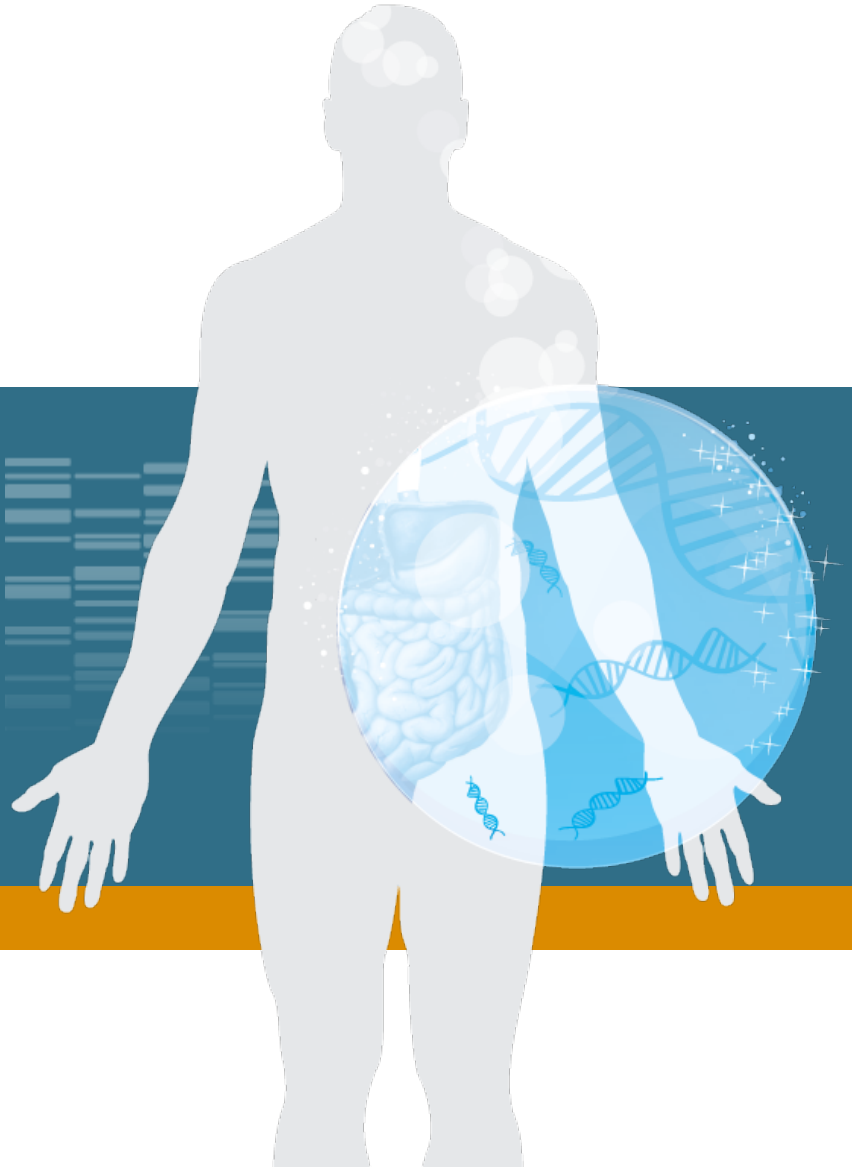
université PARIS DIDEROT PARIS 7



Introduction au tidyverse

Module R – Session 2

Diplôme Universitaire en Biologie intégrative - 2021

magali.berland@inrae.fr

metagenopolis mgps.eu

INRAE

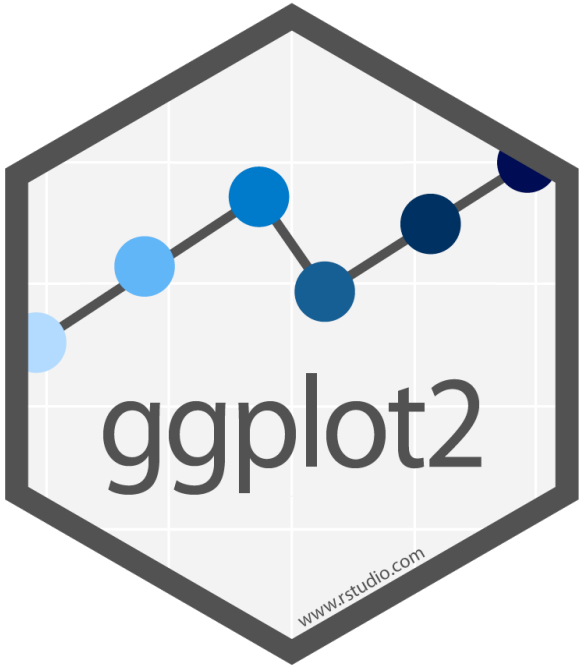
R packages for data science

The tidyverse is an opinionated **collection of R packages** designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

Install the complete tidyverse with:

```
install.packages("tidyverse")
```

Visualisation graphique avec ggplot



Le package ggplot

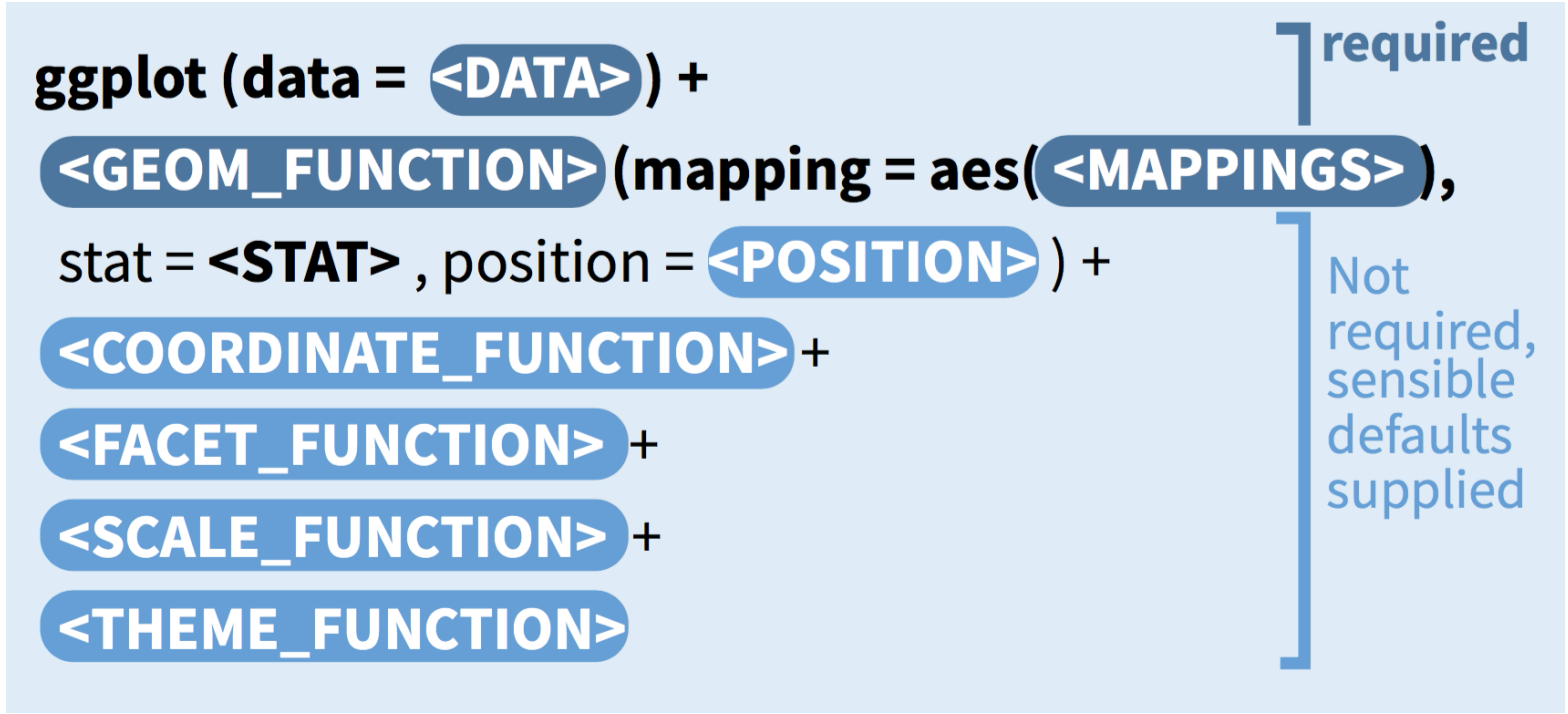
Pour commencer

```
library(ggplot2) # ou library(tidyverse)
```

Chaque objet ggplot2 a trois composantes clés:

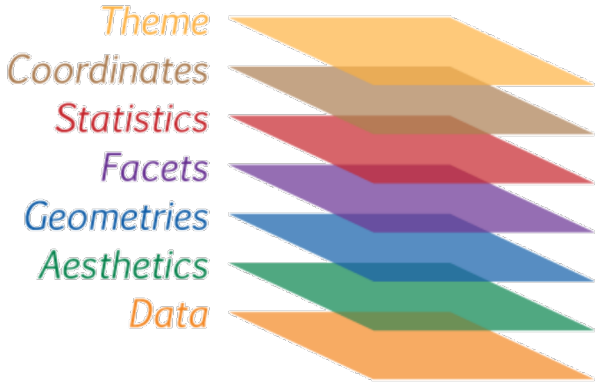
- Les données sous forme d'un **data frame** (**data**)
- Un ensemble de '**aesthestic mapping**' entre les variables du data frame et leurs propriétés visuelles (couleur, taille, etc.)
- Au moins un calque décrivant comment rendre chaque observation; généralement créé avec la fonction **geom**.

Les calques d'un graphe : 'Layered Grammar of Graphics'



Les calques de ggplot

Les calques de ggplot2



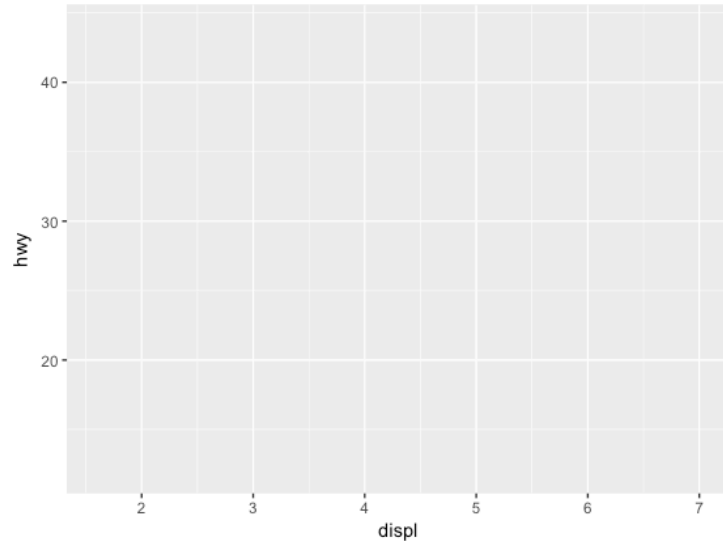
- **Data** : Le jeu de données utilisé (data frame)

```
ggplot(data = mpg)
```

Les calques de ggplot2

- **Data** : Le jeu de données utilisé (data frame)
- **Aesthetics mapping** : Les variables à représenter et leurs propriétés graphiques

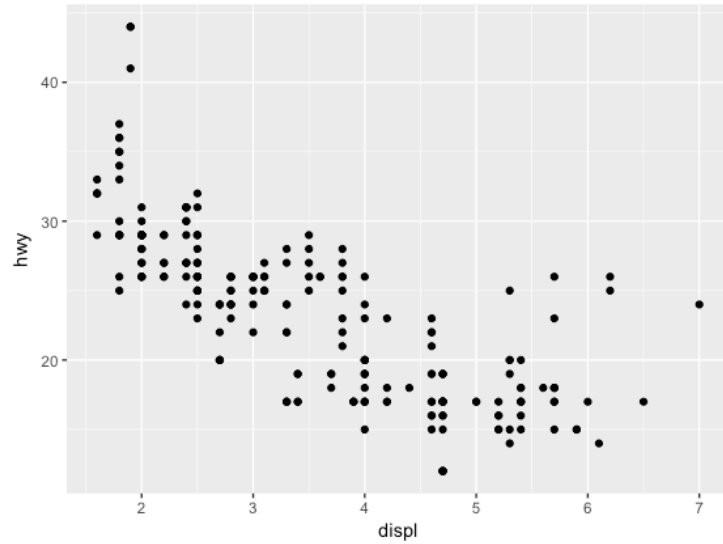
```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy))
```



Les calques de ggplot2

- **Data** : Le jeu de données utilisé (data frame)
- **Aesthetics mapping** : Les variables à représenter et leurs propriétés graphiques
- **Geometries** : Objet géométrique utilisé pour représenter les données

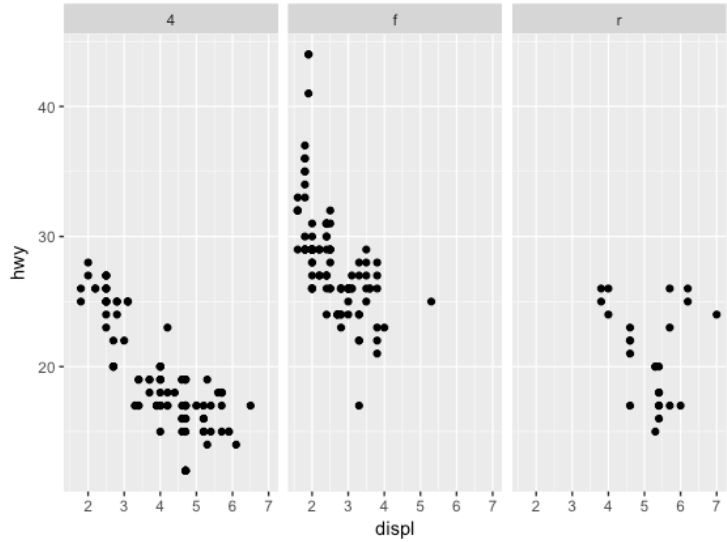
```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy)) +
       geom_point()
```



Les calques de ggplot2

- **Data** : Le jeu de données utilisé (data frame)
- **Aesthetics mapping** : Les variables à représenter et leurs propriétés graphiques
- **Geometries** : Objet géométrique utilisé pour représenter les données
- **Facets** : Tableau (lignes et colonnes) de graphes

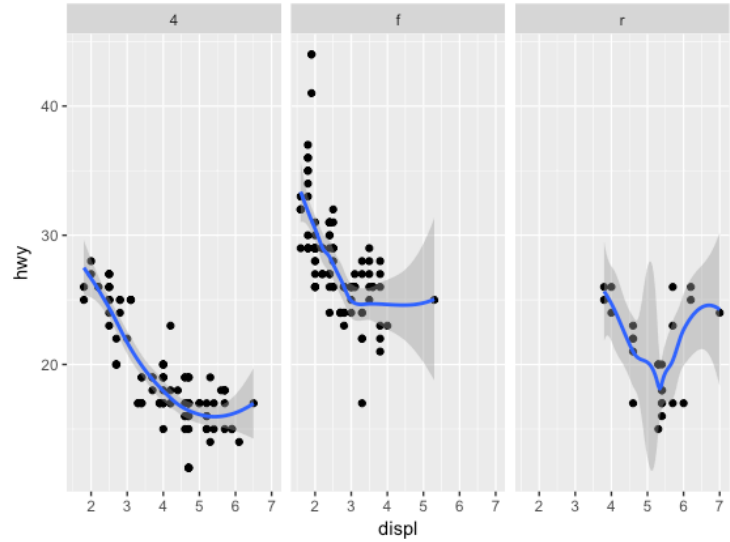
```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy)) +
       geom_point() +
       facet_wrap(~ class)
```



Les calques de ggplot2

- **Data** : Le jeu de données utilisé (data frame)
- **Aesthetics mapping** : Les variables à représenter et leurs propriétés graphiques
- **Geometries** : Objet géométrique utilisé pour représenter les données
- **Facets** : Tableau (lignes et colonnes) de graphes
- **Statistics** : Modèles ou transformations statistiques des données

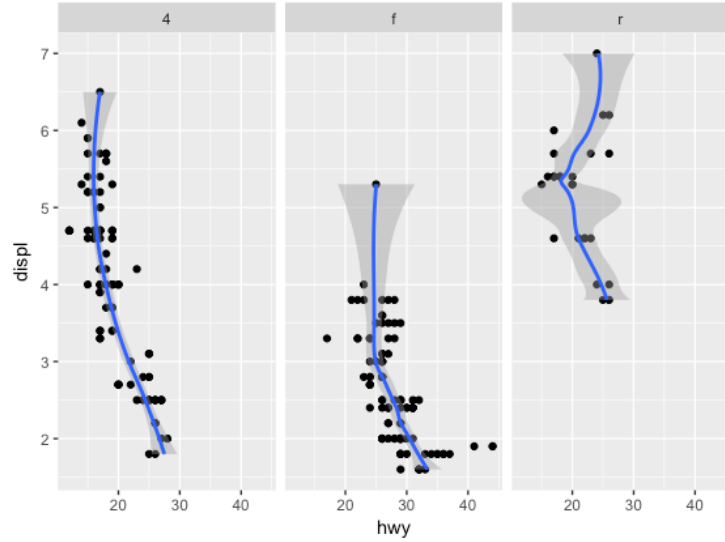
```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy)) +
       geom_point() +
       facet_wrap(~ class) +
       stat_smooth()
```



Les calques de ggplot2

- **Data** : Le jeu de données utilisé (data frame)
- **Aesthetics mapping** : Les variables à représenter et leurs propriétés graphiques
- **Geometries** : Objet géométrique utilisé pour représenter les données
- **Facets** : Tableau (lignes et colonnes) de graphes
- **Statistics** : Modèles ou transformations statistiques des données
- **Coordinates** : L'espace de représentation (horizontal, vertical, échelle log)

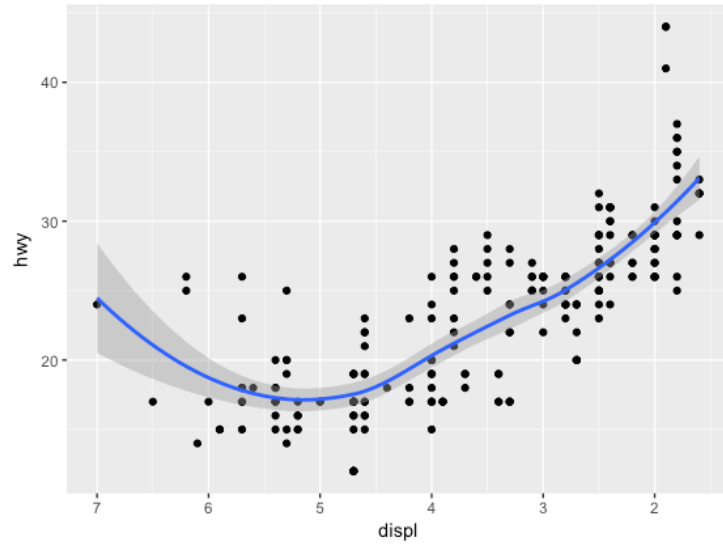
```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy)) +
       geom_point() +
       facet_wrap(~ drv) +
       stat_smooth() +
       coord_flip()
```



Les calques de ggplot2

- **Data** : Le jeu de données utilisé (data frame)
- **Aesthetics mapping** : Les variables à représenter et leurs propriétés graphiques
- **Geometries** : Objet géométrique utilisé pour représenter les données
- **Facets** : Tableau (lignes et colonnes) de graphes
- **Statistics** : Modèles ou transformations statistiques des données
- **Coordinates** : L'espace de représentation (horizontal, vertical, échelle log)
- **Scales** : L'échelle des axes (linéaire, logarithmique, à l'envers), les couleurs de remplissage

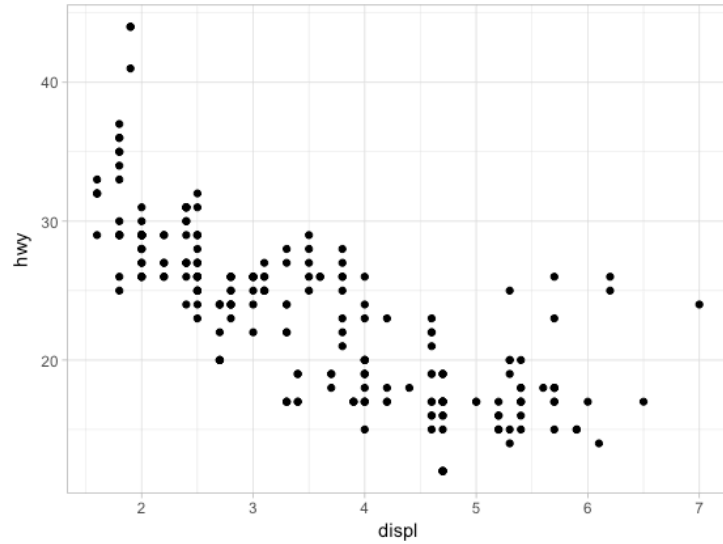
```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy)) +
       geom_point() +
       stat_smooth() +
       scale_x_reverse()
```



Les calques de ggplot2

- **Data** : Le jeu de données utilisé (data frame)
- **Aesthetics mapping** : Les variables à représenter et leurs propriétés graphiques
- **Geometries** : Objet géométrique utilisé pour représenter les données
- **Facets** : Tableau (lignes et colonnes) de graphes
- **Statistics** : Modèles ou transformations statistiques des données
- **Coordinates** : L'espace de représentation (horizontal, vertical, échelle log)
- **Scales** : L'échelle des axes (linéaire, logarithmique, à l'envers), les couleurs de remplissage
- **Theme** : Description de l'arrière plan

```
ggplot(data = mpg,
       mapping = aes(x = displ, y = hwy)) +
       geom_point() +
       theme_light()
```



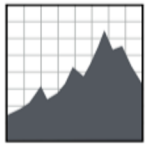
Quelle représentation utiliser ?

- Variables continues (quantitatives)
- Variables discrètes (qualitatives)
  - Ordinales
  - Nominales

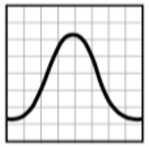
Une variable continue

Continue

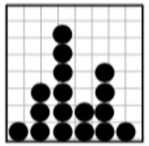
a <- ggplot(mpg, aes(hwy))



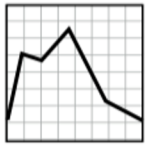
a + geom_area(stat = "bin")
x, y, alpha, color, fill, linetype, size
b + geom_area(aes(y = ..density..), stat = "bin")



a + geom_density(kernel = "gaussian"**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_density(aes(y = ..county..))

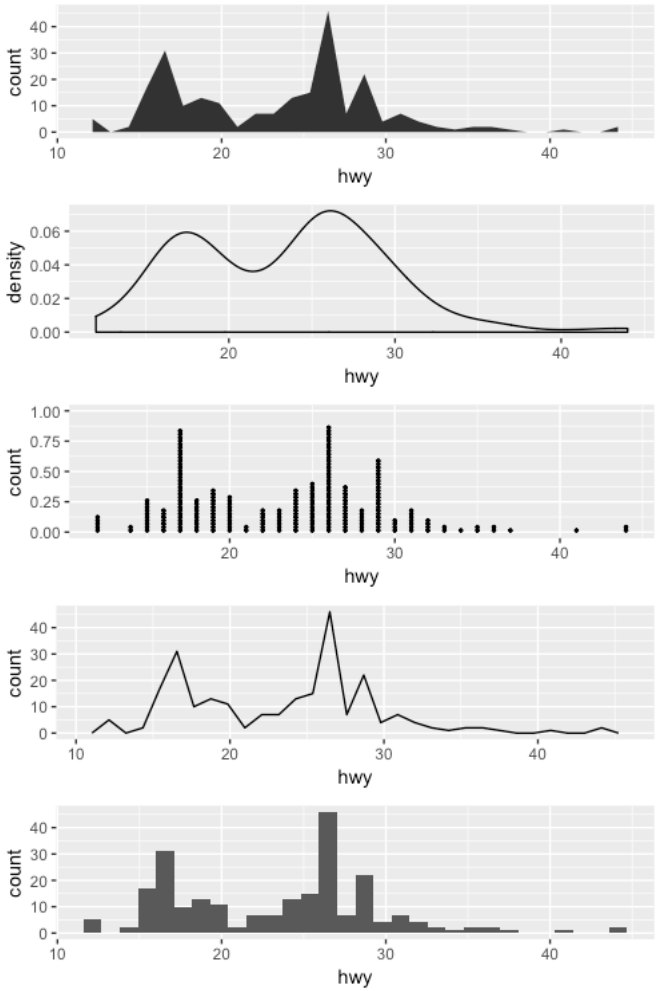


a + geom_dotplot()
x, y, alpha, color, fill



a + geom_freqpoly()
x, y, alpha, color, linetype, size
b + geom_freqpoly(aes(y = ..density..))

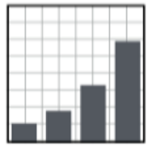
a + geom_histogram(binwidth = 5**)**
x, y, alpha, color, fill, linetype, size, weight
b + geom_histogram(aes(y = ..density..))



Une variable discrète

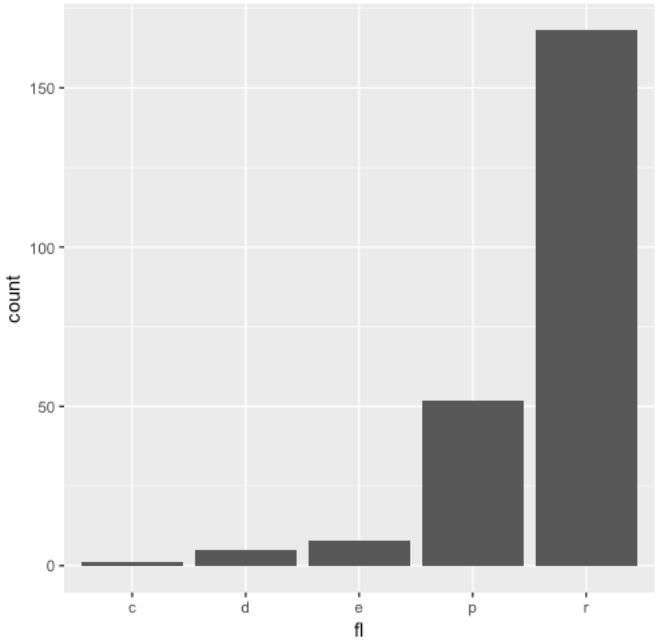
Discrète

```
b <- ggplot(mpg, aes(fl))
```



b + geom_bar()

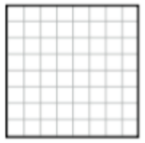
x, alpha, color, fill, linetype, size, weight



Deux variables continues

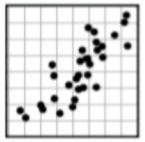
X Continue,Y Continue

f <- ggplot(mpg, aes(cty, hwy))



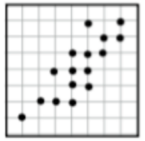
f + geom_blank()

(Utile pour étendre les limites)



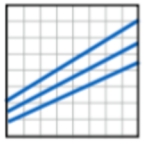
f + geom_jitter()

x, y, alpha, color, fill, shape, size



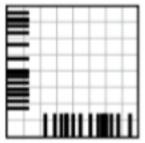
f + geom_point()

x, y, alpha, color, fill, shape, size



f + geom_quantile()

x, y, alpha, color, linetype, size, weight

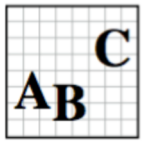


f + geom_rug(sides = "bl"**)**

alpha, color, linetype, size

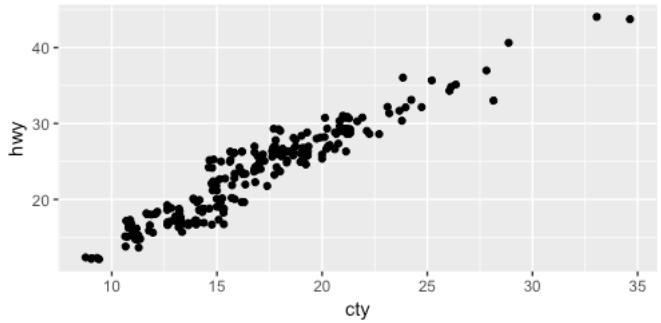
f + geom_smooth(model = lm**)**

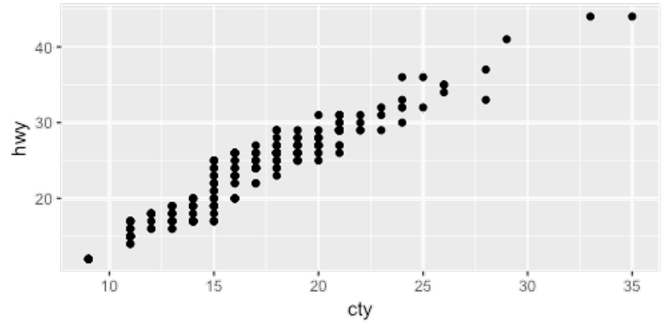
x, y, alpha, color, fill, linetype, size, weight

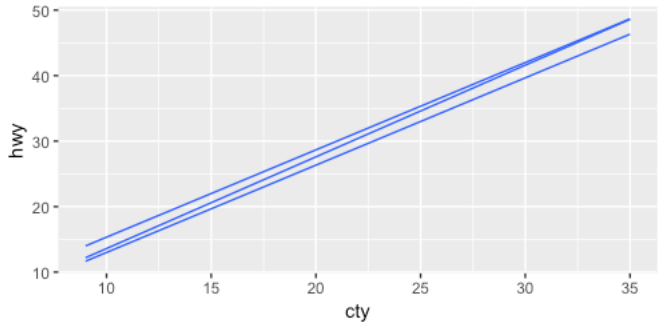


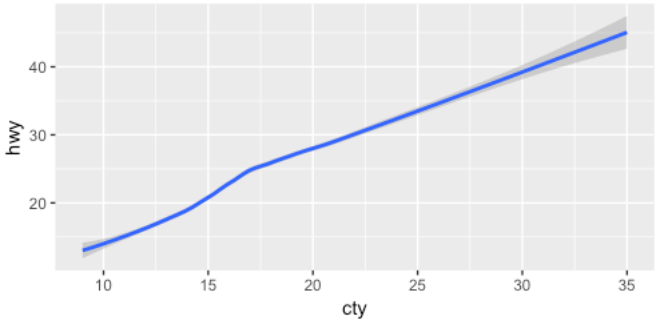
f + geom_text(aes(label = cty)**)**

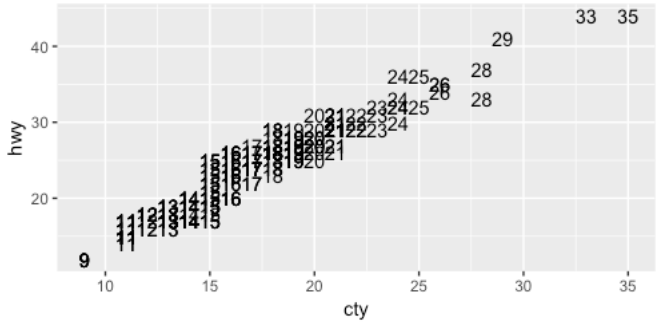
x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust







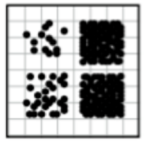




Deux variables discrètes

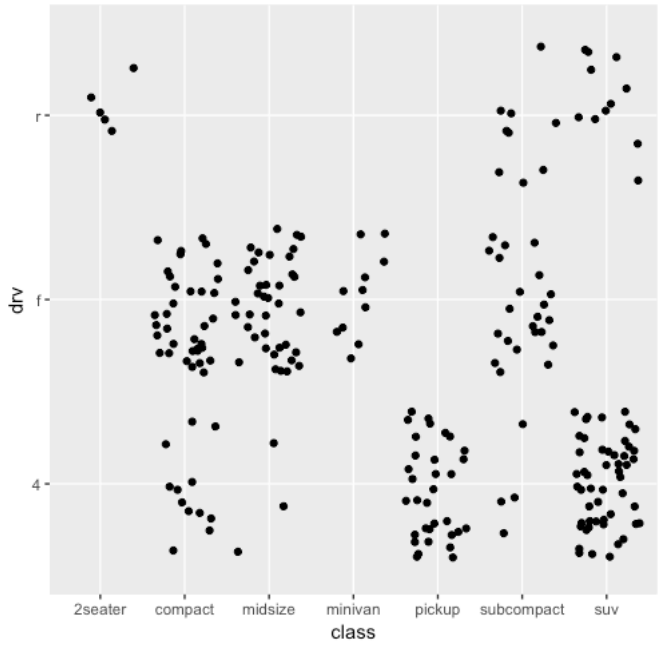
X Discrète, Y Discrète

```
h <- ggplot(mpg, aes(class, drv))
```



h + geom_jitter()

x, y, alpha, color, fill, shape, size



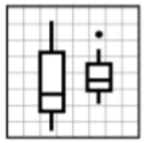
Deux variables, une continue et une discrète

X Discrète, Y Continue

g <- ggplot(mpg, aes(class, hwy))

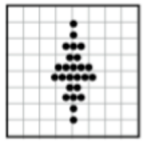
g + geom_bar(stat = "identity")

x, y, alpha, color, fill, linetype, size, weight



g + geom_boxplot()

lower, middle, upper, x, ymax, ymin, alpha, color, fill, linetype, shape, size, weight

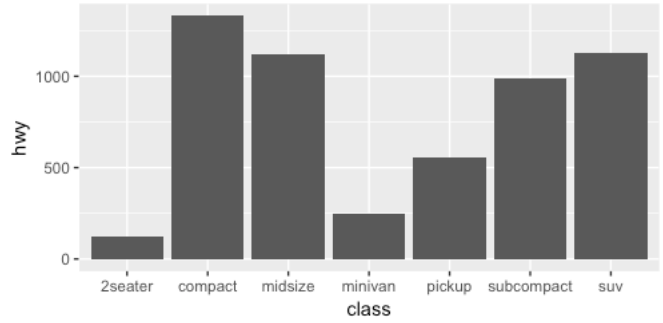


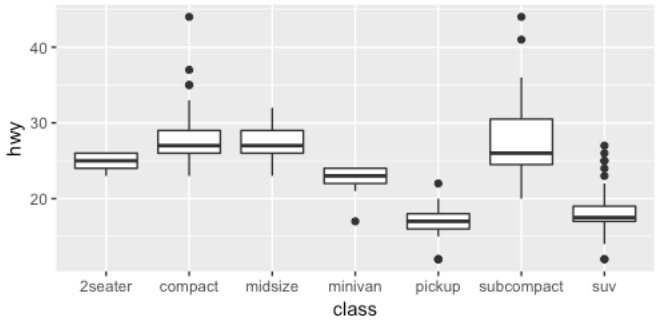
g + geom_dotplot(binaxis = "y", stackdir = "center"**)**

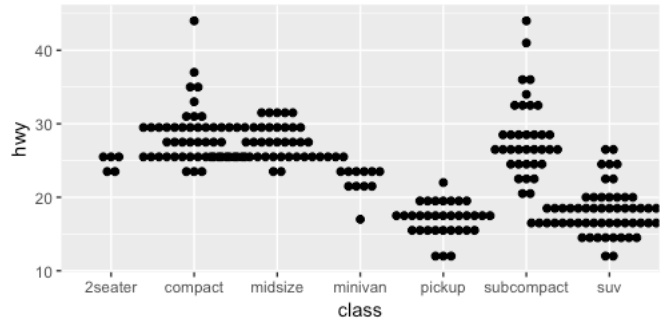
x, y, alpha, color, fill

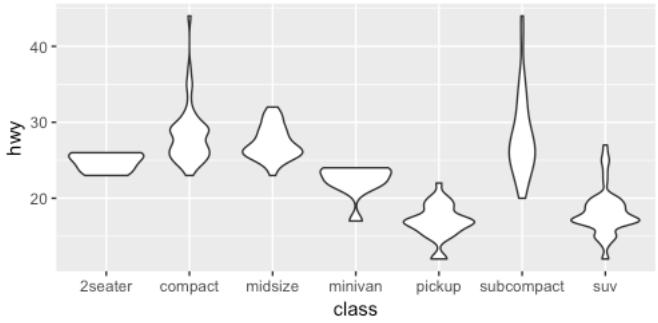
g + geom_violin(scale = "area"**)**

x, y, alpha, color, fill, linetype, size, weight



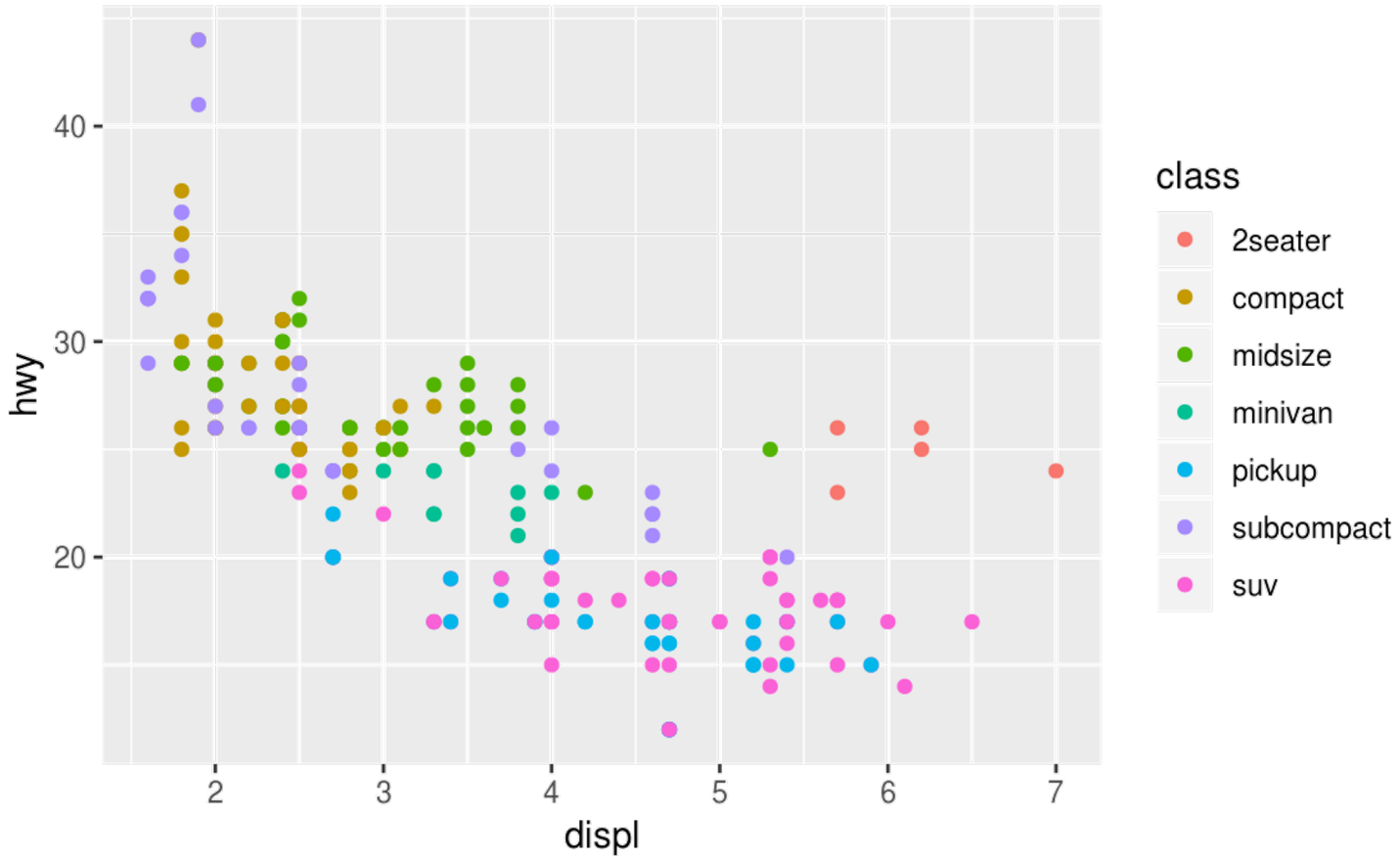






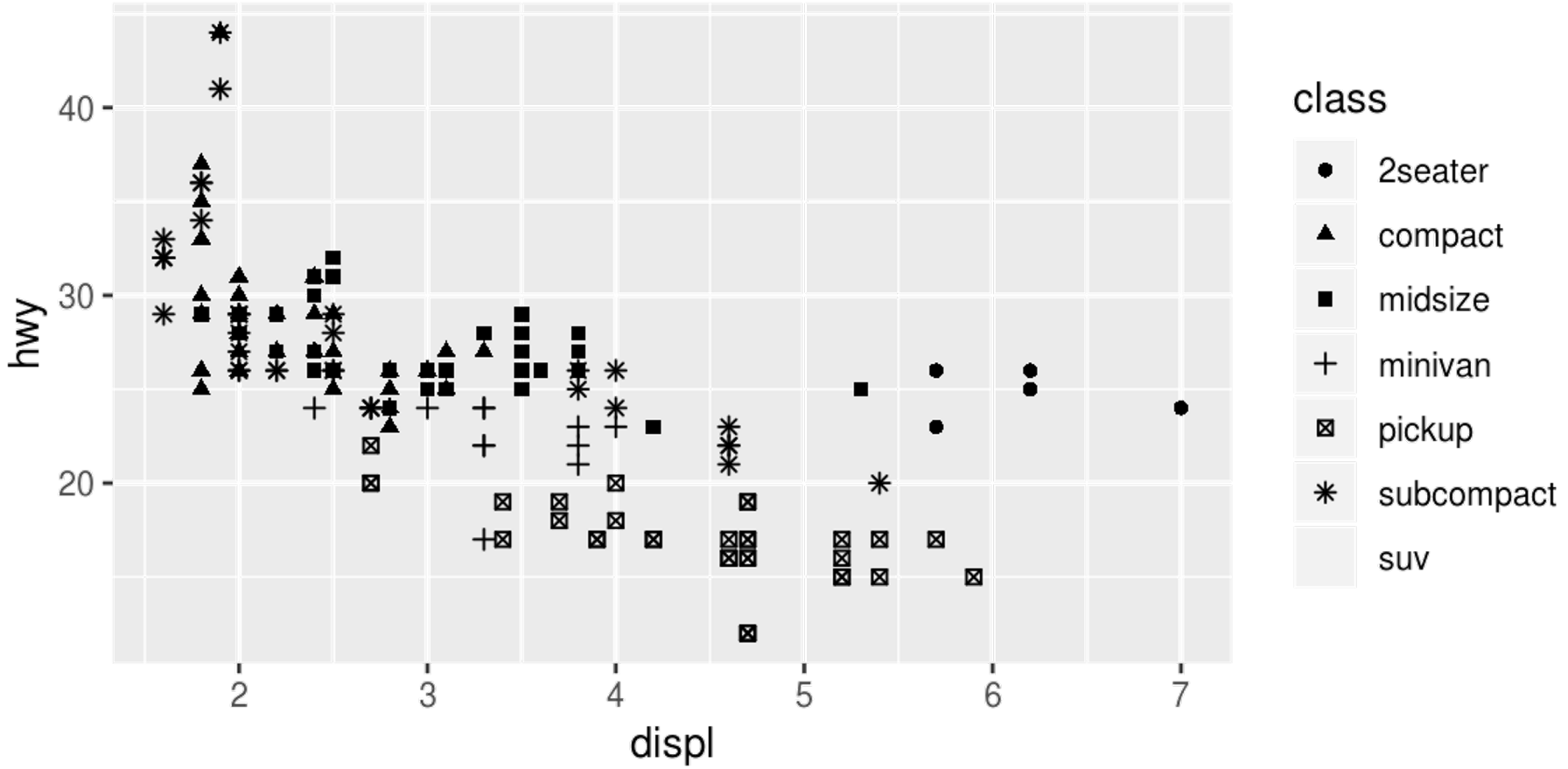
Trois variables, deux continues et une discrète

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, color = class))
```



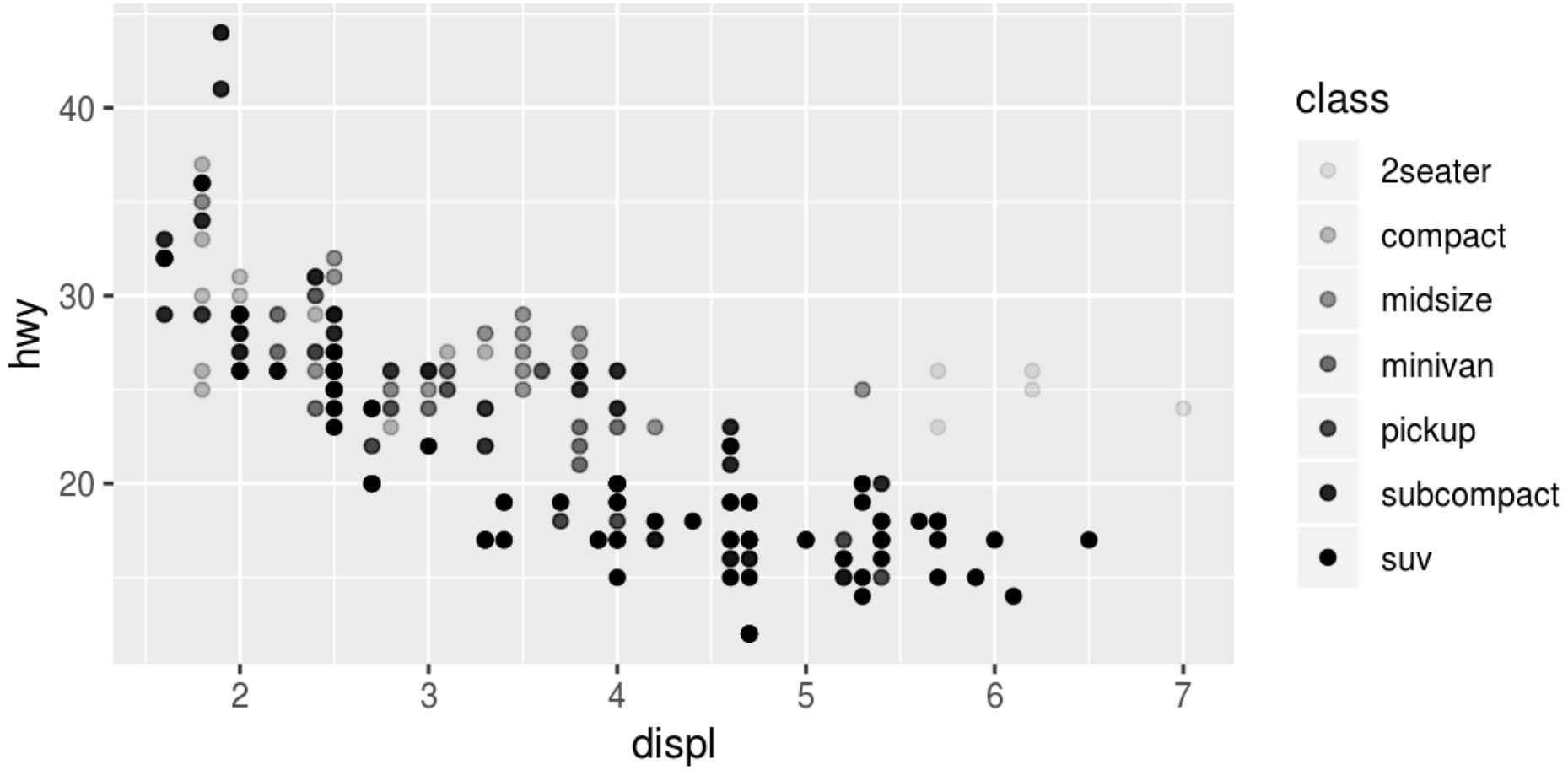
Trois variables, deux continues et une discrète

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, shape = class))
```



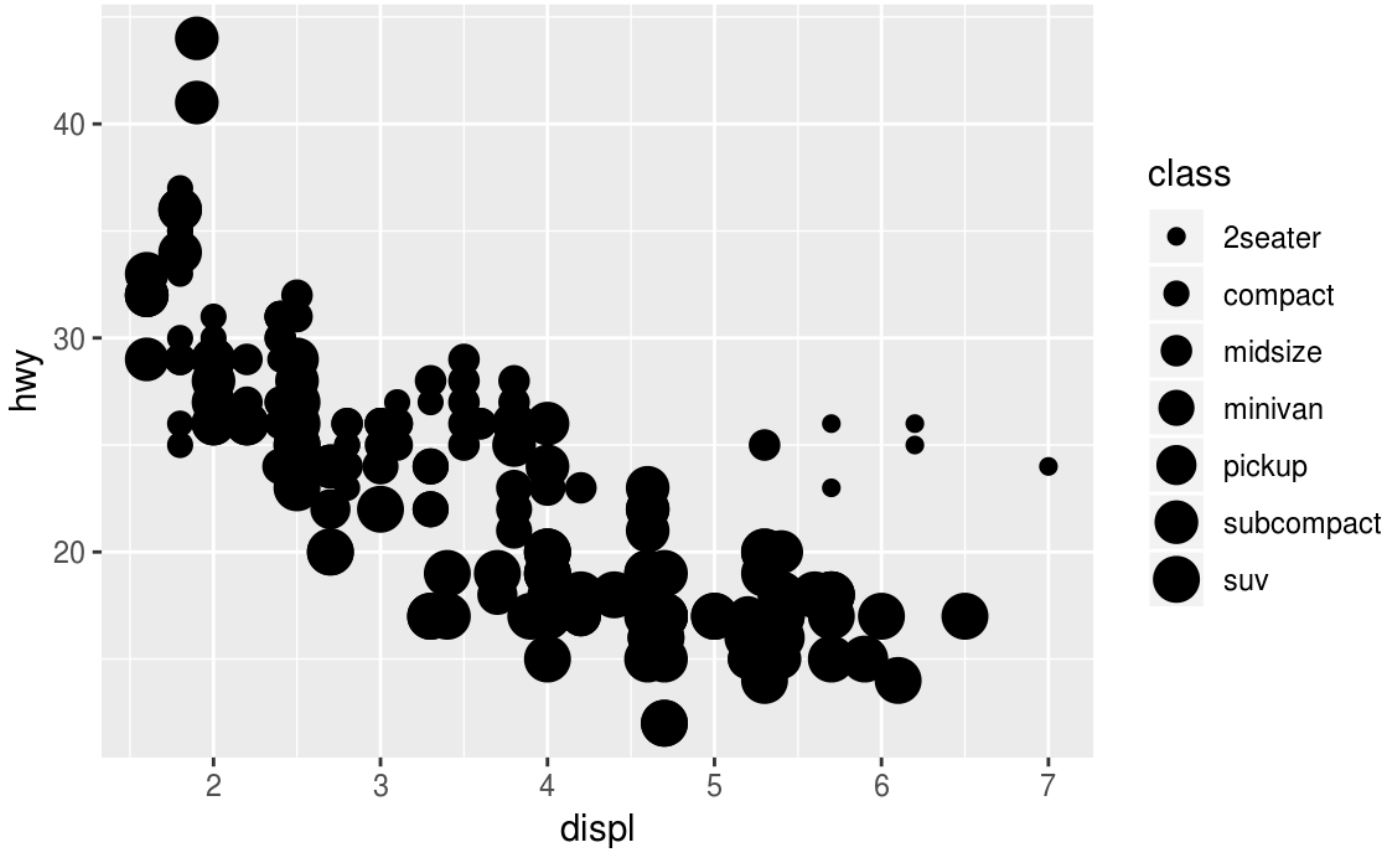
Trois variables continues

```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, alpha = class))
```



Trois variables continues

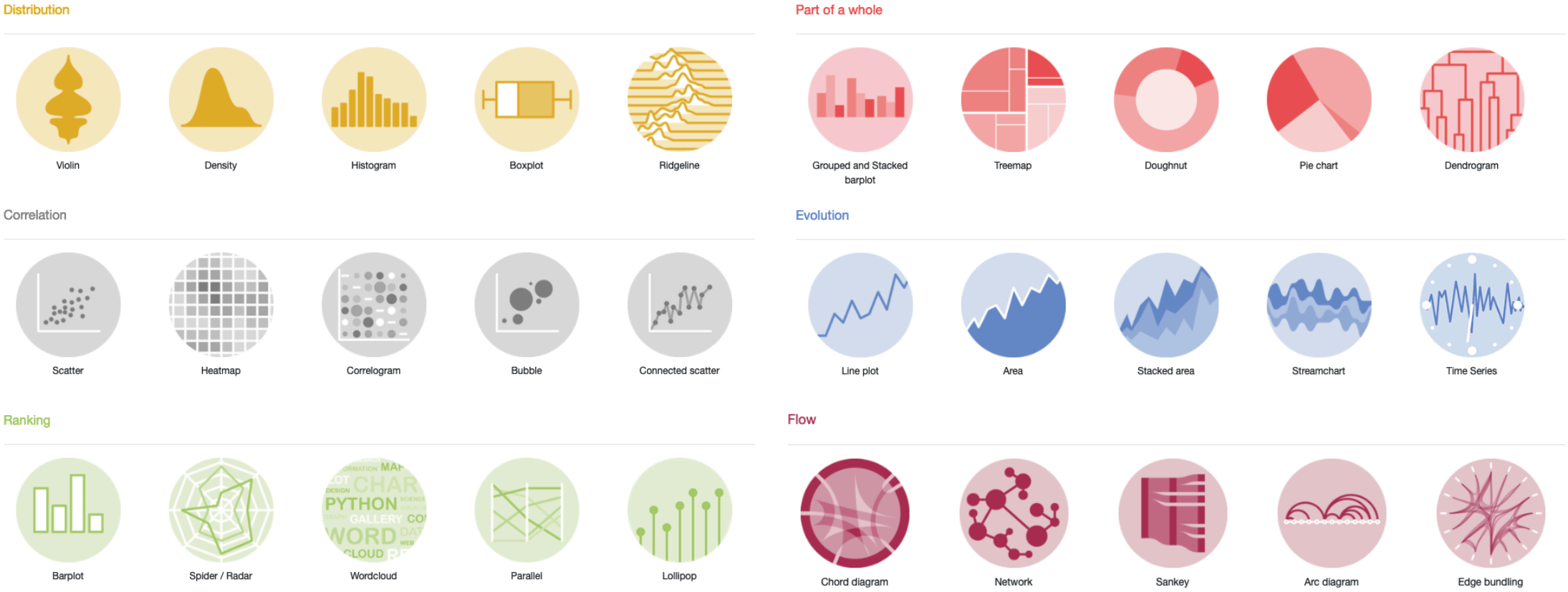
```
ggplot(data = mpg) +
  geom_point(mapping = aes(x = displ, y = hwy, size = class))
#> Warning: Using size for a discrete variable is not advised.
```



Et pour plus de trois variables ?

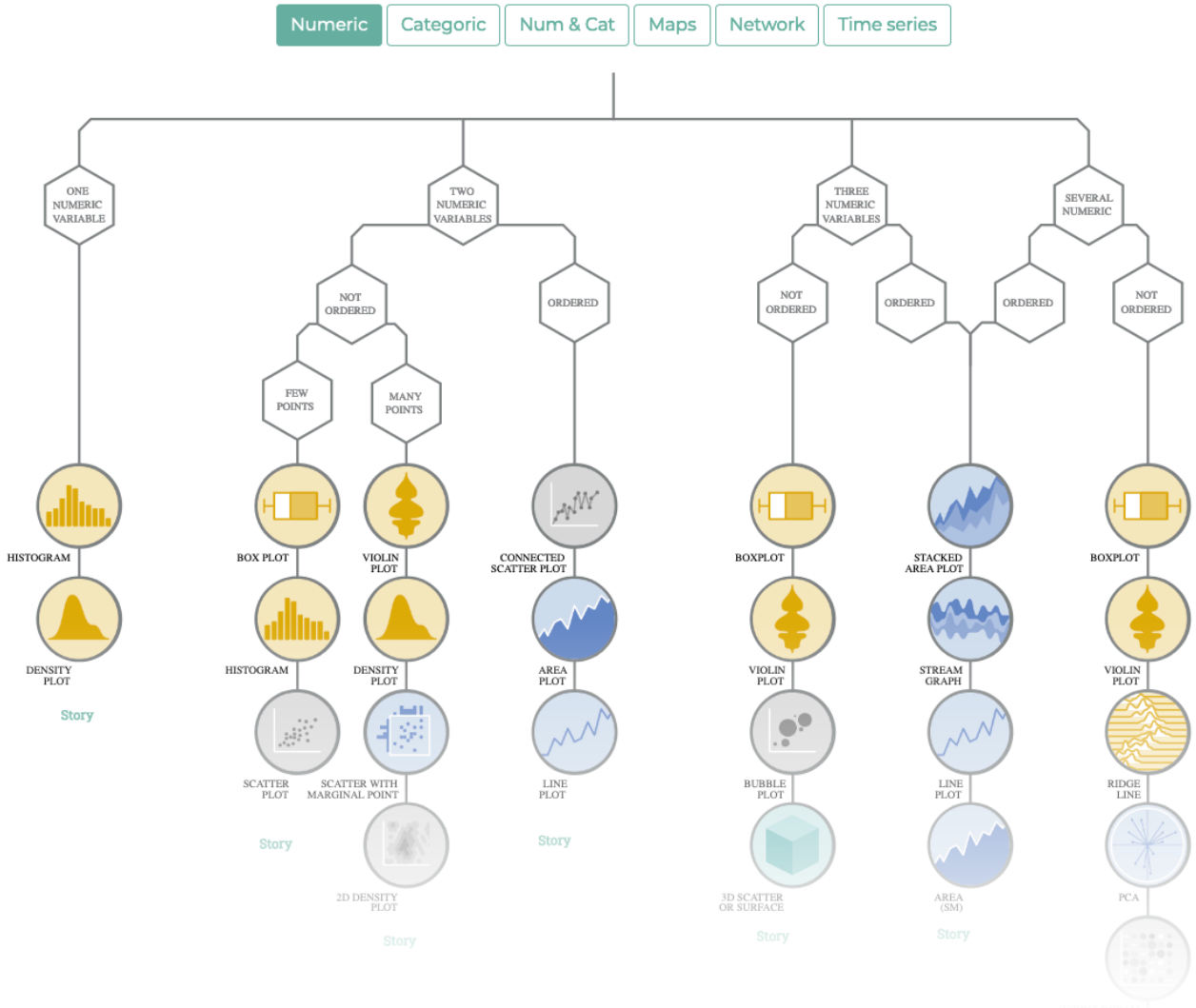
Ressources

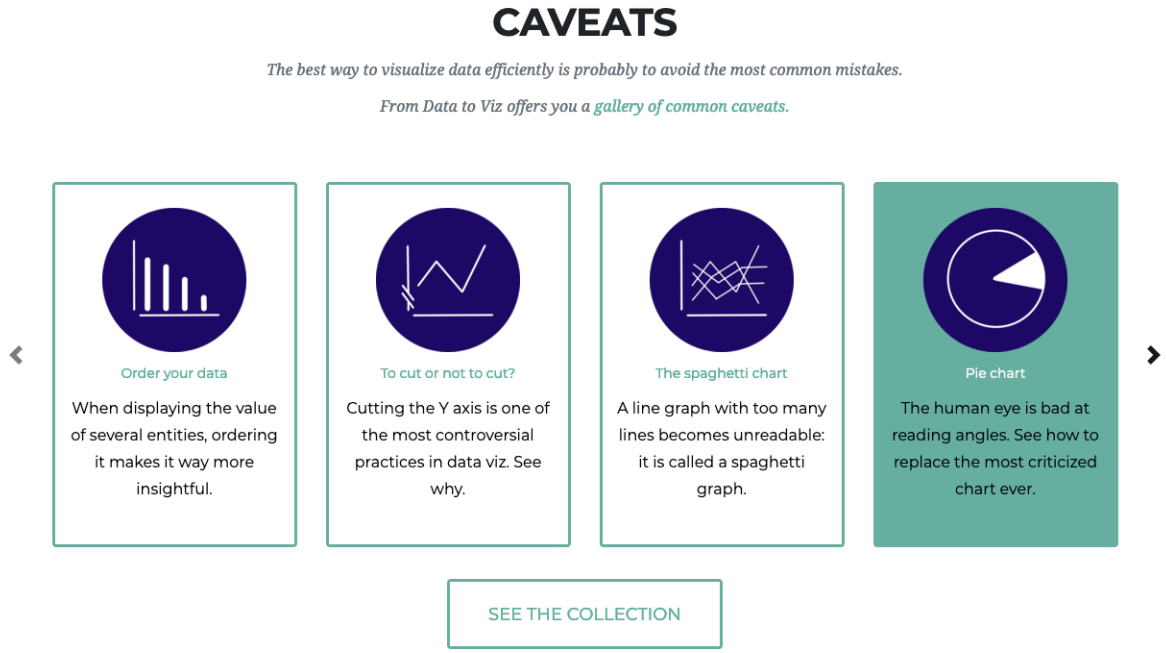
- https://www.r-graph-gallery.com/



Ressources

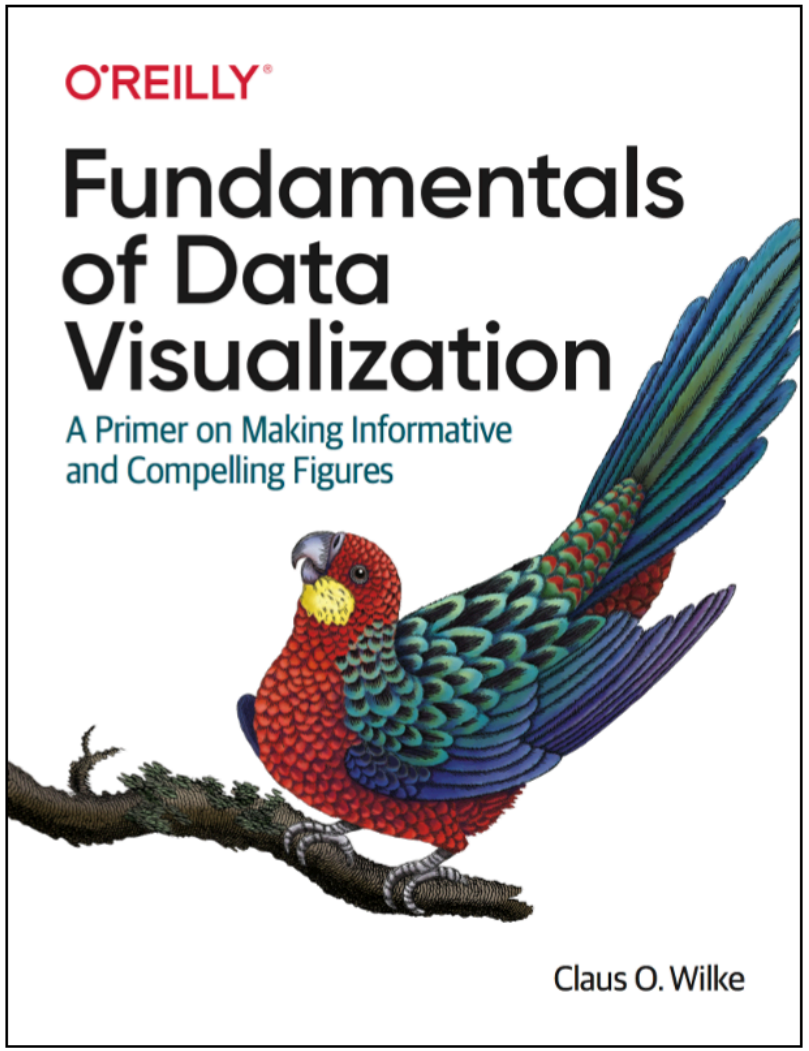
- https://www.data-to-viz.com/





Ressources

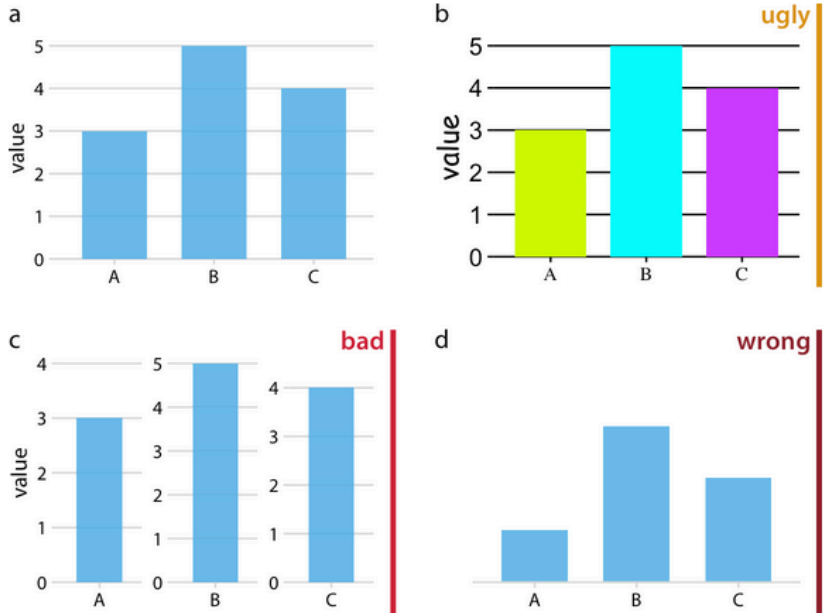
- https://clauswilke.com/dataviz/



Ugly, bad, and wrong figures

Throughout this book, I frequently show different versions of the same figures, some as examples of how to make a good visualization and some as examples of how not to. To provide a simple visual guideline of which examples should be emulated and which should be avoided, I am clearly labeling problematic figures as "ugly", "bad", or "wrong" (Figure 1.1):

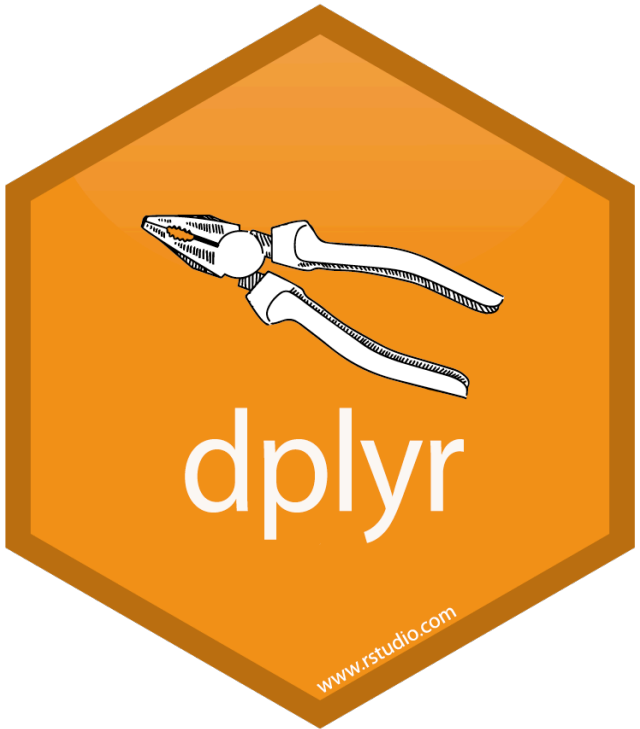
- **ugly**—A figure that has aesthetic problems but otherwise is clear and informative.
- **bad**—A figure that has problems related to perception; it may be unclear, confusing, overly complicated, or deceiving.
- **wrong**—A figure that has problems related to mathematics; it is objectively incorrect.



TD : 1ère partie

***

Remanier les données



Tibbles

- Les tibbles sont des data frame, mais légèrement modifiées pour mieux fonctionner dans le tidyverse.
- Principales différences :
  - Affichage concis dans la console de R
  - Pas de rownames
  - Information sur le type de chaque variable
    - Int [entiers], dbl [nombres réels], chr [chaine de caractère], fctr [facteur],
    - dttm [date-heure], date
    - lgl [logique]

Tibbles

```
swiss
```

```
##              Fertility Agriculture Examination Education Catholic
## Courtelary        80.2        17.0          15        12     9.96
## Delemont          83.1        45.1           6         9    84.84
## Franches-Mnt      92.5        39.7           5         5    93.40
## Moutier           85.8        36.5          12         7    33.77
## Neuveville        76.9        43.5          17        15     5.16
## Porrentruy        76.1        35.3           9         7    90.57
## Broye             83.8        70.2          16         7    92.85
## Glane             92.4        67.8          14         8    97.16
## Gruyere           82.4        53.3          12         7    97.67
## Sarine            82.9        45.2          16        13    91.38
## Veveyse           87.1        64.5          14         6    98.61
## Aigle             64.1        62.0          21        12     8.52
## Aubonne           66.9        67.5          14         7     2.27
## Avenches          68.9        60.7          19        12     4.43
## Cossonay          61.7        69.3          22         5     2.82
## Echallens         68.3        72.6          18         2    24.20
## Grandson          71.7        34.0          17         8     3.30
## Lausanne          55.7        19.4          26        28    12.11
## La Vallee         54.3        15.2          31        20     2.15
```

Tibbles

```
library(tibble)
as_tibble(swiss, rownames = "Province")
```

```
## # A tibble: 47 x 7
##    Province     Fertility Agriculture Examination Education Catholic
##    <chr>            <dbl>       <dbl>       <int>     <int>    <dbl>
##  1 Courtelary        80.2        17.0          15        12     9.96
##  2 Delemont          83.1        45.1           6         9    84.8
##  3 Franches-Mnt      92.5        39.7           5         5    93.4
##  4 Moutier           85.8        36.5          12         7    33.8
##  5 Neuveville        76.9        43.5          17        15     5.16
##  6 Porrentruy        76.1        35.3           9         7    90.6
##  7 Broye             83.8        70.2          16         7    92.8
##  8 Glane             92.4        67.8          14         8    97.2
##  9 Gruyere           82.4        53.3          12         7    97.7
## 10 Sarine            82.9        45.2          16        13    91.4
## # ... with 37 more rows, and 1 more variable: Infant.Mortality <dbl>
```

Les bases de dplyr : six verbes pour remanier les données

- Les cinq fonctions clés de dplyr :
  - Choisissez les observations par leurs valeurs (**filter()**)
  - Réorganisez les lignes (**arrange()**)
  - Choisissez les variables par leur nom (**select()**)
  - Créez de nouvelles variables avec des fonctions de variables existantes (**mutate()**)
  - Réduisez de nombreuses valeurs en un seul résumé (**summarise()**)
- **group_by()** : modifie la portée de chaque fonction pour qu'elle fonctionne groupe par groupe

Les bases de dplyr : six verbes pour remanier les données

- Tous les verbes fonctionnent de la même manière:
  - Le premier argument est un data frame ou un tibble.
  - Les arguments suivants décrivent quoi faire avec le data frame, en utilisant les noms de variables (sans guillemets).
  - Le résultat est un nouveau data frame.
- Ensemble, ces propriétés facilitent l'enchaînement de plusieurs étapes simples pour obtenir un résultat complexe.

Filtrer les lignes avec filter()

```
filter(flights, month == 1, day == 1)
#> # A tibble: 842 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      517            515         2      830
#> 2  2013     1     1      533            529         4      850
#> 3  2013     1     1      542            540         2      923
#> 4  2013     1     1      544            545        -1     1004
#> 5  2013     1     1      554            600        -6      812
#> 6  2013     1     1      554            558        -4      740
#> # ... with 836 more rows, and 12 more variables: sched_arr_time <int>,
#> #   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#> #   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>, hour <dbl>,
#> #   minute <dbl>, time_hour <dttm>
```

Trier les lignes avec arrange()

```
arrange(flights, year, month, day)
#> # A tibble: 336,776 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>
#> 1  2013     1     1      517            515         2      830
#> 2  2013     1     1      533            529         4      850
#> 3  2013     1     1      542            540         2      923
#> 4  2013     1     1      544            545        -1     1004
#> 5  2013     1     1      554            600        -6      812
#> 6  2013     1     1      554            558        -4      740
#> # ... with 3.368e+05 more rows, and 12 more variables:
#> #   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,
#> #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,
#> #   distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

`arrange(flights, -year)` : tri dans l'ordre décroissant

Sélectionner les colonnes avec select()

```
select(flights, year, month, day)
```

```
select(flights, year:day)
```

```
select(flights, -(year:day))
```

Aides à la sélection :

starts_with()	`starts_with("abc")`
ends_with()	`ends_with("xyz")`
contains()	`contains("ijk")`
matches()	`matches("(.)\\1")`
num_range()	`num_range("x", 1:3)`

Ajouter de nouvelles variables avec mutate()

```
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  speed = distance / air_time * 60
)
```

mutate () ajoute de nouvelles variables et préserve celles existantes.

Ajouter de nouvelles variables avec mutate() et transmute()

```
mutate(flights_sml,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

mutate () ajoute de nouvelles variables et préserve celles existantes.

```
transmute(flights,
  gain = dep_delay - arr_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

transmute () ajoute de nouvelles variables et supprime celles existantes.

Résumer les données avec summarise()

```
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
#> # A tibble: 1 x 1
#>   delay
#>   <dbl>
#> 1  12.6
```

Résumer les données avec group_by() et summarise()

```
by_day <- group_by(flights, year, month, day)
summarise(by_day, delay = mean(dep_delay, na.rm = TRUE))
#> # A tibble: 365 x 4
#> # Groups:   year, month [?]
#>    year month   day delay
#>   <int> <int> <int> <dbl>
#> 1  2013     1     1 11.5
#> 2  2013     1     2 13.9
#> 3  2013     1     3 11.0
#> 4  2013     1     4  8.95
#> 5  2013     1     5  5.73
#> 6  2013     1     6  7.15
#> # ... with 359 more rows
```

Pour retirer un groupement et retrouver les opérations sur les données non groupées, utiliser ungroup().

Autres verbes utiles…

- **rename**(mpg, hwy = rendement) : renomme les colonnes d'un data frame
- **distinct**(iris) : enlève les lignes dupliquées
- **slice**(iris, 10:15) : sélectionne des lignes par position

Enchainer les opérations

Syntaxe : le pipe « %>% »

❖ Permet de rendre le code plus lisible

- Everyday language

> Tom eats an apple

Subject - Verb - Complement

- Programming language

```
eat(Tom, apple)
```

Verb - Subject - Complement

Syntaxe : le pipe « %>% »

❖ Permet de rendre le code plus lisible

- Everyday language

  > Tom eats an apple

Subject - Verb - Complement

- Programming language

  ```
  eat(Tom, apple)
  ```

Verb - Subject - Complement

❖ %>% passe l'objet situé à gauche comme premier argument de la fonction située à droite

```
x %>% f(y) ⇔ f(x, y)
y %>% f(x, z) ⇔ f(y, x, z)
```

Enchainer les opérations avec l'opérateur « pipe » %>%

```
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
```

Enchainer les opérations avec l'opérateur « pipe » %>%

```r
by_dest <- group_by(flights, dest)
delay <- summarise(by_dest,
  count = n(),
  dist = mean(distance, na.rm = TRUE),
  delay = mean(arr_delay, na.rm = TRUE)
)
delay <- filter(delay, count > 20, dest != "HNL")
```

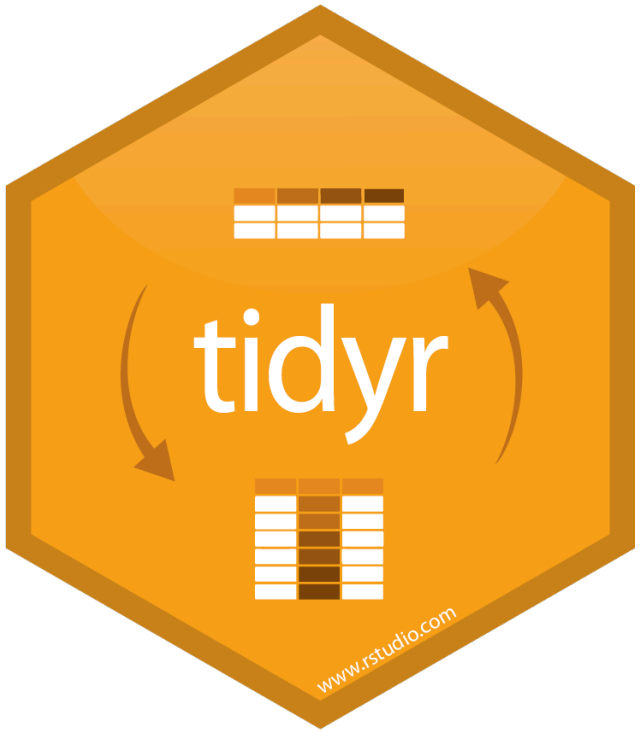
Ensuite

```r
delays <- flights %>%
  group_by(dest) %>%
  summarise(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest != "HNL")
```

TD : 2ème partie

***

Réordonner les données



Jeu de données ordonné (tidy data)

> “Happy families are all alike; every unhappy family is unhappy in its own way.” — Leo Tolstoy

> “Tidy datasets are all alike, but every messy dataset is messy in its own way.” — Hadley Wickham

Jeu de données ordonné (tidy data)

> "Happy families are all alike; every unhappy family is unhappy in its own way." — Leo Tolstoy

> "Tidy datasets are all alike, but every messy dataset is messy in its own way." — Hadley Wickham

- Un jeu de données est ordonné quand :
  - chaque **variable** se trouve dans **une colonne**
  - chaque **observation** compose **une ligne**
  - les éléments sont contenus dans **le même dataset**

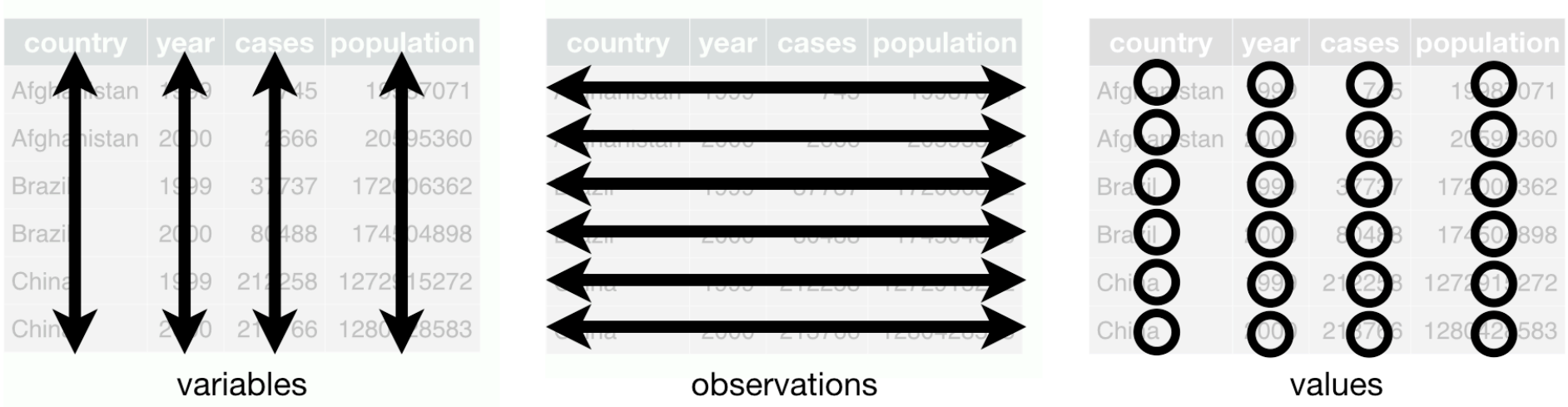
  - Une ligne = « un individu statistique »

  - Permet de faire référence aux variables et aux observations de manière cohérente tout au long de l'analyse

Jeu de données ordonné (tidy data)

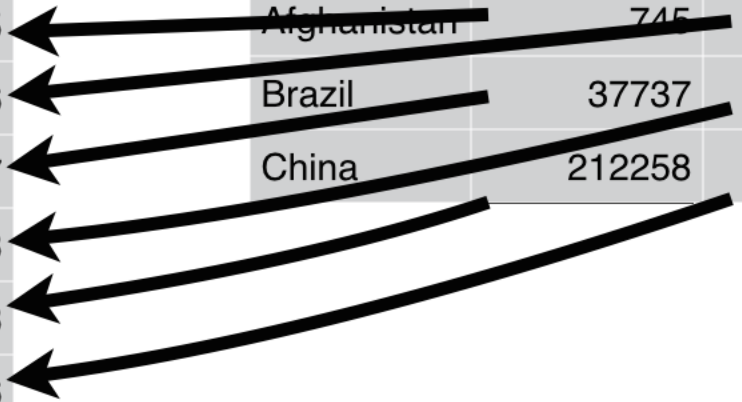
> "Happy families are all alike; every unhappy family is unhappy in its own way." — Leo Tolstoy

> "Tidy datasets are all alike, but every messy dataset is messy in its own way." — Hadley Wickham



Rassembler avec gather() et étendre avec spread()

country	year	cases
Afghanistan	1999	745
Afghanistan	2000	2666
Brazil	1999	37737
Brazil	2000	80488
China	1999	212258
China	2000	213766



country	1999	2000
Afghanistan	745	2666
Brazil	37737	80488
China	212258	213766

Rassembler avec gather() et étendre avec spread()

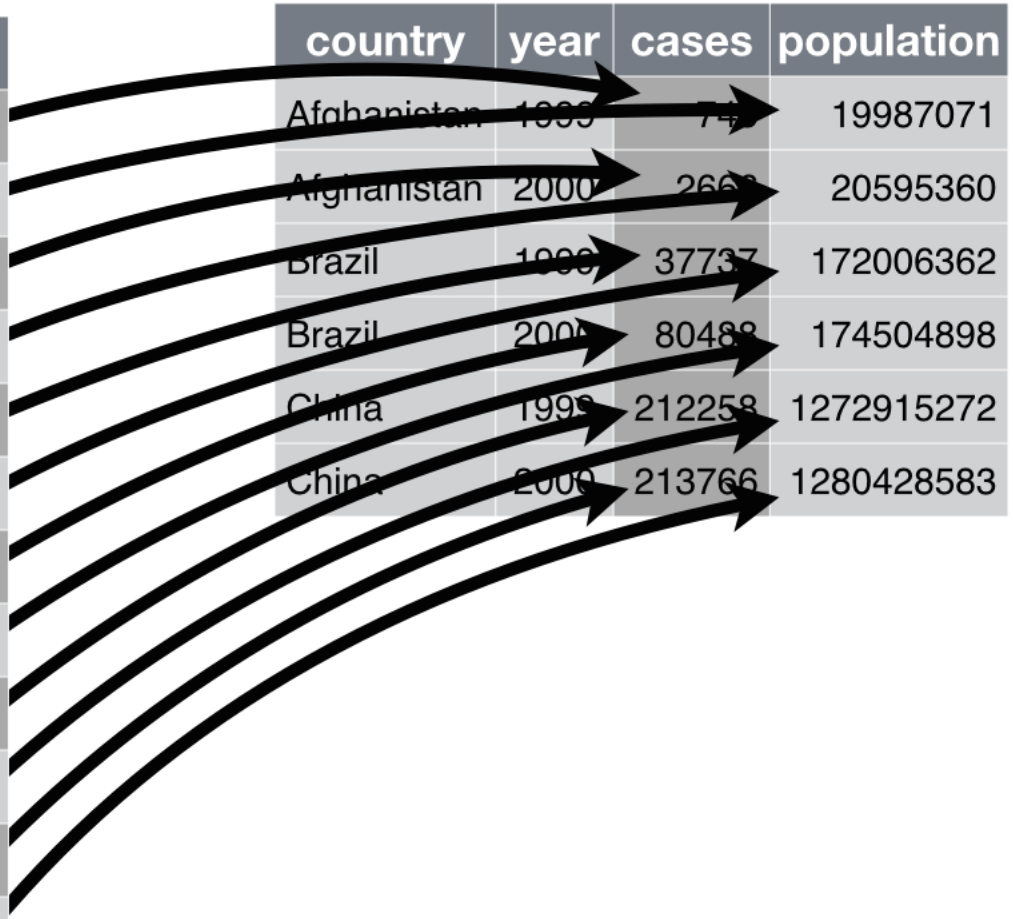
gather : regroupe des colonnes en lignes

```
gather(data, key = "key", value = "value", ...)

data  A data frame.
key, value  Names of new key and value columns,
...   A selection of columns.
      > If empty, all variables are selected.
      > You can supply bare variable names,
      > Select all variables between x and z with x:z,
      > Exclude y with -y
```

Rassembler avec gather() et étendre avec spread()

country	year	key	value
Afghanistan	1999	cases	745
Afghanistan	1999	population	19987071
Afghanistan	2000	cases	2666
Afghanistan	2000	population	20595360
Brazil	1999	cases	37737
Brazil	1999	population	172006362
Brazil	2000	cases	80488
Brazil	2000	population	174504898
China	1999	cases	212258
China	1999	population	1272915272
China	2000	cases	213766
China	2000	population	1280428583



country	year	cases	population
Afghanistan	1999	745	19987071
Afghanistan	2000	2666	20595360
Brazil	1999	37737	172006362
Brazil	2000	80488	174504898
China	1999	212258	1272915272
China	2000	213766	1280428583

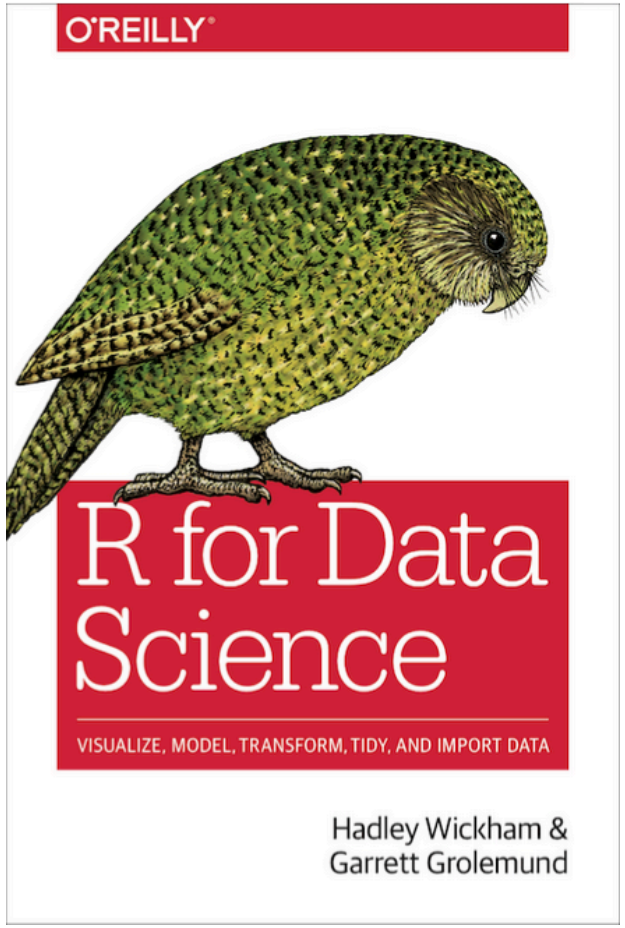
Séparer avec separate() et unir avec unite()

- **separate()** : sépare une colonne en plusieurs colonnes

- **unite()** : réuni plusieurs colonnes en une seule (inverse de separate)

Ressources

- https://r4ds.had.co.nz/



TD – 3ème partie

***