

# Exercices d'initiation à la manipulation de données avec R

Gilles San Martin - Centre Wallon de Recherche Agronomique (CRA-W)

14 November 2015

## Contents

Exercices de prise en main de R . . . . .	2
Exercices de manipulation de formats de données . . . . .	4
Exercice 1 . . . . .	4
Exercice 2 . . . . .	6
Exercices d'importation de données . . . . .	8
Exercices subsidiaires (pas de R) . . . . .	10
Exercices sur l'extraction de données (Subscripting) . . . . .	11
Exercices de manipulation de caractères . . . . .	14
Exercice 1 . . . . .	14
Exercice 2 . . . . .	17
Structures de contrôle : fonctions, boucles, exécutions conditionnelles,... . . . .	18
Exercice 0 . . . . .	18
Exercice 1 . . . . .	19
Exercice 2 . . . . .	20
Exercice 3 . . . . .	22
Exercice 4 . . . . .	22
Exercice 5 . . . . .	23
Agrégation et reshaping . . . . .	25
Exercice 1 . . . . .	25
Exercice 2 . . . . .	26

## Exercices de prise en main de R

Le but de cette série d'exercices est de se familiariser avec l'utilisation basique de R, d'acquérir le réflexe de chercher dans l'aide et de comprendre et utiliser la vectorisation. On vous demandera donc par moment d'utiliser des fonctions qui n'ont pas été vues dans la partie théorique. Pas de panique, cherchez simplement dans l'aide...

- 1) Calculer les racines carrées arrondies à 2 décimales de tous les nombres de 1 à 100 (fonctions `round` et `sqrt`)

```
round(sqrt(1:100), 2)
round((1:100)^0.5, 2) # résultat identique
round(sqrt(seq(1,100,1)), 2) # résultat identique
```

- 2) Calculer la surface des cercles de rayon variant de 0 à 250 cm par pas de 10 cm (26 cercles donc) (fonction `seq`)

```
rayon <- seq(0, 250, 10)
surface <- pi*rayon^2
surface
```

- 3) Ecrivez les phrases suivantes : “Un cercle de 0 cm de rayon a une surface de 0 m<sup>2</sup>”, “Un cercle de 10 cm de rayon a une surface de 0.03 m<sup>2</sup>”, etc... pour tous les cercles de rayon variant de 0 à 250 cm par pas de 10 cm (fonction `paste`)

```
paste("Un cercle de ", rayon, " cm de rayon a une surface de ", round(surface/10000, 2),
      " m2", sep= "")
```

- 4) Tracer un graphique montrant la relation entre le périmètre et la surface d'un cercle. Ajoutez un titre explicite au graphique

```
rayon <- 1:1000
surface <- pi*rayon^2
perimetre <- 2*pi*rayon
plot(surface ~ perimetre, type="l",
      main = "Relation entre la surface et \nle périmètre d'un cercle")
# NB à la place du paramètre graphique "main", on peut aussi utiliser la fonction "title"
```

- 5) Utilisez les 3 vecteurs ci-dessous : “jour”, “mois”, “année” pour créer un vecteur nommé “date” sous la forme 28/9/2012. (fonction `paste`) Essayez de comprendre comment sont construits ces 3 vecteurs (regardez l'aide des fonctions utilisées: `floor`, `runif`, `set.seed`).

```
set.seed(123)
jour <- floor(runif(30, 1, 31))
mois <- floor(runif(30, 1, 12))
année <- floor(runif(30, 1900, 2012))
```

```
date <- paste(jour, mois, année, sep="/")
date
```

```
# On a utilisé la fonction runif qui génère des nombres aléatoire selon une distribution
# uniforme. Pour les 3 vecteurs on demande à la fonction de générer 30 de ces nombres.
# On génère des nombres entre 1 et 30 pour les jours, entre 1 et 12 pour les mois et entre
# 1900 et 2012 pour les années.
# Pour obtenir des nombres entiers (pour éliminer les décimales) on utilise la fonction
# floor qui garde uniquement la partie entière des nombres (sans les arrondir).
# La fonction set.seed sert simplement à ce que les nombres aléatoires générés soient
# toujours les mêmes.
```

- 6) Créez 2 variables aléatoires de distribution normale (fonction `rnorm`) ayant chacune une moyenne différente. Réalisez ensuite un test t de student pour comparer les moyennes de ces deux variables (cherchez dans l'aide la fonction adéquate...).
- Faites un boxplot horizontal de ces deux variables sur le même graphique. Vous devez pour ce faire d'abord les coller côte à côte dans un même objet avec la fonction `cbind` (par exemple `cbind(a,b)`)

```
a <- rnorm(n=30, mean= 10, sd=1)
b <- rnorm(30, 15)
# Taper ??student pour trouver dans l'aide la fonction faisant le test de t
# (ou cherchez sur internet)
t.test(a, b)
# Taper ?boxplot pour voir les différents paramètres possibles pour cette fonction et
# notamment celui qui permet de tracer des boxplots horizontaux.
boxplot(cbind(a,b), horizontal =TRUE)
```

- 7) Créez une variable aléatoire avec une distribution de Poisson (une fonction similaire à `rnorm` et `runif`). Tracez un histogramme de cette variable de couleur rouge.

```
# La seule difficulté ici est de trouver la bonne fonction en utilisant l'aide.
# Voir par exemple l'aide de ?rnorm : dans la section "see also" il y a un lien vers
# toutes les distributions fournies par R
a <- rpois(1000, 1)
hist(a, col = "red")
```

- 8) Voici deux vecteurs représentant le nombre de mâles et de femelles observées dans une population et dans 5 catégories de couleurs : noir, jaune, rouge, bleu, blanc.
- Tracez un barplot représentant côte à côte les valeurs des mâles et des femelles pour chaque couleur (et avec le nom des couleurs dans les étiquettes de l'axe x).
- Pour ce faire commencez par coller les 2 vecteurs de manière à former une matrice à 2 lignes et 5 colonnes (fonction `rbind`). Ajoutez les 5 couleurs dans les noms de colonnes (fonction `colnames`) et ensuite, tracez le graphique.

```
males <- c(3, 5, 7, 9, 10)
femelles <- c(0, 2, 5, 14, 16)

mat <- rbind(males, femelles)
colnames(mat) <- c("noir", "jaune", "rouge", "bleu", "blanc")

barplot(mat, beside = TRUE)
```

## Exercices de manipulation de formats de données

### Exercice 1

L'exercice suivant n'est pas très réaliste. Le but est de vous faire manipuler dans des cas simples les formats de données. Voici 2 vecteurs de 10 nombres a ( 10, 20, 30, etc) et b (5, 5, 5, etc).

```
a <- seq(10,100,10)
b <- rep(c(5,10), c(7,3))
```

- 1) Faites la somme de chaque vecteur (somme a = 550, somme b = 65)

```
sum(a)
sum(b)
```

- 2) Concaténez ces deux vecteurs l'un à la suite de l'autre de façon à obtenir un vecteur de longueur 20 et dont la somme vaut 615

```
c(a,b)
length(c(a,b))
sum(c(a,b))
```

- 3) Collez les nombres de ces vecteurs a et b 2 à 2 (pour obtenir 105, 205, 305 etc) et sauvez le résultat dans un nouveau vecteur "ab". Ce vecteur a une longueur de 10 éléments. Quel est son mode ?

```
ab <- paste(a, b, sep="")
ab
mode(ab)
```

- 4) Faites la somme de ce nouveau vecteur (réponse = 29865).

```
sum(ab) # ne fonctionne pas
sum(as.numeric(ab))
```

- 5) Faites un graphique de a en fonction de b (plot(a~b))

```
plot(a~b)
```

- 6) Transformez le vecteur b en facteur et sauvez ce résultat dans un nouveau vecteur appelé "f"

```
f <- as.factor(b)
```

- 7) Faites un graphique de a en fonction de f

```
plot(a~f)
```

- 8) Faites la somme de f (ça ne fonctionne pas...)

```
sum(f) # ne fonctionne pas
```

- 9) Retransformez f en numérique pour pouvoir en faire la somme (vous devrez passer par un format intermédiaire avant de le transformer en numérique). Vous devriez obtenir le même résultat que sum(b) soit 65 (et pas 13).

```
sum(as.numeric(f)) # ne donne pas la bonne réponse
sum(as.numeric(as.character(f)))
```

- 10) A partir du vecteur a, créez une matrice de 5 colonnes et 2 lignes avec sur la première ligne les valeurs 10, 20, 30, etc... (et pas 10, 30, 50 etc...). Sauvez cette matrice dans un objet appelé “mat”.

```
matrix(a, ncol = 5, nrow=2)
mat <- matrix(a, ncol = 5, nrow=2, byrow=TRUE)
mat
```

- 11) Ajoutez comme nom de colonnes col1, col2, col3 etc... à cette matrice.

```
colnames(mat) <- c("col1", "col2", "col3", "col4", "col5")
colnames(mat) <- paste0("col", 1:ncol(mat)) # même résultat mais plus flexible
```

- 12) Au moyen de l’opérateur \$ essayez d’extraire la colonne 2 (mat\$col2). Que faut-il faire pour que cela fonctionne ? (dans quel format faut-il transformer mat?)

```
mat$col2 # ne fonctionne pas car $ ne fonctionne qu'avec des listes ou des data.frame
mat <- as.data.frame(mat)
mat$col2
```

- 13) Créez un objet à 2 colonnes et 3 lignes contenant dans la première colonne les valeurs A, B et C et dans la deuxième colonne 3, 6 et 9.

```
a <- c("A", "B", "C")
b <- c(3,6,9)
df <- data.frame(a=a, b=b)
df
```

- 14) Faites la somme de la deuxième colonne de l’objet ainsi créé

```
sum(df$b)
```

```
# On aurait pu être tenté de procéder comme suit mais ça ne fonctionne pas car si on
# mélange du texte et des nombres dans une matrice, les nombres sont transformés en texte
mat <- matrix(c("A", "B", "C", 3,6,9), 3, 2)
colnames(mat) <- c("a", "b")
mat <- as.data.frame(mat)
sum(mat$b)
# on peut par contre retransformer cette colonne en numérique à la volée :
sum(as.numeric(mat$b))
```

## Exercice 2

Dans cet exercice plus réaliste, vous devrez à de nombreuses reprises passer d'un format de données à un autre (dates, character, numeric,...) pour pouvoir effectuer les opérations demandées.

Chargez le fichier "biche.txt" avec la commande suivante : `d <- read.table("biche.txt")` après avoir défini votre répertoire de travail où se trouve le fichier "biche.txt" avec la fonction `setwd`, par exemple : `setwd("/home/gilles/exercices")` (NB : vous devez utiliser des slashes "/" et pas de backslashes "\" dans le chemin du répertoire).

Ce fichier contient la date, l'heure et les points gps (coordonnées x y) de position d'une biche. Les colonnes `datet0`, `timet0`, `xt0`, `yt0`, contiennent pour chaque ligne les positions et temps de l'observation (point GPS) précédente. L'objectif général est de déterminer la période de l'année (saison) pour chaque point et de calculer la vitesse de déplacement entre deux points successifs.

- 1) Visualisez le contenu de votre jeu de données avec `summary(d)`, `head(d)` et `str(d)`. Identifiez le type de chaque variable.

```
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
d <- read.table("data/biche.txt")
summary(d)
head(d)
```

- 2) Passez les colonnes `time` et `timet0` en format date-temps (fonction `strptime`). Vous devrez au préalable coller la date et le temps dans une même variable (fonction `paste`). Utilisez la fonction `summary` pour visualiser la transformation du type de données

```
d$time <- paste(d$date, d$time)
d$time <- strptime(d$time, format = "%d/%m/%Y %H:%M:%S")

d$timet0 <- paste(d$datet0, d$timet0)
d$timet0 <- strptime(d$timet0, format = "%d/%m/%Y %H:%M:%S")

summary(d)
```

- 3) Créez une colonne pour l'année et le mois (par exemple avec la fonction `strftime`). Pour rappel, vous pouvez ajouter une colonne "mois" au dataframe "d" par exemple comme ceci : `d$mois <- c(1, 3, 5, 7)` Utilisez la fonction `head` pour visualiser les premières lignes du jeu de données

```
d$year <- as.numeric(strftime(d$time, "%Y"))
d$month <- as.numeric(strftime(d$time, "%m"))
head(d)
```

- 4) Sur base de votre colonne "mois", créez une colonne "saison" divisée en 3 périodes : nourrissage de janvier à avril (inclus), "été" de mai à septembre (inclus) et "chasse" de octobre à décembre. Vous pouvez utiliser la fonction `cut` pour ce faire. Faites un `summary()` de cette nouvelle colonne. Vous devriez avoir 2192 données en période de nourrissage, 969 en été et 1265 en période de chasse.

```
d$season <- cut(d$month, breaks = c(0,4,9,12), labels = c("nourrissage", "été", "chasse"))
summary(d$season)
# NB on aurait pu utiliser ici la fonction cut sur une date au format Date (cfr ?cut.Date)
```

- 5) Calculez le temps écoulé entre 2 points gps successifs (2 lignes successives). Lorsque vous faites la différence entre deux heures (classe `POSIXct`), Vous obtenez une variable au format "difftime". Vous pouvez voir les unités utilisées au moyen de la fonction `units`. Voir l'aide pour `difftime` pour plus d'infos. Les 3 premières valeurs sont : NA, 3.800833 et 7.999167 heures.

```
d$deltatime <- d$time - d$timet0
units(d$deltatime)
d$deltatime <- as.numeric(d$deltatime)/60
head(d$deltatime)
```

- 6) Calculez pareillement la distance parcourue. Les coordonnées x et y sont des lambert belges (coordonnées projetées) en mètres.  
Pour Rappel, le théorème de Pythagore pour un triangle rectangle :  $\text{Hypoténuse}^2 = \text{base}^2 + \text{hauteur}^2$   
Les 3 premières valeurs sont NA, 0.149308406 et 0.286062930 km

```
d$dist <- sqrt((d$x - d$xt0)^2 + (d$y - d$yt0)^2)/1000
head(d$dist)
```

- 7) Calculez la vitesse en km/h entre deux points successifs (attentions aux unités). Quelle est la vitesse maximale ? (réponse : 3.312303 km/h et pas NA ! Regardez dans l'aide en cas de besoin...)

```
d$speed <- d$dist/d$deltatime
max(d$speed, na.rm = TRUE)

summary(d)
head(d)
```

## Exercices d'importation de données

Le but de ces exercices est de se familiariser avec les cas les plus fréquents d'importation de données dans R (données dans un tableur ou dans un fichier texte). Les premiers exercices sont assez simples mais on a progressivement et volontairement introduit de nombreux pièges. L'objectif n'est point de jouir du plaisir sadique de vous voir patagner mais bien de vous confronter aux problèmes fréquents que l'on rencontre pour importer ses données. Quand vous devez importer un fichier texte, ouvrez le d'abord dans un bon éditeur de texte (par exemple avec Rstudio) et regardez comment il est structuré (est-ce qu'il y a des entêtes de colonne, des valeurs manquantes, comment sont séparés les champs, quels sont les séparateur de décimales, etc...).

Pour les fichiers xls, exportez les en format texte en respectant les recommandations vues dans la partie théorique. Après l'importation visualisez systématiquement votre jeu de données avec `summary()` et en imprimant le jeu de données.

Les deux derniers exercices sont subsidiaires et ne sont pas à proprement parler des exercices de R, l'essentiel du travail devant se faire dans le tableur. Ils montrent des fichiers Excel bien structurés et clairs mais assez mal adaptés à l'analyse et à l'importation dans R (ou tout autre logiciel). Le but est de montrer qu'il vaut mieux structurer ses fichiers de données brutes en réfléchissant au traitement des données.

- 1) Importez le fichier `data1.txt`. Calculez la moyenne de la variable "haraxy" (réponse : 5.85 )

```
# Peu de difficultés ici. Il faut bien spécifier na.rm = TRUE dans la fonction mean sans
# quoi elle retourne une valeur manquante NA
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
mydata <- read.table("data/data1.txt", sep=";", header = TRUE)
summary(mydata)
mean(mydata$haraxy, na.rm = TRUE)
```

- 2) Importez le fichier `data2.txt`. Faites la moyenne de la variable "adabip" (=31.77) Utilisez la fonction `plot` pour faire un graphique de `adabip` en fonction de `site` : `plot(adabip ~ site, data=mydata)`. Vous obtenez un boxplot. Faites de même pour `adabip` en fonction de `date` : `plot(adabip ~ date, data=mydata)`. Vous n'obtenez pas un boxplot parce que `date` n'est pas dans le bon format. Transformez la date en facteur de façon à ce que : `plot(adabip ~ date, data=mydata)` vous donne un boxplot.

```
# Il faut bien spécifier le type de séparateur, le caractère des décimales et des valeurs
# manquantes.

setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
mydata <- read.table("data/data2.txt", sep="\t", dec=".", na.string="", header = TRUE)
summary(mydata)
mean(mydata$adabip, na.rm = TRUE)
plot(adabip ~ site, data=mydata)
plot(adabip ~ date, data=mydata) # ce n'est pas ce qu'on veut

mydata$date <- as.factor(mydata$date)
plot(adabip ~ date, data=mydata)

# Une même analyse statistique ne donnera pas les mêmes résultats selon que les variables
# sont des facteurs ou non...
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
mydata <- read.table("data/data2.txt", sep="\t", dec=".", na.string="", header = TRUE)
lm(adabip ~ date, data=mydata)
mydata$date <- as.factor(mydata$date)
lm(adabip ~ date, data=mydata)
```

- 3) Importez le fichier `data2bis.csv`. Avec la ligne de code suivante (où "mydata" est le nom de votre jeu de données), calculez la moyenne d'`adabip` pour le site "l'île de Niverlée". Vous devez obtenir 33.35.  
`mean(mydata[mydata$site == "L'île de Niverlée", "adabip"])`.



Si ça ne fonctionne pas, n'oubliez pas de regarder le jeu de données que vous avez importé pour essayer de comprendre la source du problème.

```
# Si vous êtes sous windows la colonne site ne sera normalement pas importée correctement.
# Tous les caractères accentués sont remplacés par d'autres caractères.
# Ceci est dû à l'encodage des caractères qui est UTF-8 pour ce fichier alors que sous
# Windows l'encodage par défaut est latin1 (qui porte aussi d'autres noms).
# Vous devez donc soit modifier l'encodage des caractères dans un éditeur de texte avancé
# soit utiliser l'option fileEncoding de read.tables.
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
mydata <- read.table("data/data2bis.csv", sep=";", dec=".", header = TRUE,
                    fileEncoding = "UTF-8")
summary(mydata)
mean(mydata[mydata$site == "L'île de Niverlée", "adabip"])
```

- 4) Importez le fichier data3.xls. Calculez la moyenne de adabip (=31.77) et adadec (=10.42)

```
# Depuis le tableur, exporter le fichier en format texte et ensuite l'importer avec
# read.table.
# Petite difficulté : il y a eu une erreur d'encodage dans adadec : un point virgule à la
# place d'une virgule. Cette colonne est donc reconnue comme du texte et importée comme
# un facteur comme on le voit dans le résultat d'un summary()
# La solution la plus simple dans ce cas-ci est sans doute d'aller modifier le fichier
# d'origine. Il est possible aussi de le faire dans R si on a pas la possibilité de
# modifier les données sources.
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
```

```
mydata <- read.table("data/Exercices_import/data3.csv", header=TRUE, sep="\t",
                    dec=";", na.string="")
summary(mydata)
mean(mydata$adabip, na.rm = TRUE)
mean(mydata$adadec, na.rm = TRUE)

# Avec les données corrigées dans le fichier orriginal
mydata <- read.table("data/Exercices_import/data3_corrige.csv", header=TRUE, sep="\t",
                    dec=";", na.string="")
summary(mydata)
mean(mydata$adabip, na.rm = TRUE)
mean(mydata$adadec, na.rm = TRUE)
```

- 5) Importez le fichier data4.xls. Avec la fonction mean, calculez la moyenne de calqua (=7.19) et oencon (=3.95) (NB : Inspiré de faits réels...)

```
# Plusieurs difficultés ici. Le fichier Excel comprend des valeurs "fantômes" invisibles
# sur certaines lignes et colonnes en dehors de la matrice de données.
# Le plus simple pour éviter ce genre de problèmes est de sélectionner la matrice de
# données et de la copier-coller dans une feuille vierge avant l'exportation en format
# texte.
# Ensuite le titre de la colonne OENCON a été encodé comme OENCON, le deuxième "O" a été
# remplacé par un zéro...
# Il faut donc corriger le titre au bien l'utiliser avec le zéro.
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
mydata <- read.table("data/Exercices_import/data4.csv", header=TRUE, sep="\t", dec=";",
                    na.string="")
summary(mydata)
mean(mydata$CALQUA, na.rm = TRUE)
mean(mydata$OENCON, na.rm = TRUE)
```

- 6) Importez le fichier data5.xls et vérifiez que l'importation s'est faite correctement. Si l'importation ne s'est pas faite correctement, essayez d'importer le fichier sans la dernière colonne et essayez de comprendre l'origine du problème

```
# La difficulté ici vient du fait qu'il y a un champ "mémo" qui contient des informations
# diverses encodées par les utilisateurs.
# On a typiquement des problèmes avec ce genre de champs parce qu'ils contiennent des
# caractères spéciaux qui interfèrent avec la lecture du fichier :
# point virgule, tabulation (plus rare), retour à la ligne, guillemets (fermés ou ouverts).
# La première défense contre ce problème est de bien demander au logiciel lors de
# l'exportation au format texte d'entourer les champs textes par des guillemets.
# Dans bien des cas ça suffit. Mais dans certains cas il faut beaucoup plus chipoter...
# Une mesure radicale est de ne pas importer ces colonnes si elles ne sont pas nécessaires.
# Si non R indique en général le numéro de la ligne qui pose problème. On peut alors
# essayer de comprendre ce qui se passe en ouvrant le fichier dans un éditeur de texte.
# Dans bien des cas il suffit de faire quelques corrections à la main. Si ce n'est pas
# possible, il faudra utiliser readLines ou scan et des expressions régulières pour
# traiter le fichier.
#
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
mydata <- read.table("data/Exercices_import/data5.csv", header=TRUE, sep="\t",
                    dec=",", na.string="")
summary(mydata)
```

### Exercices subsidiaires (pas de R)

- 7) Importez les données contenues dans le fichier "insectes\_feuilles.xls". Calculez la proportion de feuilles F5 où les insectes étaient présents tous traitements et toutes variétés confondus NB : l'essentiel du travail se fait ici dans le tableur  
L'essentiel du travail consiste ici à réorganiser le fichier xls dans le tableur pour qu'il soit exploitable. Il faut éliminer les sous-totaux, fusionner les deux feuilles de calcul, rajouter une colonne "traitement" et une colonne "variété. Cette manière d'encoder les données n'est pas idéale pour le traitement. Il vaut mieux en général encoder les données brutes et faire les statistiques descriptives à part (par exemple avec des tableaux croisés dynamiques/pilotes de données) Lors de l'importation dans R on prendra garde au fait que les cases vides correspondent à des données manquantes (argument na.string de read.table).
- 8) Importez les données contenues dans le fichier "météo.xlsx" issu d'un site internet. Calculez la température minimale moyenne et la quantité totale de précipitations sur la période. NB : l'essentiel du travail se fait ici dans le tableur  
Ici aussi il y a pas mal de boulot à faire dans le tableur pour pouvoir sauver ce fichier dans un format texte exploitable. Il faut défusionner les cellules fusionnées, ensuite faire un tri sur la date pour séparer les lignes contenant les heures de prise de mesure des mesures proprement-dites. Il faut éliminer les unités "°C" et "mm" avec des chercher-remplacer. Attention dans certains cas (Excel 2007 ?) après un rechercher remplacer sur "°C", il reste un caractère invisible qu'il faut également éliminer.

## Exercices sur l'extraction de données (Subscripting)

Chargez le fichier “ladybirds.txt”. Il s'agit de données de comptage de coccinelles sur 3 espèces d'arbres (colonne “tree”) : pins (P), tilleuls (L cfr “Lime”), érables (M cfr “Maple”) dans 3 types de paysages (landsc) : Urbain (U), Suburbain (S) et Rural (R), sur une série de sites et à 4 dates. Les colonnes après la quatrième correspondent aux espèces de coccinelles (nombre d'individus).

```
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
d <- read.table("data/ladybirds.txt", sep="\t", header = TRUE)
summary(d)
```

- 1) Sélectionnez toutes les colonnes sauf la 6, 9 et 12

```
d[, -c(6, 9, 12)]
```

- 2) Sélectionnez les 10 premières lignes et uniquement les colonnes correspondant aux espèces

```
d[1:10, 5:ncol(d)]
d[1:10, -c(1:4)] # équivalent
```

- 3) Sélectionnez les données de l'espèce “haraxy” sur pin.

```
d[d$tree == "P", "haraxy"]
```

- 4) Ne gardez que les observations (lignes) où l'espèce “anaoce” était présente

```
d[d$anaoce > 0,]
```

- 5) Ne gardez que les observations (lignes) “rurales” où “anaoce” était présente

```
d[d$landsc == "R" & d$anaoce > 0,]
```

- 6) Sélectionnez les données des sites PS1, MU2, LS3, PU2, LR3, LR2, MR1

```
d[d$site %in% c("PS1", "MU2", "LS3", "PU2", "LR3", "LR2", "MR1"),]
```

- 7) Sélectionnez toutes les données sauf celles de ces sites (PS1, MU2, LS3, PU2, LR3, LR2, MR1)

```
d[!d$site %in% c("PS1", "MU2", "LS3", "PU2", "LR3", "LR2", "MR1"),]
```

- 8) Sélectionnez les données sur Pins à la date 1 et sur feuillus (tilleuls et érables) pour toutes les dates sauf la date1

```
d[(d$tree == "P" & d$date == "date1") | (d$tree %in% c("M", "L") & d$date != "date1"),]
```

- 9) Sélectionnez les données collectées sur érable ou tilleul en milieu suburbain et pour les espèces haraxy, calqua, exoqua, harqua et aphobl

```
d[ (d$tree == "M" | d$tree == "L") & d$landsc == "S",
  c("tree", "landsc", "date", "haraxy", "calqua", "exoqua", "harqua", "aphobl")]
```

- 10) Sélectionnez 10 lignes aléatoirement (à l'aide de la fonction sample)

```
d[sample(1:nrow(d), 10),]
```

- 11) Sélectionnez une ligne sur 5 (à l'aide de la fonction seq)

```
d[seq(1, nrow(d), 5),]
```

- 12) Transformez les données d'abondance en données binaires (présence/absence). (n'écrasez pas le jeu de donnée)

```
sp <- d[, -c(1:4)]
sp[sp>0] <- 1
cbind(d[, 1:4], sp)
```

- 13) Remplacez les données de plus de 150 individus par des valeurs manquantes

```
# NB dans ce cas les messages d'avertissement (warnings) sont sans importance
d[d>150] <- NA
```

- 14) Remplacez les données d'abondance d'espèce du site PS1 à la date 3 par des valeurs manquantes

```
d[d$site == "PS1" & d$date == "date3", -c(1:4)] <- NA
```

- 15) Multipliez les valeurs des sites sur pins par 5/4

```
d[d$tree == "P", -c(1:4)] <- d[d$tree == "P", -c(1:4)] * 5/4
```

- 16) Réordonnez le jeu de données de manière à ce qu'on ait d'abord la colonne date puis site puis landsc puis tree suivies des colonnes espèces

```
d[, c(4:1, 5: ncol(d))]
```

- 17) Ne gardez dans le jeu de données que les espèces pour lesquelles on a plus de 100 individus observés au total. (utilisez la fonction colSums)

```
colSums(d[, -c(1:4)])
colSums(d[, -c(1:4)], na.rm = TRUE)
colSums(d[, -c(1:4)], na.rm = TRUE) >100

# plusieurs solutions :
spp <- colSums(d[, -c(1:4)], na.rm = TRUE) >100
spp
d[, c(TRUE, TRUE, TRUE, TRUE, spp)]
cbind(d[, 1:4], d[, -c(1:4)][, spp])
d[, c(1:4, which(spp) + 4)]
```

- 18) Triez les lignes par tree, landsc et date (dans cet ordre). Vous devrez utiliser la fonction order.

```
d[order(d$tree, d$landsc, d$date), ]
```

- 19) Comment faire pour que la colonne landsc se classe dans cet ordre : U, S, R ? (utilisez la fonction factor et le paramètre levels= pour changer l'ordre des niveaux du facteur landsc. Ensuite réutilisez la même ligne de code que ci-dessus.)

```
d$landsc <- factor(d$landsc, levels = c("U", "S", "R"), ordered = TRUE)
d[order(d$tree, d$landsc, d$date), ]
```

- 20) Réordonnez colonnes du jeu de données de manière à ce que les espèces soient présentées de la plus abondante à la moins abondante (ie harqua, adabip, myroct, adadec, etc...)

```
or <- order(colSums(d[, -c(1:4)], na.rm = TRUE), decreasing = TRUE)
or
d[, c(1:4, or+4 )]
```

## Exercices de manipulation de caractères

Quelques rappels :

- les fonctions `grep` et `grepl` ont un argument `ignore.case` qui permet d'ignorer la casse des caractères quand on le désire
- dans les expressions régulières, si vous voulez rechercher par exemple un point “.” il faudra chercher “\.” car le point a une signification particulière dans la syntaxe des expressions régulières.

### Exercice 1

Voici un jeu de données sur lequel on va travailler (espèce, nombre d'individus, site)

```
d <- data.frame(
  sp = c("COCCINELLA SEPTEMPUNCTATA", "Coccinella septempunctata",
        "Coccinella Septempunctata Brucki", "Adalia spp", "COCCINELLIDAE",
        "coccinella hieroglyphica", "ADALIA BIPUNCTATA REVIELERI",
        "Adalia bipunctata bipunctata", "Adalia decempunctata", "Formicidae",
        "Harmonia spp.", "Adalia bipunctata", "Coccinella quinquepunctata",
        "Coccinella sp.", "Myrmica speciosides", "NOMADA BISPINOSA",
        "NOMADA sp.", "Formicinae", "Apoidea", "Dinocampus coccinellae", "COLEOPTERA",
        "Aphaenogaster spinosa", "MYRMICA SP.", "Lasius niger", "Formicinae"),
  n = c(1, 5, 10, 33, 2, 1, 4, 6, 1, 3, 1, 1, 2, 1, 9, 3, 1, 1, 3, 2, 6, 1, 2, 1, 3),
  site = c("Mazée", "Namur", "Res. Nat de Matagne", "Treignes",
           "res nat du coupu tienne", "Vaucelles", "Rés. Nat. Roche Madou", "Vierves",
           "natoye", "Olloy", "Le Mesnil", "resnat du Chamousias", "Natoye", "Mazée",
           "Réserve Naturelle de la Haie Gabaux", "Dourbes", "Doische",
           "Oignies", "Mazée", "Vaucelles", "Rés. nat. de la Montagne de la Carrière",
           "Regniessart", "Vireux", "Haybes", "res nat Al Florée")
)
summary(d)
d
```

Rappels sur la nomenclature des noms d'espèces et la taxonomie.

Le nom scientifique d'une espèce est toujours composé du nom de genre suivi du nom d'espèce. Par exemple : *Adalia bipunctata*, *Adalia* étant le nom de genre, *bipunctata*, le nom d'espèce. Lorsqu'on ne sait pas identifier une espèce jusqu'à l'espèce, on indique en general le nom de genre suivi de “sp” ou “spp” (“species”). Par exemple *Adalia sp.* Lorsqu'il y a trois noms à la suite (pex *Adalia bipunctata bipunctata*), le 3ème désigne une sous-espèce. Lorsqu'on a un seul nom terminant par “inae”, “idae”, “oidea”, “era”,... il s'agit de niveaux taxonomiques supérieurs au nom de genre (sous-famille, famille, ordre,...)

- 1) Dans la colonne `sp`, la casse des noms scientifiques n'est pas uniforme. Mettez tous les caractères en majuscule.

```
d$sp <- toupper(d$sp)
```

- 2) Sélectionnez les données du genre *Adalia* (n=5)

```
d[grepl("ADALIA", d$sp), ]
```

- 3) Sélectionnez les données correspondant au genre *Coccinella* (attention aux parasites !). (n = 6) Comment faire pour éviter de sélectionner *Dinocampus coccinellae* (un parasite de coccinelles...) sans changer la casse des caractères ?

```
d[grepl("COCCINELLA", d$sp), ] # pas bon car on inclut Dinocampus coccinellae
# il suffit de spécifier que Coccinella doit être en début de chaîne et éventuellement
# de spécifier qu'il est suivi d'une espace
d[grepl("^COCCINELLA ", d$sp), ]
```

- 4) Sélectionnez les données des genres Adalia et Coccinella (n = 11)

```
d[grepl("^ADALIA |^COCCINELLA ", d$sp), ]
```

- 5) Transformez le vecteur de noms scientifiques “sp” de manière à avoir la première lettre en majuscule et les suivantes en minuscule. Vous pouvez simplement utiliser ici la fonction substring (avec paste, toupper et tolower). Les expressions régulières ne sont nécessaires (il est aussi possible d'utiliser les expressions régulières mais c'est plus compliqué dans ce cas).

```
d$sp <- paste(toupper(substring(d$sp, 1, 1)), tolower(substring(d$sp, 2, 1000)), sep="")
d$sp
# autre solution avec les expressions régulières :
paste0(toupper(gsub("(^.).*", "\\1", d$sp)), tolower(gsub("(^.).*", "\\1", d$sp)))
```

- 6) Sélectionnez uniquement les 6 données qui n'ont PAS été identifiées au moins jusqu'au genre (familles, sous-familles, etc...). Piste : il s'agit des éléments de la colonne “sp” contenant au moins une espace (utilisez regexpr ou grepl).

```
d[!grepl(" ", d$sp), ] # solution 1
d[regexpr(" ", d$sp) < 0, ] # solution 2
```

- 7) Sélectionnez les données qui ont été identifiées au niveau générique ou infra générique (genre espèces, sous-espèces). NB : ne cherchez pas midi à 14 heures. Il suffit normalement de rajouter ou de changer un seul caractère à la ligne de code précédente.

```
d[grepl(" ", d$sp), ] # solution 1
d[!regexpr(" ", d$sp) < 0, ] # solution 2
d[regexpr(" ", d$sp) > 0, ] # solution 3
```

- 8) Créez un code composé des 3 premières lettres du genre suivies des 3 premières lettres de l'espèce. Pour les données identifiées au niveau supragénérique remplacez le code obtenu par une chaîne de caractère vide.  
2 pistes :

- a) utilisez une combinaison de substring et de regexpr pour trouver la position du premier caractère espace
- b) utiliser gsub et les possibilités de capture des expressions régulières

```
# Solution a
esp_position <- regexpr(" ", d$sp)
code <- paste(substring(d$sp, 1, 3), substring(d$sp, esp_position+1, esp_position + 3),
              sep="")
code[esp_position < 0] <- ""
code

# Solution b
code <- gsub("(.{3}).* (.{3}).*", "\\1\\2", d$sp)
code[!grepl(" ", d$sp)] <- ""
code
```

- 9) Sélectionner les données qui ont été identifiées au niveau de la sous-espèce. (NB : utilisez grep ou grepl mais pas gregexpr)

```
# On crée une expression régulière correspondant au début de la chaîne suivi d'un nombre
# quelconque de caractères,
# puis une espace puis un nombre quelconque de caractères puis une deuxième espace puis
# encore un nombre quelconque de
# caractères.
d[grep("^.* .* .*", d$sp),]

# On pourrait en fait utiliser gregexpr pour identifier les noms contenant plus d'une
# espace mais le résultat est sous forme de liste
# Et on a pas encore vu à ce stade comment par exemple aller mesurer la longueur des
# éléments d'une liste (avec sapply).
# De plus, la solution avec grep est plus simple.
sapply(gregexpr(" ", d$sp), length) # nombre d'espace dans chaque élément
d[sapply(gregexpr(" ", d$sp), length) > 1, ]
```

- 10) Pour ces espèces identifiées au niveau de la sous-espèce, transformer le nom de manière à ne garder que le Genre et l'espèce. Utilisez gsub et les possibilités de capture des expressions régulières

```
gsub("(^.* .*).*", "\\1", d$sp)
```

- 11) Sélectionnez uniquement les données qui ont été identifiées au niveau générique mais pas spécifique (ie Adalia sp., Adalia spp, etc...). NB : il y en a 5, pas 7 !

```
# Le code suivant sélectionne les taxons contenant une espace suivie de sp mais ce n'est
# pas suffisant car il sélectionne aussi Myrmica specioides et Aphaenogaster spinosa
d[grepl(" sp", d$sp), "sp"]

# Avec le code suivant, on sélectionne les taxons contenant une espace suivie de sp suivi
# de 0 ou 1 "p" suivi de 0 ou 1 point (le double backslash \\ est nécessaire pour
# échapper à la signification du point tout seul qui vaut pour n'importe quel caractère)
d[grepl(" spp?\\.?$", d$sp), "sp"]
```

- 12) Sélectionnez toutes les données observées dans des réserves naturelles. (NB il y en a 7, pas 9) Dans la colonne "site" les données observées dans des réserves naturelles sont systématiquement indiquées mais de manière peu uniformisée (res. nat, résnat, etc...)

```
d[grep("r[eé]s.* ?nat.*", d$site, ignore.case=TRUE),]

# plus simple mais pas correct car on sélectionne aussi les données de Natoye
d[grep("nat", d$site, ignore.case=TRUE),]
```

- 13) Sélectionnez les données du genre Adalia observées dans des réserves naturelles (2 données). Il faut impérativement utiliser ici grepl, la version de grep qui retourne un vecteur logique

```
d[grepl("r[eé]s.* ?nat.*", d$site, ignore.case=TRUE) &
  grepl("^Adalia ", d$sp, ignore.case=TRUE),]
```



## Exercice 2

Voici un vecteur de coordonnées géographiques en Degrés Minutes Secondes (DMS).

```
DMS <- c("50°45'58.32\"N", "50°51'12.73\"S", "4°32'5.66\"E", "3°34'53.66\"W",  
        "2°28'7.26\"0", "50°36'23.88\"", "50°36'23\"", "50°36'23.65\"'N (N2)",  
        "50°36'23.65\"'N (N2)", "50°36'07,26\"N", " 50 ° 45 ' 58,32 \" N")
```

On veut le transformer en degrés décimaux ( $DD = D + M/60 + S/(60*60)$ ). Les 4 premières coordonnées sont correctes et cohérentes. Les suivantes n'ont pas toutes été encodées de la même manière (NB : inspiré de faits réels...)

Utilisez les expressions régulières pour extraire les valeurs correspondant aux degrés, aux minutes, aux secondes et à l'hémisphère (par défaut on considère qu'on est dans l'hémisphère nord et à l'est du méridien de Greenwich).

Conseil : Commencez par transformer les 4 premières valeurs puis copiez progressivement vos expressions pour prendre en compte tous les cas particuliers...

Vous devriez obtenir les valeurs suivantes :

```
c(50.7662, -50.853536, 4.534906, -3.581572, -2.468683, 50.606633, 50.606389, 50.606569, 50.606569, 50.602017, 50.7662)
```

```
D <- as.numeric(gsub(pattern = " *([0-9]{1,2}) *°.*", "\\1", x = DMS))  
M <- as.numeric(gsub(pattern = ".*° *([0-9]{1,2}) *'.*", "\\1", x = DMS))  
  
S <- gsub(pattern = ".*°.*' *([0-9]{1,2}[\\.,\\,]{0,1}[0-9]{0,2}) *([\\\"' ]).*",  
           "\\1", x = DMS)  
S  
# remplacer la virgule par un point puis transformer en nombres  
S <- as.numeric(gsub(",", "\\.", S))  
S  
  
sign <- gsub(pattern = ".*°.*'.*([\\\"' ]) *([NSEOW]).*", "\\1", x = DMS)  
sign  
sign <- ifelse(sign %in% c("S", "O", "W"), -1, 1)  
sign  
  
DD <- (D + M/60 + S/(60*60)) * sign  
DD
```

## Structures de contrôle : fonctions, boucles, exécutions conditionnelles,...

### Exercice 0

Petits exercices simples de mise en jambes...

- 1) Créez une fonction “salut” qui écrit simplement “Bonjour”. Cette fonction n’a pas d’arguments : salut() écrira “Bonjour”
- 2) Ajoutez un argument “qui” pour pouvoir choisir un nom quelconque: salut(qui = “Gilles”) écrira “Bonjour Gilles”, salut(“Louis”) écrira “Bonjour Louis”
- 3) Ajoutez un argument “english” de type vrai/faux permettant de choisir la langue : salut(qui = “Gilles”, english = FALSE) écrira “Bonjour Gilles” et salut(“Philippe”, english = TRUE) écrira “Hello Philippe”

```
salut <- function() {print("Bonjour")}
salut()

salut <- function(qui = "") {
  a <- paste("Bonjour", qui)
  print(a)
}
salut()
salut("Georges")

salut <- function(qui = "", english = FALSE) {
  if(english == TRUE) {
    a <- paste("Hello", qui)
  } else {
    a <- paste("Bonjour", qui)
  }
  print(a)
}

salut()
salut(english = TRUE)
salut("Philippe")
salut("Philippe", TRUE)
```

- 4) Au moyen d’une boucle for, calculez pour chaque ligne de la matrice suivante, la moyenne et l’écart type (sd). Stockez ces valeurs dans une nouvelle matrice à 2 colonnes et 10 lignes. NB : en pratique on ne devrait pas utiliser de boucle pour ce genre de calculs (on utiliserait rowMeans et apply).

```
mat <- matrix(c(1:40), 10, 4, byrow=TRUE)
```

```
meansd <- matrix(NA, 10, 2) # matrice vide pour stocker les résultats
colnames(meansd) <- c("mean", "sd")

for(i in 1:nrow(mat)) {
  meansd[i,1] <- mean(mat[i,])
  meansd[i,2] <- sd(mat[i,])
}
meansd
```

- 5) Utilisation de la fonction merge. On a un dataset “obs” qui contient un nombre d’observations pour une série d’espèces (sp) identifiées par un code à 6 lettres.

On veut ajouter une colonne à ce dataset avec le nom scientifique complet correspondant. Les correspondances code - nom scientifiques se trouvent dans le dataset "tax".

Attention certains codes de la table obs n'ont pas de correspondance dans "tax". Veuillez malgré tout à ne pas perdre de données dans la table "obs".

```
obs <- data.frame (
  sp = c("cocund", "cocsep", "cocsep", "scysut", "psyvig", "adabip", "rhychr", "adabip",
        "psyvig", "cocsep"),
  nbr = c(2,3,46,3,4,8,9,20,4,3)
)

tax <- data.frame(
  code = c("adabip", "cocsep", "cocund", "psyvig", "subvig", "epiarg"),
  taxon = c("Adalia bipunctata", "Coccinella septempunctata",
            "Coccinella undecimpunctata", "Psyllobora vigintiduopunctata",
            "Subcoccinella vigintiquatuorpunctata", "Epilachna argus")
)
```

```
merge(obs, tax, by.x = "sp", by.y="code", all.x = TRUE)
```

## Exercice 1

- Construisez une fonction qui crée un code composé des 3 premières lettres du nom de genre et des 3 premières lettres du nom d'espèce. L'exercice a déjà été fait précédemment il suffit ici de l'encapsuler dans une fonction. Pour rappel les lignes de code suivant permettaient de faire le travail (en dehors d'une fonction) :

```
esp_position <- regexpr(" ", x)
code <- paste(substring(x, 1, 3), substring(x, esp_position + 1, esp_position + 3), sep="")
N'oubliez pas de créer un vecteur exemple de noms scientifiques pour tester votre fonction.
```

```
noms <- c("Libellula depressa", "Coenagrion mercuriale", "Ceriagrion tenellum",
        "Lestes sponsa")

codesp <- function (x) {
  esp_position <- regexpr(" ", x)
  code <- paste(substring(x, 1, 3), substring(x, esp_position + 1, esp_position + 3),
               sep="")
  return(code)
}
codesp(noms)
```

- Modifiez ensuite cette fonction de façon à ce qu'on puisse choisir le nombre de lettres prises en compte

```
codesp <- function (x, nblettres = 3) {
  esp_position <- regexpr(" ", x)
  code <- paste(substring(x, 1, nblettres),
               substring(x, esp_position + 1, esp_position + nblettres), sep="")
  return(code)
}
codesp(noms)
codesp(noms, 5)
```

- Modifiez encore votre fonction de façon à ce que l'utilisateur puisse choisir de mettre toutes les lettres du code en majuscule

```
codesp <- function (x, nblettres = 3, uppercase = FALSE) {
  esp_position <- regexpr(" ", x)
  code <- paste(substring(x, 1, nblettres),
               substring(x, esp_position+1, esp_position + nblettres), sep="")
  if(uppercase) code <- toupper(code)
  return(code)
}
codesp(noms, 5)
codesp(noms, 5, TRUE)
```

- Modifiez votre fonction de façon à ce que l'utilisateur puisse choisir de mettre toutes les lettres du code en majuscule ou en minuscule ou de pas y toucher

```
codesp <- function (x, nblettres = 3, case = c("upper", "lower", "asis")) {
  esp_position <- regexpr(" ", x)
  code <- paste(substring(x, 1, nblettres),
               substring(x, esp_position+1, esp_position + nblettres), sep="")
  case <- match.arg(case) #permet d'éviter un message et vérifie que les arg. sont bons
  if(case == "upper") code <- toupper(code) else
    if(case == "lower") code <- tolower(code)
  return(code)
}
codesp(noms, 5)
codesp(noms, 5, "lower")
codesp(noms, 5, "asis")
```

```
codesp(noms, 5, "nimportequoi")
```

## Exercice 2

- Chargez le fichier “ladybirds\_aggr.txt”. Changez les valeurs des deux premières colonnes pour des valeurs plus explicites : tree : M = Mapple, L = Lime, P = Pine ; landsc : U = Urban, S = Suburban, R = Rural (le plus simple est d'utiliser la fonction `levels()` <-)

```
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
d <- read.table("data/ladybirds_aggr.txt", sep="\t", header=TRUE)

levels(d$landsc) <- c("Rural", "Suburban", "Urban")
levels(d$tree) <- c("Lime", "Mapple", "Pine")
```

- Réordonnez le tableau selon l'espèce d'arbre (tree) et ensuite du paysage (landsc) et dans l'ordre inverse de l'ordre alphabétique.

```
d <- d[order(d$tree, d$landsc, decreasing=TRUE),]
```

- Tracez un barplot de la première ligne du tableau représentant l'abondance des différentes espèces de coccinelles sur Pins en milieu urbain. Ajoutez dans le titre l'espèce d'arbre et le paysage correspondant (paramètre “main” de la fonction `plot`).

```
# dev.new(width = 10/2.54, height = 8/2.54)
par(mar = c(4,3,3,1), las=2)
barplot(as.matrix(d[1, -c(1,2)]), main = paste(d[1,1], d[1,2], sep="-"))
```

- A l'aide d'une boucle "for" répétez ensuite le même graphique pour chaque ligne du tableau en adaptant les titres. Juste avant la boucle for placez la commande suivante : `par(mfrow = c(3,3), mar = c(4,3,3,1), las=2, cex = 0.7)` mfrow divise la fenêtre graphique en 9 parties, mar ajuste la taille des marges, las force les étiquettes à être perpendiculaires à l'axe et cex diminue la taille des caractères (on détaillera ces options plus loin dans la partie sur les graphiques).  
Ne vous tracassez pas trop de la mise en forme des graphiques qui sera abordée plus loin.

```
# dev.new(width = 17/2.54, height = 12/2.54)
par(mfrow = c(3,3), mar = c(4,3,3,1), las=2, cex = 0.7)
for (i in 1:nrow(d)){
  barplot(as.matrix(d[i, -c(1,2)]), main = paste(d[i,1], d[i,2], sep="-"))
}
```

- Faites ensuite un barplot pour une espèce de coccinelle en fonction des différentes combinaisons d'arbre et de paysage (on va travailler sur les colonnes, avec une barre par ligne du tableau de données). Vous devrez ajouter vous-même les étiquettes au moyen de l'argument "names.arg" de la fonction barplot. Pour créer ces étiquettes, collez les valeurs des deux premières colonnes et séparez les par un caractère "\n" qui représente un retour à la ligne.
- Faites ensuite un graphique similaire pour chacune des 12 espèces au moyen d'une boucle. Avant la boucle for, ajoutez la ligne de code suivante : `par(mfrow = c(4,3), mar = c(4.5,2,3,1), las=2, cex = 0.65)`  
N'oubliez pas d'ajouter les titres appropriés.

```
# dev.new(width = 10/2.54, height = 7/2.54)
par(mfrow=c(1,1),mar = c(4.5,2,3,1), las=2, cex = 0.8)
barplot(d[, "haraxy"], names.arg = paste(d$tree, d$landsc, sep="\n"),
        main = colnames(d)[3], col = c("red", "orange", "yellow"))
```

```
# dev.new(width = 18/2.54, height = 18/2.54)
par(mfrow = c(4,3), mar = c(4.5,2,3,1), las=2, cex = 0.65)
for(i in 3:ncol(d)){
  barplot(d[,i], names.arg = paste(d$tree, d$landsc, sep="\n"),
        main = colnames(d)[i], col = c("red", "orange", "yellow"))
}
```

*# Pour info : avec le package graphique ggplot2, on peut se passer de boucle et obtenir  
# un résultat plus "fini"*

```
# dev.new(width = 17/2.54, height = 12/2.54)
library(ggplot2)
library(reshape)
mnd <- melt(d, id = 1:2, variable_name="sp")
ggplot(mnd, aes(y = value, x = sp)) +
  geom_bar(stat = "identity") +
  facet_grid(tree ~ landsc, scales = "free") +
  theme_bw(11) +
  theme(axis.text.x = element_text(angle=45, hjust=1, vjust=1))
```

```
# dev.new(width = 18/2.54, height = 18/2.54)
mnd$landsc <- factor(mnd$landsc, levels = c("Urban", "Suburban", "Rural"))
ggplot(mnd, aes(y = value, x = tree, fill = landsc)) +
  geom_bar(stat = "identity", position = "dodge") +
  facet_wrap(~ sp, scales = "free", nrow = 4) +
  theme_bw(10) +
  scale_fill_manual(name = "Landscape", values = c("red", "orange", "yellow"))
```

### Exercice 3

Chargez les fichiers `biche1.csv` (séparateur : tabulations) et `ephemerides.csv` (séparateur : point virgule).

Le fichier “biche” contient les dates, heures et localisations GPS d’une biche. Le fichier éphémérides contient les heures de coucher (sunrise) et de coucher de soleil (sunset) à Uccle.

On veut pour chaque position de la biche, déterminer si il faisait jour ou nuit au moment de l’enregistrement.

Il faut commencer par formater correctement les colonnes date, sunset et sunrise du fichier d’éphémérides et les colonnes date et time du fichier biche. Attention, les éphémérides sont en heure civile pour l’Europe centrale (tz=“CET”) alors que le fichier biche est en temps universel (tz=“UTC”). Afin que les comparaisons et calculs soient corrects, il faut donc bien indiquer le fuseau horaire au moyen de l’argument tz au moment où on formate les temps (avec `strptime`). Pour certaines opérations, R retourne un message d’avis quand les temps impliqués ne sont pas dans le même tz mais les comparaisons sont néanmoins correctes.

Fusionnez ensuite les deux tables (merge) et créez une colonne jour/nuit. Il y a 155 données de jour et 288 données de nuit dans ce fichier. Vous pouvez le vérifier par un `summary()` sur votre colonne jour/nuit après l’avoir transformée en facteur.

```
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
biche <- read.table("data/biche1.csv", sep="\t", header=TRUE)
ephem <- read.table("data/ephemerides.csv", sep=";", header=TRUE)

# On transforme les données de ephem en dates et temps, en spécifiant le fuseau horaire
summary(ephem)
ephem$date <- as.Date(ephem$date, format = "%d %m %Y")
ephem$sunrise <- strptime(paste(ephem$date, ephem$sunrise),
                          format = "%Y-%m-%d %H:%M", tz="CET")
ephem$sunset <- strptime(paste(ephem$date, ephem$sunset),
                        format = "%Y-%m-%d %H:%M", tz="CET")

# On transforme les variables temps et date du jeu de données biche
summary(biche)
biche$date <- as.Date(biche$date, format = "%d/%m/%Y")
biche$time <- strptime(paste(biche$date, biche$time),
                      format = "%Y-%m-%d %H:%M:%S", tz="UTC")

# fusion des tables
biche <- merge(biche, ephem[, 1:3], by = "date", all.x=TRUE)
summary(biche)

biche$period <- ifelse(biche$time > biche$sunrise & biche$time < biche$sunset,
                      "day", "night")
biche$period <- as.factor(biche$period)
summary(biche$period)
```

### Exercice 4

Un des but de l’exercice précédent était d’utiliser la fonction `merge`. Cependant, il serait plus efficace dans ce cas précis de pouvoir déterminer la période de la journée sans passer par un fichier extérieur. C’est ce que permet de faire la fonction `sunriset` du package `maptools` qui calcule l’heure de coucher ou de lever de soleil sur base des coordonnées géographiques et de la date. La fonction `crepuscule` permet également de déterminer les heures de pénombre.

Ecrivez une fonction qui détermine pour n’importe quelle date-heure en un point précis si il s’agit du jour ou de la nuit. Vous devez utiliser les colonnes longitude et latitude et la méthode appelée “## S4 method for signature ‘matrix,POSIXct’” dans l’aide de `sunriset`. Attention, par défaut le résultat de `sunriset` est donné dans le fuseau horaire local. Si le fuseau horaire du vecteur date-temps reçu en argument par `sunriset` est bien spécifié, la valeur retournée par `sunriset` sera dans ce fuseau horaire. Il peut y avoir de légères différences entre les résultats de `sunriset` et le fichier `ephemerides.csv`, ce dernier étant calculé pour Uccle (il y a 6 différences). Vous pouvez utiliser les latitude et longitude de l’observatoire d’Uccle comme valeurs par défaut à votre fonction (long = 4.357886, lat = 50.79873)

```

# long lat par défaut calculé pour l'observatoire d'Uccle
daynight <- function(time, long = 4.357886, lat = 50.79873) {

  require(maptools)

  crd <- as.matrix(cbind(long, lat))
  time <- as.POSIXct(time)

  sunset <- sunriset(crd= crd, dateTime = time, POSIXct.out=TRUE,
                    direction = "sunset")[,2]
  sunrise <- sunriset(crd= crd, dateTime = time, POSIXct.out=TRUE,
                    direction = "sunrise")[,2]

  ifelse(time > sunrise & time < sunset, "day", "night")
}

biche$periodbis <- daynight(biche$time, biche$long, biche$lat)
biche[1:20,c("time", "sunrise", "sunset", "period", "periodbis")]
sum(biche$period != biche$periodbis) # nombre de différences

# à l'observatoire d'Uccle --> les deux approches correspondent parfaitement
biche$periodbis <- daynight(biche$time)
biche[1:20,c("time", "sunrise", "sunset", "period", "periodbis")]
sum(biche$period != biche$periodbis) # nombre de différences

```

## Exercice 5

Au moyen d'une boucle, modifiez le code des deux exercices précédents de façon à répéter ces opérations pour les 10 fichiers biche1.csv, biche2.csv,... Votre programme devrait fonctionner pour n'importe quel nombre de fichiers. Utilisez la fonction `list.files` pour faire la liste des fichiers visés et la stocker dans un vecteur. Utilisez ensuite ce vecteur pour lire successivement les différents fichiers dans une boucle et ajoutez leur la colonne jour/nuit. Ajoutez en plus une colonne identifiant chaque biche (chaque fichier). Vous pouvez stocker chaque jeu de données dans les éléments d'une liste.

```

setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices/data")
fichiers <- list.files(pattern = "^biche[0-9]+\\.csv$")

# liste vide qui va servir à stocker les jeux de données
multibiches <- as.list(vector(length = length(fichiers)))

for (i in 1:length(fichiers)) {

  biche <- read.table(fichiers[i], sep="\t", header=TRUE)

  #' On transforme les variables temps et date
  biche$date <- as.Date(biche$date, format = "%d/%m/%Y")
  biche$time <- strptime(paste(biche$date, biche$time), format = "%Y-%m-%d %H:%M:%S",
                        tz="UTC")

  biche$period <- daynight(time = biche$time , lat = biche$lat, long = biche$long)

  # on ajoute le nom de fichier dans une colonne
  nom <- gsub("(^biche[0-9]+\\.csv$", "\\1", fichiers[i])
  biche$file <- nom
}

```

```

multibiches[[i]] <- biche
# on ajoute aussi le nom de fichier dans les noms des éléments de la liste
names(multibiches)[i] <- nom
}

summary(multibiches)
str(multibiches)

# On peut coller les jeux de données les uns à la suite des autres

res <- rbind(multibiches[[1]], multibiches[[2]], multibiches[[3]]) # 3 fichiers
res <- do.call(rbind, multibiches) # tous les fichiers
summary(res)
head(res)

```



## Agrégation et reshaping

NB1 : dans ces exercices, on utilisera jamais de boucle explicite.

NB2 : par “créer une fonction à la volée”, on entend créer une fonction à usage unique qui n’est pas sauvée dans un objet. On utilise typiquement ce genre de fonctions dans les fonctions de la famille apply.

### Exercice 1

Chargez le jeu de données “ladybirds.txt”. On a déjà utilisé à plusieurs reprises un jeu de données similaire. Les 4 premières colonnes contiennent : l’espèce d’arbre (tree : Pine, Mapple ou Lime), le paysage (landsc : Urban, Suburban, Rural), un code identifiant le site (site) et la période d’échantillonnage (date). Les colonnes suivantes contiennent des abondances de coccinelles identifiées par un code à 6 lettres.

```
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
d <- read.table("data/ladybirds.txt", header=TRUE, sep = "\t")
summary(d)
```

- 1) faites le total du nombre d’individus pour chaque espèce et par site. Utilisez la fonction aggregate et sauvez le résultat dans un dataset appelé “site”. Gardez dans ce jeu de données les colonnes tree, landsc et site dont vous aurez besoin dans les exercices suivants.

```
# Deux notations possibles pour aggregate avec des résultats identiques :
aggregate(.~tree + landsc + site, data=d[, -4], FUN = sum)
site <- aggregate(d[, 5:ncol(d)], d[, c("tree", "landsc", "site")], FUN = sum)
```

- 2) Au moyen de la fonction apply, calculez le nombre moyen et l’écart-type de chaque espèce de coccinelles entre les sites (autrement dit : la moyenne et l’écart-type des colonnes espèces du dataset “site”).  
NB : on pourrait utiliser ici la fonction colMeans pour la moyenne mais il n’y a pas d’équivalent pour l’écart-type.  
haraxy : moyenne = 6.62, sd = 12.22

```
apply(site[, 4:ncol(site)], 2, mean)
apply(site[, 4:ncol(site)], 2, sd)
```

- 3) Utilisez les fonctions lapply et puis sapply pour obtenir le même résultat que la fonction précédente (la présentation peut être différente mais pas les chiffres). Le jeu de donnée étant un data.frame, on peut le traiter comme une liste dont les éléments sont les colonnes du data.frame, ce qui permet d’utiliser les fonctions sapply et lapply.

```
lapply(site[, 4:ncol(site)], mean)
sapply(site[, 4:ncol(site)], mean)
```

- 4) Au moyen de la fonction apply, calculez le nombre total de coccinelles (toutes espèces confondues) pour chaque site. (on veut donc faire la somme des lignes du dataset “site”)  
NB1 : on devrait de préférence utiliser ici la fonction rowSums  
NB2 : impossible ici d’utiliser sapply ou lapply car on travaille sur les lignes

```
apply(site[, 4:ncol(site)], 1, sum)
```

- 5) Ecrivez une fonction qui calcule l’indice de diversité de Simpson d’un site sur base du nombre d’individus de chaque espèce. Cet indice se calcule comme 1 moins la somme du carré des abondances relatives de chaque espèce. Calculez ensuite cet indice de simpson pour chaque site. (indice pour le site LR1 = 0.76)

```
simpson <- function(x) { 1-sum((x/sum(x))^2) }
apply(site[, 4:ncol(site)], 1, simpson)
```

- 6) En partant du dataset “site”, calculez l’abondance moyenne et l’écart type pour chaque combinaison d’arbre et de paysage.  
Vous devrez utiliser la fonction aggregate. Le résultat est un tableau à 9 lignes et 14 colonnes.

```
aggregate(site[,4:ncol(site)], site[, c("tree","landsc")], FUN = mean)
aggregate(site[,4:ncol(site)], site[, c("tree","landsc")], FUN = sd)
```

- 7) Faites la même opération que dans la question précédente mais avec un résultat arrondi à 2 décimales (vous pouvez le faire uniquement pour la moyenne). Essayez de répondre à cette question en utilisant les 2 approches différentes suivantes :
  - Appliquez simplement la fonction round au résultat de la fonction aggregate de la question précédente (moins les 2 premières colonnes).
  - Au lieu d’utiliser la fonction mean seule dans l’aggregate, créez à la volée une fonction qui calcule la moyenne et l’arrondit à 2 décimales.

*# Solution 1 :*

```
a <- aggregate(site[,4:ncol(site)], site[, c("tree","landsc")], FUN = mean)
round(a[,-c(1,2)],2)
```

*# Solution 2 :*

```
aggregate(site[,4:ncol(site)], site[, c("tree","landsc")], FUN = function(x) {round(mean(x), 2)})
```

- 8) Si vous le désirez, explorez les possibilités de la fonction cast du package reshape. Cette fonction permet de faire tout ce que aggregate et tapply font mais avec une syntaxe beaucoup plus simple.  
Vous devrez commencer par passer le jeu de données en format “long” avec la fonction melt()

```
library(reshape)
md <- melt(d, id = 1:4, variable_name = "sp")
cast(md, tree + landsc ~ sp, fun = sum)
cast(md, tree + landsc + site ~ sp, fun = sum)
cast(md, sp + landsc ~ date, fun = sum)
cast(md, landsc ~ date | sp, fun = sum)
cast(md, sp ~ tree + landsc, fun = sum)
```

## Exercice 2

Chargez le jeu de données “dff.csv” et visualisez sa structure. Il s’agit d’un jeu de données faunistiques (de 15000 lignes) typique des jeu de données récoltés par des naturalistes volontaires : espèces observée (sp), position géographique (utm1, utm5 : mailles de 1 et 5 km<sup>2</sup>, x, y : coordonnées géographiques), date (dat2), la plante hôte (plante), le nombre d’individus (n) et diverses infos comme la méthode de capture (capt), le stade de développement (devl), le type de support (micr).

```
# On lit le jeu de données, on affiche un résumé et on visualise les 10 premières lignes
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
d <- read.table("data/dff.csv", header=TRUE, sep=";", quote="\"")
summary(d)
d[1:10,]
```

- 1) Combien a-t-on d'observations par espèce ?  
Essayez d'arriver au même résultat avec la fonction `table` (la plus simple dans ce cas) et `aggregate`. Pour `aggregate` utilisez la fonction “`length`” pour compter le nombre d'observations.  
(réponse : `sp_01` : 1133 obs, etc)

```
table(d$sp)
# même résultat avec aggregate (le premier argument peut être n'importe quelle colonne)
aggregate(d$sp, list(d$sp), length)
```

- 2) Combien a-t-on de données par espèce ? On définira la donnée comme une combinaison unique de `sp` x `utm1` x `dat2`.  
Utilisez la fonction “`unique`” pour agréger les valeurs identiques suivie de `table`.  
(réponse : `sp_01` : 946 données etc)

```
prov <- unique(d[,c("sp", "utm1", "dat2")])
table(prov$sp)
```

- 3) Quels sont les 30 carrés `utm` de 5 km<sup>2</sup> (`utm5`) avec le plus d'espèces ? (réponse : 31UER775775 et 16 autres carrés avec 9 espèces)

```
prov <- unique(d[,c("sp", "utm5")]) # combinaison unique de espèces x carrés
sort(table(prov$utm5), decreasing = TRUE) [1:30]
```

- 4) Extrairez l'année, le mois et le jour de `dat2` (utilisez simplement `substring`) et stockez les dans 3 nouvelles colonnes du jeu de données

```
d$Y <- substring(d$dat2, 1,4)
d$M <- substring(d$dat2, 5,6)
d$D <- substring(d$dat2, 7,8)
```

- 5) Faites un graphique (par exemple `barplot`) de l'évolution du nombre total d'observations au cours des années (240 observations en 2011).  
Faites de même un graphique montrant la phénologie de toutes les espèces confondues par mois (17 observations en Janvier).

```
barplot(table(d$Y))
barplot(table(d$M), las=1)
```

- 6) Faites un graphique phénologique par décade (toutes espèces confondues). Vous pouvez par exemple utiliser la fonction `cut` pour subdiviser le vecteur “jours” en 3 groupes et coller ce nouveau vecteur avec le vecteur mois. (3ème décade de Mars : 55 données)

```
d$decade <- cut(as.numeric(d$D), breaks = c(0,11,21,31), labels = c("1", "2", "3"))
d$decade <- as.factor(paste(d$M, d$decade, sep="_"))
barplot(table(d$decade))
```

- 7) Faites un tableau espèces x décade. Appliquez la fonction `barplot` à chaque ligne de ce tableau (fonction `apply`).  
Avant la ligne de code `apply`, insérez la commande suivante qui divise la fenêtre graphique en 9 zones et modifie les marges :  
`par(mfrow=c(3,3), mar = c(3,3,1,1))`  
NB : il n'y a pas de manière très directe d'ajouter un titre explicite aux graphiques en utilisant `apply` (en récupérant les noms de colonne). Dans ce cas une boucle peut être plus adaptée.

```
# dev.new(width = 17/2.54, height = 12/2.54)
pheno <- table(d$sp, d$decade)
par(mfrow=c(3,3), mar = c(3,3,1,1))
apply(pheno, 1, barplot)
```

- 8) Calculez le nombre d'observations par année pour chaque espèce (sp\_1 en 2011 : 94 données). Utilisez deux approches différentes pour obtenir les mêmes chiffres (mais présentés différemment) :
  - fonction table (le plus simple et le plus direct dans ce cas) : vous obtiendrez une table espèces x années à 9 lignes x 14 colonnes
  - fonction aggregate avec comme fonction d'agrégation "length". Les mêmes valeurs seront dans un tableau à 3 colonnes.

```
table(d$sp, d$Y)
ag <- aggregate(d$sp, d[, c("Y", "sp")], length)
ag[1:20,]
```

- 9) De façon similaire à la question précédente, calculez le nombre maximum d'individus(max(n)) observés pour chaque année et chaque espèce avec les deux approches possibles. NB : Vous devrez utiliser tapply à la place de table ici.

```
tapply(d$n, d[, c("Y", "sp")], max)
ag <- aggregate(d$n, d[, c("Y", "sp")], max)
ag[1:20,]
```

- 10) Trouvez la date d'observation la plus précoce pour chaque année et chaque espèce. (= 215 en 1998 pour sp\_1) Commencez par créer une colonne au format Date. Utilisez ensuite la fonction aggregate et min. Transformez ensuite la colonne date du résultat de votre aggregate en nombre de jours depuis le premier janvier de chaque année. Vous pouvez utiliser pour ce faire la fonction "format" avec pour argument format "%j" à transformer ensuite en numérique (voir aide de strptime). Il n'est pas nécessaire de passer en format POSIX. Réorganisez ensuite le jeu de données obtenu en un tableau espèces x années. Unstack ne fonctionne pas bien ici à cause des valeurs manquantes pour certaines années. Vous pouvez utiliser soit la fonction cast du package reshape, soit la fonction tapply avec comme fonction d'agrégation la somme ou la moyenne. Comme il n'y a qu'une seule valeur par combinaison d'espèce x année cette opération ne changera pas les valeurs et ne fera que réorganiser le jeu de données. NB : Vous pouvez aussi essayer tapply à la place de aggregate dès le début mais le format date est alors directement transformé en numérique et les manipulations sont un peu plus compliquées. Pour pouvoir transformer cette table en format date, il faut d'abord transformer la table en data.frame avec as.data.frame.matrix. Ensuite on peut transformer ces valeurs en appliquant as.Date à chaque colonne avec lapply et en spécifiant comme origine le 1er janvier 1970.

```
d$date <- as.Date(as.character(d$dat2), format = "%Y%m%d")
res <- aggregate(d$date, list(Y= d$Y, sp = d$sp), min)
res[1:20,]
res$x <- as.numeric(format(res$x, "%j"))
```

```
# unstack : pas très adapté ici à cause du fait qu'il y a des valeurs manquantes
# pour certaines années
unstack(res[, c("Y", "x")], x ~ Y)
```

```
# tapply
tapply(res$x, res[,c("sp", "Y")], sum)
```

```
# cast
library(reshape)
cast(res, sp ~ Y)

# avec tapply à la place de aggregate
a <- tapply(d$date, list(Y= d$Y, sp = d$sp), min)
a
b <- as.data.frame.matrix(a)
b[,1:ncol(b)] <- lapply(b, as.Date, origin = "1970-01-01")
b
```

- 11) Ajoutez au jeu de données un vecteur de dates en nombre de jours depuis le premier janvier de cette année. (format(x, format = "%j") Sur base de ce nouveau vecteur, trouvez les 5 dates les plus précoces par année et par espèce et calculez en la moyenne (pour chaque année et chaque espèce). Vous pouvez utiliser aggregate ou tapply au choix. Vous devrez construire une fonction personnalisée qui trie les dates, extrait les 5 premières et en calcule la moyenne (sans oublier de gérer les NA).

```
d$date2 <- as.numeric(format(d$date, "%j"))

res <- aggregate(d$date2, list(Y= d$Y, sp = d$sp), function(x) mean(sort(x)[1:5],
                                                                    na.rm=TRUE))

res[1:20,]

tapply(d$date2, list(Y= d$Y, sp = d$sp), function(x) mean(sort(x)[1:5], na.rm=TRUE))
```

- 12) Pour chaque espèce faites une liste des carrés de 5 km<sup>2</sup> (utm5) où cette espèce a été observée au moins 3 fois. Ensuite comptez les nombres de ces carrés pour chaque espèce. Commencez par faire une table utm5 x sp. Sauvez les noms de lignes (noms des utm5) dans un vecteur. Ensuite avec l'aide de la fonction apply, récupérez pour chaque espèce les noms des utm5 pour lesquelles le nombre d'observations est > 3 (vous devrez construire une fonction à la volée). Le résultat sera une liste. Avec les fonctions lapply puis sapply comptez le nombre d'utm5 pour chaque espèce. Visualisez les différences de format entre ces deux fonctions.

```
utm <- table(d$utm5, d$sp)
head(utm)
utmnames <- row.names(utm)
l <- apply(utm, 2, function(x) utmnames[x>3])
l[6:9]
lapply(l, length)
sapply(l, length)
```

- 13) Pour chaque espèce calculez le nombre de carrés utm5 où l'espèce a été vue au moins 0, 1, 2, 3, ... fois. Appliquez simplement (avec apply) la fonction table à chaque colonne de votre tableau utm5 x sp

```
apply(utm, 2, table)
```

- 14) Créez un tableau espèces de plantes x espèces d'insectes donnant le nombre d'observations pour chaque combinaison. Éliminez les combinaisons plante x espèce pour lesquelles on a seulement une observation (transformez les 1 en 0) et ensuite éliminez les plantes pour lesquelles on a aucune observations (pex en créant un vecteur logique sur base de rowSums). Combien y a-t-il de plantes hôtes par espèce sur base de ce tableau ? (131 plantes pour sp\_1)

```

plant <- table(d$plante, d$sp)
plant[plant == 1] <- 0
plant <- plant[rowSums(plant)>0,]
plant[1:20,]

# nombre de plantes hôtes
colSums(plant>0)

```

- 15) Pour chaque espèce, établissez la liste des dix plantes hôtes les plus utilisées avec le nombre d'observations pour chaque plante. Vous devez d'abord transformer votre tableau plante x espèce en data.frame. Si votre tableau est encore au format "table" vous devez utiliser as.data.frame.matrix. Si il est déjà au format "matrix" (ça dépend des opérations faites sur votre table d'origine) vous pouvez utiliser juste as.data.frame ou as.data.frame.matrix. Ensuite vous pouvez utiliser lapply pour appliquer à chaque colonne du tableau plante x espèce une fonction personnalisée qui trie les abondances par ordre décroissant en extrait les 10 premières. Avec une simple fonction comme celle-là vous aurez les nombres d'observations mais vous aurez perdu les noms des lignes (plantes). Il faut donc ajouter un argument à la fonction qui reçoit les noms des plantes et les ajouter au sein de votre fonction personnalisée. Vous pouvez ensuite modifier votre fonction de manière à ajouter par exemple une colonne de % d'observations (nb obs / sum (nb obs)) et à la fin le nombre total d'observations et le nombre total d'espèces de plantes.

```

class(plant)
plant <- as.data.frame.matrix(plant)
class(plant)

# avec le code suivant on a pas les noms de plantes
lapply(plant, function(x) {sort(x, decreasing = TRUE)[1:10]})

# fonction modifiée avec les noms de plantes comme argument
preferred_plants <- function(x, nmax = 10, plnames) {
  names(x) <- plnames
  as.matrix(sort(x, decreasing = TRUE)[1:nmax])
}

lapply(plant, preferred_plants, plnames = row.names(plant), nmax = 5)
lapply(plant, preferred_plants, plnames = row.names(plant), nmax = 10)

preferred_plants <- function(x, nmax = 10, plnames) {
  tot <- sum(x)
  nbtaxa <- sum(x>0)
  res <- data.frame(nbobs = x, pct = round(x*100/tot,1) )
  row.names(res) <- as.character(plnames)
  res <- res[order(res$nbobs, decreasing = TRUE),][1:nmax,]
  res <- rbind(res, c( "----", "----"), c( tot, "100"), c(nbtaxa, ""))
  row.names(res)[(nmax+1):(nmax+3)] <- c("----", "Total", "nb taxa")
  return(res)
}

lapply(plant, preferred_plants, plnames = row.names(plant))

```

- 16) On veut calculer les indices d'Ellenberg pour chaque espèce. Les indices d'Ellenberg sont des valeurs indicatrices (lumière, température, humidité, richesse du sol,...) pour une série d'espèces de plantes. On calcule en général un indice synthétique pour un site en multipliant l'abondance des plantes du site par leurs indices d'Ellenberg puis en prenant la somme de ces valeurs divisée par l'abondance totale (une valeur par indice). Il s'agit donc de la moyenne des indices des plantes présentes sur le site pondérée par leur abondance. On peut calculer également ces indices pour les biotopes occupés par des insectes.

- Commencez par charger le fichier Ellenberg.txt, ensuite, fusionnez les noms de lignes de votre tableau plantes x espèces avec le dataset Ellenberg (merge) afin que les lignes de ce dataset Ellenberg correspondent parfaitement aux lignes de votre tableau plantes x espèces.
- Calculez ensuite les indices d'Ellenberg pour sp\_1 uniquement. Vous pouvez multiplier la matrice d'Ellenberg par le vecteur sp\_1, diviser par le nombre total d'observations pour sp\_1 et faire la somme de chaque colonne. Attention cependant le nombre total d'observations pour sp\_1 ne devrait prendre en compte que les observations sur les plantes pour lesquelles l'indice d'Ellenberg est disponible. Le dénominateur est donc différent pour chaque indice d'Ellenberg.
- Généralisez ensuite ces calculs pour toutes les espèces (ie les colonnes du tableau plantes x espèces) par exemple au moyen de la fonction sapply.

```
setwd("/home/gilles/stats/Formation_R_stats/Formation_Stats_1_R/Exercices")
ellenberg <- read.table("data/ellenberg.txt", header=TRUE, sep="\t")
summary(ellenberg)
ellenberg[1:15,]

# Fusion des jeux de données
plantname <- data.frame(name = row.names(plant))
ellenberg <- merge(plantname, ellenberg, by.x = "name", by.y = "Taxprio_final",
                  all.x=TRUE)
ellenberg[1:10,]

# Calcul des indices d'Ellenberg pour une espèce
# On pourrait être tenté de faire le calcul suivant :
colSums(plant[,1] * ellenberg[,-1] / sum(plant[,1]), na.rm = TRUE)

# Avec le calcul ci-dessus, on ne tient pas compte du fait que certaines espèces de
# plantes n'ont pas de valeurs pour certains indices d'Ellenberg. Le dénominateur n'est
# donc pas bon on divise systématiquement par un nombre trop grand.
# On doit faire pour chaque indice la somme du nombre d'observations pour
# lesquelles il existe un indice d'ellenberg, comme ceci par exemple :
colSums(plant[,1] * ellenberg[,-1], na.rm = TRUE) /
  colSums(plant[,1] * (!is.na(ellenberg[,-1])))

# Calcul des indices d'Ellenberg pour toutes les espèces
result <- sapply(plant, function(x) {round(colSums(x * ellenberg[,-1], na.rm = TRUE) /
                                           colSums(x * (!is.na(ellenberg[,-1]))), 1)})

result
t(result)

# strictement équivalent ici :
result <- apply(plant, 2, function(x) {round(colSums(x * ellenberg[,-1], na.rm = TRUE) /
                                           colSums(x * (!is.na(ellenberg[,-1]))), 1)})
```