FACULTÉ DES SCIENCES ECONOMIQUES

UNIVERSITÉ DE RENNES 1

M1 Statistique & Économétrie
Ewen Gallic [1]
http://egallic.fr

2015

Logiciel R et programmation

Exercices

Partie 1 : Données

Exercice 1 (manipulation de vecteurs)
Considérons le vecteur suivant : $x = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}$.

1. Créer ce vecteur dans R et le stocker dans un objet que l'on appellera x ;

```
# Une première solution
x <- c(1, 2, 3, 4, 5)
# Une seconde
x <- seq(1, 5)
# Une troisième
x <- seq_len(5)
```

2. Afficher le mode de x, puis sa longueur ;

```
mode(x)
## [1] "numeric"
length(x)
## [1] 5
```

3. Extraire le premier élément, puis le dernier ;

1. ewen.gallic[at]gmail.com

```
   "["(x,1) ; "["(x,5)
```

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

```
   x[1] ; x[5]
```

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

```
   # Pour le dernier, on peut également faire comme suit :
   x[length(x)]
```

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

4. Extraire les trois premier éléments et les stocker dans un vecteur que l'on nommera `a` ;

```
  a <- x[c(1,2,3)]
  (a <- x[1:3])
```

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

5. Extraire les éléments en position 1, 3, 5 ; les stocker dans un vecteur que l'on nommera `b` ;

```
  b <- x[c(1,3,5)]
```

6. Additionner le nombre `10` au vecteur `x`, puis multiplier le résultat par `2` ;

```
  (x + 10)*2
```

```
## [1] 22 24 26 28 30
```

7. Effectuer l'addition de `a` et `b`, commenter le résultat ;

```
a + b
```

```
## [1] 2 5 8
```

Les éléments de `a` et `b` sont ajoutés un par un, le i^e élément de `b` est additionné au i^e élément de `a`.

8. Effectuer l'addition suivante : `x+a`, commenter le résultat, puis regarder le résultat de `a+x` ;

Le i^e élément de `a` est additionné au i^e élément de `x`, mais comme la longueur de `a` est inférieure à celle de `b`, `R` procède à ce que l'on appelle le recyclage, c'est-à-dire que le vecteur de plus courte taille est répété jusqu'à obtenir une taille égale à celle du plus long, puis est tronqué si la taille de lélément répété dépasse celle du vecteur initialement plus long.

```
  x + a
```

```
## Warning in x + a: la taille d'un objet plus long n'est pas multiple de
la taille d'un objet plus court
```

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

9. Multiplier le vecteur par le scalaire `c` que l'on fixera à 2 ;

```
  c <- 2
  x*c
```

```
## [1]  2  4  6  8 10
```

10. Effectuer la multiplication de `a` et `b`, commenter le résultat ;

```
a * b
```

```
## [1]  1  6 15
```

Les éléments de `a` et `b` sont multipliés un par un, le i^e élément de `b` est multiplié au i^e élément de `a`.

11. Effectuer la multiplication suivante : `x*a`, commenter le résultat ;

Comme pour l'addition, `R` procède au recyclage.

```
  x
```

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

```
  a
```

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

```
  x * a
```

```
## Warning in x * a: la taille d'un objet plus long n'est pas multiple de
la taille d'un objet plus court
```

```
## [1]  1  4  9  4 10
```

12. Récupérer les positions des multiples de 2 et les stocker dans un vecteur que l'on nommera `ind`, puis conserver uniquement les multiples de 2 de `x` dans un vecteur que l'on nommera `mult_2` ;

```
  ind <- which(x %% 2 == 0)
  (mult_2 <- x[ind])
```

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

13. Afficher les éléments de `x` qui sont multiples de 3 **et** multiples de 2 ;

```
  x[which(x %% 2 == 0 & x %% 3 == 0)]
```

```
## integer(0)
```

14. Afficher les éléments de `x` qui sont multiples de 3 **ou** multiples de 2 ;

```
  x[which(x %% 2 == 0 | x %% 3 == 0)]
```

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

15. Calculer la somme des éléments de `x` ;

```
  sum(x)
```

```
## [1] 15
```

16. Remplacer le premier élément de `x` par un `4` ;

```
  x[1] <- 4
  x
```

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

17. Remplacer le premier élément de `x` par la valeur `NA`, puis calculer la somme des éléments de `x` ;

```
  x[1] <- NA
  sum(x)
```

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

```
  sum(x, na.rm=TRUE)
```

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

18. Lister les objets en mémoire dans la session `R` ;

```
  ls()
```

```
## [1] "a"      "b"      "c"      "ind"    "mult_2" "x"
```

19. Supprimer le vecteur ;

```
  rm(x)
```

20. Supprimer la totalité des objets de la session.

```
  rm(list=ls())
```

Lorsqu'on désire une session fraîche, il est préférable de redémarrer la session plutôt que de détruire les objets présents dans celle-ci. Le fait de redémarrer la session `R` décharge également les *packages* éventuellement chargés dans la session courante.

Exercice 2 (manipulation de listes)

1. Évaluer le code suivant : `TRUE+FALSE+TRUE*4` et le commenter ;

```
  TRUE + FALSE + TRUE * 4
## [1] 5
```

Dans `R`, `TRUE` et `FALSE` sont des booléens, et répondent à une condition logique. Lorsqu'on additionne des booléens, `R` les convertit en `integer` : `TRUE` prend la valeur 1 et `FALSE` 0.

2. Évaluer les expressions suivantes : `c(1, 4, TRUE)`, et `c(1, 4, TRUE, "bonjour")`, commenter ;

```
  c(1, 4, TRUE)
## [1] 1 4 1
  c(1, 4, TRUE, "bonjour")
## [1] "1"       "4"       "TRUE"    "bonjour"
```

Les éléments d'un vecteur doivent tous être de même type. Dans les deux cas, `R` convertit les données dans le type le plus général. Ainsi, dans le premier exemple, `TRUE` est converti en `numeric`, dans le second, la présence d'une chaîne de caractères force `R` à convertir tous les autres éléments du vecteur en `character`.

3. Créer une liste que l'on appellera `l` et qui contient les éléments `1`, `4` et `TRUE` en première, seconde et troisième positions respectivement ;

```
  l <- list(1, 4, TRUE)
```

4. Extraire le premier élément de la liste `l`, et afficher son mode. En faire de même avec le troisième élément, et commenter ;

```
  # Extraction du premier élément de l
  l[[1]]
## [1] 1
  "[["(l, 1)
## [1] 1
  # Mode du premier élément de l
  mode(l[[1]])
## [1] "numeric"
  # Idem pour le troisième élément
  mode(l[[3]])
## [1] "logical"
```

Contrairement aux vecteurs, les listes peuvent contenir des éléments de classe différente. Cela fait de la liste un objet très polyvalent.

5. Ajouter un quatrième élément à la liste `l` : `"bonjour"`, puis afficher la structure de `l` ;

```r
  l <- c(l, "bonjour")
  str(l)
```

```
## List of 4
##  $ : num 1
##  $ : num 4
##  $ : logi TRUE
##  $ : chr "bonjour"
```

6. Retirer le troisième élément de la liste `l` ;

```r
  l[[3]] <- NULL
  unlist(l)
```

```
## [1] "1"       "4"       "bonjour"
```

7. Créer une liste de trois éléments : votre nom, votre prénom, et votre année de naissance. Ces trois éléments de la liste devront être nommés respectivement `"nom"`, `""prenom"` et `année de naissance`. Stocker la liste ainsi créée dans un objet nommé `moi` ;

```r
  moi <- list(nom = "Vader", prenom = "Darth",
              `année de naissance` = 41.9)
  moi
```

```
## $nom
## [1] "Vader"
##
## $prenom
## [1] "Darth"
##
## $`année de naissance`
## [1] 41.9
```

8. Extraire le prénom de la liste `moi` de deux manières : en utilisant l'indice, et en utilisant le nommage ;

```r
  moi[[2]]
```

```
## [1] "Darth"
```

```r
  moi$prenom
```

```
## [1] "Darth"
```

```r
  # Remarque : on peut effectuer la recherche de l'indice
  which(names(moi) == "prenom")
```

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

9. Créer une liste avec la même structure que celle de `moi`, en la remplissant avec les informations d'une autre personne et la nommer `toi`. Puis, créer la liste `personnes`, qui contiendra les listes `toi` et `moi` ;

```r
toi <- list(nom = "Skywalker", prenom = "Luke",
            `année de naissance` = 19)
personnes <- list(toi, moi)
```

10. Extraire la liste `toi` de `personnes` (en première position) ;

```r
personnes[[1]]
```

```
## $nom
## [1] "Skywalker"
##
## $prenom
## [1] "Luke"
##
## $`année de naissance`
## [1] 19
```

Attention, si on fait `personnes[1]`, on garde uniquement le premier élément de la liste `personnes`, mais on conserve la structure de liste ; dans ce cas précis, cela revient à retirer tous les éléments de la liste sauf le premier.

11. Extraire directement depuis `personne` le prénom de l'élément en première position.

```r
personnes[[1]]$prenom
```

```
## [1] "Luke"
```

Exercice 3 (manipulation de matrices)

1. Créer la matrice suivante : $A = \begin{bmatrix} -3 & 5 & 6 \\ -1 & 2 & 2 \\ 1 & -1 & -1 \end{bmatrix}$;

```r
A <- matrix(c(3,5,6, -1,2,2, 1,-1,-1), ncol=3, byrow=TRUE)
```

Si on ne précise pas `byrow=TRUE`, `R` remplit la matrice colonne par colonne.

2. Afficher la dimension de `A`, son nombre de colonnes, son nombre de lignes et sa longueur ;

```
  dim(A)
## [1] 3 3
  ncol(A)
## [1] 3
  nrow(A)
## [1] 3
  length(A)
## [1] 9
```

3. Extraire la seconde colonne de `A`, puis la première ligne ;

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

4. Extraire l'élément en troisième position à la première ligne ;

```
 A[1, 3]
## [1] 6
```

5. Extraire la sous-matrice de dimension 2×2 du coin inférieur de `A`, c'est-à-dire $\begin{bmatrix} 2 & 2 \\ -1 & -1 \end{bmatrix}$;

```
  A[2:3, 2:3]
##      [,1] [,2]
## [1,]    2    2
## [2,]   -1   -1
```

6. Calculer la somme des colonnes puis des lignes de `A` ;

```
  colSums(A)
## [1] 3 6 7
  rowSums(A)
## [1] 14  3 -1
```

7. Afficher la diagonale de `A` ;

```
  diag(A)
```

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

8. Rajouter le vecteur $\begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top$ à droite de la matrice `A` et stocker le résultat dans un objet appelé `B` ;

```
  B <- cbind(A, c(1,2,3))
  B
```

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

9. Retirer le quatrième vecteur de `B` ;

```
  B <- B[,-4]
  B
```

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

10. Retirer la première et la troisième ligne de `B` ;

```
  B[-c(1,3),]
```

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

11. Ajouter le scalaire `10` à `A` ;

```
 A + 10
```

```
##      [,1] [,2] [,3]
## [1,]   13   15   16
## [2,]    9   12   12
## [3,]   11    9    9
```

12. Ajouter le vecteur $\begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top$ à A ;

```
  A + c(1,2,3)
```

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

13. Ajouter la matrice identité I_3 à `A` ;

```
  diag(3)
```

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

```
  A + diag(3)
```

```
##      [,1] [,2] [,3]
## [1,]    4    5    6
## [2,]   -1    3    2
## [3,]    1   -1    0
```

14. Diviser tous les éléments de la matrice `A` par 2 ;

```
  A / 2
```

```
##      [,1] [,2] [,3]
## [1,]  1.5  2.5  3.0
## [2,] -0.5  1.0  1.0
## [3,]  0.5 -0.5 -0.5
```

15. Multiplier la matrice `A` par le vecteur $\begin{bmatrix}1 & 2 & 3\end{bmatrix}^\top$;

```
  A %*% c(1,2,3)
```

```
##      [,1]
## [1,]   31
## [2,]    9
## [3,]   -4
```

À ne pas confondre avec `A * c(1,2,3)`, qui effectue la multiplication terme à terme.

16. Afficher la transposée de `A` ;

```
  t(A)
```

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

17. Effectuer le produit avec transposition $A^\top A$.

```r
t(A) %*% B
```

```
##      [,1] [,2] [,3]
## [1,]   11   12   15
## [2,]   12   30   35
## [3,]   15   35   41
```

```r
crossprod(A,A)
```

```
##      [,1] [,2] [,3]
## [1,]   11   12   15
## [2,]   12   30   35
## [3,]   15   35   41
```

Exercice 4 (importation et exportation)

1. Télécharger le fichier `csv` à l'adresse suivante : egallic.fr/Enseignement/R/Exercices/donnees/notes.csv et le placer dans le répertoire courant du projet. Importer son contenu dans `R` ;

On peut soit utiliser la fonction `read.csv()` du *package* `utils`, soit la fonction `read_csv()` du *package* `readr` (plus rapide sur des gros volumes). Avec la première solution, le contenu est stocké sous forme de `data frame`, avec la seconde sous forme de `local data frame`.

```
# Téléchargement du fichier
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes.csv"
download.file(lien, destfile = "./notes.csv")
df <- read.csv("notes.csv")
library(readr)
df <- read_csv("notes.csv")
```

2. Importer à nouveau les données dans `R`, mais en utilisant fournissant cette fois le l'url directement à la fonction d'importation ;

```
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes.csv"
df <- read_csv(lien)
```

3. À présent, importer le contenu du fichier egallic.fr/Enseignement/R/Exercices/donnees/notes_decim.csv. Le séparateur de champs est un point virgule et le séparateur décimal est une virgule ;

Au lieu d'utiliser la fonction `read_csv()`, on fait appel à `read_csv2()`, qui fait en réalité appel la fonction `read_delim()`, en précisant le caractère de séparation *via* le paramètre `delim`.

```
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes_decim.csv"
df <- read_csv2(lien)
# Ou de manière équivalente
df <- read_delim(lien, delim=";")
```

4. Importer le contenu du fichier egallic.fr/Enseignement/R/Exercices/donnees/notes_h.csv. Le nom des colonnes n'est pas présent ;

Avec la fonction `read.csv`, il faut indiquer que l'en-tête est absente *via* le paramètre `header`. En utilisant la fonction `read_csv()`, il suffit de préciser que le nom des colonnes est absent *via* le paramètre `col_names`.

```
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes_h.csv"
df <- read_csv(lien, col_names=FALSE)
```

5. Importer le contenu du fichier egallic.fr/Enseignement/R/Exercices/donnees/notes_h_s.csv. La première ligne n'est pas à importer ;

On préciser à la fonction utilisée de sauter un certain nombre de lignes avec le paramètre `skip`.

```
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes_h_s.csv"
df <- read_csv(lien, skip=1)
```

6. Importer le contenu de la première feuille du fichier Excel egallic.fr/Enseignement/R/Exercices/donnees/notes.xlsx ;

Pour importer le contenu d'un classeur Excel dans `R`, le plus pratique est d'utiliser la fonction `read_excel()` du *package* `read_xl`. La version de ce *package* disponible sur le CRAN à la date d'écriture de cet exercice (0.1.0) ne permet pas encore d'importer un fichier en ligne. La version courante sur GitHub le permet, donc cette fonctionnalité devrait apparaître sur le CRAN dans les versions futures du *package*. Pour l'heure, nous allons d'abord télécharger le fichier dans le répertoire courant de la session `R`, puis le charger.

```
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes.xlsx"
download.file(lien, destfile = "./notes.xlsx", mode = "wb")
library(readxl)
df <- read_excel("notes.xlsx")
```

7. Importer le contenu de la seconde feuille (`notes_h_s`) du fichier Excel `egallic.fr/Enseignement/R/Exercices/donnees/notes.xlsx`. La première ligne est un commentaire à ne pas considérer durant l'importaiton ;

```
df <- read_excel("notes.xlsx", sheet = "notes_h_s", skip=1)
```

8. Importer le fichier `egallic.fr/Enseignement/R/Exercices/donnees/notes.rda` dans `R` ;

Pour charger un fichier de données au format `rda` ou `RData`, on utilise la fonciton `load()`. Dans cet exemple, le fichier source est sur l'Internet, donc il faut d'abord appliquer la méthode `url()` à notre chaîne de caractères indiquant l'emplacement du fichier.

```
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes.rda"
load(url(lien))
head(notes)
```

9. Exporter le contenu de l'objet `notes` de la question précédente au format `csv` (virgule en séparateur de champs, point en séparateur décimal, ne pas conserver le numéro des lignes).

Avec la fonction `write.csv()` du *package* `utils`, il est nécessaire de préciser que l'on ne souhaite pas inclure le nom des lignes *via* le paramètre `row.names`, ce qui est fait par défaut avec la fonction `write_csv()` du *package* `readr`.

```
write.csv(notes, file = "notes_2.csv", row.names = FALSE)
write_csv(notes, path = "notes_3.csv")
```

10. Importer le contenu du fichier `notes_2012.csv` contenu dans l'archive disponible à l'adresse suivante : `http://egallic.fr/Enseignement/R/Exercices/donnees/notes.zip`

Dans un prmeier temps, il s'agit de télécharger le fichier dans un répertoire temporaire.

```
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes.zip"
# Chemin vers dossier temporaire
td <- tempdir()
# Creation du fichier temporaire qui contiendra l'archive
tf <- tempfile(tmpdir=td, fileext=".zip")
# Telechargement dans cette archive
download.file(lien, tf)
```

Puis, on regarde le contenu du fichier :

```r
# Le nom des fichiers
unzip(tf, list=TRUE)$Name
```

```r
# Le nom du premier fichier
nom_fichier <- unzip(tf, list=TRUE)$Name[1]

# Charger les données dans R
df <-
  unz(description = tf, filename = nom_fichier) %>%
  read_csv2()
```

Exercice 5 (manipulation de tableaux de données)

1. À l'aide de la fonction `read_excel()` du *package* `readr`, importer le contenu de la feuille intitulée `notes_2012` du fichier Excel disponible à l'adresse suivante : http://egallic.fr/Enseignement/R/Exercices/donnees/notes_etudiants.xlsx et le stocker dans une variable que l'on nommera `notes_2012` ;

```r
library(readxl)
lien <- "http://egallic.fr/Enseignement/R/Exercices/donnees/notes_etudiants.xlsx"
notes_2012 <- read_excel(lien, sheet = "notes_2012")
```

2. Afficher les 6 premières lignes du jeu de données, puis les dimensions du tableau ;

```r
head(notes_2012)
dim(notes_2012)
ncol(notes_2012)
nrow(notes_2012)
```

3. Conserver uniquement la colonne `note_stat` du tableau de données `notes_2012` dans un objet que l'on appellera `tmp` ;

```r
notes_2012$note_stat
notes_2012[, "note_stat"]
select(notes_2012, note_stat)
```

4. Conserver uniquement les colonnes `num_etudiant`, `note_stat` et `note_macro` dans l'objet `tmp` ;

```r
notes_2012[, c("num_etudiant", "note_stat", "note_macro")]
notes_2012 %>% select(num_etudiant, note_stat, note_macro)
notes_2012 %>% select(num_etudiant:note_macro)
```

5. Remplacer le contenu de `tmp` par les observations de `notes_2012` pour lesquelles l'individu a obtenu une note de stat supérieure (strictement) à 10 ;

```
notes_2012 %>%
  filter(note_stat > 10)
```

6. Remplacer le contenu de `tmp` par les observations de `notes_2012` pour lesquelles l'individu a obtenu une note comprise dans l'intervalle $(10, 15)$;

```
notes_2012 %>%
  filter(note_stat > 10, note_stat < 15)
```

7. Regarder s'il y a des doublons dans le tableau de données `notees_2012` ; le cas échéant, les retirer du tableau ;

```
any(duplicated(notes_2012))
# Ou encore
notes_2012 <-
  notes_2012 %>%
  distinct()
```

8. Afficher le type des données de la colonne `num_etudiant`, puis afficher le type de toutes les colonnes de `notes_2012` ;

```
class(notes_2012$num_etudiant)
lapply(notes_2012, class)
str(notes_2012)
```

9. Ajouter au tableau `notes_2012` les colonnes suivantes :
   (a) `note_stat_maj` : la note de stat (`note_stat`) majorée d'un point,
   (b) `note_macro_maj` : la note de macro (`note_macro`) majorée de trois points (le faire en deux étapes : d'abord deux points en plus, puis un point) ;

```
notes_2012 %>%
  mutate(note_stat_maj = note_stat + 1)
notes_2012 %>%
  mutate(note_stat_maj = note_stat + 1,
         note_macro_maj = note_macro + 2,
         note_macro_maj = note_macro_maj + 1)
```

10. Renommer la colonne `year` en `annee` ;

```
notes_2012 <-
  notes_2012 %>%
  rename(annee = year)
```

11. Depuis le fichier `notes_etudiants.xlsx` (c.f. question 1), importer le contenu des feuilles `notes_2013`, `notes_2014` et `prenoms` et le stocker dans les objets `notes_2013`, `notes_2014` et `prenoms` respectivement ;

```
notes_2013 <- read_excel(lien,
                         sheet = "notes_2013", na = "NA")
notes_2014 <- read_excel(lien,
                         sheet = "notes_2014", na = "NA")
prenoms <- read_excel(lien,
                      sheet = "prenoms")
```

12. Empiler le contenu des tableaux de données `notes_2012`, `notes_2013` et `notes_2014` dans un objet que l'on nommera `notes` ;

```
notes <- rbind(notes_2012, notes_2013, notes_2014)
```

13. Fusionner les tableaux `notes` et `prenoms` à l'aide d'une jointure gauche, de manière à rajouter les informations contenues dans le tableau `prenoms` aux observations de `notes`. La jointure doit se faire par le numéro détudiant et l'année, l'objet final viendra remplacer le contenu de `notes` ;

```
notes <-
  notes %>%
  left_join(prenoms, by = c("num_etudiant", "annee"))
```

14. Trier le tableau `notes` par années croissantes et notes de macro décroissantes ;

```
notes <- arrange(notes, annee, desc(note_macro))
```

15. Changer le type des colonnes `annee` et `sexe` en facteur ;

```
notes <- note %>%
  mutate(annee = factor(annee),
         sexe = factor(sexe, levels = c("F", "H"),
                       labels = c("Femme", "Homme")))
```

16. Créer une colonne `apres_2012` qui prend la valeur `TRUE` si l'observation concerne une note attribuée après 2012 ;

```
notes <- note %>%
  mutate(apres_2012 = ifelse(annee > 2012, yes = TRUE, no = FALSE))
```

17. À l'aide de la fonction `summarize()` du *package* `dplyr`, calculer :
    (a) la moyenne et l'écart-type annuels des notes pour chacune des deux matières,

```
infos_notes <-
  notes %>%
  group_by(annee) %>%
  summarize(moyenne_stat = mean(note_stat, na.rm=TRUE),
            sd_stat = sd(note_stat, na.rm=TRUE),
            moyenne_macro = mean(note_macro, na.rm = TRUE),
            sd_macro = sd(note_macro, na.rm=TRUE))
```

(b) la moyenne et l'écart-type annuels et par sexe des notes pour chacune des deux matières ;

```
infos_notes_2 <-
  notes %>%
  group_by(annee, sexe) %>%
  summarize(moyenne_stat = mean(note_stat, na.rm=TRUE),
            sd_stat = sd(note_stat, na.rm=TRUE),
              moyenne_macro = mean(note_macro, na.rm = TRUE),
              sd_macro = sd(note_macro, na.rm=TRUE))
```

18. En utilisant la fonction `gather()` du *package* `tidyr`, créer un tableau dans lequel chaque ligne renseigne le numéro d'étudiant, l'année, le prénom, le sexe, l'enseignement (macro ou stat) et la note ;

```
library(tidyr)
notes_l <-
  notes %>%
  gather(key = note, value = enseignement, note_stat, note_macro)
```

19. En repartant de l'objet obtenu à la question précédente, utiliser la fonction `spread()` du *package* `tidyr` pour retomber sur le même tableau que `notes`.

```
notes_l %>%
  spread(note, enseignement)
```

D'autres manières de répondre à chaque question de cet exercice :

```r
notes_2012_df <- data.frame(notes_2012)
head(notes_2012_df)
dim(notes_2012_df)
ncol(notes_2012_df)
nrow(notes_2012_df)
notes_2012_df$note
notes_2012_df[, "note"]
notes_2012_df[, c("num_etudiant", "note_stat", "note_macro")]
notes_2012_df[notes_2012_df$note_stat>10,]
notes_2012_df[notes_2012_df$note_stat>10 & notes_2012_df$note_stat<15,]
any(duplicated(notes_2012_df))
notes_2012_df <- notes_2012_df[!duplicated(notes_2012_df),]
class(notes_2012_df$num_etudiant)
lapply(notes_2012_df, class)
str(notes_2012_df)
within(notes_2012_df,{note_stat_maj = note_stat + 1}) %>%
  head()
within(notes_2012_df,{
  note_stat_maj <- note_stat + 1
  note_macro_maj <- note_macro + 2
  note_macro_maj <- note_macro_maj + 1}) %>%
  head()
ind <- which(colnames(notes_2012_df) == "year")
colnames(notes_2012_df)[ind] <- "annee"
notes_2013_df <- read_excel("donnees/notes_etudiants.xlsx",
                            sheet = "notes_2013", na = "NA") %>%
  data.frame()
notes_2014_df <- read_excel("donnees/notes_etudiants.xlsx",
                            sheet = "notes_2014", na = "NA") %>%
  data.frame()
notes_df <- rbind(notes_2012_df, notes_2013_df, notes_2014_df)
prenoms_df <- read_excel("donnees/notes_etudiants.xlsx",
                         sheet = "prenoms") %>%
  data.frame()
notes_df <-
  merge(notes_df, prenoms_df, by = c("num_etudiant", "annee"))
notes_df <-
  notes_df[with(notes_df, order(annee, desc(note_macro))),]
aggregate(notes_df[, c("note_stat", "note_macro")],
          by = list(annee = notes_df$annee),
          FUN = function(x) cbind(moy = mean(x, na.rm=T),
                                  sd = sd(x, na.rm=T)))

aggregate(notes_df[, c("note_stat", "note_macro")],
          by = list(annee = notes_df$annee, sexe = notes_df$sexe),
          FUN = function(x) cbind(moy = mean(x, na.rm=T),
                                  sd = sd(x, na.rm=T)))
```

Exercice 6 (manipulation de chaînes de caractères)

1. Créer les objets `a` et `b` afin qu'il contiennent respectivement les chaînes de caractères suivantes : `23 à 0` et `C'est la piquette, Jack !` ;

```
a <- "23 à 0"
b <- "C'est la piquette, Jack !"
```

2. Créer le vecteur `phrases` de longueur 2, dont les deux éléments sont `a` et `b` ;

```
phrases <- c(a, b)
```

3. À l'aide de la fonction appropriée dans le *package* `stringr`, afficher le nombre de caractères de `a`, de `b`, puis appliquer la même fonction à l'objet `phrases` ;

```
library(stringr)
str_length(a)
str_length(b)
str_length(phrases)
```

4. En utilisant la fonction `str_c()`, concaténer `a` et `b` dans une seule chaîne de caractères, en choisissant la virgule comme caractère de séparation ;

```
str_c(a, b, sep = ",")
```

5. Concaténer les deux éléments du vecteur `phrases` en une seule chaîne de caractères, en les séparant par le caractère de retour à la ligne, puis utiliser la fonction `cat()` pour afficher le résultat dans la console ;

```
library(stringr)
str_c(phrases, collapse = "\n")
cat(str_c(phrases, collapse = "\n"))
# Différent de :
str_c(phrases, sep = "\n")
```

6. Appliquer la même fonction que dans la question précédente à l'objet suivant : `c(NA, phrases)` et commenter ;

```
str_c(c(phrases, NA), collapse = "\n")
cat(str_c(c(phrases, NA), collapse = "\n"))
```

Il y a l'objet `NA` dans le vecteur pour lequel on souhaite effectuer une concaténation. Cette dernière échoue donc, et le résultat dans la console, suite à l'appel de la fonction `cat()` sera également `NA`.

7. Mettre en majuscules, puis en minuscules les chaînes du vecteur `phrases` (afficher le résultat, ne pas modifier `phrases`) ;

```
str_to_upper(phrases)
str_to_lower(phrases)
```

8. À l'aide de la fonction `word()` du *package* `stringr`, extraire le mot `la`, puis `Jack` de la chaîne `b` ;

```
word(b, 2)
word(b, -2)
```

9. Même question que la précédente, en utilisant la fonction `str_sub()` ;

```
str_sub(b, 20, 23)
str_sub(b, -6, -3)
# Ne pas spécifier le paramètre end :
str_sub(b, 20)
```

10. À l'aide de la fonction `str_detect()`, rechercher si le motif `piqu` puis `mauvais` sont présents dans `b` ;

```
str_detect(b, "piqu")
str_detect(b, "mauvais")
```

11. À l'aide de la fonction `str_detect()`, rechercher si le motif `piqu` est présent dans les éléments du vecteur `phrases` ;

```
str_detect(phrases, "piqu")
```

12. À l'aide de la fonction `str_detect()`, rechercher si le motif `piqu` ou le motif `à` sont présents dans les éléments du vecteur `phrases` ;

```
str_detect(phrases, "piqu|à")
```

13. En utilisant la fonction `str_locate()`, retourner les positions de la première occurence du caractère `a` dans la chaîne `b`, puis essayer avec le caractère `w` pour observer le résultat retourné ;

```
str_locate(b, "a")
str_locate(b, "w")
```

14. Retourner toutes les positions du motif `a` dans la chaîne `b` ;

```
str_locate_all(b, "a")
```

15. En utilisant la fonction `str_replace()`, remplacer la première occurence du motif `a`, par le motif `Z` (afficher le résultat, ne pas modifier `phrases`) ;

```
str_replace(b, "a", "Z")
```

16. Remplacer toutes les occurences de `a` par `Z` dans la chaîne `b` (afficher le résultat, ne pas modifier `phrases`) ;

```
str_replace_all(b, "a", "Z")
```

17. Utiliser la fonction `str_split()` pour séparer la chaîne `b` en utilisant la virgule comme séparateur de sous-chaînes ;

```
str_split(b, ",")
```

18. Retirer tous les caractères de ponctuation de la chaîne `b`, puis utiliser la fonction `str_trim()` sur le résultat pour retirer les caractères blancs du début et de la fin de la chaîne.

```
str_replace_all(b, "[:punct:]", "") %>%
  str_trim()
```

Exercice 7 (manipulation de dates)

1. En utilisant la fonction `as.Date()`, stocker la date du 29 août 2015 dans un objet que l'on appellera `d` puis afficher la classe de l'objet ;

```
d <- as.Date("2015-08-29")
class(d)
unclass(d)
```

2. À l'aide de la fonction appropriée, afficher la date du jour ;

```
as.Date(Sys.time())
```

3. À l'aide de la fonction `as.Date()`, stocker sous forme de date la chaîne de caractères suivante : `29-08-2015` ;

```
d <- as.Date("29-08-2015", format = "%d-%m-%Y")
```

4. Utiliser les fonctions `as.POSIXct()` et `as.POSIXlt` pour stocker la chaîne de caractères `2015-08-29 20:30:56` sous forme de dates dans des objets nommés `d_ct` et `d_lt` respectivement ; utiliser ensuite la fonction `unclass()` sur les deux objets pour comparer la façon dont `R` a stocké l'information :

```
d_ct <- as.POSIXct("2015-08-29 20:30:56")
d_lt <- as.POSIXlt("2015-08-29 20:30:56")
```

Les objets `POSIXct` sont stockés en secondes écoulées depuis le premier janvier 1970, les objets `POSIXlt` sont une liste dont les élṁeents correspondent aux comosantes de la date.

```
unclass(d_ct)
unlist(unclass(d_lt))
```

```
unclass(as.POSIXct("1970-01-01 00:00:10"))
unclass(as.POSIXct("1970-01-01 00:00:10", tz = "UTC"))
```

5. Utiliser la fonction appropriée du *package* `lubridate` pour stocker la chaîne de caractères `2015-08-29` sous forme de date ;

```
library(lubridate)
d_l <- ymd("2015-08-29")
class(d_l)
```

6. Même question avec la chaîne `2015-08-29 20:30:56` ;

```
ymd_hms("2015-08-29 20:30:56")
```

7. Utiliser la fonction `ymd_hms()` pour stocker la date et l'heure actuelle, en précisant le fuseau horaire, puis afficher la date et l'heure correspondantes à New York City ;

```
date_paris <- ymd_hms(Sys.time(), tz = "Europe/Paris")
with_tz(date_paris, "America/New_York")
```

8. Considérons le vecteur `x` :

```
x <- c(ymd_hms("2015-08-29 20:30:56", tz = "Europe/Paris"),
       ymd_hms("2015-09-15 08:10:33", tz = "Europe/Paris"))
```

   Extraire l'année, le mois, le jour, les heures, les minutes et les secondes du premier élément de `x` à l'aide des fonctions appropriées du *package* `lubridate` ;

```
year(x[1])
month(x[1])
day(x[1])
hour(x[1])
minute(x[1])
second(x[1])
```

9. Appliquer les mêmes fonctions au vecteur `x` ;

   Les fonctions de la question précédente sont vectorisées.

```
year(x)
month(x)
day(x)
hour(x)
minute(x)
second(x)
```

10. Au premier élément de `x`, ajouter :
    — une seconde,

    Les questions qui suivent permettent de voir la différence entre l'ajout de durées ou d'époques. Les époques ne prennent pas en compte les années bissextiles. Pour ajouter une durée, il suffit d'utiliser la fonction appropriée du *package* `lubridate`, dont le nom correspond à la durée au pluriel, et fournir en paramètre la valeur souhaitée. Pour les époques, il suffit de rajouter le préfixe `d` au nom de la fonction.

    ```
    x[1] + 1
    x[1] + seconds(1)
    ```

    — un jour,

    ```
    x[1] + days(1)
    ```

    — un mois

    ```
    x[1] + months(1)
    ```

    — deux années ;

    ```
    x[1] + years(2)
    x[1] + dyears(2)
    ```

    On voit bien la différence entre une durée et une époque ici, puisque l'année 2016 est bissextile :

    ```
    leap_year(2016)
    ```

11. Tester si la date du premier élément de `x` vient avant celle du second élément ;

```
x[1] < x[2]
```

12. En utilisant la fonction `new_interval()` du *package* `lubridate`, créer un intervalle de dates entre les deux éléments de `x`, puis afficher le nombre de jours, puis le nombre d'heures, puis le nombre d'années séparant les deux dates ;

```
new_interval(x[1], x[2]) / ddays(1)
new_interval(x[1], x[2]) / dhours(1)
new_interval(x[1], x[2]) / dyears(1)
```

13. En utilisant la fonction `seq()`, créer une séquence de dates avec un intervalle de 5 jours entre chaque date, commençant à la date du premier élément de `x` et se terminant à la date du second élément de `x` (la séquence sera tronquée avant) ;

```
    seq(x[1], x[2], by = "5 days")
```

14. Convertir en date les deux chaînes de caracères suivantes : `Sam 29 Août 2015` et `Sat 29 Aug 2015` ;

```
    parse_date_time("Sam 29 Août 2015", orders = "dmy", locale="fr_fr")
    parse_date_time("Sat 29 Aug 2015", orders = "dmy", locale="en_gb")
```

Sous Windows, le paramètre `locale` doit prendre une valeur différente :

```
    parse_date_time("Sam 29 Août 2015", orders = "dmy", locale="french_fr")
    parse_date_time("Sat 29 Aug 2015", orders = "dmy", locale="english_gb")
```