Code

Data management et exploitation des données Reddit: Post Diabetes

DE CLERCQ Allan

2024-12-13

Projet R : Data management et exploitation des données Reddit

0. Contexte

Nous disposons d'un ou plusieurs jeux de données. Ces jeux de données sont des extractions de forums reddit, chaque fichier correspondant à un forum. Tous les fichiers ont la même structure.

Si nous disposons de plusieurs fichiers (dans /data), voici quelques consignes supplémentaires :

Au début du script, rassembler les données des différents fichiers au sein d'un même dataframe. Veillez à conserver le nom du forum de chaque enregistrement.

Il est possible que chacun des fichiers n'ait pas la même plage temporelle. Exemple : un fichier peut couvrir la période de janvier 2024 à novembre 2024 et un autre fichier la période de juin 2024 à novembre 2024. Identifiez les dates de début de chacun des fichiers.

1. Premières Consignes

1.1. Consignes

Compléter l'en-tête de ce fichier .Rmd avec votre nom (author), et le nom du jeu de données (title).

Complétez ce .Rmd comme vous le souhaitez, pour répondre aux points / questions si dessous.

- **C1 :** Conservez (au moins) les dataframes d'entrées (après chargement des fichiers csv) dans le cache. (1 point). Modifier les colonnes existantes ou créez de nouvelles colonnes si besoin.
- **C2 :** Exécutez le .Rmd pour produire un fichier .html. (-5 si le .html n'est pas présent dans le dossier)
- **C3 :** Propreté du code : 1 point. Compléter le fichier r_eval_consignes avec 0/1 à chaque consigne que vous avez traité, 0/1 si vous pensez que votre réponse est correcte.

1.2 Requêtes des consignes:

Il faut dans un premier temps déclarer les options de knitr pour le cache et le répertoire de travail.

> **Attention**: Pour reéxécuter le code et récupérer le cache, il faut bien définir le répertoire de travail.

Ensuite, il faut charger les packages nécessaires et importer les données.

- Hide code -

```r
library(RColorBrewer)
library(wordcloud)
library(tidytext)
library(SnowballC)
library(tm)
library(wordcloud)

# Rajouter les autres packages nécessaires ci-dessous
library(dplyr)
library(lubridate)
library(emoji)
library(ggplot2)
library(Cairo)

library(devtools)
library(ggthemr) # devtools::install_github('Mikata-Project/ggthemr')
# Changer le thème ggplot
theme <- ggthemr('grape', set_theme = TRUE)

library(ggstatsplot)
library(nortest)
library(FSA)
library(stopwords)

library(tidyr)
library(proxy)
library(plotly)
library(tibble)
library(FactoMineR)
library(factoextra)
```

Par la suite, il faut importer les données. Dans le cas des données sur le diabète, il y a trois fichiers à importer qui sont stockés dans le dossier `data/` . Le cache sera utilisé pour stocker les données importées.

- `data/new_post_diabetes_241201.csv`
- `data/new_post_diabetes_t1_241201.csv`
- `data/new_post_diabetes_t2_241201.csv`

Les trois différentes données et le dataframe rassemblé seront stockés dans le cache avec des dossiers distincts pour plus de visibilité.

- `cache/data_1/`
- `cache/data_2/`
- `cache/data_3/`
- `cache/data_gathered/`

- Hide code -

```r
data_1 <- read.csv("data/new_post_diabetes_241201.csv", header = TRUE, sep = ";")
```

- Hide code -

```r
data_2 <- read.csv("data/new_post_diabetes_t1_241201.csv", header = TRUE, sep = ";")
```

- Hide code -

```r
data_3 <- read.csv("data/new_post_diabetes_t2_241201.csv", header = TRUE, sep = ";")
```

Il est nécessaire de vérifier que les colonnes des trois jeux de données sont identiques et que les types de données sont les mêmes pour éviter des erreurs lors de la fusion des données.

- Hide code -

```r
# Vérifiacation des données : égalité des colonnes et leur type de données
cols_identical <- all.equal(colnames(data_1), colnames(data_2)) == TRUE &&
        all.equal(colnames(data_1), colnames(data_3)) == TRUE

types_identical <- all.equal(sapply(data_1, class), sapply(data_2, class)) == TRUE &&
        all.equal(sapply(data_1, class), sapply(data_3, class)) == TRUE

if (cols_identical && types_identical) {
  print("Les colonnes sont identiques")
} else {
  print("Les colonnes ne sont pas identiques")
}
```

```
## [1] "Les colonnes sont identiques"
```

Nous pouvons désormais rassembler les données dans un seul dataframe. Également re-vérifier que le nombre de lignes du dataframe rassemblé est bien égal à la somme des lignes des trois jeux de données. Puis supprimer de la mémoire vive les données dorénavant inutiles.

- Hide code -

```
# Rassembler les données
  data_gathered <- rbind(data_1, data_2, data_3)

# Vérifier le nombre de lignes
  print(paste0("Le fichier réunit contient la somme des lignes des 3 données ? = ",
               nrow(data_gathered) == nrow(data_1) + nrow(data_2) + nrow(data_3)))
```

```
## [1] "Le fichier réunit contient la somme des lignes des 3 données ? = TRUE"
```

- Hide code -

```
# Supprimer les données dorénavent inutiles
  rm(data_1, data_2, data_3, cols_identical, types_identical)
```

1.3 Data management:

Dans un premier temps, il est nécessaire de nettoyer les données. Pour cela, il faut vérifier s'il y a des valeurs manquantes dans les données.

- Hide code -

```
# Vérification des valeurs manquantes
data_gathered %>%
  summarise_all(~sum(is.na(.)))
```

```
##   id_post titre auteur texte texte_resume categorie nb_commentaires
## 1       0     0      0     0            0         0               0
##   nb_reactions date_heure_post date_post jour_post heure_post longueur_post
## 1            0               0         0         0          0             0
##   reseau_social forum_lower forum date_extraction
## 1             0           0     0               0
```

Dans notre cas, il n'y a pas de valeurs manquantes. Nous pouvons donc dorénavant re-définir les types de données des colonnes, les colonnes à garder et les colonnes à supprimer. Nous nous retrouverons avec le dataframe `data_gathered_clear` . La fonction `emoji_replace_name()` permet de modifier les emoji par leurs noms, exemple: 😊 en `":smile:"` Les colonnes gardées sont les suivantes :

- `id` : Identifiant du post (caractère)
- `title` : Titre du post (caractère, en minuscule, emojis remplacés)
- `author` : Auteur du post (caractère, emojis remplacés)
- `text` : Texte du post (caractère, en minuscule, emojis remplacés)
- `summary_text` : Résumé du texte (caractère, en minuscule, emojis remplacés)
- `categorie` : Catégorie du post (facteur)
- `nb_com` : Nombre de commentaires (entier)
- `nb_react` : Nombre de réactions (entier)
- `date_post` : Date du post (année-mois-jour heure:minute:seconde)
- `day_post` : Jour du post (facteur)
- `length_post` : Longueur du post (entier)
- `forum` : Forum du post (facteur)
- `date_extract` : Date d'extraction (année-mois-jour heure:minute:seconde)

- Hide code -

```
data_gathered_clear <- data_gathered %>%
 mutate( id = as.character(id_post),
         title = as.character(tolower(emoji_replace_name(titre))),
         author = as.character(emoji_replace_name(auteur)),
         text = as.character(tolower(emoji_replace_name(texte))),
         summary_text = tolower(emoji_replace_name(texte_resume)),
         categorie = as.factor(ifelse(categorie == "", "Inconnu", categorie)),
         nb_com = as.integer(nb_commentaires),
         nb_react = as.integer(nb_reactions),
         date_post = ymd_hms(date_heure_post),
         day_post = wday(date_post, label = TRUE, abbr = FALSE),
         length_post = as.integer(longueur_post),
         forum = as.factor(forum_lower),
         date_extract = ymd(sub(" .*", "", date_extraction))
         ) %>%
   select(id, title, author, text, summary_text, categorie, nb_com, nb_react, date_post, day_post, length_post, for
um, date_extract)
```

2. Par jour

2.1. Consignes

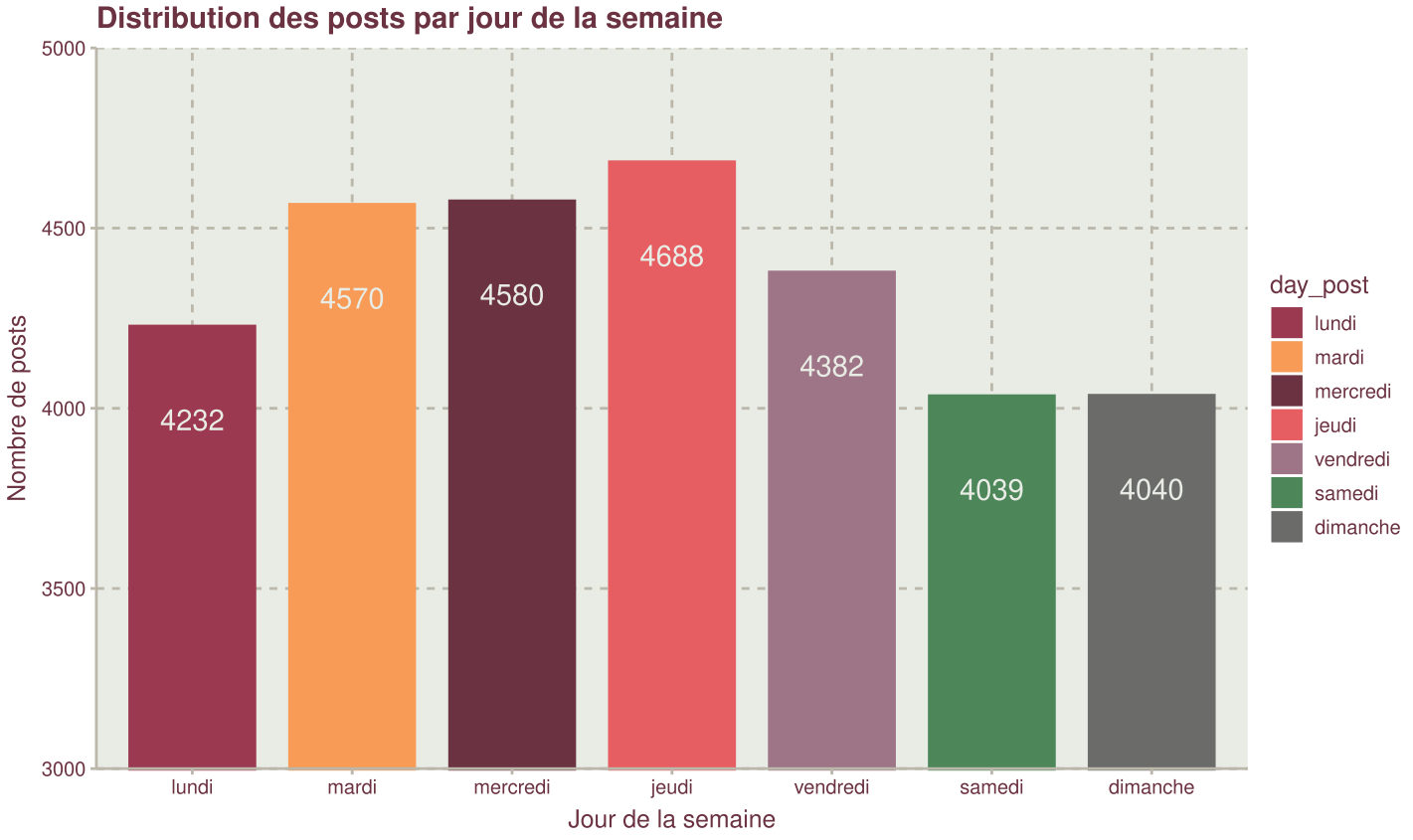
- **C4 :** Proposer un graphique présentant la distribution des posts par jour de publication, de lundi à dimanche. (2 point)

2.2. Requêtes

Nous cherchons à représenter une distribution en fonction d'une donnée catégorielle (jour de la semaine). Pour cela, nous pouvons utiliser plusieurs graphiques : barplot, lollipop, doughnut, pie chart, etc. Dans notre cas, l'utilisation d'un barplot est plus adaptée:

- Hide code -

```
# Barplot de la distribution des posts par jour de la semaine
data_gathered_clear %>%
        mutate(day_post = factor(day_post,
                                 levels = c("lundi", "mardi", "mercredi", "jeudi", "vendredi", "samedi", "dimanche")
)) %>%  # ré-ordonner les jours
        ggplot(aes(x = day_post, fill = day_post)) + # barplot
        geom_bar(stat = "count", width = 0.8) +  # Barplot avec bordures noires
        geom_text(stat = "count", aes(label = after_stat(count)), vjust = 5, size = 5, color = theme$palette$backgr
ound) +  # Ajouter les labels avec une couleur contrastée
        labs(
                title = "Distribution des posts par jour de la semaine", # Titre
                x = "Jour de la semaine", # Nom de l'axe des abscisses
                y = "Nombre de posts" # Nom de l'axe des ordonnées
        ) +
        scale_fill_manual(values = theme$palette$swatch[-1]) +
        scale_y_continuous( # Zoomer sur une plage de l'axe Y
                limits = c(0, 8000),
                expand = c(0, -3000),
                breaks = seq(1000, 8000, by = 500)
        )
```



3. Par heure

3.1. Consignes

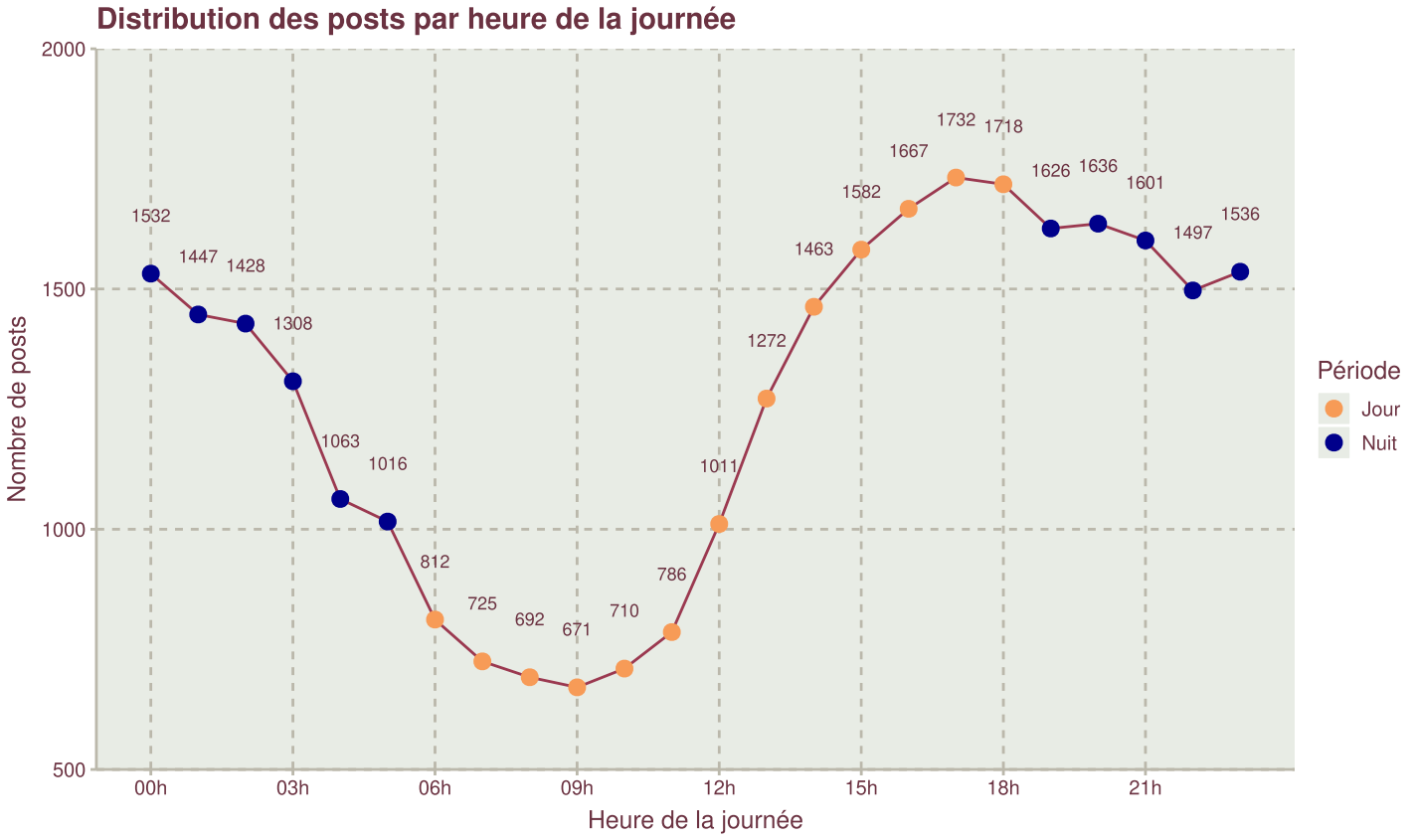
- **C5 :** Proposer un graphique présentant la distribution des posts par heure de publication. Les heures étant arrondies à l'heure, de 0 à 23. (2 points)

3.2. Requêtes

Pour représenter la distribution des posts par heure de publication, nous pouvons utiliser un barplot Dans notre cas, l'utilisation d'un Scatterplot serait plus adaptée. un graphique en bar serait trop chargé et peu lisible. Alors qu'un Scatterplot permet de voir la donnée la plus importante : distribution des posts par heure de la journée. J'ai également ajouté des lignes connectant les points pour une meilleure visualisation et pouvoir apercevoir la **tendance** de la distribution des posts.

- Hide code -

```
# Barplot de la distribution des posts par heure de la journée
data_gathered_clear %>%
        ggplot(aes(x = hour(date_post))) +
        geom_line(stat = "count", group = 1, aes(y = ..count..), linewidth = 0.5) +  # Lignes connectant les points
        geom_point(stat = "count", aes(y = ..count.., color = ifelse(hour(date_post) >= 6 & hour(date_post) <= 18,
"day", "night")), size = 3) +
        scale_color_manual(values = c("day" = theme$palette$swatch[3], "night" = "darkblue"),
                           labels = c("Jour", "Nuit")) + # Labels pour la légende)
        geom_text(stat = "count", aes(label = after_stat(count)), vjust = -4, size = 3, color = theme$palette$swatc
h[4]) +  # Ajouter des étiquettes de texte
        labs(
                title = "Distribution des posts par heure de la journée",
                x = "Heure de la journée",
                y = "Nombre de posts"
        ) +
        scale_y_continuous(
                limits = c(0, 2500),
                expand = c(0, -500)
        ) +
        scale_x_continuous(
                 breaks = seq(0, 23, by = 3),
                 labels = c("00h", "03h", "06h", "09h", "12h", "15h", "18h", "21h")
        ) +
        guides(color = guide_legend(title = "Période"))
```



4. Taille des posts

4.1. Consignes

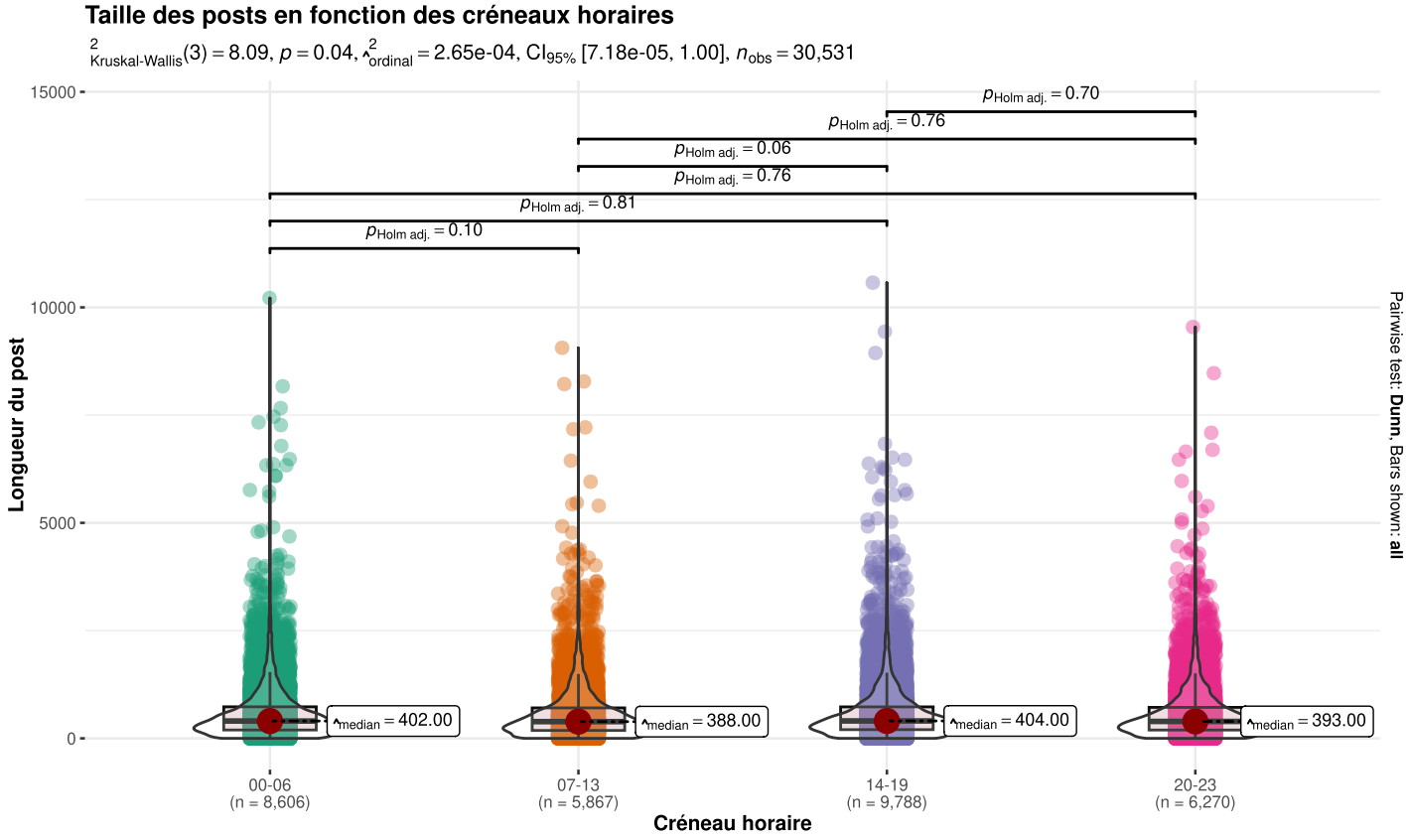
- **C5 :** Proposer un graphique permettant de comparer la taille des posts en fonction des créneaux horaires suivants : (00-06) (07-13) (14-19) (20-23) Vous pouvez ajuster l'axe des ordonnées et exclure les valeurs extrêmes si nécessaire. (2 points)
- **C6 :** A l'aide d'un test statistique, évaluez si la différence de longueur de posts est significative en fonction du créneau horaire. (1 point)

4.2. Requêtes

Pour comparer la taille des posts en fonction des créneaux horaires, nous pouvons faire un boxplot avec un violonplot. Les donnés sont de types par

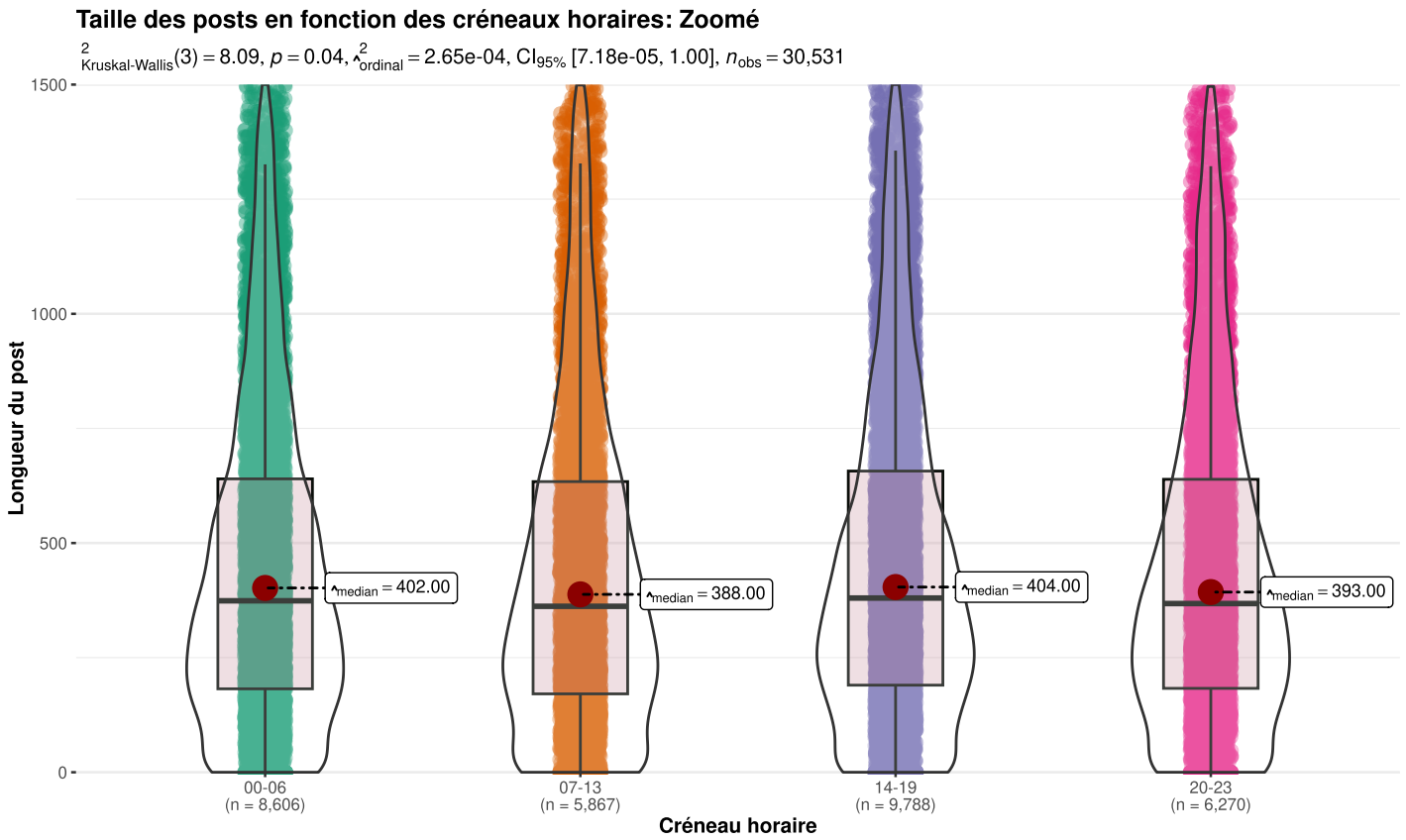
- Hide code -

```
data_stats <- data_gathered_clear %>%
  mutate(hour_group = as.factor(cut(hour(date_post),
                                    breaks = c(0, 6, 13, 19, 23),
                                    labels = c("00-06", "07-13", "14-19", "20-23"),
                                    include.lowest = TRUE))) %>%
  select(id, length_post, hour_group)


p <- data_stats %>%
  ggbetweenstats(
    x = hour_group,
    y = length_post,
    type = "nonparametric",  # afficher la moyenne tronquée.
    xlab = "Créneau horaire",
    ylab = "Longueur du post",
    plot.type = "violin",  # Utilisation d'un violin plot
    pairwise.display = "all",  # Comparaison de toutes les paires de groupes
    p.adjust.method = "holm",  # Appliquer la méthode de correction de Holm
    bf.message = TRUE,  # Afficher le message Bayesien
    results.subtitle = TRUE  # Ajouter un sous-titre avec les résultats du test
  )

p + labs(title = "Taille des posts en fonction des créneaux horaires")
```



- Hide code -

```
p + labs(title = "Taille des posts en fonction des créneaux horaires: Zoomé") +
  scale_y_continuous(limits = c(0, 1500), expand = c(0, 0))
```



- Hide code -

```
# Test de normalité de la longueur des posts  pour chaque créneau: Anderson-Darling
ad_test_results <- data_stats %>%
  group_by(hour_group) %>%
  summarise(p_value = ad.test(length_post)$p.value)

ad_test_results %>%
  mutate(result = ifelse(p_value < 0.05, "ne suit pas une loi normale", "suit une loi normale")) %>%
  print()
```

```
## # A tibble: 4 × 3
##   hour_group p_value result
##   <fct>        <dbl> <chr>
## 1 00-06      3.7e-24 ne suit pas une loi normale
## 2 07-13      3.7e-24 ne suit pas une loi normale
## 3 14-19      3.7e-24 ne suit pas une loi normale
## 4 20-23      3.7e-24 ne suit pas une loi normale
```

On peut voir que la longueur des posts ne suit pas une loi normale. Nous allons donc utiliser un test statistique non paramétrique, le test de Kruskal-Wallis. Ce test nous permettra de savoir s'il existe une différence significative dans la longueur des posts en fonction des créneaux horaires.

- Hide code -

```
# Utilisation d'un test statistique non paramétrique : Kruskal-Wallis
kruskal_test <- kruskal.test(length_post ~ hour_group, data = data_stats)
p_value <- round(kruskal_test$p.value, 5)
significance <- ifelse(p_value < 0.05, "Il existe", "Il n'existe pas")
print(paste0(significance, " une différence significative dans la longueur des posts en fonction des créneaux horai
res: p-value = ", p_value))
```

```
## [1] "Il existe une différence significative dans la longueur des posts en fonction des créneaux horaires: p-valu
e = 0.04426"
```

Il existe donc une différence significative dans la longueur des posts en fonction des créneaux horaires. Pour savoir quel créneau horaire est significativement différent, nous allons utiliser un test post-hoc, le test de Dunn.

- Hide code -

```
# Test post-hoc pour connaitre qul créneau horaire est significativement différent : Dunn test
dunn_test <- dunnTest(data_stats$length_post, data_stats$hour_group, kw = TRUE, label = TRUE, wrap = TRUE)
print(dunn_test$dtres)
```

```
##  [1] "  Kruskal-Wallis rank sum test"
##  [2] ""
##  [3] "data: x and g"
##  [4] "Kruskal-Wallis chi-squared = 8.0865, df = 3, p-value = 0.04"
##  [5] ""
##  [6] ""
##  [7] "                             Comparison of x by g                              "
##  [8] "                                    (Holm)                                     "
##  [9] "Col Mean-|"
## [10] "Row Mean |      00-06      07-13      14-19"
## [11] "---------+--------------------------------"
## [12] "   07-13 |   2.315196"
## [13] "         |     0.1030"
## [14] "         |"
## [15] "   14-19 |  -0.234562  -2.583965"
## [16] "         |     0.8145     0.0586"
## [17] "         |"
## [18] "   20-23 |   1.112973  -1.140594   1.356708"
## [19] "         |     0.5314     0.7621     0.6995"
## [20] ""
## [21] "alpha = 0.05"
## [22] "Reject Ho if p <= alpha"
```

Ce qui nous intéresse ici est la p-value ajustée. Si la p-value ajustée est inférieure à 0.05, alors il existe une différence significative entre les deux groupes. Malheureusement on ne relève pas de différence significative entre les groupes 1 à 1, mais on peut voir que les groupes `07-13` et `14-19` ont un p-value de `0.058`, ce qui est proche de 5%. On peut donc dire que ces deux groupes sont proches d'avoir une différence significative.

Il faut être prudent pour tirer une conclusion de ce test, car il est possible que le test de Dunn ne soit pas assez puissant pour détecter une différence significe entre les groupes. **Nous avons donc besoin de plus de données pour réfuter l'hypothèse d'égalité qui semble très proche de la significativité (différence entre les heures `07-13` et `14-19`)**.

5. Nuage de mots

5.1 : Consignes : Création des listes de tokens

La tokenisation est un processus du traitement automatique des langues (TAL). Elle consiste à diviser un texte en unités plus petites appelées tokens. Ces tokens peuvent être des mots, des phrases, des caractères ou d'autres unités analytiques en fonction de l'objectif de l'analyse. Dans le reste du script, les tokens correspondront à des mots.

- **C7 :** Expliquez le fonctionnement de la fonction **get_tokens**, définies en bas. (1 point)
- **C8 :** Utilisez la fonction **get_tokens** définie plus haut pour créer deux listes de mots : tokens_titre à partir de la colonne *titre* et tokens_texte à partir de la colonne *texte*. Stocker le résultat de cette opération dans le cache. (1 points)

- Hide code -

```
get_tokens <- function(texte) {

  # Convertir le corpus en dataframe tibble pour tidytext
  text_df <- data.frame(texte = texte,
                        stringsAsFactors = FALSE)

  # Tokenization
  unnest_tokens <- text_df %>%
    unnest_tokens(word, texte)

  # Lemmatisation (si nécessaire)
  tokens <- unnest_tokens %>%
    mutate(word = tolower(word),
           word = removePunctuation(word, ucp = TRUE),
           word = removeNumbers(word),
           word = removeWords(word, stopwords()),
           word = trimws(word),
           word = wordStem(word)) %>%
    filter(word != "")

  return(tokens)
}
```

5.2 : Requêtes : Création des listes de tokens

La fonction `get_tokens` permet de transformer un texte en une liste. Elle prend en paramètre un texte et retourne une liste de mots. Les étapes de la tokenisation sont les suivantes :

Conversion du texte en dataframe tibble pour tidytext : On extrait le texte du dataframe et on le stocke dans un dataframe tibble qui permet de manipuler les données plus efficacement.

Tokenization : On utilise la fonction `unnest_tokens` de tidytext pour transformer le texte en tokens. Cette fonction divise une colonne (ici `texte`) en tokens (mots), en applatissant les données en un tableau (un token par ligne).

Lemmatisation (si nécessaire) : Cette étape permet de normaliser les mots en retirant: - Les majuscules (tolower) - La ponctuation (removePunctuation) - Les chiffres (removeNumbers) - Les mots vides (removeWords) - Les mots très fréquents et peu informatifs tels que “le”, “la”, “et”, “ou” en français et “the”, “and”, “or” en anglais (stopwords) - Et en racinisant les mots tel que “winning” devient “win” (wordStem)

Puis on filtre les mots vides et on retourne la liste de mots.

- Hide code -

```
# Création des listes de tokens
tokens_titre <- get_tokens(data_gathered_clear$title)
tokens_texte <- get_tokens(data_gathered_clear$text)
```

5.3 Consignes : Création d’une fonction nuage de mot

- **C9 :** Proposez une fonction **nuage_de_mots** générant un nuage de mots à partir d’un vecteur de tokens. Cette fonction utilisera la fonction **wordcloud** du package **wordcloud**. Cette fonction commence par compter le nombre d’occurrence de chaque token. A l’aide du paramètre **nb_tokens_max**, le nuage de mots n’affichera que les **nb_tokens_max** mots les plus fréquents (pour ne pas surcharger le nuage de mots). A l’aide du paramètre **nb_occurrences_min**, la fonction n’affichera que les tokens pour lesquels le nombre d’occurences du tokens sera supérieur à **nb_occurrences_min**. (3 points)

5.4 Requêtes : Création d’une fonction nuage de mot

- Hide code -

```
# Fonction pour créer un nuage de mots
nuage_de_mots <- function(list_tokens, nb_occurrences_min, nb_tokens_max) {

  # Compter le nombre d'occurrences de chaque token
  word_freq <- list_tokens %>%
    count(word)

  # Créer le nuage de mots
  wordcloud(words = word_freq$word,
            freq = word_freq$n,
            min.freq = nb_occurrences_min,
            max.words = nb_tokens_max,
            random.order = FALSE,
            colors = theme$palette$swatch[-1],
            scale = c(5, 0.5)
    )
}
```

5.5 Consignes : Utilisation 1

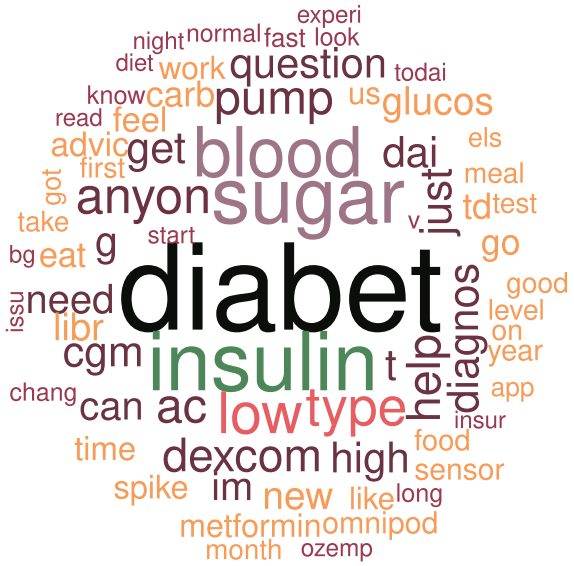
- **C10 :** Utilisez cette fonction pour créer un nuage de mots à partir (1) des mots contenus dans la colonne titre, (2) des mots contenus dans la colonne “texte”. Ajustez les paramètres **nb_tokens_max** et **nb_occurrences_min** pour que les nuages de mots soient lisibles et informatifs. (2 points)

5.6 Requêtes : Utilisation 1

- Hide code -

```
# Nuage de mots à partir de mots contenus dans la colonne titre
nuage_de_mots(tokens_titre, nb_occurrences_min = 100, nb_tokens_max = 70)
title("Nuage de mots à partir des titres")
```

Nuage de mots à partir des titres



- Hide code -

```
# Nuage de mots à partir de mots contenus dans la colonne texte
nuage_de_mots(tokens_texte, nb_occurrences_min = 1000, nb_tokens_max = 70)
title("Nuage de mots à partir des textes")
```

Nuage de mots à partir des textes

5.7 Consignes : Utilisation 2

Dans les deux nuages de mots précédents, il est possible que plusieurs mots n'apportent pas d'information. Ils n'ont pas été filtrés par l'instruction *word = removeWords(word, stopwords())* de la fonction **get_tokens**.

- **C11 :** Adaptez la fonction **nuage_de_mots** avec un paramètre supplémentaire, **liste_stop_words**, une liste de mots supplémentaires qui seront retirés du nuage de mots. Utilisez la fonction mise à jour pour produire à nouveau les nuages de mots à partir des tokens de titre et de

texte.(2 points)

5.8 Requêtes : Utilisation 2

Pour ce faire, nous allons utiliser la fonction `stopwords::stopwords("en",source = "stopwords-iso")` qui permet de récupérer une liste de stopwords en anglais. Nous allons également remplacer le mot “g” par “dexcomg” car il il fait ici référence au produit Dexcom G6.

- Hide code -

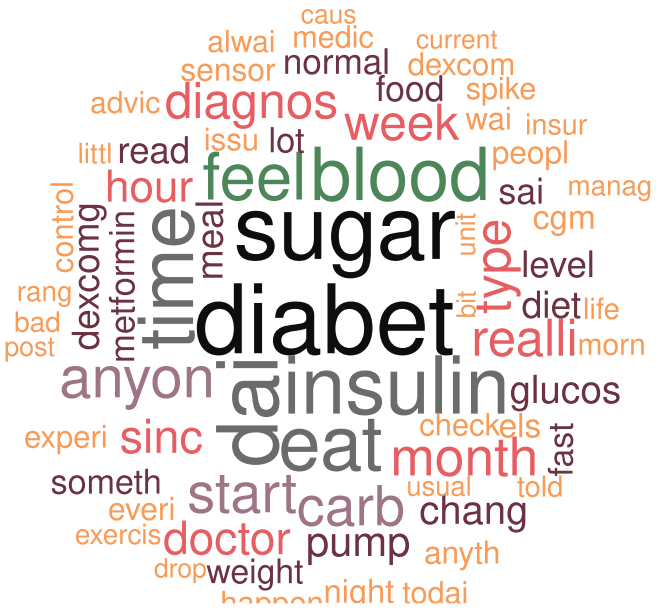
```
# Fonction pour créer un nuage de mots avec des stopwords optionnels
nuage_de_mots <- function(list_tokens, nb_occurrences_min = 1, nb_tokens_max = 100) {
  # Filtrer les tokens pour enlever les stopwords
  list_tokens_filtered <- list_tokens %>%
    mutate(word = gsub("\\bg\\b", "dexcomg", word)) %>%
    filter(!word %in% stopwords::stopwords("en",source = "stopwords-iso")) %>% # Enlever les stopwords en anglais d
u
    filter(!nchar(word) < 3) # Enlever les mots d'une seule lettre

  # Compter le nombre d'occurrences de chaque token
  word_freq <- list_tokens_filtered %>%
    count(word)

  # Créer le nuage de mots
  wordcloud(words = word_freq$word,
            freq = word_freq$n,
            min.freq = nb_occurrences_min,
            max.words = nb_tokens_max,
            random.order = FALSE,
            colors = theme$palette$swatch[-1],
            scale = c(4, 0.5)
  )
}

# Appeler la fonction pour créer un nuage de mots en enlevant les stopwords
nuage_de_mots(tokens_titre, nb_occurrences_min = 100, nb_tokens_max = 70)
title("Nuage de mots à partir des titres (sans stopwords)")
```

Nuage de mots à partir des titres (sans stopwords)

- Hide code -

```
# Appeler la fonction pour créer un nuage de mots en enlevant les stopwords
nuage_de_mots(tokens_texte, nb_occurrences_min = 1000, nb_tokens_max = 70)
title("Nuage de mots à partir des textes (sans stopwords)")
```

Nuage de mots à partir des textes (sans stopwords)



6.Graphique libre

6.1 Consignes

- **C12 :** Proposez un graphique pour continuer d'explorer les jeux de données. (2 points)

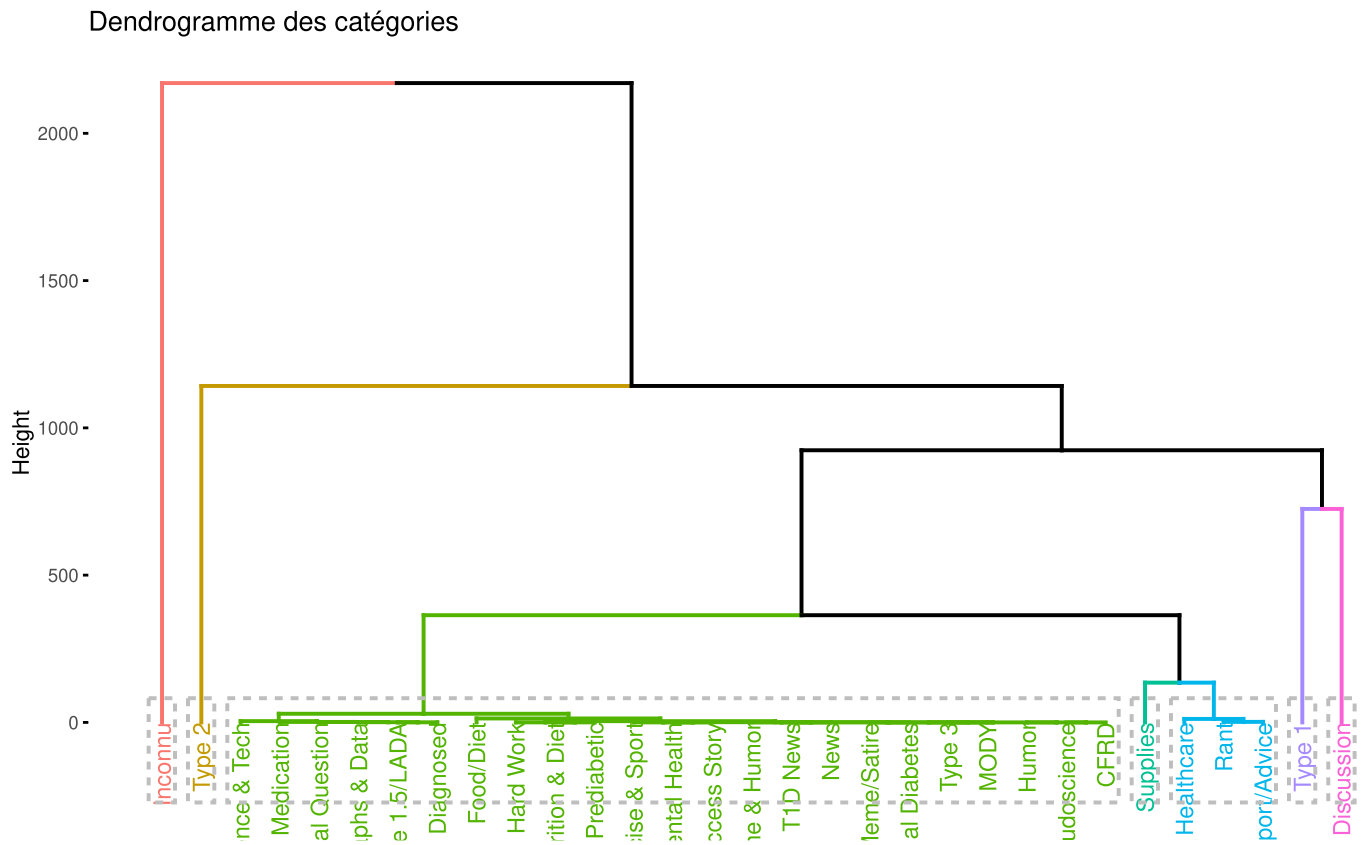
6.2 Requêtes

- Hide code -

```
# Regrouper les tokens par catégorie
tokens_by_category <- data_gathered_clear %>%
        group_by(categorie) %>%
        summarise(tokens = list(get_tokens(text)$word[
                                        !get_tokens(text)$word %in% stopwords::stopwords("en",source = "stopwords
-iso")& nchar(get_tokens(text)$word) > 2]
                                )) %>%
        ungroup()

# Calcul des fréquences des tokens et les limiter aux tokens apparaissant au moins 2 fois dans chaque catégorie
tokens_freq <- tokens_by_category %>%
        unnest(tokens) %>%
        count(categorie, tokens) %>%
        group_by(categorie) %>%
        filter(n >= 2) %>%
        ungroup()

total_words_per_category <- tokens_freq %>%
        group_by(categorie) %>%
        summarise(total = sum(n))  # Total de mots par catégorie

tokens_freq <- tokens_freq %>%
        left_join(total_words_per_category, by = "categorie") %>%
        mutate(frequency = n / total) %>%  # Calcul de la fréquence relative
        select(categorie, tokens, frequency)  # Sélectionner les colonnes d'intérêt



# Réorganiser les données en une matrice avec les tokens
tokens_matrix <- tokens_freq %>%
        spread(key = tokens, value = frequency, fill = 0)

tokens_matrix <- column_to_rownames(tokens_matrix, var = "categorie")

# Réaliser la PCA sur la matrice avec les catégories
pca_result <- PCA(tokens_matrix, graph = FALSE)


# HCPC pour le clustering hiérarchique et nb.clust = -1 pour choisir automatiquement le nombre de clusters
hcpc_result <- HCPC(pca_result, nb.clust = -1, graph = FALSE)
```

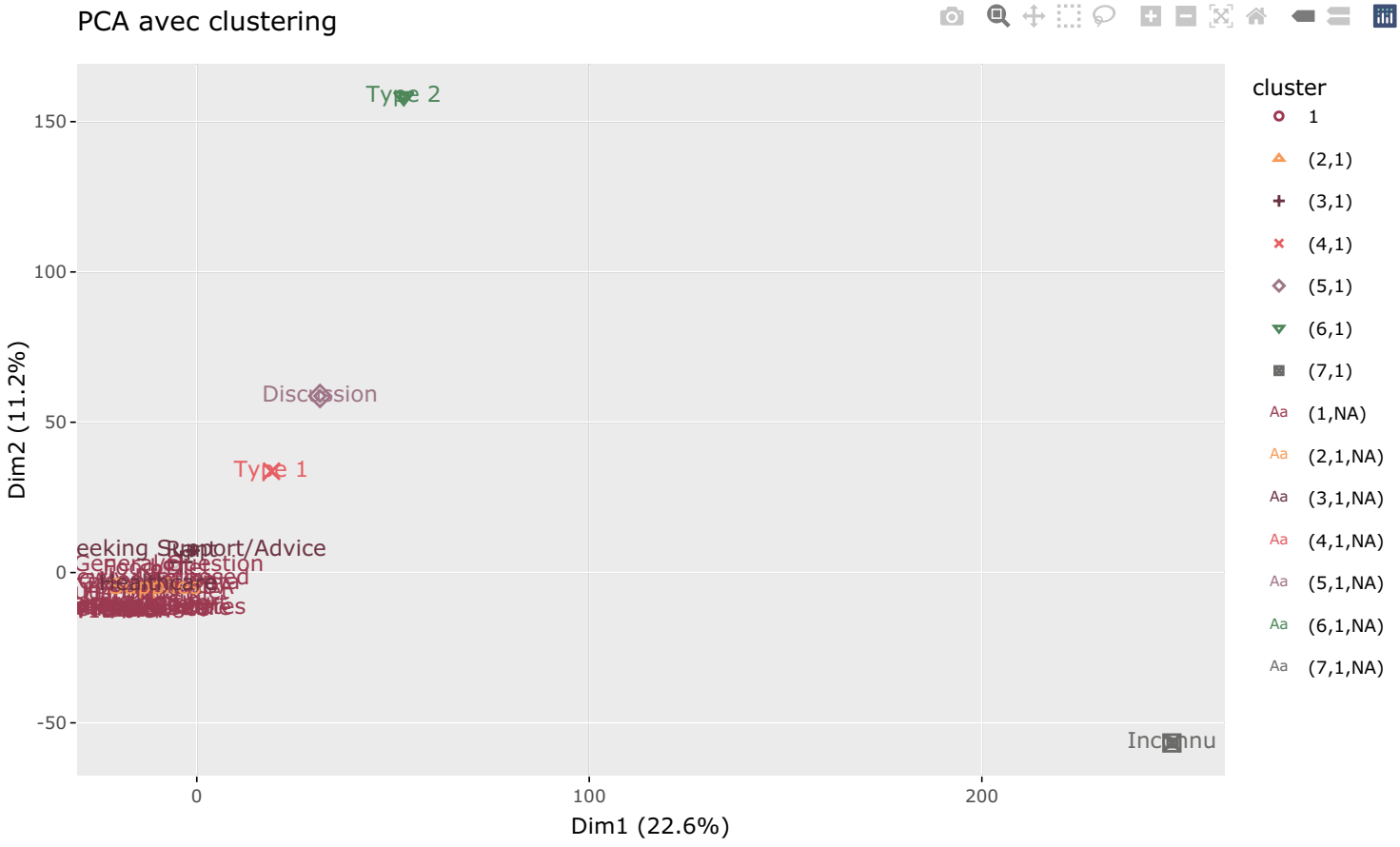
- Hide code -

```
# Dendrogramme avec noms des catégories
fviz_dend(hcpc_result,
          cex = 0.8,                # Taille des étiquettes
          rect = TRUE,              # Ajouter des rectangles autour des clusters
          show_labels = TRUE,       # Afficher les étiquettes
          main = "Dendrogramme des catégories",
          scale = "none")
```



- Hide code -

```
p <- fviz_cluster(hcpc_result,
                  show.clust.cent = TRUE,
                  main = "PCA avec clustering",
                  axes = c(1, 2))
ggplotly(p)
```

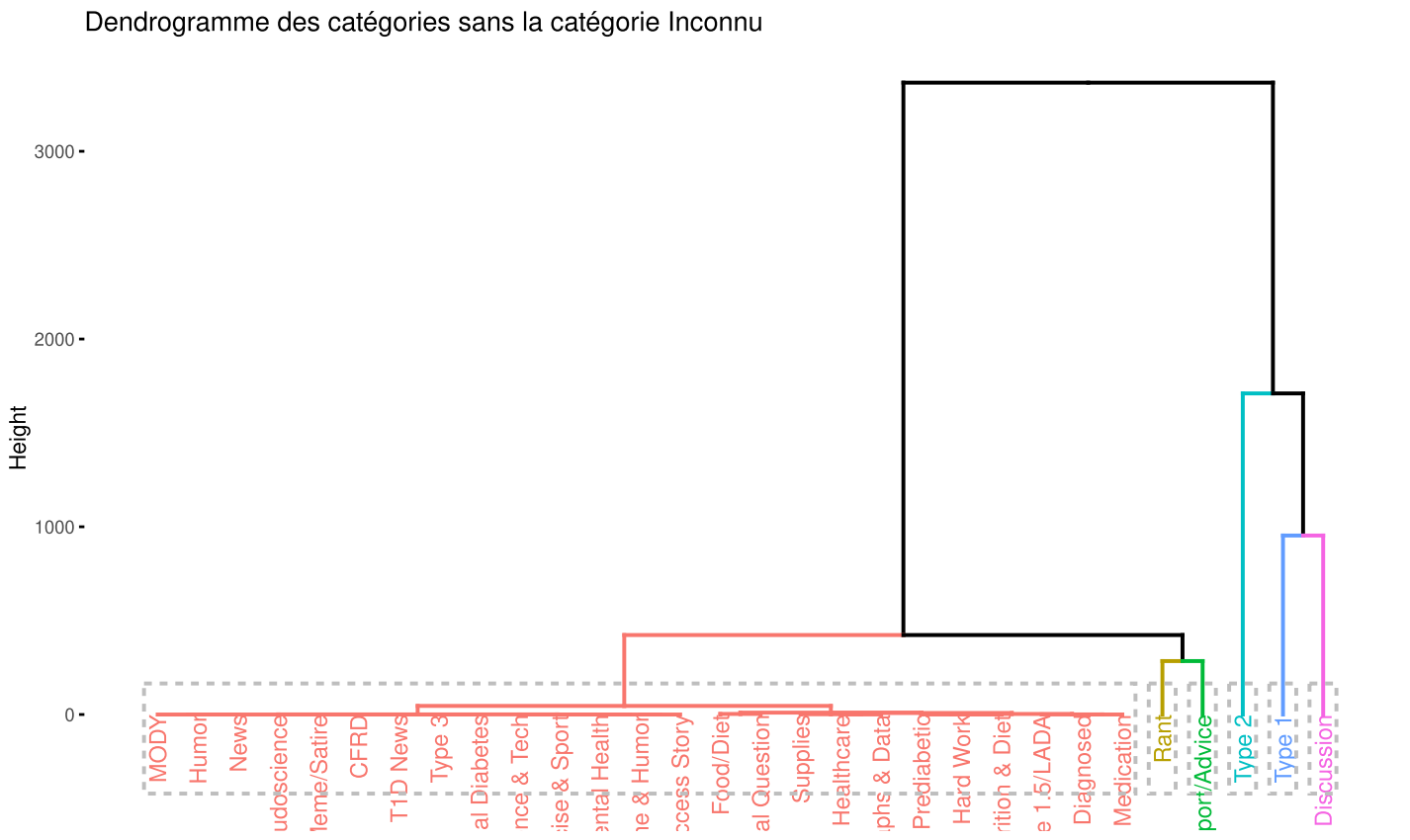


- Hide code -

```
# supprimer la ligne d'index "Inconnu"
tokens_matrix_2 <- tokens_matrix[row.names(tokens_matrix) != "Inconnu", ]

# Réaliser la PCA sur la matrice avec les catégories
pca_result_2 <- PCA(tokens_matrix_2, graph = FALSE)


# HCPC pour le clustering hiérarchique et nb.clust = -1 pour choisir automatiquement le nombre de clusters
hcpc_result_2 <- HCPC(pca_result_2, nb.clust = -1, graph = FALSE)
```

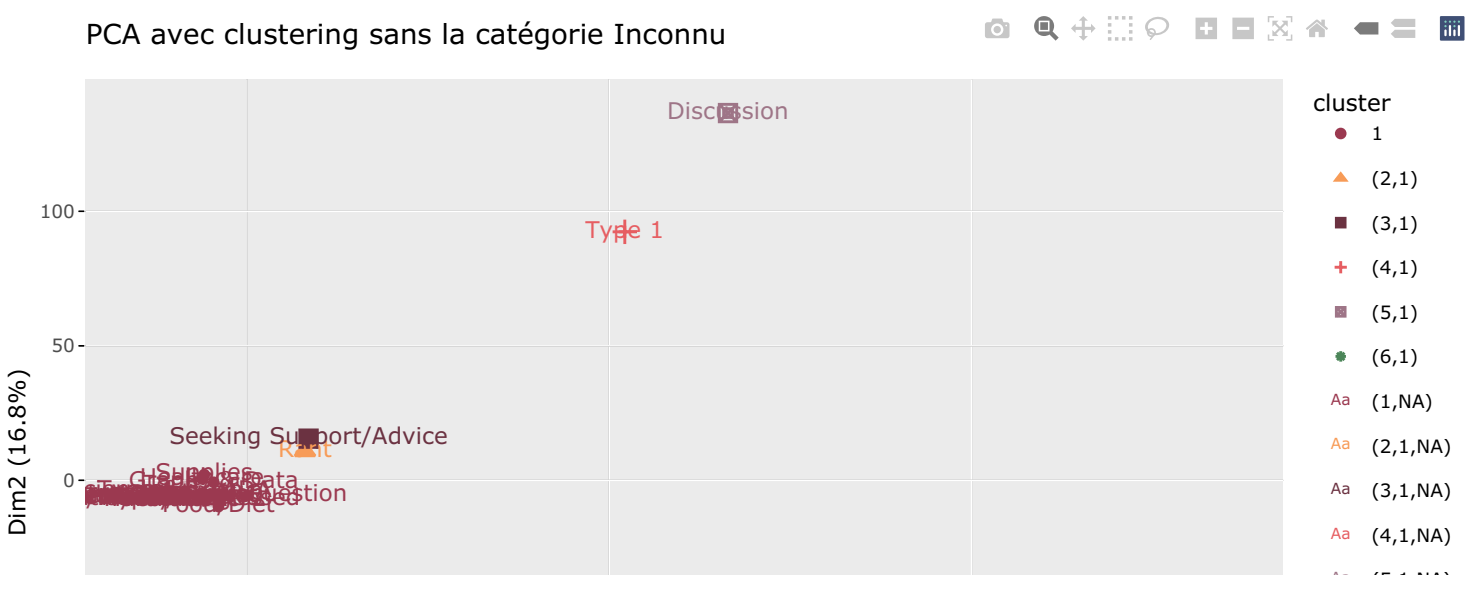
- Hide code -

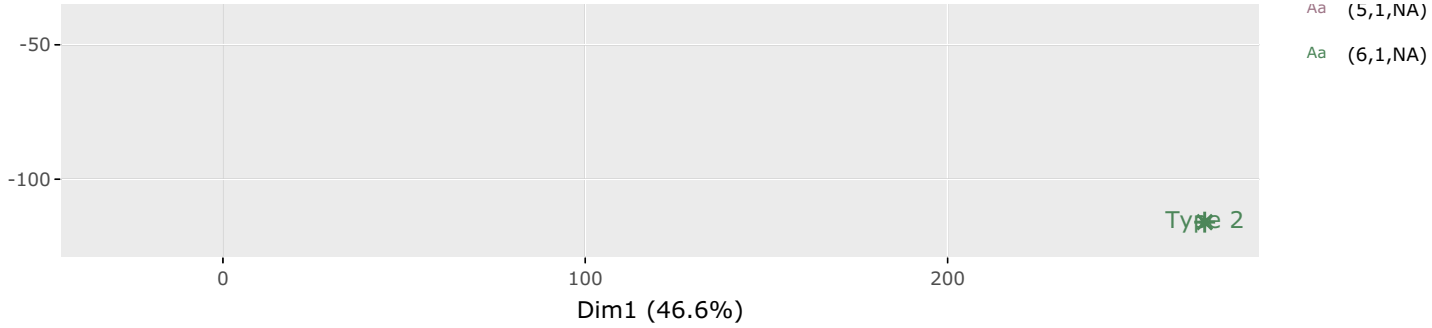
```
# Dendrogramme avec noms des catégories
fviz_dend(hcpc_result_2,
          cex = 0.8,                # Taille des étiquettes
          rect = TRUE,              # Ajouter des rectangles autour des clusters
          show_labels = TRUE,       # Afficher les étiquettes
          main = "Dendrogramme des catégories sans la catégorie Inconnu")
```



- Hide code -

```
p2 <- fviz_cluster(hcpc_result_2,
                   show.clust.cent = TRUE,
                   main = "PCA avec clustering sans la catégorie Inconnu",
                   axes = c(1, 2))
ggplotly(p2)
```




-50
-100
0
100
200
Dim1 (46.6%)
Type 2
Aa (5,1,NA)
Aa (6,1,NA)