Analyse des Données

Travaux Pratiques 0

Introduction

La maison-page du logiciel se trouve à l'adresse **www.r-project.org**. Toutes les réponses aux questions relatives à la bonne exécution de ce TP (commandes, manipulation d'objets, ...) peuvent être trouvées en utilisant l'aide en ligne de R. Une bonne base pour débuter est de commencer par taper help.start() et de suivre le lien **An Introduction to R** ou de consulter les documentations en français disponibles sur **http ://cran.r-project.org/other-docs.html**.

Ce TP a pour but de vous familiariser avec les commandes de base de R. En fin de TP, vous serez capable de lire, écrire et afficher des données sous R.

Ce TP est séparé en trois parties, la première vous donnant les concepts fondammentaux du langage R, la seconde mettant en application ces principes sur un exemple et la troisième, plus exploratoire, vous permettra d'effectuer votre première "analyse de données".

Pour bien commencer :

1. Placer les fichiers de données dont vous aurez besoin pour ce TP dans un répertoire que vous appellerez 'mes_donnees',
2. lancer le logiciel R,
3. positionner le 'working directory' (dans le menu 'Tools' pour un mac et menu 'File ->change dir' pour windows) sur votre répertoire 'mes_donnees'.

1 Principes de base de R

Dans cette section, en partant de la documentation indiquée ci-dessus, nous décrivons quelques principes de base de **R**. Le logiciel **R** est un langage qui est *orienté-objet* et qui est *interprété*. Il fournit, via une interface utilisateur simple (cf. le menu *File, Edit, Packages*) et surtout via une fenêtre principale de commandes, un langage et un environnement de programmation essentiellement dédié à l'analyse statistique. Les commandes sont entrées depuis la fenêtre principale (appelée aussi *console*). L'assignation d'une valeur à un objet s'effectue à l'aide de l'opérateur `<-` et selon la forme générale :

```
> objet <- valeur(s)
```

Dans cette section, nous présentons les principales structures de données, c.-à-d. les *vecteurs*, les *listes*, les *tableaux de données*, ... Pour finir, nous présentons la notion de *fonction*.

1.1 Vecteur

Le type de données de base dans **R** est le *vecteur*. Un vecteur désigne une collection ordonnée d'éléments de même nature (appelé *mode* dans **R**). Selon son mode, un vecteur est *numérique*, *logique* ou *alphanumérique*.
On peut créer soi-même un vecteur en utilisant la fonction concaténation *c(element 1, element 2, ...)*. Les nombres décimaux doivent comporter un point décimal, les chaines de caractères être entourées de guillemets doubles `‘‘ ’’`, et les valeurs logiques être codées par TRUE et FALSE, soit T et F en abrégé. Une donnée manquante est codée par la valeur NA.

Exemple.
Entrer la commande suivante :

```
> AGE <- c(34.5, 23, 18, 42.5, 75, 7)
```

et vérifier son résultat avec la commande qui permet d'afficher `AGE` :

```
> AGE
```

Le résultat doit être :

```
[1] 34.5  23  18  42.5  75  7
```

Le chiffre `1` entre crochets indique le numéro d'index du premier élément affiché sur la ligne.

Les vecteurs peuvent être utilisés tels quels dans des expressions arithmétiques classiques. *Les opérations sont effectués élément par élément.* Lorsque les vecteurs intervenant dans l'expression n'ont pas la même longueur, la longueur du vecteur résultat est celle du plus long vecteur de l'expression. Dans ce cas, les valeurs des vecteurs plus courts sont 'recyclées', éventuellement partiellement, de manière à atteindre cette longueur.

Exemple.

```
> TEMPS <- c(1, 0.5)
> x <- AGE * TEMPS ; y <- AGE + TEMPS
> x
> y
> x2 <- TEMPS * AGE ; x2
```

Dans le même esprit, une expression logique peut être appliquée à un vecteur : le résultat est alors un vecteur logique contenant le résultat de l'expression appliquée sur chaque élément du vecteur de départ.

Exemple.

```
> AGE > 20
```

L'extraction d'éléments d'un vecteur s'effectue via un vecteur d'index placé entre crochet. Un vecteur d'index peut être de quatre types distincts : *vecteur logique, vecteur d'entiers positifs, vecteur d'entiers négatifs, vecteur de chaines de caractères.*

Extraction par vecteur logique. Le vecteur logique et le vecteur indexé sont de même longueur. Les éléments extraits sont ceux qui génèrent une valeur TRUE par application du vecteur logique, et le vecteur résultat a donc pour longueur le nombre déléments ayant une valeur TRUE.

Exemple.

```
> AGE[AGE > 20]
```

Extraction par vecteur d'entiers positifs. Le vecteur d'index, *i.e.* ici le vecteur d'entiers positifs, est de longueur quelconque. Chaque élément dont la position correspond à la valeur mentionnée dans le vecteur d'index est extrait. Le résultat a donc pour longueur celle du vecteur d'index.

Exemple.

```
> AGE[c(3, 5, 6)]
```

Extraction par vecteur d'entiers negatifs. Chaque élément dont la position correspond à la valeur absolue du nombre mentionnée dans le vecteur d'index est exclus de la selection.

Exemple.

```
> AGE[-c(3, 5, 6)]
```

Extraction par vecteur de chaines de caractères. Chaque élément du vecteur indexé peut être préalablement nommé explicitement par la fonction `names()`. Les éléments dont le nom correspond aux chaines de caractères du vecteur d'index, sont alors sélectionné et accompagnés de leur nom.

Exemple.

```
> names(AGE) <- c(‘‘Fred’’, ‘‘Paul’’, ‘‘Marie’’, ‘‘Jo’’, ‘‘Yves’’, ‘‘Jack’’)
> AGE ; AGE[c(‘‘Paul’’, ‘‘Marie’’)]
```

1.2 Autres structures : listes, tableaux de données et matrices

1.2.1 Listes

Une liste (`list`) est une collection ordonnée d'objets, non nécessairement du même mode. Les éléments d'une liste peuvent être n'importe quel objet défini dans **R**. Cette propriété

est utilisée par certaines fonctions pour renvoyer des résultats complexes sous la forme d'un unique objet liste.
La constitution d'une liste s'effectue par la fonction `list` : *list(nom1 = élément1, nom2 = élément2, ...)*, l'utilisation des noms étant facultative. On peut accéder à chaque élément de la liste par son index numérique entre double crochets [[...]], ou par son nom précédé du signe $.

Exemple.

```
> liste <- list(ages=AGE, agesplus=y, agenoms=c(``Paul'', ``Marie''))
> # extraction du deuxieme element de la liste par index
> liste[[2]]
> # extraction du deuxieme element de la liste par nom
> liste$agesplus
> # extraction d'un sous-element de agesplus par index
> liste[[2]][1]
```

Il est possible de concaténer plusieurs listes en une seule au moyen de la fonction *c(liste1, liste2, ...)*, comme pour les vecteurs : l'utilisation de la fonction *list(liste1, liste2, ...)* conduirait à l'obtention d'une liste de listes ...

1.2.2 Tableaux de données

Un tableau de données (`data.frame`) est une classe particulière de listes, qui a pour but de stocker des données qui sont à analyser. Chaque composant (le plus souvent un vecteur) de la liste est l'équivalent d'une colonne ou variable, tandis que les différents éléments de ces composants correspondent aux lignes ou individus.
On peut appliquer à cette classe d'objets des méthodes spécifiques destinées à faciliter la visualisation et l'analyse de leur contenu, comme par exemple les méthodes `plot()`, `summary()`, ...

Un tableau de données peut être créé par rassemblement de ses composants individuels via la fonction *data.frame(nom1 = élément1, nom2 = élément2, ...)*, qui est l'analogue de *list*, ou via la conversion d'un objet existant par *as.data()*, ou enfin via la fonction *read.table* qui permet la lecture des données dans un fichier externe. Les vecteurs inclus dans un data.frame doivent être de même longueur, ou si un des éléments est plus court il doit pouvoir être 'recyclé' *uniquement* un nombre entier de fois.

Exemple. Vérifier que, selon la règle indiquée ci-dessus, seule la dernière des trois instructions utilisant *data.frame* est incorrecte.

```
> x <- 1:6 ; y <- 10 ; z <- c(20,30) ; w <- 1:4
> data.frame(x,y)
> data.frame(x,z)
> data.frame(x,w)
```

1.2.3 Matrice

Une matrice est en fait un vecteur qui possède un argument supplémentaire (*dim*) qui est lui-même un vecteur numérique de longueur 2 et qui définit les nombres de lignes et de colonnes de la matrice. Une matrice peut être créée avec la fonction *matrix* dont les paramètres par défaut sont :

```
matrix(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
```

L'option *byrow* indique si les valeurs données par *data* doivent remplir successivement les colonnes (option par défaut) ou les lignes (si TRUE). L'option *dimnames* permet de donner des noms aux et aux colonnes.

Exemple.

```
> matrix(1:6, 2, 3)
> matrix(1:6, 2, 3, byrow = TRUE)
> # on peut aussi utiliser l'attribut 'dim' pour transformer un
> # vecteur en matrice.
> x <- 1:6 ; dim(x) <- c(2,3) ; x
```

1.3 Fonctions

Les fonctions de **R** sont considérées comme des objets élémentaires. Elles peuvent contenir une suite d'instructions, y compris des appels à d'autres fonctions.

1.3.1 Syntaxe

La syntaxe d'appel d'une fonction est simple : *nom_fonction(argument1 = valeur1, argument2 = valeur2, ...)*. Chaque argument peut être défini de deux façons : par son nom, ou par sa position dans la liste des arguments.

Exemple.

```
> HAUTEUR <- c(185, 170, 172, 180, 168, 175)
> plot(x=AGE, y=HAUTEUR, type=''p'')
> plot(x=AGE, y=HAUTEUR, type=''l'')
> plot(AGE, HAUTEUR, type=''p'')
```

La documentation de **R** sur chaque fonction et ses paramètres est disponible via la commande *help(nom_fonction)*. On peut aussi afficher des exemples d'utilisation d'une fonction de **R** grâce à *example(nom_fonction)*.

1.3.2 Définition d'une fonction

Le logiciel **R** permet de définir des fonctions personnalisées à l'aide de l'expression suivante :

nom_fonction <- *function(arg1[=expr1], arg2[expr2], ...)* { *bloc d'instructions* }.

Les accolades indiquent le début et la fin du code source de la fonction en cours de définition. Les crochets ne sont pas nécessaires : ils indiquent la valeur par défaut d'un argument facultatif.

Pour définir une fonction, on peut également utiliser la commande *fix(nom_fonction)* qui permet d'écrire le code source de la fonction via un éditeur de texte (par défaut, le Bloc notes). L'instruction *nom_fonction2* <- *edit(nom_fonction1)* permet à la fois de renommer et de modifier la définition d'une fonction.

Les valeurs initiales des arguments d'une fonction ne sont pas modifiées par les expressions contenues dans la fonction.

Exemple.

```
> x <- 3
> carré <- function(x) {x <- x*x ; return(x)}
> carré(x)
> x
```

Exercice. Définir une fonction *fact(n)* qui pour chaque entier positif retourne la valeur $n! = n \times (n-1) \times ... \times 2 \times 1$. On pourra pour cela utiliser une boucle *for* dont la syntaxe est : *for(i in 1 : n)* { *instructions* }.

2 Mise en application des principes de base

Cette partie vous permettra de lire un fichier, manipuler les données, créer des scripts et fabriquer de petites fonctions. Bref, que du bonheur.

Toutes les instructions à faire vous sont données, mais sont bien évidemment présentes dans l'aide de R. Ceci vous permettra d'avoir un petit pense-bête à portée de main.

On utilisera pour cette partie les données du tableau "titanic.dat" rendant compte pour chaque passager de :

- son type de billet (de la première à la troisième classe, une valeur 0 étant destinée aux membres d'équipage),
- son age (1=adulte, 0=enfant),
- son sexe (1=homme, 0=femme, 2=indéterminé),
- son trépas (0=a succombé, 1=a survécu)

2.1 Lecture de fichier

2.1.1 On commence.

Placer les données du fichier **titanic.dat** dans la variable 'titanic'.

Élements de solution :

```
# les champs sont séparés par des virgules, on utilise donc read.csv()
# header=TRUE est important (cf. doc)
```

2.1.2 Qu'a-t-on récupéré ?

Ceci fait, afficher la variable "titanic (elle doit posséder 2201 lignes) ainsi que ses attributs.

2.1.3 De plus en plus fort

Afficher uniquement le sexe des passagers, puis uniquement les caractéristiques du passager numéro 1974.

2.2 Modification de variable

Après de longues analyses, la personne ayant un sexe indéterminé s'est révélée être un homme.

2.2.1 On vérifie qu'on sait faire

Trouver le passager ayant un sexe indéterminé et afficher ses caractéristiques.

Élements de Solution :

```
#utiliser une boucle for et la commande print
```

2.2.2 Lever le doute

Corriger l'indétermination.

2.2.3 Sauvegarde de la langue française

Traduiser les noms des différentes colonnes en Français.

Élements de solution :

```
#names(titanic) donne les noms des colonnes sous forme de vecteur
```

2.2.4 Attention, ça va aller très vite

Pour finir, un petit jeu.
écrire en une seule ligne une commande R qui remplace le nom de chaque ligne par "P_x" où x est le numéro du passager (*i.e.* le numéro de ligne).

Élements de solution :

```
#row.names(titanic) rend le vecteur des noms des lignes
#utiliser la commande paste pour aller plus vite :)
```

2.3 Fichier de script

2.3.1 American history R

Créer un fichier contenant toutes les commandes précédentes, appeler le 'tp_1.R'. Effacer ensuite les objets créés et lancer le script pour tout retrouver.

Solution :

```
#on considere que le script s'appelle "Robert.R"
# on pourra utiliser la commande history() pour le creer
#
# lire la doc de rm() et ls()
# ici on efface tout
rm(list=ls())
source("Robert.R")
```

2.3.2 Une petite fonction pour la route

Écrire une fonction qui rend les statistiques descriptives d'un tableau donné en entrée, et écrit un petit mot gentil à l'utilisateur.

Élements de solution :

```
#utiliser la commande summary pour avoir les statistiques descriptives
```

3 Mais où est donc Leonardo ?

3.1 "les femmes et les enfants d'abord"

En utilisant la fonction *table()*, vérifier que l'adage "les femmes et les enfants d'abord" est vérifié.

3.1.1 Pourcentages

1. Calculer le nombre total de survivants, le nombre de femmes (non enfants) et le nombre d'enfants surviants.
2. Déduire les pourcentages de femmes (non enfants) et d'enfants parmi les survivants.
3. Caluler le nombre de femmes (non enfants) et le nombre d'enfants parmi les passagers.

4. Calculer le pourcentage de femmes (non enfant) survivantes parmi les toutes les femmes (non enfants), le pourcentage d'enfant survivants parmi tous les enfants, et enfin le pourcentage de femmes et d'enfants survivants parmi toutes les femmes et enfants.

3.1.2 La Classe, c'est classe

La classe a-t-elle une importance dans la survie du passager ? Pour résoudre cette question épineuse, reprendre les questions précédentes en séparant les différentes classes.

3.2 Graphiques

Les différentes fonctions utiles pour cette partie sont :
- plot() : pour dessiner des trucs,
- points() : pour rajouter des choses à un graphique.

3.2.1 Tracés

Pour les deux graphiques suivants, on supposera que l'axe des abcisses représente les différentes classes.
- superposer sur un même graphique les courbes du nombre total de passager, du nombre total de femmes et d'enfants, et du nombre d'enfants
- superposer sur un même graphique les courbes du nombre total de passager, du nombre total de survivants, du nombre de femmes et d'enfants survivants et le nombre d'enfants survivants.

3.2.2 Question bonus

représenter les deux graphiques ci-avant dans une même fenêtre. On utilisera pour cela la fonction *par()*.