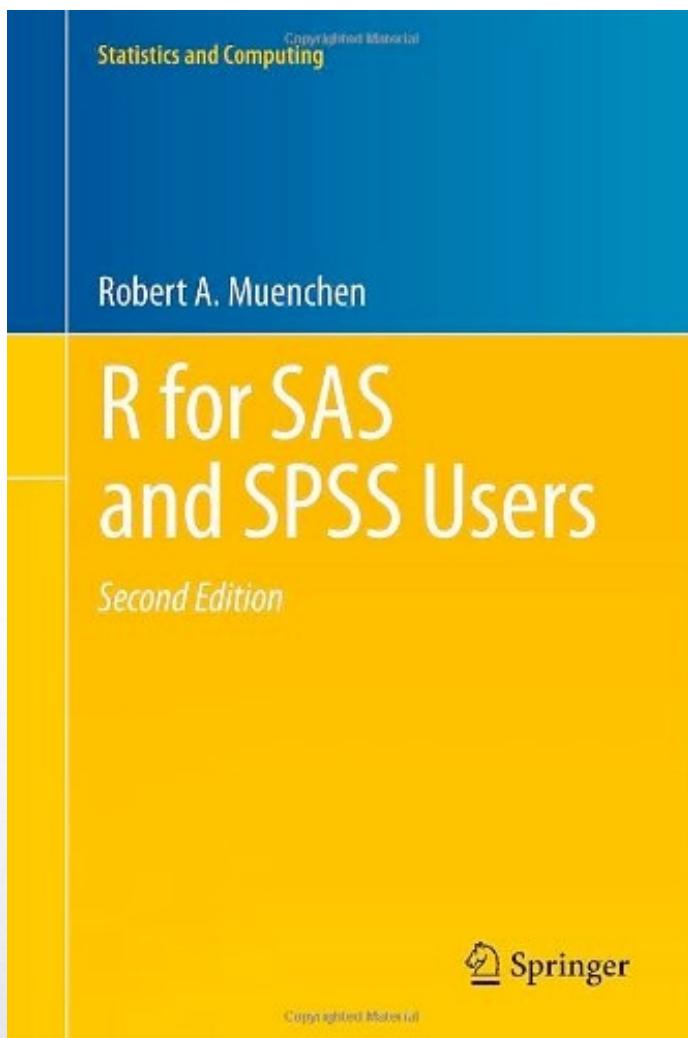




# Utilisation de

---

Dominique Muller  
(dominique.muller@univ-grenoble-alpes.fr)

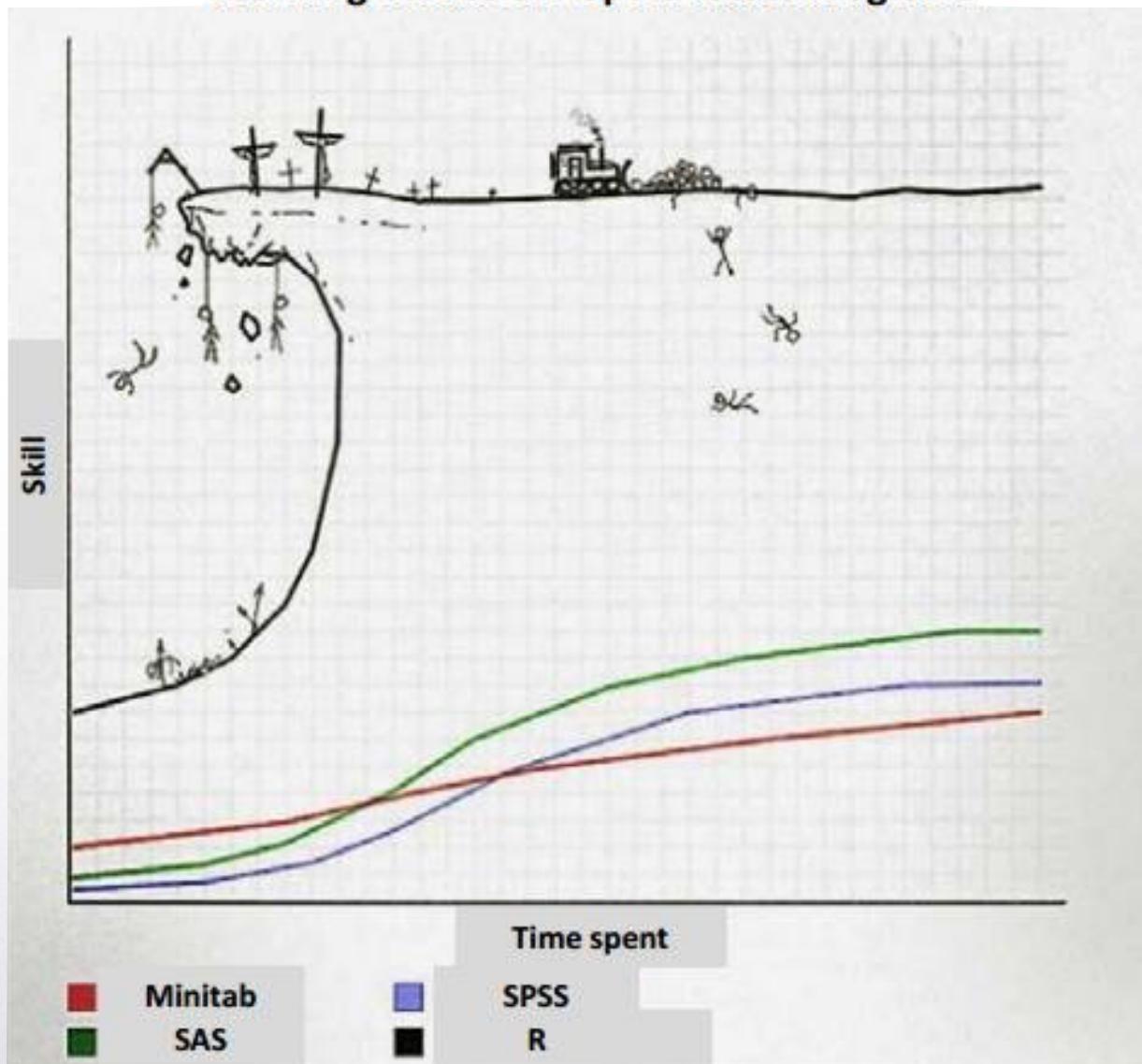




# Pourquoi utiliser R ?

- Flexibilité et puissance
- Simplicité du langage

## Learning Curves of Popular Stats Programs





# Pourquoi utiliser R ?

- Flexibilité et puissance
- SimPLICITÉ du langage
- Multiplateforme (Windows, Mac, Linux)
- C'est gratuit !



Rendez vous sur :  
[webcom.upmf-grenoble.fr/LIP/DMuller/](http://webcom.upmf-grenoble.fr/LIP/DMuller/)



The image shows an aerial night view of a dense urban area, likely Grenoble, France. The city is filled with numerous buildings, many of which are lit up with warm yellow and orange lights. The streets below are also illuminated, creating a complex network of light trails. In the foreground, there's a darker area that appears to be a park or a less densely built part of the city. The overall atmosphere is vibrant and suggests a bustling city at night.

[Home](#)   [Research](#)   [Team](#)   [Publications >](#)   [Resources >](#)   [Highlights](#)

Cours sur R

Cours M2R



# Installation

- <http://www.r-project.org>
- Sur le site, téléchargement du logiciel et des différents packages (ce qu'on peut aussi faire directement dans R)



# Les différentes parties du logiciel

Console R

R version 2.15.1 (2012-06-22) -- "Roasted Marshmallows"  
Copyright (C) 2012 The R Foundation for Statistical Computing  
ISBN 3-900051-07-0  
Platform: i386-apple-darwin9.8.0/i386 (32-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.  
  
[R.app GUI 1.52 (6188) i386-apple-darwin9.8.0]  
  
[Historique recherché depuis /Users/dominiquemuller/Dropbox/r/donnees/.Rapp.history]  
  
>

Console

ImportsDonnees

```
1 #####  
2 # Importation des données #  
3 #####  
4  
5  
6  
7 # Solution 1 : Si le fichier a été enregistré en format txt séparateur tabulation :  
8 # (à noter qu'ici on n'indique pas le chemin car R sait où regarder,  
9 # si ce n'est pas le cas, il faut indiquer le chemin)  
10 DF<-read.table("shooterShort.txt",header=TRUE,sep="\t",dec=".")  
11  
12 # A voir à l'usage  
13 DF<-read.table("shooterShort.txt",header=TRUE,dec=".") # semble marcher également  
14  
15 str(DF)  
16  
17 # Solution 2 : Si le fichier a été enregistré en format csv séparateur ";" :  
18 DF<-read.table("shooterShort.csv",header=TRUE,sep=";",dec=".")  
19  
20  
21 # Solution 3 : Si le fichier a été enregistré en format xlsx :  
22 # Nous devons installer puis activer un package  
23 install.packages("xlsx") # télécharge  
24 library(xlsx) # active le package  
25  
26 DF<-read.xlsx("shooterShort.xlsx",1) # le 1 indique que les données sont dans feuille 1  
27  
28  
29  
30  
31 # Pour aller plus loin :
```

Feuille de script

# Pour lancer une commande

```
## Importation des données ##

# Solution 1 : Si le fichier a été enregistré en format txt séparateur tabulation :
# (à noter qu'ici on n'indique pas le chemin car R sait où regarder,
# si ce n'est pas le cas, il faut indiquer le chemin)
DF<-read.table("shooterShort.txt",header=TRUE,sep="\t",dec=",") # A voir à l'usage
DF<-read.table("shooterShort.txt",header=TRUE,dec=",") # semble marcher également

str(DF)

# Solution 2 : Si le fichier a été enregistré en format csv séparateur ";" :
DF<-read.table("shooterShort.csv",header=TRUE,sep=";",dec=",") # Nous devons installer puis activer un package
install.packages("xlsx") # télécharge
library(xlsx) # active le package
DF<-read.xlsx("shooterShort.xlsx",1) # le 1 indique que les données sont dans feuille 1

# Pour aller plus loin :
```

Dans la feuille de script :

- soit on place le curseur sur la ligne et on utilise **CTRL + R** (Windows) ou **CMD + Entrée** (Mac)
- soit on sélectionne l'ensemble des lignes ou éléments d'une ligne qu'on veut lancer et ensuite **CTRL + R** (Windows) ou **CMD + Entrée** (Mac)



# Grandes notions

- Fonctions
- Création et types d'objets
- Notion d'environnements

# Grandes notions

- Fonctions
- Création et types d'objets
- Notion d'environnements

# Fonctions

- Chaque opération sur R est effectuée à partir d'une fonction
- Il existe :
  - des fonctions qu'on trouve par défaut dans R ("in built")
  - des fonctions ou ensembles de fonctions regroupées dans des packages
  - des fonctions que l'on crée nous-mêmes
- A noter qu'il existe des fonctions et des fonctions génériques (des fonctions qui ne font pas la même chose selon le type d'objet qu'on leur donne)

# Fonctions

- fonction(Argument 1, Argument 2, ...)
- Trois exemples fonctionnant aussi bien :
  - `seq(from=1,to=10,by=1)` (va nous donner [1] 1 2 3 4 5 6 7 8 9 10)
  - `seq(1,10,1)`
  - `seq(by=1,from=1,to=10)`
  - `seq(1,10) # Marche car certains arguments ont une valeur par défaut, ici il s'agit de 1`
- Pour connaître les différents arguments possibles, il suffit d'utiliser l'aide : par exemple => `?seq`



# Installation de packages

Pour installer un package, deux étapes :

- le télécharger (une seule fois)

ex : `install.packages("readxl")`

- l'activer (à chaque fois qu'on ouvre R)

ex : `library(readxl)`



# Grandes notions

- Fonctions
- Création et types d'objets
- Notion d'environnements



# Grandes notions

- Fonctions
- Création et types d'objets
- Notion d'environnements

# Création d'objets

- Noms d'objets doivent commencer par une lettre
- R est "case sensitive"
- Utilisation du symbole "`<-`" pour créer l'objet du nom choisi
- Exemple :

```
Bob1 <- 5
```

```
Bob2 <- "garcon"
```

```
Bob3 <- c(5,2)      (ici le "c" indique qu'on "combine" différents éléments)
```

```
Bob4 <- c(« garcon », "fille")
```

# Création d'objets

1. Nous créons des objets avec "<- " (par exemple : `Bob1 <- 5`). Ici R, ne nous montre pas le contenu de Bob1, il l'a juste créé
2. Ensuite, nous pouvons voir le contenu de Bob1, en tapant simplement `Bob1`

```
Bob1
```

```
[1] 5
```

3. Pour voir les objets créés : `ls()`

```
ls()
```

```
[1] "Bob1" "Bob2" "Bob3" "Bob4"
```



# Types d'objets

- Vecteurs (et facteurs)
- Matrices
- Data Frames
- Listes

# Vecteurs

- Définition : ensemble d'éléments de même nature
- Exemple :
  - `workshop1<-c(1,2,1,2,1,2,1,2,3,4)`
  - `workshop2<-c("R", "SAS", "R", "SAS", "R", "SAS", "R", "SAS", "SPSS", "STATA")`
- Un vecteur peut donc être "numérique" ou "caractère" :
  - `mode(workshop1)`, nous donne : [1] "numeric"
  - `mode(workshop2)`, nous donne : [1] "character"
- Pour une table de fréquence, utilisez : `table(workshop2)`

workshop2

R	SAS	SPSS	STATA
4	4	1	1

# Facteurs

- Définition : variables catégorielles

- Exemple 1 :

- `workshop<-c("R", "SAS", "R", "SAS", "R", "SAS", "R", "SAS", "SPSS", "STATA")`
  - `workshopF<-factor(workshop)`

- Exemple 2 :

- `workshop<-c(1,2,1,2,1,2,1,2)`  
`workshopF<-factor(workshop,`  
`levels=c(1,2,3,4),`  
`labels=c("R", "SAS", "SPSS", "STATA"))`

`table(workshopF)`

`workshopF`

R	SAS	SPSS	STATA
4	4	0	0

# Matrice

- Définition : ensemble de vecteurs (variables) de même nature (numériques ou caractères)
- Exemple avec 4 vecteurs :
  - `q1<-c(1,2,2,3,4,5,5,4)`
  - `q2<-c(1,1,2,1,5,4,3,5)`
  - `q3<-c(5,4,4,NA,2,5,4,5) # On note qu'il y a une valeur manquante`
  - `q4<-c(1,1,3,3,4,5,4,5)`
- qu'ensuite nous combinons en une matrice avec la fonction `cbind()` ("column bind") :
  - `myMatrix<-cbind(q1,q2,q3,q4)`
  - `myMatrix # Si on s'arrête à la ligne précédente, R ne montre pas le contenu de l'objet (j'omettrai souvent cette commande dans la suite des diapositives)`

	q1	q2	q3	q4
[1,]	1	1	5	1
[2,]	2	1	4	1
[3,]	2	2	4	3
[4,]	3	1	NA	3
[5,]	4	5	2	4
[6,]	5	4	5	5
[7,]	5	3	4	4
[8,]	4	5	5	5

# Data frames

- Définition : ensemble de vecteurs (variables) de même nature ou de natures différentes (nos tableaux de données habituels)
- Exemple :
  - `myDF<-data.frame(workshop,q1,q2,q3,q4)`

Row names

	workshop	q1	q2	q3	q4
1	R	1	1	5	1
2	SAS	2	1	4	1
3	R	2	2	4	3
4	SAS	3	1	NA	3
5	R	4	5	2	4
6	SAS	5	4	5	5
7	R	5	3	4	4
8	SAS	4	5	5	5

Même résultat avec `myDF<-data.frame(workshop,myMatrix)`

Autre exemple : `Df<-read.table("shooterShort.txt",header=TRUE,sep="\t",dec=",")`

# Data frames

- Ajoutons une variable "participant" (pp)

```
pp<-c(1,2,3,4,6,7,8,9)
```

- Sans utiliser row.names :

```
myDF<-data.frame(pp,workshop,q1,q2,q3,q4)
```

	pp	workshop	gender	q1	q2	q3	q4
1	1	R	f	1	1	5	1
2	2	SAS	f	2	1	4	1
3	3	R	f	2	2	4	3
4	4	SAS	<NA>	3	1	NA	3
5	6	R	m	4	5	2	4
6	7	SAS	m	5	4	5	5
7	8	R	m	5	3	4	4
8	9	SAS	m	4	5	5	5

- En utilisant row.names :

```
myDF<-data.frame(myDF, row.names="pp")
```

	workshop	gender	q1	q2	q3	q4
1	R	f	1	1	5	1
2	SAS	f	2	1	4	1
3	R	f	2	2	4	3
4	SAS	<NA>	3	1	NA	3
6	R	m	4	5	2	4
7	SAS	m	5	4	5	5
8	R	m	5	3	4	4
9	SAS	m	4	5	5	5



# Listes

- Définition : Une liste est un objet qui peut lui-même contenir des objets de tous les types déjà mentionnés
- Exemple :
  - `myList<-list(q1,workshop,myDF,myMatrix)`

Ici myList contient donc un vecteur, un facteur, un dataframe et une matrice

# Exercice I

- Créer un vecteur de type caractère nommé "sujet" et comportant 4 prénoms
- Créer un vecteur numérique nommé "age" et comportant 4 valeurs d'âge (en année)
- Créer un vecteur numérique nommé "taille" et comportant 4 valeurs de taille en cm
- Créer une matrice combinant les vecteurs age et taille
- Créer le dataframe "DFexI" combinant les vecteurs sujet, age et taille

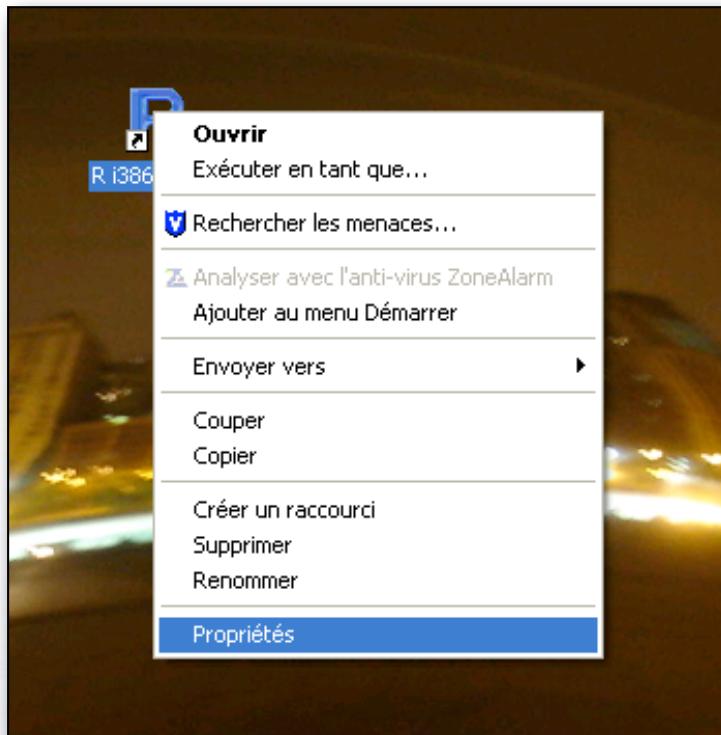
# Importation des données

Pour définir un espace de travail, il existe (au moins) trois stratégies :

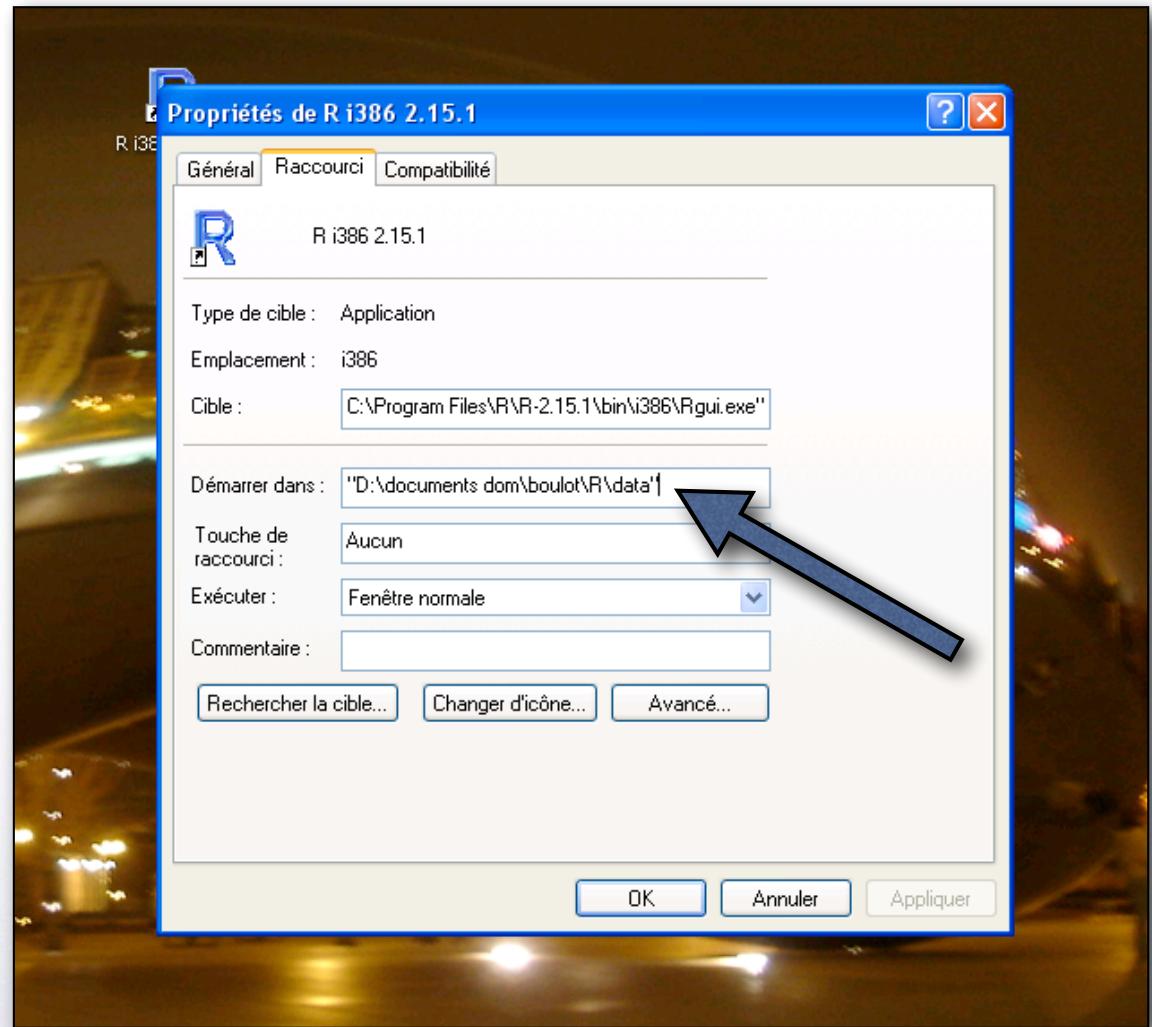
- Soit définir un répertoire fixe dans lequel nous mettrons toutes les données
- Soit redéfinir à chaque fois le répertoire où se trouve les données
- Soit indiquer le chemin complet à chaque fois dans la syntaxe



# Pour utiliser un répertoire fixe (Windows)

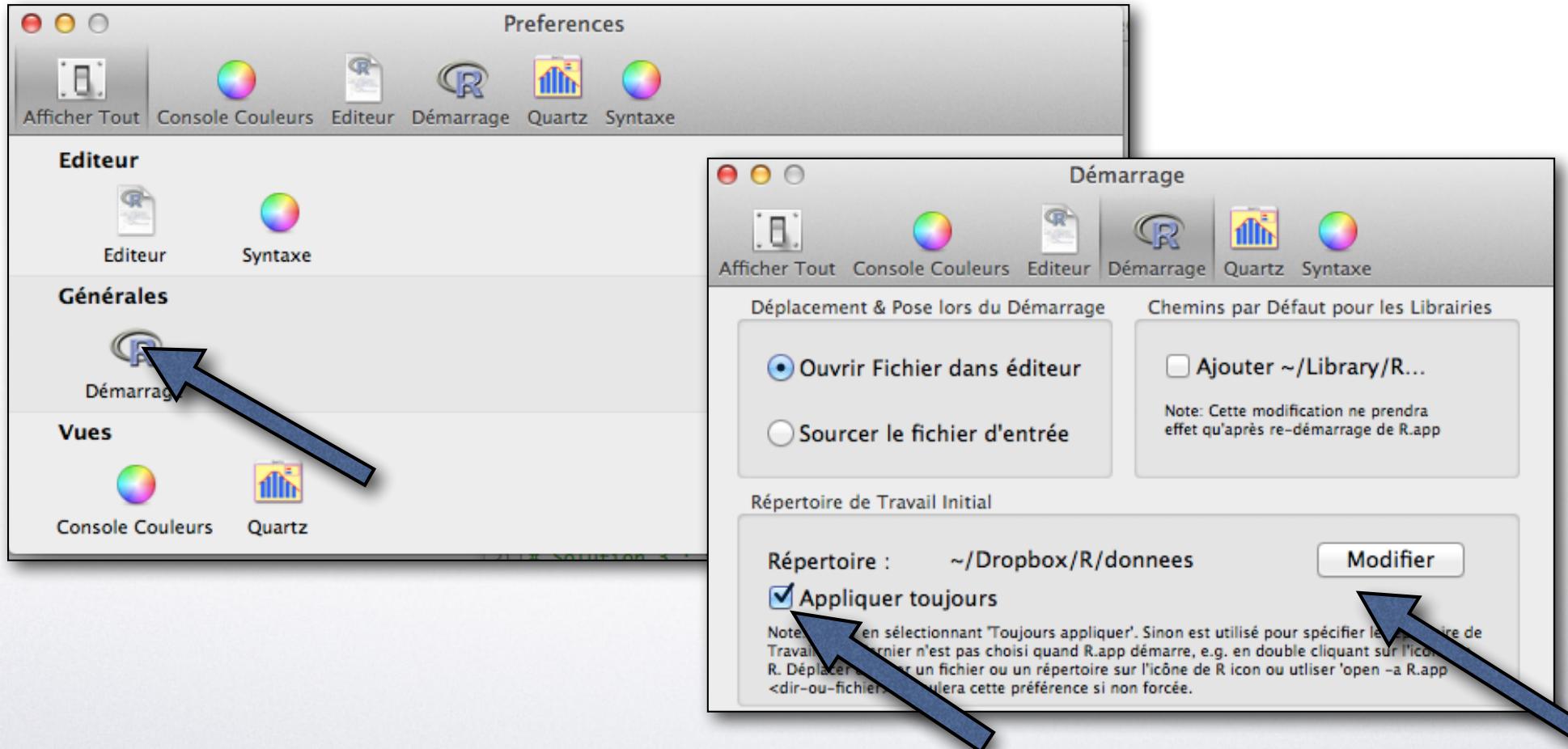


Clic droit sur l'icône R



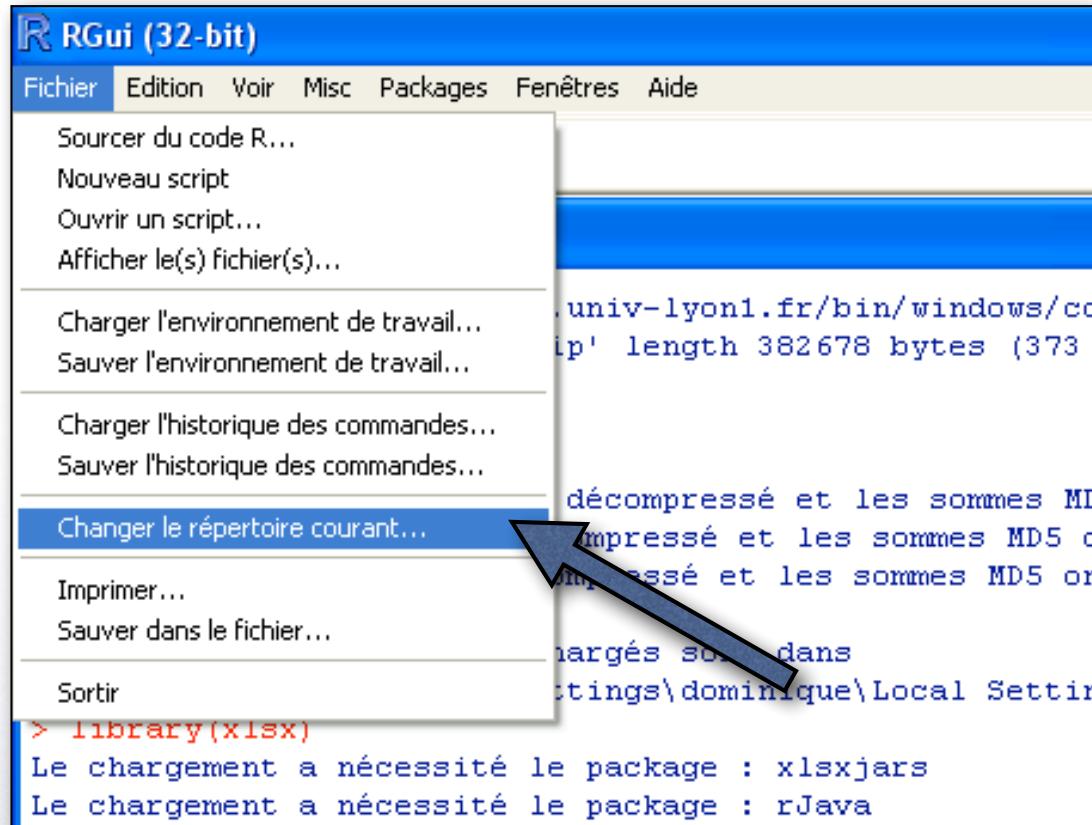


# Pour utiliser un répertoire fixe (Mac)



Attention, peut-être parfois nécessaire de redémarrer R ensuite

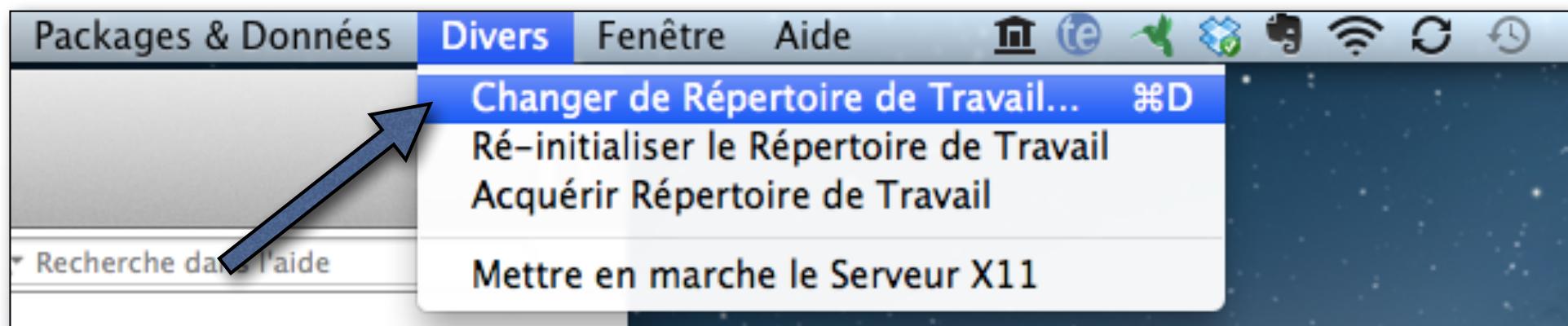
# Définir un répertoire temporaire (Windows)



Attention, n'apparaît que si la fenêtre active est la console



# Définir un répertoire temporaire (Mac)



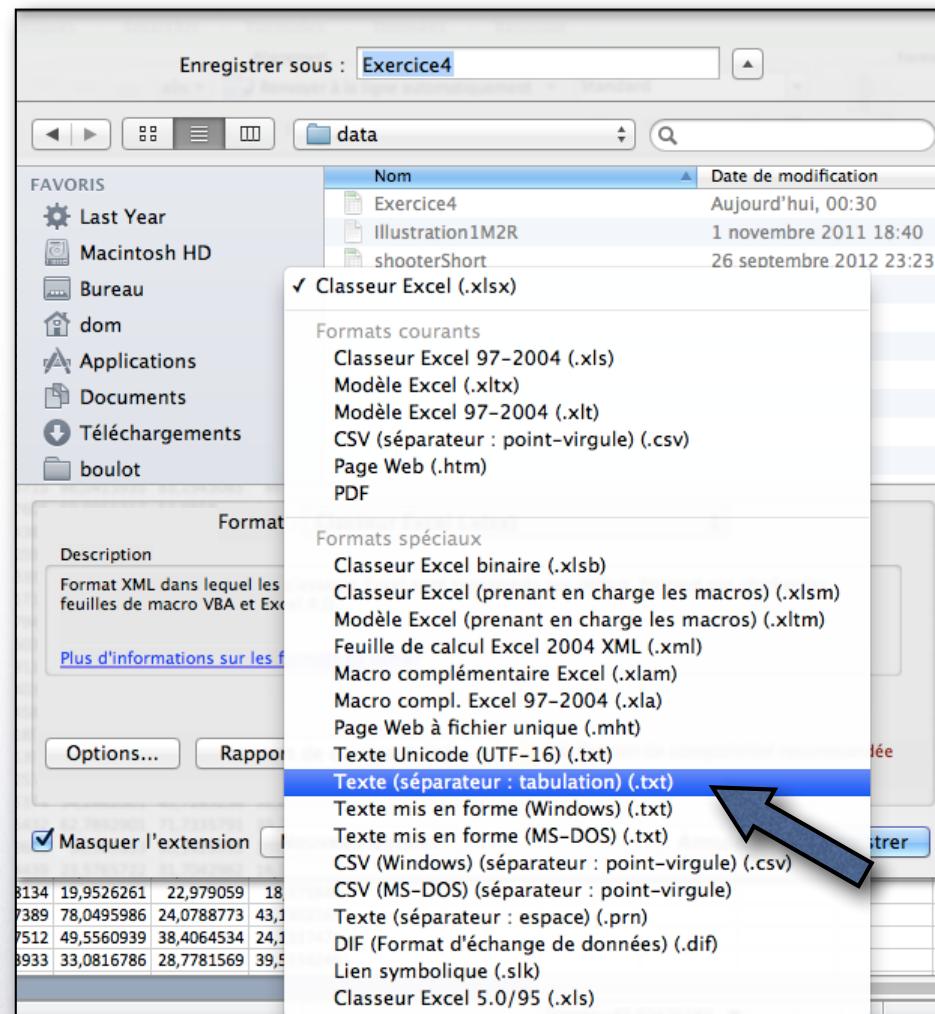
Attention, ici il ne faut pas redémarrer R

# Importation des données

- Sans package ("in built") avec enregistrement au format txt séparateur tabulation :

# Importation des données

Dans Excel :



# Importation des données

- Sans package ("in built") avec enregistrement au format txt séparateur tabulation :

```
DFt<-read.table("shooterShort.txt",header=TRUE,sep="\t",dec=",")
```

- "DFt<-" indique que nous souhaitons créer un objet s'appelant "DFt" (DF pour Data Frame)
- `read.table` est une fonction, qui sert à importer des données
- `header=TRUE` indique que la première ligne comporte le nom des variables
- `sep="\t"` indique l'utilisation d'un séparateur tabulation
- `dec=", "` indique que l'on utilise des virgules pour les décimales

# Importation des données

- Après avoir créé le data frame DFt, nous pouvons voir son contenu avec la commande `DFt`
- Pour les data frames un peu longs, on préféra utiliser la fonction `head()`

`head(DFt)`

```
pnum black_gun black_nogun white_gun white_nogun
1   1  542.5833    672.2000  551.4583   658.6667
2   2  604.0400    662.8750  587.3913   632.2000
3   3  546.1200    633.9167  536.0870   614.6400
4   4  536.6800    620.5600  539.9600   603.2000
5   5  571.7917    632.7826  586.1739   622.7391
6   6  533.6000    621.2800  566.6250   616.1304
```

Pour voir plus d'observations, on rajoutera l'Argument 2 : par exemple, `head(DFt, 10)`



# Importation des données

- Avec le package "readxl" (après installation et activation de celui-ci) :

```
DFx<-read_excel("shooterShort.xlsx",1)
```

- `1` indique que les données sont dans la feuille 1
- marche également avec le nom de la feuille entre guillemets

- Attention, cette commande produit un data frame particulier qu'on appelle un "tibble"

# Importation des données

- Ainsi, si on demande à voir le contenu, nous avons :

```
# A tibble: 36 x 5
  pnum black_gun black_nogun white_gun white_nogun
  <dbl>     <dbl>      <dbl>     <dbl>      <dbl>
1     1  542.5833   672.2000  551.4583  658.6667
2     2  604.0400   662.8750  587.3913  632.2000
3     3  546.1200   633.9167  536.0870  614.6400
4     4  536.6800   620.5600  539.9600  603.2000
5     5  571.7917   632.7826  586.1739  622.7391
6     6  533.6000   621.2800  566.6250  616.1304
7     7  562.4000   645.9545  576.9167  642.9583
8     8  589.9200   678.9200  635.3913  637.5600
9     9  529.5238   652.8421  573.8696  638.9545
10    10  547.4091   665.7273  564.6250  639.2083
# ... with 26 more rows
```

Avec plus de colonnes, celles-ci ne seront pas présentées, mais listées après « with 26 more rows »

- Pour revenir un format data frame classique, nous pouvons utiliser :

```
DFx<-as.data.frame(DFx)
```

# Importation des données

L'importation ne fonctionne pas :

- Vérifiez qu'il n'y a pas de faute de frappe dans la formule, attention notamment au nom du fichier (extension comprise), sachant que R est sensible à la case.
- Vérifiez que le fichier est bien enregistré dans le répertoire de travail (cela implique donc de bien repérer le répertoire de travail défini dans R et de vérifier que le fichier est bien dans ce répertoire).
- Vérifiez que le fichier ne comporte pas de caractères spéciaux problématiques (e.g., #, accents).

# Exercice 2

- Importez les données shooterShort.xlsx, tout d'abord avec la fonction `read.table` (nommez DFt ce data frame)
- Importez les données shooterShort.xlsx, avec la fonction `read_excel` (nommez DFx ce data frame)
- Importez les données IllustrationIM2R.txt (nommez `DFill1` ce data frame)
- Utilisez la fonction `mean()` pour essayer de calculer la moyenne de BEPC (l'une des variables de ce data frame)



# Grandes notions

- Fonctions
- Création et types d'objets
- Notion d'environnements



# Grandes notions

- Fonctions
- Création et types d'objets
- Notion d'environnements

# Notion d'environnements

- Un environnement est un ensemble d'objets
- Pour voir le contenu de l'Environnement I, on peut utiliser : `ls(1)`

```
> ls(1)
[1] "age"        "DFex1"       "DFill1"      "DFt"        "maMatrice"   "sujet"       "taille"
```

- On voit donc qu'il contient 7 objets

# Notion d'environnements

- Pour bien comprendre R, il est essentiel de comprendre sa structure
- Deux fonctions sont importantes pour cela : `search()` et `ls()`

```
> search()
[1] ".GlobalEnv"          "tools:RGUI"           "package:stats"      "package:graphics"
[5] "package:grDevices"   "package:utils"        "package:datasets"   "package:methods"
[9] "Autoloads"            "package:base"

> ls(1)
[1] "age"                 "DFex1"                "DFill1"              "DFT"                  "maMatrice"            "sujet"                "taille"
```

Notons que `ls(1)`, `ls(".GlobalEnv")` et `ls()` donnent le même résultat, c'est-à-dire le contenu de l'Environnement I (`GlobalEnv`)

# Notion d'environnements

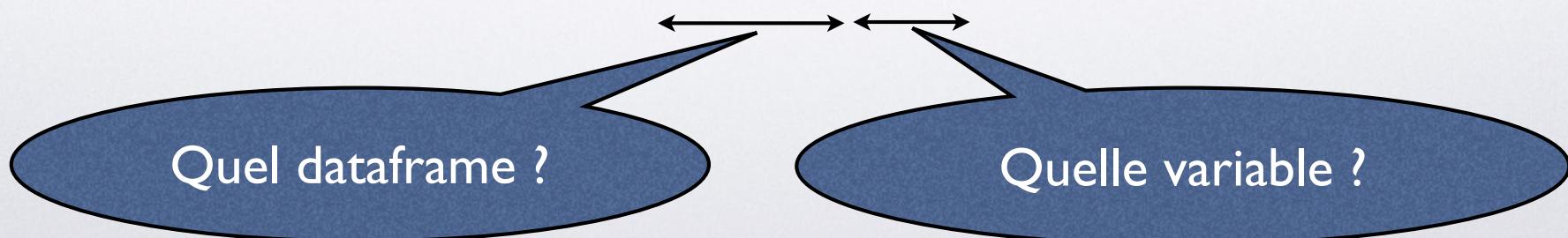
- Pour faire le ménage dans l'environnement de travail : `remove()` ou `rm()`

```
> rm(sujet,age,taille,maMatrice)
> ls(1)
[1] "DFex1"  "DFill1" "DFT"
```

- Pourquoi `mean(BEPC)` ne marche-t-il pas ?

Parce que R ne trouve pas un objet s'appelant BEPC. R ne descend pas au niveau du contenu des dataframes, sauf si on lui demande :

`mean(DFill1$BEPC)`



# Notion d'environnements

- Il existe un moyen (parfois risqué) de ne pas avoir à préciser le DF dans lequel se trouve la variable => la fonction attach()

AVANT :

```
> search()
```

```
[1] ".GlobalEnv"           "tools:RGUI"          "package:stats"      "package:graphics"  
[5] "package:grDevices"    "package:utils"        "package:datasets"   "package:methods"  
[9] "Autoloads"            "package:base"
```

PUIS : `attach(DFill1)`

APRES :

```
> search()
```

```
[1] ".GlobalEnv"           "DFill1"             "tools:RGUI"          "package:stats"  
[5] "package:graphics"     "package:grDevices"  "package:utils"        "package:datasets"  
[9] "package:methods"      "Autoloads"          "package:base"
```

Maintenant, si nous utilisons `mean(BEPC)`, cela fonctionne, car **après** avoir regardé dans `.GlobalEnv`, R regarde dans `DFill1`

# Notion d'environnements

> `search()`

```
[1] ".GlobalEnv"      "DFill1"           "tools:RGUI"        "package:stats"  
[5] "package:graphics" "package:grDevices" "package:utils"     "package:datasets"  
[9] "package:methods"   "Autoloads"         "package:base"
```

- **APRES** avoir cherché dans `.GlobalEnv`, R ira chercher dans `DFill1`
- **ATTENTION**, l'environnement `DFill1` est la copie de `DFill1` au moment t où nous utilisons la fonction `attach()`. Voilà d'ailleurs ce qui se passe si on réutilise la même fonction :

> `attach(DFill1)`

The following object(s) are masked from 'DFill1 (position 3)':  
BEPC, Pourcentage, pp

> `search()`

```
[1] ".GlobalEnv"      "DFill1"           "DFill1"           "tools:RGUI"  
[5] "package:stats"    "package:graphics" "package:grDevices" "package:utils"  
[9] "package:datasets" "package:methods"  "Autoloads"         "package:base"
```

- Pour éviter ces désagréments : toujours faire suivre un `attach()` par un `detach()`. Ici il faudrait utiliser deux fois `detach(DFill1)` pour revenir à l'état initial.



# Exercice 3

- Manipulez les fonctions search, ls et attach pour vérifier que vous avez compris ce que nous venons de voir

# Quelques connecteurs

- Arithmétiques : +, -, \*, /, ^, sqrt(), log(), exp()
- Relationnels : ==, <, >, <=, >=, != (différent)
- Logiques : &, | (ou) # sur Mac : alt + shift + L

# Adressage

- A concevoir comme les indices i et j dans  $DF_{i,j}$  c'est-à-dire par exemple  $DF_{1,3}$
- donc : `DF[ligne, colonne]`
- On peut utiliser le numéro de colonne ou le nom de la variable :  
`DF[1, 2]` ou `DF[1, "BEPC"]`
- `DF[1]` indiquera également la colonne 1, mais :
  - `DF[, 1]` => vecteur
  - `DF[1]` => data frame
- `DF[1, ]` indiquera la ligne 1 (format data frame)

# Utilisation adressage

- Pour créer un nouveau DF avec une sélection de variables :
  - En gardant uniquement BEPC : `DFbepc<-DF["BEPC"]` ou `DFbepc<-DF[2]`
  - Si nous voulons garder plus d'une variable : `DFbepc<-DF[c("pp", "BEPC")]`
- Pour créer un nouveau DF avec une sélection d'observations :
  - En gardant uniquement observations 4 à 6 : `DFshort<-DF[4:6, ]`
  - En gardant uniquement le participant 4 : `DFshort<-DF[DF$pp==4, ]`
  - En gardant uniquement les participants 4 à 6 : `DFshort<-DF[DF$pp>=4 & DF$pp<=6, ]`
- Pour exclure des observations ou des participants :
  - Par exemple, les observations 2 et 4 : `DF<-DF[!row.names(DF) %in% c(2,4), ]`
  - Par exemple, les participants 2 et 4 : `DF<-DF[!DF$pp %in% c(2,4), ]`



# Exercice 3bis

Manipulez les différentes commandes que nous venons de voir à partir du DFILL (Exercice 2)

# Utilisation adressage

- Pour remplacer une valeur spécifique dans le DF
  - Pour remplacer tous les temps de réponse > à 2000 par 2000 :  
`DF$RT[DF$RT>2000]<-2000`
  - S'il y a eu une erreur de saisie pour le participant 4 dont la note au BEPC était en réalité de 13,9886 :  
`DF$BEPC[DF$pp==4]<-13.9886`
- Pour changer l'ordre d'un DF
  - Si on veut classer par ordre ascendant de note au BEPC :  
`DF<-DF[order(DF$BEPC),]`
  - Si on veut classer par ordre descendant de note au BEPC :  
`DF<-DF[order(-DF$BEPC),]`

# Sélection d'observations

- Outre l'adressage, nous pouvons aussi utiliser la fonction subset :
  - Pour garder uniquement des temps de réponses > 200 ms :

```
DF<-subset(DF, RT >= 200)
```
  - Pour exclure les réponses à la phase d'entraînement et les essais "catch" :

```
DF<-subset(DF,phase != "training" & Val != "Catch")
```

# Création de variables

- Quand nous créons une variable, deux questions se posent :
  - Où sont la ou les variables utilisées pour créer la nouvelle variable ?
  - Où voulons-nous créer la nouvelle variable ?
- C'est pourquoi une formule comme `NewVar <- BEPC * 2` n'est pas une bonne formule a priori
  - Nous pourrions utiliser `NewVar <- DF$BEPC * 2` mais NewVar serait créée en dehors de DF
  - Il faut donc préciser à partir de quoi et où :  
`DF$NewVar <- DF$BEPC * 2`

# Création de variables

- La technique précédente fonctionne, mais elle est lourde quand nous voulons créer plusieurs variables. Nous allons donc utiliser une fonction, la fonction "within" qui indique une fois pour toutes "où trouver les variables ?" et "où les créer ?"

```
DF <- within(DF, {  
})
```

Tout ce qui sera créé entre les deux accolades sera "within" DF

# Création de variables

- Par exemple :

```
DF <- within(DF,{  
  # Création d'un contraste pour la variable genre  
  genrec <- -0.5*(gender == "f") + 0.5*(gender == "m")  
  # Inversion d'un score (item contre trait)  
  q3R <- 6 - q3  
  # Création d'un score moyen  
  score<-rowMeans(cbind(q1,q2,q3R,q4),na.rm=TRUE)  
  # Création d'une forme centrée à la moyenne pour la variable VIcont  
  VIcontc <- scale(VIcont,scale=FALSE)  
  # Création d'une forme centrée réduite de la variable VIcont  
  VIcontz <- scale(VIcont)  
})
```

# Création de variables

- Par exemple :

```
DF <- within(DF, {  
  
# Création d'un contraste pour la variable genre  
genrec <- -0.5*(gender == "f") + 0.5*(gender == "m")  
  
})
```

Notons que dans cette formule (qui marche dans quasiment tous les logiciels--R, SAS, SPSS, Statistica, Excel), on remplace donc "f" par -0.5 et "h" par 0.5

# Création de variables

- Par exemple :

```
DF <- within(DF, {  
  
  # Création d'un score moyen  
  score<-rowMeans(cbind(q1,q2,q3R,q4),na.rm=TRUE)  
  
})
```

- Notons que dans cette formule, on crée une nouvelle variable qui est la moyenne de q1, q2, q3R et q4.
- rowMeans est donc une fonction qui crée pour chaque ligne (d'où le "row") la moyenne de plusieurs variables.
- na.rm=TRUE permet d'indiquer à R de ne pas renvoyer NA dès qu'il existe une seule valeur manquante (argument qu'on retrouve souvent)

# Equivalent d'un tableau croisé dynamique

- A partir d'un format déplié (c'est-à-dire une ligne par réponse d'un sujet et non une ligne par sujet), nous voulons parfois parvenir à un format court avec une ligne par sujet.
- Pour ce faire, nous allons utiliser le package "reshape2" et la fonction "dcast"
- Par exemple (fichier "illustrationArmes.R") :

pp	phase	trialnb	Val	CueType	Stimuli	ACC	RT
18	10	phase1	18	val	Weapon	Gun2	1 351
19	10	phase1	19	val	Control	Light2	1 426
21	10	phase1	21	val	Control	Centre3	1 306
23	10	phase1	23	inv	Weapon	Grenade2	1 328
26	10	phase1	26	val	Weapon	Kalach3	1 453
27	10	phase1	27	val	Control	Whisk2	1 322
28	10	phase1	28	val	Control	Whisk4	1 343
29	10	phase1	29	inv	Weapon	Knife2	1 405

# Equivalent d'un tableau croisé dynamique

- A partir d'un format déplié (c'est-à-dire une ligne par réponse d'un sujet et non une ligne par sujet), nous voulons parfois parvenir à un format court avec une ligne par sujet.
- Pour ce faire, nous allons utiliser le package "reshape2" et la fonction "dcast"
- Par exemple (fichier "illustrationArmes.R") :

```
DFcourt<-dcast(DF3,pp~CueType+Val,value.var="RT",mean,na.rm=TRUE)
```

- `DF3` indique le DF d'origine (`DF3` dans cet exemple)
- `pp~CueType+Val` indique que nous voulons une ligne par participant (`pp`) et qu'en colonne nous voulons un croisement entre `CueType` et `Val`
- `value.var="RT"` indique que la VD est "RT"
- `mean` indique que nous voulons que soit calculé la moyenne (nous aurions pu demander la médiane), et `na.rm=TRUE` que les NA n'empêchent pas le calcul.

# Statistiques descriptives

- Si toutes les variables nous intéressent, nous pouvons commencer par la fonction **summary** (voir également la fonction « **mean** » si seule la moyenne nous intéresse).
- Par exemple :

```
summary(DFcourt)
```

Control_inv	Control_val	Weapon_inv	Weapon_val
Min. :393.7	Min. :385.7	Min. :433.3	Min. :355.5
1st Qu.:477.4	1st Qu.:428.0	1st Qu.:484.2	1st Qu.:420.7
Median :515.9	Median :453.6	Median :532.7	Median :441.0
Mean :526.5	Mean :477.1	Mean :544.6	Mean :466.9
3rd Qu.:550.4	3rd Qu.:516.7	3rd Qu.:601.0	3rd Qu.:504.2
Max. :716.6	Max. :658.4	Max. :711.5	Max. :636.7

- Si une seule nous intéresse, nous pouvons spécifier laquelle :

```
summary(DFcourt$Control_inv)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
393.7	477.4	515.9	526.5	550.4	716.6

# Statistiques descriptives

- Pour calculer les moyennes par condition quand nous avons une variable inter, nous pouvons utiliser la fonction tapply

- Par exemple (illustration avec l'étude "workshop" et après avoir utilisé un attach(DF)) :

```
tapply(score,workshop,mean,na.rm=TRUE)
```

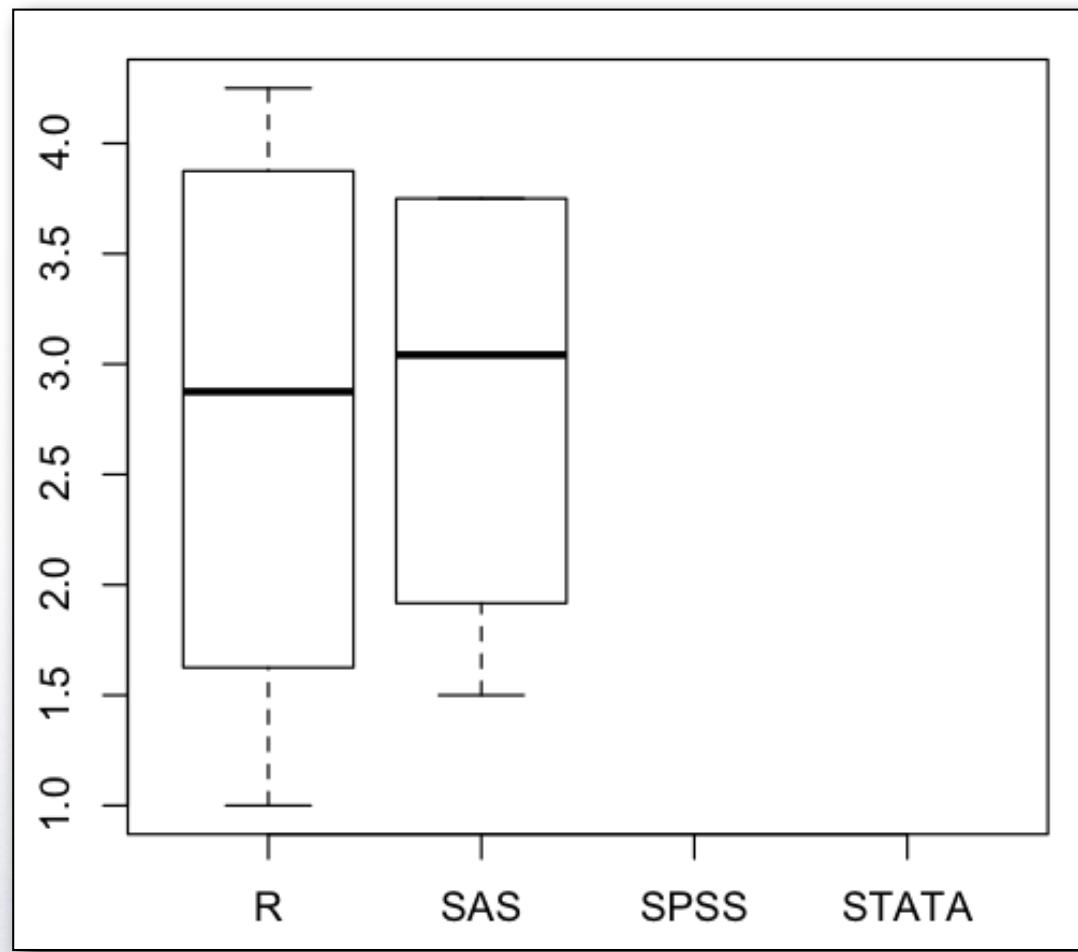
R	SAS	SPSS	STATA
2.750000	2.833333	NA	NA

- La fonction tapply applique sur k groupes définis par l'Argument 2 (ici workshop), la fonction définit par l'Argument 3 (ici la moyenne avec son argument na.rm=TRUE), avec comme VD ce que nous indiquons en Argument 1.

- Pour obtenir le boxplot correspondant, nous pouvons utiliser :

```
boxplot(score~workshop)
```

# Statistiques descriptives



# Statistiques descriptives

- Nous pouvons également utiliser la fonction `tapply` pour obtenir les moyennes correspondant au croisement de deux variables.
- Par exemple (illustration avec l'étude "workshop" et après avoir utilisé un `attach(DF)`) :

```
tapply(score, list(workshop, gender), mean, na.rm=TRUE)
```

	f	m
R	1.625	3.875
SAS	1.500	3.750
SPSS	NA	NA
STATA	NA	NA

# Statistiques descriptives

- Pour calculer les moyennes par condition quand nous avons deux variables inter (ou une seule) et plusieurs niveaux en intra nous pouvons utiliser la fonction aggregate
- Par exemple (illustration avec l'étude "workshop" et après avoir utilisé un `attach(DF)`) :

```
aggregate(cbind(q1,q2,q3R,q4),by=data.frame(workshop,gender),mean,na.rm=TRUE)
```

	workshop	gender	q1	q2	q3R	q4
1	R	f	1.5	1.5	1.5	2
2	SAS	f	2.0	1.0	2.0	1
3	R	m	4.5	4.0	3.0	4
4	SAS	m	4.5	4.5	1.0	5

# Statistiques inférentielles

- Faisons maintenant une régression avec le genre comme prédicteur (codé en -0,5 et 0,5) et le score comme VD
- Avant de faire cette régression, nous utilisons la fonction `outliers` pour vérifier que nous n'avons pas de sujets déviants. Cette fonction n'existe pas dans R, ni dans les packages, on placera donc le fichier "outliersFunction" dans le répertoire de travail et on activera cette fonction avec la commande :

```
source("outliersFunction.R")
```

- Ensuite, nous utilisons donc :`outliers(score ~ genderc, DF)`

	score	genderc	sdr	fstar	cookd	leverage
3	2.25	-0.5	2.51	6.30	0.77	0.33
1	1.00	-0.5	-1.88	3.53	0.59	0.33
5	4.25	0.5	1.11	1.23	0.20	0.25
7	3.50	0.5	-0.74	0.55	0.10	0.25
2	1.50	-0.5	-0.20	0.04	0.01	0.33

- Notons que nous pouvons également classer par le levier ou le D de cook (cf fichier `IllustrationSimple`).

# Statistiques inférentielles

- Faisons maintenant une régression avec le genre comme prédicteur (codé en -0,5 et 0,5) et le score comme VD
- Pour la régression proprement dite, nous utilisons la fonction `lm` (pour linear model)  
`fit1<-lm(score ~ genderc, DF)`
- Pour l'instant, nous avons juste créé l'objet `fit1` qui est un objet de class "lm"
- Si nous demandons à voir ce qu'il y a dans `fit1`, nous obtenons :

Call:

```
lm(formula = score ~ genderc, data = DF)
```

Coefficients:

(Intercept)	genderc
2.698	2.229

- Mais cet objet `fit1` contient bien plus que ça. Pour avoir accès aux autres informations, nous devons utiliser d'autres fonctions.



# Statistiques inférentielles

- Pour obtenir le résultat habituel d'une régression, nous allons utiliser la fonction `summary` :  
`summary(fit1)`

Call:

```
lm(formula = score ~ genderc, data = DF)
```

Residuals:

1	2	3	5	6	7	8
-0.58333	-0.08333	0.66667	0.43750	-0.06250	-0.31250	-0.06250

Coefficients:

	Estimate	Std. Error	t value	Pr(> t )
(Intercept)	2.6979	0.1782	15.141	2.28e-05 ***
genderc	2.2292	0.3564	6.255	0.00153 **

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.4666 on 5 degrees of freedom

(1 observation deleted due to missingness)

Multiple R-squared: 0.8867, Adjusted R-squared: 0.864

F-statistic: 39.13 on 1 and 5 DF, p-value: 0.001531

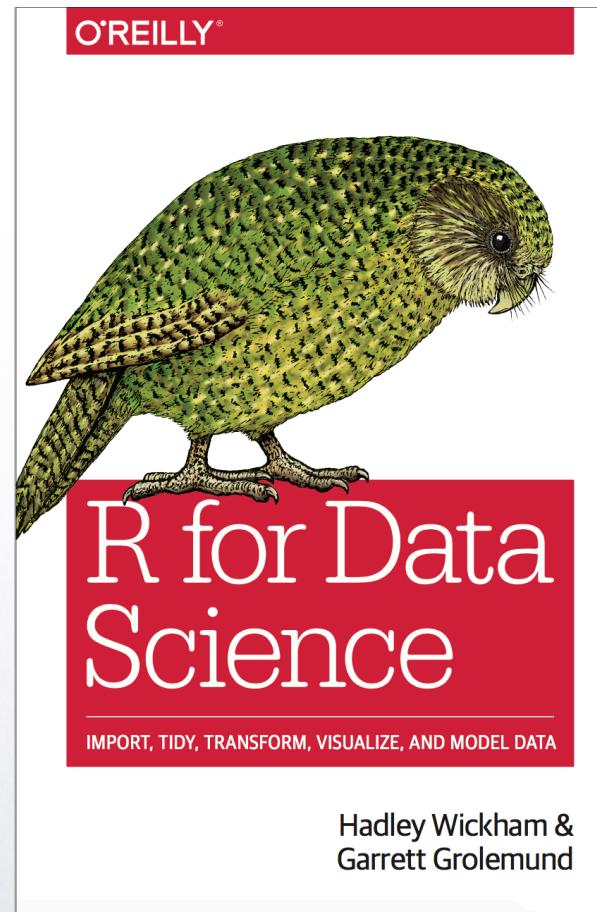
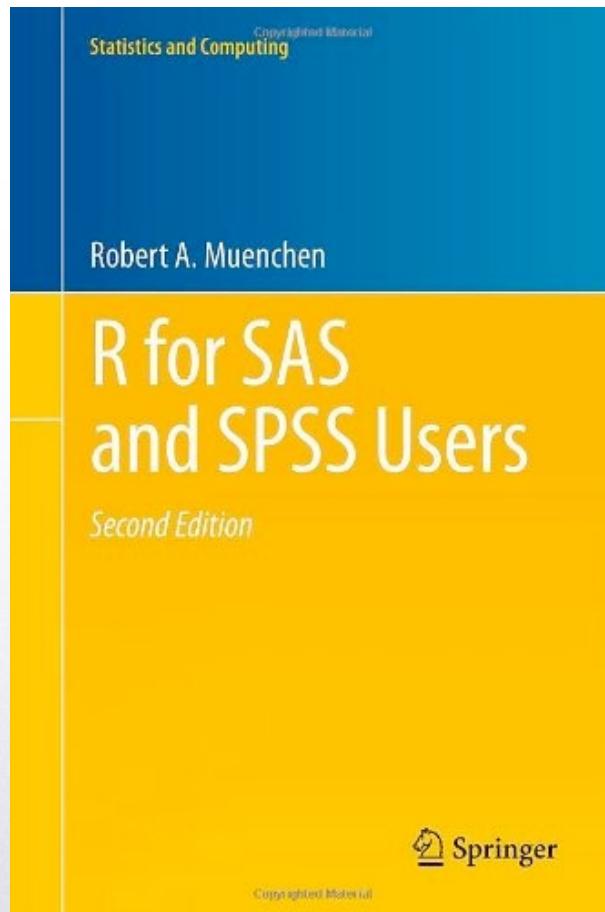
# Exercice 4

- Créez un DF correspondant aux données de “Exercice 4.xlsx”
- Retirez de ce DF toutes les valeurs de vd1, vd2, vd3 et vd4 qui seraient supérieures à 100.
- Créez les variables suivantes :
  - catc, soit un recodage en -0,5 et 0,5 pour la variable cat
  - score, soit un score moyen reprenant vd1, vd2, vd3 et vd4
- Calculez les moyennes de score par condition
- Faites un box plot représentant le score en fonction des conditions expérimentales
- Réalisez une régression avec score comme vd et catc comme prédicteur

# Exercice 5

- Créez un DF correspondant aux données de shooterLong
- Retirez de ce DF les observations correspondant à la phase 0 (variable "phase") et gardez uniquement les réponses correctes (variable "ResponseLabel").
- Créez les variables suivantes :
  - racec, soit un recodage en -0,5 et 0,5 respectivement pour "black" et "white"
  - objectc, soit un recodage en -0,5 et 0,5 respectivement pour "gun" et "nogun"
- Créez un "tableau court" en faisant l'équivalent d'un tableau croisé dynamique. Ce tableau fera apparaître les numéros de participants (pnum) et une colonne par condition du croisement 2 (race : black vs white) \* 2 (object : gun vs nogun) en utilisant comme VD la variable Time.
- Calculez les moyennes par condition
- Oups, vous deviez utiliser non pas les temps de réaction, mais les temps de réaction avec transformation logarithmique, recommencez tout ce qui est nécessaire.

# Pour aller plus loin



[r4ds.had.co.nz](http://r4ds.had.co.nz)