Introduction à R

Programmation et Logiciel

Cours 1

Sources

- *R fundamentals and Programming Techniques*

  Thomas Lumley, 2006

  R Core Development Team

  Univ. Of Washington Dept. Of Biostatistics
- *The Art of R Programming*

  Norman Matloff

  No Starch Press 2011

Que sont R et S+ ?

- R est une implémentation libre d'un dialecte du langage S, qui est un environnement statistique et graphique pour l'analyse de données
- S a été sciemment conçu pour effacer la distinction entre utilisateurs et programmeurs
- S-PLUS est un système commercial (Insightful Co)

Que sont R et S+ ?

- S est un système pour l'analyse interactive des données. Il a été conçu avec le souci d'interactivité.
- S est un langage de programmation de haut niveau, ayant certains points communs avec Scheme et Python.
- C'est un bon système pour le développement rapide d'applications statistiques.

Les inconvénients de R

- R (et S) sont accusés d’être lents, gourmands en mémoire et de ne permettre de manipuler que des tableaux de taille relativement faible.

- *C’est tout à fait exact.*

Les inconvénients de R

- Cependant et heureusement, les ordinateurs actuels sont rapides et dotés de mémoires importantes.
- Un tableau de données de quelques dizaines de milliers d’observations peut être abordé avec 256 Mo de mémoire et 1 Go permet déjà d’envisager des tableaux de données de taille respectable.
- De plus, il existe des outils permettant d’interfacer R avec des bases de données mais cela n’est pas transparent pour l’utilisateur.

Les inconvénients de R

- R ne possède pas de GUI ni de support commercial, contrairement à S+.
- S+ propose également un ensemble de boîtes à outils spécialisées qui peuvent justifier son acquisition.

GUI (Interface graphique)

  - R ne dispose pas de GUI pour les analyses statistiques mais en propose pour la manipulation de fichiers, *scripts*, fenêtres, …etc
- Inclus : Interfaces Windows et Mac
- Interfaces multiplateformes : Eclipse, JGR et Emacs/ESS peuvent être téléchargés sur Internet

Analogies entre R et S+

- Pour l'analyse de données en ligne de commande, ils sont très semblables
- La plupart des programmes écrits dans un dialecte peuvent être traduits de façon immédiate dans l'autre
- De longs programmes requièrent presque toujours une traduction
- R dispose d'un système de *packages* très efficace pour distribuer code et données, ce qui n'est pas le cas de S+

Lecture de données

- Fichiers textes
- Tableaux SAS, Stata, SPSS
- Pages web
- (Bases de données)

- Une information détaillée est fournie dans le manuel Data Import/Export.

Lecture de données texte

- Le format le plus simple se présente avec les noms des variables dans la première ligne

Case	id	gender	deg	yrdeg	field	startyr	year
1	1	F	Other	92	Other	95	95
2	2	M	Other	91	Other	94	94
3	2	M	Other	91	Other	94	95
4	4	M	PhD	96	Other	95	95

et des champs de valeurs séparés par des espaces ou des tabulations.

Lecture de données texte

- Sous R, utiliser la commande

```
salaire <- read.table("salary.txt", header=TRUE)
```

Pour lire les données du fichier salary.txt dans le tableau de données (data frame) salaire.

Syntaxe

- On peut laisser sans problème des espaces entre les commandes
- Par contre, faire attention à la distinction entre majuscules et minuscules
- TRUE (et FALSE) sont des constantes logiques
- Contrairement à d'autres systèmes, R ne fait pas la distinction entre les commandes et les fonctions : tout commande est une fonction, donc renvoie une valeur

Syntaxe

- Les arguments des fonctions peuvent être identifiés (`header=TRUE`) ou non (`« salary.txt »`)

- Tout tableau de données (data frame) est stocké dans une variable. *On peut donc manipuler plusieurs tableaux de données en même temps*.

Lecture de données texte

- Parfois, les colonnes sont séparés par des virgules ou des tabulations

```
Ozone,Solar.R,Wind,Temp,Month,Day
41,190,7.4,67,5,1
36,118,8,72,5,2
12,149,12.6,74,5,3
18,313,11.5,62,5,4
NA,NA,14.3,56,5,5
```

- Dans ce cas, utiliser

```
ozone <- read.table("ozone.csv", header=TRUE, sep=",")
```

ou

```
ozone <- read.csv("ozone.csv")
```

Syntaxe

- Les fonctions peuvent avoir des arguments optionnels (sep).

- Pour avoir une description complète d'une fonction et des arguments possibles, on peut utiliser help (par exemple `help(read.table)`)

Syntaxe

- NA est utilisé pour coder les valeurs manquantes
- R manipule NA de façon intuitive: 1+NA, NA & FALSE,…etc renvoient NA
- Pour tester si une valeur est manquante, on utilise la fonction `is.na()`

Lecture de données texte

- Parfois les noms des variables ne sont pas inclus

```
1 0.2 115 90 1 3 68 42 yes
2 0.7 193 90 3 1 61 48 yes
3 0.2 58  90 1 3 63 40 yes
4 0.2 5   80 2 3 65 75 yes
5 0.2 8.5 90 1 2 64 30 yes
```

- On peut alors les spécifier directement

```
psa <- read.table("psa.txt",
          col.names=c("ptid","nadirpsa",
          "pretxpsa", "ps","bss","grade","age",
          "obstime","inrem"))
```

ou en deux temps

```
psa <- read.table("psa.txt")
names(psa) <- c("ptid","nadirpsa","pretxpsa", "ps",
        "bss","grade","age","obstime","inrem"))
```

Syntaxe

- Pour assigner un vecteur (ou n'importe quel objet) à une variable, on utilise la même syntaxe que pour lui assigner un tableau de données

- `c()` est une fonction qui transforme ses arguments en un vecteur

- `names` est une fonction qui permet d'accéder aux noms des variables d'un tableau de données

Données issues d'autres logiciels statistiques

```
library(foreign)
stata <- read.dta("salary.dta")
spss <- read.spss("salary.sav", to.data.frame=TRUE)
sasxport <- read.xport("salary.xpt")
epiinfo <- read.epiinfo("salary.rec")
```

- De nombreuses fonctions R sont disponibles dans des packages
- La fonction `library()` permet de lister les packages disponibles, d'obtenir de l'aide et de charger les packages en mémoire

Données issues d'autres logiciels statistiques

- Le package foreign est inclus dans la distribution standard de R. Il est consacré à l'importation/exportation de données.

- Des centaines d'autres packages sont disponibles sur

http://cran.r-project.org

Données sur le Web

- Les fichiers chargés à l'aide de read.table peuvent se trouver sur le web

```
F <- read.table("http://univ-ubs.fr/lardjane/
       donnees/coursR.dat", header=TRUE)
```

- Il est également possible de lire des fichiers dans des bases de données en lignes

Manipulation de données

- Comme R peut manipuler plusieurs tableaux de données à la fois, il est nécessaire de spécifier dans quel tableau se trouve une variable
- La syntaxe

```
antibiotics$duree
```

signifie que la variable duree se trouve dans le tableau antibiotics.

Manipulation de données

```
## Ceci est un commentaire
## pour convertir une température de Fo en Do

antibiotics$tempC <- (antibiotics$temp-32)*5/9

## pour afficher la moyennes, les quartiles de
## toutes les variables

summary(antibiotics)
```

Manipulation de données

- Tout objet R est en fait un vecteur (mais certains n'ont qu'un élément)
- Utiliser [ ] pour extraire des éléments d'un vecteur (vu en TD)

```
## Premier élément
antibiotics$temp[1]

## Tous les éléments sauf le premier
antibiotics$temp[-1]

## Eléments 5 à 10
antibiotics$temp[5:10]
```

Manipulation de données

```
## Eléments 5 et 7
antibiotics$temp[c(5,7)]

## Personnes ayant reçus des antibiotiques (noter ==)
antibiotics$temp[ antibiotics$antib==1 ]

## ou
with(antibiotics, temp[antib==1])
```

Syntaxe

- Les indices positifs sélectionnent des éléments, les indices négatifs éliminent des éléments (vu en TD)
- 5:10 est la suite 5, 6, 7, … 10 (vu en TD)
- Pour tester l'égalité, utiliser == , le signe = ne convient pas
- with() définit temporairement un tableau de données comme environnement par défaut où chercher les variables
- Une autre possibilité est d'utiliser attach()

Manipulation de données

- Pour les tableaux de données, on utilise deux indices

```
## Première ligne
antibiotics[1,]
## Deuxième colonne
antibiotics[,2]
## Plusieurs lignes et colonnes
antibiotics[3:7, 2:4]
```

Manipulation de données

```
## Colonnes par noms
antibiotics[, c("id","temp","wbc")]
## Personnes ayant reçu des antibiotiques
antibiotics[antibiotics$antib==1, ]
## Créer un nouveau tableau de données
yes <- antibiotics[antibiotics$antib==1,]
```

Calculs

```
mean(antibiotics$temp)
median(antibiotics$temp)
var(antibiotics$temp)
sd(antibiotics$temp)
mean(yes$temp)
mean(antibiotics$temp [antibiotics$antib==1]
with(antibiotics, mean(temp[sex==2]))
extr <- with(antibiotics, temp>99)
mean(extr)
```

Facteurs

- Les facteurs représentent des variables catégorielles. On ne peut leur appliquer les opérateur mathématiques (excepté ==, !=)

```
table (salary$rank, salary$field)
antibiotics$antib <- factor(antibiotics$antib,
        labels=c("Yes","No"))
antibiotics$agegp<-cut(antibiotics$antib,
            c(0,18,65,100))
table(antibiotics$agegp)
```

Chaînes de caractères

- Les fonctions de manipulation de chaînes de caractères sont basées sur les expressions régulières
- Celles-ci s'interprètent comme de petits programmes représentant des chaînes de caractères
- "a" représente la lettre a
- "TATAA" représente la chaîne TATAA
- "ACGT.C" correspond à ACGT suivie d'un caractère quelconque, suivi de C
- "a*b*" correspond à aucun ou plusieurs a suivi d'aucun ou plusieurs b

Chaînes de caractères

- "[A-Za-z] +" correspond à un ou plusieurs caractères entre A et Z ou entre a et z
- "[[:alpha:]]+" correspond à une ou plusieurs lettres dans l'alphabet local
- "\\.csv$" correspond à une chaîne de caractères avec .csv à la fin
- "([[:alpha:]]+) [[:space:] [:punct:]]+\\1[[:space:] [:punct:]]+"

Correspond à tout mot répété.

Fonctions d'expressions régulières

- grep(regexp, vecteur) identifie toutes les chaînes de caractères dans le vecteur qui contiennent une sous-chaînes correspondant à l'expression régulière
- sub(regexp, nouvelle, vecteur) remplace la première sous-chaîne correspondant à l'expression régulière avec la nouvelle (pour chaque élément du vecteur).
- gsub fait la même chose, mais peut faire plus d'un remplacement par chaîne

Fonctions pour expressions régulières

- regexpr(regexp, vecteur) renvoie la position de la première correspondance pour chaque chaîne de caractères
- gregexpr fait la même chose mais renvoie toutes les correspondances
- strsplit() scinde une chaîne de caractère à chaque correspondance avec une expression régulière
- glob2rx() convertit des noms de fichiers en expressions régulières
- On peut également manipuler des expressions régulières Perl mais avec une syntaxe différente

Aide (vu en TD)

- help(fn) pour de l'aide sur fn
- help.search("sujet") pour les pages d'aide en lien avec "sujet"
- apropos("tab") pour les fonctions dont le nom contient "tab"
- Davantage de fonctions sur le site

  http://www.r-project.org

Graphiques

- R (et S+) peuvent produire des graphiques dans plusieurs formats, parmi lesquels :
- Graphiques sur écran
- Fichiers PDF pour inclure dans des documents LATEX ou envoyer par e-mail
- Fichiers PNG ou JPEG pour les pages Web (ou pour inclure dans des documents MS Office).

Graphiques

- Le format PNG est également utile pour les graphes de données nombreuses.
- Sous Windows, metafiles pour Word, Powerpoint, et les programmes analogues

Mise au point

- Les graphiques sont généralement mis au point à l'écran puis sauvegardés dans un fichier PDF par exemple
- Pour les graphiques à imprimer, on obtient un meilleur résultat en mettant le graphique au point aux dimensions dans lesquelles il sera imprimé, par exemple :

```
## sous Windows
windows (height=400, width=600)
```

Mise au point

- Word ou LATEX peuvent redimensionner le graphique mais dans ce cas les légendes sont aussi redimensionnées...

Finition

- Quand on a les bonnes commandes pour réaliser un graphique, on peut le sauvegarder dans n'importe quel format; par exemple :

```
## ouvrir un fichier pdf
pdf ("graphique.pdf" ,height=400,width=600)
## instructions à suivre
...
### fermer le fichier PDF
dev.off()
```

Plot

- On utilise généralement plot() pour créer un graphique puis lines(), points(), legend(), text(), et d'autres commandes pour le compléter
- plot() est une fonction générique : elle détermine la représentation appropriée pour différents types d'arguments

Plot

```
## nuage de points
plot (salary$year, salary$salary)
## boxplot
plot (salary$rank, salary$salary)
## diagrammes en barres empilés
plot (salary$field, salary$rank)
```

Syntaxe

- La commande plot() admet la syntaxe

```
plot(salary˜rank, data=salary)
```

où l'on utilise le système de formule des modèles de régression.

- Les variables de la formule sont recherchées dans l'argument data.

Mise au point d'un graphique

- Deux aspects importants lors de la mise au point d'un graphique
- Il doit transmettre une information
- Il doit être lisible

Mise au point d’un graphique

- Les axes doivent être labellisés (et les unités spécifiées)
- La couleur peut être utile (mais pas si le graphique doit être imprimé en noir et blanc)
- Des styles de points ou de lignes différents doivent généralement être labellisés
- Na pas superposer des points à d’autres points

Options

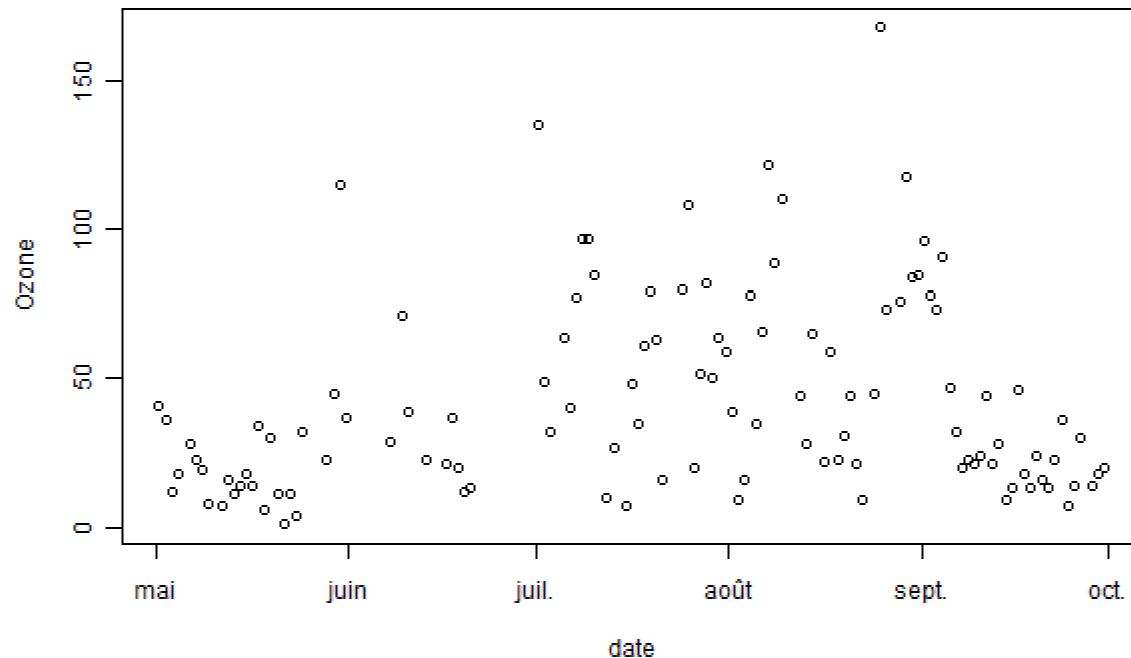
- Charger un tableau de données : concentration journalière d'ozone dans une ville, été 1973

```
data(airquality)
names (airquality)
airquality$date<-with(airquality,
        ISOdate(1973,Month,Day))
```

- Tous les graphes ci-après ont été mis au point en 400×600 pixels et sauvegardés comme fichiers png

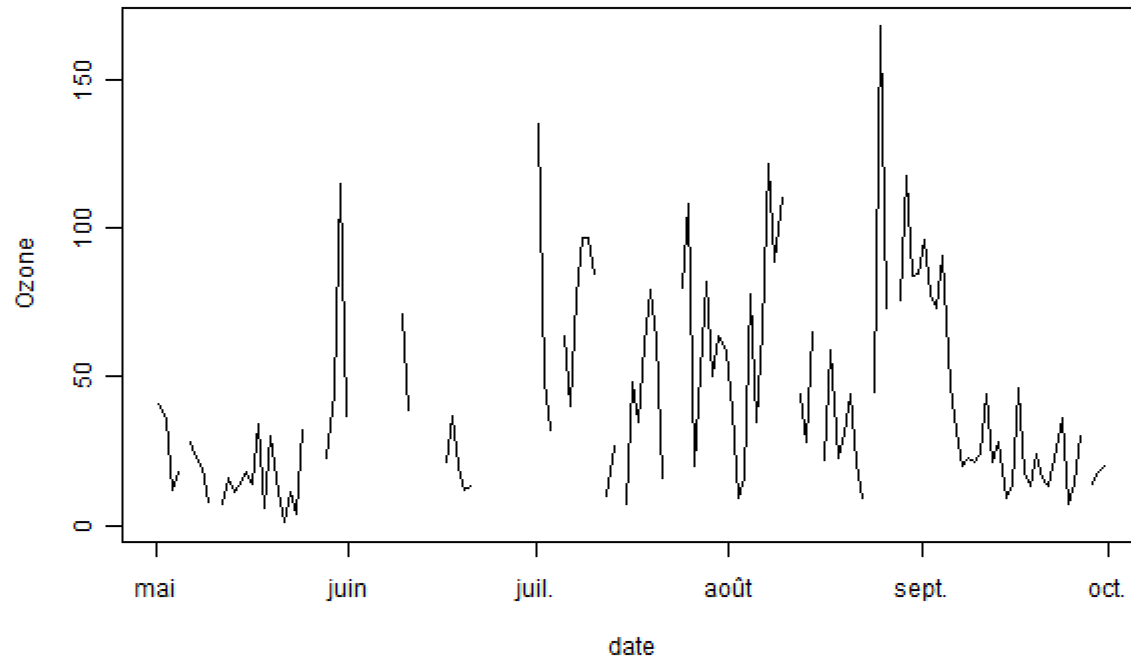
Graphiques

- plot(Ozone~date, data=airquality)



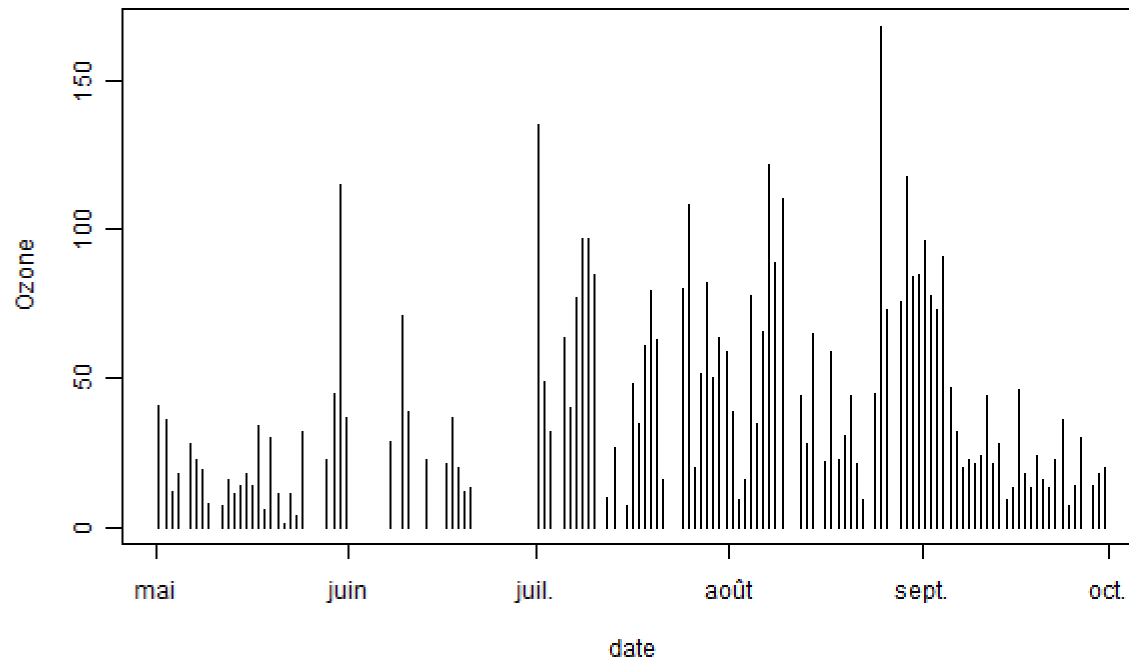
Graphiques

- plot(Ozone~date, data=airquality,type="l")



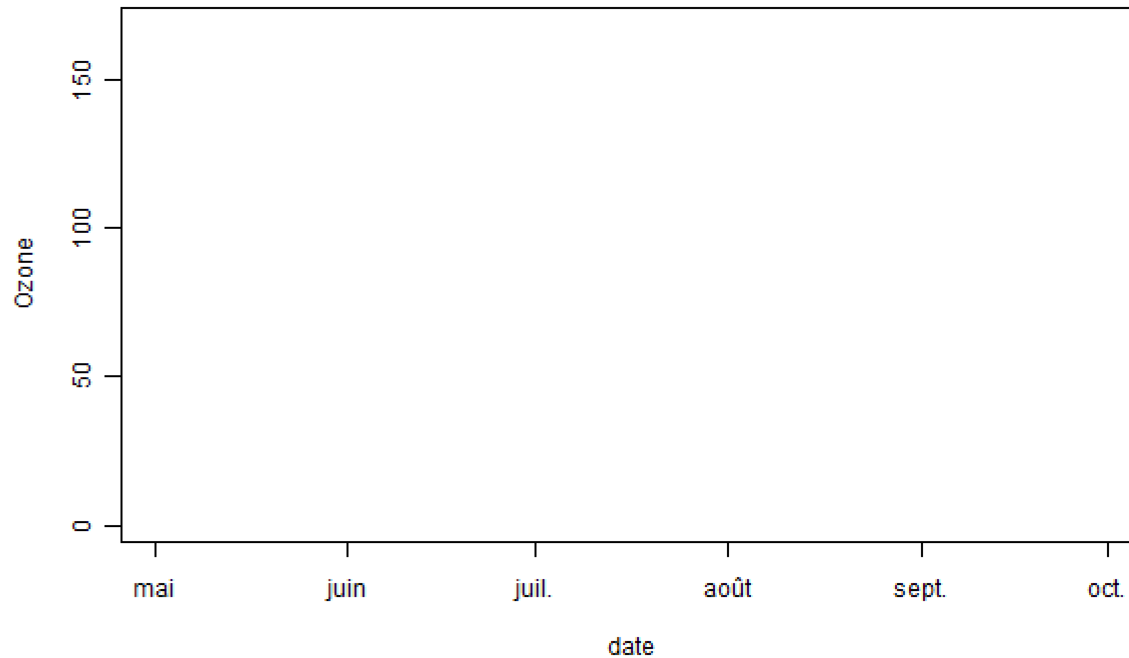
Graphiques

- plot(Ozone~date, data=airquality,type="h")



Graphiques

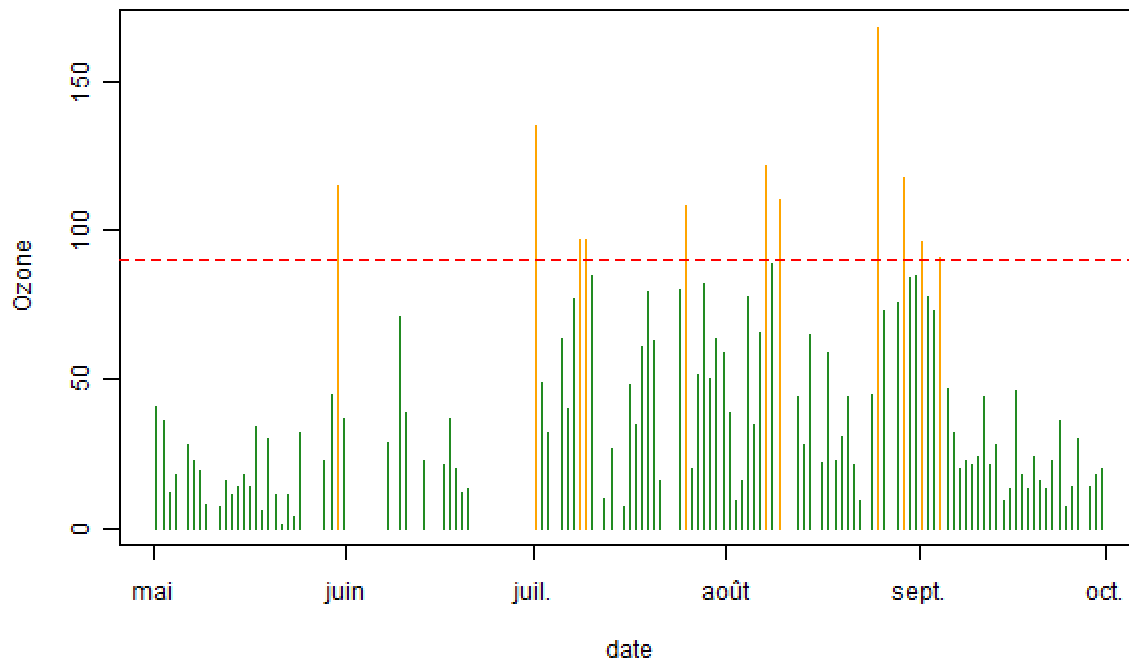
- plot(Ozone~date, data=airquality,type="n")



Graphiques

```
mauv <-ifelse(airquality$Ozone>=90,
        "orange","forestgreen")
plot(Ozone~date,
        data=airquality,type="h",col=mauv)
abline(h=90,lty=2,col="red")
```

Graphiques



Syntaxe

- type = spécifie le type de graphique.
- type="n" est parfois utile; cela permet de créer un graphique à compléter ensuite.
- Les couleurs peuvent être spécifiées par leurs noms (la fonction colors() fournit tous les noms possibles), par leurs valeurs rouge/vert/bleu (#rrggbb 6 chiffres en base 16) ou par leur position dans une palette standard de 8 couleurs.
- Pour pdf() et quartz(), on peut spécifier des couleurs partiellement transparentes sous la forme #rrggbbaa.

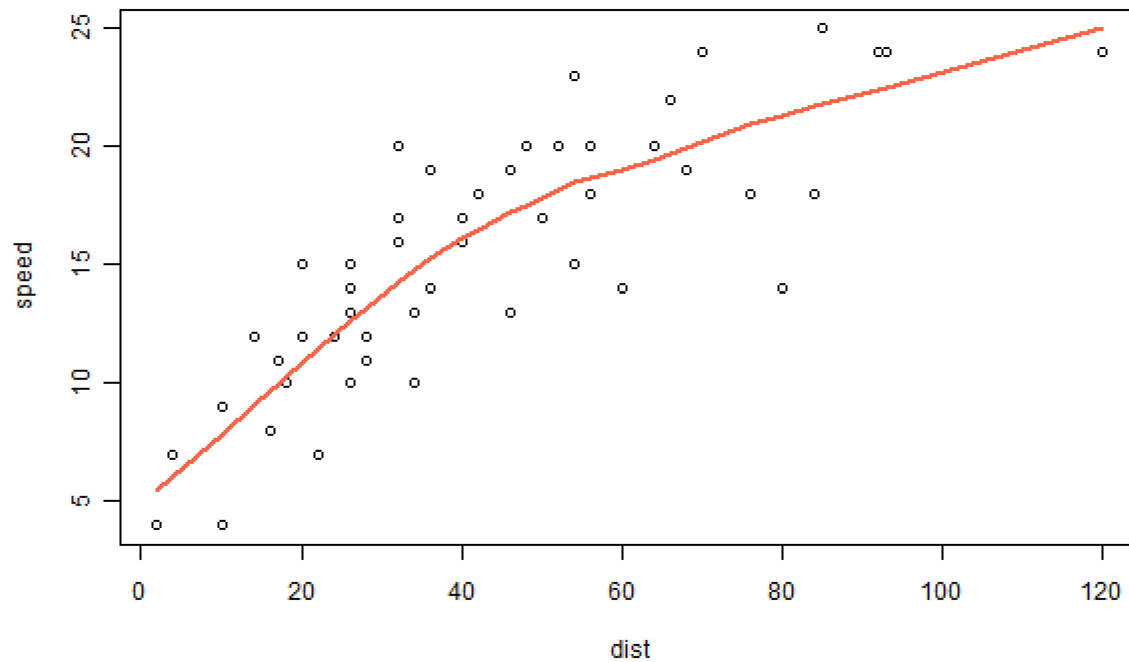
Syntaxe

- abline() rajoute une droite à un graphique
- ifelse() sélectionne l’une ou l’autre valeur en se basant sur une condition logique
- lty spécifie le type de ligne : 1 pour solide, 2 pour tirets, 3 pour pointillés

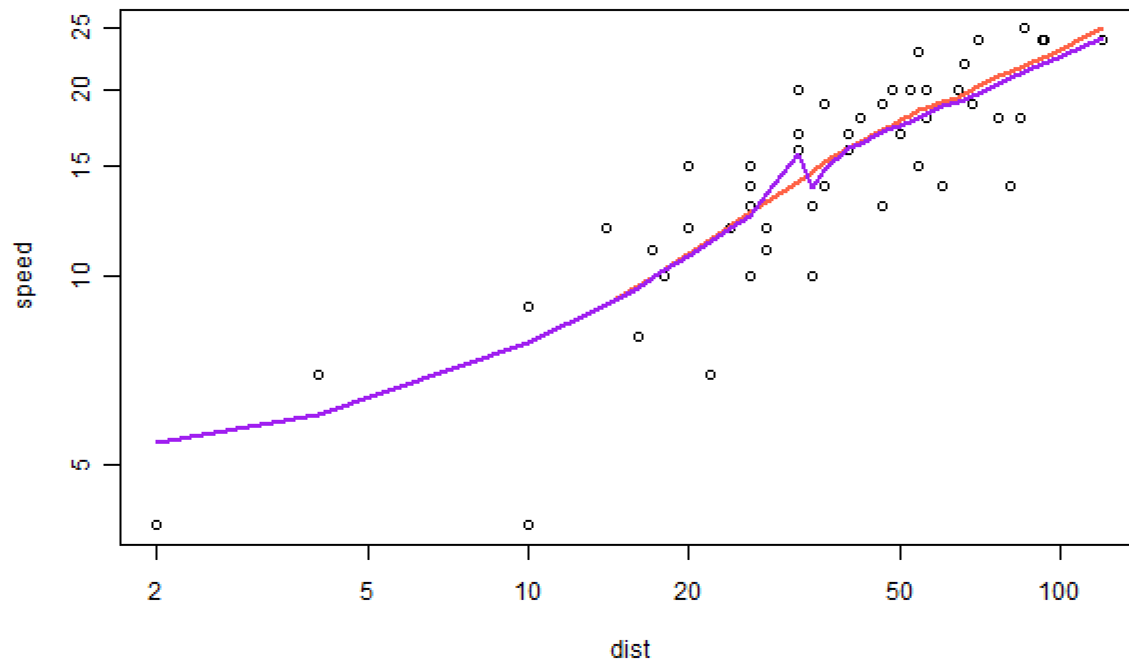
Compléter un graphique

```
data(cars)
plot(speed~dist ,data=cars)
with(cars, lines(lowess(dist,speed), col="tomato", lwd=2))
plot (speed~dist , data=cars, log="xy")
with(cars, lines(lowess(dist,speed), col="tomato", lwd=2))
with(cars, lines(supsmu(dist,speed), col="purple", lwd=2))
legend(2,25, legend=c("lowess" , "supersmoother")
        ,bty="n", lwd=2,col=c("tomato" , "purple"))
```

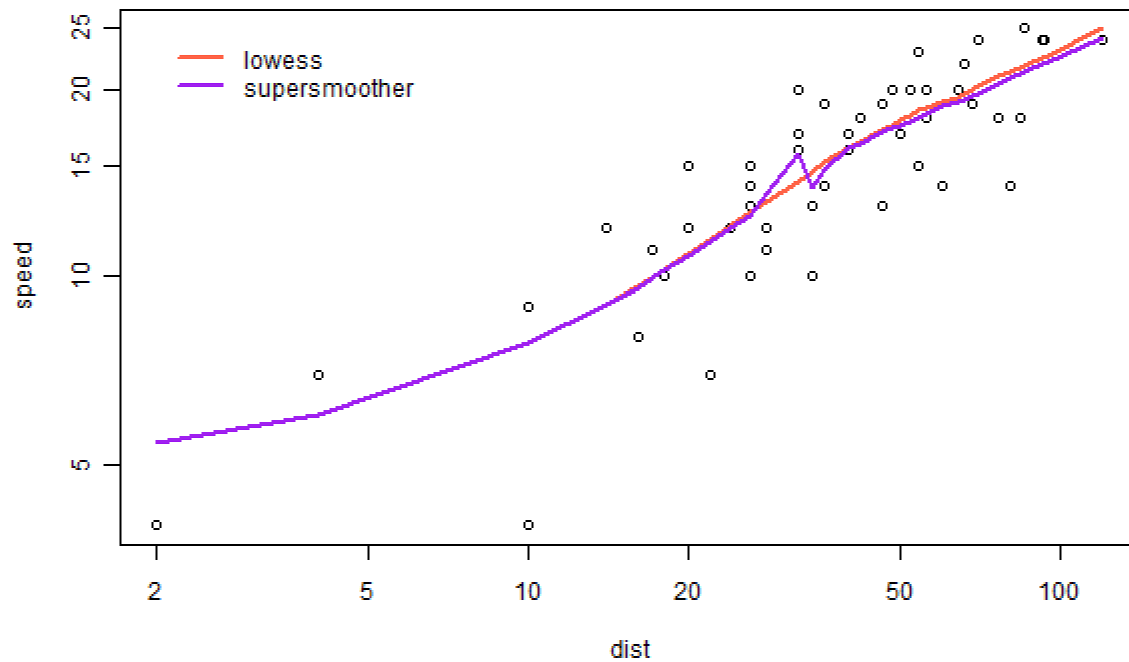
Compléter un graphique



Compléter un graphique



Compléter un graphique



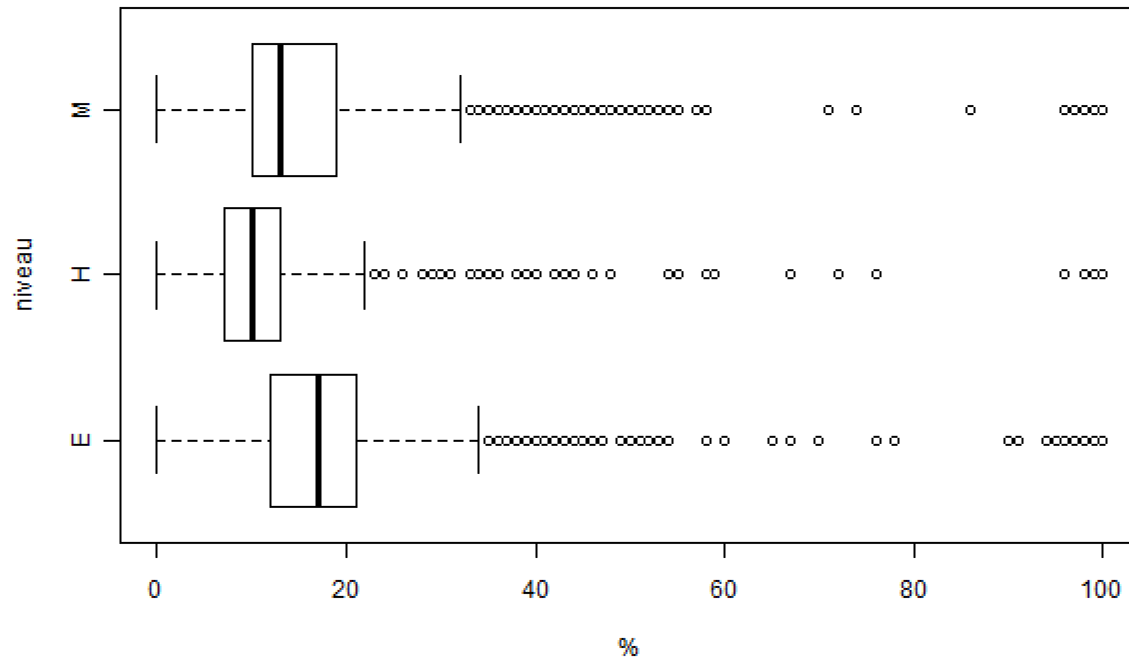
Syntaxe

- lines rajoute des lignes à un graphique existant (points() rajoute des points).
- lowess() et supsmu() permettent de lisser un nuage de points. Elles tracent des courbes lisses qui s'ajustent localement à la dépendance entre y et x
- log="xy" spécifie que les deux axes doivent être logarithmiques (si log="x", cela concerne uniquement les abcisses)
- legend() rajoute une légende

Boxplots (diagrammes en boîtes)

```
data(api, package="survey")
boxplot(mobility˜stype ,data=apipop,
        horizontal=TRUE,xlab="%",ylab="niveau")
```

Boxplots (diagrammes en boîtes)



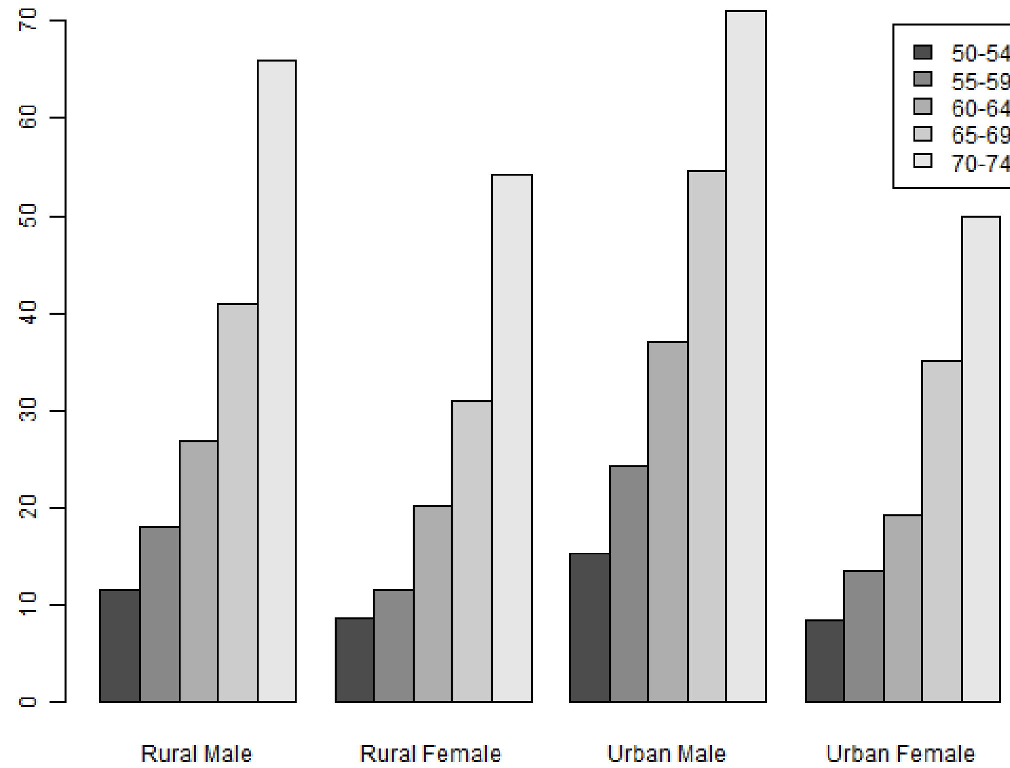
Syntaxe

- boxplot permet d'obtenir des boxplots
- horizontal=TRUE spécifie que le boxplot doit être horizontal
- xlab et ylab sont des options permettant de spécifier les labels des abcisses et des ordonnées

Diagramme en barres

barplot (VADeaths ,beside=TRUE, legend=TRUE)

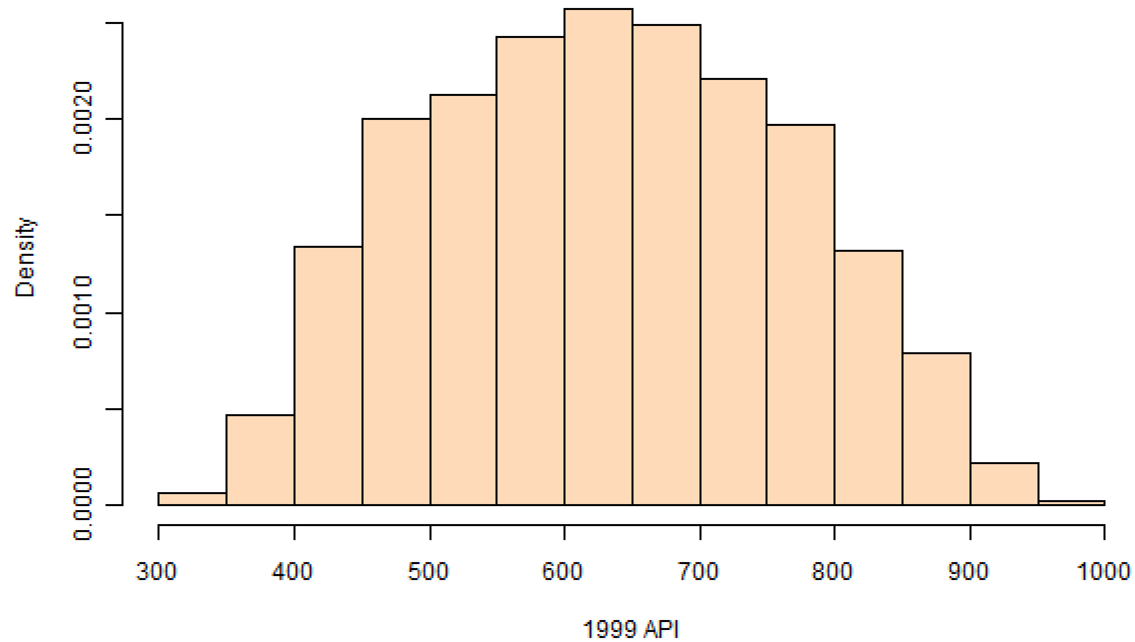
Diagramme en barres



Histogrammes

- hist(apipop$api99,col="peachpuff" ,xlab="1999 API",main="" ,prob=TRUE)

Histogrammes



Syntaxe

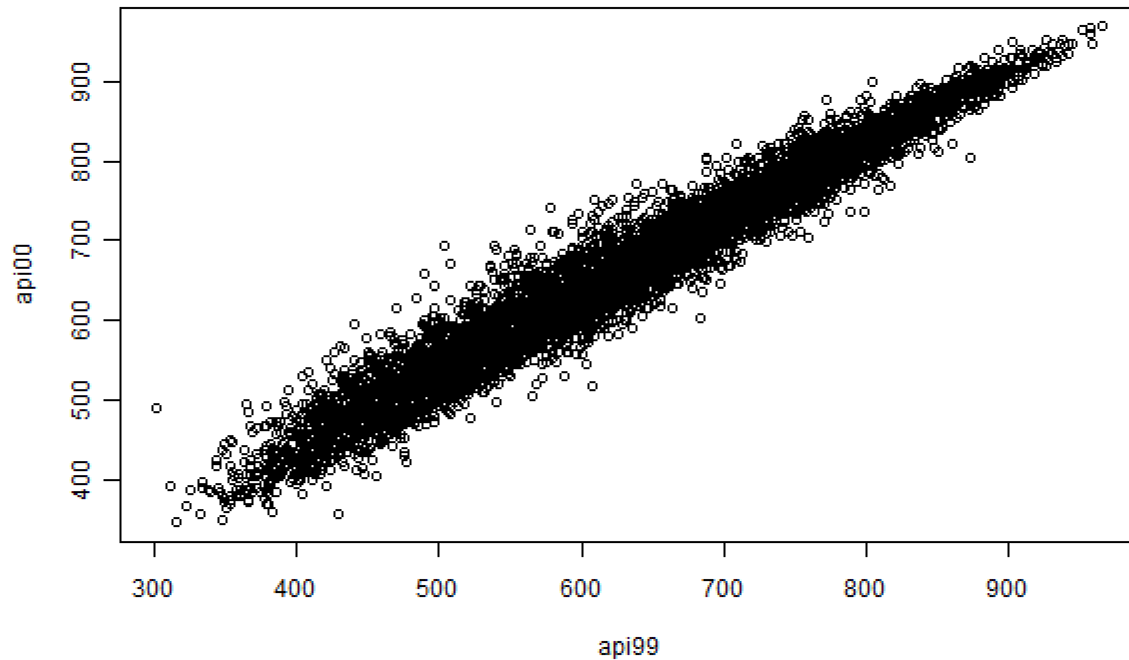
- main= specifies le titre du graphique
- prob=TRUE spécifie que l’histogramme doit être fait en fréquences relatives

Données nombreuses

- Des nuages de points peuvent être rapidement saturés, par exemple

```
data(api,package="survey")
plot(api00~api99,data=apipop)
```

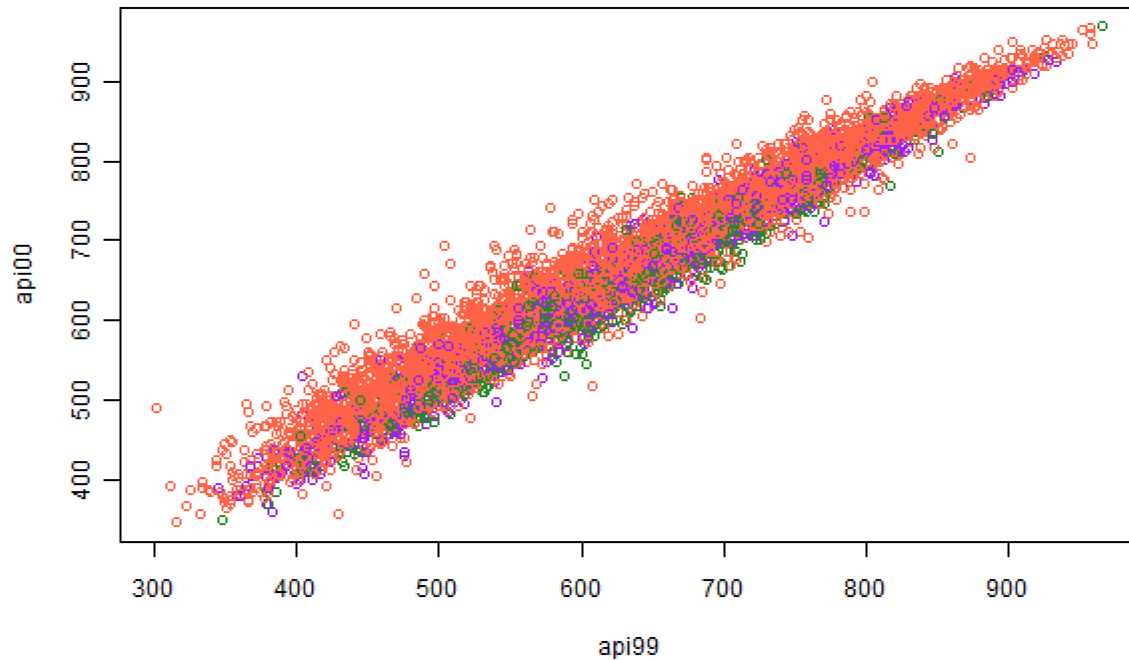
Données nombreuses



Données nombreuses

```
colors<-c("tomato" , "forestgreen",
        "purple")[apipop$stype]
plot (api00~api99 , data=apipop, col=colors)
```

Données nombreuses

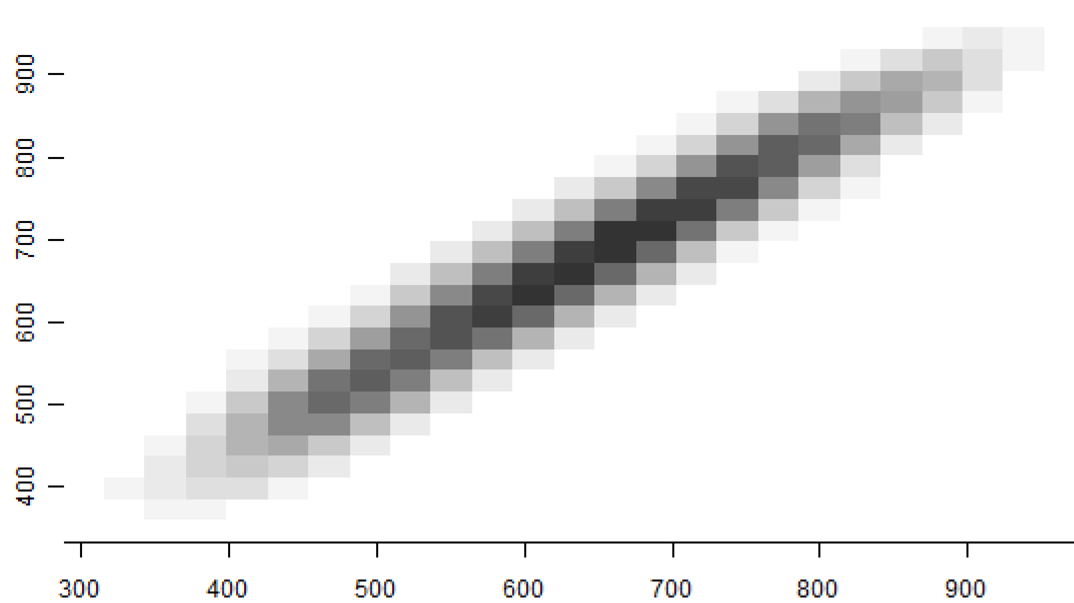


Graphiques de Densité

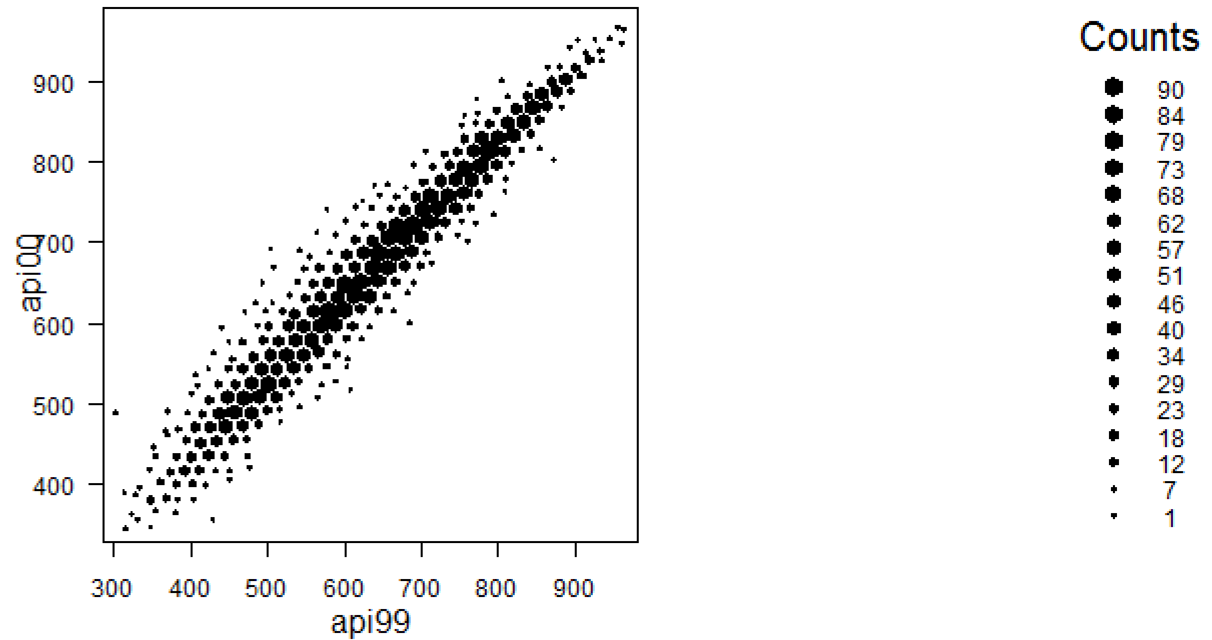
- Pour des données nombreuses, on peut avoir recours à des estimations de densité

```
library (MASS)
with(apipop, image(kde2d(api99,api00),
        col=grey(seq(1 ,0.2, length=20))))
library (hexbin)
with(apipop, plot(hexbin(api99,api00),
        style="centroids"))
```

Graphiques de Densité



Graphiques de Densité



Syntaxe

- kde2d dans le package MASS permet d’obtenir une estimation par noyau d’une densité en deux dimensions. Elle renvoie la densité des points sur une grille rectangulaire.
- image permet d’obtenir une image à partir de données sur une grille triangulaire
- Des variantes sont contour, qui trace des contour et filled.contour qui trace les contours et colorie le graphique obtenu.

Syntaxe

- hexbin dans le package hexbin package calcule le nombre de points dans chaque cellule hexagonale
- L'option style=centroids représente des hexagones pleins dont la taille dépend du nombre de points qu'ils contiennent et dont l'emplacement correspond au barycentre de ceux-ci

Gérer le code et les données

- Il existe différentes façons de sauvegarder l'information sous R

1. Le Workspace. Lorsque R démarre, il lit le fichier .RData et lorsqu'il est refermé, il propose de sauvegarder le workspace dans ce fichier

On peut sauvegarder le Workspace à tout moment avec la commande save.image(). Cela permet de tout sauvegarder à l'exception des packages chargés au cours de la session.

Gérer le code et les données

2. Les fichiers binaires. La commande save() permet de sauvegarder des fonctions ou des données dans un fichier binaire. On peut y accéder avec attach() ou load().

3. Le code source. Au lieu de sauvegarder les résultats et les données, ont peut les reconstituer à partir de code source.

Gérer le code et les données

- Il y a deux façons de gérer des projets multiples sous R

1. Sauvegarder chaque projet dans un répertoire distinct et utiliser le fichier .Rdata pour tout regrouper. Pour partager des fonctions ou objets entre projets, ils doivent être exportés et importés explicitement. *Tout est basé sur le fichier .Rdata*. Les scripts ou fichiers sources ne servent qu'à la documentation.

Gérer le code et les données

2. Tout sauvegarder sous la forme de code source. Pour chaque analyse, créer un fichier qui lit les données, les transforme et éventuellement les sauvegarde dans un nouveau fichier. *Tout est basé sur le code source*. Les fichiers sauvegardés ne servent qu'à économiser du temps.

Gérer le code et les données

- La seconde méthode, basée sur le code source, est préférable à la première car
- S'il y a un problème de lecture du fichier .Rdata, on perd tout dans la première approche
- Il est plus facile d'organiser et de récupérer l'information avec la seconde approche

Rediriger les sorties

- Pour rediriger les sorties textes vers un fichier

```
sink("fichier")
```

- Pour annuler la redirection

```
sink()
```

- Les messages d'erreur ne sont pas redirigés.
- Pour que les sorties s'affichent simultanément à la console, utiliser

```
sink("fichier",split=TRUE)
```

Mise en forme des sorties

- Les sorties R sont par défaut au format texte
- La fonction xtable() du package xtable permet d'obtenir des tableaux LATEX ou HTML à partir de matrices ou des sorties des fonctions de modélisation statistique.
- Les sorties HTML peuvent être sauvegardées dans un fichier et récupérées sous Word ou Powerpoint.

Débogage et optimisation

- Le premier souci lorsqu'on programme est d'obtenir un programme correct, ce qui est plus facile si le programme est écrit de façon simple et claire.
- Certaines clarifications ont l'avantage d'aboutir à un code plus rapide. Par exemple, travailler directement sur les vecteurs plutôt que sur leurs éléments. D'autres n'ont pas d'impact sur la rapidité, comme les fonctions *apply. Certaines ralentissent l'exécution comme l'affectation de noms aux composantes des vecteurs.
- Une fois que le code obtenu est correct, l'étape suivante est de déterminer les parties qui sont trop lentes pour les optimiser.

Chronométrage

- R dispose de la fonction proc.time(), qui renvoie l'instant courant. On peut le sauvegarder avant le début d'une partie d'un programme et le soustraire à la valeur correspondante obtenue après.
- R dispose de la fonction system.time() pour chronométrer l'évaluation d'une expression
- Sous R, Rprof(fichier) lance le profilage et Rprof(NULL) le stoppe. Le profilage consiste à écrire dans le fichier une liste des fonctions utilisées plusieurs fois par seconde.
- summaryRprof(fichier) permet d'obtenir un résumé du fichier de profilage obtenu, notamment le temps écoulé pour chaque fonction.

Mémoire

- R dispose de la fonction memory.size, qui permet d'obtenir la valeur de la mémoire allouée courante et le maximum atteint de mémoire allouée
- gc() (garbage collector) fournit le maximum alloué depuis le dernier appel de gc(reset=TRUE)
- gcinfo(TRUE) spécifie qu'un rapport sur l'utilisation de la mémoire doit être fourni chaque fois que le garbage collector est lancé.

Débogage

- traceback() identifie où a eu lieu la dernière erreur : quelle fonction était appelée, d'où elle était appelée, etc jusqu'au niveau le plus haut.
- options(error=dump.frames) sauvegarde l'état d'un programme lorsqu'une erreur a lieu. debugger() permet alors de lancer le débogueur pour examiner l'état des fonctions appelées.
- options(error=recover) lance le débogueur dès qu'une erreur a lieu.

Débogage

- La commande browser() lance le débogueur de l'endroit où elle est insérée
- options(warn=2) convertit tous les warnings en erreurs.
- debug(fnom) lance le débogueur lorsque la fonction fnom() est appelée.
- Le débogueur consiste en une ligne de commande interactive qui permet d'examiner l'état des paramètres d'une fonction.
- Un accès plus souple au débogueur est fourni par la fonction trace() (voir l'aide).

Optimiser le code

- Les opérations sur les vecteur sont rapides
- Les opérations sur les matrices peuvent être plus rapide qu'en C
- Les fonctions ayant peu d'options et peu de contrôles d'erreur sont plus rapide, par exemple : sum(x)/length(x) est plus rapide que mean(x)
- L'allocation de la mémoire en une fois est plus rapide qu'une allocation incrémentale, par exemple : utiliser

x<-numeric(10000); x[i]<-f(i) plutôt que

x<-c(x,f(i))

Optimiser le code

- Les opérations sur les tableaux de données sont plus lentes que les opérations sur les matrices, notamment pour de grands tableaux
- Manquer de mémoire ralentit beaucoup l’exécution des programmes, notamment sous Windows
- Avoir recours à C pour certaines parties d’un programme peut améliorer sa rapidité de façon spectaculaire

Objets

- Un objet peut être vu comme une structure d'information sur laquelle peuvent agir un certains nombre de fonctions.
- Sous R, les vecteurs, matrices, tableaux de données,…etc sont des objets. Il est possible d'accéder à leurs caractéristiques internes directement.
- Pour identifier les caractéristiques internes d'un objet, on peut utiliser str et names.

Fonctions génériques

- De nombreuses fonctions de R sont génériques. Cela signifie que la fonction en elle-même (par ex. plot, summary, mean) ne fait rien. La tâche correspondante est réalisée par une méthode adaptée à la représentation graphique, au résumé ou à la moyennisation d'un certain type d'objets.
- Si on passe un tableau de données en argument de summary, R va chercher la fonction summary.data.frame et appelle cette fonction à la place. S'il n'existe pas de méthode spécifique à l'objet en argument, R lance summary.default
- On peut déterminer tous les types d'objets que R sait résumer à l'aide de la fonction methods()

Fonctions génériques

```
> methods("summary")

 [1] summary.aov           summary.aovlist        summary.aspell*
 [4] summary.connection    summary.data.frame     summary.Date
 [7] summary.default       summary.ecdf*          summary.factor
[10] summary.glm           summary.infl           summary.lm
[13] summary.loess*        summary.manova         summary.matrix
[16] summary.mlm           summary.nls*           summary.packageStatus*
[19] summary.POSIXct       summary.POSIXlt        summary.ppr*
[22] summary.prcomp*       summary.princomp*      summary.rsnns*
[25] summary.srcfile       summary.srcref         summary.stepfun
[28] summary.stl*          summary.table          summary.tukeysmooth*
```

Méthodes

- Le système objet/méthodes permet d'enrichir facilement la liste des structures d'information gérées par R tout en assurant leur compatibilité avec celles existantes en terme de fonctionnement.
- Quelques méthodes standard : print, summary, plot, image, coef, vcov, anova, logLik, AIC, residuals.

Nouvelles classes

- Créer une nouvelle classe d'objets se fait très simplement

```
class(x) <- "L3.MIS"
```

- R recherchera automatiquement les méthodes print.L3.MIS, summary.L3.MIS, …etc.