DATA MINING AVEC R
① Le langage R

Christophe Lalanne

ch.lalanne@gmail.com
www.aliquote.org

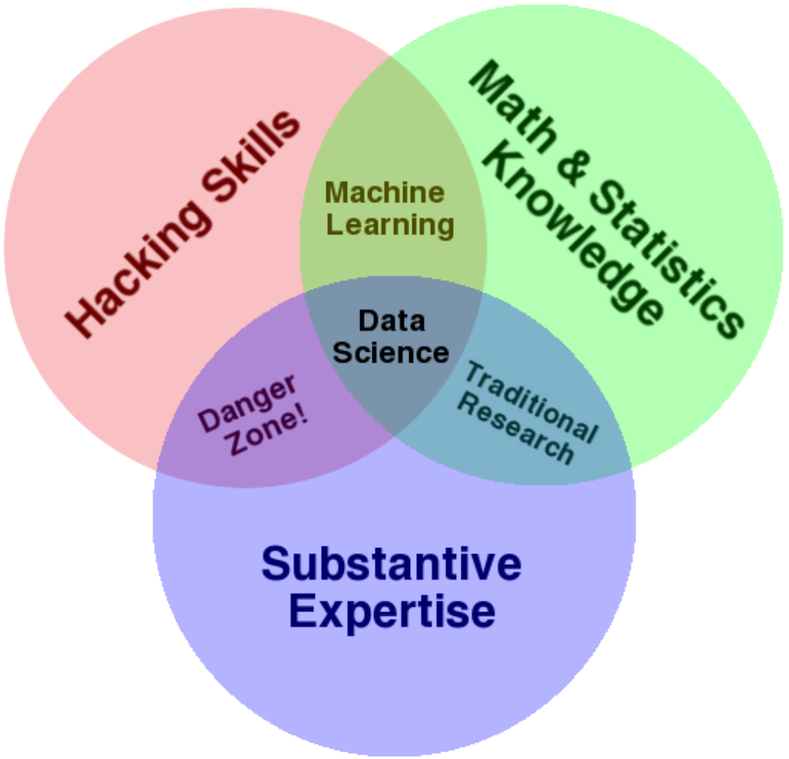
Synopsis

- Éléments du langage
- Structures de contrôle
- Gestion de données
- Environnement RStudio
- Applications

http://r-project.org

R est un langage versatile pour la manipulation et le traitement des données. C'est un logiciel open-source (dérivé du langage S), offrant à la fois un langage de programmation et des routines d'analyse statistiques puissantes. De nombreux packages, disponibles sur le site CRAN, fournissent des extensions aux fonctionnalités de base de R.

Data Science



Source: http://drewconway.com/zia/2013/3/26/the-data-science-venn-diagram

Autres outils

- Terminal Unix (Bash ou Zsh), Git, éditeur de texte
- sed/awk, csvkit, jq
- sqlite, Redis
- Weka, vowpal, libsvm (liblinear)

Git

Si Git n'est pas installé sur votre système, l'installation est relativement rapide. Pour vérifier que Git fonctionne, il suffit de taper dans un Terminal :

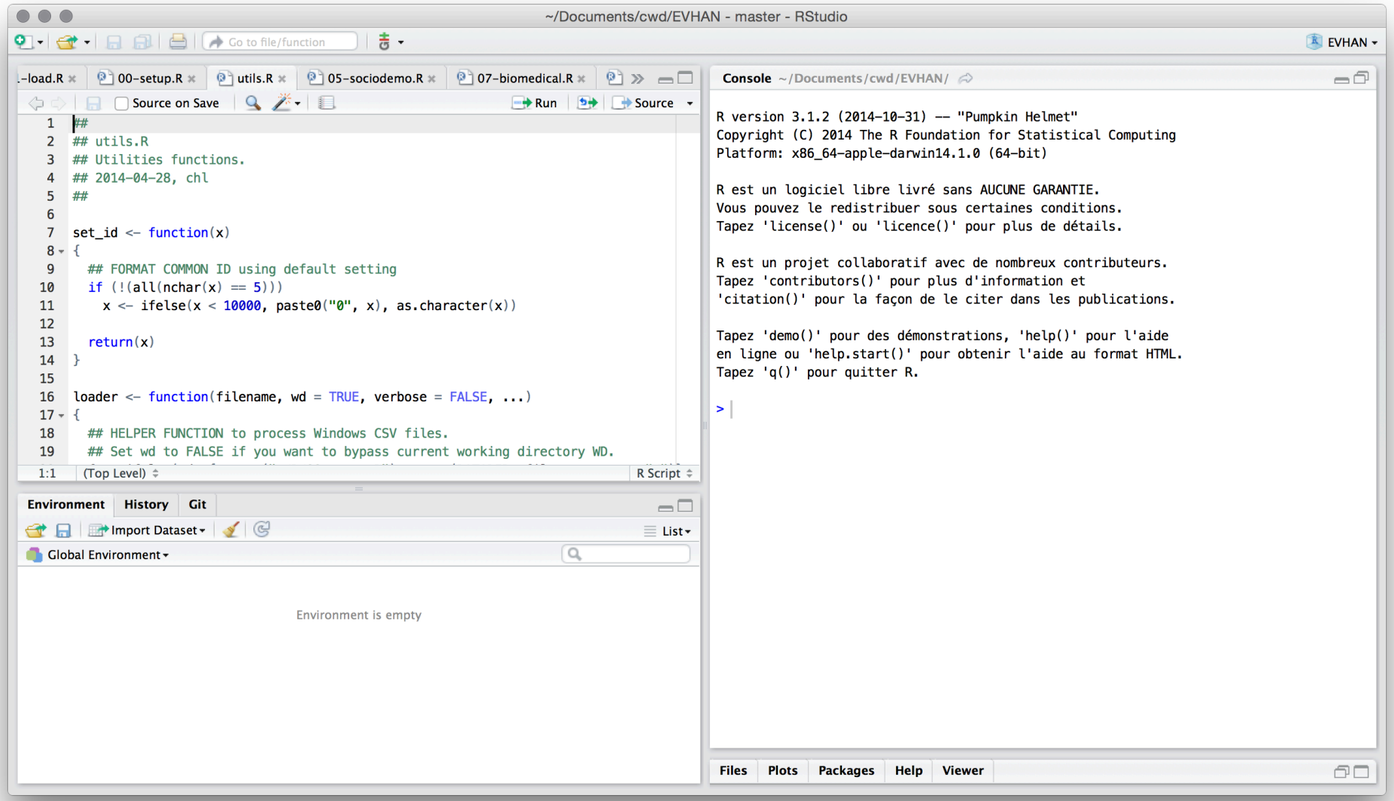
```
$ git --version
```

SourceTree offre une interface à Git et Mercurial. BitBucket permet de maintenir des dépôts privés gratuitement.

Ressources

1. Matloff (2011). The Art of R Programming. No Starch Press.
2. Wickham (2015). Advanced R.
3. Hastie et al. (2009). The Elements of Statistical Learning. Springer.
4. James et al. (2015). An Introduction to Statistical Learning. Springer. [**ISLR**]
5. Leskovec et al. (2011). Mining of Massive Datasets. Cambridge

RStudio


~/Documents/cwd/EVHAN - master - RStudio
Go to file/function
EVHAN
.-load.R
00-setup.R
utils.R
05-sociodemo.R
07-biomedical.R
Source on Save
Run
Source
1 ##
2 ## utils.R
3 ## Utilities functions.
4 ## 2014-04-28, chl
5 ##
6
7 set_id <- function(x)
8 {
9 ## FORMAT COMMON ID using default setting
10 if (!(all(nchar(x) == 5)))
11 x <- ifelse(x < 10000, paste0("0", x), as.character(x))
12
13 return(x)
14 }
15
16 loader <- function(filename, wd = TRUE, verbose = FALSE, ...)
17 {
18 ## HELPER FUNCTION to process Windows CSV files.
19 ## Set wd to FALSE if you want to bypass current working directory WD.
1:1
(Top Level)
R Script
Environment
History
Git
Import Dataset
List
Global Environment
Environment is empty
Console ~/Documents/cwd/EVHAN/
R version 3.1.2 (2014-10-31) -- "Pumpkin Helmet"
Copyright (C) 2014 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin14.1.0 (64-bit)
R est un logiciel libre livré sans AUCUNE GARANTIE.
Vous pouvez le redistribuer sous certaines conditions.
Tapez 'license()' ou 'licence()' pour plus de détails.
R est un projet collaboratif avec de nombreux contributeurs.
Tapez 'contributors()' pour plus d'information et
'citation()' pour la façon de le citer dans les publications.
Tapez 'demo()' pour des démonstrations, 'help()' pour l'aide
en ligne ou 'help.start()' pour obtenir l'aide au format HTML.
Tapez 'q()' pour quitter R.
>
Files
Plots
Packages
Help
Viewer

R comme calculateur

Read **E**val **P**rint **L**oop

```
> r <- 5
> 2 * pi * r^2
[1] 157.0796
```

Variable, constantes, opérateur, assignation, passage par valeur/ référence…

Lecture conseillée (plus tard) : Burns (2011). R Inferno.

Aide en ligne

`help(command)` ou `?command` (sauf quelques cas particuliers)

```
> ?pi
Constants                  package:base                  R Documentation

Built-in Constants

Description:

     Constants built into R.

Usage:

     LETTERS
     letters
     month.abb
     month.name
     pi

Details:
```

Packages

Packages de base : `stats`, `lattice`, MASS, etc.

```
library(MASS)
help(package = MASS)
data(package = MASS)
MASS::lm.ridge  # espace de nom
if (!require(dplyr))
  install.packages("dplyr", dependencies = TRUE)
```

CRAN : liste des packages, Task Views, documentation.

```
> sessionInfo()
R version 3.1.2 (2014-10-31)
Platform: x86_64-apple-darwin14.1.0 (64-bit)

locale:
[1] fr_FR.UTF-8/fr_FR.UTF-8/fr_FR.UTF-8/C/fr_FR.UTF-8/fr_FR.UTF-8

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
[1] tools_3.1.2
```

Script R

Plutôt que de saisir toutes les commandes dans la console, il est souvent plus judicieux d'enregistrer ses commandes dans un script R : simple fichier texte avec extension `.r` ou `.R`.

Intérêt : construire une suite d'instructions réutilisables, modifiables et partageables.

Espace de travail

On distingue le répertoire courant de travail et l'espace de travail ou environnement.

```
getwd()
setwd("~/Documents")
dir(pattern = ".R")
source("./monfichier.R")
ls()
```

RStudio facilite la gestion de l'espace de travail, l'édition de scripts, et la navigation dans le système de fichiers.

Les objets R

L'élément de base est le **vecteur** (un scalaire est alors un vecteur de longueur 1) que l'on peut créer avec la commande `c()`, entre autres, et dont les éléments sont de type :

- `numeric`, des nombres (entiers ou flottants)
- `logical`, des booléens à valeur dans {`TRUE`, `FALSE`}
- `character`, des caractères ASCII

On peut ajouter `integer` et `complex`.

Les expressions suivantes sont valides :

```
nombre <- 3.141593
v <- c(1,2,3,4)
b <- c(T,F,T,F)
s <- c("h","e","l","l","o")
a <- b <- rnorm(2)
u <- b
```

Adresser les éléments d'un vecteur

On utilise [pour indexer les n éléments d'un vecteur, sachant que l'index du premier élément vaut 1 (et non 0 comme dans certains langages) :

```
v[1]
v[c(1,4)]
v[1:4]
```

On peut adresser les élements d'un vecteur en utilisant les valeurs contenues dans un autre vecteur (principe d'un **dictionnaire**), par exemple :

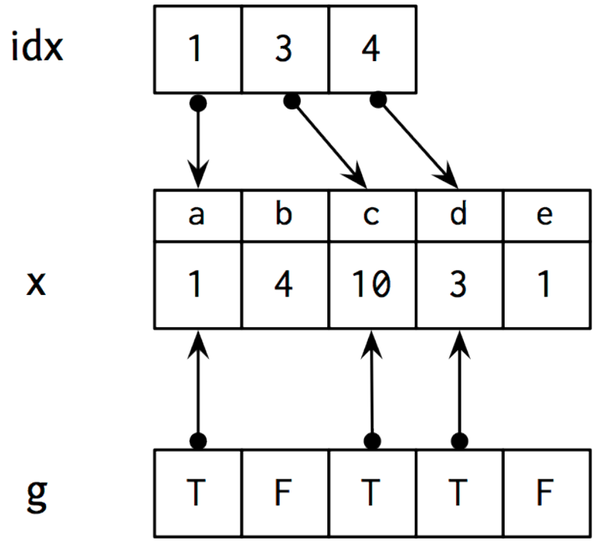
```
v <- c(1,2,3,4)
b <- c(T,F,T,F)
s <- c("h","e","l","l","o")
v[b]    # valeurs de v telles que b vaut TRUE
s[v[2]] # v[2]=2, d'où s[2] qui vaut "e"
```

Illustration

```
x <- c(1,4,10,3,1)
names(x) <- letters[1:length(x)]
idx <- c(1,3,4)
g <- c(T,F,T,T,F)
```

Sélection indexée des éléments de x :

```
x[idx]
x[g]
```



Trier, classer

On peut également effectuer des **opérations de tri** sur un vecteur :

```
x <- 1:10
xs <- sample(x)
sort(xs, decreasing = TRUE)
order(xs)
```

ou obtenir le **rang** de ses élements

```
rank(rev(xs))
```

Nombres aléatoires

Dès que l'on manipule des nombres aléatoires, il est nécessaire de fixer la graine du générateur congruentiel, autrement on ne pourra pas reproduire la séquence (e.g., cas des simulations).

```
> runif(6)
[1] 0.1791854 0.7889672 0.8164497 0.2004527 0.2382137 0.9693377
> runif(6)
[1] 0.80375195 0.01630027 0.45203035 0.58823877 0.47040014 0.14629810
```

Illustration en lançant R depuis un Terminal :

```
% Rscript -e "print(runif(6))"
[1] 0.44252118 0.61164540 0.24009883 0.01861185 0.45266777 0.54880932
% Rscript -e "print(runif(6))"
[1] 0.04160656 0.42925736 0.89571922 0.62046257 0.94939723 0.59708779
% Rscript -e "set.seed(101); print(runif(6))"
[1] 0.37219838 0.04382482 0.70968402 0.65769040 0.24985572 0.30005483
% Rscript -e "set.seed(101); print(runif(6))"
[1] 0.37219838 0.04382482 0.70968402 0.65769040 0.24985572 0.30005483
```

Codage des données manquantes

Le symbole NA est utilisé pour désigner une donnée manquante (différent d'une valeur non représentable en machine telle que `-Inf`).

```
x <- 1:10
x[5] <- NA
is.na(x)
sum(is.na(x))
```

Représentation des données catégorielles

R utilise le terme `factor()` pour représenter une variable catégorielle possédant plusieurs modalités (ou niveaux) mutuellement exclusives. Hors cas des variables binaires (0/1), l'utilisation de facteur est recommandée car de nombreuses fonctions de R ont un comportement spécifique en présence de facteurs. Ceux-ci sont également utiles dans les notations par formules, les graphiques, les procédures d'aggrégation ou de fusion de données, et de nombreux modèles statistiques.

```
> state <- sample(c("on", "off"), 20, rep = TRUE)
> class(state)
[1] "character"
> head(state)
[1] "off" "off" "on"  "off" "on"  "off"
> state <- factor(state)
> class(state)
[1] "factor"
> head(state)
[1] off off on  off on  off
Levels: off on
> levels(state)
[1] "off" "on"
> nlevels(state)
[1] 2
```

Un facteur R possède des **niveaux** (`levels=`) et des **étiquettes** (`labels=`) associées à chacun de ces niveaux. Le premier niveau est appelé **niveau de référence**.

```
> evt <- c(0,1)
> y <- sample(evt, 10, rep = TRUE)
> y
 [1] 1 1 1 0 1 1 0 0 1 0
> factor(y, levels = 0:1, labels = c("non", "oui"))
 [1] oui oui oui non oui oui non non oui non
Levels: non oui
```

Manipulation de chaînes de caractères

La plupart des opérations habituelles sur les chaînes de caractères sont possibles sous R :

- recherche de motif par regex
- extraction de sous-chaînes
- découpage selon le motif
- remplacement de sous-chaînes

Pour plus de souplesse : package stringr et stringi

```
> a <- c("toto-24", "titi-13")
> substr(a, 1, 4)
[1] "toto" "titi"
> strsplit(a, "-")
[[1]]
[1] "toto" "24"

[[2]]
[1] "titi" "13"
> gsub("[1-9]*$", "", a)
[1] "toto-" "titi-"
```

Manipulation de dates

Attention à la notation anglo-saxonne (ou localisation du système) : mm/dd/yyyy (vs. dd/mm/yyyy en français)

```
> d1 <- c("22/01/1998", "24/06/1999")
> d2 <- as.POSIXct(d1, format = "%d/%m/%Y")
> d2
[1] "1998-01-22 CET"  "1999-06-24 CEST"
> diff(d2)
Time difference of 517.9583 days
```

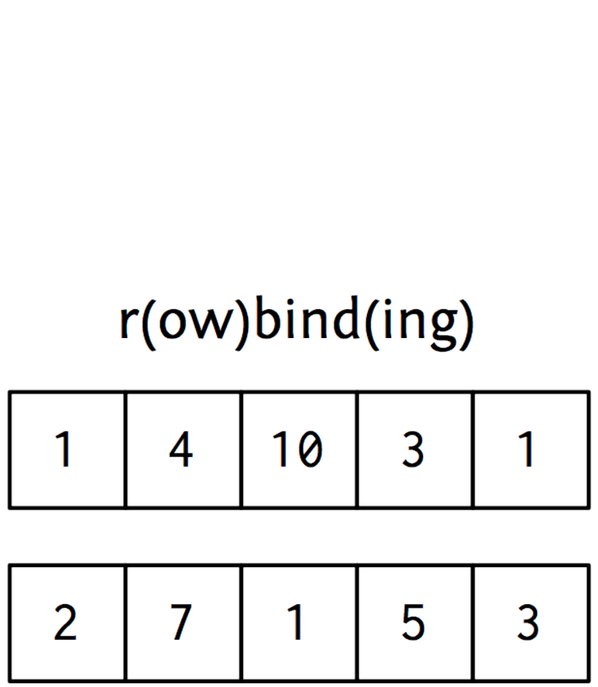
Pour plus de souplesse : package lubridate

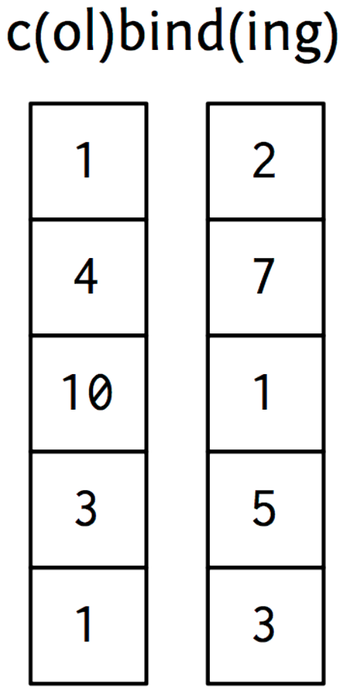
Autres types d'objets

Matrice

Collection de valeurs de même type (e.g., nombres ou caractères) arrangées dans une structure à deux dimensions (`array`, n-dimensions).

```
x <- c(1,4,10,3,1)
y <- c(2,7,1,5,3)
cbind(x, y)
rbind(x, y)
```





Liste

Collection de valeurs de différents types, de taille éventuellement variable.

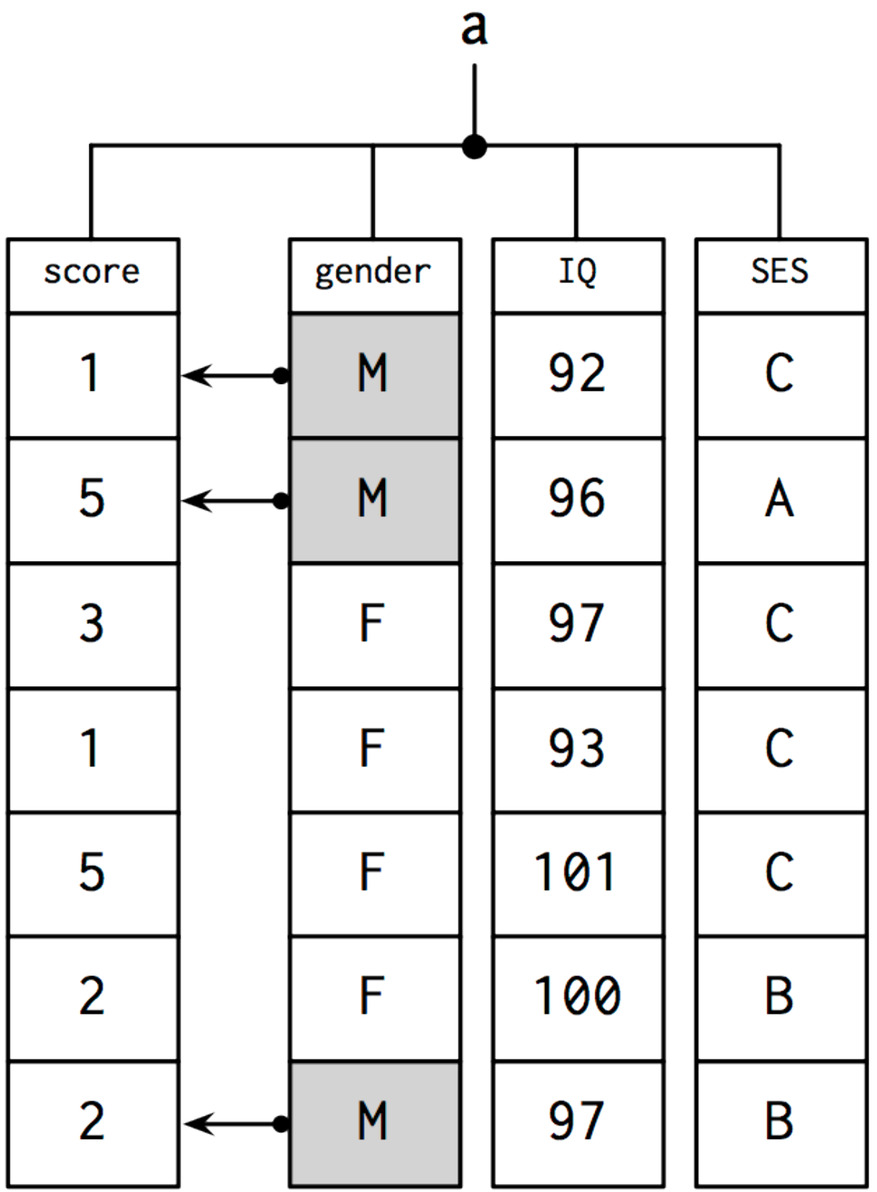
```
x <- c(1,4,10,3,1)
y <- c("a", "b", "c")
z <- list(alpha = x, beta = y)

> str(z)
List of 2
 $ alpha: num [1:5] 1 4 10 3 1
 $ beta : chr [1:3] "a" "b" "c"
```

Data frame

Collection de valeurs de différents types, de même taille (restriction par rapport aux listes).

```
x <- c(1,4,10,3,1)
y <- letters[1:5]
z <- data.frame(alpha = x, beta = y)
```

```
> str(z)
'data.frame':   5 obs. of  2 variables:
 $ alpha: num  1 4 10 3 1
 $ beta : Factor w/ 5 levels "a","b","c","d",..: 1 2 3 4 5
```


a
score
gender
IQ
SES
1
M
92
C
5
M
96
A
3
F
97
C
1
F
93
C
5
F
101
C
2
F
100
B
2
M
97
B

Structures de contrôle

Branchement conditionnel (If-else)

```
test <- TRUE
if (test) print("Hello") else print("Bye-bye")
ifelse(test, "Hello", "Bye-bye")
```

La dernière expression est vectorisée :

```
n <- 20
x <- sample(letters[1:3], n, rep = TRUE)
y <- sample(1:100, n)
z <- ifelse(x == "a", 1, 0)
```

Itération (for)

```
s <- 0
for (i in 1:10) {
    s <- s+i
}
```

En plus simple :

```
cumsum(1:10)
```

Souvent, on peut exploiter les fonctions de la famille `*apply()` :

```
val <- c(2, 8, 3, 4)
f <- function(x) x+2
res <- numeric(4)
for (i in 1:4) res[i] <- f(val[i])
```

En plus simple :

```
sapply(val, f)
```

Fonctions

Comme dans d'autres langages de programmation, une fonction accepte des arguments.

```
f <- function(x, y = NULL, ...) {
  ...
  return(val)
}
```

```
> args(sapply)
function (X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
NULL
```

```r
f <- function(x) mean(x)
v <- sample(1:100, 10)
f(v)
```

Ajout de paramètres avec valeur par défaut :

```r
f <- function(x, na.rm = FALSE) mean(x, na.rm = na.rm)
v[sample(1:length(v), 5)] <- NA
f(v)  ## na.rm = FALSE
f(v, na.rm = TRUE)
```

Application 1

1. Générer une séquence de n événements 0/1, avec p = 0,5.
2. Calculer la proportion de résultats positifs obtenus en (1).
3. Répéter l'expérience m = 1000 fois et vérifier la distribution des proportions observées.
4. Quel est le 1er événement positif pour une expérience ? En moyenne pour m = 1000 expériences ?

Your turn.

Différents format de données

- fichiers de type CSV (RFC4180)
- fichiers structurés (XML, JSON)
- fichiers binaires (`.mat`, `.sav`, `.dta`)

Les données peuvent être dispatchées dans plusieurs fichiers, e.g. fichiers Nifti `*.hdr+*.img` en IRM (globbing), ou PLINK `*.bed +*.bim` en Génétique (merging).

Lecture de fichiers et data frames

Il existe plusieurs commandes pour lire des fichiers plats, mais dans le cas des **fichiers rectangulaires** (variables en colonnes et observations en lignes), on utilisera généralement `read.csv()` ou `read.table()`.

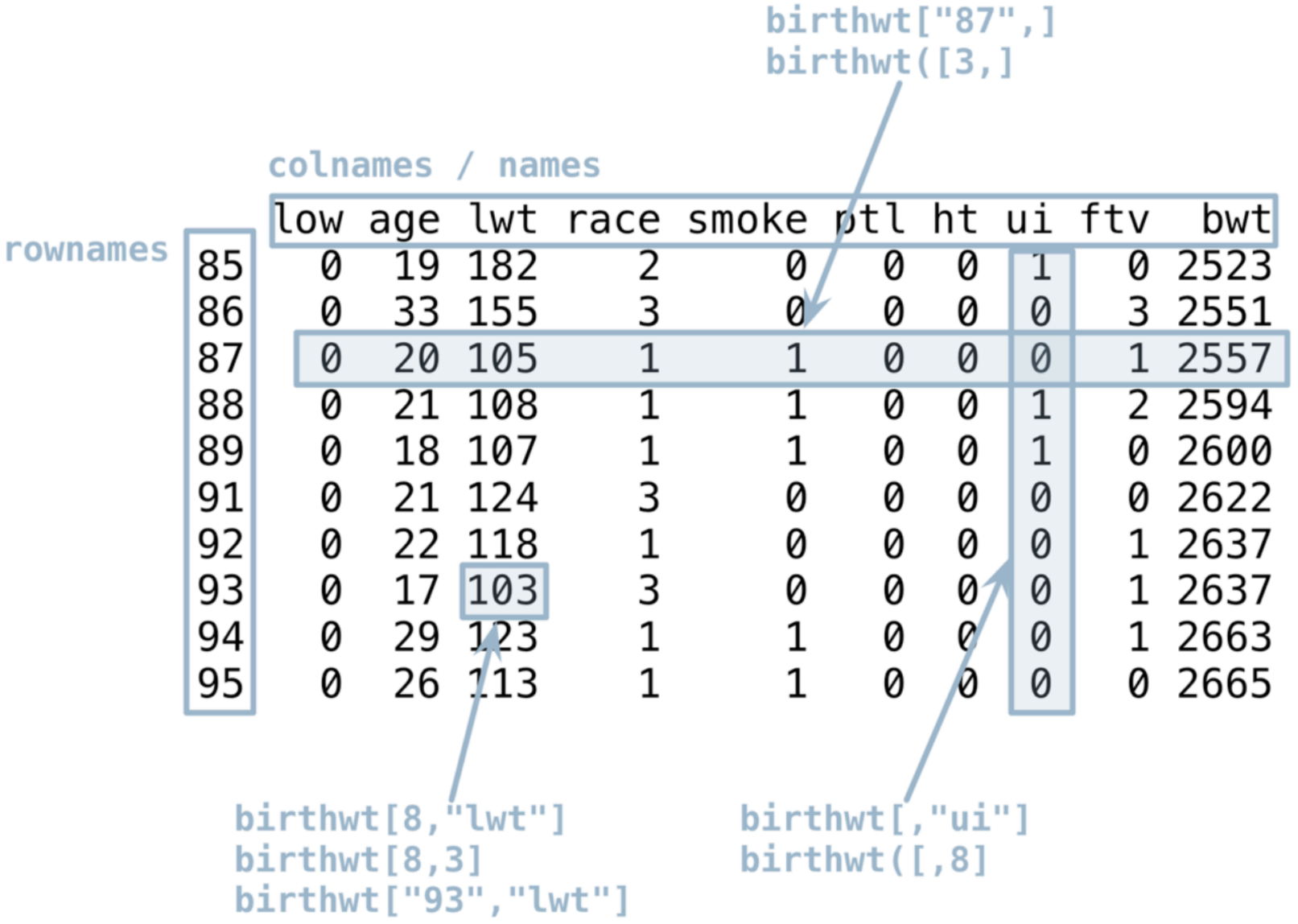
Exemple 1 : birthwt.csv (189 lignes, 10 colonnes)

```
0,19,182,2,0,0,0,1,0,2523
0,33,155,3,0,0,0,0,3,2551
0,20,105,1,1,0,0,0,1,2557
```

Exemple 2 : birthwt.dat (189 lignes, 10 colonnes)

```
0 19 182 2 0 0 0 1 0 2523
0 33 155 3 0 0 0 0 3 2551
0 20 105 1 1 0 0 0 1 2557
```

```
birthwt <- read.table("birthwt.dat", header = FALSE)
varnames <- c("low", "age", "lwt", "race", "smoke",
              "ptl", "ht", "ui", "ftv", "bwt")
names(birthwt) <- varnames
head(birthwt, n = 10)
```


birthwt["87",]
birthwt([3,]
colnames / names
rownames
low age lwt race smoke ptl ht ui ftv bwt
85 0 19 182 2 0 0 0 1 0 2523
86 0 33 155 3 0 0 0 0 3 2551
87 0 20 105 1 1 0 0 0 1 2557
88 0 21 108 1 1 0 0 1 2 2594
89 0 18 107 1 1 0 0 1 0 2600
91 0 21 124 3 0 0 0 0 0 2622
92 0 22 118 1 0 0 0 0 1 2637
93 0 17 103 3 0 0 0 0 1 2637
94 0 29 123 1 1 0 0 0 1 2663
95 0 26 113 1 1 0 0 0 0 2665
birthwt[8,"lwt"]
birthwt[8,3]
birthwt["93","lwt"]
birthwt[,"ui"]
birthwt([,8]

Autres packages pour la lecture de gros fichiers plats : data.table (`fread()`), sqldf (`read.csv.sql()`), readr (`read_csv()`).

Exemple : `flight.csv` (16.35 MB)

```
> system.time(d <- fread("flights.csv"))
utilisateur      système       écoulé
      0.265        0.013        0.282
> system.time(d <- tbl_df(read.csv("flights.csv",
                                   stringsAsFactors = FALSE)))
utilisateur      système       écoulé
      5.825        0.085        6.031
```

Source de données : Hadley Wickham, dplyr tutorial

RStudio

Présentation

Application 2

Données `birthwt` du package `MASS`.

1. Recoder les variables en facteur selon l'aide en ligne.
2. Combien y'a-t-il d'événements positifs `ht` et `ui`, séparéméent ou les deux ensemble ?
3. Combien y'a-t-il d'événements positifs `ht` pour les femmes ayant un poids < 55 kg ?
4. Enregistrer toutes les instructions dans un script.

Your turn.

TODO LIST

1. Installer les packages suivants (avec leur dépendances) : `dplyr` et `ggplot2`
2. Créer un compte GitHub au besoin
3. Lire ISLR §2.1