Séance 1 : Introduction à R

Laurent Tournier

12 septembre 2022

Table des matières

1 Introduction aux statistiques descriptives

Les statistiques descriptives visent à décrire un ensemble, en général important, de données, c'est-à-dire à en résumer certaines particularités, sous la forme de représentations graphiques ou de grandeurs numériques. L'interprétation des résultats est ensuite propre à chaque champ d'application. Cette partie des statistiques

n'utilise pas de probabilités, à la différence des statistiques mathématiques, ou inférentielles, qui supposent que les données sont des réalisations d'un variable aléatoire dont on cherche à mieux connaître la loi.

1.1 Vocabulaire, représentations simples

En statistiques, les données dont on dispose associent à chaque *individu* d'une certain ensemble, appelé la *population*, une ou plusieurs *variables* qui quantifient ou qualifient certains caractères des individus (remarque : les individus peuvent être des personnes, mais aussi des objets, des dates, des lieux...). Ces données sont aussi appelées une *série statistique*.

Pour toute série statistique, il est important de bien connaître la façon dont les individus ont été choisis (liste exhaustive, choix aléatoire selon une certaine méthode, choix selon un critère, précisions de lieu et période de temps...), et ce que représentent précisément les variables (mode de détermination, unité,...). Une grande précision est ici indispensable pour pouvoir interpréter les données.

On dispose de données $(x_1, y_1, \ldots), (x_2, y_2, \ldots), \ldots, (x_n, y_n, \ldots)$ où n est l'*effectif total* (taille de la population), et $(x_i, y_i, \ldots)$ sont les *observations* des variables associées au i-ième individu.

Une variable à valeurs numériques est dite *quantitative*. Dans le cas contraire, elle est dite *qualitative*, c'est-à-dire que ses valeurs ne sont pas des nombres : par exemple, des chaînes de caractères (nom, adresse,...), des couleurs, ou des pays.

Si une variable qualitative prend ses valeurs dans un ensemble fini, et prédéfini, elle est dite *modale* (ou *catégorielle* ou *factorielle*) et les valeurs possibles sont appelées les *modalités* (ou *catégories*, ou *facteurs*) de la variable. S'il n'y a que deux valeurs, on parle de variables *binaires*. Si les modalités sont ordonnées, on parle de variables *ordinales*, et de variables *nominales* sinon.

Une variable quantitative est *discrète* si ses valeurs possibles sont restreintes à un ensemble fini, et elle est *continue* sinon. Une variable quantitative discrète peut d'ailleurs se voir comme une variable qualitative.

Pour une variable qualitative (ou quantitative discrète),

- l'*effectif* d'une modalité (ou d'une valeur) est le nombre de fois où elle est présente dans la population. On représente graphiquement les effectifs par un *diagramme en bâtons* ou *histogramme*.
- la *fréquence* d'une modalité (ou d'une valeur) est le quotient de l'effectif de cette valeur par l'effectif total. On représente graphiquement les fréquences par un *diagramme circulaire*, ou par un diagramme en bâtons (éventuellement empilés).

Pour représenter graphiquement des observations une variable quantitative continue, un *nuage de points* consiste à dessiner un point par donnée, dont la position sur un axe est donnée par la valeur de la variable ; le principe s'étend en 2D ou 3D pour des données bidimensionnelles voire tridimensionnelles. Dès que le nombre de points ne permet pas de les compter visuellement, on choisit plutôt un nombre fini d'intervalles, on classifie les individus selon l'intervalle qui contient leur valeur et on représente l'histogramme des effectifs ou des fréquences associé : ainsi, on approche la variable continue par une variable discrète. Attention, le choix des intervalles est essentiel et ne suit pas une règle générale.

1.2 Statistiques usuelles

Une première approche pour décrire un ensemble de données consiste à calculer certaines statistiques.

Supposons que l'on dispose d'observations $x_1, \ldots, x_n$ d'une variable quantitative. La *fonction de répartition empirique*

$$F(x) = \frac{1}{n}\mathrm{Card}\{i \in \{1, \ldots, n\} \,|\, x_i \leq x\}.$$

donne, pour tout x, la proportion d'observations inférieures à x.

Pour $\alpha \in [0, 1]$, un *quantile d'ordre* α est un réel q_α tel qu'une proportion $\geq \alpha$ des données est dans $]-\infty, q_\alpha]$, et une proportion $\geq 1-\alpha$ des données est dans $[q_\alpha, +\infty[$. Autrement dit q_α est tel que $F\big((q_\alpha)^-\big) < \alpha \leq F(q_\alpha)$ (où $F(x^-) = \frac{1}{n}\mathrm{Card}\{i \in \{1, \ldots, n\} \,|\, x_i < x\}$).

En pratique, pour $0 < \alpha < 1$, si on ordonne les données $x_{(1)} \leq \cdots \leq x_{(n)}$,

— si $\alpha n \notin \mathbb{N}$, et $k - 1 < \alpha n < k$ avec $k \in \mathbb{N}$, alors $x_{(k)}$ est l'unique quantile d'ordre α ;
— si $\alpha n = k \in \mathbb{N}^*$, les quantiles d'ordre α sont les éléments de $[x_{(k)}, x_{(k+1)}]$.

Certains sont d'usage fréquent :

— Une *médiane* est un quantile d'ordre 1/2 : elle sépare les données en deux parties égales.
— Un *premier quartile* est un quantile d'ordre 1/4.
— Un *troisième quartile* est un quantile d'ordre 3/4.
— Le *minimum* est le plus grand quantile d'ordre 0, et le *maximum* est le plus petit d'ordre 1.

On peut représenter ces valeurs par une *boîte de dispersion* (ou « boîte à moustaches »).

Si $x_1, \ldots, x_n$ sont les observations d'une variable quantitative, la *moyenne* de cette série statistique est

$$\overline{x} = \frac{x_1 + \cdots + x_n}{n}.$$

Pour une variable quantitative discrète, de valeurs $v_1, \ldots, v_r$, ayant pour effectifs $n_1, \ldots, n_r$ et fréquences $f_1, \ldots, f_r$, on a aussi

$$\overline{x} = \frac{n_1 v_1 + \cdots + n_r v_r}{n} = f_1 v_1 + \cdots + f_r v_r.$$

Moyenne et médiane sont des valeurs qui « approchent » au mieux toutes les données :

— La moyenne est l'unique réel c qui minimise $\sum_{i=1}^n (x_i - c)^2$.
— Les médianes sont les réels c qui minimisent $\sum_{i=1}^n |x_i - c|$.

Remarquons que la moyenne est sensible aux valeurs aberrantes, à la différence de la médiane.

— La *variance* de la série statistique est

$$s_x^2 = \frac{1}{n}\sum_{i=1}^n (x_i - \overline{x})^2 = \frac{1}{n}\sum_{i=1}^n x_i^2 - \overline{x}^2 \quad (= \overline{x^2} - \overline{x}^2) \geq 0,$$

et son *écart type* est $\sigma_x = \sqrt{s_x^2}$. C'est une mesure de la dispersion des valeurs autour de la moyenne : si le quotient $\frac{\sigma_x}{|\overline{x}|}$ est petit (et $\overline{x} \neq 0$), les données sont concentrées autour de $\overline{x}$.

On suppose que l'on observe deux variables pour chaque individu, si bien que l'on dispose de données $(x_1, y_1), \ldots, (x_n, y_n)$.

— La *covariance* des deux variables est

$$s_{x,y} = \frac{1}{n}\sum_{i=1}^n (x_i - \overline{x})(y_i - \overline{y}) = \frac{1}{n}\sum_{i=1}^n x_i y_i - \overline{x}\,\overline{y} \quad (= \overline{xy} - \overline{x}\,\overline{y}),$$

(donc $s_x^2 = s_{x,x}$)
— Leur *corrélation* est $\rho_{x,y} = \frac{s_{x,y}}{\sigma_x \sigma_y} \in [-1, 1]$. Une corrélation positive (et proche de 1) indique que les variables ont tendance à être simultanément grandes, ou simultanément petites, tandis qu'une corrélation négative (et proche de -1) indique des variations en sens opposés.

2 Introduction à R

R est un langage de programmation et un logiciel libre destiné à l'analyse des données et leur visualisation. C'est le plus utilisé dans la communauté statistique académique, mais il est aussi de plus en plus utilisé en entreprise à la place de logiciels commerciaux.

Ce logiciel est maintenu et enrichi par une large communauté autour du R Project et du site d'archive CRAN qui fournit la dernière version et inventorie toutes les extensions (“packages”) mis à disposition.

2.1 Interaction avec R

R est, comme Python ou Matlab/Octave, un langage interprété : il n'est pas nécessaire de compiler les programmes, ceux-ci sont lus et interprétés ligne-à-ligne.

On peut l'utiliser en ligne de commande (via la commande `R`, utilisée en mode interactif ou en lui passant les fichiers selon la syntaxe `R CMD BATCH infile outfile`), ou aussi via diverses interfaces graphiques qui facilitent l'interaction avec R, comme par exemple RStudio (www.rstudio.com), que l'on utilisera durant ce cours.

Dans RStudio, plusieurs sous-fenêtres se présentent, d'une façon similaire à Matlab/Octave :

- éditeur de code
- console : interaction directe avec R
- environnement : liste des variables en mémoire vive
- historique des commandes envoyées à la console
- fichier d'aide
- graphiques
- liste des packages disponibles/installés

Leur disposition est amplement paramétrable.

2.2 Fichiers `.R` et `.Rmd`

R script. Les programmes en R sont par convention enregistrés avec une extension `.R`.

Dans le programme, on peut inclure des commentaires (du texte non interprété) en les précédant d'un `#` ; ceci permet d'expliquer le contenu du code. NB. En R, il n'y a pas de commande pour commenter plusieurs lignes à la fois.

Raccourci RStudio : Ctrl-Maj-Entrée pour exécuter le programme

R markdown On peut aussi mêler texte et programmes en R dans des fichiers "R Markdown", dont l'extention est `.Rmd`.

Ce sont des fichiers texte brut avec une syntaxe simple pour faire une mise en forme sommaire comme

- de l'*italique* : `*italique*`
- du **gras** : `**gras**`
- des sections : `# Section`
- des listes
- des équations $e^{i\pi}+1$: `e^{i\pi}+1`
- etc.,

(cf. Menu "Help/Markdown quick reference")

mais aussi inclure des cellules (*chunks*) de code R (ou autre) :

```
2+2
```

```
## [1] 4
```

```
x <- rnorm(1)
x = rnorm(1)
x
```

```
## [1] -0.1623783
```

```
x =2
exp(2)
```

```
## [1] 7.389056
```

Raccourci RStudio : Ctrl-Alt-i pour créer une cellule de code ; Ctrl-Maj-Entrée pour l'envoyer vers la console (ou la flèche verte juste au-dessus). Le résultat apparaît alors juste en-dessous.

Le fichier R Markdown peut être converti dans divers formats (pdf, html, word) via le bouton "Knit" de RStudio, ou Ctrl-Maj-k.

C'est donc une façon pratique de simultanément réaliser une analyse de donnée et de produire un document présentant les résultats.

On peut vouloir relancer toutes les cellules (Run All) ; dans ce cas, si une des cellules est longue ou inutile à relancer (chargement de données), on peut préciser `cache = T` dans l'en-tête pour éviter de la relancer à moins que le code ait été changé depuis la dernière exécution.

```
df <- read.csv(file='données.csv')
```

2.3 Environnement

À chaque fois que l'on exécute un code R dans une session RStudio, on travaille dans un même environnement : les variables restent en mémoire (et apparaisssent dans l'interface), les packages restent chargés,... même si on change de fichier. Quand on quitte RStudio, une fenêtre demande si on souhaite sauvegarder l'environnement courant, ce qui permet de reprendre la session telle quelle une fois prochaine.

Inversement, on peut vouloir effacer tous les objets en mémoire.

```
# Dans le fichier principal, un bon réflexe est d'inclure cette ligne au début.
rm(list=ls())
# Elle efface la liste de tous les objets présents en mémoire.
```

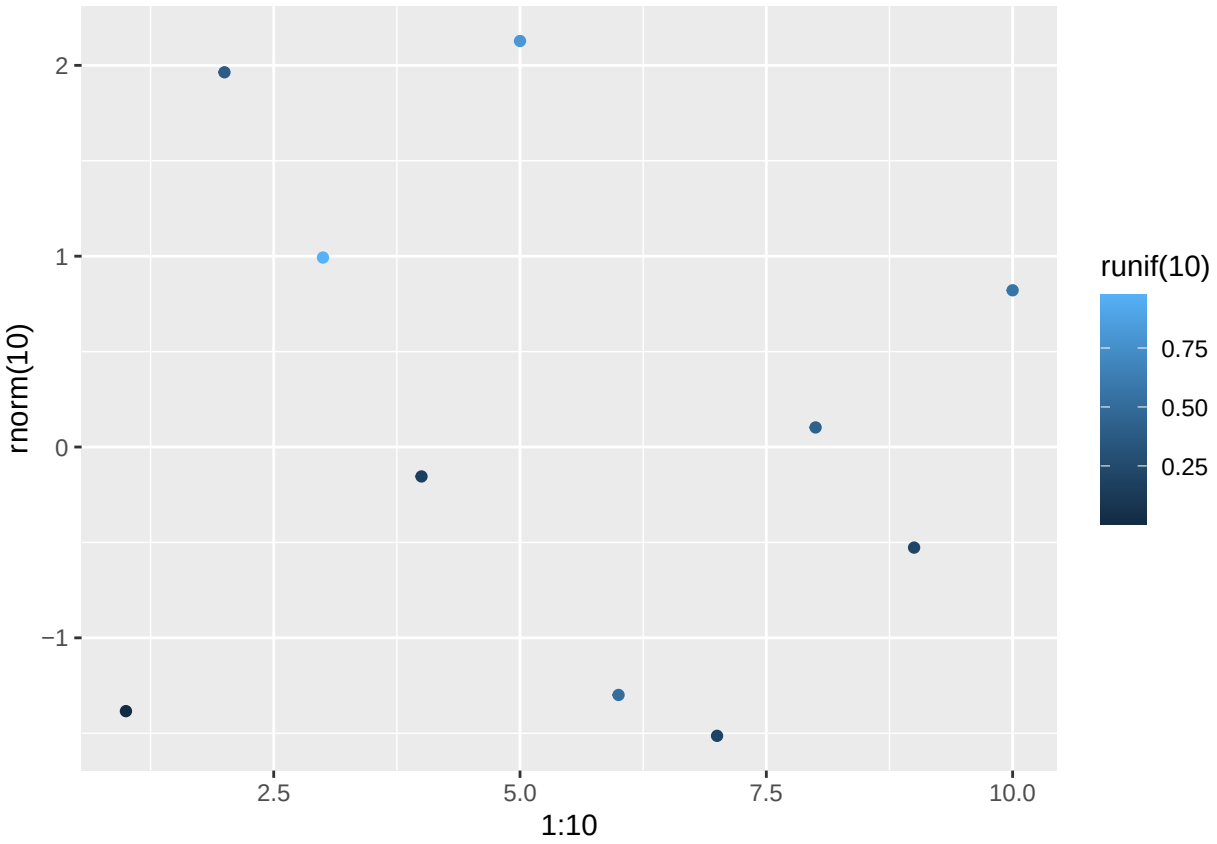
2.4 Packages

`R` dispose d'une communauté très active. Les très nombreux *packages* sont développés par les utilisateurs et développeurs. On peut les installer simplement sur son ordinateur. Par exemple, pour faire appel à quelques outils avancés de représentation graphique, on peut utiliser le package `ggplot2`. On exécute dans la console :

```
install.packages('ggplot2')
```

(on peut aussi passer par l'interface graphique, onglet "Packages") puis on charge le package :

```
library('ggplot2')
ggplot(mapping=aes(x=1:10,y=rnorm(10),colour=runif(10)))+geom_point()
```



Quelle est cette fonction `ggplot` ? La réponse est dans la sous-fenêtres `help`, après avoir exécuté la ligne suivante.

```
?ggplot
```

```
## Aucune documentation pour 'ggplot' n'a été trouvée dans les packages et les bibliothèques :
## vous pourriez essayer '??ggplot'
```

L'aide s'applique aussi aux packages. Par exemple,

```
?ggplot2
```

```
## Aucune documentation pour 'ggplot2' n'a été trouvée dans les packages et les bibliothèques :
## vous pourriez essayer '??ggplot2'
```

Dans RStudio, dans l'onget "Package", les packages chargés apparaissent cochés. Certains packages d'usage fréquent sont chargés par défaut au lancement de RStudio.

3 Bases du langage

3.1 Commandes

Deux types de commandes

Expressions : si on entre un objet ou une opération, la console R affiche sa valeur.

```
cos(pi)
```

```
## [1] -1
```

```
log(1)
```

```
## [1] 0
```

Affectations : une ligne contenant une affectation n'affiche pas de valeur. On utilise l'opérateur d'affectation `<-` ou simplement `=`.

```
# x=1+2
x <- 1+2
x
```

```
## [1] 3
```

```
y = 4
x == y
```

```
## [1] FALSE
```

```
ma.variable <- 5  # On peut utiliser un "." dans les noms de variable
```

A l'aide de `;` on peut taper deux commandes sur la même ligne avant leur exécution :

```
e <- exp(1); log(e)
```

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

Exemple d'opérateurs :

```
3*4; 10/3; 10%/%3; 2^3; sqrt(16)
```

```
## [1] 12
```

```
## [1] 3.333333
```

```
## [1] 3
```

```
## [1] 8
```

```
## [1] 4
```

```
1==2; 1!=1 ;
```

```
## [1] FALSE
```

```
## [1] FALSE
```

```
!FALSE
```

```
## [1] TRUE
```

```
FALSE & TRUE
```

```
## [1] FALSE
```

```
FALSE | TRUE
```

```
## [1] TRUE
```

3.2 Structures, modes

Dans R, tout est un objet, ayant une certaine structure et contenant des éléments (selon la structure) le même mode ou des modes possiblement différents.

Les principaux **modes** sont :

— `numeric` : nombres réels

— `character` : chaînes de caractères (donnée entre apostrophes ou guillemets)
— `logical` : valeurs logiques vrai (`T` ou `TRUE`)/faux (`F` ou `FALSE`)
— `function` : fonction
— `list` : toute collection d'objets ayant (potentiellement) divers modes

La fonction `mode(x)` donne le mode de `x`.

```
mode(42)
```

```
## [1] "numeric"
```

```
mode("texte")
```

```
## [1] "character"
```

```
mode(1==2)
```

```
## [1] "logical"
```

```
mode(mode)
```

```
## [1] "function"
```

```
mode(list(45,"a"))
```

```
## [1] "list"
```

Les principales **structures** sont :

— `vector` : tableau unidimensionnel, dont les éléments ont le même mode (un scalaire est un vecteur)
— `matrix` : tableau bi-dimensionnel, dont les éléments ont le même mode
— `array` : tableau multi-dimensionnel, dont les éléments ont le même mode
— `list` : "tableau" unidimensionnel, dont les éléments ont des modes qui peuvent différer
— `data.frame` : "tableau bi-dimensionnel", dont les éléments d'une même colonne ont le même mode. Utilisé pour stocker des données : les colonnes sont les **variables** et les lignes les **observations**.

La fonction `str(x)` détaille la structure de `x`.

```
x <- list(1,"abc",T,list("cd",5))
str(x)
```

```
## List of 4
##  $ : num 1
##  $ : chr "abc"
##  $ : logi TRUE
##  $ :List of 2
##   ..$ : chr "cd"
##   ..$ : num 5
```

Pour tous ces modes/structures, on dispose de

— fonctions pour interroger : `is.numeric(x)` renvoie `TRUE` ou `FALSE`
— fonctions de conversion : `as.numeric(x)`

3.3 Vecteurs

```
x <- c(1,3,4,5,7) ; x
```

```
## [1] 1 3 4 5 7
```

```
length(x)
```

```
## [1] 5
```

```
x[1]
```

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

```
2*x
```

```
## [1]  2  6  8 10 14
```

```
x^2
```

```
## [1]  1  9 16 25 49
```

```
sqrt(x)
```

```
## [1] 1.000000 1.732051 2.000000 2.236068 2.645751
```

```
x*x*x
```

```
## [1]   1  27  64 125 343
```

```
2:5
```

```
## [1] 2 3 4 5
```

```
x[2:5]
```

```
## [1] 3 4 5 7
```

```
x[c(1,4,5)]
```

```
## [1] 1 5 7
```

```
x>2
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE
```

```
x[x>2]
```

```
## [1] 3 4 5 7
```

```
which(x>2)
```

```
## [1] 2 3 4 5
```

```
x[which(x>2)]
```

```
## [1] 3 4 5 7
```

```
x[-2]
```

```
## [1] 1 4 5 7
```

```
x[-c(2,3)]
```

```
## [1] 1 5 7
```

```
x <- 2:7 ; x
```

```
## [1] 2 3 4 5 6 7
```

```
y <- 5:1 ; y
```

```
## [1] 5 4 3 2 1
```

```
c(x,y)
```

```
##  [1] 2 3 4 5 6 7 5 4 3 2 1
```

```r
rep(x,4)
```

```
##  [1] 2 3 4 5 6 7 2 3 4 5 6 7 2 3 4 5 6 7 2 3 4 5 6 7
```

```r
seq(from=5,by=2,to=17)
```

```
## [1]  5  7  9 11 13 15 17
```

```r
seq(5,17,2)
```

```
## [1]  5  7  9 11 13 15 17
```

```r
x %in% 1:4
```

```
## [1]  TRUE  TRUE  TRUE FALSE FALSE FALSE
```

Différences avec Matlab : Comme en C, on utilise des crochets pour accéder aux éléments. Les opérations ^, * opèrent élément par élément.

```r
x <- 2:7 ; x
```

```
## [1] 2 3 4 5 6 7
```

```r
y <- 5:1 ; y
```

```
## [1] 5 4 3 2 1
```

Nommage des indices : Les structures de R étant destinées à contenir des données ayant une signification, il est possible de nommer les colonnes. Les noms peuvent alors servir pour manipuler la structure, et apparaissent dans les retours (texte ou graphique).

```r
x <- 1:5 ; x
```

```
## [1] 1 2 3 4 5
```

```r
names(x) <- c("a1","b","c","d","e"); x
```

```
## a1  b  c  d  e
##  1  2  3  4  5
```

```r
x["b"]
```

```
## b
## 2
```

```r
x[c("b","d")]
```

```
## b d
## 2 4
```

```r
x[2]
```

```
## b
## 2
```

```r
x[c(2,4)]
```

```
## b d
## 2 4
```

```r
# x[-("b")]  # Erreur
x[!(names(x) %in% c("b"))]
```

```
## a1  c  d  e
```

```
##  1  3  4  5
```

```
x <- c(a=1, individu2=2, c=3, d=4) ; x
```

```
##         a individu2         c         d
##         1         2         3         4
```

```
x[5] = 5 ; x
```

```
##         a individu2         c         d
##         1         2         3         4         5
```

```
x["e"]=6 ; x
```

```
##         a individu2         c         d                 e
##         1         2         3         4         5         6
```

3.4 Données manquantes

Un objet spécial est la valeur manquante `NA` (*not available*). On l'utilise pour spécifier qu'une donnée n'est pas disponible. On peut l'utiliser dans n'importe quelle structure.

```
x <- c(1,4,NA,NA,2) ; x
```

```
## [1]  1  4 NA NA  2
```

```
y <- NA + 5 ; y
```

```
## [1] NA
```

```
NA * NA
```

```
## [1] NA
```

```
z <- NA
y == z
```

```
## [1] NA
```

```
is.na(y)
```

```
## [1] TRUE
```

3.5 Listes

```
ma.liste = list("Jean","Lenoir",35); ma.liste
```

```
## [[1]]
## [1] "Jean"
##
## [[2]]
## [1] "Lenoir"
##
## [[3]]
## [1] 35
```

```
ma.liste[[2]]
```

```
## [1] "Lenoir"
```

```
ma.liste = list(prénom="Jean", nom="Lenoir", age=35) ; ma.liste
```

```
## $prénom
## [1] "Jean"
##
## $nom
## [1] "Lenoir"
##
## $age
## [1] 35
```

```
ma.liste[["nom"]]
```

```
## [1] "Lenoir"
```

```
ma.liste$nom    # Ne fonctionne pas pour les vecteurs
```

```
## [1] "Lenoir"
```

3.6 Matrices

```
M <- matrix(1:6, nrow=2, ncol=3); M
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

```
N <- matrix(1:6, nrow=2, ncol=3, byrow=T); N
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
```

```
dim(M)
```

```
## [1] 2 3
```

```
M[1,3]
```

```
## [1] 5
```

```
M[,3]
```

```
## [1] 5 6
```

```
M[5]
```

```
## [1] 5
```

```
M[1,]
```

```
## [1] 1 3 5
```

```
M[,-2]
```

```
##      [,1] [,2]
## [1,]    1    5
## [2,]    2    6
```

```
cbind(M,N) # bind by columns
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    3    5    1    2    3
## [2,]    2    4    6    4    5    6
```

```
rbind(M,N) # bind by rows
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
## [3,]    1    2    3
## [4,]    4    5    6
```

```
t(M)
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]    5    6
```

```
M*M
```

```
##      [,1] [,2] [,3]
## [1,]    1    9   25
## [2,]    4   16   36
```

```
M %*% t(M)
```

```
##      [,1] [,2]
## [1,]   35   44
## [2,]   44   56
```

```
diag(c(1,2,3),3)
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    0    2    0
## [3,]    0    0    3
```

On peut aussi nommer les lignes et colonnes.

```
colnames(M) <- c("a","b","c")
rownames(M) <- c("d","e")
M
```

```
##   a b c
## d 1 3 5
## e 2 4 6
```

```
M["d","b"]
```

```
## [1] 3
```

```
n <- 4
M <- matrix(runif(n^2),nrow=n,ncol=n)
M
```

```
##           [,1]       [,2]       [,3]      [,4]
## [1,] 0.56464091 0.79990778 0.62574671 0.6710058
## [2,] 0.33418788 0.04185471 0.01552273 0.8807744
## [3,] 0.37712612 0.58922995 0.68769580 0.5013319
## [4,] 0.04645702 0.48229379 0.63025313 0.4478148
```

```
x <- eigen(M)
x
```

```
## eigen() decomposition
## $values
## [1]  1.8367644+0.0000000i -0.4266830+0.0000000i  0.1659624+0.1549994i
## [4]  0.1659624-0.1549994i
##
## $vectors
##              [,1]          [,2]                  [,3]                  [,4]
## [1,] 0.6715440+0i -0.3575164+0i  0.6124814+0.0000000i  0.6124814+0.0000000i
## [2,] 0.3173041+0i  0.8576324+0i -0.2377446+0.4298381i -0.2377446-0.4298381i
## [3,] 0.5498355+0i -0.1897698+0i  0.2853168-0.4258917i  0.2853168+0.4258917i
## [4,] 0.3821357+0i -0.3172319+0i -0.3465625+0.0262348i -0.3465625-0.0262348i
```

```
x$values
```

```
## [1]  1.8367644+0.0000000i -0.4266830+0.0000000i  0.1659624+0.1549994i
## [4]  0.1659624-0.1549994i
```

3.7 Facteurs

Un facteur est un vecteur utilisé pour contenir une variable qualitative (valeurs discrètes). Ses valeurs, ou catégories ou encore modalités, sont appelées les `levels` en R.

```
ville <- c('paris', 'lyon', 'lyon', 'paris', 'nantes')
fact.ville <- factor(ville); fact.ville
```

```
## [1] paris  lyon   lyon   paris  nantes
## Levels: lyon nantes paris
```

```
levels(fact.ville)
```

```
## [1] "lyon"   "nantes" "paris"
```

```
mode(fact.ville)
```

```
## [1] "numeric"
```

```
as.numeric(fact.ville)
```

```
## [1] 3 1 1 3 2
```

On souhaite parfois que les facteurs soient ordonnés (ce sera utile pour des représentations graphiques, par exemple).

```
tailles <- c('S', 'S', 'M', 'S', 'L', 'M')
fact.tailles <- factor(tailles,levels=c('S','M','L'),ordered=TRUE)
fact.tailles
```

```
## [1] S S M S L M
## Levels: S < M < L
```

```
fact.tailles[1]<fact.tailles[3]
```

```
## [1] TRUE
```

et on peut vouloir donner des étiquettes plus explicites que les “levels” :

```
tailles <- c('S', 'S', 'M', 'S', 'L', 'M')
fact.tailles <- factor(tailles,levels=c('S','M','L'),labels=c('petit','moyen','grand'),ordered=TRUE)
fact.tailles
```

```
## [1] petit petit moyen petit grand moyen
## Levels: petit < moyen < grand
```

3.8 Data frames

Les data frame sont le principal type dédié aux données. Ce sont (presque) des listes de vecteurs dont tous les éléments ont la même longueur. Normalement, dans un data frame les colonnes sont les **variables** (les quantités observées) et les lignes les **observations**. Contrairement aux matrices, les éléments d'un data frame peuvent avoir des modes différents.

```
id <- c('id.442', 'id.443', 'id.444', 'id.445', 'id.446', 'id.447', 'id.448', 'id.449')
age <- c(19, 45, 67, 53, 17, 30, 27, 35)
fumeur <- c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE, NA, FALSE)
data1 <- data.frame(Id=id, Age=age, Fumeur=fumeur)
dim(data1); nrow(data1); ncol(data1)
```

```
## [1] 8 3
```

```
## [1] 8
```

```
## [1] 3
```

```
names(data1)
```

```
## [1] "Id"     "Age"    "Fumeur"
```

Pour extraire les éléments :

```
data1$Id # une colonne de caractères est par défaut transformée en facteur
```

```
## [1] "id.442" "id.443" "id.444" "id.445" "id.446" "id.447" "id.448" "id.449"
```

```
data1[,2]
```

```
## [1] 19 45 67 53 17 30 27 35
```

```
data1[3,]
```

```
##       Id Age Fumeur
## 3 id.444  67   TRUE
```

```
data1$Age[data1$Fumeur==FALSE]
```

```
## [1] 45 17 NA 35
```

```
data1[data1$Fumeur==FALSE, ]
```

```
##        Id Age Fumeur
## 2  id.443  45  FALSE
## 5  id.446  17  FALSE
## NA   <NA>  NA     NA
## 8  id.449  35  FALSE
```

On peut éviter de taper le nom du data frame à chaque fois en l'attachant :

```
attach(data1)
Age
```

```
## [1] 19 45 67 53 17 30 27 35
```

```
Age[Fumeur==F]
```

```
## [1] 45 17 NA 35
```

```
detach(data1)

# Ou
```

```
with(data1,{
  Age
  Age[Fumeur==F]
})
```

```
## [1] 45 17 NA 35
```

```
AgeF = with(data1,Age[Fumeur==F])
```

Pour afficher seulement les six premières lignes :

```
head(data1)
```

```
##       Id Age Fumeur
## 1 id.442  19   TRUE
## 2 id.443  45  FALSE
## 3 id.444  67   TRUE
## 4 id.445  53   TRUE
## 5 id.446  17  FALSE
## 6 id.447  30   TRUE
```

et les six dernières avec `tail()`.

On aurait aussi pu utiliser une colonne comme nom de lignes, via le paramètre `row.names`.

```
data1 <- data.frame(Age=age, Fumeur=fumeur, row.names=id)
data1
```

```
##        Age Fumeur
## id.442  19   TRUE
## id.443  45  FALSE
## id.444  67   TRUE
## id.445  53   TRUE
## id.446  17  FALSE
## id.447  30   TRUE
## id.448  27     NA
## id.449  35  FALSE
```

```
data1["id.443",]
```

```
##        Age Fumeur
## id.443  45  FALSE
```

```
data1$AgeMois <- data1$Age*12
data1
```

```
##        Age Fumeur AgeMois
## id.442  19   TRUE     228
## id.443  45  FALSE     540
## id.444  67   TRUE     804
## id.445  53   TRUE     636
## id.446  17  FALSE     204
## id.447  30   TRUE     360
## id.448  27     NA     324
## id.449  35  FALSE     420
```

Importation. En général, les données sont stockées dans des fichiers, et importées sous R.

Le format le plus simple est un fichier texte, où la première ligne (*entête*) donne le nom des variables, puis chaque ligne contient une observation, avec des valeurs séparées par des virgules. On parle de format CSV

(*comma separated value*) et l'extension est souvent `.csv`. L'import est réalisé par la fonction `read.table` dont les paramètres autorisent d'autres séparateurs que la virgule, et la présence ou non de l'entête. Le paramètre `dec` précise le symbole utilisé pour le séparateur décimal. Par défaut c'est le point, mais en français on utilise aussi parfois la virgule (il faut y être vigilent).

```
donnees <- read.table("données.csv",sep=",",dec=".",header=T)
head(donnees)
```

```
##       Nom Prénom Age Poids Réponse
## 1 Lelarge  Émile  42  80.4     Oui
## 2  Martin  Louis  55  78.1     Non
## 3 Meunier Sophie  22    NA     Oui
```

```
df1<-read.table('Iris.txt', header= TRUE, sep=';')
head(df1)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

De même on dispose de `write.table` pour écrire un fichier de données en format texte.

Des packages permettent de lire et écrire des fichiers Excel (cf. le fichier `readxl` et la fonction `read_excel(fichier)`) et bien d'autres formats.

4 Programmation

4.1 Fonctions définies par l'utilisateur

Exemple :

```
ma.fonction <- function(x,y=10){
  # la valeur par défaut de y est 10
  z=x-2*y
  return(z)
}
ma.fonction(2)
```

```
## [1] -18
```

```
ma.fonction(2,4)
```

```
## [1] -6
```

```
ma.fonction(y=1, x=4)
```

```
## [1] 2
```

Toute variable définie dans une fonction est *locale* et n'apparaît pas dans l'espace de travail : essayer d'exécuter

```
z
```

4.2 Structures de contrôle

Traitement conditionnel, If :

```r
x <- runif(1) # valeur tirée selon une loi uniforme dans [0,1]
if(x<1/6){
  print('gagné')
}else{
  print('perdu')
}
```

```
## [1] "perdu"
```

Traitement itératif, boucle `for` :

```r
y=rep(NA,5)
for(i in 1:5){
  y[i] <- exp(i)
}
y
```

```
## [1]   2.718282   7.389056  20.085537  54.598150 148.413159
```

Comme les boucles ne sont pas très efficaces, il faut essayer de les remplacer par des opérations sur vecteur :

```r
exp(1:5)
```

```
## [1]   2.718282   7.389056  20.085537  54.598150 148.413159
```

Traitement itératif, boucle `while` :

```r
  tirage_disque <- function(){
    x <- 1; y<-1;
    while(x^2+y^2 >1){
      x=runif(1)*2-1
      y=runif(1)*2-1
    }
    return(c(x,y))
  }

  tirage_disque()
```

```
## [1] -0.2926226  0.7651999
```

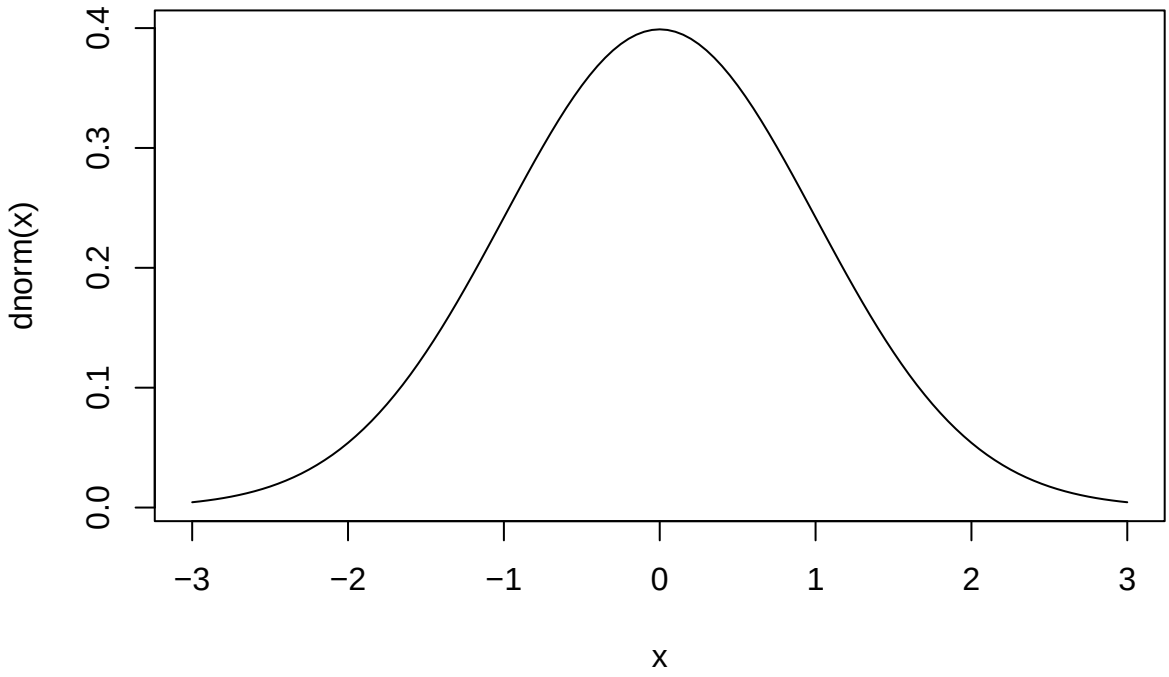
4.3 Fonctions de génération aléatoire de base

Pour chaque loi classique, R dispose de 4 fonctions associées, commençant par

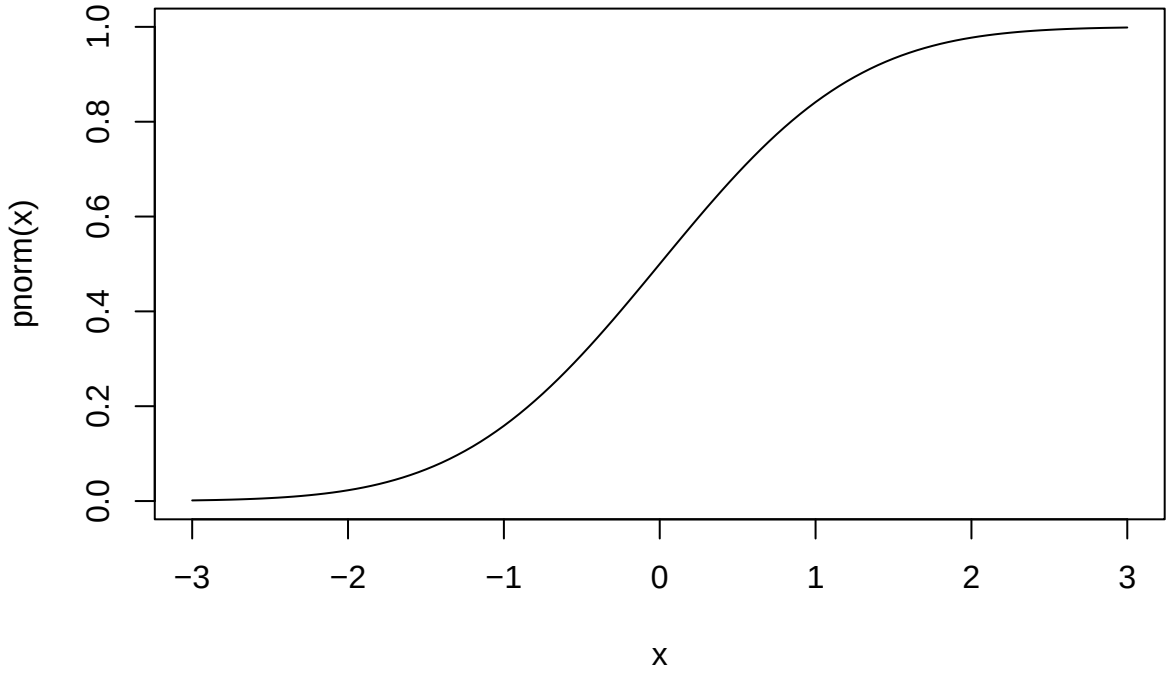
- `d` : densité de probabilité
- `p` : fonction de répartition (*probability distribution function*)
- `q` : quantiles
- `r` : génération aléatoire (*random generation*).

Ainsi, `dnorm`, `pnorm`, `qnorm`, `rnorm` pour la loi gaussienne.

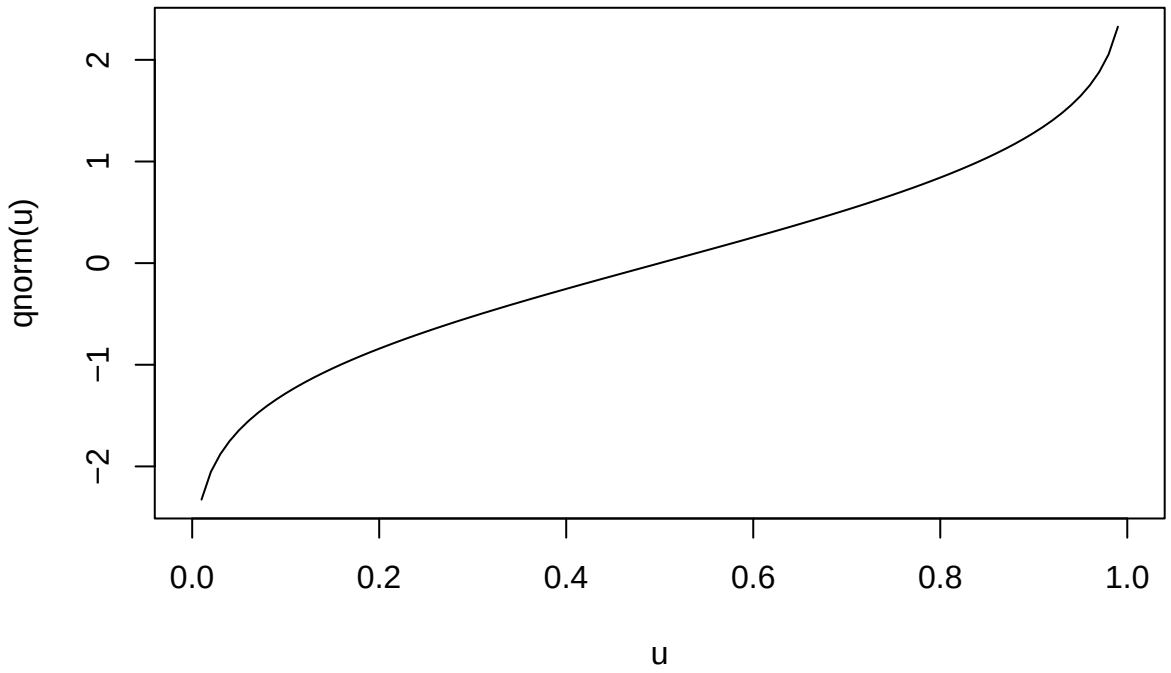
```r
  x= pretty(c(-3,3),100)
  plot(x,dnorm(x),type="l")
```



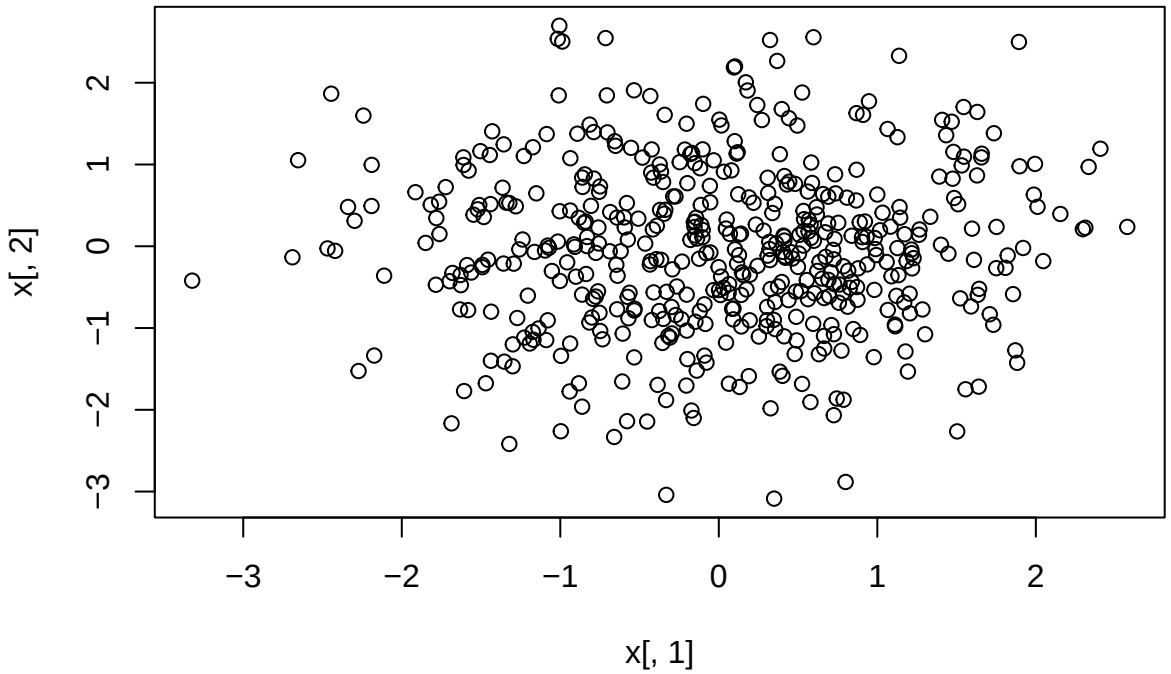

```
plot(x,pnorm(x),type="l")
```




```
u = seq(0,1,0.01)
plot(u,qnorm(u),type="l")
```




```
N=1000; x=matrix(rnorm(N),ncol=2)
plot(x[,1],x[,2])
```



De même, `binom`, `cauchy`, `exp`, `geom`, `pois`, `beta`, `gamma`, `t`,... Cf `?Distributions` et chaque fonction pour voir les paramètres.

Pour obtenir une suite reproductible de tirages pseudo-aléatoire, on peut fixer la graine du générateur avec `set.seed`.

```
runif(1)
```

```
## [1] 0.5961039
```

```
set.seed(123)
runif(1)
```

```
## [1] 0.2875775
```

```
set.seed(123)
runif(1)
```

```
## [1] 0.2875775
```

5 Statistiques descriptives de base

R est fourni avec de nombreux jeux de données réelles (dans le package `datasets`) qui permettent de traiter des exemples. Selon les cas, on utilisera des données simulées (avec les générateurs aléatoires) ou des données réelles de R.

```
?mtcars
head(mtcars)
```

```
##                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

5.1 Fonctions statistiques sur des variables quantitatives

```
x <- rnorm(n=10000)
mean(x)
```

```
## [1] -0.0195647
```

```
sd(x)
```

```
## [1] 0.9871208
```

```
var(x)
```

```
## [1] 0.9744075
```

```
range(x)
```

```
## [1] -4.055104  3.851758
```

```
quantile(x,c(0.025,0.975))
```

```
##      2.5%     97.5%
## -1.940473  1.926188
```

```
summary(x)
```

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.05510 -0.68885 -0.02417 -0.01956  0.65536  3.85176
```

```
summary(mtcars[,c("mpg","hp")])
```

```
##       mpg              hp
##  Min.   :10.40   Min.   : 52.0
```

```
##  1st Qu.:15.43   1st Qu.: 96.5
##  Median :19.20   Median :123.0
##  Mean   :20.09   Mean   :146.7
##  3rd Qu.:22.80   3rd Qu.:180.0
##  Max.   :33.90   Max.   :335.0
```

Les fonctions peuvent en général être appliquées aux vecteurs et matrices, mais aussi à chaque ligne ou colonne d'une matrice ou d'un data frame, avec `apply(X, MARGIN, FUN)` où `MARGIN` vaut 1 (ligne) ou 2 (colonne) (ou plus), et `FUN` est une fonction qui s'applique aux vecteurs.

```
apply(mtcars,1,mean)
```

```
##           Mazda RX4       Mazda RX4 Wag          Datsun 710      Hornet 4 Drive
##            29.90727            29.98136            23.59818            38.73955
##   Hornet Sportabout             Valiant          Duster 360           Merc 240D
##            53.66455            35.04909            59.72000            24.63455
##            Merc 230            Merc 280           Merc 280C          Merc 450SE
##            27.23364            31.86000            31.78727            46.43091
##          Merc 450SL         Merc 450SLC  Cadillac Fleetwood Lincoln Continental
##            46.50000            46.35000            66.23273            66.05855
##   Chrysler Imperial            Fiat 128         Honda Civic      Toyota Corolla
##            65.97227            19.44091            17.74227            18.81409
##       Toyota Corona    Dodge Challenger         AMC Javelin          Camaro Z28
##            24.88864            47.24091            46.00773            58.75273
##    Pontiac Firebird           Fiat X1-9       Porsche 914-2        Lotus Europa
##            57.37955            18.92864            24.77909            24.88027
##      Ford Pantera L        Ferrari Dino       Maserati Bora          Volvo 142E
##            60.97182            34.50818            63.15545            26.26273
```

```
apply(mtcars,2,mean)
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

Pour plusieurs variables quantitatives, on peut s'intéresser aux covariances ou aux corrélations.

$$\mathrm{Corr}(X,Y) = \frac{\mathrm{Cov}(X,Y)}{\sigma(X)\sigma(Y)} \in [-1,1]$$

```
cor(mtcars[c("mpg","disp","hp")])
```

```
##             mpg       disp         hp
## mpg   1.0000000 -0.8475514 -0.7761684
## disp -0.8475514  1.0000000  0.7909486
## hp   -0.7761684  0.7909486  1.0000000
```

On y reviendra en détails. . ..

5.2 Variables quantitatives et catégorielles

On peut souhaiter regrouper les données selon la valeur d'une variable catégorielle pour obtenir des statistiques de variables quantitatives par groupe. Pour cela, on utilise `aggregate(x,by,FUN)` : sur les données `x` groupées selon la valeurs des variables listées dans `by`, on évalue la fonction `FUN`.

```
df <- mtcars[c("mpg","cyl","hp","wt")]
aggregate(df,list(df$cyl),mean) # moyennes dans chaque groupe
```

```
##   Group.1      mpg cyl        hp       wt
## 1       4 26.66364   4  82.63636 2.285727
## 2       6 19.74286   6 122.28571 3.117143
## 3       8 15.10000   8 209.21429 3.999214
```

```
aggregate(df,list(df$cyl),sd) # écart-types dans chaque groupe
```

```
##   Group.1      mpg cyl       hp        wt
## 1       4 4.509828   0 20.93453 0.5695637
## 2       6 1.453567   0 24.26049 0.3563455
## 3       8 2.560048   0 50.97689 0.7594047
```

```
df <- df1[-5] # données Iris sans l'espèce
aggregate(df,by=list(df1$Species),mean)
```

```
##      Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     setosa        5.006       3.428        1.462       0.246
## 2 versicolor        5.936       2.770        4.260       1.326
## 3  virginica        6.588       2.974        5.552       2.026
```

5.3 Variables catégorielles

Si on s'intéresse uniquement à une ou plusieurs variables catégorielles, il est utile d'obtenir les fréquences de chaque catégorie (liste de fréquence) ou de chaque paire de catégories (table de contingence).

```
# Pour chaque valeur (4,6 ou 8) du nombre de cylindres,
# on lit le nombre d'observations correspondantes
table(mtcars$cyl)
```

```
##
##  4  6  8
## 11  7 14
```

```
# Pour chaque valeur (4,6 ou 8) du nombre de cylindres,
# et chaque valeur de vs (0 ou 1),
# on lit le nombre d'observations correspondantes
tab <- table(mtcars$cyl,mtcars$vs) ; tab
```

```
##
##      0  1
##   4  1 10
##   6  3  4
##   8 14  0
```

```
addmargins(tab)
```

```
##
##        0  1 Sum
##   4    1 10  11
##   6    3  4   7
##   8   14  0  14
##   Sum 18 14  32
```

```
# Pour obtenir des fréquences (entre 0 et 1) plutôt que des effectifs :
prop.table(tab,2)
```

```
##
##              0          1
```

```
##   4 0.05555556 0.71428571
##   6 0.16666667 0.28571429
##   8 0.77777778 0.00000000
```

L'observation de telles tables peut mener à se demander si une variable a une influence sur une autre. Pour cela, on sort du cadre de la *statistique descriptive* pour introduire un modèle aléatoire sur les données et déduire des observations certaines propriétés du modèle : il s'agit de *statistique inférentielle* ou *statistique mathématique*. En l'occurrence, en supposant que les variables catégorielles observées sont des réalisations de 2 variables aléatoires (discrètes), on souhaite tester si ces variables aléatoires sont indépendantes, et on utilise pour cela un test du χ^2 d'indépendance. En R, la fonction `chisq.test` prend en argument une table de contingence produite par la fonction `table`.

```
library(vcd);
```

```
## Le chargement a nécessité le package : grid
```

```
?Arthritis
data <- Arthritis
head(data)
```

```
##   ID Treatment  Sex Age Improved
## 1 57   Treated Male  27     Some
## 2 46   Treated Male  29     None
## 3 77   Treated Male  30     None
## 4 17   Treated Male  32   Marked
## 5 36   Treated Male  46   Marked
## 6 23   Treated Male  58   Marked
```

```
ma.table <- table(data$Treatment,data$Improved); ma.table
```

```
##
##           None Some Marked
##   Placebo   29    7      7
##   Treated   13    7     21
```

```
chisq.test(ma.table)
```

```
##
##  Pearson's Chi-squared test
##
## data:  ma.table
## X-squared = 13.055, df = 2, p-value = 0.001463
```

(pour installer le package `vcd` il peut être nécessaire d'installer le logiciel `gfortran`)

On teste ici l'indépendance entre le choix du traitement (Placebo ou Traitement véritable) et l'amélioration de l'état du patient (Aucune, Légère, Nette). L'hypothèse nulle est l'indépendance entre les deux. D'après la réponse ci-dessus, la p-valeur du test est 0.0014626 ; elle est inférieure à 5 % donc on **rejette** l'hypothèse d'indépendance, au seuil de risque 5 %. Pour rappel, cela signifie qu'il y a une probabilité 0.0014626 pour que, sous l'hypothèse d'indépendance, la statistique du test ait une valeur aussi grande que celle calculée sur les données (à savoir 13.0550199).

On considère ensuite le lien entre sexe du patient et amélioration.

```
ma.table <- table(data$Sex,data$Improved); ma.table
```

```
##
##          None Some Marked
##   Female   25   12     22
##   Male     17    2      6
```

```
chisq.test(ma.table)
```

```
## Warning in chisq.test(ma.table): L'approximation du Chi-2 est peut-être
## incorrecte
```

```
##
##  Pearson's Chi-squared test
##
## data:  ma.table
## X-squared = 4.8407, df = 2, p-value = 0.08889
```

Cette fois-ci, la p-valeur est supérieure à 5 %, les données sont donc compatibles avec l'hypothèse d'indépendance et ne permettent pas de la rejeter. Noter toutefois le message d'avertissement : "Chi-squared approximation may be incorrect". Il signifie que les conditions d'application du test ne sont pas toutes satisfaites, en l'occurrence parce que l'un des couples de catégories, à savoir (Male,Some), n'est observé que 2 fois. La conclusion d'indépendance n'est donc pas complètement étayée.

6 Exercices

Les exercices sont à faire dans un fichier R Markdown, avec une cellule de code par question.

6.1 Ex.1

En utilisant le data frame `mtcars`,

1. Définir un data frame `df` qui ne garde que les variables “mpg”, “cyl”,“wt”,“qsec”.
2. Afficher les observations (les modèles de voitures) pour lesquelles `mpg` est supérieur à 20, et `am` vaut 1.
3. Définir un nouveau data frame `df2` avec des variables “consommation”, “cylindres”, “poids”, “vitesse” qui contient les champs précédents traduits en litre/km, nombre (inchangé), tonnes, km/h. Utiliser `?mtcars` pour la signification des observations, et internet pour les conversions. (`qsec` donne le nombre de secondes nécessaires pour parcourir un quart de mile) Vérifier que les noms des lignes sont bien conservés (utiliser `rownames` si besoin).
4. Déterminer les quartiles (quantiles à 25 %, 50 %, 75 %) de “vitesse”.
5. Ajouter à “df2” une observation qui vaut “Très rapide”, “Rapide”, “Lente”, “Très lente” selon que la vitesse se trouve dans le premier quartile, le 2è, etc. Cette observation doit être un facteur.
6. Calculer la consommation moyenne dans chacun des groupes définis précédemment.
7. Produire une table de contingence avec le nombre de cylindres et le groupe de vitesse.

6.2 Ex.2

En utilisant les données du fichier “titanic.csv”,

1. Définir un data frame “df” qui ne garde que les variables “pclass” (classe du passager), “survived”, “sex”, “age”, et “fare” (prix du billet).
2. Modifier les variables “sex” et “pclass” pour en faire des facteurs. Modifier “survived” pour en faire un facteur ayant pour étiquettes “died” ou “survived” selon la valeur. Que donne `summary(df)` pour les variables factorielles ?
3. Ajouter au dataframe une variable factorielle “age.category” qui vaut “Enfant” de 0 à 17 ans, et “Adulte” au-dessus de 18 ans (et `NA` si non disponible).
4. Quel est le prix médian des billets selon la classe du passager ? Utiliser `aggregate`. Utiliser l'argument `na.rm=T` pour enlever les données manquantes (essayer d'abord sans cet argument). Est-ce justifié ici de supprimer ces données ?
5. Calculer les pourcentages de survivants selon le sexe. Puis selon la classe du passager. Puis selon la catégorie d'âge. (Un critère après l'autre, pas simultanément). Utiliser `prop.table`.
6. Utiliser la fonction `quantile()` pour déterminer les quantiles du prix du billet aux niveaux 40% et 60%.
7. Existe-t-il un passager typique, au sens où il a le sexe le plus fréquent, est dans la classe la plus fréquente, a l'âge moyen (arrondi à l'entier le plus proche), et a payé un tarif situé dans les 20% autour de la médiane (cf. question précédente) ? Comment s'appelle-t-il ?
8. R contient par défaut une base de données `Titanic` non nominative, sous forme d'une table de contingence à quatre dimensions (et non d'un dataframe) : `Titanic["1st","Male","Child","Yes"]` donne le nombre d'enfants garçons en 1re classe survivants (voir l'aide `?Titanic` pour une description). Calculer le nombre d'enfants pris en compte dans cette série statistique (utiliser `sum( )` sur un sous-tableau). À l'aide de `titanic`, deviner jusqu'à quel âge cette nouvelle table considère un passager comme enfant (il y a de petites différences dans les effectifs entre `titanic` et `Titanic`, comme signalé dans l'aide ; ne pas en tenir compte, et supposer que les `NA` sont des adultes).