Plan de la présentation

Qu'y a-t-il avant R ?

Il y a S, et non pas Q...

→ S voit le jour en 1976 des équipes du laboratoire Bell, dont la version SPlus de 1987 est commercialisée.

→ S est un langage de programmation, à la différence de logiciels statistiques tels que Stata, SAS, ...

→ R est le logiciel libre, développé par de nombreux statisticiens de par le monde et validé par une équipe agréée.

→ R permet de manipuler des objets et de développer ses propres outils.

→ De nombreux logiciels permettent aujourd'hui d'interfacer R: Ggobi (outil graphique), RExcel...

→ Améliore la flexibilité et évite la "boîte noire".

Les sites web pour travailler sous R

On dénombre un grand nombre de sites Internet offrant une source d'information variée et relativement complète:

→ site principal: www.r-project.org/

→ Librairies: cran.r-project.org/, r-forge.r-project.org/

→ Tutoriels: www.r-tutor.com/, www.statmethods.net/

→ Graphiques: http://addictedtor.free.fr/graphiques/

→ FAQs: forums.cirad.fr/logiciel-R/index.php, www.mail-archive.com/r-help@r-project.org/, r.789695.n4.nabble.com/, tolstoy.newcastle.edu.au/R/

→ Pour les actuaires! www.actuar-project.org/, vgoulet.act.ulaval.ca/

L'information y est plutôt fiable, et des exemples intéressants y sont souvent donnés!

Un langage en pleine expansion

R est extrêmement bien documenté:

→ pour débutants

- Paradis (2006). R pour débutants
- Goulet (2004). Introduction à la programmation en S.

→ économétrie: beaucoup de doc, dont:
Farnsworth (2008). Econometrics in R (sur CRAN).

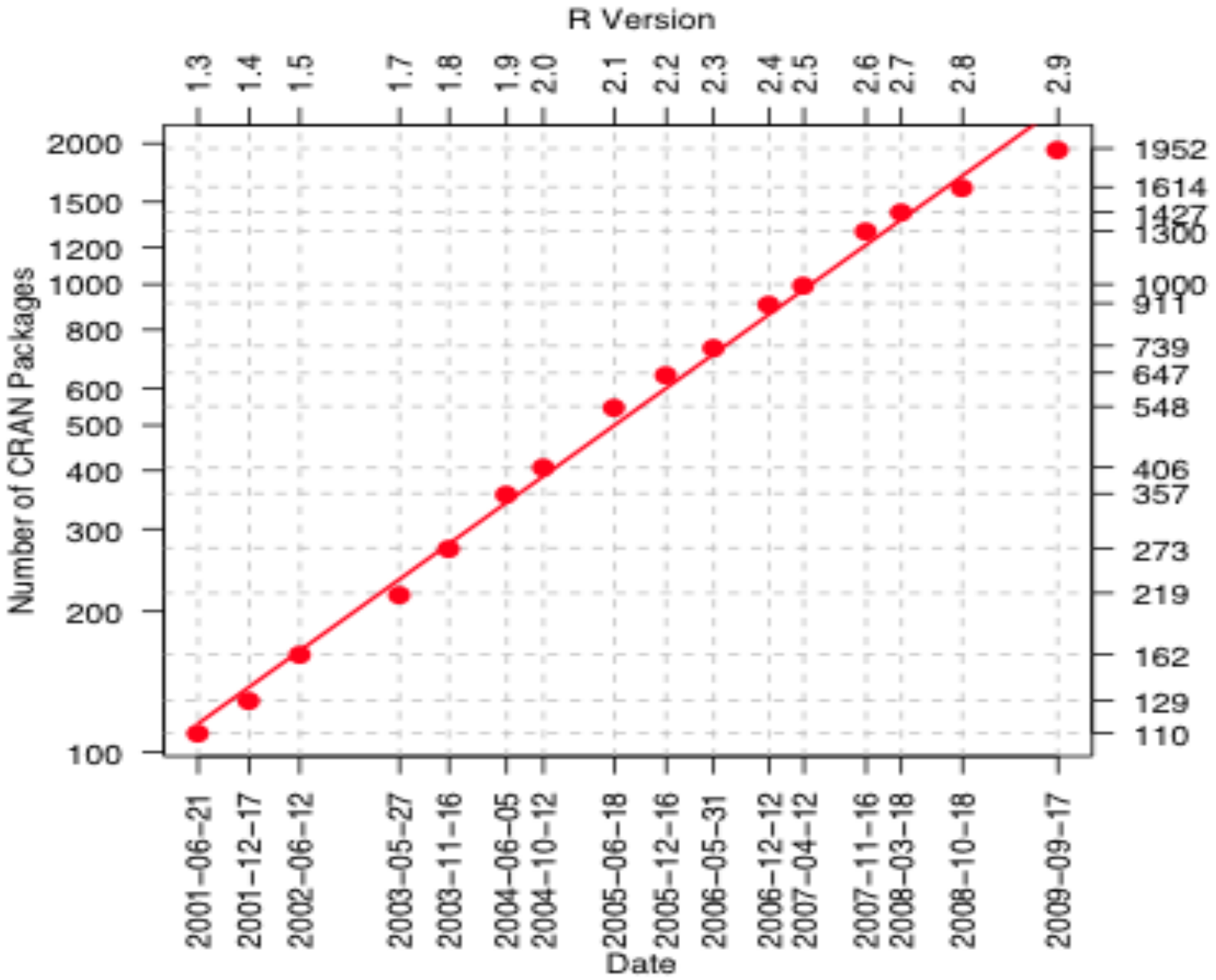
→ condensé de commandes: les refcards! (sur CRAN)

→ conférences: useR! (2008, 2009, 2010, 2011, 2012)

On constate une explosion dans le nombre d'utilisateurs et de développeurs de ce logiciel open source.

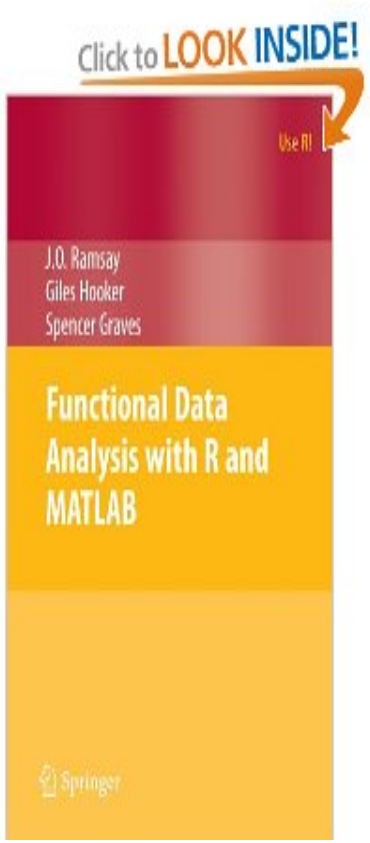
Les programmes sont validées par des équipes de chercheurs (R Core Team) assurant la validité du code.

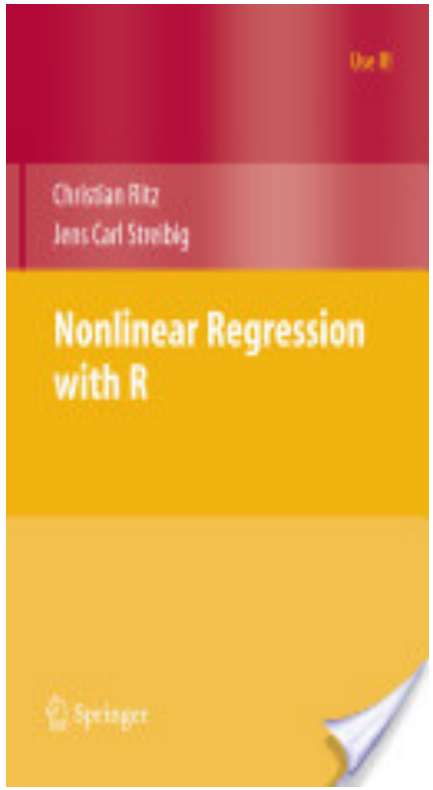
Un langage en pleine expansion (2)



Un langage en pleine expansion (3)

Un large panel de livres est en train de se développer:





Que permet de faire R?

A peu près toutes les tâches que peut vouloir faire un statisticien, à savoir:

→ Importation et exportation de données,

→ statistiques descriptives,

→ analyse de données,

→ modélisation statistique,

→ tests d'hypothèses,

→ représentations graphiques.

Voir aussi les "Task Views" qui permettent d'avoir une vision à jour des derniers développements dans chaque domaine:

http://cran.r-project.org/web/views/

Avantages et inconvénients de R

Les points forts:

→ gratuité,
→ portabilité (Windows, Unix, OS X...)
→ à jour des derniers développements de la Recherche
→ calculs rapides (coeur en C),
→ représentations graphiques de qualité,
→ code informatique très général,
→ langage objet: méthodes et attributs.

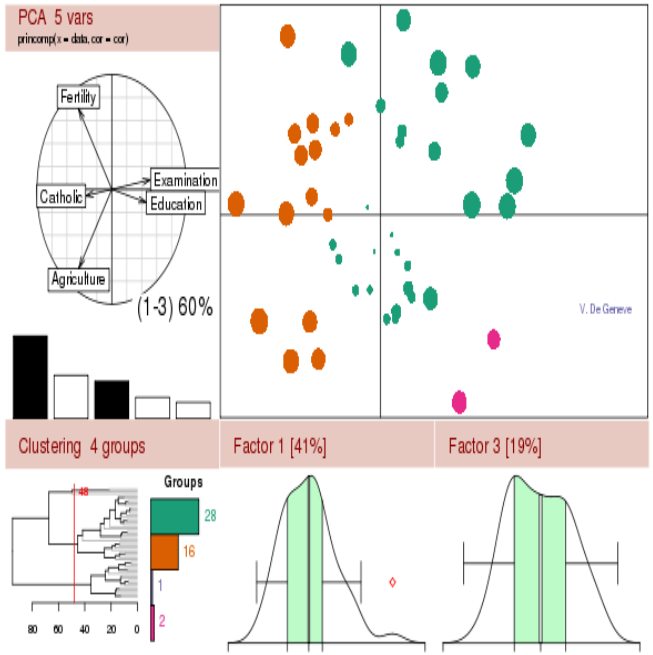
Les points faibles:

→ débuggage pas toujours évident,
→ compatibilité entre librairies,
→ taille limitée des données.

En résumé, il y a beaucoup plus de points forts!

Téléchargement de R: www.r-project.org/

The R Project for Statistical Computing



About R
What is R?
Contributors
Screenshots
What's new?

Download, Packages
CRAN

R Project
Foundation
Members & Donors
Mailing Lists
Bug Tracking
Developer Page
Conferences
Search

Documentation
Manuals
FAQs
The R Journal
Wiki
Books
Certification
Other

Misc
Bioconductor
Related Projects
User Groups
Links

Getting Started:

- R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To download R, please choose your preferred CRAN mirror.
- If you have questions about R like how to download and install the software, or what the license terms are, please read our answers to frequently asked questions before you send an email.

News:

- R 2.15.0 prerelease versions are currently available. Final release is scheduled for March 30.
- **R version 2.14.2** (Gift-Getting Season) has been released on 2012-02-29.
- **R version 2.14.1** (December Snowflakes) has been released on 2011-12-22.
- The R Journal Vol.3/2 is available.
- useR! 2012, will take place at Vanderbilt University, Nashville Tennessee, USA, June 12-15, 2012.

Paramètres par défaut

Modifier les paramètres initiaux de lancement de R:

→ le fichier "Rprofile.site", situé au répertoire ".../R/src/gnuwin32/fixed/etc", permet de modifier ces paramètres. Exemple: `options(prompt=" ")`;

→ répertoire de travail: important lors de l'importation de scripts R: `getwd()` et `setwd()`;

→ informations sur l'environnement: `Sys.getenv("R_USER")`, `Sys.putenv(R_USER=)`, `Sys.setenv(R_USER=)`.

→ sauvegarde de la session: `save.image()` et `save.history()`: enregistre respectivement les objets dans les fichiers ".RData" et les instructions dans ".RHistory".

Rq: souvent vous créez un répertoire pour un projet particulier, `setwd(...)` sera donc la plus utile...

Ouverture et fermeture de l'application

Pour ouvrir R, lancer l'application correspondante via le dossier où le programme a été installé...
Pour quitter, tapez `q()` dans la console.

```
R version 2.14.0 (2011-10-31)
Copyright (C) 2011 The R Foundation for Statistical Computing
ISBN 3-900051-07-0
Platform: x86_64-apple-darwin9.8.0/x86_64 (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.

[R.app GUI 1.42 (5933) x86_64-apple-darwin9.8.0]

[Historique recherché depuis /Users/xaviermilhaud/.Rhistory]

>
```

Les différentes fenêtres

Plusieurs fenêtres peuvent apparaitre à l'écran: en fonction de l'action de l'utilisateur, la fenêtre active peut être:

→ la console: fenêtre de l'application, par défaut à l'ouverture et toujours ouverte lorsque R tourne,

→ fenêtre graphique,

→ script: fichier "texte" contenant du code R.

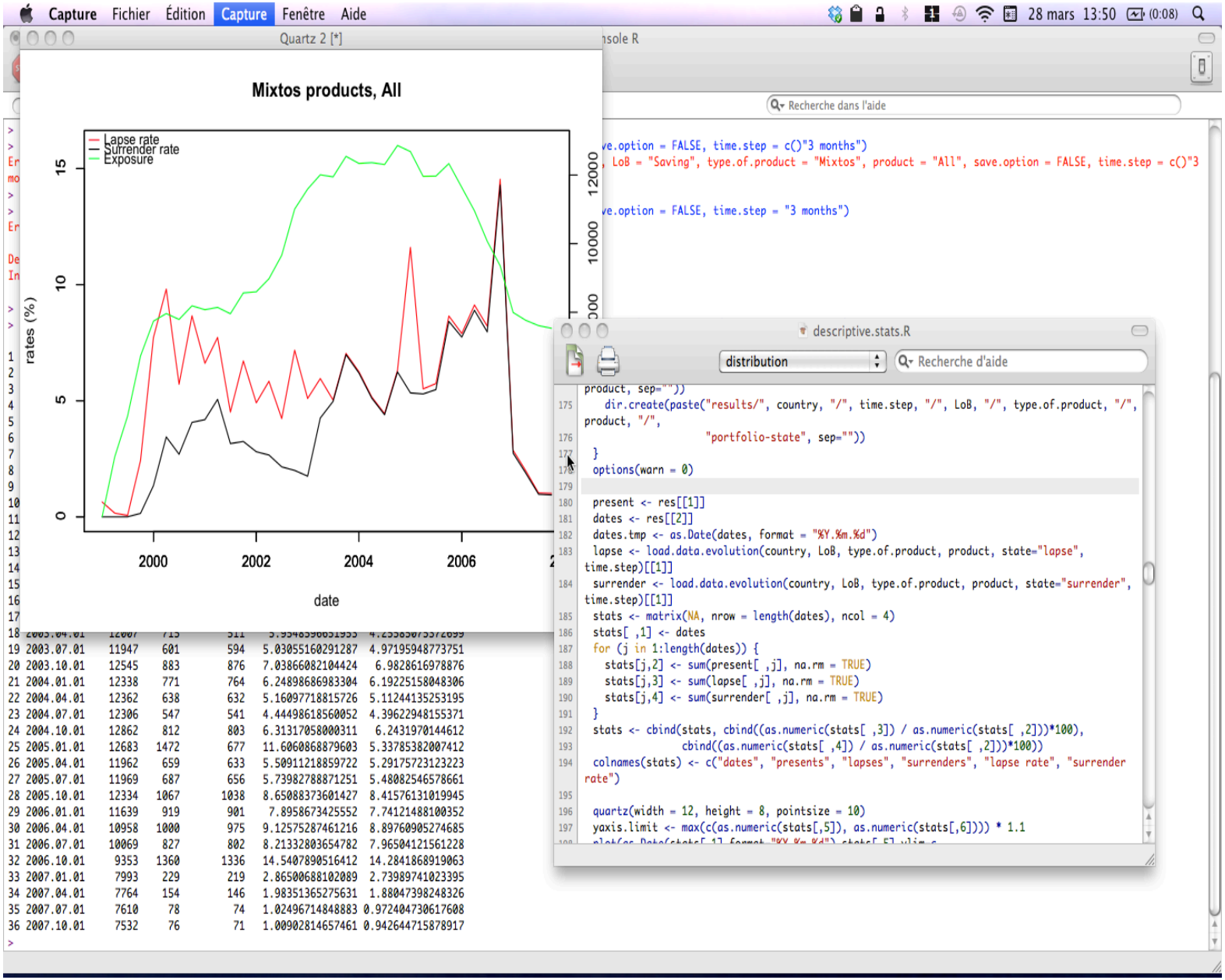
Toutes les types d'opérations sont programmables en R, et ce pour chaque fenêtre:

→ ouverture, fermeture et enregistrement;

→ lecture, écriture, exécution.

Rq: il y a un nombre limité de fenêtres graphiques (une quinzaine), donc les fermer (et/ou enregistrer) si nécessaire...

Types de fenêtre



Comment travailler sous R?

En mode "console", les commandes sont tapées directement dans le terminal:

- juste après le signe ">" ;
- "+" apparait en début de ligne si commande incomplète;

```
>
> portfolio.state(country = "Spain", LoB = "Saving", type.of.product = "Mixtos", product = "All", save.
Erreur dans file(filename, "r", encoding = encoding) :
  impossible d'ouvrir la connexion
De plus : Message d'avis :
In file(filename, "r", encoding = encoding) :
  impossible d'ouvrir le fichier 'load.data.R' : No such file or directory
>
> portfolio.state(country = "Spain", LoB = "Saving", type.of.product = "Mixtos", product = "All", save.
        dates presents lapses surrenders        lapse rate    surrender rate
1  1999-01-01     2025     12          0  0.592020620820620                 0
```

Caractéristiques de la console: objets réels et code virtuel.

Comment travailler sous R? (2)

En mode "script", pour l'écriture de code et/ou fonctions:

1. ouverture d'un fichier texte,
2. écriture/enregistrement du code R dans un fichier "*.R",
3. importation (`source(...)`) et exécution de ce code: appel de fonction ou Ctrl-R depuis le script.

auxiliary.functions.R

check.variable.type | Recherche d'aide

```
## This function removes from the input list of explanatory variables (for future glm formula) the ones
## corresponding to columns of 'myData' which were removed.
## Returns the new list of explanatory variables, updated and in line with the dataframe 'myData' considered.
remove.expl.var <- function(explanatory.var, myData)
{
  if (is.null(explanatory.var)) {
    return(NULL)
  } else {
    to.remove <- numeric()
    for (i in 1:length(explanatory.var)) {
      if (!(explanatory.var[i] %in% names(myData)))
        to.remove <- c(to.remove, i)
    }
    if (length(to.remove) != 0) {
      warning(paste("\n Following explanatory variables were removed since they are absent in the dataset:\n",
                    paste("  ", explanatory.var[to.remove], collapse=", "), "\n",sep = " "))
      explanatory.var <- explanatory.var[-to.remove]
    }
    explanatory.var
  }
}


## Function to load the credited rate corresponding to the product specified. Only available for certain spanish
## products. The product must be specified in string format (e.g. "824").
## Returns the credited rate evolution in function of time (dates).
load.credited.rate <- function(country, LoB, type.of.product,product,time.step,save.option=F,graph.option=F)
{
  if (country != "Spain") {
    warning("No available data for credited rates.")
    return(NULL)
  }
  stopifnot(is.character(product))

  myData <- read.table(paste("data/", country, "/credited-rate.txt", sep=""), header = TRUE, sep = "\t")
  #myData <- read.csv2(paste("data/", country, "/credited-rate.csv", sep=""), header = T)

  if (product != "All") {
```

Caractéristique du script: objets virtuels et code réel.

Les librairies ou "packages"

Multiples fonctionnalités pour effectuer des tâches précises, régulièrement optimisées. Pour les utiliser (correctement):

$\rightarrow$ les télécharger via CRAN ou un site mirroir,
$\rightarrow$ prendre connaissance de la documentation et des exemples (`vignette()`).

Au lancement de R, seules certaines librairies sont chargées automatiquement (celles dont *priority* ="base"):

- `installed.packages()`: retourne les librairies installées en local.
- `library()` retourne la liste des librairies installées.
- `library(lib)` ou `require(lib)` charge la librairie *lib*.
- `library(help = lib)` retourne la liste des fonctions de la librairie *lib*.
- `search()`, `searchpaths()` retourne la liste des librairies chargées.

Gestionnaire de librairies et barre d'outils

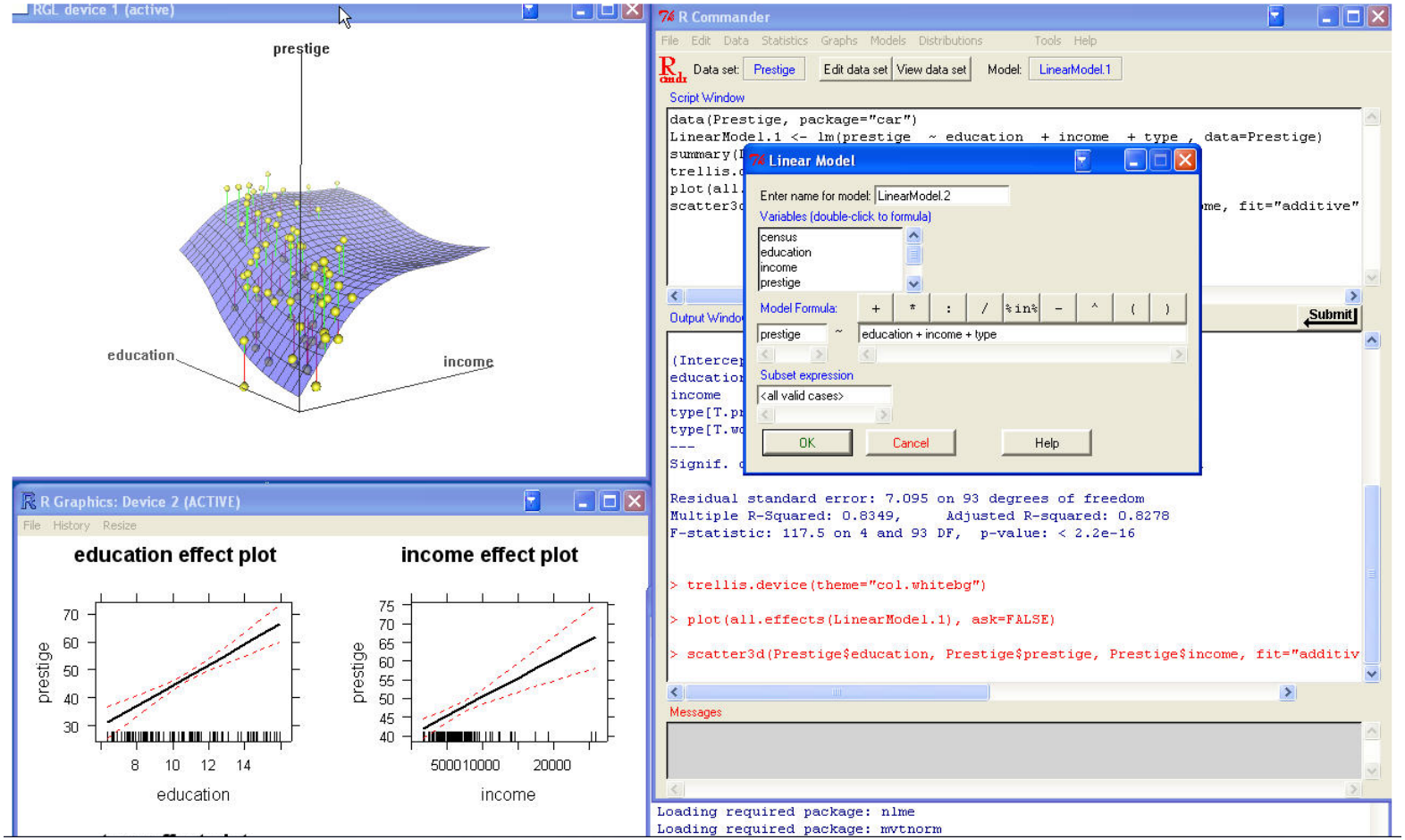
Gestionnaire de Package R

< Retour | Avancer > | Rafraîchir la liste

Statut	Package	Description
☐ non chargé	boot	Bootstrap Functions (originally by Angelo Canty for S)
☐ non chargé	class	Functions for Classification
☐ non chargé	cluster	Cluster Analysis Extended Rousseeuw et al.
☐ non chargé	codetools	Code Analysis Tools for R
☐ non chargé	compiler	The R Compiler Package
☐ non chargé	copula	Multivariate dependence with copulas
☑ chargé	datasets	The R Datasets Package
☐ non chargé	flexmix	Flexible Mixture Modeling
☐ non chargé	foreign	Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, dBase,
☑ chargé	graphics	The R Graphics Package
☑ chargé	grDevices	The R Graphics Devices and Support for Colours and Fonts
☐ non chargé	grid	The Grid Graphics Package
☐ non chargé	KernSmooth	Functions for kernel smoothing for Wand & Jones (1995)
☐ non chargé	lattice	Lattice Graphics
☐ non chargé	MASS	Support Functions and Datasets for Venables and Ripley's MASS
☐ non chargé	Matrix	Sparse and Dense Matrix Classes and Methods
☑ chargé	methods	Formal Methods and Classes
☐ non chargé	mgcv	GAMs with GCV/AIC/REML smoothness estimation and GAMMs by PQL
☐ non chargé	modeltools	Tools and Classes for Statistical Models
☐ non chargé	multcomp	Simultaneous Inference in General Parametric Models
☐ non chargé	mvtnorm	Multivariate Normal and t Distributions
☐ non chargé	nlme	Linear and Nonlinear Mixed Effects Models
☐ non chargé	nnet	Feed-forward Neural Networks and Multinomial Log-Linear Models
☐ non chargé	parallel	Support for Parallel computation in R
☐ non chargé	pspline	Penalized Smoothing Splines
☐ non chargé	rggobi	Interface between R and GGobi
☐ non chargé	RGtk2	R bindings for Gtk 2.8.0 and above
☐ non chargé	rpart	Recursive Partitioning
☐ non chargé	scatterplot3d	3D Scatter Plot
☐ non chargé	spatial	Functions for Kriging and Point Pattern Analysis
☐ non chargé	splines	Regression Spline Functions and Classes
☑ chargé	stats	The R Stats Package
☐ non chargé	stats4	Statistical Functions using S4 Classes
☐ non chargé	survival	Survival analysis, including penalised likelihood.
☐ non chargé	tcltk	Tcl/Tk Interface
☐ non chargé	tools	Tools for Package Development
☑ chargé	utils	The R Utils Package

Utilisation et interfaçage

R n'est pas très "user-friendly"... Mais il existe des développements pour pallier ce manque. Ex: "RCommander":



Permet d'utiliser les fonctionnalités de R sans forcément connaitre la syntaxe du code R sous-jacent.

Intéragir avec le système de fichiers

Un certain nombre de commandes permettent d'intéragir avec la gestion des fichiers. Il y a:

→ choix intéractif d'un fichier texte: `file.choose()`,

→ informations sur un fichier: `file.info()`,

→ contenu d'un fichier: `file.show()`,

→ destruction d'un fichier: `file.remove()`,

→ accès à un répertoire: `dirname()`,

→ contenu d'un répertoire: `dir()`.

Rq: la plupart du temps, ces commandes se font via l'interface (la barre d'outil), ce qui rend leur manipulation plus facile pour l'utilisateur.

Exercice 0

1. Créer un répertoire de travail temporaire sur votre bureau que vous appellerez “formationR-062012”;
2. Le définir comme répertoire de travail;
3. Vérifier le contenu de ce répertoire.

Solution:

Exercice 0

1. Créer un répertoire de travail temporaire sur votre bureau que vous appellerez “formationR-062012”;
2. Le définir comme répertoire de travail;
3. Vérifier le contenu de ce répertoire.

Solution:

```
## creer le repertoire
dir.create("/Users/xaviermilhaud/Desktop/FormationR-062012")
## definit le repertoire de travail
setwd("/Users/xaviermilhaud/Desktop/FormationR-062012")
## verification
getwd()
## chemin, nom et contenu du repertoire
dirname("/Users/xaviermilhaud/Desktop/FormationR-062012")
dir("/Users/xaviermilhaud/Desktop/FormationR-062012")
```

Focus sur les Task Views

Les Task Views sont à bien des égards un excellent vecteur d'information sur les outils développés en R en lien avec des besoins opérationnels concrets.

Elles fournissent:

→ un résumé sur la vocation de la Task View,

→ un ensemble de librairies utiles suivant un thème choisi,

→ une explication succinte des différences entre librairies,

→ le modérateur à contacter pour toute question.

Les sujets balayés sont divers et variés, et une attention particulière est donnée à une formalisation concrète des outils.

Sujets des Task Views

Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis & Finite Mixture Models
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
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
OfficialStatistics	Official Statistics & Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis

1. P-R-emier pas
   - Généralités
   - Fondamentaux
   - Manipulation
   - Graphiques
   - Probabilités
2. Statistiques desc-R-iptives
3. Exemples d'usages d'actuaire
4. Conclusion

Quelques remarques essentielles...

Les notions de base des commandes R à maîtriser:

$\rightarrow$ R est sensible à la casse,

$\rightarrow$ à chaque nouvelle affectation, R efface l'ancienne valeur de la variable redéfinie (ex: π).

$\rightarrow$ les noms d'objets peuvent contenir des lettres, des nombres, ".", "_".

$\rightarrow$ un nom de variable ne peut être un nombre,

$\rightarrow$ ";" pour séparer les commandes d'une même ligne.

Rq fondamentale: donner des noms explicites à vos variables et commenter bien votre code.

Rq fondamentale 2: Eviter de donner un nom commun qui pourrait correspondre à une commande de R.

Aperçu des noms de variables "interdits"

Comme évoqué précédemment, certains noms sont prédéfinis et non accessibles. On a notamment:

- l'objet nul: NULL (test par `is.null()`);
- les données manquantes: NA (cf `is.na()`, `na.omit`, ...);
- les nombres inconnus:
  → NaN,
  → Inf;
- les booléens:
  → TRUE
  → FALSE;
- les fonctions prédéfinies: `mean()`, `length()`, ...
- les mots-clefs: `break`, `if`, `case`...

→ Rq: un commentaire est annoncé par "#".

Evidemment cette liste est longue... Vigilance donc!

Demander de l'aide à R, il ne vous jugera pas!

Lorsqu'on est perdu, les commandes utiles sont:

→ `?` ou `help(., package=.)`. Ex: `> ?help`

→ pour les mots-clefs: `> ? "for"`

→ `apropos()`: nom de la fonction malconnu.

Ces appels nous renvoient vers l'aide de R, qui contient

→ *Usage*: appel de la fonction

→ *Arguments*: détails comme types acceptés, longueur, ...

→ *Value*: objet renvoyé et ses attributs.

→ *Détails*: quelques détails sur la paramétrisation en général.

→ *Bibliographie*: références de l'implémentation.

→ *Voir aussi*: suggestions de sujets connexes.

→ *Exemple*: exemples d'utilisation de la fonction.

Le mode démo

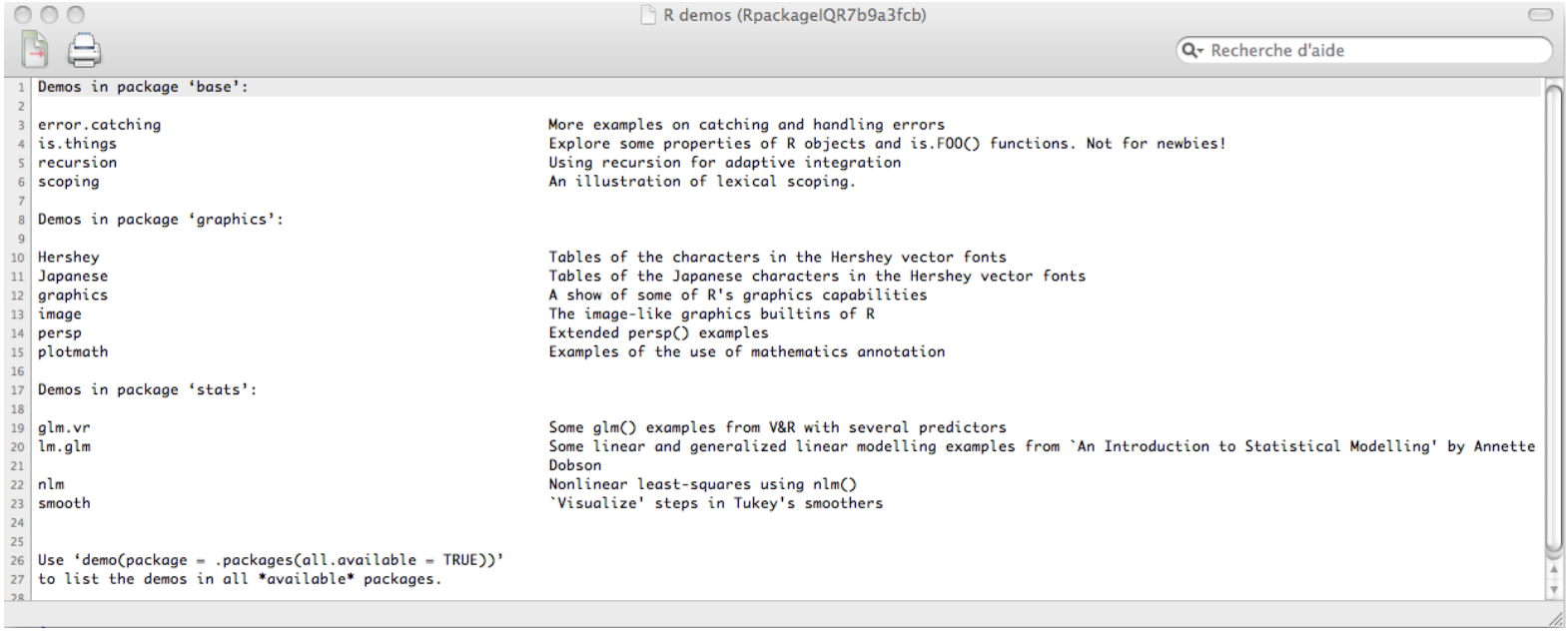
Le mode démo de R peut être très utile car:

- il facilite la compréhension des programmes;
- il facilite l'utilisation des programmes et des fonctions R.

→ `demo()`: pour obtenir la liste des démos;

→ Exemple pour une démo particulière:

```
## Display it without pausing
demo(lm.glm, package="stats", ask=FALSE)
```



Quelques commandes TRES utiles!

R est un langage objet. Il est donc intéressant de manipuler chacun des attributs, accessibles via les commandes:

→ `class()`: permet de connaître la classe à laquelle l'objet appartient. Par exemple, ce peut être un GLM, ...

→ `typeof()`: type de l'objet. est censé renvoyer "integer", "logical", ...

→ `structure()`: détaille la structure de l'objet. On accède ainsi à sa construction: ses attributs, les types, ...

→ `attributes()`: donne la liste des attributs de l'objet considéré. L'accès aux attributs se réalise par le dollar. Ex: object$res (residus).

Pour le débuggage, la commande `traceback()` remonte à l'erreur d'origine lorsqu'il y a imbrication de fonctions...

Exercice 1

Tous les exercices seront basés sur le même jeu de données.

1. Télécharger la librairie MASS;
2. Charger le jeu de données `Insurance` par `data()`;
3. Que décrit ce jeu de données?

Solution:

Exercice 1

Tous les exercices seront basés sur le même jeu de données.

1. Télécharger la librairie MASS;
2. Charger le jeu de données `Insurance` par `data()`;
3. Que décrit ce jeu de données?

Solution:

```
library(MASS)
data(Insurance)
?Insurance
```

1. P-R-emier pas
   - Généralités
   - Fondamentaux
   - Manipulation
   - Graphiques
   - Probabilités
2. Statistiques desc-R-iptives
3. Exemples d'usages d'actuaire
4. Conclusion

Les types fréquents en R...

Les principaux types de variables en R sont:

- les réels et les complexes: `real()` ou `complex()`
  → `is.real()` ou `is.complex()` pour tester,
  → `as.real()` ou `as.complex()` pour convertir.
- les entiers: `integer()`
  → `is.integer()`, `as.integer()`.
- les booléens: `logical()`
  → `is.logical()`, `as.logical()`.
- les chaines de caractères: `character()`
  → `is.character()`, `as.character()`.
- les variables catégorielles: factor()
  → `is.factor()`, par opposition à `is.numeric()`
  → `as.factor()` (attention: conv. factor/numeric).

... et les principaux objets correspondants

Les variables, en fonction de leur type, sont insérées dans:

→ `numeric()`: vecteurs de valeurs numériques (englobe les réels et les entiers).

→ `vector()`: vecteur d'objets de même type (à préciser). Tableau à 1 dimension.

→ `matrix()`: matrice de numériques. Tableau à 2 dim.

→ `array()`: tableau à n dimensions.

→ `data.frame()`: matrice avec mélange de types possible, mais dont les vecteurs sont de même taille

→ `list()`: liste d'objets de tout type.

Rq fondamentale: lors de la création de fonctions, les listes sont très utiles pour retourner un objet sur mesure!

Exercice 2

Rappel: nous travaillerons tjs avec les données `Insurance`!

1. Quel est le type de ce jeu de données?
2. Quelle en est sa classe?
3. Quelle est la dimension de ces données?
4. Donner le nombre de lignes? De colonnes?

Solution:

Exercice 2

Rappel: nous travaillerons tjs avec les données `Insurance`!

1. Quel est le type de ce jeu de données?
2. Quelle en est sa classe?
3. Quelle est la dimension de ces données?
4. Donner le nombre de lignes? De colonnes?

Solution:

```
typeof(Insurance)
attributes(Insurance)
class(Insurance)
structure(Insurance)
dim(Insurance); class(dim(Insurance))
nrow(Insurance)
dim(Insurance)[1]
ncol(Insurance)
dim(Insurance)[2]
```

Affichage et stockage

En R, les résultats sont stockés (et affichés) par:

- stockage par affectation avec `<-` ou `=`;
- affichage par l'utilisation de `()`: utile pour débugger.

```
> (a <- 2)
[1] 2
```

Par défaut, les variables et objets :

- sont stockés pendant toute la durée de la session R (sauf commande inverse explicite (ex: `rm(obj)`);
- peuvent être effacés à la fin de cette session;
- ou conservés par `save.image()` (dans le fichier ".RDataFile" du répertoire adéquat).

Rq: pour l'affichage, la fonction `substitute()` est très utile: elle permet de faire référence au nom d'un objet...

Manipulation des objets

→ Liste des objets en mémoire: `ls()` ou `objects()`

```
> ls()
 [1] "a"                                  "aggregate.data"
 [3] "economic.index.trend"   "event.distrib"
```

→ Suppression d'un objet (éviter fuites de mémoire): `rm()`

```
> rm(a)
> a
Erreur : objet 'a' introuvable
```

→ Accès aux éléments d'une liste (idem avec les data.frame) par nom: `names()`

```
> A <- list(x=1, y="blabla", z=TRUE)
> names(A)
[1] "x" "y" "z"
```

Exercice 3

1. Quelles variables contient ce jeu de données? Enumérer leurs noms.
2. Quel est le type de ces variables?

Solution:

Exercice 3

1. Quelles variables contient ce jeu de données? Enumérer leurs noms.
2. Quel est le type de ces variables?

Solution:

```
names(Insurance)
typeof(Insurance$District)
typeof(Insurance$Group)
typeof(Insurance$Age)
typeof(Insurance$Holders)
typeof(Insurance$Claims)
class(Insurance$District)
class(Insurance$Group)
class(Insurance$Age)
class(Insurance$Holders)
class(Insurance$Claims)
```

Actions et opérations sur scalaires et vecteurs

- add., soustraction, mult., div., puissance: `+`, `-`, `*`, `/`, `^`;
- moins courant: modulo `%%`, partie entière `%/%`;
- concaténation: `c()` (ex: suite arithmétique `c(1:10)`);
- généralisation: suite où l'on peut fixer
  $\rightarrow$ la raison: `seq(from=a, to=b, by=r)` où a est le premier terme, le dernier terme est $\leq b$, de raison r;
  $\rightarrow$ le nb de termes l: `seq(from=a, to=b, length=l)` où a est le premier terme, le dernier terme est $\leq b$;
- n répétitions d'un élément a: `rep(a,n)`.
- accès au i^{eme} élément du vecteur v: `v[i]`
- accès aux n premiers éléments de v: `head(v, n)`
- accès aux n derniers éléments de v: `tail(v, n)`
- accès à certains éléments de v: `v[v > 8]`
- suppression du i^{eme} élément de v: `v <- v[-i]`

Exercice 4

1. Afficher uniquement le colonne “District”.
2. Quelle est la longueur de cette colonne?
3. Attacher le dataframe. Réitérer la question précédente.

Solution:

Exercice 4

1. Afficher uniquement le colonne “District”.
2. Quelle est la longueur de cette colonne?
3. Attacher le dataframe. Réitérer la question précédente.

Solution:

```
Insurance$District
length(Insurance$District)
attach(Insurance)
Insurance$District
length(Insurance$District)
```

Travail sur les vecteurs

Fonctions utiles pour manipuler les vecteurs:

→ `sort()`: trie les éléments du vecteur;

→ `rank()`: retourne les rangs des éléments;

→ `head(a, x)` et `tail(a, x)`: déjà vues;

→ `%in%`: test de présence d'éléments dans le vecteur;

→ `match()`: retourne la position de la première occurrence d'un élément dans un vecteur;

→ `unique()` retourne le vecteur où les éléments qui étaient égaux n'apparaissent qu'une seule fois.

D'autres fonctions très utile pour les tests sont les méthodes

- `which()`: indice de l'élément satisfaisant le test,
- `which.min()`: renvoie indice du min de l'échantillon,
- `which.max()`: idem avec le max.

Actions et opérations sur matrices

- création d'une matrice: `M <- matrix()`;
- par défaut, les matrices se remplissent par colonne;
- accès à la i^{eme} ligne de la matrice M: `M[i, ]`
- accès à la j^{eme} colonne de M: `M[ ,j]`
- accès à l'élément (i,j) de M: `M[i,j]`
- suppression de la i^{eme} ligne de M: `M <- M[-i, ]`
- suppression de la j^{eme} colonne de M: `M <- M[ ,-j]`
- concaténation de plusieurs matrices:
  → par colonne: `cbind()`,
  → par ligne: `rbind()`.

Les opérations matricielles standards sont les suivantes:
→ add., soustraction: $+$, $-$,
→ produit matriciel, produit Kronecker: `%*%`, `%x%`;
→ mult. et div. terme à terme: $*$ et $/$.

Travail sur des matrices

En cas de petits calculs à faire, voici les opérations et fonctions utiles pour le calcul matriciel:

→ `det()` : calcule le déterminant;

→ `diag()` : retourne la diagonale;

→ `t()`: retourne la transposée;

→ `solve(A)`: calcule l'inverse de A

→ `solve(A,b)`: résoud le système $Ax = b$;

→ `eigen()`: calcule les valeurs propres;

→ `sapply()`: optimisation des calculs (évite les boucles);

→ `svd()`: calcule la décomposition en valeurs singulières;

→ `prod(svd(M))$d` valeur absolue du déterminant!

ATTENTION: pour des opérations entre vecteurs de taille différentes, R les complète automatiquement!

Actions et opérations sur data frame et listes

- les data frame: tableau de données.
  - création d'un data frame: `data <- data.frame()`;
  - nomination des colonnes: `names(data) <- c("vin","region")`;
  - accès aux éléments du data frame: idem que pour les matrices, ou par les noms précédés d'un `$`.
  - suppression de la colonne *region* de *data*: `data$region <- NULL`
  - concaténation de 2 data frames A et B de même nb de lignes (par colonne): `data.frame(A,B)`.
- les listes: regroupement d'objets de nature $\neq$.
  - création d'une liste: `L <- list()`;
  - accès et suppression du i^{eme} élément peut aussi se faire par `L[[i]]` et `L[[-i]]`.
  - extraire les objets d'une liste: `attach(L)` (`detach(L)`).
  - idem pour les autres points.

Rq: ces objets pouvant regrouper différents types, les opérations arithmétiques ne peuvent pas toujours s'effectuer!

Exercice 5

1. Attacher le jeu de données. Donner les identificateurs de chaque ligne du jeu de données.
2. Accéder à l'individu "10" (toutes les lignes d'un data frame ont un identifiant unique).
3. En visualiser les 5 premières lignes du jeu de données.
4. Extraire les 2 premières colonnes des 3 premières lignes.
5. Extraire les colonnes 1 et 3 des 4 premières lignes.

Solution:

Exercice 5

1. Attacher le jeu de données. Donner les identificateurs de chaque ligne du jeu de données.
2. Accéder à l'individu "10" (toutes les lignes d'un data frame ont un identifiant unique).
3. En visualiser les 5 premières lignes du jeu de données.
4. Extraire les 2 premières colonnes des 3 premières lignes.
5. Extraire les colonnes 1 et 3 des 4 premières lignes.

Solution:

```
attach(Insurance) ; rownames(Insurance)
Insurance["10", ]
head(Insurance, 5)
Insurance[1:5, ]
Insurance[1:3, 1:2]
Insurance[1:4, c(1,3)]
```

Tests

Plusieurs syntaxes pour les structures conditionnelles:

- `if (condition) { A }`: exécute les instructions A si la condition est vraie;
- `if (condition) { A } else { B }`: exécute A si la condition est vraie, sinon exécute B;
- `case` ... `end case`: cas par cas.

Dans les conditions (entre autres), on utilise des...

- ... comparaisons entre expressions: égalité (`==`), différence (`!=`), supériorité (`>=`), infériorité (`<`);
- ... comparaisons entre objets: avec `identical()`;

→ Accumuler conditions via les tests logiques : `!`, `|`, `&`, `xor`.

→ Ces tests peuvent être effectués sur vecteurs: attention, `&` est un ET logique sur chaque élément des vecteurs (renvoie un vecteur de booléens).
`&&` agit sur tous les éléments (renvoie un seul booléen).

Exercice 5 - bis

1. Extraire les individus qui appartiennent au premier "District".
2. Extraire les individus qui appartiennent au premier "District" et dont l'âge est inférieur à 25 ans ou supérieur à 35 ans.

Solution:

Exercice 5 - bis

1. Extraire les individus qui appartiennent au premier "District".
2. Extraire les individus qui appartiennent au premier "District" et dont l'âge est inférieur à 25 ans ou supérieur à 35 ans.

Solution:

```
Insurance[District == 1, ]
Insurance[which(District == 1), ]
Insurance[(District == 1) & (Age != "25-29") , ]
Insurance[which((District == 1) & (Age != "25-29")),]
Insurance[which((District == 1) && (Age != "25-29")),]
Insurance[which((District == 1) & (Age == "<25" | Age == ">35
```

Boucles (itérations)

A de multiples reprises, on ne peut malheureusement pas éviter les boucles.
Il existe en R plusieurs types de boucles itératives selon les besoins:

- `for (var in seq) { A }`: exécute les instructions A pour chaque valeur de la variable *var* appartenant à la séquence *seq*;
- `while (condition) { A }`: exécute les instructions A tant que la condition reste vraie;
- `repeat { A; if (condition) break }`: réitère les instructions A si la condition est vérifiée.

→ Eviter les boucles dans la mesure du possible car elles ralentissent l'exécution du code;

→ S'en passer grâce à: `lapply()`, `sapply()`, `outer()`.

Ecrire sa première fonction R

- La structure générale d'une fonction est:

```
> foo <- function(liste_des_parametres) {
      commandes
      objets_retournes
}
```

- Les accolades { et } définissent le début et la fin de la fonction. La dernière instruction contient le ou les objets retournés par la fonction.
- les variables déclarées ds la fonction ont 1 portée locale.
- Exécuter la fonction par `foo(...)`.

```
> stats.descriptives <- function(data){
    m=mean(data)       # commande 1
    sd=sd(data)         # commande 2
    list(m,sd)   # objet retourne
}
```

Utilisation d'une fonction R

Un certain nombre de spécificités sont à connaitre:

- programmation de la fonction:
  → si dans la console: utilisable directement;
  → si dans un script: importation par `source()`.
- définition de valeurs par défaut des arguments:

```
annuity <- function(nominal, lifeExpectancy=84,
                                discountRate)
{
      (...)
}
```

- `annuity(nominal,discountRate)` et `annuity(nominal, 84, discountRate)` identiques.
- 3 façons de spécifier les arguments:
  - par le nom: ordre sans importance,
  - par la position: ordre à respecter indispensablement,
  - avec des valeurs par défaut.

Programmation efficace

Suivant la manière d'implémenter son programme, le temps d'exécution peut varier très significativement. Citons quelques règles de base valables en environnement R:

→ les opérations vectorielles sont bien plus rapides que les opérations élément par élément sur le même vecteur,

→ les opérations matricielles sont globalement très rapides,

→ moins il y a d'options/possibilités d'erreur, plus le calcul est rapide (ex: `sum(x)/length(x)` au lieu de `mean(x)`)

→ la manipulation des data.frame est bien plus lente que celle des matrices!

→ l'allocation de mémoire est meilleure lorsqu'elle est faite d'un trait, plutôt que de manière incrémentale.

→ dépasser la capacité en mémoire vive pour faire opérations ralentit considérablement le code (fait parfois planter R).

Temps de calcul et performance

On peut récupérer des statistiques de performance. Parfois utile lorsqu'on s'absente...

C'est le cas de `system.time()` qui fournit:

→ le temps utilisé par le système,

→ le temps utilisateur,

→ le temps total nécessaire.

Exemple:

```
> system.time(rnorm(100000, 0, 1))
utilisateur     systeme      ecoule
      0.010       0.001       0.011
```

Rq: R est lent à l'usage de grosses BdD (utiliser alors la librairie `RODBC`...).

Parades pour éviter les boucles: `apply()`

Travailler avec des matrices est plus optimal!

→ Structure de la syntaxe: `apply(M, orientation, foo)`
où *M* est une matrice, *orientation* désigne les lignes ou les colonnes et *foo* est la fonction à appliquer.

→ Retourne un vecteur de l'opération sur les lignes/colonnes.

→ Exemple d'application avec `apply()`:

```
> (M <- matrix(1:4, ncol=2))
     [,1] [,2]
[1,]    1    3
[2,]    2    4
>
> apply(M,1, sum)  # somme sur lignes
[1] 4 6
```

sapply(), lapply() et tapply()

Adaptation pour travailler sur les autres structures d'objet.

- `sapply(x,foo,arg.commun)` et `lapply(...)`:
  → Avec des vecteurs/listes. *x* est la liste/vecteur, *foo* est la fonction à appliquer, et *arg.commun* est à spécifier si *foo* a plusieurs arguments.
  → Par défaut, les valeurs de *x* sont affectées au premier argument de la fonction.
  → Retourne une liste avec les résultats de la fonction.
  → `sapply()`: idem mais retourne un vecteur.
- `tapply()`: cas de variables catégorielles (sous-groupes);
  → Appliquée sur des objets de type `factor`.
  → `tapply(x, class, foo)` où *x* est le vecteur, *class* est le vecteur de type factor, et *foo* la fonction.

Rq: `by()` peut se substituer à `tapply()`.

Parades pour éviter les boucles: `outer()`

$\rightarrow$ `outer(x,y,f)`: applique f aux vecteurs x et y. Retourne une matrice M de la forme

$$M[i,j] = f(x_i, y_j).$$

`outer(x,y,f)` est particulièrement utile pr les graphiques!

Exercice: calculer $f(x,y) = \frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$ par l'utilisation

1. de boucles: $((x,y) \in [-1000, 1000]^2)$. Combien de temps?

Parades pour éviter les boucles: `outer()`

$\rightarrow$ `outer(x,y,f)`: applique f aux vecteurs x et y. Retourne une matrice M de la forme

$$M[i,j] = f(x_i, y_j).$$

`outer(x,y,f)` est particulièrement utile pr les graphiques!

Exercice: calculer $f(x,y) = \frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$ par l'utilisation

1. de boucles: $((x,y) \in [-1000, 1000]^2)$. Combien de temps?

```
> f <- function(x,y) {return(sin(sqrt(x^2+y^2))/sqrt(x^2+y^2))}
> x <- -1000:1000;  y <- -1000:1000 ; res <- matrix(NA,nrow=2001,ncol=2001)
> for (i in 1:2001) { for j in (1:2001) { res[i,j] <- f(x[i],y[j]) }}
```

Parades pour éviter les boucles: `outer()`

$\rightarrow$ `outer(x,y,f)`: applique f aux vecteurs x et y. Retourne une matrice M de la forme

$$M[i,j] = f(x_i, y_j).$$

`outer(x,y,f)` est particulièrement utile pr les graphiques!

Exercice: calculer $f(x,y) = \frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$ par l'utilisation

1. de boucles: $((x,y) \in [-1000, 1000]^2)$. Combien de temps?

```
> f <- function(x,y) {return(sin(sqrt(x^2+y^2))/sqrt(x^2+y^2))}
> x <- -1000:1000;  y <- -1000:1000 ; res <- matrix(NA,nrow=2001,ncol=2001)
> for (i in 1:2001) { for j in (1:2001) { res[i,j] <- f(x[i],y[j]) }}
```

2. de `outer()`. Comparer les temps d'exécution.

Parades pour éviter les boucles: `outer()`

→ `outer(x,y,f)`: applique f aux vecteurs x et y. Retourne une matrice M de la forme

$$M[i,j] = f(x_i, y_j).$$

`outer(x,y,f)` est particulièrement utile pr les graphiques!

Exercice: calculer $f(x,y) = \frac{sin(\sqrt{x^2+y^2})}{\sqrt{x^2+y^2}}$ par l'utilisation

1. de boucles: $((x,y) \in [-1000, 1000]^2)$. Combien de temps?

```
> f <- function(x,y) {return(sin(sqrt(x^2+y^2))/sqrt(x^2+y^2))}
> x <- -1000:1000;  y <- -1000:1000 ; res <- matrix(NA,nrow=2001,ncol=2001)
> for (i in 1:2001) { for j in (1:2001) { res[i,j] <- f(x[i],y[j]) }}
```

2. de `outer()`. Comparer les temps d'exécution.

```
> f <- function(x,y) {return(sin(sqrt(x^2+y^2))/sqrt(x^2+y^2))}
> x <- -1000:1000; y <- -1000:1000 ; res <- matrix(NA,nrow=2001,ncol=2001)
> res <- outer(x,y,f)     # + rapide, + concis
> system.time(outer(x,y,f))
```

Tâches complexes et répétitives

Lorsqu'on veut répéter l'évaluation d'une expression, une fonction utile est `replicate()`.

Syntaxe: `replicate(n, expr, simplify = ...)`, avec

- *n* est le nombre de répétitions;
- *expr* est l'expression à répéter;
- *simplify* concerne la forme du résultat de l'application de la fonction (array, vecteur, ...)

→ Cette fonction est souvent utilisée dans des problématiques de tirages aléatoires.

→ Exercice: calculer 1000 variances de 20 tirages aléatoires uniformes (par `sample()`) de support "Sepal.Width" (données `iris`):

Tâches complexes et répétitives

Lorsqu'on veut répéter l'évaluation d'une expression, une fonction utile est `replicate()`.

Syntaxe: `replicate(n, expr, simplify = ...)`, avec

- *n* est le nombre de répétitions;
- *expr* est l'expression à répéter;
- *simplify* concerne la forme du résultat de l'application de la fonction (array, vecteur, ...)

→ Cette fonction est souvent utilisée dans des problématiques de tirages aléatoires.

→ Exercice: calculer 1000 variances de 20 tirages aléatoires uniformes (par `sample()`) de support "Sepal.Width" (données `iris`):

```
> var(sample(iris$Sepal.Width, size=20, replace=F))
[1] 0.1718684
> replicate(1000,var(sample(iris$Sepal.Width,...)))
```

Application d'une fonction scalaire sur un vecteur

On considére des fonctions qui s'appliquent sur des scalaires et qui retournent un scalaire.

Ce sont généralement des fonctions usuelles, mais pas des résumés statistiques comme la moyenne etc...

- Exemples de fonction.
  - `log()`;
  - `sqrt()`;
  - `exp()`;
  - `sin()`;
- Résultat.
  - la fonction s'applique sur chaque élément du vecteur, et retourne donc également un vecteur;
  - idem pour les matrices.

Exemple: si V est un vecteur alors $\log(V)$ est un vecteur donné par $\log(V[i]), \forall i \in [1, n]$.

Résolution de système et instabilité

Assume that we want to solve in $\boldsymbol{x}$ the following linear equation $A\boldsymbol{x} = \boldsymbol{b}$ where A is some $n \times n$ invertible matrices, and $\boldsymbol{b}$ a vector of size n.

A natural idea would be to seek for the inverse of A, $\boldsymbol{x} = A^{-1}\boldsymbol{b}$.

Using the transpose of the comatrix technique needs around $n!$ operations, i.e. if $n = 100$, $n! \approx 10^{158}$.

Using Gauss's pivot technique needs around $n^3/3$ operations.

Remark Solving linear models can be numerically unstable (see e.g. Rappaz & Picasso (1998))

$$\begin{cases} 4.218613x_1 + 6.327917x_2 = 10.546530 \\ 3.141592x_1 + 4.712390x_2 = 7.853982 \end{cases}$$

has the following trivial solution $x_1 = x_2 = 1$. But if the system becomes

$$\begin{cases} 4.218611x_1 + 6.327917x_2 = 10.546530 \\ 3.141594x_1 + 4.712390x_2 = 7.853980 \end{cases}$$

the solution is still unique, and equal to $x_1 = -5$ and $x_2 = +5$.

Approximations par arrondi

Attention donc à bien gérer les décimaux (un seul arrondi en fin de calcul plutot qu'accumulation d'arrondis).

Quelques commandes qui permettent de gérer les décimaux:

→ `round()`: donne l'arrondi du nombre spécifié.

→ `floor()`: la partie entière du nombre.

→ `ceiling()`: arrondi à l'entier supérieur.

→ `trunc()`: tronque le nombre.

→ `format(object, digit=...)`: accès au format du nb.

En R, il faut faire attention aux erreurs numériques, car:

→ $\frac{5}{3} - \frac{1}{3}$ est normalement égal à $\frac{8}{3} - \frac{4}{3}$.

→ mais essayer avec $==$...

Gestion des erreurs

Les erreurs n'apparaissent qu'à l'exécution.

→ Il n'est pas toujours évident de les déceler.

→ Elles sont même parfois cachées par l'exécution:

```
> (v <- c(1:5))
[1] 1 2 3 4 5
>
> v[6]     # devrait renvoyer une erreur
[1] NA
> v[8] <- 8    # devrait renvoyer une erreur
> v
[1]  1  2  3  4  5 NA NA  8
```

→ Erreurs définies par l'utilisateur, renvoyées par `stop()`.

→ Rq: avertissements consultables par `warning()`.

Gestion des dates

En pratique, il est utile de convertir des dates. C'est fréquemment le cas lors de l'importation de fichiers. Pour cela, il existe une classe "Date" avec les attributs:

- %H : pour l'heure, entre 00 et 23;
- %d : pour le jour, entre 01 et 31;
- %m : pour le mois, entre 01 et 12
- %Y: pour l'année: sous forme de quatre chiffres.

Considérées comme des chaines de caractères dans un vecteur. Plusieurs options:
→ `as.Date()`: conversion en type "Date",

→ format de la date: %d-%m-%Y donnera 21-02-2012, etc...

→ on peut aussi faire des opérations sur les dates, comme calculer un âge par exemple (en jours par défaut).

Importation de fichiers

En pratique, nous avons bien souvent besoin d'importer des fichiers depuis d'autres logiciels ayant des extensions $\neq$:

$\rightarrow$ `read.table()`: permet d'importer des fichiers texte.

$\rightarrow$ `read.xls()`: importation de fichiers Excel.

$\rightarrow$ `read.csv()`: importation de fichiers CSV.

$\rightarrow$ `read.xport()` et `read.ssd()`: importation de fichiers de type SAS, lecture d'un script SAS pour exécution.

Les options de l'importation concernent:

$\rightarrow$ le chemin et nom du fichier par `file`.

$\rightarrow$ la denomination des colonnes par `header`.

$\rightarrow$ le type de séparateur entre les colonnes par `sep`.

$\rightarrow$ beaucoup d'autres options moins utilisées...

Exportation de données

On peut aussi vouloir exporter nos résultats (matrice, data frame, listes, ...):

- dans un fichier:
  → `write.table()` ou `sink()`: écriture dans un fichier texte.
  → `write.csv()` ou `write.csv2()`: écriture dans le logiciel Excel ss $\neq$ formats.
- dans un éditeur:
  → `edit()`: éditeur R de tableau de données.

Beaucoup d'informations utiles et détaillées à

R Data Import/Export

sur CRAN $>$ Documentation ($>$ contributed) !

Pour les grosses bases de données, voir le package `RODBC`...
Il existe un manuel sur ce sujet sur le site de R.

Exporter ses tableaux: la librairie xtable

A partir d'un data frame,

- le format HTML pour une utilisation réseau:

```
> tli.table <- xtable(tli[1:5,])
> digits(tli.table)[c(2,6)] <- 0
> print(tli.table, type="html")
<!- html table generated in R 2.14.0 by xtable 1.7-0 package ->
<TABLE border=1>
<TR> <TH>  </TH> <TH> grade </TH> <TH> sex </TH> (...)  </TR>
  <TR><TD align="right"> 1 </TD><TD align="right"> 6 ... </TR>
  <TR><TD align="right"> 2 </TD><TD align="right"> 7 ... </TR>
            (...)
 </TABLE>
```

- le format LaTeX: rédaction de note technique,

```
> print(tli.table, type="latex")
% latex table generated in R 2.14.0 by xtable 1.7-0 package
\begin{table}[ht]
\begin{tabular}{rrlllr} \hline
 & grade & sex & disadvg & ethnicty & tlimth \\  \hline
1 & 6 & M & YES & HISPANIC & 43 \\
2 & 7 & M & NO & BLACK & 88 \\
       (...) \hline
\end{tabular}
\end{table}
```

Automatisation de tâches

Lorsqu'on souhaite automatiser une tâche compliquée, on peut avoir recours au mode batch.

Exemple:

```
setwd("c:/temp")
sink("regLin.txt", append=TRUE)
data(mtcars)
cat("Donnees mtcars\n")
cat("Regression lineaire effectuee le",
format(Sys.time(),"%a %d %b %X %Y %Z"),"\n")
x <- lm(mpg~cyl, data=mtcars)
summary(x)
sink()
```

Automatisation de tâches (2)

On peut alors enregistrer cette tâche par deux techniques:

- copier ces instructions dans un fichier texte autoRegLin.txt;
- lancer l'exécution de ce programme grâce à:

```
C:\Program Files\R\rw2001\bin\R.exe" CMD BATCH
"c:/temp/autoRegLin.txt
```

- Que l'on exécute dans une console DOS:

Ou

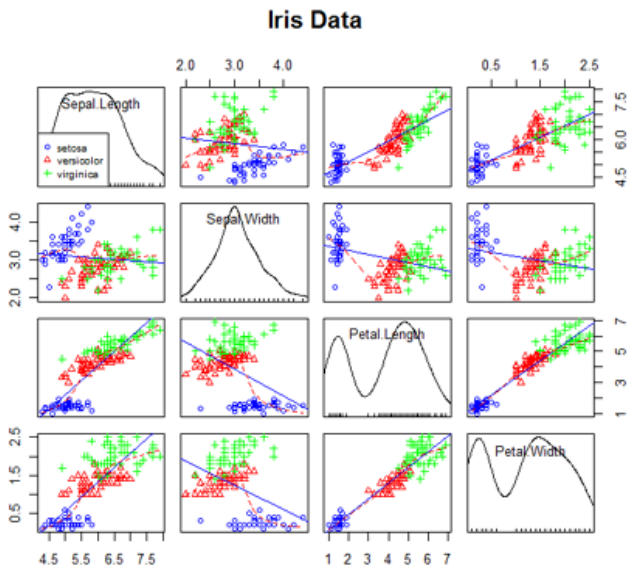
- On peut copier cette commande dans un fichier batch, autoRegLin.bat:

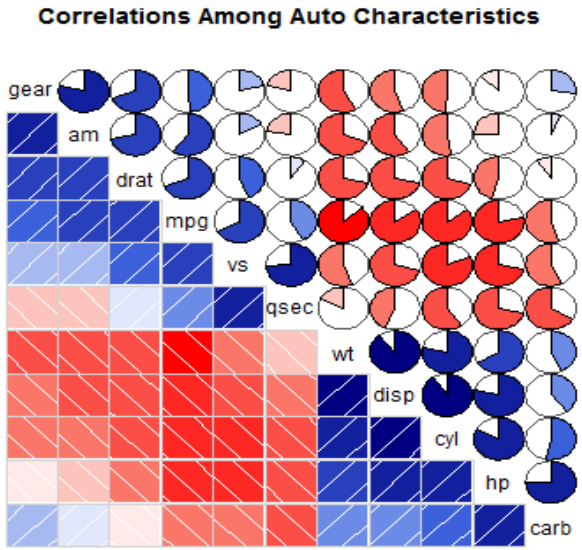
```
C:\Program Files\R\rw2001\bin\R.exe" CMD BATCH
"c:/temp/autoRegLin.r
```

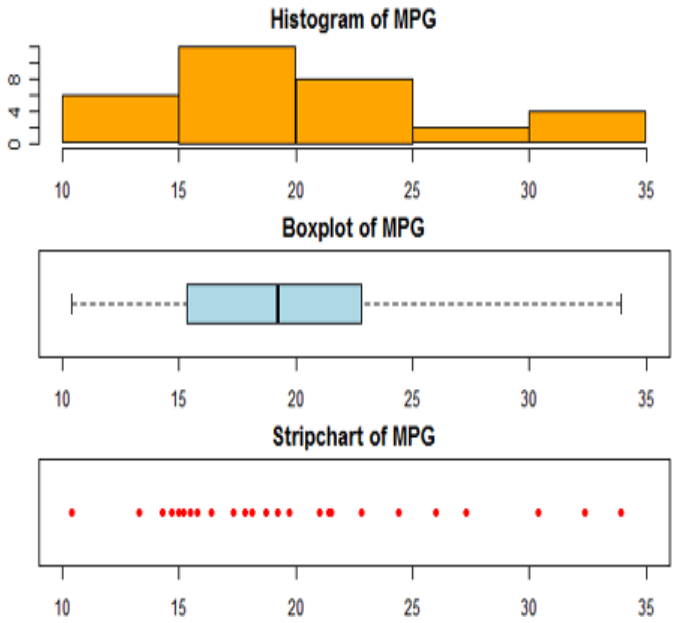
- Pour exécuter, double cliquer sur ce fichier!

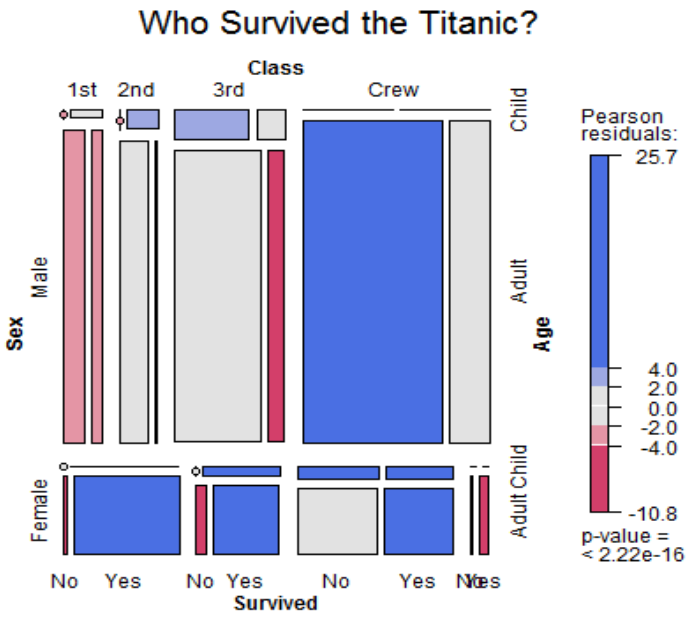
1. P-R-emier pas
   - Généralités
   - Fondamentaux
   - Manipulation
   - Graphiques
   - Probabilités
2. Statistiques desc-R-iptives
3. Exemples d'usages d'actuaire
4. Conclusion

Des graphiques variés









Créer un graphe en R

- Les graphes sont créés de manière interactive! Exemple:

```
attach(mtcars) # definit le jeu sur lequel W
plot(wt, mpg) # plots weight vs. miles/gallon.
```

- Les fenêtres graphiques contiennent plusieurs zones ("layout"), ce qui permet de tout customizer finement.
- Pour sauvegarder les graphes, on dispose des formats:
  → `pdf("graph.pdf")` : pdf file;
  → `win.metafile("graph.wmf")`: windows metafile;
  → `png("graph.png")`: png file;
  → `jpeg("graph.jpg")`: jpeg file;
  → `bmp("graph.bmp")`: bmp file;
  → `postscript("mygraph.ps")`: postscript file.

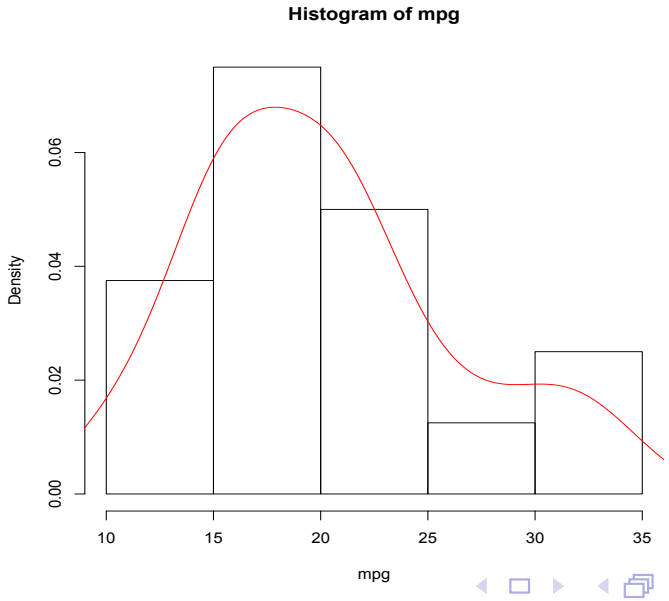
Rq: pour sauvegarder, il y a aussi `dev.print()` et `save.plot()`.

Commandes de base

- `plot()` est la fonction centrale.
- Suivant l'objet sur lequel elle est appliquée, `plot()` n'a pas le même effet (cf `methods(plot)`).
- Les fonctions `points()` ou `lines()` superposent des courbes/nuages de points sur un graphe déjà construit.

Exemple:

```
> hist(mpg, freq=F)
> lines(density(mpg), col="red")
```



Autres fonctionnalités

- On peut avoir plusieurs graphes ouverts en même temps (cf `help(dev.cur)`);
- A la création d'un 2^e graphe, le premier sera "écrasé". Pour éviter cela, ouvrir avant 1 nouvelle fenêtre par:
  → `windows()` : OS Windows;
  → `X11()`: OS Unix;
  → `quartz()`: OS Mac;
- Ajouter du texte par `text()`;
- Ajouter une légende par `legend()`;
- Ajustement de nombreux paramètres grâce à `par()`:
  → couleur par `col`, types de point par `pch`;
  → axes des abscisses et ordonnées: `xlim` et `ylim`;
  → + de graphes dans la même fenêtre: `mfrow`...

Rq: ajout de graphes ds même fenêtre: `split.screen()` ou `layout()`.

Types de graphes

Un des points attractifs de R réside dans la qualité et la variété des sorties graphiques.
De nombreux chercheurs ont développé des méthodes de visualisation adéquates suivant le phénomène observé:

- les "density plots": répartition des données,
- les "dotplots": ,
- les "barplots": données discrètes,
- les "line charts": droites,
- les "pie charts": camembert,
- les "boxplots": dispersion autour d'une valeur centrale,
- les "scatterplots": mini-graphes regroupés.

En fait on peut customiser tous ces graphes, et donc créer un graphe qui répond à ses propres besoins!

Histogramme et graphes de densité

Les histogrammes sont simples d'interprétation, mais décrivent mal les données si les classes sont mal choisies.

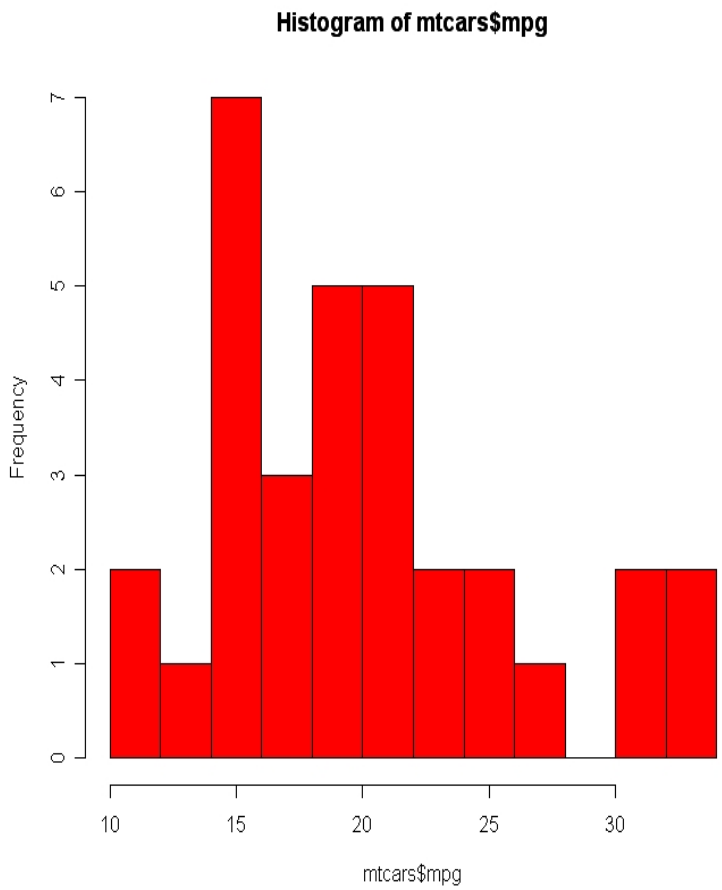
- Création: `hist(x)`, où x est un vecteur numérique;
- L'option `freq=F` permet de tracer la fréquence relative (proba) plutôt que la fréquence absolue (nombre),
- Option `breaks`: choix du découpage en classe de x.

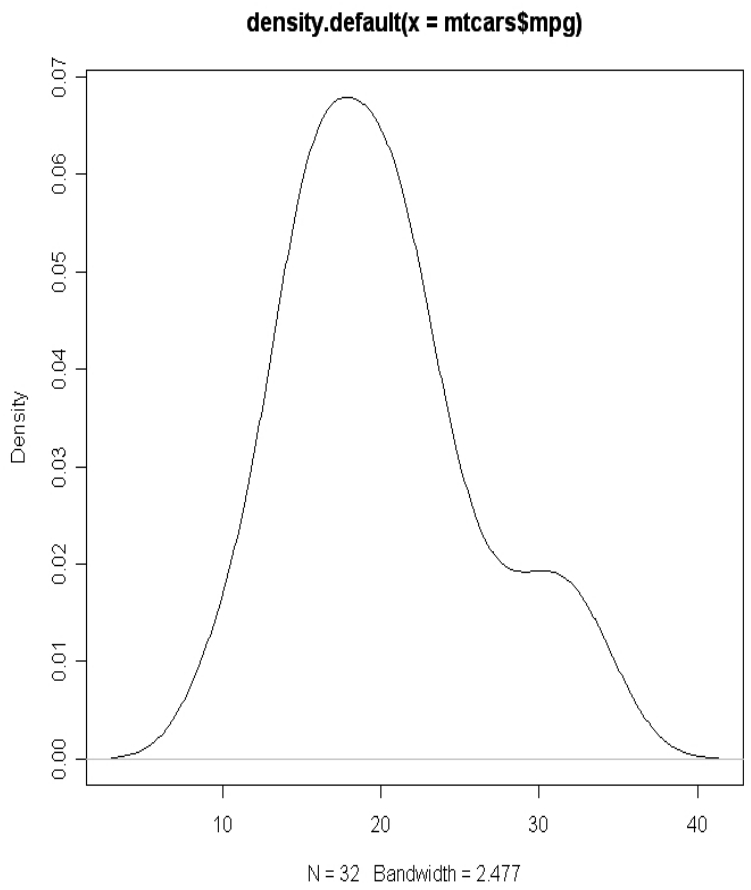
On utilise alors plutôt la densité comme estimation de la répartition des données:

- par défaut, R utilise la méthode des noyaux;
- le noyau par défaut est gaussien,
- La fonction `density(x)` estime la densité de probabilité des observations.

Le choix de la taille de fenêtre peut être un problème si les données sont volumineuses...

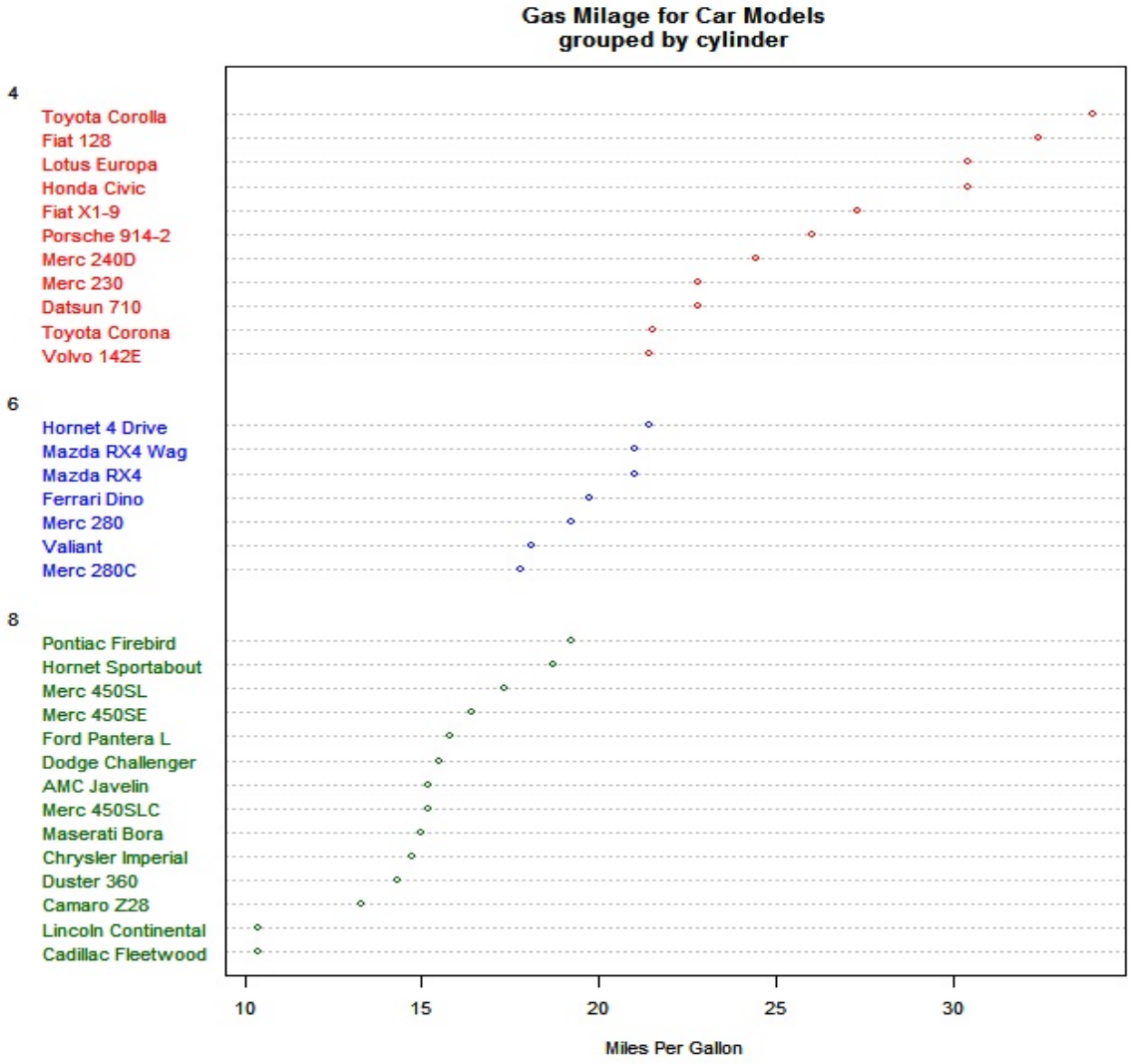
Exemple d'histogramme et de densité





Dotplots par `dotchart(x, labels=)`

- *x* est un vecteur numérique, *labels* sont les étiquettes;
- Option `group`: aggréger individus pour créer groupes,
- Option `gcolor` et `cex`: couleurs/taille des étiquettes.



Les barplots

Ressemble aux historammes, bien adapté à des données qualitatives. Basés sur des données de comptage.

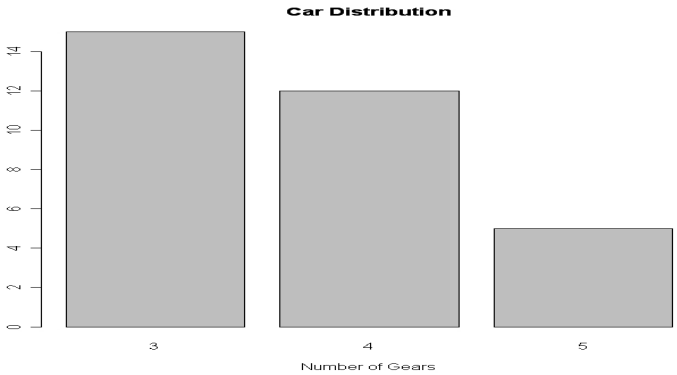
- Création: `barplot(x)`, où x est un vecteur / matrice.
- L'option `names.arg` permet d'étiqueter les bâtons (character vector),
- Option `horiz`: graphique à l'horizontal.

$\Rightarrow$ Si x est un vecteur, alors x est la hauteur des bâtons.

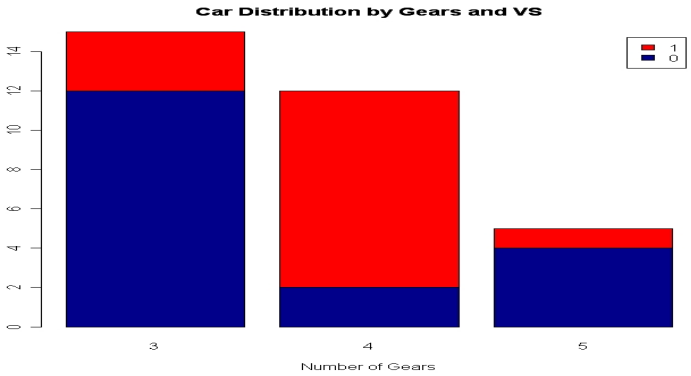
$\Rightarrow$ Si x est une matrice, alors il se présente 2 cas:

- soit l'option `beside` est vraie, auquel cas on a un "grouped" barplot.
- soit l'option `beside` est fausse, auquel cas on a un "stacked" barplot.

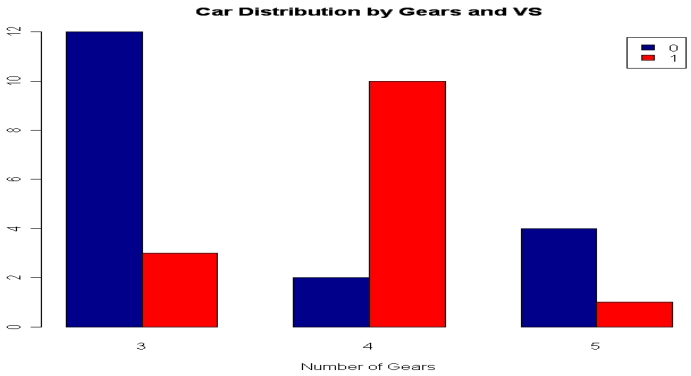
Exemple de barplots



Barplot classique



Stacked barplot



Grouped barplot

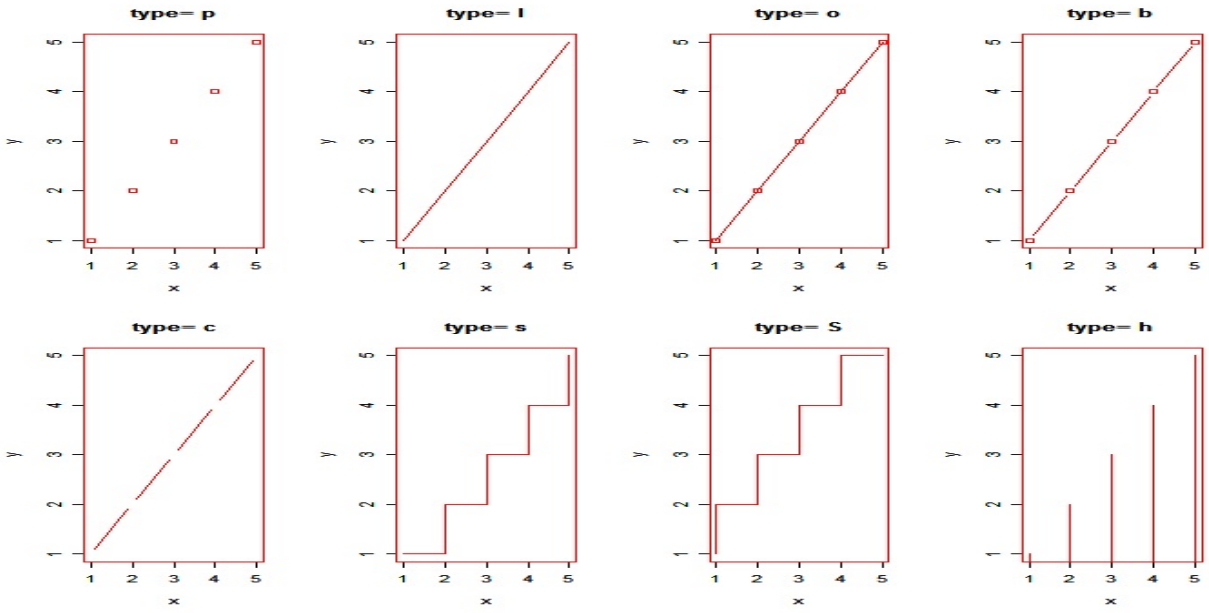
Les Lines Charts

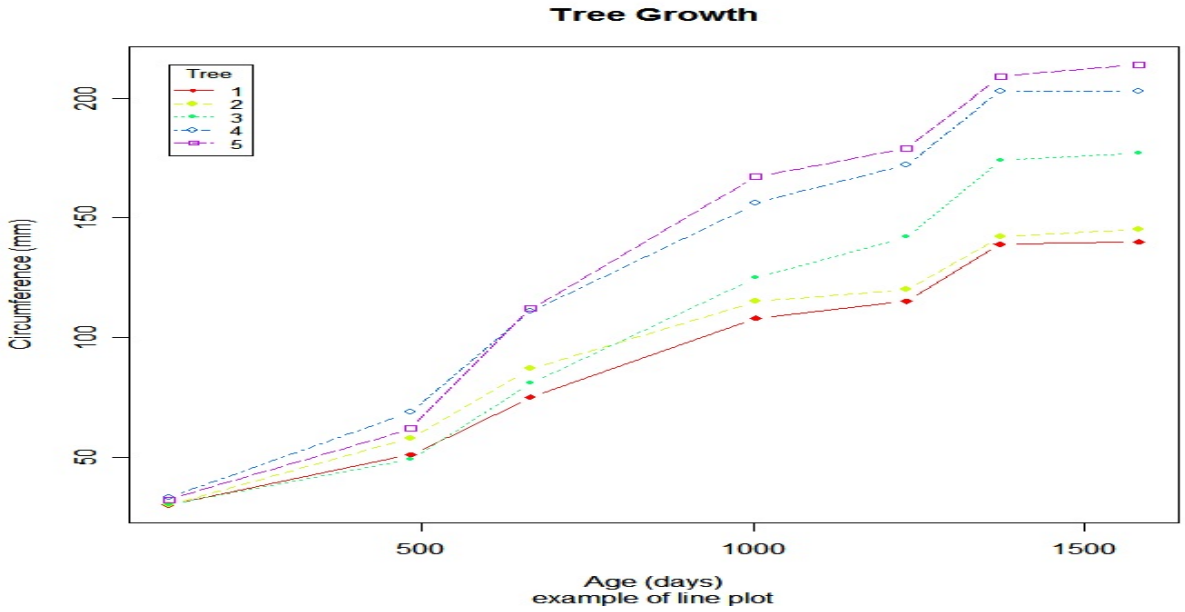
Comme leur nom l'indique, il s'agit de tracer des droites:

- Création: `lines(x,y)`, où x et y sont des vecteurs numériques.
- L'option `type` permet de choisir le type de point pour les observations. Il existe
  $\Rightarrow$ p: point,
  $\Rightarrow$ l: droite,
  $\Rightarrow$ o: points plus droites,
  $\Rightarrow$ b, c: autres points,
  $\Rightarrow$ s, S: escaliers,
  $\Rightarrow$ h: droites verticales de type histogramme,
  $\Rightarrow$ n: pas de points ou de droites.
- Option `horiz`: graphique à l'horizontal.

La fonction `lines(x,y)` ajoute de l'information à un graphe préexistant (suit en général une commande `plot(x,y)`).

Exemples de Line Charts



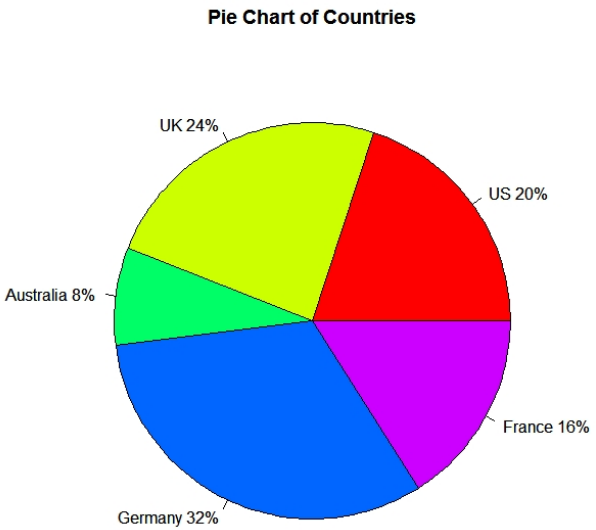


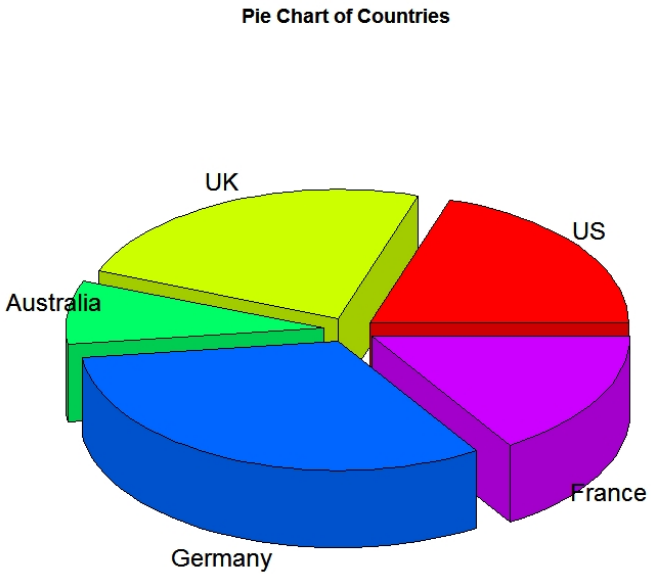
Les Pie Charts ou camemberts

Pas recommandé par la documentation R: on visualise + facilement des longueurs que des surfaces!

- Création: `pie(x)`, où x est un vecteur numérique non négatif (surface).
- L'option `labels` permet de nommer chaque quartier.
- Pas beaucoup d'options ici!

Exemples:

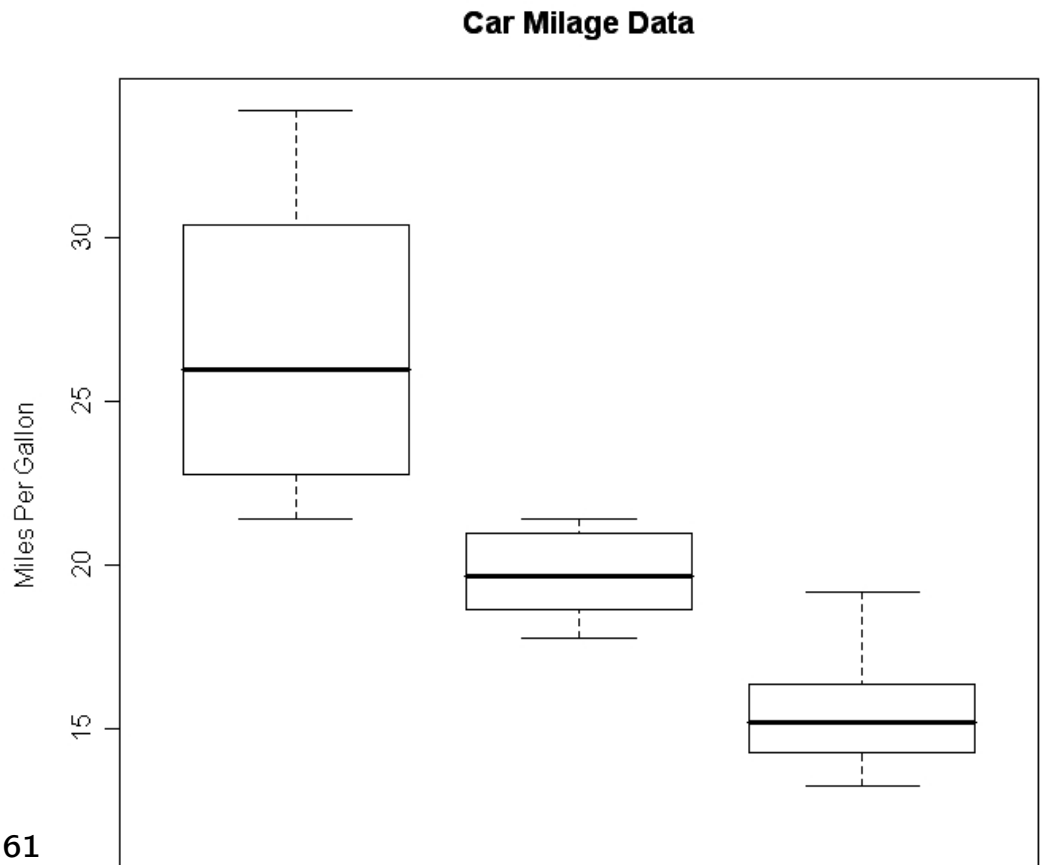




Les Boxplots, très appréciés

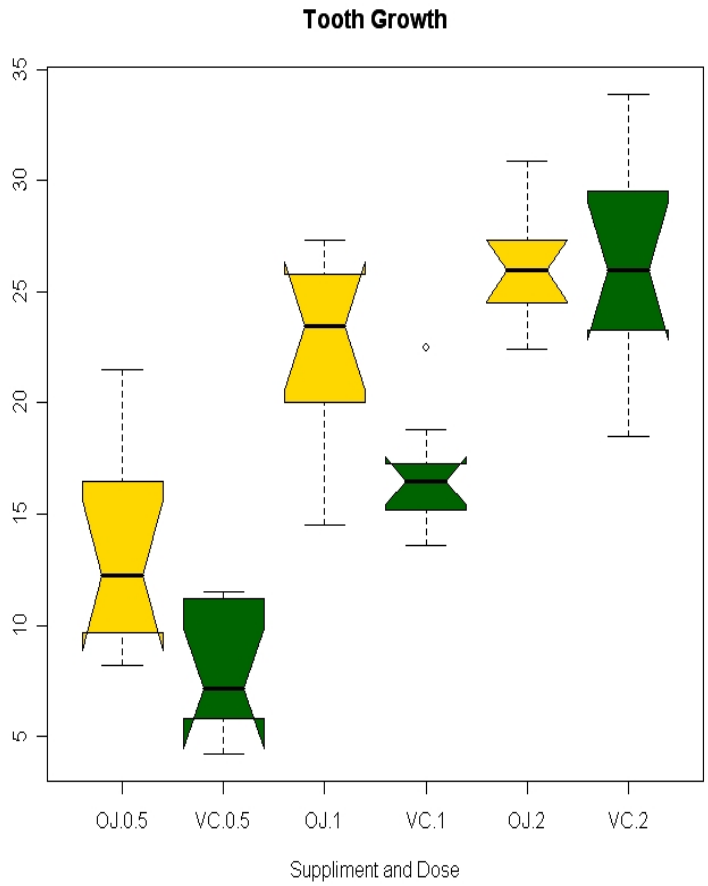
Adapté pour des variables individuelles ou pour des groupes.

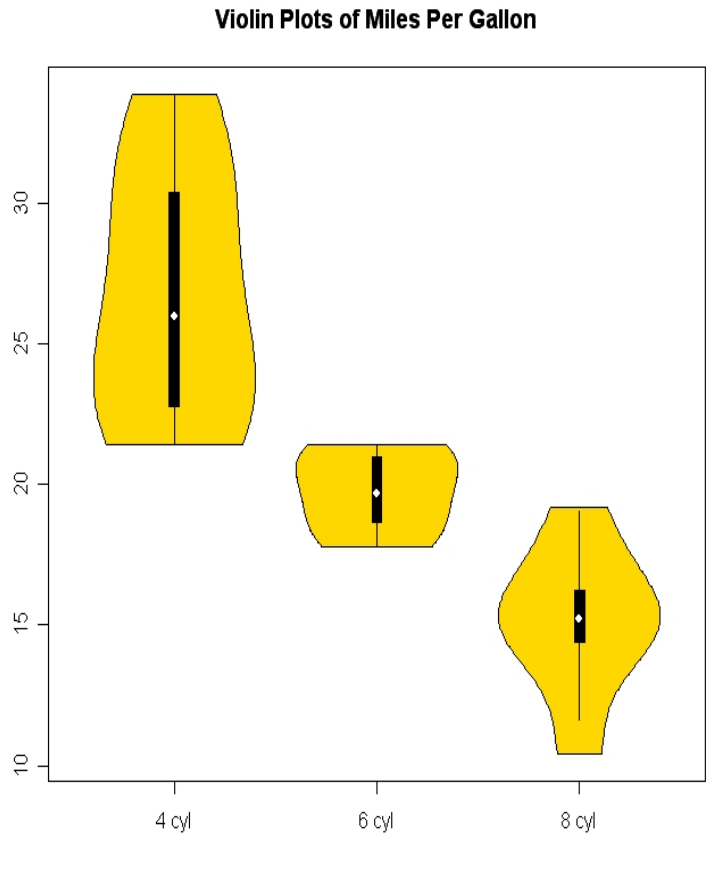
- Création: `boxplot(x, data)`, où x est une `formula` (y∼group donne un boxplot séparé pour chaque valeur de *group*), et *data* est le jeu de données.
- Options `varwidth`/`horizontal`: largeur proportionnelle à la taille des échantillons, boxplots horizontaux.



Exemples de Boxplots

Rq: ajouter de l'information avec par exemple la densité sur le boxplot. C'est le cas des graphes "violin".



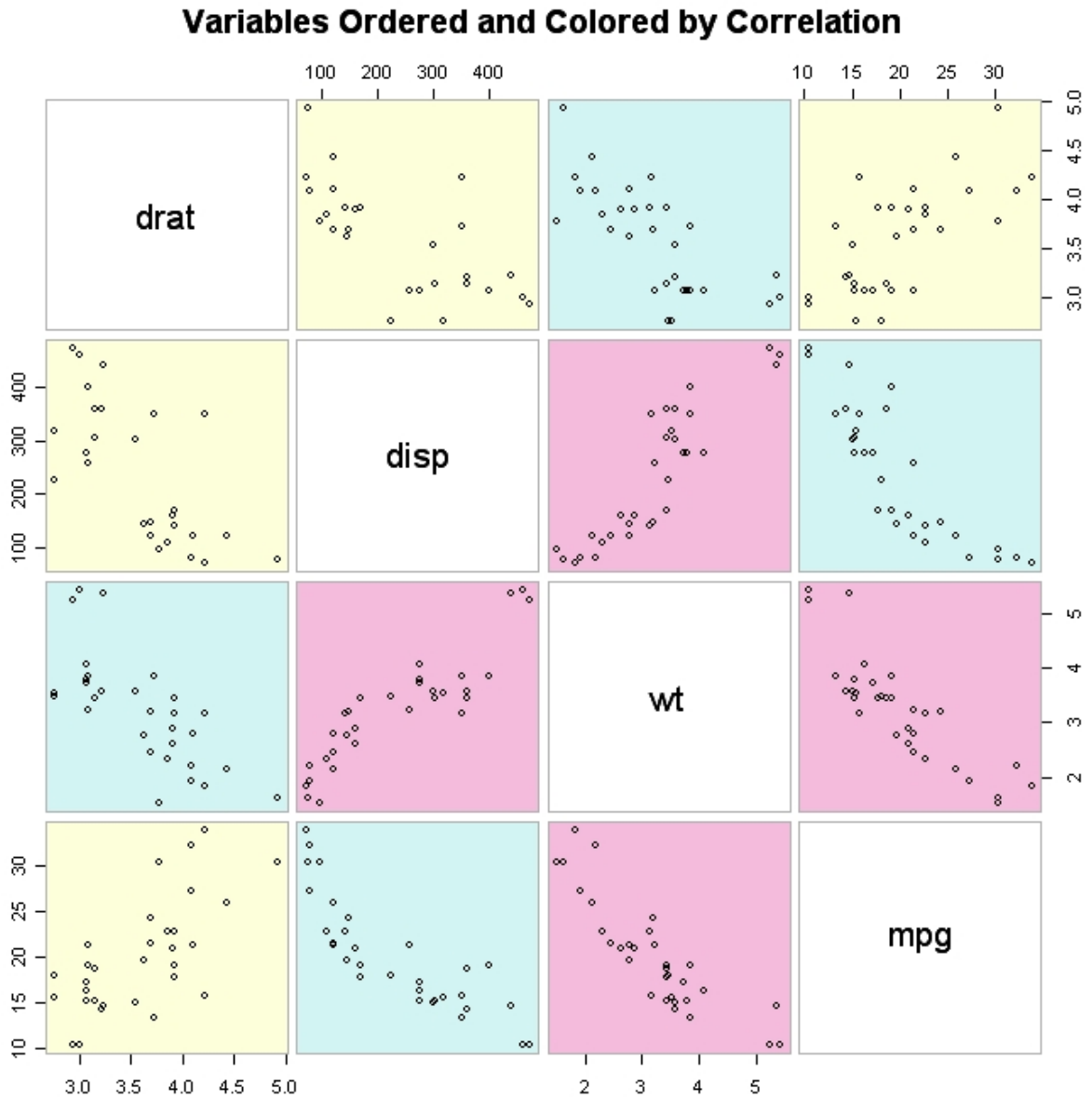


Les "scatterplot": les plus sophistiqués

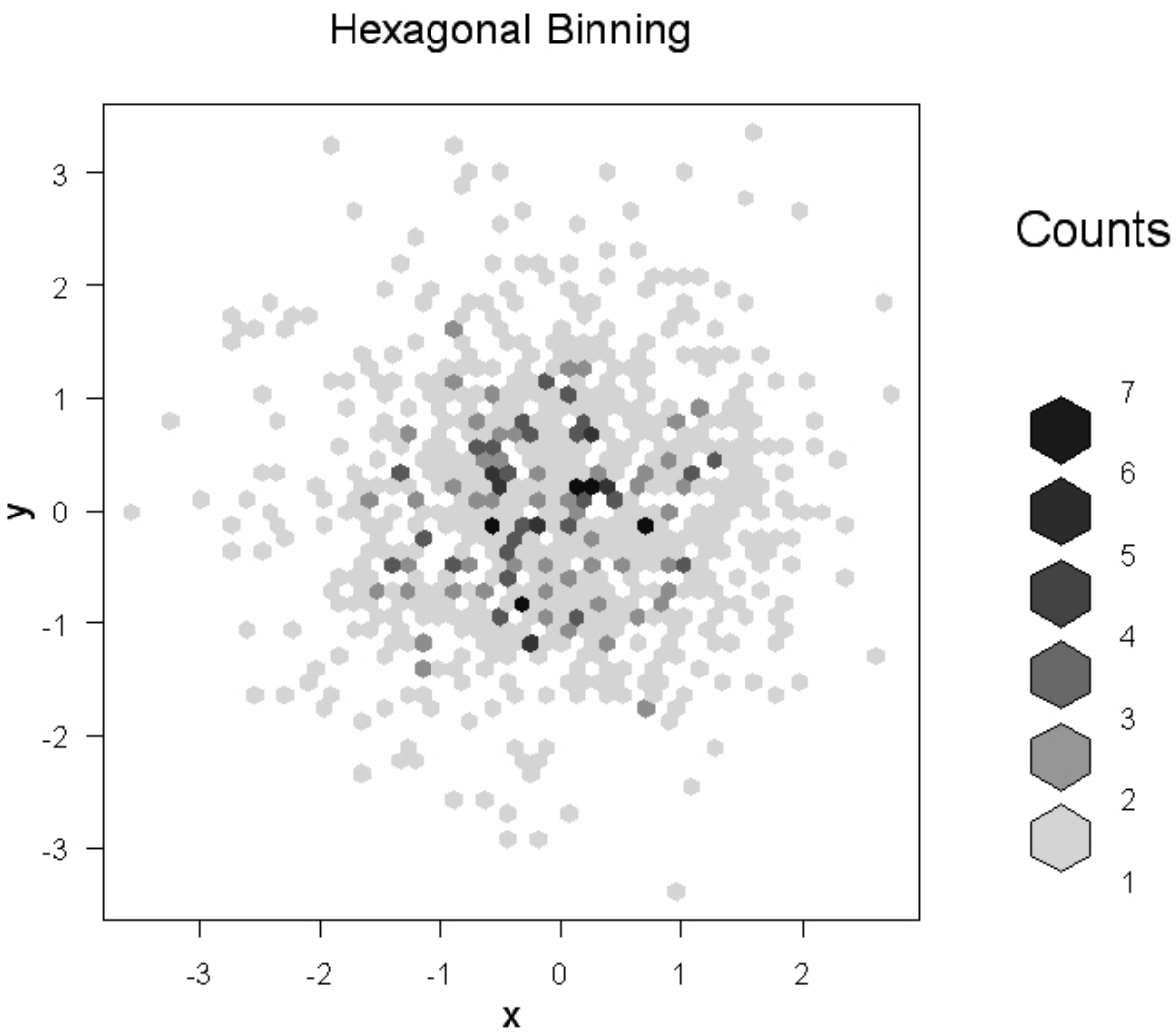
- Création simple: `plot(x,y)`, où x et y sont les vecteurs numériques des coordonnées des points.
- Plus d'options dans `scatterplot(formula,...)`, où *formula* est une formule.
- Matrice de Scatterplots par `pairs(formula)`: utiles pour une vision des dépendances entre variables...
- Plus d'options dans les librairies `lattice`, `gclus` et `car`.
- Lorsque la densité des points est trop importante, les scatterplots perdent de leur intérêt. Utiliser alors la librairie `hexbin` et les fonctions `hexbin()` ou `sunflowerplot()`.
- On peut aussi faire des graphiques en 3D.

Rq: utiliser `col2rgb()` pour connaitre le code couleur de R.

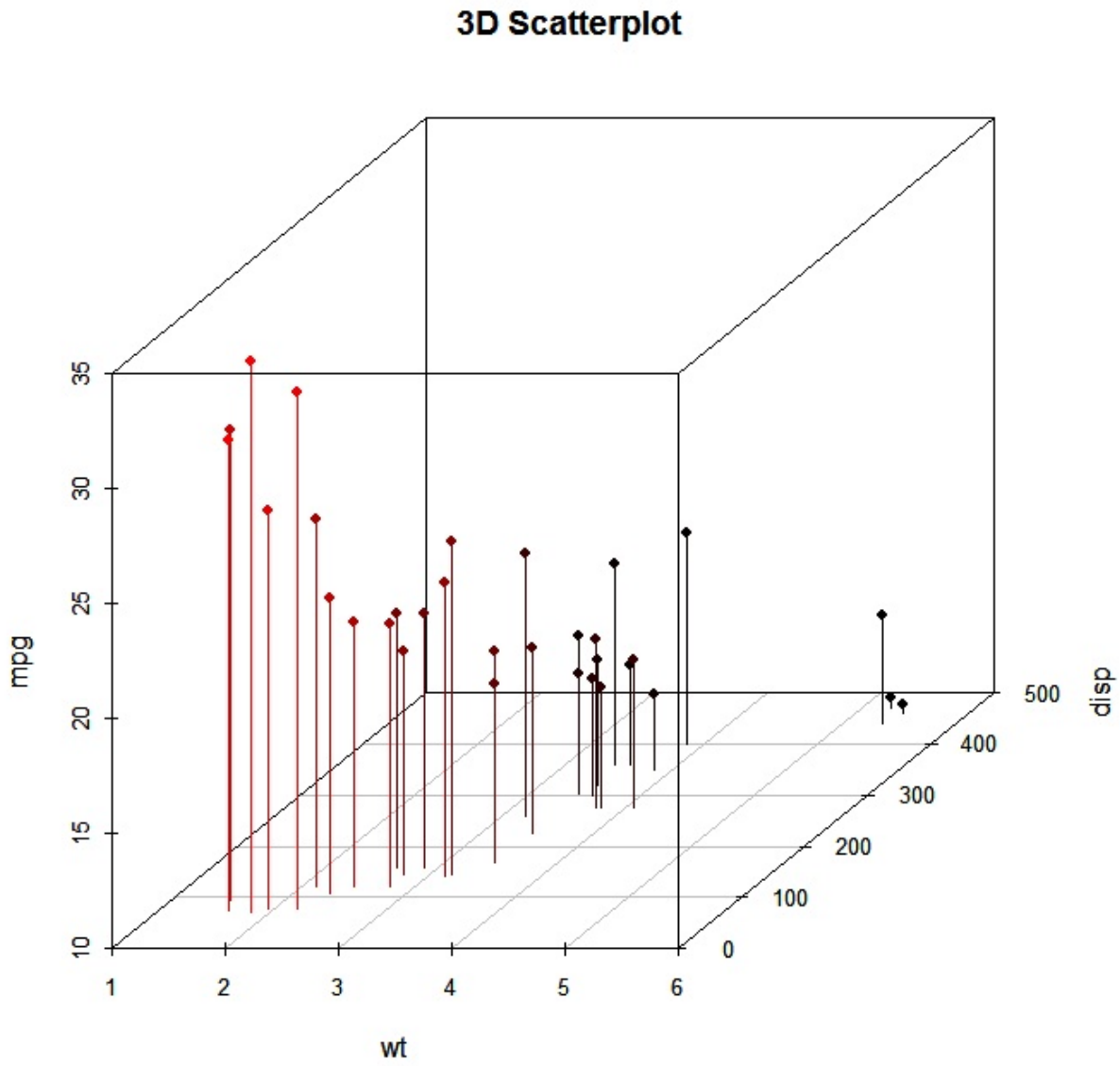
Exemples de scatterplots (1)



Exemples de scatterplots (2)



Exemples de scatterplots (3)



plot(...) et types d'objet (non exhaustif)

- sur une fonction (cosinus ici par ex.):

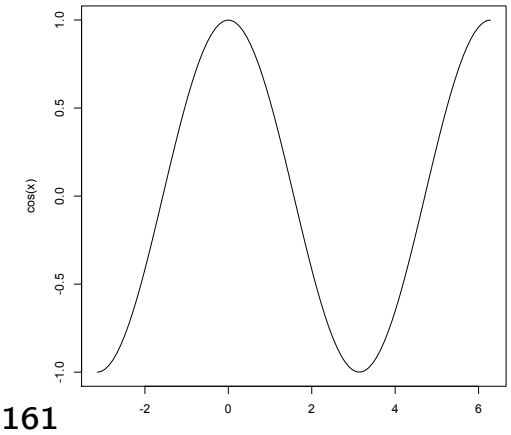
```
> plot(cos,xlim=c(-pi,pi))
```

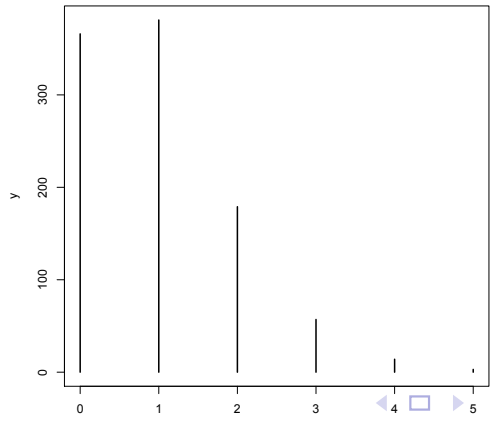
- sur un tableau:

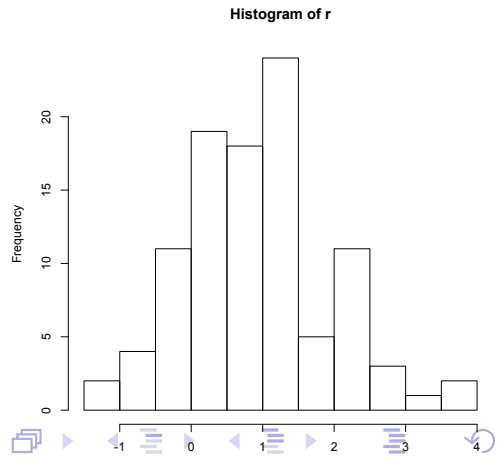
```
> x=rpois(1000,1)
> (y=table(x)) ; plot(y)
x
  0   1   2   3   4   5
366 381 179  57  14   3
```

- sur un histogramme:

```
> r=rnorm(100,1)
> z=hist(r,plot=F) ; plot(z)
```

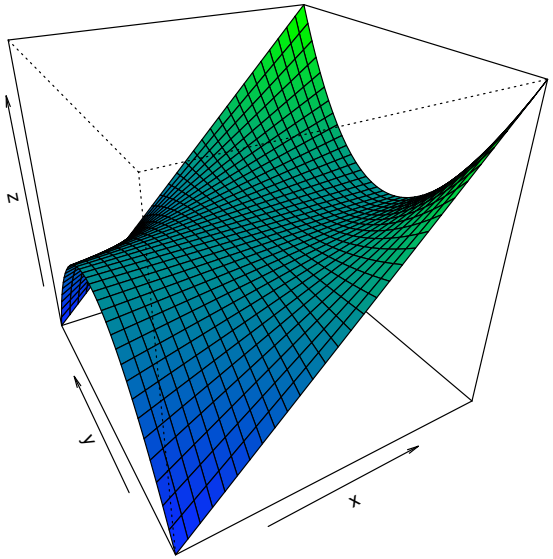


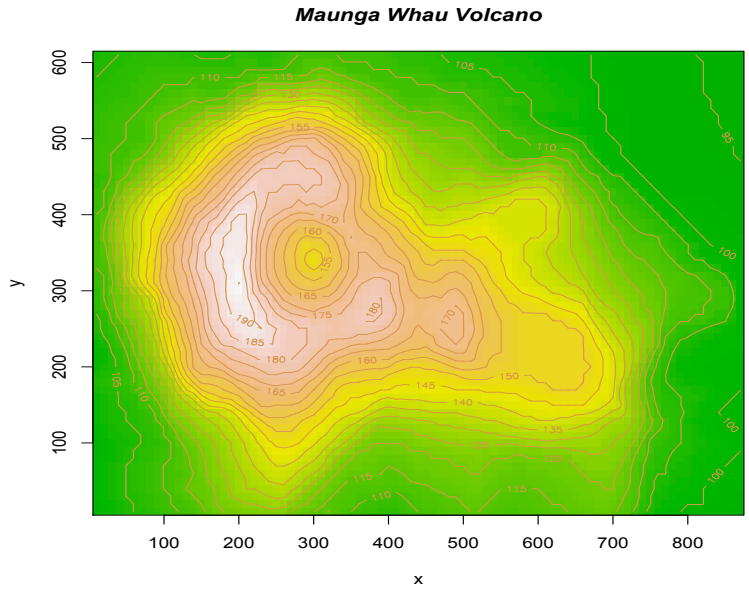




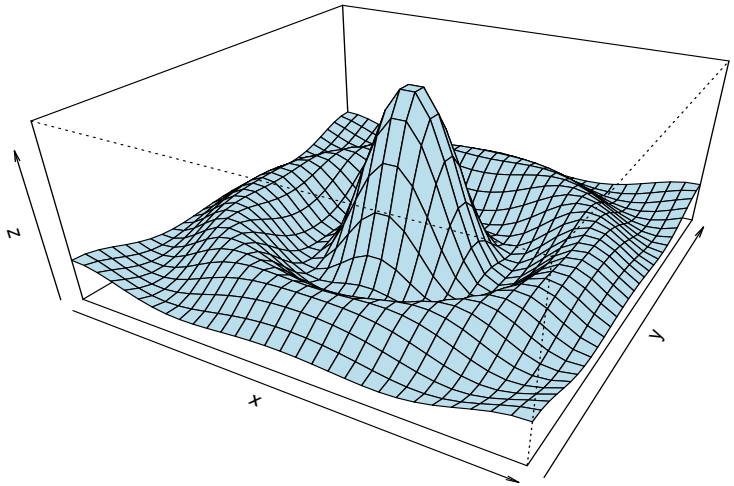
plot(...) et types d'objet 3D

- sur matrice: `contour(x,y,M)`, `image()` et `persp()`.





- sur une fonction de $\mathbb{R}^2$: utiliser `persp()`.



1. P-R-emier pas
   - Généralités
   - Fondamentaux
   - Manipulation
   - Graphiques
   - Probabilités
2. Statistiques desc-R-iptives
3. Exemples d'usages d'actuaire
4. Conclusion

Pré-requis

Rappel: Soit un espace probabilisé $(\Omega, \mathcal{F}, \mathbb{P})$. Notons X une variable aléatoire de loi $\mathbb{P}_X$, et E un événement de Ω.

On distingue alors:

- loi discrète: dans ce cas, $\mathbb{P}_X(E) = \sum_{x \in E} \mathbb{P}(X = x)$;
- loi continue: ici, $\mathbb{P}_X(E) = \int_{x \in E} f_X(x)dx$, où f_X est appelée la densité de X.

Les fonctions de R permettent d'obtenir:

1. la fonction de répartition: $F_X(x) = \mathbb{P}(X \leq x)$;
2. la densité de probabilité: $\begin{cases} \text{loi discrète: } \mathbb{P}(X = x) \\ \text{loi continue: } f_X(x) \end{cases}$;
3. les quantiles q_p où $p \in [0, 1]$: $q_p = \inf\{x : \ F_X(x) \geq p\}$;
4. effectuer des simulations de la loi.

Principales lois à disposition

Parmi les deux grandes familles de variables aléatoires, les plus usuelles sont

1. lois discrètes:
   - loi Binomiale $\mathcal{B}(n,p)$ (Bernoulli) `binom`
   - loi hypergéométrique `hyper`
   - loi de Poisson $\mathcal{P}(\mu)$ `pois`
   - loi géométrique $\mathcal{G}(\lambda)$ `geom`
   - loi à support fini $\{(a_i,p_i);\ i=1,...,I\}$ `sample`
2. lois continues:
   - loi normale $\mathcal{N}(\mu,\sigma^2)$ `norm`
   - loi uniforme $\mathcal{U}(a,b)$ `unif`
   - loi exponentielle $\mathcal{E}xp(\lambda)$ `exp`
   - loi de Student à ν degrés de liberté `t`
   - loi du χ^2 à ν degrés de liberté `chisq`

Nomenclature R

La nomenclature est standardisée.

Soit *Name* le suffixe correspondant à la loi considérée, on a:

1. la FdR au point x par `pName(x, ...)`;
2. densité de probabilité en x par `dName(x, ...)`;
3. quantile d'ordre p de la distribution par `qName(p, )`;
4. n simulations de la loi *Name* par `rName(n)`.

Rq: chaque loi a son jeu de paramètres spécifiques, consulter donc `?Name` pour plus de détails!

Exercice: soit $X \sim \mathcal{E}xp(5)$. On a donc $f_X(x) = 5e^{-5x}$.

- `pexp(x, 5)`: densité en x;
- `dexp(x, 5)`: FdR en x;
- `qexp(p, 5)`: quantile d'ordre p;
- `rexp(100, 5)`: 100 simulations de cette loi.

Simulation d'une loi discrète finie

Considérons une v.a. X à valeurs dans $\{x_1, ..., x_k\}$ t.q.

$$\mathbb{P}(X = x_i) = p_i, \quad \forall i = 1, ..., k.$$

La fonction `sample()` est un générateur de nombres aléatoires pour les lois discrètes à support fini.

La syntaxe est `sample(x, n, replace, p)`, avec:

- x est le vecteur des valeurs que peut prendre X;
- n est le nombre de valeurs simulées;
- *replace* est un booléen: s'il est vrai c'est un tirage avec remise, sinon il est sans remise (i.i.d.).
- p est le vecteur des probabilités d'apparition des x_i.

Rq: si p n'est pas spécifié, on simule une loi uniforme.

Exercice: simulation de 1000 réalisations indépendantes d'une loi uniforme à support $\{1, ..., 5\}$.

Simulation d'une loi discrète finie

Considérons une v.a. X à valeurs dans $\{x_1, ..., x_k\}$ t.q.

$$\mathbb{P}(X = x_i) = p_i, \quad \forall i = 1, ..., k.$$

La fonction `sample()` est un générateur de nombres aléatoires pour les lois discrètes à support fini.

La syntaxe est `sample(x, n, replace, p)`, avec:

- x est le vecteur des valeurs que peut prendre X;
- n est le nombre de valeurs simulées;
- *replace* est un booléen: s'il est vrai c'est un tirage avec remise, sinon il est sans remise (i.i.d.).
- p est le vecteur des probabilités d'apparition des x_i.

Rq: si p n'est pas spécifié, on simule une loi uniforme.

Exercice: simulation de 1000 réalisations indépendantes d'une loi uniforme à support $\{1, ..., 5\}$.

```
> sample((1:5), 1000, replace=T)
```

Remarque fondamentale: simulations Monte Carlo

- Les algorithmes de génération de nombres aléatoires sont basés sur une "graine" (point de départ);
- On peut fixer cette graine par `set.seed()`:

```
> set.seed(10) ; runif(1)
[1] 0.5074782
> set.seed(10) ; runif(1)
[1] 0.5074782
> runif(1)
[1] 0.3067685
```

Exercice: retrouver par la méthode de l'inverse de la FdR cette propriété en simulant une fois une loi normale standard.

Rq: utile pour calibrer un modèle sur apprentissage aléa.!

Remarque fondamentale: simulations Monte Carlo

- Les algorithmes de génération de nombres aléatoires sont basés sur une "graine" (point de départ);
- On peut fixer cette graine par `set.seed()`:

```
> set.seed(10) ; runif(1)
[1] 0.5074782
> set.seed(10) ; runif(1)
[1] 0.5074782
> runif(1)
[1] 0.3067685
```

Exercice: retrouver par la méthode de l'inverse de la FdR cette propriété en simulant une fois une loi normale standard.

```
> set.seed(10)
> pnorm(rnorm(1))  # methode inverse FdR
[1] 0.5074782
```

Rq: utile pour calibrer un modèle sur apprentissage aléa.!

Exercice 13

1. Générer une séquence de 1 à 10 par pas de 2.
2. Générer 10 000 réalisations d'une gaussienne de moyenne 5 et de variance 2.
3. En calculer la moyenne et l'écart-type empiriques.
4. Faites le graphique de la densité et de la fonction de répartition empirique de ces données sur la même fenêtre, en modifiant la couleur de 2^{eme} plan.

Solution:

Exercice 13

1. Générer une séquence de 1 à 10 par pas de 2.
2. Générer 10 000 réalisations d'une gaussienne de moyenne 5 et de variance 2.
3. En calculer la moyenne et l'écart-type empiriques.
4. Faites le graphique de la densité et de la fonction de répartition empirique de ces données sur la même fenêtre, en modifiant la couleur de 2^{eme} plan.

Solution:

```
seq(from = 1, to = 10, by = 2)
res <- rnorm(10000, 5, sqrt(2))
mean(res) ; sd(res)
split.screen(c(2,1))
screen(1)
plot(density(res))
screen(2)
ecdf(res)
plot(ecdf(res))
```

1 P-R-emier pas

2 Statistiques desc-R-iptives
- Généralités
- Types
- Indicateurs
- Corrélations
- Analyse multidimensionnelle

3 Exemples d'usages d'actuaire

4 Conclusion

Notion de base des stats descriptives

→ Elles permettent d'avoir une première vision du portefeuille,

→ Aucune hypothèse n'est requise,

→ Elles ne supposent aucun modèle sous-jacent ("tous les modèles sont faux"),

→ Elles permettent souvent une visualisation et une interprétation facile des résultats,

→ Elles guident les futurs choix de modélisation,

→ C'EST UNE ETAPE INDISPENSABLE!!!

1 P-R-emier pas

2 Statistiques desc-R-iptives
- Généralités
- Types
- Indicateurs
- Corrélations
- Analyse multidimensionnelle

3 Exemples d'usages d'actuaire

4 Conclusion

Variables discrètes

1) A valeurs dans un ensemble fini ou dénombrable,
$\Rightarrow$ dans un échantillon de taille n, l'ensemble des valeurs prises par cette variable est forcément fini.

2) On distingue deux types de variables discrètes:

- qualitatives: s'expriment par l'appartenance à une catégorie (ex: cépage),
- quantitatives: s'expriment par des nombres réels (ex: nb d'enfants).

Rq: on peut créer des variables qualitatives à partir de variables quantitatives avec la fonction `cut()`.

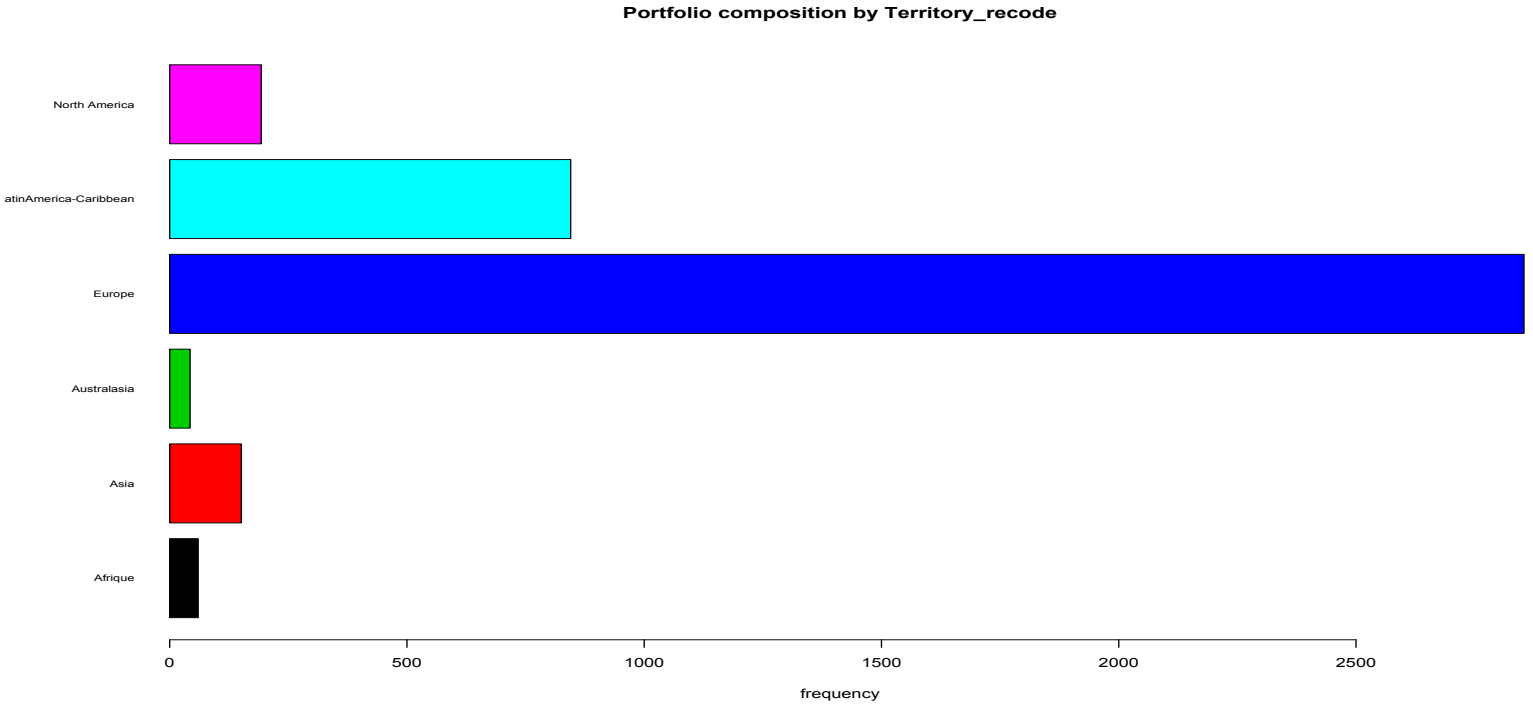
On utilise comme représentations graphiques pour les variables qualitatives:

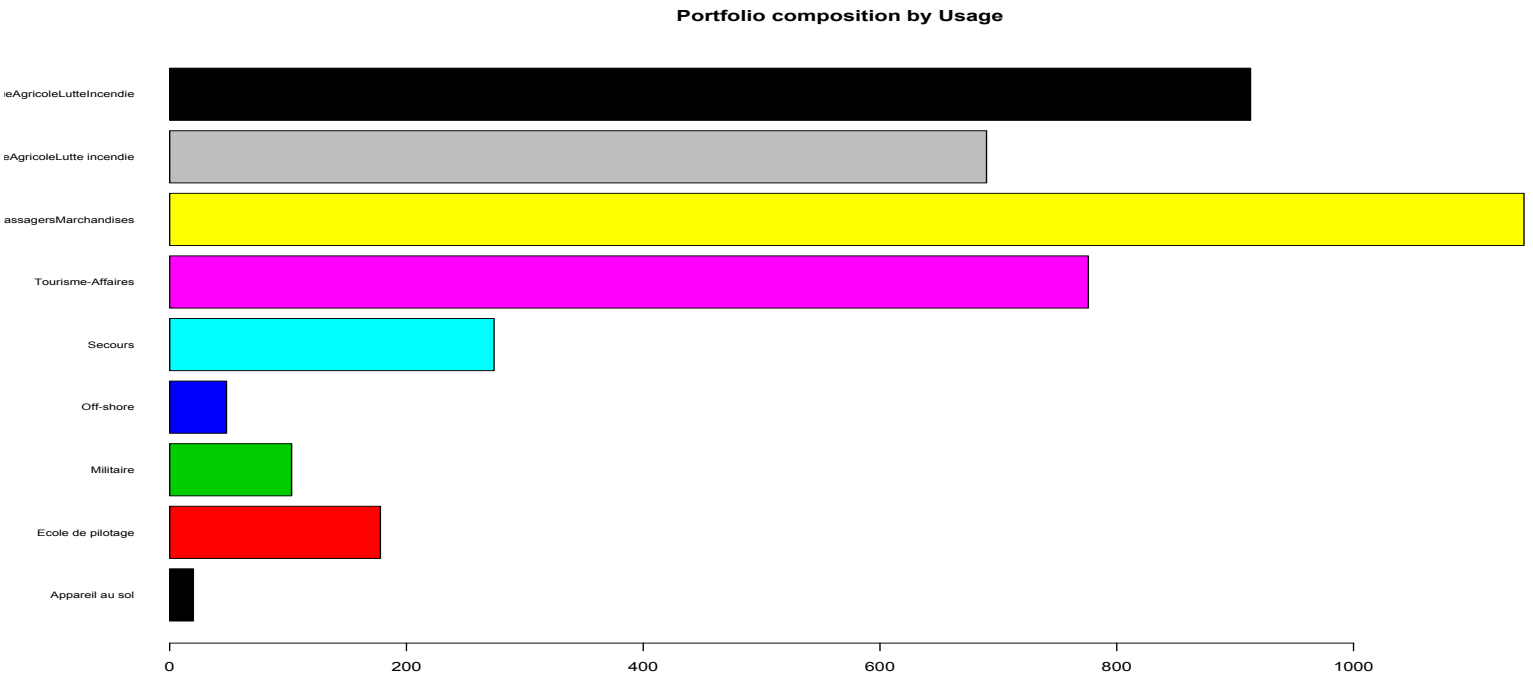
- diagrammes en bâtons: hauteur proportionnelle à la fréquence relative de chaque modalité;
- camemberts: un secteur de disque pour chaque modalité, avec une aire correspondante à la fréquence relative de la modalité.

Commandes `R` correspondantes:

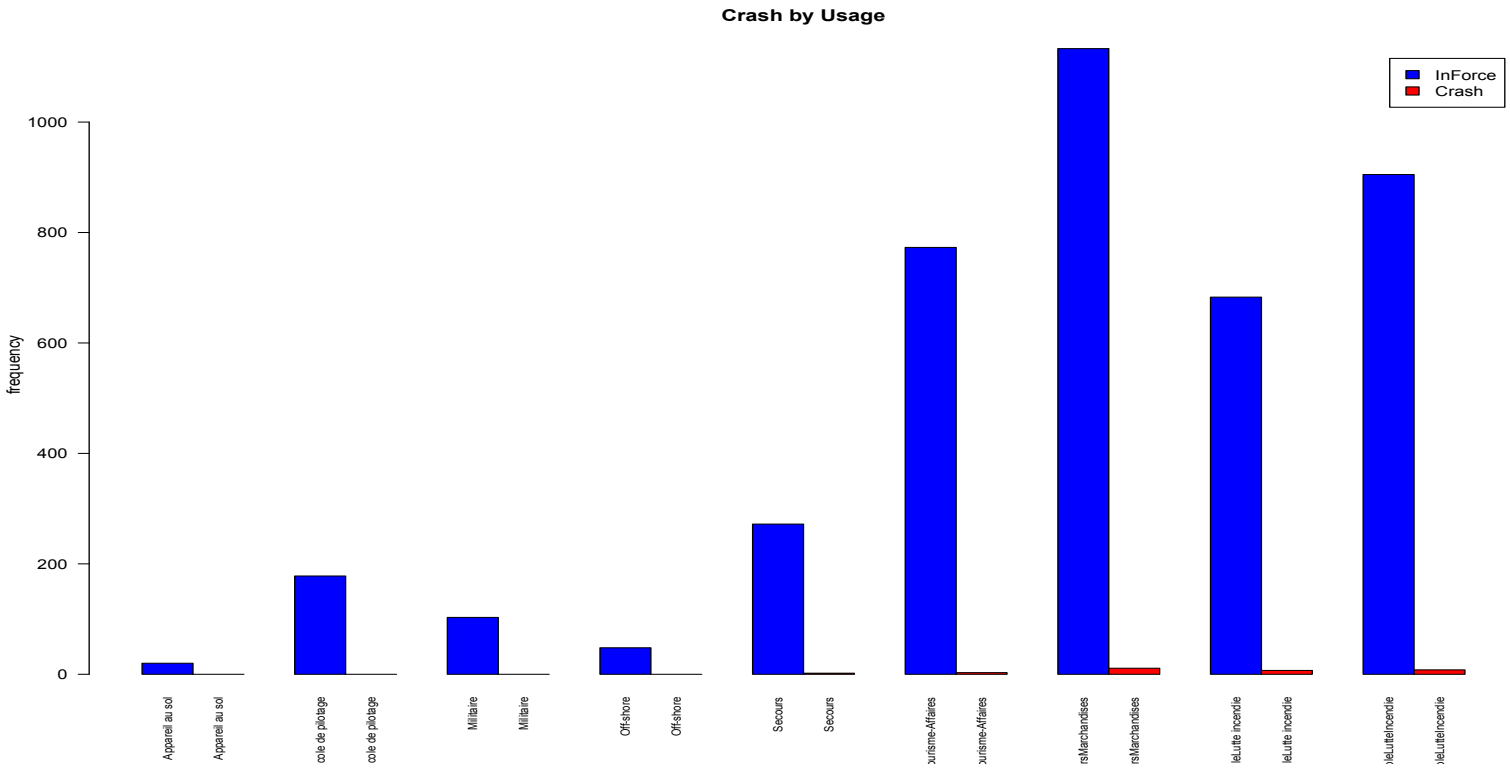
- diagramme en bâtons: `barplot()`;
- camemberts: `pie()`.

Remarque: pour les variables quantitatives, ce sont les mêmes représentations (avec fréquence absolue et relative) à part qu'il existe un ordre naturel sur les modalités.


Portfolio composition by Territory_recode
North America
atinAmerica-Caribbean
Europe
Australasia
Asia
Afrique
0
500
1000
1500
2000
2500
frequency


Portfolio composition by Usage
eAgricoleLutteIncendie
sAgricoleLutte incendie
assagersMarchandises
Tourisme-Affaires
Secours
Off-shore
Militaire
Ecole de pilotage
Appareil au sol
0
200
400
600
800
1000
frequency

Croisement avec l'événement à observer



	InForce	Crash	Crash rate (in %)
Appareil au sol	20	0	0.0000000
Ecole de pilotage	178	0	0.0000000
Militaire	103	0	0.0000000
Off-shore	48	0	0.0000000
Secours	272	2	0.7299270
Tourisme-Affaires	773	3	0.3865979
Transport-Passag.March.	1133	11	0.9615385
TravailAerien-Elingue	683	7	1.0144928
TravailAerien-saufElingue	905	8	0.8762322

Variables continues

Les données sont en général toutes distinctes les unes des autres. Les fréquences absolues valent toutes 1, d'où un autre type de représentation:

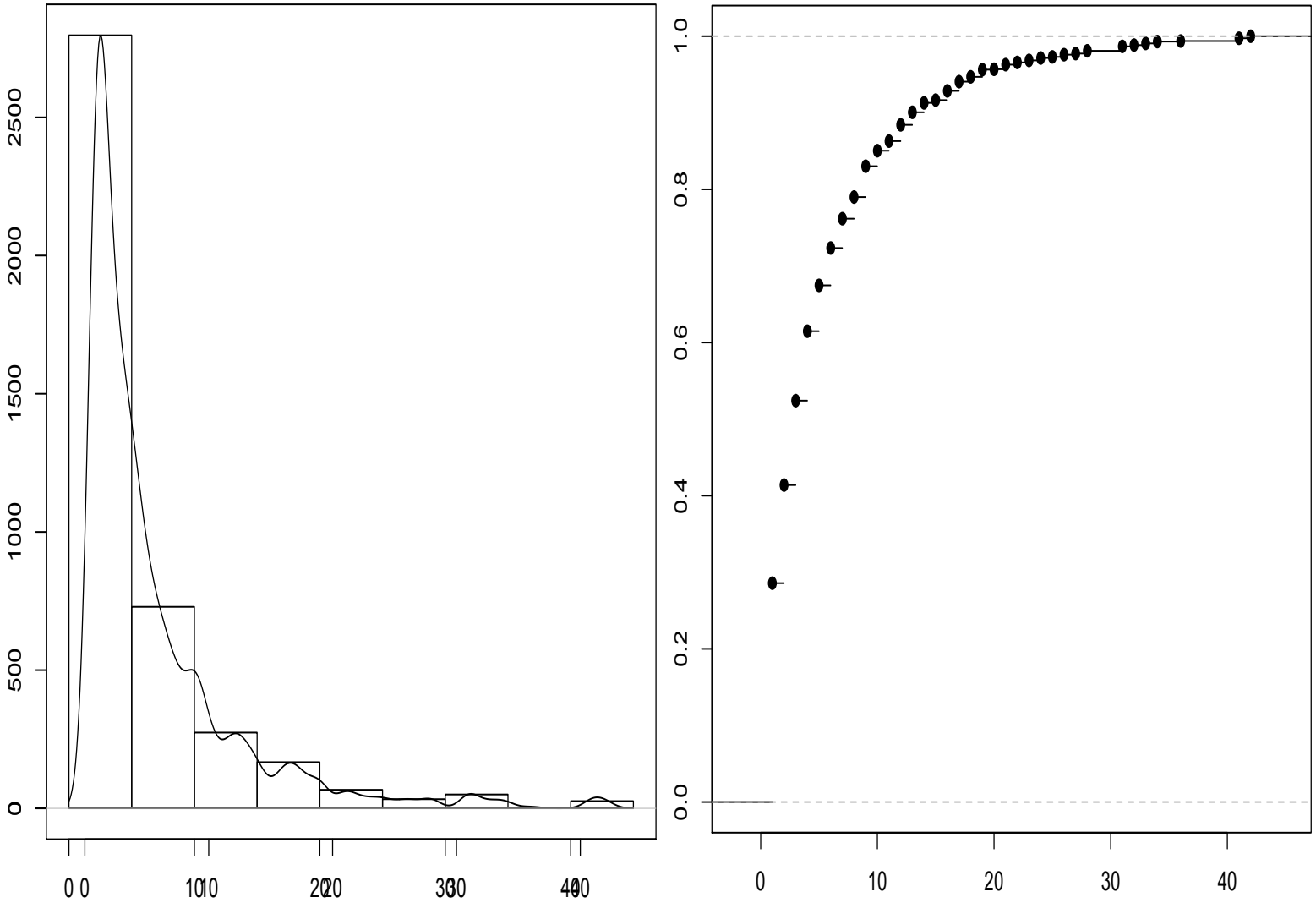
- Histogramme et polygone des fréquences: formation de classe de même largeur ou de même effectif;
- Fonction de répartition empirique et graphes de probabilités: .

Commandes `R` correspondantes:

- Histogramme: `hist()` et poly. des fréquences: `lines()`;
- FdR empirique: `ecdf()`, à intégrer dans la fonction `plot()`.

Remarque: il faut préalablement ordonner les données!

Exemple: nombre d'hélico par propriétaire



1 P-R-emier pas

2 Statistiques desc-R-iptives
- Généralités
- Types
- Indicateurs
- Corrélations
- Analyse multidimensionnelle

3 Exemples d'usages d'actuaire

4 Conclusion

Plusieurs indicateurs statistiques permettent de décrire les données, de manière plus ou moins fine (détection d'asymétrie, de points aberrants...).

Commandes R correspondantes:

1. indicateurs de tendance centrale:
   - moyenne empirique: `mean()`,
   - médiane empirique: `median()`,
2. indicateurs de valeurs extrêmes: `min()` et `max()`,
3. indicateurs de dispersion:
   - écart-type empirique: `sd()`,
   - variance empirique non-biaisée: `var()`,
   - étendue: `max()` - `min()`,
   - quantile: `quantile()`.

Remarque: la commande `summary()` permet de résumer la plupart de ces grandeurs!

Commandes summary() et quantile()

Ces commandes permettent de résumer certaines statistiques concernant la répartition des observations:

```
[1] "###### Variable NB_HELICO_ANNEE"
$summary
   Min.  1st Qu.   Median  Mean 3rd Qu.    Max.
 0.00027  0.2712  0.5992  1.172  1.004    29

$quantile
  quantile(myData[, j], probs=c(0.25,0.5,0.75,0.9))
25\%                                      0.271230
50\%                                      0.599180
75\%                                      1.003768
90\%                                      2.707400
```

Exercice 6

1. Etablir une description succinte mais complète de ces données? Remarquer la différence de résumé suivant le type de variable.
2. En donner une présentation graphique. Cela permet par exemple de vérifier la linéarité entre variables quantitatives avant une ACP...
3. Représenter graphiquement l'ensemble des relations x-y des données quantitatives.

Solution:

Exercice 6

1. Etablir une description succinte mais complète de ces données? Remarquer la différence de résumé suivant le type de variable.
2. En donner une présentation graphique. Cela permet par exemple de vérifier la linéarité entre variables quantitatives avant une ACP...
3. Représenter graphiquement l'ensemble des relations x-y des données quantitatives.

Solution:

```
summary(Insurance)
pairs(Insurance)
plot(x=Holders, y=Claims)
```

Exercice 7

1. Extraire les individus dont l'âge appartient à la tranche $[25; 29]$.
2. Quelles sont les moyennes des variables pour ces individus?
3. Quelles sont les moyennes des variables quantitatives pour chaque niveau de "Group"? Faites le calcul par 3 méthodes différentes (`tapply()`, `by()` et `aggregate()`).
4. Idem pour la variance.

Solution:

Exercice 7

Solution:

```
new.data <- Insurance[which(Age == "25-29"), ]
dim(new.data)
new.data
summary(new.data)
mean(Holders)
mean(new.data$Holders)
mean(Claims)
mean(new.data$Claims)
aggregate(Insurance[ ,(ncol(Insurance)-1):ncol(Insurance)], l
tapply(Claims, Group, mean)
by(Claims, Group, mean)
aggregate(Insurance[ ,(ncol(Insurance)-1):ncol(Insurance)], l
```

Exercice 8

1. Editer le jeu de données `Insurance`. Y introduire une donnée manquante (NA) dans la colonne "Claims" et enregistrer ce nouveau jeu de données sous le nom "new.data2".
2. Calculer la nouvelle moyenne des sinistres. En donner la somme.
3. Supprimer cette donnée manquante et recalculer ces quantités.

Solution:

Exercice 8

Solution:

```
rm(new.data)
new.data <- edit(Insurance)
new.data
# ou
new.data <- Insurance
new.data[6,5] <- NA
mean(new.data$Claims, na.rm = TRUE)
sum(new.data$Claims, na.rm = T)
mean(Insurance$Claims, na.rm = TRUE)
mean(Claims, na.rm = TRUE)
mean(Claims)
dim(new.data)
new.data2 <- na.omit(new.data)
dim(new.data2)
mean(new.data2$Claims)
```

Exercice 9

1. Combien y'a-t-il de possesseurs de véhicules par classe d'âge? Créer un tableau croisé.
2. Combien y'a-t-il de sinistres par classe d'âge?
3. Donner le taux de sinistralité par classe d'âge. Comparer avec celui établi dans le jeu de données avec introduction de NA.

Solution:

Exercice 9

1. Combien y'a-t-il de possesseurs de véhicules par classe d'âge? Créer un tableau croisé.
2. Combien y'a-t-il de sinistres par classe d'âge?
3. Donner le taux de sinistralité par classe d'âge. Comparer avec celui établi dans le jeu de données avec introduction de NA.

Solution:

```
xtabs(Claims~Age)
xtabs(Holders~Age)
class(xtabs(Holders~Age))
xtabs(Claims~Age) / xtabs(Holders~Age)
with(Insurance, {xtabs(Claims~Age) / xtabs(Holders~Age)})
with(new.data2, {xtabs(Claims~Age) / xtabs(Holders~Age)})
```

Exercice 10

1. Faire un histogramme de 15 classes du nombre de sinistres. Comment récupérer les bornes des classes? Comparer avec le barplot correspondant. Attention donc à ce que nous voulons représenter!
2. Observer un résumé statistique graphique de la répartition du nombre de sinistres par classe d'âge. Y ajouter les observations par la fonction `rug()`.
3. Quelle est la répartition en camembert des "District" ?
4. Ajouter un titre et une légende à ce graphique. Sauvegarder le en format .jpg dans votre répertoire de travail.
5. Etablir une description (boxplot, histogramme, densité) de la variable du nombre de sinistres dans la même fenêtre graphique en utilisant `par()`, puis sans l'utiliser (voir avec `layout()`).

1 P-R-emier pas

2 Statistiques desc-R-iptives
- Généralités
- Types
- Indicateurs
- Corrélations
- Analyse multidimensionnelle

3 Exemples d'usages d'actuaire

4 Conclusion

En multidimensionnel...

L'étude des corrélations entre variables peut se révéler clef lors de l'utilisation future de ces variables dans un modèle (GLM...).

Etudier la dépendance entre 2 séries de valeurs réelles x et y issues de deux variables aléatoires distinctes X et Y par:

- corrélation de Pearson:
  - hypothèse de linéarité (basé sur l'opérateur covariance),
  - hypothèse de normalité,
  - commande R: `cor()`,
- taux de Kendall:
  - coefficient basé sur les rangs des observations $\Rightarrow$ non-paramétrique,
  - commande R: `Kendall()`,

Remarque: il y a aussi le rho de Spearman (non-param.): à la main avec `cor()`. Kendall / Spearman aussi en dim. > 1.

Corrélation de Pearson

```
> cor(myData[,c("Charge","NB_HELICO_ANNEE","NB_HELICO_FLOTTE")],
+   use="complete")
                    Charge NB_HELICO_ANNEE NB_HELICO_FLOTTE
Charge           1.0000000       0.0399417        0.0504382
NB_HELICO_ANNEE  0.0399417       1.0000000        0.4906422
NB_HELICO_FLOTTE 0.0504382       0.4906422        1.0000000
```

Corrélation de Spearman

```
> cor(myData[,c("Charge","NB_HELICO_ANNEE","NB_HELICO_FLOTTE")],
+   use="complete", method="spearman")
                     Charge NB_HELICO_ANNEE NB_HELICO_FLOTTE
Charge           1.00000000      0.05331429        0.0401257
NB_HELICO_ANNEE  0.05331429      1.00000000        0.2842193
NB_HELICO_FLOTTE 0.04012570      0.28421931        1.0000000
```

Test d'indépendance du Chi-deux

Ce test s'applique sur un tableau de contingence, donc nécessite de disposer de variables X et Y catégorielles. On doit:

- créer un tableau croisé: `table(don$X, don$Y)`;
- (afficher les noms des variables du tableau:) `table(don$X, don$Y, deparse.level=2)`;
- préciser le nom des variables par `table(don$X, don$Y, dnn=c("bla","blo"))`;
- tester l'indépendance par:

```
test.data <- table(don$X, don$Y)
summary(test.data)
```

- ou simuler un tirage aléatoire:

```
chisq.test(test.data, simulate.p.value=T, B=10000)$p.value
```

1 P-R-emier pas

2 Statistiques desc-R-iptives
- Généralités
- Types
- Indicateurs
- Corrélations
- Analyse multidimensionnelle

3 Exemples d'usages d'actuaire

4 Conclusion

Réduction de dimension (librairie R ade4)

Ces méthodes sont descriptives. Intérêt: avoir une représentation graphique de l'information contenue dans les données en réduisant la dimension de l'espace des variables.

Méthodes de visualisation...

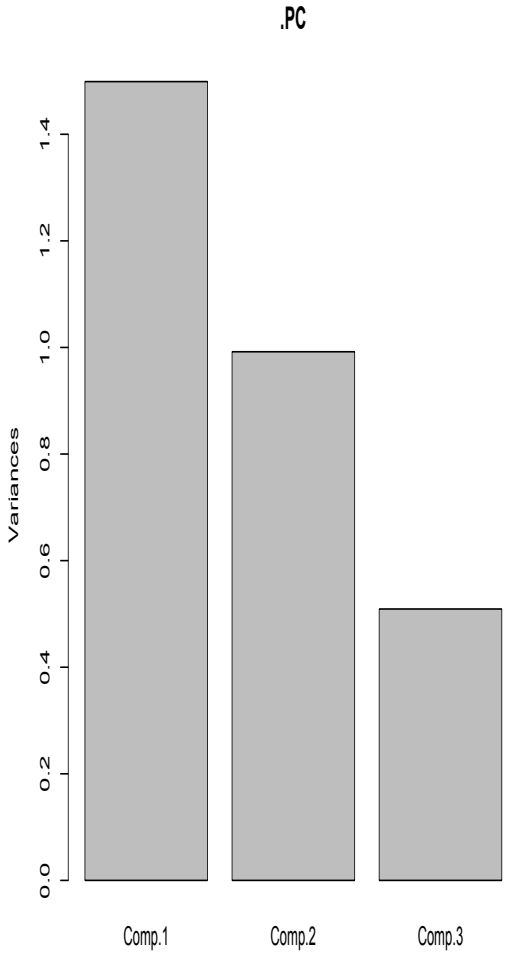
- ...sur variables continues: analyse en composantes principales (ACP), commande R: `dudi.pca()`,
- ...sur variables catégorielles:
  - analyse factorielle des correspondances (AFC), commande R: `dudi.coa()`,
  - analyse factorielle des correspondances Multiples (AFCM), commande R: `dudi.coa()` et `acm.disjonctif()`.

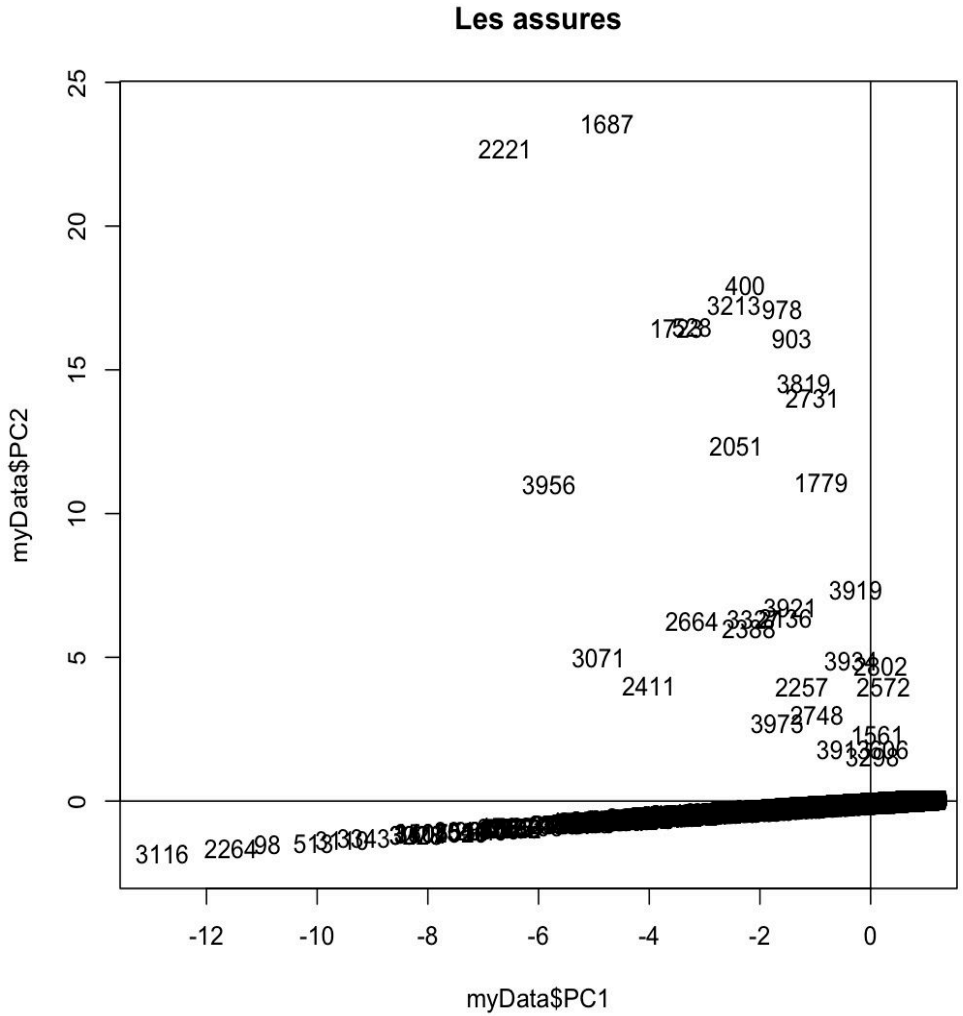
- Objectif: minimiser la déformation de l'info. d'origine due à la réduction de dimension.
- Visualisation: dans le plan (dim. 2), appelé plan principal.

FactoMineR: ACP aux données d'hélicoptères

```
> .PC <- princomp(~Charge+NB_HELICO_ANNEE+NB_HELICO_FLOTTE,cor=TRUE,
+   data=myData)

> unclass(loadings(.PC))  # component loadings
                     Comp.1      Comp.2      Comp.3
Charge           -0.1270913  0.99177376 -0.01525149
NB_HELICO_ANNEE  -0.7006984 -0.10065314 -0.70632191
NB_HELICO_FLOTTE -0.7020466 -0.07908068  0.70772647

> .PC$sd^2  # component variances
   Comp.1    Comp.2    Comp.3
1.4988308 0.9919246 0.5092446

> summary(.PC) # proportions of variance
Importance of components:
                          Comp.1    Comp.2    Comp.3
Standard deviation     1.2242674 0.9959541 0.7136138
Proportion of Variance 0.4996103 0.3306415 0.1697482
Cumulative Proportion  0.4996103 0.8302518 1.0000000

> screeplot(.PC)
> plot(myData$PC1, myData$PC2, type = "n", main="Les assures")
> abline(h=0, v=0)
> text(myData$PC1, myData$PC2, rownames(myData))
```

Eboulis et plan principal





Exercice 11

1. Ajouter une colonne représentant le taux de sinistralité. La nommer "Taux" et vérifier son type.
2. Calculer la corrélation de Pearson entre le nb de "Holders" et le nb de sinistres. Idem entre "Holders" et le taux de sinistralité (indique var. à exclure ds régression).
3. Construire une régression linéaire de ce taux par toutes les variables explicatives à disposition (`lm()`).
4. Que contient le modèle? Comment accède-t-on à ses coefficients? Effectuer des prévisions.
5. Représenter graphiquement les résidus.
6. Afficher les résultats de cette modélisation. Conclure.
7. Réaliser une analyse de variance d'un nombre de sinistres expliqué par l'âge, le groupe et leurs effets croisés. Tester la normalité des résidus.
8. Construire une régression linéaire où le taux est expliqué uniquement par la classe d'âge. Donner une représentation graphique des données empiriques.
9. Construire un modèle dans lequel le taux de sinistralité

Exercice 11

```
data <- data.frame(Insurance, data.frame(Claims/Holders))
names(data)[ncol(data)] <- "Taux"
class(data$Taux)
reg.lin <- lm(Taux ~ District + Group + Age)
summary(reg.lin)
attributes(reg.lin)
coef(reg.lin)
predict(reg.lin, newdata = data)
plot(reg.lin$residuals)
obj <- aov(Taux ~ Group*Age)
shapiro.test(obj$residuals)
plot(obj$residuals)
hist(obj$residuals, probability=TRUE, xlab="", main="Les rÕsi
reg.lin <- lm(Taux ~ Age)
plot(data$Age, data$Taux)
reg.lin <- lm(Taux ~ Holders)
plot(data$Holders, data$Taux)
abline(reg.lin)
```

Exercice 14

L'objectif est de créer une fonction qui permettent de réaliser le programme suivant:

```
# lecture des donnees
donnees <- Insurance
# regression lineaire
z <- aov(donnees$Claims ~ donnees$Age)
# graphe XY
plot(donnees$Age , donnees$Claims , ylab="Nb de sinistres", xl
```

Solution:

Exercice 14

L'objectif est de créer une fonction qui permettent de réaliser le programme suivant:

```
# lecture des donnees
donnees <- Insurance
# regression lineaire
z <- aov(donnees$Claims ~ donnees$Age)
# graphe XY
plot(donnees$Age , donnees$Claims , ylab="Nb de sinistres", xl
```

Solution:

```
graphe2D <- function(X, Y, data) {
attach(data)
modele <- aov(Y ~ X)
titre <- paste("Relation entre ", substitute(X), " et ", subst
plot(X, Y, ylab = substitute(Y), xlab = substitute(X), main =
}
graphe2D(Claims, Age, Insurance)
```

1 P-R-emier pas

2 Statistiques desc-R-iptives

3 Exemples d'usages d'actuaire
- Classification
- Un modèle économétrique classique: le cas des GLM
- Modèles de crédibilité

4 Conclusion

Généralités

Les méthodes de classification ont principalement 2 objectifs:

- réduire la dimension: soit du nombre d'observations, soit du nombre de variables explicatives;
- établir un regroupement des individus afin de constituer des groupes homogènes (clustering).

$\Rightarrow$ Il existe de nombreuses librairies permettant de réaliser un processus de classification en R: `cluster`, `fastcluster`, `flashClust`, `skmeans`, `flexclust`, `rpart`...

$\Rightarrow$ Nous utiliserons très souvent par la suite des fonctions comprises dans les bibliothèques standards de R!

Principe: aggrégations successives d'objets (d'individus) par le calcul d'une métrique (ex: distance).

Pour notre culture générale...en non-hiérarchique!

Le nombre de groupes / classes doit être spécifié en amont de la construction de l'arbre.

Il existe principalement deux méthodes, dont l'avantage est leur relative simplicité de mise en oeuvre:

- les k-plus proches voisins: aggrégation autour de barycentres avec mise à jour, commande R: `kmeans()`;
- généralisation des k-plus proches voisins où le barycentre est défini par un groupe d'individus (cf `flashClust`).

Inconvénient: avoir une bonne idée au préalable du nombre de classes, sinon le résultat peut être surprenant!

Avantage: simplicité de mise en oeuvre, large panel d'outil open-source.

CAH: la classification hiérarchique ascendante

C'est la plus utilisée des méthodes de classification. Ses caractéristiques sont:

- Méthode par aggrégation;
- Selon le critère d'aggrégation choisi, le résultat est souvent très différent;
- Lors de la classification, on calcule la distance entre des groupes d'individus;
- En pratique, c'est souvent le critère de la distance euclidienne qui est considéré;
- Il n'existe pas de moyen de connaître le critère optimal pour un jeu de données.

$\Rightarrow$ Regroupement d'individus homogènes, réduction de la dimension des individus.

Mise en oeuvre avec R: *dist*() et *hclust*()

$\exists$ plusieurs mesures de distance/critères d'aggrégation:

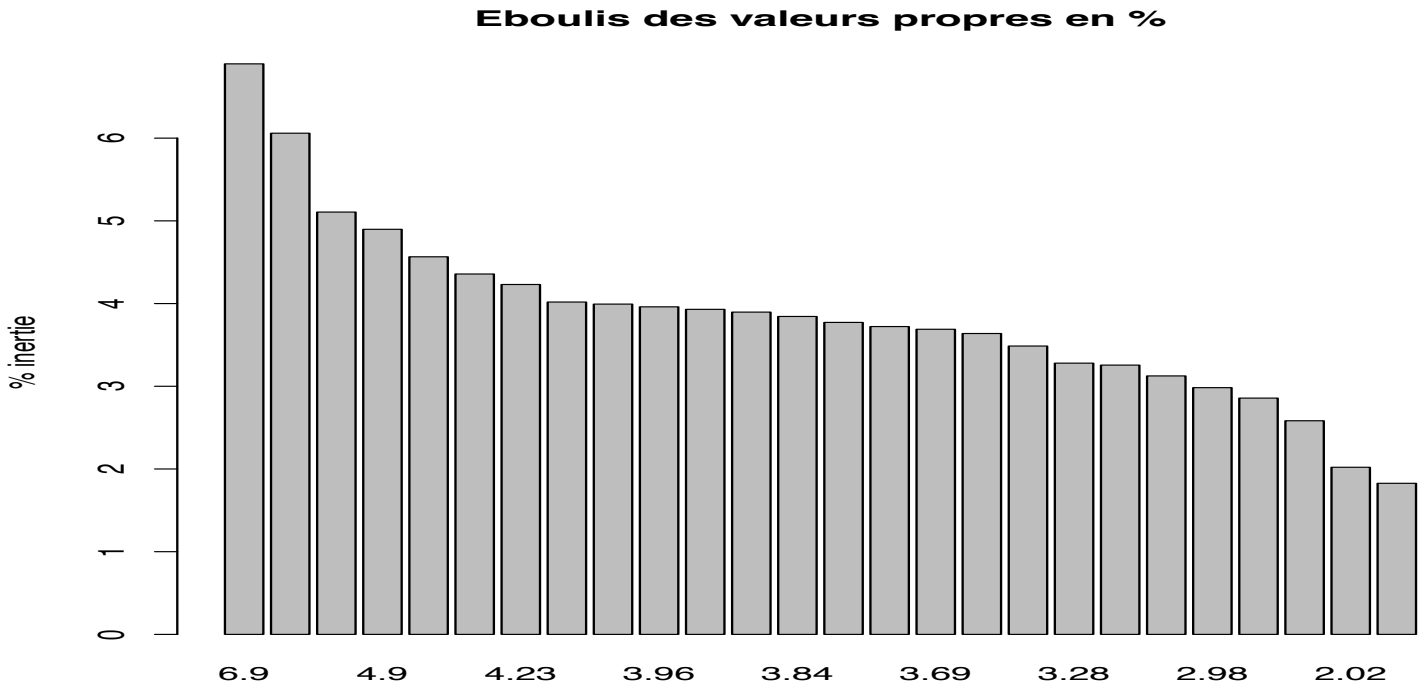
- distance: euclidienne, euclidienne^2, Manhattan...
- aggrégation: simple, liens moyens, Ward, McQuitty...

Ici, données qualitatives et quantitatives $\Rightarrow$ 1ere ètape: AFCM pour obtenir par la suite un tableau de distance.

```
>  head(myData, n=3)
  Assure          Type_recode          Usage          Annee    Territory
  55    Turbine - Mono-moteur<2500 Appareil au sol  2007      Europe
  255          Turbine - Bi-moteur Appareil au sol  2007     Europe
  160    Turbine - Mono-moteur<2500 Appareil au sol  2008      Asia
...
Departement NB_HELICO_FLOTTE NB_HELICO_ANNEE NB_Accidents Charge
  AGF                  1          1.00000              0      0
   AGF                 2          2.00548              0      0
   AGI                 1          0.50137              0      0
```

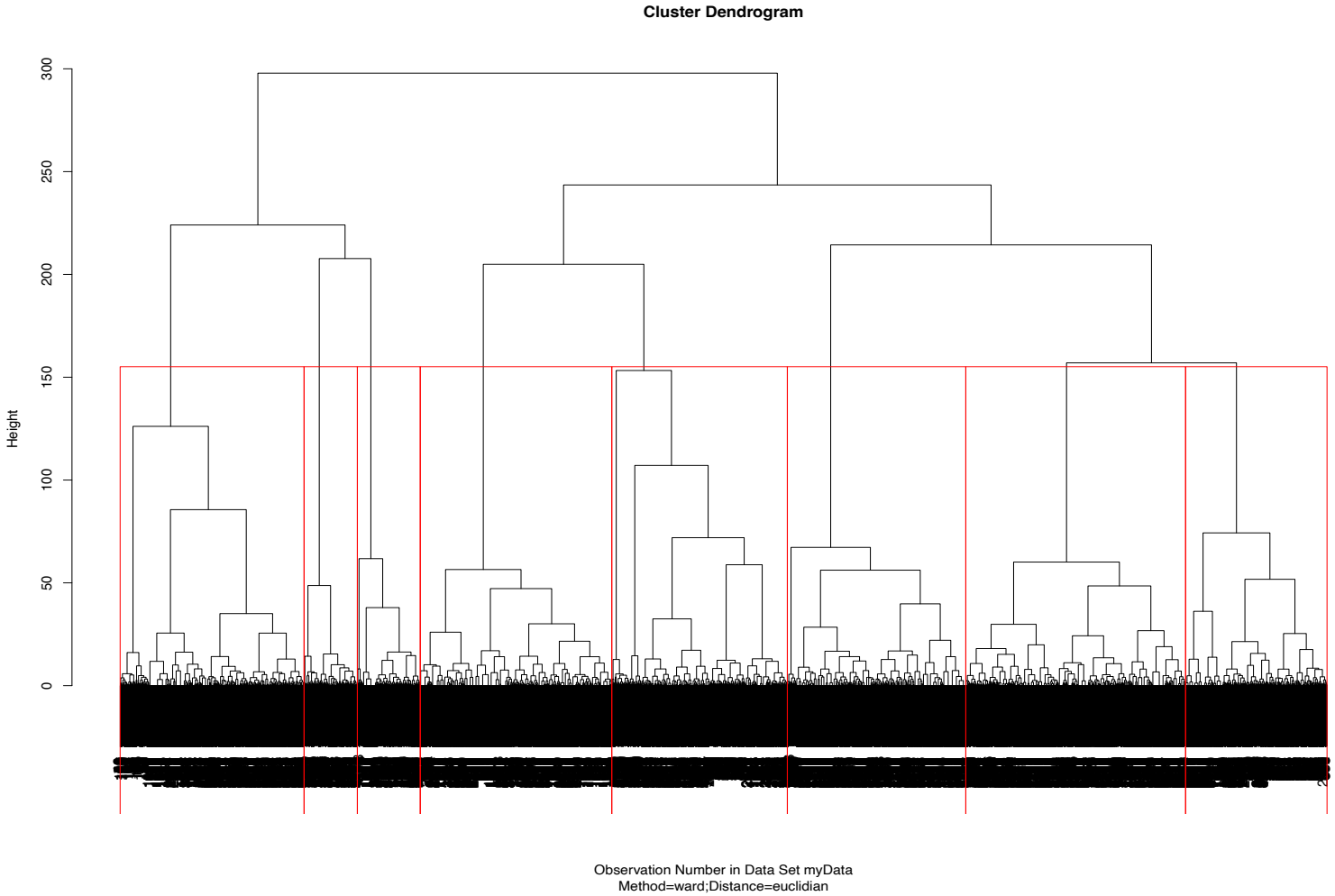
Construction de l'AFCM et de la CAH:

```
 # Construction du tableau disjonctif
> myData$NB_HELICO_FLOTTE <-
+          categorisation.variableNum(myData$NB_HELICO_FLOTTE)
> myData$NB_HELICO_ANNEE <-
+          categorisation.variableNum(myData$NB_HELICO_ANNEE)
> library(ade4)
> tableau.disjonctif <- acm.disjonctif(myData[ ,-c(1,10)])
> resultats.AFC <- dudi.coa(df = tableau.disjonctif, scannf=F, nf=7)

> inertie <- (resultats.AFC$eig / sum(resultats.AFC$eig)) * 100
> barplot(inertie, ylab = "% inertie", names.arg = round(inertie, 2))
> title("Eboulis des valeurs propres en %")
> # Explication de la variance
> round((resultats.AFC$eig / sum(resultats.AFC$eig)) * 100, 2)
> scatter.coa(resultats.AFC,method=1,sub="Helico",posieig="none")

> # une representation du plan factoriel: les assures
> plot(resultats.AFC$li[ ,1], resultats.AFC$li[ ,2], type = "n",
+        xlab="Axe 1", ylab = "Axe 2")
> text(resultats.AFC$li[ ,1], resultats.AFC$li[ ,2],
+       label = row.names(tableau.disjonctif))
> title("Helico - Plan des individus")
> abline(h=0, v=0)
```




```
> myData$AFCM1 <- resultats.AFC$li[ ,1] ;
> myData$AFCM2 <- resultats.AFC$li[ ,2] ;
...
> myData$AFCM6 <- resultats.AFC$li[ ,6]
> myData$AFCM7 <- resultats.AFC$li[ ,7]
> matrice.de.design <- model.matrix(~-1 + AFCM1+AFCM2+AFCM3+AFCM4+
+                                   AFCM5+AFCM6+AFCM7, myData)
> distance <- dist(matrice.de.design)
> ClassifHierarchAsc <- hclust(distance , method = "ward")
> plot(ClassifHierarchAsc, main= "Cluster Dendrogram",
+      xlab= "Observation Number in Data Set myData",
+      sub="Method=ward;Distance=euclidian")
```

Dendogramme

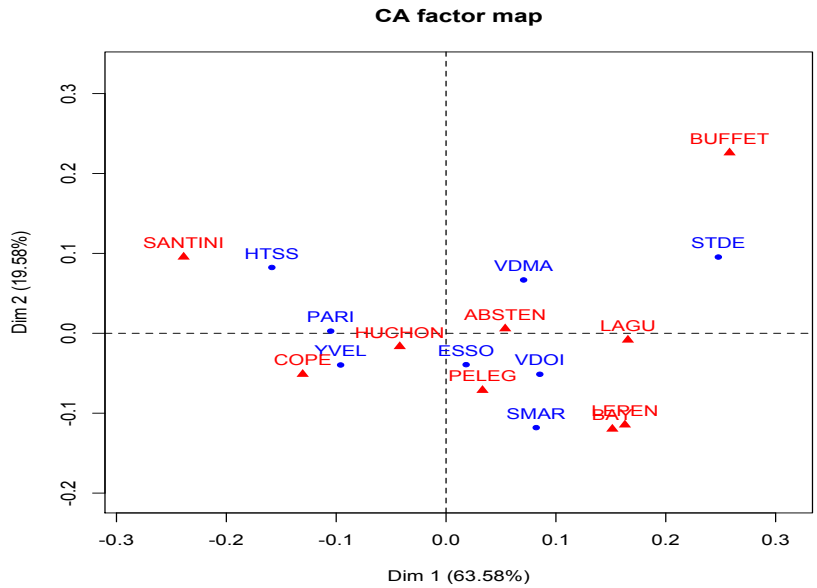


Récupération des clusters:

```
> new.clusters <- rect.hclust(ClassifHierarchAsc, k=8)  # 8 clusters
> # pour connaitre les individus de chaque cluster, on utilise
> print(new.clusters[[i]])  # ou i est le num. de cluster
```

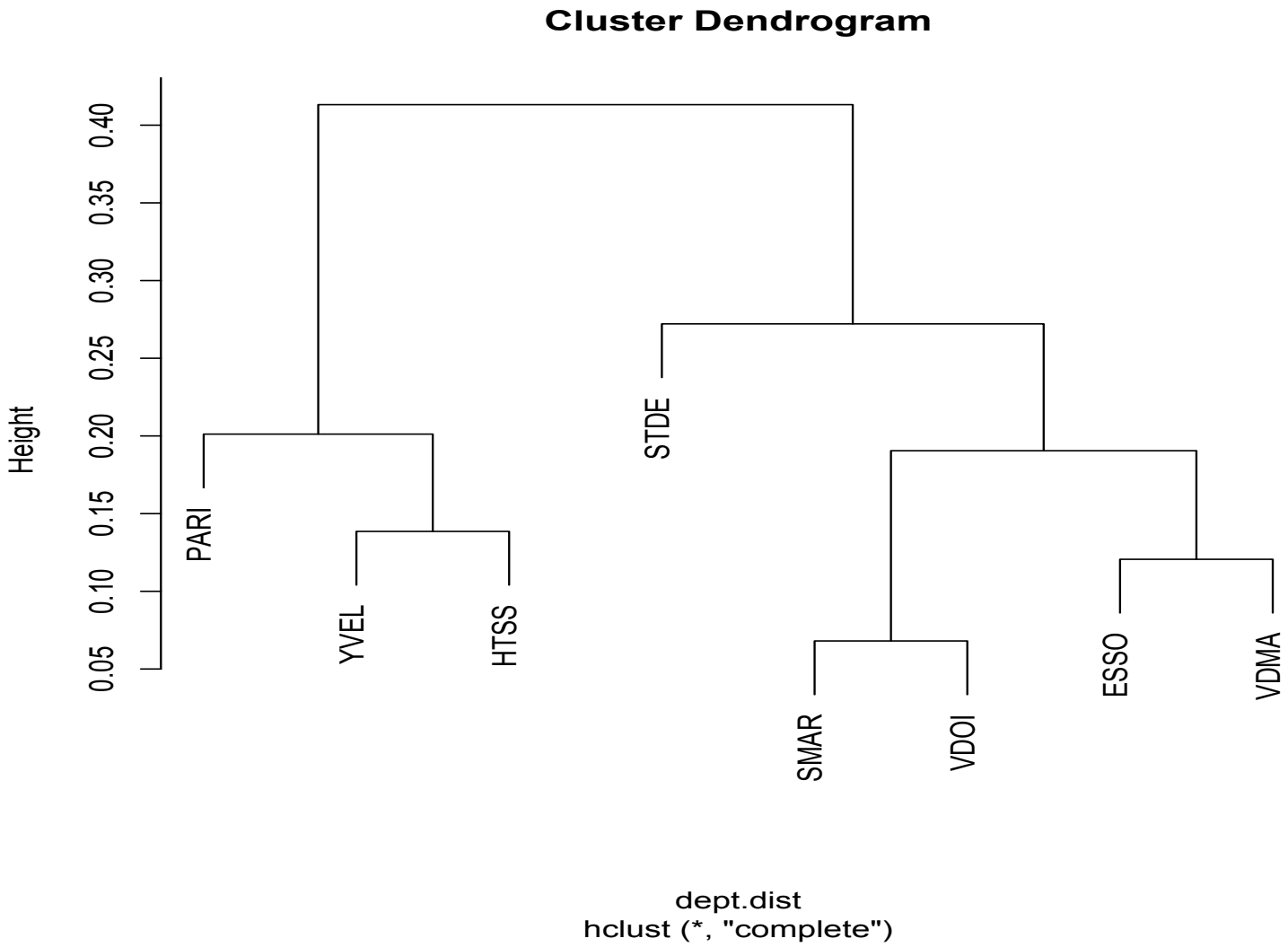
Exemple...F.G. Carpentier (élections IdF-2004)

	HUCHON	COPE	SANTINI	LEPEN	BUFFET	LAGU	PELEG	BAY	ABSTEN
PARI	258495	184419	114222	57183	39052	22479	13277	5006	434078
SMAR	128715	114003	48782	71897	25732	19738	11980	7085	301478
YVEL	150141	140634	96746	61676	23292	15998	13939	6486	329626
ESSO	144581	95451	59967	54309	26732	17545	12108	5346	270414
HTSS	143444	136677	122610	47279	32987	16438	11322	4690	314964
STDE	107327	61507	40081	54412	49535	19619	8393	5176	287618
VDMA	126569	93049	60234	47074	41897	17308	10969	4557	286913
VDOI	111176	82524	47903	55165	24693	17018	9876	4825	262458



```
> library(FactoMineR)
> regionales.CA <- CA(regionales.data, ncp=3)  # AFC a 3 comp.
> regionales.CA$row$coord
           Dim 1        Dim 2        Dim 3
PARI -0.10498220  0.002720771 -0.101621294
SMAR  0.08205669 -0.118055641  0.033233371
YVEL -0.09597746 -0.039670407  0.055520970
ESSO  0.01826232 -0.039252973 -0.035484742
HTSS -0.15856949  0.082371853  0.075166586
STDE  0.24784182  0.095411070  0.001689557
VDMA  0.07059721  0.066742451 -0.011532432
VDOI  0.08536010 -0.051274382  0.020568553
>
> dept.dist <- dist(regionales.CA$row$coord)
> dept.dist
           PARI       SMAR       YVEL       ESSO       HTSS
SMAR 0.26030034
YVEL 0.16300855 0.19579872
ESSO 0.14603099 0.12248168 0.14605800
HTSS 0.20117119 0.31595970 0.13855694 0.24146595
STDE 0.37914299 0.27211718 0.37330486 0.26874352 0.41320582
VDMA 0.20746784 0.19048788 0.20872703 0.12061378 0.24551655 (...)
VDOI 0.23254253 0.06805179 0.18503955 0.08825304 0.28345001
>
```

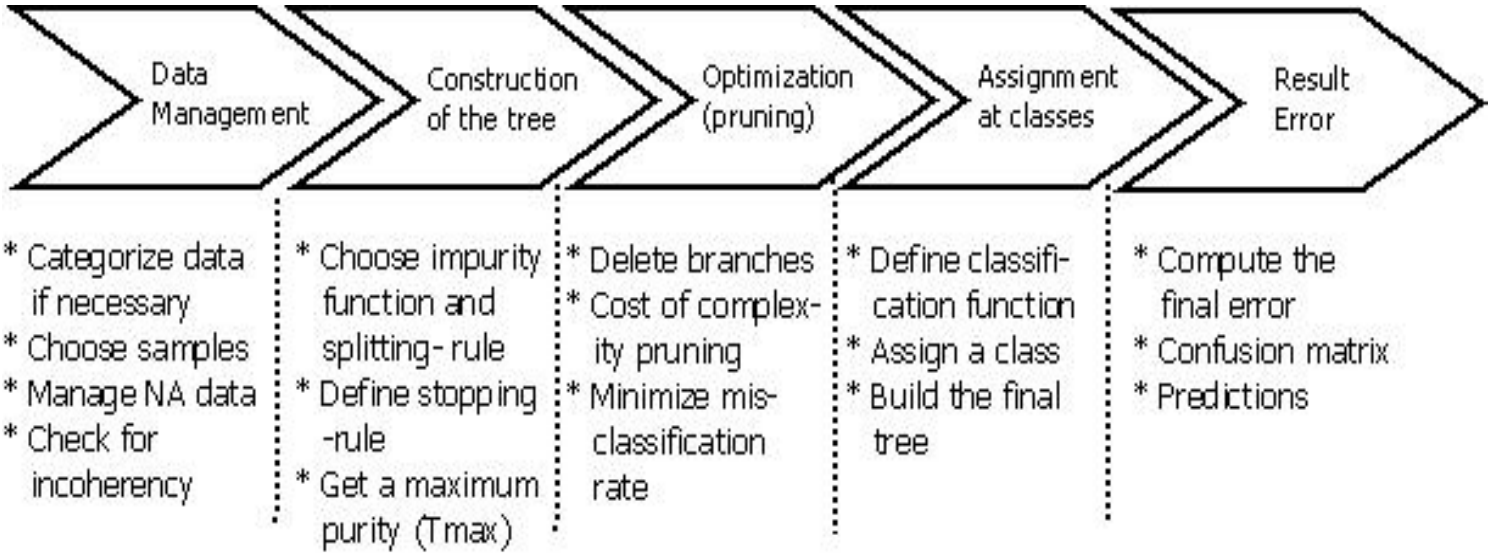
```
> dept.cah <- hclust(dept.dist)
>  plot(dept.cah)
```



La classification hiérarchique descendante

⇒ Méthode par division de l'échantillon total initial.

⇒ Réduction du nombre de variables explicatives.



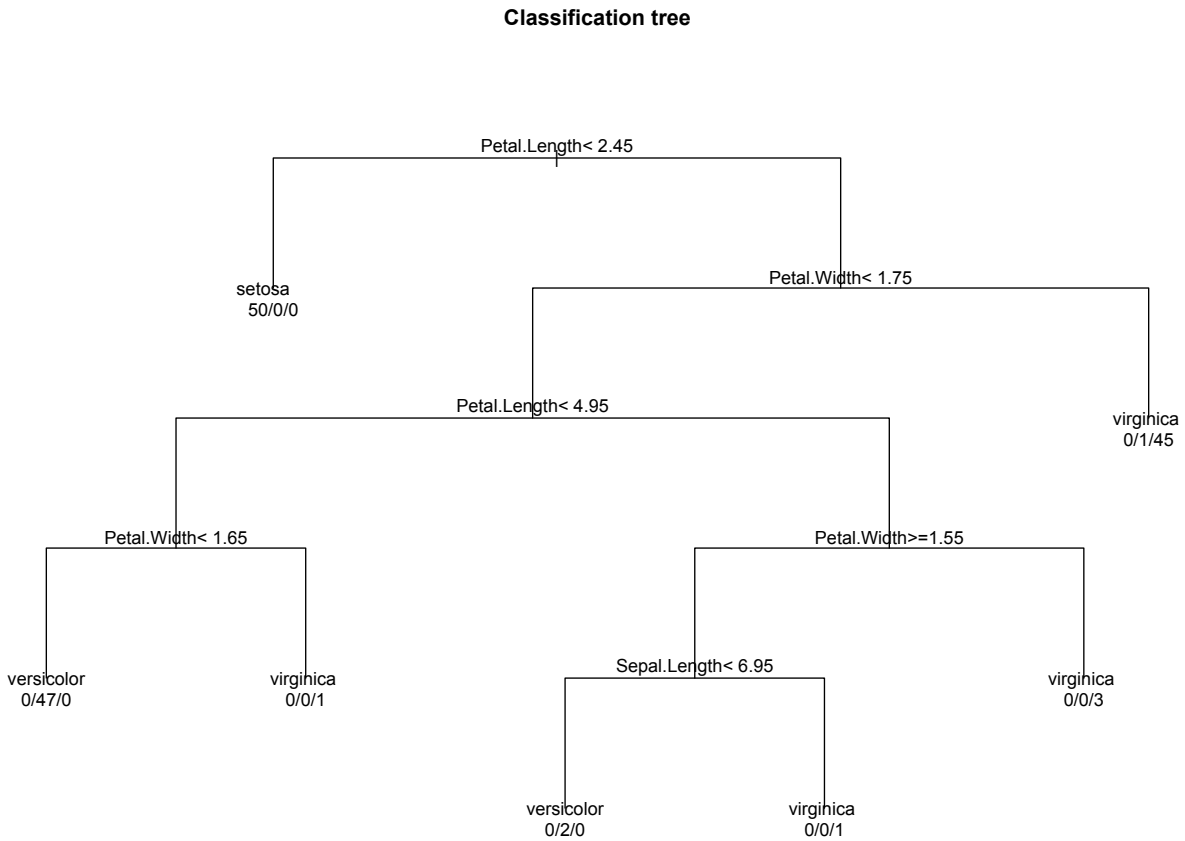
Minimisation de "variance", création de noeuds homogènes.

Remarque: l'analyse discriminante linéaire est une méthode alternative de sélection de variable (commande R: `lda()`).

Mise en oeuvre avec R: *library(rpart)*

```
>  crash.maximalTree <- rpart(NB_Accidents ~ Type_recode + Usage +
+ Annee + Territory_recode + Departement, method="class", data=myData,
+ control=rpart.control(minsplit=1,minbucket=1,cp=0, maxcompete=2))
>  plotcp(crash.maximalTree)
Erreur dans plotcp(crash.maximalTree) :
  'cptable' ne contients aucuns r\'esultats de validation crois\'ee
> crash.maximalTree$cptable
  CP nsplit rel error
1  0      0         1

>  ## Change le jeu de donnees
>  data(iris) ; head(iris, n = 2)
  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
>  iris.maximalTree <- rpart(Species ~ ., data=iris, method="class")
>  ## choose the cp corresponding to the smallest xerror
>  xerror.min <- min(iris.maximalTree$cptable[ ,4] +
+             iris.maximalTree$cptable[ ,5])
>  cp <- iris.maximalTree$cptable[which((iris.maximalTree$cptable[,4]+
+             iris.maximalTree$cptable[ ,5])==xerror.min), 1]
>  iris.cart <- prune(iris.maximalTree, cp = cp[1])
```

```
> plot(iris.cart, uniform=T, branch=1.0, compress=T, margin=0.1)
> text(iris.cart, use.n = T)
```



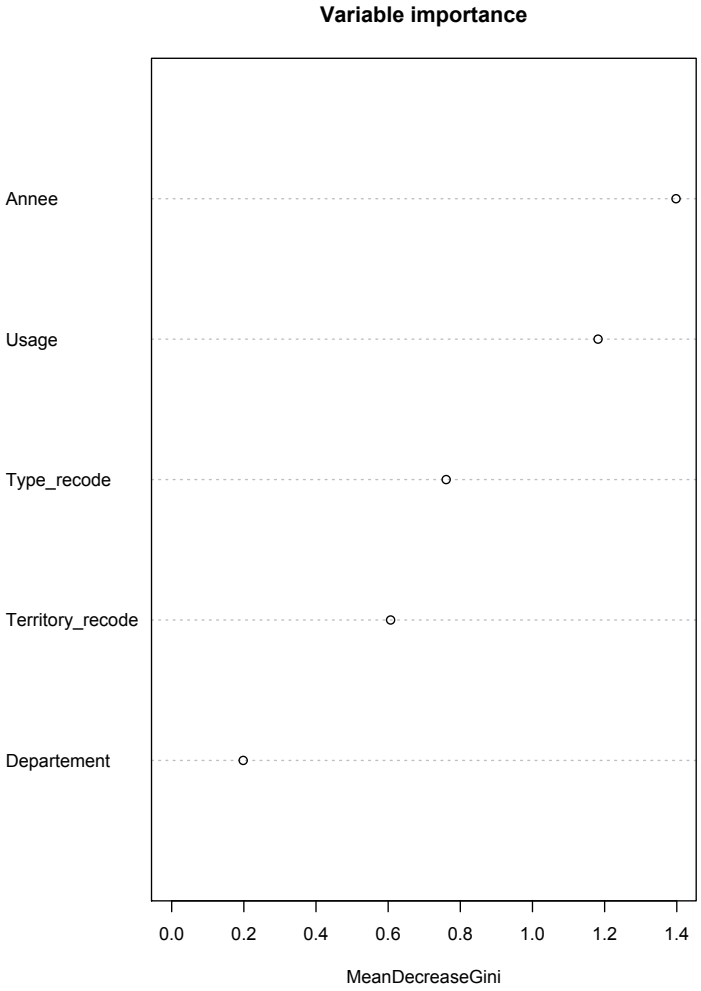
Exemple d'arbre final obtenu par l'algorithme.

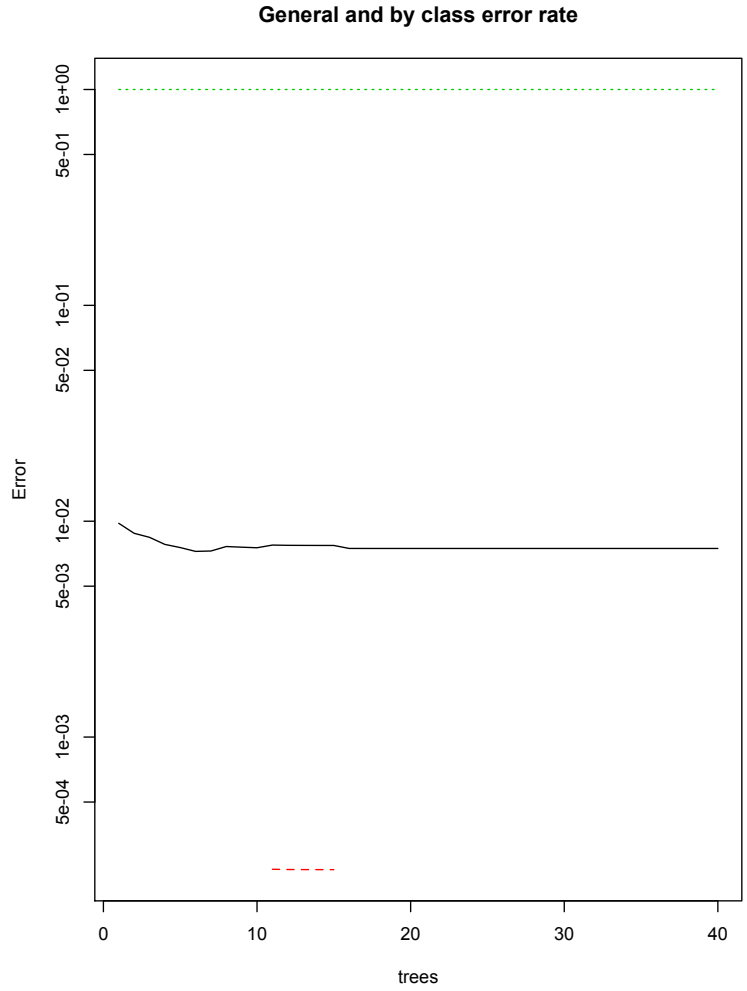
Importance des variables explicatives

La librairie *randomForest* permet d'appréhender l'importance des variables dans le processus de classification, par une méthode "*MonteCarlo*".

```
>  ## search for the best value of mtry : mtry <- tuneRF()
>  crash.RF <- randomForest(NB_Accidents ~ Type_recode + Usage +
+ Annee + Territory_recode + Departement, data = myData,
+ ntree=40, keep.forest=F, importance=T, mtry=2)

>  ## plots the importance of explanatory variables
>  varImpPlot(crash.RF, type = 2, main = "Variable importance")

>  ## to see the number of trees to stabilize the oob error
>  plot(crash.RF, log="y", main = "General and by class error rate")

>  confusion.matrix <- crash.RF$confusion[ ,1:2]
> confusion.matrix
     0 1
0 4115 0
1   31 0
```

Graphes des résultats





1 P-R-emier pas

2 Statistiques desc-R-iptives

3 Exemples d'usages d'actuaire
- Classification
- Un modèle économétrique classique: le cas des GLM
  - Notions de base
  - Implémentation
  - Calibrage
  - Sélection
- Modèles de crédibilité

4 Conclusion

Composantes d'un GLM

Un modèle GLM est caractérisé pour l'individu i par:

1. la loi de la réponse aléatoire Y_i: la réponse suit une distribution de la famille exponentielle;
2. le prédicteur $\eta_i = \sum_{k=1}^{p} \beta_k X_{ik}$, linéaire er déterministe: les facteurs explicatifs le constituent;
3. la fonction de lien g: monotone, différentiable et inversible et telle que $g(\mathbb{E}[Y_i]) = \eta_i$.

$\Rightarrow$ il faut donc les caractériser dans R!

Exemple du modèle linéaire:

$$g = Id \qquad \eta_i = \sum_{k=1}^{p} \beta_k X_{ik} \qquad Y_i \sim \mathcal{N}(\eta_i, \sigma^2)$$

.

Les fonctions de liens

Elles sont résumées dans le tableau suivant:

Loi	Lien	Moyenne	Utilisation
$\mathcal{N}(\mu, \sigma^2)$	Id: $\eta = \mu$	$\mu = X\beta$	Rég. lin.
$\mathcal{B}(\mu)$	logit: $\eta = ln(\frac{\mu}{1-\mu})$	$\mu = \frac{exp(X\beta)}{1+exp(X\beta)}$	Taux
$\mathcal{P}(\mu)$	log: $\eta = ln(\mu)$	$\mu = exp(X\beta)$	Fréquence
$\mathcal{G}(\alpha, \beta)$	inverse: $\eta = \frac{1}{\mu}$	$\mu = (X\beta)^{-1}$	Sévérité
$\mathcal{IN}(\mu, \lambda)$	inverse^2: $\eta = -\frac{1}{\mu^2}$	$\mu = (X\beta)^{-2}$	Sévérité

En pratique dans notre exemple, on prendra donc un lien *logit*!

Un premier essai sur nos données

Sur notre exemple, les résultats sont médiocres...

```
>  model.glm <- glm(NB_Accidents ~ Type_recode + Usage + Territory_recode + Departement,
+    data = myData, weights = myData$NB_HELICO_ANNEE, family = binomial(link = "logit"))
>  summary(model.glm)
  ...
                                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                                 -37.32633 3716.31177  -0.010   0.9920
Type Turbine - Bi-moteur                     -0.22222    1.12793  -0.197   0.8438
Type Turbine - Mono-moteur<2500               2.21555    1.01221   2.189   0.0286 *
Type Turbine - Mono-moteur>2500              -0.09627    1.50835  -0.064   0.9491
Usage Pilotage                                0.37742 3522.03384   0.000   0.9999
Usage Militaire                              -0.49936 3531.05702   0.000   0.9999
Usage Off-shore                               0.33164 3815.09842   0.000   0.9999
Usage Secours                                15.92829 3199.43901   0.005   0.9960
UsageTourisme-Affaires                       14.21893 3199.43904   0.004   0.9965
UsageTransport-PassMarchan.                  16.00699 3199.43897   0.005   0.9960
UsageTravailAerien                           16.12803 3199.43898   0.005   0.9960
UsageTravailAerien-saufElingue..             16.05347 3199.43898   0.005   0.9960
Territory Asia                                0.56848 2250.19052   0.000   0.9998
Territory Australasia                        -0.73963 3544.11463   0.000   0.9998
Territory Europe                             15.94826 1890.65134   0.008   0.9933
Territory LatinAmerica-Caribbean             16.38645 1890.65136   0.009   0.9931
Territory North America                      -0.34405 2274.93235   0.000   0.9999
DepartementAGI                               -0.42543    0.29241  -1.455   0.1457
---
Signif. codes:  0 Ô***Õ 0.001 Ô**Õ 0.01 Ô*Õ 0.05 Ô.Õ 0.1 Ô Õ 1
```

Amélioration possible

Trop de modalités ⇒ recodage!

- Type: mono-moteur et bi-moteur;
- Usage: formation, affaires ou travail;
- Territoire: Europe, Amerique latine ou autres.

⇒ Pertinence des méthode de réduction des dimensions pour le choix des variables / modalités à garder!

Exemple:

```
myData$Type_recode <- as.character(myData$Type_recode)
myData$Type_recode[grep("Mono-moteur", myData$Type_recode)] <- "Mono-moteur"
myData$Type_recode[grep("Bi-moteur", myData$Type_recode)] <- "Bi-moteur"
myData$Type_recode <- factor(myData$Type_recode)
table(myData$Type_recode)

myData$Territory_recode <- as.character(myData$Territory_recode)
myData$Territory_recode[which(myData$Territory_recode == "Europe")] <- "Europe"
myData$Territory_recode[which(myData$Territory_recode == "LatinAmerica-Caribbean")]
+          <- "LatinAmerica"
myData$Territory_recode[which(myData$Territory_recode != "LatinAmerica" &
+          myData$Territory_recode != "Europe")] <- "Others"
myData$Territory_recode <- factor(myData$Territory_recode)
table(myData$Territory_recode)
```

Nouveau résultat: meilleur mais mauvais!

A comparer avec les stats descriptives...
Explication: sous-représentation très forte des sinistres!

```
>  model.glm <- glm(NB_Accidents ~ Type_recode + Usage + Annee + Territory_recode +
+        Departement , data = myData, family = binomial(link="logit"))

>  summary(model.glm)

Call:
glm(formula = NB_Accidents ~ Type_recode + Usage + Annee + Territory_recode +
    Departement, family = binomial(link = "logit"), data = myData)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-0.2204  -0.1648  -0.1102  -0.0373   3.6635

Coefficients:
                              Estimate Std. Error z value Pr(>|z|)
(Intercept)                   -7.56991    1.20037  -6.306 2.86e-10 ***
Type_recodeMono-moteur         2.42129    1.01809   2.378   0.0174 *
UsageUsage-Formation         -16.54874 1476.10278  -0.011   0.9911
UsageUsage-Travail             0.26436    0.36854   0.717   0.4732
Annee2007                      0.04785    0.77049   0.062   0.9505
Annee2008                      0.74566    0.67840   1.099   0.2717
Annee2009                      0.86799    0.67824   1.280   0.2006
Annee2010                      0.60802    0.71568   0.850   0.3956
Territory_recodeLatinAmerica   0.32248    0.42712   0.755   0.4503
Territory_recodeOthers       -16.65279 1310.23506  -0.013   0.9899
DepartementAGI                -0.01185    0.48836  -0.024   0.9806
---
Signif. codes:  0 Ô***Õ 0.001 Ô**Õ 0.01 Ô*Õ 0.05 Ô.Õ 0.1 Ô Õ 1
```

Autre amélioration possible

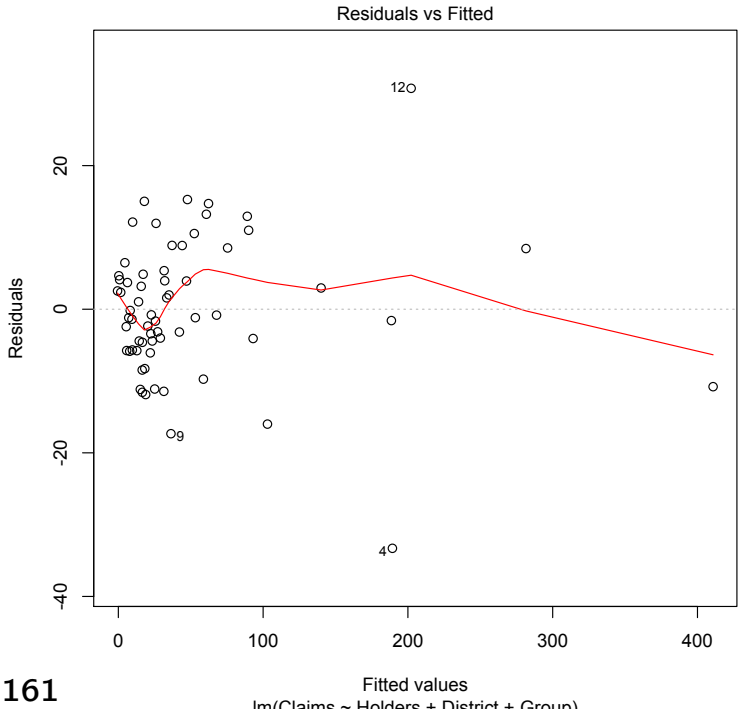
Se servir du processus de classification pour faire tourner les GLM sur des clusters homogènes...Mais ici les résultats sont encore pires!

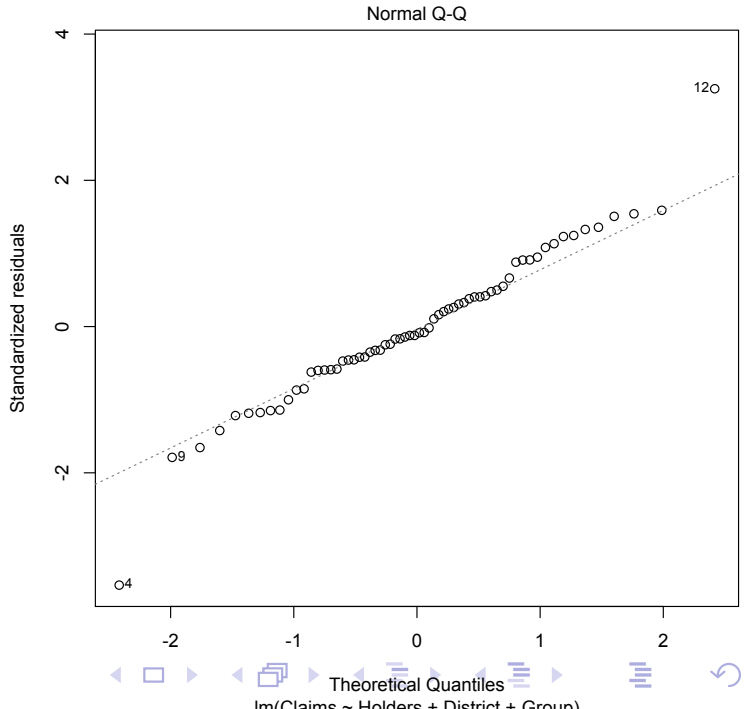
(autre possibilité: "importance resampling"...)

```
 ## GLM sur clusters obtenus par la classification : pire
> glm.cluster1 <- glm(NB_Accidents ~ Type_recode + Usage + Territory_recode +
 +   Departement , data = myData[new.clusters[[7]], ],
 + family = binomial(link = "logit"))
> summary(glm.cluster1)

Coefficients:
                                                Estimate Std. Error z value Pr(>|z|)
(Intercept)                                   -2.657e+01  3.669e+05       0        1
Type_recodeTurbine - Bi-moteur                -9.515e-30  7.377e+04       0        1
Type_recodeTurbine - Mono-moteur<2500         -1.152e-29  7.344e+04       0        1
Type_recodeTurbine - Mono-moteur>2500         -1.249e-29  8.835e+04       0        1
UsageTourisme-Affaires                         3.184e-13  3.666e+05       0        1
UsageTransport-PassagersMarchandises           3.184e-13  3.633e+05       0        1
UsageTravailAerien-ElingueAgricoleLutte incendie  3.184e-13  3.627e+05       0        1
UsageTravailAerien-saufElingueAgricoleLutteIncendie  3.184e-13  3.629e+05       0        1
Territory_recodeLatinAmerica-Caribbean        -1.057e-29  6.549e+04       0        1
DepartementAGI                                -2.539e-30  5.008e+04       0        1
```

Nouvelle illustration de modele GLM

```
## Change le jeu de donnees car decevant...
library(MASS)
data(Insurance)
head(Insurance, n = 5)
# suivre avec les commandes de Christophe
index <- sample(nrow(Insurance), 5)
regression.lineaire <- lm(Claims ~ Holders + District + Group,
+                         data = Insurance)
coefficients(regression.lineaire)
plot(regression.lineaire)
```





Comparaison des prévisions-modèle linéaire/GLM

Comparaison entre observations, prévisions par modèle linéaire et prévisions par modèle GLM:

```
>  glm.reg <- glm(Claims ~ Holders + District + Group +
+                          Age, data = Insurance, family = poisson)
```

```
>  head(cbind(obs = Insurance$Claims, reg.lin =
+       fitted(regression.lineaire), glm = fitted(glm.reg)), n = 5)
```

	obs	reg.lin	glm
1	38	24.24666	17.15981
2	35	34.37169	30.30146
3	20	31.25037	34.02503
4	156	190.50488	155.11423
5	63	48.14359	46.86296

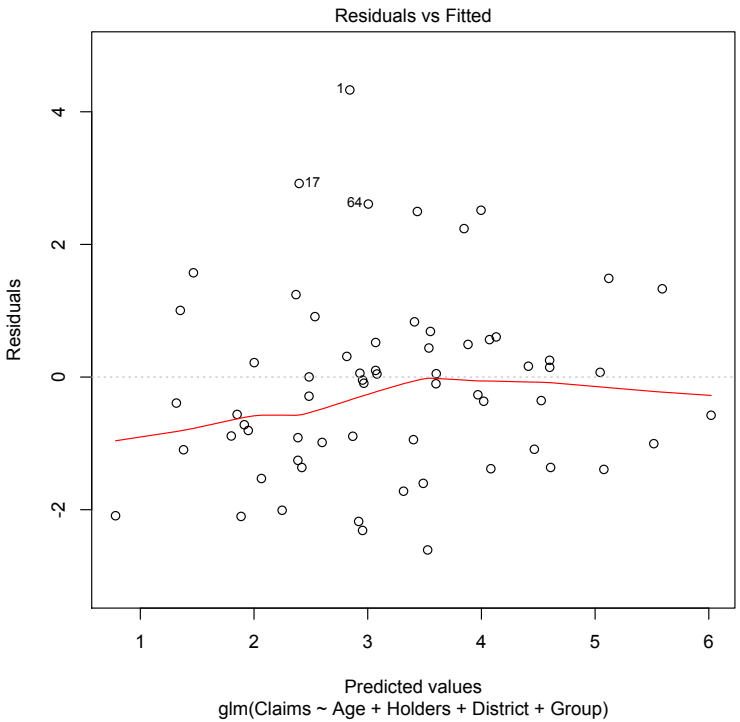
Qualité d'ajustement

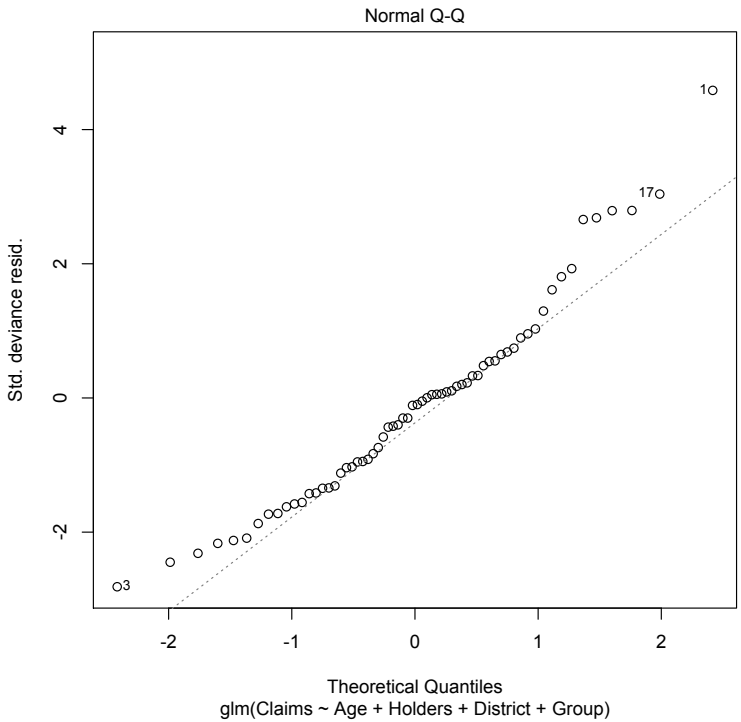
Le calibrage des coefficients de régression est donnée par la fonction *summary*().

```
>  summary(glm.reg)
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept)  3.935e+00  4.373e-02  89.987  < 2e-16 ***
Age.L        1.518e+00  6.342e-02  23.942  < 2e-16 ***
Age.Q        4.850e-01  5.422e-02   8.946  < 2e-16 ***
Age.C        4.200e-01  4.980e-02   8.433  < 2e-16 ***
Holders     -1.582e-05  3.592e-05  -0.440     0.66
District2   -4.474e-01  4.777e-02  -9.366  < 2e-16 ***
District3   -9.306e-01  6.129e-02 -15.184  < 2e-16 ***
District4   -1.465e+00  7.906e-02 -18.537  < 2e-16 ***
Group.L     -5.209e-01  5.388e-02  -9.666  < 2e-16 ***
Group.Q     -1.038e+00  5.134e-02 -20.215  < 2e-16 ***
Group.C      2.249e-01  3.835e-02   5.865 4.49e-09 ***
```

Résidus

```
> ## Residuals: i.i.d. et asymptotiquement gaussien
> plot(glm.reg, which = 1:2)
```





Déviance: mesure la qualité d'adéquation du modèle (en comparant la vraisemblance du modèle courant à celle du modèle saturé), suit le Khi-deux.

```
Null deviance: 4236.68  on 63  degrees of freedom
Residual deviance:  121.12  on 53  degrees of freedom
AIC: 460.44
```

Sélection de modèle

Sélection de modèle par une analyse de variance. On utilise la fonction *anova*() du package *car*, pour comparer:

- le modèle courant au modèle nul:

```
> Anova(glm.varCorrelees)
Analysis of Deviance Table (Type II tests)

Response: Claims
          LR Chisq Df Pr(>Chisq)
Age         815.27  3  < 2.2e-16 ***
Group       461.04  3  < 2.2e-16 ***
Holders       5.28  1    0.02156 *
District    362.77  3  < 2.2e-16 ***
Age:Group    52.23  9  4.087e-08 ***
```

- deux modèles GLM entre eux:

```
>  glm.classic <- glm(Claims ~ Age + Holders + District + Group, data = Insurance,
+                                    family = poisson)
>  glm.varCorrelees <- glm(Claims ~ Age * Group + Holders + District,
+                                 data = Insurance, family = poisson)
>  library(car)
>  anova(glm.classic, glm.varCorrelees, test = "Chisq")
Analysis of Deviance Table

Model 1: Claims ~ Age + Holders + District + Group
Model 2: Claims ~ Age * Group + Holders + District
  Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        53    121.118
2        44     68.887  9   52.231 4.087e-08 ***
```

Sélection de variable

Utilisation de tests d'hypothèses.

On utilise la fonction *stepAIC*() du package *MASS*:

- approche ascendante: modèle nul et ajout de variables,

```
> glm.classic.forward <- stepAIC(glm(Claims ~ Age * Holders *
+         District * Group, data = Insurance, family = poisson),
+         direction="forward")
```

- approche descendante: modèle saturé et suppression.

```
> glm.classic.backward <- stepAIC(glm(Claims ~ Age * Holders *
+         District * Group, data = Insurance, family = poisson),
+         direction="backward")
```

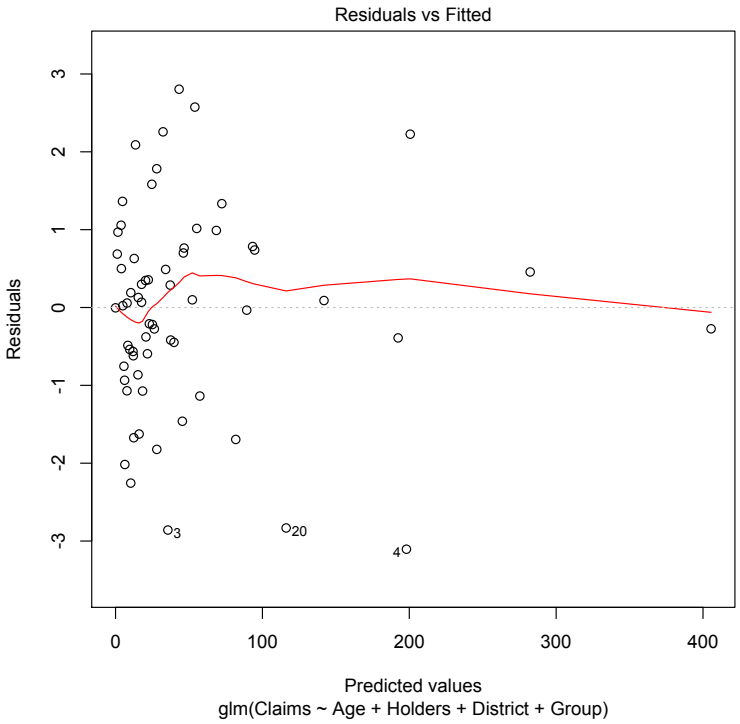
$\Rightarrow$ Ces tests sont basés sur...

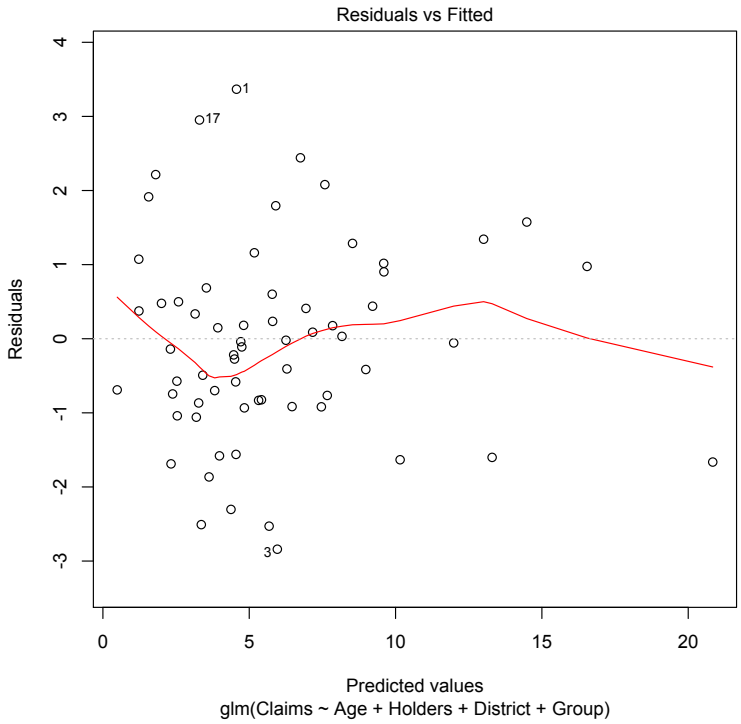
Choix de la fonction de lien

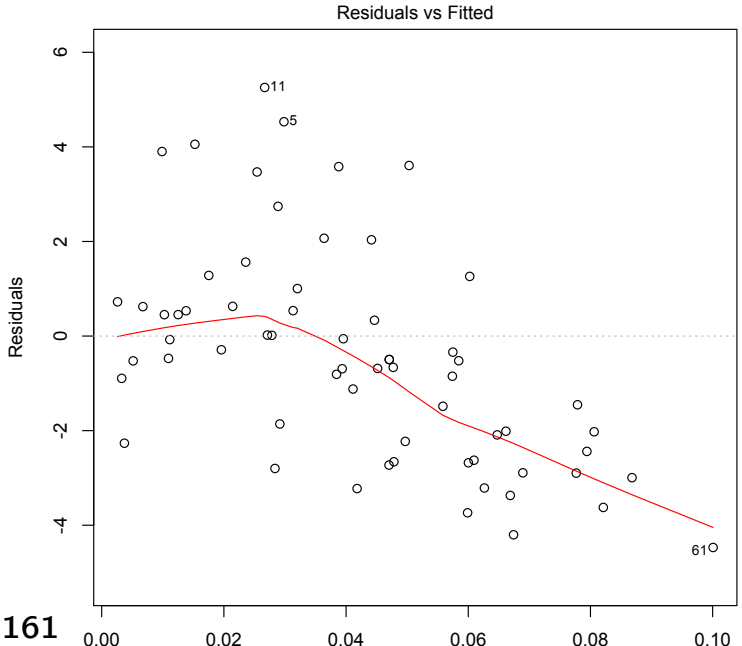
- La détection d'une tendance systématique des résidus indique probablement un mauvais choix de lien,
- Suivant la distribution de l'erreur, il y a un choix limité de fonctions de lien possibles.

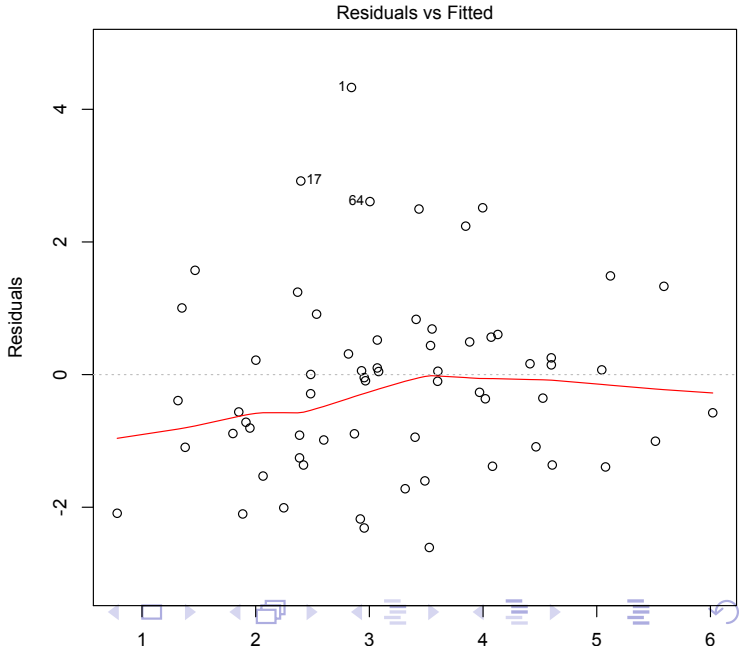
Ex: pour une erreur de loi de Poisson, nous pouvons considérer comme liens: *identite*, *sqrt*, *inverse* et *log*.

```
## Choix de la fonction de lien
glm.reg <- glm(Claims ~ Age + Holders + District + Group,
+           data = Insurance, family = poisson(link = "identity"))
plot(glm.reg, which = 1)
glm.reg <- glm(Claims ~ Age + Holders + District + Group,
+          data = Insurance, family = poisson(link = "sqrt"))
glm.reg <- glm(Claims ~ Age + Holders + District + Group,
+          data = Insurance, family = poisson(link = "inverse"))
glm.reg <- glm(Claims ~ Age + Holders + District + Group,
+           data = Insurance, family = poisson(link = "log"))
```









1 P-R-emier pas

2 Statistiques desc-R-iptives

3 Exemples d'usages d'actuaire
- Classification
- Un modèle économétrique classique: le cas des GLM
- Modèles de crédibilité
  - Tarif ind.
  - Implémentation

4 Conclusion

Idée de base

Les modèles de crédibilité permettent de tarifier plus justement les assurés en fonction de leur historique de sinistre(s).

Pondération expérience ind./expérience coll. :

$$P^{cred} = z\ P^{ind} + (1 - z)\ P^{coll}, \text{ (combinaison linéaire)}$$

avec z *facteur de crédibilité*, $z \in [0, 1]$.

- moins on possède d'expérience sur un contrat et plus z sera petit,
- plus l'expérience individuelle est importante et plus z sera grand.

Différents modèles de crédibilité

L'ensemble des modèles de crédibilité sont implémentés dans le package `actuar` de `R`. Il s'agit notamment des modèles:

- Buhlmann: + ancien, aucun poids aux données;
- Buhlmann-Straub: données pondérées, inadapté au données présentant une tendance;
- modèle hiérarchique: intègre une structure hiérarchique de prime;
- modèle de régression (Hachemeister): permet d'avoir une tendance due par exemple a un facteur exogène.

L'actuaire calcule les paramètres de structure: prime collective, variance intra et inter-contrats (groupes de risque).

http://cran.r-project.org/web/packages/actuar/index.html

Mise en oeuvre : librairie *actuar*

$\rightarrow$ Il s'agit de mettre en oeuvre la théorie de la crédibilité sur les contrats de notre portefeuille.

$\rightarrow$ On pourrait établir un lien entre un modèle de crédibilité appliqué à une classe et un GLM et retrouver sensiblement les mêmes résultats, mais...

- la taille de la classe doit être importante;
- nous ne le ferons pas puisque les applications GLM sont mauvaises sur notre exemple...

```
> ## Modeles de credibilite
> library(actuar)

> myData.charge <- read.csv2(file = "Application-cred-charges.csv")
> myData.poids <- read.csv2(file = "Application-cred-poids.csv")
> myData <- data.frame(myData.charge, myData.poids)
> myData <- myData[ ,-7] ; myData[ ,1] <- as.numeric(myData[ ,1])
> colnames(myData) <- c("Assure","charge.1","charge.2","charge.3",
  +       "charge.4","charge.5","poids.1","poids.2","poids.3",
  +       "poids.4", "poids.5")
```

Création de classes de risque

```
> ## recupere et aggrege les assures en groupe en fonction des
+       clusters trouves a la CAH
> group.1<- myData[new.clusters[[1]],]; risk.1<- colSums(group.1)[-1]
> group.2<- myData[new.clusters[[2]],]; risk.2<- colSums(group.2)[-1]
> group.3<- myData[new.clusters[[3]],]; risk.3<- colSums(group.3)[-1]
> group.4<- myData[new.clusters[[4]],]; risk.4<- colSums(group.4)[-1]
> group.5<- myData[new.clusters[[5]],]; risk.5<- colSums(group.5)[-1]
> group.6<- myData[new.clusters[[6]],]; risk.6<- colSums(group.6)[-1]
> group.7<- myData[new.clusters[[7]],]; risk.7<- colSums(group.7)[-1]
> group.8<- myData[new.clusters[[8]],]; risk.8<- colSums(group.8)[-1]
> donnees.cred <- rbind(c(risk.1, risk.2, risk.3, risk.4, risk.5,
+                                 risk.6, risk.7, risk.8))
> donnees.cred <- cbind(1:8, donnees.cred)
> colnames(donnees.cred)[1] <- "risk.group"

> ## Buhlmann-Straub model
> cred.model <- cm(~ risk.group, data = donnees.cred,
+             ratios = charge.1:charge.5, weights = poids.1:poids.5)

> predict(cred.model)
> summary(cred.model)
```

Sorties R

```
>  cred.model <- cm(~ risk.group, donnees.cred,
+         ratios = charge.1:charge.5, weights = poids.1:poids.5)
>  predict(cred.model)
    risk.1     risk.2     risk.3     risk.4     risk.5     risk.6
  60614.93 5147467.34  121565.92   43378.46   59805.02  140769.44
    risk.7     risk.8
 158819.45   59515.84

>  summary(cred.model)
        (...)
Structure Parameters Estimators
  Collective premium: 723992
  Between risk.group variance: 6.392769e+12
  Within risk.group variance: 1.738949e+15
Detailed premiums (Level: risk.group)
    risk.group Indiv. mean Weight Cred. factor Cred. premium
    1                   0   2977    0.9162768      60614.93
    2             5325518   6758    0.9613062    5147467.34
    3                   0   1348    0.8320894     121565.92
    4                   0   4268    0.9400843      43378.46
    5                   0   3021    0.9173955      59805.02
    6                   0   1127    0.8055649     140769.44
    7                   0    968    0.7806337     158819.45
    8                   0   3037    0.9177949      59515.84
```

Bilan

R est plus qu'un logiciel, c'est un langage! Attention: c'est un langage interprété, mais pas compilé...

Finalement, toutes les étapes d'une étude peuvent se faire en R, pourvu que l'on sache comment les implémenter:

1. importer des fichiers;
2. travailler dessus: statistiques descriptives, modélisation, calcul d'erreurs...;
3. combiner et imbriquer des fonctions pour finalement arriver à programmer des calculs complexes;
4. éventuellement développer une interface pour un outil;
5. exporter les résultats dans des fichiers.

Synthèse très pratique des fonctions R les plus souvent utilisées: voir les RefCards (short).

Exercices Bonus

1. Réaliser une analyse en composantes principales...
2. Réaliser une analyse factorielle des correspondances mutliples...
3. Construire un arbre de classification hiérarchique descendante (algorithme CART).
4. Expliquer le taux de sinistralité à l'aide d'un modèle linéaire généralisé (lien logit).
5. Expliquer le nombre de sinistres à l'aide d'un modèle linéaire généralisé (lien log).