RÉPUBLIQUE DU SÉNÉGAL

Un Peuple - Un But - Une Foi

Ministère de l'Économie, du Plan et de la Coopération

RÉPUBLIQUE DU SÉNÉGAL
Un Peuple - Un But - Une Foi
MINISTÈRE DE L'ÉCONOMIE
DU PLAN ET DE LA COOPÉRATION

Agence Nationale de la Statistique et de la Démographie (ANSD)

École Nationale de la Statistique et de l'Analyse Économique Pierre Ndiaye (ENSAE)

PROJET STATISTIQUES SOUS R

TP10 : Traitement des questions ouvertes avec R : Text mining

---

Rédigé par :
Mame Balla BOUSSO
Paul BALAFAI
Elèves ingénieurs statisticiens économistes

Sous la supervision de :
M. Aboubacar HEMA
ANALYSTE DE RECHERCHE CHEZ IFPRI

Année scolaire : 2024/2025

/newpage

1. Importation et Nettoyage des Données

Chargement des packages

```
library(readxl)        # Pour lire les fichiers Excel
library(topicmodels)   # Pour la modélisation thématique
library(ggplot2)       # Pour les visualisations
library(dplyr)         # Pour la manipulation de données
library(tidytext)      # Pour le traitement de texte
library(tidyr)         # Pour la gestion des données
library(wordcloud)     # Pour les nuages de mots
library(tidyverse)     # Collection de packages pour la science des données
library(tm)            # Pour le text mining
library(SnowballC)     # Pour le stemming
library(stringr)       # Pour la manipulation de strings
library(udpipe)        # Pour la lemmatisation.
```

Importation des données

```
Enquête_dopinion_relative_à_la_journée_dintégration_ <- read_excel("Data/Enquête_do
Texte_JI <- Enquête_dopinion_relative_à_la_journée_dintégration_
head(Enquête_dopinion_relative_à_la_journée_dintégration_)
```

```
## # A tibble: 6 x 4
##      id `Classe de l'étudiant :` `Nationalité de l'étudiant` Texte
##   <dbl> <chr>                    <chr>                       <chr>
## 1     1 AS2                      Congo                       <NA>
## 2     2 ISEP1                 Cameroun                Donner à temps le ~
## 3     3 AS2                      Congo                       <NA>
## 4     4 AS1                   Sénégal                 Faire des sketchs ~
## 5     5 AS1                   Cameroun                Intégrer un court-~
## 6     6 ISE1 Eco                 Sénégal                     <NA>
```

```
colnames(Texte_JI)
```

Vérification des colonnes

```
## [1] "id"                          "Classe de l'étudiant :"
## [3] "Nationalité de l'étudiant" "Texte"
```

3. Filtrage des données

Identification des textes vides

```
text_id_empty <- Texte_JI %>%
  group_by(id) %>%
  summarise(nb_mots = sum(!is.na(Texte))) %>%
  filter(nb_mots == 0) %>%
  pull(id)

Texte_JI_filtered <- Texte_JI %>%
  filter(!(id %in% text_id_empty))

head(Texte_JI_filtered)
```

```
## # A tibble: 6 x 4
##      id `Classe de l'étudiant :` `Nationalité de l'étudiant` Texte
##   <dbl> <chr>                    <chr>                       <chr>
## 1    2 ISEP1                  Cameroun                 Donner à temps le ~
## 2    4 AS1                    Sénégal                  Faire des sketchs ~
## 3    5 AS1                    Cameroun                 Intégrer un court-~
## 4    7 ISE2                   Cameroun                 Faire des sketch c~
## 5    9 ISEP2                  Cameroun                 Améliorer le son p~
## 6   11 ISE2                   Togo                     Une meilleure sono~
```

```
nrow(Texte_JI_filtered)
```

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

On remarque que le nombre de ligne a diminué passant de 128 à 45. Seulement 45 lignes contient des textes.

Nettoyage des textes

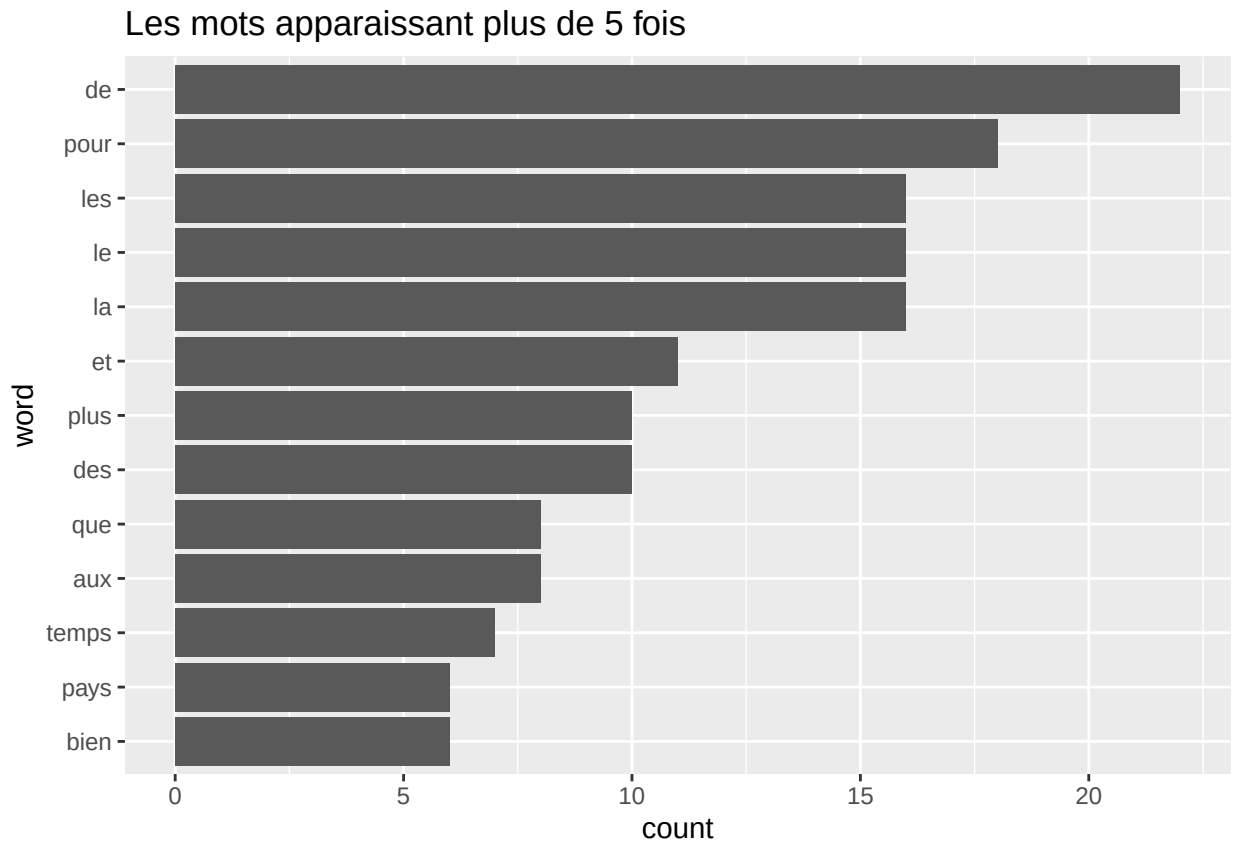
On crée une fonction pour traiter les textes afin de faciliter leur analyse

```
clean_text <- function(text) {
  text <- tolower(text)             # Conversion en minuscules
  text <- removePunctuation(text)  # Suppression de la ponctuation
  text <- removeNumbers(text)      # Suppression des chiffres
  text <- stripWhitespace(text)    # Suppression des espaces superflus
  return(text)
}

data <- Texte_JI_filtered %>%
  mutate(Texte_corrige = sapply(Texte, clean_text))
```

2. Exploration et Prétraitement Textuel

Dans le processus de prétraitement des données, on va tokeniser la base de données pour analyser non pas les textes, mais les mots directement.

```
tokenized_textes <- data %>%
  select(id, Texte_corrige) %>%
  unnest_tokens(input = 'Texte_corrige', output = 'word')


# Visualisation Tokenisation

tokenized_textes %>%
  count(word, sort = TRUE) %>%
  rename(count = n) %>%
  filter(count > 5) %>%
  mutate(word = reorder(word, count)) %>%
  ggplot(aes(x = count, y = word)) +
  geom_col()  +
  labs(title = "Les mots apparaissant plus de 5 fois") +
  scale_x_continuous(breaks = seq(0, 50, 5))
```




```
# Nombre total de lignes après tokenisation
nrow(tokenized_textes)
```

```
## [1] 511
```

Comme nous pouvons le voir sur le graphique ci-dessus, de nombreux mots présents n'apportent aucune réelle valeur à notre analyse. Des mots comme "de", "pour", "les", "le", "la" sont ce qu'on appelle des mots vides (stop words).

Nous allons supprimer ces mots en utilisant la commande anti_join(stop_words).

Charger les stop words en français

```
stop_words_fr <- tibble(word = stopwords("fr"))
head(stop_words_fr)
```

```
## # A tibble: 6 x 1
##   word
##   <chr>
## 1 au
## 2 aux
## 3 avec
## 4 ce
## 5 ces
## 6 dans
```

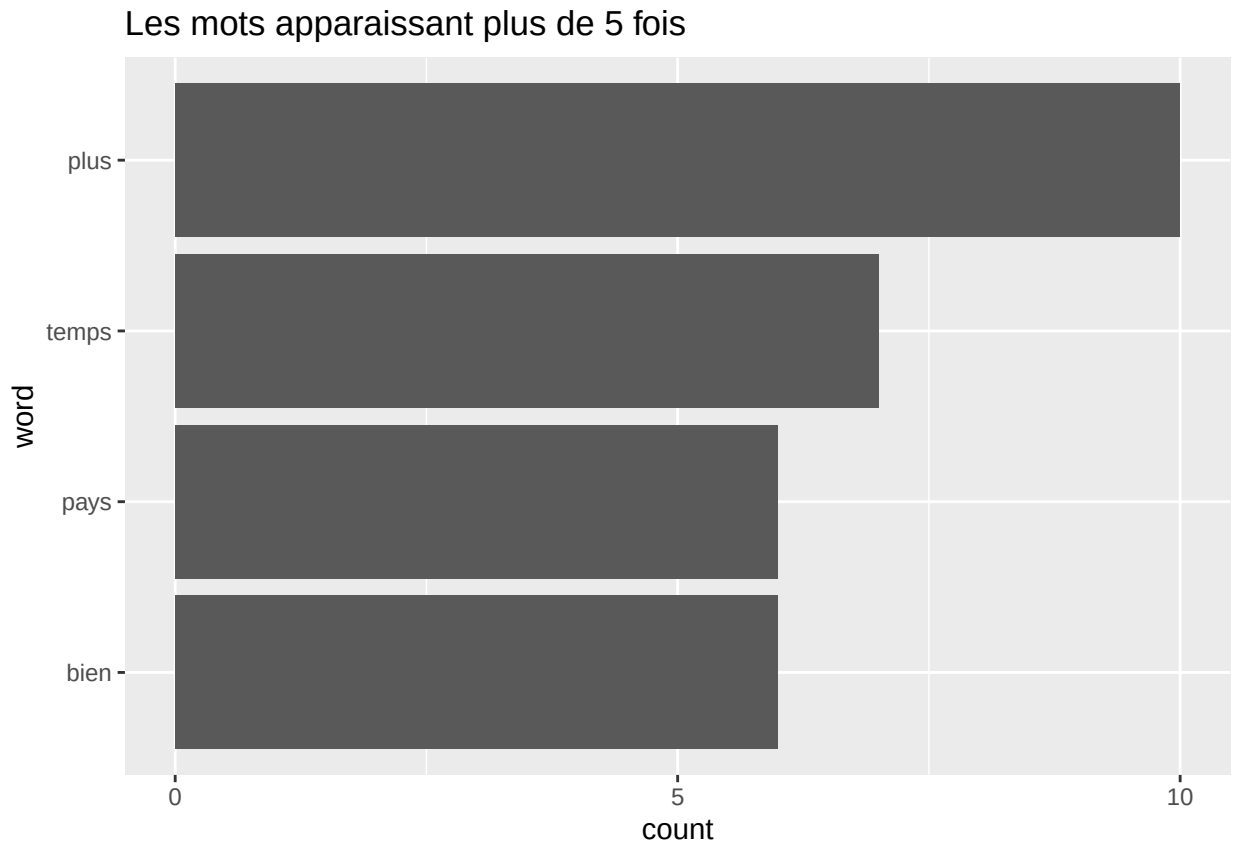
```
Stop_texte <- tokenized_textes %>%
  anti_join(stop_words_fr, by = "word")

nrow(Stop_texte)
```

```
## [1] 317
```

On voit bien que le nombre de mots diminue suite à la supression des stop word

Comme vous pouvez le voir sur le graphique ci-dessous, il reste moins de mots, mais ils sont beaucoup plus pertinents pour l'analyse.

```
Stop_texte %>%
  anti_join(stop_words_fr) %>% #finds where tweet words overlap with predefined sto
  count(word, sort = TRUE) %>%
  rename(count = n) %>%
  filter(count > 5) %>%
  mutate(word = reorder(word, count)) %>%
  ggplot(aes(x = count, y = word)) +
  geom_col() +
  labs(title = "Les mots apparaissant plus de 5 fois") +
  scale_x_continuous(breaks = seq(0, 50, 5))
```




```
library(ggwordcloud) # Une autre manière de visualiser

Stop_texte %>%
  anti_join(stop_words_fr) %>%
  count(word, sort = TRUE) %>%
  filter(n > 4) %>%
  ggplot(aes(label = word, size = n, color = n)) +
  geom_text_wordcloud() +
  scale_size_area(max_size = 15)
```

On peut également voir ci-dessous, la fréquence standard des termes (TF) pour tous les mots

```
Stop_texte %>%
  count(word, sort = TRUE) %>%
  rename(count = n) %>%
  mutate(total=sum(count))%>%
  mutate(tf=count/total) %>%
  head()
```

```
## # A tibble: 6 x 4
##   word   count total     tf
##   <chr>  <int> <int>  <dbl>
## 1 plus      10   317 0.0315
## 2 temps      7   317 0.0221
## 3 bien       6   317 0.0189
## 4 pays       6   317 0.0189
## 5 donner     5   317 0.0158
## 6 faire      5   317 0.0158
```

L'application des stop word diminue le nombre de mots. Ceci le montre

```
Stop_texte %>%
  count(word, sort = TRUE)%>%
  nrow()
```

```
## [1] 215
```

Racinisation

En racinisant, les mots *cultures* et *culture* par exemple se réduisent en *culture*. Voilà pourquoi le nombre total de mots diminue comme le résultat de cette commande

```
# stemming
Stop_texte = Stop_texte %>%
  mutate(stem = wordStem(word))

# unique count of words after stemming
Stop_texte %>%
  count(stem, sort = TRUE) %>%
  nrow()
```

```
## [1] 201
```

Fréquence des mots avant le stemming

```
# Fréquence des mots avant le stemming
Stop_texte %>%
  count(word) %>%
  arrange(desc(n)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    word          n
##    <chr>     <int>
##  1 plus         10
##  2 temps         7
##  3 bien          6
##  4 pays          6
##  5 donner        5
##  6 faire         5
##  7 organiser     5
##  8 tout          5
##  9 chaque        4
## 10 culture       4
```

Fréquence des mots après le stmming

```
# Fréquence des mots après le stemming
Stop_texte %>%
  count(stem) %>%
  arrange(desc(n)) %>%
  head(10)
```

```
## # A tibble: 10 x 2
##    stem          n
##    <chr>     <int>
##  1 plu          10
##  2 cultur        8
##  3 temp          7
##  4 bien          6
##  5 organis       6
##  6 pai           6
##  7 culturel      5
##  8 donner        5
```

```
##  9 fair          5
## 10 journé        5
```

On remarque que le stemming ne semble pas respecter la logique de certains mots. En effet, la racinisation supprime les lettres *s* à la fin des mots comme *plus* et *temps*. Egalement le mot *paix* est réduit à *pai*. C'est pourquoi dans ce qui suit, nous ferons fi de cette racinisation en utilisanat désormais seulement mes textes issus de l'application des stop word.

les analyse TF-IDF

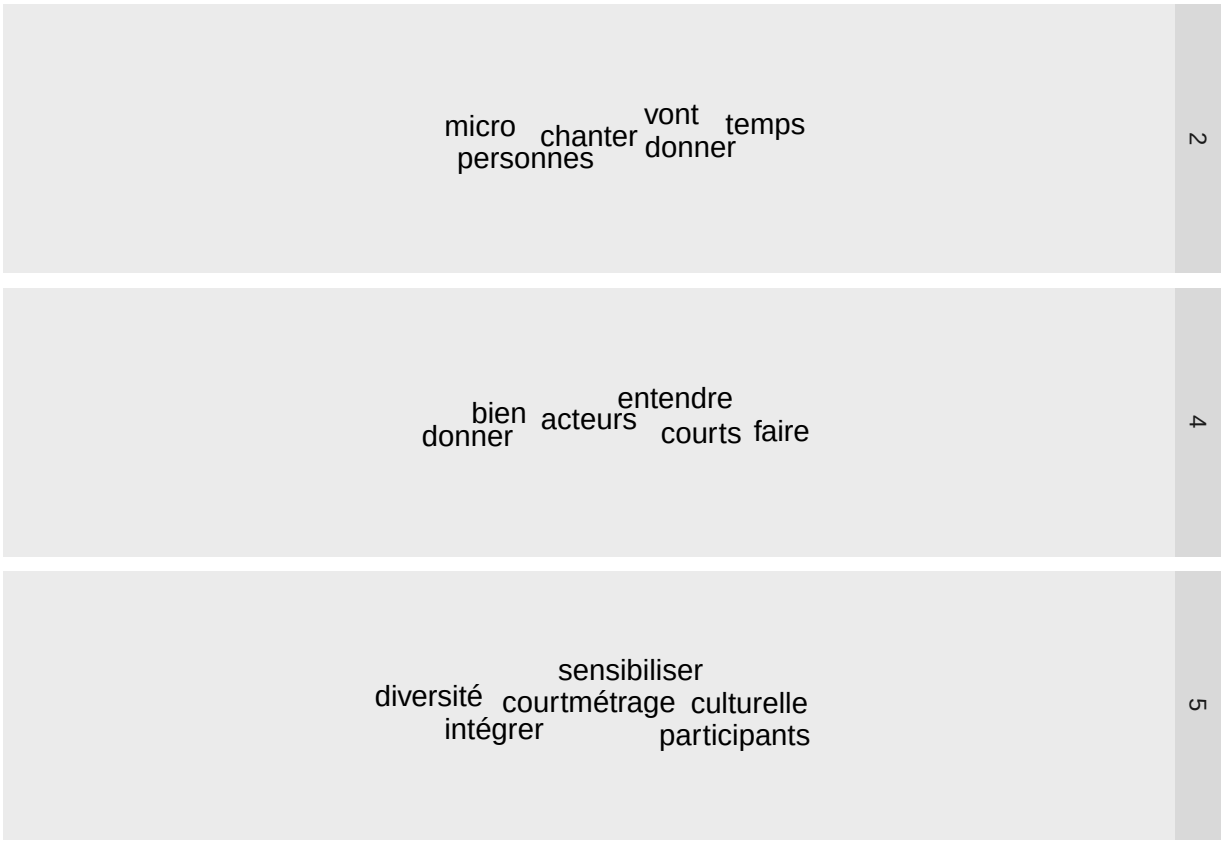
Ci-dessous, nous voyons l'intégralité du tableau TF-IDF. Ce qui nous intéresse le plus, c'est la colonne tf_idf, car elle nous donne le classement pondéré ou l'importance des mots dans notre texte.

```
texte_tf_idf <- Stop_texte %>%
  count(word, id, sort = TRUE) %>%
  rename(count = n) %>%
  bind_tf_idf(word, id, count)
head(texte_tf_idf)
```

```
## # A tibble: 6 x 6
##   word          id count    tf   idf tf_idf
##   <chr>      <dbl> <int> <dbl> <dbl>  <dbl>
## 1 bien          65     2 0.25   2.20  0.549
## 2 cultures     117     2 0.105  2.71  0.285
## 3 dominante    102     2 0.25   3.81  0.952
## 4 pays          75     2 0.143  2.20  0.314
## 5 tout          32     2 0.222  2.42  0.538
## 6 a             39     1 0.2    3.81  0.761
```

Les simples décomptes de fréquences de mots peuvent être trompeurs et peu utiles pour bien comprendre nos données. Il est en fait intéressant les mots les plus fréquents dans chaque texte.

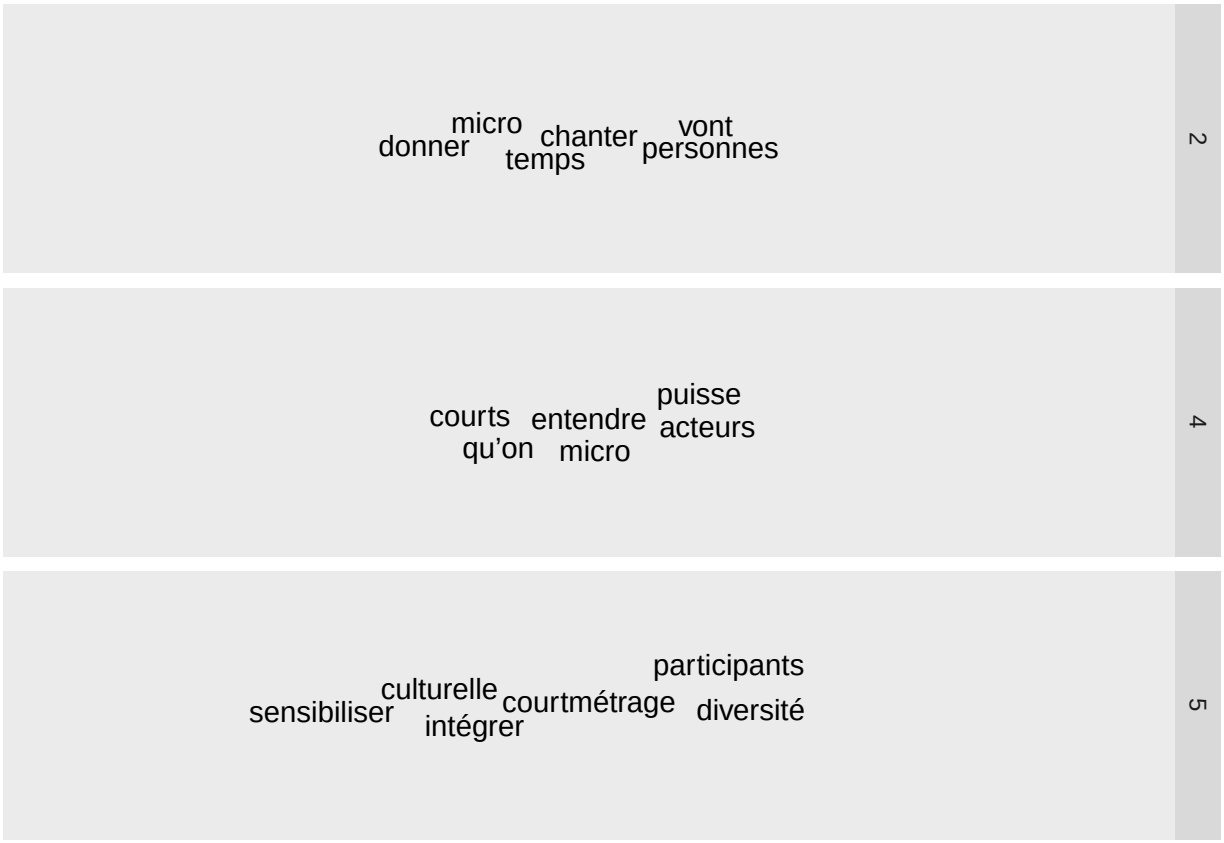
```
texte_tf_idf %>%
  select(word, id, tf_idf, count) %>%
  group_by(id) %>%
  slice_max(order_by = count, n = 6, with_ties=FALSE) %>% #takes top 5 words from e
  filter(id < 6) %>% #just look at 5 textes
  ggplot(aes(label = word)) +
  geom_text_wordcloud() +
  facet_grid(rows = vars(id))
```



On s'est limité au cinq premiers textes. Mais les textes correspondant aux identifiants 1 et 3 sont des NA et donc ont été isolés.

Par ailleurs le résultat qui suit montre aussi qu'il ne faut pas se limiter à un simple dénombrement des textes mais à leur fréquence.

```
texte_tf_idf %>%
  select(word, id, tf_idf) %>%
  group_by(id) %>%
  slice_max(order_by = tf_idf,n = 6, with_ties=FALSE) %>% #takes top 5 words from e
  filter(id < 6) %>% #just look at 5 tweets
  ggplot(aes(label = word)) +
  geom_text_wordcloud() +
  facet_grid(rows = vars(id))
```



Relations entre les mots

Jusqu'à présent, nous avons seulement examiné les mots individuellement. Mais que faire si nous voulons connaître les relations entre les mots dans un texte ? Cela peut être accompli grâce aux n-grammes, où n est un nombre. Auparavant, nous avions effectué une tokenisation mot par mot, mais nous pouvons aussi tokeniser par groupes de n mots. Créons maintenant des bigrams (groupes de deux mots) à partir de tous les tweets, puis comptons-les et trions-les.

```
textes_bigram <- data %>%
  select(id, Texte_corrige) %>%
  unnest_tokens(bigram, Texte_corrige, token = 'ngrams', n = 2)
head(textes_bigram)
```

```
## # A tibble: 6 x 2
##      id bigram
##   <dbl> <chr>
## 1     2 donner à
## 2     2 à temps
## 3     2 temps le
## 4     2 le micro
## 5     2 micro aux
## 6     2 aux personnes
```

Comme vous pouvez le voir dans le dataframe ci-dessus, certains bigrammes contiennent des mots vides (stop words) qui n'apportent pas beaucoup de valeur. Supprimons ces mots vides. Pour cela, nous allons d'abord séparer la colonne des bigrammes en deux colonnes distinctes nommées 'word1' et 'word2'. Ensuite, nous utilis-

erons deux fonctions de filtre pour supprimer les mots vides.

```
textes_bigram <- textes_bigram %>%
  separate(bigram, c("word1", "word2"), sep = " ") %>%#separates on whitespace
  filter(!word1 %in% stop_words_fr$word) %>%
  filter(!word2 %in% stop_words_fr$word)

head(textes_bigram)
```

```
## # A tibble: 6 x 3
##      id word1     word2
##   <dbl> <chr>     <chr>
## 1     2 vont      chanter
## 2     4 sketchs   courts
## 3     4 bien      donner
## 4     4 acteurs   qu'on
## 5     4 qu'on     puisse
## 6     5 diversité culturelle
```

On peut maintenant compter les bigram et voir le résultat

```
bigram_counts <- textes_bigram %>%
  count(word1, word2, sort = TRUE)
head(bigram_counts)
```

```
## # A tibble: 6 x 3
##   word1       word2         n
##   <chr>       <chr>     <int>
## 1 chaque      pays          3
## 2 différentes cultures      2
## 3 donner      plus          2
## 4 faut        réduire       2
## 5 plus        grande        2
## 6 soirée      dansante      2
```

Comme précédemment, on peut aussi créer une mesure TF-IDF avec des n-grammes. Faisons-le maintenant.

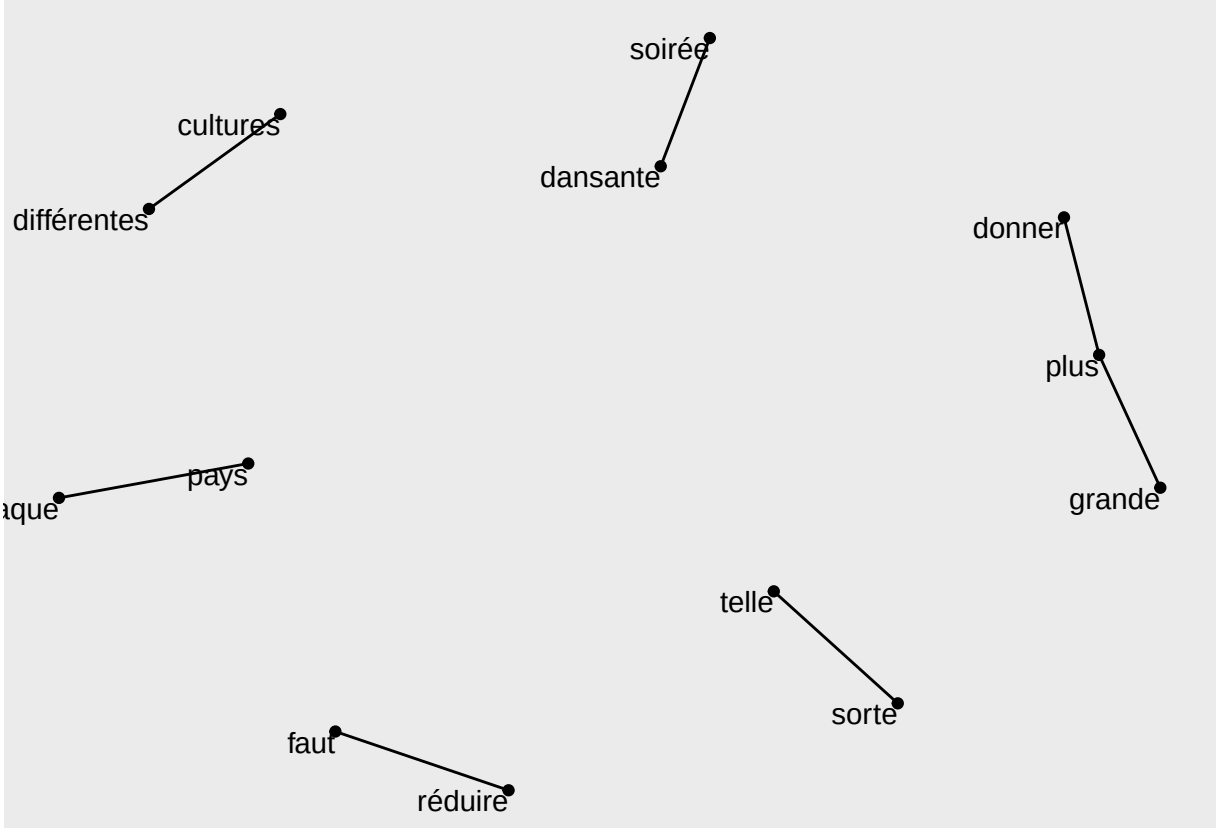
```
data %>%
  select(id, Texte_corrige) %>%
  unnest_tokens(bigram, Texte_corrige, token = 'ngrams', n = 2) %>%
  count(id, bigram) %>%
  bind_tf_idf(bigram, id, n) %>%
  group_by(id) %>%
  arrange(id, desc(tf_idf)) %>%
  head()
```

```
## # A tibble: 6 x 6
## # Groups:   id [1]
##      id bigram            n    tf   idf tf_idf
##   <dbl> <chr>         <int> <dbl> <dbl>  <dbl>
## 1     2 aux personnes     1 0.111  3.81  0.423
## 2     2 donner à          1 0.111  3.81  0.423
## 3     2 temps le          1 0.111  3.81  0.423
## 4     2 vont chanter      1 0.111  3.81  0.423
## 5     2 à temps           1 0.111  3.81  0.423
## 6     2 personnes qui     1 0.111  3.11  0.346
```

Comme on peut le voir ci-dessus, beaucoup de valeurs TF-IDF sont identiques. Cela est en partie dû à la petite

taille des textes. Jetons maintenant un coup d'œil visuel aux relations entre les mots dans tous les textes, en utilisant un graphe en réseau.

```
library('igraph')
library('ggraph')
bi_graph <- bigram_counts %>%
  filter(n > 1) %>%
  graph_from_data_frame()

ggraph(bi_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```



Comme on peut le voir ci-dessus, de nombreux noms et d'autres informations ont été extraits des données.

```
texte_trigram <- data %>%
  select(id, Texte_corrige) %>%
  unnest_tokens(trigram, Texte_corrige, token = 'ngrams', n = 3) %>%
  separate(trigram, c("word1", "word2", "word3"), sep = " ") %>% #separates on whi
  filter(!word1 %in% stop_words_fr$word) %>%
  filter(!word2 %in% stop_words_fr$word) %>%
  filter(!word3 %in% stop_words_fr$word)

head(texte_trigram)
```

```
## # A tibble: 6 x 4
##      id word1        word2          word3
```

```
##   <dbl> <chr>        <chr>          <chr>
## 1     4 acteurs      qu'on          puisse
## 2     7 scène        car            habituellement
## 3    11 meilleure    sonorisation   côté
## 4    11 sonorisation côté           technique
## 5    14 dautres      journées       continuellement
## 6    14 favoriser    lapprentissage culturel
```

On peut aussi compter les trigram et voir le résultat

```
trigram_counts <- texte_trigram %>%
  count(word1, word2, word3, sort = TRUE)
head(trigram_counts)
```

```
## # A tibble: 6 x 4
##   word1    word2     word3            n
##   <chr>    <chr>     <chr>        <int>
## 1 acteurs  qu'on     puisse           1
## 2 ajouter  plus      d'activités      1
## 3 aménager plus      despace          1
## 4 bien     organiser l'événement      1
## 5 car      c'est     souvent          1
## 6 certains modules   statistiques     1
```

Comme précédemment, on peut aussi créer une mesure TF-IDF avec des trigrammes. Faisons-le maintenant.

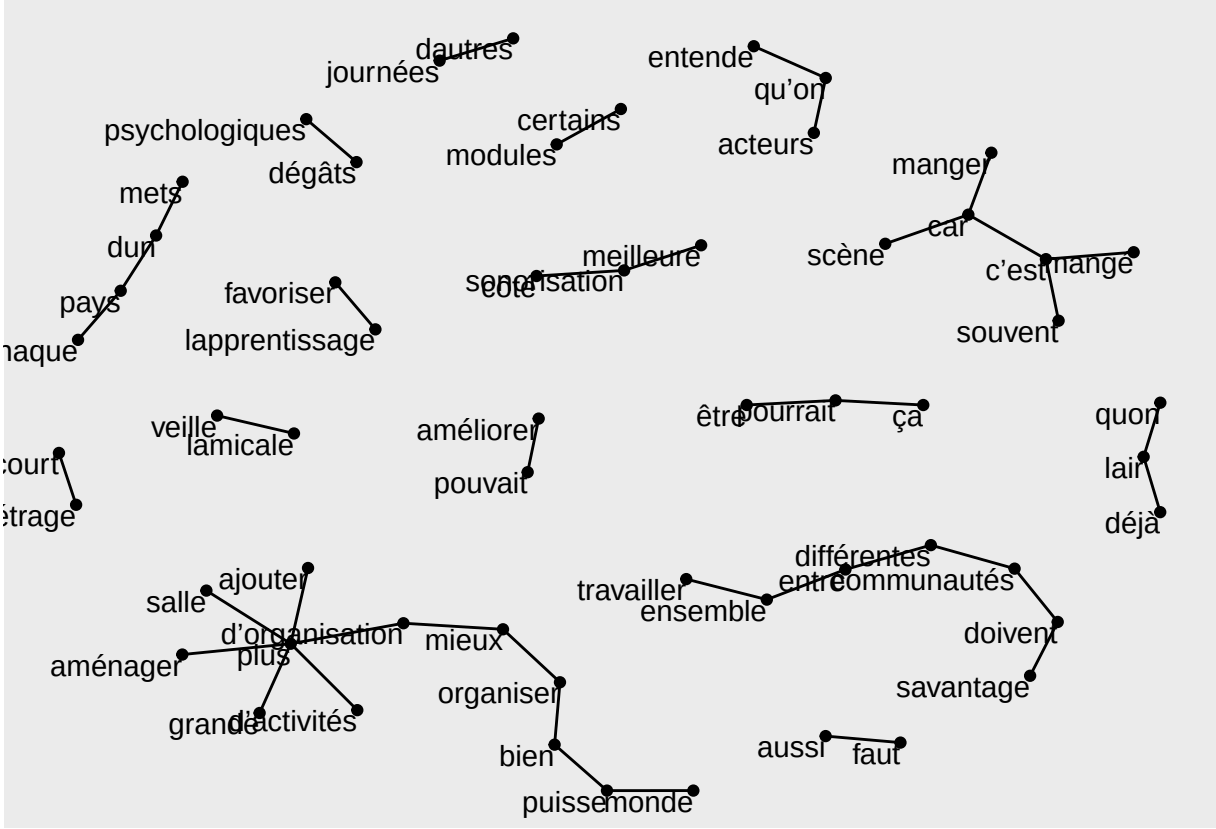
```
data %>%
  select(id, Texte_corrige) %>%
  unnest_tokens(trigram, Texte_corrige, token = 'ngrams', n = 3) %>%
  count(id, trigram) %>%
  bind_tf_idf(trigram, id, n) %>%
  group_by(id) %>%
  arrange(id, desc(tf_idf)) %>%
  head()
```

```
## # A tibble: 6 x 6
## # Groups:   id [1]
##      id trigram                 n    tf   idf tf_idf
##   <dbl> <chr>               <int> <dbl> <dbl>  <dbl>
## 1     2 aux personnes qui       1 0.125  3.81  0.476
## 2     2 donner à temps          1 0.125  3.81  0.476
## 3     2 micro aux personnes     1 0.125  3.81  0.476
## 4     2 personnes qui vont      1 0.125  3.81  0.476
## 5     2 qui vont chanter        1 0.125  3.81  0.476
## 6     2 temps le micro          1 0.125  3.81  0.476
```

Comme on peut le voir ci-dessus, beaucoup de valeurs TF-IDF sont identiques. Cela est en partie dû à la petite taille des textes comme remarqué dans le cas bigram. Jetons maintenant un coup d'œil visuel aux relations entre les mots dans l'ensemble des textes, en utilisant un graphe en réseau.

```
tri_graph <- trigram_counts %>%
  filter(n > 0) %>% # Ici, on garde TOUS les trigrammes présents au moins une fois
  graph_from_data_frame()

ggraph(tri_graph, layout = "fr") +
  geom_edge_link() +
  geom_node_point() +
```

```
  geom_node_text(aes(label = name), vjust = 1, hjust = 1)
```



Normalement, on mettrait n > 1 ou n > 2 pour filtrer les trigrammes peu fréquents. MAIS : dans notre cas, les tweets sont très courts, donc les trigrammes se répètent très peu. Résultat : quasiment aucun trigramme n'apparaît plus d'une fois. Du coup, si on filtre avec n > 1 ou n > 2 → le graphe devient vide (aucun trigramme à afficher).

En mettant n > 0, on garde tous les trigrammes possibles, même ceux présents une seule fois. Cela permet d'obtenir un graphe, même si les connexions sont faibles (juste 1 apparition).

Dans notre base de données, le champ contenant les suggestions n'est pas obligatoire, ce qui signifie que plusieurs enregistrements présentent des valeurs manquantes (NA). Lors de l'application initiale du modèle LDA sur l'ensemble de la base, nous avons constaté que certains textes vides étaient malgré tout classés dans une catégorie, simplement parce qu'ils étaient présents dans les données en entrée. En d'autres termes, le modèle attribuait un sujet à un champ vide, ce qui n'a pas de sens et fausse l'interprétation : dans la nouvelle variable contenant les catégories issues du LDA, on retrouvait ainsi des lignes avec un texte vide associé à une thématique, comme si le modèle avait 'catégorisé du vide'.

Pour éviter ce biais, nous avons adopté une nouvelle approche plus rigoureuse. Nous avons d'abord isolé les textes non vides, c'est-à-dire les enregistrements contenant effectivement une suggestion. Le modèle LDA a donc été appliqué uniquement sur cette sous-base, ce qui garantit que chaque catégorisation repose sur un contenu textuel réel.

En parallèle, nous avons soigneusement conservé les identifiants (IDs) des textes vides, afin de pouvoir les réintégrer dans la base complète après classification. Cela permet de reconstituer une base cohérente, où :

les textes contenant une suggestion sont associés à une catégorie issue du LDA,

et les textes vides conservent leur place, avec éventuellement une étiquette neutre comme 'Non renseigné' ou

NA dans la variable de catégorie.

Cette méthode permet ainsi de respecter la structure initiale de la base, d'éviter des classifications erronées sur des données absentes, et de garantir une analyse fiable et interprétable.

```
# 1. Extraire les ID avant prétraitement (tweetsDF)
Id_initial_texte <- unique(Texte_JI$id)
length(Id_initial_texte)
```

```
## [1] 128
```

```
# 2. Extraire les ID après prétraitement (text_df)
Id_final_texte <- unique(Stop_texte$id)
length(Id_final_texte)
```

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

```
#. Identifier les tweets manquants (présents avant, absents après)
Id_texte_NA <- setdiff(Id_initial_texte, Id_final_texte)
length(Id_texte_NA)
```

```
## [1] 83
```

3. Analyse Thématique (LDA)

Il est courant d'avoir une collection de documents, comme des articles de presse ou des publications sur les réseaux sociaux, que l'on souhaite diviser en thèmes. Autrement dit, on veut savoir quel est le sujet principal dans chaque document. Cela peut se faire grâce à une technique appelée modélisation thématique (topic modeling). Ici, nous allons explorer la modélisation thématique à travers la méthode LDA (Latent Dirichlet Allocation).

LDA repose sur deux grands principes : Chaque document est un mélange de plusieurs sujets

Chaque sujet est un mélange de mots

Un exemple classique serait de supposer qu'il existe deux grands sujets dans les actualités : la politique et le divertissement. Le sujet politique contiendra des mots comme élu, gouvernement,

Tandis que le sujet divertissement contiendra des mots comme film, acteur. Mais certains mots peuvent apparaître dans les deux, comme prix ou budget.

LDA va identifier : les mélanges de mots qui composent chaque sujet, et les mélanges de sujets qui composent chaque document. Voyons cela à travers un exemple : On commence par créer notre modèle LDA. La fonction LDA() nécessite en entrée une matrice document-terme (DocumentTermMatrix), que l'on peut créer à partir de notre base déjà prétraitée que nous avons généré précédemment.

Création d'une matrice document-thème

```
# création d'une matrice document-thème pour LDA
df_dtm <- Stop_texte %>%
  count(id, word) %>%
  cast_dtm(id, word, n)
```

Choix du nombre k de thèmes

Dans le cadre de la modélisation thématique avec LDA (Latent Dirichlet Allocation), un des éléments clés du paramétrage est le choix du nombre de thèmes (K). Ce paramètre n'est pas déterminé automatiquement par le modèle ; il doit être choisi par l'utilisateur, en fonction des données et des objectifs de l'analyse. Or, le nombre de thèmes a un impact direct sur la qualité et la lisibilité du modèle :

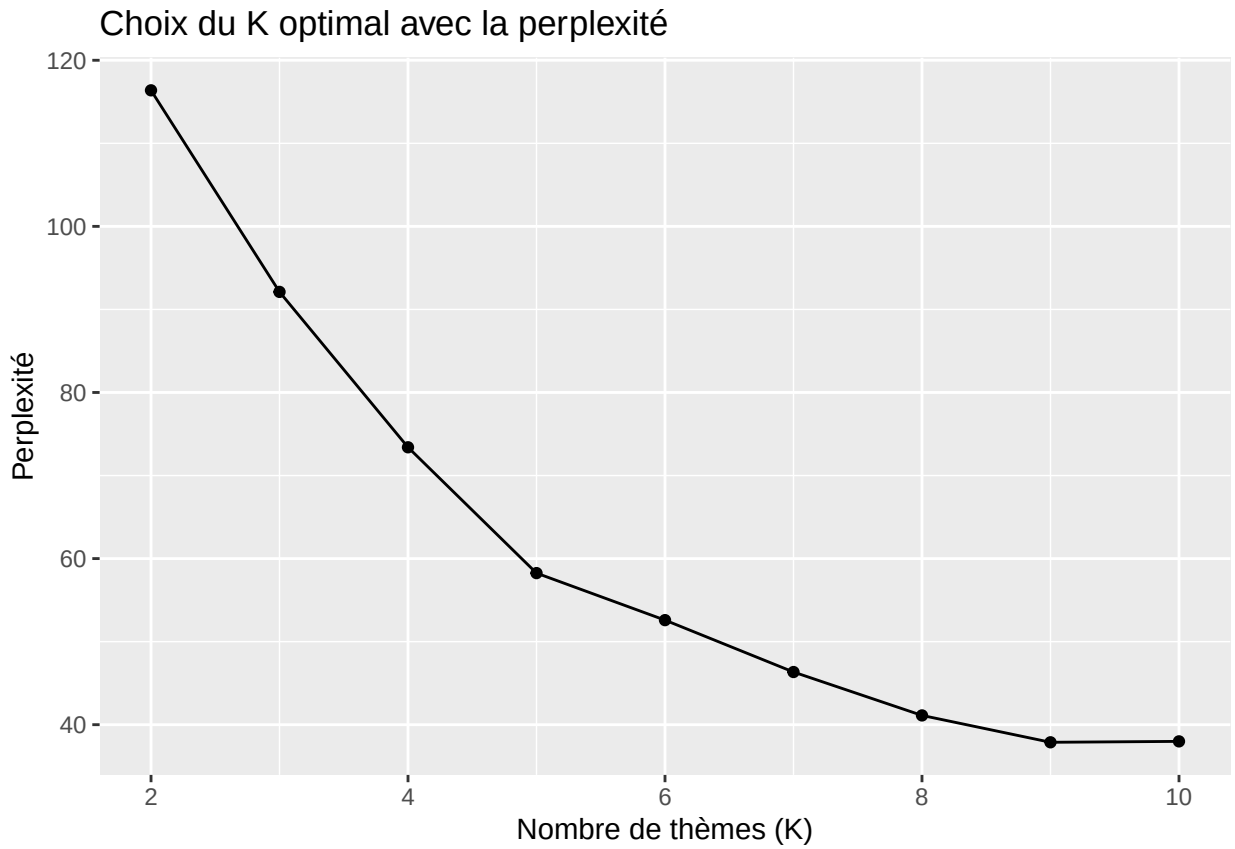
Un K trop petit risque de regrouper des thématiques très différentes dans un même sujet, rendant le résultat peu précis.

Un K trop grand peut sur-segmenter les données, en produisant des thèmes trop spécifiques ou redondants, souvent difficiles à interpréter.

C'est pourquoi il est important de trouver un équilibre, c'est-à-dire un K optimal qui capte suffisamment de variété sans trop complexifier le modèle.

Afin de déterminer lenombre K optimal de thèmes, on utilisons la perplexité du modèle pour plusieurs valeurs de K La perplexité est une mesure standard issue de la modélisation probabiliste, souvent utilisée pour évaluer les modèles de langage. Dans le contexte de LDA, elle mesure dans quelle mesure le modèle 'explique' les données textuelles

```
k_values <- 2:10
perplexities <- sapply(k_values, function(k) {
  lda_model <- LDA(df_dtm, k = k, control = list(seed = 1234))
  perplexity(lda_model)
})

# Afficher le graphique
ggplot(data.frame(K = k_values, Perplexity = perplexities), aes(x = K, y = Perplexi
  geom_point() + geom_line() +
  labs(title = "Choix du K optimal avec la perplexité",
       x = "Nombre de thèmes (K)", y = "Perplexité")
```



Le graphique montre une forte diminution de la perplexité entre K = 2 et K = 7, ce qui indique que chaque thème ajouté dans cette plage apporte une réelle amélioration du modèle. Ensuite, à partir de K ≈ 8, la courbe commence à s'aplatir : les gains supplémentaires deviennent de plus en plus faibles.

Ce comportement suggère qu'à partir de K = 8, ajouter davantage de thèmes n'améliore plus significativement la qualité du modèle, tout en augmentant sa complexité. On peut donc considérer K = 8 comme un bon compromis, car il permet de capter une diversité raisonnable de thématiques sans trop fragmenter les données.

Cela justifie donc le choix de 8 thèmes comme valeur optimale dans notre modélisation LDA.

Généralement En théorie, la perplexité est censée diminuer à mesure que le nombre de thèmes (K) augmente. En effet, un modèle avec plus de thèmes dispose de plus de 'flexibilité' pour représenter les textes de manière fine. Cela se traduit généralement par une meilleure capacité à prédire les mots observés dans les documents — donc une perplexité plus faible.

Cependant, ce comportement n'est pas garanti dans tous les cas. Il peut arriver que la perplexité stagne voire augmente à partir d'un certain K, ou ne suive pas une baisse régulière. Ce phénomène peut être lié à plusieurs facteurs, notamment à la nature des textes analysés.

Un cas courant :

Les mots utilisés dans les documents peuvent être très variés même s'ils expriment des idées similaires. Par exemple, des mots comme gouvernement, État, autorités, institution peuvent tous renvoyer à la même notion politique, mais être traités comme des termes distincts par le modèle. Cela peut fragmenter artificiellement les thèmes, ou faire croire à une diversité de contenus plus grande qu'en réalité.

Dans ces situations, la perplexité peut ne plus refléter fidèlement la 'cohérence sémantique' des thèmes. Elle devient donc une mesure limitée, surtout si les textes sont courts, informels ou s'ils contiennent beaucoup de synonymes ou paraphrases.

Une solution souvent utilisée : Groupage par thème

```
text_df2 <- Stop_texte %>%
  mutate(text_semantic = word) %>%  # dupliquer la colonne lemmatisée
  mutate(
    text_semantic = str_replace_all(text_semantic, "\\b(manger|plat|mets|piment|res
    text_semantic = str_replace_all(text_semantic, "\\b(sketch|micro|temps|chanter|
    text_semantic = str_replace_all(text_semantic, "\\b(audible|sonorisation|techni
    text_semantic = str_replace_all(text_semantic, "\\b(communication|organisation|
  )
```

Limites : pas trop flexible surtout en cas de grands volumes de données

Dans ce qui suit, nous continuerons directement avec les données déjà prétraitées et visualisées sans appliquer un groupage supplémentaire.

```
lda_model <- LDA(df_dtm, k = 8, control = list(seed = 1234))

# Termes par thème
terms_by_topic <- tidy(lda_model, matrix = "beta")
terms_by_topic
```

```
## # A tibble: 1,720 x 3
##    topic term         beta
##    <int> <chr>       <dbl>
##  1     1 chanter 5.02e-154
##  2     2 chanter 4.94e-324
##  3     3 chanter 5.02e-154
##  4     4 chanter 1.88e-154
##  5     5 chanter 1.81e-  2
##  6     6 chanter 3.48e-154
##  7     7 chanter 2.77e-154
##  8     8 chanter 1.88e-154
##  9     1 donner  1.14e-153
## 10     2 donner  4.88e-  2
## # i 1,710 more rows
```

La colonne beta représente la probabilité qu'un mot donné appartienne à un thème particulier. En d'autres termes, plus la valeur de beta est élevée pour un mot dans un thème, plus ce mot est représentatif de ce thème

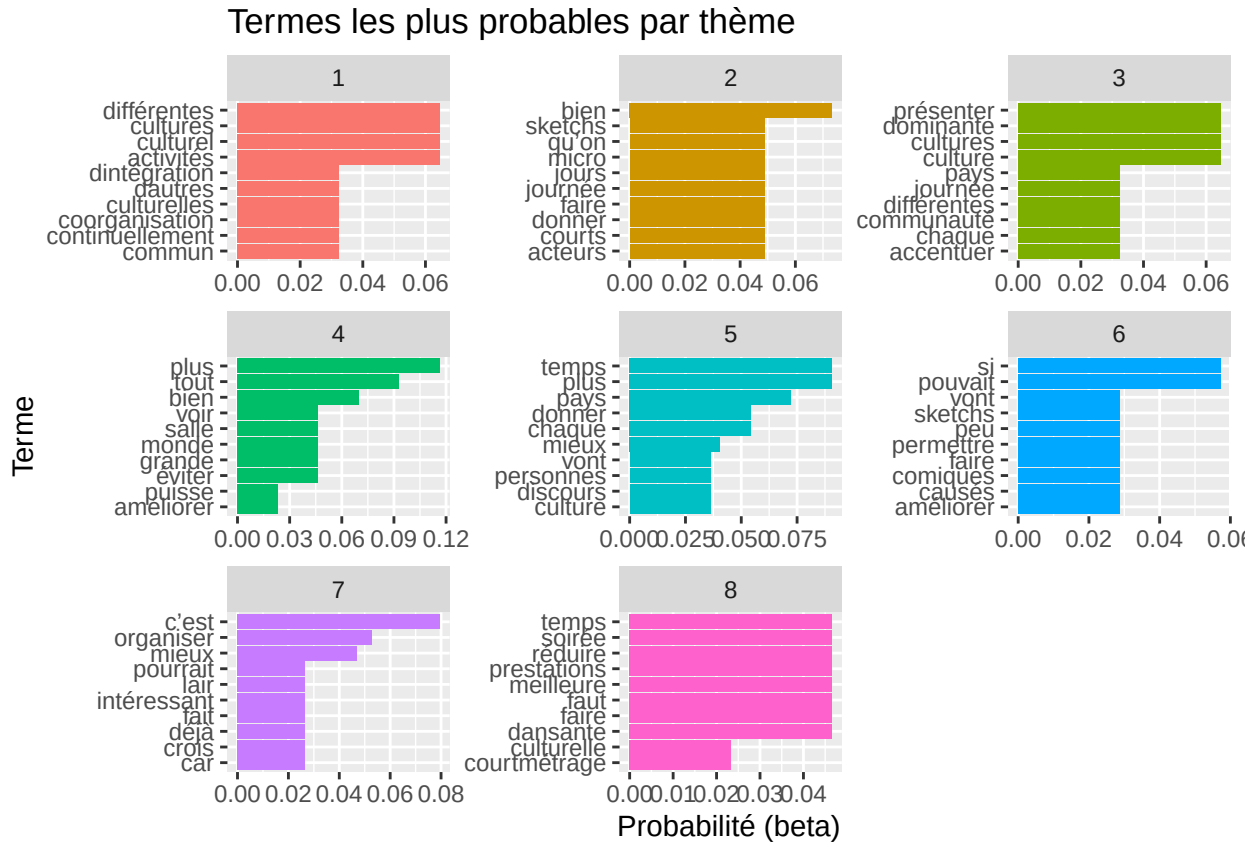
```
top_terms <- terms_by_topic %>%
  group_by(topic) %>%
  slice_max(beta, n = 10, with_ties = FALSE) %>%  # Prend exactement 10 termes par
  ungroup() %>%
  arrange(topic, -beta)  # Trie par thème et par probabilité décroissante

# Vérification du nombre de termes sélectionnés par thème
top_terms %>%
  count(topic)
```

```
## # A tibble: 8 x 2
##   topic     n
```

```
##    <int> <int>
## 1      1    10
## 2      2    10
## 3      3    10
## 4      4    10
## 5      5    10
## 6      6    10
## 7      7    10
## 8      8    10
```

10 termes ont été sélectionnés par thèmes. On peut les visualiser également

```
top_terms %>%
  mutate(term = reorder_within(term, beta, topic)) %>%
  ggplot(aes(beta, term, fill = factor(topic))) +
  geom_col(show.legend = FALSE) +
  facet_wrap(~ topic, scales = "free") +
  scale_y_reordered() +
  labs(title = "Termes les plus probables par thème",
       x = "Probabilité (beta)", y = "Terme")
```




```
# Étape 4 : Classification des textes par thème
textes_gamma <- tidy(lda_model, matrix = "gamma")
# Afficher les textes avec leur thème dominant
textes_classified <- textes_gamma %>%
  group_by(document) %>%
  slice_max(gamma) %>%
```

```
  ungroup()
```

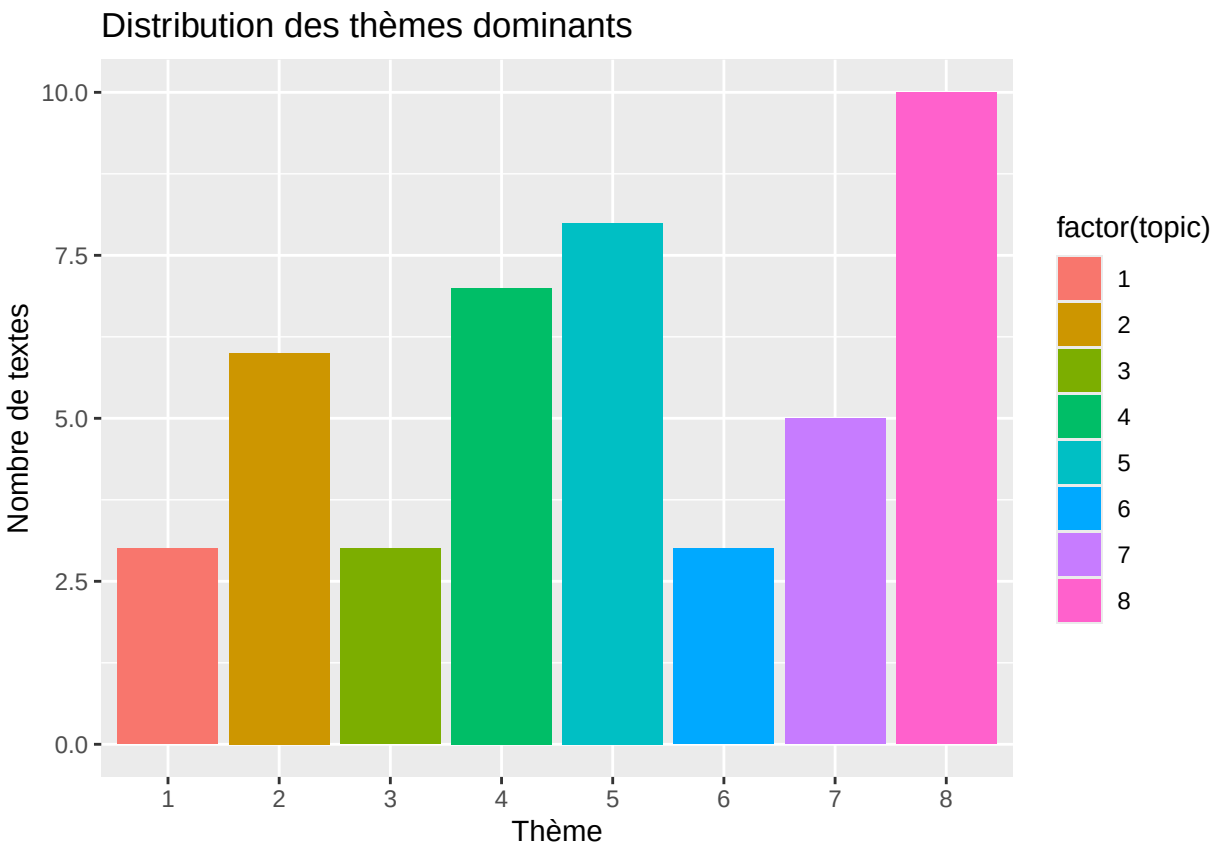
Ici pour chaque texte, on peut voir la probabilité qu'il a d'appartenir à chacun des thèmes.

```
# Nombre de texte dans chaque thème
textes_classified %>%
  count(topic)
```

```
## # A tibble: 8 x 2
##   topic     n
##   <int> <int>
## 1     1     3
## 2     2     6
## 3     3     3
## 4     4     7
## 5     5     8
## 6     6     3
## 7     7     5
## 8     8    10
```

Pour chaque thème, on voit le nombre de textes

```
# Visualisation des nombres de texte dans chaque thème
textes_classified %>%
  ggplot(aes(factor(topic), fill = factor(topic))) +
  geom_bar() +
  labs(title = "Distribution des thèmes dominants",
       x = "Thème", y = "Nombre de textes")
```



Labellisation (très subjective)

```
# Labelliser les thèmes

textes_gamma <- textes_gamma %>%
  mutate(topic_label = case_when(
    topic == 1 ~ "Diversité culturelle et activités communes",
    topic == 2 ~ "Performances artistiques et intervention scénique",
    topic == 3 ~ "Présentation des cultures par les communautés",
    topic == 4 ~ "Aménagement de l'espace et la gestion du temps",
    topic == 5 ~ "Organisation",
    topic == 6 ~ "Suggestions d'amélioration",
    topic == 7 ~ "Organisation générale et impression globale",
    topic == 8 ~ "animation",
    TRUE ~ "Autre"
  ))
```

4. Approche Alternative avec BERTopic

BERTopic est un outil puissant de topic modeling (modélisation de sujets) qui permet d'extraire automatiquement des thèmes principaux à partir de textes non structurés. Il se distingue des approches classiques comme

LDA par sa capacité à capturer des relations sémantiques en se basant sur le texte et non des motstokenisés.

Installation de miniconda et chargement du package reticulate

Pour utiliser les bibliothèques Python dans R (comme bertopic, sentence-transformers, etc.) qui sont nécessaire à notre analyse, on utilise le package reticulate, qui agit comme un pont entre R et Python. Afin d'assurer que tout fonctionne dans un environnement propre et contrôlé, nous allons installer Miniconda, une version légère de Conda, qui sert à gérer les environnements Python.

```
library(reticulate)

# Voir l'environnement actif de reticulate

py_config()
```

```
## python:      C:/Users/lenovo/AppData/Local/R/cache/R/reticulate/uv/cache/archiv
v0/_3TGLwPWf9or45vcJp-5G/Scripts/python.exe
## libpython:    C:/Users/lenovo/AppData/Local/R/cache/R/reticulate/uv/python/cpyt
3.11.12-windows-x86_64-none/python311.dll
## pythonhome:    C:/Users/lenovo/AppData/Local/R/cache/R/reticulate/uv/cache/archi
v0/_3TGLwPWf9or45vcJp-5G
## virtualenv:    C:/Users/lenovo/AppData/Local/R/cache/R/reticulate/uv/cache/archi
v0/_3TGLwPWf9or45vcJp-5G/Scripts/activate_this.py
## version:      3.11.12 (main, Apr  9 2025, 04:03:34) [MSC v.1943 64 bit (AMD64)]
## Architecture:   64bit
## numpy:       C:/Users/lenovo/AppData/Local/R/cache/R/reticulate/uv/cache/archive
v0/_3TGLwPWf9or45vcJp-5G/Lib/site-packages/numpy
## numpy_version:  2.2.4
##
## NOTE: Python version was forced by VIRTUAL_ENV
```

```
# Installation des packages nécessaires dans l'environnement actif de reticulate
reticulate::py_install(
  packages = c("sentence-transformers", "hdbscan", "umap-learn", "bertopic"),
  pip = TRUE
)
```

```
## Using virtual environment "C:/Users/lenovo/AppData/Local/R/cache/R/reticulate/uv
v0/_3TGLwPWf9or45vcJp-5G" ...
```

Importation des modules Python

Chaque module que nous importons est lié à une fonctionnalité clé :

- sentence_transformers : gestion des modèles d'embedding de texte

- hdbscan : algorithme de clustering utilisé par BERTopic

- bertopic : la librairie principale pour la modélisation de sujets

```
sentence_transformers <- import("sentence_transformers")
hdbscan <- import("hdbscan")
bertopic <- import("bertopic")
umap <- import("umap")  # important pour fixer le random_state
```

```
# 🔹 Modèle d'embedding (changeable par d'autres plus bas)
embedding_model <- sentence_transformers$SentenceTransformer("paraphrase-MiniLM-L6-
```

- umap : utilisé pour projeter les embeddings dans un espace de plus faible dimension Ici on utilise 'paraphrase-MiniLM-L6-v2', un modèle rapide et efficace. Mais il existe d'autres variétés plus puissante mais qui sont plus robuste. Le tableau qui suit donne quelques détails.

```
# 🔹 Comparaison de modèles d'embedding pour BERTopic
embedding_models <- data.frame(
  Modele = c(
    "paraphrase-distilbert-base-nli-stsb",
    "bert-base-nli-mean-tokens",
    "all-mpnet-base-v2"
  ),
  Taille = c(768, 768, 768),
  Precision_Semantique = c(
    "Bonne précision sémantique",
    "Très bonne précision sémantique",
    "Excellente précision sémantique"
  )
)
# 🔹 Affichage du tableau
knitr::kable(embedding_models, caption = "Tableau comparatif de modèles d'embeddin
```

Table 1: Tableau comparatif de modèles d'embedding utilisables avec BERTopic

Modele	Taille	Precision_Semantique
paraphrase-distilbert-base-nli-stsb	768	Bonne précision sémantique
bert-base-nli-mean-tokens	768	Très bonne précision sémantique
all-mpnet-base-v2	768	Excellente précision sémantique

```
hdbscan_model <- hdbscan$HDBSCAN(
  min_cluster_size = reticulate::r_to_py(3L),
  min_samples = reticulate::r_to_py(1L)
)

# 🔹 Réduction de dimension via UMAP avec seed fixée pour reproductibilité
umap_model <- umap$UMAP(
  n_neighbors = 15L,
  n_components = 5L,
  min_dist = 0.0,
  metric = "cosine",
  random_state = 42L  # 🔹 Seed fixée ici
)
```

```
# 🔧 Création du modèle BERTopic
topic_model <- bertopic$BERTopic(
  language = "french",
  embedding_model = embedding_model,
  hdbscan_model = hdbscan_model,
  umap_model = umap_model
)

# 📊 Préparation des données

docs <- Texte_JI_filtered$Texte
ids <- Texte_JI_filtered$id  # on garde l'id associé à chaque texte

result <- topic_model$fit_transform(docs)


# 🔍 Extraction des résultats
topics <- result[[1]]
probs <- result[[2]]

topics
```

```
##  [1] 4 3 5 3 1 0 0 0 2 2 1 0 2 2 3 0 5 0 3 4 1 1 1 1 3 3 1 0 0 2 0 4 0 0 0 0 1 2
## [39] 1 5 4 0 1 0 4
```

```
probs
```

```
##  [1] 0.7649678 1.0000000 1.0000000 0.7850124 1.0000000 0.8767291 1.0000000
##  [8] 1.0000000 1.0000000 1.0000000 0.9240548 1.0000000 0.8713598 1.0000000
## [15] 1.0000000 1.0000000 1.0000000 0.9533261 1.0000000 1.0000000 1.0000000
## [22] 0.9240548 0.9210614 1.0000000 0.8374228 0.6877206 1.0000000 1.0000000
## [29] 1.0000000 0.8713598 1.0000000 1.0000000 0.9533261 1.0000000 0.8585130
## [36] 1.0000000 1.0000000 0.9615890 0.9210614 1.0000000 1.0000000 1.0000000
## [43] 1.0000000 1.0000000 0.8038874
```

```
# 🔧 Reconstruction du data.frame avec id + texte + classe
base_categorisee <- data.frame(
  id = ids,
  texte = docs,
  classe = topics,
  proba = probs
)

# 📊 Affichage des infos sur les thèmes trouvés
topic_info <- topic_model$get_topic_info()
head(topic_info)
```

```
##   Topic Count                                  Name
## 1     0    15                 0_de_la_pays_les
## 2     1    10              1_bien_pour_soit_son
## 3     2     6           2_tout_une_soirée_monde
## 4     3     6      3_sketchs_des_courts_acteurs
## 5     4     5    4_temps_prestations_réduire_le
## 6     5     3 5_court_métrage_culturelle_intégrer
##                                                  Representation
```

```
## 1                         de, la, pays, les, présenter, culture, organisation, cultur
## 2                                 bien, pour, soit, son, jours, espace, le, est, jou
## 3                               tout, une, soirée, monde, voir, salle, grande, dansa
## 4                                 sketchs, des, courts, acteurs, les, aux, micro, e
## 5                        temps, prestations, réduire, le, il, faut, des, chanter, acc
## 6 court, métrage, culturelle, intégrer, créativité, preuve, diversité, participant
##
## 1 Que chaque pays présente sa culture de telle sorte les personnes qui ne connaissai
organisation pour permettre aux différentes cultures de nous présenter des activités c
## 2
## 3
## 4
## 5
## 6
métrage sur la diversité culturelle pour sensibiliser les participants
```

```r
topic_info$label <- c(
  "Célébration et partage des cultures nationales",                # Topic 0
  "Aspect technique et organisation",                              # Topic 1
  "Mieux aménager l'espace",                                       # Topic 2
  "Sketchs et prestations",                                        # Topic 3
  "Gestion du timing lors des interventions",                      # Topic 4
  "Touche créative et courmétrages"                                  # Topic 5
)


base_categorisee <- merge(
  base_categorisee,
  topic_info[, c("Topic", "label")],
  by.x = "classe",
  by.y = "Topic",
  all.x = TRUE
)


base_categorisee <- subset(base_categorisee, select = -c(classe, proba))
```

```r
doc_vides <- data.frame(
  id = Id_texte_NA
)


# 2. Identifier les noms des autres colonnes (sauf "document")
autres_colonnes <- setdiff(names(base_categorisee), "id")

# 3. Ajouter des NA pour les autres colonnes
doc_vides[autres_colonnes] <- NA

# 4. Fusionner avec la base existante
textes_by_topic_complet <- rbind(base_categorisee, doc_vides)

# 5. Optionnel: trier par document si nécessaire
textes_by_topic_complet <- textes_by_topic_complet[order(textes_by_topic_complet$i
```

```r
# Joindre les thèmes dominants avec les tweets originaux

Texte_JI$Texte <- NULL

textes_classified <- Texte_JI %>%
  inner_join(textes_by_topic_complet, by = c("id" = "id"))

head(textes_classified)
```

```
## # A tibble: 6 x 5
##    id `Classe de l'étudiant :` `Nationalité de l'étudiant` texte      label
##  <dbl> <chr>                   <chr>                       <chr>      <chr>
## 1     1 AS2                     Congo                       <NA>       <NA>
## 2     2 ISEP1                   Cameroun                    Donner à tem~ Gest~
## 3     3 AS2                     Congo                       <NA>       <NA>
## 4     4 AS1                     Sénégal                     Faire des sk~ Sket~
## 5     5 AS1                     Cameroun                    Intégrer un ~ Touc~
## 6     6 ISE1 Eco                Sénégal                     <NA>       <NA>
```

Table 2: Suggestions pour améliorer l'organisation de la journée d'intégration

id	texte	label
1	NA	NA
2	Donner à temps le micro aux personnes qui vont chanter.	Gestion du timing lors des interventions
3	NA	NA
4	Faire des sketchs courts et bien donner le micro aux acteurs.	Sketchs et prestations
5	Intégrer un court-métrage sur la diversité culturelle pour sensibiliser.	Touche créative et courmétrages
6	NA	NA
7	Faire des sketch courts et donner le micro aux acteurs sur la scène.	Sketchs et prestations
8	NA	NA
9	Améliorer le son pour que tout soit bien audible.	Aspect technique et organisation
10	NA	NA

CONCLUSION

L'analyse des questions ouvertes à l'aide des techniques de text mining met en lumière l'importance du *prétraitement* des données textuelles. Cette étape cruciale permet de nettoyer, normaliser et structurer le texte afin de le rendre exploitable pour les algorithmes d'analyse. Cependant, il est important de noter que les outils de prétraitement sont plus adaptés et optimisés pour *l'anglais*, notamment en ce qui concerne les listes de *stop words*, les outils de *stemming* ou de *lemmatisation*. Cela constitue un frein lorsqu'on travaille sur des textes en français ou dans d'autres langues moins représentées.

Parmi les méthodes explorées, *LDA (Latent Dirichlet Allocation)* permet d'identifier des thématiques en se basant sur la fréquence des mots. Toutefois, cette approche présente des *limites importantes* : elle repose uniquement sur la *co-occurrence de mots*, sans prendre en compte leur sens réel ou leur contexte sémantique. Ainsi, des textes exprimant des idées similaires avec des mots différents peuvent ne pas être associés au même thème, ce qui réduit la pertinence de l'analyse dans certains cas.

C'est dans ce cadre que *BERTopic* se distingue. En s'appuyant sur des modèles d'embeddings comme BERT, il permet de capter la sémantique des phrases. Il devient alors possible de regrouper des textes similaires même si les mots employés sont différents. Cette approche offre une compréhension plus fine et plus pertinente des idées exprimées dans les données.

Cela dit, il est essentiel de garder à l'esprit que le traitement des textes reste une tâche *complexe* et *imparfaite*. La diversité des styles, des formulations, des niveaux de langue ou encore des erreurs d'écriture rend l'analyse automatique difficile. Les résultats doivent donc être interprétés avec précaution, et idéalement complétés par une validation *humaine* pour garantir leur fiabilité.