Notes de leçon | Modifier les colonnes

February 2024

Introduction

Vous savez désormais comment conserver ou supprimer des colonnes et des lignes de votre ensemble de données. Aujourd'hui, vous allez apprendre comment modifier des variables existantes ou en créer de nouvelles, en utilisant la fonction `mutate()` de {dplyr}. C'est une étape essentielle dans la plupart des projets d'analyse de données.

Allons-y!

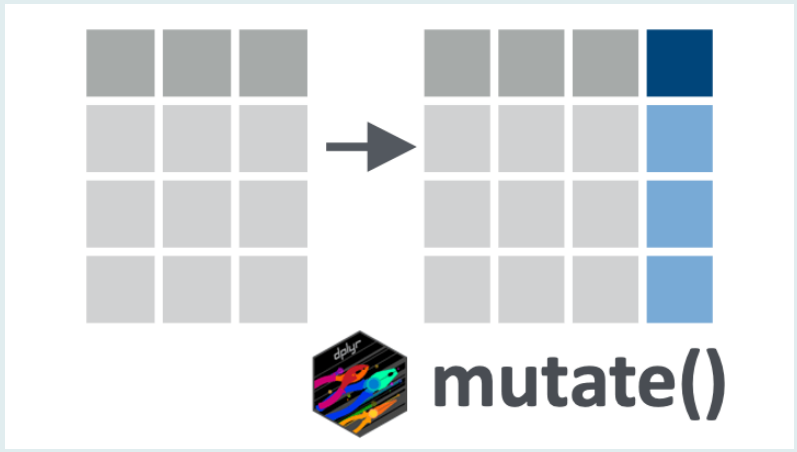

Fig: la fonction `mutate()`.

Objectifs d'apprentissage

1. Vous pouvez utiliser la fonction `mutate()` du package `{dplyr}` pour créer de nouvelles variables ou modifier des variables existantes.

2. Vous pouvez créer de nouvelles variables numériques, caractères, facteurs et booléennes.

Packages

Cette leçon nécessitera les "packages" chargés ci-dessous:

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

Jeux de données

Dans cette leçon, nous utiliserons à nouveau les données de l'enquête sérologique COVID-19 menée à Yaoundé, au Cameroun. Ci-dessous, nous importons l'ensemble de données `yaounde` et créons un sous-ensemble plus petit appelé `yao`. Notez que cet ensemble de données est légèrement différent de celui utilisé dans la leçon précédente.

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

ous-ensemble plus petit de variables
aounde %>% select(date_surveyed,
                  age,
                  weight_kg, height_cm,
                  symptoms, is_smoker)
```

date_sur…	age	weight_kg	height_cm	symptoms	is_smoker
2020-10-22	45	95	169	Muscle p…	Non-smo…
2020-10-24	55	96	185	No sympt…	Ex-smoker
2020-10-24	23	74	180	No sympt…	Smoker
2020-10-22	20	70	164	Rhinitis--…	Non-smo…
2020-10-22	55	67	147	No sympt…	Non-smo…
2020-10-25	17	65	162	Fever--C…	Non-smo…
2020-10-25	13	65	150	Sneezing	Non-smo…
2020-10-24	28	62	173	Headache	Non-smo…
2020-10-24	30	73	170	Fever--R…	Non-smo…
2020-10-24	13	56	153	No sympt…	Non-smo…

1–10 of 971 rows Previous **1** 2 3 4 5 ... 98 Next

Nous utiliserons également un jeu de données issu d'une étude transversale visant à déterminer la prévalence de la sarcopénie chez la population âgée (>60 ans) au Karnataka, en Inde. La sarcopénie est une condition courante chez les personnes âgées, caractérisée par une perte progressive et généralisée de la masse et de la force musculaires squelettiques. Les données ont été obtenues de Zenodo ici, et la publication source peut être trouvée ici.

Ci-dessous, nous importons et visualisons cet ensemble de données :

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

ıia
```

number	age	age_gr…	sex_ma…	marital_…	height_…	wei
7	60.8	Sixties	0	married	1.57	58
8	72.3	Seventies	1	married	1.65	72
9	62.6	Sixties	0	married	1.59	64
12	72	Seventies	0	widow	1.473	54.5
13	60.1	Sixties	0	married	1.55	47
19	60.6	Sixties	0	married	1.422	64
45	60.1	Sixties	1	widower	1.68	60
46	60.2	Sixties	0	married	1.8	64.6
51	63	Sixties	0	married	1.6	57.8
56	60.4	Sixties	0	married	1.6	71.8

1–10 of 239 rows Previous **1** 2 3 4 5 ... 24 Next

Notez que le jeu de donnée COVID-19 Yaoundé et sarcopénie sont en anglais !

Pour cette leçon, nous utiliserons ces versions en anglais. Mais dans d'autres leçons, nous utiliserons des versions partialement en français.

Introduction à `mutate()`

La fonction `mutate()`. (Dessin adapté d'Allison Horst)

Nous utilisons `dplyr::mutate()` pour créer de nouvelles variables ou modifier des variables existantes. La syntaxe est assez intuitive et ressemble généralement à `df %>% mutate(nouveau_nom_de_colonne = ce_qu'il_contient)`.

Voyons un exemple rapide.

L'ensemble de données `yaounde` contient actuellement une colonne appelée `height_cm`, qui montre la taille, en centimètres, des répondants à l'enquête. Créons un dataframe, `yao_height`, avec juste cette colonne, pour une illustration facile :

```
ght <- yaounde %>% select(height_cm)
ght
```

```
## # A tibble: 5 × 1
##   height_cm
##       <dbl>
## 1       169
## 2       185
## 3       180
## 4       164
## 5       147
```

Que faire si vous vouliez **créer une nouvelle variable**, appelée `height_meters`, où les hauteurs sont converties en mètres? Vous pouvez utiliser `mutate()` pour cela, avec l'argument `height_meters = height_cm/100` :

```
ght %>%
e(height_meters = height_cm/100)
```

```
## # A tibble: 5 × 2
##   height_cm height_meters
##       <dbl>         <dbl>
## 1       169          1.69
## 2       185          1.85
## 3       180          1.8
## 4       164          1.64
## 5       147          1.47
```

Super. La syntaxe est magnifiquement simple, n'est-ce pas?

SIDE NOTE

Il est parfois utile de penser aux fonctions de manipulation de données dans le contexte d'un logiciel de tableur familier. Voici à quoi correspondrait la commande R `mutate(height_m = height_cm/100)` dans Google Sheets :

	A	B
1	**height_cm**	
2	169	
3	185	
4	180	
5	164	
6	147	
7	162	
8	150	

Maintenant, imaginez qu'il y avait une petite erreur dans l'équipement utilisé pour mesurer les hauteurs des répondants, et que toutes les hauteurs sont inférieures de 5 cm. Vous aimeriez donc ajouter 5 cm à toutes les hauteurs de l'ensemble de données. Pour ce faire, plutôt que de créer une nouvelle variable comme vous l'avez fait auparavant, vous pouvez **modifier la variable existante** avec mutate :

```
ght %>%
e(height_cm = height_cm + 5)
```

```
## # A tibble: 5 × 1
##   height_cm
```

```
##       <dbl>
## 1       174
## 2       190
## 3       185
## 4       169
## 5       152
```

Encore une fois, très facile à faire!

La dataframe `sarcopenia` a une variable `weight_kg`, qui contient le poids des répondants en kilogrammes. Créez une nouvelle colonne, appelée `weight_grams`, avec le poids des répondants en grammes. Stockez votre réponse dans l'objet `Q_weight_to_g`. (1 kg équivant à 1000 grammes.)

```
étez le code avec votre réponse:
t_to_g <-
penia %>%
_____________
```

J'espère que vous voyez maintenant que la fonction `mutate` est très conviviale. En théorie, nous pourrions terminer la leçon ici, car vous savez maintenant comment utiliser `mutate()` 😀. Mais bien sûr, le diable est dans les détails - ce qui est intéressant, ce n'est pas `mutate()` lui-même mais ce qui se trouve *à l'intérieur* de l'appel `mutate()`.

Le reste de la leçon traitera de quelques cas d'utilisation du verbe `mutate()`. Dans ce processus, nous aborderons plusieurs nouvelles fonctions que vous n'avez pas encore rencontrées.

Création d'une variable à partir de zéro (indice de ligne)

Souvent, vous allez créer des variables qui **référencent des variables existantes** (par exemple, la variable `height_meters` créée ci-dessus fait référence à la variable `height_cm`).

Parfois, vous créez des variables **"à partir de zéro"** sans faire référence à d'autres.

Regardons a un exemple. Nous allons créer un indice de ligne avec `mutate()` et `1:n()` pour générer une séquence d'indices de lignes.

Le `n()` dans dplyr retourne le nombre de lignes du dataframe.

```
ght %>%
e(row_index = 1:n())
```

```
## # A tibble: 5 × 2
##   height_cm row_index
##       <dbl>     <int>
## 1       169         1
## 2       185         2
## 3       180         3
## 4       164         4
## 5       147         5
```

Ajoutez une variable `respondent_id` à `sarcopenia` pour contenir le numéro de ligne.

```
étez le code :
penia_resp_id <-
penia %>%
e(_____________________________)
```

Note : La base de données `sarcopenia` comporte 10 colonnes, qui ne tiendront probablement pas toutes dans votre console lorsque vous imprimerez cette base de données. Votre nouvelle colonne, `respondent_id` sera probablement cachée.

```
 5    13  60.6                0 above poverty l… married
 6    19  60.5                0 above poverty l… married
 7    45  60.4                1 above poverty l… widower
 8    46  60.4                0 above poverty l… married
 9    51  63                  0 above poverty l… married
10    56  60.3                0 above poverty l… married
# … with 229 more rows, and 6 more variables:
#   height_meters <dbl>, weight_kg <dbl>,
#   gait_speed_meters_per_second <chr>, gripstrength_kg <dbl>,
#   skeletal_muscle_index <dbl>, respondent_id <int>
> Column hidden
```

Ainsi, pour voir réellement la nouvelle colonne, vous pouvez utiliser `View()` pour visualiser l'ensemble du cadre de données ou `select()` pour sélectionner les colonnes concernées.

Création d'une variable booléenne

Vous pouvez utiliser `mutate()` pour créer une variable booléenne pour catégoriser une partie de votre population.

Ci-dessous, nous créons une variable booléenne, `is_child` qui est soit `TRUE` si le sujet est un enfant ou `FALSE` si le sujet est un adulte (d'abord, nous sélectionnons uniquement la variable `age` pour qu'il soit facile de voir ce qui est fait ; il est probable que vous n'ayez pas besoin de cette présélection pour vos propres analyses).

```
t(age) %>%
e(is_child = age <= 18)
```

```
## # A tibble: 5 × 2
##     age is_child
##   <dbl> <lgl>
## 1    45 FALSE
## 2    55 FALSE
## 3    23 FALSE
## 4    20 FALSE
## 5    55 FALSE
```

Le code `age <= 18` évalue si chaque âge est inférieur ou égal à 18. Les âges qui correspondent à cette condition (18 ans et moins) sont `TRUE` et ceux qui ne répondent pas à la condition sont `FALSE`.

Une telle variable est utile pour, par exemple, compter le nombre d'enfants dans le jeu de données. Le code ci-dessous le fait avec la fonction `janitor::tabyl()` :

```
e(is_child = age <= 18) %>%
(is_child)
```

```
##  is_child   n   percent
##     FALSE 662 0.6817714
##      TRUE 309 0.3182286
```

Vous pouvez observer que 31,8% (0,318…) des répondants de l'ensemble de données sont des enfants.

Voyons un autre exemple, car le concept de variables booléennes peut être un peu déroutant. La variable `symptoms` rapporte tous les symptômes respiratoires ressentis par le patient :

```
t(symptoms)
```

```
## # A tibble: 5 × 1
##   symptoms
##   <chr>
## 1 Muscle pain
## 2 No symptoms
```

```
## 3 No symptoms
## 4 Rhinitis--Sneezing--Anosmia or ageusia
## 5 No symptoms
```

Vous pourriez créer une variable booléenne, appelée `has_no_symptoms`, qui est définie sur `TRUE` si le répondant n'a signalé aucun symptôme :

```
(symptoms) %>%
(has_no_symptoms = symptoms == "No symptoms")
```

```
## # A tibble: 5 × 2
##   symptoms
##   <chr>
## 1 Muscle pain
## 2 No symptoms
## 3 No symptoms
## 4 Rhinitis--Sneezing--Anosmia or ageusia
## 5 No symptoms
##   has_no_symptoms
##   <lgl>
## 1 FALSE
## 2 TRUE
## 3 TRUE
## 4 FALSE
## 5 TRUE
```

De même, vous pourriez créer une variable booléenne appelée `has_any_symptoms` qui est définie sur `TRUE` si le répondant a signalé des symptômes. Pour cela, vous échangerez simplement le code `symptoms == "No symptoms"` pour `symptoms != "No symptoms"` :

```
(symptoms) %>%
(has_any_symptoms = symptoms != "No symptoms")
```

```
## # A tibble: 5 × 2
##   symptoms
##   <chr>
## 1 Muscle pain
## 2 No symptoms
## 3 No symptoms
## 4 Rhinitis--Sneezing--Anosmia or ageusia
## 5 No symptoms
##   has_any_symptoms
##   <lgl>
## 1 TRUE
## 2 FALSE
## 3 FALSE
```

```
## 4 TRUE
## 5 FALSE
```

Toujours confus par les exemples booléens? C'est normal. Prenez une pause et jouez un peu avec le code ci-dessus. Ensuite, essayez la question de pratique ci-dessous.

Les femmes ayant une force de préhension inférieure à 20 kg sont considérées comme ayant une faible force de préhension. Avec un sous-ensemble féminin du dataframe `sarcopenia`, ajoutez une variable appelée `low_grip_strength` qui est `TRUE` pour les femmes avec une force de préhension < 20 kg et FALSE pour les autres femmes.

```
étez le code avec votre réponse :
_low_grip_strength <-
penia %>%
(sex_male_1_female_0 == 0) %>%  # d'abord, nous filtrons
 l'ensemble de données pour seulement les femmes
(_____________________) # code mutate ici
```

Quel pourcentage de femmes interrogées ont une faible force de préhension selon la définition ci-dessus? Entrez votre réponse sous forme de nombre sans guillemets (par exemple, 43.3 ou 12.2), à une décimale près.

```
omen_low_grip_strength <- "VOTRE RÉPONSE"
```

Création d'une variable numérique basée sur une formule

Maintenant, regardons un exemple de création d'une variable numérique, l'indice de masse corporelle (IMC), qui est un indicateur de santé couramment utilisé. La formule de l'indice de masse corporelle peut être écrite comme suit :

$$IMC = \frac{poids(kilogrammes)}{taille(mètres)^2}$$

Vous pouvez utiliser `mutate()` pour calculer l'IMC dans l'ensemble de données `yao` comme suit :

```
t(weight_kg,  height_cm) %>%

ord obtenir la taille en mètres
e(height_meters = height_cm/100) %>%

s utiliser la formule de l'IMC
e(bmi = weight_kg / (height_meters)^2)
```

```
## # A tibble: 5 × 4
##   weight_kg height_cm height_meters   bmi
##       <dbl>     <dbl>         <dbl> <dbl>
## 1        95       169          1.69  33.3
## 2        96       185          1.85  28.0
## 3        74       180          1.8   22.8
## 4        70       164          1.64  26.0
## 5        67       147          1.47  31.0
```

Sauvegardons le dataframe avec les IMC pour plus tard. Nous l'utiliserons dans la section suivante.

```
<-
>%
t(weight_kg,  height_cm) %>%
ord obtenir la taille en mètres
e(height_meters = height_cm/100) %>%
s utiliser la formule de l'IMC
e(bmi = weight_kg / (height_meters)^2)
```

La masse musculaire appendiculaire (MMA), un indicateur de santé utile, est la somme de la masse musculaire dans les 4 membres. Elle peut être prédite avec la formule suivante, appelée équation de Lee :

$$MMA(kg) = (0.244 \times poids(kg)) + (7.8 \times taille(m)) + (6.6 \times sexe) - ($$

La variable `sex` dans la formule suppose que les hommes sont codés comme 1 et les femmes comme 0 (ce qui est déjà le cas pour notre ensemble de données `sarcopenia`). Le `- 4.5` à la fin est une constante utilisée pour les Asiatiques.

Calculez la valeur MMA pour tous les individus dans l'ensemble de données `sarcopenia`. Cette valeur devrait être dans une nouvelle colonne appelée `asm`


```
étez le code avec votre réponse:
alculation <-
penia %>%
e(asm = _____)
```

Changer le type d'une variable

Dans votre flux de travail d'analyse de données, vous avez souvent besoin de redéfinir les *types* de variables. Vous pouvez le faire avec des fonctions telles que `as.integer()`, `as.factor()`, `as.character()` et `as.Date()` dans votre appel `mutate()`. Voyons un exemple de cela.

Entier : `as.integer`

`as.integer()` convertit toutes les valeurs numériques en entiers :

```
%>%
e(bmi_integer = as.integer(bmi))
```

```
## # A tibble: 5 × 5
##   weight_kg height_cm height_meters   bmi
##       <dbl>     <dbl>          <dbl> <dbl>
## 1        95       169           1.69  33.3
## 2        96       185           1.85  28.0
## 3        74       180           1.8   22.8
## 4        70       164           1.64  26.0
## 5        67       147           1.47  31.0
##   bmi_integer
##         <int>
## 1          33
## 2          28
## 3          22
## 4          26
## 5          31
```

Notez que cela *tronque* les entiers plutôt que de les arrondir vers le haut ou le bas, comme vous pourriez vous y attendre. Par exemple, le BMI 22.8 à la troisième ligne est tronqué à 22. Si vous voulez des nombres arrondis, vous pouvez utiliser la fonction `round` de base R.

Utiliser `as.integer()` sur une variable factorielle est un moyen rapide de codifier des chaînes en nombres. Cela peut être essentiel pour certains traitements de données d'apprentissage automatique.

```
%>%
(bmi_integer = as.integer(bmi),
 bmi_rounded = round(bmi))
```

```
## # A tibble: 5 × 6
##   weight_kg height_cm height_meters   bmi
##       <dbl>     <dbl>         <dbl> <dbl>
## 1        95       169          1.69  33.3
## 2        96       185          1.85  28.0
## 3        74       180          1.8   22.8
## 4        70       164          1.64  26.0
## 5        67       147          1.47  31.0
##   bmi_integer bmi_rounded
##         <int>       <dbl>
## 1          33          33
## 2          28          28
## 3          22          23
## 4          26          26
## 5          31          31
```

La fonction `round()` de base R arrondit "à la moitié vers le bas". C'est-à-dire que le nombre 3.5, par exemple, est arrondi à 3 par `round()`. C'est étrange. La plupart des gens s'attendent à ce que 3.5 soit arrondi *vers le haut* à 4, et non pas vers le bas à 3. La plupart du temps, vous voudrez donc utiliser la fonction `round_half_up()` de janitor.

Dans les leçons futures, vous découvrirez comment manipuler les dates et comment convertir en un type de date en utilisant `as.Date()`.

Utilisez `as_integer()` pour convertir les âges des répondants dans l'ensemble de données `sarcopenia` en entiers (en les tronquant au

passage). Cela devrait aller dans une nouvelle colonne appelée `age_integer`.

PRACTICE (in RMD)

```
étez le code avec votre réponse :
ıteger <-
)enia %>%
e(age_integer =  _____________________________)
```

En Résumé !

Comme vous pouvez l'imaginer, transformer des données est une étape essentielle dans tout flux de travail d'analyse de données. Il est souvent nécessaire de nettoyer les données et de les préparer pour d'autres analyses statistiques ou pour créer des graphiques. Et comme vous l'avez vu, il est assez simple de transformer des données avec la fonction `mutate()` de dplyr, bien que certaines transformations soient plus délicates à réaliser que d'autres.

Félicitations d'être arrivé jusqu'ici.

Mais votre périple de manipulation de données n'est pas encore terminé ! Dans nos prochaines leçons, nous apprendrons à créer des résumés de données complexes et comment créer et travailler avec des groupes de data frame. Intrigué ? À bientôt dans la prochaine leçon.

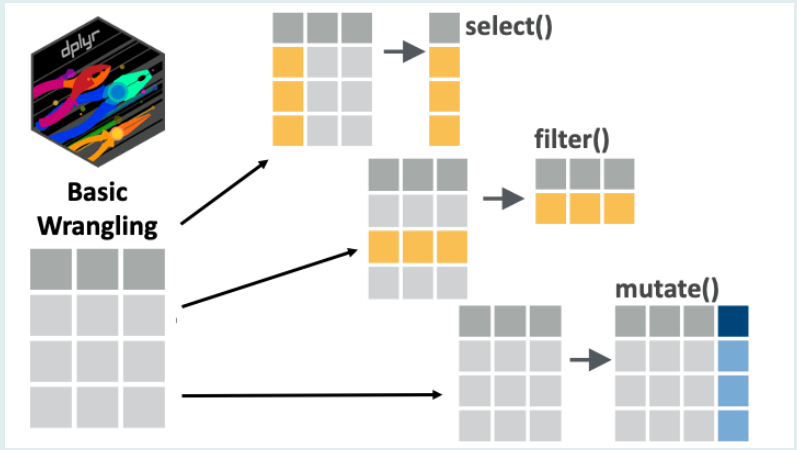

Fig : Manipulation basique des données avec `select()`, `filter()`, et `mutate()`.

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

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

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)
- *Créer, modifier, et supprimer des colonnes — Mutate*. (n.d.). Consulté le 21 février 2022, à partir de https://dplyr.tidyverse.org/reference/mutate.html
- *Appliquer une fonction (ou des fonctions) sur plusieurs colonnes — Across*. (n.d.). Consulté le 21 février 2022, à partir de https://dplyr.tidyverse.org/reference/across.html

Les illustrations ont été adaptées de :

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

Autres références :

- Lee, Robert C, ZiMian Wang, Moonseong Heo, Robert Ross, Ian Janssen, et Steven B Heymsfield. “Total-Body Skeletal Muscle Mass: Development and Cross-Validation of Anthropometric Prediction Models.” *The American Journal*

of Clinical Nutrition 72, no. 3 (2000): 796–803. https://doi.org/10.1093/ajcn/72.3.796.