Nettoyage de données II : corriger les incohérences

Introduction

Dans cette leçon, nous allons explorer l'essentielle compétence du nettoyage des données, la prochaine étape cruciale pour raffiner vos jeux de données en vue d'une analyse robuste. Alors que nous passons de l'analyse exploratoire des données, où nous avons identifié des problèmes potentiels, nous nous concentrons maintenant sur la résolution systématique de ces problèmes. Commençons !

Objectifs d'apprentissage

- Vous savez comment nettoyer automatiquement et manuellement les noms de colonnes
- Vous pouvez facilement supprimer les colonnes et lignes vides
- Vous êtes capable de supprimer les ligne en double
- Vous savez comment corriger les valeurs en caractère de chaîne
- Vous pouvez changer les types de données

Packages

Les packages ci-dessous seront nécessaires pour cette leçon :

```
# Charger les packages
if(!require(pacman)) install.packages("pacman")
pacman::p_load(tidyverse,
               janitor,
               inspectdf,
               skimr,
               dplyr)
```

Jeu de données

Le jeu de données que nous utiliserons pour cette leçon est une version légèrement modifiée du jeu de données utilisé dans la première leçon `Nettoyage des données` ; ici, nous avons ajouté un peu plus d'erreurs à nettoyer ! Consultez la leçon 1 pour une explication de ce jeu de données.

```
non_adherence <- read_csv(here("data/non_adherence_desordre.csv"))
```

```
non_adherence
```

```
## # A tibble: 5 × 15
##   id_patient District `Unite de sante` Sexe  Age_35
##        <dbl>    <dbl>            <dbl> <chr> <chr>
## 1      10037        1                1 Homme plus de 35 ans
## 2      10537        1                1 F     plus de 35 ans
## 3       5489        2                3 F     Moins de 35 ans
## 4       5523        2                3 Homme Moins de 35 ans
## 5       4942        2                3 F     plus de 35 ans
## # ℹ 10 more variables: `Age a l'initiation du ARV` <dbl>,
## #   EDUCATION_DU_PATIENT <chr>, …
```

Nettoyage des noms de colonnes

Noms de colonnes

En règle générale, les noms de colonnes doivent avoir une syntaxe “propre” et standardisée afin que nous puissions facilement travailler avec eux et que notre code soit lisible par d'autres codeurs.

Idéalement, les noms de colonnes :

- devraient être courts
- ne devraient pas contenir d'espaces ou de points (remplacer les espaces et les points par des tirets bas “_”)
- ne devraient pas contenir de caractères inhabituels (&, #, <, >)
- devraient avoir un style similaire

Pour voir nos noms de colonnes, nous pouvons utiliser la fonction `names()` de base R.

```
names(non_adherence)
```

```
##  [1] "id_patient"                    "District"
##  [3] "Unite de sante"                "Sexe"
##  [5] "Age_35"                        "Age a l'initiation du ARV"
##  [7] "EDUCATION_DU_PATIENT"          "OCCUPATION_DU_PATIENT"
##  [9] "Etat...civil"                  "Statut OMS a l'initiation du ARV"
## [11] "IMC_initiation_ARV"            "CD4_initiation_ARV"
## [13] "regime.1"                      "Nmbr_comprimes_jour"
## [15] "NA"
```

Ici, nous pouvons voir que :

- certains noms contiennent des espaces
- certains noms contiennent des caractères spéciaux comme `...`
- certains noms sont en majuscules alors que d'autres non

Nettoyage automatique

Une fonction pratique pour standardiser les noms de colonnes est `clean_names()` du package `janitor`.

La fonction `clean_names()` :

- Convertit tous les noms pour qu'ils ne contiennent que des underscores, des chiffres et des lettres.
- Analyse les casse et séparateurs selon un format cohérent. (par défaut snake_case)
- Gère les caractères spéciaux (&, #, <, >) ou les caractères accentués.

```
non_adherence %>%
  clean_names() %>%
  names()
```

```
##  [1] "id_patient"                    "district"
##  [3] "unite_de_sante"                "sexe"
##  [5] "age_35"                        "age_a_linitiation_du_arv"
##  [7] "education_du_patient"          "occupation_du_patient"
##  [9] "etat_civil"                    "statut_oms_a_linitiation_du_arv"
## [11] "imc_initiation_arv"            "cd4_initiation_arv"
## [13] "regime_1"                      "nmbr_comprimes_jour"
## [15] "na"
```

D'après cette sortie, nous pouvons voir que :

- les noms de variables en majuscules ont été convertis en minuscules (par ex. `EDUCATION_DU_PATIENT` est devenu `education_du_patient`)
- les espaces dans les noms de variables ont été convertis en underscores (par ex. `Age a l'initiation du ARV` est devenu `age_a_linitiation_du_arv`)
- les points (`.`) ont tous été remplacés par des underscores (par ex. `Etat...civil` est devenu `etat_civil`)

** Q: Nettoyage automatique **

L'ensemble de données suivant a été adapté d'une étude qui a utilisé des données rétrospectives pour caractériser les dynamiques temporelles et spatiales des épidémies de fièvre typhoïde à Kasene, Ouganda.

```
typhoide <- read_csv(here("data/typhoid_uganda.csv"))
```

```
## Rows: 215 Columns: 31
## ── Column specification
──────────────────────────────────────────────────────────
## Delimiter: ","
## chr (18): Householdmembers, Positioninthehousehold,
Watersourcedwithinhouseh...
## dbl (11): UniqueKey, CaseorControl, Age, Sex, Levelofeducation,
Below10years...
## lgl  (2): NA, NAN
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this
message.
```

Utilisez la fonction `clean_names()` de `janitor` pour nettoyer les noms de variables dans le jeu de données `typhoide`.

Nettoyage automatique puis manuel des noms de colonnes

Parfois, le nettoyage automatique seul ne suffit pas pour s'assurer que nos noms de colonnes sont propres et ont un sens pour notre jeu de données. Une combinaison de nettoyage automatique et manuel peut être nécessaire. Nous pouvons commencer par le nettoyage automatique, puis vérifier s'il reste des noms de colonnes étranges qui nécessiteraient un nettoyage manuel.

Voici un exemple combinant le nettoyage automatique et manuel : un nettoyage plus approprié serait de standardiser d'abord les noms de colonnes, racourcir les noms de certain variables, puis de supprimer la fin `_du_patient` de toutes les colonnes. Créons un nouveau dataframe appelé `non_adh_col_propre`.

```
non_adh_col_propre <- non_adherence %>%
          # standardiser la syntaxe des noms de colonnes
          clean_names() %>%
          # racourcir manullement les noms de deux variables
          rename("age_initiation_arv" = "age_a_linitiation_du_arv",
                 "statut_oms_initiation_arv" =
"statut_oms_a_linitiation_du_arv") %>%
          # enlever manuellement "_de_patient" de toutes les colonnes
          rename_with(str_replace, pattern = "_du_patient", replacement = "")
non_adh_col_propre
```

```
## # A tibble: 5 × 15
##   id_patient district unite_de_sante sexe  age_35
##        <dbl>    <dbl>          <dbl> <chr> <chr>
## 1      10037        1              1 Homme plus de 35 ans
## 2      10537        1              1 F     plus de 35 ans
## 3       5489        2              3 F     Moins de 35 ans
## 4       5523        2              3 Homme Moins de 35 ans
## 5       4942        2              3 F     plus de 35 ans
```

```
## # ℹ 10 more variables: age_initiation_arv <dbl>,
## #   education <chr>, occupation <chr>, etat_civil <chr>, …
```

Cela semble parfait ! Nous utiliserons ce jeu de données avec les colonnes propres dans la prochaine section.

Q: Nettoyage complet des noms de colonnes

Standardisez les noms de colonnes dans le jeu de données `typhoide` puis:

- remplacez `or_` par _
- remplacez `of` par _
- renommez les variables `below10years` `n1119years` `n2035years` `n3644years`, `n4565years` `above65years` en `num_below_10_yrs` `num_11_19_yrs` `num_20_35_yrs` `num_36_44_yrs`, `num_45_65_yrs` `num_above_65_yrs`

Supprimer des colonnes et lignes vides

Une ligne/colonne **VIDE** est une ligne/colonne où toutes les valeurs sont NA. Lorsque vous chargez un jeu de données, vous devez toujours vérifier s'il contient des lignes ou colonnes vides et les supprimer. Le but est que **chaque ligne soit un point de données significatif** et que **chaque colonne soit une variable significative**. Commençons par nos colonnes.

Suppression des colonnes vides

Pour identifier les colonnes vides, nous allons utiliser la fonction `inspect_na()` du package `inspectdf` pour identifier les colonnes vides.

```
inspectdf::inspect_na(non_adh_col_propre)
```

D'après la sortie, nous voyons que le `pcnt` indique 100% de vide (c'est-à-dire valeurs NA) pour notre colonne `na` : il y a une valeur NA dans chaque ligne.

Pour supprimer les colonnes vides du dataframe, nous utiliserons la fonction `remove_empty()` du package `janitor`. Cette fonction supprime toutes les colonnes d'un dataframe composées entièrement de valeurs NA.

Nous appliquerons la fonction sur le jeu de données `non_adh_clean_names` et supprimerons la colonne vide identifiée précédemment.

```
ncol(non_adh_col_propre)
```

```
## [1] 15
```

```
non_adh_col_propre <- non_adh_col_propre %>%
  remove_empty("cols")

ncol(non_adh_col_propre)
```

```
## [1] 14
```

Ici, nous pouvons voir que la colonne na a été supprimée des données.

Q: Supprimer des colonnes vides

Supprimez les colonnes vides du jeu de données `typhoide`.

Suppression des lignes vides

Bien qu'il soit relativement facile d'identifier les colonnes vides à partir de la sortie `skim()`, ce n'est pas aussi facile pour les lignes vides. Heureusement, la fonction `remove_empty()` fonctionne également s'il y a des lignes vides dans les données. Le seul changement dans la syntaxe est de spécifier `"rows"` au lieu de `"cols"`.

```
nrow(non_adh_col_propre)
```

```
## [1] 1420
```

```
non_adh_col_propre <- non_adh_col_propre %>%
  remove_empty("rows")

nrow(non_adh_col_propre)
```

```
## [1] 1417
```

Le nombre de lignes est passé de 1420 à 1417, ce qui suggère qu'il y avait des lignes vides dans les données qui ont été supprimées.

Q: Supprimer des colonnes et lignes vides

Supprimez à la fois les lignes et colonnes vides du jeu de données `typhoide`.

Suppression des lignes en double

Très souvent dans vos jeux de données, il y a des situations où vous avez des valeurs en double, lorsqu'une ligne a exactement les mêmes valeurs qu'une autre ligne. Cela peut se produire lorsque vous combinez des données de sources multiples ou lorsque vous avez reçu plusieurs réponses à un sondage.

Il est donc nécessaire d'identifier et de supprimer toutes les valeurs en double de vos données afin de garantir des résultats précis. Pour cela, nous utiliserons deux fonctions : `janitor::get_dupes()` et `dplyr::distinct()`.

`janitor::get_dupes()`

Une façon simple de vérifier rapidement les lignes qui ont des doublons est la fonction `get_dupes()` du package `janitor`.

La syntaxe est `get_dupes(x)`, où x est **un dataframe**.

Essayons-le sur notre dataframe `non_adh_clean_names`.

```
non_adh_col_propre %>%
  get_dupes()
```

```
## No variable names specified - using all columns.
```

La sortie est composée de 8 lignes : il y a 2 lignes pour chaque paire de doublons. Vous pouvez facilement voir que ce sont des doublons d'après la variable `patient_id` qui identifie de façon unique nos observations.

Q: Supprimer des lignes en double

Identifiez les éléments qui sont des doublons dans le jeu de données `typhoide`.

La fonction `get_dupes()` est utile pour extraire vos données en double avant de les supprimer. Parfois, si vous avez beaucoup de doublons, cela peut indiquer un problème qui nécessite une enquête plus approfondie, comme un problème de collecte de données ou de fusion. Une fois que vous avez examiné vos doublons, vous pouvez les supprimer avec la fonction suivante.

`dplyr::distinct()`

`distinct()` est une fonction du package `dplyr` qui ne conserve que les lignes uniques/distinctes d'un dataframe. S'il y a des lignes en double, seule la première ligne est conservée.

Vérifions le nombre de lignes avant et après avoir appliqué la fonction `distinct()`.

```
nrow(non_adh_col_propre)
```

```
## [1] 1417
```

```
non_adh_distinct <- non_adh_col_propre %>%
  distinct()

nrow(non_adh_distinct)
```

```
## [1] 1413
```

Initialement, notre jeu de données avait 1417 lignes. L'application de la fonction `distinct()` réduit les dimensions du jeu de données à 1413 lignes. Cela fait sens, car comme nous l'avons vu précédemment, 4 de nos valeurs étaient des doublons.

Q: Affichage des décomptes

Supprimez les lignes en double du jeu de données `typhoide`. Assurez-vous que seules des lignes uniques restent dans le jeu de données.

Transformations

Correction des chaînes de caractères

Il y a souvent des moments où vous devez corriger certaines incohérences dans les chaînes de caractères qui pourraient interférer avec l'analyse des données, y compris les fautes d'orthographe et les erreurs de casse.

Ces problèmes peuvent être corrigés manuellement dans la source de données brute ou nous pouvons apporter la modification dans le pipeline de nettoyage. Cette dernière option est plus transparente et reproductible pour toute personne cherchant à comprendre ou à répéter votre analyse.

Changer les valeurs de chaînes avec `case_match()`

Nous pouvons utiliser la fonction `case_match()` dans la fonction `mutate()` pour changer des valeurs spécifiques et pour concilier des valeurs orthographiées différemment.

La syntaxe est `case_match(nom_colonne, ancienne_valeur_colonne ~ nouvelle_valeur_colonne)`

Tout d'abord, regardons la colonne `sex` :

```
non_adh_distinct %>% count(sexe)
```

```
## # A tibble: 2 × 2
##   sexe      n
##   <chr> <int>
## 1 F      1084
## 2 Homme   329
```

Dans cette variable, nous pouvons voir qu'il y a des incohérences dans la façon dont les niveaux ont été codés. Utilisons la fonction `case_match()` pour que `F` soit changé en `Female`.

```
non_adh_distinct %>%
  mutate(sexe = case_match(sexe, "F" ~ "Femme", .default=sexe)) %>%
  count(sexe)
```

```
## # A tibble: 2 × 2
##   sexe      n
##   <chr> <int>
## 1 Femme  1084
## 2 Homme   329
```

Cela semble parfait ! `case_match()` est un moyen facile de corriger les valeurs qui ne correspondent pas dans votre jeu de données. Parfois, nous avons des modèles plus complexes à corriger, comme le recodage conditionnel et la suppression de caractères spéciaux. Pour plus d'informations sur la gestion des chaînes de caractères, consultez la leçon sur les chaînes de caractères !

Si vous ne spécifiez pas l'argument `.default=nom_colonne`, toutes les valeurs de cette colonne qui ne correspondent pas à celles que vous changez et que vous mentionnez explicitement dans votre fonction

`case_match()` seront renvoyées comme NA. Dans le cas ci-dessus, cela signifie que tous les `Males` auraient été définis comme manquants.

Q: Corriger les chaînes de caractères

La variable `householdmembers` du jeu de données `typhoide` devrait représenter le nombre de personnes dans un ménage. Affichez les différentes valeurs de la variable.

Transformer en miniscule

Il y a une valeur `01–May` dans la variable `householdmembers` du jeu de données `typhoide`. Recodez cette valeur en 1–5.

Homogénéiser toutes les chaînes dans l'ensemble du jeu de données

Vous vous souviendrez peut-être qu'avec la sortie `skim` de la première leçon sur le pipeline de nettoyage des données, nous avions des cas où nos caractères de chaînes n'étaient pas cohérents en ce qui concerne la casse. Par exemple, pour notre variable `occupation`, nous avions à la fois `Professor` et `professor`.

Pour résoudre ce problème, nous pouvons transformer toutes nos chaînes en minuscules à l'aide de la fonction `tolower()`. Nous sélectionnons toutes les colonnes de type caractère avec `where(is.character)`

```
non_adh_distinct %>% count(occupation)
```

```
non_adh_distinct %>%
  mutate(across(where(is.character),
                ~ tolower(.x))) %>%
  count(occupation)
```

Comme nous pouvons le constater, nous sommes passés de 51 à 49 niveaux uniques pour la variable `occupation` lorsque nous transformons tout en minuscules !

Transformez toutes les chaînes de caractères du jeu de données `typhoide` en minuscules.

Nettoyage des types de données

Les colonnes contenant des valeurs qui sont des nombres, des facteurs ou des valeurs logiques (TRUE/FALSE) se comportent comme prévu uniquement si elles sont correctement classées. En tant que tel, vous devrez peut-être redéfinir le type ou la classe de votre variable.

KEY POINT

KEY POINT

R a 6 types de données de base/classes.

- `character` : chaînes ou caractères individuels, entre guillemets
- `numeric` : n'importe quel nombre réel (comprend les décimales)
- `integer` : tout(s) nombre(s) entier(s)
- `logical` : variables composées de TRUE ou FALSE
- `factor` : variables catégorielles/qualitatives
- `Date/POSIXct` : représente les dates et heures du calendrier

En plus de ceux répertoriés ci-dessus, il y a aussi `raw` qui ne sera pas abordé dans cette leçon.

Vous vous souvenez peut-être que dans la dernière leçon, notre jeu de données contenait 5 variables de type caractère et 9 variables numériques (ainsi qu'une variable logique NA qui a depuis été supprimée car elle était complètement vide). Jetons un coup d'œil rapide à nos variables à l'aide de la fonction `skim()` de la dernière leçon :

```
  skim(non_adh_distinct) %>%
  select(skim_type) %>%
  count(skim_type)
```

Dans la dernière leçon, on a vu que toutes nos variables sont catégorielles, sauf `age_initiation_arv`, `imc_initiation_arv`, `cd4_initiation_arv` et `nmbr_comprimes_jour`. Transformons toutes les autres en variables factorielles à l'aide de la fonction `as.factor()` !

```
non_adh_distinct %>%
  mutate(across(!c(age_initiation_arv,
                   imc_initiation_arv,
                   cd4_initiation_arv,
                   nmbr_comprimes_jour),
                ~ as.factor(.x))) %>%
  skim() %>%
  select(skim_type) %>%
  count(skim_type)
```

SIDE NOTE

SIDE NOTE

pour indiquer où les variables fournies dans `across()` sont utilisées.

Q: Changer les types de données

Convertissez les variables aux positions 13 à 29 dans le jeu de données `typhoide` en facteurs.

Parfait, c'est exactement ce que nous voulions !

Mettre tout ensemble

Maintenant, appliquons ensemble les transformations de chaînes de caractères et de types de données !

```
non_adherence_final <- non_adh_distinct %>%
  # Recodage des valeurs de sexe
  mutate(sexe = case_match(sexe, "F" ~ "Femme", .default=sexe)) %>%
  # Tout en minuscules
  mutate(across(where(is.character),
                ~ tolower(.x))) %>%
  # Changement des types de variables
    mutate(across(!c(age_initiation_arv,
                     imc_initiation_arv,
                     cd4_initiation_arv,
                     nmbr_comprimes_jour),
                  ~ as.factor(.x)))

non_adherence_final
```

En Résumé

Félicitations pour avoir terminé cette leçon en deux parties sur le pipeline de nettoyage des données ! Vous êtes désormais mieux équipé pour relever les complexités des jeux de données réels. N'oubliez pas, le nettoyage des données ne consiste pas seulement à arranger des données en désordre ; il s'agit de garantir la fiabilité et la précision de vos analyses. En maîtrisant des techniques telles que la gestion des noms de colonnes, l'élimination des entrées vides, le traitement des doublons, l'affinage des valeurs de chaînes et la gestion des types de données, vous avez perfectionné vos capacités à transformer des données de santé brutes en une base propre pour des informations significatives !

Answer Key

Q: Nettoyage automatique

```
clean_names(typhoide)
```

Q: Nettoyage complet des noms de colonnes

```
typhoide %>%
  clean_names %>%
  rename("num_below_10_yrs" = "below10years",
         "num_11_19_yrs" = "n1119years",
         "num_20_35_yrs" = "n2035years",
         "num_36_44_yrs" = "n3644years",
         "num_45_65_yrs" = "n4565years",
         "num_above_65_yrs" = "above65years") %>%
  rename_with(str_replace_all, pattern = "or_",replacement = "_") %>%
  rename_with(str_replace_all, pattern = "of",replacement = "_") %>%
  names()
```

```
##  [1] "unique_key"                  "case_control"
##  [3] "age"                         "sex"
##  [5] "level_education"             "householdmembers"
##  [7] "num_below_10_yrs"            "num_11_19_yrs"
##  [9] "num_20_35_yrs"               "num_36_44_yrs"
## [11] "num_45_65_yrs"               "num_above_65_yrs"
## [13] "positioninthehousehold"      "watersourcedwithinhousehold"
## [15] "borehole"                    "river"
## [17] "tap"                         "rainwatertank"
## [19] "unprotectedspring"           "protectedspring"
## [21] "pond"                        "shallowwell"
## [23] "stream"                      "jerrycan"
## [25] "bucket"                      "county"
## [27] "subcounty"                   "parish"
## [29] "village"                     "na"
## [31] "nan"
```

Q: Supprimer des colonnes vides

```
typhoide %>%
  remove_empty("cols")
```

Q: Supprimer des colonnes et lignes vides

```
typhoide %>%
  remove_empty("cols") %>%
  remove_empty("rows")
```

Q: Supprimer des lignes en double

```
# Identify duplicates
get_dupes(typhoide)
```

```
## No variable names specified - using all columns.
```

```
# Remove duplicates
typhoide_distinct <- typhoide %>%
  distinct()

# Ensure all distinct rows left
get_dupes(typhoide_distinct)
```

```
## No variable names specified - using all columns.
```

```
## No duplicate combinations found of: UniqueKey, CaseorControl, Age, Sex,
Levelofeducation, Householdmembers, Below10years, N1119years, N2035years, ...
and 22 other variables
```

Q: Affichage des décomptes

```
typhoide %>%
  count(Householdmembers)
```

Q: Corriger les chaînes de caractères

```
typhoide %>%
  mutate(Householdmembers = case_match(Householdmembers, "01-May" ~ "1-5",
.default=Householdmembers)) %>%
  count(Householdmembers)
```

Q: Transformer en miniscule

```
typhoide %>%
  mutate(across(where(is.character),
                ~ tolower(.x)))
```

Q: Changer les types de données

```
typhoide %>%
  mutate(across(13:29, ~as.factor(.)))
```

Contributors

The following team members contributed to this lesson:

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

AMANDA MCKINLEY

R Developer and Instructor, the GRAPH Network