

Traitement des données avec le logiciel R

– Travaux Dirigés –

EXERCICE N°1

1. Créer un dossier dans votre répertoire personnel pour accueillir les fichiers utiles à ces TD.
2. Ouvrir RStudio et faire de votre dossier le répertoire de travail (à l'aide du menu approprié).
3. Récupérer le chemin de votre répertoire de travail à l'aide de l'instruction `getwd()`.
4. Créer un nouveau document que vous pouvez appeler TD. La première instruction de votre fichier sera `setwd(...)` appliquée au bon chemin. Enregistrer ce document que vous pouvez nommer « TD.R ».
5. Enregistrer votre espace de travail (qui est pour l'instant vide) sous le nom « TD.RData ».

Rappelons le fichier `.R` contiendra votre script et que le fichier `.RData` contiendra les objets créés, votre base de données.

EXERCICE N°2

Que renvoient les instructions suivantes ?

$1 / 2^2$	<code>a <- 2 < 1</code>	<code>c(1,2,3) + 1</code>	<code>as.integer(c(1,"a"))</code>
$1 : 3 \times 2$	<code>a</code>	<code>c(1,2,3) + c(1,2)</code>	<code>as.logical(c(0,1,2,3))</code>
$2^2 \wedge 3$	<code>b <- sqrt(-2)</code>	<code>b <- c(1,"a")</code>	<code>c(TRUE,FALSE,0,1,2)</code>
<code>a <- sqrt(4)</code>	<code>T <- 3</code>	$(1 : 3) \times 2$	<code>c <- c(2,4,8,16)</code>
<code>a <- 1 / 0</code>	<code>TRUE <- 3</code>	<code>c(1,2) \times 1 : 4</code>	<code>c[2] ; c[-2]</code>
<code>a</code>	<code>1a <- 3</code>	<code>2 \times \text{rep}(1 : 3, \text{times}=2)</code>	<code>d <- c("a","b","c","d","e")</code>
		<code>rep(1 : 3, \text{each}=2)+2</code>	<code>d[c(TRUE,FALSE)]</code>

EXERCICE N°3

Combien d'objets avez-vous créé dans l'exercice précédent ? Les supprimer à l'aide de la fonction `rm(...)`.

EXERCICE N°4

Uniquement à l'aide des fonctions `" : "`, `seq()` et `rep()`, générer les vecteurs suivants :

1. $1 - 3 - 5 - 7 - 9 - 11$;
2. $1 - 2 - 3 - 4 - 1 - 2 - 3 - 4 - 1 - 2 - 3 - 4$;
3. $10 - 8 - 6 - 4 - 2 - 0$.

EXERCICE N°5

1. Affichez la taille mémoire de des vecteurs `1 : 10` et `seq(1,10,by=1)` à l'aide de la fonction `object.size(...)`.
2. Expliquer cette différence en affichant le type de données des deux vecteurs.

EXERCICE N°6

1. Utiliser la fonction `sys.time()` pour déterminer le temps de chargement du fichier `Emails.csv` à l'aide de la fonction `read.csv()`.
2. Sauvegarder l'objet obtenu dans un fichier `.RData`, le supprimer puis déterminer le temps pour le recharger.
Conclusion!

EXERCICE N°7

1. Créer un vecteur `a` contenant les valeurs suivantes : 2, 4, 5, 5.
2. Afficher successivement les vecteurs `mean(a)`, `a-mean(a)`, $(a-\text{mean}(a))^2$ puis `mean((a-mean(a))^2)`.

3. Afficher le vecteur $\text{mean}(a^2) - \text{mean}(a)^2$.
4. Êtes-vous surpris? Que vient-on de calculer?
5. La fonction $\text{sd}(\dots)$ calcule l'écart type des valeurs d'un vecteur. Afficher $\text{sd}(a)^2$. Pourquoi n'obtient-on pas le même résultat que dans les questions 2 et 3?

EXERCICE N°8

Le fichier TD_temperatures.csv contient les températures journalières minimales à La Rochelle depuis 1979.

1. a) Importer ces données dans un objet que vous appellerez Température à l'aide de la fonction `read.table(...)`. *Attention aux paramètres sep, header, stringsAsFactors.*
b) Quelle est la classe de l'objet Température?
c) Température possède 2 variables. Quels sont les noms de ces variables? Quels sont les types de leurs données?
d) Enregistrer `Température$temp` dans un vecteur nommé `t`.
2. a) `min(t)` donne la valeur minimale contenue dans `t`. Que vaut-elle? Quelle est votre explication?
b) Afficher successivement les vecteurs `t== -9999`, `sum(t== -9999)`. À quoi correspondent ces vecteurs?
c) Que permet de faire l'instruction `t <- t[t != -9999]`?
d) Combien le nouveau vecteur `t` contient-il de valeurs?
e) `t` est un vecteur contenant les températures en dixième de degré. Passer les valeurs en degré.
3. a) D'un point de vue statistique, quelle est la nature de la variable `t`?
b) Rappeler les paramètres et graphiques adaptés à ce type de variable.
c) Essayer les instructions `summary(t)`, `boxplot(t)` et `hist(t)`.
4. En vous inspirant de la méthode vue en dans la question 2 b :
a) Déterminer le nombre et la fréquence de valeurs négatives contenus dans `t`.
b) Déterminer le nombre et la fréquence de valeurs inférieures à la moyenne.

EXERCICE N°9

1. a) Utiliser la fonction `sample()` pour générer un vecteur `de1` contenant les résultats de 1 000 lancers d'un dé.
b) Créer de même un vecteur `de2`.
c) Créer un vecteur `somme` contenant la somme des deux dés.
d) Utiliser la fonction `table()` pour dénombrer les différentes valeurs du vecteur `somme`.
e) Représenter cette table à l'aide de la fonction `plot()`.
f) Quelle est la valeur la plus fréquente? Pourquoi?
2. a) En utilisant une boucle `for`, créer un vecteur contenant un échantillon de taille 1 000 de la somme de 30 dés.
b) Déterminer la moyenne m et l'écart type σ de cet échantillon.
c) Représenter graphiquement les fréquences de la série obtenue.
d) À l'aide de la fonction `curve(...)`, ajouter au graphique la courbe de la fonction de densité de la loi normale d'espérance m et d'écart type σ .
e) Quel théorème vient-on d'illustrer?

Nous verrons plus loin une méthode plus efficace pour générer ce vecteur à l'aide des matrices.

*
* *

 Lire la section du cours concernant les `data.frame`

EXERCICE N°10

Les données ci-dessous sont issues d'une étude menée sur des enfants de 3 ans :

Prénom	Érika	Célia	Éric	Paul	Adan	Louis	Léo	Jean
Sexe	F	F	H	H	H	H	H	H
Poids (en kilogrammes)	16	13	13,5	16,5	17	14,8	16,7	16,5
Taille (en centimètres)	100	97	95,5	95	103	98	100	115

1. Entrer ces données dans un `data.frame` que vous pouvez appeler `tableau`. `stringsAsFactors = FALSE`.
2. a) Écrire une instruction permettant d'afficher la taille d'Éric.
b) Écrire une instruction permettant de n'afficher que les prénoms et les poids des enfants.
c) Écrire une instruction permettant ne n'afficher que les données concernant les filles.
d) Écrire une instruction permettant ne n'afficher que les prénoms et les poids des enfants mesurant plus de 99 cm.
e) Écrire une instruction permettant ne n'afficher que les prénoms des enfants mesurant plus de 95 cm et pesant plus de 15 kg. Combien sont-ils?
f) Écrire une instruction dénombrant les garçons.
g) Jean est vraiment trop grand. Le supprimer du tableau.
3. Rappeler la nature de la variable `Sexe` et quelques graphiques appropriés à la représentation de ce type de variable. Essayer les instructions `table(tableau$Sexe)`, `barplot(table(tableau$Sexe))` et `pie(table(tableau$Sexe))`. *Le paramètre `main` permet d'ajouter un titre...*
4. a) Déterminer le poids moyen et la taille moyenne de ces enfants et plus généralement un résumé de ces deux séries.
b) Déterminer le poids moyen des filles.
5. Représenter à l'aide de la fonction `plot(...)` la taille des enfants en fonction de leur poids. Ne pas oublier le titre et l'intitulé des axes (`main`, `xlab`, `ylab`).
6. Ajouter une colonne indiquant l'IMC (arrondi au dixième) de ces enfants (poids, en kilogrammes, divisé par le carré de la taille, en mètres).
7. À cet âge, un enfant est considéré en surpoids si son IMC dépasse 18 et en insuffisance pondérale si son IMC est inférieur à 14.

On souhaite donc discrétiser notre variable IMC de la façon suivante :

IMC	Corpulence
de 0 à 14	Insuffisance pondérale
de 14 à 18	Poids normal
au dessus de 18	Surpoids

À l'aide de la fonction `cut(...)`, ajouter une colonne donnant la corpulence des enfants.

Un peu d'aide ? Trois vecteurs sont nécessaires :

- la variable à discrétiser,
- un vecteur donnant les intervalles `c(0,14,18,25)`
- un vecteur donnant les modalités à associer à chaque intervalle.

Et `help(cut)` ...

EXERCICE N°11

Installer et charger le package `xlsx` qui permet d'importer (et d'exporter) des données Excel.

L'installation doit se faire lors de la première utilisation du package à l'aide de l'instruction `install.packages(...)` ou directement à l'aide des menus RStudio.

Une fois installé, le package doit être chargé au début de chaque lancement de R à l'aide de l'instruction `library(...)`.

EXERCICE N°12

Le fichier « TD_etablissements_scolaires.xls » contient les effectifs de l'ensemble des établissements du secondaire, privé et public en 2014-2015.

⚠ L'effectif est parfois indiqué « < à 5 ». Cela aura des conséquences lors de l'import des données qu'il faudra gérer.

1. Ouvrir le document avec Excel pour voir sa structure. Remarquer notamment que le tableau de données ne commence qu'à la ligne 3...
La colonne A contient un numéro unique par établissement : le code UAI (Unité Administrative Immatriculée). Les trois premiers caractères de ce code indique le département.
2. a) Utiliser la fonction `read.xlsx(...)` ou `read.xlsx2(...)` du package `xlsx` pour importer ces données sous **R** dans un tableau de données que vous nommerez `Etablissements`.
Un peu d'aide pour les paramètres : `help(read.xlsx)` et toujours `stringsAsFactors...`

- b) Renommer les variables : UAI, Academie, Type, Secteur, Effectif.
 - c) Quelle est le type de la variable Effectifs? Pourquoi **R** a-t-il fait cela?
 - d) Convertir ce vecteur en un vecteur d'entiers.
⚠ Des valeurs NA pour « not available » vont apparaître. Il faudra les gérer par exemple à l'aide du paramètre `na.rm` des fonctions `min`, `max`, `mean`.
3. Dans les manipulations que vous allez effectuer dans les questions suivantes, prenez garde de ne pas effacer votre tableau de données sous peine de devoir le réimporter.
- a) Quel est l'établissement ayant le plus grand nombre d'élèves?
✎ `max(...)` indique le maximum d'un vecteur (attention aux valeurs manquantes) et `which.max(...)` indique l'indice de ce maximum.
 - b) Quelle est la proportion d'établissements privés en France?
 - c) Quelle est cette proportion parmi les lycées?
 - d) Déterminer le nombre total d'élèves dans les lycées publics et dans les lycées privés.
 - e) À l'aide de la fonction `hist(...)`, tracer un histogramme de la variable Effectif.
 - f) Tracer des histogrammes ainsi que les boîtes à moustaches de la variable Effectif pour le privé et pour le public.
✎ `par(mfrow = c(2,1))` permet de tracer les deux graphiques l'un sous l'autre.
4. a) En utilisant la fonction `substr(...)`, créer dans le tableau de données Etablissements une nouvelle variable Num_dep contenant les trois chiffres indiquant le numéro des départements (079 pour les Deux-Sèvres).
- b) Quel est le département ayant le moins d'élèves?
 - c) Quel est le département ayant le moins de collégiens?

EXERCICE N°13

Le fichier Excel « TD_regions_departements_pop.xlsx » contient 3 feuilles :

- La liste des départements avec leur population 2016;
- La liste des départements et leur ancienne région;
- La liste des anciennes régions et des grandes régions.

1. Ouvrir le fichier à l'aide d'Excel pour voir sa structure.
2. Importer les données sous **R** dans trois tableaux de données.
⚠ Le paramètre « `encoding = "UTF-8"` » devra sans doute être ajouté pour gérer les accents.
3. À l'aide de la fonction `merge(...)`, créer un tableau unique contenant le nom des départements, leur population, l'ancienne région et la nouvelle grande région.
4. a) À l'aide de la fonction `aggregate(...)`, créer de même un tableau comportant le nom des nouvelles grandes régions ainsi que leur population totale.
 b) Présenter ces données à l'aide d'un graphique adapté.
✎ Quelques paramètres pourront être utiles : `horiz` pour tracer les barres horizontalement et `las` pour l'orientation des intitulés de l'axe.
- c) Exporter votre graphique, en lignes de commande, au format PDF Cf paragraphe 6.3 du cours.
- d) Exporter le tableau sur une nouvelle feuille (nommée « Population des grandes régions ») du fichier « region_departement_pop.xlsx ».
Indication : `help(write.xlsx)` pour voir les paramètres existants.

EXERCICE N°14

En reprenant les données des deux exercices précédents, créer un graphique présentant les proportions d'élèves par rapport à la population totale dans chaque grande région.

*
* *

EXERCICE N°15

Le package maps permet de réaliser des cartes et de les enrichir avec des informations statistiques.

La fonction principale du package est `map(...)` dont les principaux paramètres sont :

- `database` : la zone que l'on veut représenter ("world", pour représenter l'ensemble des pays, "france", ...)
- `regions` : vecteur indiquant les noms des différentes régions que l'on veut représenter. Ces régions doivent faire partie de la zone choisie.

Le monde est en effet divisé en différentes zones qui sont elles-mêmes divisées en régions. Par exemple, les régions de la zone "france" sont essentiellement ses départements.

`map("france", plot = FALSE)$names` affiche l'ensemble des régions définies pour la France.

- `exact = TRUE` : à mettre (voir l'aide en ligne...)
- `add = TRUE` pour ajouter la carte à la carte existante.
- `fill = TRUE` pour colorier l'intérieur des régions.
- `col` : vecteur indiquant les couleurs de remplissage des régions (si `fill=TRUE`).

Si l'on veut représenter plusieurs régions, les vecteurs `regions` et `col` doivent avoir le même nombre d'éléments pour que les couleurs indiquées correspondent aux régions sélectionnées.

1. Installer et charger le package `maps`.
2. Essayer et comprendre le code ci-dessous.

```
# Dessine la France
> map("france")
# Ajoute les Deux-Sèvres sur la carte existante
> map("france", regions="Deux-Sevres", col="red", fill=T, exact=T, add=T)
# Ajoute un point de longitude et latitude indiquées
> points(-0.46, 46.32, pch=20)
# Ajoute du texte
> text(-0.46, 46.32, "Niort", pos=4, cex=0.6)
# Création d'un tableau de données
> dep <- data.frame(nom=c("Deux-Sevres", "Sarthe"),
                    pref=c("Niort", "Le Mans"),
                    long=c(-0.4606, 0.1969),
                    lat=c(46.326, 48.0041),
                    couleur=c("red", "blue"),
                    stringsAsFactors = F)

> dep
      nom    pref    long    lat couleur
1 Deux-Sevres Niort -0.4606 46.3260    red
2 Sarthe Le Mans  0.1969 48.0041    blue
> map("france")
> map("france", regions=dep$nom, col=dep$couleur, fill=T, exact=T, add=T)
> points(dep$long, dep$lat, pch=20)
> text(dep$long, dep$lat, dep$pref, pos=4, cex=0.6)
> legend("bottomleft", legend=dep$nom, col=dep$couleur, pch=15)
```

3. Les noms des départements utilisés dans le package `maps` ne correspondent pas aux noms officiels utilisés notamment par l'INSEE.

`map("france", plot = FALSE)$names` affiche l'ensemble des régions définies pour la France. Elles sont ordonnées dans l'ordre des latitudes décroissantes.

Importer le fichier « TD_lien_maps_dep.csv » un `data.frame` qui contiendra deux colonnes : les noms officiels des départements et ceux utilisés dans le package `maps`.

EXERCICE N°16

Représenter chacune des nouvelles grandes régions de différentes couleurs. Ajouter une légende. Exporter votre carte au format JPEG.

Je vous invite par exemple à créer un tableau de données contenant les colonnes suivantes :

- Le nom officiel du département ;
- Le nom du département utilisé par le package `maps` ;

- Le nom de la grande région;
- La couleur attribuée à chaque département (cette couleur devant être la même pour les différents départements d'une même région).

 Quelques informations concernant les couleurs :

- `colors()` affiche les noms de l'ensemble des couleurs pré-définies dans **R**;
- La fonction `rgb(...)` permet de créer facilement des dégradés;
- Le package `RColorBrewer` propose différentes palettes de couleurs.

EXERCICE N°17

Le fichier « `sl_cho_2019T1.xls` » contient les taux de chômage par département depuis le premier trimestre 1982.

Réaliser pour chaque trimestre présent dans le fichier une carte colorée indiquant le taux de chômage par département. Exporter chacune de ces cartes au format jpeg afin de réaliser une animation à l'aide d'un logiciel adapté.

Contrôlez vos cartes : <http://www.insee.fr/fr/statistiques/2012804>

*
* *

 Lire la section du cours concernant les listes.

EXERCICE N°18

- Créer une liste que vous appellerez `maliste` dont les éléments sont les 100 vecteurs suivants :

`1 :1; 1 :2; 1 :3; 1 :4; ... ; 1 :100`

- Répondre aux questions suivantes sans l'ordinateur :

- Quelle est la classe de chacun des objets suivants :

`maliste[c(1,2,3)], maliste[1], maliste[[1]]`

- Que renvoient les expressions :

`maliste[[4]], maliste[[50]][20], maliste[[c(50,20)]], maliste[[c(50,60)]]`

- Calculer la somme de chacun des vecteurs de `maliste` à l'aide de la fonction `sapply(...)`.

EXERCICE N°19

- Écrire une fonction permettant de supprimer les chaînes vides d'un vecteur de caractères.

La fonction reçoit par exemple le vecteur `c("a", "b", "", "c", "", "d")` et renvoie le vecteur `c("a", "b", "c", "d")`.

- À l'aide de la fonction `sapply(...)`, supprimer toutes les chaînes vides présentes dans la liste suivante :

`list(a=c("1","2","3"),b=c("", "2","3","4"),c=c("", "", "3","4","5"))`

*
* *

 Effectuer au besoin une petite recherche sur les expressions régulières.

EXERCICE N°20

Le fichier « `TD_Fleurs_du_mal.txt` » contient l'intégral (ou presque) des *Fleurs du mal* de Charles Baudelaire.

Dans ce fichier, les titres des différentes poésies sont écrits en majuscule ce qui va nous aider à les repérer. Par ailleurs, certaines poésies sont composées de plusieurs parties numérotées en chiffres romains.

- Importer le texte à l'aide de la fonction `readLines(...)`. La fonction `readChar(...)` permet également de récupérer l'ensemble du texte au sein d'une unique chaîne. Le paramètre `nchars` peut être fixé à l'aide de l'instruction `file.info("monFichier.txt")$size`

- Utilisation d'expressions régulières**

Les fonctions **R** utilisant les expressions régulières sont : `grep`, `grepl`, `regexpr`, `gregexpr` (localisation de motifs), `sub`, `gsub` (substitution de motifs) et `strsplit` (découpage d'un texte).

- `grep(...)` renvoie un vecteur contenant les indices des éléments contenant un motif satisfaisant à l'expression régulière.
- `gsub(...)` substitue aux motifs satisfaisants à l'expression régulière la chaîne souhaitée.
- `strsplit(...)` découpe une chaîne au niveau des motifs satisfaisants à l'expression régulière. Les motifs sont supprimés.

- Afficher les vers contenant « amour » ou « Amour » (mais pas « AMOUR », ni « amoureuse », ...).
- Afficher les vers contenant soit le mot « amour », soit le mot « enfer ».
- Afficher les vers contenant simultanément « amour » et « enfer ».
- Afficher les vers contenant des mots de 4 lettres commençant par un « r » et se terminant par « e ».
- Afficher les vers se terminant par un point d'interrogation.
- Afficher les vers contenant des mots se terminant par un « s ».

3. Statistiques sur l'ensemble des vers

- Supprimer toutes les lignes vides.
- Utiliser la fonction `gsub(...)` pour supprimer tous les espaces en début de vers.
- En utilisant par exemple la fonction `toupper(...)`, supprimer toutes les lignes en majuscule.
- Combien l'œuvre contient-elle de vers?
- Supprimer la ponctuation.
- Supprimer les espaces multiples entre les mots pour les remplacer par un seul espace.
- Utiliser la fonction `strsplit(...)` pour découper les vers et en récupérer les mots.
Quelle est la classe de l'objet retourné?
- Supprimer au besoin l'ensemble des chaînes vides introduites par `strsplit(...)`. *Normalement, il n'y en a pas...*
- Quel est le nombre minimum, maximum et moyen de mots par vers? Quel est le nombre total de mots?
- Tracer un graphique donnant la fréquence des vers en fonction du nombre de mots qu'ils contiennent.
- Quel est le vers le plus long? Le plus court?
- Quel est nombre total de lettres dans le recueil?

4. Statistiques sur l'ensemble des mots

- Passer l'ensemble du texte en minuscule. *Cette opération aurait pu être effectuée plus tôt.*
- Utiliser la fonction `unlist(...)` pour récupérer un vecteur contenant l'ensemble des mots de l'œuvre.
- Quels sont les dix mots les plus fréquents?
- Utiliser par exemple la fonction `removeWords(...)` du package `tm` (text mining) pour supprimer ces mots inintéressants. `stopwords("french")` donne une liste (un vecteur) déjà complète de mots à supprimer.
- Réaliser un nuage de mots à l'aide du package `wordcloud` ou `wordcloud2`.

5. Statistiques par poème

- Combien le recueil contient-il de poèmes?
Δ Certains poèmes sont composés de plusieurs parties numérotées en chiffres romains. `grep(...)` et une expression régulière adaptée doit permettre de les repérer.
- Étudier la série statistique constituée du nombre de mots par poème.
- Quels sont les poèmes contenant le plus et le moins de mots.

EXERCICE N°21

Vous trouverez sur moodle les déclarations de politique générale prononcées par les différents premiers ministres depuis 1976.

L'objectif de l'exercice est de créer une table de contingence entre les noms des premiers ministres et les mots employés dans leur déclaration.

Cette table de contingence pourra ensuite être utilisée pour réaliser une AFC (analyse factorielle de correspondance).

Quelques précisions et indications :

- La fonction `table(...)` crée une table de contingence à partir d'un tableau de données. Je vous invite donc à créer le tableau de données contenant deux variables : les mots contenus dans les textes et leur auteur.

- Vous veillerez à supprimer les mots peu intéressants (`stopwords("french")`) et à ne conserver que les mots apparaissant plus de 150 fois dans l'ensemble du corpus.
- Pour réaliser l'AFC, vous pouvez utiliser la fonction `CA(...)` du package `FactoMineR`. Le package `factoextra` propose des visualisations plus avancées.
- Essayez d'éviter les boucles en ayant recours le plus souvent possible à la fonction `sapply(...)`.
- Vous réaliserez un programme complètement automatisé : il devra fonctionner avec un autre corpus. La fonction `dir(...)` permet par exemple de lister l'ensemble des fichiers contenus dans un répertoire. Notez que la valeur de 150 évoquée plus haut n'est alors pas toujours pertinente.
- Écrire alors une fonction prenant en entrée le chemin contenant le corpus ainsi que le nombre de minimum de mots souhaités et réalisant en sortie l'AFC.

EXERCICE N°22

L'algorithme `Word2Vec` développé par des ingénieurs de Google permet de :

- Prédire un contexte à partir d'un mot;
- Prédire un mot à partir d'un contexte.

L'algorithme repose sur des réseaux de neurones et nécessite donc un apprentissage.

📖 Plus d'informations : <https://dataanalyticspost.com/Lexique/word2vec/>

La librairie `wordVectors` implémente l'algorithme sous **R**. Essayons-la sur une des déclarations de politique générale.

1. L'installation du package nécessite `devtools` :

```
> library(devtools)
> install_github("bmschmidt/wordVectors")
> library(wordVectors)
```

2. La première étape consiste à nettoyer le fichier qui servira ensuite pour l'entraînement :

```
> prep_word2vec(origin = "edouard_philippe.txt",
               destination = "edouard_philippe_propre.txt",
               lowercase = T, bundle_ngrams = 2)
```

La fonction `prep_word2vec(...)` prend en entrée un fichier texte et crée un nouveau fichier dépouillé de toute ponctuation, saut à la ligne, espace superflu,... Le paramètre `bundle_ngrams = 2` permet de repérer des couples de mots à ne pas dissocier (en deçà, parce que,...).

Ouvrez le fichier « `edouard_philippe_propre.txt` » pour voir le résultat. De l'aveu même de l'auteur, cette fonction est « extraordinairement inefficace » ! Mais vous savez faire mieux, à l'aide des expressions régulières. Les fonctions `readChar(...)` et `writeChar(...)` peuvent alors être utiles.

```
> readChar("edouard_philippe.txt",
          nchars = file.info("edouard_philippe.txt")$size)
> ...
> writeChar(...,"edouard_philippe_propre.txt")
```

3. L'étape suivante consiste à utiliser ce fichier propre pour entraîner le réseau de neurones :

```
> modele <- train_word2vec(train_file = "edouard_philippe_propre.txt",
                          output_file = "modele.bin")
> unlink("edouard_philippe_propre.txt")
> unlink("modele.bin")
```

Le modèle est enregistré dans le fichier « `modele.bin` » et peut être rechargé (grâce à la fonction `read.vectors(...)`) ou supprimé (`unlink(...)`).

4. Le modèle peut maintenant être utilisé pour prédire un contexte lié à un mot :

```
> closest_to(modele,"projet")
```

Ou pour prédire un mot à partir d'un contexte :

```
> closest_to(modele,c("dialogue","réforme"))
```

📖 Aller plus loin : <https://github.com/bmschmidt/wordVectors/blob/master/vignettes/introduction.Rmd>

Effectuer au besoin des recherches sur le format de document XML et le langage XPath

<https://www.w3schools.com/xml/default.asp>

EXERCICE N°23

On s'intéresse dans cet exercice aux prix des carburants en France.

1. a) Rendez-vous sur le site <https://www.prix-carburants.gouv.fr/rubrique/opendata/> et télécharger les données du 30 novembre 2021. L'arborescence du fichier XML (les nœuds) et le sens des différents attributs y sont indiqués.
- b) Ouvrir le document avec le bloc note pour en comprendre la structure puis avec Excel pour comprendre l'intérêt du format XML par rapport au format CSV pour ce jeu de données.
- c) Importer les données dans R à l'aide de la fonction `xmlParse(...)` du package XML. *Utiliser l'encodage Latin-1 semble-t-il.*

Dans la suite, j'ai nommé mon objet « carburants ».

2. La fonction `getNodeSet(...)` permet d'obtenir la liste des noeuds d'un fichier XML correspondant au chemin XPath spécifié.

On peut alors appliquer une fonction aux différents éléments de la liste grâce à `sapply(...)`.

Le package XML fournit trois fonctions qui pourront être utiles :

- `xmlAttrs(...)` récupère au sein d'une matrice l'ensemble des attributs du nœud racine.
- `xmlGetAttr(...)` récupère au sein d'un vecteur l'attribut spécifié dans le nœud racine.
- `xmlValue(...)` récupère au sein d'un vecteur tous les nœuds « texte » contenu dans le nœud racine.

Essayer ces quelques exemples :

- a) Récupération de l'ensemble des villes :

```
> getNodeSet(carburants, "/pdv_liste/pdv/ville")
> sapply(getNodeSet(carburants, "/pdv_liste/pdv/ville"), xmlValue)
```

- b) Récupération des prix du gazole :

```
> getNodeSet(carburants, "//pdv/prix[@nom='Gazole'][position()=1]")
> sapply(getNodeSet(carburants, "//pdv/prix[@nom='Gazole'][position()=1]"),
  xmlGetAttr, "valeur")
```

⚠ Le fichier comporte des doublons (anormaux me semble-t-il). D'où l'argument « `position()=1` ». Essayer sans pour s'en convaincre. À retenir pour la suite...

- c) Récupération des identifiants des stations disposant d'une boutique alimentaire :


```
> getNodeSet(carburants, "/pdv_liste/pdv[services/service='Boutique alimentaire']")
> sapply(getNodeSet(carburants,
  "/pdv_liste/pdv[services/service='Boutique alimentaire']"),
  xmlGetAttr, "id")
```

- d) Récupération du prix du gazole dans les Deux-Sèvres :


```
> getNodeSet(carburants,
  "/pdv_liste/pdv[contains('79', substring(@cp, 1, 2))]/prix[@nom='Gazole']")
> sapply(getNodeSet(carburants,
  "/pdv_liste/pdv[contains('79', substring(@cp, 1, 2))]/prix[@nom='Gazole']"),
  xmlGetAttr, "valeur")
```

- e) Récupération des attributs concernant les deux stations proposant plus de 22 services :

```
> getNodeSet(carburants, "/pdv_liste/pdv[services[count(service)>22]]")
> sapply(getNodeSet(carburants, "/pdv_liste/pdv[services[count(service)>22]]"),
  xmlAttrs)
```

 Notons l'existence de la fonction `xpathSApply(...)` qui conjugue l'utilisation des fonctions `getNodeSet(...)` et `sapply(...)`.

3. Comparer à l'aide de boîtes à moustaches les prix du gazole sur route et sur autoroute.


 Je vous invite pour cela à créer un `data.frame` contenant deux colonnes : le prix et le type de voie.

4. Proposer une carte de l'ensemble des stations délivrant du gazole, colorée en fonction du prix proposé (par exemple par décile).


Voici un code utilisant le package `leaflet` permettant d'afficher la carte de France et des points :


```
> leaflet() %>%
>   addTiles() %>%
>   setView(lng = 2.4302778, lat = 46.5397222, zoom = 6) %>%
>   addCircles(lng,lat)
```

5. Positionner les stations services se trouvant à moins de 10 kilomètres (à vol d'oiseau) de l'espace NiortTech (coordonnées GPS : lng = -0.458698, lat = 46.32088).

 La distance à vol d'oiseau se calcule à l'aide de la formule suivante (coordonnées en radians) :


$$\text{acos}(\sin(\text{lat1}) * \sin(\text{lat2}) + \cos(\text{lat1}) * \cos(\text{lat2}) * \cos(\text{lng2} - \text{lng1})) \times 6371$$

6. Déterminer le département dans lequel le prix moyen du gazole sur route est le moins cher.
7. Créer un vecteur contenant toutes les adresses des stations niortaises.  Attention à la casse.
8. Tracer une courbe donnant l'évolution du prix moyen du gazole dans les Deux-Sèvres sur les trente derniers jours.

 Vous aurez besoin pour cela d'automatiser l'importation des fichiers XML. Voici une solution :

```
# Téléchargement dans un fichier "temp"
> download.file("https://donnees.roulez-eco.fr/opendata/jour/20211130","temp")
# Sur PC, ajouter le paramètre mode = "wb" pour éviter un problème de décompression
# download.file("https://donnees.roulez-eco.fr/opendata/jour/20211130","temp",mode="wb")
# Décompression du fichier
> fichier <- unzip(temp)
> carburants <- xmlParse(fichier,encoding="Latin-1")
# suppression des fichiers temp et fichier
> unlink("temp")
> unlink(fichier)
# suppression de l'objet fichier
> rm(fichier)
```

9. Un automobiliste, garé sur la Brèche, dispose de suffisamment de gazole pour parcourir 20 kilomètres. Tracer la route lui permettant de gagner la station la plus éloignée possible.

 Je vous invite à remarquer que notre automobiliste n'ira pas à plus de 20 kilomètres à la ronde et à utiliser la librairie `osrmr`. Voici par exemple un code permettant d'afficher la distance et le trajet entre Niort et Poitiers :

```
> trajet <- viaroute(lat1 = 46.323716,lng1 = -0.464777,lat2 = 46.580224,lng2 = 0.340375,
                    instructions = T,
                    api_version = 5,
                    localhost = FALSE,
                    timeout = 0.001)
> trajet$routes[[1]]$distance/1000
> route <- decode_geom(trajet$routes[[1]]$geometry, precision = 5)
> leaflet() %>%
  addTiles() %>%
  setView(lng = -0.458698, lat = 46.32088, zoom = 8) %>%
  addPolylines(route$lng,route$lat)
```

Ce programme peut être l'occasion d'utiliser la fonction `tryCatch(...)` pour en renforcer la robustesse.

*
* *