Exercices de programmation R - corrigé

Marie Laure Delignette-Muller

2024-01-10

Contents

Concepts de base **2**

Création d'un objet R simple . . . 3

Création d'un vecteur ou d'une matrice . . . 6

Indexation d'un vecteur ou d'une matrice . . . 11

Les jeux de données **13**

Création d'un jeu de données directement dans R . . . 13

Indexation d'un jeu de données . . . 14

Ajout de colonnes calculées à partir des colonnes préexistantes . . . 15

Les facteurs . . . 16

Importation et exploration rapide d'un jeu de données . . . 16

Manipulation de données **18**

Importation d'un jeu de données plus complexe . . . 18

Recodage simple d'une variable numérique en facteur . . . 21

Modification des noms et de l'ordre des modalités d'un facteur et sélection de lignes . . . 21

Transformation d'une variable quantitative en variable qualitative par définition de classes . . . 23

Création d'une nouvelle variable quantitative par calcul . . . 25

Manipulation de données (*suite OPTIONNELLE pour les plus rapides*) **25**

Création des variables Revenu et RevenuF . . . 25

Recodage des nombres de chiens et nombres de chats . . . 27

Automatisation des calculs **28**

Structures conditionnelles `if` et itératives `for` . . . 28

Ecriture d'une fonction et application d'une fonction par groupe à l'aide de la fonction `tapply()` . . . 30

Sauvegarde du jeu de données augmenté **31**

Visualisation de données avec ggplot2 **32**

Concepts de base

A FAIRE en début, en cours et en fin de session Rstudio

- En début de session on **ouvre un script R** (ou .rmd si on veut d'emblée intégrer le script dans un rapport d'analyse), et si c'est un nouveau script on **le sauve d'emblée avec l'extension .R** (ou .Rmd pour un rapport d'analyse), dans le même dossier que celui où sont les fichiers de données (pas obligatoire mais plus simple).
- On **change le répertoire de travail** (Onglet Session, Set Working Directory) et on le définit au répertoire où se trouve le script et les données (option **To Source File location**).
- On **sauve le script très régulièrement** !!!
- En fin de session **Rstudio vous demande si vous voulez sauver l'espace de travail** (workspace). Il est fortement conseillé de **toujours refuser** pour éviter des problèmes (sauf cas très exceptionnels qui ne devraient pas vous concerner de suite).

A SAVOIR avant de commencer :

- R est sensible à la casse (majuscule / minuscule)
- Dans un code R , tout ce qui se trouve à droite du signe # est considéré comme un commentaire, donc non interprété comme du code (mais bien utile pour comprendre son code).
- Le séparateur de décimale dans R est le point `.`.
- Le symbole <- est utilisé pour affecter une valeur à un objet. Mieux vaut ne pas le remplacer par le symbole = même si souvent ça fonctionne (mais pas toujours !). Le symbole = est à réserver pour la définition des valeurs des arguments dans l'appel à une fonction.
- Nommez les objets R avec des noms parlants (pas trop courts donc) mais pas trop longs non plus.
- Evitez de donner des noms qui sont susceptibles de correspondre à des noms de fonctions R (ex. : `median`)
- Ne commencez pas le nom d'un objet par un chiffre.
- Evitez les accents, les espaces et les caractères spéciaux dans les noms des objets R ainsi que dans les fichiers de données que vous allez importer dans R (donc y penser bien en amont dès la saisie des données dans un tableur par exemple). Les seuls caractères spéciaux conseillés dans les noms d'objets, tant qu'ils ne sont pas en début de nom, sont `_` et `.` (ex. : `animal_prefere`).
- Dans les données, codez les variables quantitatives par des nombres (donc pas '`J1`, `J3`, `J10` par ex.) et les variables qualitatives par des chaînes de caractères (donc pas 1 pour les mâles et 0 pour les femelles par ex.). Cela vous évitera une étape de recodage dans R.
- Les données manquantes doivent être codées `NA` (pas de cellule vide dans un tableau de données que l'on souhaite importer dans R).

Quelques liens qui vous seront sans doute utiles :

Pour l'utilisation de R

- la "cheat sheet" de la base de R : https://iqss.github.io/dss-workshops/R/Rintro/base-r-cheat-sheet.pdf
- la carte de référence de R en français : https://www.apmep.fr/IMG/pdf/R_RefCard.pdf
- la carte de référence de R (version 2 plus moderne) en anglais : https://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf
- Le "cookbook" de R : http://www.cookbook-r.com/

Pour l'utilisation de rmarkdown pour inclure le code R (et les sorties) dans un rapport d'analyse

- le "cookbook" de rmarkdown : https://bookdown.org/yihui/rmarkdown-cookbook/
- la "cheat sheet" de rmardown : https://rstudio.github.io/cheatsheets/rmarkdown.pdf

Pour faire des graphes avec ggplot2

- le "cookbook" de ggplot2 : https://r-graphics.org/
- la "cheat sheet" de ggplot2 : https://rstudio.github.io/cheatsheets/data-visualization.pdf

Création d'un objet R simple

> **UTILISATION du signe d'affectation <-, des fonctions `class()`, `rm()`, `print()`, `paste()`, `is.numeric()`, `is.character()`, `is.logical()` et `as.numeric()`, `as.character()`, `as.logical()` et des opérateurs mathématiques et logiques (ET : &, OU : |, EGAL: ==, DIFFERENT DE !=, NON : !, SUPERIEUR >, SUPERIEUR ou EGAL>=, etc.),**

Pour vous approprier les bases, exécutez chaque ligne de code une à une en prenant le temps d'examiner ce qui est produit.

Pour la plupart des exercices, nous vous indiquons le nom des fonctions à utiliser. Ayez le réflexe à chaque fois d'aller voir l'aide en ligne des fonctions que vous ne connaissez encore pas bien en tapant `?nom_de_la_fonction`.

```
# --- les nombres (`numeric`) ---
age <- 45 # affectation d'une valeur à un objet R
age # pour voir l'objet
```

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

```
print(age) # une autre façon de voir l'objet
```

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

```
class(age) # pour connaître la classe de l'objet
```

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

```
rm(age) # pour effacer un objet de l'espace de travail
# regarder l'effet de cette commande dans onglet Environment en haut à droite de Rstudio

(age <- 45) # pour faire l'affectation et voir l'objet en même temps
```

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

```
(age_en_mois <- age * 12)
```

```
## [1] 540
```

```
(age_au_carre <- age^2)
```

```
## [1] 2025
```

```r
# --- les chaînes de caractères (`character`) ---
genre <- "masculin"
pays <- "Autriche"
class(genre)
```

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

```r
# collage de chaîne de caractères sans séparateur
(phrase <- paste("Ce répondant est de genre ", genre, ".", sep = ""))
```

```
## [1] "Ce répondant est de genre masculin."
```

```r
# --- les booléens (`logical`) ---
(jeune <- age < 30)
```

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

```r
class(jeune)
```

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

```r
# utilisation des opérateurs logiques
(homme_jeune <- (age < 30) & (genre == "masculin"))
```

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

```r
# autres façons d'écrire la même chose
(homme_jeune <- (age < 30) & (genre != "feminin"))
```

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

```r
(homme_jeune <- !(age >= 30) & (genre == "masculin"))
```

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

```r
(homme_jeune <- !((age >= 30) | (genre == "feminin")))
```

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

```r
# Vérification de la classe et changement de classe d'un objet
is.numeric(age)
```

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

```r
is.numeric(jeune)
```

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

```r
(jeune_num <- as.numeric(jeune))
```

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

```r
is.numeric(jeune_num)
```

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

EXERCICE pour appliquer ce que vous avez découvert précédemment :

1. Créez une phrase sous forme d'une chaîne de caractères avec la fonction `paste()` en utilisant les deux objets créés (`age` et `pays`) pour indiquer l'âge et le pays du répondant, et affichez-la à l'écran pour vérifier le résultat.
2. Créez un booléen indiquant si le répondant est autrichien d'au moins 18 ans et vérifier sa valeur bien sûr.
3. Créez un booléen indiquant si le répondant a moins de 16 ans ou plus de 70 ans.
4. Calculez l'âge du répondant en log en utilisant la fonction `log()`. Celle-ci donne-t-elle le logarithme népérien ou décimal de l'âge ? Trouvez à l'aide sur la fonction `log()` (en tapant `?log`) comment on obtient l'autre log.

```r
## 1
(phrase2 <- paste("Le répondant vient d'", pays, " et a ", age," ans."))
```

```
## [1] "Le répondant vient d' Autriche  et a  45  ans."
```

```r
## 2
(autrichien_majeur <- (pays == "Autriche") & (age >= 18))
```

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

```r
## 3
(potentiel_inactif <- (age < 16) | (age > 70))
```

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

```r
## 4
log(age) # sans faire de calcul on se doute que c'est le népérien car le décimal est en 1 et 2
```

```
## [1] 3.806662
```

```r
log10(age)
```

```
## [1] 1.653213
```

Création d'un vecteur ou d'une matrice

> **UTILISATION du calcul vectorisé et des fonctions** `c()`, `rep()`, `seq()`, `matrix()`, `rbind()`, `cbind()`. `rownames()`, `colnames()`, `is.vector()`, `is.matrix()`, `length()`, `dim()`, `nrow()`, `ncol()`.

Un vecteur est un objet R utilisé notamment pour stocker toutes les valeurs d'une variable en statistique. Il est à une dimension et ne contient que des éléments de même type. En statistique il permettra par exemple de stocker les valeurs d'une variable quantitative (vecteur de nombres), ou d'une variable qualitative (vecteur de chaînes de caractères). Une matrice, c'est la même chose en dimension 2, toujours ne contenant que des éléments de même type.

On peut créer un vecteur, par exemple,

- avec la fonction `c()` pour concaténer plusieurs valeurs,
- avec la fonction `rep()`pour répéter plusieurs fois une même valeur,
- avec la fonction `seq()` pour générer une séquence régulière de valeurs,
- ou par calcul à partir d'autres vecteurs créés auparavant. Le calcul est en effet vectorisé par défaut dans R, ce qui veut dire que si l'on écrit un calcul impliquant un ou plusieurs vecteurs, il fait automatiquement le calcul pour tous les éléments du (ou des) vecteur(s) impliqué(s), élément par élément (*i.e.* ligne à ligne).

Pour bien comprendre l'usage de ces fonctionnalités, exécutez chaque ligne de code une à une en prenant le temps d'examiner ce qui est produit.

```r
# Création de vecteurs avec c()
(Age <- c(71, 29, 45, 32, 81))
```

```
## [1] 71 29 45 32 81
```

```r
is.vector(Age)
```

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

```r
class(Age) # donne le type du vecteur
```

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

```r
length(Age)
```

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

```r
Pays <- c("Autriche", "Danemark", "Danemark", "UK", "Autriche")
class(Pays)
```

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

```r
Genre <- c("feminin", "feminin", "masculin", "masculin", "feminin")

# Création d'un vecteur avec rep()
(Annee <- rep(2023, times = 5))
```

```
## [1] 2023 2023 2023 2023 2023
```

```r
# Création de séquences régulières
(Numero <- 1:5)
```

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

```r
(Numero <- seq(from = 1, to = 5, by = 1))
```

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

```r
(Numero10en10 <- seq(from = 10, to = 60, by = 10))
```

```
## [1] 10 20 30 40 50 60
```

```r
# Ajout de valeurs
(Agecomplete <- c(Age, 14, 17))
```

```
## [1] 71 29 45 32 81 14 17
```

```r
class(Agecomplete)
```

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

```r
length(Agecomplete)
```

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

```r
# Que se passe-t-il si on concatène des valeurs qui ne sont pas du même type ?
(Agecomplete2 <- c(Age, "mineur", "mineur"))
```

```
## [1] "71"     "29"     "45"     "32"     "81"     "mineur" "mineur"
```

```r
class(Agecomplete2)
```

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

```r
# Calcul vectorisé
Age^2
```

```
## [1] 5041  841 2025 1024 6561
```

```r
Annee - Age
```

```
## [1] 1952 1994 1978 1991 1942
```

```r
(PaysETGenre <- paste(Pays, Genre, sep = "."))
```

```
## [1] "Autriche.feminin"  "Danemark.feminin"  "Danemark.masculin"
## [4] "UK.masculin"       "Autriche.feminin"
```

```r
(Majeur <- Agecomplete >= 18)
```

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

```r
class(Majeur)
```

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

EXERCICE pour appliquer ce que vous avez découvert précédemment (en vérifiant le résultat à chaque fois bien sûr) :

1. Créez un vecteur de type chaîne de caractères avec la fonction `paste()` en utilisant les deux vecteurs créés (`Genre` et `Age`) pour indiquer le genre et l'âge de chaque répondant, avec comme séparateur `_`.
2. Créez un vecteur de type booléen (`logical`) indiquant si le répondant est autrichien ou du Royaume Uni et a au moins 50 ans.
3. Créez un vecteur de type numérique codant avec un 1 les femmes et un 0 les garçons en utilisant le calcul vectorisé avec les opérateurs logiques et la fonction `as.numeric()`.

```r
## 1
(GenreETage <- paste(Genre, Age, sep = "_"))
```

```
## [1] "feminin_71"  "feminin_29"  "masculin_45" "masculin_32" "feminin_81"
```

```r
class(GenreETage)
```

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

```r
## 2
(autrichien_OU_UK_ET_sup30ans <- (Pays == "UK" | Pays == "Autriche") & (Age >= 50))
```

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

```r
(autrichien_OU_UK_ET_sup30ans <- (Pays != "Danemark" & Age >= 50)) # equivalent
```

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

```r
## 3
(Femme <- as.numeric(Genre == "feminin"))
```

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

```r
class(Femme)
```

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

Pour créer une matrice on peut par exemple utiliser la fonction `matrix()` ou les fonctions `rbind()` et `cbind()` qui permettent de juxtaposer des vecteurs de même type respectivement par ligne ou colonne. Voici quelques exemples à explorer par vous mêmes.

```r
# Création d'une matrice par remplissage par ligne
(Matrice1 <- matrix(c(1, 2, 3, 4, 11, 12, 13, 14), byrow = TRUE, ncol = 4))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]   11   12   13   14
```

```r
class(Matrice1)
```

```
## [1] "matrix" "array"
```

```r
is.matrix(Matrice1)
```

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

```r
is.vector(Matrice1)
```

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

```r
dim(Matrice1)
```

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

```r
nrow(Matrice1)
```

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

```r
ncol(Matrice1)
```

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

```r
# en changeant le nombre de colonnes
(Matrice2 <- matrix(c(1, 2, 3, 4, 11, 12, 13, 14), byrow = TRUE, ncol = 2))
```

```
##      [,1] [,2]
## [1,]    1    2
## [2,]    3    4
## [3,]   11   12
## [4,]   13   14
```

```r
# Création d'une matrice par remplissage par colonne
(Matrice3 <- matrix(c(1, 2, 3, 4, 11, 12, 13, 14), byrow = FALSE, ncol = 2))
```

```
##      [,1] [,2]
## [1,]    1   11
## [2,]    2   12
## [3,]    3   13
## [4,]    4   14
```

```r
# Création de noms de colonnes
colnames(Matrice3) <-
c("i", "10+i")
Matrice3
```

```
##      i 10+i
## [1,] 1   11
## [2,] 2   12
## [3,] 3   13
## [4,] 4   14
```

EXERCICE pour manipuler les matrices (en vérifiant le résultat à chaque fois bien sûr) :

1. Créez une nouvelle matrice de la même dimension que `Matrice3` ne contenant que des valeurs à 100.
2. Créez une nouvelle matrice comme la somme de `Matrice3` et de celle que vous venez de créer
3. Utilisez la fonction `colnames()` pour changer les noms de colonnes de cette matrice "100+i" et "110+i".
4. *En OPTIONNEL pour les plus rapides, créez une matrice sur le même format mais pour i allant de 1 à 9, en utilisant 1:9, nommez ces colonnes "100+i" et "110+i" et ses lignes "ligne_i" avec i allant de 1 à 9. Vous pouvez explorer plusieurs solutions.*

```r
## 1
(Matrice4 <- matrix(rep(100, times = 8), ncol = 2))
```

```
##      [,1] [,2]
## [1,]  100  100
## [2,]  100  100
## [3,]  100  100
## [4,]  100  100
```

```r
## 2
(Matrice5 <- Matrice3 + Matrice4)
```

```
##        i 10+i
## [1,] 101  111
## [2,] 102  112
## [3,] 103  113
## [4,] 104  114
```

```r
## 3
colnames(Matrice5)
```

```
## [1] "i"    "10+i"
```

```r
colnames(Matrice5) <- c("100+i", "110+i")
Matrice5
```

```
##      100+i 110+i
## [1,]   101   111
## [2,]   102   112
## [3,]   103   113
## [4,]   104   114
```

```r
## 4
Matrice6 <- matrix(rep(100, times = 18) + c(1:9, 11:19), ncol = 2, byrow = FALSE)
colnames(Matrice6) <- c("100+i", "110+i")
rownames(Matrice6) <- paste("ligne", 1:9, sep = "_")
Matrice6
```

```
##         100+i 110+i
## ligne_1   101   111
## ligne_2   102   112
## ligne_3   103   113
## ligne_4   104   114
## ligne_5   105   115
## ligne_6   106   116
## ligne_7   107   117
## ligne_8   108   118
## ligne_9   109   119
```

Indexation d'un vecteur ou d'une matrice

On peut indexer un vecteur (ou une matrice) par **position**, en indiquant le numéro de ligne (et de colonne pour une matrice) entre crochets (séparés par une virgule dans le cas d'une matrice), par **nom** (si les lignes - resp. les colonnes - sont nommées), ou par **condition logique**. Pour bien comprendre comment cela fonctionne explorez les codes suivants.

```r
# Indexation par position d'un vecteur
Age # juste pour avoir en tête toutes les valeurs
```

```
## [1] 71 29 45 32 81
```

```r
Age[1]
```

```
## [1] 71
```

```r
Age[2:4]
```

```
## [1] 29 45 32
```

```r
Age[length(Age)]
```

```
## [1] 81
```

```r
Age[c(-1, -3)] # pour enlever des éléments
```

```
## [1] 29 32 81
```

```r
## Indexation par condition d'un vecteur
Age[Age > 30]
```

```
## [1] 71 45 32 81
```

```r
Age[Genre == "masculin"]
```

```
## [1] 45 32
```

Pour une matrice on indique toujours entre les crochets ce qui correspond aux lignes, une virgule, puis ce qui correspond aux colonnes. La virgule doit être mise même si on ne fait une sélection que sur les lignes ou que sur les colonnes. Explorez les exemples ci-dessous.

```r
# création de Matrice6 pour ceux qui ne l'auraient pas créée auparavant
Matrice6 <- matrix(rep(100, times = 18) + c(1:9, 11:19), ncol = 2, byrow = FALSE)
colnames(Matrice6) <- c("100+i", "110+i")
rownames(Matrice6) <- paste("ligne", 1:9, sep = "_")
Matrice6
```

```
##         100+i 110+i
## ligne_1   101   111
## ligne_2   102   112
## ligne_3   103   113
## ligne_4   104   114
## ligne_5   105   115
## ligne_6   106   116
## ligne_7   107   117
## ligne_8   108   118
## ligne_9   109   119
```

```r
# Indexation par position d'une matrice
Matrice6[3, 2] # toujours [numéro de ligne, numéro de colonne]
```

```
## [1] 113
```

```r
Matrice6[1:5, ]
```

```
##         100+i 110+i
## ligne_1   101   111
## ligne_2   102   112
## ligne_3   103   113
## ligne_4   104   114
## ligne_5   105   115
```

```
Matrice6[ , 2]
```

```
## ligne_1 ligne_2 ligne_3 ligne_4 ligne_5 ligne_6 ligne_7 ligne_8 ligne_9
##     111     112     113     114     115     116     117     118     119
```

```
## Indexation par nom
Matrice6["ligne_1",  ]
```

```
## 100+i 110+i
##   101   111
```

```
Matrice6["ligne_1", "110+i"]
```

```
## [1] 111
```

Les jeux de données

Création d'un jeu de données directement dans R

UTILISATION des fonctions `data.frame()`, `str()`, `is.data.frame()`, `as.data.frame()`

Un jeu de données est un **objet R de dimension 2**, comme une matrice, mais à la différence d'une matrice il ne contient **pas forcément des éléments tous du même type**. Il est composé de vecteurs colonnes qui peuvent être de types différents. Un jeu de données est une juxtaposition de vecteurs dans lesquels sont stockées en colonne les valeurs de différentes variables (qualitatives ou quantitatives), chaque ligne correspondant aux valeurs des ces variables pour un même individu, ou plus généralement une même observation. Ci-dessous nous allons créer un petit jeu de données "jouet" à partir des vecteur créés précédemment et utiliser quelques fonctions. Explorez ce code.

```
# Création d'un jeu de données à partir de vecteurs
(d_jouet <- data.frame(age = Age, genre = Genre, pays = Pays))
```

```
##   age    genre     pays
## 1  71  feminin Autriche
## 2  29  feminin Danemark
## 3  45 masculin Danemark
## 4  32 masculin       UK
## 5  81  feminin Autriche
```

```
class(d_jouet)
```

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

```
str(d_jouet) # structure du jeu de données
```

```
## 'data.frame':    5 obs. of  3 variables:
##  $ age  : num  71 29 45 32 81
##  $ genre: chr  "feminin" "feminin" "masculin" "masculin" ...
##  $ pays : chr  "Autriche" "Danemark" "Danemark" "UK" ...
```

```r
is.data.frame(d_jouet)
```

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

```r
# Création d'un jeu de données à partir d'une matrice
# dans ce cas le jeu de données ne contiendra soit que des variables quantitatives
# soit que des variables qualitatives
is.data.frame(Matrice6)
```

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

```r
(d_matrice6 <- as.data.frame(Matrice6))
```

```
##         100+i 110+i
## ligne_1   101   111
## ligne_2   102   112
## ligne_3   103   113
## ligne_4   104   114
## ligne_5   105   115
## ligne_6   106   116
## ligne_7   107   117
## ligne_8   108   118
## ligne_9   109   119
```

```r
str(d_matrice6)
```

```
## 'data.frame':    9 obs. of  2 variables:
##  $ 100+i: num  101 102 103 104 105 106 107 108 109
##  $ 110+i: num  111 112 113 114 115 116 117 118 119
```

Indexation d'un jeu de données

La façon la plus courante de sélectionner une colonne (donc une variable) d'un jeu de données est d'utiliser le nom du jeu de données et d'y accoler $ suivi du nom de la variable, comme ci-dessous, mais on peut aussi l'indexer de la même façon qu'une matrice pour sélectionner ligne(s) ou colonne(s). Testez les lignes de code ci-dessous pour bien comprendre.

```r
# Sélection d'une colonne (variable)
d_jouet$age
```

```
## [1] 71 29 45 32 81
```

```r
d_jouet$pays
```

```
## [1] "Autriche" "Danemark" "Danemark" "UK"       "Autriche"
```

```r
d_jouet[, 2]
```

```
## [1] "feminin"  "feminin"  "masculin" "masculin" "feminin"
```

```r
# Sélection de lignes
d_jouet[2:4, ]
```

```
##   age    genre     pays
## 2  29  feminin Danemark
## 3  45 masculin Danemark
## 4  32 masculin       UK
```

```r
d_jouet[d_jouet$age > 30, ]
```

```
##   age    genre     pays
## 1  71  feminin Autriche
## 3  45 masculin Danemark
## 4  32 masculin       UK
## 5  81  feminin Autriche
```

```r
# Sélection de colonnes par noms
d_jouet[, c("age", "pays")]
```

```
##   age     pays
## 1  71 Autriche
## 2  29 Danemark
## 3  45 Danemark
## 4  32       UK
## 5  81 Autriche
```

```r
# Sélection sur lignes et colonnes
d_jouet[d_jouet$age > 30, ]$pays
```

```
## [1] "Autriche" "Danemark" "UK"       "Autriche"
```

Ajout de colonnes calculées à partir des colonnes préexistantes

Il est assez fréquent, quand on manipule des données, que l'on souhaite ajouter des variables calculées à partir des colonnes de base (*i.e.* des données dites brutes), et il vaut bien mieux le fare dans R, que dans le tableur (pour des raisons de traçabilité notamment). Pour créer une nouvelle variable dans le jeu de données, il suffit de lui donner comme nom dans l'affection le nom du jeu de données suivi de $ suivi du nom que l'on veut donner à la cette variable, comme ci-dessous :

```r
d_jouet$age_log_10 <- log10(d_jouet$age)
d_jouet$pays_genre <- paste(d_jouet$pays, d_jouet$genre, sep = "_")
d_jouet$au_moins_50ans <- d_jouet$age >= 50
d_jouet
```

```
##   age    genre     pays age_log_10        pays_genre au_moins_50ans
## 1  71  feminin Autriche   1.851258  Autriche_feminin           TRUE
## 2  29  feminin Danemark   1.462398  Danemark_feminin          FALSE
## 3  45 masculin Danemark   1.653213 Danemark_masculin          FALSE
## 4  32 masculin       UK   1.505150       UK_masculin          FALSE
## 5  81  feminin Autriche   1.908485  Autriche_feminin           TRUE
```

Les facteurs

UTILISATION des fonctions `is.factor()`, as.factor(), `factor()`, `levels()`

Pour que R considère les vecteurs de chaînes de caractères (type `char`) comme des variables qualitatives (type `factor`), il faut les tranformer en facteurs. Ensuite il est souvent utile de **modifier l'ordre des modalités d'un facteur** (par défaut il les ordonne par ordre alphabétique) et/ou de **changer le nom des modalités**. Découvrez ces fonctionalités en testant le code ci-dessous.

```
# Transformation en facteur
class(d_jouet$pays)
```

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

```
d_jouet$pays <- as.factor(d_jouet$pays)
class(d_jouet$pays)
```

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

```
levels(d_jouet$pays)
```

```
## [1] "Autriche" "Danemark" "UK"
```

```
# Changement des noms des modalités
levels(d_jouet$pays) <- c("Autriche", "Danemark", "Royaume-Uni")

# Changement de l'ordre des modalités
d_jouet$pays <- factor(d_jouet$pays, levels = c("Danemark", "Royaume-Uni", "Autriche"))
levels(d_jouet$pays)
```

```
## [1] "Danemark"    "Royaume-Uni" "Autriche"
```

Importation et exploration rapide d'un jeu de données

UTILISATION des fonctions `read.table()`, `str()`, `head()`, `nrow()`, `dim(d)`, `summary()`.

Généralement les jeux de données ne sont pas créés directement dans R mais importés par exemple à partir d'un fichier texte exporté depuis un tableur. Avant d'importer un fichier contenant le jeu de données à importer, il est important de savoir :

- **s'il comporte ou non une en-tête** (première ligne qui correspond aux noms des variables, pour définir l'**argument `header`** de la fonction d'import),

- quel **séparateur de colonnes** a été utilisé (le point virgule `";"`, l'espace `" "`, la tabulation `"\t"`, ..., pour définir l'**argument** `sep` de la fonction d'import),
- quel **séparateur de décimales** a été utilisé, pour définir l'**argument** `dec` de la fonction d'import. (ATTENTION, dans R le point `.` est utilisé, alors que sur la plupart des ordinateurs paramétrés en France, la virgule `", "` est utilisée. Il faudra donc indiquer `dec = ","` pour importer correctement un fichier venant d'un tel ordinateur paramétré en français pour le séparateur de décimales).

Pour importer des données à partir d'un fichier .txt exporté classiquement avec comme **séparateur de colonnes la tabulation**, nous utiliserons la fonction `read.table()` avec ses arguments `header = TRUE` si le fichier comporte une en-tête, et `stringsAsFactors = TRUE` pour que les vecteurs de chaînes de caractères soient automatiquement transformés en facteurs.

Importez et explorez un jeu de données qui nous servira plus tard à l'aide des lignes de codes suivantes :

```
dENQ <- read.table("ENQ9697.txt", header = TRUE, stringsAsFactors = TRUE)
str(dENQ)
```

```
## 'data.frame':    107 obs. of  7 variables:
##  $ SEXE    : Factor w/ 2 levels "F","M": 1 2 1 2 1 1 2 1 1 1 ...
##  $ AGE     : int  22 21 19 20 19 21 21 19 20 22 ...
##  $ POIDS   : int  53 67 63 60 48 58 77 61 52 70 ...
##  $ TAILLE  : int  175 175 172 175 167 171 187 170 161 168 ...
##  $ CADRE   : Factor w/ 2 levels "C","V": 2 1 2 2 2 1 1 1 1 1 ...
##  $ DECISION: Factor w/ 3 levels "A","E","T": 2 1 1 1 1 2 1 1 2 2 ...
##  $ FILIERE : Factor w/ 7 levels "A","C","E","I",..: 6 6 3 2 6 1 2 1 6 6 ...
```

```
head(dENQ)
```

```
##   SEXE AGE POIDS TAILLE CADRE DECISION FILIERE
## 1    F  22    53    175     V        E       R
## 2    M  21    67    175     C        A       R
## 3    F  19    63    172     V        A       E
## 4    M  20    60    175     V        A       C
## 5    F  19    48    167     V        A       R
## 6    F  21    58    171     C        E       A
```

```
nrow(dENQ)
```

```
## [1] 107
```

```
dim(dENQ)
```

```
## [1] 107   7
```

```
summary(dENQ)
```

```
##  SEXE        AGE            POIDS           TAILLE      CADRE  DECISION FILIERE
##  F:69   Min.   :18.00   Min.   :47.00   Min.   :152.0   C:52   A:31     A:11
##  M:38   1st Qu.:19.50   1st Qu.:54.00   1st Qu.:163.5   V:55   E:50     C:35
##         Median :20.00   Median :60.00   Median :170.0          T:26     E:16
##         Mean   :20.36   Mean   :61.23   Mean   :170.8                   I: 3
##         3rd Qu.:21.00   3rd Qu.:67.00   3rd Qu.:178.0                   P: 9
##         Max.   :23.00   Max.   :95.00   Max.   :190.0                   R:31
##                                                                         S: 2
```

Manipulation de données

Importation d'un jeu de données plus complexe

> **UTILISATION des fonctions `read.table()`, `str()`, `head()`, `nrow()`, `dim(d)`, `summary()`, `is.na()`, `complete.cases()` et `sum()`.**

Pour illustrer les manipulations classiques de jeux de données, nous allons travailler sur des données réelles, déposées sur Zenodo par les auteurs de l'article suivant :

Sandøe, P., Palmer, C., Corr, S. A., Springer, S., & Lund, T. B. (2023). Do people really care less about their cats than about their dogs? A comparative study in three European countries. Frontiers in Veterinary Science, 10, 1237547. https://www.ncbi.nlm.nih.gov/pmc/articles/PMC10625900/pdf/fvets-10-1237547.pdf

Vous utiliserez le fichier Excel Sandoee_et_al-Frontiers_2023_v2_numeric.xls, récupéré sur zenodo, auquel j'ai ajouté deux feuilles pour vous simplifier la tâche, l'une nommée data4R dans lequel on a fait une sélection des colonnes du premier, colonnes que j'ai renommées de façon plus explicite (à partir du matériel supplémentaire de l'article), l'autre nommée data4R_withNA dans lequel j'ai remplacé toutes les cellules vides par des NA pour indiquer qu'il s'agit de données manquantes.

1. Sauvez la feuille data4R depuis Excel en format **Texte (séparateur : tabulation)**, et tentez de l'importer dans R avec la ligne de code suivante puis analysez le message d'erreur qui devrait vous apparaître.

```
d <- read.table("Sandoe2023_sansNA.txt", stringsAsFactors = TRUE, header = TRUE)
```

2. Ensuite utilisez la feuille dans laquelle on a remplacé les cellules vides par des NA pour importer le jeu de données de la même façon, mais sans message d'erreur cette fois-ci.
3. Explorez les données importées (noms des variables, types des variables, nombre de lignes, ...) à l'aide des fonctions `str()`, `head()`, `nrow()`, `dim(d)` et `summary()`. Listez quelques raisons pour lesquels un petit travail de recodage va être nécessaire avant d'analyser ces données.

```
# 2
d <- read.table("Sandoe2023.txt", stringsAsFactors = TRUE, header = TRUE)
# 3
str(d)
```

```
## 'data.frame':    4610 obs. of  14 variables:
##  $ Country           : int  1 2 2 1 2 2 3 1 2 1 ...
##  $ Age               : int  71 29 59 63 34 71 49 39 29 20 ...
##  $ Gender            : int  2 2 2 2 1 1 2 2 1 2 ...
##  $ INCOME_AT         : int  1 NA NA 6 NA NA NA 3 NA 2 ...
##  $ INCOME_DK         : int  NA 3 3 NA 9 2 NA NA 12 NA ...
##  $ INCOME_UK         : int  NA NA NA NA NA NA 2 NA NA NA ...
##  $ companionship_dogs: int  NA NA 0 1 NA NA NA NA 1 0 ...
##  $ companionship_cats: int  NA 1 NA NA NA NA 1 NA NA 0 ...
##  $ nb_of_dogs        : int  NA 1 2 3 NA NA 1 NA 2 4 ...
##  $ nb_of_cats        : int  NA 3 1 1 NA NA 2 NA 1 3 ...
##  $ insurance_dog1    : int  NA NA 1 NA NA NA NA NA 1 NA ...
##  $ insurance_cat1    : int  NA NA NA NA NA NA 2 NA NA NA ...
##  $ LAPS              : int  NA 63 65 52 NA NA 52 NA 44 44 ...
##  $ Favorite_species  : int  NA NA NA NA NA NA NA NA NA 1 ...
```

```
head(d)
```

```
##   Country Age Gender INCOME_AT INCOME_DK INCOME_UK companionship_dogs
## 1       1  71      2         1        NA        NA                 NA
## 2       2  29      2        NA         3        NA                 NA
## 3       2  59      2        NA         3        NA                  0
## 4       1  63      2         6        NA        NA                  1
## 5       2  34      1        NA         9        NA                 NA
## 6       2  71      1        NA         2        NA                 NA
##   companionship_cats nb_of_dogs nb_of_cats insurance_dog1 insurance_cat1 LAPS
## 1                 NA         NA         NA             NA             NA   NA
## 2                  1          1          3             NA             NA   63
## 3                 NA          2          1              1             NA   65
## 4                 NA          3          1             NA             NA   52
## 5                 NA         NA         NA             NA             NA   NA
## 6                 NA         NA         NA             NA             NA   NA
##   Favorite_species
## 1               NA
## 2               NA
## 3               NA
## 4               NA
## 5               NA
## 6               NA
```

```
nrow(d)
```

```
## [1] 4610
```

```
dim(d)
```

```
## [1] 4610   14
```

```
summary(d)
```

```
##     Country           Age             Gender        INCOME_AT
##  Min.   :1.000   Min.   : 18.00   Min.   :1.000   Min.   : 1.000
##  1st Qu.:1.000   1st Qu.: 32.00   1st Qu.:1.000   1st Qu.: 2.000
##  Median :2.000   Median : 47.00   Median :2.000   Median : 4.000
##  Mean   :2.013   Mean   : 48.12   Mean   :1.562   Mean   : 5.316
##  3rd Qu.:3.000   3rd Qu.: 63.00   3rd Qu.:2.000   3rd Qu.: 7.000
##  Max.   :3.000   Max.   :999.00   Max.   :4.000   Max.   :13.000
##                                                   NA's   :3110
##    INCOME_DK        INCOME_UK      companionship_dogs companionship_cats
##  Min.   : 1.000   Min.   : 1.000   Min.   :0.000      Min.   :0.000
##  1st Qu.: 4.000   1st Qu.: 2.000   1st Qu.:1.000      1st Qu.:0.000
##  Median : 6.000   Median : 3.000   Median :1.000      Median :1.000
##  Mean   : 6.671   Mean   : 4.635   Mean   :0.769      Mean   :0.741
##  3rd Qu.:10.000   3rd Qu.: 6.000   3rd Qu.:1.000      3rd Qu.:1.000
##  Max.   :13.000   Max.   :13.000   Max.   :1.000      Max.   :1.000
##  NA's   :3058     NA's   :3052     NA's   :3365       NA's   :3337
##    nb_of_dogs     nb_of_cats    insurance_dog1  insurance_cat1       LAPS
```

```
##  Min.   :1.00   Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   : 0.00
##  1st Qu.:1.00   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:1.000   1st Qu.:38.00
##  Median :2.00   Median :2.000   Median :1.000   Median :2.000   Median :47.00
##  Mean   :1.68   Mean   :1.872   Mean   :1.491   Mean   :1.763   Mean   :46.69
##  3rd Qu.:2.00   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:2.000   3rd Qu.:58.00
##  Max.   :6.00   Max.   :6.000   Max.   :3.000   Max.   :3.000   Max.   :69.00
##  NA's   :2271   NA's   :2271    NA's   :3612    NA's   :3855    NA's   :2271
##  Favorite_species
##  Min.   :1.000
##  1st Qu.:1.000
##  Median :1.000
##  Mean   :1.874
##  3rd Qu.:2.000
##  Max.   :9.000
##  NA's   :3857
```

4. *En OPTIONNEL pour les plus rapides*, pour savoir si on a une information sur les revenus de chaque répondant, calculez combien il y a de lignes sans données manquantes respectivement dans les colonnes INCOME_AT, INCOME_DK, INCOME_UK et vérifiez que la somme correspond au nombre total de répondants. Utilisez pour cela les fonctions `is.na()`, l'opérateur logique ! et la fonction `sum()`.
5. Ensuite, *toujours en OPTIONNEL*, calculez combien il y a de lignes sans aucune donnée manquante (toutes colonnes prises en compte expecté celles sur le revenu) en utilisant en plus la fonction `complete.cases()`.

```
# 4
# Calcul du nombre de lignes sans données manquantes dans les colonnes codant pour le revenu
(nbI_AT <- sum(!is.na(d$INCOME_AT)))
```

```
## [1] 1500
```

```
(nbI_DK <- sum(!is.na(d$INCOME_DK)))
```

```
## [1] 1552
```

```
(nbI_UK <- sum(!is.na(d$INCOME_UK)))
```

```
## [1] 1558
```

```
# Calcul de la somme et comparaison au nombre de répondants
nbI_AT + nbI_DK + nbI_UK
```

```
## [1] 4610
```

```
nrow(d)
```

```
## [1] 4610
```

```
# 5
# Calcul du nombre de lignes sans aucune donnée manquante sur toutes les colonnes
# excepté celles sur le revenu (colonnes 4 à 6)
lignes_sans_NA <- complete.cases(d[,-(4:6)])
class(lignes_sans_NA)
```

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

```
length(lignes_sans_NA)
```

```
## [1] 4610
```

```
sum(lignes_sans_NA)
```

```
## [1] 188
```

Recodage simple d'une variable numérique en facteur

> **UTILISATION des fonctions `class()`, `as.factor()`, `levels()`, `factor()` et `table()`.**

Dans ce jeu de données le pays (colonne Country) a été codé 1 pour l'Autriche, 2 pour le Danemark et 3 pour le Royaume Uni, dans la variable Country. Nous allons créer dans le même jeu de données une nouvelle variable que l'on appelera "Pays" et qui sera définie comme un facteur, avec comme modalités "Danemark", "Autriche" et "UK", dans cet ordre. Nous calulerons aussi les effectifs de répondants dans chaque pays à l'aide de la fonction `table()`.

Faites tournez le code ci-dessous en regardant les objets créés pour bien comprendre ce que fait chaque instruction.

```
class(d$Country)
```

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

```
d$Pays <- as.factor(d$Country)
class(d$Pays)
```

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

```
levels(d$Pays)
```

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

```
levels(d$Pays) <- c("Autriche", "Danemark", "UK")
table(d$Pays) # pour vérifier la cohérence avec la table 2 de l'article
```

```
##
## Autriche Danemark       UK
##     1500     1552     1558
```

Modification des noms et de l'ordre des modalités d'un facteur et sélection de lignes

> **UTILISATION des fonctions `subset()`, `class()`, `as.factor()`, `levels()`, `factor()` et `table()`.**

1. Le genre (colonne Gender) a été codé numériquement en 1 pour les hommes, 2 pour les femmes, 3 pour les non genrés et 4 pour ceux qui ne voulaient pas donner leur genre. Créez une variable Genre à partir de Gender, qui soit un facteur, de modalités "femme", "homme", "non genre" et "sans reponse", dans cet ordre.
2. Les auteurs de l'article avaient choisi d'exclure les données correspondant aux deux dernières catégories qui étaient en faible nombre. On fera de même. Utilisez la fonction `subset()` pour créer un nouveau jeu de données ds uniquement avec les lignes correspondant aux modalités 3 et 4 du genre. Vérifiez la cohérence des résultats obtenus avec la table 2 de l'article en tapant `table(d$Genre, d$Pays)`.
3. *En OPTIONNEL pour les plus rapides*, Refaites une table des effectifs des modalités du Genre à partir du sous jeu de données. Que constatez-vous. Réappliquez la fonction `factor()` à la variable dans ce nouveau jeu de données et refaites la table. Que constatez-vous ?

```
# 1
# Transformation en facteur
class(d$Gender)
```

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

```
d$Genre <- as.factor(d$Gender)
class(d$Genre)
```

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

```
levels(d$Genre)
```

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

```
# Changement des noms des modalités
levels(d$Genre) <- c("homme", "femme", "non genre", "sans reponse")
table(d$Genre)
```

```
##
##        homme        femme    non genre sans reponse
##         2035         2563            8            4
```

```
# Changement de l'ordre des modalités
d$Genre <- factor(d$Genre, levels = c("femme", "homme", "non genre", "sans reponse"))
levels(d$Genre)
```

```
## [1] "femme"        "homme"        "non genre"    "sans reponse"
```

```
table(d$Genre)
```

```
##
##        femme        homme    non genre sans reponse
##         2563         2035            8            4
```

```
# Table à deux entrée pour comparaison avec Table 2 de l'article
table(d$Genre, d$Pays)
```

```
##
##                Autriche Danemark  UK
##   femme             854      851 858
##   homme             641      698 696
##   non genre           4        1   3
##   sans reponse        1        2   1
```

```
# 2
# Exclusion des modalités 3 et 4
nrow(d)
```

```
## [1] 4610
```

```
ds <- subset(d, Gender < 3)
nrow(ds)
```

```
## [1] 4598
```

```
# 3
# Redéfinition du facteur pour oublier les modalités disparues du fait de la sélection
table(ds$Genre)
```

```
##
##        femme        homme    non genre sans reponse
##         2563         2035            0            0
```

```
levels(ds$Genre)
```

```
## [1] "femme"        "homme"        "non genre"    "sans reponse"
```

```
ds$Genre <- factor(ds$Genre)
table(ds$Genre)
```

```
##
## femme homme
##  2563  2035
```

Dans la suite nous continuerons à travailler sur le jeu de données complet `d` et non sur la sélection faite précédemment `ds`.

Transformation d'une variable quantitative en variable qualitative par définition de classes

> **UTILISATION des fonctions `min()`, `max()` (et de leur argument `na.rm`), `which()`, `table()` et `cut()`.**

Dans le jeu de données, la variable Age donne l'âge du répondant en années. Dans l'article l'âge semble être manipulé sous forme d'une variable qualitative à 5 modalités, correspondant aux classes 18-29 ans, 30-39 ans, 40-49 ans, 50-59 ans et plus de 60 ans.

1. Créez une nouvelle variable qualitative que vous nommerez AgeF à l'aide de ce découpage en 5 classes, en utilisant la fonction `cut()`. En amont calculez le valeurs min et max de l'âge des répondants, et gérez le problème mis en évidence en remplaçant les valeurs irréalistes par des valeurs manquantes (`NA`).
2. En *OPTIONNEL pour les plus rapides*, vérifiez la cohérence des résultats obtenus avec la table 2 de l'article en tapant `table(d$AgeF, d$Pays)` et essayez de comprendre d'où viennent les différences observées (pour ma part je n'y suis pas arrivée).

```
#1
min(d$Age)
```

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

```
max(d$Age)
```

```
## [1] 999
```

```
# Identification du ou des valeurs extrêmes irréalistes
which(d$Age > 130)
```

```
## [1] 2611 3994
```

```
# Remplacement des valeurs extrêmes irréalistes par NA
d$Age[d$Age > 130] <- NA
min(d$Age, na.rm = TRUE)
```

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

```
max(d$Age, na.rm = TRUE)
```

```
## [1] 100
```

```
# Définition de la nouvelle variable
d$AgeF <- cut(d$Age, breaks = c(17, 29, 39, 49, 59, Inf))
str(d$AgeF)
```

```
##  Factor w/ 5 levels "(17,29]","(29,39]",..: 5 1 4 5 2 5 3 2 1 1 ...
```

```
table(d$AgeF)
```

```
##
##  (17,29]  (29,39]  (39,49]  (49,59] (59,Inf]
##      988      714      791      775     1340
```

```
# 2
# Table à deux entrées
table(d$AgeF, d$Pays)
```

```
##
##           Autriche Danemark  UK
##   (17,29]       355      299 334
##   (29,39]       269      217 228
##   (39,49]       264      264 263
##   (49,59]       260      262 253
##   (59,Inf]      352      508 480
```

```
# Essai infructueux pour comprendre les incohérences avec la table 2
# non expliquées par les données aberrantes remplacées par des NA
d$AgeFbis <- cut(d$Age, breaks = c(17, 29, 39, 49, 59, Inf), right = FALSE)
table(d$AgeFbis, d$Pays)
```

```
##
##           Autriche Danemark  UK
##   [17,29)       327      274 291
##   [29,39)       271      223 238
##   [39,49)       275      250 269
##   [49,59)       252      249 254
##   [59,Inf)      375      554 506
```

Création d'une nouvelle variable quantitative par calcul

> **UTILISATION de la vectorisation des calculs, de la fonction `scale()` et des fonctions `mean()` et `sd()` avec leur argument `na.rm` .**

Pour le cas où l'on voudrait utiliser ultérieurement l'âge en tant que variable quantitative, mais en le centrant (*i.e.* enlevant l'âge moyen à chaque valeur) et en le réduisant (*i.e.* le divisant par son écart type), créez la variable quantitative nommée AgeCR à l'aide de la fonction `scale()`. Vérifiez que cela a bien fonctionné en calculant la moyenne et l'écart type de cette nouvelle variable.

```
d$AgeCR <- scale(d$Age)
# vérification
mean(d$AgeCR, na.rm = TRUE)
```

```
## [1] 1.418686e-18
```

```
sd(d$AgeCR, na.rm = TRUE)
```

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

Manipulation de données (*suite OPTIONNELLE pour les plus rapides*)

Création des variables Revenu et RevenuF

> **UTILISATION de l'indexation d'un jeu de données par un vecteur de noms de colonnes, des fonctions `summary()` et `rowSums()` et de son argument `na.rm`.**

Les auteurs de l'article ont fait trois colonnes nommées INCOME_AT (pour les revenus des australiens), INCOME_DK (pour les revenus des danois) et INCOME_UK (pour les revenus des anglais). Ils ont codé le revenu par un score de 1 à 13 défini à partir de la classe d'appartenance de chaque revenu, définies pour chaque pays. Les scores de 1 à 11 correspondent à des revenus croissants, la classe 12 à (préfère ne pas répondre) et la classe 13 à "je ne sais pas".

1. Comparez les distributions des revenus dans les 3 pays en appliquant la fonction `summary()` à un sous jeu de données contenant uniquement les trois colonnes INCOME_xx.
2. Créez dans `d` une colonne unique intitulée "Revenu" et contenant avec les revenus des tous les participants à l'enquête, en utilisant la fonction `rowSums()` avec son argument `na.rm`. Vérifiez notamment à l'aide de la fonction `summary()` que votre code a bien fonctionné.
3. Dans l'article le revenu est manipulé sous la forme d'une variable qualitative à 3 modalités, "revenu_haut", "revenu_bas", "revenu_inconnu". Créez une telle variable, nommée `RevenuF`, sachant que pour le Royaume-Uni et l'Autriche le revenu est considéré comme bas s'il est inférieur ou égal à 3 dans codage original, mais que pour le Danemark il est considéré comme bas s'il est inférieur ou égal à 5. Une façon de faire est de créer à l'aide de la fonction `factor()` un facteur défini avec ces 3 modalités, rempli uniquement avec des "revenu_inconnu", puis de remplacer les valeurs qui doivent être à "revenu_bas" et "revenu_haut" en fonction des valeurs de
   "INCOME_AT", "INCOME_DK" et "INCOME_UK". Vérifiez la cohérence des résultats obtenus avec la table 2 de l'article en tapant `table(d$RevenuF, d$Pays)`.

```r
# 1
summary(d[, c("INCOME_AT", "INCOME_DK", "INCOME_UK")])
```

```
##    INCOME_AT        INCOME_DK        INCOME_UK
##  Min.   : 1.000   Min.   : 1.000   Min.   : 1.000
##  1st Qu.: 2.000   1st Qu.: 4.000   1st Qu.: 2.000
##  Median : 4.000   Median : 6.000   Median : 3.000
##  Mean   : 5.316   Mean   : 6.671   Mean   : 4.635
##  3rd Qu.: 7.000   3rd Qu.:10.000   3rd Qu.: 6.000
##  Max.   :13.000   Max.   :13.000   Max.   :13.000
##  NA's   :3110     NA's   :3058     NA's   :3052
```

```r
# 2
d$Revenu <- rowSums(d[, c("INCOME_AT", "INCOME_DK", "INCOME_UK")], na.rm = TRUE)
summary(d[, "Revenu"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   3.000   4.000   5.542   8.000  13.000
```

```r
# 3
d$RevenuF <- factor(x = rep("revenu_inconnu", nrow(d)), levels = c("revenu_haut", "revenu_bas", "revenu
d$RevenuF[d$INCOME_AT <= 3] <- "revenu_bas"
d$RevenuF[d$INCOME_UK <= 3] <- "revenu_bas"
d$RevenuF[d$INCOME_DK <= 5] <- "revenu_bas"
d$RevenuF[(d$INCOME_AT > 3) & (d$INCOME_AT <= 11)] <- "revenu_haut"
d$RevenuF[(d$INCOME_UK > 3) & (d$INCOME_UK <= 11)] <- "revenu_haut"
d$RevenuF[(d$INCOME_DK > 5) & (d$INCOME_DK <= 11)] <- "revenu_haut"
table(d$RevenuF, d$Pays)
```

```
##
##                  Autriche Danemark  UK
```

```
##   revenu_haut          535      570 586
##   revenu_bas           690      709 796
##   revenu_inconnu       275      273 176
```

Recodage des nombres de chiens et nombres de chats

Les colonnes nb_of_dogs et nb_of_cats codent pour le nombre de chiens et le nombre de chats, mais avec un code un peu particulier. 1 correspond à 0, 2 à 1, 3 à 2, 4 à 3, 5 à 4 et 6 à plus de 4 chiens (resp. chats).

1. Définissez de nouvelles variables qualitatives Nchiens et Nchats sous forme de facteurs codant pour ces variables, avec des noms de modalités plus explicites.
2. Créez ensuite des variables Proprio_Chien et Proprio_Chat indiquant TRUE selon que le répondant est ou non propriétaire de chien(s) (resp. chat(s)).
3. Créez une variable ChatChien à partir des deux précédentes, avec comme modalités "chien", "chat", "chienETchat" ou "NIchienNIchat" (dans cet ordre) suivant que le répondant a seulement un ou des chiens, seulement un ou des chats, les deux, ou ni l'un ni l'autre. On peut par exemple commencer par créer un facteur rempli de NA avec la fonction rep() et la remplir au fur et à mesure pour chaque modalité.
4. Faites une table à deux entrées avec `ChatChien` et `Pays` pour vérifier la cohérence avec la table 1 de l'article.

```
# 1
modNchiens <- c("0 chien", "1 chien", "2 chiens", "3 chiens",
                "4 chiens", "plus de 4 chiens")
d$Nchiens <- modNchiens[d$nb_of_dogs]
modNchats <- c("0 chat", "1 chat", "2 chats", "3 chats",
               "4 chats", "plus de 4 chats")
d$Nchats <- modNchats[d$nb_of_cats]

# 2
d$Proprio_Chien <- d$nb_of_dogs > 1
d$Proprio_Chat <- d$nb_of_cats > 1

# 3
d$ChatChien <- factor(rep(NA, nrow(d)),
                      levels = c("chien", "chat", "chienETchat", "NIchienNIchat"))
d$ChatChien[d$Proprio_Chien & d$Proprio_Chat] <- "chienETchat"
d$ChatChien[d$Proprio_Chien & !d$Proprio_Chat] <- "chien"
d$ChatChien[!d$Proprio_Chien & d$Proprio_Chat] <- "chat"
d$ChatChien[!d$Proprio_Chien & !d$Proprio_Chat] <- "NIchienNIchat"
class(d$ChatChien)
```

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

```
d$ChatChien <- as.factor(d$ChatChien)

# 4
table(d$ChatChien, d$Pays)
```

```
##
##                 Autriche Danemark  UK
##   chien              225      308 311
##   chat               391      241 240
##   chienETchat        184       77 140
##   NIchienNIchat       84       62  76
```

Automatisation des calculs

Structures conditionnelles `if` et itératives `for`

Afin de comprendre comment on utilise les structures conditionnelles et itératives, testez le code suivant.

```
# Utilisation de la boucle for
for (i in 1:5){
  print(paste("Itération", i, sep = " "))
}
```

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

```
# Utilisation d'une condition if
valeur <- 40
if (valeur > 50){
  print("OK")
} else {
  print("KO")
}
```

```
## [1] "KO"
```

```
# Boucle contenant une condition
valeurs <- c(12, 40, 45, 52, 67, 78)
for (i in 1:length(valeurs)){
  if (valeurs[i] > 50){
    print("OK")
  } else {
    print("KO")
  }
}
```

```
## [1] "KO"
## [1] "KO"
## [1] "KO"
## [1] "OK"
## [1] "OK"
## [1] "OK"
```

Maintenant que vous avez compris comme cela fonctionne, nous appliquer ces structures de contrôle sur notre jeu de données. Si vous ne l'avez pas déjà importé vous pouvez le faire avec la commande ci-dessous :

```
d <- read.table("Sandoe2023.txt", stringsAsFactors = TRUE, header = TRUE)
```

Le score d'attachement (LAPS - Lexingto attachment to pets scale) a été évalué pour chaque propriétaire, à partir de 23 questions relatives à son attachement à son animal favori (Favorite_species). Cette dernière variable a été codée en 1 pour "dog", 2 pour "cat", et de 3 à 9 pour les autres espèces.

1. Appliquez la fonction `table()` à la variable Favorite_species pour voir ce qu'elle contient.
2. Faites une nouvelle colonne Animal_favori que vous coderez en "chien", "chat", "autre". Pour vous entraîner à utiliser les structures conditionnelles et de boucle, faites cela en créant une nouvelle colonne de type facteur (avec `factor()`), en indiquant au départ juste ses modalités dans l'ordre "chien", "chat", "autre", et en la remplissant avec des NA (à l'aide de la fonction `rep()`), puis parcourez ce vecteur (avec la structure itérative `for`) en affectant sa modalité à partir de ce qui est codé dans Favorite_species, en utilisant la strucure itérative if. Pour gérer les NA dans Favorite_species, vous pouvez utilisez une structure conditionnelle avec un if initial. Comme on aura mis des NA par défaut dans le nouveau facteur, pas besoin de réaffecter les données manquantes.
3. Appliquez ensuite la fonction `table()` sur ce facteur pour voir les effectifs pour chaque modalité et vérifiez que c'est cohérent avec votre première table faite sur Favorite_species.

```
# 1
table(d$Favorite_species)
```

```
##
##   1   2   3   4   5   6   7   8   9
## 414 232  36  20   8  17  10  11   5
```

```
# 2
d$Animal_favori <- factor(rep(NA, nrow(d)), levels = c("chien", "chat", "autre"))
class(d$Animal_favori)
```

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

```
levels(d$Animal_favori)
```

```
## [1] "chien" "chat"  "autre"
```

```
for (i in 1:nrow(d)){
  if (! is.na(d$Favorite_species[i])) {
    if (d$Favorite_species[i] == 1) {
      d$Animal_favori[i] <- "chien"
    } else {
      if (d$Favorite_species[i] == 2) {
        d$Animal_favori[i] <- "chat"
      } else {
        d$Animal_favori[i] <- "autre"
      }
    }
  }
}

# 3
table(d$Animal_favori)
```

```
##
## chien  chat autre
##   414   232   107
```

Ecriture d'une fonction et application d'une fonction par groupe à l'aide de la fonction `tapply()`

Afin de comprendre comment on écrit une fonction et comment on peut l'appliquer par groupe à l'aide de `tapply()` testez le code suivant :

```
# Création d'une fonction
Q1 <- function(x){
  quartile1 <- quantile(x, probs = 0.25, na.rm = TRUE)
  return(quartile1)
}

# Création de la variable Genre (si non fait auparavant)
## création d'un vecteur de modalités dans l'ordre
## dans lesquelles elles ont été codées dans Gender
## en donnant le même nom aux deux dernières
Genres <- c("homme", "femme", "autre", "autre")
## Utilisation du code dans Gender comme index dans Genres
d$Genre <- as.factor(Genres[d$Gender])

# Application de cette fonction à l'âge de tous les répondants puis manuellement par genre
Q1(d$Age)
```

```
## 25%
##  32
```

```
Q1(d$Age[d$Genre == "femme"])
```

```
## 25%
##  30
```

```
Q1(d$Age[d$Genre == "homme"])
```

```
## 25%
##  36
```

```
# Application automatique de la fonction par genre
tapply(d$Age, d$Genre, Q1)
```

```
## autre femme homme
## 21.75 30.00 36.00
```

1. A l'aide de la fonction `tapply()`, calculez la médiane des LAPS pour chaque modalité de la variable Animal_favori que l'on vient de créer. Faites la même chose pour calculer les moyennes.
2. Faites une petite fonction qui affiche les quartiles, le min et le max, en utilisant les fonctions `quantile()`, `min()` et `max()` et appliquez cette fonction pour calculer ces statistiques sur le LAPS par espèce (modalité de Animal_favori) en utilisant la fonction `tapply()`. Pour gérer les données manquantes, utilisez les arguments `na.rm` de ces fonctions.

```r
# 1
tapply(d$LAPS, d$Animal_favori, median)
```

```
## chien  chat autre
##    52    45    44
```

```r
tapply(d$LAPS, d$Animal_favori, mean)
```

```
##    chien     chat    autre
## 51.22947 45.84483 44.03738
```

```r
# 2
resume_stat <- function(v) {
  quartiles <- quantile(v, probs = c(0.25, 0.5, 0.75), na.rm = TRUE)
  min <- min(v, na.rm = TRUE)
  max <- max(v, na.rm = TRUE)
  res <- c(min, quartiles, max)
  names(res) <- c("min", "Q1", "Q2", "Q3", "max")
  return(res)
}
tapply(d$LAPS, d$Animal_favori, resume_stat)
```

```
## $chien
## min  Q1  Q2  Q3 max
##   6  44  52  61  69
##
## $chat
## min  Q1  Q2  Q3 max
##   6  37  45  59  69
##
## $autre
##  min   Q1   Q2   Q3  max
##  4.0 34.0 44.0 55.5 69.0
```

Sauvegarde du jeu de données augmenté

A l'aide de la fonction `write.table()`, sauvegardez le jeu de données augmenté avec toutes les nouvelles variables que vous avez créées, dans un fichier que l'on nommera `Sandoe23augmente.txt` et importez-le dans Excel pour l'y retrouver.

```r
# Petit regard sur la structure du jeu de données augmenté avant sa sauvegarde
str(d)
```

```
## 'data.frame':    4610 obs. of  16 variables:
##  $ Country           : int  1 2 2 1 2 2 3 1 2 1 ...
##  $ Age               : int  71 29 59 63 34 71 49 39 29 20 ...
##  $ Gender            : int  2 2 2 2 1 1 2 2 1 2 ...
##  $ INCOME_AT         : int  1 NA NA 6 NA NA NA 3 NA 2 ...
##  $ INCOME_DK         : int  NA 3 3 NA 9 2 NA NA 12 NA ...
```

```
##  $ INCOME_UK         : int  NA NA NA NA NA NA 2 NA NA NA ...
##  $ companionship_dogs: int  NA NA 0 1 NA NA NA NA 1 0 ...
##  $ companionship_cats: int  NA 1 NA NA NA NA 1 NA NA 0 ...
##  $ nb_of_dogs        : int  NA 1 2 3 NA NA 1 NA 2 4 ...
##  $ nb_of_cats        : int  NA 3 1 1 NA NA 2 NA 1 3 ...
##  $ insurance_dog1    : int  NA NA 1 NA NA NA NA NA 1 NA ...
##  $ insurance_cat1    : int  NA NA NA NA NA NA 2 NA NA NA ...
##  $ LAPS              : int  NA 63 65 52 NA NA 52 NA 44 44 ...
##  $ Favorite_species  : int  NA NA NA NA NA NA NA NA NA 1 ...
##  $ Animal_favori     : Factor w/ 3 levels "chien","chat",..: NA NA NA NA NA NA NA NA NA 1 ...
##  $ Genre             : Factor w/ 3 levels "autre","femme",..: 2 2 2 2 3 3 2 2 3 2 ...
```

```r
# Sauvegarde du jeu de données en format texte
write.table(d, file = "Sandoe23augmente.txt", row.names = FALSE)
```

Si vous souhaitez avoir la virgule comme séparateur de décimales (important ici uniquement pour la variable AgeCR), il faut mettre l'argument `dec = ","`, mais dans ce cas il faudra bien penser à faire de même si vous voulez importer à nouveau ce jeu de données dans R avec la fonction `read.table()`.

Visualisation de données avec ggplot2

A partir du jeu de données recodé, faites des représentations graphiques intéressantes pour visualiser notamment les variables LAPS de l'animal favori en fonction de l'âge, du genre et du revenu de son propriétaire, de son pays, et de l'espèce, et de l'espèce de l'animal favori.

```r
require(ggplot2)
# Exclusion des individus de genre inconnu ou non défini
d <- subset(d, Genre == "femme" | Genre == "homme")
d$Genre <- factor(d$Genre)

# quelques exemples parmi de nombreux possibles
ggplot(aes(x = Animal_favori, y = LAPS), data = d) + geom_boxplot() +
  facet_wrap(~ Pays) + theme_bw()
ggplot(aes(x = Animal_favori, y = LAPS, col = Genre), data = d) + geom_boxplot() +
  facet_wrap(~ Pays) + theme_bw()
ggplot(aes(x = Animal_favori, y = LAPS, col = Genre), data = d) + geom_boxplot() +
  facet_grid(AgeF ~ Pays) + theme_bw() # trop d'info, on n'y voit plus grand chose
ggplot(aes(x = Age, y = LAPS, col = Genre), data = d) + geom_point() +
  facet_wrap(~ Pays) + theme_bw()
ggplot(aes(x = RevenuF, y = LAPS, col = Pays), data = d) + geom_boxplot() + theme_bw()
```