

# Introduction à ggplot2

Café des doctorants

Lorenzo Gaborini

2019-03-26



# Contrôles

Appuyer sur **h** ou **?** pour contrôler la présentation.

# Buts

- R moderne : nouveaux outils
- Introduction au `tidyverse`
- Familiariser avec la librairie `ggplot2`
- Extensions
- `ggplot2` est énorme : **concepts** et **autonomie** !
- Pour informations :
  - Stack Overflow
  - Cheatsheets : <https://www.rstudio.com/resources/cheatsheets/>
  - Documentation : <https://ggplot2.tidyverse.org/reference/>


# Prérequis

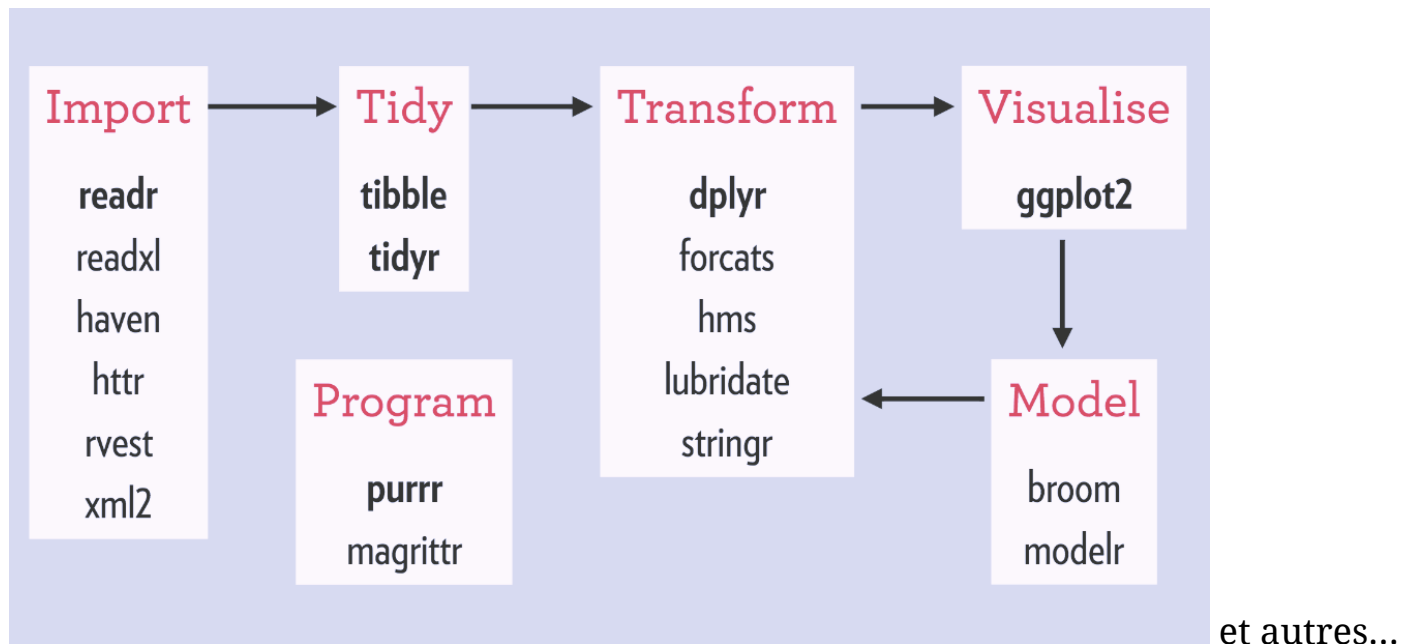
## Logiciels

- R (nécessaire)
- connaissances de base
- Ressources:
  - [R for Data Science](#) : guide au **tidyverse**
  - [TidyTuesday](#) : **a weekly social data project in R**
  - [ggplot2 cookbook](#)
- [RStudio](#) (conseillé)
  - interface complète pour **écrire, développer et produire**
  - literate programming : **R Markdown**, articles, présentations, sites web, ...

# R : les bases

# R moderne : le tidyverse





-  [tidyverse](https://www.tidyverse.org/) : écosystème de packages qui travaillent bien ensemble



- Installation et chargement :  
`install.packages("tidyverse")`  
`library(tidyverse)`

# R moderne : le tidyverse

Les plus importants :

-  [readr](#) : lecture des données de fichiers (conversion en `data.frames`)
-  [tidyr](#) : rangement de `data.frames`
-  [dplyr](#) : manipulation de `data.frames` :
  - filtrage, transformations, regroupage (voir : `plyr`, mais mieux !)
-  [ggplot2](#) : visualisation de données

Auteur principal : Hadley Wickham (<https://github.com/hadley/>)

# Caractéristiques communes

- Le premier paramètre de chaque fonction est un `data.frame` qui contient l'information
- la valeur de retour est toujours un `data.frame` (ou `tibble`)
- fonctions faciles et prévisibles :  
p.ex., `read_csv`, `mutate`, `select`, `summarize`, `filter`, `geom_point`, `geom_line`
- comportement prévisible : paramètres et valeurs de retour, erreurs, ...
- très bonne documentation : R help, sites Internet, blogs, github



# Caractéristiques communes : pipe

- le **flux d'informations** est indiqué avec l'opérateur `%>%` (**pipe**)

`f(x,y)` une fonction  
`x %>% f(y)` est équivalent à `f(x,y)`

- Conséquence :

**Lecture de gauche à droite !**  
`x %>% f(y) : x "then" f(y)`



- Clavier (R Studio) : par défaut
  - Windows : CTRL + ⬆ + M
  - Mac : ⌘ + ⬆ + M

# Caractéristiques communes : données rectangulaires

Dans le **tidyverse** on travaille avec :

## données rectangulaires (tidy data)

chaque ligne est une observation, chaque colonne est une variable

 **tidyr** vous aide ! (anciennement :  **reshape2**)

**gather**(data, key, value, ..., na.rm = FALSE,  
convert = FALSE, factor\_key = FALSE)

gather() moves column names into a **key**  
column, gathering the column values into a  
single **value** column.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

key value

**spread**(data, key, value, fill = NA, convert = FALSE,  
drop = TRUE, sep = NULL)

spread() moves the unique values of a **key**  
column into the column names, spreading the  
values of a **value** column across the new columns.

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

→

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

key value

Aussi: **separate**, **unite**, ...

 documentation ! : [Data Import cheatsheet](#)

# R moderne : exemple

Données sur les vols de New York en 2013 :  [nycflights13](#)

```
library(nycflights13)
head(flights, 20)
```

Search: <input type="text"/>									
	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time	sched_arr_time	arr_
1	2013	1	1	517	515	2	830		819
2	2013	1	1	533	529	4	850		830
3	2013	1	1	542	540	2	923		850
4	2013	1	1	544	545	-1	1004		1022
5	2013	1	1	554	600	-6	812		837
6	2013	1	1	554	558	-4	740		728
7	2013	1	1	555	600	-5	913		854

Showing 1 to 7 of 20 entries

Previous

1

2

3

Next

# R moderne : exemple

Devinez le résultat !

```
df <- flights %>%  
  filter(month %in% c(12, 1, 2)) %>%  
  group_by(dest) %>%  
  summarise(  
    count = n(),  
    dist = mean(distance, na.rm = TRUE),  
    delay = mean(arr_delay, na.rm = TRUE)  
  ) %>%  
  filter(count > 20, dest != "HNL")
```

(on a utilisé `dplyr` : fonctions `filter`, `group_by`, `summarise`, `n` et `%>%`)

# R moderne : exemple

Voilà :

```
head(df, 10)
```

```
## # A tibble: 10 x 4
##   dest  count  dist delay
##   <chr> <int> <dbl> <dbl>
## 1 ABQ      31 1826  30.3
## 2 ALB     174  143  26.8
## 3 ATL    4126  757.  9.14
## 4 AUS     521 1515. 13.4
## 5 AVL      31  583  12.1
## 6 BDL     112  116  11.3
## 7 BGR      56  378  19.7
## 8 BHM      58  866  17.5
## 9 BNA    1358  759. 16.6
## 10 BOS   3586  191.  3.25
```

# R moderne : exemple

`nycflights13` contient aussi des informations sur les aéroports :

```
head(airports, 5)
```

```
## # A tibble: 5 x 8
##   faa   name                lat   lon   alt   tz dst tzone
##   <chr> <chr>                <dbl> <dbl> <int> <dbl> <chr> <chr>
## 1 04G   Lansdowne Airport      41.1 -80.6  1044   -5 A   Americ
## 2 06A   Moton Field Municipal ~ 32.5 -85.7   264   -6 A   Americ
## 3 06C   Schaumburg Regional    42.0 -88.1   801   -6 A   Americ
## 4 06N   Randall Airport        41.4 -74.4   523   -5 A   Americ
## 5 09J   Jekyll Island Airport  31.1 -81.4    11   -5 A   Americ
```

# R moderne : exemple

Jointures avec `dplyr` :

```
df_airports <- df %>%  
  left_join(airports, by = c('dest' = 'faa'))
```

```
## # A tibble: 7 x 11  
##   dest  count  dist delay name      lat   lon  alt  tz dst  
##   <chr> <int> <dbl> <dbl> <chr>    <dbl> <dbl> <int> <dbl> <chr>  
## 1 ABQ      31 1826  30.3 Albuquerque~ 35.0 -107.  5355  -7 A  
## 2 ALB     174  143  26.8 Albany Intl  42.7 -73.8   285  -5 A  
## 3 ATL    4126  757.  9.14 Hartsfield~ 33.6 -84.4  1026  -5 A  
## 4 AUS     521 1515. 13.4 Austin Ber~ 30.2 -97.7   542  -6 A  
## 5 AVL      31  583  12.1 Asheville ~ 35.4 -82.5  2165  -5 A  
## 6 BDL     112  116  11.3 Bradley In~ 41.9 -72.7   173  -5 A  
## 7 BGR      56  378  19.7 Bangor Intl  44.8 -68.8   192  -5 A
```

# R moderne : exemple

Visualisation avec `ggplot2` :

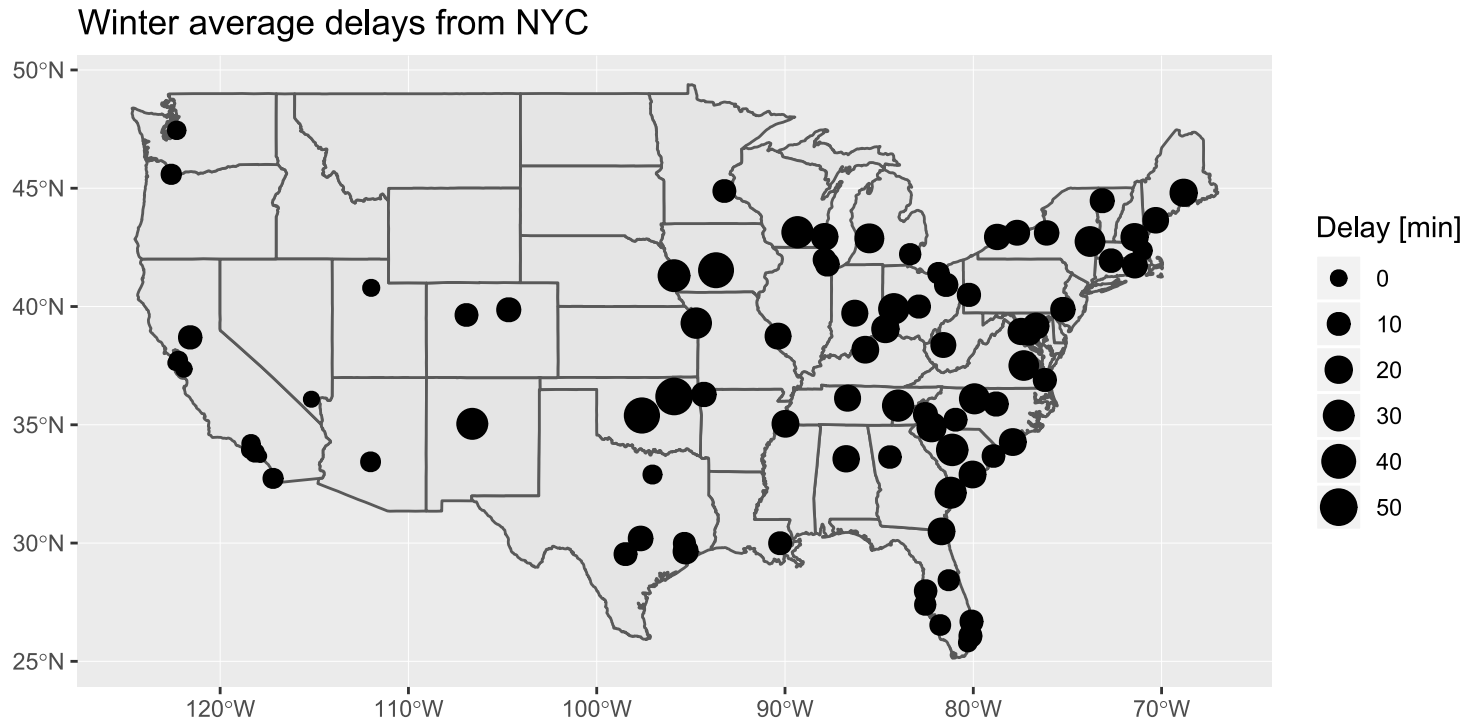
```
library(sf)
library(maps)

# Read the USA map
us_states <- sf::st_as_sf(
  maps::map("state", plot = FALSE, fill = TRUE)
)

p <- df_airports %>%
  ggplot() +
  geom_sf(data = us_states) +
  geom_point(aes(x = lon, y = lat, size = delay)) +
  ggtitle('Winter average delays from NYC') +
  labs(x = '', y = '') +
  scale_size_continuous('Delay [min]')
```



# R moderne : exemple



# La grammaire des données

# R : dplyr

## Grammaire des données

- Juste quelques notions !
- Les données sont des `data.frame` (`tibble`)
- **verbes** qui les transforment
  - filtrage par lignes : `filter`
  - filtrage par colonnes : `select`
  - ajout de colonnes : `mutate`, `rename`
  - groupage de lignes : `group_by`
  - rangement : `arrange`
  - résumé : `summarise`
- Connexions avec les bases des données plus communes (p.ex. `sqlite`)
- Paradigme **split-apply-combine**
  - données → "je regroupe → je transforme → je simplifie"

# Les graphiques

# R : base graphics

Ce qu'on apprend aux premiers courses : "base"

(p.ex. données : `mtcars`, caractéristiques de 32 automobiles)

- Scatter plot, points et lignes

```
plot(x = mtcars$mpg, y = mtcars$wt, col = ..., pch = ..., ...)  
lines(x = mtcars$mpg, y = mtcars$wt, col = ..., pch = ..., ...)  
points(x = mtcars$mpg, y = mtcars$wt, col = ..., pch = ..., ...)
```

- Diagrammes à barres

```
hist(x = mtcars$mpg)
```

- Densité

```
plot(density(x = mtcars$wt))
```

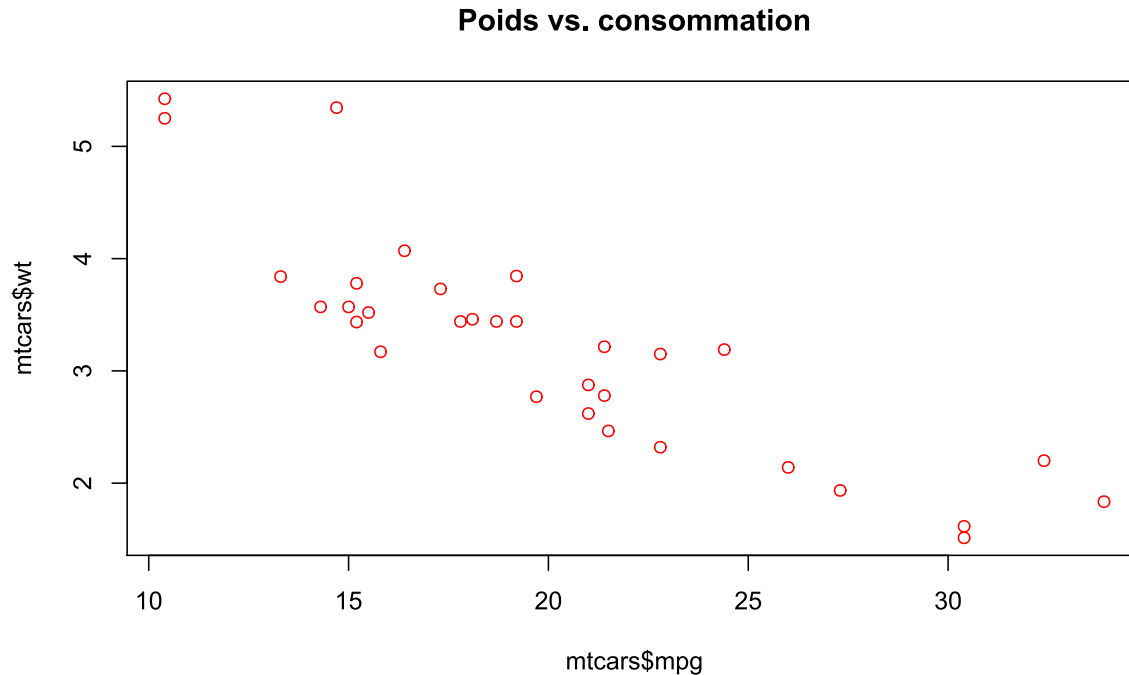
- Boxplot

```
boxplot(mtcars$wt)
```

# R : base graphics

Exemple :

```
plot(x = mtcars$mpg, y = mtcars$wt, type = 'p', col = 'red')  
title('Poids vs. consommation')
```



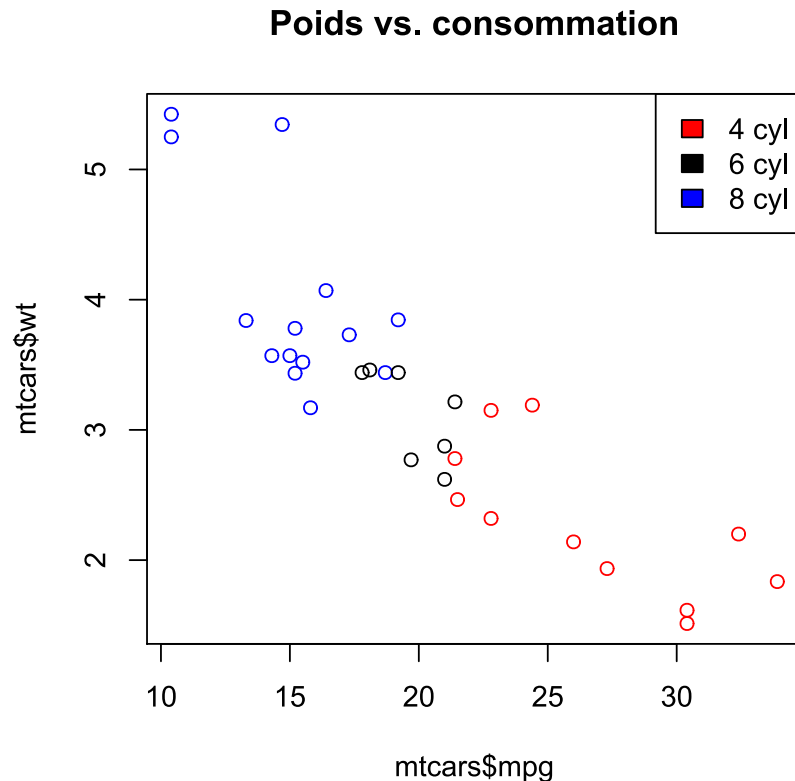
# R : base graphics, groupes

## Avec **données groupées** ça se complique !

Exemple : (coloration par cylindres)

```
# Couleurs ici
cols <- mtcars$cyl
cols[mtcars$cyl == 4] <- 'red'
cols[mtcars$cyl == 6] <- 'black'
cols[mtcars$cyl == 8] <- 'blue'

plot(x = mtcars$mpg, y =
      mtcars$wt,
      type = 'p', col = cols)
title('Poids vs. consommation')
legend('topright',
      c('4 cyl', '6 cyl', '8
cyl'),
      fill = c('red', 'black',
'blue'))
```



# R : base graphics, multiples

Graphiques multiples : il faut un cycle...

```
par(mfrow = c(3,1))
for (cyl_sub in
unique(mtcars$cyl)) {
  mtcars_sub <-
mtcars[mtcars$cyl == cyl_sub, ]
  plot(x = mtcars_sub$mpg, y =
mtcars_sub$wt, type = 'p')
  title(paste('Poids vs
consommation:', cyl_sub,
'cylindres'))
}
```



# R : base graphics

Inconvénients :

- aspect graphique pas agréable
- mécanisme compliqué
- interface pas uniforme
- duplication et répétition : Pr(erreur) augmente !
- ...

# La grammaire des graphiques

# Grammaire des graphiques

Un graphique est une combinaison d'éléments indépendants (**layers**) :

- **data** : un jeu de données
- **aesthetic mapping** :  
correspondance entre les variables dans **data** et les variables graphiques
- **geom** : un objet graphique (point, ligne, cercle, ...) qui sera montré
- **position** : le déplacement des objets de type **geom** (souvent: identité)
- **stat** : un objet qui calcule des statistiques (moyennes, ...) : (souvent: identité)
- **coord** : le système de coordonnées (cartésiennes, polaires, ...)
- **scale** : contrôle de la gamme des valeurs pour les données
- **facet** : ajout de sous-graphiques
- **theme** : réglage fin d'éléments graphiques
- Chaque graphique est une **somme** de ces niveaux.

# Grammaire des graphiques : exemple

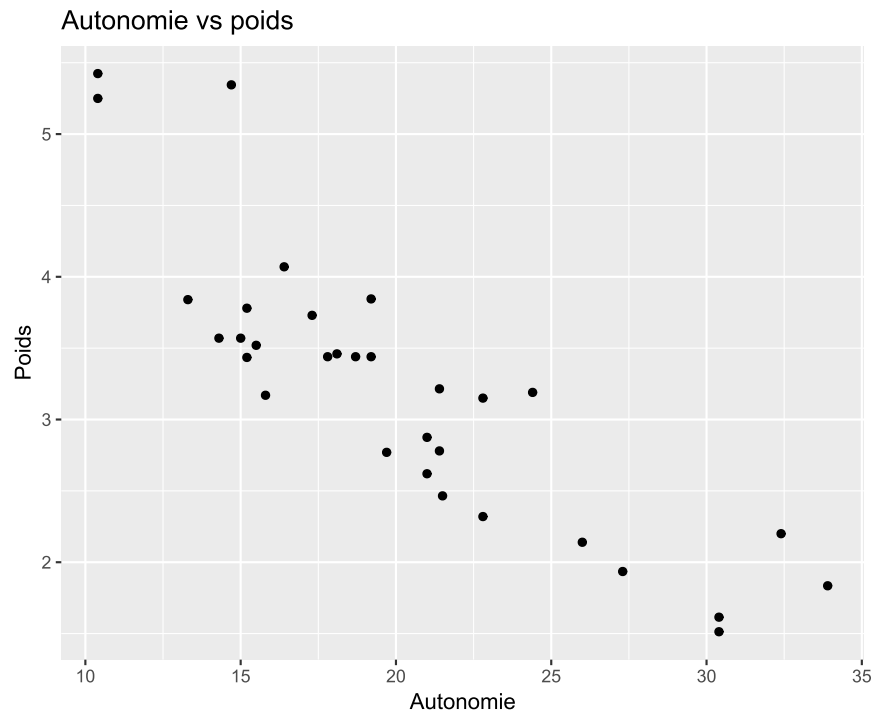
`ggplot2` est une implémentation de la grammaire des graphiques.

- L'objet de base est `ggplot(data = ...)`
- Puis on y somme les autres niveaux :

```
library(ggplot2)

ggplot(mtcars) +
  geom_point(aes(x = mpg, y =
wt)) +
  ggtitle('Autonomie vs poids')
+
  labs(x = 'Autonomie', y =
'Poids')
```

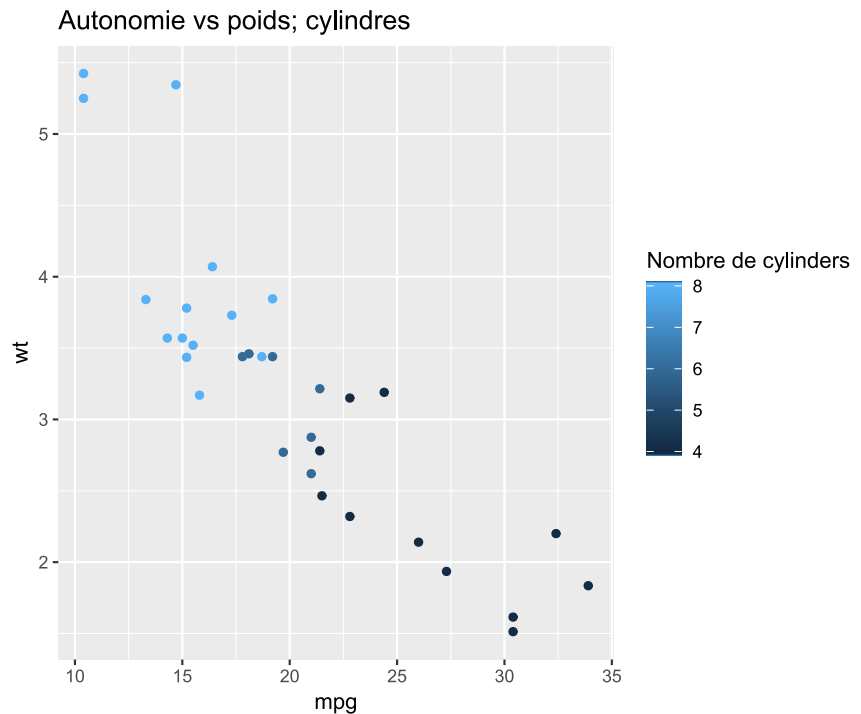
- `ggtitle` : le titre
- `labs` : étiquettes des axes, titre, sous-titre, ...



# Grammaire des graphiques : exemple avec groupes

```
p <- ggplot(mtcars) +  
  geom_point(aes(x = mpg, y =  
wt, col = cyl)) +  
  ggtitle('Autonomie vs poids;  
cylindres') +  
  
  scale_color_continuous('Nombre  
de cylindres')  
p
```

- **data** : `mtcars`
- **geom\_point** : ajout d'une couche graphique de type "scatter plot"  
Variables graphiques : `x`, `y`, `col` (couleur : facultative)
- **aes** : **aesthetic mapping**  
dans `data`: `mpg`, `wt`, `cyl` ⇒  
variables `x`, `y`, `col` dans le graphique
- **ggplot** adopte une échelle de couleur automatique pour des valeurs continues

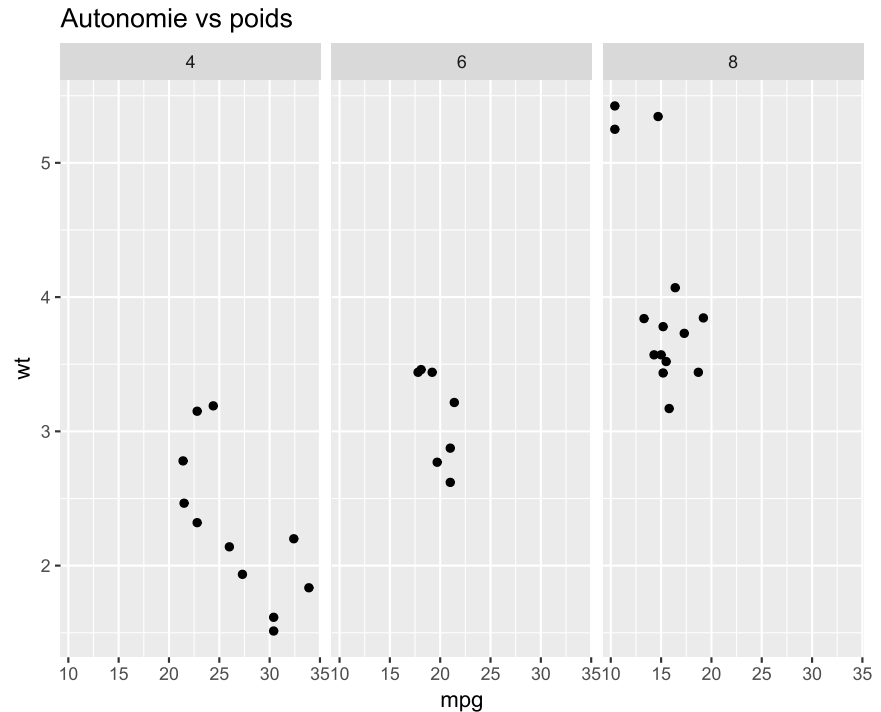


# Grammaire des graphiques : exemple

L'exemple avec les sous-groupes : facile !

```
ggplot(mtcars) +  
  geom_point(aes(x = mpg, y =  
wt)) +  
  facet_wrap( ~ cyl) +  
  ggtitle('Autonomie vs poids')
```

- Le sous-groupe est la couche `facet_wrap`
- `ggplot` crée des sous-graphiques avec les valeurs de son argument
- sous-graphiques automatiquement étiquetées



# Géométries

# Géométries

Les plus importantes :

- `geom_point` : points
- `geom_line` : lignes
- `geom_text`, `geom_label` : du texte
- `geom_abline`, `geom_vline`, `geom_hline` :  
lignes selon pente / intersection avec les axes
- `geom_histogram`, `geom_boxplot`, `geom_density`
- `geom_polygon`, `geom_path` :  
un polygone ou parcours, vertex en ordre de apparition dans `data`
- `geom_raster`, `geom_contour` : pour données 2D

Mini-guide :  [Data visualization cheatsheet](#)

## Anatomie d'une géométrie :

- `geom_*(mapping = ..., data = ..., stat = ..., position = ..., ...)`
- `mapping` spécifie les **esthétiques** (correspondances entre variables)
- `data = ...` est utile pour lire un `data` modifié




# Esthétiques

Presque toutes les géométries supportent les esthétiques :

- `x`, `y`, `col` (bord), `fill` (remplissage), `alpha` (transparence).

Autres esthétiques :

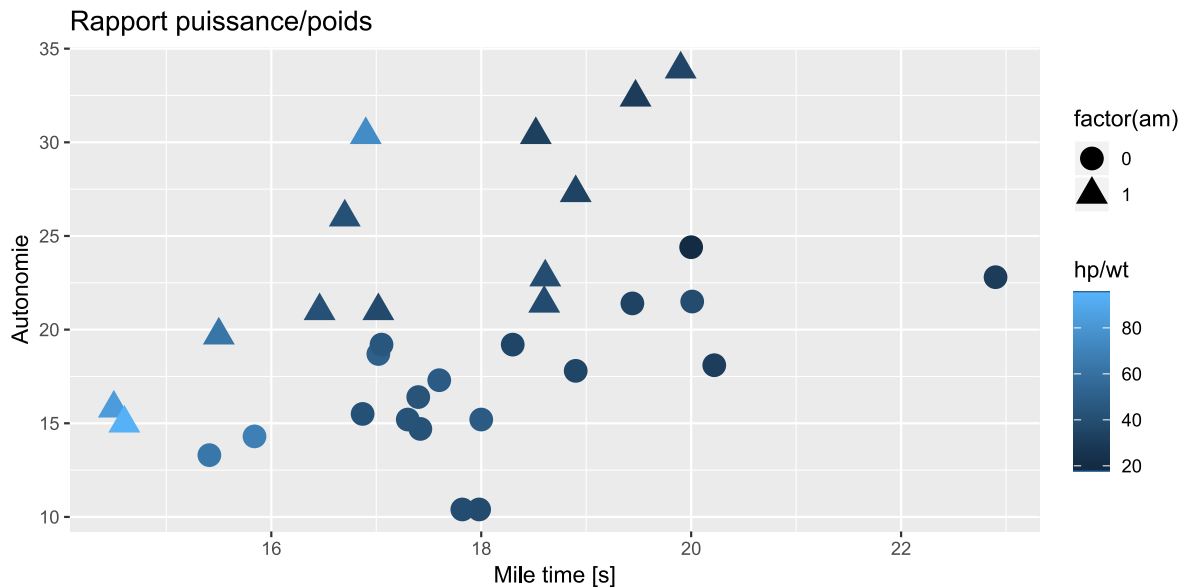
- `shape`, `lwd` (poids des lignes), `lty` (type des lignes), `size` (dimension des objets), ...
- Géométries particulières :  
`xmin`, `xmax`, `ymin`, `ymax`, `group`, ... →  documentation !
- On peut les spécifier ailleurs :  
`geom_*(aes(...))`, ou `ggplot2(aes(...))` (partagées parmi les `geoms`)
- Variables non in `data` : placés en dehors de `aes`

```
my_size <- 0.5
ggplot(mtcars, aes(x = mpg, y = wt)) +
  geom_point(aes(col = cyl), size = my_size)
```

# Esthétiques

Vous pouvez utiliser des expressions :

```
ggplot(mtcars) +  
  geom_point(  
    aes(x = qsec, y = mpg,  
        col = hp/wt,  
        shape = factor(am)),  
    size = 5) +  
  labs(title = 'Rapport puissance/poids', x = 'Mile time [s]', y = 'Autonomie')
```

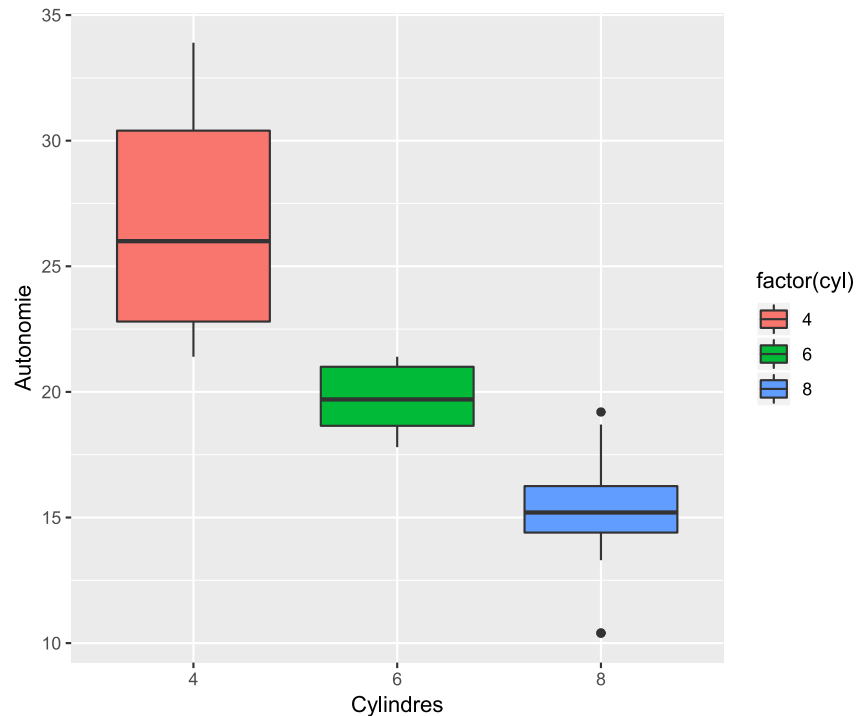


# Un boxplot...

Consommation vs nombre de cylindres

```
mtcars %>%  
  ggplot(aes(x = factor(cyl), y  
= mpg)) +  
  geom_boxplot(aes(fill =  
factor(cyl))) +  
  labs(x = 'Cylindres', y =  
'Autonomie')
```

`labs(...)` : titres des axes



Boxplot : médiane, quartiles, moustaches ... mais qui les a calculés ?

# Les statistiques

# Les statistiques

Avant de plotter :

- **data** est séparé en groupes selon les variables discrètes :
  - ce qu'on trouve dans **facets**
  - ce qu'on assigne à **col**, **fill**, **group**, ...
- Pour chaque sous-ensemble de **data**, le niveau **stat** calcule tout le nécessaire
- Conséquence : à chaque **geom** correspond un **stat** (souvent caché !)

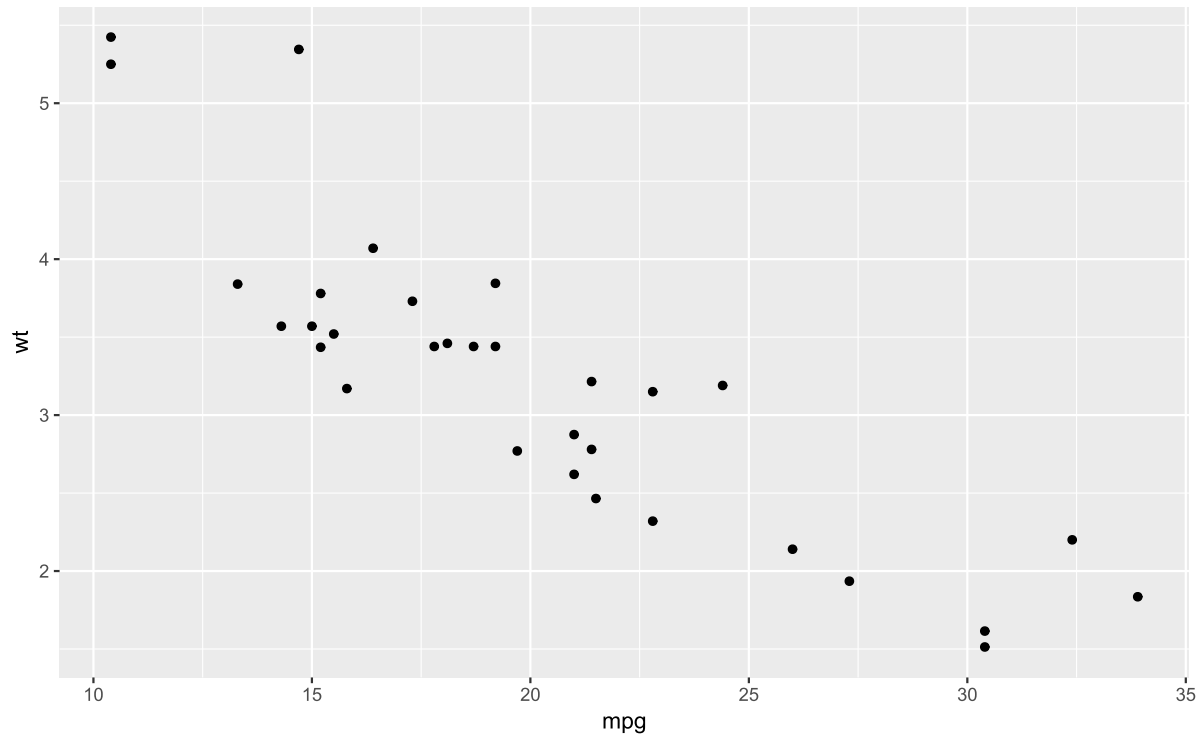
En pratique :

- **stat** ajoute des **colonnes** à **data** avec des noms fixes

# Les statistiques : exemple

Consommation vs poids : notez les esthétiques partagées

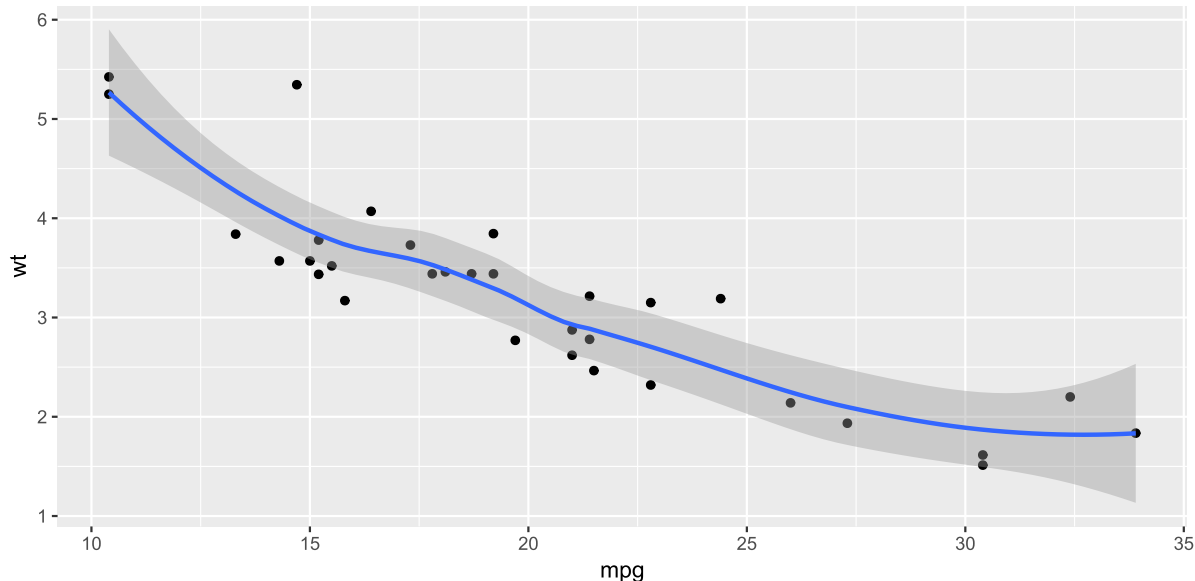
```
p <- ggplot(mtcars, aes(x = mpg, y = wt)) +  
  geom_point()  
p
```



# Les statistiques : exemple

- Ajout du niveau `geom_smooth` : smoothing
  - linéaire : `method = "lm"` (et autres)
  - non-linéaire : dans ce cas `method = "loess"` (et autres)
- `geom_smooth` adore `stat_smooth` :
  - calcule les prédictions `y`, intervalles `ymin`, `ymax`, erreur `se`
  - représentation graphique optimale

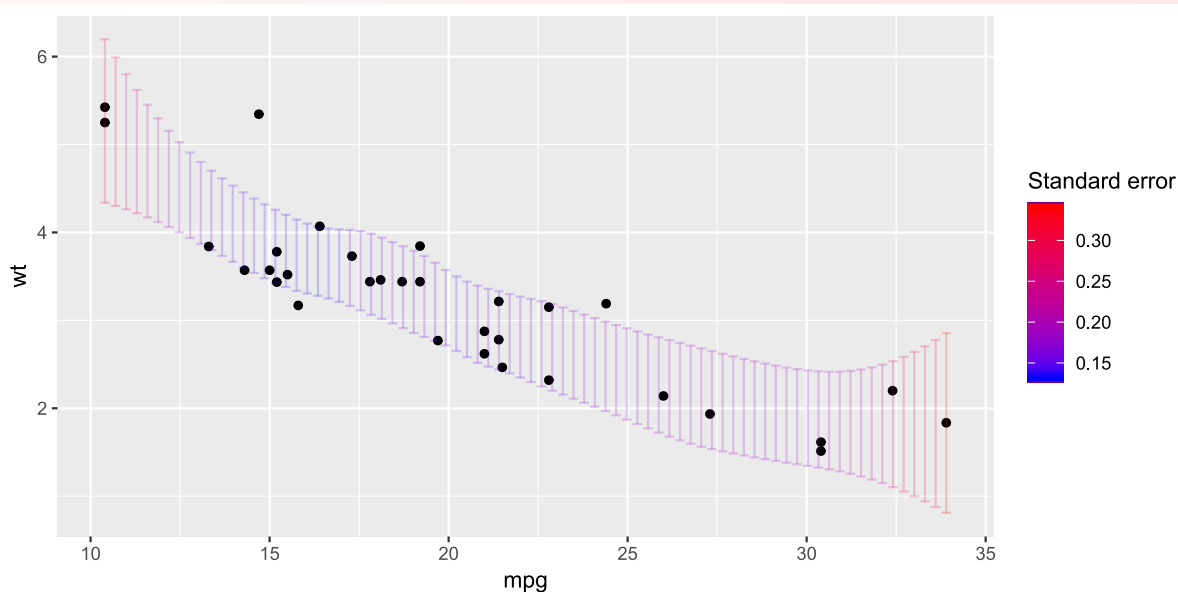
```
p + geom_smooth(method = 'loess')
```



# Les statistiques : exemple

- On peut accéder aux valeurs avec `stat(...)` dans le même niveau !  
(vieux code : les nouvelles variables commencent par `...`)
- Moyenne + intervalles de confiance pour la moyenne,  $6\sigma$

```
p +
  geom_errorbar(aes(ymin = y - 3*stat(se), ymax = y + 3*stat(se), col = stat(se)),
    alpha = 0.2,
    stat = 'smooth', method = 'loess') +
  scale_color_gradient('Standard error', low = 'blue', high = 'red')
```



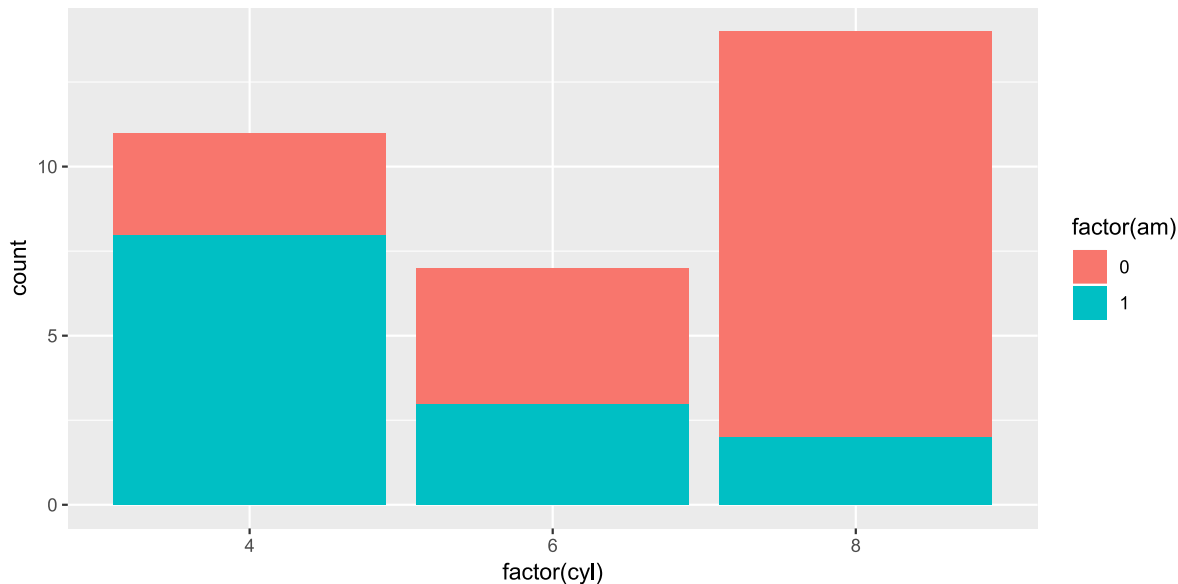


# Position

Quand on a des **éléments superposés** : (p.ex. diagramme à barre)

- Il y a une `stat_count` cachée ! (définit *y*)
- Par défaut, empilés (`geom_bar`) ou superposés (`geom_point`)

```
ggplot(mtcars) +  
  geom_bar(aes(x = factor(cyl), fill = factor(am)))
```

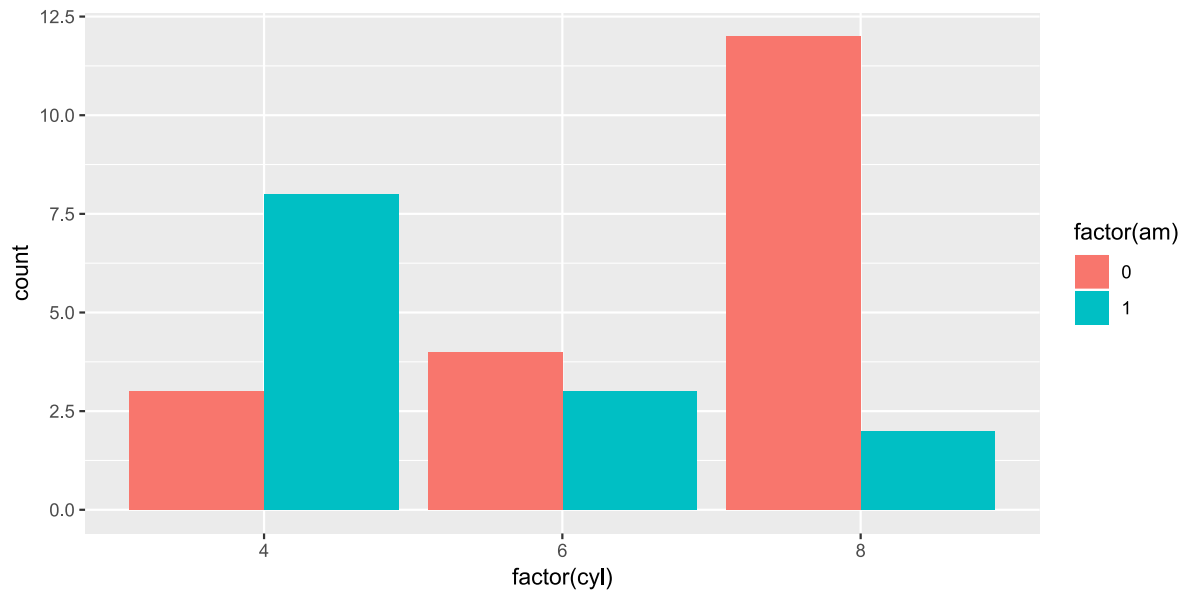


# Position

Paramètre `position` :

- `dodge` : à côté

```
ggplot(mtcars) +  
  geom_bar(aes(x = factor(cyl), fill = factor(am)),  
            position = 'dodge')
```

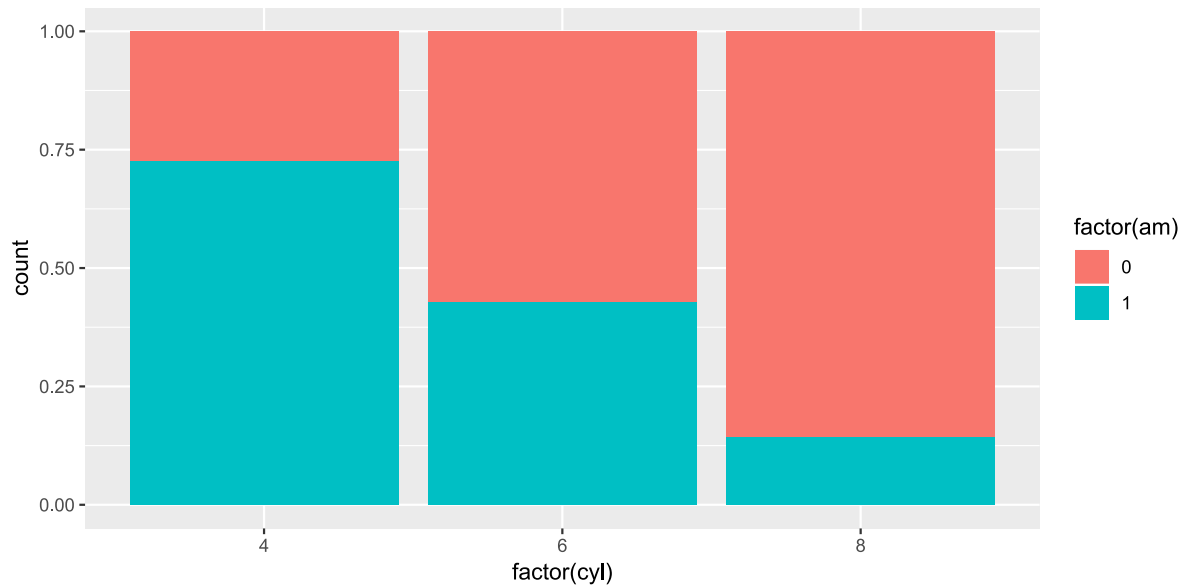


# Position

Paramètre `position` :

- `fill` : normalise

```
ggplot(mtcars) +  
  geom_bar(aes(x = factor(cyl), fill = factor(am)),  
           position = 'fill')
```

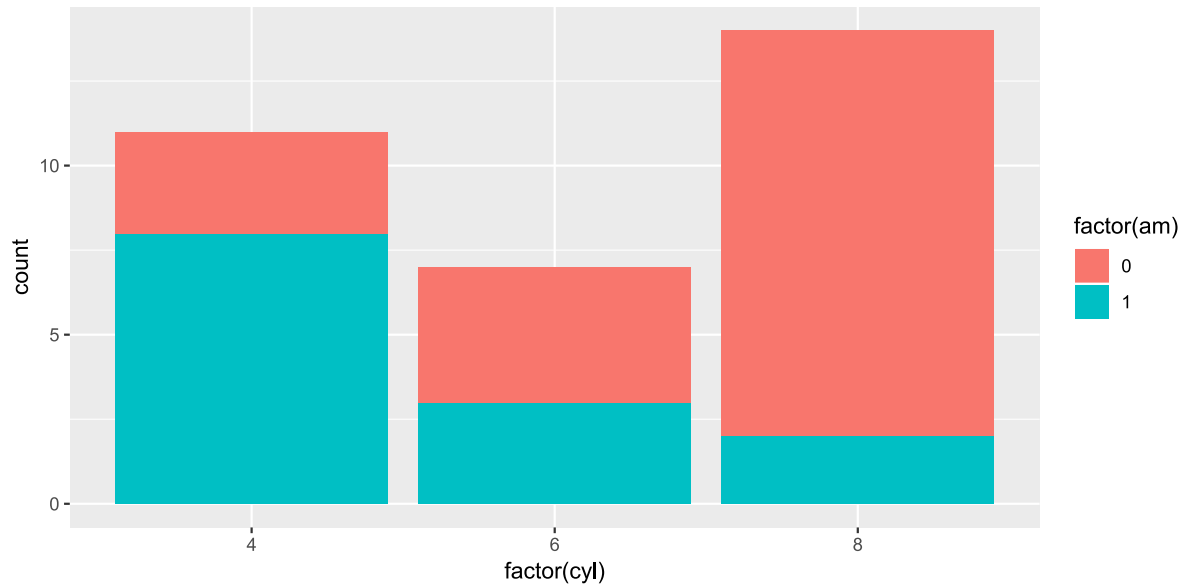


# Position

Paramètre `position` :

- `stack` : empiler

```
ggplot(mtcars) +  
  geom_bar(aes(x = factor(cyl), fill = factor(am)),  
           position = 'stack')
```

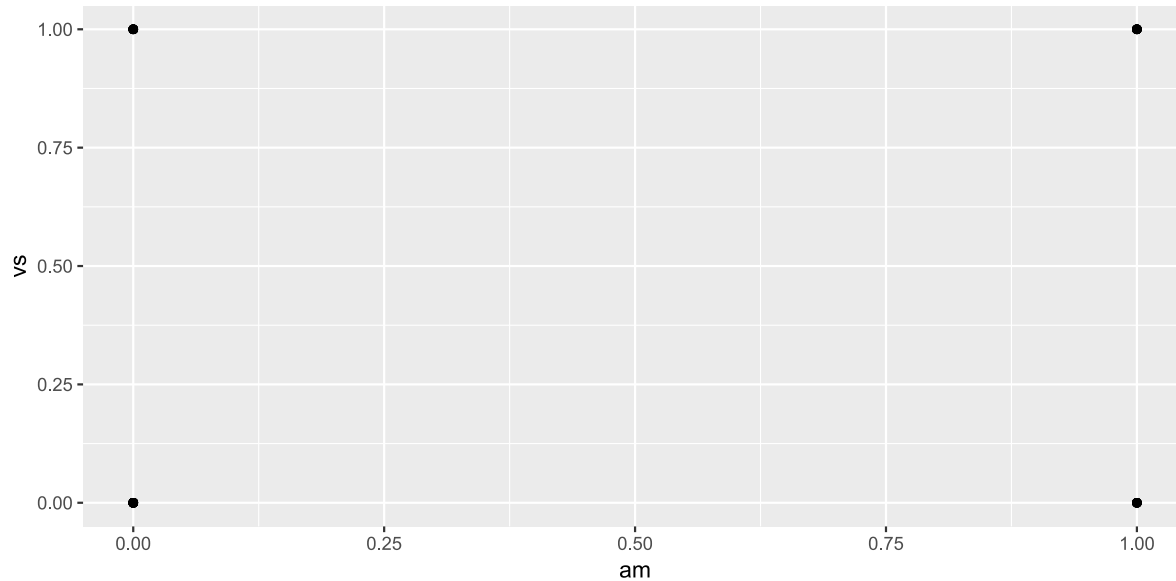


# Position

Paramètre **position** :

- **jitter** : ajout du bruit pour séparer

```
ggplot(mtcars) +  
  geom_point(aes(x = am, y = vs))
```

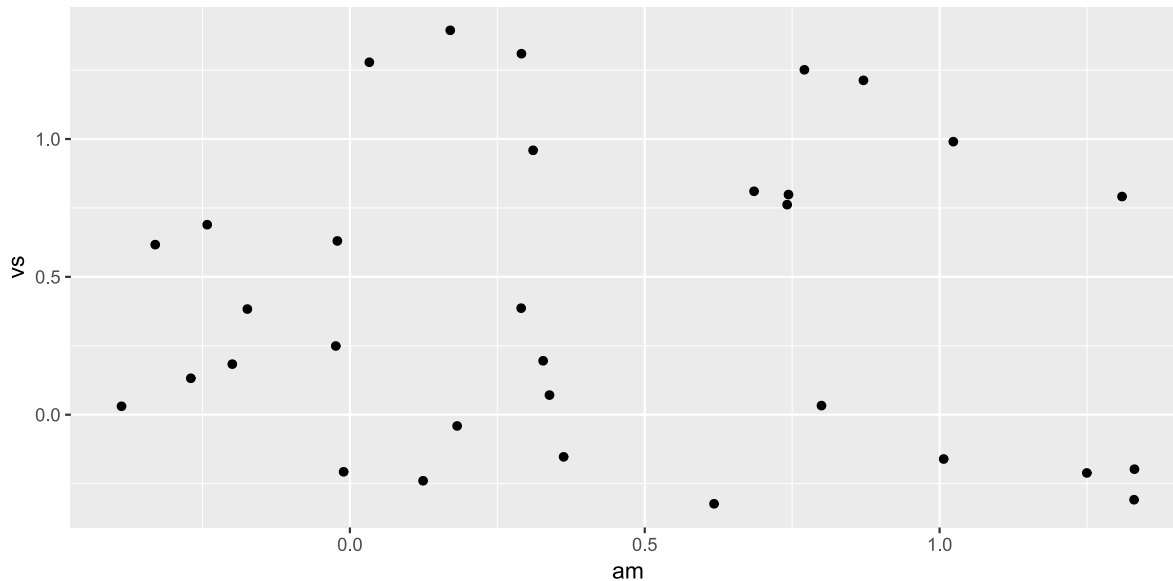


# Position

Paramètre **position** :

- **jitter** : ajout du bruit pour séparer

```
ggplot(mtcars) +  
  geom_point(aes(x = am, y = vs),  
             position = 'jitter')
```

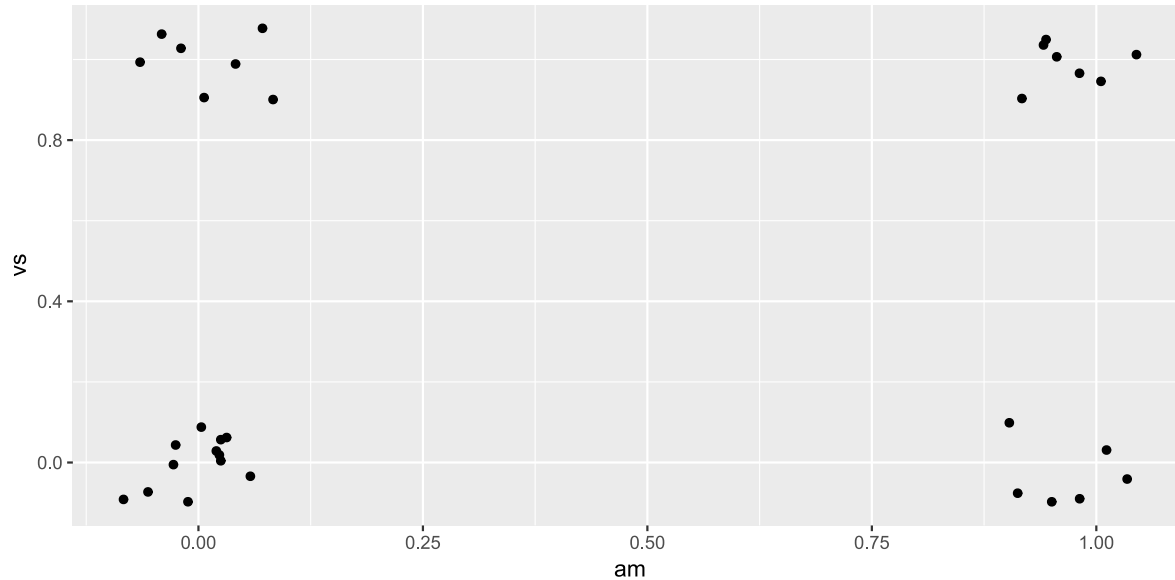


# Position

Paramètre `position` :

- `jitter` : ajout du bruit pour séparer  
Réglage ou `geom_jitter()`

```
ggplot(mtcars) +  
  geom_point(aes(x = am, y = vs),  
             position = position_jitter(width = 0.1, height = 0.1))
```



# Coordinates

- Cordonnées cartésiennes x-y  
`coord_cartesian(xlim = ..., ylim = ..., expand = ...)`
  - `xlim, ylim` : contrôle du **zoom**
  - `expand (TRUE/FALSE)` : ajout d'espace avant/après les extrêmes
- Cordonnées cartésiennes aux proportions fixés  
`coord_fixed(ratio = ...)`
- Cordonnées polaires  
`coord_polar(theta = 'x', ...)`
  - `theta` : qui gère l'angle
- Cordonnées inversées  
`coord_flip()`  
x devient y



# Scales

Correspondance entre les valeurs des **données** et les valeurs dans le **graphique**.

Échelles aux valeurs ...

- continues : types `numeric`, `int`, `Date`, ...
- discrètes : tout le reste (`factor`, `char`, ...)
- Noms : `scale_` + esthétique + type d'échelle (`continuous`, `discrete`, `manual`, ...)

## Anatomie d'une échelle

Toutes échelles acceptent au moins :

- `name` : nom dans la légende
- `breaks` : les valeurs seuil où on trouve des droites, des signes, ... (défaut : intelligent)
- `labels` : les chaînes de caractères associées aux `breaks` (défaut : intelligent)
- `trans` : transformation données  $\rightarrow$  (`breaks`, `labels`) (p.ex. `'log10'`)
- `limits` : les valeurs aux extrêmes (`NA` : min/max données)  
Aussi : fonctions `xlim( )`, `ylim( )`

# Scales : x, y

Exemples :

- `scale_x_continuous(trans = 'log10')` :  
échelle  $x$  logarithmique (`scale_x_log10()`)
- `scale_y_discrete(breaks = c('a', 'b'), labels = c('Premier', 'Deuxième'))` :  
échelle  $y$  discrète
- `scale_y_datetime()` : échelle  $y$  pour dates  
(= `scale_y_continuous(trans = 'date')`)

## Scales vs coord\_x\_\* et coord\_y\_\*

**scales** : les valeurs au delà des limites sont **remplacés** avec NA

⚠ coupure d'éléments graphiques ! ⚠

# Scales : couleurs, remplissage

Fini avec les couleurs de R !

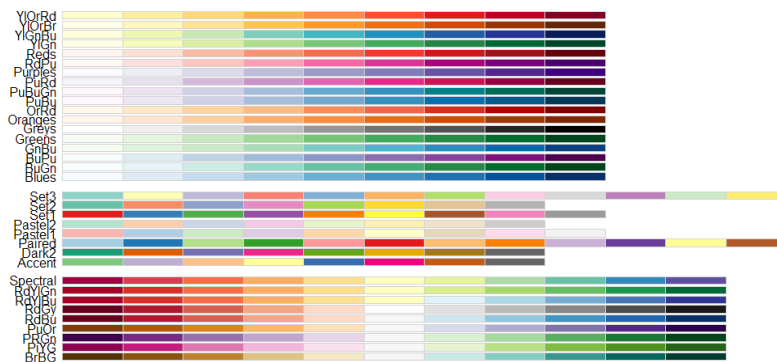
- Échelles chromatiques **optimisées** et **automatiques**
- Pour les esthétiques `col`, `fill`

Continues :

- `scale_color_continuous` : générique
- `scale_color_gradient`, `scale_color_gradient2`, `scale_color_gradientn`
- `scale_color_gray` : valeurs de gris

Discrètes :

- `scale_color_brewer` : palettes de [ColorBrewer](http://colorbrewer2.org/) : <http://colorbrewer2.org/>
  - 3 types : **diverging**, **qualitative**, **sequential**
  - liste : `RColorBrewer::display.brewer.all()`



- `scale_fill_viridis_d`:  
même principe mais avec la palette Viridis (Python : `matplotlib`)



# Facets

Deux façons :

- `facet_wrap(...)` : aligner sur une dimension
- `facet_grid(...)` : aligner en deux dimensions

Les axes seront partagés entre les graphiques !  
(sauf avec p.ex. `scales = "free_x"`)

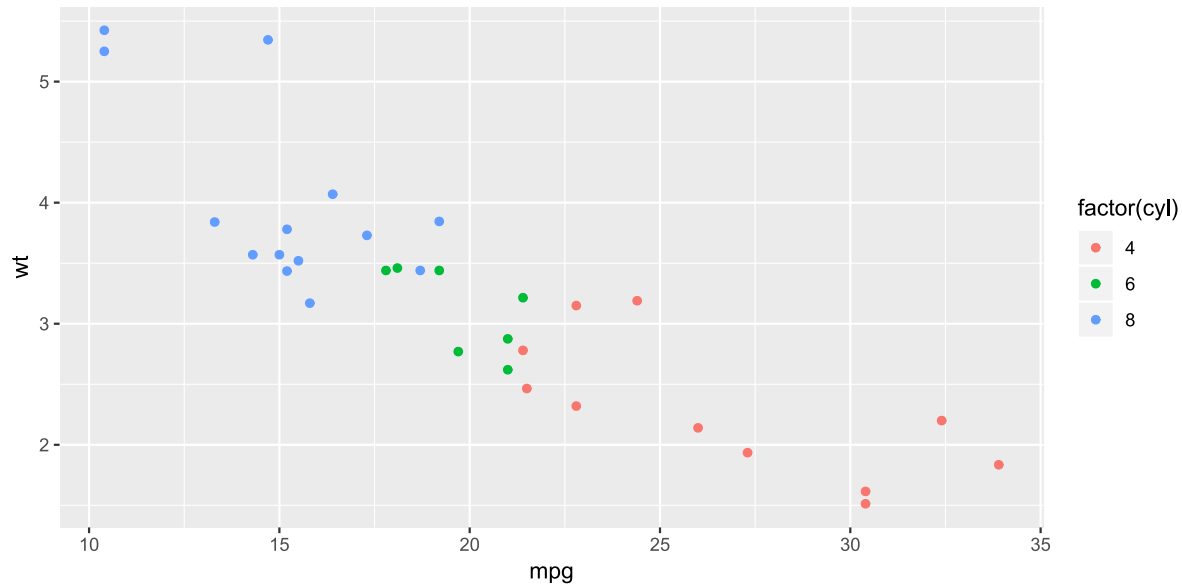
Paramètres :

- `...` :
  - une formule du type `colonnes ~ lignes` (vieille interface)
  - `vars(colonnes), vars(lignes)` (nouvelle interface)
- `labeller` : types d'étiquettes pour les variables dans les onglets
  - `labeller = label_value` : que la valeur
  - `labeller = label_both` : 'nom : valeur'
  - `labeller = label_parsed` : si `valeur` est une expression (`plotmath`)

# Facets : exemple

Sans onglets :

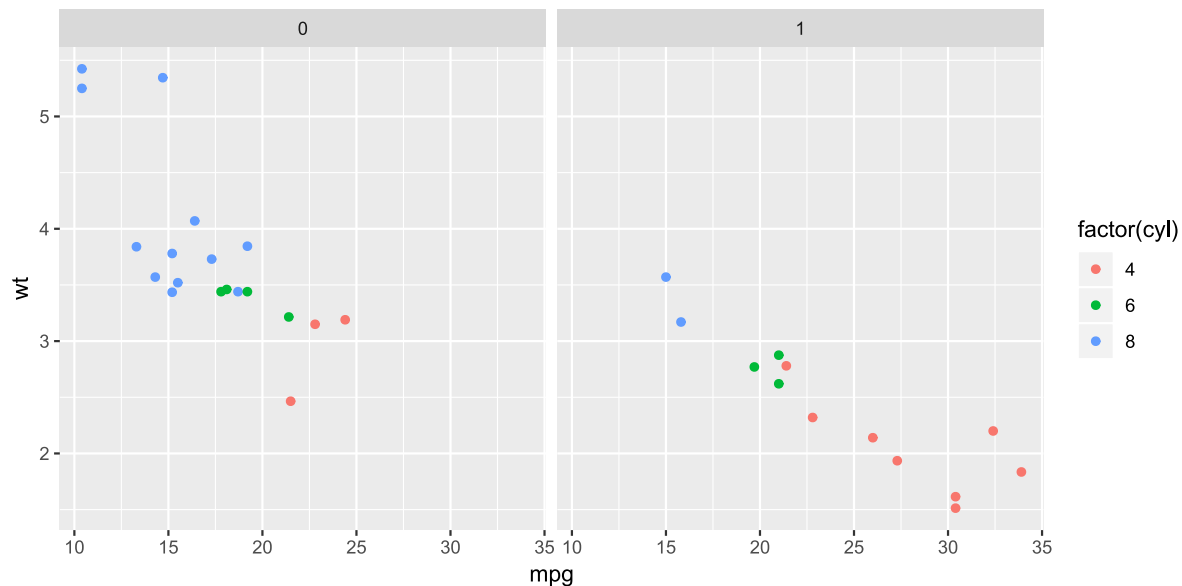
```
mtcars %>%  
  ggplot(aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point()
```



# Facets : exemple

`facet_wrap` : par transmission (automatique / manuelle)

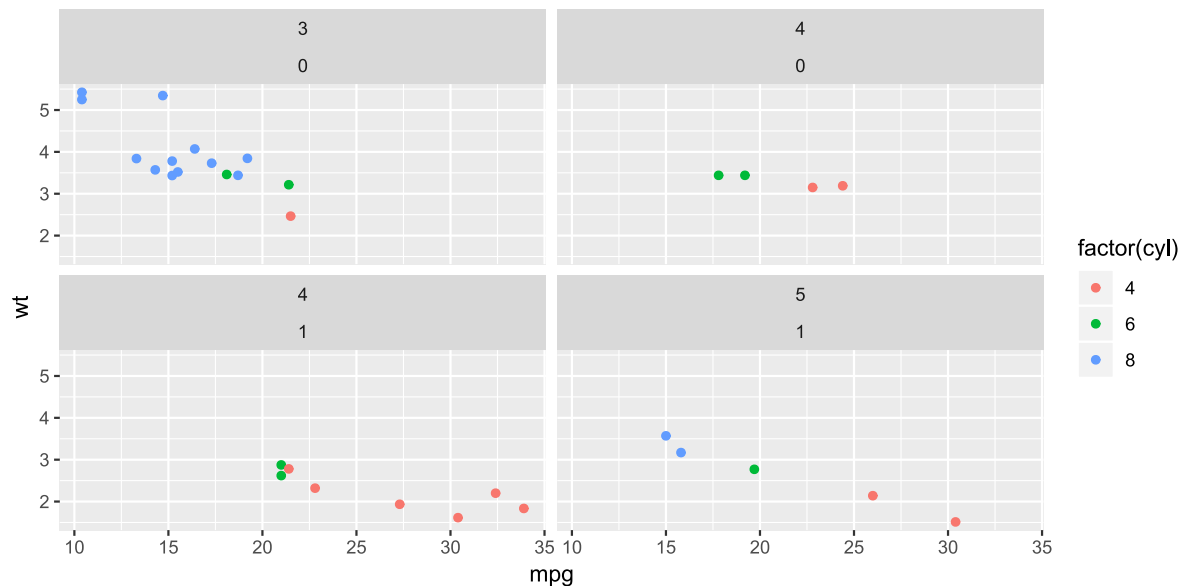
```
mtcars %>%  
  ggplot(aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point() +  
  facet_wrap(~ am) # vars(am)
```



# Facets : exemple

`facet_wrap` avec deux variables (nombre vitesses, transmission)

```
mtcars %>%  
  ggplot(aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point() +  
  facet_wrap(gear ~ am) # ou vars(gear, am)
```

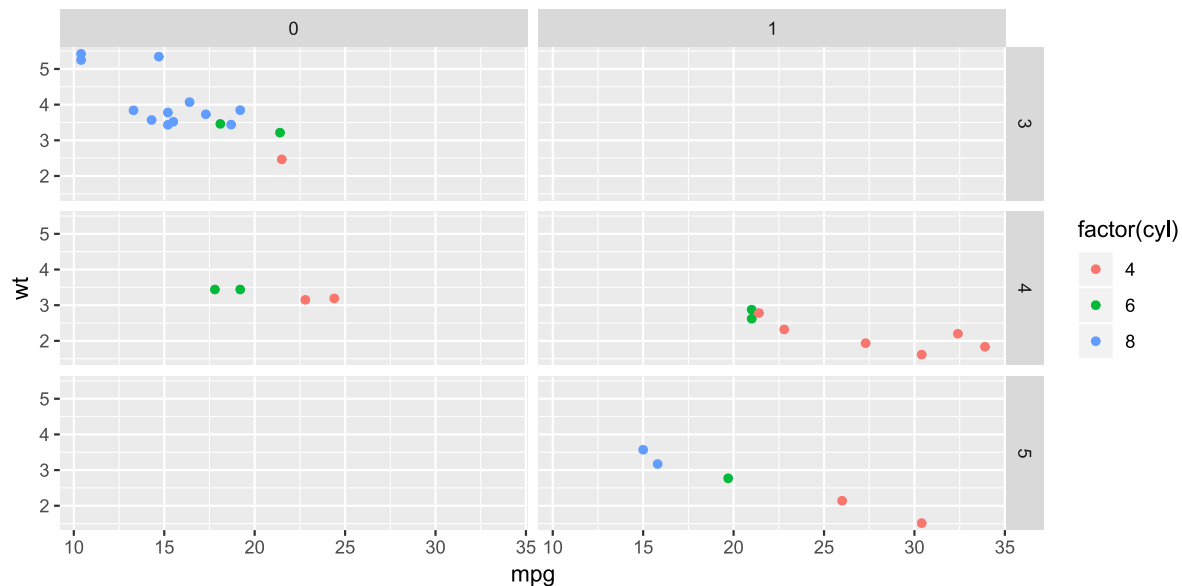




# Facets : exemple

`facet_grid` : mieux !

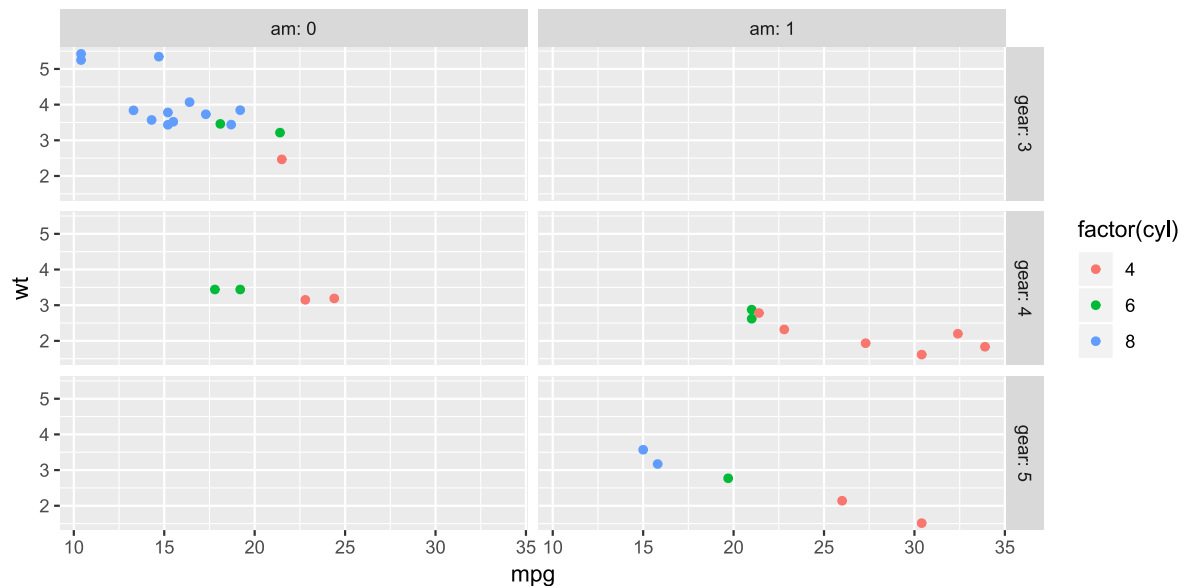
```
mtcars %>%  
  ggplot(aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point() +  
  facet_grid(gear ~ am)
```



# Facets : exemple

`facet_grid` : mieux, avec étiquettes

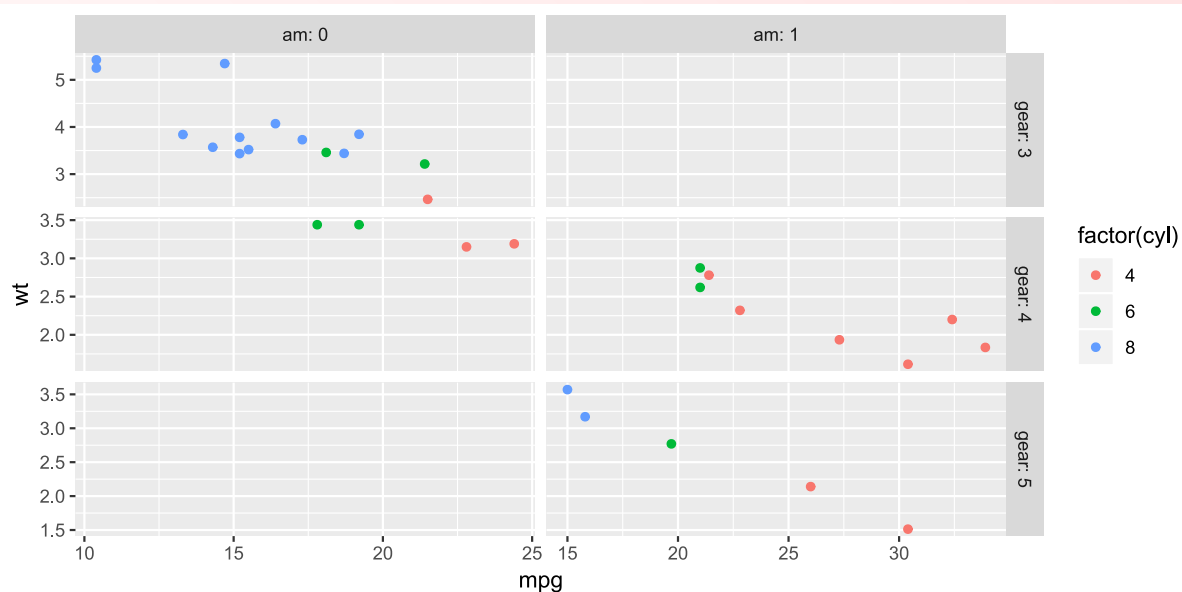
```
mtcars %>%  
  ggplot(aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point() +  
  facet_grid(gear ~ am, labeller = label_both)
```



# Facets : exemple

`facet_grid` : découplage des axes (mais toujours partagés !)

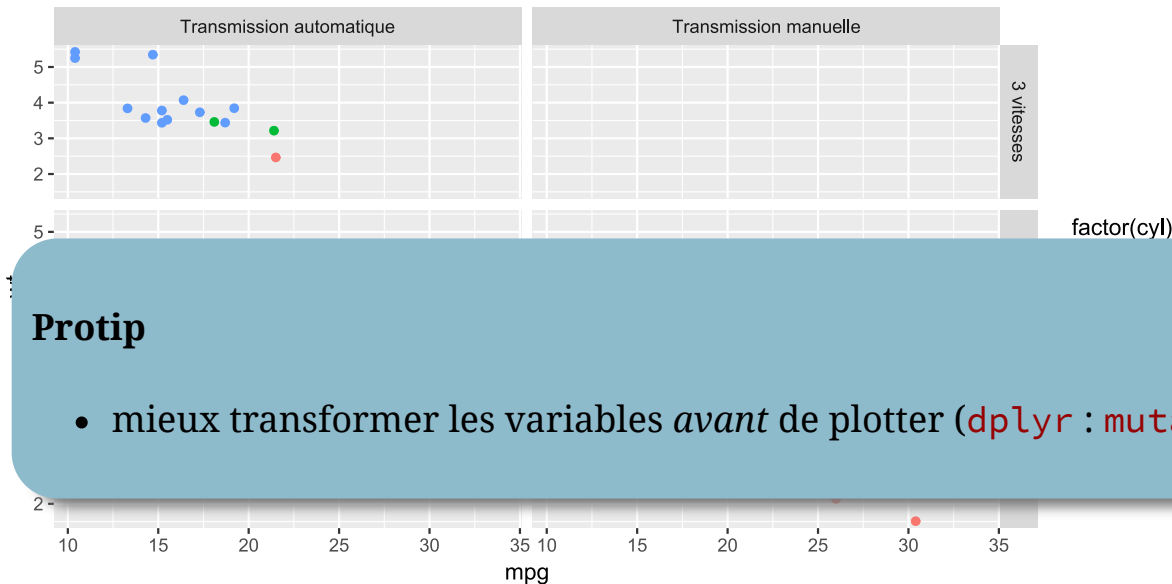
```
mtcars %>%
  ggplot(aes(x = mpg, y = wt, col = factor(cyl))) +
  geom_point() +
  facet_grid(gear ~ am, labeller = label_both, scales = 'free')
```



# Facets : exemple

Rien vous empêche d'utiliser des expressions !


```
mtcars %>%  
  ggplot(aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point() +  
  facet_grid(paste(gear, 'vitesses') ~  
             paste('Transmission',  
                   ifelse(am == 0, 'automatique', 'manuelle')))
```




## Protip

- mieux transformer les variables *avant* de plotter (`dplyr : mutate`)

# Thèmes

Avec `theme(element = valeur, ...)` vous pouvez modifier les éléments graphiques  
(→  documentation !)

## element

- `title` : tous les titres
- `axis.title.*` : les titre des axes
- `axis.text.*` :  
textes dans les marques sur les axes
- `axis.ticks.*` :  
marques sur les axes
- `axis.line.*` :  
les droites des axes
- `legend.*` :  
les légendes
- `panel.*` : fond du graphique
- `plot.*` : autour le graphique
- `strip.*` : composants de `facets`
- ...  documentation !

## valeur

Tous les éléments sont du type :

- `element_line(...)` : droits
- `element_rect(...)` :  
rectangles (bords, fonds)
- `element_text(...)` : texte
- `element_blank(...)` :  
pour effacer

# Thèmes : un exemple

Base

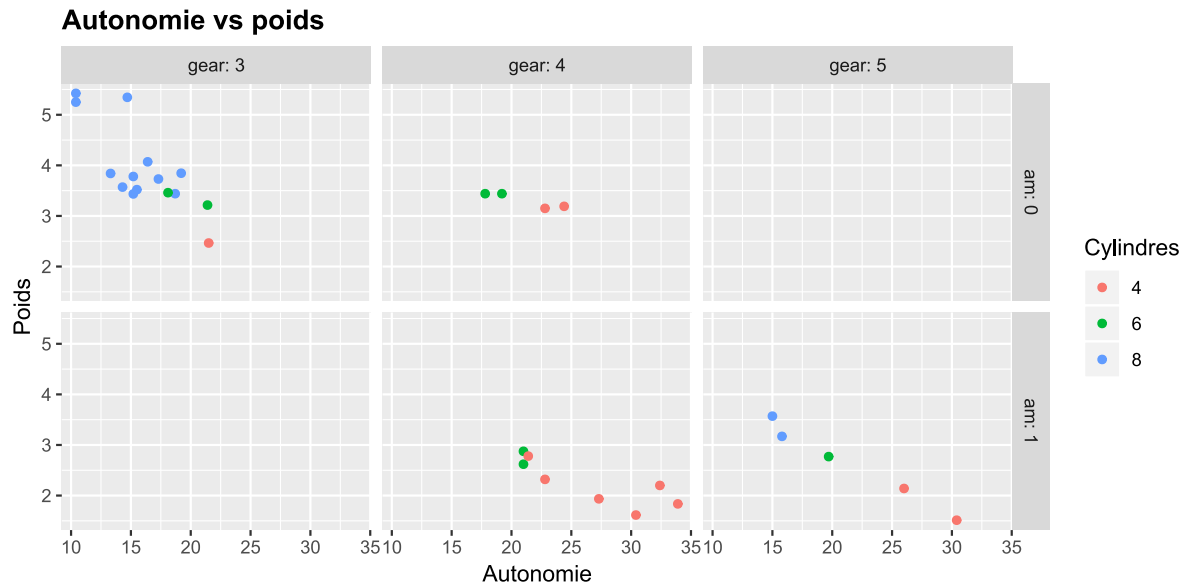
```
p <- ggplot(mtcars, aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point() +  
  ggtitle('Autonomie vs poids') +  
  scale_color_discrete('Cylindres') +  
  labs(x = 'Autonomie', y = 'Poids') +  
  facet_grid(am ~ gear, labeller = label_both)
```

p

# Thèmes : un exemple

Titre gras

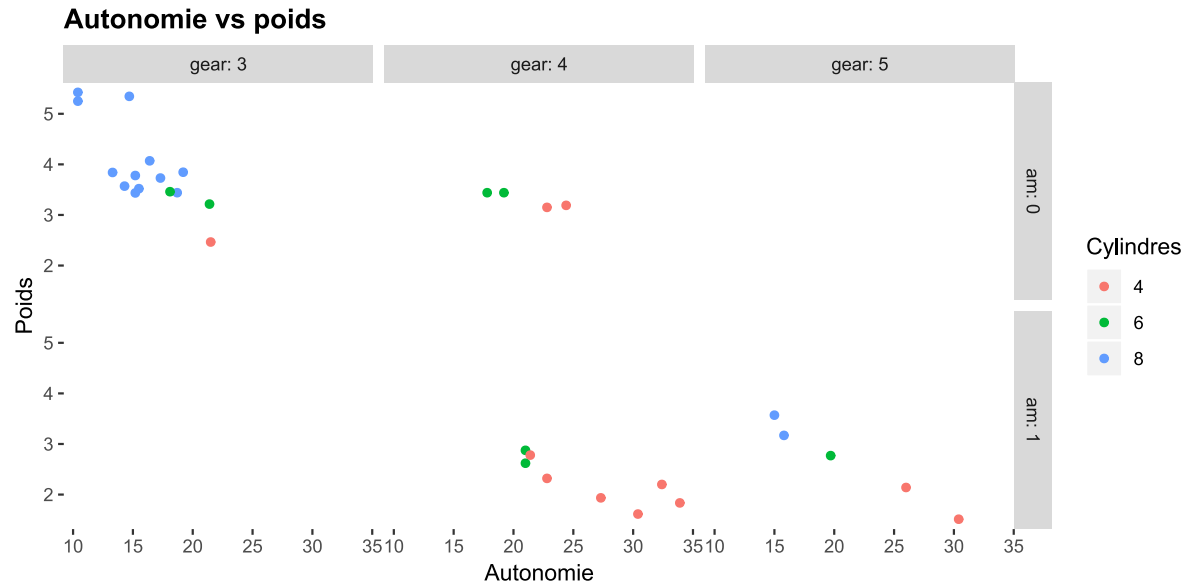
```
p + theme(  
  plot.title = element_text(face = 'bold'))
```



# Thèmes : un exemple

Sans le fond

```
p + theme(  
  plot.title = element_text(face = 'bold'),  
  panel.background = element_blank())
```

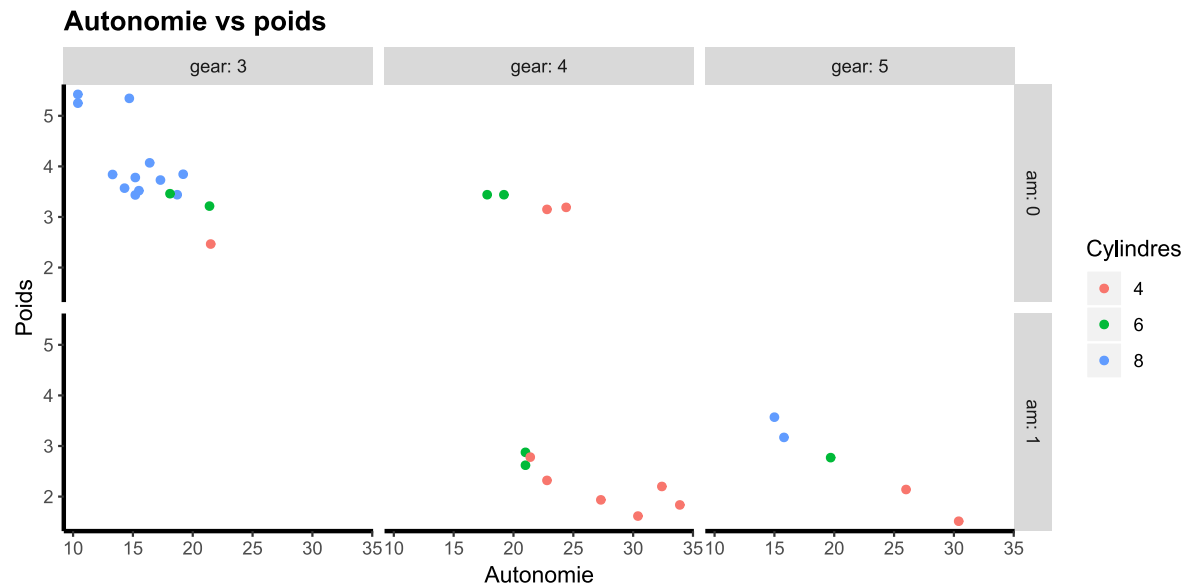




# Thèmes : un exemple

Avec axes

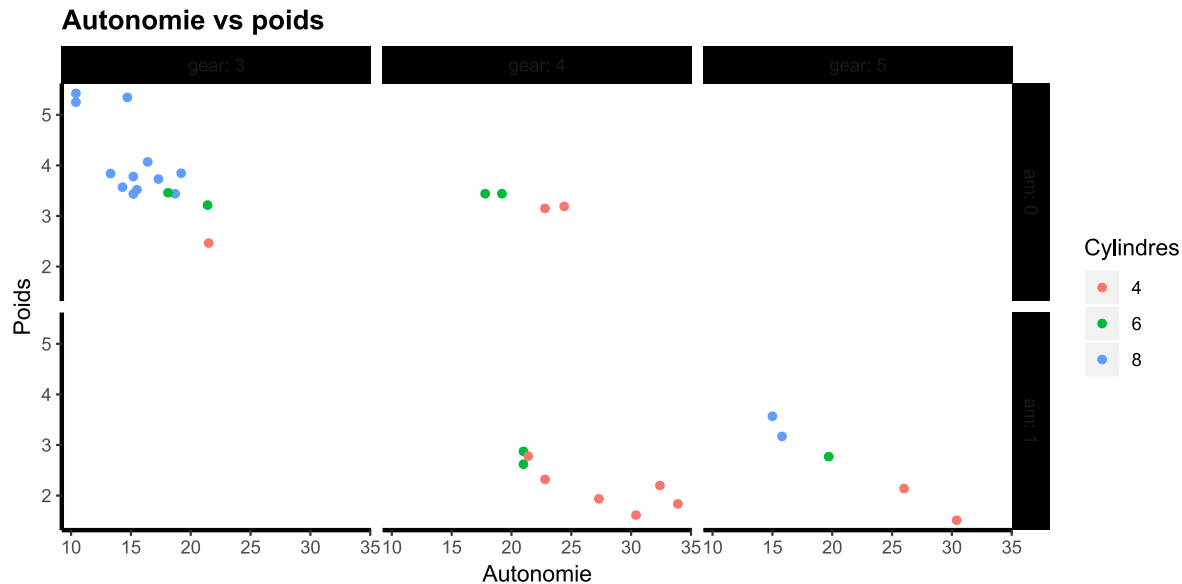
```
p + theme(  
  plot.title = element_text(face = 'bold'),  
  panel.background = element_blank(),  
  axis.line = element_line(size = 1))
```



# Thèmes : un exemple

## Fond des onglets

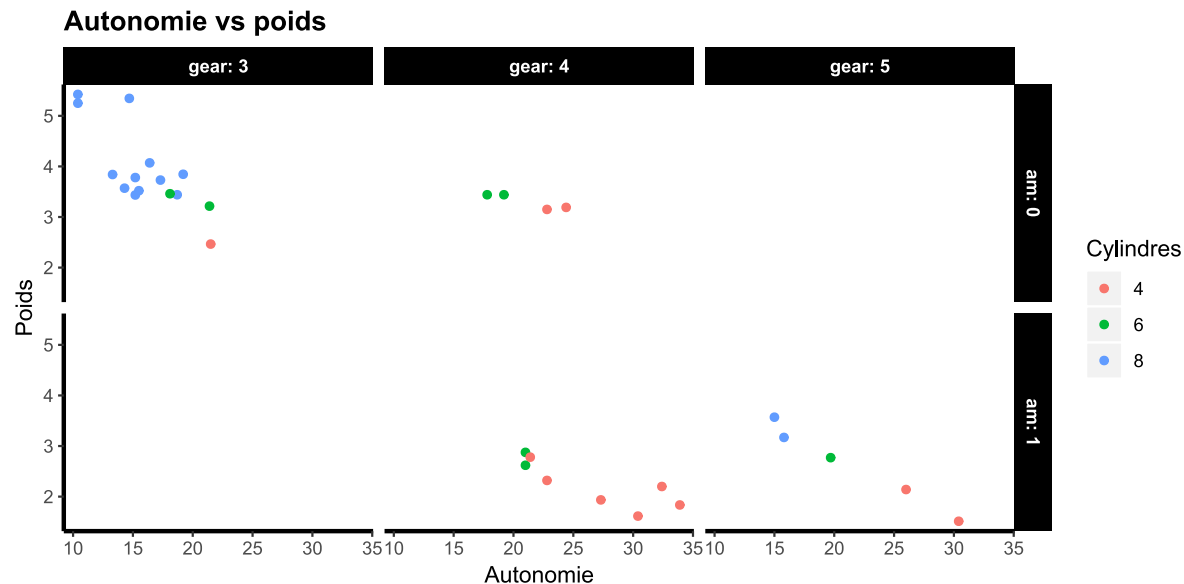
```
p + theme(  
  plot.title = element_text(face = 'bold'),  
  panel.background = element_blank(),  
  axis.line = element_line(size = 1),  
  strip.background = element_rect(fill = 'black'))
```



# Thèmes : un exemple

Texte des onglets

```
p + theme(  
  plot.title = element_text(face = 'bold'),  
  panel.background = element_blank(),  
  axis.line = element_line(size = 1),  
  strip.background = element_rect(fill = 'black'),  
  strip.text = element_text(color = 'white', face = 'bold'))
```



# Thèmes : un exemple

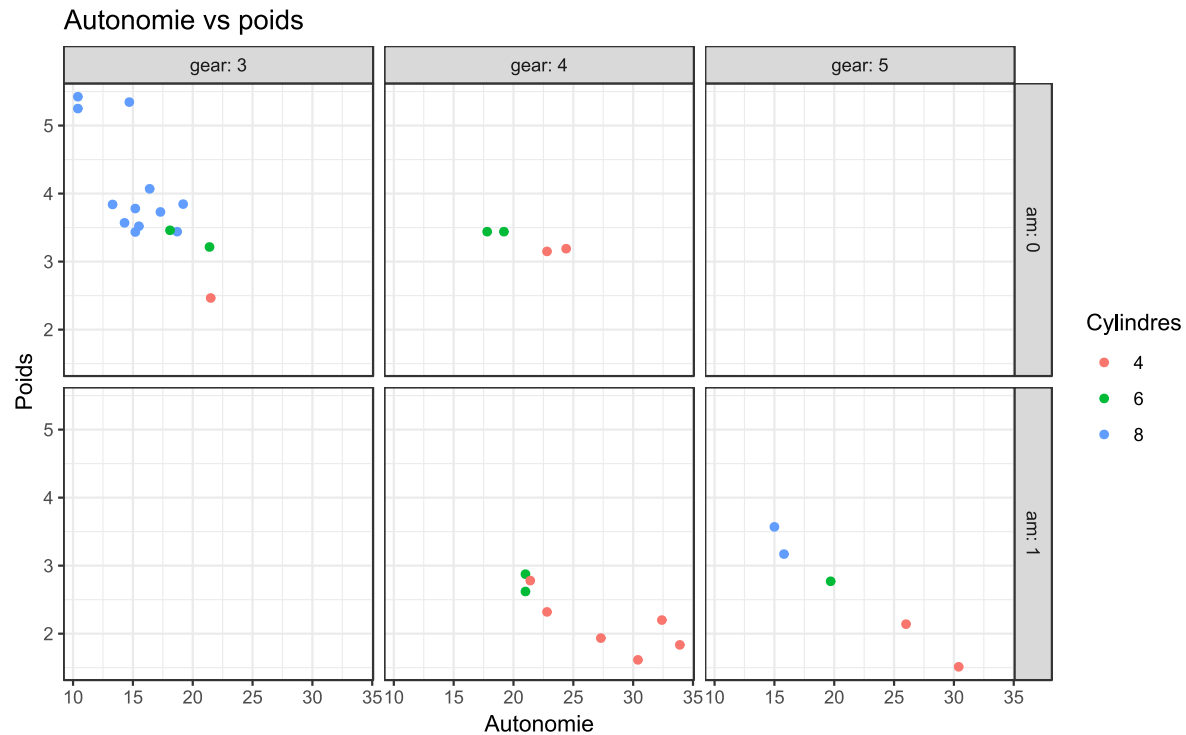
Étiquettes axe x

```
p + theme(  
  plot.title = element_text(face = 'bold'),  
  panel.background = element_blank(),  
  axis.line = element_line(size = 1),  
  strip.background = element_rect(fill = 'black'),  
  strip.text = element_text(color = 'white', face = 'bold'),  
  axis.text.x = element_text(angle = 90))
```

# Thèmes : exemples

Inclus :

- `theme_bw()` : noir et blanc

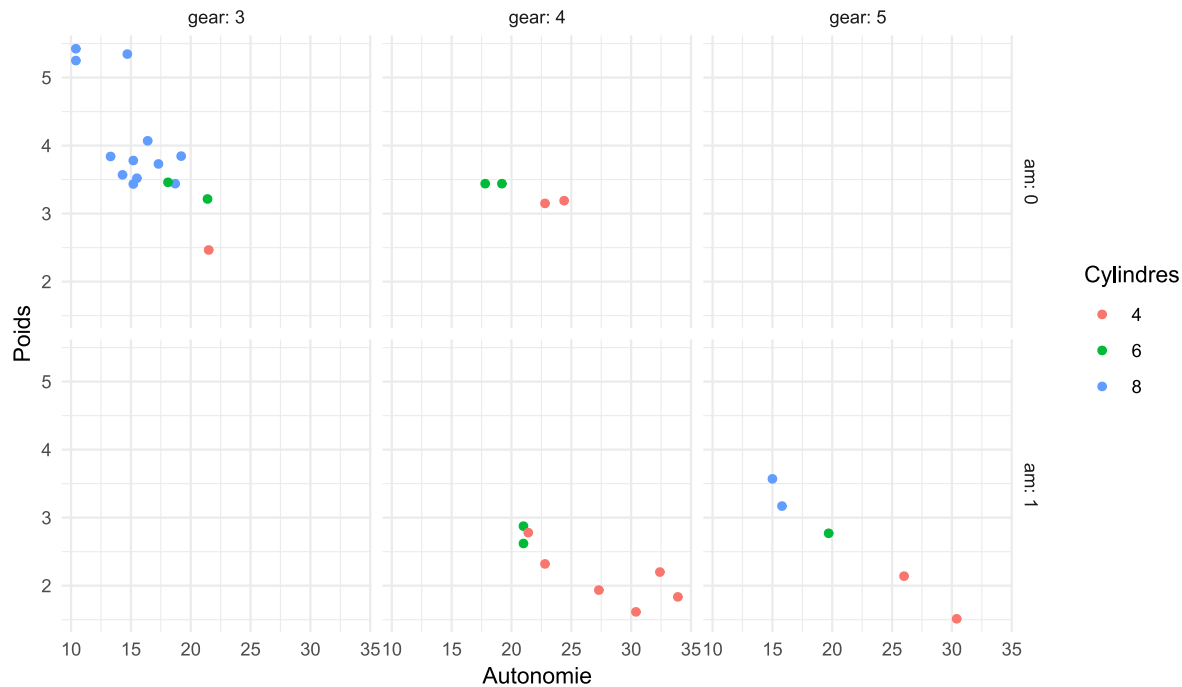


# Thèmes : exemples

Inclus :

- `theme_minimal()` : minimaliste

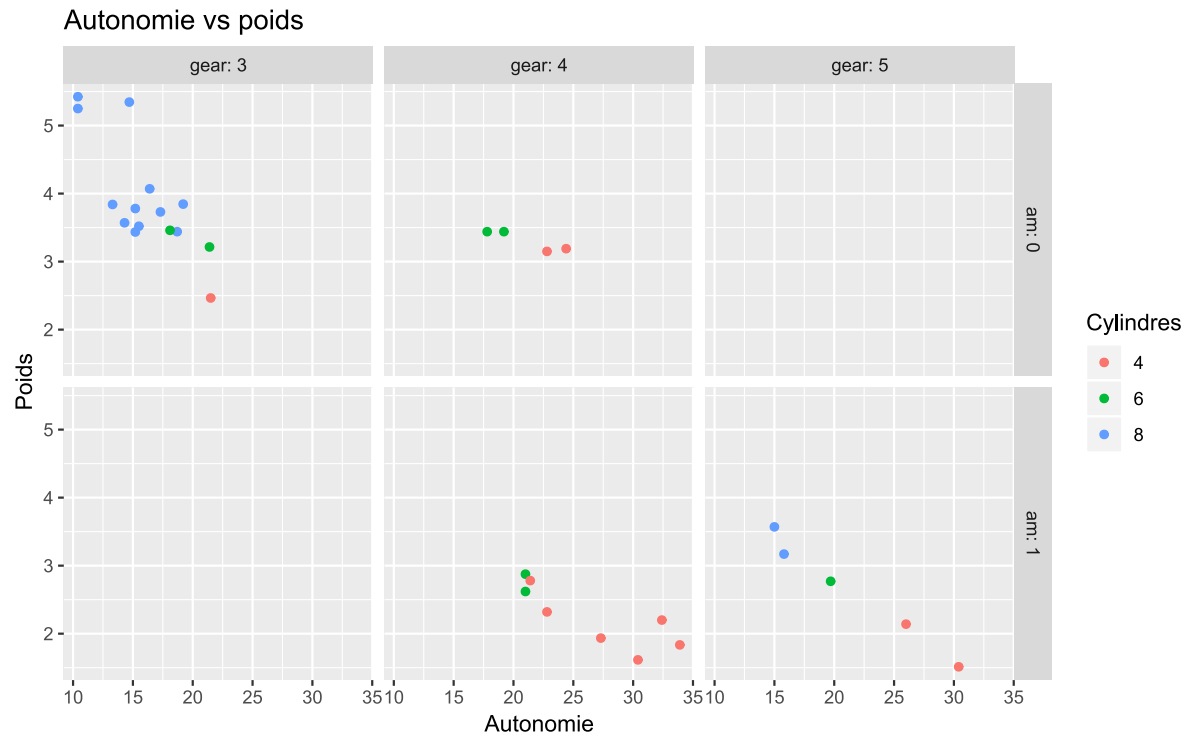
## Autonomie vs poids



# Thèmes : exemples

Inclus :

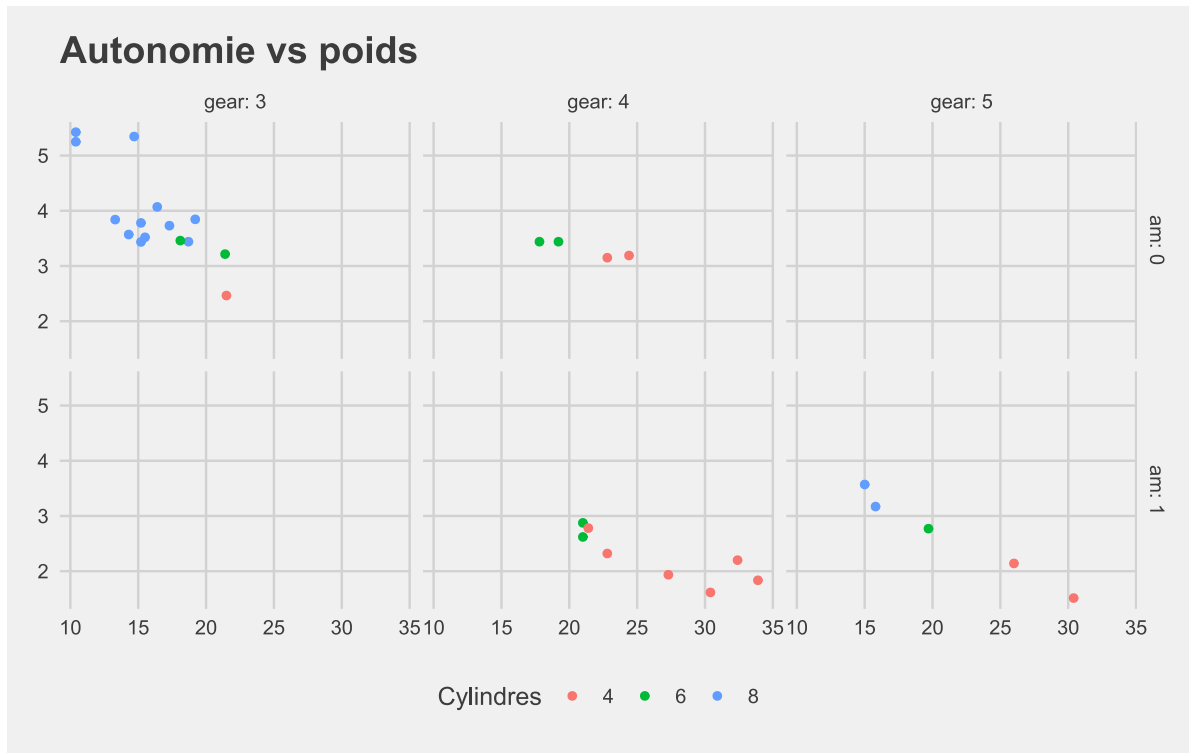
- `theme_gray()` : gris



# Thèmes : exemples

Package  [ggthemes](#) :

- `theme_fivethirtyeight()`



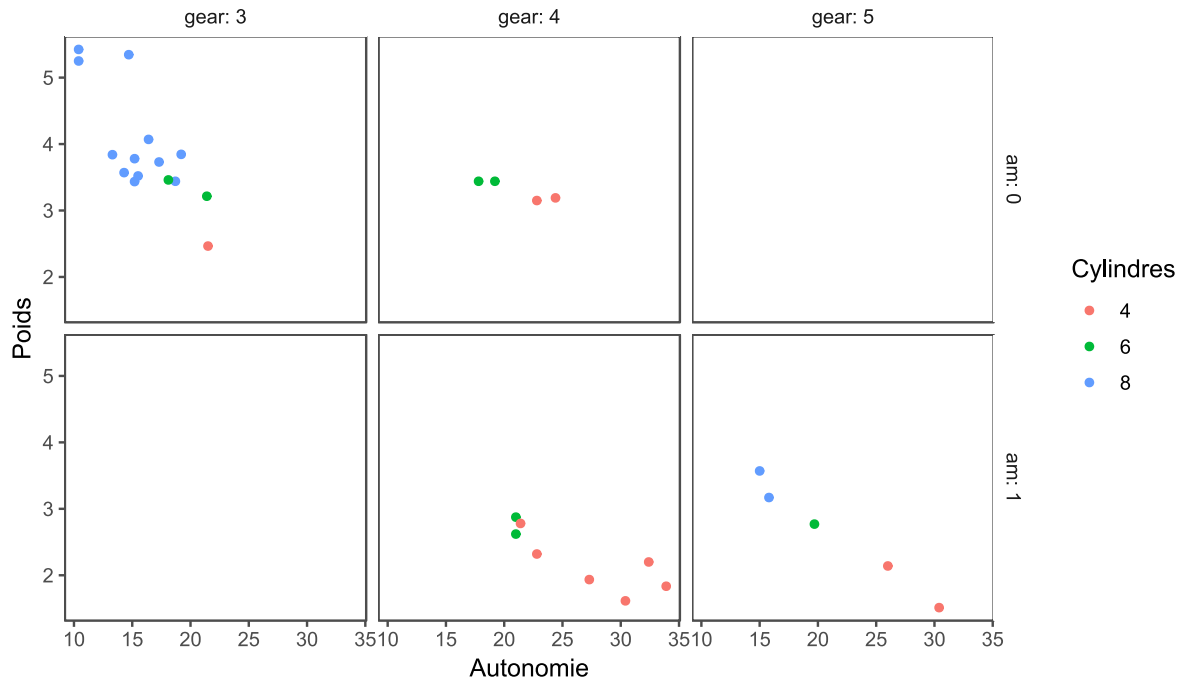


# Thèmes : exemples

Package  [ggthemes](#) :

- `theme_few()`

Autonomie vs poids

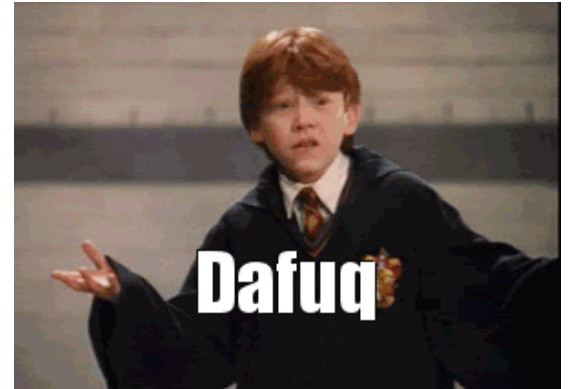
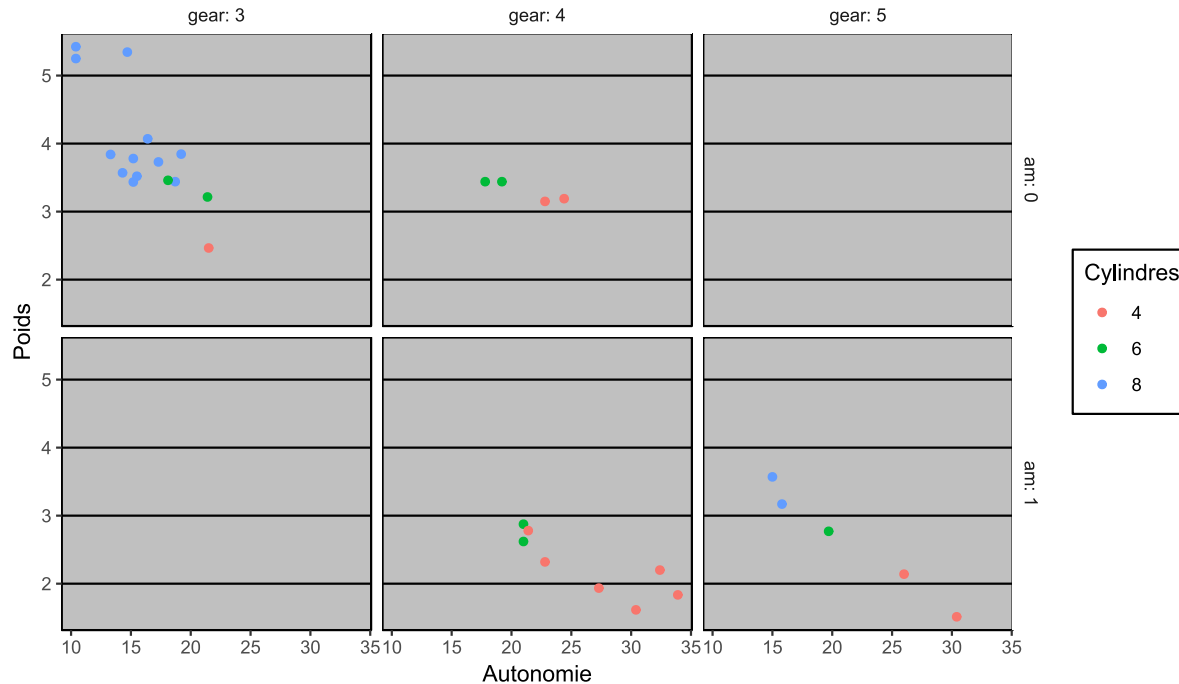


# Thèmes : exemples

Package  [ggthemes](#) :

- `theme_excel()`

Autonomie vs poids



# Thèmes : composition

## Construction

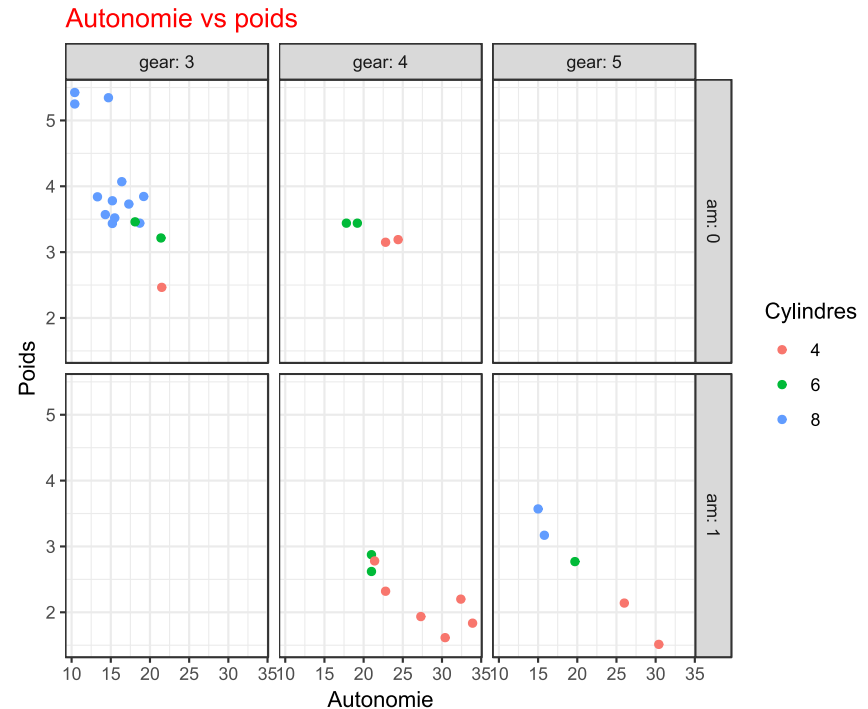
Les thèmes sont composables :

```
my_theme <- theme_bw() +  
  theme(plot.title =  
    element_text(colour = 'red'))  
  
p + my_theme
```

## Réglage global

Pour changer toutes les graphiques :

```
ggplot2::theme_set(my_theme)
```



# Sauvegarde

Facile : fonction `ggsave`

Dernier plot (ou paramètre `plot = ...`)

```
ggsave('myfile.pdf', width = 8, height = 5, units = 'cm')  
ggsave('myfile.eps', width = 8, height = 5, units = 'cm')  
ggsave('myfile.png', width = 8, height = 5, units = 'cm', dpi = 300)
```

Cairo PDF (en cas de polices de caractères personnalisées) :

```
ggsave('myfile.pdf', device = cairo_pdf)
```

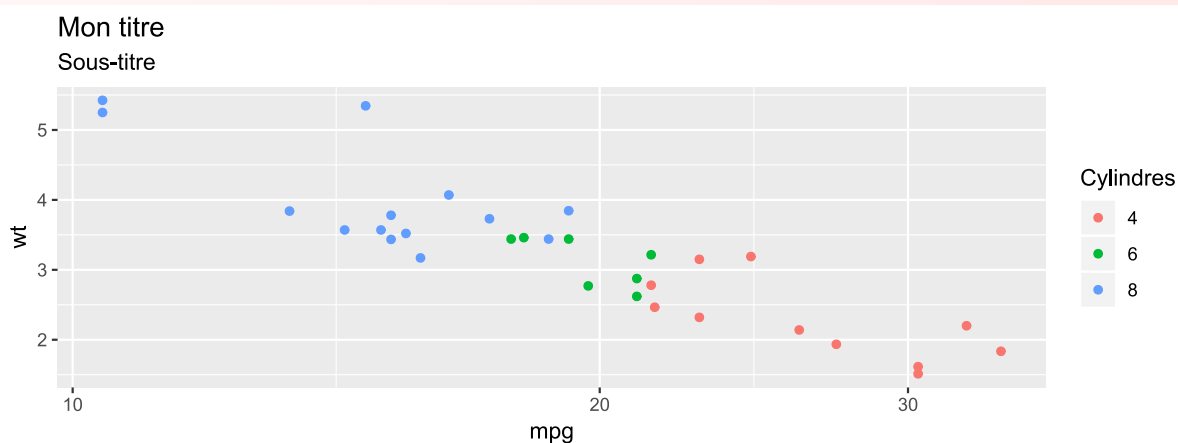
# Composition d'éléments

Pour éviter de répéter des éléments chaque fois :

- objet `ggplot` + objet `list`

```
my_opts <- list(  
  scale_x_log10(),  
  scale_color_discrete('Cylindres'),  
  ggtitle('Mon titre', subtitle = 'Sous-titre')  
)
```

```
ggplot(mtcars, aes(x = mpg, y = wt, col = factor(cyl))) +  
  geom_point() +  
  my_opts
```

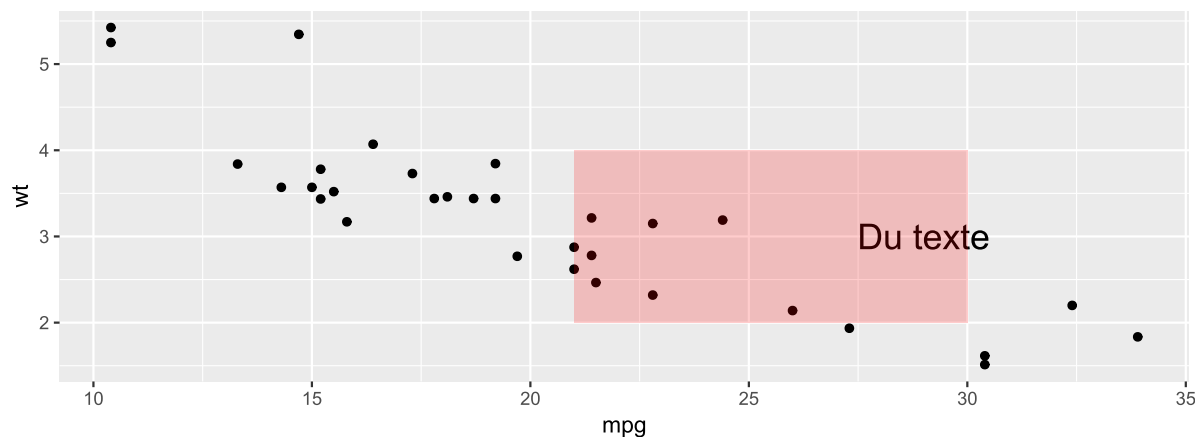


# Annotations


Vous pouvez ajouter des annotations où vous voulez :

- couche `annotate(geom = ...)`
- géométrie à choix, unités du graphique
- pas de lien avec des variables !

```
ggplot(mtcars) +  
  geom_point(aes(x = mpg, y = wt)) +  
  annotate('text', x = 29, y = 3, label = 'Du texte', size = 6) +  
  annotate("rect", xmin = 21, xmax = 30, ymin = 2, ymax = 4,  
    alpha = .2, fill = 'red')
```



# Géométries : paramètre `data`

`data` : un `data.frame`, mais aussi une fonction ( documentation !):

A function will be called with a single argument, the plot data.  
The return value must be a `data.frame`, and will be used as the layer data.

## Ou : comment étiqueter les outliers

Outliers : automobiles trop lourdes

Préliminaires : noms des automobiles dans une colonne ( [tidyr](#))

```
mtcars_with_models <- mtcars %>%  
  rownames_to_column('model')  
  
p <- mtcars_with_models %>%  
  ggplot(aes(x = mpg, y = wt)) +  
    geom_point(aes(col =  
      factor(cyl)))
```

# Ou : comment étiqueter les outliers

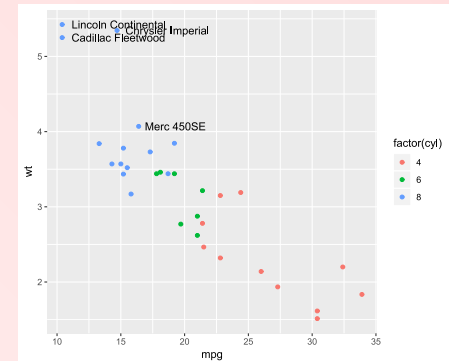


# Ou : comment étiqueter les outliers

## Approche 1 : une étiquette vide

```
heavy_cars <- quantile(mtcars$wt, 0.9)

p <- mtcars_with_models %>%
  ggplot(aes(x = mpg, y = wt)) +
    geom_point(aes(col = factor(cyl))) +
    geom_text(aes(label =
      ifelse(wt > heavy_cars, model, ''),
      hjust = -0.1, vjust = 0.5)
```



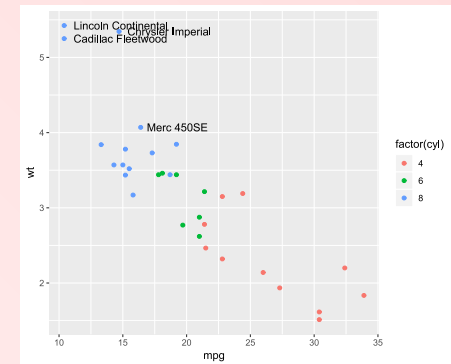
# Ou : comment étiqueter les outliers

## Approche 2 : filtrage du **data**

Répétition de **data** (sans ne pouvoir rien ajouter avant **ggplot** !)

```
heavy_cars <- quantile(mtcars$wt, 0.9)

p <- mtcars_with_models %>%
  ggplot(aes(x = mpg, y = wt)) +
    geom_point(aes(col = factor(cyl))) +
    geom_text(aes(label = model),
              hjust = -0.1, vjust = 0.5,
              data = filter(mtcars_with_models,
                            wt > heavy_cars)
    )
```



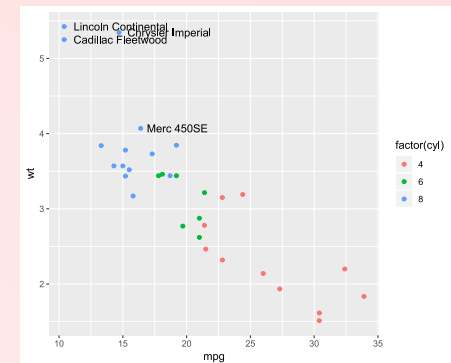
# Ou : comment étiqueter les outliers

## Approche 3 : pipe

Dans une pipe, `.` indique le premier argument (le résultat avant `ggplot`).<sup>1</sup>

```
heavy_cars <- quantile(mtcars$wt, 0.9)

p <- mtcars_with_models %>%
  ggplot(aes(x = mpg, y = wt)) +
  geom_point(aes(col = factor(cyl))) +
  geom_text(aes(label = model),
    hjust = -0.1, vjust = 0.5,
    data = . %>% filter(wt > heavy_cars)
  )
```



[1] Mon approche préférée !

# Ou : comment étiqueter les outliers

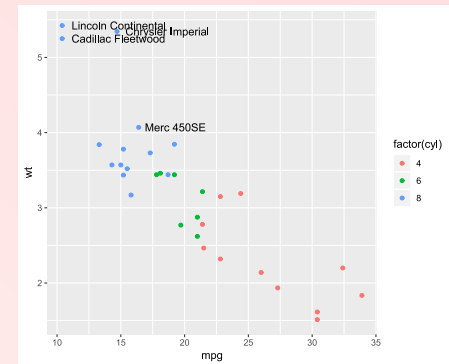
## Approche 4 : une fonction

J'utilise la caractéristique du paramètre ! <sup>1</sup>

```
heavy_cars <- quantile(mtcars$wt, 0.9)

my_filter <- function(x) {
  filter(x, wt > heavy_cars)
}

p <- mtcars_with_models %>%
  ggplot(aes(x = mpg, y = wt)) +
    geom_point(aes(col = factor(cyl))) +
    geom_text(aes(label = model),
              hjust = -0.1, vjust = 0.5,
              data = my_filter
    )
```



[1] Pourquoi est-ce que `my_filter` arrive à bien trouver `heavy_cars` ? ( difficile ! )  
La réponse est [ici](#)...

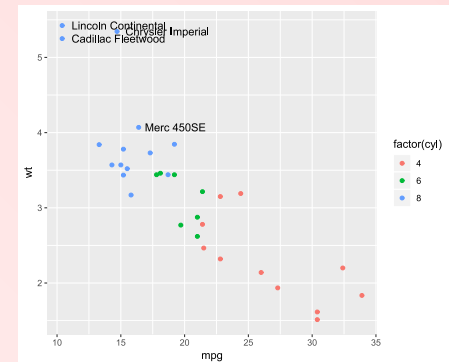
# Ou : comment étiqueter les outliers

## Approche 5 : une fonction qui retourne une fonction...

Pour conditions multiples<sup>1</sup> ! Quasiquotation<sup>2</sup>!

```
filter_df <- function(...) {
  condition <- enquos(...)
  function(df) {
    df %>% filter(!!!condition)
  }
}

p <- mtcars_with_models %>%
  ggplot(aes(x = mpg, y = wt)) +
  geom_point(aes(col = factor(cyl))) +
  geom_text(aes(label = model),
    hjust = -0.1, vjust = 0.5,
    data = filter_df(wt > quantile(wt, 0.9))
  )
```



[1] Ça permet d'enchaîner plusieurs conditions à la `filter` : p.ex.

```
filter_df(wt > quantile(wt, 0.9), str_detect(model, 'Audi'))
```


[2] Encore plus difficile ! Explications [ici](#)...

# Extensions


# Ce qui reste

 [gridExtra](#) : coller des graphiques différents dans le même graphique

- in base R : `par(mfrow=...)`
- [vignette](#)

 [ggRidges](#) : ridgeline plots

- [vignette](#)

 [gghighlight](#) : mise en évidence d'éléments


- [vignette](#)

 [ggraph](#) : **grammaire des graphs**

- Deux `data.frames` : nœuds et liens...

# plotly

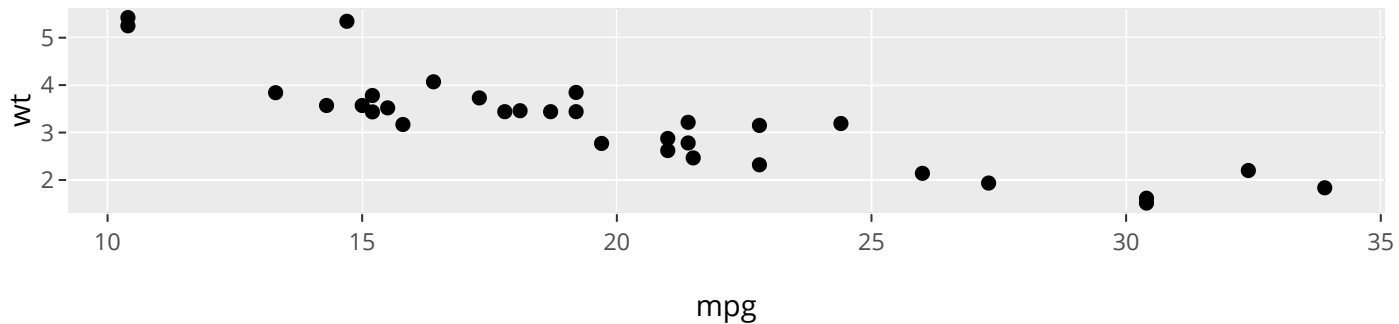
ADVANCED!

- Package  [plotly](#) : librairie similaire à `ggplot2`, mais interactive (touchez les slides !)
- traducteur de `ggplot2` à `plotly` : fonction `ggplotly`

```
library(plotly)
```

```
pl <- ggplot(mtcars, aes(x = mpg, y = wt)) + geom_point()
```

```
ggplotly(pl)
```





# Animations

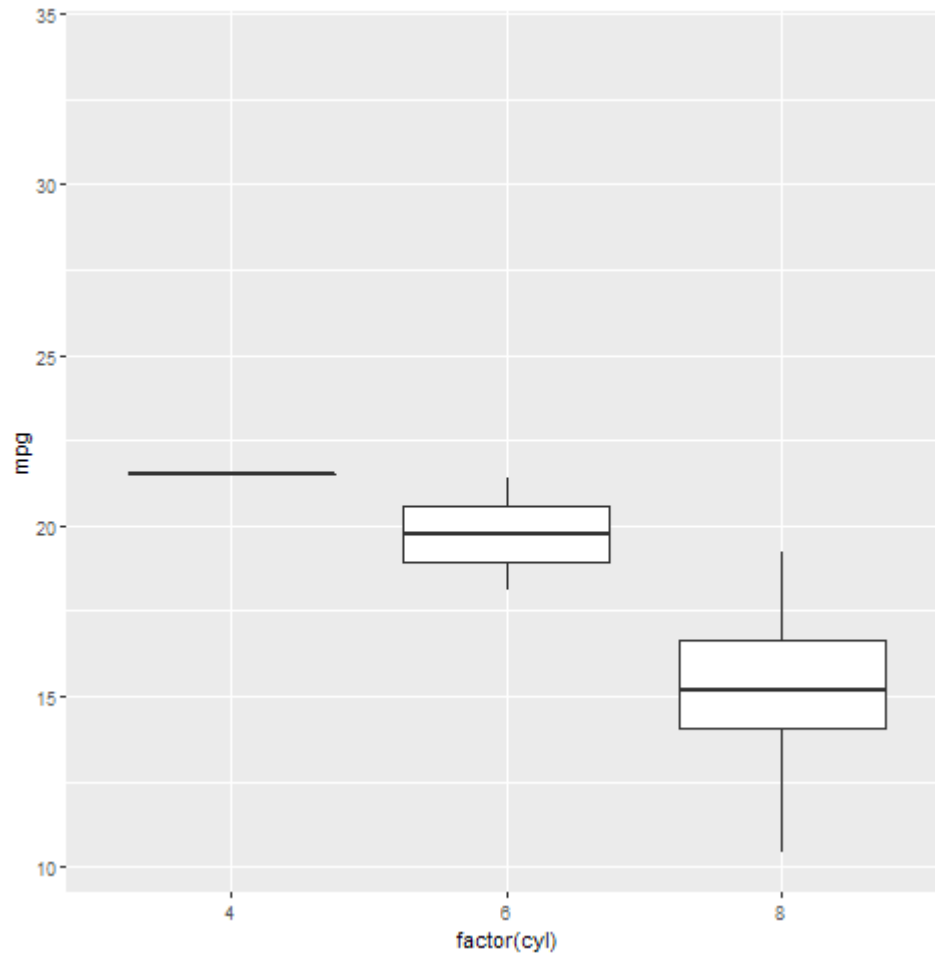
Package  [gganimate](#)

Vous ajoutez quelques lignes...

```
library(ggplot2)
library(gganimate)

p <- ggplot(mtcars, aes(factor(cyl), mpg)) +
  geom_boxplot() +
  transition_states(
    gear, transition_length = 2, state_length = 1
  ) +
  enter_fade() +
  exit_shrink() +
  ease_aes('sine-in-out')
```

# Animations



# Cartographie

Approche à la **tidyverse** : package  [sf](#)



```
library(sf)

# Shapefile des cantons
shp <- sf::st_read('data/switzerland_shapefiles/gadm36_CHE_1.shp', quiet = TRUE)

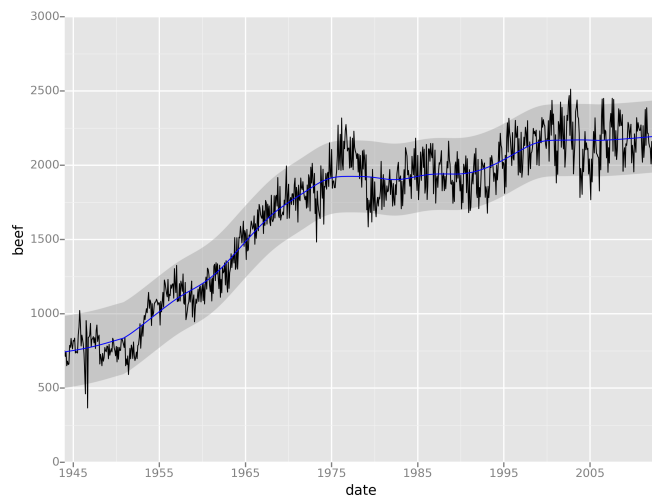
ggplot(shp) +
  geom_sf(aes(fill = (NAME_1 == 'Ticino')))) +
  geom_sf_text(aes(label = NAME_1), size = 3) +
  theme_bw() +
  scale_fill_discrete('', breaks = c(TRUE, FALSE), labels = c('Rigole', 'Travaille'))
```

# ggplot2 in Python

Diverses implémentations de la grammaire des graphiques :

-  [plotnine](#)
-  [ggplot](#)

```
from ggplot import *  
  
ggplot(aes(x='date', y='beef'), data=meat) +\  
  geom_line() +\  
  stat_smooth(colour='blue', span=0.2)
```



Pour aller plus loin...

# Pour aller plus loin...

## Galerie des extensions

- [ggplot gallery](https://www.ggplot2-exts.org/gallery/) : <https://www.ggplot2-exts.org/gallery/>

## Conseils :


- suivez des blogs
  - [RStudio Blog](#)
  - [R-bloggers](#)
  - [Timo Grossenbacher \(SRF Data\)](#) (DE)
  - [SRF Data](#) (DE)
- copiez les exemples de GitHub
- expérimentez !

## Courses de visualisation des données


- [Fundamentals of Data Visualization](#) (théorie)
- [BBC data journalism cookbook](#) (que `ggplot2`)
- [Edward Tufte](#) : vieux (et exagéré ? )

# Ces slides...

## R Markdown

- **Markdown** : langage simple pour rédiger du texte formaté
- GitHub, blogs...
- **R Markdown** = Markdown avec code intégré pour générer
  - pages web, blogs, articles, présentations, interfaces, livres, ...
- [Galerie](#)
- [Guide](#) (faite avec R Markdown +  [blogdown](#))

## Cette présentation

-  [xaringan](#) basé sur la librairie [remark.js](#)
- Mélange : R Markdown + CSS (quelques détails)
- [Documentation](#) (c'est une présentation!)
- [Sources](#)