# Data Cleaning 1: Exploratory Data Analysis

## Introduction

Le nettoyage des données est le processus qui consiste à transformer des données brutes et "désordonnées" en des données fiables pouvant être analysées correctement. Cela implique d'identifier les points de données **inexacts**, **incomplets** ou **improbables** et de résoudre les incohérences ou les erreurs dans les données, ainsi que de renommer les noms de variables pour les rendre plus clairs et simples à manipuler.

Les tâches de nettoyage de données peuvent souvent être fastidieuses. Une blague courante chez les analystes de données dit : "80% de la science des données consiste à nettoyer les données et les 20% restants à se plaindre du nettoyage des données". Mais le nettoyage des données est une étape essentielle du processus d'analyse des données. Un peu de nettoyage au début du processus d'analyse des données contribuera grandement à améliorer la qualité de vos analyses et la facilité avec laquelle ces analyses peuvent être effectuées. Et une gamme de packages et de fonctions dans R peut enormoement simplifier le processus de nettoyage des données.

Dans cette leçon, nous allons commencer à examiner un pipeline de nettoyage de données typique dans R. Les étapes de nettoyage couvertes ici ne correspondent probablement pas exactement à ce dont vous aurez besoin pour vos propres données, mais elles constituent certainement un bon point de départ.

Commençons !

---

Voici la traduction en français des sections suivantes :

## Objectifs d'apprentissage

- Vous pouvez énumérer les opérations typiques impliquées dans le processus de nettoyage des données
- Vous pouvez diagnostiquer les problèmes de jeux de données qui nécessitent un nettoyage des données grâce à des fonctions telles que :
    - `visdat::vis_dat()`
    - `skimr::skim()`
    - `inspectdf::inspect_cat()`
    - `inspectdf::inspect_num()`
    - `DataExplorer::create_report()`

---

## Packages

Les packages chargés ci-dessous seront nécessaires pour cette leçon :

```
if(!require("pacman")) install.packages("pacman")
pacman::p_load(visdat,
               skimr,
               inspectdf,
               DataExplorer,
               tidyverse)
```

---

## Jeu de données

Le jeu de données principal que nous utiliserons dans cette leçon provient d'une étude menée dans trois centres de santé de Zambezia, au Mozambique. L'étude a examiné les facteurs individuels associés au délai avant le non-respect des services de soins et de traitement du VIH. Pour les besoins de cette leçon, nous ne regarderons qu'un sous-ensemble modifié de l'ensemble de données complet.

L'ensemble de données complet peut être obtenu auprès de Zenodo, et l'article peut être consulté ici.

Jetons un coup d'œil à cet ensemble de données :

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

```
non_adherence
```

```
## # A tibble: 5 × 15
##   patient_id District `Health unit` Sex   Age_35
##        <dbl>    <dbl>         <dbl> <chr> <chr>
## 1      10037        1             1 Male  over 35
## 2      10537        1             1 F     over 35
## 3       5489        2             3 F     Under 35
## 4       5523        2             3 Male  Under 35
## 5       4942        2             3 F     over 35
## # ℹ 10 more variables: `Age at ART initiation` <dbl>,
## #   Education <chr>, Occupation <chr>, …
```

La première étape du nettoyage des données consistera à explorer cet ensemble de données afin d'identifier les problèmes potentiels qui nécessitent un nettoyage. Cette étape préliminaire est parfois appelée "analyse exploratoire des données" ou AED.

Regardons quelques commandes AED simples dans R qui vous aideront à identifier les erreurs et incohérences de données possibles.

---

## Visualisation des données manquantes avec `visdat::vis_dat()`

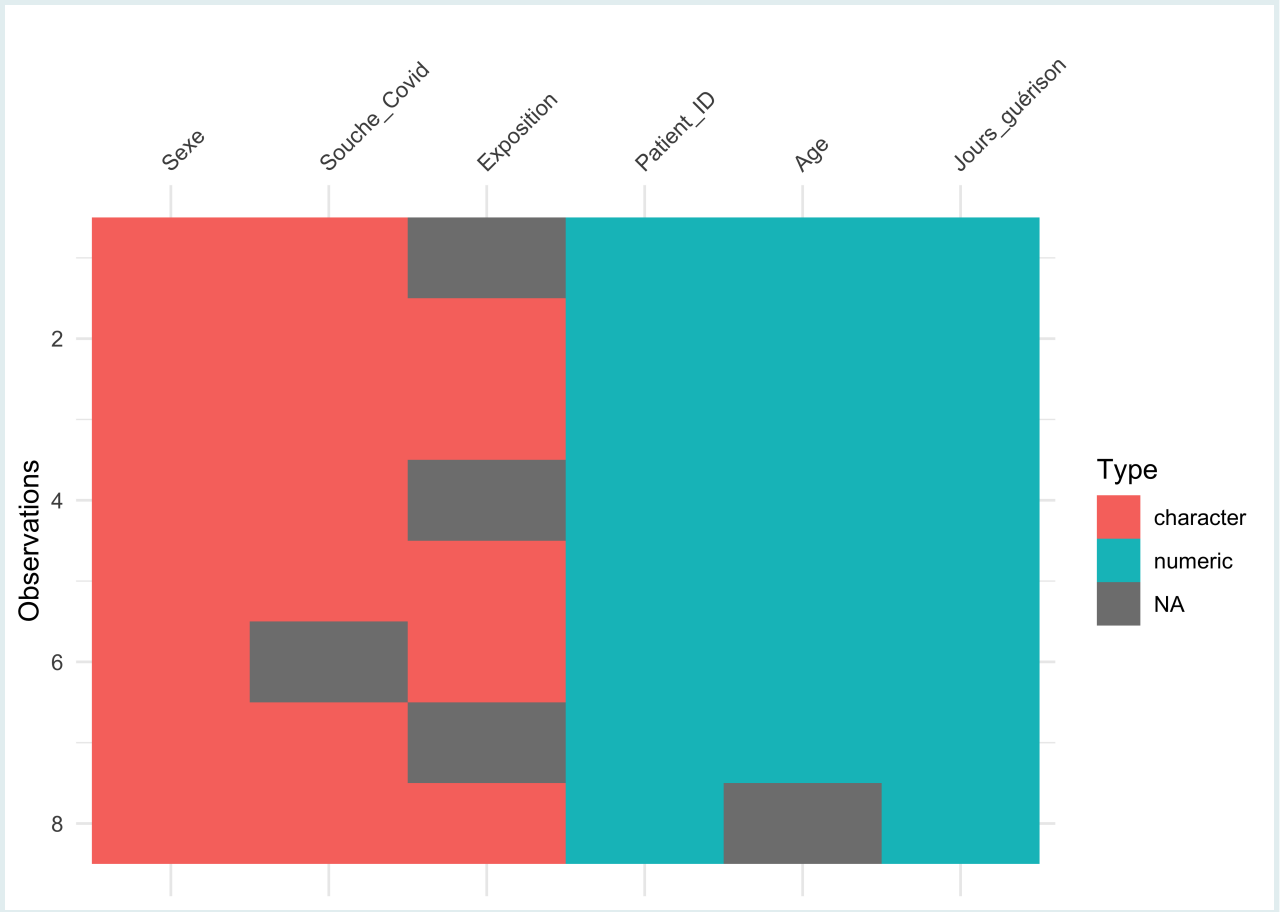
La fonction `vis_dat()` du package `visdat` est un excellent moyen de visualiser rapidement les types de données et les valeurs manquantes dans un jeu de données. Elle crée un graphique qui montre une vue "zoomée" de votre dataframe : chaque ligne du dataframe est représentée par une seule ligne sur le graphique.

Essayons d'abord avec un petit jeu de données fictif pour comprendre son fonctionnement. Copiez le code suivant pour créer un dataframe de 8 patients et leurs informations de diagnostic et de guérison du COVID-19. Comme vous pouvez le voir ci-dessous, certaines informations sont manquantes pour certains patients, représentées par NA.

```
        covid_pat <- tribble(
          ~Patient_ID, ~Age, ~Sexe, ~Souche_Covid, ~Exposition,
~Jours_guérison,
           1, 25, "Homme", "Alpha", NA, 10,
           2, 32, "Femme", "Delta", "Hôpital", 15,
           3, 45, "Homme", "Beta", "Voyage", 7,
           4, 19, "Femme", "Omicron", NA, 21,
           5, 38, "Homme", "Alpha", "Inconnu", 14,
           6, 55, "Femme", NA, "Communauté", 19,
           7, 28, "Femme", "Omicron", NA, 8,
           8, NA, "Femme", "Omicron", "Voyage", 26
        )
        covid_pat
```

Maintenant, utilisons la fonction `vis_dat()` sur notre dataframe pour obtenir une représentation visuelle des types de données et des valeurs manquantes.

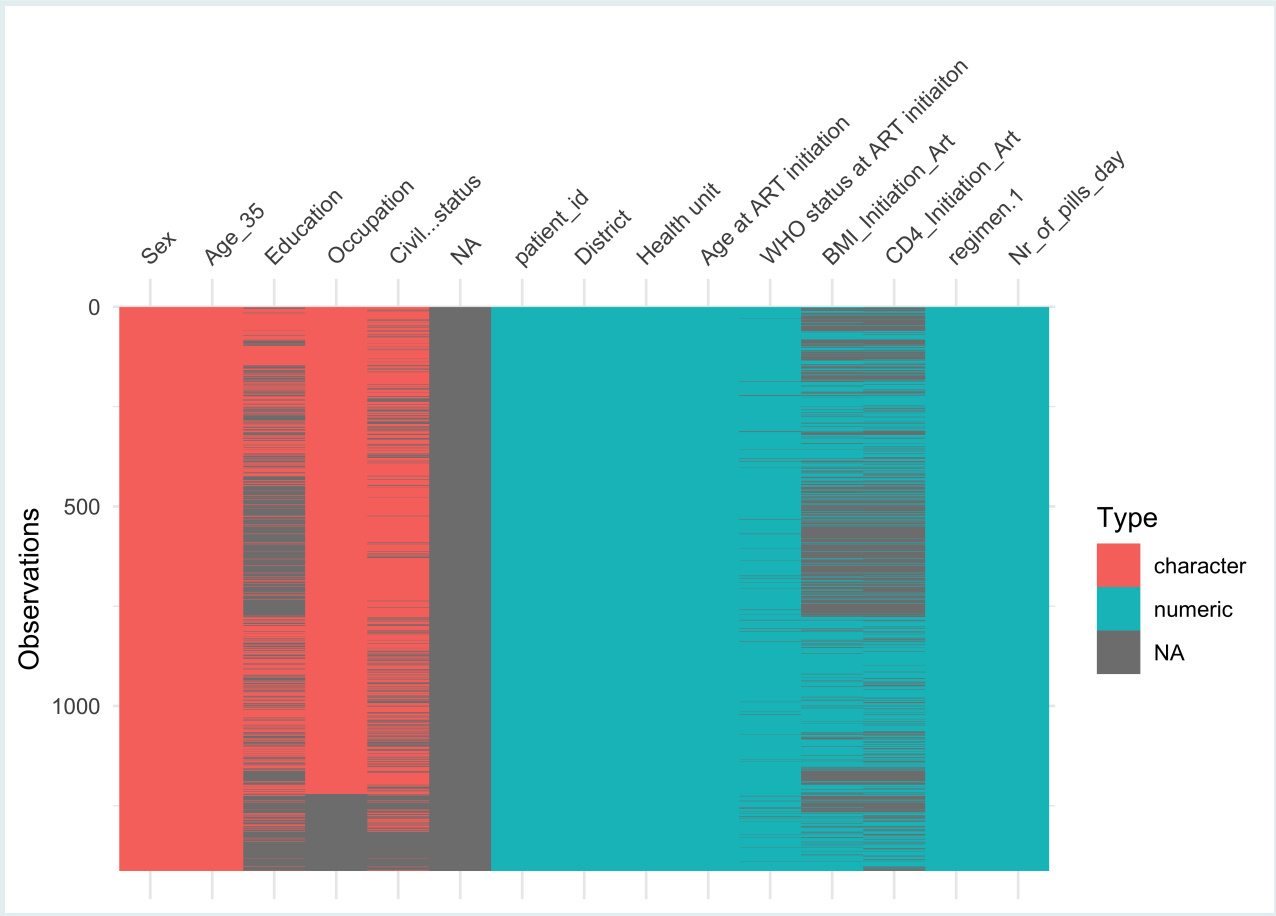
```
        vis_dat(covid_pat)
```




C'est parfait ! Chaque ligne de notre dataframe est représentée par une seule ligne dans le graphique et différentes couleurs sont utilisées pour illustrer les variables caractères (rose) et numériques (bleu), ainsi que les valeurs manquantes (gris). À partir de ce graphique, nous pouvons voir que plusieurs patients de notre jeu de données ont des données manquantes pour la variable `Exposition`.

Regardons maintenant notre vrai jeu de données du monde réel, qui est beaucoup plus grand et plus désordonné. Les grands jeux de données réels peuvent présenter des motifs complexes dans les données qui sont difficiles à repérer sans visualisation, donc l'utilisation de fonctions comme `vis_dat()` peut être particulièrement utile. Essayons-le maintenant !

```
vis_dat(non_adherence)
```




Génial ! À partir de cette vue, nous pouvons déjà voir certains problèmes :

- Il semble y avoir une colonne complètement vide dans les données (la colonne `NA` qui est entièrement grise)
- Plusieurs variables ont beaucoup de valeurs manquantes (comme `Education`, `BMI_Initation_Art` et `CD4_Initation_ART`)
- Les noms de certaines variables sont peu clairs/non nettoyés (par exemple, `Age at ART initation` et `WHO status at ART initiaion` ont des espaces dans leurs noms et `Civil...status` et `regimen.1` ont des caractères spéciaux `.`)

Dans la prochaine leçon, nous essayerons de remédier à ces problèmes et à d'autres dans la mesure du possible pendant le processus de nettoyage des données. Mais pour l'instant, l'objectif est que nous comprenions les fonctions utilisées pour les identifier.

Maintenant que nous avons une solide compréhension de la façon de visualiser les données manquantes avec `vis_dat()`, jetons un coup d'œil à un autre package et fonction qui peut nous aider à générer des statistiques sommaires de nos variables !

L'ensemble de données suivant a été adapté d'une étude qui a examiné les occasions manquées de dépistage du VIH chez les patients se présentant pour la première fois pour des soins contre le VIH dans un hôpital universitaire suisse. L'ensemble de données complet peut être trouvé ici.

```
missed_ops <- read_csv(here("data/HIV_missed_ops.csv"))
```

Utilisez la fonction `vis_dat()` pour obtenir une représentation visuelle des données. Quels problèmes potentiels pouvez-vous repérer en fonction de la sortie ?

---

Voici la traduction en français de la section suivante :

## Génération de statistiques sommaires avec `skimr::skim()`

La fonction `skim()` du package `skimr` fournit une vue d'ensemble dans la console du dataframe et un résumé de chaque colonne (par classe/type). Essayons-la d'abord sur nos données fictives `covid_pat` pour comprendre son fonctionnement. Tout d'abord, pour rappel, voici notre dataframe `covid_pat` :

```
covid_pat
```

Parfait, lançons maintenant la fonction `skim()` !

```
skimr::skim(covid_pat)
```

Table 1: Data summary

| | |
|---|---|
| Name | covid_pat |
| Number of rows | 8 |
| Number of columns | 6 |
| _____ | |
| Column type frequency: | |
| character | 3 |
| numeric | 3 |
| _____ | |
| Group variables | None |

**Variable type: character**

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| Sexe | 0 | 1.00 | 5 | 5 | 0 | 2 | 0 |

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| Exposition | 3 | 0.62 | 6 | 10 | 0 | 4 | 0 |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 | hist |
|---|---|---|---|---|---|---|---|---|---|---|
| Patient_ID | 0 | 1.00 | 4.50 | 2.45 | 1 | 2.75 | 4.5 | 6.25 | 8 |  |
| Age | 1 | 0.88 | 34.57 | 12.39 | 19 | 26.50 | 32.0 | 41.50 | 55 |  |
| Jours_guérison | 0 | 1.00 | 15.00 | 6.68 | 7 | 9.50 | 14.5 | 19.50 | 26 |  |

Incroyable ! Comme nous pouvons le voir, cette fonction fournit :

- Un aperçu des lignes et des colonnes du dataframe
- Le type de données pour chaque variable
- `n_missing`, le nombre de valeurs manquantes pour chaque variable
- `complete_rate`, le taux d'exhaustivité pour chaque variable
- Un ensemble de statistiques sommaires : la moyenne, l'écart-type et les quartiles pour les variables numériques ; et la fréquence et les proportions pour les variables catégorielles
- Des histogrammes spark pour les variables numériques

Maintenant, nous pouvons voir pourquoi cela est si précieux avec de grands dataframes ! Revenons à notre ensemble de données `adh` et exécutons la fonction `skim()` dessus.

```
non_adherence
```

```
skimr::skim(non_adherence)
```

Table 2: Data summary

| | |
|---|---|
| Name | non_adherence |
| Number of rows | 1413 |
| Number of columns | 15 |
| _____ | |
| Column type frequency: | |
| character | 5 |
| logical | 1 |
| numeric | 9 |
| _____ | |
| Group variables | None |

**Variable type: character**

| skim_variable | n_missing | complete_rate | min | max | empty | n_unique | whitespace |
|---|---|---|---|---|---|---|---|
| Sex | 0 | 1.00 | 1 | 4 | 0 | 2 | 0 |
| Age_35 | 0 | 1.00 | 7 | 8 | 0 | 2 | 0 |
| Education | 776 | 0.45 | 4 | 10 | 0 | 5 | 0 |
| Occupation | 193 | 0.86 | 4 | 22 | 0 | 50 | 0 |
| Civil...status | 412 | 0.71 | 6 | 12 | 0 | 4 | 0 |

**Variable type: logical**

| skim_variable | n_missing | complete_rate | mean | count |
|---|---|---|---|---|
| NA | 1413 | 0 | NaN | : |

**Variable type: numeric**

| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
|---|---|---|---|---|---|---|---|---|---|
| skim_variable | n_missing | complete_rate | mean | sd | p0 | p25 | p50 | p75 | p100 |
| patient_id | 0 | 1.00 | 7364.75 | 3312.95 | 147.0 | 4705.00 | 7355.0 | 10098.0 | 24377.0 |
| District | 0 | 1.00 | 1.35 | 0.48 | 1.0 | 1.00 | 1.0 | 2.0 | 2.0 |
| Health unit | 0 | 1.00 | 1.87 | 0.90 | 1.0 | 1.00 | 2.0 | 3.0 | 3.0 |
| Age at ART initiation | 0 | 1.00 | 32.07 | 9.51 | 15.1 | 25.20 | 30.4 | 36.8 | 75.3 |
| WHO status at ART initiaiton | 45 | 0.97 | 1.80 | 0.98 | 1.0 | 1.00 | 1.0 | 3.0 | 4.0 |
| BMI_Initiation_Art | 588 | 0.58 | 20.37 | 3.54 | 9.0 | 18.12 | 20.1 | 22.2 | 52.0 |
| CD4_Initiation_Art | 674 | 0.52 | 398.19 | 243.64 | 3.0 | 233.00 | 343.0 | 538.0 | 1401.0 |
| regimen.1 | 0 | 1.00 | 4.64 | 1.54 | 1.0 | 3.00 | 6.0 | 6.0 | 7.0 |
| Nr_of_pills_day | 0 | 1.00 | 1.51 | 0.59 | 1.0 | 1.00 | 1.0 | 2.0 | 3.0 |

À partir de cette sortie, nous pouvons identifier certains problèmes potentiels avec nos données :

- Nous pouvons confirmer que la colonne NA, que vous avez vue dans la sortie `vis_dat()`, est bien complètement vide : elle a un `complete_rate` de 0
- La distribution de `Age at ART initation` est asymétrique

Utilisez `skim()` pour obtenir un aperçu détaillé de l'ensemble de données `missed_ops`.

Super ! Maintenant que nous savons comment générer un bref rapport de statistiques sommaires pour nos variables. Dans la prochaine section, nous découvrirons quelques fonctions utiles du package `inspectdf` qui nous permettent de visualiser différentes statistiques sommaires.

---

## Visualisation des statistiques sommaires avec des fonctions de `inspectdf`

Bien que la fonction `skimr::skim()` vous donne des résumés de variables dans la console, vous voudrez parfois plutôt des résumés de variables sous une forme graphique plus riche. Pour cela, les fonctions `inspectdf::inspect_cat()` et `inspectdf::inspect_num()` peuvent être utilisées.

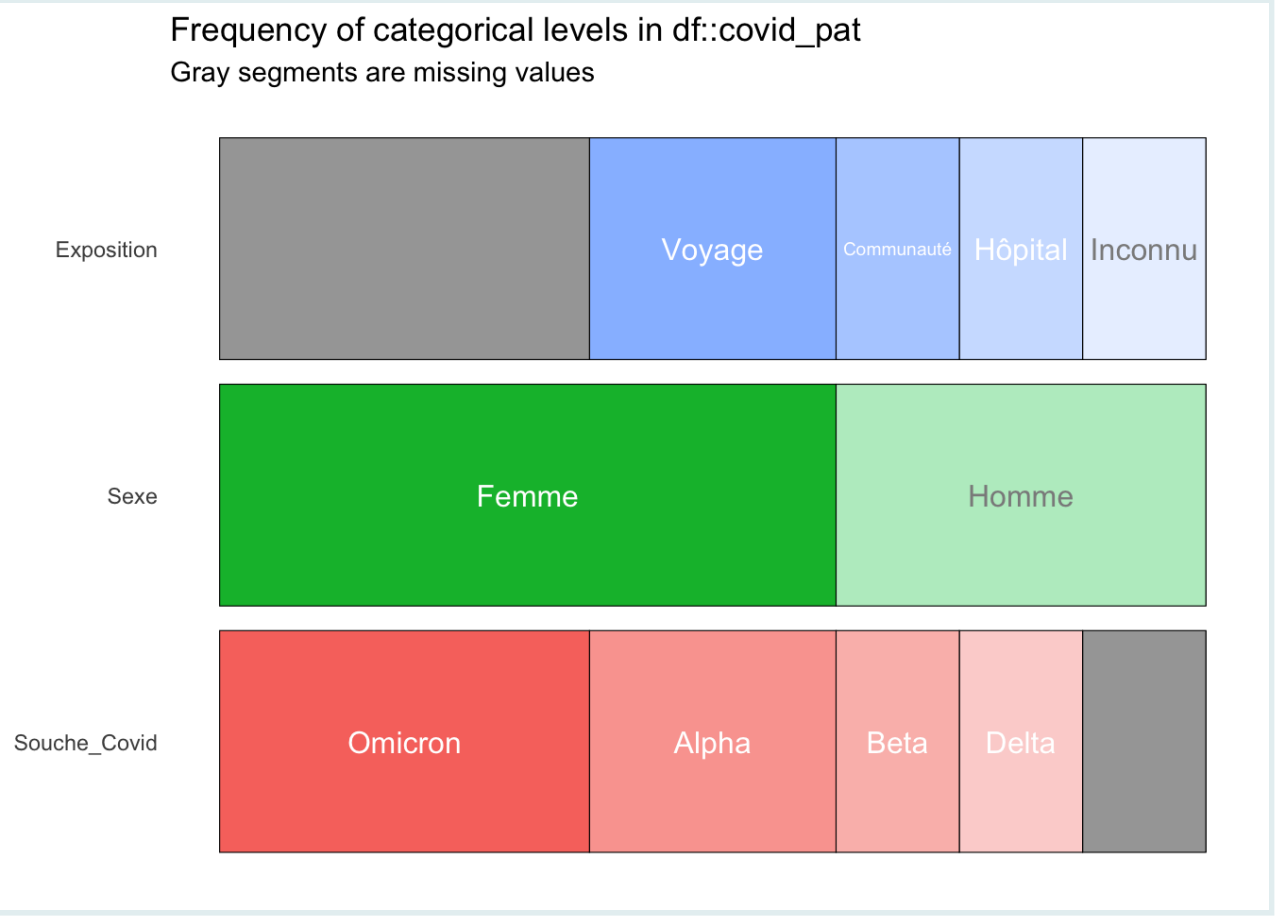
Si vous exécutez `inspect_cat()` sur un objet de données, vous obtenez un résumé tabulaire des variables catégorielles dans le jeu de données (les informations importantes sont cachées dans la colonne `levels`). Essayons-le d'abord sur le jeu de données `covid_pat`. Pour rappel, voici notre jeu de données :

```
covid_pat
```

```
inspect_cat(covid_pat)
```

La magie se produit lorsque vous exécutez `show_plot()` sur le résultat de `inspect_cat()` :

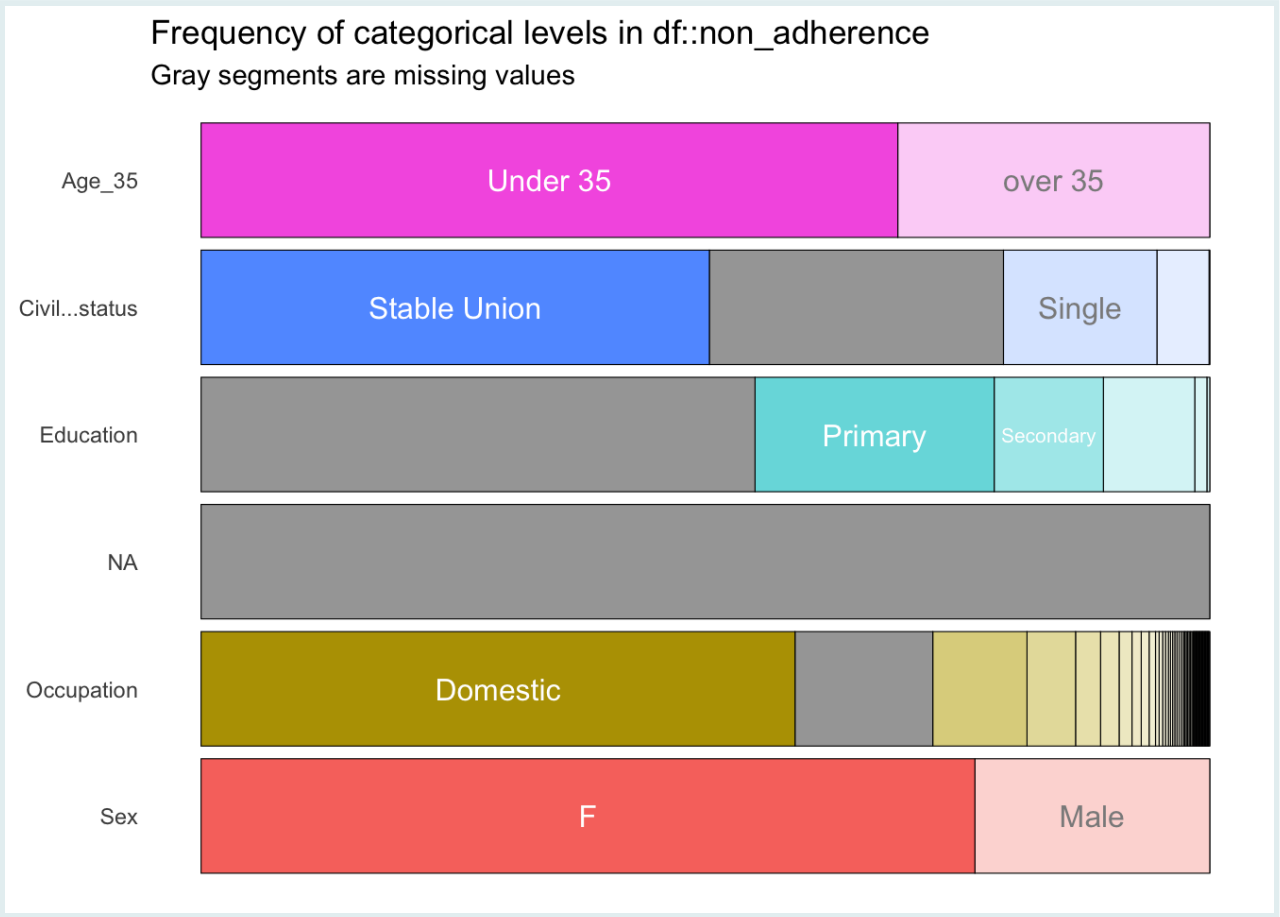
```
inspect_cat(covid_pat) %>%
  show_plot()
```




Cela semble parfait ! Vous obtenez une belle figure récapitulative montrant la distribution des variables catégorielles ! Les niveaux de variable sont également joliment étiquetés (s'il y a suffisamment d'espace pour afficher une étiquette).

Maintenant, essayons-le sur notre jeu de données `non_adherence` :

```
inspect_cat(non_adherence) %>%
  show_plot()
```




À partir de là, vous pouvez observer certains problèmes avec quelques variables catégorielles :

- Le niveau de variable `Under 35` commence par une majuscule, tandis que `over 35` commence par une minuscule. Il pourrait être bon de standardiser cela.
- La variable sexe a les niveaux `F` et `Male`. Cela pourrait également valoir la peine d'être normalisé.
- Comme nous l'avons vu précédemment, `NA` est complètement vide.

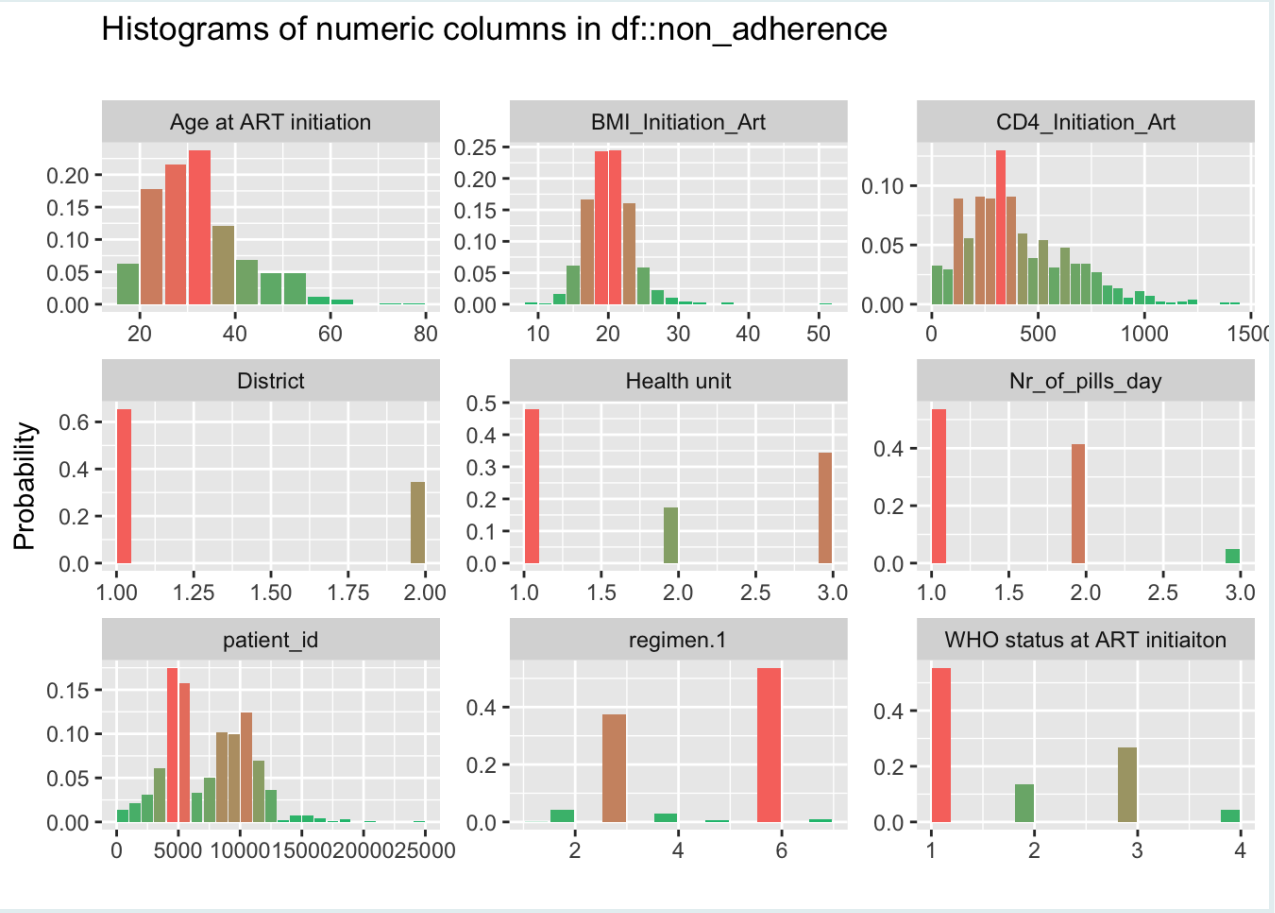
Complétez le code suivant pour obtenir un résumé visuel des variables catégorielles dans le jeu de données `missed_op`.

```
        inspect___() %>%
          __________
```

Combien de problèmes de données potentiels pouvez-vous repérer ?

De même, vous pouvez obtenir un graphique de résumé pour les variables numériques du jeu de données avec `inspect_num()`. Exécutons cela sur notre jeu de données `non_adherence` :

```
        inspect_num(non_adherence) %>%
          show_plot()
```




À partir de cette sortie, vous pouvez remarquer que de nombreuses variables qui devraient être des variables factor sont codées comme numériques. En fait, les seules vraies variables numériques sont `Age at ART initiation`, `BMI_Initiation_ART`, `CD4_Initation_ART` et `Nr_of_pills_day`. Nous corrigerons ces problèmes dans la prochaine leçon lorsque nous passerons au nettoyage des données. Pour l'instant cependant, jetons un coup d'œil à notre dernière fonction pour l'analyse exploratoire des données !

Utilisez `inspect_num` pour créer des histogrammes de vos variables numériques dans le jeu de données `missed_op`. Les types de variables numériques sont-ils corrects ?

---

Voici la traduction en français de la section suivante :

## Création de rapports de données avec `DataExplorer::create_report()`

Enfin, la fonction `create_report()` du package `DataExplorer` crée un profil complet d'un dataframe : un fichier HTML avec des statistiques de base et des visualisations de distribution.

Exécutons cette fonction sur le dataframe `adh` et observons sa sortie. Notez que cela peut prendre un certain temps. Si cela prend trop de temps, vous pouvez l'exécuter sur un échantillon du jeu de données plutôt que sur l'ensemble du jeu de données.

```
        create_report(non_adherence)
```

Comme vous pouvez le voir, le rapport est assez complet. Nous ne passerons pas en revue toutes les sorties de ce rapport de données car de nombreuses sorties graphiques sont les mêmes que celles que nous avons vues avec les fonctions précédentes ! Cependant, certaines nouvelles sorties incluent :

- Un graphique QQ pour évaluer la normalité des variables numériques
- Une analyse de corrélation (lorsqu'il y a suffisamment de lignes complètes)
- Une analyse en composantes principales (lorsqu'il y a suffisamment de lignes complètes)

N'hésitez pas à explorer la documentation du package par vous-même.

Créez un rapport de données pour vos données `missed_ops` en utilisant la fonction `create_report()` !

---

## En Résumé

En nous familiarisant avec les données, nous avons pu identifier certains problèmes potentiels qu'il faudra peut-être résoudre avant d'utiliser les données dans une analyse.

Et comme vous l'avez vu, le code nécessaire pour explorer ces données est en réalité très peu important ; d'autres développeurs R ont fait le travail difficile pour nous en construisant d'incroyables packages pour analyser rapidement les jeux de données et identifier les problèmes.

À partir de la prochaine leçon, nous allons aborder ces problèmes identifiés un par un, en commençant par le problème de noms de variables incohérents et désordonnés.

On se retrouve à la prochaine leçon !

[XXX DEMO HERE XXX]

REMINDER

[XXX SIDE NOTES HERE XXXX]

RECAP

[XXX RECAP HERE XXX]

PRACTICE

(in RMD)

[XXX PRACTICE QUESTIONS HERE XXX] - NOTE: Make sure to add the answers to all practice at the end of the lesson! We are no longer using autograders for the GF project.

[XXX PRACTICE QUESTIONS HERE XXX] - NOTE: Make sure to add the answers to all practice at the end of the lesson! We are no longer using autograders for the GF project.

CHALLENGE

[XXX CHALLENGE PRACTICE QUESTIONS HERE XXX] - NOTE: Make sure to add the answers to all practice at the end of the lesson! We are no longer using autograders for the GF project.

PRO TIP

[XXX PRO-TIPS HERE XXX]

VOCAB


[XXX RELEVANT VOCUBULARY WORDS HERE XXX]


[XXX SIDE NOTES HERE XXXX]


[XXX TIPS HERE XXX]


[XXX KEY POINTS HERE XXX]


[XXX ERRORS HERE XXX]


[XXX RSTUDIO CLOUD TIPS HERE XXX]

## WRAP UP!

[XXX NICE WRAP UP MESSAGE OR SUMMARY IF NEEDED HERE XXX]

## Answer Key

## Contributors

The following team members contributed to this lesson:

### KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about world improvement

### LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network
A firm believer in science for good, striving to ally programming, health and education

(make sure to update the contributor list accordingly!)