

---

# Notes de leçon | Regroupement et résumé des données

January 2024



Introduction .....	
Objectifs d'apprentissage .....	
Le jeu de données COVID-19 de Yaoundé .....	
Qu'est-ce qu'une statistique récapitulative ? .....	
Introduction à <code>dplyr::summarize()</code> .....	
Résumés groupés avec <code>dplyr::group_by()</code> .....	
Regroupement par plusieurs variables (groupement imbriqué) .....	
Dégroupement avec <code>dplyr::ungroup()</code> (pourquoi et comment) .....	
Comptage des lignes .....	
Compter les lignes qui répondent à une condition .....	
<code>dplyr::count()</code> .....	
Inclure les combinaisons manquantes dans les statistiques récapitulatives .....	
Conclusion .....	
Solutions des exercices pratiques .....	

---

## Introduction

Vous savez déjà comment conserver les jeux de données qui vous intéressent, comment garder les variables pertinentes et comment les modifier ou en créer de nouvelles.

Maintenant, nous allons aller un peu plus loin dans la manipulation de vos données en comprenant comment extraire facilement des statistiques récapitulatives, grâce au verbe `summarize()`, comme le calcul de la moyenne d'une variable.

De plus, nous commencerons à explorer un verbe crucial, `group_by()`, capable de regrouper vos variables ensemble pour effectuer des opérations groupées sur votre jeu de données.

Allons-y !

---

---

## Objectifs d'apprentissage

1. Vous pouvez utiliser `dplyr::summarize()` pour extraire des statistiques récapitulatives des jeux de données.
2. Vous pouvez utiliser `dplyr::group_by()` pour regrouper les données par une ou plusieurs variables avant d'effectuer des opérations sur elles.
3. Vous comprenez pourquoi et comment dégroupier les jeux de données groupés.
4. Vous pouvez utiliser `dplyr::n()` avec `group_by()`-`summarize()` pour compter les lignes par groupe.

5. Vous pouvez utiliser `sum()` avec `group_by()`-`summarize()` pour compter les lignes qui répondent à une condition.
6. Vous pouvez utiliser `dplyr::count()` comme une fonction pratique pour compter les lignes par groupe.

---

## Le jeu de données COVID-19 de Yaoundé

Dans cette leçon, nous allons à nouveau utiliser les données de l'enquête sérologique COVID-19 menée à Yaoundé, au Cameroun.

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

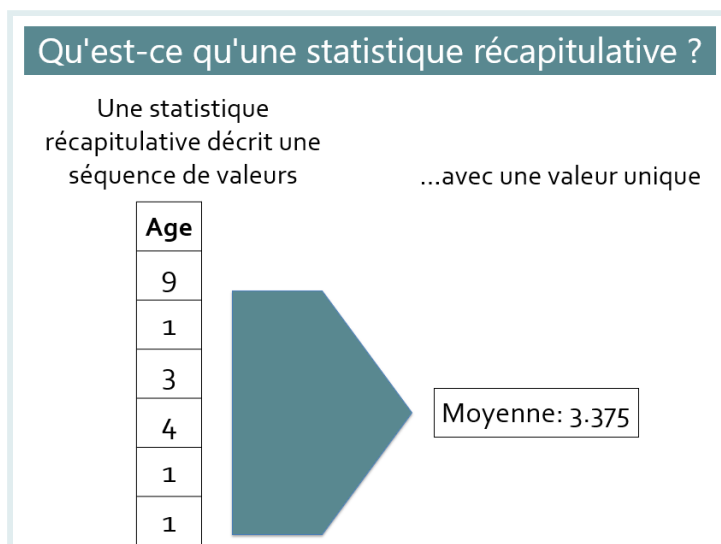
# ensemble plus petit de variables
yaounde %>% select(
  age_cat_3, sexe, poids_kg, taille_cm,
  quartier, fumeur, enceinte, occupation,
  saisons_traitement, symptomes, jours_absence_travail, jours_alite,
  aide, resultat_igg)
```

```
## # A tibble: 971 × 15
##   age_cat_3 sexe    poids_kg taille_cm quartier
##   <dbl> <chr>    <dbl>    <dbl> <chr>
## 1     45 Adult  Female     95     169 Briqueterie
## 2     55 Adult  Male       96     185 Briqueterie
## 3     23 Adult  Male       74     180 Briqueterie
## 4     20 Adult  Female     70     164 Briqueterie
## 5     55 Adult  Female     67     147 Briqueterie
## 6     17 Child  Female     65     162 Briqueterie
## 7     13 Child  Female     65     150 Briqueterie
## 8     28 Adult  Male       62     173 Briqueterie
## 9     30 Adult  Male       73     170 Briqueterie
## 10    13 Child  Female     56     153 Briqueterie
## # i 961 more rows
## # i 9 more variables: fumeur <chr>, enceinte <chr>, ...
```

Consultez la première leçon de ce chapitre pour plus d'informations sur ce jeu de données.

## Qu'est-ce qu'une statistique récapitulative ?

Une statistique récapitulative est une valeur unique (telle qu'une moyenne ou une médiane) qui décrit une séquence de valeurs (généralement une colonne dans votre jeu de données).



Les statistiques récapitulatives peuvent décrire le centre, la dispersion ou l'étendu d'une variable, ou les nombres et les positions des valeurs au sein de cette variable. Certaines statistiques récapitulatives courantes sont présentées dans le diagramme ci-dessous :

## Exemples de statistiques récapitulatives

```
age <- (9, 1, 4, 2, 2, 2)
```

Statistique récapitulative	Code R	Résultat
<b>Nombres</b>		
No. d'éléments	<code>dplyr::n(age)</code>	6
No. d'éléments distincts	<code>dplyr::n_distinct(age)</code>	4
<b>Position</b>		
Premier élément	<code>dplyr::first(age)</code>	9
Dernier élément	<code>dplyr::last(age)</code>	2
3 <sup>ème</sup> élément	<code>dplyr::nth(age, 3)</code>	4
<b>Centre</b>		
Moyenne	<code>mean(age)</code>	3.3
Médiane	<code>median(age)</code>	2
<b>Dispersion</b>		
Ecart-type	<code>sd(age)</code>	2.9
Ecart interquartile	<code>IQR(age)</code>	1.5
<b>Etendu</b>		
Minimum	<code>min(age)</code>	1
Maximum	<code>max(age)</code>	9
25 <sup>ème</sup> quantile	<code>quantile(age, 0.25)</code>	2

Le calcul des statistiques récapitulatives est une opération très courante dans la plupart des processus d'analyse de données, il sera donc important de devenir compétent pour les extraire de vos jeux de données. Et pour cette tâche, il n'y a pas de meilleur outil que la fonction `summarize()` de {dplyr} ! Alors voyons comment utiliser cette puissante fonction.

### Introduction à `dplyr::summarize()`

Pour commencer, il est préférable de voir d'abord comment obtenir des statistiques récapitulatives simples *sans* utiliser `summarize()`, puis nous verrons pourquoi vous devriez *réellement* utiliser `summarize()`.

Imaginez que l'on vous demande de trouver l'âge moyen des répondants dans le jeu de données `yao`. Comment pourriez-vous le faire en R de base ?

Tout d'abord, rappelons que la fonction du signe dollar, `$`, vous permet d'extraire une colonne d'un jeu de données vers un vecteur :

```
# extraire la colonne `age` de `yao`
```

Pour obtenir la moyenne, vous passez simplement ce vecteur `yao$age` dans la fonction `mean()` :

```
age)
```

```
## [1] 29.01751
```

Et c'est tout ! Vous avez maintenant une statistique récapitulative simple. Extrêmement facile, n'est-ce pas ?

Alors, pourquoi avons-nous besoin de `summarize()` pour obtenir des statistiques récapitulatives si le processus est déjà si simple sans lui ? Nous reviendrons sur la question du *pourquoi* bientôt. D'abord, voyons *comment* obtenir des statistiques récapitulatives avec `summarize()`.

En revenant à l'exemple précédent, la syntaxe correcte pour obtenir l'âge moyen avec `summarize()` serait :

```
size(mean_age = mean(age))
```

```
## # A tibble: 1 × 1  
##   mean_age  
##   <dbl>  
## 1      29.0
```

L'anatomie de cette syntaxe est présentée ci-dessous. Vous devez simplement entrer le nom de la nouvelle colonne (par exemple `mean_age`), la fonction récapitulative (par exemple `mean()`), et la colonne à résumer (par exemple `age`).

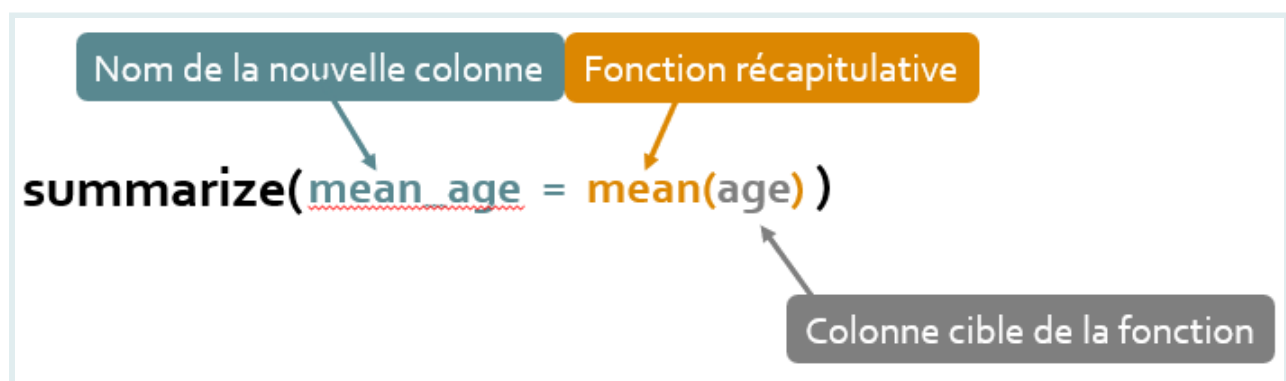


Fig. Syntaxe de base pour la fonction `summarize()`.

Vous pouvez également calculer plusieurs statistiques récapitulatives dans une seule commande `summarize()`. Par exemple, si vous vouliez à la fois l'âge moyen et l'âge médian, vous pourriez exécuter :

```
size(mean_age = mean(age),  
     median_age = median(age))
```

```
## # A tibble: 1 × 2
##   mean_age median_age
##   <dbl>      <dbl>
## 1      29.0         26
```

Sympa !

Maintenant, vous devriez vous demander pourquoi `summarize()` place les statistiques récapitulatives dans un jeu de données, avec chaque statistique dans une colonne différente.

Le principal avantage de cette structure de jeu de données est de faciliter la production de résumés *groupés* (et la création de tels résumés groupés sera le principal avantage de l'utilisation de `summarize()`).

Nous examinerons ces résumés groupés dans la section suivante. Pour l'instant, essayez de répondre aux questions de pratique ci-dessous.

### Pratique 1

Utilisez `summarize()` et les fonctions récapitulatives pertinentes pour obtenir la moyenne, la médiane et l'écart type des poids des répondants à partir de la variable `poids_kg` du jeu de données `yao`.

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

mean_poids_kg	median_poids_kg	sd_poids_kg
---------------	-----------------	-------------

>%

### Pratique 2



Utilisez `summarize()` et les fonctions récapitulatives pertinentes pour obtenir les tailles minimale et maximale des répondants à partir de la variable `taille_cm` du jeu de données `yao`.

Votre sortie doit être un jeu de données avec deux colonnes nommées comme indiqué ci-dessous :

min_taille_cm	max_taille_cm
---------------	---------------



## PRACTICE



>%

## Résumés groupés avec `dplyr::group_by()`

Comme son nom l'indique, `dplyr::group_by()` vous permet de regrouper un jeu de données par les valeurs d'une variable (par exemple le sexe masculin vs féminin). Vous pouvez ensuite effectuer des opérations qui sont divisées selon ces groupes.

Quel effet `group_by()` a-t-il sur un jeu de données ? Essayons de regrouper le jeu de données `yao` par sexe et observons l'effet :

```
_by(sexe)
```

```
## # A tibble: 971 × 15
## # Groups:   sexe [2]
##   age cat_age_3 sexe   poids_kg taille_cm quartier
##   <dbl> <chr>   <chr>   <dbl>   <dbl> <chr>
## 1    45 Adult   Female     95     169 Briqueterie
## 2    55 Adult   Male      96     185 Briqueterie
## 3    23 Adult   Male      74     180 Briqueterie
## 4    20 Adult   Female    70     164 Briqueterie
## 5    55 Adult   Female    67     147 Briqueterie
## 6    17 Child   Female    65     162 Briqueterie
## 7    13 Child   Female    65     150 Briqueterie
## 8    28 Adult   Male      62     173 Briqueterie
## 9    30 Adult   Male      73     170 Briqueterie
## 10   13 Child   Female    56     153 Briqueterie
## # i 961 more rows
## # i 9 more variables: fumeur <chr>, enceinte <chr>, ...
```

Hmm. Apparemment, rien ne s'est passé. La seule chose que vous *pourriez* remarquer est une nouvelle section dans l'en-tête qui vous indique la variable groupée—sex—et le nombre de groupes—2 :

```
# A tibble: 971 × 10
👉 # Groups:   sexe [2] 👉
```

Mis à part cet en-tête, cependant, le jeu de données semble inchangé.

Mais voyez ce qui se passe lorsque nous chaînons le `group_by()` avec l'appel `summarize()` que nous avons utilisé dans la section précédente :

```
by(sexe) %>%
  summarize(mean_age = mean(age))
```

```
## # A tibble: 2 × 2
##   sexe    mean_age
##   <chr>    <dbl>
## 1 Female    29.5
## 2 Male     28.4
```

Vous obtenez une statistique récapitulative différente pour chaque groupe ! Les statistiques pour les femmes sont dans une ligne et celles pour les hommes sont dans une autre. (À partir de ce jeu de données de sortie, vous pouvez dire par exemple que, l'âge moyen pour les répondantes est de 29.5, tandis que pour les répondants masculins, il est de 28.4)

Comme mentionné précédemment, ce type de résumé groupé est la raison principale pour laquelle la fonction `summarize()` est si utile !

Voyons un autre exemple d'une opération simple `group_by() + summarize()`.

Supposons que l'on vous ait demandé d'obtenir les poids maximum et minimum pour les individus dans différents quartiers dans le jeu de données `yao`. D'abord, vous feriez un `group_by()` sur la variable `neighbourhood`, puis vous appelleriez les fonctions `max()` et `min()` à l'intérieur de `summarize()` :

```
by(quartier) %>%
  summarize(max_poids = max(poids_kg),
            min_poids = min(poids_kg))
```

```
## # A tibble: 9 × 3
##   quartier    max_poids min_poids
##   <chr>        <dbl>    <dbl>
## 1 Briqueterie    128        20
## 2 Carriere       129        14
## 3 Cité Verte     118        16
## 4 Ekoudou        135        15
## 5 Messa          96        19
## 6 Mokolo         162        16
## 7 Nkomkana       161        15
## 8 Tsinga         105        15
## 9 Tsinga Oliga   100        17
```

Super ! Avec seulement quelques lignes de code, vous êtes capable d'extraire beaucoup d'informations.

Voyons encore un exemple pour faire bonne mesure. La variable `jours_absence_travail` nous indique le nombre de jours où les répondants ont été absents au travail en raison de symptômes similaires à ceux du COVID. Les individus qui n'ont signalé aucun symptôme semblable à celui du COVID ont un `NA` pour cette variable :

```
sum(jours_absence_travail)
```

```
## # A tibble: 971 × 1
##   jours_absence_travail
##   <dbl>
## 1 0
## 2 NA
## 3 NA
## 4 7
## 5 NA
## 6 7
## 7 0
## 8 0
## 9 0
## 10 NA
## # i 961 more rows
```

Pour compter le nombre total de jours de travail manqués pour chaque groupe de sexe, vous pourriez essayer d'exécuter la fonction `sum()` sur la variable `jours_absence_travail` :

```
sum(jours_absence_travail, na.rm = TRUE)
sum(jours_absence_travail)
```

```
## # A tibble: 2 × 2
##   sexe total_jours_absence
##   <chr> <dbl>
## 1 Female NA
## 2 Male NA
```

Hmmm. Cela vous donne des résultats `NA` car certaines lignes dans la colonne `jours_absence_travail` ont des `NA` en elles, et R ne peut pas trouver la somme de valeurs contenant un `NA`. Pour résoudre ce problème, l'argument `na.rm = TRUE` est nécessaire :

```
sum(jours_absence_travail, na.rm = TRUE)
sum(jours_absence_travail, na.rm = TRUE)
```

```
## # A tibble: 2 × 2
##   sexe    total_jours_absence
##   <chr>          <dbl>
## 1 Female            256
## 2 Male              272
```

La sortie nous dit qu'au total, parmi toutes les femmes de l'échantillon, 256 jours de travail ont été manqués en raison de symptômes similaires à ceux du COVID, et parmi tous les hommes, 272 jours.

J'espère que vous voyez maintenant pourquoi `summarize()` est si puissant. En combinaison avec `group_by()`, il vous permet d'obtenir des résumés de vos jeux de données groupés très informatifs avec très peu de lignes de code.

Produire de tels résumés est une partie très importante de la plupart des processus d'analyse de données, cette compétence sera donc probablement utile très prochainement !



**`summarize()` produit des “Tableaux croisés dynamiques”**

Les jeux de données récapitulatifs créés par `summarize()` sont souvent appelés des tableaux croisés dynamiques dans le contexte des logiciels de tableur comme Microsoft Excel.

### Pratique 3



Utilisez `group_by()` et `summarize()` pour obtenir le poids moyen (kg) en fonction du statut de fumeur dans le jeu de données `yao`.

Nommez la colonne de poids moyen `poids_moyen`

Le jeu de données de sortie doit ressembler à ceci :

fumeur	poids_moyen
Ex-fumeur	
Non-fumeur	
Fumeur	
NA	

## PRACTICE



(in RMD)

⌂

## Pratique 4

Utilisez `group_by()`, `summarize()` et les fonctions de statistiques récapitulatives pertinentes pour obtenir les tailles minimum et maximum pour chaque sexe dans le jeu de données `yao`.

## PRACTICE



(in RMD)

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

sexe	taille_min_cm	taille_max_cm
Female		
male		

⌂

## Pratique 5

Utilisez `group_by()`, `summarize()`, et la fonction `sum()` pour calculer le nombre total de jours alités (de la variable `jours_alite`) rapportés par les répondants de chaque sexe.

## PRACTICE



(in RMD)

Votre sortie doit être un jeu de données avec deux colonnes nommées comme indiqué ci-dessous :

sexe	total_jours_alités
Female	
Male	

⌂

## Regroupement par plusieurs variables (groupement imbriqué)

Il est possible de regrouper un jeu de données par plus d'une variable. Ceci est parfois appelé "groupement imbriqué".

Prenons un exemple. Supposons que vous voulez connaître l'âge moyen des hommes et des femmes *dans chaque quartier* (plutôt que l'âge moyen de *toutes* les femmes), vous pourriez mettre à la fois `sexe` et `quartier` dans l'instruction

`group_by()` :

```
by(sexe, quartier) %>%
size(age_moyen = mean(age))
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the `.groups` argument.
```

```
## # A tibble: 18 × 3
## # Groups:   sexe [2]
##   sexe    quartier    age_moyen
##   <chr> <chr>         <dbl>
## 1 Female Briqueterie    31.6
## 2 Female Carriere      28.2
## 3 Female Cité Verte    31.8
## 4 Female Ekoudou      29.3
## 5 Female Messa        30.2
## 6 Female Mokolo       28.0
## 7 Female Nkomkana     33.0
## 8 Female Tsinga       30.6
## 9 Female Tsinga Oliga  24.3
## 10 Male   Briqueterie    33.7
## 11 Male   Carriere      30.0
## 12 Male   Cité Verte    27.0
## 13 Male   Ekoudou      25.2
## 14 Male   Messa        23.9
## 15 Male   Mokolo       30.5
## 16 Male   Nkomkana     29.8
## 17 Male   Tsinga       28.8
## 18 Male   Tsinga Oliga  24.3
```

À partir de ce jeu de données de sortie, vous pouvez voir que, par exemple, les femmes de Briqueterie ont un âge moyen de 31,6 ans, tandis que les hommes de Briqueterie ont un âge moyen de 33,7 ans.

L'ordre des colonnes listées dans `group_by()` est interchangeable. Donc, si vous exécutez `group_by(quartier, sexe)` au lieu de `group_by(sexe, quartier)`, vous obtiendrez le même résultat, bien qu'il soit ordonné différemment :

```
by(quartier, sexe) %>%
size(age_moyen = mean(age))
```

```
## `summarise()` has grouped output by 'quartier'. You can
## override using the `.groups` argument.
```

```
## # A tibble: 18 × 3
## # Groups:   quartier [9]
##   quartier     sexe  age_moyen
##   <chr>        <chr>    <dbl>
## 1 Briqueterie Female    31.6
## 2 Briqueterie Male      33.7
## 3 Carriere    Female    28.2
## 4 Carriere    Male      30.0
## 5 Cité Verte  Female    31.8
## 6 Cité Verte  Male      27.0
## 7 Ekoudou     Female    29.3
## 8 Ekoudou     Male      25.2
## 9 Messa       Female    30.2
## 10 Messa      Male      23.9
## 11 Mokolo     Female    28.0
## 12 Mokolo     Male      30.5
## 13 Nkomkana   Female    33.0
## 14 Nkomkana   Male      29.8
## 15 Tsinga     Female    30.6
## 16 Tsinga     Male      28.8
## 17 Tsinga Oliga Female    24.3
## 18 Tsinga Oliga Male      24.3
```

Maintenant, l'ordre des colonnes est différent : `quartier` est la première colonne, et `sexe` est la deuxième. Et l'ordre des lignes est également différent : les lignes sont d'abord ordonnées par `quartier`, puis ordonnées par `sexe` à l'intérieur de chaque quartier.

Mais les statistiques de résumé sont les mêmes. Par exemple, vous pouvez à nouveau voir que les femmes de Briqueterie ont un âge moyen de 31,6 ans, tandis que les hommes de Briqueterie ont un âge moyen de 33,7 ans.

## Pratique 6

### PRACTICE



En utilisant le jeu de données `yao`, groupez vos données par `sexe` (`sexe`) et traitements (`combinaisons_de_traitements`) en utilisant `group_by`. Ensuite, en utilisant `summarize()` et la fonction de statistique récapitulative appropriée, calculez le poids moyen (`poids_kg`) pour chaque groupe.

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

sexe	combinaisons_de_traitements	poids_moyen_kg
------	-----------------------------	----------------

```
>%
```

#### PRACTICE



En utilisant le jeu de données `yao`, groupez vos données par catégorie d'âge (`categorie_d_age_3`), genre (`sexe`), et résultats d'IgG (`resultat_igg`) en utilisant `group_by`. Ensuite, en utilisant `summarize()` et la fonction de statistique récapitulative appropriée, calculez le nombre moyen de jours alités (`n_jours_alités`) pour chaque groupe.

Votre sortie doit être un jeu de données avec quatre colonnes nommées comme indiqué ci-dessous :

categorie_d_age_3	sexe	resultat_igg	moyenne_n_jours_alités
-------------------	------	--------------	------------------------

```
>%
```

## Dégroupement avec `dplyr::ungroup()` (pourquoi et comment)

Quand vous utilisez `group_by()` pour plus d'une variable avant d'utiliser `summarize()`, le jeu de données de sortie reste groupé. Ce regroupement persistant peut avoir des effets indésirables en aval, vous devrez donc parfois utiliser `dplyr::ungroup()` pour dégroupier les données avant de faire une analyse plus poussée.

Pour comprendre *pourquoi* vous devriez utiliser `ungroup()` sur les données, considérez d'abord l'exemple suivant, où nous ne regroupons qu'une seule variable avant de calculer une statistique récapitulative :

```
by(sexe) %>%
size(mean_age = mean(age))
```

```
## # A tibble: 2 × 2
##   sexe    mean_age
##   <chr>    <dbl>
```



```
## 1 Female      29.5
## 2 Male        28.4
```

Les données sont produites comme un jeu de données normal ; il n'est pas groupé. Vous pouvez le voir parce qu'il n'y a pas d'information sur les groupes dans l'en-tête.

Mais considérez maintenant quand vous regroupez par deux variables avant de calculer une statistique récapitulative :

```
by(sexe, quartier) %>%
  size(mean_age = mean(age))
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the `.groups` argument.
```

```
## # A tibble: 18 × 3
## # Groups:   sexe [2]
##   sexe    quartier    mean_age
##   <chr> <chr>         <dbl>
## 1 Female Briqueterie    31.6
## 2 Female Carriere      28.2
## 3 Female Cité Verte    31.8
## 4 Female Ekoudou       29.3
## 5 Female Messa         30.2
## 6 Female Mokolo        28.0
## 7 Female Nkomkana      33.0
## 8 Female Tsinga        30.6
## 9 Female Tsinga Oliga  24.3
## 10 Male  Briqueterie    33.7
## 11 Male  Carriere      30.0
## 12 Male  Cité Verte    27.0
## 13 Male  Ekoudou       25.2
## 14 Male  Messa         23.9
## 15 Male  Mokolo        30.5
## 16 Male  Nkomkana      29.8
## 17 Male  Tsinga        28.8
## 18 Male  Tsinga Oliga    24.3
```

Maintenant, l'en-tête vous indique que les données sont toujours groupées par la première variable dans `group_by()`, ici `sexe` :

```
# A tibble: 18 × 3
👉 # Groups:   sexe [2] 👉
```

Quelle est l'implication de ce regroupement persistant dans le jeu de données ? Cela signifie que le jeu de données peut montrer un comportement qui semble étrange lorsque vous essayez d'appliquer certaines fonctions {dplyr} dessus.

Par exemple, si vous essayez de `select()` une seule variable, peut-être la variable `mean_age`, vous devriez normalement pouvoir utiliser `select(mean_age)` :

```
by(sexe, quartier) %>%
size(mean_age = mean(age)) %>%
c(mean_age) # ne fonctionne pas comme prévu
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the `.groups` argument.
## Adding missing grouping variables: `sexe`
```

```
## # A tibble: 18 × 2
## # Groups:   sexe [2]
##   sexe    mean_age
##   <chr>    <dbl>
## 1 Female    31.6
## 2 Female    28.2
## 3 Female    31.8
## 4 Female    29.3
## 5 Female    30.2
## 6 Female    28.0
## 7 Female    33.0
## 8 Female    30.6
## 9 Female    24.3
## 10 Male     33.7
## 11 Male     30.0
## 12 Male     27.0
## 13 Male     25.2
## 14 Male     23.9
## 15 Male     30.5
## 16 Male     29.8
## 17 Male     28.8
## 18 Male     24.3
```

Mais comme vous pouvez le voir, la variable groupée, `sexe`, est *toujours* sélectionnée, même si nous n'avons demandé que `mean_age` dans l'instruction `select()`.

C'est l'un des nombreux exemples de comportements uniques des jeux de données groupés. D'autres verbes dplyr comme `filter()`, `mutate()` et `arrange()` agissent également de manière spéciale sur les données groupées. Nous aborderons cela en détail dans une leçon future.

Vous savez donc maintenant *pourquoi* vous devriez dégroupier les données lorsque vous n'en avez plus besoin. Voyons maintenant *comment* dégroupier les données. C'est assez simple : il suffit d'ajouter la fonction `ungroup()` à votre chaîne de pipe. Par exemple :

```
by(sexe, quartier) %>%
  summarise(mean_age = mean(age)) %>%
  print()
```

## `summarise()` has grouped output by 'sexe'. You can override  
## using the `.groups` argument.

```
## # A tibble: 18 × 3
##   sexe    quartier    mean_age
##   <chr>  <chr>         <dbl>
## 1 Female Briqueterie    31.6
## 2 Female Carriere      28.2
## 3 Female Cité Verte    31.8
## 4 Female Ekoudou       29.3
## 5 Female Messa         30.2
## 6 Female Mokolo        28.0
## 7 Female Nkomkana      33.0
## 8 Female Tsinga        30.6
## 9 Female Tsinga Oliga  24.3
## 10 Male   Briqueterie    33.7
## 11 Male   Carriere      30.0
## 12 Male   Cité Verte    27.0
## 13 Male   Ekoudou       25.2
## 14 Male   Messa         23.9
## 15 Male   Mokolo        30.5
## 16 Male   Nkomkana      29.8
## 17 Male   Tsinga        28.8
## 18 Male   Tsinga Oliga    24.3
```

Maintenant que le jeu de données est dégroupé, il se comportera à nouveau comme un jeu de données normal. Par exemple, vous pouvez `select()` n'importe quelle colonne(s) que vous voulez ; vous n'aurez pas certaines colonnes indésirables qui vous suivent :

```
by(sexe, quartier) %>%
  summarise(mean_age = mean(age)) %>%
  print() %>%
  select(mean_age)
```

## `summarise()` has grouped output by 'sexe'. You can override  
## using the `.groups` argument.

```
## # A tibble: 18 × 1
##   mean_age
##   <dbl>
## 1      31.6
```

```
## 2      28.2
## 3      31.8
## 4      29.3
## 5      30.2
## 6      28.0
## 7      33.0
## 8      30.6
## 9      24.3
## 10     33.7
## 11     30.0
## 12     27.0
## 13     25.2
## 14     23.9
## 15     30.5
## 16     29.8
## 17     28.8
## 18     24.3
```

## Comptage des lignes

Vous pouvez faire beaucoup de science des données en *comptant* simplement et occasionnellement en *divisant*. - Hadley Wickham, Scientifique Senior chez RStudio

Une tâche courante de statistique récapitulative des données est de compter combien d'observations (lignes) il y a pour chaque groupe. Vous pouvez y parvenir avec la fonction spéciale `n()` de {dplyr}, qui est spécifiquement conçue pour être utilisée dans `summarise()`.

Par exemple, si vous voulez compter combien d'individus se trouvent dans chaque groupe de quartier, vous exécuteriez :

```
by(quartier) %>%
  size(nombre = n())
```

```
## # A tibble: 9 × 2
##   quartier      nombre
##   <chr>         <int>
## 1 Briqueterie    106
## 2 Carriere      236
## 3 Cité Verte     72
## 4 Ekoudou       190
## 5 Messa         48
## 6 Mokolo        96
## 7 Nkomkana      75
## 8 Tsinga        81
## 9 Tsinga Oliga   67
```

Comme vous pouvez le voir, la fonction `n()` ne nécessite aucun argument. Elle “connait son travail” dans le jeu de données !

Bien sûr, vous pouvez inclure d’autres statistiques récapitulatives dans le même appel `summarize()`. Par exemple, ci-dessous, nous calculons également l’âge moyen par quartier.

```
by(quartier) %>%
  summarize(nombre = n(),
            mean_age = mean(age))
```

```
## # A tibble: 9 × 3
##   quartier      nombre mean_age
##   <chr>         <int>   <dbl>
## 1 Briqueterie     106     32.5
## 2 Carriere       236     28.9
## 3 Cité Verte      72     29.9
## 4 Ekoudou       190     27.6
## 5 Messa          48     27.3
## 6 Mokolo         96     29.1
## 7 Nkomkana       75     31.7
## 8 Tsinga         81     29.7
## 9 Tsinga Oliga   67     24.3
```

## Pratique 7

Groupez votre jeu de données `yao` par l’occupation des répondants (`occupation`) et utilisez `summarize()` pour créer des colonnes qui montrent :



- combien d’individus il y a avec chaque occupation (pensez à la fonction `n()`)
- le nombre moyen de jours de travail manqués (`jours_absence_travail`) par ceux ayant cette occupation

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

occupation	count	mean_jours_absence_travail
------------	-------	----------------------------

```
>%
```

## Compter les lignes qui répondent à une condition

Plutôt que de compter *toutes* les lignes comme ci-dessus, il est parfois plus utile de compter seulement les lignes qui répondent à des conditions spécifiques. Cela peut être fait facilement en plaçant les conditions requises dans la fonction `sum()`.

Par exemple, pour compter le nombre de personnes de moins de 18 ans dans chaque quartier, vous placez la condition `age < 18` à l'intérieur de `sum()` :

```
by(quartier) %>%  
size(nombre_inferieur_18 = sum(age < 18))
```

```
## # A tibble: 9 × 2  
##   quartier      nombre_inferieur_18  
##   <chr>                <int>  
## 1 Briqueterie          28  
## 2 Carriere             58  
## 3 Cité Verte           19  
## 4 Ekoudou             66  
## 5 Messa                18  
## 6 Mokolo               32  
## 7 Nkomkana             22  
## 8 Tsinga               23  
## 9 Tsinga Oliga        25
```

De même, pour compter le nombre de personnes ayant un doctorat dans chaque quartier, vous placez la condition `edu_haute == "Doctorate"` à l'intérieur de `sum()` :

```
by(quartier) %>%  
size(nombre_avec_doctorates = sum(edu_haute == "Doctorate"))
```

```
## # A tibble: 9 × 2  
##   quartier      nombre_avec_doctorates  
##   <chr>                <int>  
## 1 Briqueterie          2  
## 2 Carriere             1  
## 3 Cité Verte           1  
## 4 Ekoudou              1  
## 5 Messa                2  
## 6 Mokolo               0  
## 7 Nkomkana             4  
## 8 Tsinga               3  
## 9 Tsinga Oliga        3
```

### CHALLENGE



## Sous le capot : compter avec des conditions

Pourquoi pouvez-vous utiliser `sum()`, qui est censé ajouter des nombres, sur une condition comme `edu_haute == "Doctorate"` ?

Utiliser `sum()` sur une condition fonctionne parce que la condition évalue aux valeurs booléennes `TRUE` et `FALSE`. Et ces valeurs booléennes sont traitées comme des nombres (où `TRUE` est égal à 1 et `FALSE` est égal à 0), et les nombres peuvent, bien sûr, être sommés.

Le code ci-dessous démontre ce qui se passe sous le capot de manière étape par étape. Exécutez-le et voyez si vous pouvez suivre.

```
condition_sums <- yao %>%
  filter(edu_haute) %>%
  mutate(avec_doctorate = edu_haute == "Doctorate") %>%
  mutate(numerique_avec_doctorate = as.numeric(avec_doctorate))

condition_sums
```

### CHALLENGE



```
## # A tibble: 971 × 3
##   edu_haute   avec_doctorate numerique_avec_doctorate
##   <chr>      <lgl>                <dbl>
## 1 Secondary FALSE                      0
## 2 University FALSE                      0
## 3 University FALSE                      0
## 4 Secondary FALSE                      0
## 5 Primary   FALSE                      0
## 6 Secondary FALSE                      0
## 7 Secondary FALSE                      0
## 8 Doctorate TRUE                       1
## 9 Secondary FALSE                      0
## 10 Secondary FALSE                      0
## # i 961 more rows
```

Les valeurs numériques peuvent ensuite être ajoutées pour produire un décompte des lignes remplissant la condition `edu_haute == "Doctorate"` :

```
condition_sums %>%
  summarise(nombre_avec_doctorate = sum(numerique_avec_doctorate))
```

```
## # A tibble: 1 × 1
##   nombre_avec_doctorate
```

## CHALLENGE



```
## <dbl>
## 1 17
```

Pour une illustration finale du comptage avec des conditions, considérez la variable `combinaisons_traitement`, qui liste les traitements reçus par les personnes présentant des symptômes similaires à ceux du COVID. Les personnes qui n'ont reçu aucun traitement ont une valeur `NA` :

```
table(combinaisons_traitement)
```

```
## # A tibble: 971 × 1
##   combinaisons_traitement
##   <chr>
## 1 Paracetamol
## 2 <NA>
## 3 <NA>
## 4 Antibiotics
## 5 <NA>
## 6 Paracetamol--Antibiotics
## 7 Traditional meds.
## 8 Paracetamol
## 9 Paracetamol--Traditional meds.
## 10 <NA>
## # i 961 more rows
```

Si vous voulez compter le nombre de personnes qui n'ont reçu *aucun traitement*, vous additionneriez celles qui répondent à la condition

`is.na(combinaisons_traitement)` :

```
table(
  by(quartier) %>%
  size(traitement_inconnu = sum(is.na(combinaisons_traitement)))
)
```

```
## # A tibble: 9 × 2
##   quartier      traitement_inconnu
##   <chr>          <int>
## 1 Briqueterie      82
## 2 Carriere        192
## 3 Cité Verte       46
## 4 Ekoudou         133
## 5 Messa           35
## 6 Mokolo          65
## 7 Nkomkana        53
## 8 Tsinga          56
## 9 Tsinga Oliga     47
```



Ce sont les personnes ayant des valeurs `NA` pour la colonne `combinaisons_traitement`.

Pour compter les personnes qui *ont* reçu un traitement, vous pouvez simplement nier la fonction `is.na()` avec `!`:

```
group_by(quartier) %>%
  size(traitement_connu = sum(!is.na(combinaisons_traitement)))
```

```
## # A tibble: 9 × 2
##   quartier      traitement_connu
##   <chr>          <int>
## 1 Briqueterie      24
## 2 Carriere        44
## 3 Cité Verte      26
## 4 Ekoudou         57
## 5 Messa           13
## 6 Mokolo           31
## 7 Nkomkana         22
## 8 Tsinga           25
## 9 Tsinga Oliga     20
```

VEUILLEZ IGNORER LA QUESTION PRATIQUE SUR LA VÉRIFICATION DES SYMPTÔMES CHEZ LES ADULTES. NOUS AVONS DÉCIDÉ DE LA SUPPRIMER.

### `dplyr::count()`

La fonction `dplyr::count()` regroupe plusieurs choses en une seule ligne de code conviviale pour vous aider à trouver les comptages d'observations par groupe.

Utilisons `dplyr::count()` sur notre variable `occupation`:

```
(occupation)
```

```
## # A tibble: 28 × 2
##   occupation      n
##   <chr>          <int>
## 1 Farmer          5
## 2 Farmer--Other    1
## 3 Home-maker      65
## 4 Home-maker--Farmer 2
## 5 Home-maker--Informal worker 3
## 6 Home-maker--Informal worker--Farmer 1
## 7 Home-maker--Trader 3
## 8 Informal worker 189
## 9 Informal worker--Other 2
## 10 Informal worker--Trader 4
## # i 18 more rows
```

Notez que c'est la même sortie que :

```
by(occupation) %>%
size(n = n())
```

```
## # A tibble: 28 × 2
##   occupation      n
##   <chr>      <int>
## 1 Farmer          5
## 2 Farmer--Other    1
## 3 Home-maker      65
## 4 Home-maker--Farmer 2
## 5 Home-maker--Informal worker 3
## 6 Home-maker--Informal worker--Farmer 1
## 7 Home-maker--Trader 3
## 8 Informal worker 189
## 9 Informal worker--Other 2
## 10 Informal worker--Trader 4
## # i 18 more rows
```

Vous pouvez également appliquer `dplyr::count()` de manière imbriquée :

```
(sexe, occupation)
```

```
## # A tibble: 40 × 3
##   sexe occupation      n
##   <chr> <chr>      <int>
## 1 Female Farmer          3
## 2 Female Home-maker      65
## 3 Female Home-maker--Farmer 2
## 4 Female Home-maker--Informal worker 3
## 5 Female Home-maker--Informal worker--Farmer 1
## 6 Female Home-maker--Trader 3
## 7 Female Informal worker 77
## 8 Female Informal worker--Trader 1
## 9 Female No response      8
## 10 Female Other           6
## # i 30 more rows
```

## PRACTICE



(in RMD)

## Pratique 8

Le verbe `count()` vous donne des informations clés sur votre ensemble de données de manière très rapide. Regardons nos

résultats IgG stratifiés par catégorie d'âge et par sexe en une seule ligne de code.

En utilisant le jeu de données `yao`, comptez les différentes combinaisons de genre (`sex`), de catégories d'âge (`cat_age_3`) et de résultats IgG (`resultat_igg`).

Votre sortie doit être un jeu de données avec quatre colonnes nommées comme indiqué ci-dessous :

sexe	cat_age_3	resultat_igg	n
------	-----------	--------------	---

#### PRACTICE



(in RMD)

```
>%
```

En utilisant le jeu de données `yao`, comptez les différentes combinaisons de catégories d'âge (`cat_age_3`) et de nombre de jours alités (`jours_alite`).

Votre sortie doit être un jeu de données avec trois colonnes nommées comme indiqué ci-dessous :

cat_age_3	jours_alite	n
-----------	-------------	---

```
_bedridden_age_categories <-  
>%
```

L'inconvénient de `count()` est qu'il ne peut vous donner qu'une seule statistique récapitulative dans le jeu de données. Lorsque vous utilisez `summarize()` et `n()`, vous pouvez inclure plusieurs statistiques récapitulatives. Par exemple :

```
by(sexe, quartier) %>%  
size(count = n(),  
      median_age = median(age))
```

```
## `summarise()` has grouped output by 'sexe'. You can override  
## using the `.groups` argument.
```

```
## # A tibble: 18 × 4  
## # Groups:   sexe [2]  
##   sexe    quartier    count median_age  
##   <chr>  <chr>      <int>      <dbl>  
## 1 Female Briqueterie     61         28
```

```
## 2 Female Carriere      140      25.5
## 3 Female Cité Verte    44       28
## 4 Female Ekoudou      110      26.5
## 5 Female Messa        26      27.5
## 6 Female Mokolo       53       23
## 7 Female Nkomkana     43       28
## 8 Female Tsinga       42       29
## 9 Female Tsinga Oliga 30      23.5
## 10 Male  Briqueterie   45       28
## 11 Male  Carriere     96       27
## 12 Male  Cité Verte   28      22.5
## 13 Male  Ekoudou     80      21.5
## 14 Male  Messa       22      24.5
## 15 Male  Mokolo      43       32
## 16 Male  Nkomkana    32       27
## 17 Male  Tsinga     39       27
## 18 Male  Tsinga Oliga 37       21
```

Mais `count()` ne peut produire que des comptages :

```
by(sexe, quartier) %>%
  count()
```

```
## # A tibble: 18 × 3
## # Groups:   sexe, quartier [18]
##   sexe    quartier      n
##   <chr>  <chr>      <int>
## 1 Female Briqueterie    61
## 2 Female Carriere     140
## 3 Female Cité Verte    44
## 4 Female Ekoudou     110
## 5 Female Messa        26
## 6 Female Mokolo       53
## 7 Female Nkomkana     43
## 8 Female Tsinga       42
## 9 Female Tsinga Oliga 30
## 10 Male  Briqueterie   45
## 11 Male  Carriere     96
## 12 Male  Cité Verte   28
## 13 Male  Ekoudou     80
## 14 Male  Messa       22
## 15 Male  Mokolo      43
## 16 Male  Nkomkana    32
## 17 Male  Tsinga     39
## 18 Male  Tsinga Oliga 37
```

## Inclure les combinaisons manquantes dans les statistiques récapitulatives

Lorsque vous utilisez `group_by()` et `summarize()` sur plusieurs variables, vous obtenez une statistique récapitulative pour chaque combinaison unique des variables groupées. Par exemple, considérez le code et la sortie ci-dessous, qui comptent le nombre d'individus dans chaque groupe d'âge et de sexe :

```
group_by(sexe, cat_age_3) %>%
  summarise(nombre_d_individus = n())
```

## `summarise()` has grouped output by 'sexe'. You can override  
## using the ``.groups` argument.

## # A tibble: 6 × 3  
## # Groups: sexe [2]  
## sexe cat\_age\_3 nombre\_d\_individus  
## <chr> <chr> <int>  
## 1 Female Adult 368  
## 2 Female Child 155  
## 3 Female Senior 26  
## 4 Male Adult 267  
## 5 Male Child 136  
## 6 Male Senior 19

Dans le jeu de données de sortie, il y a une ligne pour chaque combinaison de sexe et de groupe d'âge (Femme—Adulte, Femme—Enfant, etc.).

Mais que se passe-t-il si l'une de ces combinaisons n'est pas présente dans les données ?

Créons un exemple artificiel pour observer cela. Avec le code ci-dessous, nous supprimons artificiellement tous les enfants de sexe masculin du jeu de données `yao` :

```
yao_no_male_children <-
  yao %>%
  filter(!(sexe == "Male" & cat_age_3 == "Child"))
```

Maintenant, si vous exécutez le même appel `group_by()` et `summarize()` sur `yao_no_male_children`, vous remarquerez la combinaison manquante :

```
yao_no_male_children %>%
  group_by(sexe, cat_age_3) %>%
  summarise(number_of_individuals = n())
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the `.groups` argument.
```

```
## # A tibble: 5 × 3
## # Groups:   sexe [2]
##   sexe   cat_age_3 number_of_individuals
##   <chr>  <chr>                <int>
## 1 Female Adult                368
## 2 Female Child                155
## 3 Female Senior                26
## 4 Male   Adult                267
## 5 Male   Senior                19
```

En effet, il n'y a pas de ligne pour les enfants de sexe masculin.

Mais parfois, il est utile d'inclure ces combinaisons manquantes dans le jeu de données de sortie, avec une valeur NA ou 0 pour la statistique récapitulative.

Pour ce faire, vous pouvez exécuter le code suivant à la place :

```
male_children %>%
  # Convertir les variables en facteurs
  mutate(sexe = as.factor(sexe),
         cat_age_3 = as.factor(cat_age_3)) %>%
  # Supprimer l'argument .drop = FALSE
  summarise(number_of_individuals = n())
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the `.groups` argument.
```

```
## # A tibble: 6 × 3
## # Groups:   sexe [2]
##   sexe   cat_age_3 number_of_individuals
##   <fct>  <fct>                <int>
## 1 Female Adult                368
## 2 Female Child                155
## 3 Female Senior                26
## 4 Male   Adult                267
## 5 Male   Child                 0
## 6 Male   Senior                19
```

Que fait ce code ?

- D'abord, il convertit les variables de regroupement en facteurs avec `as.factor()` (dans un appel à `mutate()`)

- Ensuite, il utilise l'argument `.drop = FALSE` dans la fonction `group_by()` pour éviter de supprimer les combinaisons manquantes.

Vous avez maintenant un compte clair de 0 pour le nombre d'enfants de sexe masculin !

Voyons un autre exemple, cette fois sans modifier artificiellement nos données.

Le code ci-dessous calcule l'âge moyen par sexe et par niveau d'éducation :

```
group_by(sexe, edu_haute) %>%
  summarise(mean_age = mean(age))
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the ``.groups` argument.
```

```
## # A tibble: 13 × 3
## # Groups:   sexe [2]
##   sexe    edu_haute    mean_age
##   <chr>   <chr>         <dbl>
## 1 Female Doctorate      28
## 2 Female No formal instruction 45.6
## 3 Female No response      35
## 4 Female Primary        26.8
## 5 Female Secondary      28.8
## 6 Female University      31.5
## 7 Male   Doctorate      42.2
## 8 Male   No formal instruction 37.9
## 9 Male   No response      22
## 10 Male   Other          5.5
## 11 Male   Primary        22.9
## 12 Male   Secondary      29.4
## 13 Male   University      31.9
```

Remarquez que dans le jeu de données de sortie, il y a 7 lignes pour les hommes mais seulement 6 lignes pour les femmes, car aucune femme n'a répondu "Autre" à la question sur le niveau d'éducation le plus élevé.

Si vous voulez néanmoins inclure la ligne "Femme—Autre" dans le jeu de données de sortie, vous exécuteriez :

```
df %>%
  mutate(sexe = as.factor(sexe),
         edu_haute = as.factor(edu_haute)) %>%
  group_by(sexe, edu_haute, .drop = FALSE) %>%
  summarise(mean_age = mean(age))
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the `.groups` argument.
```

```
## # A tibble: 14 × 3
## # Groups:   sexe [2]
##   sexe   edu_haute      mean_age
##   <fct> <fct>         <dbl>
## 1 Female Doctorate          28
## 2 Female No formal instruction 45.6
## 3 Female No response         35
## 4 Female Other              NaN
## 5 Female Primary           26.8
## 6 Female Secondary         28.8
## 7 Female University        31.5
## 8 Male   Doctorate          42.2
## 9 Male   No formal instruction 37.9
## 10 Male  No response         22
## 11 Male  Other              5.5
## 12 Male  Primary           22.9
## 13 Male  Secondary         29.4
## 14 Male  University        31.9
```

## Pratique 9

En utilisant le jeu de données `yao`, calculons l'âge médian en regroupant par quartier, catégorie d'âge et sexe.

Notez que nous voulons toutes les combinaisons possibles de ces trois variables (pas seulement celles présentes dans nos données).

### PRACTICE



(in RMD)

Faites attention à deux impératifs de préparation des données !

- convertissez vos variables de regroupement en facteurs au préalable en utilisant `mutate()`
- calculez votre statistique, la médiane, tout en supprimant les valeurs `NA`.

Votre sortie doit être un jeu de données avec quatre colonnes nommées comme indiqué ci-dessous :

quartier	cat_age_3	sexe	median_age
----------	-----------	------	------------



## PRACTICE



(in RMD)

%>%

## Pourquoi inclure les combinaisons manquantes ?

Ci-dessus, nous avons mentionné que l'inclusion de combinaisons manquantes est souvent utile dans le processus d'analyse de données. Voyons un cas d'utilisation : la conception des graphiques avec {ggplot}. Si vous n'avez pas encore appris {ggplot}, ce n'est pas grave, concentrez-vous simplement sur les sorties de graphique.

Pour réaliser un diagramme à barres composé avec les comptes d'âge-sexe de `yao_no_male_children`, vous pourriez exécuter :

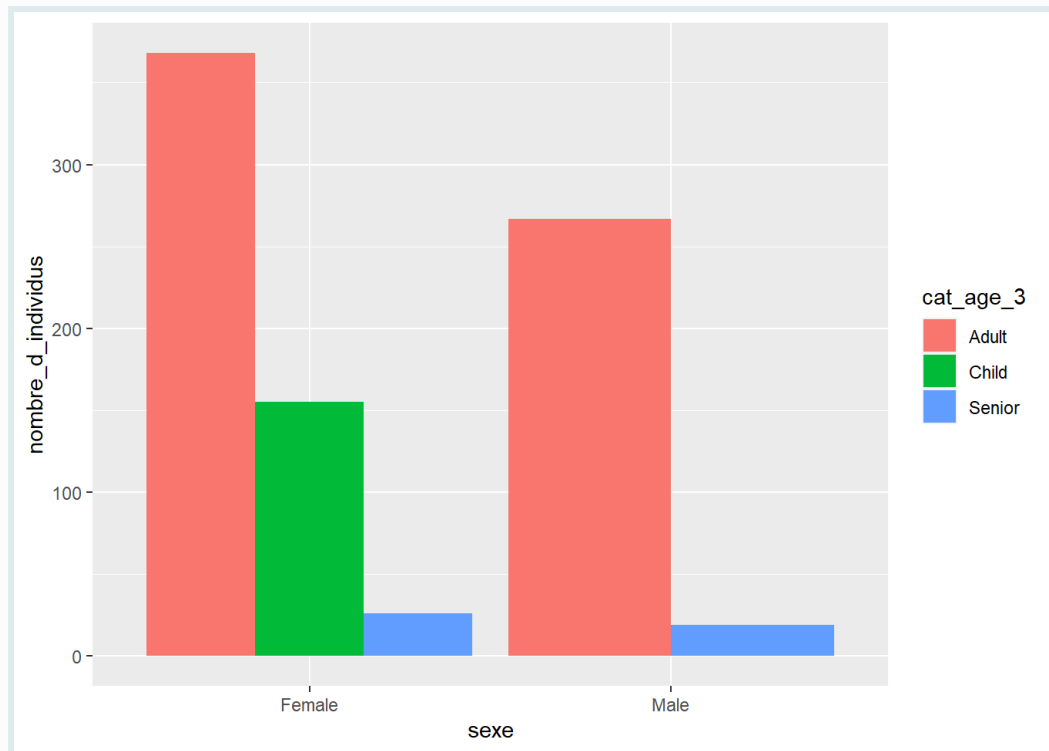
## SIDE NOTE



```
yao_no_male_children %>%  
  summarise(n = sum(is.na(cat_age_3))) %>%  
  filter(n > 0) %>%  
  print()
```

```
# Insérer la sortie à ggplot  
ggplot() +  
  geom_bar(aes(x = sexe, y = nombre_d_individus, fill = cat_age_3),  
            position = "dodge")
```

```
## `summarise()` has grouped output by 'sexe'. You can  
## override  
## using the `.groups` argument.
```



#### SIDE NOTE



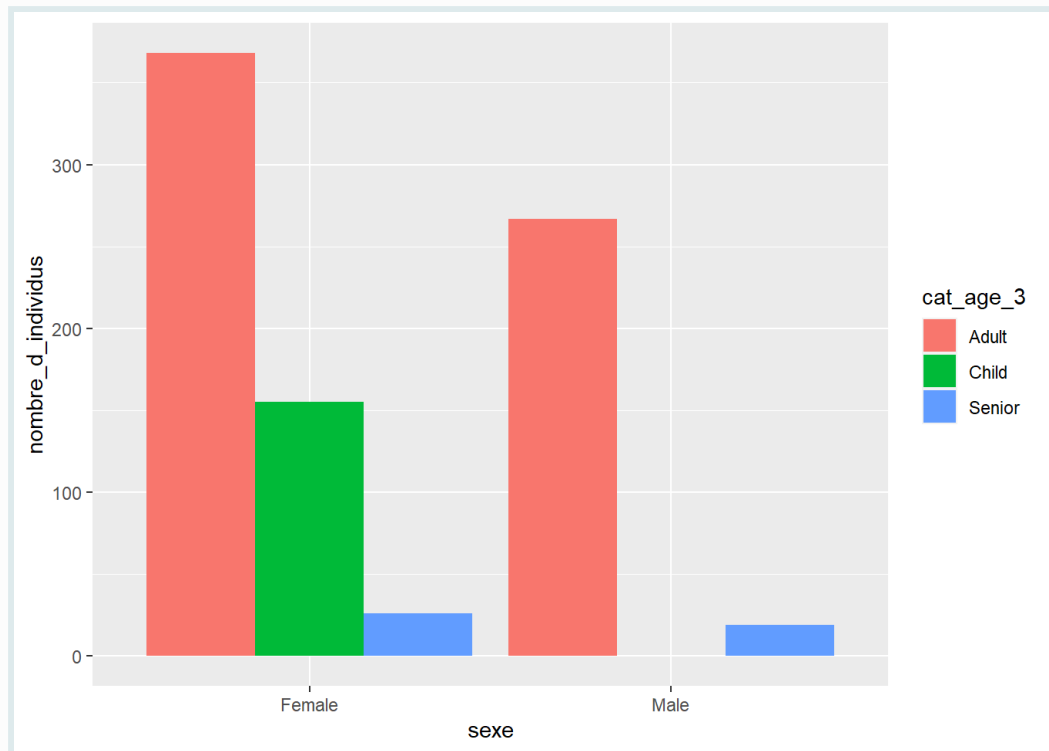
Pas très élégant ! Idéalement, il devrait y avoir un espace vide indiquant 0 pour le nombre d'enfants de sexe masculin.

Si vous mettez en œuvre la procédure pour inclure les combinaisons manquantes, vous obtenez un diagramme à barres composé plus naturel, avec un espace vide pour les enfants de sexe masculin :

```
male_children %>%
  summarise(sexe = as.factor(sexe),
            cat_age_3 = as.factor(cat_age_3)) %>%
  group_by(sexe, cat_age_3, .drop = FALSE) %>%
  summarise(nombre_d_individus = n()) %>%
  arrange(sexe, cat_age_3) %>%
  drop_na() %>%

# mettre la sortie à ggplot
ggplot() +
  geom_bar(aes(x = sexe, y = nombre_d_individus, fill = cat_age_3),
           position = "dodge")
```

```
## `summarise()` has grouped output by 'sexe'. You can
## override
## using the `.groups` argument.
```



#### SIDE NOTE



Beaucoup mieux !

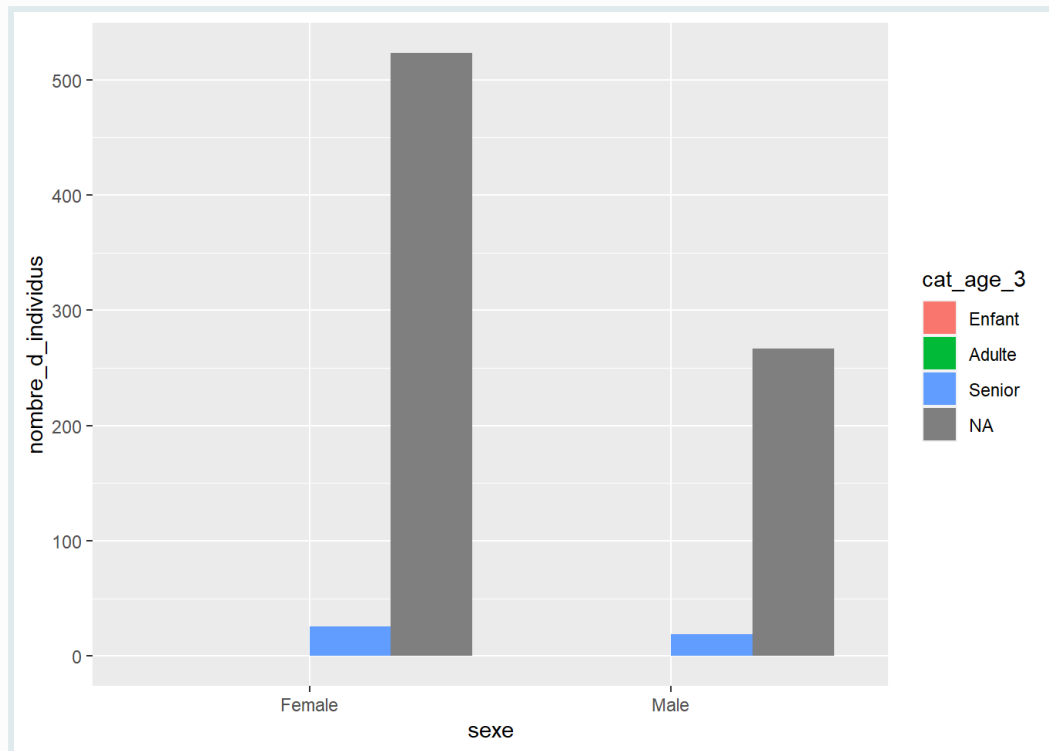
Au fait, cette sortie peut être légèrement améliorée en définissant les niveaux de facteur pour l'âge dans leur ordre croissant correct : d'abord "Enfant", puis "Adulte" puis "Senior" :

```
male_children %>%
  summarise(sexe = as.factor(sexe),
            cat_age_3 = factor(cat_age_3,
                              levels = c("Enfant",
                                           "Adulte",
                                           "Senior"))) %>%
  group_by(sexe, cat_age_3, .drop = FALSE) %>%
  summarise(nombre_d_individus = n()) %>%
  arrange() %>%

# remettre la sortie à ggplot
ggplot() +
  geom_col(aes(x = sexe, y = nombre_d_individus, fill = cat_age_3),
           position = "dodge")
```

```
## `summarise()` has grouped output by 'sexe'. You can
## override
## using the `.groups` argument.
```

#### SIDE NOTE



## Conclusion

Vous avez maintenant vu comment obtenir des statistiques récapitulatives rapides à partir de vos données, soit pour l'exploration de données, soit pour une présentation ou une visualisation de données supplémentaires.

De plus, vous avez découvert l'une des merveilles de {dplyr}, la possibilité de grouper vos données à l'aide de `group_by()`.

`group_by()` combiné avec `summarize()` est l'une des manipulations de regroupement les plus courantes.

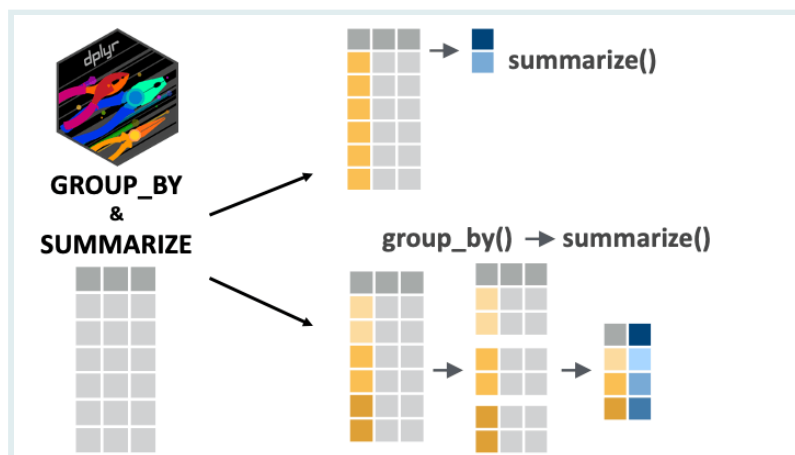


Fig: `summarize()` et son utilisation combinée avec `group_by()`.

Cependant, vous pouvez également combiner `group_by()` avec de nombreux autres verbes `{dplyr}` : c'est ce que nous couvrirons dans notre prochaine leçon. À bientôt !

---

## Contributeurs

Les membres suivants de l'équipe 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



### ANDREE VALLE CAMPOS

R Developer and Instructor, the GRAPH Network

Motivated by reproducible science and education



### KENE DAVID NWOSU

Data analyst, the GRAPH Network

Passionate about world improvement

Merci à [Alice Osmaston](#) et [Saifeldin Shehata](#) pour leurs commentaires et leur revue.

---

## Références

Certaines informations de cette leçon ont été adaptées des sources suivantes :

- Horst, A. (2022). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Travail original publié en 2020)
- *Group by one or more variables*. (s.d.). Consulté le 21 février 2022, sur [https://dplyr.tidyverse.org/reference/group\\_by.html](https://dplyr.tidyverse.org/reference/group_by.html)
- *Summarise each group to fewer rows*. (s.d.). Consulté le 21 février 2022, sur <https://dplyr.tidyverse.org/reference/summarize.html>
- The Carpentries. (s.d.). *Grouped operations using `dplyr`*. Grouped operations using `dplyr` – Introduction to R/tidyverse for Exploratory Data Analysis.

Consulté le 28 juillet 2022, sur [https://tavareshugo.github.io/r-intro-tidyverse-gapminder/06-grouped\\_operations\\_dplyr/index.html](https://tavareshugo.github.io/r-intro-tidyverse-gapminder/06-grouped_operations_dplyr/index.html)

L'œuvre d'art a été adaptée de :

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

---

## Solutions des exercices pratiques

### Solution exercice pratique 1

```
size(mean_poids_kg = mean(poids_kg),
     median_poids_kg = median(poids_kg),
     sd_poids_kg = sd(poids_kg))

## # A tibble: 1 × 3
##   mean_poids_kg median_poids_kg sd_poids_kg
##           <dbl>           <dbl>      <dbl>
## 1          64.4             66         23.1
```

### Solution exercice pratique 2

```
size(min_taille_cm = min(taille_cm),
     max_taille_cm = max(taille_cm))

## # A tibble: 1 × 2
##   min_taille_cm max_taille_cm
##           <dbl>      <dbl>
## 1           54         196
```

### Solution exercice pratique 3

```
by(fumeur) %>%
  size(poids_mean = mean(poids_kg))

## # A tibble: 4 × 2
##   fumeur      poids_mean
##   <chr>          <dbl>
## 1 Ex-smoker      76.4
## 2 Non-smoker     63.0
```

```
## 3 Smoker          72.4
## 4 <NA>            73
```

### Solution exercice pratique 4

```
by(sexe) %>%
  summarise(min_taille_cm = min(taille_cm),
            max_taille_cm = max(taille_cm))
```

```
## # A tibble: 2 × 3
##   sexe   min_taille_cm max_taille_cm
##   <chr>         <dbl>         <dbl>
## 1 Female           54           185
## 2 Male            96           196
```

### Solution exercice pratique 5

```
by(sexe) %>%
  summarise(total_jours_alite = sum(jours_alite, na.rm = T))
```

```
## # A tibble: 2 × 2
##   sexe   total_jours_alite
##   <chr>         <dbl>
## 1 Female           175
## 2 Male            163
```

### Solution exercice pratique 6\_1

```
by(sexe, combinaisons_traitement) %>%
  summarise(mean_poids_kg = mean(poids_kg, na.rm = T))
```

```
## `summarise()` has grouped output by 'sexe'. You can override
## using the `.groups` argument.
```

```
## # A tibble: 47 × 3
## # Groups:   sexe [2]
##   sexe   combinaisons_traitement   mean_poids_kg
##   <chr> <chr>                        <dbl>
## 1 Female Antibiotics                54.7
## 2 Female Antibiotics--Other anti-inflamm. 67
## 3 Female Hydrocortisone             59.7
## 4 Female Other                      68
## 5 Female Other anti-inflamm.        71.5
## 6 Female Paracetamol                74.4
```

```
## 7 Female Paracetamol--Antibiotics 69
## 8 Female Paracetamol--Antibiotics--Hydrocort... 54
## 9 Female Paracetamol--Antibiotics--Other 65.9
## 10 Female Paracetamol--Antibiotics--Other ant... 38.7
## # i 37 more rows
```

## Solution exercice pratique 6\_2

```
by(cat_age_3, sexe, resultat_igg) %>%
  summarise(mean_jours_alite = mean(jours_alite, na.rm = T))
```

## `summarise()` has grouped output by 'cat\_age\_3', 'sexe'. You  
## can override using the `.groups` argument.

```
## # A tibble: 12 × 4
## # Groups:   cat_age_3, sexe [6]
##   cat_age_3 sexe   resultat_igg mean_jours_alite
##   <chr>    <chr> <chr>          <dbl>
## 1 Adult    Female Negative      1.07
## 2 Adult    Female Positive    0.976
## 3 Adult    Male   Negative    0.889
## 4 Adult    Male   Positive    1.59
## 5 Child    Female Negative    0.731
## 6 Child    Female Positive    1.36
## 7 Child    Male   Negative    1.44
## 8 Child    Male   Positive    1.12
## 9 Senior   Female Negative      1
## 10 Senior  Female Positive      3
## 11 Senior  Male   Negative      3
## 12 Senior  Male   Positive      2
```

## Solution exercice pratique 7

```
by(occupation) %>%
  summarise(count = n(),
            mean_jours_absence_travail = mean(jours_absence_travail, na.rm=TRUE))
```

```
## # A tibble: 28 × 3
##   occupation count mean_jours_abse...1
##   <chr>      <int>      <dbl>
## 1 Farmer      5          0
## 2 Farmer--Other 1          2
## 3 Home-maker  65         0.929
## 4 Home-maker--Farmer 2      NaN
## 5 Home-maker--Informal worker 3      NaN
## 6 Home-maker--Informal worker--Far... 1      NaN
## 7 Home-maker--Trader 3          0
## 8 Informal worker 189        2.17
```



```
## 9 Informal worker--Other 2 0
## 10 Informal worker--Trader 4 0
## # i 18 more rows
## # i abbreviated name: 'mean_jours_absence_travail'
```

### Solution exercice pratique 8\_1

```
(sexe, cat_age_3, resultat_igg)
```

```
## # A tibble: 12 × 4
##   sexe   cat_age_3 resultat_igg     n
##   <chr> <chr>      <chr>      <int>
## 1 Female Adult    Negative    265
## 2 Female Adult    Positive    103
## 3 Female Child    Negative    112
## 4 Female Child    Positive     43
## 5 Female Senior   Negative     18
## 6 Female Senior   Positive      8
## 7 Male   Adult    Negative    171
## 8 Male   Adult    Positive     96
## 9 Male   Child    Negative     94
## 10 Male   Child    Positive     42
## 11 Male   Senior   Negative      9
## 12 Male   Senior   Positive     10
```

### Solution exercice pratique 8\_2

```
(cat_age_3, jours_alite)
```

```
## # A tibble: 28 × 3
##   cat_age_3 jours_alite     n
##   <chr>      <dbl> <int>
## 1 Adult          0    147
## 2 Adult          1     10
## 3 Adult          2     28
## 4 Adult          3     18
## 5 Adult          4      3
## 6 Adult          5      3
## 7 Adult          6      2
## 8 Adult          7      4
## 9 Adult          8      1
## 10 Adult         14      1
## # i 18 more rows
```

### Solution exercice pratique 9

```

e(quartier = as.factor(quartier),
  cat_age_3 = as.factor(cat_age_3),
  sexe = as.factor(sexe)) %>%
by(quartier, cat_age_3, sexe, .drop=FALSE) %>%
size(median_age = median(age, na.rm=TRUE))

```

```

## `summarise()` has grouped output by 'quartier', 'cat_age_3'.
## You can override using the `.groups` argument.

```

```

## # A tibble: 54 × 4
## # Groups:   quartier, cat_age_3 [27]
##   quartier    cat_age_3 sexe    median_age
##   <fct>      <fct>    <fct>      <dbl>
## 1 Briqueterie Adult    Female      31
## 2 Briqueterie Adult    Male      37.5
## 3 Briqueterie Child    Female     13
## 4 Briqueterie Child    Male       9
## 5 Briqueterie Senior   Female     69
## 6 Briqueterie Senior   Male     72
## 7 Carriere     Adult    Female     30
## 8 Carriere     Adult    Male     30
## 9 Carriere     Child    Female     10
## 10 Carriere    Child    Male     12
## # i 44 more rows

```