

Les analyses statistiques dans R

par
Odile Wolber

1. Les packages d'analyse statistique

De nombreux packages sont disponibles pour réaliser des analyses statistiques. Certains font partie des packages standards distribués avec une installation de base de R, il s'agit notamment :

- du package *stats* qui permet de réaliser un large éventail d'analyses statistiques : tests classiques, modèles linéaires (régression par les moindres carrés, modèles linéaires généralisés, analyse de la variance), lois de distribution, résumés statistiques, classifications hiérarchiques, séries temporelles, moindres carrés non linéaires, analyses multivariées.
- du package *stats4* qui contient des fonctions statistiques utilisant les classes du système de classes de S version 4 ;
- du package *splines* qui permet de réaliser des analyses utilisant les représentations polynomiales.

Seul le package *stats* est chargé au démarrage de R. Les autres packages doivent être chargés avec l'instruction `library()`.

De nombreux packages contribués allongent la liste des analyses possibles avec R. Une liste complète de ces packages contribués, accompagnée d'une description, se trouve sur le site Web du CRAN (<http://cran.r-project.org/src/contrib/PACKAGES.html>).

Certains de ces packages sont regroupés parmi les packages recommandés car ils couvrent des méthodes souvent utilisées en analyse des données. Ils sont brièvement décrits dans le tableau ci-dessous.

Principaux packages contribués

<code>boot</code>	méthodes de ré-échantillonnage et de bootstrap
<code>class</code>	méthodes de classification
<code>cluster</code>	méthodes d'aggrégation
<code>KernSmooth</code>	méthodes pour le calcul de fonctions de densité
<code>mgcv</code>	modèles additifs généralisés
<code>nlme</code>	modèles linéaires ou non-linéaires à effets mixtes
<code>nnet</code>	réseaux neuronaux et modèles log-linéaires multinomiaux
<code>rpart</code>	méthodes de partitionnement récursif
<code>sm</code>	estimation non paramétrique de la densité et régression non paramétrique
<code>spatial</code>	analyses spatiales (< kriging >, covariance spatiale, . . .)
<code>survival</code>	analyses de survie

Le tableau suivant fournit la liste des principales fonctions d'analyse statistique. Lorsqu'on ne le précise pas, la fonction est disponible dans le package *stats*.

Principales fonctions statistiques

Analyses factorielles

<code>prcomp</code> , <code>princomp</code>	analyse en composantes principales
<code>corresp</code> (<i>package MASS</i>)	analyse factorielle des correspondances
<code>mca</code> (<i>package MASS</i>)	analyse des correspondances multiples
<code>lda</code>	analyse discriminante
<code>cca</code> (<i>package ade4</i>)	analyse canonique

Principales fonctions statistiques

Classifications

hclust classification hiérarchique. On peut ensuite utiliser la fonction `plot` pour visualiser le dendrogramme avec `plot` : par exemple, `plot(hclust(dist(a)))`, où `a` est une `data.frame` avec un identifiant et des variables numériques.

Typologies

La fonction `clusplot` (package `cluster`) permet de représenter les classes obtenues dans un plan. `Clusplot` est une fonction générique (cf. 4.2).

kmeans typologie – méthode des centres mobiles
pam (package `cluster`) partition autour des médoïdes¹ (plus robuste que `kmeans`)
clara (package `cluster`) partition à partir de larges échantillons (on tire d'abord un sous-échantillon sur lequel s'effectue la partition)
daisy (package `cluster`) partition à partir d'une matrice de dissimilarité ; variables qualitatives ou quantitatives
dist partition à partir d'une matrice de dissimilarité ; variables quantitatives uniquement
fanny (package `cluster`) classification floue

Régressions

lm modèle linéaire
glm modèle linéaire généralisé. Exemple pour une régression de Poisson de y sur x_1 , x_2 et x_3 : `summary(glm(y ~ x1 + x2 + x3, family=poisson))`.
nls moindres carrés non linéaires
gls (package `nlme`) moindres carrés généralisés
tsls (package `sem`) variables instrumentales, système d'équations structurelles

Econométrie des variables qualitatives

logit (package `gtools`) modèle logit

Econométrie des panels

lme (package `nlme`) économétrie des panels
Analyse de la variance

aov analyse de la variance

Séries temporelles

(cf. plus particulièrement le package `pastecs`)

arima modélisation ARIMA

spectrum analyse spectrale

2. Les formules

Les formules utilisées sont les mêmes pour presque toutes les fonctions. Une formule est typiquement de la forme $y \sim \text{model}$ où y est la réponse analysée et `model` est un ensemble de termes pour lesquels les paramètres sont estimés. Ces termes sont séparés par des symboles arithmétiques qui ont une signification particulière.

a+b effets additifs de a et de b

X si X est une matrice, ceci équivaut à un effet additif de toutes ses colonnes, c'est-à-dire $X[,1]+X[,2]+\dots+X[,n\text{col}(X)]$; certaines de ces colonnes peuvent être sélectionnées avec l'indexation numérique (ex. : $X[,2:4]$)

a:b effet interactif entre a et b

a*b effets additifs et interactifs (identique à $a+b+a:b$)

poly(a, n) polynôme de a jusqu'au degré n

¹ L'objet représentatif d'une classe, le médoïde, est l'objet pour lequel la dissimilarité moyenne par rapport à tous les objets dans la classe est la plus petite.

n	inclut toutes les interactions jusqu'au niveau n , c'est-à-dire $(a+b+c)^2$ est identique à $a+b+c+a:b+a:c+b:c$
$b \%in\% a$	les effets de b sont hiérarchiquement inclus dans a (identique à $a+a:b$ ou a/b)
$-b$	supprime l'effet de b , par exemple : $(a+b+c)^2-a:b$ est identique à $a+b+c+a:c+b:c$
-1	$y \sim x-1$ force la régression à passer par l'origine (idem pour $y \sim x+0$ ou $0+y \sim x$)
1	$y \sim 1$ ajuste un modèle sans effets (juste l'intercept)
<code>offset(...)</code>	ajoute un effet au modèle sans estimer de paramètre (par ex. <code>offset(3*x)</code>)

Les opérateurs arithmétiques des formules utilisées dans R un sens différent de celui qu'ils ont dans une expression classique.

Exemples :

La formule $y \sim x_1+x_2$ définira le modèle $y = \beta_1 x_1 + \beta_2 x_2 + \alpha$, et non pas $y = \beta(x_1 + x_2) + \alpha$.

Pour obtenir le modèle $y = \beta(x_1 + x_2) + \alpha$, on utilisera la fonction `l` : la formule $y \sim l(x_1+x_2)$ définira alors le modèle.

De même, pour définir le modèle $y = \beta_1 x + \beta_2 x^2 + \alpha$, on utilisera la formule $y \sim \text{poly}(x,2)$ et non pas $y \sim x + x^2$.

Il est possible d'inclure une fonction dans une formule afin de transformer une variable. Ainsi, dans l'exemple suivant, on veut utiliser la racine carrée (`sqrt`) d'une variable :

```
aov.spray <- aov(sqrt(InsectSprays[, 1]) ~ InsectSprays[, 2])
```

La fonction `aov()` accepte une syntaxe particulière pour spécifier les effets aléatoires. Par exemple, $y \sim a + \text{Error}(b)$ signifie effets additifs d'un terme fixe (a) et d'un terme aléatoire (b).

3. Attributs des résultats des analyses

Les résultats des analyses statistiques sont des listes. Les listes, comme on l'a vu à la session R01, contiennent différents objets de types et de modes différents. Le contenu des listes retournées par les fonctions statistiques est déterminé par la classe de la fonction. Ainsi, la fonction `aov` (analyse de la variance) retourne une liste de classe "aov", `lm` retourne une liste de classe "lm".

Selon que la classe de la liste à afficher, les informations ne seront pas les mêmes. Les fonctions utilisées pour extraire les résultats agissent spécifiquement en fonction de la classe de la liste : c'est le propre des fonctions dites "génériques", qui ont une syntaxe unique quelle que soit la liste passée en argument.

4. Affichage des résultats

Les fonctions statistiques produisent une liste dont les différents éléments correspondent aux résultats de l'analyse. Nous allons présenter sur un exemple de régression linéaire la manière de procéder pour afficher les résultats.

Exemple : nous reprenons l'exemple du cours d'Arthur Tenenhaus sur le prix des appartements, tiré du livre de Michel Tenenhaus « méthodes statistiques en gestion ». On dispose pour 28 appartements du prix, de la surface et du prix au m². Les données stockées dans la `data.frame` `appart` sont présentées dans le tableau suivant :

	Rue	Surf	Prix	Prix au m2	Rue	Surf	Prix	Prix au m2
1	censier	28	650	23.21	15	montparnasse	40 1000	25.00
2	contrescarpe	50	1400	28.00	16	rue d'assas	260 7500	28.85
3	saint-simon	106	3250	30.66	17	St-germain	70 1625	23.21
4	Rapp	196	4000	20.41	18	île St-Louis	117 4750	40.6
5	St-A.des arts	55	1340	24.36	19	jussieu	90 1890	21.00
6	près quais	190	3950	20.79	20	quartier-latin	30 390	13.00

	Rue	Surf	Prix	Prix au m2	Rue	Surf	Prix	Prix au m2	
7	gobelins	110	2500	22.73	21	montparnasse	105	1875	17.86
8	gobelins	60	1600	26.67	22	rue mazarine	52	1000	19.23
9	censier	48	1250	26.04	23	censier	80	1350	16.88
10	panthéon	35	1250	35.71	24	assas	60	1475	24.58
11	rue madame	86	1750	20.35	25	observatoire	140	4950	35.36
12	rue de seine	65	1500	23.08	26	rue de savoie	20	425	21.25
13	panthéon	32	775	24.22	27	Luxembourg	100	2475	24.75
14	S. -babylone	52	1225	23.56	28	gobelins	28	425	15.18

On souhaite faire la régression linéaire du prix en fonction de la surface. On utilise la fonction `lm`.

```
appart.lm=lm(appart$Prix~appart$Surf)
```

4. 1. Affichage simple du contenu de la liste créée par la fonction statistique

Reprenons l'exemple précédent : on peut afficher les résultats en utilisant simplement l'instruction `appart.lm` :

```
appart.lm
```

Call:

```
lm(formula = appart$Prix ~ appart$Surf)
```

Coefficients:

```
(Intercept) appart$Surf
-147.33      26.77
```

On peut regarder la structure de l'objet créé par `lm` avec la fonction `str` :

```
str(appart.lm,max.level=-1)
```

List of 12

```
- attr(*, "class")= chr "lm"
```

Une autre façon de regarder cette structure est d'afficher les noms des éléments de l'objet :

```
names(appart.lm)
```

```
[1] "coefficients" "residuals" "effects" "rank" "fitted.values" "assign" "qr"
[8] "df.residual" "xlevels" "call" "terms" "model"
```

Les éléments peuvent ensuite être extraits comme vu précédemment :

```
appart.lm$coefficients
```

```
(Intercept) appart$Surf
-147.32895 26.76582
```

```
appart.lm$residuals
```

```
 1      2      3      4      5      6      7      8      9     10
47.88604 209.03805 560.15224 -1098.77138 15.20896 -988.17647 -296.91103 141.37987 112.56968 460.52532
11     12     13     14     15     16     17     18     19     20
-404.53140 -92.44922 65.82277 -19.49359 76.69623 688.21627 -101.27831 1765.72824 -371.59467 -265.64559
21     22     23     24     25     26     27     28
-788.08194 -244.49359 -643.93649 16.37987 1350.11443 37.01259 -54.25285 -177.11396
```

4. 2. Affichage des résultats à partir des fonctions génériques

4. 2. 1. La fonction `summary`

On peut accéder à un résultat plus détaillé que l'instruction `appart.lm` à partir de la fonction `summary` :

```
summary(appart.lm)
```

Call:
lm(formula = appart\$Prix ~ appart\$Surf)

Residuals:
Min 1Q Median 3Q Max
-1098.771 -273.462 -2.142 119.772 1765.728

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) -147.33 206.23 -0.714 0.481
appart\$Surf 26.77 2.07 12.931 7.86e-13 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 614.7 on 26 degrees of freedom
Multiple R-Squared: 0.8654, Adjusted R-squared: 0.8603
F-statistic: 167.2 on 1 and 26 DF, p-value: 7.862e-13

Notons que la fonction summary crée également une liste.

str(summary(appart.lm))

List of 11

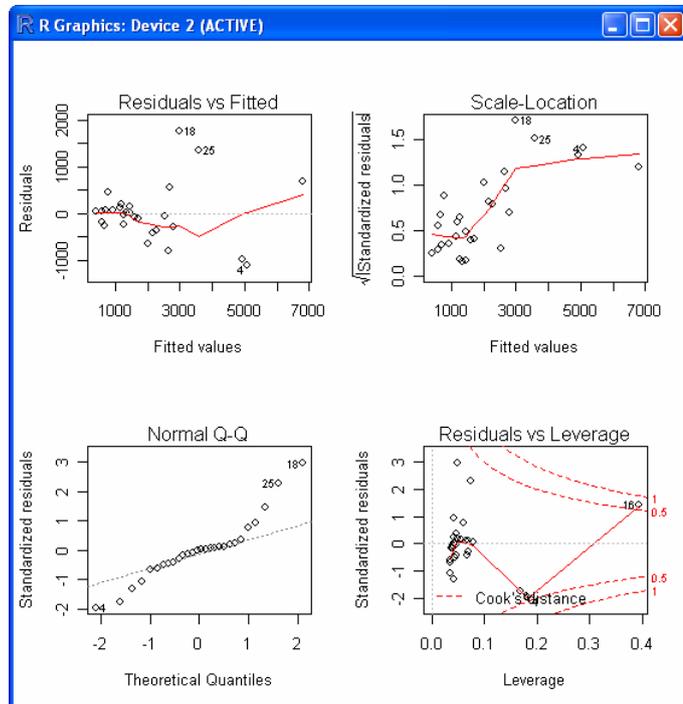
```
$ call           : language lm(formula = appart$Prix ~ appart$Surf)
$ terms          :Classes 'terms', 'formula' length 3 appart$Prix ~ appart$Surf
- attr(*, "variables")= language list(appart$Prix, appart$Surf)
- attr(*, "factors")= int [1:2, 1] 0 1
- attr(*, "dimnames")=List of 2
  $ : chr [1:2] "appart$Prix" "appart$Surf"
  $ : chr "appart$Surf"
- attr(*, "term.labels")= chr "appart$Surf"
- attr(*, "order")= int 1
- attr(*, "intercept")= int 1
- attr(*, "response")= int 1
- attr(*, ".Environment")=length 3 <environment>
- attr(*, "predvars")= language list(appart$Prix, appart$Surf)
- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
  - attr(*, "names")= chr [1:2] "appart$Prix" "appart$Surf"
$ residuals      : Named num [1:28] 47.9 209.0 560.2 -1098.8 15.2 ...
- attr(*, "names")= chr [1:28] "1" "2" "3" "4" ...
$ coefficients    : num [1:2, 1:4] -147.329 26.766 206.228 2.070 -0.714 ...
- attr(*, "dimnames")=List of 2
  $ : chr [1:2] "(Intercept)" "appart$Surf"
  $ : chr [1:4] "Estimate" "Std. Error" "t value" "Pr(>|t|)"
$ aliased        : Named logi [1:2] FALSE FALSE
- attr(*, "names")= chr [1:2] "(Intercept)" "appart$Surf"
$ sigma          : num 615
$ df             : int [1:3] 2 26 2
$ r.squared      : num 0.865
$ adj.r.squared  : num 0.86
$ fstatistic     : Named num [1:3] 167 1 26
- attr(*, "names")= chr [1:3] "value" "numdf" "dendf"
$ cov.unscaled   : num [1:2, 1:2] 1.13e-01 -9.33e-04 -9.33e-04 1.13e-05
- attr(*, "dimnames")=List of 2
  $ : chr [1:2] "(Intercept)" "appart$Surf"
  $ : chr [1:2] "(Intercept)" "appart$Surf"
```

4. 2. 2. La fonction plot

La fonction plot propose également différentes sorties graphiques en fonction de la classe de la liste passée en argument.

Exemple : le graphique suivant illustre l'utilisation de la fonction plot sur le résultat de la régression mise en œuvre sur le fichier appart avec la fonction lm.

```
layout(matrix(1:4,2,2))  
plot(appart.lm)
```



4. 2. 3. Autres fonctions génériques

Le tableau suivant résume les autres principales fonctions génériques qui permettent d'extraire des informations d'un objet qui résulte d'une analyse.

print	retourne un résumé succinct
df.residual	retourne le nombre de degrés de liberté résiduel
coef	retourne les coefficients estimés (avec parfois leurs erreurs-standards)
residuals	retourne les résidus
deviance	retourne la déviance
fitted	retourne les valeurs ajustées par le modèle
logLik	calcule le logarithme de la vraisemblance et le nombre de paramètre d'un modèle
AIC	calcule le critère d'information d'Akaike ou AIC (dépend de logLik())

Le tableau suivant indique certaines fonctions génériques qui font des analyses supplémentaires à partir d'un objet qui résulte d'une analyse faite au préalable, l'argument principal étant cet objet, mais dans certains cas un argument supplémentaire est nécessaire comme pour predict ou update.

add1	teste successivement tous les termes qui peuvent être ajoutés à un modèle
drop1	teste successivement tous les termes qui peuvent être enlevés d'un modèle
step	sélectionne un modèle par AIC (fait appel à add1 et drop1)
anova	calcule une table d'analyse de variance ou de déviance pour un ou plusieurs modèles

predict calcule les valeurs prédites pour de nouvelles données à partir d'un modèle. On peut ensuite utiliser la fonction termplot pour obtenir le graphe des effets (partiels) d'un modèle de régression.

update ré-ajuste un modèle avec une nouvelle formule ou de nouvelles données

Exemple : on poursuit l'affichage des résultats de la précédente analyse. On désire maintenant afficher les intervalles de prédiction et les intervalles de confiance grâce à la fonction predict :

```
pred.w.plim=predict(lm(appart$Prix~appart$Surf), interval="prediction")
```

	fit	lwr	upr
1	602.1140	-704.37931	1908.607
2	1190.9620	-102.25810	2484.182
3	2689.8478	1400.01973	3979.676
...			
26	387.9874	-924.95477	1700.930
27	2529.2529	1241.16735	3817.338
28	602.1140	-704.37931	1908.607

```
pred.w.clim=predict(lm(appart$Prix~appart$Surf), interval="confidence")
```

	fit	lwr	upr
1	602.1140	269.79548	934.4324
2	1190.9620	915.40975	1466.5142
3	2689.8478	2430.68129	2949.0142
...			
26	387.9874	31.15681	744.8180
27	2529.2529	2278.90275	2779.6030
28	602.1140	269.79548	934.4324

Pour visualiser la bande confiance et de prédiction, on peut utiliser le programme ci-dessous. Le tri préalable du fichier appart est nécessaire pour le tracé des lignes pour l'intervalle de la droite de régression et l'intervalle des observations.

```
ord=order(appart$Prix)
appart$Prix=appart$Prix[ord]
appart$Surf=appart$Surf[ord]
reg=lm(appart$Prix~appart$Surf)

reg=lm(appart$Prix~appart$Surf)

# Intervalle de confiance pour la droite de régression
pred1=predict(reg,interval="confidence")

# Intervalle de prédiction
pred2=predict(reg,interval="prediction")

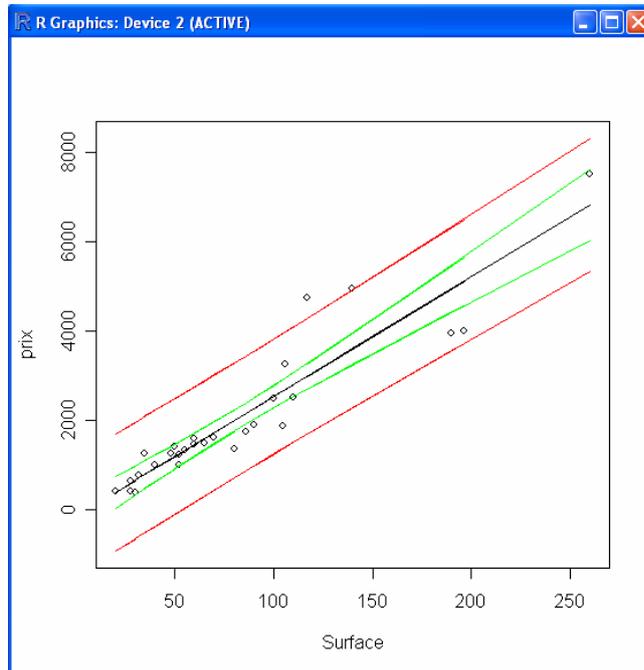
# Trouver les valeurs minimales et maximales pour les intervalles
l.min=min(pred1[,2],pred2[,2])
l.max=max(pred1[,3],pred2[,3])

plot(appart$Surf,appart$Prix,ylim=c(l.min,l.max),
+ xlab="Surface",ylab="prix")

# droite de régression
lines(appart$Surf,pred1[,1],lwd=1)

# lignes pour l'intervalle de la droite de régression
lines(appart$Surf,pred1[,2],lwd=1,col="green")
lines(appart$Surf,pred1[,3],lwd=1,col="green")
```

```
# lignes pour l'intervalle des observations
lines(appart$Surf,pred2[,2],lwd=1,col="red")
lines(appart$Surf,pred2[,3],lwd=1,col="red")
```



Si on souhaite tester le modèle sur de nouvelles données X_{new} , on utilise la fonction `predict(appart.lm, Xnew)`.

5. Trois exemples d'analyses statistiques

Nous reprenons dans cette partie les exemples des polycopiés de cours d'Arthur Tenenhaus. Nous réalisons successivement : une analyse discriminante, une typologie et une analyse en composantes principales, avec pour chaque analyse une représentation graphique des résultats.

5. 1. Analyse discriminante : fonction `lda`

L'analyse discriminante est réalisée sur le fichier des iris de Fisher, disponible dans le package `datasets`. On dispose des variables suivantes : `Sepal.Length` (longueur des sépales), `Sepal.Width` (largeur des sépales), `Petal.Length` (longueur des pétales), `Petal.Width` (largeur des pétales), `Species` (espèce).

On veut construire les axes discriminants des différentes espèces d'iris. On utilise la fonction `lda` (*linear discriminant analysis*) du package `MASS`.

```
library(MASS)
iris.lm=lda(iris[,1:4],iris[,5])
```

On peut afficher le contenu de l'analyse avec l'instruction `str(iris.lm)` :

List of 8

```
$ prior : Named num [1:3] 0.333 0.333 0.333
- attr(*, "names")= chr [1:3] "setosa" "versicolor" "virginica"
$ counts : Named int [1:3] 50 50 50
- attr(*, "names")= chr [1:3] "setosa" "versicolor" "virginica"
$ means : num [1:3, 1:4] 5.01 5.94 6.59 3.43 2.77 ...
- attr(*, "dimnames")=List of 2
 $ : chr [1:3] "setosa" "versicolor" "virginica"
 $ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
```

```

$ scaling : num [1:4, 1:2] 0.8294 1.5345 -2.2012 -2.8105 0.0241 ...
- attr(*, "dimnames")=List of 2
 $ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
 $ : chr [1:2] "LD1" "LD2"
$ lev     : chr [1:3] "setosa" "versicolor" "virginica"
$ svd     : num [1:2] 48.64 4.58
$ N       : int 150
$ call    : language lda(x = iris[, 1:4], iris[, 5])
- attr(*, "class")= chr "lda"

```

L'instruction `iris.lda` affiche les resultants suivants :

Call:

```
lda(iris[, 1:4], iris[, 5])
```

Prior probabilities of groups:

```

setosa    versicolor  virginica
0.3333333 0.3333333 0.3333333

```

Group means:

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
setosa	5.006	3.428	1.462	0.246
versicolor	5.936	2.770	4.260	1.326
virginica	6.588	2.974	5.552	2.026

Coefficients of linear discriminants:

	LD1	LD2
Sepal.Length	0.8293776	0.02410215
Sepal.Width	1.5344731	2.16452123
Petal.Length	-2.2012117	-0.93192121
Petal.Width	-2.8104603	2.83918785

Proportion of trace:

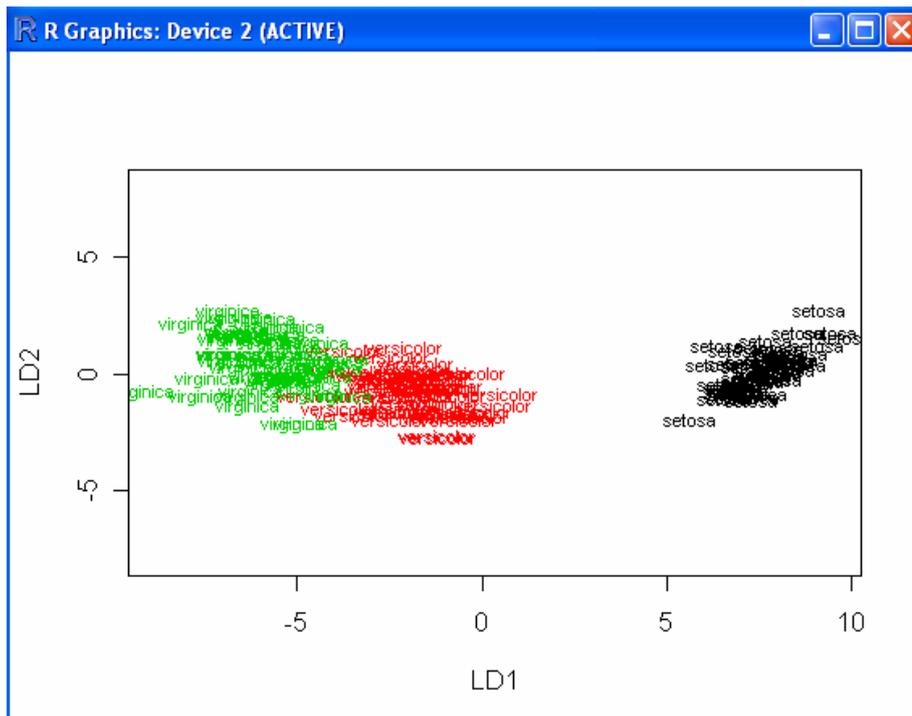
```

LD1 LD2
0.9912 0.0088

```

On peut représenter les données sur les axes discriminants avec l'instruction :

```
plot(iris.lda, col=as.numeric(iris[,5]))
```



Si on souhaite tester le modèle sur de nouvelles données X_{new} , on utilise la fonction `predict(iris.lda, Xnew)`.

On peut voir ce que donne le modèle sur les données précédentes avec les instructions suivantes :

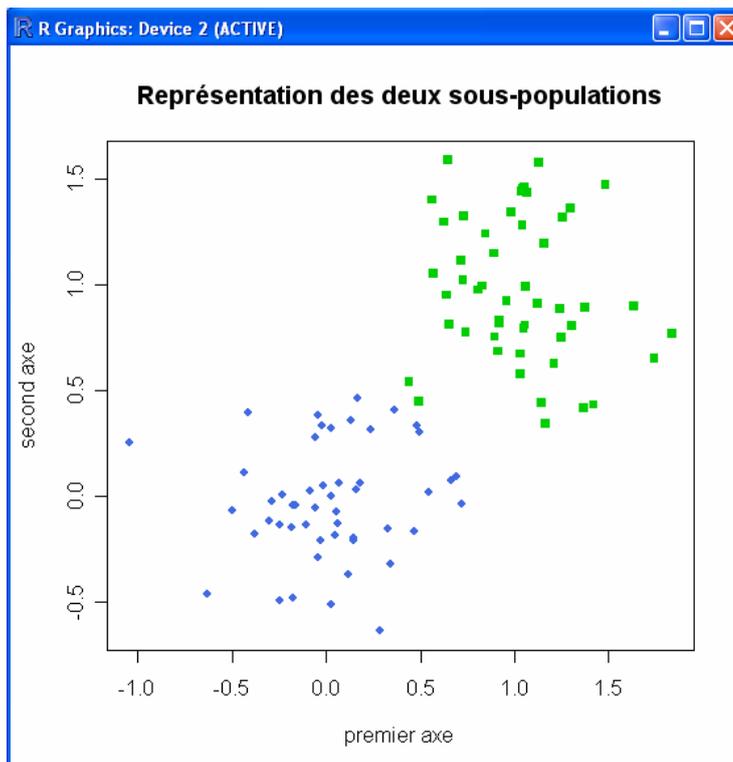
```
result.lda=predict(iris.lda,iris[,1:4])  
result.lda
```

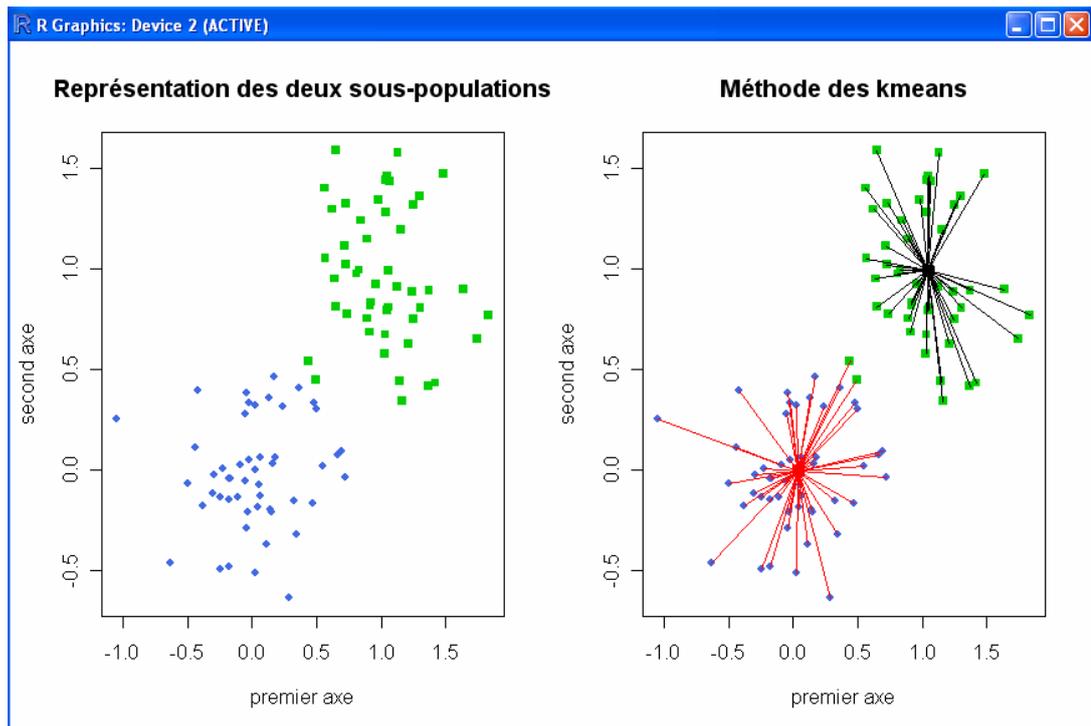
On aurait également pu construire le modèle sur la moitié du fichier, et le tester sur l'autre moitié du fichier.

5. 2. Typologie : fonction kmeans

L'objectif des kmeans est de partitionner des données en G groupes (G étant fourni par l'utilisateur). Dans l'exemple suivant, on crée deux sous-populations artificielles à partir desquelles on construit une matrice unique. On cherche ensuite une partition en deux groupes des observations de la matrice. On peut ainsi vérifier a posteriori que les deux groupes obtenus coïncident plus ou moins avec les deux sous-populations de départ.

```
# Création d'une matrice de données artificielles  
# contenant deux sous-populations  
C1=matrix(rnorm(100, sd=0.3), ncol=2)  
C2=matrix(rnorm(100, mean=1, sd=0.3), ncol=2)  
mat=rbind(C1, C2)  
  
# Visualisation des données générées  
layout(t(matrix(1:2)))  
  
plot(C1, col="royalblue", pch=16,  
      xlim=range(mat[,1]),  
      ylim=range(mat[,2]),  
      xlab="premier axe",  
      ylab="second axe",  
      main="Représentation des deux sous-populations")  
points(C2, col="green3", pch=15)
```





5. 3. Analyse en composantes principales : fonction princomp

L'exemple est tiré du livre de Michel Tenenhaus « Méthodes statistiques en gestion ». On dispose des caractéristiques de 24 voitures, reportées dans le tableau ci-dessous.

	Cylindree	Puissance	Vitesse	Poids	Largeur	Longueur
Citroen C2 1.1 Base	1124	61	158	932	1659	3666
Smart Fortwo Coupe	698	52	135	730	1515	2500
Mini 1.6 170	1598	170	218	1215	1690	3625
Nissan Micra 1.2 65	1240	65	154	965	1660	3715
Renault Clio 3.0 V6	2946	255	245	1400	1810	3812
Audi A3 1.9 TDI	1896	105	187	1295	1765	4203
Peugeot 307 1.4 HDI 70	1398	70	160	1179	1746	4202
Peugeot 407 3.0 V6 BVA	2946	211	229	1640	1811	4676
Mercedes Classe C 270 CDI	2685	170	230	1600	1728	4528
BMW 530d	2993	218	245	1595	1846	4841
Jaguar S-Type 2.7 V6 Bi-Turbo	2720	207	230	1722	1818	4905
BMW 745i	4398	333	250	1870	1902	5029
Mercedes Classe S 400 CDI	3966	260	250	1915	2092	5038
Citroen C3 Pluriel 1.6i	1587	110	185	1177	1700	3934
BMW Z4 2.5i	2494	192	235	1260	1781	4091
Audi TT 1.8T 180	1781	180	228	1280	1764	4041
Aston Martin Vanquish	5935	460	306	1835	1923	4665
Bentley Continental GT	5998	560	318	2385	1918	4804
Ferrari Enzo	5998	660	350	1365	2650	4700
Renault Scenic 1.9 dCi 120	1870	120	188	1430	1805	4259
Volkswagen Touran 1.9 TDI 105	1896	105	180	1498	1794	4391
Land Rover Defender Td5	2495	122	135	1695	1790	3883
Land Rover Discovery Td5	2495	138	157	2175	2190	4705
Nissan X-Trail 2.2 dCi	2184	136	180	1520	1765	4455

On effectue une analyse en composantes principales à partir de la matrice de corrélation (option cor=TRUE de la fonction princomp).

```
auto.pca=princomp(auto,cor=TRUE)
```

Si on avait un nom de variable pour la première colonne (le nom des modèles), l'instruction précédente ne serait pas valide. Il faudrait préciser une formule pour ne sélectionner que les variables numériques, par exemple :

```
auto.pca=princomp(data=auto,~Cylindree+Puissance+Vitesse+Poids+Largeur+Longueur,cor=TRUE)
```

Le problème dans ce cas se pose ultérieurement, lorsqu'on veut obtenir le graphique des observations et des variables dans un plan factoriel. En effet, on n'aura pas le nom des modèles, mais simplement un numéro d'observation.

Les informations que l'on obtient avec l'instruction `auto.pca` sont très parcellaires :

```
auto.pca
```

```
Call:
```

```
princomp(x = auto, cor = TRUE)
```

```
Standard deviations:
```

```
  Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  
2.1003018 0.9238018 0.6600484 0.4856651 0.2267966 0.1111369
```

```
6 variables and 24 observations.
```

On obtient également très peu d'informations avec la fonction générique `summary` :

```
summary(auto.pca)
```

```
Importance of components:
```

```
              Comp.1  Comp.2  Comp.3  Comp.4  Comp.5  Comp.6  
Standard deviation  2.1003018 0.9238018 0.6600484 0.4856651 0.2267966 0.1111369  
Proportion of Variance 0.7352113 0.1422350 0.07261066 0.03931176 0.00857278 0.002058567  
Cumulative Proportion 0.7352113 0.8774462 0.95005689 0.98936865 0.99794143 1.000000000
```

Pour accéder aux résultats détaillés de la liste par l'indexation ou par le nom. La liste est composée des éléments suivants :

```
str(auto.pca)
```

```
List of 7
```

```
$ sdev      : Named num [1:6] 2.100 0.924 0.660 0.486 0.227 ...  
- attr(*, "names")= chr [1:6] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...  
$ loadings  : loadings [1:6, 1:6] -0.458 -0.440 -0.422 -0.360 -0.381 ...  
- attr(*, "dimnames")=List of 2  
  $ : chr [1:6] "Cylindree" "Puissance" "Vitesse" "Poids" ...  
  $ : chr [1:6] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...  
- attr(*, "class")= chr "loadings"  
$ center    : Named num [1:6] 2723 207 215 1487 1838 ...  
- attr(*, "names")= chr [1:6] "Cylindree" "Puissance" "Vitesse" "Poids" ...  
$ scale     : Named num [1:6] 1484.5 152.4 55.4 379.3 216.2 ...  
- attr(*, "names")= chr [1:6] "Cylindree" "Puissance" "Vitesse" "Poids" ...  
$ n.obs     : int 24  
$ scores    : num [1:24, 1:6] 2.59592 4.15015 1.38191 2.51334 0.00313 ...  
- attr(*, "dimnames")=List of 2  
  $ : chr [1:24] "Citroen C2 1.1 Base" "Smart Fortwo Coupe" "Mini 1.6 170" "Nissan Micra  
1.2 65" ...  
  $ : chr [1:6] "Comp.1" "Comp.2" "Comp.3" "Comp.4" ...  
$ call      : language princomp(x = auto, cor = TRUE)  
- attr(*, "class")= chr "princomp"
```

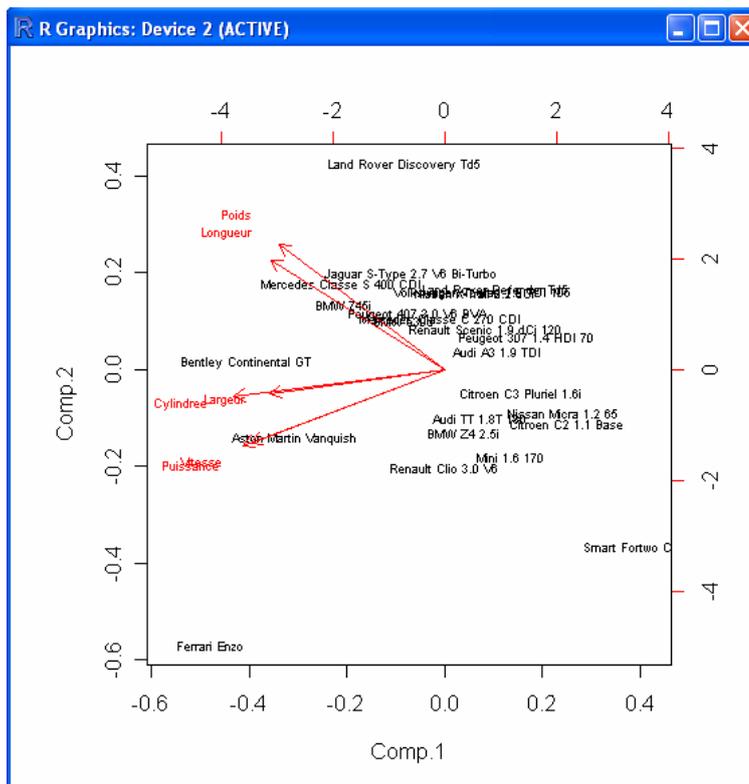
Par exemple, pour accéder aux composantes, on utilise l'instruction suivante :

```
auto.pca$scores
```

	Comp.1	Comp.2	Comp.3	Comp.4	Comp.5	Comp.6
Citroen C2 1.1 Base	2.595915393	-0.50997398	0.17914010	-0.16570416	0.207127333	0.031611419
Smart Fortwo Coupe	4.150149792	-1.66590562	-0.27433215	0.92410926	0.029269629	-0.033602334
Mini 1.6 170	1.381906978	-0.81571698	-0.37235974	0.05104958	-0.464039063	0.050992642
....						
Nissan X-Trail 2.2 dCi	0.614388049	0.72225714	0.04553763	-0.11663312	0.174152022	0.118669211

On peut utiliser la fonction biplot pour obtenir la représentation des variables et des observations dans un plan factoriel. Il s'agit en fait de la fonction biplot.princomp qui trace le graphique d'objets de classe princomp ou prcomp.

`biplot(auto.pca,cex=0.6)` représente les données dans le premier plan factoriel :



Exercices

Exercice 1 - Modèle linéaire

Dans le package datasets est disponible le jeu de données cars. Ce jeu de données est composé de deux variables la première indiquant la vitesse de voitures et la seconde le temps de freinage.

- 1.1. Tracer le graphe bivarié de la deuxième colonne sur la première. Que constate t'on ?
- 1.2. Construire un modèle linéaire reliant la vitesse au temps de freinage.
- 1.3. Ajouter au graphe bivarié construit à la question 1.1 la droite de régression résultant du modèle de la question 1.2.
- 1.4. Donner les valeurs prédites par le modèle pour chacune des observations.
- 1.5. Construire des intervalles de prédiction et de confiance.

Exercice 2 - Régression logistique

Vous disposez du jeu de données bordeaux_R.txt, composé des 5 colonnes suivantes :

- TEMPERAT = somme des températures moyennes journalières (°C)
- SOLEIL = Durée d'insolation (h)
- CHALEUR = Nombre de jours de grande chaleur
- PLUIE = hauteur de pluie en (mm)
- QUALITÉ : 1 = Bon, 2 = Moyen et 3 = Médiocre

- 2.1. Importer bordeaux_R.txt dans la data.frame nommé A.
- 2.2. Tracer tous les graphes bivariés et colorer les points en fonction de leur qualité.
- 2.3. Construire un modèle logistique ordonné à l'aide de la fonction polr disponible sur le package MASS.
- 2.4. Accéder aux probabilités d'appartenance de chaque observation aux différentes classes.
- 2.5. Donner la précision du modèle en termes de pourcentage de bonne classification.

Exercice 3 - Analyse en composante principale (ACP)

Pour illustrer l'analyse en composante principale, nous allons, une nouvelle fois, utiliser le jeu de données bordeaux_R.txt

- 3.1. Charger le jeu de données bordeaux_R en mémoire.
- 3.2. Construire une analyse en composante principale (à partir des 4 variables température, soleil, chaleur, pluie) sur la matrice de corrélation
- 3.3. Afficher sur un même graphique les variables et les individus (biplot) sur les deux premiers axes principaux.
- 3.4. Afficher la qualité des vins en labels des individus.
- 3.5. Interpréter le résultat de l'analyse.

Exercice 4 - Clustering : les Kmeans

- 4.1. Récupérer les deux premiers scores de l'analyse en composante principale de l'exercice précédent et les stocker dans un objet nommé X.
- 4.2. Tracer le graphe bivarié du premier score sur le deuxième et colorer les points selon leur qualité.
- 4.3. Construire une partition de 3 groupes de ce nuage de points via l'algorithme des kmeans.

- 4.4. Calculer le taux de bonne classification et comparer à la régression logistique.
- 4.5. Lier chacune des observations à son centroïde par un segment.

Corrigé

Correction de l'exercice 1

1.1. `plot(cars$dist, cars$speed)`

On constate une relation linéaire entre la distance de freinage et la vitesse du véhicule. Comme on pouvait s'y attendre, plus le véhicule est rapide, plus le temps d'arrêt est important.

1.2. `result.lm = lm(cars$speed ~ cars$dist)`

1.3. `abline(result.lm)`

1.4. `Yhat = predict(result.lm)`

1.5. intervalle de prediction :

`predict(result.lm, as.data.frame(cars$dist), interval="prediction")`

intervalle de confiance :

`predict(result.lm, as.data.frame(cars$dist), interval="confidence")`

Correction de l'exercice 2

2.1. `A = read.table("/bordeaux_R.txt", header = TRUE, sep = "\t")`

2.2. `plot(A[,2:5], col = A$QUALITE)`

2.3. `res.log = polr(as.factor(A$QUALITE) ~ A$TEMPERAT + A$SOLEIL + A$CHALEUR + A$PLUIE)`

2.4. `result.logistic[4]`

2.5. `Yhat = predict(res.log)`

`performance = A$QUALITE - as.numeric(Yhat)`

`length(performance[performance == 0])`

`accuracy = length(performance[performance == 0])/nrow(A)`

`accuracy`

Correction de l'exercice 3

3.1. `A = read.table("/bordeaux_R.txt", header = TRUE, sep = "\t")`

3.2. `result.pca = princomp(A[,2:5], cor = TRUE)`

3.3. `biplot(result.pca)`

3.4. `nom = c("bon", "moyen", "medicore")[as.numeric(A$QUALITE)];`

`biplot(result.pca, xlab = nom)`

3.5. On remarque qu'une année peu pluvieuse jumelée à de fortes chaleurs fournira des vins de bonne qualité.

Correction de l'exercice 4

```
4.1. A = read.table("/bordeaux_R.txt", header = TRUE, sep = "\t")
      result.pca = princomp(A[,2:5], cor = TRUE)
      X = result.pca[[6]][,1:2]
```

```
4.2. plot(X, col = A$QUALITE)
```

```
4.3. result.kmeans = kmeans(X, 3)
```

```
4.4. Yhat = result.kmeans$cluster
```

```
      Yobs = A$QUALITE
```

```
      perf = Yhat - Yobs
```

```
      accuracy = length(perf[perf == 0])/nrow(A)
```

Le clustering fournit des résultats moins bons que la régression logistique.

Ceci peut s'expliquer par deux points :

- le pourcentage de variance capturée par l'ACP n'est pas complète
- les kmeans partitionne l'espace sans se servir de la variable à expliquer (ici la qualité) tandis que la régression logistique utilise cette information pour construire son modèle.

```
4.5. plot(X, col = A$QUALITE,
```

```
       xlab = "premiere composante principale",
```

```
       ylab = "deuxieme composante principale ",
```

```
       main = "Les vins de bordeaux")
```

```
      points(result.kmeans$centers, col = 1:3, pch = 7, lwd = 3)
```

```
      segments(X[result.kmeans$cluster == 1, ][, 1],
               X[result.kmeans$cluster == 1, ][, 2],
               result.kmeans$centers[1, 1],
               result.kmeans$centers[1, 2],
               col = 1
              )
```

```
      segments(X[result.kmeans$cluster == 2, ][, 1],
               X[result.kmeans$cluster == 2, ][, 2],
               result.kmeans$centers[2, 1],
               result.kmeans$centers[2, 2],
               col = 2
              )
```

```
      segments(X[result.kmeans$cluster == 3, ][, 1],
               X[result.kmeans$cluster == 3, ][, 2],
               result.kmeans$centers[3, 1],
               result.kmeans$centers[3, 2],
               col = 2
```

```
      )
```