04 - Types de données et transformations

PRO1036 - Analyse de données scientifiques en R

Tim Bollé

October 6, 2025

Pourquoi s’intéresser aux types de données ?

Exemple: Cat lovers

Un sondage a demandé à des gens leur nom et le nombre de chat qu'ils possèdent. Les instructions indiquaient d'entrer le nombre de chats en chiffres.

```
cat_lovers <- read_csv("data/cat-lovers.csv")
```

```
# A tibble: 60 × 3
   name            number_of_cats handedness
   <chr>           <chr>          <chr>
 1 Bernice Warren  0              left
 2 Woodrow Stone   0              left
 3 Willie Bass     1              left
 4 Tyrone Estrada  3              left
 5 Alex Daniels    3              left
 6 Jane Bates      2              left
 7 Latoya Simpson  1              left
 8 Darin Woods     1              left
 9 Agnes Cobb      0              left
10 Tabitha Grant   0              left
# ℹ 50 more rows
```

Statistique simple…

```
cat_lovers %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
# A tibble: 1 × 1
  mean_cats
      <dbl>
1        NA
```

Ca ne marche pas…

Pourquoi ?

```
1  ?mean
```

mean {base} R Documentation

Arithmetic Mean

Description

Generic function for the (trimmed) arithmetic mean.

Usage

```
mean(x, ...)

## Default S3 method:
mean(x, trim = 0, na.rm = FALSE, ...)
```

Arguments

`x`	An `R` object. Currently there are methods for numeric/logical vectors and date, date-time and time interval objects. Complex vectors are allowed for `trim = 0`, only.
`trim`	the fraction (0 to 0.5) of observations to be trimmed from each end of `x` before the mean is computed. Values of trim outside that range are taken as the nearest endpoint.
`na.rm`	a logical value indicating whether `NA` values should be stripped before the computation proceeds.
`...`	further arguments passed to or from other methods.

Et maintenant ?

```
cat_lovers %>%
  summarise(mean_cats = mean(number_of_cats, na.rm = TRUE))
```

```
# A tibble: 1 × 1
  mean_cats
      <dbl>
1        NA
```

Toujours pas…

Regardons les données

```
          1  glimpse(cat_lovers)
```

```
Rows: 60
Columns: 3
$ name           <chr> "Bernice Warren", "Woodrow Stone", "Willie Bass", "Tyro…
$ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1", "0", "0", "0", …
$ handedness     <chr> "left", "left", "left", "left", "left", "left", "left",…
```

Il suffit de convertir !

```
cat_lovers %>%
  mutate(
    number_of_cats = as.numeric(number_of_cats)
  ) %>%
  summarise(mean_cats = mean(number_of_cats, na.rm = TRUE))
```

```
# A tibble: 1 × 1
  mean_cats
      <dbl>
1     0.776
```

Ok mais …

```
Warning: There was 1 warning in `mutate()`.
ℹ In argument: `number_of_cats = as.numeric(number_of_cats)`.
Caused by warning:
! NAs introduits lors de la conversion automatique
```

Regardons cela de plus près…

Show 10 entries Search:

	name	number_of_cats	handedness
1	Bernice Warren	0	left
2	Woodrow Stone	0	left
3	Willie Bass	1	left
4	Tyrone Estrada	3	left
5	Alex Daniels	3	left
6	Jane Bates	2	left
7	Latoya Simpson	1	left
8	Darin Woods	1	left
9	Agnes Cobb	0	left
10	Tabitha Grant	0	left

Showing 1 to 10 of 60 entries Previous 1 2 3 4 5 6 Next

Respecter les types de données

```
cat_lovers %>%
  mutate(
    number_of_cats = case_when(
      name == "Ginger Clark" ~ "2",
      name == "Doug Bass"    ~ "3",
      TRUE                   ~ number_of_cats
      ),
    number_of_cats = as.numeric(number_of_cats)
    ) %>%
  summarise(mean_cats = mean(number_of_cats))
```

```
# A tibble: 1 × 1
  mean_cats
      <dbl>
1     0.833
```

Enregistrer

```
1  cat_lovers <- cat_lovers %>%
2    mutate(
3      number_of_cats = case_when(
4        name == "Ginger Clark" ~ "2",
5        name == "Doug Bass"    ~ "3",
6        TRUE                   ~ number_of_cats
7        ),
8      number_of_cats = as.numeric(number_of_cats)
9      )
```

Morale

- Si vos données ne se comportent pas comme vous l’attendez, il se peut qu’il s’agisse d’un probplème de type de données.
- Explorez et investiguez vos données, appliquez les modifications, *sauvegardez vos données*, vivez heureux

Types de données

Types de données dans R

- logical
- double
- integer
- character
- Il y a en d’autres mais nous les utiliserons peu

Logical et character

logical - Valeurs booléennes TRUE et FALSE

```
1  typeof(TRUE)
```

```
[1] "logical"
```

character - Texte

```
1  typeof("Hello")
```

```
[1] "character"
```

Double et integer

double - Nombres à virgule flottante (type par défaut pour les nombres)

```
typeof(1.5)
```

```
[1] "double"
```

```
typeof(7)
```

```
[1] "double"
```

integer - Nombres entiers (indiqué par un L)

```
typeof(1L)
```

```
[1] "integer"
```

```
typeof(1:3)
```

```
[1] "integer"
```

Concatenation

Des vecteurs peuvent être construits à l’aide de la fonction c( )

```
          1  c(1, 2, 3)
[1] 1 2 3
          1  c("Hello", "World!")
[1] "Hello"  "World!"
          1  c(c("hi", "hello"), c("bye", "jello"))
[1] "hi"    "hello" "bye"   "jello"
```

Conversion

… intentionnelle

```
1 x <- 1:3
2 x
```

```
[1] 1 2 3
```

```
1 typeof(x)
```

```
[1] "integer"
```

```
1 x <- c(TRUE, FALSE)
2 x
```

```
[1]  TRUE FALSE
```

```
1 typeof(x)
```

```
[1] "logical"
```

```
1 y <- as.character(x)
2 y
```

```
[1] "1" "2" "3"
```

```
1 typeof(y)
```

```
[1] "character"
```

```
1 y <- as.numeric(x)
2 y
```

```
[1] 1 0
```

```
1 typeof(y)
```

```
[1] "double"
```

Conversion

… accidentelle

R va faire les conversions sans se poser de questions, surtout quand on mets différentes choses dans un même vecteur.

```
          1  c(1, "Hello")
[1] "1"     "Hello"
          1  c(FALSE, 3L)
[1] 0 3
```

```
          1  c(1.2, 3L)
[1] 1.2 3.0
          1  c(2L, "two")
[1] "2"   "two"
```

Conversion

- Explicite - Utilisez les fonctions `as.*()`
  - `as.logical()`
  - `as.numeric()`
  - `as.integer()`
  - `as.character()`
  - `as.double()`
- Implicite - R va faire les conversions pour vous, soyez vigilant

Cas spéciaux

Cas spéciaux

- `NA` - Valeur manquante
- `Inf` - Infini
- `NaN` - Not a Number

```r
# division par zéro -> Inf
7/0
```

```
[1] Inf
```

```r
-1/0
```

```
[1] -Inf
```

```r
Inf - Inf
```

```
[1] NaN
```

NA

```r
x <- c(1, 2, 3, 4, NA)
```

```r
mean(x)
```

```
[1] NA
```

```r
mean(x, na.rm = TRUE)
```

```
[1] 2.5
```

```r
summary(x)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   1.00    1.75    2.50    2.50    3.25    4.00       1
```

NA

Les NA sont utilisés par R pour représenter des valeurs manquantes. Ils sont de type logique.

```
typeof(NA)
```

```
[1] "logical"
```

```
# TRUE or NA
TRUE | NA
```

```
[1] TRUE
```

```
# FALSE or NA
FALSE | NA
```

```
[1] NA
```

Classes de données

Classes de données

Nous avons parlé des *types* des données et nous allons maintenant parler des *classes* des données:

- Vecteurs sont comme des briques LEGO
- On les colle ensemble pour construire des structures plus compliquées, notamment des *représentations de données*

Par exemples:

- factor
- date
- data frame

factor

Les facteurs sont utilisés pour gérer les variables catégorielles, c'est-à-dire les variables qui ont un ensemble fixe et connu de valeurs possibles.

```
x <- factor(c("BS", "MS", "PhD", "MS"))
x
```

```
[1] BS  MS  PhD MS
Levels: BS MS PhD
```

```
typeof(x)
```

```
[1] "integer"
```

```
class(x)
```

```
[1] "factor"
```

factor

Pour chaque facteur on a:

- `levels` - Les valeurs possibles
- `labels` - Les étiquettes associées aux valeurs

```
1 glimpse(x)
```

```
Factor w/ 3 levels "BS","MS","PhD": 1 2 3 2
```

```
1 as.integer(x)
```

```
[1] 1 2 3 2
```

Dates

```
y <- as.Date("2024-01-01")
y
```

```
[1] "2024-01-01"
```

```
typeof(y)
```

```
[1] "double"
```

```
class(y)
```

```
[1] "Date"
```

Dates

Les dates sont en réalité un entier (le nombre de jours depuis l'origine, 1 Jan 1970) et un entier (l'origine) collés ensemble

```
          1  as.integer(y)
```

```
[1] 19723
```

```
          1  as.integer(y) / 365
```

```
[1] 54.03562
```

Data frames

Les data frames sont des structures de données qui sont comme des vecteurs de différentes classes.

```
df <- data.frame(x = 1:2, y = 3:4)
df
```

```
  x y
1 1 3
2 2 4
```

```
typeof(df)
```

```
[1] "list"
```

```
class(df)
```

```
[1] "data.frame"
```

Listes

Les listes sont des conteneurs génériques pour des vecteurs de n’importe quel type.

```r
l <- list(
  x = 1:4,
  y = c("hi", "hello", "jello"),
  z = c(TRUE, FALSE)
)
l
```

```
$x
[1] 1 2 3 4

$y
[1] "hi"    "hello" "jello"

$z
[1]  TRUE FALSE
```

Listes et data frames

- Un data frame est une liste spéciale contenant des vecteurs de longueur égale
- Un `tibble` est un data frame du Tidyverse

```
df
```

```
  x y
1 1 3
2 2 4
```

```
df %>%
  as_tibble()
```

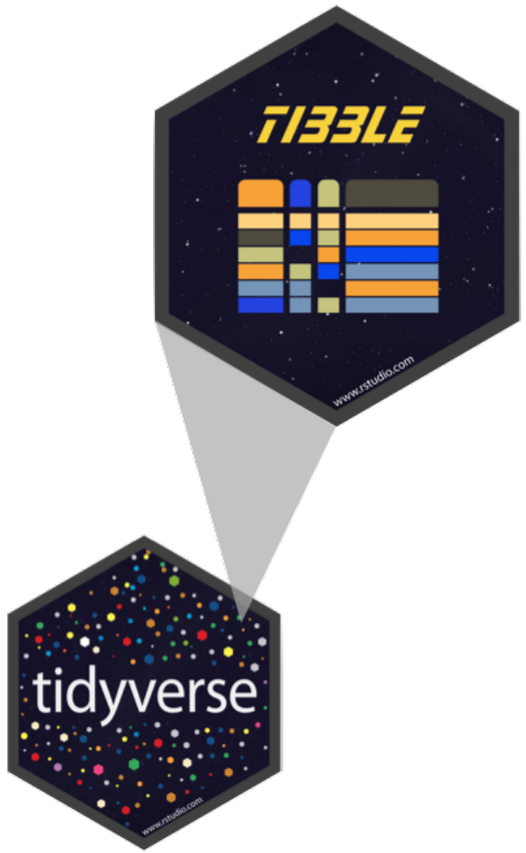
```
# A tibble: 2 × 2
      x     y
  <int> <int>
1     1     3
2     2     4
```

```
df %>%
  as_tibble() %>%
  class()
```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

Tibbles

Les Tibbles sont des data frames modernes. Ils sont adaptés au tidyverse (en pratique, toutes les fonctions du tidyverse retournent des `tibbles` au lieu de `data.frame`).



- Les tibbles ont un affichage limité, adapté à la console
- Il est possible d'accéder à une colonne à l'aide des double crochets (`df[["x"]]`) en plus du `$` (`df$x`). Il est ainsi possible d'utiliser le numéro de colonne au lieu du nom (`df[[1]]`)
- Les tibbles fournissent également plus d'indications lorsque quelque chose ne va pas (messages de *warnings*).

Travailler avec des facteurs

Exemple: Cat lovers

Nous commençons avec un data frame avec des caractères

```
glimpse(cat_lovers)
```

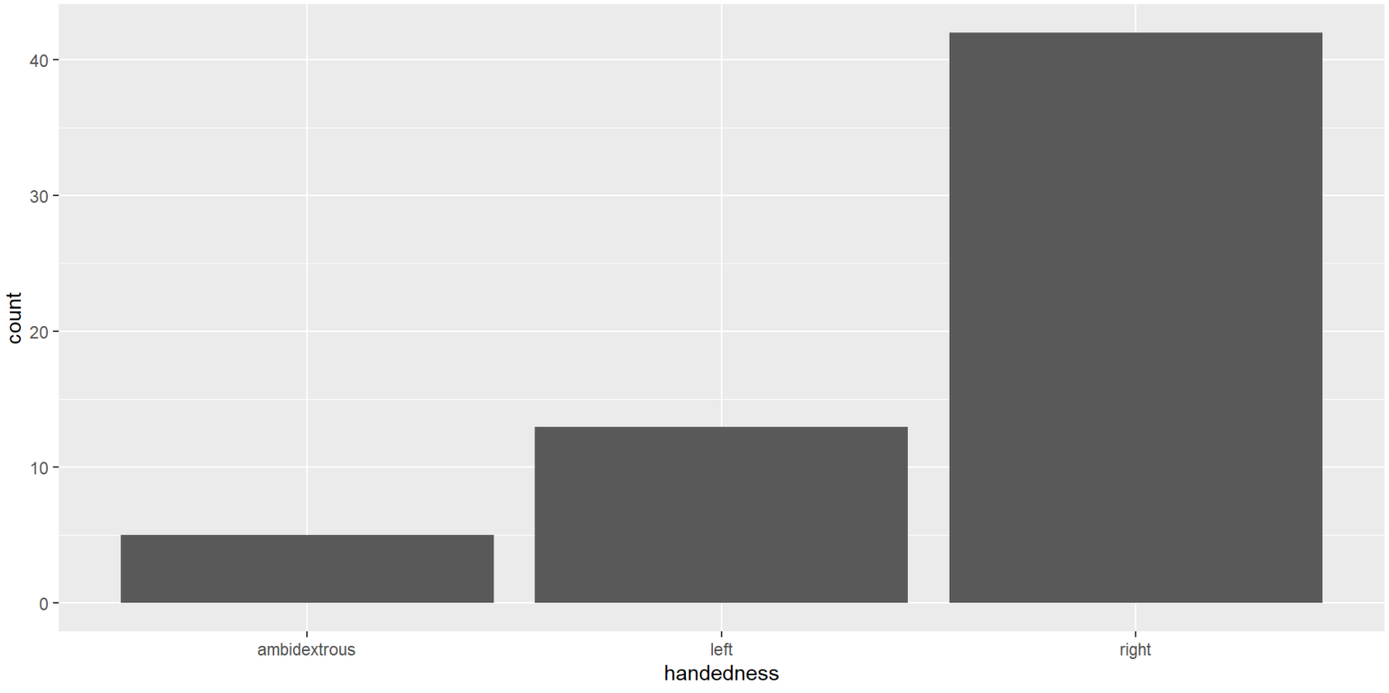
```
Rows: 60
Columns: 3
$ name           <chr> "Bernice Warren", "Woodrow Stone", "Willie Bass", "Tyro…
$ number_of_cats <chr> "0", "0", "1", "3", "3", "2", "1", "1", "0", "0", "0", …
$ handedness     <chr> "left", "left", "left", "left", "left", "left", "left",…
```

Plotting

Au moment de faire un graphique, R va faire une conversion de type

Si on lui demande de plotter des catégories, il va créer des facteurs

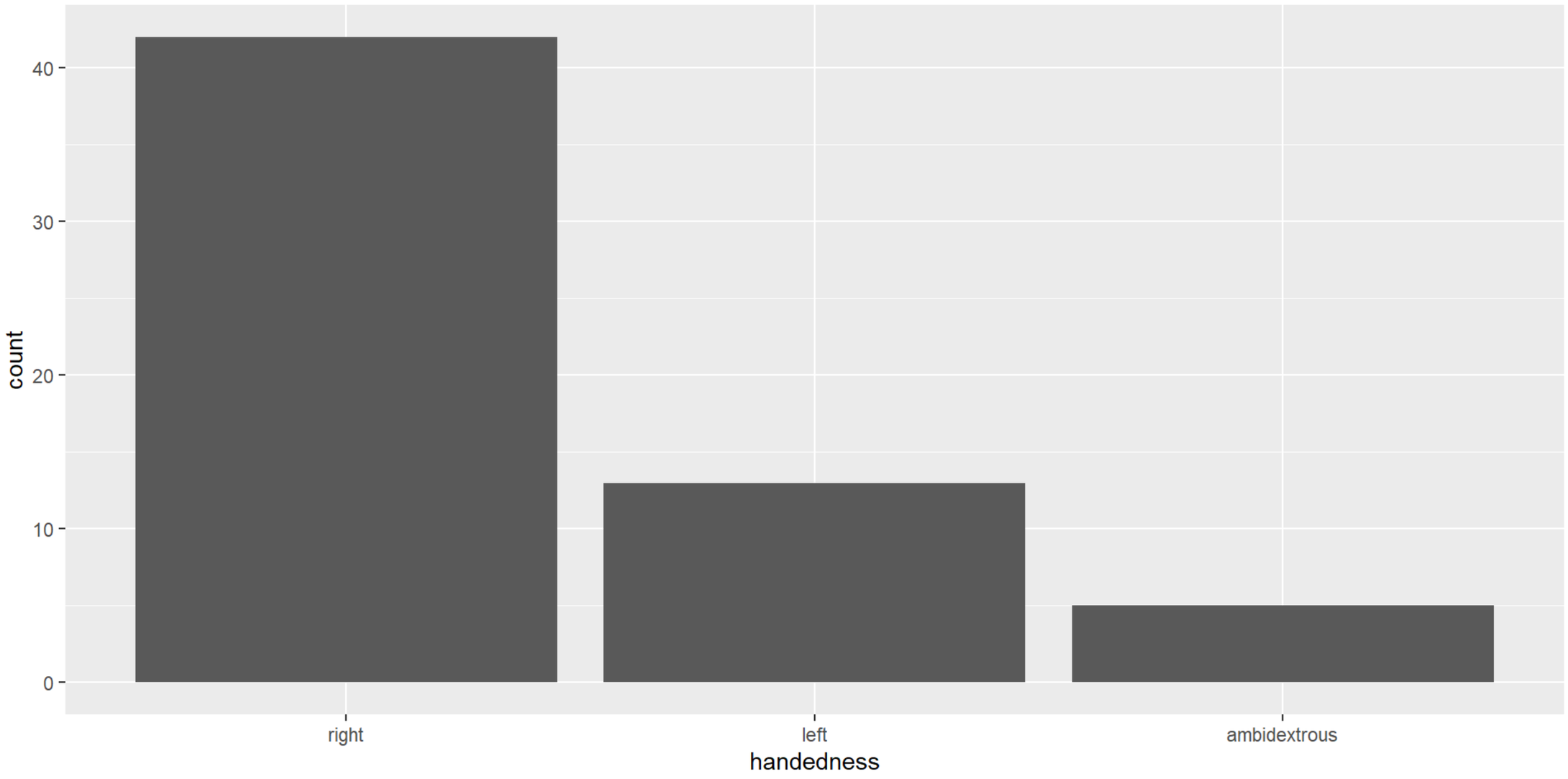
```
1  ggplot(cat_lovers, mapping = aes(x = handedness)) +
2    geom_bar()
```



Les facteurs sont ordonnées par ordre alphabétique par défaut

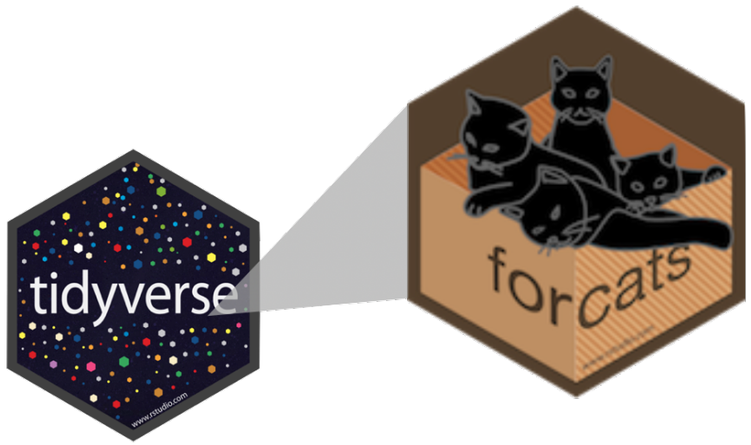
On peut manipuler les facteurs avec `forcats`

```
1  cat_lovers %>%
2    mutate(handedness = fct_infreq(handedness)) %>%
3    ggplot(mapping = aes(x = handedness)) +
4    geom_bar()
```



La grammaire de la manipulation de données

Basé sur des fonctions qui correspondent à des verbes permettant de manipuler des dataframes.



- Les facteurs sont utiles lorsque vous avez des données catégorielles et que vous voulez remplacer l'ordre des vecteurs de caractères pour améliorer l'affichage
- Le package **forcats** fournit une suite d'outils utiles qui résolvent des problèmes courants avec les facteurs

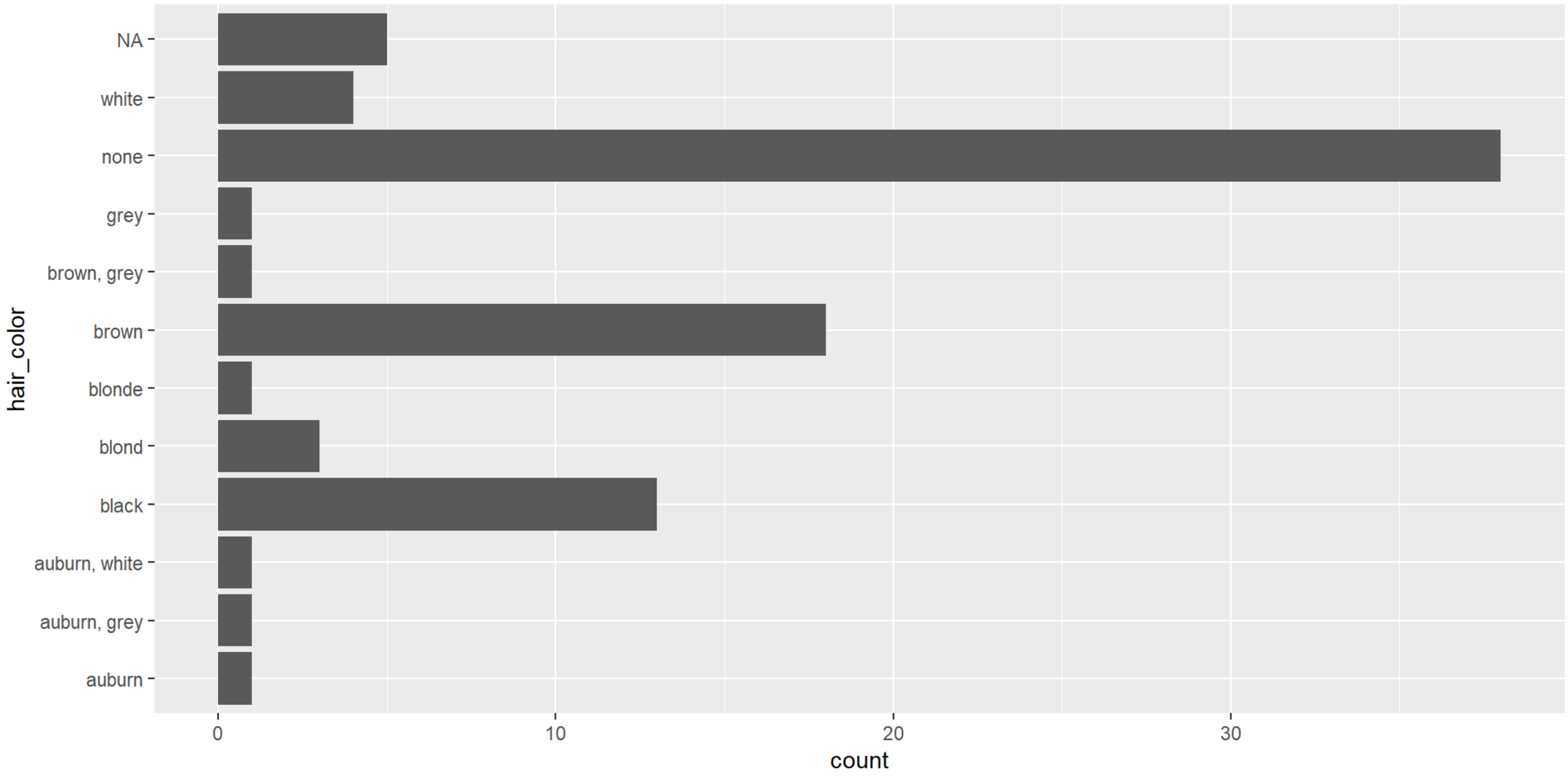
Les fonctions de `forcats`

`focats` possède plusieurs fonctions pour manipuler les facteurs:

- `fct_reorder()` - Réordonne les niveaux d'un facteur en fonction d'une autre variable
- `fct_relevel()` - Réordonne les niveaux d'un facteur
- `fct_infreq()` - Réordonne les niveaux d'un facteur en fonction de leur fréquence
- `fct_lump()` - Regroupe les niveaux d'un facteur en "autres"
- `fct_explicit_na()` - Ajoute un niveau pour les valeurs manquantes

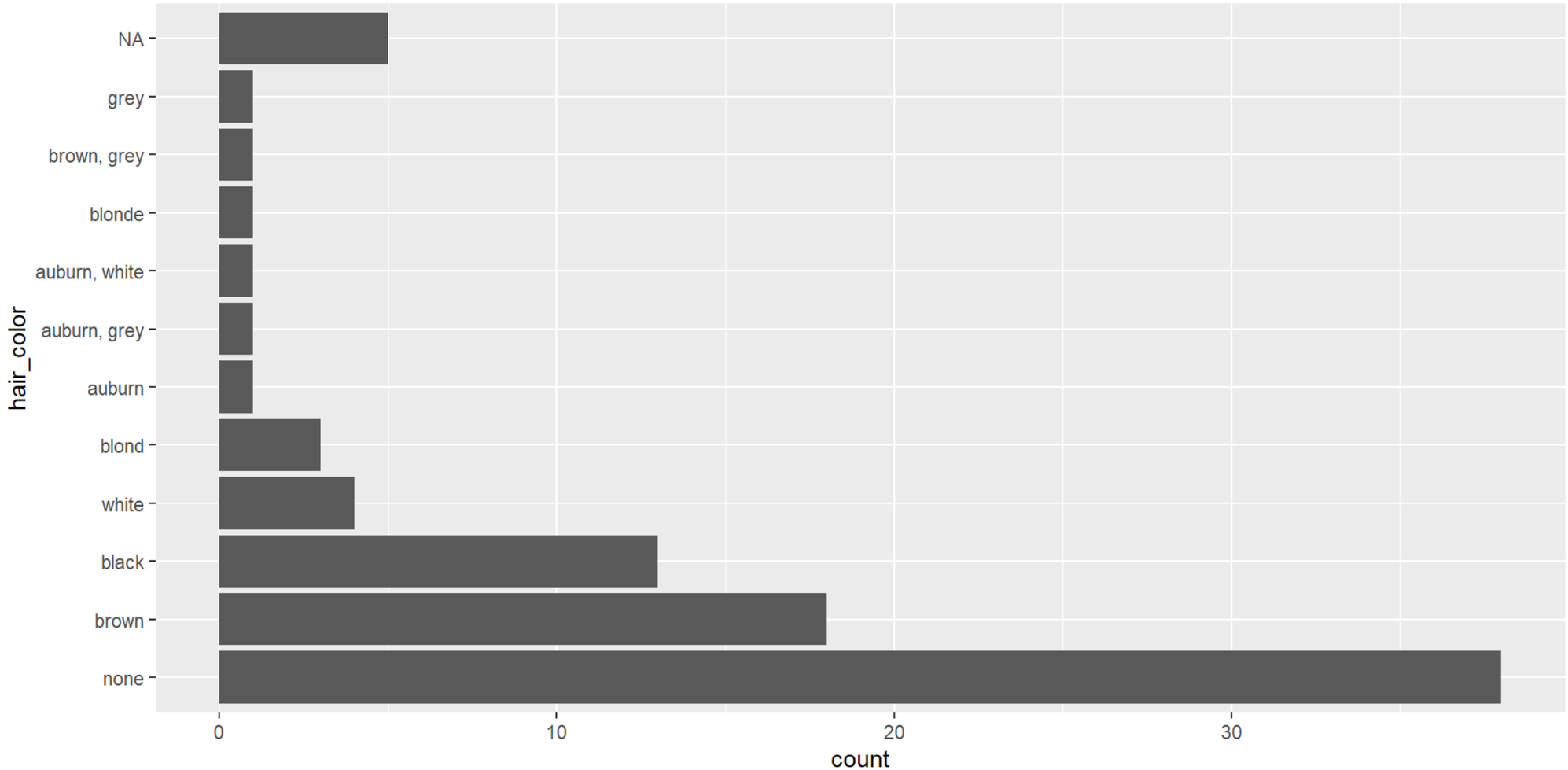
Exemple: starwars

```
1  starwars %>%
2    ggplot(aes(y = hair_color)) +
3    geom_bar()
```



Ordre selon la fréquence : `fct_infreq()`

```
1  starwars %>%
2    mutate(
3      hair_color = fct_infreq(hair_color)
4    ) %>%
5    ggplot(aes(y = hair_color)) +
6    geom_bar()
```



Regroupement de facteurs

Utile quand on a beaucoup de niveaux

```
starwars %>%
  count(skin_color, sort = TRUE)
```

```
# A tibble: 31 × 2
   skin_color     n
   <chr>      <int>
 1 fair          17
 2 light         11
 3 dark           6
 4 green          6
 5 grey           6
 6 pale           5
 7 brown          4
 8 blue           2
 9 blue, grey     2
10 none           2
# ℹ 21 more rows
```

… 31 niveaux dans cet exemple !

Regroupement de facteurs : `fct_lump()`

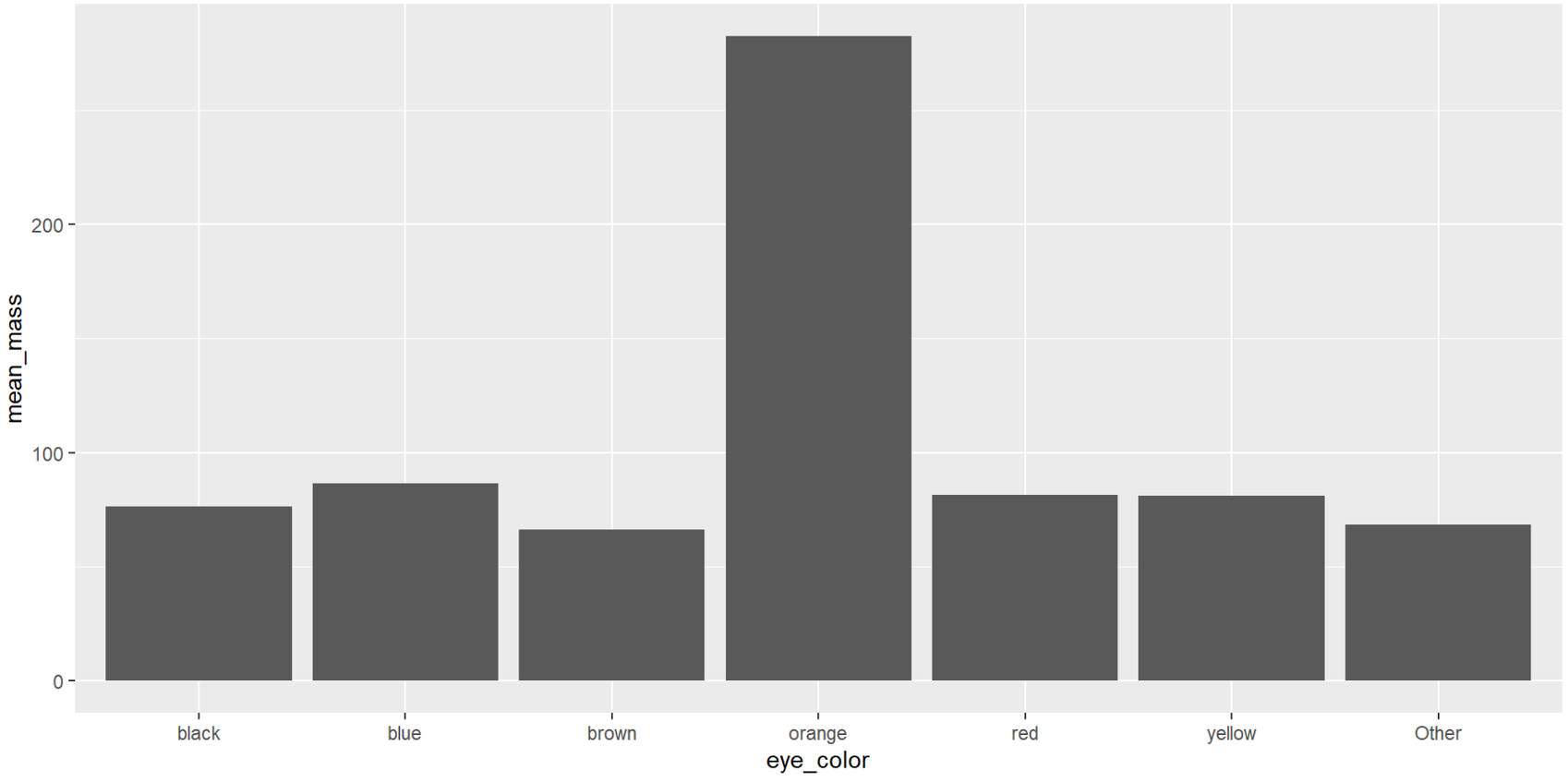
```
starwars %>%
  mutate(skin_color = fct_lump(skin_color, n = 5)) %>%
  count(skin_color, sort = TRUE)
```

```
# A tibble: 6 × 2
  skin_color     n
  <fct>      <int>
1 Other         41
2 fair          17
3 light         11
4 dark           6
5 green          6
6 grey           6
```

Regroupement de facteurs

Regardons maintenant la masse moyenne selon la couleur des yeux:

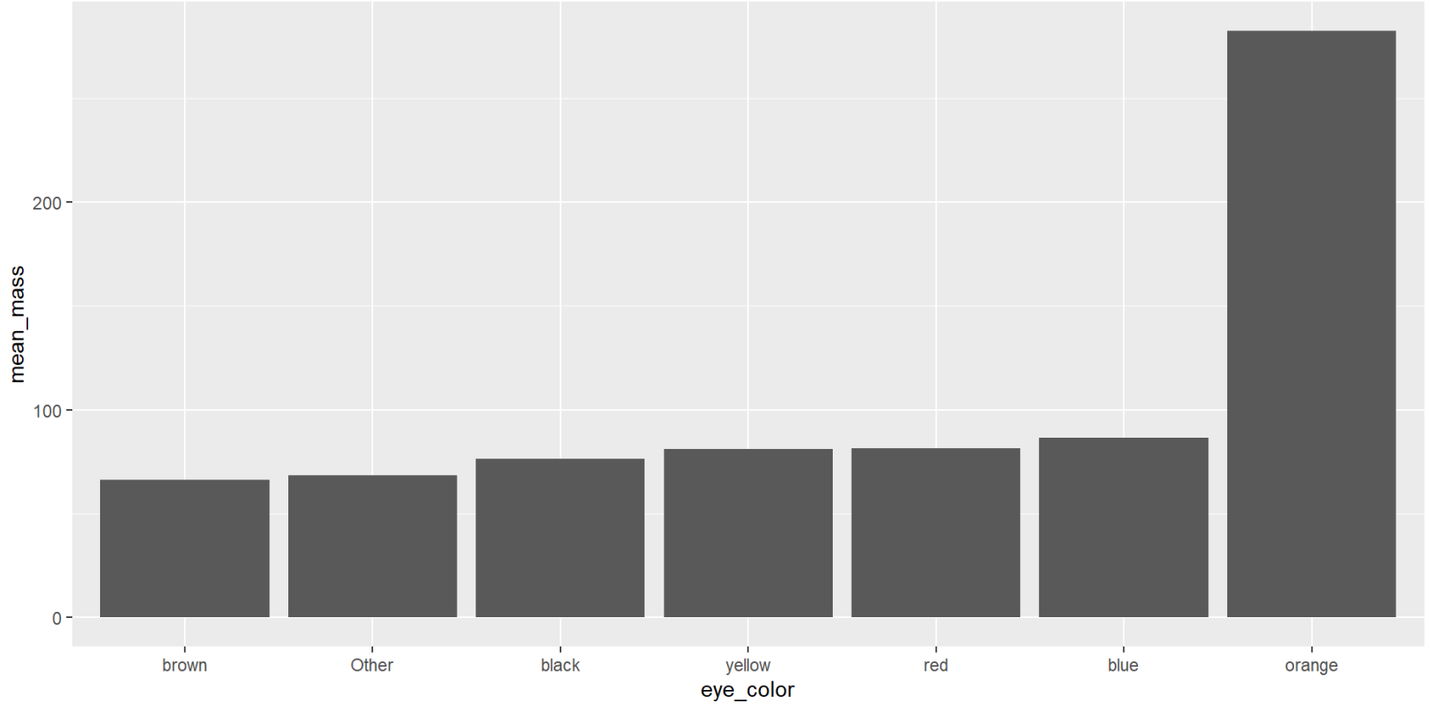
```
1  starwars %>%
2    mutate(eye_color = fct_lump(eye_color, n = 6)) %>%
3    group_by(eye_color) %>%
4    summarise(mean_mass = mean(mass, na.rm = TRUE)) %>%
5    ggplot(aes(x = eye_color, y = mean_mass)) +
6    geom_col()
```



Regroupement et changement d'ordre !

Nous allons maintenant regrouper les couleurs des yeux et les ordonner selon la masse moyenne

```
1  starwars %>%
2    mutate(eye_color = fct_lump(eye_color, n = 6)) %>%
3    group_by(eye_color) %>%
4    summarise(mean_mass = mean(mass, na.rm = TRUE)) %>%
5    mutate(eye_color = fct_reorder(eye_color, mean_mass)) %>%
6    ggplot(aes(x = eye_color, y = mean_mass)) +
7    geom_col()
```



Exemple: Hotels

Exemple: Hotels

Choix manuel de l'ordre : `fct_relevel()`

La variable `month.name` est une variable intégrée dans R qui contient les noms des mois en anglais et dans le bon ordre.

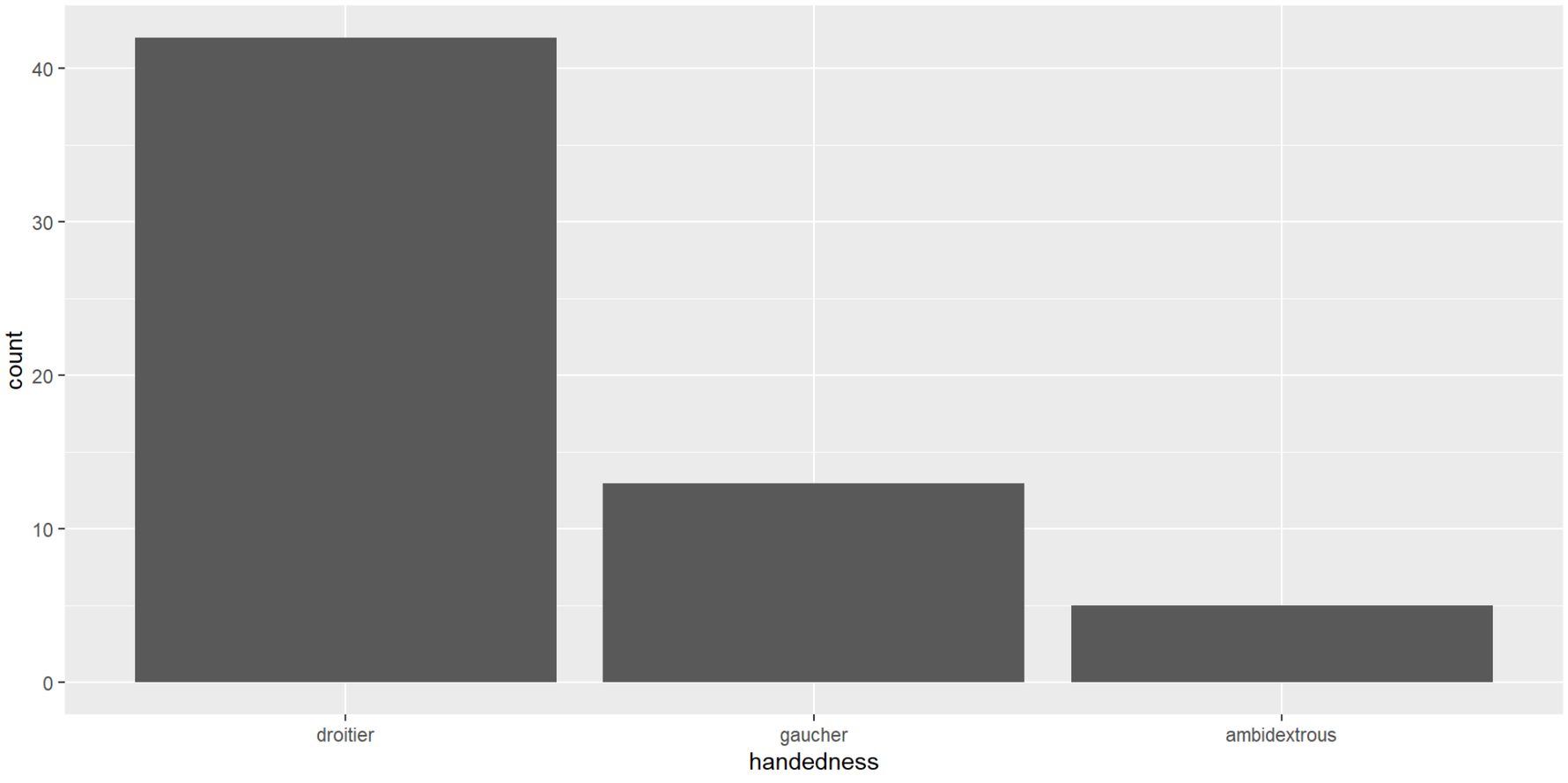
```r
hotels <- readr::read_csv("data/hotels.csv")

hotels %>%
  mutate(arrival_date_month = fct_relevel(arrival_date_month, month.name)) %>% # On réord
  group_by(hotel, arrival_date_month) %>%    # group by type d'hotel et mois d'arrivée
  summarise(mean_adr = mean(adr)) %>%        # calcul de l'ADR moyen pour chaque groupe
  ggplot(aes(
    x = arrival_date_month,
    y = mean_adr,                            # mean_adr sur y-axis
    group = hotel,                           # Groupe les lignes par type
    color = hotel)                           # couleur par type
    ) +
  geom_line() +                              # On veut des lignes
  scale_y_continuous(labels = label_dollar()) +
  theme_minimal() +                          # Utilise le minimal theme
  labs(x = "Arrival month",                  # On met à jour les labels
       y = "Mean ADR (average daily rate)",
       title = "Comparison of resort and city hotel prices across months",
       subtitle = "Resort hotel prices soar in the summer while city hotel prices remain\
       color = "Hotel type") +
  scale_x_discrete(guide = guide_axis(check.overlap = TRUE)) # On s'assure que les labels
```

Renommer les niveaux : `fct_recode()`

Si on veut simplement renommer les niveaux, on peut utiliser `fct_recode()`

```
1  cat_lovers %>%
2    mutate(handedness = fct_recode(handedness, gaucher = "left", droitier = "right")) %>%
3    mutate(handedness = fct_infreq(handedness)) %>%
4    ggplot(mapping = aes(x = handedness)) +
5    geom_bar()
```



Travailler avec les dates

Dates

- **lubridate** est un package tidyverse-friendly qui facilite la manipulation des dates
- Il ne fait pas partie du coeur tidyverse, donc il doit être installé avec `install.packages("lubridate")` mais il n'est pas chargé avec lui, et doit être explicitement chargé avec `library(lubridate)`.
  - ...

Exemple des hotels

Calculer et visualiser le nombre de réservations à une date d'arrivée donnée

Nous allons voir les bases mais travailler avec des dates peut s'avérer complexe mais cela peut apporter beaucoup à une analyse.

Show 10 entries Search:

	hotel	is_canceled	lead_time	arrival_date_year	arrival_date_month	arrival_date_week_number
1	Resort Hotel	0	342	2015	July	27
2	Resort Hotel	0	737	2015	July	27
3	Resort Hotel	0	7	2015	July	27

Showing 1 to 3 of 3 entries Previous 1 Next

Étape 1 - Construire la date

```
library(glue) # glue permet de coller des éléments ensemble

hotels %>%
  mutate(
    arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_of_mo
    ) %>%
  relocate(arrival_date) # pour afficher la nouvelle colonne à gauche
```

```
# A tibble: 119,390 × 33
   arrival_date hotel is_canceled lead_time arrival_date_year arrival_date_month
   <glue>       <chr>       <dbl>     <dbl>             <dbl> <chr>
 1 2015 July 1  Reso…           0       342              2015 July
 2 2015 July 1  Reso…           0       737              2015 July
 3 2015 July 1  Reso…           0         7              2015 July
 4 2015 July 1  Reso…           0        13              2015 July
 5 2015 July 1  Reso…           0        14              2015 July
 6 2015 July 1  Reso…           0        14              2015 July
 7 2015 July 1  Reso…           0         0              2015 July
 8 2015 July 1  Reso…           0         9              2015 July
 9 2015 July 1  Reso…           1        85              2015 July
10 2015 July 1  Reso…           1        75              2015 July
# ℹ 119,380 more rows
# ℹ 27 more variables: arrival_date_week_number <dbl>,
#   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,
#   stays_in_week_nights <dbl>, adults <dbl>, children <dbl>, babies <dbl>,
#   meal <chr>, country <chr>, market_segment <chr>,
#   distribution_channel <chr>, is_repeated_guest <dbl>,
#   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>, …
```

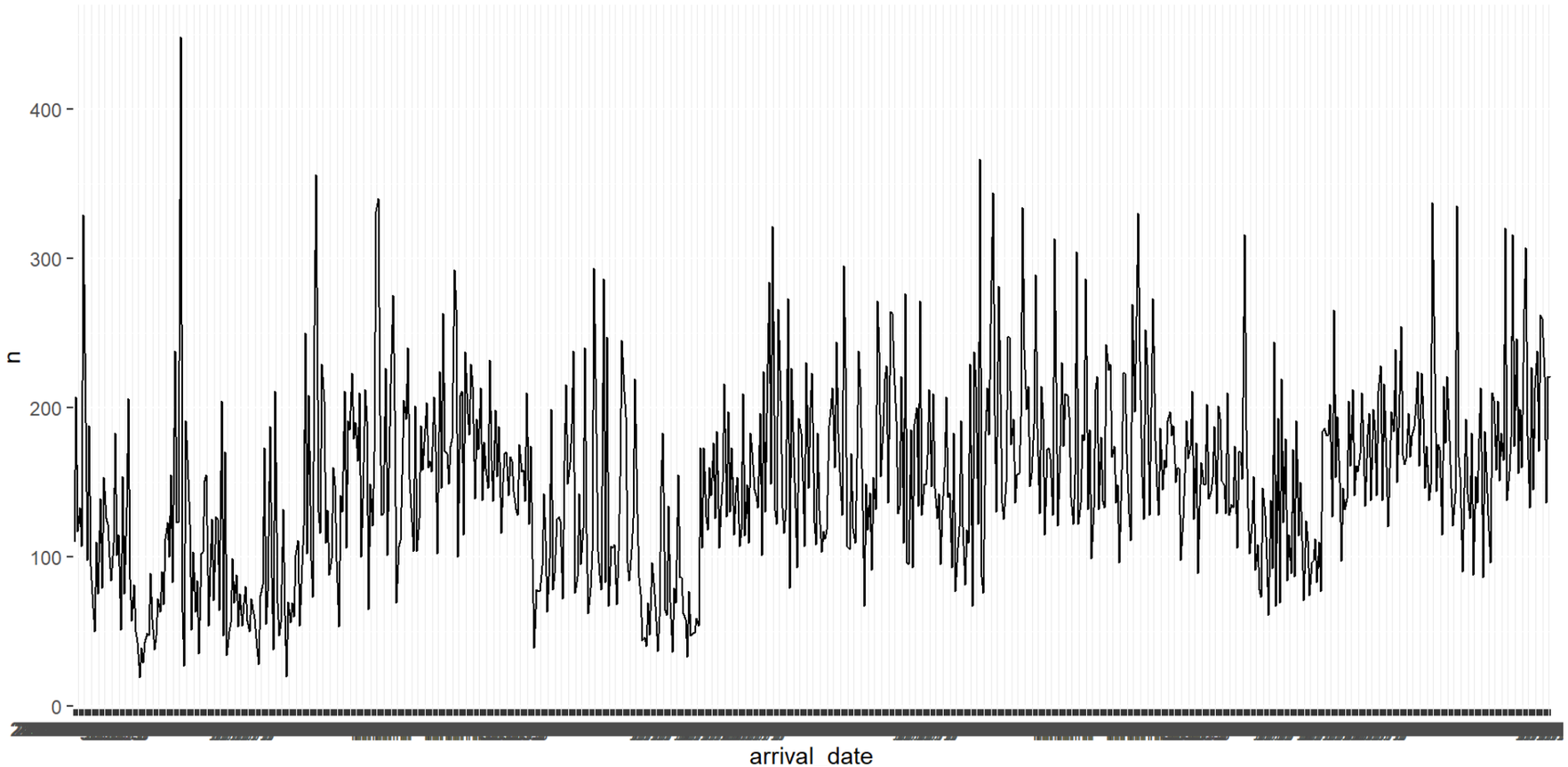
Étape 2 - Compter les réservations par dates

```
hotels %>%
  mutate(arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_
  count(arrival_date)
```

```
# A tibble: 793 × 2
   arrival_date       n
   <glue>         <int>
 1 2015 August 1    110
 2 2015 August 10   207
 3 2015 August 11   117
 4 2015 August 12   133
 5 2015 August 13   107
 6 2015 August 14   329
 7 2015 August 15   190
 8 2015 August 16    98
 9 2015 August 17   188
10 2015 August 18    94
# ℹ 783 more rows
```

Étape 4 - Visualiser

```
1 hotels %>%
2   mutate(arrival_date = glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_
3   count(arrival_date) %>%
4   ggplot(aes(x = arrival_date, y = n, group = 1)) +
5   geom_line()
```



Quel est le problème ici ?

Étape 1 - Construire la date (comme une date)

```
1 library(lubridate) # pour les dates
2
3 hotels %>%
4   mutate(
5     arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_day_o
6     ) %>%
7   relocate(arrival_date)
```

```
# A tibble: 119,390 × 33
   arrival_date hotel is_canceled lead_time arrival_date_year arrival_date_month
   <date>       <chr>       <dbl>     <dbl>             <dbl> <chr>
 1 2015-07-01   Reso…           0       342              2015 July
 2 2015-07-01   Reso…           0       737              2015 July
 3 2015-07-01   Reso…           0         7              2015 July
 4 2015-07-01   Reso…           0        13              2015 July
 5 2015-07-01   Reso…           0        14              2015 July
 6 2015-07-01   Reso…           0        14              2015 July
 7 2015-07-01   Reso…           0         0              2015 July
 8 2015-07-01   Reso…           0         9              2015 July
 9 2015-07-01   Reso…           1        85              2015 July
10 2015-07-01   Reso…           1        75              2015 July
# ℹ 119,380 more rows
# ℹ 27 more variables: arrival_date_week_number <dbl>,
#   arrival_date_day_of_month <dbl>, stays_in_weekend_nights <dbl>,
#   stays_in_week_nights <dbl>, adults <dbl>, children <dbl>, babies <dbl>,
#   meal <chr>, country <chr>, market_segment <chr>,
#   distribution_channel <chr>, is_repeated_guest <dbl>,
#   previous_cancellations <dbl>, previous_bookings_not_canceled <dbl>, …
```

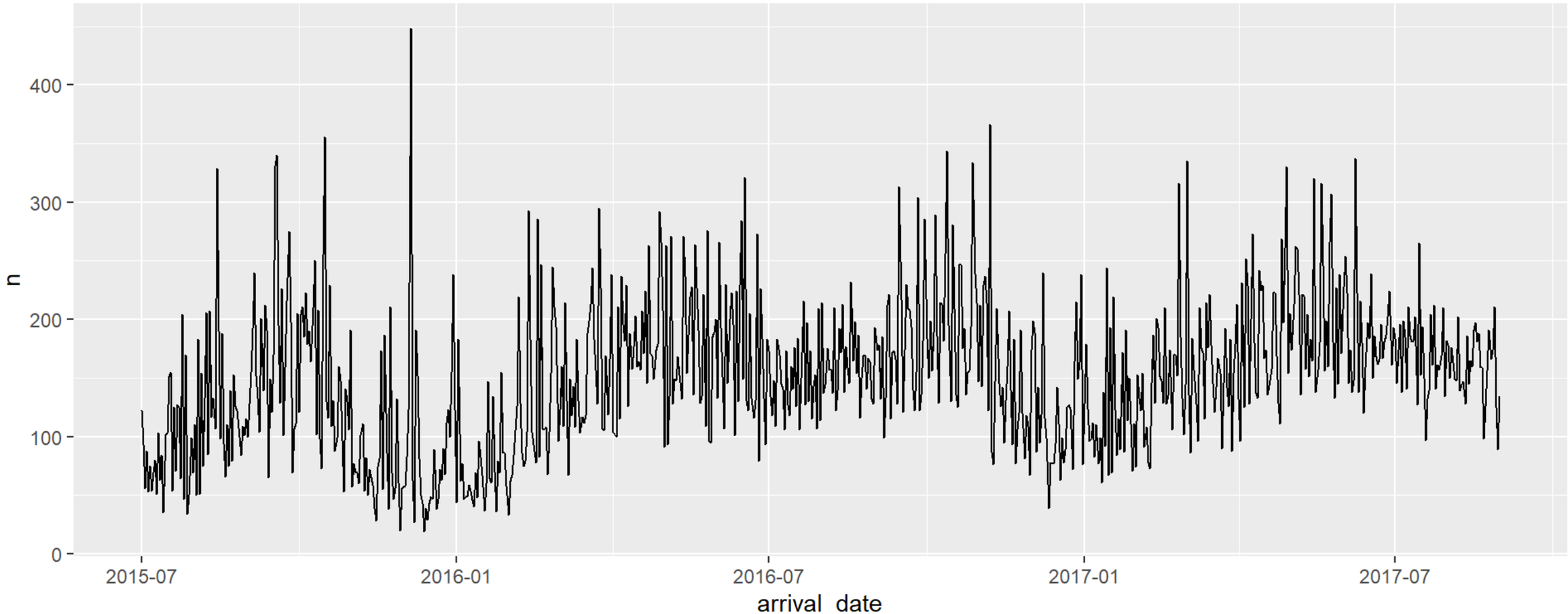
Étape 2 - Compter les réservations par dates

```
hotels %>%
  mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_
  count(arrival_date)
```

```
# A tibble: 793 × 2
   arrival_date     n
   <date>       <int>
 1 2015-07-01     122
 2 2015-07-02      93
 3 2015-07-03      56
 4 2015-07-04      88
 5 2015-07-05      53
 6 2015-07-06      75
 7 2015-07-07      54
 8 2015-07-08      69
 9 2015-07-09      80
10 2015-07-10      51
# ℹ 783 more rows
```

Étape 3a - Visualiser

```
1  hotels %>%
2    mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_
3    count(arrival_date) %>%
4    ggplot(aes(x = arrival_date, y = n, group = 1)) +
5    geom_line()
```



Étape 3b - Visualiser avec une courbe

```
1  hotels %>%
2    mutate(arrival_date = ymd(glue("{arrival_date_year} {arrival_date_month} {arrival_date_
3    count(arrival_date) %>%
4    ggplot(aes(x = arrival_date, y = n, group = 1)) +
5    geom_smooth() #<<
```