sp1    20
5        man1 site3 date1 sp1    27
6        man1 site3 date2 sp1    22
7        man2 site1 date1 sp1     5
8        man2 site1 date2 sp1     4
9        man2 site2 date1 sp1    13
10       man2 site2 date2 sp1    11
11       man2 site3 date1 sp1    11
(...)
19       man1 site1 date1 sp2     1
20       man1 site1 date2 sp2     5
21       man1 site2 date1 sp2     6
22       man1 site2 date2 sp2     3
23       man1 site3 date1 sp2     4
(...)
```

NB : il existe une version plus récente du package appelée `reshape2`
Le gros avantage est la rapidité qui a été fortement améliorée
--> à privilégier sur des gros jeux de données

`melt` s'utilise exactement de la même façon
`cast` devient `dcast` quand on travaille sur des data.frame. La syntaxe est la même mais on perd la possibilité de subdiviser en liste avec le pipe |

Alternative à aggregate : cast

Moyenne des dates - résultat identique à :

by

```
> aggregate(cbind(sp1, sp2, sp3, sp4, sp5) ~ management + site, data=d, mean)
```

lignes colonnes

```
> cast(md, management + site ~ sp, fun.aggregate = mean)
```

```
  management  site  sp1 sp2 sp3 sp4 sp5
1       man1 site1 21.0 3.0 1.5 6.0 1.5
2       man1 site2 16.0 4.5 2.5 4.0 0.5
3       man1 site3 24.5 3.5 1.0 4.0 1.5
4       man2 site1  4.5 1.5 1.0 6.0 2.5
5       man2 site2 12.0 1.5 0.5 7.0 1.0
6       man2 site3 10.5 1.0 1.0 5.5 5.0
7       man3 site1  1.0 0.0 0.0 6.0 2.5
8       man3 site2  4.5 0.0 0.0 3.0 1.0
9       man3 site3  3.0 0.0 0.0 4.0 1.5
```

Alternative à aggregate : cast

Toutes les possibilités vues précédemment restent valables :

```
> cast(md, site ~ sp | management, fun.aggregate = mean)

$man1
   site  sp1 sp2 sp3 sp4 sp5
1 site1 21.0 3.0 1.5   6 1.5
2 site2 16.0 4.5 2.5   4 0.5
3 site3 24.5 3.5 1.0   4 1.5

$man2
   site  sp1 sp2 sp3 sp4 sp5
1 site1  4.5 1.5 1.0 6.0 2.5
2 site2 12.0 1.5 0.5 7.0 1.0
3 site3 10.5 1.0 1.0 5.5 5.0

$man3
   site sp1 sp2 sp3 sp4 sp5
1 site1 1.0   0   0   6 2.5
2 site2 4.5   0   0   3 1.0
3 site3 3.0   0   0   4 1.5
```

Alternative à aggregate : cast

Il est possible d'ajouter des totaux et sous-totaux :

```
> cast(md, management + site ~ sp, fun.aggregate = mean, margins = TRUE)
   management  site        sp1      sp2       sp3      sp4      sp5    (all)
1        man1 site1 21.000000 3.000000 1.5000000 6.000000 1.500000 6.600000
2        man1 site2 16.000000 4.500000 2.5000000 4.000000 0.500000 5.500000
3        man1 site3 24.500000 3.500000 1.0000000 4.000000 1.500000 6.900000
4        man1 (all) 20.500000 3.666667 1.6666667 4.666667 1.166667 6.333333
5        man2 site1  4.500000 1.500000 1.0000000 6.000000 2.500000 3.100000
6        man2 site2 12.000000 1.500000 0.5000000 7.000000 1.000000 4.400000
7        man2 site3 10.500000 1.000000 1.0000000 5.500000 5.000000 4.600000
8        man2 (all)  9.000000 1.333333 0.8333333 6.166667 2.833333 4.033333
9        man3 site1  1.000000 0.000000 0.0000000 6.000000 2.500000 1.900000
10       man3 site2  4.500000 0.000000 0.0000000 3.000000 1.000000 1.700000
11       man3 site3  3.000000 0.000000 0.0000000 4.000000 1.500000 1.700000
12       man3 (all)  2.833333 0.000000 0.0000000 4.333333 1.666667 1.766667
13      (all) (all) 10.777778 1.666667 0.8333333 5.055556 1.888889 4.044444
> cast(md, management + site ~ sp, fun.aggregate = mean, margins = c("grand_col",
"grand_row"))
   management  site       sp1      sp2       sp3      sp4      sp5    (all)
1        man1 site1 21.00000 3.000000 1.5000000 6.000000 1.500000 6.600000
2        man1 site2 16.00000 4.500000 2.5000000 4.000000 0.500000 5.500000
3        man1 site3 24.50000 3.500000 1.0000000 4.000000 1.500000 6.900000
4        man2 site1  4.50000 1.500000 1.0000000 6.000000 2.500000 3.100000
5        man2 site2 12.00000 1.500000 0.5000000 7.000000 1.000000 4.400000
6        man2 site3 10.50000 1.000000 1.0000000 5.500000 5.000000 4.600000
7        man3 site1  1.00000 0.000000 0.0000000 6.000000 2.500000 1.900000
8        man3 site2  4.50000 0.000000 0.0000000 3.000000 1.000000 1.700000
9        man3 site3  3.00000 0.000000 0.0000000 4.000000 1.500000 1.700000
10      (all) (all) 10.77778 1.666667 0.8333333 5.055556 1.888889 4.044444
```

Alternative à aggregate : cast

Exemples avec une jeu de données moins "carré" (non balancé)

```
> set.seed(1234)

> obs <- data.frame(
+     site = sample(c("site1", "site2", "site3", "site4", "site5"), 20, replace = TRUE),
+     sp = sample(c("sp1", "sp2", "sp3"), 20, replace = TRUE),
+     abund = rpois(20, 10),
+     date = paste("2012", sample(c("0510", "0612", "0711"),20, replace = TRUE), sep="")
+ )

> summary(obs)
    site     sp         abund             date
 site1:3   sp1:11   Min.   : 5.00   20120510:14
 site2:6   sp2: 3   1st Qu.: 8.00   20120612: 2
 site3:2   sp3: 6   Median :10.00   20120711: 4
 site4:6            Mean   :10.60
 site5:3            3rd Qu.:13.25
                    Max.   :20.00
```

Alternative à aggregate : cast

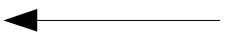
NB si aucune colonne appelée "value" existe, `cast` essaye d'utiliser la dernière colonne à la place
Utiliser l'argument "value" pour spécifier avec quelle colonne il doit travailler

```
> cast(obs, site + date ~ sp, fun.aggregate= length)
Using date as value column.  Use the value argument to cast to override this choice
Erreur dans `[.data.frame`(data, , variables, drop = FALSE) :
  undefined columns selected

> cast(obs, site + date ~ sp, fun.aggregate= length, value="abund")
   site     date sp1 sp2 sp3
1 site1 20120510   1   1   1
2 site2 20120510   1   0   1
3 site2 20120612   1   0   0
4 site2 20120711   2   0   1
5 site3 20120510   2   0   0
6 site4 20120510   2   1   1
7 site4 20120612   0   0   1
8 site4 20120711   1   0   0
9 site5 20120510   1   1   1
```

Alternative à aggregate : cast

On peut lui demander d'ajouter les combinaisons pour lesquelles on a aucune observations.
Par défaut il ajoute comme valeur la fonction choisie appliquée à un vecteur de longueur 0

```
> cast(obs, site + date ~ sp, fun.aggregate= length, value = "abund", add.missing =
TRUE)
    site     date sp1 sp2 sp3
1  site1 20120510   1   1   1
2  site1 20120612   0   0   0
3  site1 20120711   0   0   0
4  site2 20120510   1   0   1
5  site2 20120612   1   0   0
6  site2 20120711   2   0   1
7  site3 20120510   2   0   0
8  site3 20120612   0   0   0
9  site3 20120711   0   0   0
10 site4 20120510   2   1   1
11 site4 20120612   0   0   1
12 site4 20120711   1   0   0
13 site5 20120510   1   1   1
14 site5 20120612   0   0   0
15 site5 20120711   0   0   0
```



Combinaisons sans observations

Alternative à aggregate : cast

On peut lui spécifier une autre valeur par défaut

```
> cast(obs, site + date ~ sp, fun.aggregate= length, value = "abund", add.missing =
TRUE, fill = NA)

    site     date sp1 sp2 sp3
1  site1 20120510   1   1   1
2  site1 20120612  NA  NA  NA
3  site1 20120711  NA  NA  NA
4  site2 20120510   1  NA   1
5  site2 20120612   1  NA  NA
6  site2 20120711   2  NA   1
7  site3 20120510   2  NA  NA
8  site3 20120612  NA  NA  NA
9  site3 20120711  NA  NA  NA
10 site4 20120510   2   1   1
11 site4 20120612  NA  NA   1
12 site4 20120711   1  NA  NA
13 site5 20120510   1   1   1
14 site5 20120612  NA  NA  NA
15 site5 20120711  NA  NA  NA
```

table - tapply

`table` permet de construire des tables de contingences où on compte le nombre d'observations d'une combinaison de facteurs

```
# jeu de données exemple :
> obs
    site  sp abund     date
1  site1 sp1    10 20120510
2  site4 sp1     8 20120510
3  site4 sp1     8 20120510
4  site4 sp1    11 20120711
5  site5 sp1     7 20120510
6  site4 sp3     8 20120612
7  site1 sp2    10 20120510
8  site2 sp3    10 20120510
(...)
```

table - tapply

Mêmes résultats avec cast

```
> table(obs$site, obs$sp)

       sp1 sp2 sp3
  site1   1   1   1
  site2   4   0   2
  site3   2   0   0
  site4   3   1   2
  site5   1   1   1

> cast(obs, site ~ sp, fun.aggregate = length)
Using date as value column.  Use the value argument to cast to override this choice

   site sp1 sp2 sp3
1 site1   1   1   1
2 site2   4   0   2
3 site3   2   0   0
4 site4   3   1   2
5 site5   1   1   1


> table(obs$site, obs$date, obs$sp)
(...)
> cast(obs, site ~ date ~ sp, fun.aggregate = length, value = "abund")
(...)
```

table - tapply

`ftable` pour "flat tables" : formatage différent

```
> ftable(obs$site, obs$date, obs$sp)
                sp1 sp2 sp3

site1 20120510    1   1   1
      20120612    0   0   0
      20120711    0   0   0
site2 20120510    1   0   1
      20120612    1   0   0
      20120711    2   0   1
site3 20120510    2   0   0
      20120612    0   0   0
      20120711    0   0   0
site4 20120510    2   1   1
      20120612    0   0   1
      20120711    1   0   0
site5 20120510    1   1   1
      20120612    0   0   0
      20120711    0   0   0
```

table - tapply

`tapply` fait la même chose (tableaux croisés) sauf qu'on peut utiliser n'importe quelle fonction et l'appliquer sur un vecteur

```
> tapply(X= obs$abund, INDEX = list(obs$site, obs$sp), FUN=sum)
      sp1 sp2 sp3
site1  10  10  14
site2  32  NA  16
site3  28  NA  NA
site4  27  20  20
site5   7  14  14

> tapply(X= obs$abund, INDEX = list(obs$site, obs$date, obs$sp), FUN=mean)
, , sp1
      20120510 20120612 20120711
site1       10       NA       NA
site2       13        5        7
site3       14       NA       NA
site4        8       NA       11
site5        7       NA       NA

, , sp2
      20120510 20120612 20120711
site1       10       NA       NA
site2       NA       NA       NA
site3       NA       NA       NA
site4       20       NA       NA
site5       14       NA       NA
(...)
```

table - tapply

Comparaison avec aggregate :
aggregate ne fait pas de tableaux croisés et son argument x peut être un tableau (matrix, data.frame) alors que tapply ne peut travailler que sur un vecteur (données en format "long")

```
> tapply(X= obs$abund, INDEX = list(obs$site, obs$sp), FUN=sum)
      sp1 sp2 sp3
site1  10  10  14
site2  32  NA  16
site3  28  NA  NA
site4  27  20  20
site5   7  14  14

> aggregate(x= obs$abund, by = list(site=obs$site, sp=obs$sp), FUN=sum)
    site  sp  x
1  site1 sp1 10
2  site2 sp1 32
3  site3 sp1 28
4  site4 sp1 27
5  site5 sp1  7
6  site1 sp2 10
7  site4 sp2 20
8  site5 sp2 14
9  site1 sp3 14
10 site2 sp3 16
11 site4 sp3 20
12 site5 sp3 14
```

split

`split` permet de diviser un jeu de données selon les niveaux de un ou plusieurs facteurs. Le résultat est une liste.
Souvent utilisé en combinaison avec lapply et sapply.

Par exemple : faire la régression longueur vs largeur de sépale pour chaque espèce

```
> data(iris)
> summary(iris)
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width          Species
 Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
 1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
 Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
 Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
 3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
 Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

split

```
> split(iris, iris$Species)
$setosa
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
2           4.9         3.0          1.4         0.2  setosa
(...)
$versicolor
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
51           7.0         3.2          4.7         1.4 versicolor
52           6.4         3.2          4.5         1.5 versicolor
(...)
$virginica
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
101          6.3         3.3          6.0         2.5 virginica
102          5.8         2.7          5.1         1.9 virginica
(...)


> lapply(split(iris, iris$Species), function(x) lm(Sepal.Length ~ Sepal.Width, data=x))
$setosa
Coefficients:
(Intercept)  Sepal.Width
     2.6390       0.6905

$versicolor
Coefficients:
(Intercept)  Sepal.Width
     3.5397       0.8651

$virginica
Coefficients:
(Intercept)  Sepal.Width
     3.9068       0.9015
```

by

`aggregate` et `tapply` ne peuvent appliquer une fonction que sur un vecteur à la fois

`by` fait la même chose mais peut travailler sur des data.frame entiers divisés selon un ou plusieurs facteurs

Par exemple : faire la régression longueur vs largeur de sépale pour chaque espèce d'iris

(même exemple que le précédent)

by

```
> res <- by(iris, iris$Species,
+ function(x) lm(Sepal.Length ~ Sepal.Width, data=x))
> res
iris$Species: setosa

Call:
lm(formula = Sepal.Length ~ Sepal.Width, data = x)

Coefficients:
(Intercept)  Sepal.Width
     2.6390       0.6905

------------------------------------------------------------------------------
iris$Species: versicolor

Call:
lm(formula = Sepal.Length ~ Sepal.Width, data = x)

Coefficients:
(Intercept)  Sepal.Width
     3.5397       0.8651

------------------------------------------------------------------------------
iris$Species: virginica

Call:
lm(formula = Sepal.Length ~ Sepal.Width, data = x)

Coefficients:
(Intercept)  Sepal.Width
     3.9068       0.9015
```

by

`by` retourne un objet de classe "`by`" proche d'une liste

```
> sapply(res, coef)
              setosa versicolor virginica
(Intercept) 2.6390012  3.5397347 3.9068365
Sepal.Width 0.6904897  0.8650777 0.9015345
```

```
> res <- by(iris, iris$Species,
+           function(x) coef(lm(Sepal.Length ~ Sepal.Width, data=x)))
> do.call(rbind, res)
           (Intercept) Sepal.Width
setosa        2.639001   0.6904897
versicolor    3.539735   0.8650777
virginica     3.906836   0.9015345
```

`by` fonctionne exactement comme la combinaison de `split` et `lapply`
il retourne juste une liste de classe "by" qui s'imprime de façon différente

by

Parfois un peu difficile de récupérer les noms des facteurs

Ex : Analyse en composante principale pour chaque combinaison de "management" et "date", puis récupération des valeurs propres.

```
> res <- by(d[,4:8], d[, c("management", "date")], princomp)
> eig <- lapply(res, function(x) x$sdev)

> do.call(rbind, eig)
       Comp.1   Comp.2    Comp.3       Comp.4    Comp.5
[1,] 4.060581 2.058867 1.6752283 8.540871e-01 0.7189508
[2,] 2.867186 2.200201 1.3750028 7.425494e-01 0.4542525
[3,] 3.089745 1.178408 0.9923842 0.000000e+00 0.0000000
[4,] 3.625028 1.639126 1.3157466 1.203047e+00 0.5421507
[5,] 2.507144 2.100457 1.5097396 9.402869e-01 0.4887276
[6,] 1.662929 1.096013 0.9712993 4.763187e-09 0.0000000
>
> cbind(expand.grid(dimnames(res)), do.call(rbind, eig))

  management  date   Comp.1   Comp.2    Comp.3       Comp.4    Comp.5
1       man1 date1 4.060581 2.058867 1.6752283 8.540871e-01 0.7189508
2       man2 date1 2.867186 2.200201 1.3750028 7.425494e-01 0.4542525
3       man3 date1 3.089745 1.178408 0.9923842 0.000000e+00 0.0000000
4       man1 date2 3.625028 1.639126 1.3157466 1.203047e+00 0.5421507
5       man2 date2 2.507144 2.100457 1.5097396 9.402869e-01 0.4887276
6       man3 date2 1.662929 1.096013 0.9712993 4.763187e-09 0.0000000
```

unique

`unique` permet d'éliminer les doublons - agrégation de valeurs identiques

```
> obs
    site  sp abund     date
1  site1 sp1    10 20120510
2  site4 sp1     8 20120510
3  site4 sp1     8 20120510
4  site4 sp1    11 20120711
5  site5 sp1     7 20120510
6  site4 sp3     8 20120612
7  site1 sp2    10 20120510
(...)

> unique(obs[,1:2])
    site  sp
1  site1 sp1
2  site4 sp1
5  site5 sp1
6  site4 sp3
7  site1 sp2
8  site2 sp3
10 site3 sp1
11 site4 sp2
13 site2 sp1
14 site5 sp2
16 site5 sp3
19 site1 sp3
```

Résumé : fonctions de la famille apply

Principe général :

1) **Input** : reçoivent un objet d'un certain type en entrée
Argument : `x, X` ou `data` de la fonction

2) **by** : découpent cet objet selon une règle
L'argument correspondant de la fonction peut s'appeler :
`MARGINS, by, INDEX, f, INDICES` selon la fonction

3) **FUN** : appliquent une fonction à chaque élément résultant de la découpe

4) **Output** : recolle les résultats pour et les retourne dans un format préférentiel (si possible)

Résumé : fonctions de la famille apply

Fonctionnement typique :

	Input	by	Output
`apply(X,MARGIN,FUN)`	matrix/ data.frame	ligne (MARGIN=1) ou colonnes (MARGIN=2)	vecteur
`lapply(X,FUN)` `sapply(X,FUN)`	liste	éléments de la liste	liste (toujours) vecteur/matrix
`aggregate(x,by,FUN)`	vecteur(s)	liste de factors	data.frame
`tapply(X,INDEX,FUN)` `(table(x))`	vecteur(s)	liste de factors	matrix/array (table)
`split(x, f) + lapply`	matrix/ data.frame	liste de factors	liste
`by(data,INDICES,FUN)`	matrix/ data.frame	liste de factors	liste de classe by

Pour coller les éléments d'une liste : pex : `do.call(rbind, maliste)`
Pour transformer une table en data.frame :
`as.data.frame` (format long) ou `as.data.frame.matrix` (format large)
La fonction `reshape::cast` peut remplacer avantageusement `aggregate`, `tapply`, etc...

Exercices :
aggregation - reshaping

Interagir avec le système

Nombreuses fonctions pour lister, copier, renommer, effacer,... des fichiers ou des dossiers

```
list.files(path = ".", pattern = NULL, all.files = FALSE,
           full.names = FALSE, recursive = FALSE,
           ignore.case = FALSE, include.dirs = FALSE, no.. = FALSE)

list.dirs(path = ".", full.names = TRUE, recursive = TRUE)

basename(path)
dirname(path)
```

```
file.create(..., showWarnings = TRUE)
file.remove(...)
file.rename(from, to)
file.copy(from, to, overwrite = recursive, recursive = FALSE,
          copy.mode = TRUE, copy.date = FALSE)
```

```
file.info(..., extra_cols = TRUE)
file.size(...)
```

Interagir avec le système

```
> wd <- "/home/gilles/photos/0_rename"
> list.files(wd)
 [1] "20151029_185926_2860.JPG"  "20151029_190043_2862.JPG"
 [3] "20151029_191711_2865.JPG"  "20151030_153812_2876.JPG"
(...)
[51] "20151104_133540_3257.JPG"  "20151104_133755_3271.JPG"
[53] "focus_stacking"            "rename_pictures.R"
[55] "ToSmall"

> # fichiers correspondant à un pattern (photos modifiées ici)
> list.files(wd, pattern = "M\\.JPG$")
[1] "20151031_133132_2981M.JPG" "20151031_133204_2983M.JPG"
[3] "20151031_134441_3036M.JPG" "20151031_134939_3055M.JPG"
[5] "20151104_110806_3179M.JPG" "20151104_110917_3186M.JPG"
>
> # pour chercher aussi dans les sous-dossiers (sous dossier "Focus_stacking" ici)
> list.files(wd, recursive = TRUE)
 [1] "20151029_185926_2860.JPG"
(...)
[53] "focus_stacking/20151001_Dendroctonus_adult_021_001_stack164_opts04_M.JPG"
[54] "focus_stacking/20151001_Dendroctonus_adult_022_002_stack108_opts05_M.JPG"
[55] "focus_stacking/20151020_Contarinia_001_stack80_opts05_M.JPG"
[56] "focus_stacking/20151020_Contarinia_009_stack38_opts05_M.JPG"
[57] "rename_pictures.R"
[58] "ToSmall"
>
```

Interagir avec le système

```
> # Chemins complets
> pr <- list.files(wd, full.names = TRUE)
> pr[1:4]
[1] "/home/gilles/photos/0_rename/20151029_185926_2860.JPG"
[2] "/home/gilles/photos/0_rename/20151029_190043_2862.JPG"
[3] "/home/gilles/photos/0_rename/20151029_191711_2865.JPG"
[4] "/home/gilles/photos/0_rename/20151030_153812_2876.JPG"
>
> # Extraction des différentes parties
> basename(pr)[1:4]
[1] "20151029_185926_2860.JPG" "20151029_190043_2862.JPG"
[3] "20151029_191711_2865.JPG" "20151030_153812_2876.JPG"

> dirname(pr)[1:4]
[1] "/home/gilles/photos/0_rename" "/home/gilles/photos/0_rename"
[3] "/home/gilles/photos/0_rename" "/home/gilles/photos/0_rename"
>
```

Interagir avec le système

Exemple : mesures d'ailes de papillons :

4 points (XY) pour l'aile antérieure (fw)

4 points(XY) pour l'aile postérieure (hw)

Un fichier par individu

On veut calculer la surface des ces polygones pour chaque aile

Le nom du fichier contient la description du traitement, famille, etc...

NB : on ne travaille ici que sur un sous échantillon des fichiers pour gagner de la place sur les slides

```
> setwd("/home/gilles/stats/R_Demo/merge_morphometric_files/completeset")
> f <- list.files(, pattern = "\\.txt$", )
> set.seed(1234)
> f <- sample(f, 30)
> f
 [1] "C190605M.txt" "C250505M.txt" "C250501M.txt" "C250504M.txt" "C280507M.txt"
 [6] "C250507M.txt" "C190104M.txt" "C210203M.txt" "C250602M.txt" "C240207M.txt"
[11] "C250904M.txt" "C250103M.txt" "C210310M.txt" "C280806M.txt" "C210501M.txt"
[16] "C280109M.txt" "W210103M.txt" "C210305M.txt" "C190905M.txt" "C210201M.txt"
[21] "C210505M.txt" "W210101M.txt" "C190805M.txt" "C190302M.txt" "C210105M.txt"
[26] "C270603M.txt" "C240201M.txt" "C280209M.txt" "C270606M.txt" "C190303M.txt"
>
```

Interagir avec le système

Exemple : mesures d'ailes de papillons

```
> # empty list to store the results
> d <- as.list(vector(length = length(f)))
> d <- vector(mode = "list", length = length(f)) # another possibility
>
> # read each file and save its content in a list
> for(i in 1:length(f)) {
+     d[[i]] <- as.matrix(read.table(f[i], header = TRUE, dec = ",", sep = "\t")[1:8,6:7])
+ }

>
> d[1:3]
[[1]]
     X   Y
1 691 499
2 334 564
3 245 341
4 212 104
5 761 344
6 442 652
7 258 460
8 406 140

[[2]]
     X   Y
1 618 486
2 225 521
3 125 320
4 157  74
5 707 309
6 298 536
7 158 308
8 385  37

[[3]]
     X   Y
1 654 490
2 249 531
3 171 321
4 199  69
5 647 320
6 325 614
7 119 413
8 263  90
```

Interagir avec le système

Exemple : mesures d'ailes de papillons

```
> # paste all the coordinates into one dataframe and add information
> # about the measured trait (fore or hind wing), and individual

> d <- do.call(rbind.data.frame, d)
> d$trait <- factor(rep(rep(c("fw", "hw"), each = 4), length(f)))
> f <- rep(f, each = 8)
> d$indiv <- factor(substring(f,1,8))

> summary(d)
       X                Y          trait          indiv
 Min.   : 51.0   Min.   :  4.0   fw:120   C190104M:  8
 1st Qu.:192.0   1st Qu.:226.0   hw:120   C190302M:  8
 Median :282.0   Median :410.0            C190303M:  8
 Mean   :353.8   Mean   :376.4            C190605M:  8
 3rd Qu.:589.2   3rd Qu.:521.2            C190805M:  8
 Max.   :811.0   Max.   :732.0            C190905M:  8
                                          (Other) :192
> d[1:10,]
     X   Y trait    indiv
1   691 499    fw C190605M
2   334 564    fw C190605M
3   245 341    fw C190605M
4   212 104    fw C190605M
5   761 344    hw C190605M
6   442 652    hw C190605M
7   258 460    hw C190605M
8   406 140    hw C190605M
11  618 486    fw C250505M
21  225 521    fw C250505M
```

Interagir avec le système

Exemple : mesures d'ailes de papillons

```
> # Caclulate the surface for each wing and each individual with areapl
> # function from splancs package

> library(splancs)
> res <- by(d[,c("X", "Y")], as.list(d[,c(3:4)]),
+           FUN = function(x) areapl(as.matrix(x)))
>
> res <- as.data.frame(as.table(res))

> # Add descriptive information about each individual present into the file name
> res$type <- factor(substring(res$indiv,1,1))
> res$temperature <- as.numeric(substring(res$indiv,2,3))
> res$family <- factor(paste(substring(res$indiv,1,1),substring(res$indiv,4,5), sep=""))

> res
   trait    indiv     Freq type temperature family
1     fw C190104M 112231.0    C           19    C01
2     hw C190104M 143618.0    C           19    C01
3     fw C190302M  93466.0    C           19    C03
4     hw C190302M 125020.5    C           19    C03
5     fw C190303M 100433.5    C           19    C03
6     hw C190303M 120001.5    C           19    C03
(...)
55    fw C280806M 108140.5    C           28    C08
56    hw C280806M 146557.5    C           28    C08
57    fw W210101M 116224.0    W           21    W01
58    hw W210101M 154817.0    W           21    W01
59    fw W210103M 105458.0    W           21    W01
60    hw W210103M 142232.5    W           21    W01
```

Interagir avec le système

`system()` : permet de faire exécuter une commande au système d'exploitation
Permet notamment de commander d'autres programmes depuis R

Exemple : extraction des métadonnées exifs de fichiers photos avec le programme en ligne de commande exiv2
(à installer séparément http://www.exiv2.org/)

--> utilisation de ces données pour renommer les photos

Nom original des photos :

```
[1] "small_DSC_3113.JPG" "small_DSC_3135.JPG" "small_DSC_3180.JPG" "small_DSC_3187.JPG"
```

On veut les renommer selon la date et heure de prise de vue + le numéro d'origine de la photo :
YYYYMMDD_HHMMSS_nbr.JPG

```
[1] "20151101_170317_3113.JPG" "20151101_181552_3135.JPG" "20151104_110806_3180.JPG"
[4] "20151104_110918_3187.JPG"
>
```

Interagir avec le système

`system()` - Exemple : extraction des métadonnées (exifs) de fichiers photos avec le programe exiv2 (à installer séparément)

```
> setwd("/home/gilles/stats/R_Demo/rename_pictures")
> photos <- list.files(getwd(), pattern= "\\.JPG$")
> photos
[1] "small_DSC_3113.JPG" "small_DSC_3135.JPG" "small_DSC_3180.JPG" "small_DSC_3187.JPG"

> cmd <- paste("exiv2 -pa \"",photos[1], "\"", sep="")
> cmd
[1] "exiv2 -pa \"small_DSC_3113.JPG\""
```

Commande exiv2 telle qu'on l'aurait exécutée dans une console sans les \ pour afficher les exifs

On envoie la commande vers le système et on stocke le résultat dans un objet R `(intern = TRUE)`

```
> exif <- system(cmd, intern = TRUE)
> exif[1:5]
[1] "Exif.Image.Make                                Ascii      18  NIKON CORPORATION"
[2] "Exif.Image.Model                               Ascii      12  NIKON D7100"
[3] "Exif.Image.Orientation                         Short       1  top, left"
[4] "Exif.Image.XResolution                         Rational    1  300"
[5] "Exif.Image.YResolution                         Rational    1  300"


> exif[grepl("Exif.Photo.DateTimeOriginal", exif)]
[1] "Exif.Photo.DateTimeOriginal                    Ascii      20  2015:11:01 17:03:17"
>
```

On extrait la ligne qui contient la date et l'heure de prise de vue

Interagir avec le système

`system()` - Exemple : renommer des photos sur base des métadonnées (exifs) avec le programme exiv2 (à installer séparément)

```
> newname_photos <- function(filename) {
+
+     # read metadata and store it into a vector
+     a <- system(paste("exiv2 -pa \"",filename, "\"", sep=""), intern=TRUE,
+                 ignore.stdout = FALSE, ignore.stderr = TRUE)
+     a <- a[grepl("Exif.Photo.DateTimeOriginal", a)]
+     a <- strsplit(a, "( +)") # split each line where there is one or more spaces
+     if(length(a) != 0) {a <- paste(gsub(":", "", a[[1]][4]),
+                                    gsub(":", "", a[[1]][5]), sep="_")}
+
+     extention <- toupper(substring(filename, regexpr("\\..{3}$", filename),
+                                    nchar(filename)))
+     number <- gsub(".*([0-9]{4})\\..{3,4}$", "\\1", filename)
+
+     return(paste(a, "_", number, extention, sep=""))
+ }
>
> new_names <- unlist(lapply(photos, newname_photos))
> new_names
[1] "20151101_170317_3113.JPG" "20151101_181552_3135.JPG" "20151104_110806_3180.JPG"
[4] "20151104_110918_3187.JPG"
> photos
[1] "small_DSC_3113.JPG" "small_DSC_3135.JPG" "small_DSC_3180.JPG" "small_DSC_3187.JPG"

> file.rename(from = photos, to = new_names)
[1] TRUE TRUE TRUE TRUE
```

Interagir avec le système

`eval()` et `parse()` :

exécuter une chaîne de texte comme si c'était une commande R

```
> eval("1+1")
[1] "1+1"
```

evalue "1+1" qui n'est qu'une chaîne de texte

```
> parse(text = "1+1")
expression(1+1)
```

transforme une chaîne de texte en expression

```
> eval(parse(text = "1+1"))
[1] 2
```

`eval()` évalue la chaîne de texte transformée en expression par `parse()`

Interagir avec le système

`eval()` et `parse()` :

Relativement peu utilisé et peu recommandé mais parfois utile

```
> library(fortunes)

> fortune(106)
If the answer is parse() you should usually rethink the question.
   -- Thomas Lumley
      R-help (February 2005)

> fortune('106')
Personally I have never regretted trying not to underestimate my own
future stupidity.
   -- Greg Snow (explaining why eval(parse(...)) is often suboptimal,
answering a question triggered by the infamous fortune(106))
      R-help (January 2007)
```

Pour plus d'informations cherchez avec les mots clés
"Computing on the language"

Voir par exemple :
http://adv-r.had.co.nz/Computing-on-the-language.html

Exercices :
Essayez de répéter les exemples vus ci-dessus

surface des ailes de papillons
(data/butterflies_wings)

modifier les noms des photos
(data/pictures)