Chapitre 5 : Mise en forme des données

*Traduit par Zoé Lackner*

# 5.1 Objectifs d’apprentissage

## 5.1.1 Niveau débutant

1. Être capable d’utiliser les 6 principaux one-table verbes de dplyr :
   * [select()](https://psyteachr.github.io/msc-data-skills/dplyr.html#select)
   * [filter()](https://psyteachr.github.io/msc-data-skills/dplyr.html#filter)
   * [arrange()](https://psyteachr.github.io/msc-data-skills/dplyr.html#arrange)
   * [mutate()](https://psyteachr.github.io/msc-data-skills/dplyr.html#mutate)
   * [summarise()](https://psyteachr.github.io/msc-data-skills/dplyr.html#summarise)
   * [group\_by()](https://psyteachr.github.io/msc-data-skills/dplyr.html#group_by)

## 5.1.2 Niveau intermédiaire

1. Aussi apprendre à utiliser ces one-table verbs additionnels :
   * [rename()](https://psyteachr.github.io/msc-data-skills/dplyr.html#rename)
   * [distinct()](https://psyteachr.github.io/msc-data-skills/dplyr.html#distinct)
   * [count()](https://psyteachr.github.io/msc-data-skills/dplyr.html#count)
   * [slice()](https://psyteachr.github.io/msc-data-skills/dplyr.html#slice)
   * [pull()](https://psyteachr.github.io/msc-data-skills/dplyr.html#pull)

## 5.1.3 Niveau avancé

1. Avoir une bonne maîtrise des opérations [select()](https://psyteachr.github.io/msc-data-skills/dplyr.html#select_helpers)
2. Utiliser les [fonctions windows](https://psyteachr.github.io/msc-data-skills/dplyr.html#window)

# 5.2 Ressources

*N.B. Les ressources marquées (EN) sont en anglais.*

* [Chapter 5: Data Transformation](http://r4ds.had.co.nz/transform.html) in *R for Data Science (EN)*
* [Data transformation cheat sheet](https://github.com/rstudio/cheatsheets/raw/master/source/pdfs/data-transformation-cheatsheet.pdf) *(EN)*
* [Lecture slides on dplyr one-table verbs](https://psyteachr.github.io/msc-data-skills/slides/04_dplyr_slides.pdf) *(EN)*
* [Chapter 16: Date and times](http://r4ds.had.co.nz/dates-and-times.html) in *R for Data Science (EN)*

# 5.3 Configuration

Vous aurez besoin des packages suivants.

|  |
| --- |
| *# libraries needed for these examples*  **library**(tidyverse)  **library**(lubridate)  **set.seed**(8675309) *# makes sure random numbers are reproducible* |

## 5.4 Le jeu de données disgust

Ces exemples vous aideront à utiliser les données issues de [disgust.csv](https://psyteachr.github.io/data/disgust.csv) *(non accessible)*, qui contient des données issues de [Three Domain Disgust Scale](http://digitalrepository.unm.edu/cgi/viewcontent.cgi?article=1139&context=psy_etds) *(EN)* (échelle mesurant les différences inter-individuelles en terme de sensibilité au dégoût). Chaque participant est identifié par un unique user\_id et chaque questionnaire rempli est identifié par son unique id.

|  |
| --- |
| disgust <- **read\_csv**("https://psyteachr.github.io/msc-data-skills/data/disgust.csv") |

*Instructions du Questionnaire* : Les items suivants décrivent une variété de concepts. Veuillez évaluer à quel point vous trouvez dégoûtant les concepts décrits à chaque item, où 0 signifie que vous ne trouvez pas du tout le concept dégoûtant, et 6 signifie que vous trouvez le concept extrêmement dégoûtant.

|  |  |
| --- | --- |
| **nom de col** | **question** |
| moral1 | Voler une friandise dans une supérette. |
| moral2 | Voler à un voisin. |
| moral3 | Un étudiant qui triche pour obtenir de bons résultats. |
| moral4 | Abuser d’un ami. |
| moral5 | Imiter la signature de quelqu’un sur un document légal. |
| moral6 | Doubler la queue pour acheter un des derniers tickets pour un spectacle. |
| moral7 | Mentir intentionnellement pendant une transaction financière. |
| sexual1 | Ecouter deux inconnus avoir des rapports sexuels. |
| sexual2 | Pratiquer le sexe oral. |
| sexual3 | Regarder une vidéo pornographique. |
| sexual4 | Découvrir que quelqu’un que vous n’appréciez pas éprouve des fantasmes sexuels vous concernant. |
| sexual5 | Ramener dans votre chambre quelqu’un que vous venez de rencontrer pour avoir un rapport sexuel. |
| sexual6 | Une personne inconnue, du sexe opposé, qui caresse volontairement votre cuisse dans un ascenseur. |
| sexual7 | Avoir un rapport annal avec une personne de sexe opposé. |
| pathogen1 | Marcher dans un caca de chien. |
| pathogen2 | S’asseoir à côté de quelqu’un qui a des plaies infectées sur les bras. |
| pathogen3 | Serrer la main d’un inconnu qui a les mains moites. |
| pathogen4 | Apercevoir de la moisissure dans de vieux restes du réfrigérateur. |
| pathogen5 | Être à côté de quelqu’un qui a une forte odeur corporelle. |
| pathogen6 | Apercevoir un cafard courir sur le sol. |
| pathogen7 | Toucher accidentellement la coupure ensanglantée de quelqu’un. |

# 

# 5.5 Les six principaux verbes dplyr

La plupart des procédures de mise en forme des données que vous souhaiterez faire avec des données en psychologie impliquent les verbes tidyr que vous avez appris au [Chaptitre 3](https://psyteachr.github.io/msc-data-skills/tidyr.html#tidyr) et les six principaux verbes dplyr : select, filter, arrange, mutate, summarise, et group\_by.

## 5.5.1 select()

Permet de sélectionner les colonnes par leur nom ou numéro.

Vous pouvez sélectionner chaque colonne individuellement, séparées alors par des virgules (e.g., col1, col2). Mais vous pouvez aussi sélectionner toutes les colonnes comprises entre deux colonnes, en les séparant par deux points (e.g., start\_col:end\_col).

|  |
| --- |
| moral <- disgust %>% **select**(user\_id, moral1:moral7)  **names**(moral) |

|  |
| --- |
| ## [1] "user\_id" "moral1" "moral2" "moral3" "moral4" "moral5"  ## [7] "moral6" "moral7" |

Vous pouvez aussi sélectionner des colonnes par leur numéro, ce qui est utile quand le nom des colonnes est long ou compliqué.

|  |
| --- |
| sexual <- disgust %>% **select**(2, 11:17)  **names**(sexual) |

|  |
| --- |
| ## [1] "user\_id" "sexual1" "sexual2" "sexual3" "sexual4" "sexual5"  ## [7] "sexual6" "sexual7" |

De plus, vous pouvez utiliser un symbole moins pour dé-sélectionner des colonnes, conservant alors toutes les autres colonnes du tableau. Si vous souhaitez exclure un intervalle de colonnes, il faut tout d’abord placer des parenthèses autour de l’intervalle (e.g., -(moral1:moral7), pas-moral1:moral7).

|  |
| --- |
| pathogen <- disgust %>% **select**(-id, -date, -(moral1:sexual7))  **names**(pathogen) |

|  |
| --- |
| ## [1] "user\_id" "pathogen1" "pathogen2" "pathogen3" "pathogen4"  ## [6] "pathogen5" "pathogen6" "pathogen7" |

Vous pouvez aussi sélectionner des colonnes en fonction de critères qui sont compris dans le nom de la colonne.{#select\_helpers}

### 5.5.1.1 starts\_with()

Sélectionne les colonnes qui commencent par une chaîne de caractères.

|  |
| --- |
| u <- disgust %>% **select**(**starts\_with**("u"))  **names**(u) |

|  |
| --- |
| ## [1] "user\_id" |

### 5.5.1.2 ends\_with()

Sélectionne les colonnes qui terminent par une chaîne de caractères.

|  |
| --- |
| firstq <- disgust %>% **select**(**ends\_with**("1"))  **names**(firstq) |

|  |
| --- |
| ## [1] "moral1" "sexual1" "pathogen1" |

### 5.5.1.3 contains()

Sélectionne les colonnes qui contiennent une chaîne de caractères.

|  |
| --- |
| pathogen <- disgust %>% **select**(**contains**("pathogen"))  **names**(pathogen) |

|  |
| --- |
| ## [1] "pathogen1" "pathogen2" "pathogen3" "pathogen4" "pathogen5"  ## [6] "pathogen6" "pathogen7" |

### 5.5.1.4 num\_range()

Sélectionne les colonnes avec un nom qui correspond à la structure prefix.

|  |
| --- |
| moral <- disgust %>% **select**(**num\_range** ("moral", 2:4))  **names**(moral\_4) |

|  |
| --- |
| ## [1] "moral2" "moral3" "moral4" |

|  |  |
| --- | --- |
|  | Vous pouvez utiliser width pour paramétrer le nombre de chiffres avant le zéro. Par exemple, num\_range(‘var\_’, 8:10, width=2) permet de sélectionner les colonnes var\_08, var\_09, et var\_10. |

## 5.5.2 filter()

Permet de sélectionner des lignes en fonction de critères de la colonne.

Ce code permet de sélectionner toutes les lignes dont le user\_id est de 1.

|  |
| --- |
| disgust %>% **filter**(user\_id == 1) |

|  |
| --- |
| ## # A tibble: 1 x 24  ## id user\_id date moral1 moral2 moral3 moral4 moral5  ## <dbl> <dbl> <date> <dbl> <dbl> <dbl> <dbl> <dbl>  ## 1 1 1 2008-07-10 2 2 1 2 1  ## # … with 16 more variables: moral6 <dbl>, moral7 <dbl>,  ## # sexual1 <dbl>, sexual2 <dbl>, sexual3 <dbl>, sexual4 <dbl>, ## # sexual5 <dbl>, sexual6 <dbl>, sexual7 <dbl>,  ## # pathogen1 <dbl>, pathogen2 <dbl>, pathogen3 <dbl>,  ## # pathogen4 <dbl>,pathogen5 <dbl>, pathogen6 <dbl>,  ## # pathogen7 <dbl> |

|  |  |
| --- | --- |
|  | N’oubliez pas d’utiliser == et non = pour vérifier si deux choses sont équivalentes. Un seul = permet d’assigner la valeur de droite à la variable de gauche et permet (en général) de l’évaluer comme TRUE. |

Vous pouvez sélectionner plusieurs critères en les séparant par des virgules.

|  |
| --- |
| amoral <- disgust %>% **filter**(  moral1 == 0,  moral2 == 0,  moral3 == 0,  moral4 == 0,  moral5 == 0,  moral6 == 0,  moral7 == 0  ) |

Vous pouvez utiliser des symboles &, |, et ! pour vouloir dire “et”, “ou”, et “pas”. Vous pouvez utiliser des opérateurs pour faire des équations.

|  |
| --- |
| *#* *everyone who chose either 0 or 7 for question moral1*  moral\_extremes <- disgust %>%  **filter**(moral1 == 0 | moral1 == 7)  *# everyone who chose the same answer for all moral questions*  moral\_consistent <- disgust %>%  **filter**(  moral2 == moral1 &  moral3 == moral1 &  moral4 == moral1 &  moral5 == moral1 &  moral6 == moral1 &  moral7 == moral1  )  *# everyone who did not answer 7 for all 7 moral questions*  moral\_no\_ceiling <- disgust %>%  **filter**(moral1+moral2+moral3+moral4+moral5+moral6+moral7 != 7\*7) |

Parfois, vous devez exclure des participants IDs pour des raisons que nous ne formulerons pas ici. L’opérateur %in% est utile pour tester si la valeur d’une colonne est dans une liste. Il faut entourer l’équation de parenthèses et ajouter un ! devant pour s’assurer qu’une ou plusieurs valeurs soient retirées de la liste.

|  |
| --- |
| no\_researchers <- disgust %>%  **filter**(!(user\_id %in% **c**(1,2))) |

### 5.5.2.1 Dates

Vous pouvez utiliser le package lubridate pour travailler avec les dates. Par exemple, vous pouvez utiliser la fonction year() pour retourner l’année de la colonne date et ensuite sélectionner les données uniquement collectées en 2010.

|  |
| --- |
| disgust2010 <- disgust %>%  **filter**(**year**(date) == 2010) |

Vous pouvez aussi sélectionner les données d’il y a 5 ans. Vous pouvez alors utiliser la fonction range pour vérifier les dates minimales et maximales du jeux de données résultant.

|  |
| --- |
| disgust\_5ago <- disgust %>%  **filter**(date < **today**() - **dyears**(5))  **range** (disgust\_5ago$date) |

|  |
| --- |
| ## [1] "2008-07-10" "2014-08-04" |

## 5.5.3 arrange()

Permet d’ordonner ses données.

|  |
| --- |
| disgust\_order <- disgust %>%  **arrange**(id)  **head**(disgust\_order) |

|  |
| --- |
| ## # A tibble: 6 x 24  ## id user\_id date moral1 moral2 moral3 moral4 moral5  ## <dbl> <dbl> <date> <dbl> <dbl> <dbl> <dbl> <dbl>  ## 1 1 1 2008-07-10 2 2 1 2 1  ## 2 3 155324 2008-07-11 2 4 3 5 2  ## 3 4 155366 2008-07-12 6 6 6 3 6  ## 4 5 155370 2008-07-12 6 6 4 6 6  ## 5 6 155386 2008-07-12 2 4 0 4 0  ## 6 7 155409 2008-07-12 4 5 5 4 5  ## # … with 16 more variables: moral6 <dbl>, moral7 <dbl>,  ## # sexual1 <dbl>, sexual2 <dbl>, sexual3 <dbl>, sexual4 <dbl>,  ## # sexual5 <dbl>, sexual6 <dbl>, sexual7 <dbl>,  ## # pathogen1 <dbl>, pathogen2 <dbl>, pathogen3 <dbl>,  ## # pathogen4 <dbl>, pathogen5 <dbl>, pathogen6 <dbl>,  ## # pathogen7 <dbl> |

En utilisant la fonction desc() vous pouvez inverser le sens.

|  |
| --- |
| disgust\_order <- disgust %>%  **arrange**(**desc**(id))  **head**(disgust\_order) |

|  |
| --- |
| ## # A tibble: 6 x 24  ## id user\_id date moral1 moral2 moral3 moral4 moral5  ## <dbl> <dbl> <date> <dbl> <dbl> <dbl> <dbl> <dbl>  ## 1 39456 356866 2017-08-21 1 1 1 1 1  ## 2 39447 128727 2017-08-13 2 4 1 2 2  ## 3 39371 152955 2017-06-13 6 6 3 6 6  ## 4 39342 48303 2017-05-22 4 5 4 4 6  ## 5 39159 151633 2017-04-04 4 5 6 5 3  ## 6 38942 370464 2017-02-01 1 5 0 6 5  ## # … with 16 more variables: moral6 <dbl>,moral7 <dbl>,  ## # sexual1 <dbl>, sexual2 <dbl>, sexual3 <dbl>, sexual4 <dbl>,  ## # sexual5 <dbl>, sexual6 <dbl>, sexual7 <dbl>,  ## # pathogen1 <dbl>, pathogen2 <dbl>, pathogen3 <dbl>,  ## # pathogen4 <dbl>,pathogen5 <dbl>, pathogen6 <dbl>,  ## # pathogen7 <dbl> |

### 

## 5.5.4 mutate()

Permet de créer de nouvelles colonnes. C’est une des fonctions les plus utiles du package tidyverse.

Pour partir de variables déjà présentes dans le jeux de données, il faut se référer à leur nom (sans les guillemets). Vous pouvez créer plus d’une colonne, en séparant les colonnes créée par une virgule. Une fois que vous avez créé une nouvelle colonne, vous pouvez immédiatement vous en servir pour créer une nouvelle variable (e.g., total ci-dessous).

|  |
| --- |
| disgust\_total <- disgust %>%  **mutate**(  pathogen = pathogen1 + pathogen2 + pathogen3 + pathogen4 + pathogen5 + pathogen6 + pathogen7,  moral = moral1 + moral2 + moral3 + moral4 + moral5 + moral6 + + moral7,  sexual = sexual1 + sexual2 + sexual3 + sexual4 + sexual5 + sexual6 + sexual7,  total = pathogen + moral + sexual,  user\_id = **paste0**("U", user\_id)  )  **head**(disgust\_total) |

|  |  |
| --- | --- |
|  | Vous pouvez écraser une colonne en donnant à une nouvelle colonne le même nom que celle-ci. Attention, vous ne pourrez alors plus accéder à l’ancienne colonne. |

## 

## 5.5.5 summarise()

Permet de créer un résumé des statistiques pour le jeux de données. Vérifier le [Formulaire de mise en forme des données](https://www.rstudio.org/links/data_wrangling_cheat_sheet) *(EN)* et le [Formulaire de transformation des données](https://github.com/rstudio/cheatsheets/raw/master/source/pdfs/data-transformation-cheatsheet.pdf) *(EN)* pour accéder à plusieurs fonctions de résumé. Quelques fonctions communes sont : mean(), sd(), n(), sum(), et quantile().

|  |
| --- |
| disgust\_total %>%  **summarise**(  n = **n**(),  q25 = **quantile**(total, .25, na.rm = TRUE),  q50 = **quantile**(total, .50, na.rm = TRUE),  q75 = **quantile**(total, .75, na.rm = TRUE),  avg\_total = **mean**(total, na.rm = TRUE),  sd\_total = **sd**(total, na.rm = TRUE),  min\_total = **min**(total, na.rm = TRUE),  max\_total = **max**(total, na.rm = TRUE)  ) |

|  |
| --- |
| ## # A tibble: 1 x 8  ## n q25 q50 q75 avg\_total sd\_total min\_total  ## <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  ## 1 20000 59 71 83 70.7 18.2 0  ## # … with 1 more variable: max\_total <dbl> |

### 

## 5.5.6 group\_by()

Permet de créer un sous-ensemble de données. Vous pouvez notamment l’utiliser pour créer des résumés, pour apprécier la valeur moyenne de chaque groupe expérimental par exemple.

Ici, nous allons utiliser mutate pour créer de nouvelles colonnes s’appelant year, grouper les données en fonction de year, et calculer les scores moyens.

|  |
| --- |
| disgust\_total %>%  **mutate**(year = **year**(date)) %>%  **group\_by**(year) %>%  **summarise**(  n = **n**(),  avg\_total = **mean**(total, na.rm = TRUE),  sd\_total = **sd**(total, na.rm = TRUE),  min\_total = **min**(total, na.rm = TRUE),  max\_total = **max**(total, na.rm = TRUE)  ) |

|  |
| --- |
| ## # A tibble: 10 x 6  ## year n avg\_total sd\_total min\_total max\_total  ## <dbl> <int> <dbl> <dbl> <dbl> <dbl>  ## 1 2008 2578 70.3 18.5 0 126  ## 2 2009 2580 69.7 18.6 3 126  ## 3 2010 1514 70.6 18.9 6 126  ## 4 2011 6046 71.3 17.8 0 126  ## 5 2012 5938 70.4 18.4 0 126  ## 6 2013 1251 71.6 17.6 0 126  ## 7 2014 58 70.5 17.2 19 113  ## 8 2015 21 74.3 16.9 43 107  ## 9 2016 8 67.9 32.6 0 110  ## 10 2017 6 57.2 27.9 21 90 |

Vous pouvez aussi combiner les fonctions filter et group\_by. L’exemple suivant retourne le plus petit score total pour chaque année.

|  |
| --- |
| disgust\_total %>%  **mutate**(year = **year**(date)) %>%  **select**(user\_id, year, total) %>%  **group\_by**(year) %>%  **filter**(**rank**(total) == 1) %>%  **arrange**(year) |

|  |
| --- |
| ## # A tibble: 7 x 3  ## # Groups: year [7]  ## user\_id year total  ## <chr> <dbl> <dbl>  ## 1 U236585 2009 3  ## 2 U292359 2010 6  ## 3 U245384 2013 0  ## 4 U206293 2014 19  ## 5 U407089 2015 43  ## 6 U453237 2016 0  ## 7 U356866 2017 21 |

Une autre possibilité est d’utiliser mutate après group\_by. L’exemple qui suit calcule un score centré sur la moyenne pour chaque sujet en groupant les scores en fonction de user\_id, puis de soustraire la moyenne spécifique à chaque groupe par chaque score. Notez qu’il faut d’abord utiliser gather pour organiser les données en un long format.

|  |
| --- |
| disgust\_smc <- disgust %>%  **gather**(“question”, “score”, moral1:pathogen7) %>%  **group\_by**(user\_id) %>%  **mutate**(score\_smc = score - **mean**(score, na.rm = TRUE)) |

## 5.5.7 Combinaison des différentes fonctions

Beaucoup des actions que nous avons réalisées plus tôt seraient facilitées avec des données dans un format tidy. Nous pourrions alors utiliser group\_by pour calculer différents scores.

|  |  |
| --- | --- |
|  | C’est un bon exercice d’utiliser ungroup() après avoir utilisé group\_by et summarise. Oublier de rompre les groupes dans le jeux de données n’affectera pas les traitements ultérieurs, cependant, cela pourrait beaucoup perturber d’autres actions. |

Ainsi, nous pouvons répartir les 3 domaines : calculer le score total, supprimer les lignes où le total est absent (NA) et calculer la moyenne par an.

|  |
| --- |
| disgust\_tidy <- **read\_csv**("data/disgust.csv") %>%  **gather**(“question”, “score”, moral1:pathogen7) %>%  **separate**(question, **c**("domain","q\_num"), sep = -1) %>%  **group\_by**(user\_id) %>%  **summarise**(score = **mean**(score)) %>%  **mutate**(score\_smc = score - **mean**(score, na.rm = TRUE)) |

|  |
| --- |
| ## Parsed with column specification:  ## cols(  ## .default = col\_double(),  ## date = col\_date(format = "")  ## ) |

|  |
| --- |
| ## See spec(...) for full column specifications. |

|  |
| --- |
| disgust\_tidy2 <- disgust\_tidy %>%  **spread**(domain, score) %>%  **mutate**(  total = moral + sexual + pathogen,  year = **year**(date)  ) %>%  **filter**(!**is.na**(total)) %>%  **arrange**(user\_id)  disgust\_tidy3 <- disgust\_tidy2 %>%  **group\_by**(year) %>%  **summarise**(  n = **n**(),  avg\_pathogen = **mean**(pathogen),  avg\_moral = **mean**(moral),  avg\_sexual = **mean**(sexual),  first\_user = **first**(user\_id),  last\_user = **last**(user\_id)  ) |

## 

# 5.6 Verbes one-table dplyr additionnels

Vous pouvez utiliser les exemples de codes qui suivent et la page d’aide pour trouver à quoi servent les verbes one-table qui suivent. La plupart ont un nom explicite.

## 5.6.1 rename()

|  |
| --- |
| iris\_underscore <- iris %>%  **rename**(  sepal\_length = Sepal.Length,  sepal\_width = Sepal.Width,  petal\_length =Petal.Length,  petal\_width = Petal.Width)  **name**(iris\_underscore) |

|  |
| --- |
| ## [1] "sepal\_length" "sepal\_width" "petal\_length" "petal\_width"  ## [5] "Species" |

|  |
| --- |
| La plupart du monde s'emmêle les pinceaux un jour ou l’autre avec rename() et essaie de mettre le nom original de la colonne à gauche et le nouveau nom à droite. Vous pouvez essayer pour voir le message d’erreur qui apparaît à cette occasion. |

### 

## 5.6.2 distinct()

|  |
| --- |
| *# create a data table with duplicated values*  dupes <- **tibble**(  id = **rep**(1 : 5, 2),  dv = **rep**(LETTERS[1 : 5], 2)  )  **distinct**(dupes) |

|  |
| --- |
| ## # A tibble: 5 x 2  ## id dv  ## <int> <chr>  ## 1 1 A  ## 2 2 B  ## 3 3 C  ## 4 4 D  ## 5 5 E |

### 

## 5.6.3 count()

|  |
| --- |
| *# how many observations from each species are in iris?*  **count**(iris, Species) |

### 

|  |
| --- |
| ## # A tibble: 3 x 2  ## Species n  ## <fct> <int>  ## 1 setosa 50  ## 2 versicolor 50  ## 3 virginica 50 |

### 

## 5.6.4 slice()

|  |
| --- |
| **tibble**(  id = 1 : 10,  condition = **rep**(**c**(“A”, “B”), 5)  ) %>%  **slice**(3:6, 9) |

|  |
| --- |
| ## # A tibble: 5 x 2  ## id condition  ## <int> <chr>  ## 1 3 A  ## 2 4 B  ## 3 5 A  ## 4 6 B  ## 5 9 A |

### 

## 5.6.5 pull()

|  |
| --- |
| iris %>%  **group\_by**(Species) %>%  **summarise\_all**(mean) %>%  **pull**(Sepal.Length) |

|  |
| --- |
| ## [1] 5.006 5.936 6.588 |

# 5.7 Fonctions Window

Les fonctions Window utilisent l’ordre des lignes pour calculer des valeurs. Vous pouvez les utiliser pour réaliser des actions qui nécessitent un rang ou un ordre, comme choisir le score maximal pour chaque classe ou accéder à la première et dernière ligne, et comme calculer des sommes ou des moyennes cumulatives.

La [vignette de fonction dplyr window](https://dplyr.tidyverse.org/articles/window-functions.html) *(EN)* donne de nombreuses explications de ces fonctions, mais nous allons tout de même vous fournir quelques exemples des fonctions les plus utiles ci-dessous.

## 5.7.1 Fonctions pour attribuer un rang à ses données

|  |
| --- |
| **tibble** (  id = 1:5,  “Data Skills” = **c**(16, 17, 17, 19, 20),  “Statistics” = **c**(14, 16, 18, 18, 19)  ) %>%  **gather**(class, grade, 2:3) %>%  **group\_by**(class) %>%  **mutate**(row\_number = **row\_number**(),  rank = **rank**(grade),  min\_rank = **min\_rank**(grade),  dense\_rank = **dense\_rank**(grade),  quartile = **ntile**(grade, 4),  percentile = **ntile**(grade, 100)) |

### 

|  |
| --- |
| ## # A tibble: 10 x 9  ## # Groups: class [2]  ## id class grade row\_number rank min\_rank dense\_rank  ## <int> <chr> <dbl> <int> <dbl> <int> <int>  ## 1 1 Data… 16 1 1 1 1  ## 2 2 Data… 17 2 2.5 2 2  ## 3 3 Data… 17 3 2.5 2 2  ## 4 4 Data… 19 4 4 4 3  ## 5 5 Data… 20 5 5 5 4  ## 6 1 Stat… 14 1 1 1 1  ## 7 2 Stat… 16 2 2 2 2  ## 8 3 Stat… 18 3 3.5 3 3  ## 9 4 Stat… 18 4 3.5 3 3  ## 10 5 Stat… 19 5 5 5 4  ## # … with 2 more variables: quartile <int>, percentile <int> |

|  |
| --- |
| * Quelles sont les différences entre row\_number(), rank(), min\_rank(), dense\_rank(), et ntile() ? * Pourquoi row\_number() ne nécessite pas d’argument ? * Que se passerait t’il si vous écriviez l’argument grade ou class pour row\_number() ? * Que pensez-vous qu’il arriverait si vous supprimiez la ligne group\_by(class), ci-dessus ? * Et que pensez-vous qu’il surviendrait si vous ajoutiez id comme argument au verbe group\_by ? * Que se passe t’il si vous changez l’ordre des lignes? * Que permet de faire le second argument dans ntile() ? |

Vous pouvez utiliser une fonction window pour regrouper les données en quantiles.

|  |
| --- |
| iris %>%  **group\_by**(tertile = **ntile**(Sepal.Length, 3)) %>%  **summarise**(mean.Sepal.Length = **mean**(Sepal.Length)) |

### 

|  |
| --- |
| ## # A tibble: 3 x 2  ## tertile mean.Sepal.Length  ## <int> <dbl>  ## 1 1 4.94  ## 2 2 5.81  ## 3 3 6.78 |

## 5.7.2 Fonctions offset

|  |
| --- |
| **tibble** (  trial = 1:10,  cond = **rep**(**c**(“exp”, “ctrl”), **c**(6, 4)),  score = **rpois**(10, 4)  ) %>%  **mutate**(  score\_change = score - **lag**(score, order\_by = trial),  last\_cond\_trial = cond != **lead**(cond, default = TRUE)  ) |

### 

|  |
| --- |
| ## # A tibble: 10 x 5  ## trial cond score score\_change last\_cond\_trial  ## <int> <chr> <int> <int> <lgl>  ## 1 1 exp 2 NA FALSE  ## 2 2 exp 4 2 FALSE  ## 3 3 exp 5 1 FALSE  ## 4 4 exp 5 0 FALSE  ## 5 5 exp 3 -2 FALSE  ## 6 6 exp 5 2 TRUE  ## 7 7 ctrl 9 4 FALSE  ## 8 8 ctrl 6 -3 FALSE  ## 9 9 ctrl 6 0 FALSE  ## 10 10 ctrl 4 -2 TRUE |

|  |
| --- |
| Veuillez maintenant regarder les pages d’aide pour lag() et lead().   * Qu’arrive t’il si vous supprimez l’argument order\_by ou que vous remplacez trial par cond ? * Que fait l’argument default ? * Pouvez-vous trouver des circonstances pour lesquelles les fonctions lag() et lead() vous seraient utiles pour traiter vos propres données ? |

## 5.7.3 Fonctions cumulatives

cumsum(), cummin(), et cummax() sont des fonctions R qui permettent de calculer des moyennes cumulées, des minimums cumulés, ainsi que des maximums cumulés. Le package dplyr introduit cumany() et cumall(), qui renvoient TRUE dès qu’une ou plusieurs valeur(s) précédente(s) répond(ent) à leur critère.

|  |
| --- |
| **tibble** (  time = 1:10,  obs = **c**(1, 0, 1, 2, 4, 3, 1, 0, 3, 5)  ) %>%  **mutate**(  cumsum = **cumsum**(obs),  cummin = **cummin**(obs),  cummax = **cummax**(obs),  cumany = **cumany**(obs == 3),  cumall = **cumall**(obs < 4)  ) |

### 

|  |
| --- |
| ## # A tibble: 10 x 7  ## time obs cumsum cummin cummax cumany cumall  ## <int> <dbl> <dbl> <dbl> <dbl> <lgl> <lgl>  ## 1 1 1 1 1 1 FALSE TRUE  ## 2 2 0 1 0 1 FALSE TRUE  ## 3 3 1 2 0 1 FALSE TRUE  ## 4 4 2 4 0 2 FALSE TRUE  ## 5 5 4 8 0 4 FALSE FALSE  ## 6 6 3 11 0 4 TRUE FALSE  ## 7 7 1 12 0 4 TRUE FALSE  ## 8 8 0 12 0 4 TRUE FALSE  ## 9 9 3 15 0 4 TRUE FALSE  ## 10 10 5 20 0 5 TRUE FALSE |

|  |
| --- |
| * Qu’est ce qui surviendrait si vous remplaciez cumany(obs == 3) par cumany(obs > 2) ? * Qu’est ce qui surviendrait si vous changiez cumall(obs < 4) par cumall(obs < 2) ? * Pouvez-vous trouver des circonstances pour lesquelles les fonctions cumany() et cumall() vous seraient utiles pour traiter vos propres données ? |

# 5.8 Exercices

Téléchargez les [exercices](https://psyteachr.github.io/msc-data-skills/exercises/05_dplyr_exercise.Rmd) *(EN)*. Vous pouvez aussi consulter les [solutions](https://psyteachr.github.io/msc-data-skills/exercises/05_dplyr_answers.Rmd) *(EN)* une fois après avoir essayé de répondre à toutes les questions.