TP 0: Introduction à R

Vos noms

Statistiques biomédicales

Table des matières

1. Introduction

`R` est est un langage de programmation et un logiciel libre destiné aux statistiques et à la science des données soutenu par la R Foundation for Statistical Computing. Il fait partie de la liste des paquets GNU3 et est écrit en `C` (langage), Fortran et `R`. `R` est un logiciel open source (gratuit et modifiable) (cran.r-project.org). Il est très largement utilisé à partir de l'interface graphique `RStudio` (www.rstudio.com).

Il peut-être utilisé sur des fichiers `R script` pour faire du code pur. Les commentaires sont alors à introduire sur une ligne précédée d'un `#` afin que ceux-ci ne soit pas interprétés.

```
# ceci n'est pas interprété.
```

```
2+2
```

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

Nous l'utiliserons sur des fichiers `Rmarkdown` (ou `R notebook`), pour le confort. Il est alors possible de produire des fichiers `html` ou `pdf` (dans l'en-tête changer `html` par `pdf`, selon votre goût). On ouvre une cellule de code avec le raccourci `Ctrl+Alt+i`. On l'exécute en cliquant sur la flèche verte en bout de ligne (ou en la saisissant plus `Ctrl+ Entrée`). Le fichier s'éxecute en cliquant sur `Knit`.

Comme `python`, `R` est un langage interprété :

```
print('Hello world!')
```

```
## [1] "Hello world!"
```

On peut aussi exécuter un script, c'est à dire des commandes dans un fichier `.R`. Attention, il faut que le fichier soit situé dans le même dossier (ou alors on donne le chemin complet). Si l'on souhaite que des fichiers

`.R` situé dans le même dossier s'appelle entre eux sans spécifier le chemin, il faut situer la session dans le dossier commun : `Session` puis `set working directory`.

```
source('Fichier1.R')
```

```
## [1] "Hello world!?"
```

`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 quelques outils avancés de combinatoires, le package `combinat`, on exécute dans la console : `install.packages('combinat')`, puis :

```
#Je n'ai pas exécuté la ligne suivante, car, en faîte, combinat est déjà installé. Elle sert à mettre l
#library('combinat')
combn(1:6,3)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13] [,14]
## [1,]    1    1    1    1    1    1    1    1    1     1     2     2     2     2
## [2,]    2    2    2    2    3    3    3    4    4     5     3     3     3     4
## [3,]    3    4    5    6    4    5    6    5    6     6     4     5     6     5
##      [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]     2     2     3     3     3     4
## [2,]     4     5     4     4     5     5
## [3,]     6     6     5     6     6     6
```

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

```
?combn
```

De même pour `rnorm`

```
?rnorm
```

Chercher sur le web permet de trouver de nombreuses informations. Entre autre : https://stats.stackexchange.com/.

La fenêtre de `RStudio` se divise en quatre sous-fenêtres : :

1. un éditeur de texte (fichier `.R` ou `.Rmd`)
2. l'espace de travail ou d'historique de commandes
3. le navigateur de fichiers, graphiques, packages, documentations
4. la console `R`.

Enfin, le très bon livre de Vincent Goulet *Introduction à la programmation en R*, en accès libre sur le site du CRAN :

https://cran.r-project.org/doc/contrib/Goulet_introduction_programmation_R.pdf

Afin de profiter de ce TP, lisez, exécutez et/ou modifiez (on enregistre mieux en agissant) chacune des lignes qui suivent ... en commençant par celles-ci.

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

2. Bases du langage

2.1 Commandes

Deux types de commandes

Expressions

```
cos(pi)
```

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

```
log(1)
```

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

Affectations et expressions

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

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

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

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

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; 12/3; 2^3; sqrt(16)
```

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

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

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

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

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

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

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

```
# et
FALSE & TRUE
```

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

```
# ou
FALSE | TRUE
```

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

Question : Introduire deux variables `x` et `y` contenant des valeurs. Ecrire l'expression qui renvoit `TRUE` si `x` et `y` ne sont pas égales.

2.2 Modes, longueurs et classes de données

Dans R, tout est un objet. Les principaux modes sont :

- `numeric` : nombres réels
- `character` : chaînes de caractères
- `logical` : valeurs logiques vrai/faux
- `list` : liste, collection d'objets
- `function` : fonction

Les objets de mode `numeric`, `character` et `logical` peuvent contenir des données d'un seul type. Au contraire, les objets de mode `list` peuvent contenir différents types d'objets.

On peut accéder au mode d'un objet avec la fonction `mode()` :

```
# La fonction de concatenation c() permet de créer des vecteurs
# Note: Les parenthèses servent aux fonctions
age=c(33,28, 33)
mode(age)
```

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

```
noms <- c('Siham', 'Jeanne', 'Ting')
mode(noms)
```

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

```
ma.liste <- list(noms=noms, age=age)
mode(ma.liste)
```

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

```
mode(is.integer(pi))
```

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

```
mode(mode)
```

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

```
print(ma.liste)
```

```
## $noms
## [1] "Siham"  "Jeanne" "Ting"
##
## $age
## [1] 33 28 33
```

La longueur d'un objet est donné par la fonction `length` :

```
length(age)
```

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

```
length(noms)
```

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

```
length(ma.liste)
```

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

Un objet spécial est la valeur manquante `NA`. Par défaut, son mode est `logical`, cependant `NA` n'est ni `TRUE` ni `FALSE`. Pour tester si une valeur est manquante on utilisera la fonction `is.na()` :

```
NA==NA
```

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

```
is.na(NA)
```

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

```
is.na(mean(c(1,4,NA)))
```

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

La *classe* d'un objet spécifie son interaction avec opérations et fonctions. Les data frame sont des listes spéciales dont les éléments ont tous la même longueur. La classe d'un data frame est différente de celle des listes standards et les data frame ont un système d'indiçage qui n'existe pas pour les autres listes :

```
class(ma.liste)
```

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

```
mon.data.frame=data.frame(noms,age)
mode(mon.data.frame)
```

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

```
class(mon.data.frame)
```

```
## [1] "data.frame"
```

Question : Essayer les commandes suivantes. Puis changer l'âge de romain dans `mon.data.frame`.

```
# ma.liste[1,2]
# mon.data.frame[1,2]
# ma.liste[[1]][2]
# mon.data.frame[[1]][2]
# ma.liste$noms
# ma.liste$noms[2]
# mon.data.frame$noms[2]
```

2.3 Vecteurs et matrices

2.3.1 Vecteurs

En `R`, le vecteur est l'élément de base pour les calculs : un scalaire est un vecteur de longueur un. La fonction la plus utilisée pour créer un vecteur est la concaténation :

```
prix <- c(150, 162, 155, 157); prix
```

```
## [1] 150 162 155 157
```

L'indiçage est fait par les crochets et non les parenthèses, qui servent aux fonctions.

```
# le premier indice n'est pas 0 comme en python, mais 1
prix[1]
```

```
## [1] 150
```

```
prix[c(1,3)]
```

```
## [1] 150 155
```

```
prix[1:3]
```

```
## [1] 150 162 155
```

```
prix[-(1)]
```

```
## [1] 162 155 157
```

On peut aussi utiliser un vecteur booléen pour indicer : les éléments extraits sont ceux correspondants aux valeurs `TRUE`.

```
prix[prix>156]
```

```
## [1] 162 157
```

On peut utiliser l'indiçage pour changer un élément :

```
prix[1] <- 0; prix
```

```
## [1]   0 162 155 157
```

Il est possible de donner des étiquettes aux éléments d'un vecteur et d'extraire des éléments sur la base de celles-ci :

```
names(prix)
```

```
## NULL
```

```
names(prix) <- c('model.1', 'model.2', 'model.3', 'model.4')
prix
```

```
## model.1 model.2 model.3 model.4
##       0     162     155     157
```

```
prix['model.3']
```

```
## model.3
##     155
```

Dans un vecteur, tous les éléments ont le même mode.

```
x <- c(1,2,'a', 'b'); x
```

```
## [1] "1" "2" "a" "b"
```

```
mode(x)
```

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

Pour générer le vecteur des n premiers entiers on utilise la syntaxe `1:n`

```
1:10
```

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

```
2:6
```

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

Pour générer des suites plus générales on utilise la fonction `seq()` :

```
seq(from=2, to=20, by=2)
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```
# ou plus simplement
seq(2,20,2)
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

On peut créer un vecteur d'éléments répétés avec `rep()` :

```
rep(1,5)
```

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

```
rep(NA,4)
```

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

2.3.2 Matrices

Une matrice est un vecteur avec un attribut `dim` de longueur deux. Tous les élèments d'une matrice ont donc le même mode. Pour créer une matrice :

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

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

Par défaut `matrix()` remplit la nouvelle matrice par colonne. L'indiçage se fait avec les crochets :

```
M[2,] # 2e ligne
```

```
## [1] 2 4 6
```

```
M[,3] # 3e colonne
```

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

```
M[2,3]
```

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

```
M[3]
```

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

```
M[,-2] # extrait toutes les colonnes sauf la 2e
```

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

Pour fusionner verticalement (horizontalement) deux matrices on utilise `rbind()` (resp. `cbind()`) :

```
cbind(M,-M)
```

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

```
rbind(M,2*M)
```

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

```
## [2,]    2    4    6
## [3,]    2    6   10
## [4,]    4    8   12
```

2.3.3 Opérations sur vecteurs et matrices numériques

Elément par élément :

```
v <- c(3,4,1,6)
v + 2
```

```
## [1] 5 6 3 8
```

```
v * 2
```

```
## [1]  6  8  2 12
```

```
v * v
```

```
## [1]  9 16  1 36
```

```
v/2
```

```
## [1] 1.5 2.0 0.5 3.0
```

```
v/v
```

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

```
v + v^2
```

```
## [1] 12 20  2 42
```

```
sqrt(M)
```

```
##          [,1]     [,2]     [,3]
## [1,] 1.000000 1.732051 2.236068
## [2,] 1.414214 2.000000 2.449490
```

```
M * M
```

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

```
# Essayer les commandes suivantes:
# M + v
# M + v[1:3]
```

Transposée, produit matriciel, inverse :

```
t(M)
```

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

```
N <- M[,-3]
N %*% diag(1,2)
```

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

```
## [2,]    2    4
```

```
solve(N)
```

```
##      [,1] [,2]
## [1,]   -2  1.5
## [2,]    1 -0.5
```

```
solve(N) %*% N
```

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

La transposée d'un vecteur est une matrice-ligne :

```
V <- t(v)
dim(V)
```

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

```
t(V)
```

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

Faire attention aux exemples suivants :

```
v %*% t(v)
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    9   12    3   18
## [2,]   12   16    4   24
## [3,]    3    4    1    6
## [4,]   18   24    6   36
```

```
t(v) %*% v
```

```
##      [,1]
## [1,]   62
```

```
v %*% v
```

```
##      [,1]
## [1,]   62
```

2.4 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 <- as.factor(ville); fact.ville
```

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

```
class(fact.ville)
```

```
## [1] "factor"
```

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

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

Un facteur a le mode `numeric` : en effet ses éléments sont stockés comme les entiers énumérant les différentes modalités :

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

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

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

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

2.5 Listes

Les listes peuvent stocker des éléments de n'importe quelle mode (y comprises d'autres listes).

Comme un vecteur, une liste est indicée par l'opérateur `[ ]`. Cependant, cela retourne une liste contenant l'élément souhaité :

```
ma.liste[1]
```

```
## $noms
## [1] "Siham"  "Jeanne" "Ting"
```

```
mode(ma.liste[1])
```

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

Pour obtenir directement l'élément, on utilise donc l'opérateur `[[ ]]` ou l'opérateur `$` suivi par le nom de l'élément (si disponible : un bon réflexe est de nommer les éléments d'une liste) :

```
ma.liste[[1]]
```

```
## [1] "Siham"  "Jeanne" "Ting"
```

```
ma.liste$age
```

```
## [1] 33 28 33
```

Les éléments d'une liste peuvent avoir des longueurs différentes :

```
ma.liste$ville <- ville
ma.liste
```

```
## $noms
## [1] "Siham"  "Jeanne" "Ting"
##
## $age
## [1] 33 28 33
##
## $ville
## [1] "paris"  "lyon"   "lyon"   "paris"  "nantes"
```

2.6 Data frames

Les data frame sont largement utilisés pour contenir des données. Ce sont des listes de classe `data.frame` dont tous les éléments ont la même longueur. Normalement, dans un data frame les colonnes sont les **variables** 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, TRUE, TRUE)
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 transformée en facteur
```

```
## [1] id.442 id.443 id.444 id.445 id.446 id.447 id.448 id.449
## Levels: 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$Age[data1$Fumeur==FALSE]
```

```
## [1] 45 17
```

Les colonnes sont directement accessibles dans l'espace de travail (sans devoir taper le nom du data frame et le $) après avoir attaché le data frame :

```
attach(data1)
Age
```

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

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 : `tail()`. Enfin, très pratique, pour obtenir un résumé façon statistiques descriptives :

```
summary(data1)
```

```
##        Id         Age          Fumeur
```

```
##  id.442 :1   Min.   :17.00   Mode :logical
##  id.443 :1   1st Qu.:25.00   FALSE:2
##  id.444 :1   Median :32.50   TRUE :6
##  id.445 :1   Mean   :36.62
##  id.446 :1   3rd Qu.:47.00
##  id.447 :1   Max.   :67.00
##  (Other):2
```

2.7 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
```

2.8 Structures de contrôle

___ Traitement conditionnel, If :___

```
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` :

```
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` : Que fait le code suivant ?

```r
x <- sample(c('pile','face'), 1, prob = c(0.8,0.2))
if(x == 'face'){
  print(paste(x, ', gagne', sep=''))
}else{
  while(x != 'face'){
    print(paste(x, ", perdu", sep=''))
    x <- sample(c('pile','face'), 1, prob = c(0.8,0.2))
  }
  print(paste(x, ', gagne', sep=''))
}
```

```
## [1] "pile, perdu"
## [1] "face, gagne"
```

2.9 Importation et exportation des données

Plusieurs fonctions permettent d'importer des données suivant le type d'enregistrement. Pour du `txt`, on utilise read.table, pour du `csv`, read.csv, pour des fichiers excel, le package `xlsx` ... et si les données proviennent d'un dataframe de `R`, on privilégiera l'extension `Rdata` pour l'enregistrement.

Pour `read.table`, les trois arguments les plus importants sont :

- `file` : nom (et adresse) du fichier, entre guillemets.
- `header` : les éléments de la première ligne sont ou ne sont pas les noms des colonnes (`TRUE` ou `FALSE`).
- `sep` : caractére séparant les éléments d'une ligne.

`read.table()` renvoie un data frame.

```r
#write.table(df1, file='Iris.txt' , sep= ';')
df1<-read.table('Iris.txt', header= TRUE, sep=';')


#save(df2, file = "heart.Rdata")
load(file='heart.Rdata')
dim(df2); names(df2)
```

```
## [1] 270  13
```

```
##  [1] "age"          "sexe"         "type_douleur" "pression"     "cholester"
##  [6] "sucre"        "electro"      "taux_max"     "angine"       "depression"
## [11] "pic"          "vaisseau"     "coeur"
```

Question : Utilisez la fonction `summary` sur `df1` et `df2`, puis décrivez brièvement les données.

Si les données sont stockées (ou doivent être sauvegardées) localement, il est nécessaire de connaître (et pouvoir modifier) le répertoire de travail :

```r
getwd() # Ou suis-je ?
setwd('~/Documents') # Dans quel état gère-t-on nos fichiers !?
```

On rappelle que dans les machines Linux et OS, `~/` est un raccourcie pour `/Users/nom_utilisateur`. Pour les machines Windows, la syntaxe des adresses est légèrement différente. Par exemple, on utilise `\` à la place de `/` .