Fiche TD avec le logiciel R : bem1

Biologie et Modélisation - Introduction au logiciel R

D. Chessel, A.B. Dufour, S. Penel, & J.R. Lobry

La fiche donne des indications pratiques très simples pour accéder au logiciel de statistique R.

Contents

1 Utiliser R dans une salle de TP

Si vous redémarez une machine en salle de TP vous pourrez choisir entre un système d'exploitation windows ou Ubuntu.

- ⋆ Sous windows le logiciel R est, en principe, pré-installé, il vous suffit de lancer le programme. Vous aurez ce que l'on appelle l'interface utilisateur graphique qui offre quelques menus déroulants. Microsoft R Open (MRO) qui est une appropriation récente de R par la société à but lucratif vendant le système d'exploitation windows n'est pas encore disponible en salle de TP.
- ⋆ Sous Ubuntu vous pouvez utiliser de même cette interface utilisateur graphique. Vous pouvez également utiliser l'environnement de travail intégré RStudio qui aurait la réputation d'être plus convivial pour les débutants.

2 Installer une extension dans une salle de TP

2.1 Qu'est ce qu'une extension?

Paquet vient de l'anglais *package* et évoque un paquet-cadeau.

R dispose dans sa version de base de la plupart des outils nécessaires pour faire des statistiques. Ses capacités peuvent être augmentées en rajoutant des extensions ou paquets. Il y a aujourd'hui, 17 Mar 2025, 22203 paquets disponibles pour R.

2.2 Quelles sont les extensions disponibles?

Quand on vient d'installer R, on dispose de la version de base. Dans la console de travail, taper :

```
library()
```

On obtient une fenêtre de documentation avec la liste des paquets installés. S'il n'est pas installé, on pourra en disposer après une petite manipulation. Il faut simplement distinguer l'installation du paquet qui consiste à télécharger et installer définitivement les fichiers formant un paquet du chargement du paquet qui consiste à mettre en mémoire ces données pour une session de travail.

2.3 Installer un paquet : `install.packages()`

Vous n'avez pas le droit en salle de TD d'écrire dans le dossier où est présent le logiciel R. Vous devez l'installer dans un dossier qui vous est propre, par exemple dans le dossier de travail courant (donné par `getwd()`). Installer le paquet `tripack` dans le dossier de travail :

```
install.packages("tripack", lib = getwd())
```

2.4 Charger un paquet : `library()`

Dans tous les cas, pour la durée d'une session de travail, pour pouvoir utiliser un paquet, il faut le charger en mémoire.

En salle de TP, vous taperez les commandes suivantes :

- ⋆ si l'installation a déjà été faite dans le dossier où est chargé le logiciel :

```
library(tripack)
```

- ⋆ si vous venez de faire l'installation dans votre dossier de travail :

```
library(tripack, lib = getwd())
```

En récompense, vous avez un pavage de Voronoï sur un semis de points aléatoires (figure 2.4) avec les commandes suivantes :

```
n <- 100
x <- runif(n)
y <- runif(n)
mosa <- voronoi.mosaic(x, y, duplicate = "remove")
plot(mosa, xlim = c(0, 1), ylim = c(0, 1), main = "Mosaïque de Voronoï",
     sub = "100 points tirés au hasard dans le carré [0,1][0,1]")
points(x, y, pch = 20, cex = 1.5, col = "green3")
box()
```

3 Consulter la documentation

Consulter la fiche de documentation est une opération fondamentale ! Pour consulter la documentation associée à une fonction il suffit de faire précéder son nom d'un point d'interrogation. Par exemple, pour la fonction `read.table()` :

```
?read.table
```

```
read.table                package:base                R Documentation

Data Input
Description:
     Reads a file in table format and creates a data frame from it,
     with cases corresponding to lines and variables to fields in the
     file.
Usage:
     read.table(file, header = FALSE, sep = "", quote = "\"'", dec = ".",
                row.names, col.names, as.is = FALSE, na.strings = "NA",
                colClasses = NA, nrows = -1,
                skip = 0, check.names = TRUE, fill = !blank.lines.skip,
                strip.white = FALSE, blank.lines.skip = TRUE,
                comment.char = "#")
```

Dans chaque fiche de documentation, on trouvera systématiquement l'objet de la fonction, une description, l'ordre d'appel, la définition des paramètres et des exemples éventuels.

Exercice : la fonction `readLines()` permet également de lire des fichiers, par exemple :

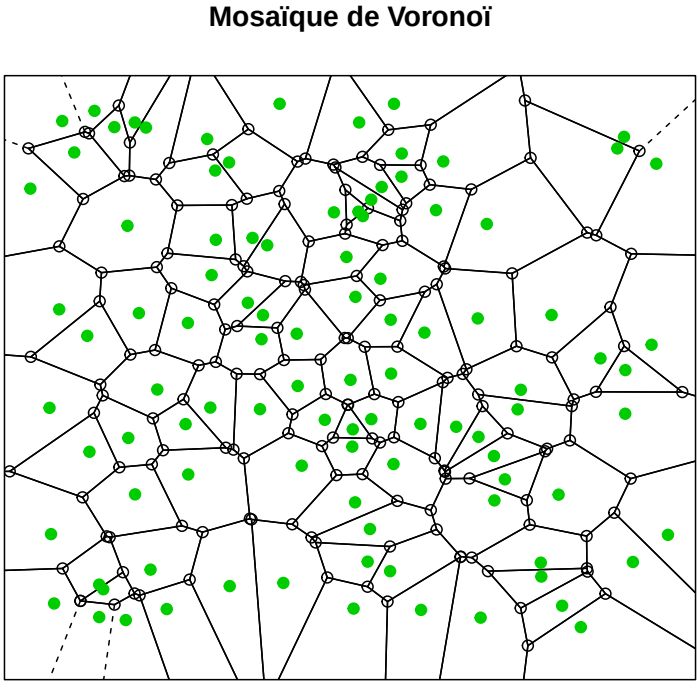


Figure 1: Pavage de Voronoï sur un semis de 100 points aléatoires effectués avec le paquet `tripack`.

```
readLines("http://pbil.univ-lyon1.fr/R/donnees/toto")
[1] "c'est toto"
```

Consultez la documentation de la fonction la fonction `readLines()` pour comprendre en quoi elle diffère de la fonction `read.table()`.

4 Objets

4.1 Logique sous R

Les trois valeurs logiques possibles dans R sont :

1. le vrai (`TRUE` que l'on peut abréger par `T`)
2. le faux (`FALSE` que l'on peut abréger par `F`)
3. l'indécidable (`NA`)

Pour faire une comparaison de deux termes, on utilise habituellement l'opérateur `==`

4.2 Affectation (<-, ->, =)

Mettre la valeur 7 dans l'objet de nom `x` :

```
x <- 7
```

Pour voir ce que contient un objet, il suffit d'entrer son nom :

```
x
[1] 7
```

On peut effectuer d'un coup l'affectation et l'affichage en entourant avec des parenthèses :

```
(x <- 7)
[1] 7
```

Exercice. Mettre la valeur 12 dans l'objet `x`, vous devez obtenir le résultat suivant :

```
x
[1] 12
```

L'affectation peut se faire avec la flèche dans l'autre sens (`7 -> x`). Cette construction est très pratique quand on a évalué une expression assez complexe que l'on rappelle avec la touche ↑. Il suffit alors de la compléter par `-> x` pour la sauvegarder dans l'objet `x`.

Exercice.

1. Calculez l'expression suivante :

   ```
   ((1+sqrt(5))/2)^2
   [1] 2.618034
   ```

2. Rappelez la commande avec la touche ↑ et sauvegardez le résultat dans `x`:

   ```
   ((1 + sqrt(5))/2)^2 -> x
   ```

3. Que vaut `x` ?

   ```
   x
   [1] 2.618034
   ```

Le signe égal (`=`) est parfois utilisé pour faire des affectations :

```
x = 7
```

Mais ceci ne fonctionnera pas dans tous les contextes, par exemple si l'affectation est elle-même encapsulée dans une fonction. Si on utilise la fonction `system.time()` pour évaluer le temps que prend une affectation, cela fonctionne parfaitement avec l'opérateur `<-` :

```
system.time(x<-7)
```

```
   user  system elapsed
      0       0       0
```

mais donnera une erreur avec l'opérateur `=` :

```
system.time(x = 7)
Error in system.time(x = 7) : unused argument(s) (x ...)
```

En effet, l'opérateur `=` est utilisé également pour donner une valeur aux paramètres d'une fonction lors de leur appel. Mais ici, il n'y a pas de paramètre `x` pour la fonction `system.time()`, d'où l'erreur produite. Pour éviter la confusion il est préférable de n'utiliser l'opérateur `=` que pour donner une valeur aux paramètres lors de l'appel d'une fonction.

4.3 Imprécision numérique (très important)

Cette section n'est pas spécifique de R mais concerne tous les langages de programmation dès que l'on cherche à comparer des valeurs numériques. À cause de la précision limitée des calculs, il y a forcément des erreurs d'arrondi, et il n'y a pas de sens à tester l'égalité de deux valeurs sans en tenir compte. Donnons un exemple concret. Soient x_1 et x_2 deux éléments de $\mathbb{R}$ tels que :

$$x_1 = \frac{5}{10} - \frac{3}{10}$$

$$x_2 = \frac{3}{10} - \frac{1}{10}$$

On peut dire que x_1 et x_2 sont égaux parce que

$$\begin{aligned} & x_1 = x_2 \\ \iff & \tfrac{5}{10} - \tfrac{3}{10} = \tfrac{3}{10} - \tfrac{1}{10} \\ \iff & \tfrac{5-3}{10} = \tfrac{3-1}{10} \\ \iff & \tfrac{2}{10} = \tfrac{2}{10} \end{aligned}$$

est une tautologie. Mais qu'en est-il en pratique ?

```
 (x1 <- 5/10 - 3/10)
[1] 0.2
 (x2 <- 3/10 - 1/10)
[1] 0.2
 x1 == x2
[1] FALSE
```

Ce n'est pas le résultat attendu ! La raison est qu'il y a toujours des erreurs d'arrondi, que l'on peut visualiser ici en augmentant le nombre de chiffres affichés après la virgule :

```
 format(x1, nsmall = 20)
[1] "0.20000000000000001110"
 format(x2, nsmall = 20)
[1] "0.19999999999999998335"
```

Heureusement R possède une fonction standard pour tester l'égalité de deux valeurs en tenant compte des imprécisions numériques : il faut utiliser la fonction `all.equal()`.

```
all.equal(x1, x2)
[1] TRUE
```

Cette fois c'est bien le résultat attendu.

4.4 Valeurs manquantes

Les valeurs manquantes sont représentées par `NA`. L'indétermination se propage *systématiquement* dans les calculs pour éviter de fausser le résultat final :

```
NA + 1
[1] NA
2*NA
[1] NA
```

La plupart des fonctions courantes possèdent un paramètre `na.rm` qui permet d'enlever les valeurs manquantes avant de faire le calcul demandé, mais il faut le faire explicitement, sa valeur par défaut est toujours `FALSE` pour éviter les catastrophes. Prenons la fonction `mean` qui permet de calculer la moyenne.

```
mean(c(pi,NA))
[1] NA
mean(c(pi,NA), na.rm = TRUE)
[1] 3.141593
```

Il y a de nombreuses fonctions utilitaires sous R pour aider à la bonne gestion des valeurs manquantes comme par exemple `is.na()` et `na.omit()`.

4.5 Les vecteurs

4.5.1 Intérêt du calcul sur les vecteurs

Dans R même une simple valeur numérique est un vecteur :

```
pi
[1] 3.141593
is.vector(pi)
[1] TRUE
```

Prenons la définition de la variance d'un échantillon de taille n d'une variable continue X :

$$\text{var}\,(\text{X}) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Dans un langage de programmation classique nous écririons quelque chose du type :

```
variance <- function(X)
{
  n <- length(X)
  sum1 <- 0
  for( i in seq(from = 1, to = n))
    {sum1 <- sum1 + X[i]}
  moyenne <- sum1/n
  sum2 <- 0
  for( i in seq(from = 1, to = n))
    {sum2 <- sum2 + (X[i] - moyenne)*(X[i] - moyenne)}
  return(sum2/n)
}
```

Prenons un échantillon constitué des 10 premiers entiers. Il s'écrit :

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

Et la variance de cet échantillon est obtenue par :

```
variance(1:10)
[1] 8.25
```

Mais dans R, nous pouvons manipuler directement les vecteurs de la façon suivante :

```
variance <- function(X)
{
  n <- length(X)
  moyenne <- sum(X)/n
  return(sum((X-moyenne)^2)/n)
}
variance(1:10)
[1] 8.25
```

Les expressions sont ainsi beaucoup plus compactes et proches des notations mathématiques.

***Exercice* (vous n'avez pas besoin de définir de nouvelle fonction pour cet exercice, il suffit d'utiliser les fonctions déjà existantes)**

1. Donner le carré des 10 premiers entiers moins un :

   ```
   [1]  0  3  8 15 24 35 48 63 80 99
   ```

2. Donner le sinus (`sin()`) des 10 premiers entiers :

   ```
   [1]  0.8414710  0.9092974  0.1411200 -0.7568025 -0.9589243 -0.2794155  0.6569866
   [8]  0.9893582  0.4121185 -0.5440211
   ```

3. Donner 10 à la puissance des 10 premiers entiers :

   ```
   [1] 1e+01 1e+02 1e+03 1e+04 1e+05 1e+06 1e+07 1e+08 1e+09 1e+10
   ```

4. Donner le logarithme népérien (`log()`) de 10 à la puissance des 10 premiers entiers :

   ```
   [1]  2.302585  4.605170  6.907755  9.210340 11.512925 13.815511 16.118096 18.420681
   [9] 20.723266 23.025851
   ```

5. Donner la racine carré (`sqrt()`) des 10 premiers entiers :

   ```
    [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000
   [10] 3.162278
   ```

6. Donner la somme (`sum()`) des 10 premiers entiers :

   ```
   [1] 55
   ```

7. Donner la somme cumulée (`cumsum()`) des 10 premiers entiers :

```
 [1]  1  3  6 10 15 21 28 36 45 55
```

8. Donner le produit (`prod()`) des 10 premiers entiers :

```
[1] 3628800
```

9. Donner les différences (`diff()`) des 10 premiers entiers avec leur précédent :

```
[1] 1 1 1 1 1 1 1 1 1
```

4.5.2 Statistiques élémentaires

Lire la documentation des fonctions pour comprendre ce qu'elles calculent.

Considérons les notes de 14 étudiants d'un groupe de TP et calculons les paramètres statistiques élémentaires. L'écriture d'un vecteur est une combinaison (cf le paragraphe sur les séries de valeurs numériques ci-après).

```
notes <- c(15,8,14,12,14,10,18,15,9,5,12,13,12,16)
sort(notes)
 [1]  5  8  9 10 12 12 12 13 14 14 15 15 16 18
length(notes)
[1] 14
min(notes)
[1] 5
max(notes)
[1] 18
range(notes)
[1]  5 18
median(notes)
[1] 12.5
quantile(notes)
   0%   25%   50%   75%  100%
 5.00 10.50 12.50 14.75 18.00
mean(notes)
[1] 12.35714
var(notes)
[1] 11.93956
sd(notes)
[1] 3.455367
unique(notes)
 [1] 15  8 14 12 10 18  9  5 13 16
sort(unique(notes))
 [1]  5  8  9 10 12 13 14 15 16 18
```

4.5.3 Séries de valeurs numériques

Séries d'entiers : ":"

L'opérateur deux points : permet de générer des séries d'entiers :

```
x <-1:12
```

Exercice. Générer la série suivante :

```
[1] -5 -4 -3 -2 -1  0  1  2  3  4  5
```

Séries de valeurs numériques : `seq()`

La fonction `seq()` permet de générer une série de nombres équidistants :

```
 seq(from = 1, to = 5)
[1] 1 2 3 4 5
 seq(from = 1, to = 2, by = 0.1)
 [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
 seq(from = 10, to = 50, length = 10)
 [1] 10.00000 14.44444 18.88889 23.33333 27.77778 32.22222 36.66667 41.11111 45.55556
[10] 50.00000
 seq(along = letters)
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
```

Exercice. Générer les séries suivantes :

```
[1]  0 -1 -2 -3 -4 -5
 [1] 0.01 0.02 0.03 0.04 0.05 0.06 0.07 0.08 0.09 0.10
[1] 1.000000 1.666667 2.333333 3.000000
```

4.5.4 Répétition de valeurs : `rep()`

```
 rep(1:5, 2)
 [1] 1 2 3 4 5 1 2 3 4 5
 rep(1:5, length = 12)
 [1] 1 2 3 4 5 1 2 3 4 5 1 2
 rep(1:5, each = 2)
 [1] 1 1 2 2 3 3 4 4 5 5
 rep(c('un', 'deux'), c(6, 3))
[1] "un"   "un"   "un"   "un"   "un"   "un"   "deux" "deux" "deux"
```

Exercice. Générer les séries suivantes :

```
[1] 1 2 3 1 2 3 1 2 3
 [1] 1 2 3 4 1 2 3 4 1 2 3 4
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
[1] "un"   "un"   "un"   "deux" "deux" "deux" "deux" "deux" "deux"
```

4.5.5 Combinaison : `c()`

```
 x <- c(1.5,7.2,pi)
 x <-c(1:3,10:12)
```

Exercice. Construire les vecteurs suivants :

```
[1] 11.1  2.7  3.3
[1] -5.00 -4.00 -3.00 -2.00 -1.00  0.00  1.00  1.10 88.77
```

Remarque :

```
 x <- c(2,5,-3,8,"a")
```

`x` est un vecteur de chaînes de caractères.

4.5.6 Mode : `mode()`

Reprenons le vecteur `x` du paragraphe précédent.

```
mode(x)
```
```
[1] "character"
```

`x` est un vecteur de chaînes de caractères.

```
x <- c(1,5,-36,3.66)
mode(x)
```
```
[1] "numeric"
```

`x` est un vecteur numérique.

```
x <- c(T,T,T,F,F)
mode(x)
```
```
[1] "logical"
```

`x` est un vecteur logique.

4.5.7 Éléments des vecteurs

Indexation par des entiers positifs

Définissons un vecteur `x`.

```
x <- c(145, -1, 28.88, 0.02, 34.5)
x
```
```
[1] 145.00  -1.00  28.88   0.02  34.50
```

Le quatrième élément :

```
x[4]
```
```
[1] 0.02
```

Du second au troisième élément :

```
x[2:3]
```
```
[1] -1.00 28.88
```

On peut reprendre plusieurs fois le même :

```
x[c(2,2,3)]
```
```
[1] -1.00 -1.00 28.88
```

Les éléments hors bornes ne sont pas disponibles (`NA`, not available, donnée manquante) :

```
x[100]
```
```
[1] NA
```

Indexation par des entiers négatifs

Tous sauf le quatrième :

```
 x[-4]
[1] 145.00  -1.00  28.88  34.50
```

Exercice 1. Donner tous les éléments sauf le premier.

```
[1] -1.00 28.88  0.02 34.50
```

Exercice 2. A l'aide de la fonction `length()`, donner tous les éléments sauf le dernier :

```
[1] 145.00  -1.00  28.88   0.02
```

On ne peut pas mélanger les indexations par des entiers positifs et négatifs simultanément. Il faut procéder en deux temps.

Exercice 3. Donner le deuxième et le troisième élément une fois que l'on a enlevé le premier élément :

```
[1] 28.88  0.02
```

Indexation par un vecteur logique

```
 x[c(T,F,F,T,T)]
[1] 145.00   0.02  34.50
 x[c(T,F)]
[1] 145.00  28.88  34.50
```

Si le vecteur logique n'est pas assez long il sera recyclé autant de fois que nécessaire.

Exercice. Donner un élément sur deux à partir du deuxième :

```
[1] -1.00  0.02
```

Le vecteur logique d'indexation peut être issu d'un calcul logique, c'est l'utilisation la plus courante :

```
 x > 10
[1]  TRUE FALSE  TRUE FALSE  TRUE
 x[x > 10]
[1] 145.00  28.88  34.50
 x > 0 & x < 1
[1] FALSE FALSE FALSE  TRUE FALSE
 x[x > 0 & x < 1]
[1] 0.02
```

Exercice. Donner la liste des éléments compris entre 10 et 50 :

```
[1] 28.88 34.50
```

Indexation par des noms

Ceci ne fonctionne que si les éléments du vecteur ont un nom. Donnons comme nom les 5 premières lettres de l'alphabet aux éléments du vecteur `x` :

```
 (names(x) <- letters[1:5])
[1] "a" "b" "c" "d" "e"
```

Quelles sont les valeurs des éléments `a` et `e` ?

```
 x[c("a","e")]
    a     e
145.0  34.5
```

Pour exclure des éléments par leur nom, il faut d'abord récupérer leurs indices avec la fonction `match()`. Quelles sont les valeurs des éléments autre que `a` et `e` ?

```
 x[-match(c("a","e"), names(x))]
    b     c     d
-1.00 28.88  0.02
```

Exercice. Quelles sont les consonnes ?

```
 [1] "b" "c" "d" "f" "g" "h" "j" "k" "l" "m" "n" "p" "q" "r" "s" "t" "v" "w" "x" "z"
```

4.6 Les facteurs : `factor()`

Les modalités : `levels()`

Les facteurs sont la représentation sous R des *variables qualitatives* (la couleur des yeux, le genre (♂, ♀), un niveau de douleur, etc). De telles données sont souvent codées numériquement, mais doivent être converties avant d'être utilisées.

```
 douleur <- c(0, 3, 2, 2, 1)
 fdouleur <- factor(douleur, levels = 0:3)
 is.numeric(fdouleur)
[1] FALSE
 is.character(fdouleur)
[1] FALSE
 is.factor(fdouleur)
[1] TRUE
 summary(fdouleur)
0 1 2 3
1 1 2 1
 table(fdouleur)
fdouleur
0 1 2 3
1 1 2 1
```

La fonction `levels()` permet de récupérer ou de modifier les modalités des variables qualitatives, la fonction `as.numeric()` donne le codage numérique des modalités :

```
 levels(fdouleur)
[1] "0" "1" "2" "3"
 levels(fdouleur) <- c("rien", "leger", "moyen", "fort")
 fdouleur
[1] rien  fort  moyen moyen leger
Levels: rien leger moyen fort
 levels(fdouleur)
[1] "rien"  "leger" "moyen" "fort"
 as.numeric(fdouleur)
[1] 1 4 3 3 2
```

4.7 Les listes

4.7.1 Création d'une liste : `list()`

Les listes sont une structure de données très flexible et très utilisée dans R. Un élément d'une liste est un objet R *quelconque*, y compris une autre liste. La fonction `list()` permet de créer des listes :

```
maliste <- list(a = pi, b = "une chaine", c = c(T,F,NA))
maliste
$a
[1] 3.141593
$b
[1] "une chaine"

$c
[1]  TRUE FALSE    NA
```

4.7.2 Extraction d'un élément d'une liste : `$` et `[[`

Un élément d'une liste est en général extrait par son nom (avec l'opérateur `$`), mais on peut aussi utiliser sa position dans la liste avec l'opérateur `[[`:

```
maliste$a
[1] 3.141593
maliste[[1]]
[1] 3.141593
```

Exercice.

1. Donner l'élément `b` de la liste :

   ```
   [1] "une chaine"
   ```

2. Donner le troisième élément de la liste :

   ```
   [1]  TRUE FALSE    NA
   ```

4.7.3 Ajout d'un élément dans une liste : `$`

```
maliste$d <-1:10
maliste
$a
[1] 3.141593
$b
[1] "une chaine"

$c
[1]  TRUE FALSE    NA

$d
 [1]  1  2  3  4  5  6  7  8  9 10
```

4.7.4 Sélection d'éléments d'une liste : `[`

L'opérateur `[` sur les listes ne fait pas d'extraction mais renvoie une liste avec les éléments sélectionnés. Par exemple, pour supprimer le premier élément de la liste :

```
maliste <- maliste[-1]
maliste
```

```
$b
[1] "une chaine"
$c
[1]  TRUE FALSE    NA

$d
 [1]  1  2  3  4  5  6  7  8  9 10
```

5 Les tableaux : `data frames`

5.1 Les objets de type `data.frame`

Les data frames sont des listes dont tous les éléments ont la même longueur. On peut mélanger dans un `data.frame` des variables quantitatives, qualitatives, logiques et textuelles[1].

1. Créer dans un tableur un fichier en entrant sur trois colonnes (`sex`, `poi`, `tai`) les données ci-dessous. Vous pouvez vous contenter des **cinq premières lignes**, nous importerons plus tard l'intégralité des données de façon plus directe.

	A	B	C
1	sexe	poi	tai
2	h	60	170
3	f	57	169
4	f	51	172
5	f	55	174
6	f	50	168
7	f	50	161
8	f	48	162
9	h	72	189
10	f	52	160
11	h	64	175
12	f	53	165
13	h	72	164
14	h	61	175
15	h	78	184
16	h	68	178
17	f	51	158
18	f	53	164
19	h	79	179
20	h	74	182
21	h	62	174
22	f	49	158

	A	B	C
23	f	50	163
24	h	74	172
25	h	80	185
26	f	53	170
27	h	73	178
28	h	70	180
29	h	72	189
30	f	70	172
31	f	62	174
32	h	77	200
33	h	70	178
34	h	76	178
35	f	51	168
36	f	52	170
37	f	57	160
38	f	53	163
39	f	55	168
40	f	66	172
41	h	65	175
42	h	75	180
43	f	50	162
44	f	53	177

	A	B	C
45	h	55	169
46	h	55	173
47	h	72	182
48	h	75	183
49	h	73	184
50	h	71	181
51	h	66	180
52	h	71	178
53	h	79	178
54	h	62	168
55	f	47	161
56	h	73	171
57	h	72	180
58	h	60	174
59	h	67	175
60	h	85	182
61	h	73	181
62	h	82	188
63	h	86	182
64	h	85	189
65	h	65	178
66	f	47	150
67	h	74	186

2. Quand c'est fini, sauvegarder les données dans un fichier au format texte en utilisant des tabulations comme séparateur.

3. Retourner dans R pour importer les données.

[1] c'est la différence avec les matrices qui elles ne peuvent contenir qu'un seul type de données

Observer d'abord ce qui se passe quand on lit les 5 premières lignes du tableau.

```
read.table("t3var.txt")[1:5,]
   V1  V2  V3
1 sexe poi tai
2    h  60 170
3    f  57 169
4    f  51 172
5    f  55 174
```

Le premier individu contient les en-têtes ! Rajouter une valeur du paramètre `header`. Quand le résultat semble correct, placer le résultat de cette lecture dans un objet. On dit qu'on fait une affectation.

```
read.table("t3var.txt", header=T)[1:5,]
t3var <- read.table("t3var.txt", h=T)
read.table("t3var.txt", h=T) -> t3var
```

Dans le cas précédent, le fichier à charger dans R était sauvegardé dans un répertoire local. Il est également possible de charger des données sauvegardées sur un répertoire distant et notamment sur un site internet. Un grand nombre de jeux de données sont disponibles sur le site

http://pbil.univ-lyon1.fr/R/donnees/

Visiter ce dossier, repérer le fichier `t3var`, l'importer à la main dans son dossier de travail, l'afficher dans le navigateur, l'ouvrir avec un éditeur, le télécharger directement dans son dossier. On peut même le lire directement par :

```
read.table("http://pbil.univ-lyon1.fr/R/donnees/t3var.txt",h=T)
read.table("http://pbil.univ-lyon1.fr/R/donnees/t3var.txt",h=T) -> t3var
```

Utiliser le fichier importé pour la suite.

5.2 Éléments des data frames : `$` et `tab[i,j]`

Les data frames étant des listes, les variables (en colonne) sont directement accessibles par leur nom :

```
t3var$tai
 [1] 170 169 172 174 168 161 162 189 160 175 165 164 175 184 178 158 164 179 182 174
[21] 158 163 172 185 170 178 180 189 172 174 200 178 178 168 170 160 163 168 172 175
[41] 180 162 177 169 173 182 183 184 181 180 178 178 168 161 171 180 174 175 182 181
[61] 188 182 189 178 150 186
```

Exercice.

1. Donner la moyenne des tailles :

   ```
   [1] 174.0606
   ```

2. Donner la médiane des poids :

   ```
   [1] 65.5
   ```

3. Donner la variance des poids :

   ```
   [1] 123.5767
   ```

4. Donner l'écart-type des poids :

```
[1] 11.11651
```

On peut également utiliser la notation de type `tab[i,j]` où `i` représente l'index des lignes et `j` celui des colonnes. Par exemple, pour avoir les 5 premiers individus :

```
t3var[1:5,]
```

```
  sexe poi tai
1    h  60 170
2    f  57 169
3    f  51 172
4    f  55 174
5    f  50 168
```

Pour avoir les première et troisième colonnes des 5 premiers individus :

```
t3var[1:5, c(1,3)]
```

```
  sexe tai
1    h 170
2    f 169
3    f 172
4    f 174
5    f 168
```

Tous les types d'indexation vu pour les vecteurs (entiers positifs, négatifs, logiques, noms) sont utilisables.

Exercice.

1. Donner les individus 1, 5 et 55 :

```
   sexe poi tai
1     h  60 170
5     f  50 168
55    h  73 171
```

2. Pour les 10 premiers individus, donner toutes les variables sauf la première :

```
   poi tai
1   60 170
2   57 169
3   51 172
4   55 174
5   50 168
6   50 161
7   48 162
8   72 189
9   52 160
10  64 175
```

3. Extraire le sexe de tous les individus :

```
 [1] "h" "f" "f" "f" "f" "f" "f" "h" "f" "h" "f" "h" "h" "h" "h" "f" "f" "h" "h" "h"
[21] "f" "f" "h" "h" "f" "h" "h" "h" "f" "f" "h" "h" "h" "f" "f" "f" "f" "f" "f" "h"
[41] "h" "f" "f" "h" "h" "h" "h" "h" "h" "h" "h" "h" "h" "f" "h" "h" "h" "h" "h" "h"
[61] "h" "h" "h" "h" "f" "h"
```

4. Tester si le sexe de tous les individus est féminin :

```
 [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE  TRUE FALSE  TRUE FALSE FALSE
[14] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE  TRUE FALSE
[27] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[40] FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[53] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE
[66] FALSE
```

5. Donner tous les individus de sexe féminin :

```
   sexe poi tai
2     f  57 169
3     f  51 172
4     f  55 174
5     f  50 168
6     f  50 161
7     f  48 162
9     f  52 160
11    f  53 165
16    f  51 158
17    f  53 164
21    f  49 158
22    f  50 163
25    f  53 170
29    f  70 172
30    f  62 174
34    f  51 168
35    f  52 170
36    f  57 160
37    f  53 163
38    f  55 168
39    f  66 172
42    f  50 162
43    f  53 177
54    f  47 161
65    f  47 150
```

6. Donner tous les individus qui font plus de 180 cm :

```
   sexe poi tai
8     h  72 189
14    h  78 184
19    h  74 182
24    h  80 185
28    h  72 189
31    h  77 200
46    h  72 182
47    h  75 183
48    h  73 184
49    h  71 181
59    h  85 182
60    h  73 181
61    h  82 188
62    h  86 182
63    h  85 189
66    h  74 186
```

7. Donner tous les individus de sexe féminin qui font plus de 175 cm :

```
   sexe poi tai
43    f  53 177
```

8. Donner le poids et la taille tous les individus de sexe féminin qui font plus de 175 cm :

```
   poi tai
43  53 177
```

5.3 Sélection des lignes : `subset()`

Il est très courant de sélectionner les individus (en ligne) en fonction de critères calculés sur les variables (en colonne). Pour éviter d'expliciter à chaque fois le nom du `data.frame` pour désigner une variable, on peut avantageusement utiliser la fonction `subset()`. Ainsi, la sélection des individus de sexe féminin qui font plus de 175 cm s'écrit aussi :

```
subset(t3var, sexe == "f" & tai > 175)
```

```
   sexe poi tai
43    f  53 177
```

6 Utiliser des scripts

Tant que ce l'on demande de faire à R tient sur quelques lignes de commande, l'utilisation de la flêche-haut pour rappeler les commandes antérieures pour les modifier est très puissant. Mais dès qu'il y a plusieurs lignes à exécuter dans un ordre déterminé cela peut devenir fastidieux. Heureusement on peut faire des scripts, c'est à dire un fichier contenant une suite de commandes. Créez un nouveau script et collez-y les commandes permettant de faire le pavage de Voronoï sur un semis de points aléatoire :

```
n <- 100
x <- runif(n)
y <- runif(n)
mosa <- voronoi.mosaic(x, y, duplicate = "remove")
plot(mosa, xlim = c(0, 1), ylim = c(0, 1), main = "Mosaïque de Voronoï",
     sub = "100 points tirés au hasard dans le carré [0,1][0,1]")
points(x, y, pch = 20, cex = 1.5, col = "green3")
box()
```

Exercice. Exécutez le script plusieurs fois : vous devez constater que le graphique change à chaque fois parce que les coordonnées des points sont tirées au hasard. Modifiez la valeur de n de 100 à 50. Exécutez le script plusieurs fois. Modifiez `green3` en `red`. Exécutez le script plusieurs fois. Modifiez `cex = 1.5` en `cex = 0.5`. Exécutez le script plusieurs fois[2].

7 Conserver les résultats

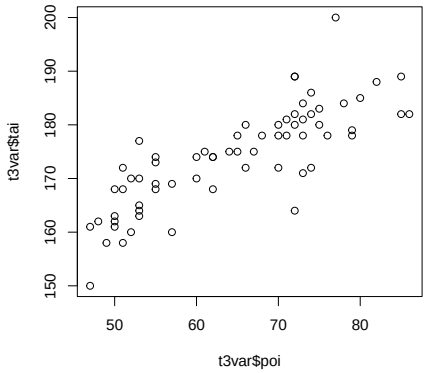
Ouvrir un fichier vierge dans votre traitement de texte favori, définir un style listing avec la police `Courier New` pour formater les lignes de commande.

```
summary(t3var)
```

```
    sexe                poi              tai       
Length:66          Min.   :47.00   Min.   :150.0  
Class :character   1st Qu.:53.00   1st Qu.:168.0  
Mode  :character   Median :65.50   Median :174.5  
                   Mean   :64.52   Mean   :174.1  
                   3rd Qu.:73.00   3rd Qu.:180.0  
                   Max.   :86.00   Max.   :200.0  
```

```
plot(t3var$poi,t3var$tai)
```

[2] Il existe un racourci-clavier pour cela, trouvez le.



Copier les commandes (sélection à la souris) et les figures (cliquer dessus avec le bouton droit) et coller dans votre document de traitement de texte. Vous savez maintenant comment faire un rapport[3].

[3] Il existe des outils puissants pour automatiser la production de rapports, pour en savoir plus voir la fiche `http://pbil.univ-lyon1.fr/R/fichestd/tdr78.pdf`.