Préambule

Le cours est divisé en 5 modules, repris du cours de **Fabien Perez** donné en 2023 et 2022 à l'UTT ; je le remercie pour cette transmission.

Certains exemples et exercices ont par ailleurs été inspirés par :

- Le cours d'Introduction à la Statistique de l'ENSAE Paris (actuellement enseigné par M. Lerasle)
- Le livre R pour la statistique et la sciences des données (sous la direction de F. Husson) `https://r-stat-sc-donnees.github.io/`
- Le Github `https://larmarange.github.io/analyse-R/` (sous la direction de J. Larmarange)

Je les remercie pour leurs contributions pédagogiques.

Le cours est calibré sur 11h et est structuré comme suit :

1. Découverte de R
2. Premières manipulations et analyses de données
3. Réduction de dimension (ACP) et clustering
4. Méthodes de régression
5. Prédiction et classification

Tout au long des 5 modules, les résultats/sorties R ne sont volontairement pas inclus dans les fichiers pour encourager la pratique individuelle du langage et du logiciel.

Deux TP supplémentaires (ACP à la main et k-means à la main) sont disponibles pour les étudiants les plus à l'aise et/ou souhaitant approfondir leurs connaissances.

Modalités d'évalution

Le cours sera évalué par rendu d'exercices. Il y a un exercice par module. La répartition des points est la suivante : 3 points pour le module 1, 4 points pour le module 2, 4 points pour le module 3, 4 points pour le module 4 et 5 points pour le module 5. **Les réponses sans commentaire ni interprétation ne seront pas valorisées**.

Veuillez rendre **la totalité des exercices dans un seul fichier .qmd ou .Rmd**. Ce fichier doit être envoyé à l'adresse mail mathilde.gerardin@insee.fr en une seule fois et **le 12 janvier 2026 au plus tard** (-2 points par jour de retard).

Quelques liens utiles

— utilitR, une documentation collaborative sur R créée par des agents de l'Insee `https://book.utilitr.org/`

— Stack Overflow, un forum de questions-réponses très fourni :`https://stackoverflow.com/`
Astuce : taper sa question en anglais dans Google + le mot-clé stackoverflow

— Une fiche aide-mémoire R : `https://cran.r-project.org/doc/contrib/Kauffmann_aide_memoire_R.pdf`

— La cheat sheet de ggplot2 en français : `https://thinkr.fr/pdf/ggplot2-french-cheatsheet.pdf`

— D'autres cheat sheets en anglais : `https://www.rstudio.com/resources/cheatsheets/`

— Kaggle (Site de compétition et d'échanges autour du Machine Learning : du code et des datasets) : `https://www.kaggle.com/`

Module 1 - Découverte de R

R est un logiciel libre conçu pour l'analyse de données. Il est donc disponible gratuitement sur le site `https://www.r-project.org/`. Il est développé par des volontaires depuis une vingtaine d'années et de nombreux outils continuent d'être implémentés dans ce langage grâce à une communauté très active.
Pour ce cours et pour vos analyses en R nous utiliserons l'environnement de développement `RStudio`, téléchargeable sur le site `https://www.rstudio.com/products/rstudio/download/`. Il existe d'autres environnement de développement pour coder en R : VSCode est par exemple de plus en plus populaire car il offre de nombreuses fonctionnalités et peut être utilisé avec une variété de langages de programmation (R, Java, JavaScript, C++...).

1 L'environnement RStudio

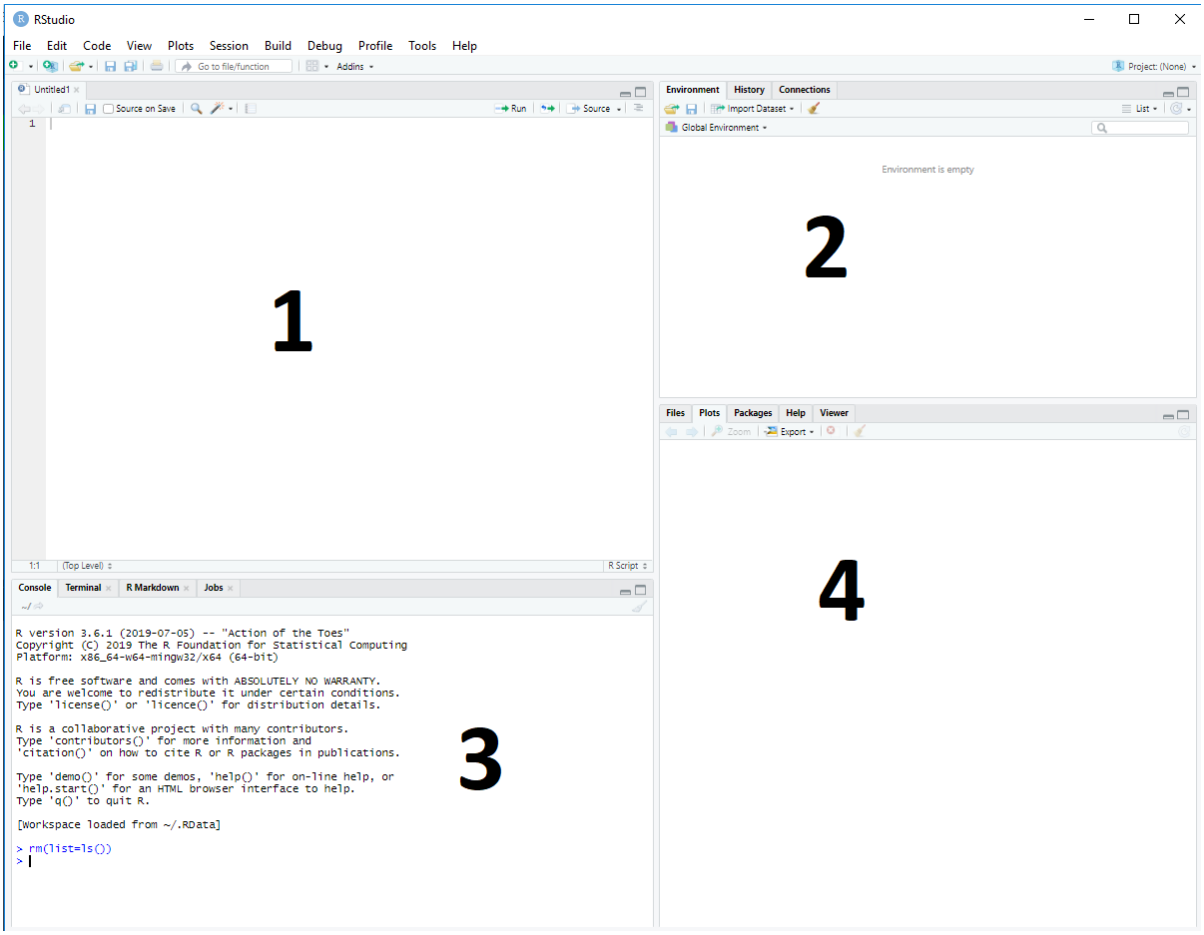


FIGURE 1 – L'environnement RStudio - 4 quadrants

L'environnement RStudio est composé de 4 quadrants comme on peut le voir sur la Figure 1 ci-dessus.

— Le quadrant 1 (en haut à gauche) correspond à la fenêtre d'éditeur. C'est ici que vous écrirez votre code et que vous alimenterez votre programme écrit en R. Pour exécuter les commandes que vous écrirez il suffit de les sélectionner et appuyer sur `Ctrl` `Ent` par exemple.

— Dans le quadrant 2 (en haut à droite) vous trouverez notamment les variables, les données que vous aurez créées et chargées ainsi qu'un historique des commandes exécutées.

— Dans le quadrant 3 (en bas à gauche) se trouve la console. Les résultats des commandes que vous exécutez y figureront. Il est d'ailleurs possible d'écrire des commandes directement dans la console. Pour pouvoir garder la trace des commandes exécutés, nous vous conseillerons cependant d'écrire le code dans le quadrant 1.

— Dans le quadrant 4 (en bas à droite) vous trouverez les graphes ("plots") ainsi que les librairies ("packages") disponibles. Un explorateur de fichier est également présent.

  Pour fixer votre dossier de travail dans R, il faut utiliser la commande `setwd()`. Cette commande indique à R le dossier racine où chercher les données et où exporter les sorties. Dans le chemin, attention à écrire les *slashs* dans le bon sens pour R.

```
setwd("C:/Users/XSY9FN/Documents/2_ENSEIGNEMENT/UTT/R")
```

2 Les premières commandes

Pour commencer à travailler sur un fichier R, il suffit de cliquer sur : `File` → `New File` → `R script`. Vous pouvez sauver le nouveau fichier en le nommant, par exemple `Module1.R`

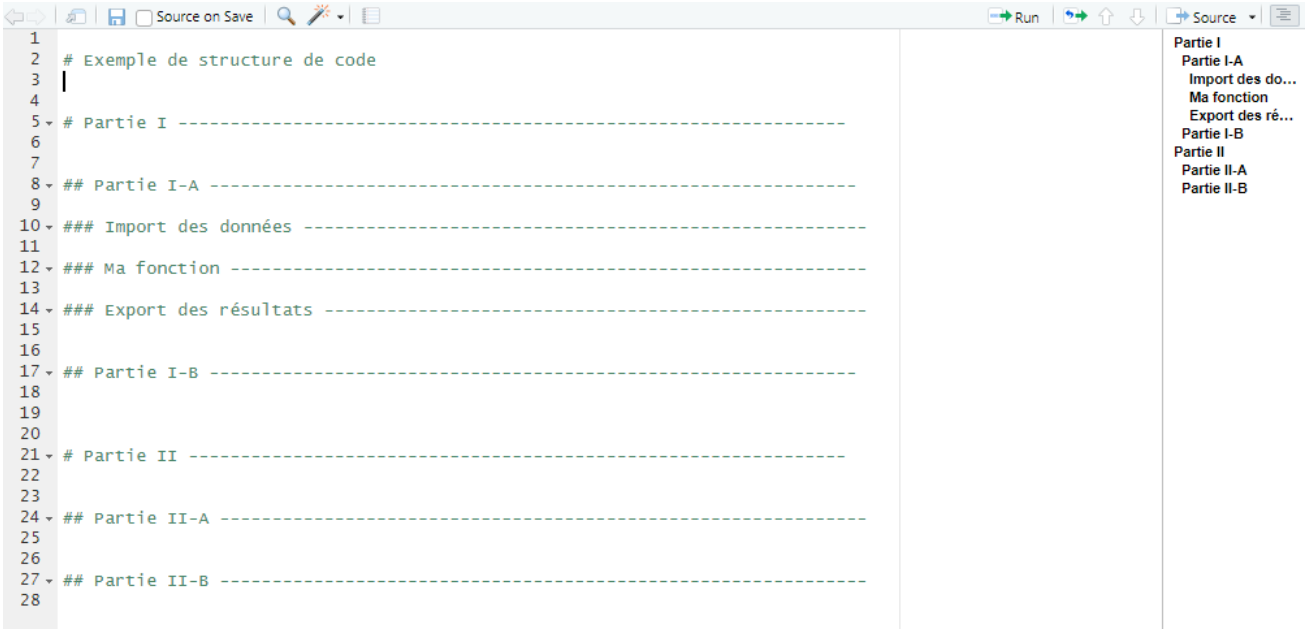
Que cela soit dans la console ou dans votre fenêtre d'éditeur essayez de lancer vos premières commandes en faisant quelques essais.

```
57*42
12+26 # Renvoie le resultat de cette operation
```

Comme vous le voyez ci-dessus, le # permet d'écrire des commentaires qui ne seront pas inter-

prétés/compilés par R.

Le # permet aussi de structurer son code R en plusieurs parties et sous-parties, en utilisant le raccourci `Ctrl` `Maj` + `R` :



Tous les opérateurs habituels sont disponibles, ainsi que les fonctions logarithmiques, trigonométriques, racine carrée, etc. `pi` est une variable pré-définie dans R dont la valeur est celle de π :

```
pi*sqrt(10)+exp(4)
```

Affectation de variable

Il existe 3 façons d'affecter une valeur à une variable (sans différence entre elles).

```
a <- 6
a = 6
6 -> a
# On utilisera la premiere facon pour unifier notre code

b <- (a - 1) * (a + 1) # On peut utiliser la valeur d'une variable
# pour creer d'autres variables
b # pour afficher la variable
```

```r
a <- 4
b #b a-t-il change ?
```

L'aide sur R vous sera sûrement utile. Deux commandes permettent d'accéder à l'aide sur une commande.

```r
help(mean)
?mean
```

Dans les deux cas, l'aide s'affichera en bas à droite. Comme tout datascientist, vous serez également amenés à utiliser les ressources présentes sur internet pour résoudre certains problèmes. Avant de rentrer dans des détails sur les objets en R, profitons de cette section pour présenter la façon de connaître la liste des variables existantes et la façon de supprimer des variables

```r
ls() # liste des variables saisies
rm(b) # permet de supprimer la variable b
rm(list=ls()) # permet de supprimer toutes les variables
# (utile en debut de fichier par exemple pour repartir a zero)
```

3 Les objets en R

Il existe trois types principaux d'objets en R **`character, numeric, boolean/logical`**. Il existe aussi le type "factor" que l'on évoquera plus tard. Les fonctions as.character/as.numeric/as.logical permettent par exemple de convertir les objets.

```r
charc <- "hello"
class(charc)
as.character(10)

charc2 <- "you"
paste(charc, charc2, sep = "␣")
paste(charc, charc2, sep = "_")
```

```r
nombre <- 10
class(nombre)

as.logical(nombre) #transforme le nombre en booleen (TRUE par défaut)

boolean <- TRUE

a<-4

boolean2 <- (a == 1)

a != 1
a > 1

class(boolean)
as.numeric(boolean2) # par défaut, FALSE = 0, TRUE = 1
```

3.1 Les vecteurs

On utilise en général la fonction c() pour créer des vecteurs. On peut utiliser les fonctions seq() et rep() également comme on peut le voir ci-dessous.

```r
# Creation d'un vecteur avec la fonction c()
# qui assemble et construit le vecteur:
vec1 = c(3,9,1,9,9,3)
vec2 = c(2,30,4)

x = seq(2, 10)
y = seq(from = 1, to = 10, by = 2)
y2 = seq(from = 10, to = -5, by = -2)

vu = runif(100) # La fonction runif()
# genere des nombres pseudo-aleatoires
# depuis une distribution de probabilite uniforme

# Pour garantir la reproductibilité, on peut fixer
```

```
# la racine du générateur aléatoire
set.seed(123)
vu = runif(100)

as.numeric(c(TRUE, FALSE))
as.logical(c(1,0,-2,1651))
```

On peut utiliser la valeur d'un vecteur pour créer d'autres vecteurs :

```
z = c(x,y)

w = rep(y,3) # répète le vecteur x 3
w = rep(y, times = 3) # idem

w = rep(y, each=3) # répète chaque élément du vecteur x 3

w = rep(y, times=3, each = 4)
```

Opérations mathématiques sur les vecteurs :

```
x+y # Attention R fait toutes ses
# operations composantes par composantes par defaut
x+7
x*x
x*y #attention
x+3
z^2
z*z
```

On peut demander la valeur du troisième élément du vecteur à l'aide de l'opérateur [] :

```
x[3]

# On peut demander la valeur des elements 1 et 3 du vecteur
```

```
# en fournissant plusieurs indices, comme suit :

x[c(1,3)] # = on peut acceder aux elements d'un vecteur
# en utilisant un autre vecteur
```

On peut changer la valeur d'un élément d'un vecteur. Et même changer tous les éléments d'un vecteur qui vérifie une condition comme on le voit sur l'exemple ci-dessous.

```
x=c(1,13,5,5,12,14,16,3,9)

x[3]<-99

x[x>6]<-6

x[0]<-3 #rien ne change
```

Les variables catégorielles (factors)

```
sex_vector = c("Homme", "Femme", "Homme", "Homme", "Femme")

# Conversion de sex_vector en factor
factor_sex_vector = factor(sex_vector)

# Afficher le factor
print(factor_sex_vector)

# Voir les modalites
levels(factor_sex_vector)
```

Deux types de factors : variables nominales/variables ordinales

```
# Animaux : variable nominale
animals_vector = c("Elephant", "Giraffe", "Donkey", "Horse")
factor_animals_vector = factor(animals_vector)
```

```
factor_animals_vector

# Temperature : variable ordinale
temperature_vector = c("High", "Low", "High","Low", "Medium")
factor_temperature_vector = factor(temperature_vector, order = TRUE,
                 levels = c("Low", "Medium", "High"))
factor_temperature_vector
```

3.2 Les matrices

```
# Creation de matrices :
x = 1:12
dim(x) = c(3,4)
?dim
xy = matrix(1:12, nrow=3, byrow=T) # Ici "T" veut dire "TRUE"
xy = matrix(1:12, nrow=3, byrow=F) # Ici "F" veut dire "FALSE"
z = matrix(1:4, nrow=2, byrow=T)
matrix(1:6, nrow = 2, ncol = 3)
matrix(1:6, nrow = 2, ncol = 3, byrow = TRUE)
```

Opérations sur les matrices :

```
t(xy) # Transposee
z^2 # Chaque element de z est multiplie par lui-meme
z*z # Chaque element de z est multiplie par lui-meme
z%*%z # Produit matriciel
```

3.3 Les listes

Les listes forment un autre types d'objet en R. Il s'agit de concaténer plusieurs objets au sein d'un plus grand objet, la liste. On peut regarder l'exemple ci-dessous.

```
vecteur=1:7
matrice=matrix(1:6, nrow = 2, ncol = 3)

maliste=list(vecteur, matrice, 16, TRUE)
```

3.4 Les data-frames/tableaux de données

Les data-frames (tableaux de données) forment un type d'objet très utilisé en R pour l'analyse de données. Nous reviendrons sur cet objet dans les prochains modules. Vous pouvez voir ci-dessous une manière de créer un data-frame.

```
Newdf=data.frame(Prenom=c("Fabien","Thomas","Marie"), age=c(29,26,23),
                 nationalite=c("francaise","francaise","francaise"))

Newdf$Prenom

sum(Newdf$age)
```

4 Les fonctions

Comme avec tout langage de programmation, il est possible de construire des fonctions en R. De nombreuses fonctions sont déjà codés en R comme on l'a vu avec les fonctions *mathématiques* précédemment. De plus, de nombreuses fonctions existantes s'appliquent aux vecteurs comme on le voit ci-dessous.

```
x=1:10

length(x)
min(x)
max(x)
range(x)
sum(x)
mean(x)
```

```
sd(x)
summary(x)
```

Pour créer une fonction, on procède comme ci-dessous en mettant entre parenthèses les arguments de la fonction que nous créons.

```
carre = function(x) {
x^2
}
carre(3)
```

5 Quelques outils de programmation en R

Les conditions en R

En R, on peut évidemment utiliser des conditions du type "si ..., alors ..." (if/else)

```
 if((2+2 == 4)) {
  print("Hello")
}
a = -3
if(a > 1) {
  print("Good␣Morning")
} else {
  print("Good␣Afternoon")
}

if(a > 1) {
  print("Good␣Morning")
} else if (a == 0){
  print("Good␣Afternoon")
} else {
  print("Good␣Evening")
}
```

Les boucles

En R, il est possible de réaliser des boucles pour exécuter des commandes un certain nombre de fois. Comme dans d'autres langages, il est possible d'utiliser les conjonctions `for` `while` pour effectuer des boucles comme on peut le voir ci-dessous.

```
#avec for
for (i in 1:10) {
print(i)}

#avec while
i=1
while(i<11){
print(i)
i<-i+1}
```

Il est important de noter qu'en R, les boucles impliquent une complexité temporelle bien plus grande que les opérations sur des vecteurs. On peut le voir avec l'exemple ci-dessous où l'on va calculer une somme de 10^8 1.

```
#avec for
a<-0
system.time(for (i in 1:10^8) {a<-a+1})

#avec vecteur
system.time(sum(rep(1,10^8)))

# tps utilisateur (user) : temps CPU utilisé
# par le processus dans l'espace utilisateur.
# = tps que le processeur a passé à exécuter
# les instructions de votre code R

# tps système (system) : temps CPU utilisé
# dans l'espace noyau (ou système) pour
# accomplir des tâches liées à votre processus

# temps écoulé (elapsed) : temps réel écoulé
# entre le début et la fin de l'exécution du code
```

Quelques autres fonctions utiles faisant appel aux données de votre système :

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

6 Les librairies/packages

Comme dit précédemment, une grande communauté d'utilisateurs contribue au développement de R. En particulier, des volontaires développent des librairies/packages qui contiennent un certain nombre de commandes qui vont faciliter les manipulations et les analyses de données. Dans ce cours, nous allons utiliser les extensions liées au package `tidyverse` (en particulier `dplyr` et `ggplot2`). Il y a deux commandes principales à connaître : la commande d'installation de package (à faire une seule fois) et la commande de chargement de la librairie en question (à faire à chaque utilisation)

```
install.packages("tidyverse")
library(tidyverse)
```

Même si un package n'a pas été chargé à l'aide de la fonction `library()`, il est possible d'appeler ses fonctions avec l'opérateur `::`. Par exemple, même sans avoir chargé le package ggplot2, il est possible d'initialiser un plot avec la commande suivante : `ggplot2::ggplot(data = df)`.

Il peut arriver qu'un de vos programmes génère une erreur parce que vous utilisez un package dans une version trop ancienne. Pour mettre à jour un package, il suffit de le réinstaller avec `install.packages()`. Cette méthode de mise à jour est la plus simple, et résout la très grande majorité des problèmes de version de packages.

L'inconvénient principal des librairies R lors des montées de version de `R` est que les packages doivent être recompilés ou réinstallés dans le nouvel environnement : cela peut être fastidieux, surtout si vous utilisez de nombreux packages ou si des dépendances complexes sont impliquées. Par ailleurs certains packages ne sont pas immédiatement mis à jour pour fonctionner avec la nouvelle version de `R`, ce qui peut être bloquant pour vos travaux. Une solution est d'utiliser l'outil et package `renv` qui créé un environnement isolé de votre installation de `R` et sauvegarde les versions des packages utilisés dans votre projet.

7 RMarkdown

RMarkdown est un outil très populaire dans l'écosystème R qui permet d'associer pour un même projet, le code, les sorties et les commentaires. Nous recommandons l'utilisation de cet outil pour rendre votre travail lisible et reproductible aisément.
Pour créer un fichier RMarkdown, il suffit de cliquer sur File → New File → R Markdown.
Les morceaux de code écrit en R seront dans des chunks :

````
```{r}
5*7
```
````

Et les sorties apparaissent automatiquement sauf si l'on précise results="hide" comme ci-dessous.

````
```{r, results="hide"}
5*7
```
````

Réciproquement on peut préciser echo=FALSE si l'on veut voir seulement le résultat et non le code, ou encore include=FALSE si l'on veut que le code tourne mais qu'on ne voie ni le code ni la sortie.
Pour écrire du texte/des commentaires il suffit d'écrire entre les chunks.
Pour écrire des titres on utilise les #. Pour l'italique et le gras on utilisera des astérisques.

```
# Titre de niveau 1
## Titre de niveau 2
### Titre de niveau 3

*En Italique*
**En Gras**

<span style="color:red"> En rouge </span>
```

Pour plus d'aides et de commandes sur Rmarkdown, nous vous conseillons de télécharger la "cheat sheet" de Rmarkdown (Help → Cheat Sheet → RMarkdown)

8 Quarto

Quarto est l'évolution de R Markdown et s'impose progressivement dans la communauté R. Quarto reprend et généralise les fonctionnalités principales de R Markdown, mais alors que R Markdown est fortement lié à R, Quarto a été conçu pour être indépendant du langage et fonctionne avec plusieurs environnements de programmation (R, Python, Julia).

Parmi les améliorations, Quarto introduit une gestion plus robuste des métadonnées, offre une meilleure prise en charge de la génération de sites web et de livres et simplifie la production de documents multi-langages.

À noter que les documents .Rmd existants peuvent généralement être convertis en .qmd (le format natif de Quarto) sans modification majeure. Cela permet de migrer facilement des projets existants vers Quarto. De même les compétences acquises en R Markdown sont directement applicables à Quarto, le coût de la transition de l'un à l'autre est donc quasiment nul.

Pour créer un fichier Quarto, il suffit de cliquer sur File → New File → Quarto Document. Voilà un exemple de code :

````
---
title: "Analyse des données"
author: "Votre Nom"
format: html
---

# Premier exemple

Voici un exemple d'analyse utilisant Quarto.

## Analyse du jeu de données *cars*

```{r}
# Code R
summary(cars)
```
````

9 Exercice - Module 1

Les exercices sont à faire en Quarto (ou Rmarkdown) et doivent être rendu dans le même fichier. Vous pouvez créer le fichier `Exercice_R_NOM_PRENOM.qmd` (ou `Exercice_R_NOM_PRENOM.Rmd`) que vous compléterez à chaque module.

1. Créer le vecteur qui contient tous les multiples de 3 entre 1 et 50.
2. Créer la fonction "tronque()" qui prend en argument un nombre x et un vecteur vec et qui change tous les éléments du vecteur vec supérieur à x en x. Tester la fonction sur le vecteur que vous avez créé dans la question 1.
3. Créer la fonction "maxi()" qui a un vecteur numérique associe son élément maximum "à la main". On n'utilisera pas les fonctions de R déjà implémentées max(),min(),sort() mais on utilisera une boucle `for`.
4. Créer un data-frame regroupant 10 étudiants avec leur nom, prénom, âge, sexe, entreprise dans laquelle est effectuée l'alternance. Calculer l'âge moyen des étudiants et la proportion de femmes. Créer une variable de tranche d'âge avec au moins trois modalités et calculer l'âge moyen et la proportion de femmes par tranche d'âge.