R-bases : Les notions de base de R
André Berchtold, 2009-2014

Obtenir R : http://www.r-project.org/
Obtenir RStudio : http://www.rstudio.com/

Principes de base

- R est un logiciel travaillant principalement à l'aide de lignes de commande. Il existe bien une interface graphique (Rcmdr), mais elle est limitée aux fonctions statistiques de base.
- Il est recommandé, en plus de R, d'installer aussi RStudio. RStudio est un environnement de travail qui permet d'utiliser R de manière beaucoup plus efficace.
- Pour utiliser R correctement, il faut écrire systématiquement toutes les commandes dans des fichiers de syntaxe (scripts) cela permet de conserver une trace de tout ce qui a été fait et de reproduire aussi souvent que nécessaire les analyses.
- Toutes les commandes R se terminent par des parenthèses (). Les paramètres des commandes (s'il y en a) doivent être placés entre ces commandes.
- R peut faire beaucoup de choses, mais créer une base de données n'est pas son fort. Pour cela, il est préférable d'utiliser d'autres moyens (Excel, SPSS, fichier texte, …).

Opérations de base

- Obtenir de l'aide : help()
- Savoir quel est le répertoire de travail actuel : getwd()
- Changer de répertoire de travail : setwd() : *setwd("c:/temp")*
- Liste des fichiers du répertoire de travail : dir()
- Liste des objets en mémoire : ls()
- Trouver le type d'un objet : is(obj)
- Charger un package supplémentaire : library()
- Rendre utilisable dans R une fonction ne faisant pas partie d'un package : *source("fonction.r ")*
- Charger un fichier au format R : load() : *load("Data_exemple.rda")*
- Sauvegarder un fichier au format R : save() : *save(Dataset,file="essai.rda")*
- Rendre les variables d'un fichier de données accessibles sans avoir à donner à chaque fois le nom du fichier (déconseillé !): attach() : *attach(Dataset)*
- Supprimer l'accessibilité directe des variables d'un fichier : detach() : *detach(Dataset)*
- Voir le code source d'une fonction : *fonction*
- Modifier le format d'affichage des résultats numériques : options(digits=7)

Manipulation de variables

- Assigner une valeur à une variable : <- (ou =) : *age_m <- mean(age)*
- Supprimer une variable de l'espace de travail : remove() : *remove(age)*
- Visualiser un jeu de données : View() : *View(Dataset)*
- Editer un jeu de données : fix() : *fix(Dataset)*
- Changer le type d'un objet : as.matrix(), as.factor(), as.numeric(), as.data.frame(), …
- Savoir si un objet est d'un certain type : is.matrix(), is.factor(), is.numeric(), is.data.frame(), …
- Accéder à la *i*-ième observation de la *j*-ième variable d'un jeu de données : *Dataset[i :j]*

- Accéder à une variable d'un jeu de données : *Dataset$Q12*
- Créer une liste : *c(), c(1,4,5), c('Olivier','Anne','Jacques','Valérie')*
- Créer une variable de longueur *n* remplie d'une valeur *x* : *rep(x,n) : rep(0, 10)*
- Voir les noms des variables d'un dataframe : *names(Dataset)*
- Renommer une variable : *names(Dataset)[3] <- c("Faculté")*
- Renommer plusieurs variables *: names(Temp)[c(1,2,3)] <- c("Votants","Vote", "Oubli")*
- Supprimer une variable d'un dataframe *D: D$Var <- NULL*
- Créer un facteur : *factor()* : *factor(c('A','B','C')), factor(c('A','A','A'),levels=c('A','B'))*
- Afficher les niveaux d'un facteur : *levels(facteur)*
- Renommer les niveaux d'un facteur : *levels(facteur) <- c("jeune","moyen","vieux")*
- Regrouper les niveaux d'un facteur : *levels(facteur) <- c("jeune","jeune","vieux")*
  Ou encore : *levels(facteur) <- c(1,1,2)*
- Réordonner les niveaux d'un facteur :
  facteur <- factor(facteur, levels=c("vieux","moyen","jeune"))
- Supprimer les niveaux inutiles d'un facteur : `X <- factor(X,exclude=NULL)`
- Assigner les niveaux d'un facteur A à un facteur B : `levels(B) <- levels(A)`
- Remplacer une valeur (ici : 20 par 50): `Age <- replace(Age,Age==20,50)`
- Déclarer des données manquantes : `Age <- replace(Age,Age==20,NA)`
- Utilisation de *recode* (librairie *car*) :
  - Attribuer une valeur aux NA : `Age <- recode(Age,"NA=-99")`
  - Recoder une variable numérique : `Age <- recode(Age,"1=17;2=22;3=27")`
  - Déclarer des données manquantes *: Dataset$Q6e <- recode(Dataset$Q6e, '-1=NA ')*
    Dataset$Q6f <- recode(Dataset$Q6f, '-1=NA ', as.factor.result=TRUE)
    Dataset$Q6f <- recode(Dataset$Q6f, '1="Non"; 2="???"; 3="Oui"; ', as.factor.result=TRUE)

Opérations sur les données

- « ou » logique : |
- « et » logique : &
- « égalité » logique : ==
- « différent de » : !=
- Recherche des NA (retourne un vecteur logique) : is.na()
- Tri par ordre croissant : sort() : *sort(Q1)*
- Tri par ordre décroissant : rev(sort())
- Recherche des emplacements des éléments pour lesquels une certaine condition est vraie : which() : *which(Q1<=4)*
- Création d'une table de fréquence pour une variable de n'importe quel type : table()
- Sélection d'un sous-ensemble d'une variable ou d'un dataframe en fonction d'une condition : *subset(Q1,Q15=="Femme").*
- Autre possibilité : *dataframe[dataframe$Q15=="Femme",]*.
- Suppression des observations comportant des données manquantes : na.omit() : *mean(na.omit(Q13))*
- Application d'une commande com sur une variable x en fonction des groupes définis par une variable y : by(x,y,com) : *by(Q13,Q15,max)*
- Repérer les cas sans données manquantes : *complete.cases()*

Calculs

- Racine carrée : sqrt() : *sqrt(9)*
- Puissance : ** ou ^ : *3**2 ou 3^2*
- Modulo : %%

- Division entière : %/%
- Arrondir x à k décimales : round(x,k) : *round(mean(na.omit(Q13)),3)*
- Minimum et maximum : min(), max()
- Exponentielle : exp()
- Logarithme naturel de x : log(x) ; logarithme de x en base k : log(x,k)
- Somme des éléments d'une variable : sum()

Calculs statistiques de base

- Résumé d'un jeu de données ou d'une variable: summary() : *summary(dataset)*
- Moyenne : mean() ; médiane : median() ; variance : var() ; écart-type : sd() ; covariance : cov() ; corrélation : cor()
  mean(age) ; mean(age,na.rm=TRUE) ;
  cov(Q11,Q12) ; cov(Q11,Q12,use="complete.obs")
  cor(Q11,Q12,use="complete.obs")
  cor(Q11,Q12,use="complete.obs", method="spearman")
- Régression linéaire: lm() ou glm() :
  summary(lm(Q12~Q2+Q14e))
  summary(glm(Q12~Q2+Q14e))
- Régression logistique binaire: glm() :
  summary(glm(Q20~Q2+Q21+Q14e,binomial))
- L'option *weights* de *glm* permet de calculer des modèles de régression sur des données pondérées:
  summary(glm(Q20~Q2+Q21+Q14e,binomial,weights=ponder))
- Supprimer la constante d'une régression : *summary(glm(Q12~+0+Q2+Q14e))*
- Imposer une catégorie comme référence : *relevel(Q12,2)*

Tables de contingence

- Créer une table de contingence : *table(var1,var2)* ou *xtabs(~var1+var2)*
  Il est possible d'utiliser plus que 2 variables.
- Test du chi-2 : *summary*(*table(var1,var2))* ou summary(*xtabs(~var1+var2))*
  Il est possible d'utiliser plus que 2 variables.
- Créer un objet table à partie de variables figurant dans un dataframe et d'une variable représentant les effectifs : *MyTable <- xtabs(Unil$effectif~Unil$Sexe+Unil$Année)*

Graphiques

- Boxplot : boxplot()
- Graphique pour une variable x : plot(x)
- Graphique de dispersion pour deux variables x et y : plot(x,y)
- Idem, mais avec des points proportionnels aux fréquences : sunflowerplot(x,y)
- Histogramme pour une variable continue x : hist(x)

Formules

Une formule est une équation de la forme

$$VD \sim VI1+VI2+VI3+\ldots$$

où VD est la variable dépendante et VIx est la x-ième variable indépendante. Dans la majorité des cas, une constante est automatiquement ajoutée au modèle. Pour supprimer cette constante, il faut inclure **0** dans la liste des variables explicatives :

$$VD \sim 0+VI1+VI2+VI3+\ldots$$

Il est possible d'inclure des interactions entre VI avec en utilisant les opérateurs ***** et **:**. L'opérateur * permet d'inclure une interaction ainsi que tous les effets d'ordres inférieurs à cette interaction. Par exemple,

$$VD \sim VI1*VI2$$

signifie en réalité

$$VD \sim VI1+VI2+VI1:VI2$$

Le modèle comporte donc les effets propres des variables *VI1* et *VI2*, ainsi que l'interaction *VI1 :VI2*. L'opérateur **:** permet de spécifier uniquement une interaction. Par exemple,

$$VD \sim VI1*VI2$$

ne rajoute que l'interaction des deux variables et non les effets d'ordres inférieurs.

Les interactions, qu'elles soient spécifiées par * ou : peuvent concerner un nombre quelconque de variables simultanément :

$$VD \sim VI1*VI2*VI3*VI4$$

Il est possible de supprimer un terme, une interaction ou un groupe d'interactions en utilisant le préfixe **–** au lieu de **+** devant ces éléments. Par exemple,

$$VD \sim VI1*VI2*VI3 - VI1:VI2 - VI1$$

spécifie le modèle suivant :

$$VD \sim VI2+VI3+VI1:VI3+VI2:VI3+VI1:VI2:VI3$$

Il est aussi possible de créer des interactions entre des groupes de variables. Par exemple,

$$VD \sim (VI1+VI2+VI3+VI4)*(VI1+VI2+VI3+VI4)$$

générera toutes les interactions d'ordre 2 entre les 4 variables, plus leurs effets propres. Il est équivalent aussi d'écrire

$$VD \sim (VI1+VI2+VI3+VI4)^2$$

Finalement, toutes les possibilités évoquées précédemment peuvent être combinées ! Par exemple

$$VD \sim 0+(VI1+VI2+VI3+VI4)^3 - (VI1+VI2+VI3+VI4)^2 + VI1+VI2 + VI1:VI2$$

Table de survie par la méthode actuarielle

On utilise la fonction *lifetab()* du package *KMsurv* :

lifetab(tis, ninit, nlost, nevent)

Cette fonction retourne une table de survie contenant notamment la probabilité de survie au début de chaque intervalle et le risque instantané, ainsi que les erreurs standard de ces quantités.

Arguments

tis Vecteur définissant les limites des différents intervalles de temps. Le premier élément est le début du premier intervalle, …, et le dernier élément est la fin du dernier intervalle. Il y a un élément de plus que le nombre d'intervalles.

ninit Le nombre de personnes étudiées au début du premier intervalle.

nlost Un vecteur indiquant le nombre de données censurées au sein de chaque intervalle. Sa longueur correspond au nombre d'intervalles.

nevent Un vecteur indiquant le nombre de personnes subissant l'événement étudié au sein de chaque intervalle. Sa longueur correspond au nombre d'intervalles.

Exemple pour les données Yamaguchi

```
tis <- c(0,12,24,36,48)
ninit <- 265
nlost <- c(0,0,3,155)
nevent <- c(59,23,18,7)

yamaguchi <- lifetab(tis, ninit, nlost, nevent)
```

Affichage de la courbe de survie

Il est possible de représenter simplement les probabilités de survie avec la commande

plot(yamaguchi$surv)

mais un graphique de meilleure qualité, avec des segments de droites correspondant à chaque intervalle, est obtenu par exemple de la manière suivante :

```
zy <- kronecker(yamaguchi $surv,c(1,1))
zx <- c(tis[1],kronecker(tis[2:4],c(1,1)),tis[5])
win.graph(width=8, height=6)
plot(zx,zy,cex=1.3,cex.axis=1.3,cex.lab=1.3,col="blue2",lwd=2,type="l",
     main="Fonction de survie",
     xlab="Temps en mois",
     ylab="Survie cumulée")
```

Analyse en temps continu : package *survival*

Le package *survival* est destiné aux analyses en temps continu. Il permet notamment de construire une courbe de survie en utilisant les méthodes de Kaplan-Meier et de Nelson-Aalen, de tester l'égalité de plusieurs courbes et d'estimer le modèle de Cox.

Pour utiliser les fonctions de ce package, il faut commencer par construire un *objet* à l'aide de la fonction

Surv(time, event)

Cet objet servira de variable dépendante pour les différents modèles calculés.

Arguments

time Vecteur indiquant le temps de survenance de chaque événement ou censure.

event Vecteur indiquant pour chaque valeur du vecteur *time* s'il s'agit d'une censure (codée 0) ou d'un événement (1).

Dans le cas où l'on travaille avec des variables évoluant dans le temps et un fichier de données au format personnes-périodes, l'objet de survie doit être construit avec la syntaxe suivante :

Surv(tstart, tstop, event)

Arguments

tstart Vecteur indiquant le moment de début de chaque épisode.

tstop Vecteur indiquant le moment de fin de chaque épisode.

event Vecteur indiquant pour chaque valeur du vecteur *time* s'il s'agit d'une censure (codée 0) ou d'un événement (1).

Table de survie en temps continu

La commande *survfit()* permet de calculer la table de survie correspondant soit à une formule, soit à un modèle de Cox. Par défaut la méthode utilisée est celle de Kaplan-Meier, mais l'argument *type="fleming-harrington"* permet d'utiliser la méthode de Nelson-Aalen.

Exemples

Construction de l'objet Survie

Survie <- Surv(time, event)

Construction de la table de survie par la méthode de Kaplan-Meier :

Table_KM <- Survfit(Survie~1)

Construction de la table de survie par la méthode de Nelson-Aalen :

Table_KM <- Survfit(Survie~1, type="fleming-harrington")

Construction de tables de survies différentes pour chaque niveau d'un facteur « fact » :

Table_KMfact <- Survfit(Survie~strata(fact))

Test de l'égalité de plusieurs courbes de survie

La commande *survdiff()* permet de tester l'égalité de plusieurs courbes de survie calculée pour un objet *Survie* en fonction des niveaux d'un facteur *Fact* :

survdiff(Survie~Fact)

Par défaut, c'est le log-rank test qui est appliqué, mais l'argument *rho=1* permet d'effectuer le test de Peto & Peto (une modification du test de Gehan-Wilcoxon) :

survdiff(Survie~Fact,rho=1)

Calcul d'un modèle de Cox

La fonction *coxph()* du package *survival* permet l'estimation d'un modèle de Cox. Sa syntaxe de base est

resultat <- coxph(formule)

où la *formule* est de la forme

survie~v1+v2+…

survie est un objet construit avec la commande *Surv()* (même objet que pour construire une courbe de Kaplan-Meier) et *v1, v2, …* sont les variables explicatives.

resultat permet d'obtenir les principaux résultats, mais *summary(resultat)* donne des informations supplémentaires.

Pour stratifier par rapport à une ou plusieurs variables, on utilise la commande *strata()* dans la formule de construction du modèle :

coxph(survie~v1+v2+strat(v3)+strata(v4)+ …)

La commande *anova* permet de comparer plusieurs modèles au moyen du test du rapport de vraisemblance :

anova(Cox_null,Cox_2,Cox_1)

La fonction *stepAIC* du package *MASS* permet d'effectuer une sélection automatique des variables explicatives en fonction du critère AIC :

stepAIC(Cox_1)

La commande *residuals()* permet d'obtenir les résidus de Schoenfeld :

Cox_1_res <- residuals(Cox_1, "scaledsch")

Le test de ces mêmes résidus s'effectue avec

cox.zph(Cox_1)

Graphiques

La commande *survfit()* permet de créer les courbes de survie correspondant à un modèle de Cox :

SurvCox1 <- survfit(Cox_1)

On affiche les courbes obtenues avec la fonction plot :

plot(SurvCox1)

Pour afficher les courbes LML plutôt que les courbes de survie d'un modèle de Cox stratifié, on rajoute l'argument *fun="cloglog"* :

plot(SurvCox1, fun="cloglog")

Pour représenter graphiquement les résidus de Schoenfeld, on utilise la commande

plot(cox.zph(Cox_1))

Lorsqu'il y a plusieurs covariables et donc plusieurs résidus, ceux de la *i-ième* covariable sont représentés graphiquement par

plot(cox.zph(Cox_1)[i])

Création d'un fichier personnes-périodes

La fonction **reshape** permet de transformer un fichier de données d'une représentation en une autre.

Exemple : On part du fichier suivant contenant 3 individus (a,b,c) et 4 variables (x1 à x4):

```
tmp <- data.frame(id=letters[1:3], x1=1:3, x2=4:6, x3=7:9, x4=10:12)
tmp
  id x1 x2 x3 x4
1  a  1  4  7 10
2  b  2  5  8 11
3  c  3  6  9 12
```

1) On considère que x1, x2, x3, x4 sont 4 observations successives de la même variable :

```
Ex1 <- reshape(tmp, direction="long", varying=list(names(tmp)[-1]))
Ex1
    id time x1
a.1  a    1  1
b.1  b    1  2
c.1  c    1  3
a.2  a    2  4
b.2  b    2  5
c.2  c    2  6
a.3  a    3  7
b.3  b    3  8
c.3  c    3  9
a.4  a    4 10
b.4  b    4 11
c.4  c    4 12
```

Pour remettre les données par ordre d'individus plutôt que par ordre de temps, on commence par créer un vecteur triant la colonne id (colonne 1 du fichier) par ordre :

```
ordre <- sort.list(Ex1[,1])
ordre
[1]  1  4  7 10  2  5  8 11  3  6  9 12
```

Ensuite, on réordonne les lignes du fichier selon ce vecteur :

```
Ex1 <- Ex1[ordre,]
Ex1
    id time x1
a.1  a    1  1
a.2  a    2  4
a.3  a    3  7
a.4  a    4 10
b.1  b    1  2
b.2  b    2  5
b.3  b    3  8
b.4  b    4 11
```

```
c.1  c   1  3
c.2  c   2  6
c.3  c   3  9
c.4  c   4 12
```

2) On considère que x1 et x2 sont deux observations successives de la même variable, alors que x3 et x4 sont deux variables fixes dans le temps :

```
v1=c("x1","x2")
Ex2 <- reshape(tmp, direction="long", varying=list(v1))
Ex2
   id x3 x4 time x1
a.1  a  7 10    1  1
b.1  b  8 11    1  2
c.1  c  9 12    1  3
a.2  a  7 10    2  4
b.2  b  8 11    2  5
c.2  c  9 12    2  6
```

3) On considère que x1 et x2 d'une part, x3 et x4 de l'autre sont deux observations successives d'une même variable :

```
v1=c("x1","x2")
v2=c("x3","x4")
Ex3 <- reshape(tmp, direction="long", varying=list(v1,v2))
Ex3
   id time x1 x3
a.1  a    1  1  7
b.1  b    1  2  8
c.1  c    1  3  9
a.2  a    2  4 10
b.2  b    2  5 11
c.2  c    2  6 12
```