
Notes de leçon | Filtrer, modifier et organiser par groupes

February 2024

Introduction	
Objectifs d'apprentissage	
Packages	
Jeux de données	
Organisation par groupe	
<code>arrange()</code> peut regrouper automatiquement	
Filtrage par groupe	
Filtrage avec des groupements imbriqués	
Modification par groupe	
Modification avec des groupes imbriqués	
Conclusion	

Introduction

La manipulation de données implique souvent d'appliquer les mêmes opérations séparément à différents groupes au sein des données. Ce motif, parfois appelé "split-apply-combine", est facilement réalisable dans {dplyr} en enchaînant le verbe `group_by()` avec d'autres verbes de manipulation tels que `filter()`, `mutate()` et `arrange()` (que vous avez déjà vus !).

Dans cette leçon, vous deviendrez confiant avec ce type de manipulations groupées.

Commençons.

Objectifs d'apprentissage

1. Vous pouvez utiliser `group_by()` avec `arrange()`, `filter()` et `mutate()` pour effectuer des opérations groupées sur un jeu de données.

Packages

Cette leçon exigera la suite de packages {tidyverse} et le package {here} :

```
library(pacman) install.packages("pacman")
p_load(tidyverse, here)
```

Jeux de données

Dans cette leçon, nous utiliserons encore les données de l'enquête sérologique COVID-19 menée à Yaoundé, au Cameroun. Ci-dessous, nous importons les données, créons un petit sous-ensemble du jeu de données, `yao` et un sous-ensemble encore plus petit, `yao_sexe_poids`.

```
read_csv(here::here('data/fr_yaounde_data.csv')) %>%
  select(sexe, age, cat_age, poids_kg, occupation, resultat_igg, resultat_igm)
```

```
## # A tibble: 5 × 7
##   sexe      age cat_age poids_kg
##   <chr>  <dbl> <chr>    <dbl>
## 1 Female    45  45 - 64      95
## 2 Male     55  45 - 64      96
## 3 Male     23  15 - 29      74
## 4 Female    20  15 - 29      70
## 5 Female    55  45 - 64      67
##   occupation      resultat_igg
##   <chr>          <chr>
## 1 Informal worker Negative
## 2 Salaried worker Positive
## 3 Student          Negative
## 4 Student          Positive
## 5 Trader--Farmer   Positive
## #> 1 more variable: resultat_igm <chr>
```

```
yao_poids <-
yao %>%
  select(sexe, poids_kg)

yao_poids
```

```
## # A tibble: 5 × 2
##   sexe      poids_kg
##   <chr>    <dbl>
## 1 Female      95
## 2 Male       96
## 3 Male       74
## 4 Female      70
## 5 Female      67
```

Pour les questions pratiques, nous utiliserons également le jeu de données sur la sarcopénie que vous avez déjà vu :

```
sia <- read_csv(here::here('data/fr_sarcopenia_elderly.csv'))
```

```
sia
```

```
## # A tibble: 5 × 9
##   numero   age groupe_d_age
##   <dbl> <dbl> <chr>
## 1     7  60.8 Sixties
## 2     8  72.3 Seventies
## 3     9  62.6 Sixties
## 4    12   72  Seventies
## 5    13  60.1 Sixties
##   sexe_homme_1_femme_0 statut_marital
##   <dbl> <chr>
## 1           0 married
## 2           1 married
## 3           0 married
## 4           0 widow
## 5           0 married
## # i 4 more variables: taille_metres <dbl>, poids_kg <dbl>,
## #   force_de_prehension_kg <dbl>, ...
```

Organisation par groupe

La fonction `arrange()` ordonne les lignes d'un jeu de données par les valeurs des colonnes sélectionnées. Cette fonction est sensible aux groupements uniquement lorsque nous définissons son argument `.by_group` à `TRUE`. Pour illustrer cela, considérons le jeu de données `yao_sexe_poids` :

```
yao_poids
```

```
## # A tibble: 5 × 2
##   sexe   poids_kg
##   <chr>   <dbl>
## 1 Female     95
## 2 Male      96
## 3 Male      74
## 4 Female     70
## 5 Female     67
```

Nous pouvons organiser ce jeu de données par poids de cette façon :

```
se_poids %>%  
  arrange(poids_kg)
```

```
## # A tibble: 5 × 2  
##   sexe    poids_kg  
##   <chr>    <dbl>  
## 1 Female      14  
## 2 Male        15  
## 3 Male        15  
## 4 Male        15  
## 5 Female      15
```

Comme prévu, les poids les plus faibles ont été placés en haut de la trame de données.

Si nous regroupons d'abord les données, nous pourrions nous attendre à un résultat différent :

```
se_poids %>%  
  group_by(sexe) %>%  
  arrange(poids_kg)
```

```
## # A tibble: 5 × 2  
## # Groups:   sexe [2]  
##   sexe    poids_kg  
##   <chr>    <dbl>  
## 1 Female      14  
## 2 Male        15  
## 3 Male        15  
## 4 Male        15  
## 5 Female      15
```

Mais comme vous le voyez, l'organisation est toujours la même.

Ce n'est que lorsque nous définissons l'argument `.by_group` à `TRUE` que nous obtenons quelque chose de différent :

```
se_poids %>%  
  group_by(sexe) %>%  
  arrange(poids_kg, .by_group = TRUE)
```

```
## # A tibble: 3 × 2  
## # Groups:   sexe [1]  
##   sexe    poids_kg  
##   <chr>    <dbl>  
## 1 Female      14  
## 2 Female      15  
## 3 Female      16
```

```
## 4 Female      16
## 5 Female      18
```

Maintenant, les données sont *d'abord* triées par sexe (toutes les femmes en premier), puis par poids.

`arrange()` peut regrouper automatiquement

En réalité, nous n'avons pas besoin de `group_by()` pour organiser par groupe ; nous pouvons simplement mettre plusieurs variables dans la fonction `arrange()` pour obtenir le même effet.

Ainsi, cette simple commande `arrange()` :

```
data_poids %>%
  arrange(sexe, poids_kg)
```

```
## # A tibble: 5 × 2
##   sexe   poids_kg
##   <chr>     <dbl>
## 1 Female      14
## 2 Female      15
## 3 Female      16
## 4 Female      16
## 5 Female      18
```

est équivalente à la commande plus complexe `group_by()`, `arrange()` utilisée précédemment :

```
data_poids %>%
  group_by(sexe) %>%
  arrange(poids_kg, .by_group = TRUE)
```

La commande `arrange(sexe, poids_kg)` dit à R d'organiser les lignes *d'abord* par sexe, puis par poids.

Évidemment, cette syntaxe, avec juste `arrange()`, et pas de `group_by()` est plus simple, donc vous pouvez vous y tenir.

Rappel

`desc()` pour l'ordre décroissant

Rappelez-vous que pour classer *en ordre décroissant*, nous pouvons mettre la variable cible dans `desc()`. Donc, par exemple, pour trier par sexe et poids, mais avec les personnes les plus lourdes en haut, nous pouvons exécuter :

```
data_poids %>%
  arrange(sexe, desc(poids_kg))
```

```
## # A tibble: 5 × 2
##   sexe   poids_kg
##   <chr>     <dbl>
## 1 Female     162
## 2 Female     161
## 3 Female     158
## 4 Female     135
## 5 Female     129
```

PRACTICE



(in RMD)

Avec une commande `arrange()`, triez les données `sarcopenia` d'abord par sexe, puis par force de préhension. (Si c'est fait correctement, la première ligne devrait être celle d'une femme avec une force de préhension de 1,3 kg). Pour rendre l'organisation claire, vous devriez d'abord `select()` les variables de sexe et de force de préhension.

Écrivez le code avec votre réponse :

```
sarcopenia_ arrangee <-
  select(sarcopenia, sexe, force_de_prehension) %>%
  arrange(sexe, force_de_prehension)
```

PRACTICE



(in RMD)

Le jeu de données `sarcopenia` contient une colonne `groupe_d_age`, qui stocke les groupes d'âge en chaîne de caractères (les groupes d'âge sont "Sixties", "Seventies" et "Eighties"). Convertissez cette variable en un facteur avec les niveaux dans le bon ordre (d'abord "Sixties", puis "Seventies" et ainsi de suite). (Astuce : Revenez sur la leçon `case_when()` si vous ne voyez pas comment réattribuer un niveau à un facteur).

Ensuite, avec une commande imbriquée à `arrange()`, classez les données d'abord par la variable facteur `groupe_d_age` nouvellement créée (les individus les plus jeunes en premier) puis par `taille_metres`, avec les individus les plus petits en premier.

Écrivez le code avec votre réponse :

```
sarcopenia_ arrangee <-
  select(sarcopenia, groupe_d_age, taille_metres) %>%
  arrange(groupe_d_age, taille_metres)
```

Filtrage par groupe

La fonction `filter()` conserve ou supprime des lignes en fonction d'une condition. Si `filter()` est appliquée à des données groupées, l'opération de filtrage est réalisée séparément pour chaque groupe.

Pour illustrer cela, considérons à nouveau le jeu de données `yao_sexe_poids` :

```
yao_poids
```

```
## # A tibble: 5 × 2
##   sexe   poids_kg
##   <chr>   <dbl>
## 1 Female     95
## 2 Male      96
## 3 Male      74
## 4 Female     70
## 5 Female     67
```

Si nous voulons filtrer les données pour la personne la plus lourde, nous pourrions exécuter :

```
yao_poids %>%
  filter(poids_kg == max(poids_kg))
```

```
## # A tibble: 1 × 2
##   sexe   poids_kg
##   <chr>   <dbl>
## 1 Female    162
```

Mais si nous voulons obtenir la personne la plus lourde par groupe de sexe (l'homme le plus lourd *et* la femme la plus lourde), nous pouvons utiliser `group_by(sex)` puis `filter()` :

```
yao_poids %>%
  group_by(sexe) %>%
  filter(poids_kg == max(poids_kg))
```

```
## # A tibble: 2 × 2
## # Groups:   sexe [2]
##   sexe   poids_kg
##   <chr>   <dbl>
## 1 Male     128
## 2 Female    162
```

Parfait ! Le code ci-dessus peut être traduit par “Pour chaque groupe de sexe, conserve la ligne avec la valeur maximale `poids_kg`”.

Filtrage avec des groupements imbriqués

La fonction `filter()` fonctionne bien avec un nombre quelconque de groupements imbriqués.

Par exemple, si nous voulons voir l’homme le plus lourd et la femme la plus lourde *par groupe d’âge*, nous pourrions exécuter le code suivant sur le jeu de données `yao` :

```
by(sexe, cat_age) %>%
  filter(poids_kg == max(poids_kg))
```

Ce code regroupe par sexe *et* catégorie d’âge, puis trouve la personne la plus lourde dans chaque sous-catégorie.

(Pourquoi avons-nous 10 lignes dans la sortie ? Eh bien, 2 groupes de sexe x 5 groupes d’âge = 10 groupements uniques.)

La sortie est un peu dispersée, donc nous pouvons enchaîner cela avec la fonction `arrange()`, pour organiser par sexe et groupe d’âge.

```
by(sexe, cat_age) %>%
  filter(poids_kg == max(poids_kg)) %>%
  arrange(sexe, cat_age)
```

Maintenant, les données sont plus faciles à lire. Toutes les femmes viennent en premier, puis les hommes. Mais nous remarquons un arrangement étrange des groupes d’âge ! Ceux âgés de 5 à 14 ans devraient venir *en premier* dans l’arrangement. Bien sûr, nous avons appris comment corriger cela avec la fonction `factor()` et son argument `levels` :

```
cat_age = factor(
  age,
  levels = c("5 - 14", "15 - 29", "30 - 44", "45 - 64", "65 +")
)
by(sexe, cat_age) %>%
  filter(poids_kg == max(poids_kg)) %>%
  arrange(sexe, cat_age)
```

Maintenant, nous avons une sortie bien organisée !

PRACTICE



PRACTICE



(in RMD)

Groupez le jeu de données `sarcopenia` par groupe d'âge et sexe, puis filtrez pour obtenir l'index de masse musculaire le plus élevé dans chaque groupe (imbriqué).

```
Écrivez le code avec votre réponse :  
_de_muscle_squelettique_max <-  
sarcopenia %>%  
  filter(VOTRE_RÉPONSE_ICI)
```

Modification par groupe

`mutate()` est utilisé pour modifier les colonnes ou pour en créer de nouvelles. Avec des données groupées, `mutate()` opère sur chaque groupe indépendamment.

Considérons d'abord un appel régulier à `mutate()`, pas un groupé. Imaginez que vous voulez ajouter une colonne qui classe les répondants par poids. Cela peut être fait avec la fonction `rank()` à l'intérieur d'un appel à `mutate()` :

```
_poids %>%  
mutate(poids_ordre = rank(poids_kg))
```

```
## # A tibble: 5 × 3  
##   sexe   poids_kg poids_ordre  
##   <chr>   <dbl>     <dbl>  
## 1 Female     95         901  
## 2 Male      96         908  
## 3 Male      74         640.  
## 4 Female     70         564.  
## 5 Female     67         502.
```

La sortie montre que la première ligne est le 901ème individu le plus léger. Mais il serait plus intuitif de classer dans l'ordre décroissant avec la personne la plus lourde en premier. Nous pouvons le faire avec la fonction `desc()` :

```
_poids %>%  
mutate(poids_ordre = rank(desc(poids_kg)))
```

```
## # A tibble: 5 × 3  
##   sexe   poids_kg poids_ordre  
##   <chr>   <dbl>     <dbl>  
## 1 Female     95         71  
## 2 Male      96         64  
## 3 Male      74        332.
```

```
## 4 Female      70      408.
## 5 Female      67      470.
```

La sortie montre que la personne de la première ligne est la 71ème personne la plus lourde.

Maintenant, essayons d'écrire un appel groupé à `mutate()`. Imaginez que nous voulons ajouter cette colonne de classement par poids *par groupe de sexe* dans le jeu de données. Autrement dit, nous voulons connaître le classement de chaque personne par poids dans leur catégorie de sexe. Dans ce cas, nous pouvons combiner `group_by(sex)` avec `mutate()` :

```
mutate(poids_order = rank(desc(poids_kg)))
```

```
## # A tibble: 5 × 3
## # Groups:   sexe [2]
##   sexe    poids_kg poids_order
##   <chr>      <dbl>      <dbl>
## 1 Female        95         53.5
## 2 Male         96         13.5
## 3 Male         74         148
## 4 Female        70         220.
## 5 Female        67         250.
```

Nous voyons maintenant que la personne de la première ligne est la 53ème femme la plus lourde. (Le .5 indique que ce rang est à égalité avec quelqu'un d'autre dans les données.)

Nous pourrions également organiser les données pour rendre les choses plus claires :

```
mutate(poids_order = rank(desc(poids_kg))) %>%
  arrange(sexe, poids_order)
```

```
## # A tibble: 5 × 3
## # Groups:   sexe [1]
##   sexe    poids_kg poids_order
##   <chr>      <dbl>      <dbl>
## 1 Female        162          1
## 2 Female        161          2
## 3 Female        158          3
## 4 Female        135          4
## 5 Female        129          5
```

Modification avec des groupes imbriqués

Bien sûr, comme avec les autres verbes que nous avons vus, `mutate()` fonctionne aussi avec des groupes imbriqués.

Par exemple, ci-dessous nous créons le groupe imbriqué d'âge *et* de sexe avec le jeu de données `yao`, puis nous ajoutons une colonne de rang avec `mutate()` :

```
by(sexe, cat_age) %>%
  mutate(rank_order = rank(desc(poids_kg)))

## # A tibble: 5 × 8
## # Groups:   sexe, cat_age [4]
##   sexe    age cat_age poids_kg
##   <chr> <dbl> <chr>    <dbl>
## 1 Female   45  45 - 64         95
## 2 Male     55  45 - 64         96
## 3 Male     23  15 - 29         74
## 4 Female   20  15 - 29         70
## 5 Female   55  45 - 64         67
##   occupation resultat_igg
##   <chr>         <chr>
## 1 Informal worker Negative
## 2 Salaried worker Positive
## 3 Student          Negative
## 4 Student          Positive
## 5 Trader--Farmer   Positive
## # 2 more variables: resultat_igm <chr>, poids_ordre <dbl>
```

La sortie montre que la personne de la première ligne est la 20ème femme la plus lourde *dans le groupe d'âge de 45 à 64 ans*.

PRACTICE



(in RMD)

Avec les données `sarcopenia`, groupez par `groupe_d_age`, puis dans une nouvelle variable appelée `force_prehension_ordre`, calculez le rang de la force de préhension de chaque individu par groupe d'âge. (Pour calculer le rang, utilisez `mutate()` et la fonction `rank()` avec sa méthode par défaut pour les égalités.)

```
# Copiez le code avec votre réponse :
# force_prehension_ordre <-
# sarcopenia %>%
#   mutate(rank_order = rank(desc(force_prehension)))
# # Copiez VOTRE RÉPONSE ICI"
```

WATCH OUT



Attention

N'oubliez pas de dégrouper les données avant de procéder à une analyse plus approfondie

Comme il a été mentionné auparavant, il est important de dégrouper vos données avant de faire une analyse plus détaillée.

Considérez ce dernier exemple, où nous avons calculé le rang de poids des individus par groupe d'âge et de sexe :

```
by(sexe, cat_age) %>%
  >(poids_ordre = rank(desc(poids_kg)))
```

WATCH OUT



```
## # A tibble: 5 × 8
## # Groups:   sexe, cat_age [4]
##   sexe      age cat_age poids_kg
##   <chr> <dbl> <chr>      <dbl>
## 1 Female    45 45 - 64         95
## 2 Male      55 45 - 64         96
## 3 Male      23 15 - 29         74
## 4 Female    20 15 - 29         70
## 5 Female    55 45 - 64         67
##   occupation      resultat_igg
##   <chr>          <chr>
## 1 Informal worker Negative
## 2 Salaried worker Positive
## 3 Student          Negative
## 4 Student          Positive
## 5 Trader--Farmer Positive
## # i 2 more variables: resultat_igm <chr>, poids_ordre <dbl>
```

Si, dans le processus d'analyse, vous avez stocké cette sortie comme un nouveau jeu de données :

```
dfie <-
  >%
  by(sexe, cat_age) %>%
  >(poids_ordre = rank(desc(poids_kg)))
```

Et puis, plus tard, vous avez repris le jeu de données et essayé une autre analyse, par exemple, en filtrant pour obtenir la personne la plus âgée dans les données :

```

.fie %>%
  filter(age == max(age))

```

```

## # A tibble: 5 × 8
## # Groups:   sexe, cat_age [5]
##   sexe      age cat_age poids_kg
##   <chr> <dbl> <chr>      <dbl>
## 1 Male      65 45 - 64          93
## 2 Male      78 65 +           95
## 3 Male      14 5 - 14          44
## 4 Female    44 30 - 44          67
## 5 Female    79 65 +           40
##   occupation      resultat_igg
##   <chr>              <chr>
## 1 Retired           Negative
## 2 Retired--Informal work... Positive
## 3 Student           Negative
## 4 Home-maker        Positive
## 5 Retired           Negative
## # 2 more variables: resultat_igm <chr>, poids_ordre <dbl>

```

WATCH OUT



Vous pourriez être confus par la sortie! Pourquoi y a-t-il 55 lignes de “personnes les plus âgées”?

Cela serait dû au fait que vous avez oublié de dégrouper les données avant de les stocker pour une analyse plus approfondie. Faisons cela correctement maintenant

```

.fie <-
  group_by(sexe, cat_age) %>%
  arrange(poids_ordre = rank(desc(poids_kg))) %>%
  ungroup()

```

Maintenant, nous pouvons obtenir correctement la/les personne(s) la/les plus âgée(s) dans le jeu de données :

```

.fie %>%
  filter(age == max(age))

```

```

## # A tibble: 2 × 8
##   sexe      age cat_age poids_kg
##   <chr> <dbl> <chr>      <dbl>
## 1 Female    79 65 +           40
## 2 Female    79 65 +           81
##   occupation resultat_igg resultat_igm
##   <chr>              <chr>          <chr>

```

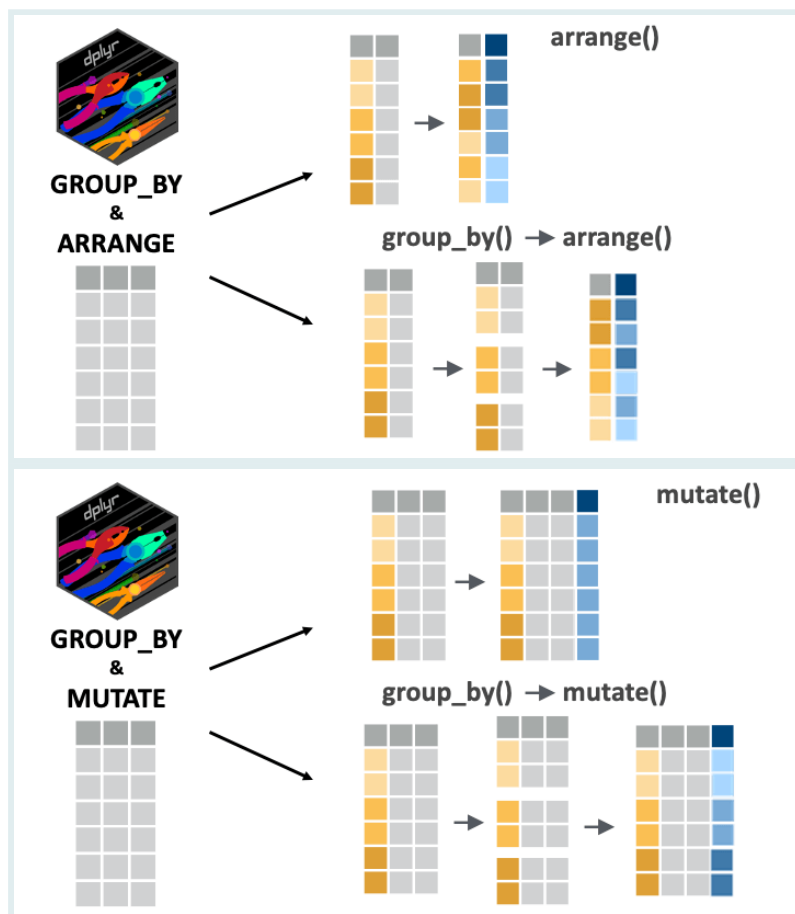
WATCH OUT



```
## 1 Retired      Negative      Negative
## 2 Home-maker  Negative      Negative
## # 1 more variable: poids_ordre <dbl>
```

Conclusion

`group_by()` est un outil merveilleux pour arranger, modifier, filtrer en fonction des groupes au sein d'une ou plusieurs variables.



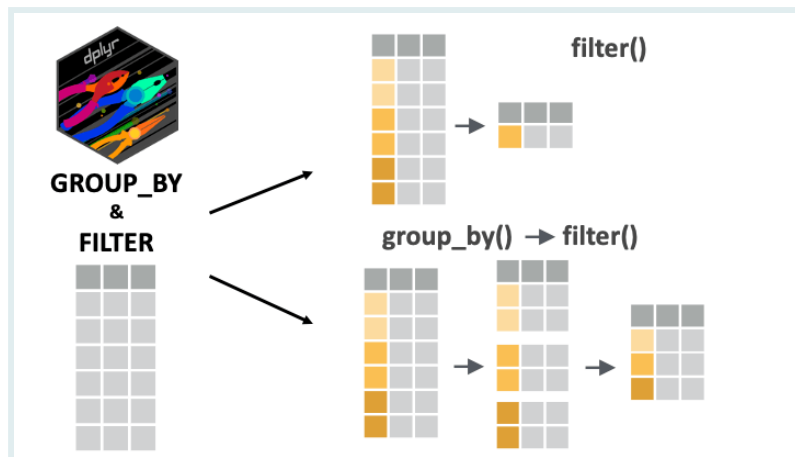


Fig: filter() et son utilisation combinée avec group_by()

Il existe de nombreuses façons de combiner ces verbes pour manipuler vos données. Nous vous invitons à prendre un peu de temps pour essayer ces verbes dans différentes combinaisons !

À la prochaine !

Contributeurs

Les membres de l'équipe suivants ont contribué à cette leçon :



LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network

A firm believer in science for good, striving to ally programming, health and education



KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement



GUY WAFEU

R Instructor and Public Health Physician

Committed to improving the quality of data analysis



SABINA RODRIGUEZ VELÁSQUEZ

Project Manager and Scientific Collaborator, The GRAPH Network

Infectiously enthusiastic about microbes and Global Health

Références

Certains matériaux de cette leçon ont été adaptés des sources suivantes :

- Horst, A. (2022). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Travail original publié en 2020)
- *Grouper par une ou plusieurs variables*. (n.d.). Consulté le 21 Février 2022, sur https://dplyr.tidyverse.org/reference/group_by.html
- *Créer, modifier et supprimer des colonnes — Mutate*. (n.d.). Consulté le 21 Février 2022, sur <https://dplyr.tidyverse.org/reference/mutate.html>
- *Sélectionner des lignes en utilisant les valeurs des colonnes — Filter*. (n.d.). Consulté le 21 Février 2022, sur <https://dplyr.tidyverse.org/reference/filter.html>
- *Arranger les lignes par valeurs de colonnes — Arrange*. (n.d.). Consulté le 21 Février 2022, sur <https://dplyr.tidyverse.org/reference/arrange.html>

L'œuvre d'art a été adaptée de :

- Horst, A. (2022). *Illustrations R & stats par Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Travail original publié en 2018)