TP4_Projet_Statistique_Sous_R

Groupe 1: Samba Dieng/ Onanena Jeanne De La Flèche /
Khadidiatou Coulibaly / Tamsir Ndong

2025-02-17

PLAN DE REDACTION

- **Introduction**
- **I°) Chargement des données**
- **II°) Nettoyage des chaînes de caractères**
- **III°) Recherche des correspondances exactes**
- **IV°) Gestion des non-correspondances**
- **V°) Défis rencontrés**
- **Conclusion et recommandations**

Introduction

Lorsqu'on fusionne deux bases de données, il est fréquent que les variables censées correspondre présentent des différences de formatage ou d'écriture (accents, espaces, apostrophes, etc.).

Dans ce projet, nous utilisons R pour harmoniser ces noms et vérifier les correspondances entre deux fichiers de données, tout en gérant les erreurs et les incohérences.

I.1°) Chargement des packages et des données

Packages utilisés

- **haven** : Permet d'importer des fichiers Stata (.dta).
- **readxl** : Permet de lire des fichiers Excel (.xlsx).
- **labelled** : Facilite la manipulation des variables factorisées.
- **stringi** : Fournit des fonctions pour le nettoyage et la manipulation de texte, notamment la suppression des accents et la transformation en majuscules.

I.2°) Importation des données

Les bases qui sont concernées sont : **HDX** et **EHCVM**.

```
library(haven)
library(readxl)
library(labelled)
library(stringi)  # Pour la gestion avancée des chaînes de cara

# Charger les données
# Base 1 : EHCVM (Stata)
data2 <- read_dta("ehcvm_individu_civ2021.dta")
data2_test <- data2  # Copie pour les tests

# Base 2 : HDX (Excel)
data_hdx <- read_excel("civ_adminboundaries_tabulardata.xlsx")
data_hdx_test <- data_hdx  # Copie pour les tests
```

II°) Nettoyage des noms des communes

Les noms de communes peuvent contenir des **accents**, des **apostrophes** et **d'autres caractères spéciaux**. Pour assurer une correspondance plus fiable, nous appliquons une transformation :

- Suppression des **accents** (stri_trans_general(text, "Latin-ASCII"))
- Conversion en **majuscules** (toupper(text)).
- Suppression des **apostrophes** (gsub("'", " ", text)).
- Suppression des **autres caractères spéciaux** (gsub("[^A-Z0-9]", " ", text)).

II.1°) Nettoyage des noms de communes dans “EHCVM”

Ce code en R ci dessous nettoie et standardise les noms de communes en supprimant les accents, en mettant les textes en majuscules et en retirant les caractères spéciaux indésirables. Il convertit d'abord la variable en facteur, extrait les niveaux uniques des communes, puis applique une fonction de nettoyage utilisant **stri_trans_general()**, **toupper()**, et **gsub()**. Cependant, une erreur est présente : def doit être remplacé par **function()** en R. De plus, **as.list(sapply(...))** est redondant et peut être simplifié en sapply(...).

```r
# Convertir la variable en facteur
data2_test$sp_commune <- to_factor(data2_test$sp_commune)

# Extraire les niveaux des communes
commune_ehcvm <- levels(data2_test$sp_commune)

# Fonction de nettoyage
def clean_text(text) {
  text <- stri_trans_general(text, "Latin-ASCII")  # Supprimer
  text <- toupper(text)  # Majuscules
  text <- gsub("'", "", text)  # Supprimer les apostrophes
  text <- gsub("[^A-Z0-9 ]", "", text)  # Supprimer autres car
  return(text)
}

# Appliquer le nettoyage
list_commune_ehcvm <- as.list(sapply(commune_ehcvm, clean_text
print(list_commune_ehcvm)  # Affichage des noms de communes ne
```

II.2°) Nettoyage des noms de communes dans la base HDX

```r
# Extraire les noms des communes
commune_hdx <- unique(data_hdx_test$ADM3_FR)

# Appliquer la transformation
commune_hdx_clean <- sapply(commune_hdx, clean_text)
print(commune_hdx_clean)  # Affichage des noms nettoyés
```

III°) Recherche des correspondances exactes On cherche d'abord les communes **présentes dans les deux bases** avec **intersect()**.

```r
# Trouver l'intersection des deux listes
communes_communes <- intersect(list_commune_ehcvm, commune_hdx_clean)

# Afficher le résultat
print(communes_communes)
```

Ensuite, on détecte les communes absentes dans la base HDX.

```
communes_uniques_ehcvm <- setdiff(list_commune_ehcvm, communes_
print(communes_uniques_ehcvm)  # Liste des communes non trouvées
```

Quand une correspondance exacte n'existe pas, on peut mesurer **la proximité entre les noms**. Nous avons les méthodes suivants:

- **Distance de Levenshtein** (stringdist::stringdist()) : Mesure le nombre d'opérations (insertion, suppression, substitution) pour transformer une chaîne en une autre.
- **Distance de Jaro-Winkler** (stringdist::stringdist()) : Plus adaptée aux erreurs typographiques et aux préfixes similaires.
- **Correspondance approximative avec** amatch() : Retourne l'indice de la meilleure correspondance.

1. Distance de Levenshtein

Principe

- Calcule le **nombre minimum d'opérations** nécessaires pour transformer une chaîne A en chaîne B.
- Les opérations sont :
  1. **Insertion** d'un caractère
  2. **Suppression** d'un caractère
  3. **Substitution** d'un caractère

Exemple

- Pour transformer `"Paris"` en `"Pariis"` :
  - On insère un "i" $\rightarrow$ 1 opération
  - Résultat : la **distance de Levenshtein** $= 1$
- Pour transformer `"Marseille"` en `"Marselle"` :
  - Il manque un "i" $\rightarrow$ 1 insertion
  - Distance $= 1$

Interprétation

- Distance **0** : les deux chaînes sont identiques.

2. Distance de Damerau-Levenshtein

Principe

- Semblable à la distance de Levenshtein, **mais** elle ajoute la possibilité de **transposer** deux lettres adjacentes comme une seule opération.
- Utile pour gérer les inversions de lettres très courantes (ex. “Marsiellle” vs “Marseille”).

Exemple

- Pour transformer `"mries"` en `"meris"`, on a besoin de permuter “r” et “i”.
  - Avec Damerau-Levenshtein, **une seule opération** suffit (transposition).
  - Avec Levenshtein classique, ce serait deux opérations (substitutions).

3. Distance de Hamming

Principe

- Compte le nombre de **positions différentes** entre deux chaînes de **même longueur**.
- **Aucune insertion ou suppression** n'est autorisée, seulement la **substitution** de caractères.

Exemple

- `"abcde"` vs. `"abzde"`
  - Seule la 3e lettre est différente ("c" vs. "z") $\rightarrow$ distance de Hamming $= 1$
- `"ab"` vs. `"ba"`
  - Les deux positions diffèrent $\rightarrow$ distance $= 2$

Limite

- Ne s'applique qu'aux chaînes de la même longueur.

4. Distances Jaro et Jaro-Winkler

Jaro

- **Compare** les positions des caractères entre deux chaînes et tient compte du nombre de **correspondances** et de **transpositions**.
- Produit une **similarité** entre 0 et 1 (1 = identique, 0 = très différent).
- Dans certains outils (ex. `stringdist`), on obtient la **distance** = `1 - similarité`.

Jaro-Winkler

- **Extension** de Jaro qui accorde **plus de poids** aux **premiers caractères** identiques.
- Particulièrement adapté pour les noms (personnes, communes...).

Exemple

- "MARTHA" vs. "MARHTA"
  - Très proches pour Jaro/Jaro-Winkler : mêmes lettres, ordre

5. Distance ou similarité de Jaccard

Principe

- On voit chaque chaîne comme un **ensemble** de caractères (ou de mots).
- La **similarité de Jaccard** =

$$\frac{\text{taille de l'intersection}}{\text{taille de l'union}}$$

- La distance de Jaccard = `1 - (similarité de Jaccard)`.

Exemple

- Chaîne A : “paris” → `{p, a, r, i, s}`
- Chaîne B : “mars” → `{m, a, r, s}`
  - Intersection : `{a, r, s}` → taille = 3
  - Union : `{p, a, r, i, s, m}` → taille = 6
  - Similarité de Jaccard = 3/6 = 0,5 → Distance = 0,5

6. Similarité cosinus (Cosine similarity)

Principe

- Conçu plutôt pour comparer des **phrases** ou **textes** plus longs.
- Transforme chaque texte en un **vecteur** (par exemple de fréquences de mots).
- Calcule l'**angle** entre ces vecteurs : plus l'angle est petit, plus la similarité est grande.

Exemple

- On construit un vecteur de la forme `[nb_occurrences_mot1, nb_occurrences_mot2, ...]`.
- On calcule

$$\text{similarité\_cosinus} = \frac{A \cdot B}{\|A\|\|B\|}$$

- Très utilisé pour la comparaison de documents.

7. Méthode phonétique (ex. Soundex, Metaphone)

Principe

- Compare la **prononciation** au lieu de l'orthographe exacte.
- Transforme chaque mot en un **code** (Soundex, Metaphone, etc.).
- Deux mots qui se prononcent de façon similaire obtiendront des **codes proches** ou identiques.

Exemple

- "Marseille" vs "Marsay"
  - On génère un code Soundex pour chacun (ex. "M624"...).
  - Si les codes sont identiques ou très semblables, on juge les mots similaires phonétiquement.

Avantages et limites

- **Avantage** : Gère les erreurs où la prononciation reste proche malgré une orthographe fautive.
- **Limite** : Peut confondre des homonymes.

V°) Les défis rencontrés

- **CAS DU TOGO ET DE LA GUINEE**

Pour ces pays, le principal problème a été que la variable du niveau admin 3 n'était pas disponible. - Elle ne contient que des NA pour le TOGO dans la base EHCVM II (variable canton). - Elle n'est présente ni dans la base EHCVM II, ni dans celle de HDX pour la Guinée. Nous sommes donc convenus de travailler avec le niveau 2 (préfectures). Après les traitements préliminaires (espaces, majuscules, caractères spéciaux, prise en compte des différents noms), il y avait, dans la base EHCVM II, 7 préfectures qui trouvaient pas de « match » dans la base HDX pour le TOGO, et 11 pour la Guinée.

V°) Les défis rencontrés

Le traitement avec les distances de Levenshtein, Jaro-Winker, Qgam (nombre de sous-chaînes semblables) et la m éthode phonique a donné les tableaux suivant. Pour la Guinée, c'est la distance de Jaro-Winker qui a donné tous les bonnes correspondances. Pour le Togo, c'est seulement celle de Levenshtein qui a donné quelques bons résultats.

- **CAS DE LA COTE D'IVOIRE** Afin de récupérer les codes des communes de la côte d'ivoire en utilisant les données de l'EHCVM 2021 et les fichiers issus de HDX, nous avons transformé tous les noms des communes en majuscules et traités tous les caractères spéciaux puis nous avons fait un premier merge. A l'issue de ce merge, nous avons eu 28 no matching que nous avons traité manuellement.

Conclusion et recommandations

- Le nettoyage des caractères est une étape cruciale pour assurer une correspondance fiable.
- La fusion des bases peut être améliorée avec des méthodes de distance pour traiter les cas où les noms ne correspondent pas exactement.
- Pour un projet plus robuste, on pourrait pondérer les scores de similarité en fonction d'autres caractéristiques (région, code postal, etc.). En appliquant ces méthodes, on réduit les erreurs et on améliore la qualité des fusions de bases de données en R.