Aide à l'utilisation de **R** pour l'analyse répétitive des données groupées

Marie Laure Delignette-Muller

16 janvier 2012

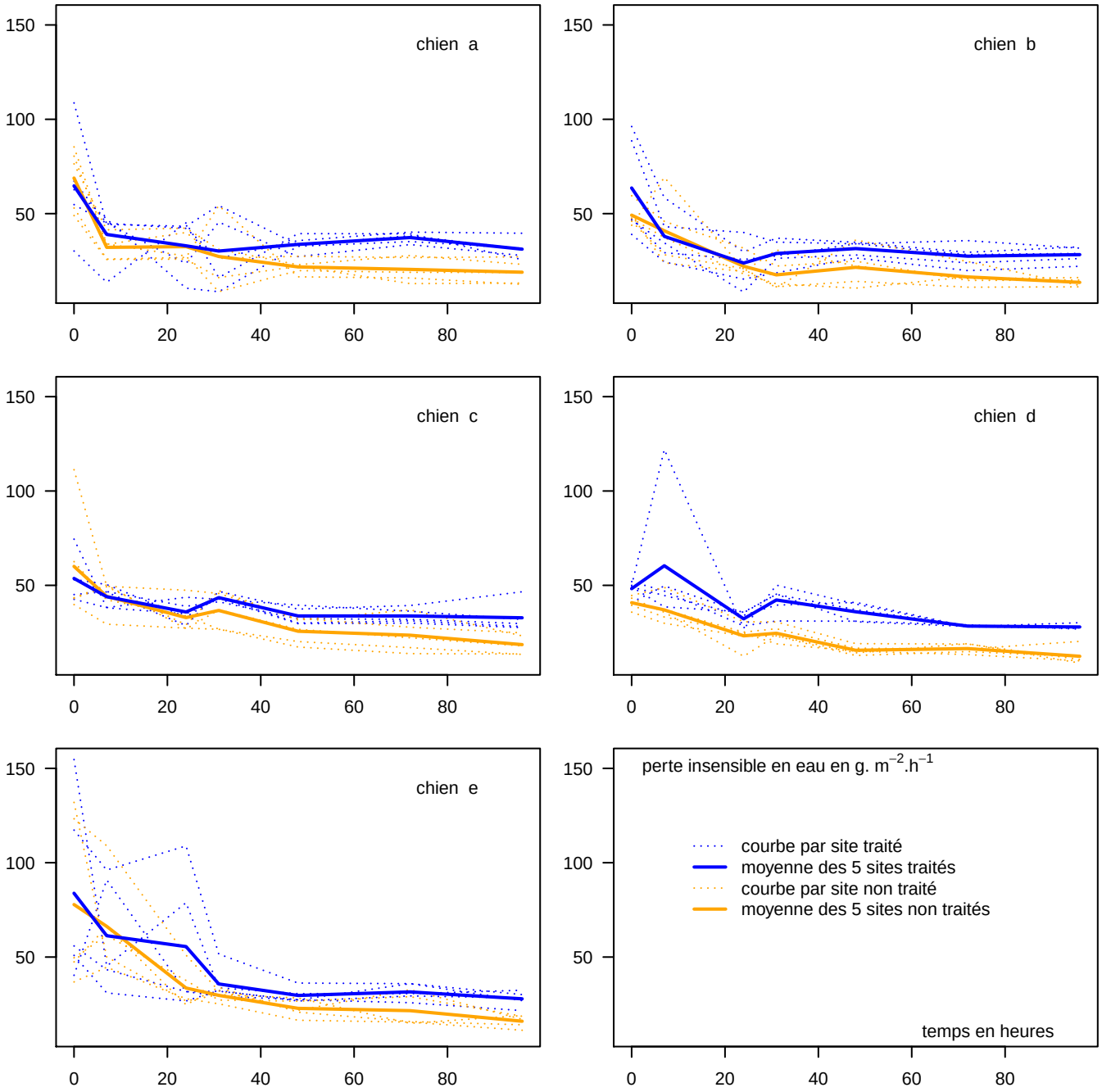


Table des matières

1 Introduction

1.1 Objectifs de ce document

L'objectif de ce document est de vous présenter les principales fonctions **R** qui vous aideront à manipuler et à analyser les données groupées de façon rapide et efficace. L'aide en ligne vous permettra ensuite d'aller plus loin. Rappelons que vous pouvez en savoir plus sur une fonction en tapant le code suivant : `help(nomdelafonction)` ou de façon équivalente `?nomdelafonction`. Une aide en ligne est aussi disponible en tapant le code : `help.start()`. Par ailleurs, vous pourrez trouver des documents permettant d'aller plus loin dans la rubrique " Documentation " du site du projet **R** à l'adresse `http://www.r-project.org/`.

*Ce document représente un complément au document " Aide à l'utilisation de **R** pour réaliser les tests paramétriques et non paramétriques ". Il suppose donc connues les notions décrites dans le premier document.*

1.2 Les différents objets R couramment manipulés

- **les vecteurs** : `vector`
  Un vecteur est un objet unidimensionnel ordonné, composé d'un ensemble de valeurs de même type appelées composantes du vecteur. En statistique un vecteur correspond souvent à une variable. Par défaut les vecteurs numériques sont considérés comme des variables quantitatives et les vecteurs de chaînes de caractères comme des variables qualitatives ou facteurs. Si une variable qualitative est codée avec des nombres, il faut appliquer au vecteur correspondant la fonction `as.factor` pour qu'elle soit bien considérée comme un facteur. Voici un exemple de définition de vecteur avec la fonction classique `c` :

```
> vec1 <- c(1, 2, 4, 6, 12)
> vec1
[1]  1  2  4  6 12
```

  On accède à la ième composante d'un vecteur en indiquant l'indice i entre crochets.

```
> vec1[3]
[1] 4
```

  Il est aussi possible de créer des vecteurs automatiquement en utilisant d'autres fonctions, comme `seq(from,to,by)` qui permet de créer une série arithmétique, et `rep(x,times)` qui permet de créer un vecteur avec une même valeurs `x` répétée `times` fois. Voici un exemple de création de vecteur utilisant ces 3 fonctions :

```
> vec2 <- c(rep(1, 3), seq(2, 5, 1), 9, 11)
> vec2
[1]  1  1  1  2  3  4  5  9 11
```

- **les matrices** : `matrix`
  Une matrice est un objet bidimensionnel ordonné sur les deux dimensions, composé aussi d'un ensemble de valeurs de même type appelées composantes de la matrice. En statistique, l'ensemble des effectifs d'une table de contingence peut par exemple être saisi dans **R** sous forme d'une matrice. Voici un exemple de création d'une matrice avec la fonction `matrix` :

```
> mat1 <- matrix(c(1, 2, 3, 4, 5, 6), nrow = 2, byrow = TRUE)
> mat1
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
```

  On accède à la composante sur la ligne i et la colonne j en indiquant l'indice de ligne suivi de l'indice de colonne entre crochets, séparés par une virgule. On peut accéder à une seule ligne ou une seule colonne en omettant un des indices mais en conservant bien la virgule (cf. les deux exemples ci-dessous).

```
> mat1[2, 2]
[1] 5
> mat1[, 2]
[1] 2 5
```

- **les listes** : `list`
  Une liste est un objet hétérogène. Contrairement aux deux précédents, il s'agit d'un ensemble ordonné d'objets nommés qui n'ont pas forcément le même type, ni la même longueur. On accède au ième objet de la liste soit en indiquant l'indice i entre doubles crochets, soit en indiquant le nom de l'objet après le nom de la liste, séparés par le caractère $ (cf. exemple suivant). On peut créer une liste à l'aide la fonction `list`.

```
> lis1 <- list(toto = c(1, 2), titi = matrix(c(1, 2, 6, 9), nrow = 2,
+     byrow = TRUE), tata = c("A", "G", "H"))
> lis1[[1]]
```

```
[1] 1 2
> lis1$tata
[1] "A" "G" "H"
```

– **les tableaux de données** : `data.frame`
L'objet le plus couramment utilisé pour représenter un tableau de données est l'objet de type `data.frame`. Il s'agit d'une liste particulière, au sens où elle est constituée de vecteurs qui ont tous la même longueur mais peuvent être de types différents, suivant qu'ils correspondent à des variable quantitatives ou qualitatives. Le jeu de données peut être importé à l'aide de la fonction `read.table` (cf. document " Aide à l'utilisation de **R** pour réaliser les tests paramétriques et non paramétriques ") et doit avoir le format suivant : **une colonne par variable, et une ligne par individu, animal ou plus largement unité d'observation**. Les données manquantes doivent obligtoirement être codées par la chaîne de caractères `"NA"`. On peut aussi créer un jeu de données directement dans **R** à l'aide de la fonction `data.frame` mais cela n'est pas recommandé pour une saisie initiale d'un gros jeu de données.

```
> dat1 <- data.frame(temps = c(0, 10, 30, 40), mesure = c(102,
+     107, 168, 289))
> dat1
  temps mesure
1     0    102
2    10    107
3    30    168
4    40    289
```

2 Manipulation de données

Voici le code permettant d'importer les deux fichiers de données que nous utiliserons tout au long de ce document (prenez garde à bien indiquer comme répertoire courant de **R** celui contenant ces fichiers pour que ces commandes marchent).

```
> d1 <- read.table("fumierI.txt", header = TRUE)
> d2 <- read.table("fumierII.txt", header = TRUE)
```

2.1 Exploration rapide d'un tableau de données

Il est prudent, avant de travailler sur un tableau de données, de l'explorer rapidement, afin de vérifier notamment qu'il n'y a pas d'erreurs grossières de saisie et que les variables qualitatives sont bien considérées come des facteurs. Les fonctions `str`, `head`, et `summary` appliquables à un tableau de données sont à ce titre bien utiles. Regardez ce qu'elles donnent sur les deux tableaux de données sur lesquels nous allons travailler.

```
> str(d1)

'data.frame':        816 obs. of  6 variables:
 $ Lieu       : Factor w/ 30 levels "B11","B12","B13",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ Temps      : int  0 1 2 3 4 6 6 7 8 9 ...
 $ Temperature: num  19.8 23 20 17 24 28.5 34 23.5 23.5 23 ...
 $ pH         : num  9.47 NA NA 10.02 NA ...
 $ TMS        : int  24 NA NA 28 NA NA NA 20 NA NA ...
 $ log10N     : num  6.61 NA 5.58 NA NA ...

> str(d2)

'data.frame':        30 obs. of  4 variables:
 $ Lieu    : Factor w/ 30 levels "B11","B12","B13",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ Tas     : Factor w/ 5 levels "T","T1","T2",..: 2 2 2 2 2 2 3 3 3 3 ...
 $ Site    : Factor w/ 6 levels "A","B","C","D",..: 1 2 3 4 5 6 1 2 3 4 ...
 $ Aeration: Factor w/ 2 levels "A","SA": 1 1 1 1 1 1 1 1 1 1 ...

> head(d1)

  Lieu Temps Temperature    pH TMS log10N
1  B11     0        19.8  9.47  24   6.61
2  B11     1        23.0    NA  NA     NA
3  B11     2        20.0    NA  NA   5.58
4  B11     3        17.0 10.02  28     NA
5  B11     4        24.0    NA  NA     NA
6  B11     6        28.5    NA  NA   3.98
```

```
> head(d2)
```

```
  Lieu Tas Site Aeration
1  B11  T1    A        A
2  B12  T1    B        A
3  B13  T1    C        A
4  B14  T1    D        A
5  B15  T1    E        A
6  B16  T1    F        A
```

```
> summary(d1)
```

```
      Lieu          Temps        Temperature          pH              TMS        
 B11    : 29   Min.   : 0.0   Min.   :11.0   Min.   :  8.02   Min.   : 12.0  
 B12    : 29   1st Qu.: 7.0   1st Qu.:23.2   1st Qu.:  9.13   1st Qu.: 18.0  
 B13    : 29   Median :14.0   Median :31.5   Median :  9.29   Median : 20.0  
 B14    : 29   Mean   :14.9   Mean   :34.3   Mean   :  9.30   Mean   : 21.4  
 B15    : 29   3rd Qu.:23.0   3rd Qu.:44.0   3rd Qu.:  9.45   3rd Qu.: 22.0  
 B16    : 29   Max.   :31.0   Max.   :68.0   Max.   : 10.02   Max.   : 78.0  
 (Other):642                                  NA's   :546.00   NA's   :516.0  
     log10N        
 Min.   :  0.000  
 1st Qu.:  0.000  
 Median :  0.699  
 Mean   :  1.269  
 3rd Qu.:  1.708  
 Max.   :  7.514  
 NA's   :480.000  
```

```
> summary(d2)
```

```
      Lieu    Tas    Site  Aeration
 B11    : 1   T :6   A:5   A :12   
 B12    : 1   T1:6   B:5   SA:18   
 B13    : 1   T2:6   C:5           
 B14    : 1   T3:6   D:5           
 B15    : 1   T4:6   E:5           
 B16    : 1          F:5           
 (Other):24                        
```

2.2 Fusion de deux tableaux de données suivant une clef

La fonction `merge` est bien utile pour alléger la saisie des données. Elle permet de fusionner deux tableaux suivant une clef, ce qui évite d'avoir à remplir, dans le fichier importé, de très nombreuses lignes avec la même valeur. Vous comprendrez mieux sur l'exemple ci-dessous. Diverses mesures (dénombrement microbien d'un pathogène, température, pH...) ont été effectuées au cours du temps sur différents lieux de différents tas de fumier, certains tas étant régulièrement aérés et d'autres non. Le tableau `d1` comporte toutes les mesures effectuées au cours du temps dans les différents lieux. Le tableau `d2` indique pour chacun des lieux, à quel tas le lieu correspond, s'il s'agit d'un tas aéré ou non et à quel site (= emplacement dans le tas) le lieu correspond. Dans le code suivant, la fonction `merge` permet d'obtenir à partir de ces deux tableaux un seul grand tableau avec toutes les colonnes des deux tableaux, la fusion étant réalisée à partir de la colonne `Lieu`

```
> d <- merge(d1, d2, by = "Lieu")
> head(d)
```

```
  Lieu Temps Temperature    pH TMS log10N Tas Site Aeration
1  B11     0        19.8  9.47  24   6.61  T1    A        A
2  B11     1        23.0    NA  NA     NA  T1    A        A
3  B11     2        20.0    NA  NA   5.58  T1    A        A
4  B11     3        17.0 10.02  28     NA  T1    A        A
5  B11     4        24.0    NA  NA     NA  T1    A        A
6  B11     6        28.5    NA  NA   3.98  T1    A        A
```

2.3 Sélection de lignes suivant un critère

La fonction `subset` permet de sélectionner (ou d'éliminer) des lignes dans un tableau de données suivant une variable logique indiquée en deuxième argument de la fonction (cf. document " Aide à l'utilisation de **R** pour réaliser

les tests paramétriques et non paramétriques "). Rappelons juste que les principaux opérateurs logiques dans **R** sont == pour "égal à", != pour "différent de", | pour le "ou" logique et & pour le "et" logique. Dans l'exemple suivant on élimine du tableau `d` les lignes correspondant au tas de modalité `"T"`.

```
> d <- subset(d, Tas != "T")
```

2.4 Partage d'un jeu de données en sous-tableaux de données suivant une variable qualitative

La fonction `split` crée une liste de sous-tableaux de données en partageant le tableau qui lui est donné en premier argument suivant les modalités d'une variable qualitative qui lui est donnée en deuxième argument (cf. exemple ci-dessous).

```
> dparLieu <- split(d, d$Lieu)
> head(dparLieu[[1]])
```

```
  Lieu Temps Temperature    pH TMS log10N Tas Site Aeration
1  B11     0        19.8  9.47  24   6.61  T1    A        A
2  B11     1        23.0    NA  NA     NA  T1    A        A
3  B11     2        20.0    NA  NA   5.58  T1    A        A
4  B11     3        17.0 10.02  28     NA  T1    A        A
5  B11     4        24.0    NA  NA     NA  T1    A        A
6  B11     6        28.5    NA  NA   3.98  T1    A        A
```

```
> head(dparLieu[[2]])
```

```
   Lieu Temps Temperature   pH TMS log10N Tas Site Aeration
30  B12     0        21.5 9.00  24  6.529  T1    B        A
31  B12     1        35.1   NA  NA     NA  T1    B        A
32  B12     2        30.0   NA  NA  4.687  T1    B        A
33  B12     3        25.0 9.59  22     NA  T1    B        A
34  B12     4        35.0   NA  NA     NA  T1    B        A
35  B12     6        44.5   NA  NA  0.699  T1    B        A
```

2.5 Création de nouveaux tableaux par collage de tableaux ou de variables

La fonction `rbind` permet de coller des lignes et donc par exemple de recréer un nouveau tableau à partir de la liste de sous-tableaux précédents en collant par exemple les deux premiers.

```
> dB11B12 <- rbind(dparLieu[[1]], dparLieu[[2]])
```

On peut aussi créer un nouveau tableau en collant des vecteurs colonnes pris par exemple dans un autre tableau à l'aide de la fonction `cbind` même si dans ce cas on utilisera plus souvent la fonction `data.frame`.

```
> dB11.tT <- cbind(dparLieu[[1]]$Temps, dparLieu[[1]]$Temperature)
> head(dB11.tT)
```

```
     [,1] [,2]
[1,]    0 19.8
[2,]    1 23.0
[3,]    2 20.0
[4,]    3 17.0
[5,]    4 24.0
[6,]    6 28.5
```

```
> dB11.tT <- data.frame(t = dparLieu[[1]]$Temps, T = dparLieu[[1]]$Temperature)
> head(dB11.tT)
```

```
  t    T
1 0 19.8
2 1 23.0
3 2 20.0
4 3 17.0
5 4 24.0
6 6 28.5
```

2.6 Travail sur les niveaux des facteurs

La fonction `levels` permet de voir les niveaux (modalités) d'un facteur (variable qualitative) et de les renommer (en les affectant à un vecteur de chaînes de caractères choisi) (cf. exemple suivant).

```
> levels(d$Site)

[1] "A" "B" "C" "D" "E" "F"

> nom_sites <- c("BAS.per", "BAS.int", "BAS.cen", "HAUT.int", "HAUT.cen",
+     "SOMMET")
> levels(d$Site) <- nom_sites
```

Il est aussi possible de changer leur ordre de classement (par défaut ordre alphabétique) en redéfinissant un facteur à partir du premier avec comme argument de la fonction `factor` `levels` défini avec les nivaux souhaités (cf. exemple ci-dessous). Cela est surtout utile lorsque l'on travaille sur une variable qualitative ordinale et que le classement alphabétique des niveaux ne correspond pas à leur classement naturel.

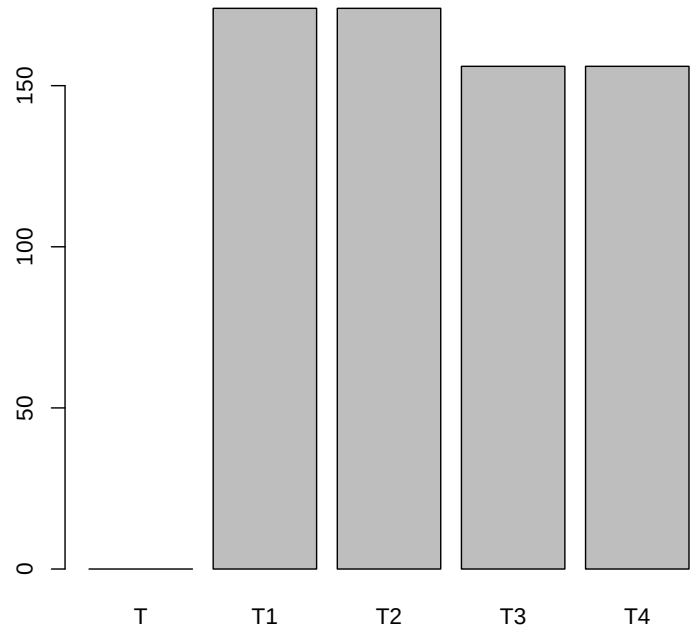
```
> levels(d$Aeration)

[1] "A"  "SA"

> d$Aeration <- factor(d$Aeration, levels = c("SA", "A"))
> levels(d$Aeration)

[1] "SA" "A"
```

Après une sélection de lignes à l'aide de la fonction `subset`, il est souvent nécessaire de redéfinir les facteurs afin de faire disparaître les niveaux qui ne sont plus utilisés du fait de la sélection. Regardons par exemple les niveaux du facteur `Tas` et comparons les aux modalités effectivement observées (accessible à l'aide la fonction `unique`).

```
> levels(d$Tas)

[1] "T"  "T1" "T2" "T3" "T4"

> unique(d$Tas)

[1] T1 T2 T3 T4
Levels: T T1 T2 T3 T4
```

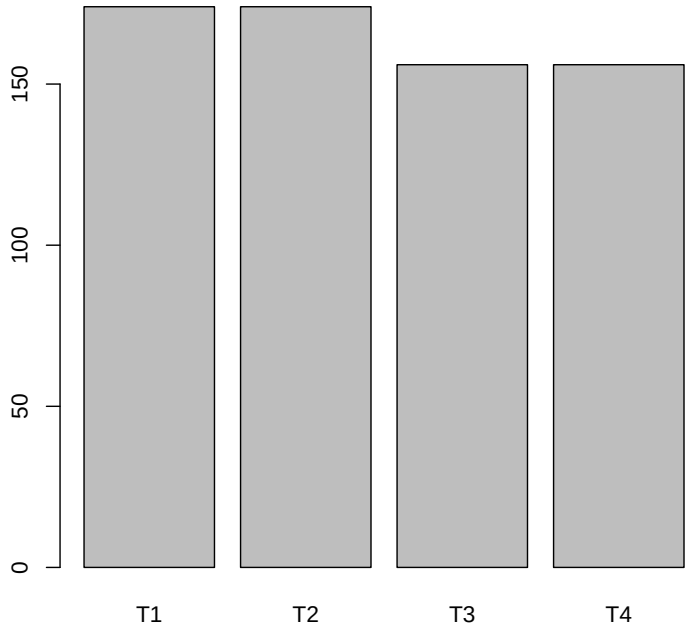
On s'aperçoit que le niveau `"T"` apparaît toujours même si les lignes qui lui correspondaient ont été enlevées du tableau de données. Ceci est gênant par exemple si on veut représenter les fréquences associées aux différentes modalités du facteur (cf. ci-dessous).

```
> barplot(table(d$Tas))
```



Mais on peut remédier à cela facilement en redéfinissant le facteur de la façon suivante (on le transforme en vecteur de chaînes de caractères puis à nouveau en facteur). Nous ferons cette manipulation aussi sur le facteur `Lieu` qui a perdu lui aussi des niveaux du fait de l'élimination du tas "T" du tableau de données.

```
> d$Tas <- factor(as.character(d$Tas))
> barplot(table(d$Tas))
> d$Lieu <- factor(as.character(d$Lieu))
```



Il est parfois utile de créer une variable numérique codant les niveaux d'un facteur de 1 au nombre de niveaux, notamment pour associer ensuite une couleur ou un type de points à chaque niveau du facteur dans une représentation graphique. Ceci peut être réalisé facilement à l'aide de la fonction `match` (cf. exemple ci-dessous).

```
> numsite <- match(d$Site, nom_sites)
> numsite
```

```
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
 [38] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 [75] 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
[112] 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6
[149] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 1 1 1 1 1 1 1 1 1 1
[186] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[223] 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[260] 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5
[297] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6
[334] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[371] 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3
[408] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
[445] 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6
[482] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1
[519] 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
[556] 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4
[593] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
[630] 5 5 5 5 5 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6
```

Il est d'ailleurs possible d'ajouter cette nouvelle variable (ou toute autre variable) au sein du tableau de données en utilisant la fonction `transform` :

```
> d <- transform(d, numsite = match(d$Site, nom_sites))
> head(d)
```

```
  Lieu Temps Temperature    pH TMS log10N Tas    Site Aeration numsite
1  B11     0        19.8  9.47  24   6.61  T1 BAS.per        A       1
2  B11     1        23.0    NA  NA     NA  T1 BAS.per        A       1
3  B11     2        20.0    NA  NA   5.58  T1 BAS.per        A       1
4  B11     3        17.0 10.02  28     NA  T1 BAS.per        A       1
5  B11     4        24.0    NA  NA     NA  T1 BAS.per        A       1
6  B11     6        28.5    NA  NA   3.98  T1 BAS.per        A       1
```

2.7 Tri d'un tableau

Il est possible d'ordonner un tableau suivant une ou plusieurs colonnes en utilisant la fonction `order` . Afin de bien voir comment fonctionne cette fonction nous allons l'appliquer à un sous-tableau `ds` contenant les données pour le temps 2 et les tas aérés. Nous voyons dans le code suivant que la fonction `order` renvoit les numéros des lignes classées dans l'ordre croissant pour la colonne numsite. Si l'on veut réordonner tout le tableau suivant cet ordre, il suffit d'utiliser ce vecteur en indice des lignes (cf. exemple ci-dessous).

```
> ds <- subset(d, Temps == 2 & Aeration == "A")
> ds
```

```
    Lieu Temps Temperature pH TMS log10N Tas     Site Aeration numsite
3    B11     2        20.0 NA  NA   5.58  T1  BAS.per        A       1
32   B12     2        30.0 NA  NA   4.69  T1  BAS.int        A       2
61   B13     2        64.0 NA  NA   2.86  T1  BAS.cen        A       3
90   B14     2        58.0 NA  NA   2.27  T1 HAUT.int        A       4
119  B15     2        63.0 NA  NA   3.04  T1 HAUT.cen        A       5
148  B16     2        58.7 NA  NA   4.86  T1   SOMMET        A       6
177  B21     2        19.0 NA  NA   5.81  T2  BAS.per        A       1
206  B22     2        33.0 NA  NA   5.61  T2  BAS.int        A       2
235  B23     2        53.0 NA  NA   3.79  T2  BAS.cen        A       3
264  B24     2        55.5 NA  NA   3.67  T2 HAUT.int        A       4
293  B25     2        64.0 NA  NA   4.00  T2 HAUT.cen        A       5
322  B26     2        59.5 NA  NA   1.00  T2   SOMMET        A       6
```

```
> os <- order(ds$numsite)
> os
```

```
 [1]  1  7  2  8  3  9  4 10  5 11  6 12
```

```
> dstri <- ds[os, ]
> dstri
```

```
    Lieu Temps Temperature pH TMS log10N Tas     Site Aeration numsite
3    B11     2        20.0 NA  NA   5.58  T1  BAS.per        A       1
177  B21     2        19.0 NA  NA   5.81  T2  BAS.per        A       1
32   B12     2        30.0 NA  NA   4.69  T1  BAS.int        A       2
206  B22     2        33.0 NA  NA   5.61  T2  BAS.int        A       2
61   B13     2        64.0 NA  NA   2.86  T1  BAS.cen        A       3
235  B23     2        53.0 NA  NA   3.79  T2  BAS.cen        A       3
90   B14     2        58.0 NA  NA   2.27  T1 HAUT.int        A       4
264  B24     2        55.5 NA  NA   3.67  T2 HAUT.int        A       4
119  B15     2        63.0 NA  NA   3.04  T1 HAUT.cen        A       5
293  B25     2        64.0 NA  NA   4.00  T2 HAUT.cen        A       5
148  B16     2        58.7 NA  NA   4.86  T1   SOMMET        A       6
322  B26     2        59.5 NA  NA   1.00  T2   SOMMET        A       6
```

2.8 Passage du format "long" au format "large" et réciproquement

Généralement lorsqu'une variable est observée sur plusieurs groupes définis par une variable contrôlée, les données sont codés au format dit "long", c'est-à-dire avec une colonne pour la variable observée et une colonne pour la variable contrôlée. Le format "large" correspond lui au codage de la variable observée en autant de colonnes qu'il y a de groupes (donc de modalités de la variable contrôlée). On peut passer facilement d'un format à l'autre à l'aide des fonctions `stack` et `unstack` (cf. exemple suivant).

```
> dslarge <- unstack(ds, log10N ~ Site)
> dslarge
```

```
  BAS.per BAS.int BAS.cen HAUT.int HAUT.cen SOMMET
1    5.58    4.69    2.86     2.27     3.04   4.86
2    5.81    5.61    3.79     3.67     4.00   1.00
```

```
> dslong <- stack(dslarge)
> colnames(dslong) <- c("log10N", "Site")
> dslong
```

```
   log10N     Site
1    5.58  BAS.per
2    5.81  BAS.per
3    4.69  BAS.int
4    5.61  BAS.int
5    2.86  BAS.cen
6    3.79  BAS.cen
7    2.27 HAUT.int
8    3.67 HAUT.int
9    3.04 HAUT.cen
10   4.00 HAUT.cen
11   4.86   SOMMET
12   1.00   SOMMET
```

3 Fonctions d'aide à l'analyse répétitive des données groupées

Comme avec tout langage de programmation, il est possible d'appliquer de façon répétitive une fonction à l'aide de boucles (instruction `for`) mais en pratique il est plus facile et plus efficace d'utiliser une des fonctions prédéfinies dans **R** que nous allons décrire dans cette partie afin de réaliser ces analyses répétitives.

3.1 Définition d'une fonction

On peut en pratique appliquer de façon répétitive une fonction prédéfinie dans **R** ou définie par l'utilisateur. Voyons donc tout d'abord comment définir une fonction à l'aide de la fonction `function`. Par exemple le code suivant permet de définir la nouvelle fonction de nom `EIQ` avec un seul argument de nom `x`. Entre les accolades ouvrante et fermante est décrit ce que fait la fonction et le résultat qu'elle retourne (ligne `return(resultat)`). Ici la fonction calcule et retourne la différence entre le troisième et le premier quartile de `x`, c'est-à-dire l'écart interquartile.

```
EIQ <- function(x)
{
    Q1 <- quantile(x,probs = 0.25,na.rm=TRUE)
    Q3 <- quantile(x,probs = 0.75,na.rm=TRUE)
    ecart.inter.quartile <- Q3-Q1
    names(ecart.inter.quartile)="eiq"
    return(ecart.inter.quartile)
}
```

Une fois définie, la fonction s'applique comme toute fonction de **R** (cf. exemple).

```
> EIQ(d$pH)
```

```
 eiq
0.34
```

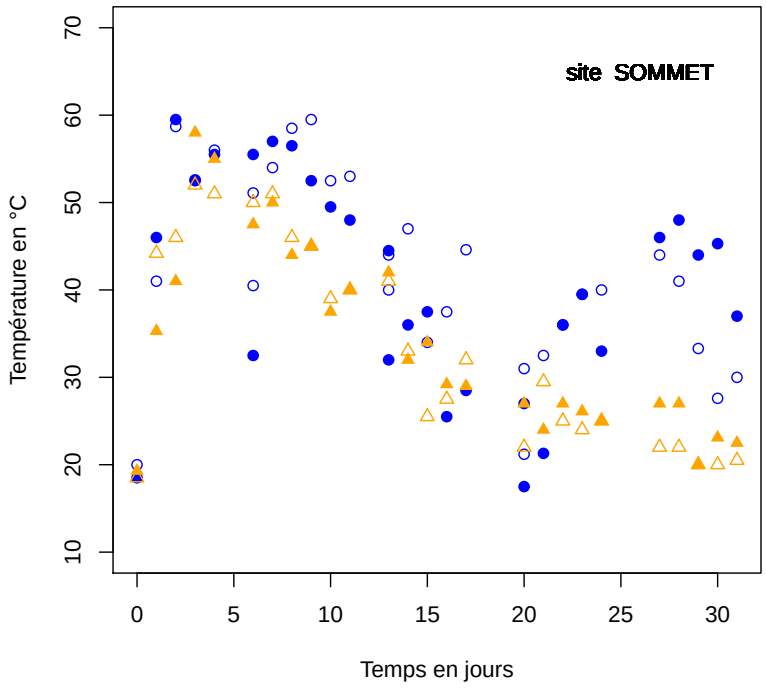
Suivant les besoins on peut définir des fonctions beaucoup plus compliquées, et notamment des fonctions qui réalisent des tracés, comme la fonction suivante qui trace, pour un sous-tableau de données correspondant à un site donné (emplacement dans le tas de fumier), la température en fonction du temps, avec des couleurs différentes pour les tas aérés et non aérés, et des types de points différents pour des lieux différents (répétitions).

```
trace1site <- function(dsite)
{
    lieux <- unique(dsite$Lieu)
    numlieu <- match(dsite$Lieu,lieux)
    aer <- unique(dsite$Aeration)
    numaer <- match(dsite$Aeration,aer)
    couleurs <- c("blue","orange")
    symboles <- c(1,19,2,17)
```

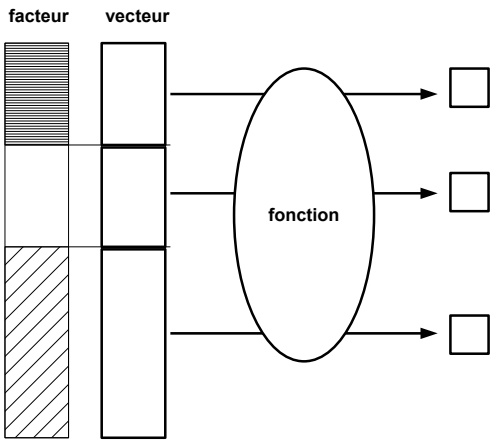
```
    plot(dsite$Temps,dsite$Temperature,
      col=couleurs[numaer],pch=symboles[numlieu],
      ylim=c(10,70),xlab="Temps en jours",ylab="Température en řC")
    text(26,65,paste("site ",dsite$Site))
}
```

Appliquons cette fonction pour le site renommé `"SOMMET"` :

```
> dsite <- subset(d, Site == "SOMMET")
> trace1site(dsite)
```



3.2 Application répétitive d'une fonction à un vecteur divisé par groupes



La fonction `tapply` permet d'appliquer une fonction (passée en troisième argument) à une variable (passée en premier argument) autant de fois qu'il y a de groupes définis par le deuxième argument (qui peut être un facteur ou une liste de facteurs). Dans l'exemple ci-dessous, l'écart inter-quartile du pH est ainsi calculé pour le groupe des tas aérés et le groupe des tas non aérés.

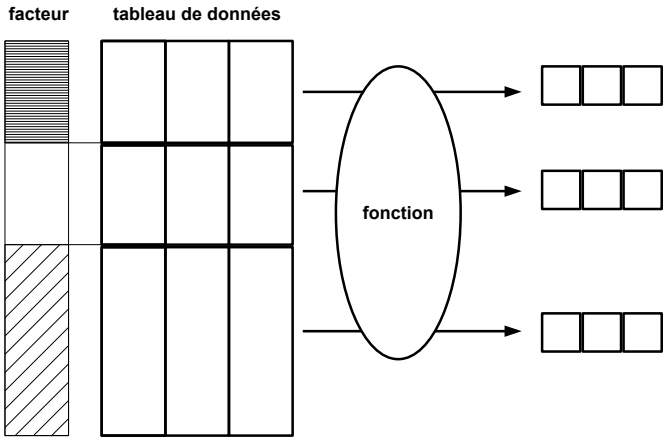
```
> tapply(d$pH, d$Aeration, EIQ)
```

```
  SA    A
0.31 0.28
```

Si besoin on peut ajouter des arguments finaux à `apply` qui seront passés à la fonction spécifiée en troisième argument. Dans l'exemple ci-dessous on ajoute l'argument `na.rm=TRUE` qui permet à la fonction `mean` de calculer la moyenne même en présence de données manquantes, en omettant ces données manquantes.

```
> tapply(d$pH, d$Aeration, mean, na.rm = TRUE)

  SA    A
9.26 9.33
```

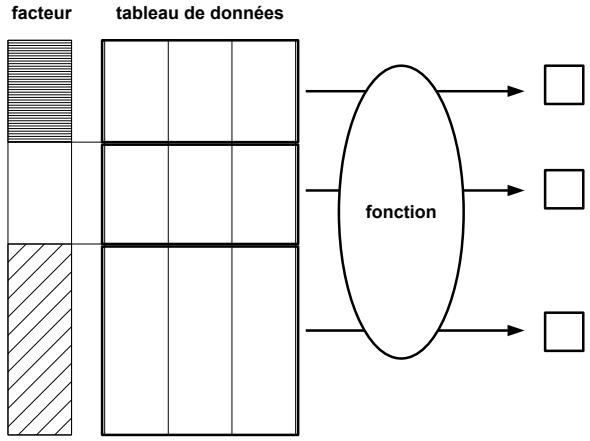
3.3 Application répétitive d'une fonction à chaque vecteur d'un tableau de données divisé par groupes



La fonction `aggregate` permet d'appliquer une fonction (passée en troisième argument) à toutes les variables d'un tableau de données (passé en premier argument) autant de fois qu'il y a de groupes définis par le deuxième argument (qui peut être un facteur ou une liste de facteurs et contrairement au cas précédent doit toujours être défini sous forme de liste même s'il n'y a qu'un facteur dans la liste). Dans l'exemple ci-dessous on calcule la moyenne de chaque colonne du sous-tableau de données comprenant les variables quantitatives, d'abord en séparant par le facteur aération, puis en séparant par le croisement du facteur aération et du facteur site.

```
> d.quant <- data.frame(d$Temperature, d$pH, d$TMS, d$log10N)
> aggregate(d.quant, list(d$Aeration), mean, na.rm = TRUE)

  Group.1 d.Temperature d.pH d.TMS d.log10N
1      SA          32.7 9.26  21.6     1.61
2       A          35.8 9.33  21.6     1.50

> aggregate(d.quant, list(d$Aeration, d$Site), mean, na.rm = TRUE)

   Group.1  Group.2 d.Temperature d.pH d.TMS d.log10N
1       SA  BAS.per          21.4 9.55  31.1    2.236
2        A  BAS.per          21.2 9.60  30.6    2.124
3       SA  BAS.int          27.6 9.29  21.1    1.982
4        A  BAS.int          27.9 9.37  20.9    1.812
5       SA  BAS.cen          35.3 9.12  18.6    1.537
6        A  BAS.cen          38.7 9.30  19.8    1.094
7       SA HAUT.int          37.2 9.21  18.7    1.079
8        A HAUT.int          40.4 9.27  20.0    1.360
9       SA HAUT.cen          41.0 9.15  19.0    0.985
10       A HAUT.cen          45.2 9.29  20.2    1.257
11      SA   SOMMET          33.8 9.25  21.0    1.817
12       A   SOMMET          41.4 9.17  18.1    1.336
```

3.4 Application répétitive d'une fonction à chaque sous-tableau d'un tableau de données divisé par groupes



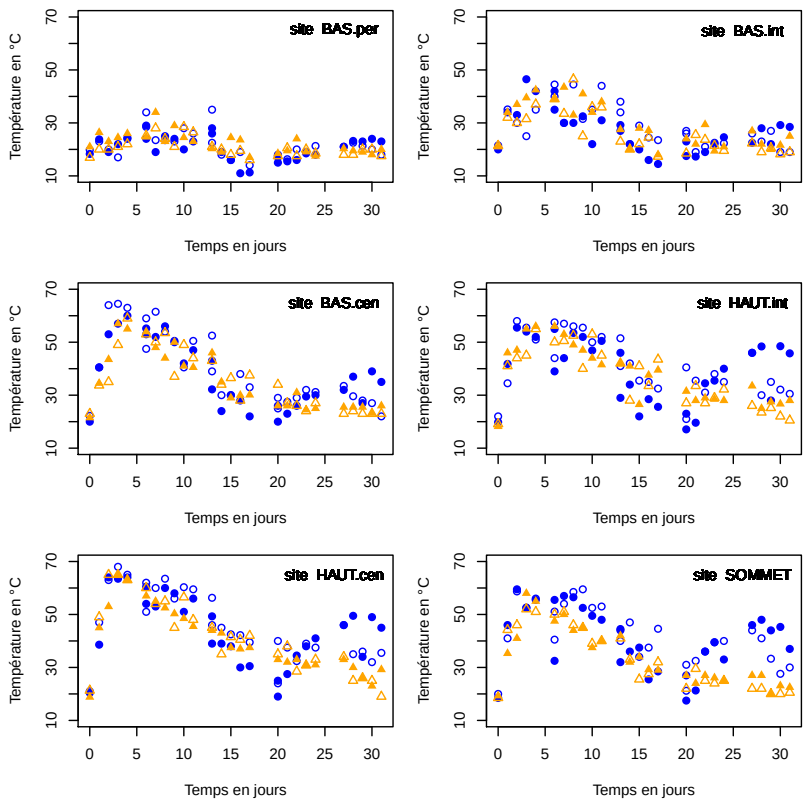
La fonction `by` permet d'appliquer une fonction (passée en troisième argument) à chaque sous-tableau d'un tableau de données (passé en premier argument) autant de fois qu'il y a de groupes définis par le deuxième argument (qui peut être un facteur ou une liste de facteurs). Dans l'exemple ci-dessous on affiche le résumé des variables de `d` en séparant par le facteur aération.

```
> by(d, d$Aeration, summary)
```

```
d$Aeration: SA
      Lieu         Temps       Temperature         pH              TMS
 B31    : 26   Min.   : 0.0   Min.   :16.0   Min.   :  8.70   Min.   : 14.0
 B32    : 26   1st Qu.: 7.0   1st Qu.:23.0   1st Qu.:  9.10   1st Qu.: 18.0
 B33    : 26   Median :14.5   Median :29.0   Median :  9.22   Median : 20.0
 B34    : 26   Mean   :15.0   Mean   :32.7   Mean   :  9.26   Mean   : 21.6
 B35    : 26   3rd Qu.:23.0   3rd Qu.:41.6   3rd Qu.:  9.41   3rd Qu.: 22.0
 B36    : 26   Max.   :31.0   Max.   :65.0   Max.   :  9.98   Max.   : 76.0
 (Other):156                                  NA's   :204.00   NA's   :192.0
     log10N        Tas            Site    Aeration    numsite
 Min.   :  0.000   T1:  0   BAS.per :52   SA:312   Min.   :1.0
 1st Qu.:  0.000   T2:  0   BAS.int :52   A :  0   1st Qu.:2.0
 Median :  0.699   T3:156   BAS.cen :52            Median :3.5
 Mean   :  1.606   T4:156   HAUT.int:52            Mean   :3.5
 3rd Qu.:  2.446            HAUT.cen:52            3rd Qu.:5.0
 Max.   :  7.514            SOMMET  :52            Max.   :6.0
 NA's   :192.000
------------------------------------------------------------
d$Aeration: A
      Lieu         Temps       Temperature         pH              TMS
 B11    : 29   Min.   : 0.0   Min.   :11.0   Min.   :  8.02   Min.   : 14.0
 B12    : 29   1st Qu.: 7.0   1st Qu.:23.7   1st Qu.:  9.20   1st Qu.: 18.0
 B13    : 29   Median :14.0   Median :34.0   Median :  9.35   Median : 20.0
 B14    : 29   Mean   :14.8   Mean   :35.8   Mean   :  9.33   Mean   : 21.6
 B15    : 29   3rd Qu.:22.0   3rd Qu.:46.0   3rd Qu.:  9.47   3rd Qu.: 22.0
 B16    : 29   Max.   :31.0   Max.   :68.0   Max.   : 10.02   Max.   : 78.0
 (Other):174                                  NA's   :240.00   NA's   :228.0
     log10N        Tas            Site    Aeration    numsite
 Min.   :  0.000   T1:174   BAS.per :58   SA:  0   Min.   :1.0
 1st Qu.:  0.000   T2:174   BAS.int :58   A :348   1st Qu.:2.0
 Median :  0.699   T3:  0   BAS.cen :58            Median :3.5
 Mean   :  1.497   T4:  0   HAUT.int:58            Mean   :3.5
 3rd Qu.:  2.199            HAUT.cen:58            3rd Qu.:5.0
 Max.   :  7.086            SOMMET  :58            Max.   :6.0
 NA's   :192.000
```

Maintenant appliquons la fonction graphique `trace1site` que nous avions définie précédemment pour obtenir la figure proposée pour tous les sites, après avoir partagé la fenêtre graphique en 6 figures et redéfinit les marges afin de ne pas laisser trop de place inoccupée.

```
> par(mfrow = c(3, 2))
> par(mar = c(4.5, 4.1, 1.1, 1.1))
> by(d, d$Site, trace1site)
```



Nous finirons par un exemple sans grand intérêt ici mais qui vous montre comment on peut récupérer facilement sous forme de tableau de donnée les résultats d'une analyse statistique faite sur un tableau de données divisé par groupes. Ici on calcule le coefficient de corrélation linéaire entre la température et le pH sur le tableau `d` divisé par lieux.

```
corTpH1lieu <- function(dlieu)
{
    corTpH <- cor(dlieu$Temperature,dlieu$pH,use="complete.obs")
    return(corTpH)
}
```

L'argument `use="complete.obs"` de la fonction `cor` permet de calculer ce coefficient de corrélation uniquement sur les lignes où il n'y a aucune donnée manquante.

```
> corTpHparLieu <- by(d, d$Lieu, corTpH1lieu)
> dcorTpH <- data.frame(Lieu = levels(d$Lieu), corTpH = as.vector(corTpHparLieu))
> head(dcorTpH)

  Lieu  corTpH
1  B11 -0.6561
2  B12  0.0426
3  B13  0.3185
4  B14  0.6397
5  B15  0.4016
6  B16 -0.6569
```

4 Conclusion

Maintenant à vous de jouer sur vos propres données. Et n'hésitez pas à y passer un peu de temps au début pour en gagner beaucoup ensuite. On pense souvent à tort qu'on aura plus vite fait de faire les manipulations de données à l'aide de son tableau et les analyses répétitives en faisant du "copier - coller" de lignes de code. Mais vous verrez que si vous investissez un minimum dans l'apprentissage des fonctions présentées dans ce document, vous serez nettement plus efficaces, vous garderez une trace de vos manipulations que vous pourrez ainsi vérifier et réutiliser, et vous prendrez beaucoup moins de risques d'erreur de manipulation.

Index

Code de réalisation du graphe de la première page de ce document

```
# Importation des données
d <- read.table("tewl.txt",header=TRUE)
str(d)

# Fonction de tracé des mesures au cours du temps pour
# un site d'un chien
traceparsite <- function(dsite)
{
if (dsite$produit[1] == 1)
lines(dsite$temps,dsite$TEWL,col="blue",lty=3)
else
lines(dsite$temps,dsite$TEWL,col="orange",lty=3)

}

# Fonction de calcul de la courbe moyenne des 5 sites d'un chien
# ayant reçu le même traitement (produit ou non)
moyenneparproduit <- function(dproduit)
{
tewlmoy <- tapply(dproduit$TEWL,dproduit$temps,mean)
if (dproduit$produit[1]==1)
lines(unique(dproduit$temps),tewlmoy,lwd=2,col="blue",lty=1)
else
lines(unique(dproduit$temps),tewlmoy,lwd=2,col="orange",lty=1)
}

# Fonction de tracé des courbes par site et des 2 moyennes pour un chien
traceparchien <- function(dchien)
{
plot(0,0,type="n",xlab = "", ylab = "",
xlim = c(min(d$temps),max(d$temps)),
ylim = c(min(d$TEWL),max(d$TEWL)),las=1 )
text(80,140,paste("chien ",dchien$chien[1]))
by(dchien,dchien$site,traceparsite)
by(dchien,dchien$produit,moyenneparproduit)
}

# Tracé des courbes des 5 chiens sur le même graphe
par(mfrow=c(3,2))
par(mar=c(2.5,2.5,1,1))
by(d,d$chien,traceparchien)

# Ajout d'une légende commune
plot(0,0,type="n",xlab = "", ylab = "",
xlim = c(min(d$temps),max(d$temps)),
ylim = c(min(d$TEWL),max(d$TEWL)),las=1 )
text(0,150,expression(paste("perte insensible en eau en g. ",m^-2,".",h^-1)),pos=4)
text(60,10,"temps en heures",pos=4)
legend(10,120,
legend=c("courbe par site traité","moyenne des 5 sites traités",
"courbe par site non traité","moyenne des 5 sites non traités"),
lty=c(3,1,3,1),col=c("blue","blue","orange","orange"),
lwd=c(1,2,1,2),
bty="n")
```