S01: Introduction

Analyse de données quantitatives avec R

Samuel Coavoux

Le langage R

Généralités

Historique

Héritier de S, l'un des premiers langages de programmation statistique, développé par Bell à partir de 1975.

R est une implémentation *libre* de S. *Libre* signifie ici *gratuit* et *open source*.

Longtemps employé principalement par des statisticiens ; désormais employé dans la plupart des champs scientifiques.

En sociologie, encore un jeune langage: poids de SAS dans la statistique publique française ; poids de SPSS, dédié aux sciences sociales, à l'international. Mais R est le langage dominant dans les *jeunes* générations de sociologues quantitativistes.

Programmation et statistique

R est un **langage de programmation** (Turing-complete). Il pourrait être employé pour écrire des logiciels, comme d'autres langages (C, Python, Java...).

Il est cependant spécifiquement orienté vers l'**analyse statistique**. Il est par conséquent peu adapté pour écrire d'autres logiciels que statistiques.

Spécificités de R

- Un langage **évolutif**. Il y a:
  - un **cœur**, les fonctions de base du langage ("R-base", développé par la "R core team") ;
  - des **packages**, qui sont des ensembles de fonctions écrits par des programmeurs indépendants (de la R core team).
- Un langage **interprété** adapté à l'usage **interactif**.
- Un langage de programmation **orienté objet**.
- Des **fonctionnalités graphiques** avancées.

Rstudio

Rstudio est un **Integrated Development Environment** (IDE) dédié à R, **open-source** et **gratuit** (dans sa version individuelle) développé par la société Rstudio. Il s'agit d'une interface qui facilite la programmation, interactive ou non:

- vue d'ensemble de l'environnement
- aide intégrée
- autocomplétion des fonctions
- aide à la mise en forme

Il existe d'autres moyens d'utiliser R: *Rgui* (environnement graphique diffusé avec R), *Rcmdr* (environnement graphique avec un menu type SPSS), *R* dans un terminal.

Prise en main

Prise en main de Rstudio

Quatre fenêtres:

- **Source**: pour écrire le code.
- **Console**: pour executer le code/écrire les instructions que l'on ne souhaite pas garder en mémoire
- **Environment / history / git**: pour voir les objets créés
- **Help / Plots / Packages**: pour voir l'aide et les graphiques

Où écrire

Dans le script (fenêtre, source), toutes les lignes peuvent être *executés* (c'est-à-dire envoyée vers la console) en cliquant sur Run ou en faisant ctrl+entrée.

Si vous souhaitez écrire un commentaire (quelque chose qui n'est pas un code R), vous pouvez commencer une ligne par #. La ligne ne sera pas executée si vous l'envoyez dans la console. (cela peut également servir à "annuler" des lignes de code dont on ne veut pas qu'elles s'executent).

Le bouton source permet enfin d'éxecuter l'ensemble du script.

R est d'abord une grosse calculatrice

```
1 + 3
```

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

```
450 / 78
```

```
## [1] 5.769231
```

```
exp(-2)
```

```
## [1] 0.1353353
```

Les opérateurs de calculs

On peut utiliser:

- opérations arithmétiques fondamentales: `+`, `-`, `*`, `/`
- exposant: `^` (`2^2`)
- reste d'une division euclidienne `%%` (`5 %% 2 = 1`)
- quotient d'une division euclidienne `%%` (`5 %/% 2 = 2`)

R respecte les règles de précédence de l'arithmétique => `*` et `/` sont exécutés avant `+` et `-`. Comme en arithmétique, on peut employer des parenthèses.

```
2 + 3 * 2
```

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

```
(2 + 3) * 2
```

```
## [1] 10
```

Objets

Principe : on crée des **objets**, qui stockent des **informations**, et à partir desquels on opère divers **traitements**.

Vocabulaire : `<-` est l'"opérateur d'assignement". Il permet de stocker le résultat d'une opération dans un objet. Il s'emploie ainsi:

```
nom_objet <- opération
```

Si aucun objet n'existe avec ce nom, l'objet est **créé** par cette opération ; si un objet de ce nom existe déjà, il **disparaît** et est remplacé par le résultat de l'opération.

Objets : exemples basiques

```
x <- -8
y <- 9
x+8
```

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

```
truc <- x+y
truc
```

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

Objets

Nature des objets

Dans R, **tout** est un objet. En effet, on peut distinguer globalement deux types d'objets:

- les vecteurs, qui contiennent des données (`x <- 1`: `x` est un vecteur contenant la valeur 1).
- les fonctions, qui contiennnent des instructions pour manipuler des vecteurs (`mean(x)` calcule la moyenne du vecteur `x`).

Tous les objets que vous avez créé au cours d'une session de travail sont affichés dans l'onglet environment de Rstudio ou en employant la fonction `ls()`.

Syntaxe des noms d'objets

- Éviter les **caractères accentués**
- R est sensible a la **casse** : `x` et `X` sont deux objets différents
- Éviter les noms tels que c, q, t, F, T, max, min, data, qui sont des noms de fonctions
- Les **espaces** sont interdits : utiliser . ou _ (privilégier _ car . est utilisé dans les noms de fonctions pour désigner des méthodes).

Inspecter un objet : `print()`

La fonction `print()` permet d'imprimer dans la console le contenu d'un objet. La sortie exacte dépend de la nature de l'objet.

```
x <- 7
print(x)
```

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

Pour tous les objets, taper le nom de l'objet dans la console est équivalent à `print()`.

```
x
```

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

Les vecteurs

Vecteurs

Le **vecteur** est un type d'objet qui contient des valeurs dans R. Tous les objets de R qui ne sont pas des fonctions sont des vecteurs.

On les différencie principalement par leur **classe**. `class(x)` renvoie la classe de l'objet `x`.

On distingue les vecteurs unidimensionnels des vecteurs pluridimensionnels:

- les *vecteurs unidimensionnels* sont des séries de valeurs homogènes (seulement des nombre ; seulement des booléens ; etc.).
- les *vecteurs pluridimensionnels* sont des assemblages de vecteurs unidimensionnels divers (à l'exception des matrices).

Classes de vecteurs

- *vecteurs unidimensionnels* :
  - numeric
  - integer
  - character
  - factor
  - logical
- *vecteurs pluridimensionnels* :
  - list
  - matrix
  - data.frame

Créer un vecteur

- operateur `c(...)` (pour combine ou concatenate) : combiner des valeurs
- operateur `rep(x, times)` (pour replicate)
- operateur `seq(from, to, by)` (pour sequence) : créer des séries équidistantes de nombres.

Vecteurs unidimensionnels

Numeric

```
tailles <- c(167, 192, 173, 174, 172, 167, 171)
tailles
```

```
## [1] 167 192 173 174 172 167 171
```

```
class(tailles)
```

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

Integer

Pour différencier un integer d'un numeric, on ajoute un L. Les valeurs s'écrivent sans guillemets.

```
serie <- seq(1L, 10L, by = 1L)
serie
```

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

```
class(serie)
```

```
## [1] "integer"
```

Character

Les valeurs d'un vecteur character s'écrivent avec des guillemets. Elles peuvent contenir tous les caractères, y compris des espaces (et donc des sauts de lignes, etc.)

```
dipl <- c("BEP", "CAP", "Bac", "Licence", "CAP",
          "Bac", "Master")
dipl
```

```
## [1] "BEP"     "CAP"     "Bac"     "Licence"
## [5] "CAP"     "Bac"     "Master"
```

```
class(dipl)
```

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

Logical

```
vr <- c(TRUE, FALSE, TRUE, TRUE)
# Il est légal mais déconseillé d'employer
# T pour TRUE et
# F pour FALSE
vr
```

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

```
class(vr)
```

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

Factor

Les factor sont une classe à part. En termes de mémoire, il s'agit d'integer ; en termes d'apparence et de comportement, ils ressemblent à des character. Il s'agit largement d'un héritage de S: à l'époque, le fait de stocker des vecteurs character sous format numérique produisait un gain de performance considérable, qui est aujourd'hui le plus souvent négligeable.

Les factor sont des vecteurs character dont le nombre de valeurs différente est fini et connu. On les appelle ses "levels".

Factor: levels

```
dipl <- c("BEP", "CAP", "Bac", "Licence", "CAP",
          "Bac", "Master")
class(dipl)
```

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

```
dipl <- factor(dipl, levels = c("BEP", "CAP", "Bac", "Licen
class(dipl)
```

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

Attributs des vecteurs

Les vecteurs ont une classe, accessible avec `class()`.

Les vecteurs unidimensionnels ont également une taille, accessible avec `length()`

```
dipl <- c("BEP", "CAP", "Bac", "Licence", "CAP",
          "Bac", "Master")
length(dipl) # nombre de valeurs
```

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

On note que tous les vecteurs ont une taille, même lorsqu'il n'y a qu'une seule valeur (dans ce cas, le vecteur est de taille 1). Cela différencie R de nombreux autres langages dans lesquels une *valeur* est différente d'une *série de valeur* (d'un vecteur).

Attributs des vecteurs: valeurs et modalités

On trouve les valeurs uniques que prend un vecteur avec `unique()`.

```
unique(dipl)
```

```
## [1] "BEP"     "CAP"     "Bac"     "Licence"
## [5] "Master"
```

Pour les factor, on peut également accéder aux valeurs avec `levels()`

```
dipl <- factor(dipl)
levels(dipl)
```

```
## [1] "Bac"     "BEP"     "CAP"     "Licence"
## [5] "Master"
```

Coup d'œil sur les vecteurs

Pour obtenir des informations de base, on peut utiliser `str()` (structure), qui donne des informations sur la nature du vecteur, ou `summary()` qui s'adapte à la classe pour donner les informations les plus pertinentes (tendance centrale et dispersion pour un vecteur numeric ou integer, tri à plat pour un factor...)

```
str(1:10)
```

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

```
summary(1:10)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00
```

Coup d'œil sur les vecteurs

```
str(dipl)
```

```
##  Factor w/ 5 levels "Bac","BEP","CAP",..: 2 3 1 4 3 1 5
```

```
summary(dipl)
```

```
##      Bac      BEP      CAP Licence  Master
##        2        1        2        1        1
```

Vecteurs pluridimensionnels

Les vecteurs pluridimensionnels

Il y en a trois types:

- matrix
- list
- data.frame

Matrice

Une matrix n'est pas un assortiment de vecteur, mais un vecteur unique à deux ou plusieurs dimensions.

```
# Première manière de créer une matrice : on crée
# un vecteur et on lui définit deux dimensions
x <- 1:12
x
```

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

```
dim(x) <- c(3,4)
x
```

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

Matrice (2)

```
# Deuxième manière : la fonction matrix
y <- matrix(1:12, nrow=3)
y
```

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

```
y <- matrix(1:12, nrow=3, byrow=TRUE)
y
```

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

Dimensions d'une matrice

```
dim(y)
```

```
## [1] 3 4
```

```
nrow(y)
```

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

```
ncol(y)
```

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

```
length(y)
```

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

Listes

Une **liste** est un objet composé d'un ou plusieurs vecteurs. Ces vecteurs peuvent être de n'importe quel type et de n'importe quelle taille.

```
maliste <- list(lettres = c("a", "b", "c", "d"),
                nombres = c(64, 75, 85, 62, 54, 45, 75),
                valeur_unique = 45)
```

Listes

```
maliste
```

```
## $lettres
## [1] "a" "b" "c" "d"
##
## $nombres
## [1] 64 75 85 62 54 45 75
##
## $valeur_unique
## [1] 45
```

Listes: accès

On peut accéder à un vecteur unique stocké dans une liste en employant $ ou [[. Avec $,on emploie le **nom** des éléments ; avec [[, on peut employer soit leur **nom**, soit leur **index** (un integer indiquant leur position dans la liste, en partant de 1)

```
maliste$lettres
```

```
## [1] "a" "b" "c" "d"
```

```
maliste[["nombres"]]
```

```
## [1] 64 75 85 62 54 45 75
```

```
maliste[[3]]
```

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

```
# Incorrect:
# maliste$1
```

Listes: attributs

Une liste à une longueur accessible avec `length()`, qui correspond au nombre d'éléments stockés ; elle a des noms, accessibles avec `names()`, qui renvoie un vecteur character. `str()` décrit la liste

```
length(maliste)
```

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

```
names(maliste)
```

```
## [1] "lettres"       "nombres"
## [3] "valeur_unique"
```

Listes: attributs

```
str(maliste)
```

```
## List of 3
##  $ lettres       : chr [1:4] "a" "b" "c" "d"
##  $ nombres       : num [1:7] 64 75 85 62 54 45 75
##  $ valeur_unique: num 45
```

data.frame

Un **data.frame** est une liste composée de vecteurs de même taille. Ces vecteurs peuvent être de différents types (contrairement aux matrices). C'est la classe qui permet de stocker des **bases de données**. Elle est spécifique à R, et elle est primordiale dans R.

```
df <- data.frame(sexe = c("F", "F", "M"),
                 dipl = c("Bac", "Licence", "CEP"),
                 age = c(32, 45, 65))
str(df)
```

```
## 'data.frame':    3 obs. of  3 variables:
##  $ sexe: Factor w/ 2 levels "F","M": 1 1 2
##  $ dipl: Factor w/ 3 levels "Bac","CEP","Licence": 1 3 2
##  $ age : num  32 45 65
```

data.frame: stringsAsFactors

Par défaut, la fonction data.frame transforme les vecteurs character en factor. Pour le désactiver, utiliser `stringsAsFactors = FALSE`.

```
df <- data.frame(sexe = c("F", "F", "M", "F"),
                 dipl = c("Bac", "Licence", "CEP", "BEP"),
                 age = c(32, 45, 65, 42),
                 stringsAsFactors=FALSE)
str(df)
```

```
## 'data.frame':    4 obs. of  3 variables:
##  $ sexe: chr  "F" "F" "M" "F"
##  $ dipl: chr  "Bac" "Licence" "CEP" "BEP"
##  $ age : num  32 45 65 42
```

data.frame: accès

Comme la liste, on peut accéder aux vecteurs qui composent le data.frame avec $ et avec [[.

```
df$sexe
```

```
## [1] "F" "F" "M" "F"
```

```
df[["dipl"]]
```

```
## [1] "Bac"     "Licence" "CEP"     "BEP"
```

```
df[[3]]
```

```
## [1] 32 45 65 42
```

data.frame

Le data.frame sert donc à stocker des bases de données. Chaque variable va dans un vecteur. L'obligation d'avoir des vecteurs de taille égale implique que toutes les variables soit renseignées (ou aient une valeur manquante, `NA`) pour chaque individu.

Par conséquent, chaque ligne d'un data.frame correspond à un individu statistique.

```
print(df)
```

```
##   sexe     dipl age
## 1    F      Bac  32
## 2    F Licence  45
## 3    M      CEP  65
## 4    F      BEP  42
```

data.frame: attributs

Comme la liste, le data.frame a une taille accessible par `length()`: le nombre de vecteurs stockés ou nombre de variables. On peut également utiliser `ncol()`. `names()` renvoie les noms des vecteurs. `nrow()` donne le nombre d'individus. Là encore, `str()` et `summary()` permettent une vue d'ensemble.

```
length(df)
```

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

```
ncol(df)
```

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

```
names(df)
```

```
## [1] "sexe" "dipl" "age"
```

Fonctions

Principe des fonctions

Nous avons déjà vu de nombreuses fonctions: `str()`, `summary()`, `length()`, `data.frame()` etc.

Les fonctions sont des **objets**, comme les vecteurs. Cela signifie que vous pouvez *en créer de nouvelles* (on verra comment dans les dernières séances) et *accéder et manipuler* le contenu de celles qui existent (en tapant le nom d'une fonction sans parenthèses).

```
ls()
```

```
## [1] "df"      "dipl"    "maliste" "serie"
## [5] "tailles" "truc"    "vr"      "x"
## [9] "y"
```

```
# afficher le code de ls()
# ls
```

Contenu d'une fonction

Une fonction:

- a un **nom** (attention à la casse : les fonctions `LDA()` et `lda()`) ;
- accepte des **arguments** ;
- *renvoie* (*return*) un **resultat** ;
- peut effectuer une **action** (dessiner un graph, lire un fichier, etc.).

Exemple : L'opérateur `c()` est une fonction qui crée un vecteur

Arguments

- Entre parenthèses apres le nom de la fonction
- Permet de préciser ce que l'on souhaite faire : objet sur lequel s'applique la fonction, parametrage...
- Un argument utilisé dans plusieurs fonctions : `na.rm=TRUE` : demande à R d'ignorer les valeurs manquantes dans le calcul.

```
ages <- c(167, NA, 192, 168)
mean(ages)
```

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

```
mean(ages, na.rm=TRUE)
```

```
## [1] 175.6667
```

Opérateurs

Tous les objets dans R sont soit des vecteurs, soit des fonctions. Cela signifie que les opérateurs que l'on a vu, qui ne sont pas des vecteurs, sont aussi des fonctions: `+ - / * %% <- [ [[ $`.

```
4 + 5
```

```
## [1] 9
```

```
`+`(4, 5)
```

```
## [1] 9
```

```
maliste$lettres
```

```
## [1] "a" "b" "c" "d"
```

```
`$`(maliste, "lettres")
```

```
## [1] "a" "b" "c" "d"
```

Bonnes pratiques

Premières régles de programmation

Source ou console?

Comment savoir si l'on doit taper une instruction dans la source ou dans la console ? La seule question à se poser est la suivante : si je dois refaire l'analyse demain, est-ce que j'aurais à nouveau besoin de cette instruction?

- instructions qui modifient R ou Rstudio (`install.packages()`, `update.packages()`, etc.) -> dans la console
- instructions qui me permettent de savoir où j'en suis de l'analyse (`str(x)`, `ls()`, etc.) -> le plus souvent dans la console
- instructions qui modifient des objets et/ou produisent des résultats statistiques -> **dans la source**

Commentaires

Commenter votre code ! L'intérêt de la programmation statistique est que l'on garde une trace écrite de ce que l'on fait. Mais encore faut-il pouvoir relire et comprendre cette trace. La règle du commentaire : **vous devez pouvoir comprendre ce que vous avez fait si vous ouvrez un fichier un mois après y avoir touché pour la dernière fois**.

Usages des commentaires:

- faire des parties, structurer le code (également, utiliser des scripts différents pour chaque opération)
- expliquer les passages complexes (usages inhabituel d'une fonction, création d'une fonction spécifique, imbrication de multiples fonctions)
- expliquer les étapes d'un travail

Propreté du code : espaces

Quelques conseils de base (cf. `http://adv-r.had.co.nz/Style.html`, `https://google.github.io/styleguide/Rguide.xml`)

+ employer des espaces autour des opérateurs

```
# bien
x <- 5 * 3 / 8
# pas bien
x<-5*3/8
```

- ajouter un espace après la virgule dans les fonctions

```
# bien
data.frame(x = 1:10, y = 2:11)
# pas bien
data.frame(x = 1:10,y = 2:11)
```

Propreté du code : indentation

Indentez les arguments dans une fonction lorsque la ligne est trop longue.

```
# bien
x <- data.frame(x = c(45, 56, 45, 65, 45),
                y = c(75, 69, 41, 32, 47))
# pas bien
x <- data.frame(x = c(45, 56, 45, 65, 45), y = c(75, 69, 41
```

Propreté du code : commentaires

Utiliser les commentaires pour faire des sections, pour expliquer des passages, expliquer des bouts de fonction.

```
##### import des données #####
x <- read.csv(...)
## Recoder les données
x <- data.frame(y = ...,
                z = ...,
                w = factor(...), # w doit être factor
                a = 1:10)
```

Aide et ressources

Inspecter les messages d'erreur

La principale difficulté de la programmation est la tendance à abandonner face aux erreurs ; la seule qualité dont vous avez besoin, c'est la persévérance et la patience. Lorsque vous tombez sur un message d'erreur:

- Repérez quelle est la partie du code qui le produit
- Vérifiez si vous n'avez pas oublié une étape.
- Lisez le message attentivement : ressemble-t-il à un message connu? Dans ce cas, la solution est connue également.
- Est-ce que le message suggère une solution? Dans ce cas, essayez-là.
- Enfin, si rien ne fonctionne, isolez la partie générique du message et cherchez la solution sur Internet/dans un livre

`https://bimestriel.framapad.org/p/Reur`

Help

L'*aide intégrée* à R inclut, pour chaque fonction, son utilité, ses arguments ainsi que leurs valeurs par défaut, et la nature de l'objet produit.

```
help("str")
?str
```

Ressources internet

Sinon, en général, tout problème trouve sa solution en demandant à **Google** ou à **Stack OverFlow**. Notez que le premier renvoie le plus souvent vers le second.

stackoverflow.com/questions/tagged/r

StackOverflow est une bonne ressource pour apprendre : allez lire des questions et essayer de repérer vous-mêmes les problèmes, puis allez lire les réponses proposée. Rapidement, vous pourrez proposer vous-mêmes des réponses/poser des questions.

Ressources écrites

Cornillon P.A., Guyader A., Husson F., Jégou N., Josse J., Kloareg M., Matzner-Løber E., Rouvière L., *Statistique avec R*, Presses universitaires de Rennes, 2013

Wickham, H., *Advanced R*, Chapman & Hall, 2014

Récapitulatif

R fonctionne avec des *objets*. On en manipule principalement deux : les *vecteurs* et les *fonctions*. Les vecteurs contiennent les données. Les fonctions prennent des *arguments*, modifient un vecteur et *retournent* un résultat.

Les résultats sont affichés (*print()*) dans la console. Pour qu'ils soient stockés en mémoire, il faut les *assigner* dans un objet.

Récapitulatif : types de vecteurs

- *vecteurs unidimensionnels*:
  - numeric
  - integer
  - character
  - factor
  - logical
- *vecteurs pluridimensionnels*:
  - list
  - matrix
  - data.frame

Récapitulatif : principales fonctions abordées

- c() crée un vecteur
- data.frame() crée un data.frame
- matrix() crée une matrice
- names() retourne le nom des colonnes
- length() retourne la taille (nombre d'éléments) de l'objet
- dim() retourne les dimensions de l'objet
- help() : votre meilleuR ami

Exercice

On a demandé à 4 ménages le *revenu du chef de ménage*, celui de son *conjoint*, et le *nombre de personnes* du ménage :

- Ménage 1 : Chef = NA ; Conjoint = 1450 ; Nb pers = 4 ; Statut = marié
- Ménage 2 : Chef = 1180 ; Conjoint = NA ; Nb pers = 2 ; Statut = célibataire
- Ménage 3 : Chef = 1750 ; Conjoint = 1690 ; Nb pers = 3 ; Statut = pacs
- Ménage 4 : Chef = 2100 ; Conjoint = NA ; Nb pers = 1 ; Statut = célibataire