

# Visualisation de données avec R

par Jonathan EL METHNI

Ce document présente un certain nombre de commandes concernant des packages **R** utiles pour faire de la visualisation de données. On commence tout d'abord par les installer puis par les charger :

```
library(ggplot2)      # Le package de référence pour faire de jolis graphiques
library(GGally)        # Un add-on au package ggplot2
library(tidyverse)      # Améliore certaines fonctionnalités de ggplot2
library(scales)         # Pour travailler avec différentes échelles
library(RColorBrewer)   # Pour avoir de jolies palettes de couleurs cohérentes
library(MASS)           # Pour travailler sur les données cats
```

Le but de la visualisation de données étant de représenter graphiquement des données brutes (ou quasi-brutes), il est souvent nécessaire de prendre en compte plusieurs variables. Nous devons donc aller plus loin que les graphiques de base (nuage de points, droite de régression, boîte à moustache, diagramme en barres ou circulaires, ...), pour les combiner par exemple.

Avec le langage de base (package **graphics** notamment), il est possible de faire des graphiques évolués, mais avec beaucoup de paramétrages *à la main*. Pour cela on va commencer par s'intéresser au jeu de données **mtcars**.

```
mtcars
```

```
##                                     mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## Cadillac Fleetwood 10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Honda Civic          30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
## Toyota Corona        21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## AMC Javelin          15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Fiat X1-9            27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Porsche 914-2        26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Lotus Europa          30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Ford Pantera L       15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
```

```

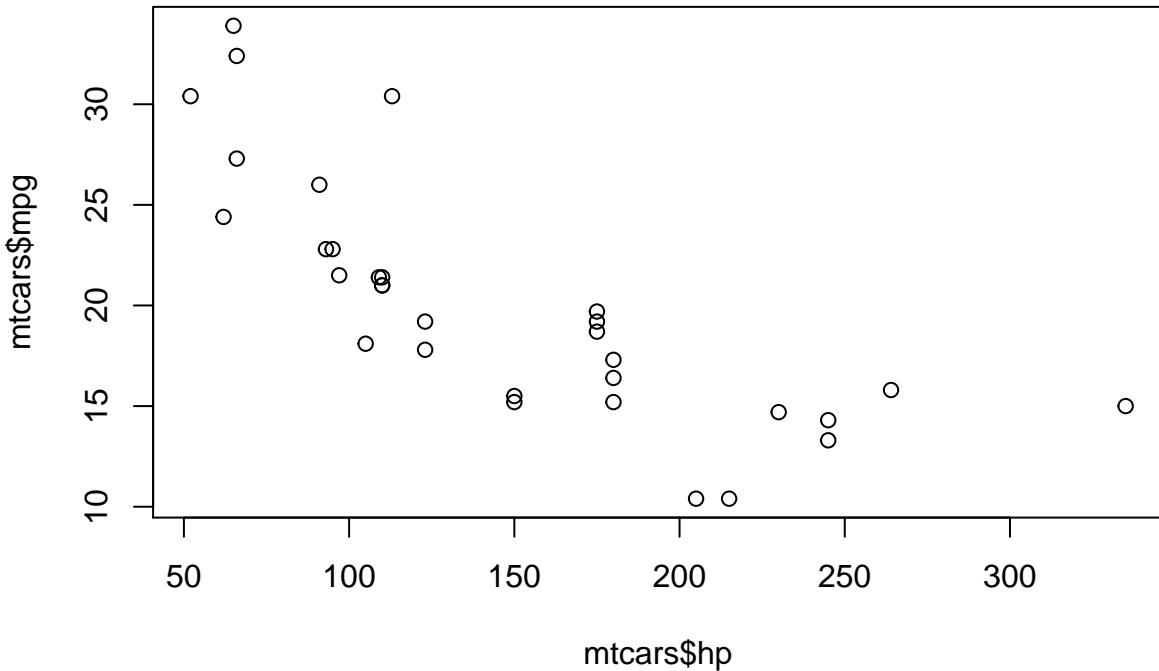
## Ferrari Dino      19.7   6 145.0 175 3.62 2.770 15.50 0 1 5 6
## Maserati Bora    15.0   8 301.0 335 3.54 3.570 14.60 0 1 5 8
## Volvo 142E        21.4   4 121.0 109 4.11 2.780 18.60 1 1 4 2

# ?mtcars ou help(mtcars)
str(mtcars)

## 'data.frame':   32 obs. of  11 variables:
## $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
## $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
## $ disp: num  160 160 108 258 360 ...
## $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
## $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
## $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
## $ qsec: num  16.5 17 18.6 19.4 17 ...
## $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
## $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
## $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
## $ carb: num  4 4 1 1 2 1 4 2 2 4 ...

plot(mtcars$hp, mtcars$mpg)

```



Voici ici un exemple de graphique avancé, représentant 4 variables (trois quantitatives et une qualitative), ainsi qu'une explication succincte des différentes fonctions utilisées, et quelques paramètres de celles-ci.

```

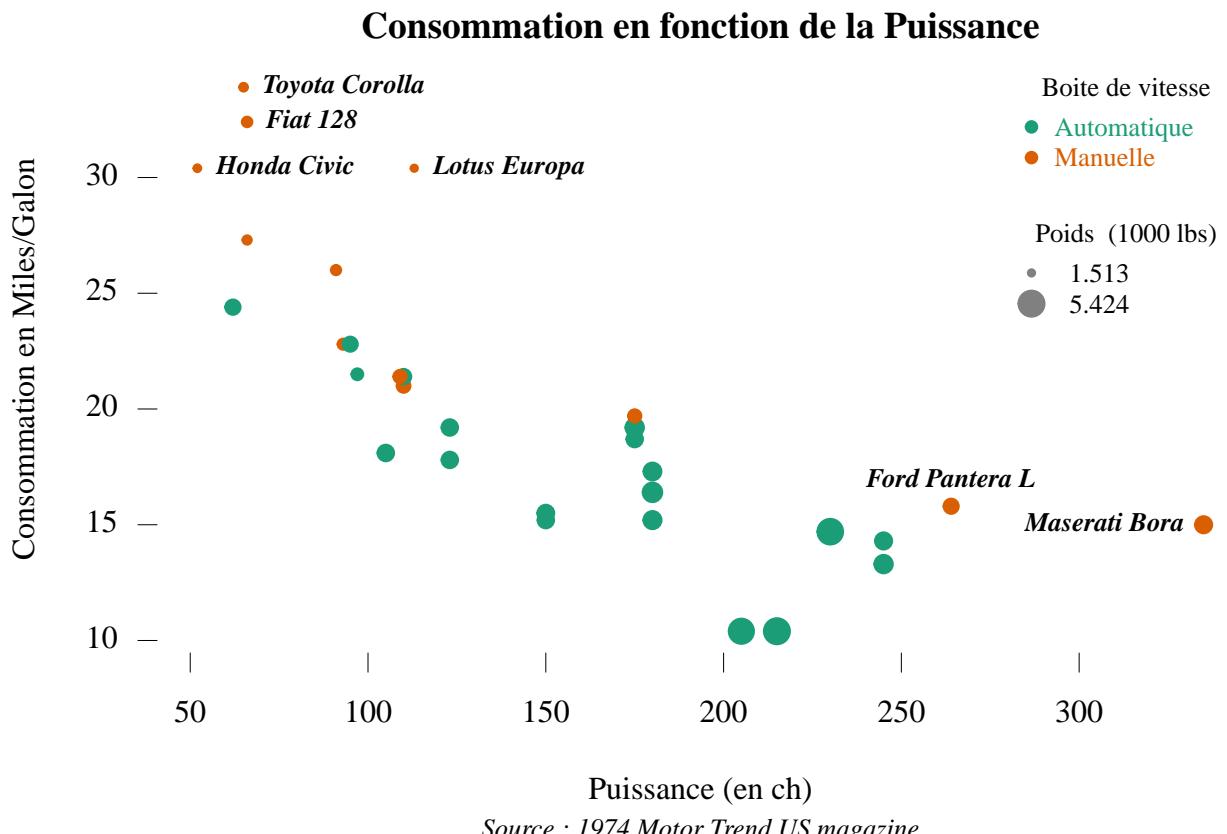
par(family = "serif", mar = c(5, 4, 2, 0)+.1)
couleurs_am = brewer.pal(3, "Dark2")
plot(mpg ~ hp,
      data = mtcars,
      pch = 19,
      cex = wt/3,
      col = couleurs_am[mtcars$am+1],
      main = "Consommation en fonction de la Puissance",
      sub = "Source : 1974 Motor Trend US magazine", font.sub = 3, cex.sub = .8,

```

```

xlab = "Puissance (en ch)",
ylab = "Consommation en Miles/Galon",
bty = "n", axes = FALSE)
axis(1, lwd = 0, lwd.ticks = .5)
at.y = axis(2, lwd = 0, lwd.ticks = .5, labels = FALSE)
text(y = at.y, x = 35, labels = at.y, srt = 0, pos = 2, xpd = TRUE)
l1 = legend("topright", legend = c("Automatique", "Manuelle"),
            col = couleurs_am, bty = "n", cex = .8, pch = 19,
            text.width = 50, text.col = couleurs_am,
            title = "Boite de vitesse", title.col = "black")
legend(l1$rect$left, l1$rect$top-l1$rect$h-1,
       legend = range(mtcars$wt), title = "Poids (1000 lbs)",
       pch = 19, pt.cex = range(mtcars$wt)/3, bty = "n", cex = .8, col = gray(.5),
       text.width = 50, adj = -.25)
outliers_hp = subset(mtcars, subset = hp > 250)
text(outliers_hp$hp, outliers_hp$mpg, row.names(outliers_hp), pos = c(3, 2), cex = .8, font = 4)
outliers_mpg = subset(mtcars, subset = mpg > 30)
text(outliers_mpg$hp, outliers_mpg$mpg, row.names(outliers_mpg), pos = 4, cex = .8, font = 4)

```



- La fonction `par()` permet de modifier les paramètres graphiques, tel que :
  - `mar` pour les marges (un vecteur numérique de taille 4 qui permet de définir les tailles des marges dans l'ordre suivant : bas, gauche, haut, droite. Par défaut il vaut `c(5.1, 4.1, 4.1, 2.1)`)
  - `family` pour la famille de police d'écriture
  - certains paramètres ci-après sont définissables globalement dans la fonction `par()`, ou localement dans les fonctions suivantes
- Les paramètres de la fonction `plot()`
  - `pch` : symbole utilisé pour chaque point (ici 19 indique un rond plein)

- **cex** : taille du point (ici en fonction de la variable **wt**)
- **col** : couleur des points (ici en fonction de la variable **am**)
- **main, sub, xlab, ylab** : resp. titre, sous-titre, intitulé en abscisse et intitulé en ordonnée
- **\*.sub** : indication spécifique pour le sous-titre
- **bty** : type de la boîte (ici n veut dire rien autour du graphique)
- **axes** : présence ou non des axes (non ici)
- La fonction **axis()** qui permet d'ajouter un axe (1 : en abscisse, et 2 : en ordonnée)
  - si rien d'indiqué pour **at**, utilisation des **ticks** par défaut
  - **lwd** : largeur de la ligne (ici non-présente)
  - **lwd.ticks** : largeur des ticks (assez fine ici)
  - renvoie les valeurs des ticks
- La fonction **text()** permet d'écrire du texte sur le graphique
  - le **x = 35** est choisi par expérience
  - **srt** : angle du texte
  - **pos** : ajustement du texte par rapport aux coordonnées indiquées
  - **xpd** : permet d'écrire en dehors du graphique (dans les marges donc)
- La fonction **legend()** permet d'ajouter une légende
  - position : soit  $(x, y)$ , soit chaîne spécifique
  - **legend** : texte des items
  - **col, bty, cex, pch** : identique à précédemment
  - **text.width** et **text.col** : largeur du texte des items et couleur(s)
  - **title** et **title.col** : titre de la légende (et couleur du titre)
  - renvoie une liste avec deux objets (**rect** qui contient des infos sur le rectangle occupé par la légende dans le graphique et **text** qui contient les coordonnées du texte des items)
  - **pt.cex** : taille des symboles
  - **adj**: ajustement du texte

## Librairie `ggplot2`

Ce package reproduit la grammaire des graphiques (cf *Grammar of Graphics*, Leland Wilkinson), avec le même formalisme. Vous pouvez trouver plus d'informations sur le site officiel et la documentation. Cet article permet de bien comprendre la philosophie du package et de la grammaire.

Le principe de cette grammaire est qu'un graphique est composé de couches :

- les **données** à représenter qui doivent être dans un dataframe !!! Si ce n'est pas le cas vous pouvez toujours le faire avec l'instruction **as.data.frame**.
- A partir de ces données nous allons définir des attributs **esthétiques** (soit identiques pour tous, soit fonction d'une des variables) :
  - les axes *x* et *y*,
  - la couleur : **color**
  - la taille : **size**
  - la transparence : **alpha**
  - le remplissage : **fill**
  - la forme : **shape**
- les attributs **géométriques** (point, droite, ...)
- les transformations **statistiques** (dénombrement, ajustement, ...)
- les **échelles**
- le système de **coordonnées** (linéaire, logarithmique, polaire, ...)
- le découpage (ou non) en **facettes**

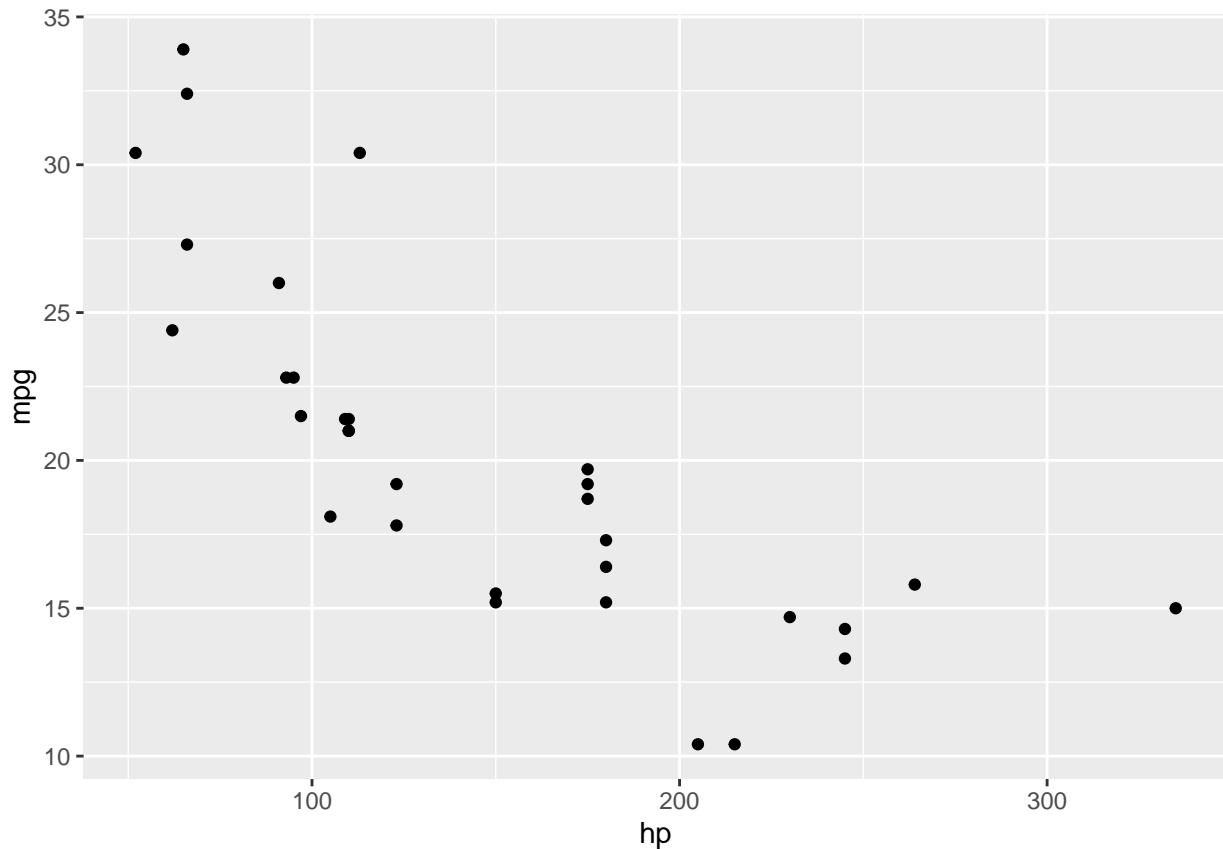
Dans cette librairie, il y a deux fonctions principales :

- `qplot` (ou `quickplot`) permettant de faire des graphiques rapidement.
- `ggplot` permettant d'initialiser un graphique auquel on va ajouter des couches successives.

### Fonction `qplot()`

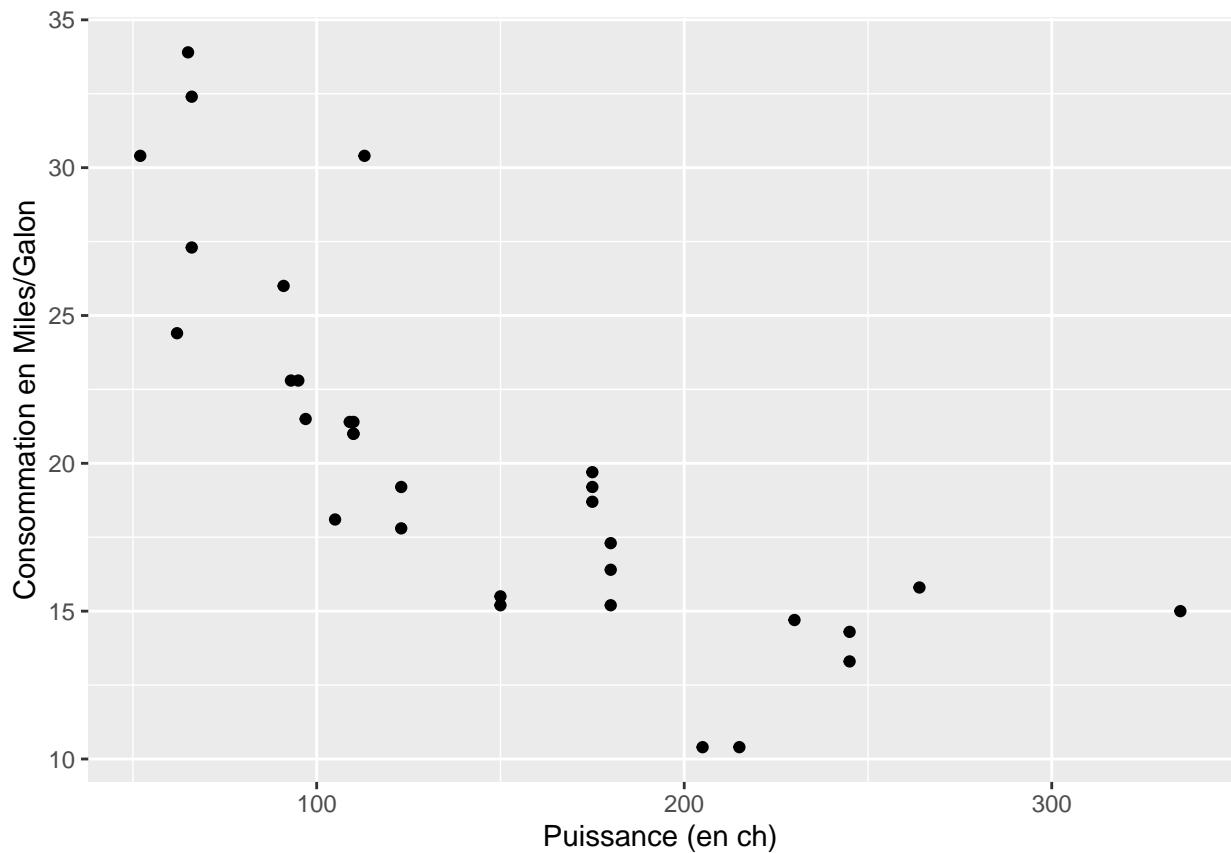
Traçons le même nuage de points que précédemment.

```
qplot(hp,mpg,data=mtcars)
```



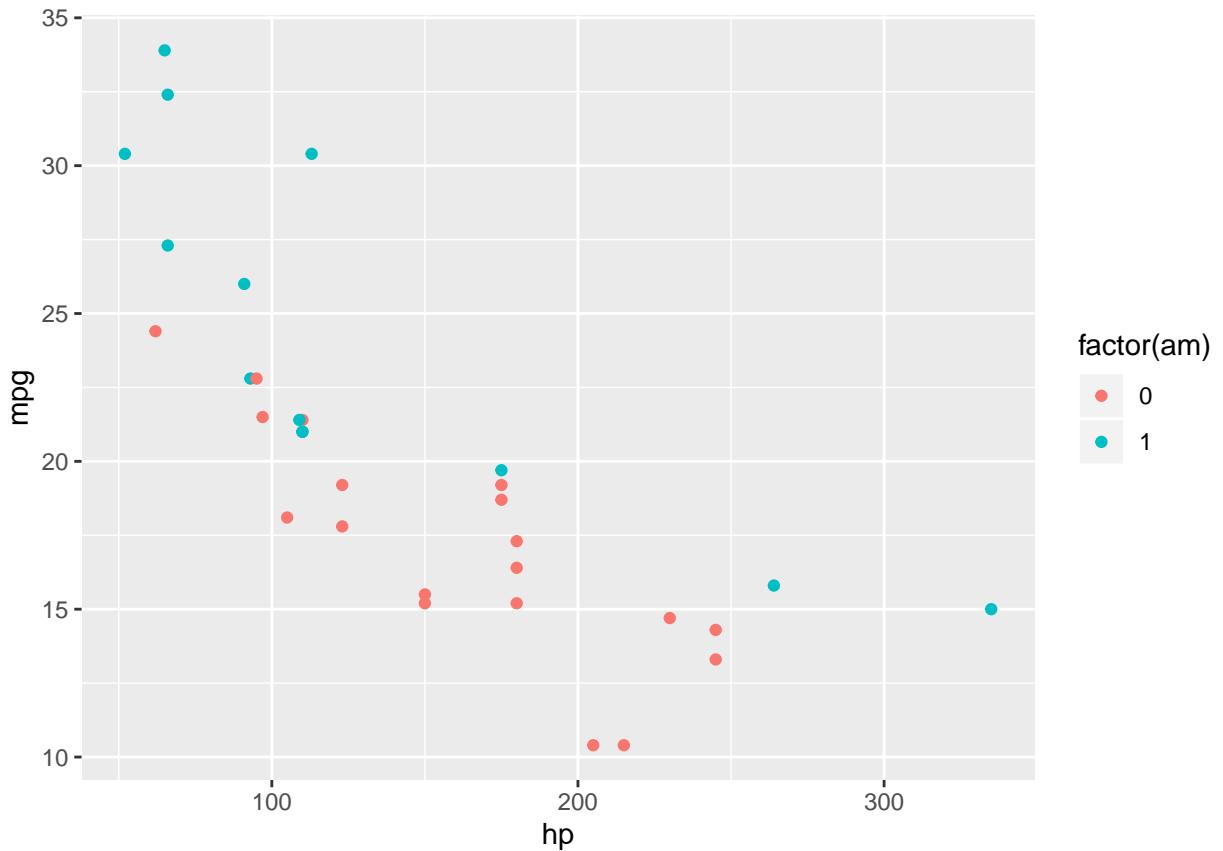
On peut y ajouter des noms pour l'axes des abscisses et celui des ordonnées.

```
p=qplot(hp,mpg,data=mtcars)  
p+xlab("Puissance (en ch)")+ylab("Consommation en Miles/Galon")
```



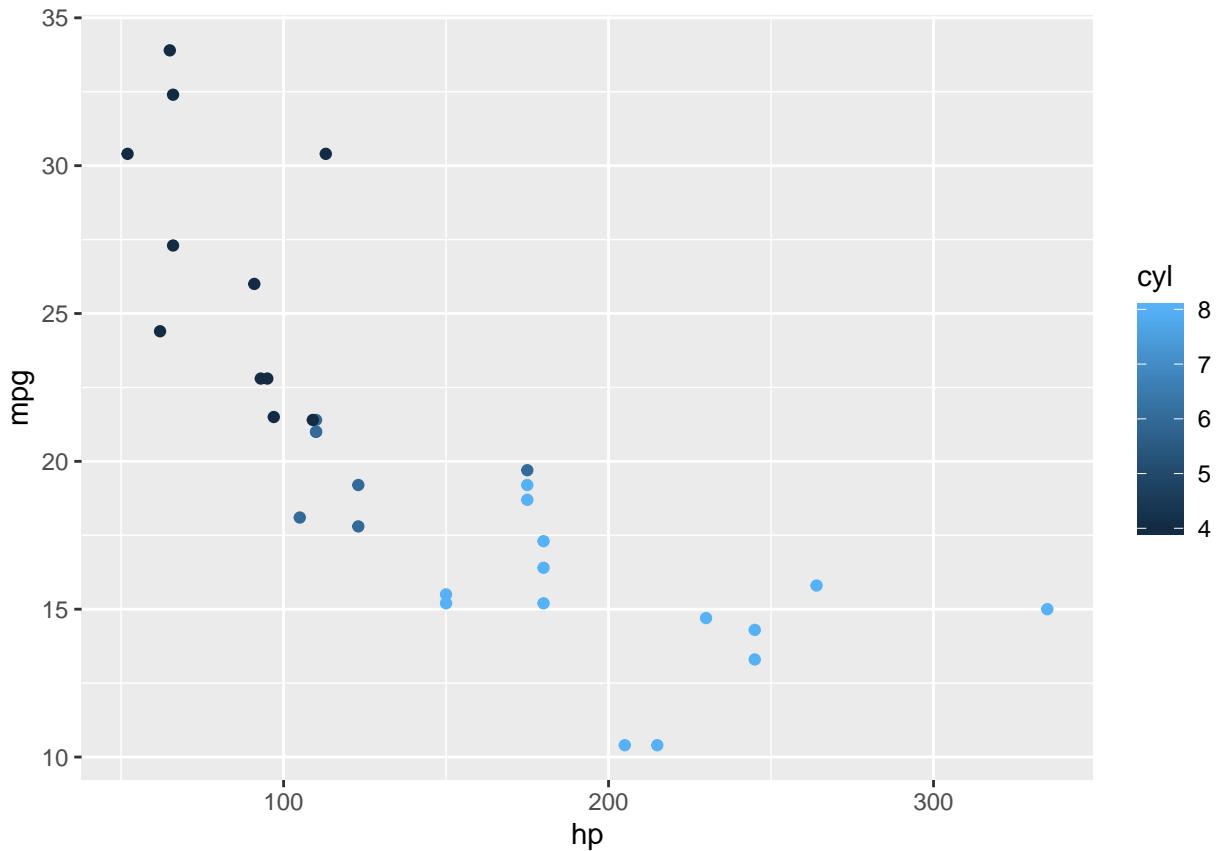
Il est possible de distinguer par des couleurs les voitures disposant d'une boîte manuelles de celles automatiques.

```
qplot(hp,mpg,data=mtcars,colour = factor(am))
```



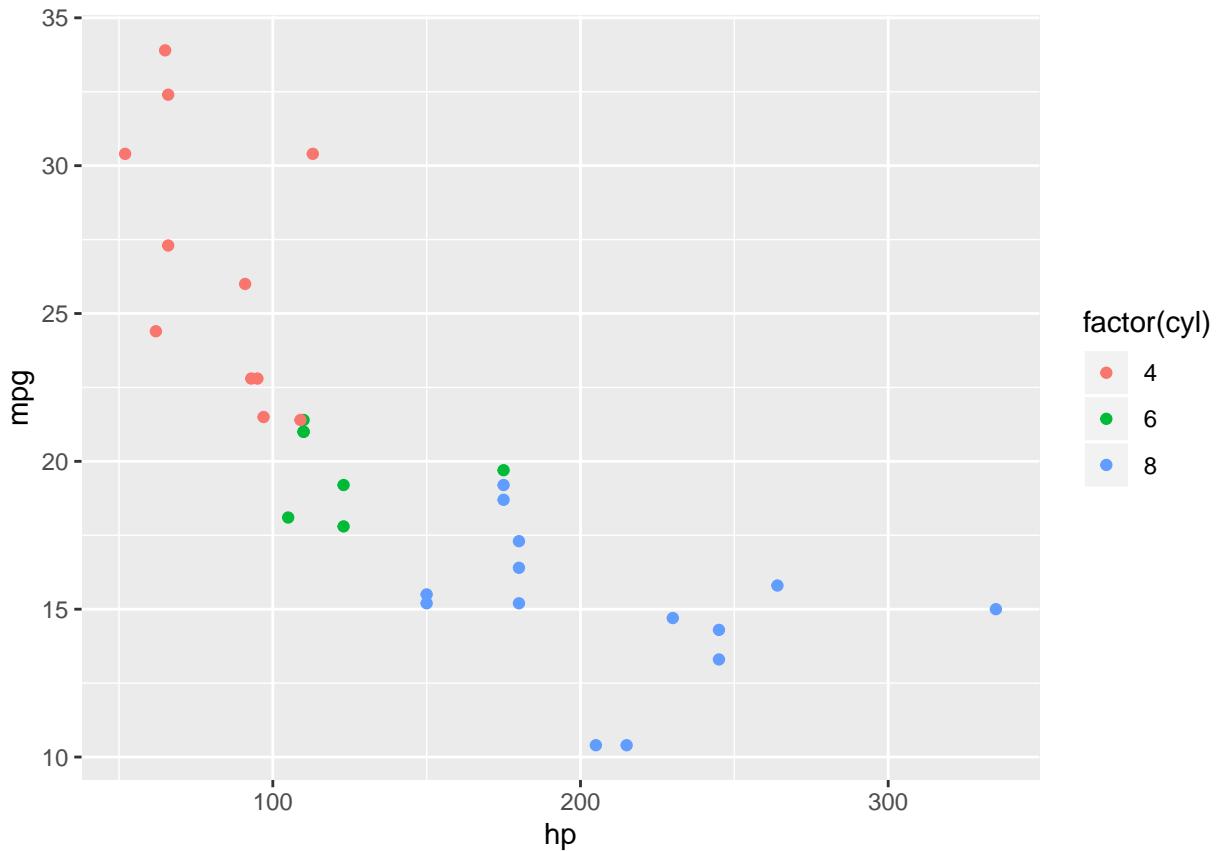
On peut mettre en avant la cylindrée des voitures (ici considérée comme une variable quantitative) par une échelle de couleur.

```
qplot(hp,mpg, data = mtcars, colour = cyl)
```



Ou le faire par un nombre de couleurs correspondant au nombre de modalités de la variable cyl (ici considérée comme une variable qualitative).

```
qplot(hp,mpg,data=mtcars,colour = factor(cyl))
```



Presque personne n'utilise la fonction `qplot` pour tracer un graphique car elle est plus difficile d'utilisation que `plot` et bien moins riche que la fonction à laquelle on va s'intéresser jusqu'à la fin du cours `ggplot`.

### Fonction `ggplot()`

La fonction `ggplot()` permet de faire plus de choses que `qplot()` mais nécessite un formalisme plus lourd, dont voici quelques détails :

- `ggplot()` crée un graphique (et le renvoie, i.e. on peut stocker un graphique dans une variable pour l'afficher plus tard, éventuellement en lui ajoutant des couches)
- `aes()` permet de définir les aspects esthétiques (`x` et `y` principalement, mais aussi `color`, `fill`, `size`, ...)
- `geom_xxx()` indique la représentation à choisir (`xxx` étant remplacé par `histogram`, `boxplot`, ...)
- `stat_xxx()` indique les transformations statistiques à utiliser, si besoin
- `scale_xxx()` s'emploie pour des changements d'échelle
- `coord_xxx()` s'utilise pour des modifications de systèmes de coordonnées
- `facet_xxx()` découpe les données (et donc le graphique) en plusieurs facettes selon les variables fournies dans la formule
- `theme_xxx()`, `labs()`, `xlab()`, `ylab()`, `ggtitle()`, ... pour des améliorations du graphique (annotations, couleurs, ...)

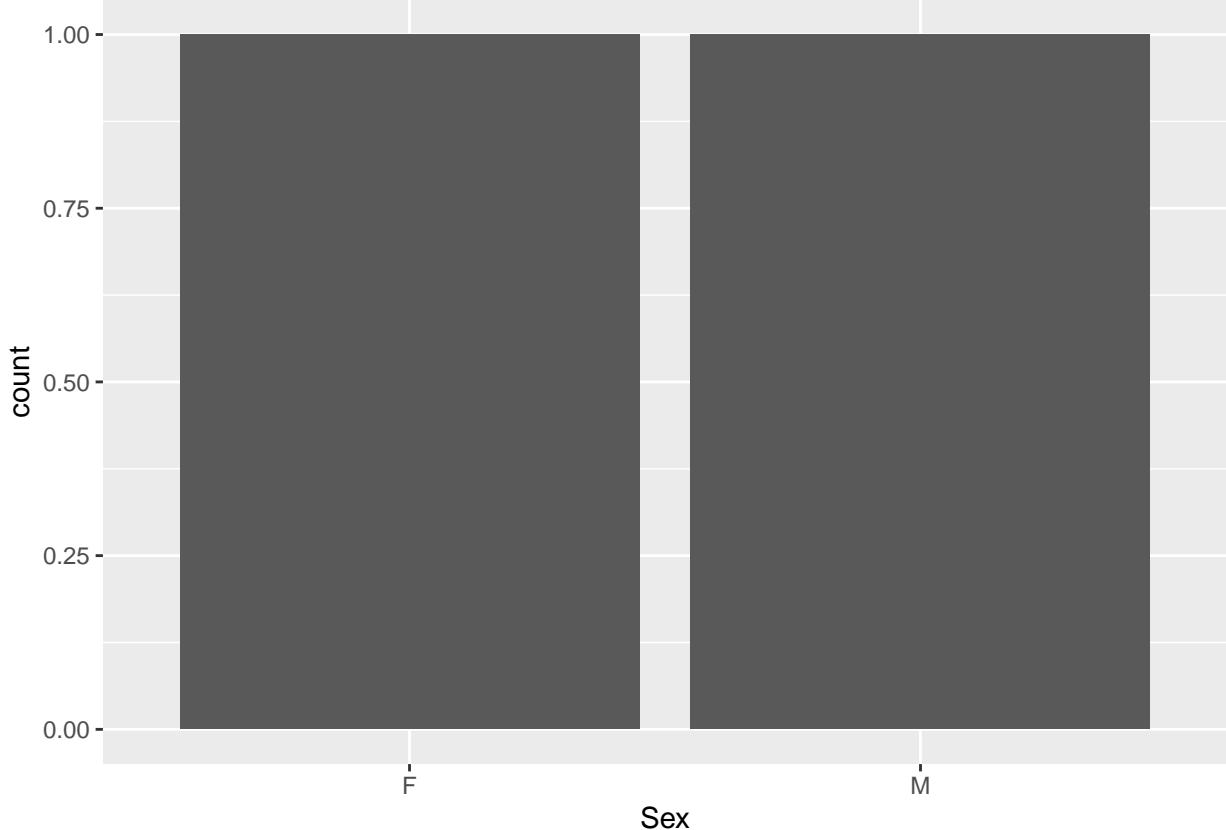
Hormis la fonction `aes()`, qui s'utilise à l'intérieur des autres, toutes ces fonctions peuvent s'ajouter pour compléter le graphique.

Le fonctionnement de la fonction `ggplot()` est donc particulier, mais une fois compris, il est facile de créer chaque graphique statistique que l'on souhaite.

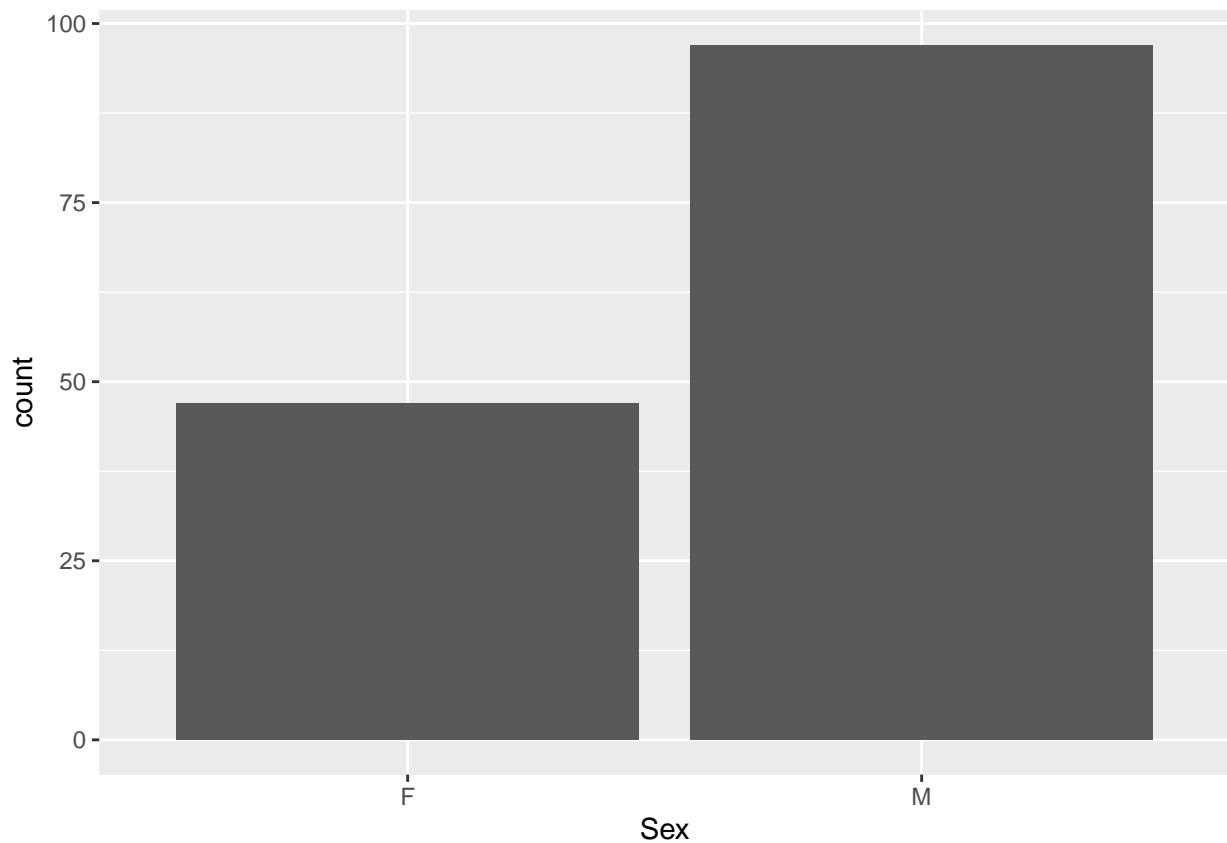
On verra au long de ce document qu'il est possible de modifier le positionnement de certains éléments graphiques :

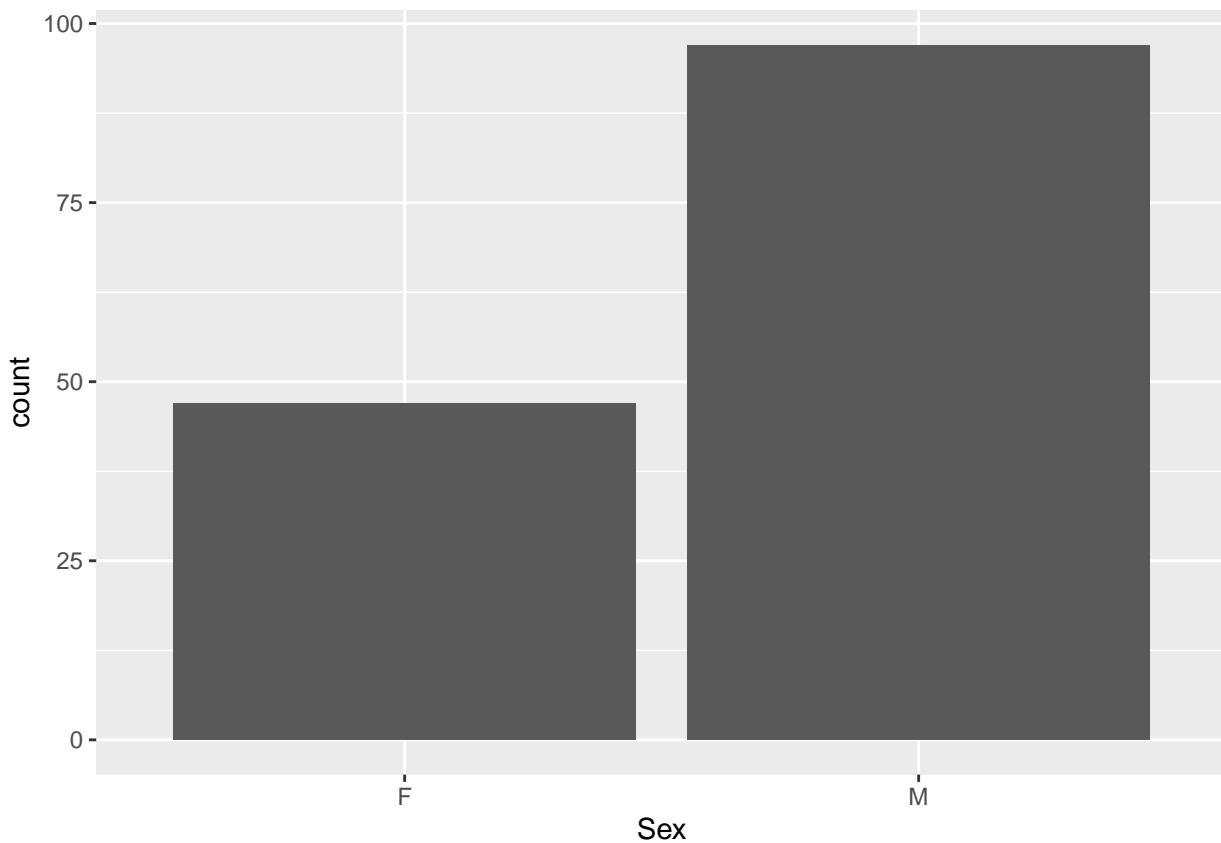
- `position="dodge"` : évite les chevauchements, place les éléments côte à côte
- `position="fill"` : empile les éléments qui se chevauchent , en normalisant pour avoir une hauteur égale
- `position="identity"` : n'ajuste pas la position
- `position="jitter"` : place les éléments côte en côte en essayant d'optimiser l'espace
- `position="stack"` : empile les éléments qui se chevauchent.

```
ggplot(data=cats, aes(x=Sex)) + geom_bar(position="fill")
```



```
ggplot(data=cats, aes(x=Sex)) + geom_bar(position="stack")
```





### Fonctionnement général

La suite présente l'utilisation de différentes fonctions à travers des exemples concrets, afin de comprendre la philosophie de cette grammaire des graphiques.

Pour cela on va s'intéresser au jeu de données `diamonds` du package `ggplot2` :

```
diamonds
```

```
## # A tibble: 53,940 x 10
##   carat cut      color clarity depth table price     x     y     z
##   <dbl> <ord>    <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23 Ideal     E     SI2     61.5    55   326  3.95  3.98  2.43
## 2 0.21 Premium   E     SI1     59.8    61   326  3.89  3.84  2.31
## 3 0.23 Good      E     VS1     56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium  I     VS2     62.4    58   334  4.2    4.23  2.63
## 5 0.31 Good      J     SI2     63.3    58   335  4.34  4.35  2.75
## 6 0.24 Very Good J     VVS2    62.8    57   336  3.94  3.96  2.48
## 7 0.24 Very Good I     VVS1    62.3    57   336  3.95  3.98  2.47
## 8 0.26 Very Good H     SI1     61.9    55   337  4.07  4.11  2.53
## 9 0.22 Fair       E     VS2     65.1    61   337  3.87  3.78  2.49
## 10 0.23 Very Good H     VS1     59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

```
# ?diamonds
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame': 53940 obs. of 10 variables:
## $ carat : num 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
```

```

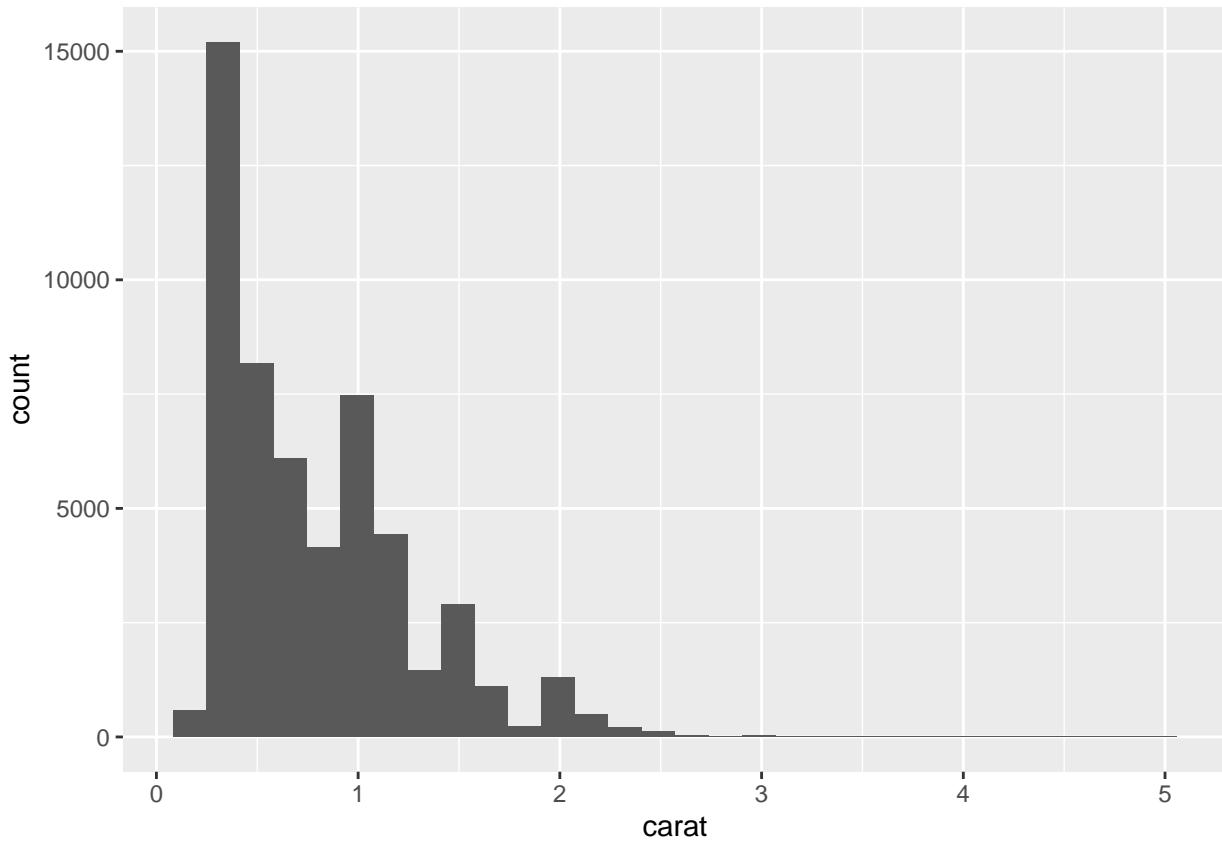
## $ cut      : Ord.factor w/ 5 levels "Fair"<"Good"<...: 5 4 2 4 2 3 3 3 1 3 ...
## $ color    : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<...: 2 2 2 6 7 7 6 5 2 5 ...
## $ clarity  : Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<...: 2 3 5 4 2 6 7 3 4 5 ...
## $ depth    : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
## $ table    : num  55 61 65 58 58 57 57 55 61 61 ...
## $ price    : int  326 326 327 334 335 336 336 337 337 338 ...
## $ x        : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
## $ y        : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
## $ z        : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...

```

## Histogramme

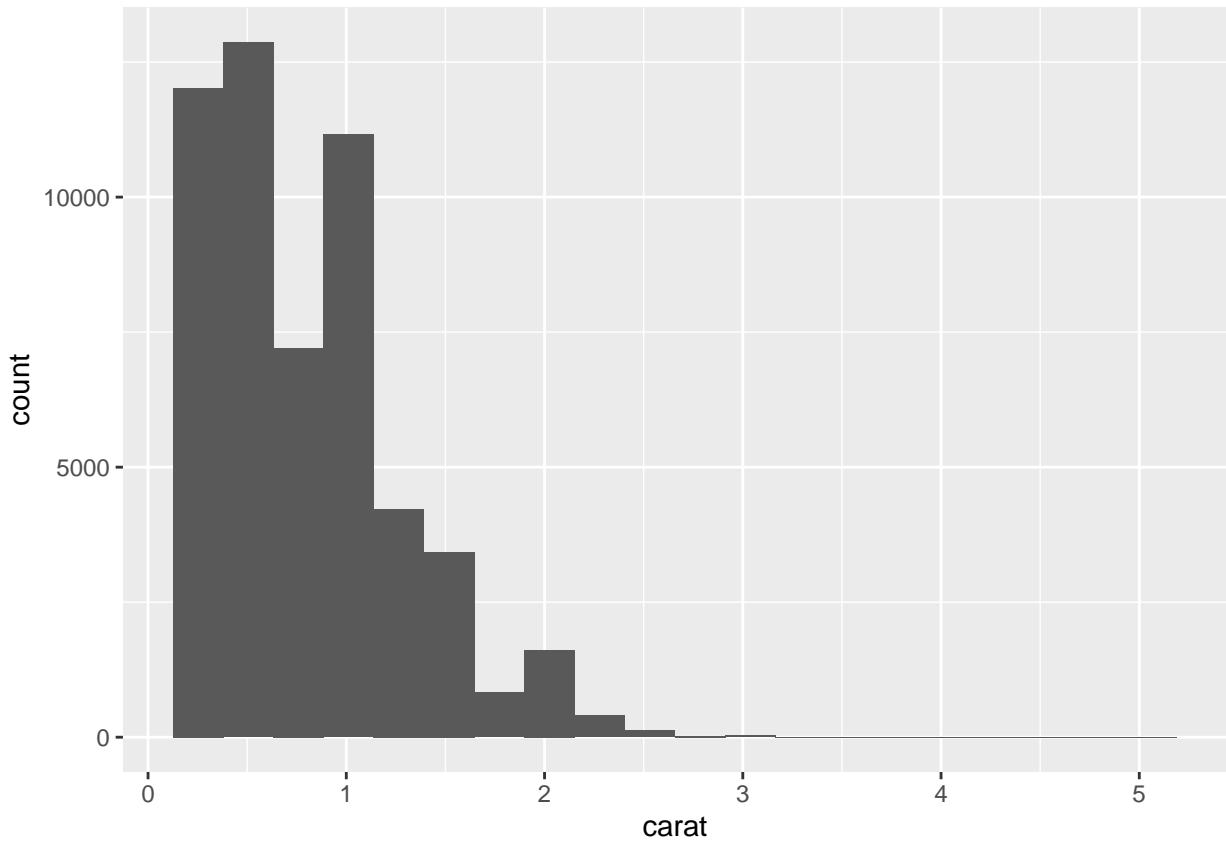
Par exemple, pour décrire une variable continue par un histogramme, nous pouvons utiliser le code suivant :

```
ggplot(data = diamonds, aes(x = carat)) +geom_histogram()
```



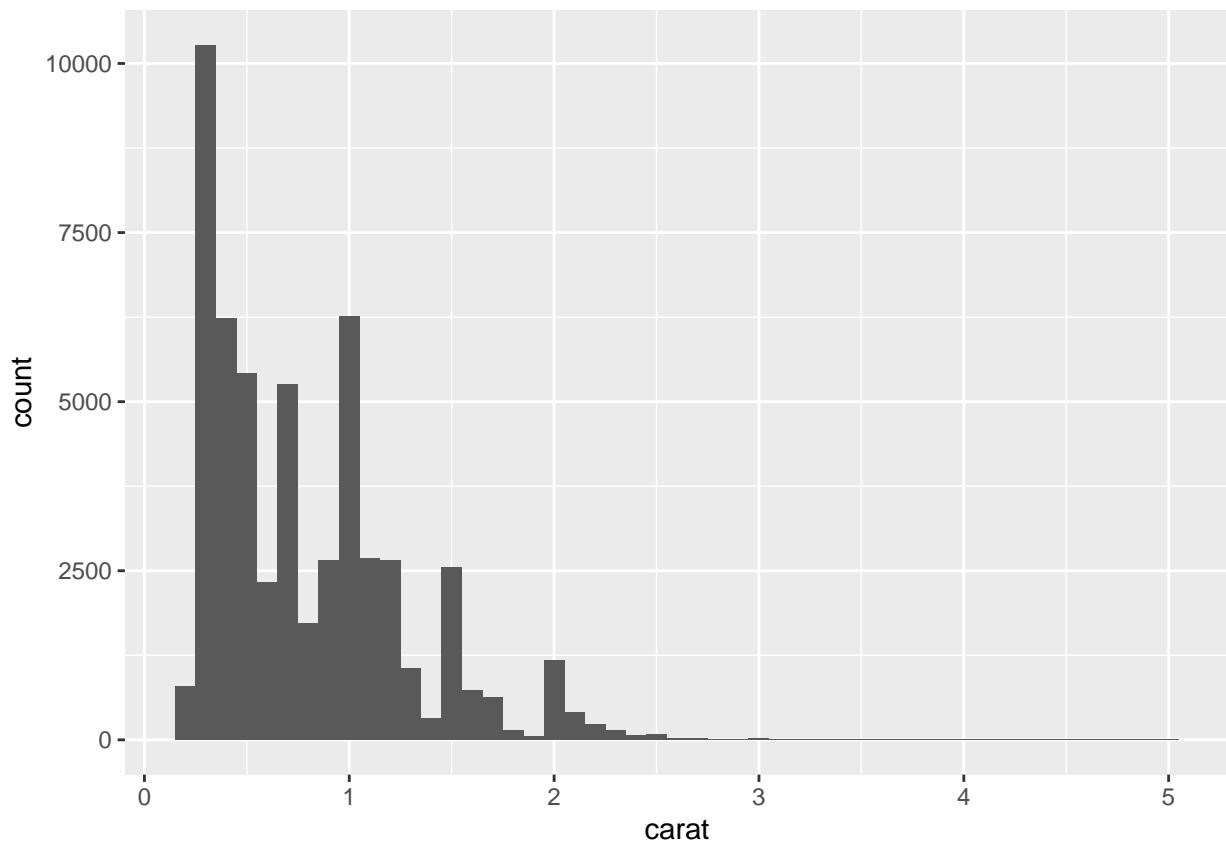
Le premier paramètre de la fonction `ggplot()` est donc le dataframe sur lequel faire le graphique (ici `diamonds`). La fonction `aes()` permet de définir les aspects esthétiques (ici la variable `carat`). Cette fonction définit les paramètres globaux du graphique. Ensuite, nous appliquons une transformation géométrique, en calculant donc un histogramme (avec 30 intervalles par défaut), à l'aide de la fonction `geom_histogram()`. L'instruction `bins` permet de choisir le nombre d'intervalle.

```
ggplot(data = diamonds, aes(x = carat)) + geom_histogram(bins = 20)
```



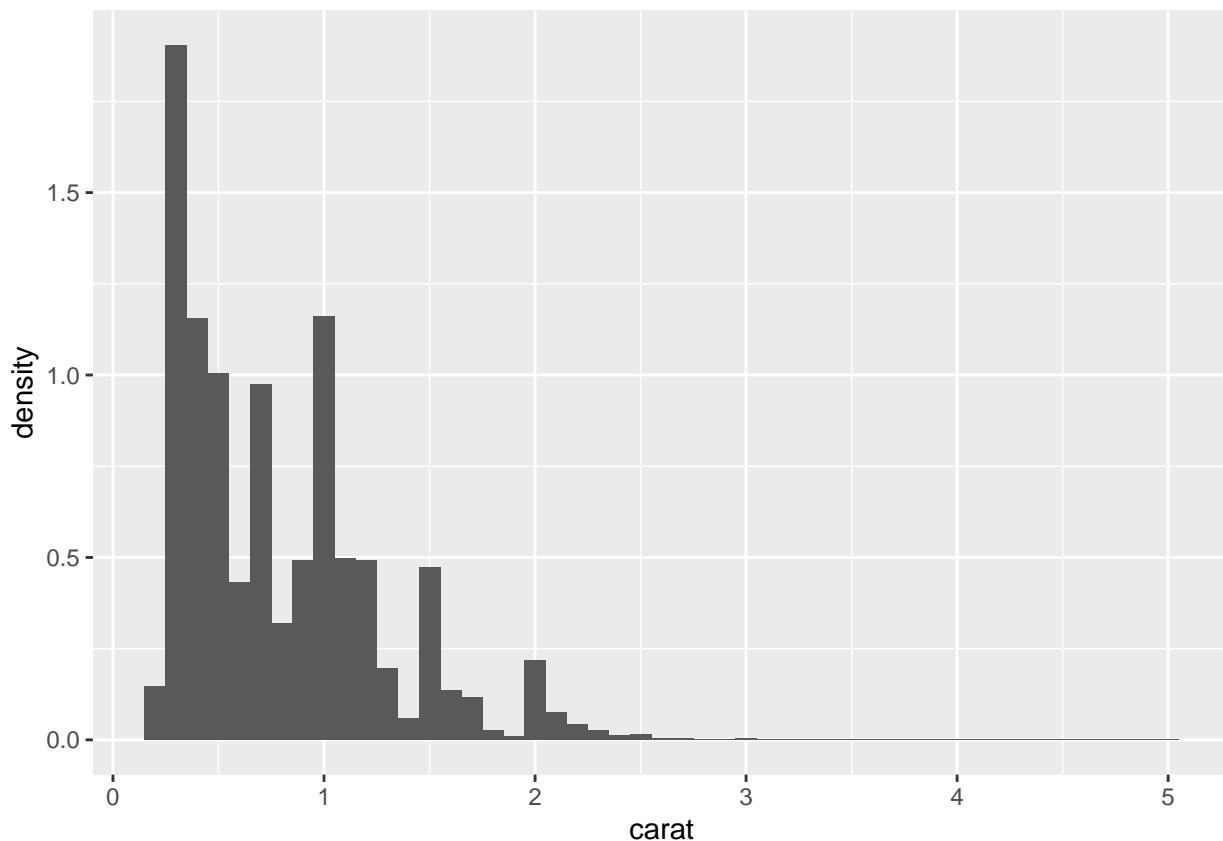
L'instruction `binwidth` permet de modifier la base des rectangles formant l'histogramme.

```
ggplot(data = diamonds, aes(x = carat)) + geom_histogram(binwidth = 0.1)
```



Tous les histogrammes présentés précédemment le sont en terme d'effectif. Il est possible de les représenter en terme de densité comme suit.

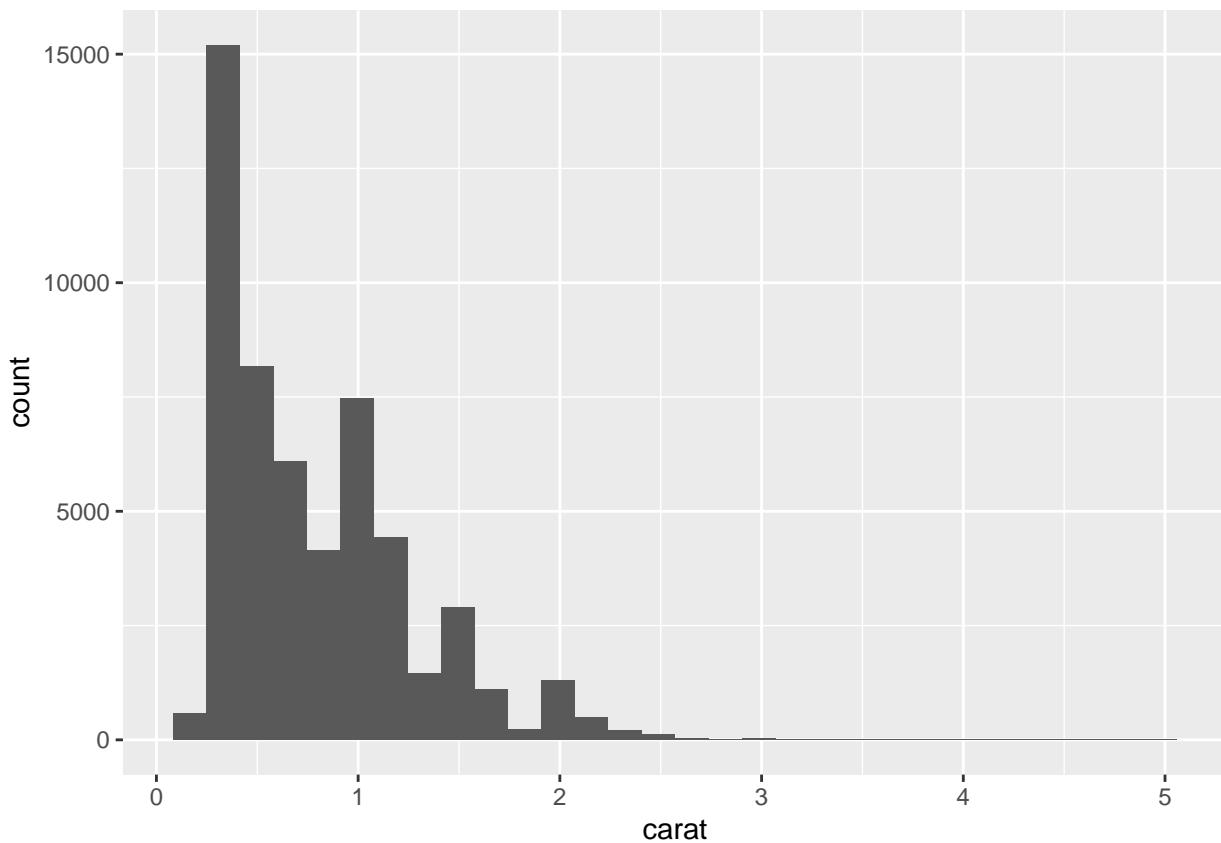
```
ggplot(data = diamonds, aes(x = carat)) + geom_histogram(aes(y=..density..), binwidth=0.1)
```



### Stockage dans une variable

Un des gros intérêt de la fonction `ggplot()` est le stockage du résultat dans une variable. Pour l'afficher, on peut soit appeler la variable, soit utiliser la fonction `print()` explicitement (voir ci-dessous).

```
histo = ggplot(data = diamonds, aes(x = carat)) + geom_histogram()  
histo # ou print(histo)
```



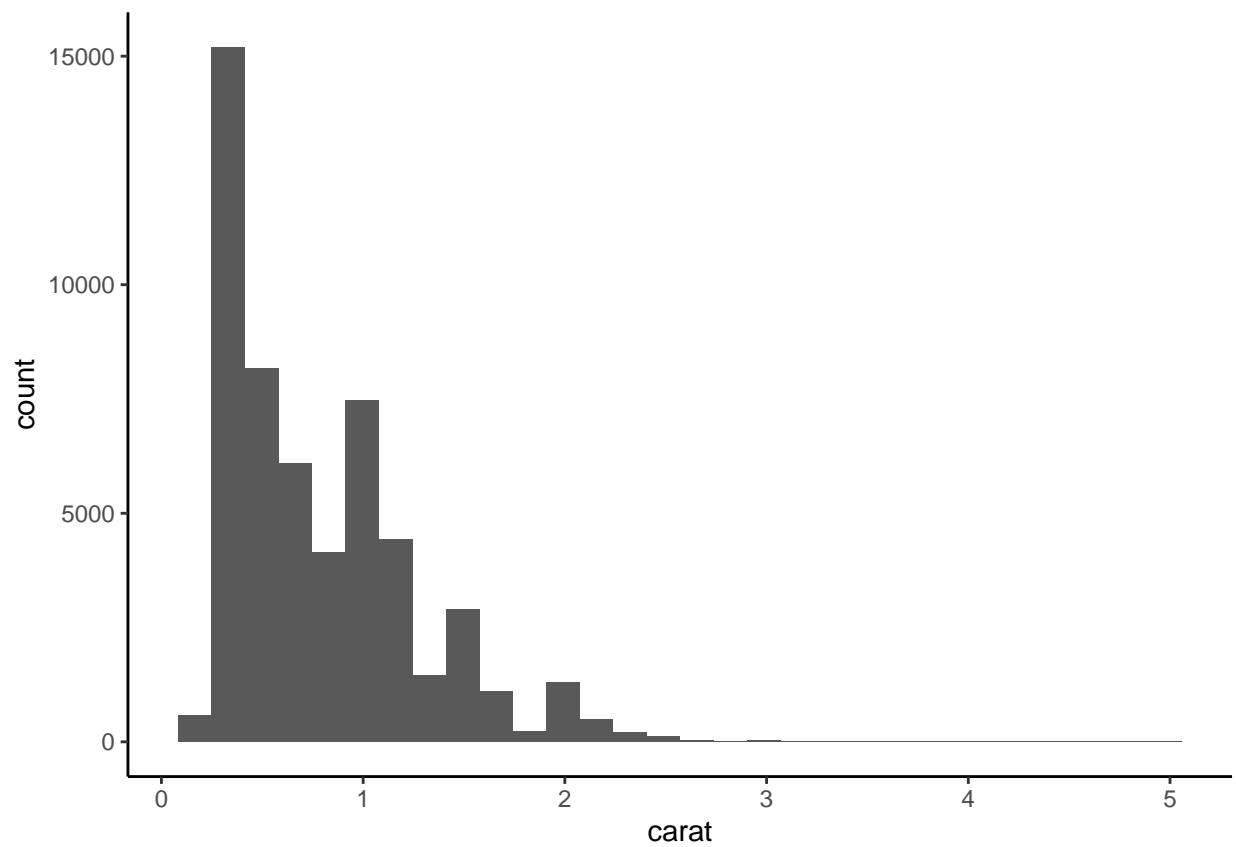
L'intérêt sera de pouvoir créer des graphiques et les stocker dans un fichier **RData**, pour les afficher plus tard et/ou les modifier.

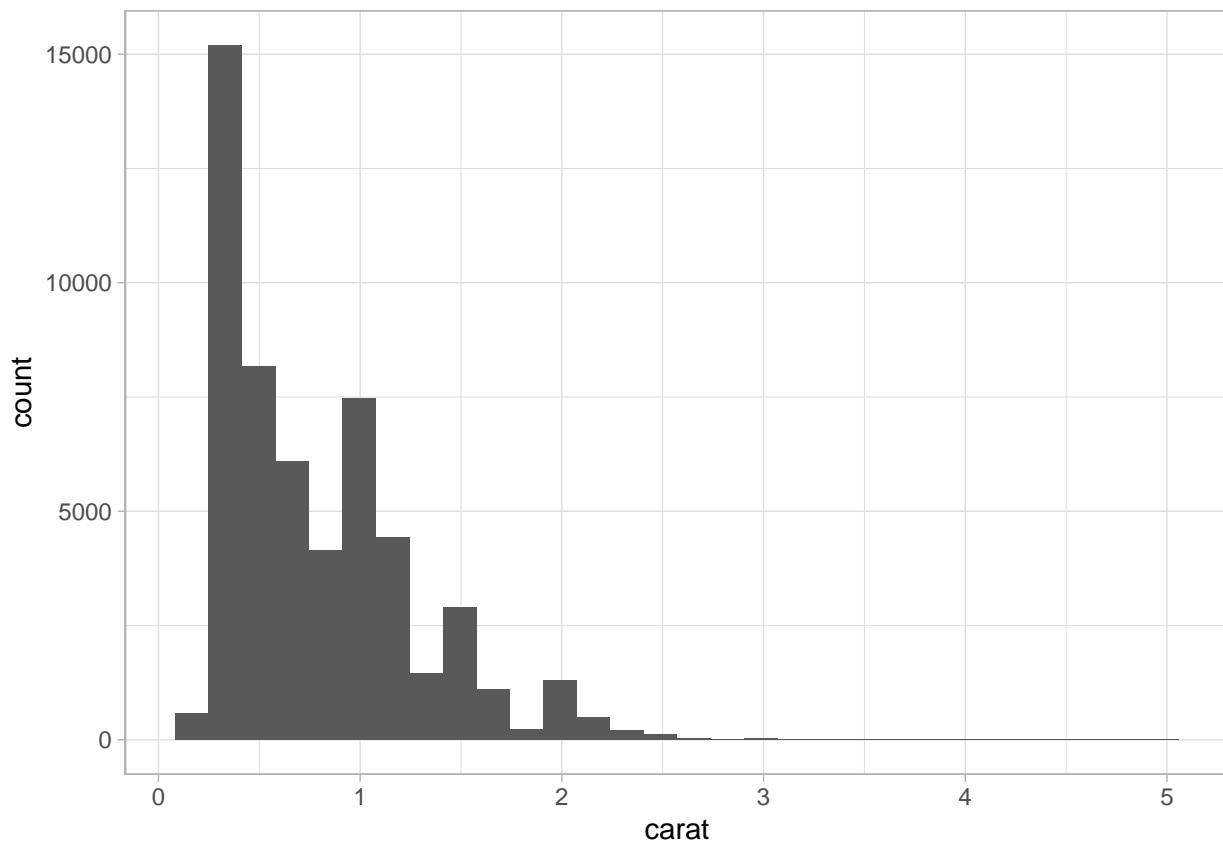
### Personnalisation du graphique

#### Thème général

On peut améliorer le graphique de différentes manières. Tout d'abord, il existe différents thèmes généraux (cf `?theme_grey` - ou un autre - pour voir la liste). Voici le thème `classic` et le thème `light`:

```
histo + theme_classic()
```





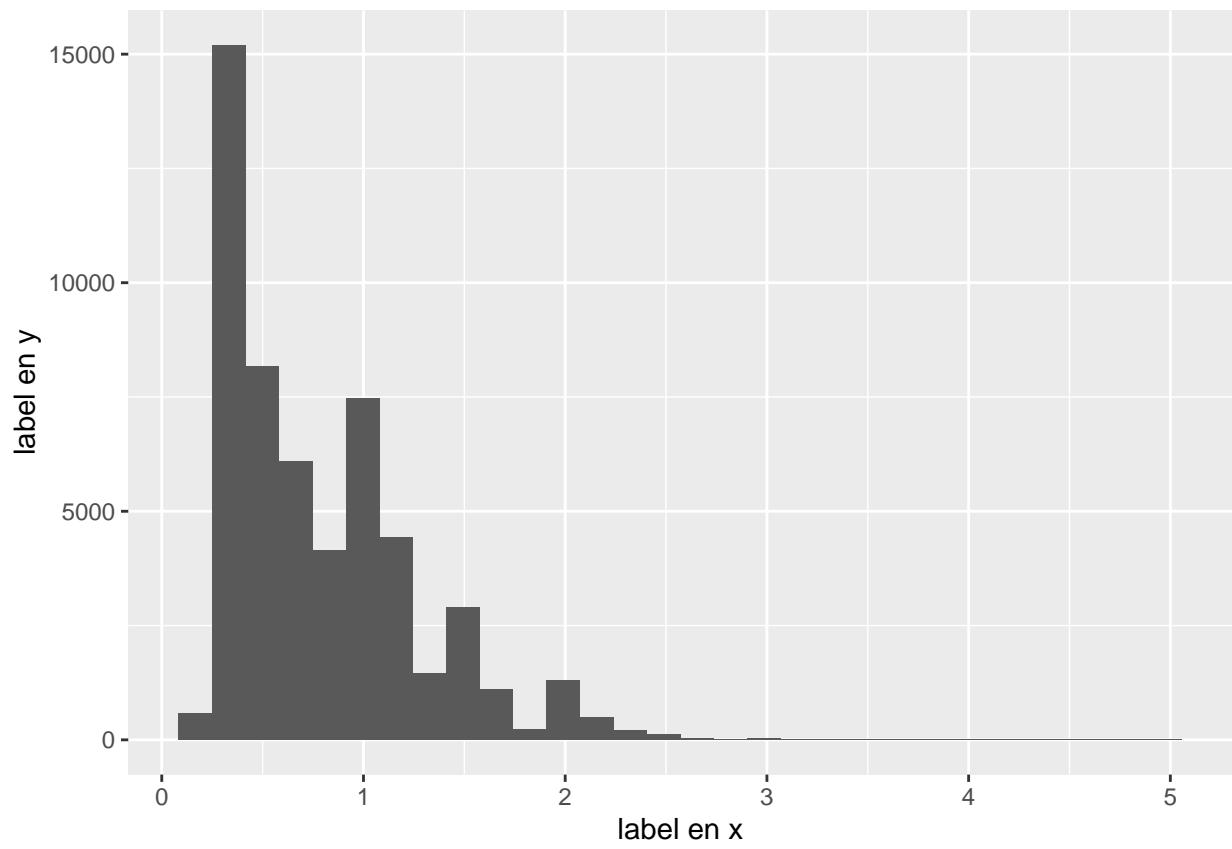
## Plus d'options

Il existe beaucoup d'autres possibilités, avec la fonction `theme()` (pas d'exemple ici car complexe).

## Labels

Une autre personnalisation courante est la redéfinition des labels des axes (et des légendes, comme nous le verrons plus tard), qui peut être faite avec la fonction `labs()`.

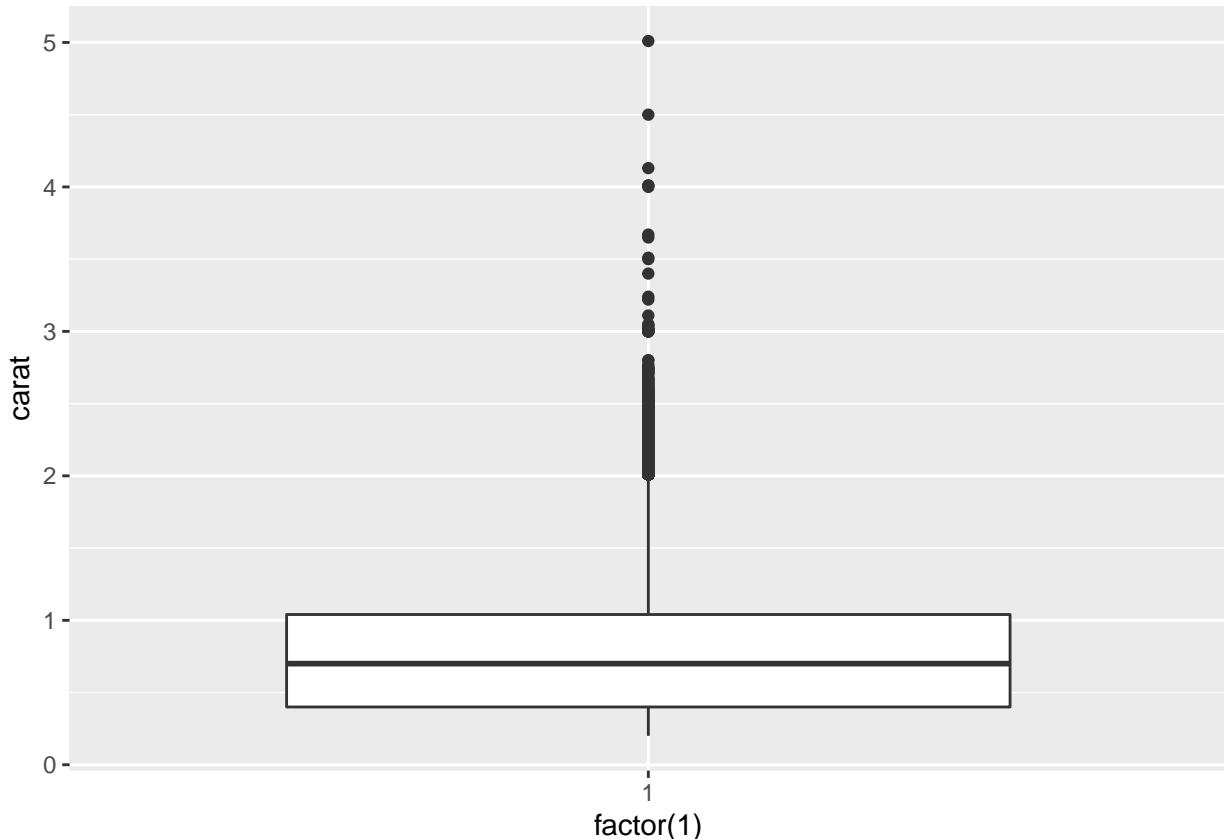
```
histo + labs(x = "label en x", y = "label en y")
```



### Boxplot

Si l'on veut tracer une boite à moustaches classique de la variable `carat` on utilise l'instruction suivante :

```
ggplot(data=diamonds,aes(x=factor(1),y=carat))+geom_boxplot()
```



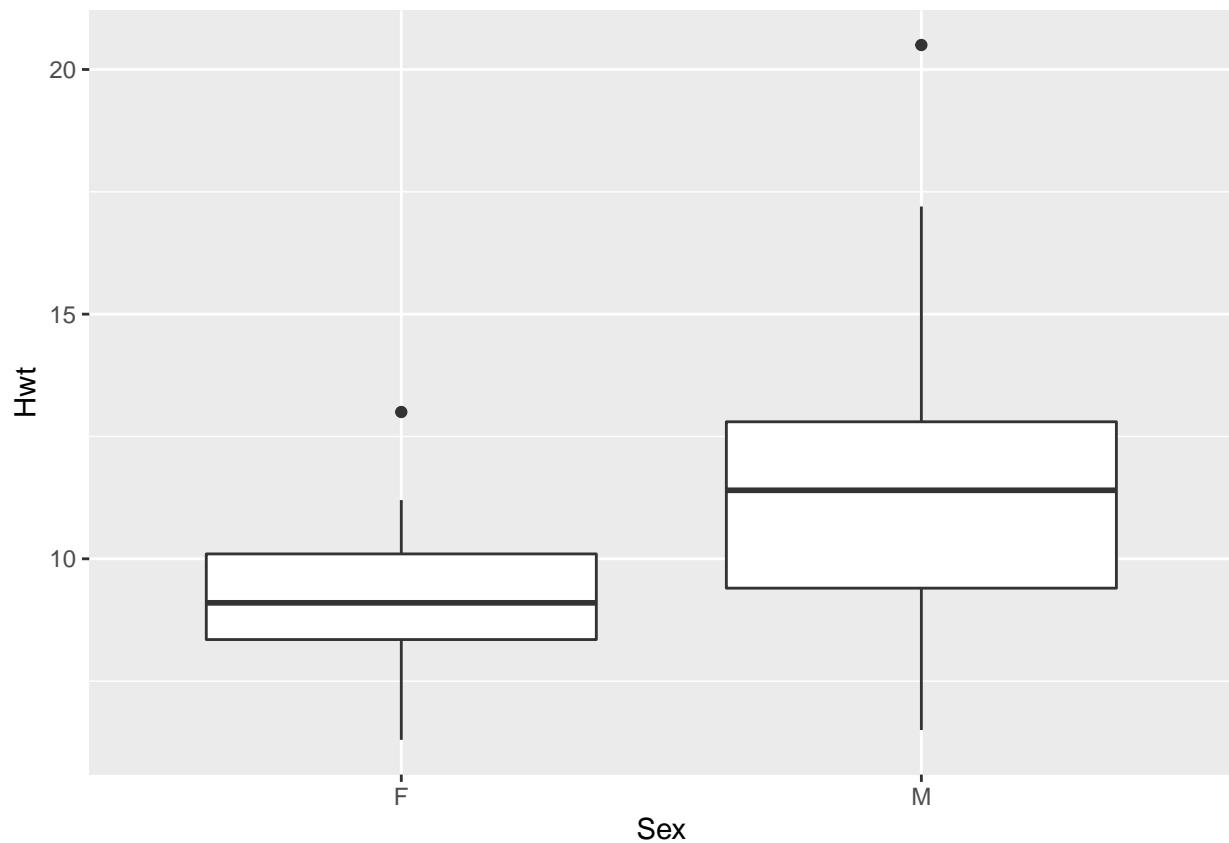
Il est en effet beaucoup plus simple de tracer plusieurs boites à moustaches (fonction d'une variable) qu'une seule boite à moustache avec `ggplot`. Regardons ce qu'il se passe sur le jeu de données `cats`

```
head(cats)

##   Sex Bwt Hwt
## 1  F  2.0 7.0
## 2  F  2.0 7.4
## 3  F  2.0 9.5
## 4  F  2.1 7.2
## 5  F  2.1 7.3
## 6  F  2.1 7.6

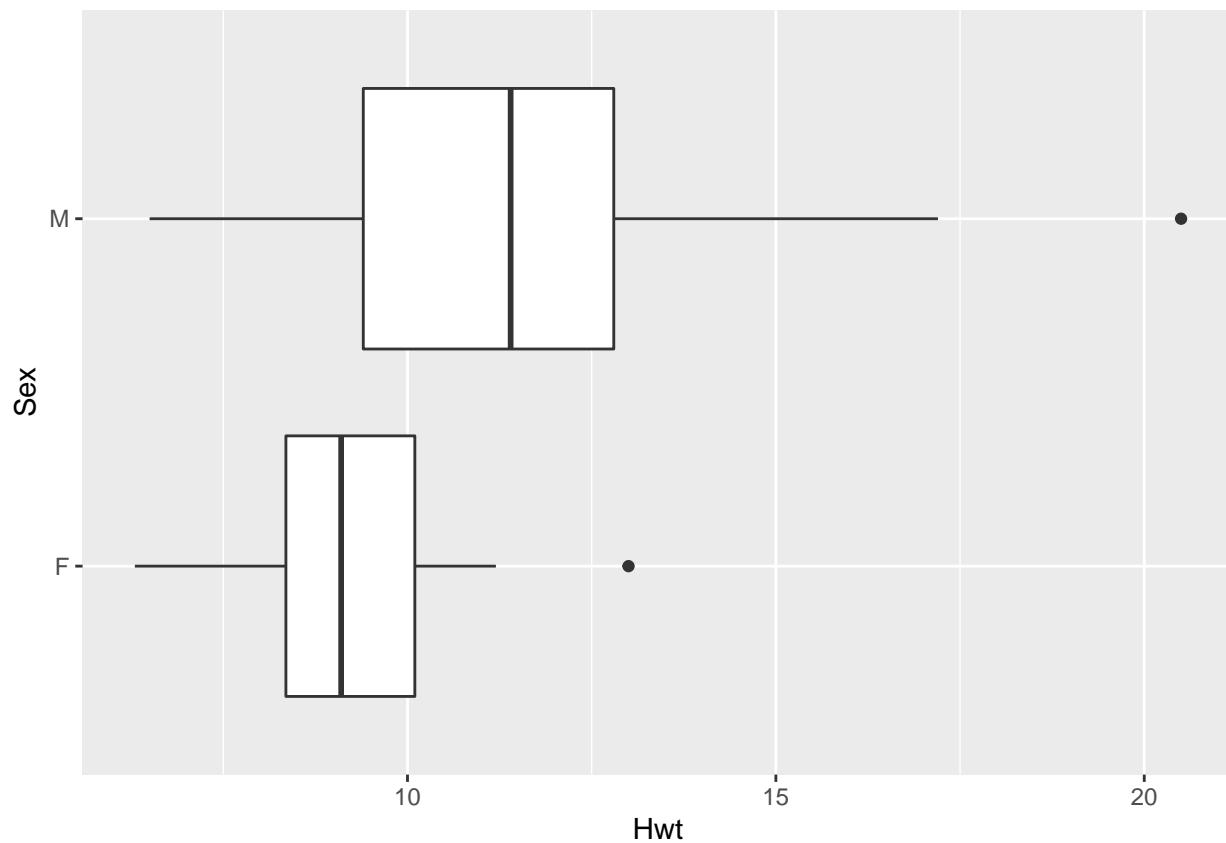
# ?cats
str(cats)

## 'data.frame':    144 obs. of  3 variables:
## $ Sex: Factor w/ 2 levels "F","M": 1 1 1 1 1 1 1 1 1 1 ...
## $ Bwt: num  2 2 2 2.1 2.1 2.1 2.1 2.1 2.1 2.1 ...
## $ Hwt: num  7 7.4 9.5 7.2 7.3 7.6 8.1 8.2 8.3 8.5 ...
box = ggplot(data=cats, aes(x=Sex,y=Hwt))
box + geom_boxplot()
```



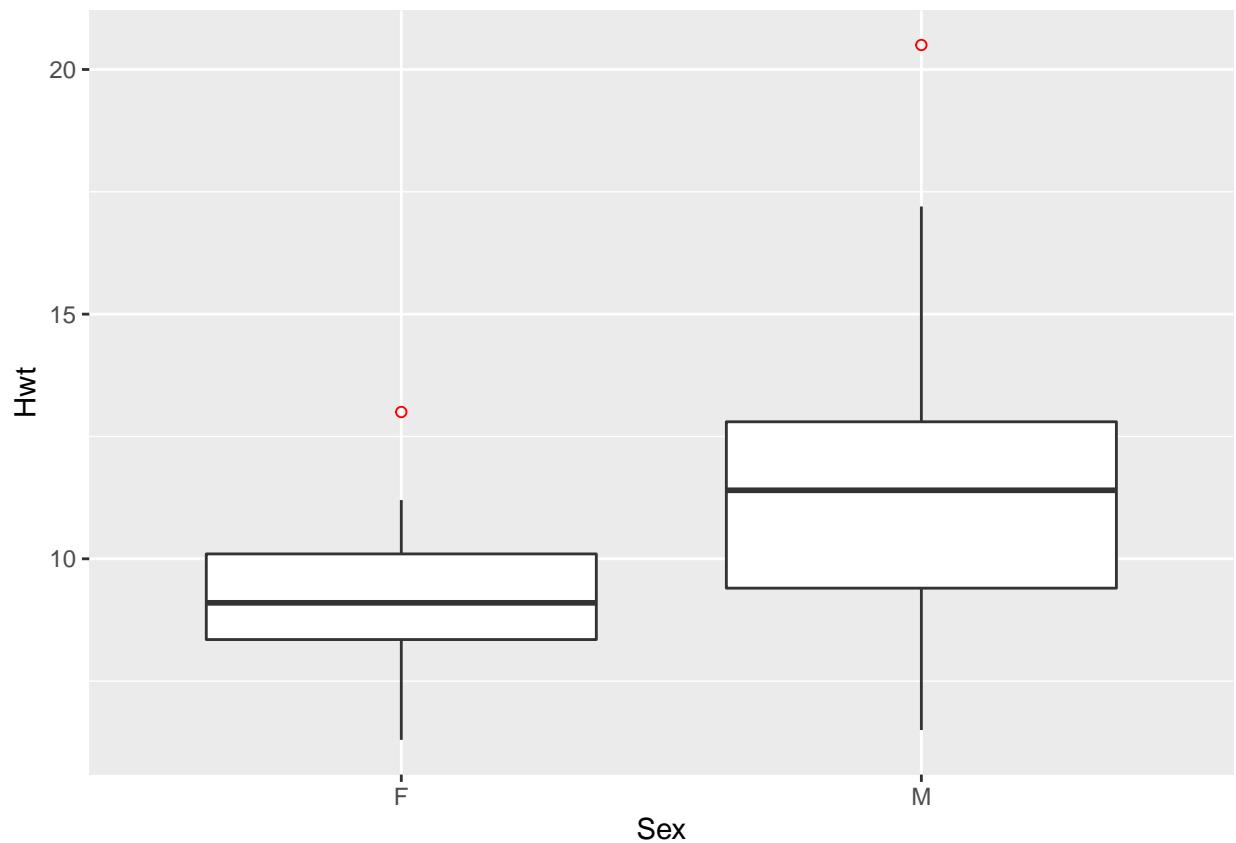
Par défaut les boites à moustaches sont dans le sens vertical mais il est très facile de les positionner à l'horizontale à l'aide de `coord_flip()`

```
box + geom_boxplot() + coord_flip()
```



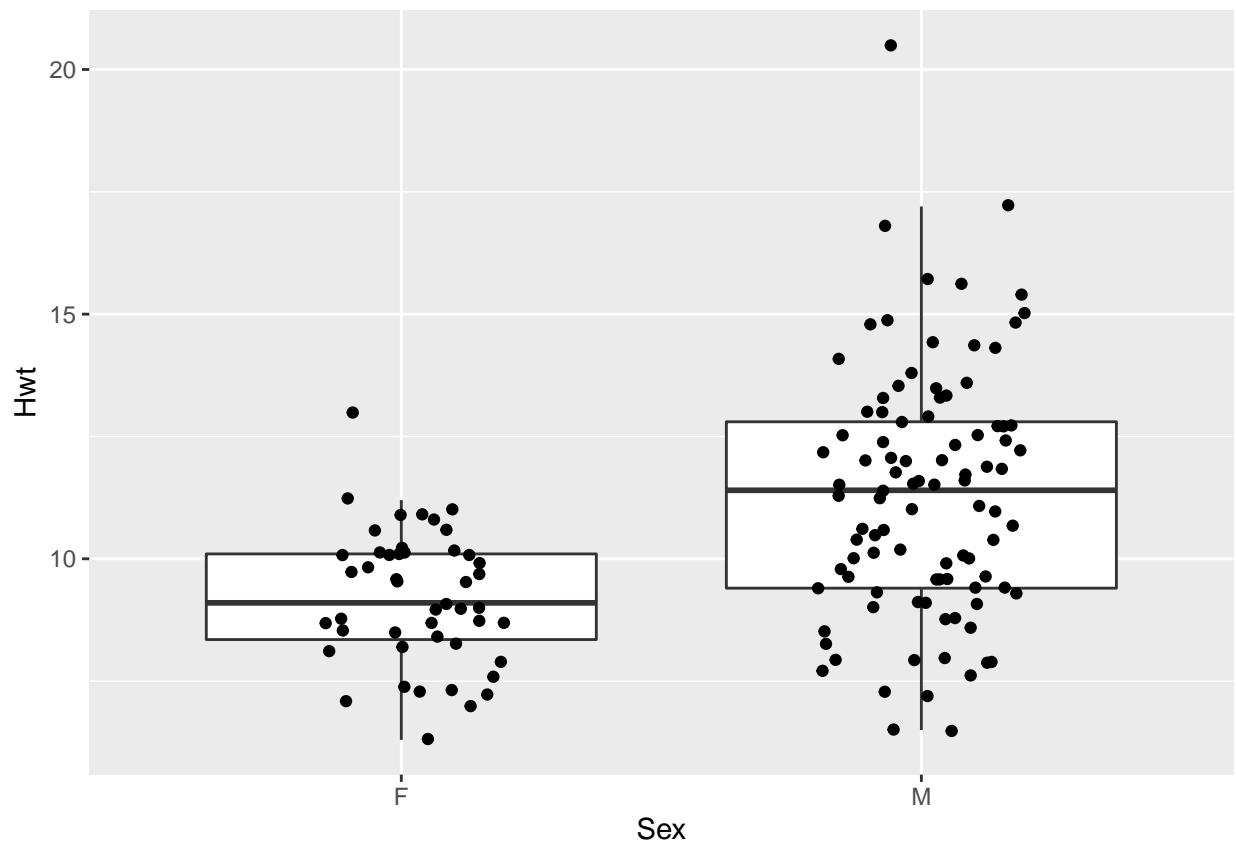
Il est possible de mettre en avant les outliers dans une autre couleur.

```
box + geom_boxplot(outlier.colour = "red", outlier.shape = 1)
```



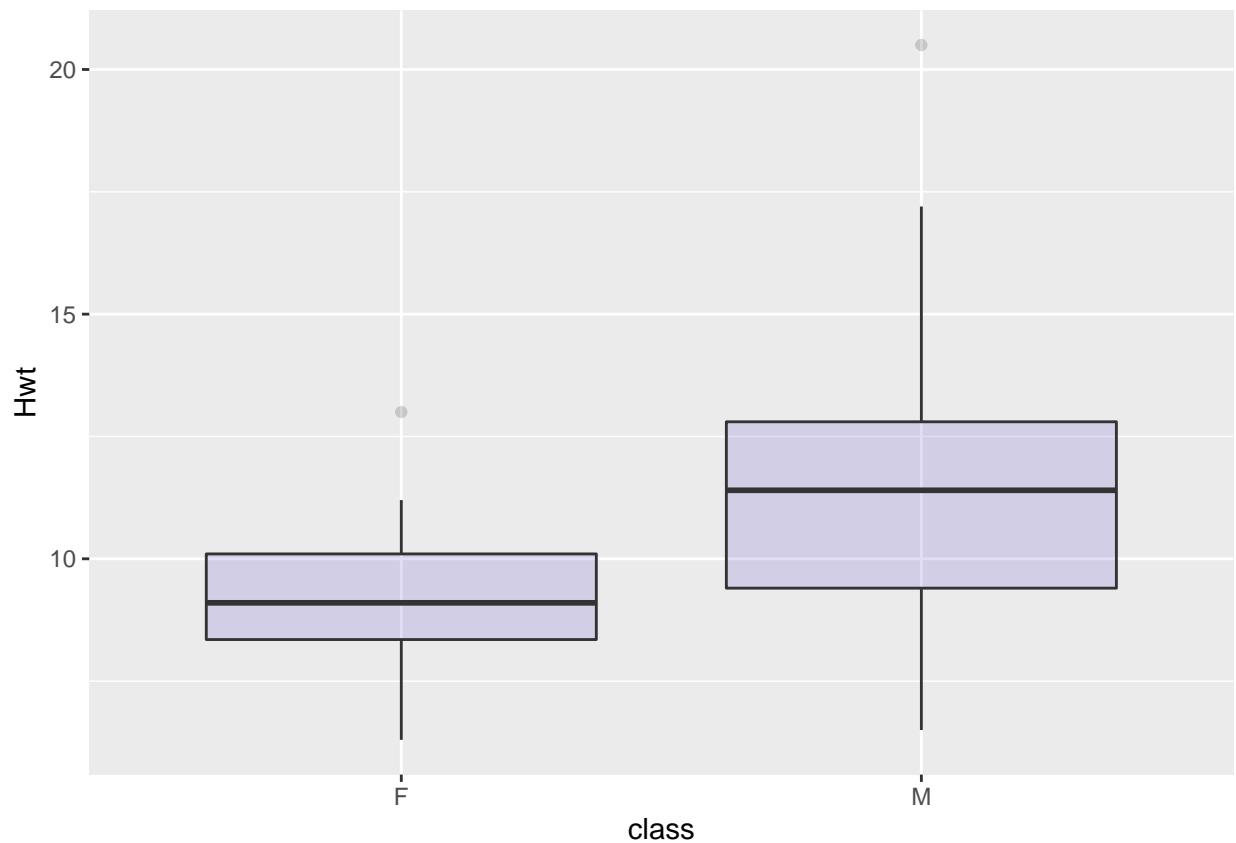
On peut mettre en évidence tous les points en les superposant aux boîtes à moustaches et en utilisant `geom_jitter` pour les décaler. Attention à bien supprimer les outliers pour pas avoir de doublons.

```
box + geom_boxplot(outlier.shape = NA) + geom_jitter(width = 0.2)
```



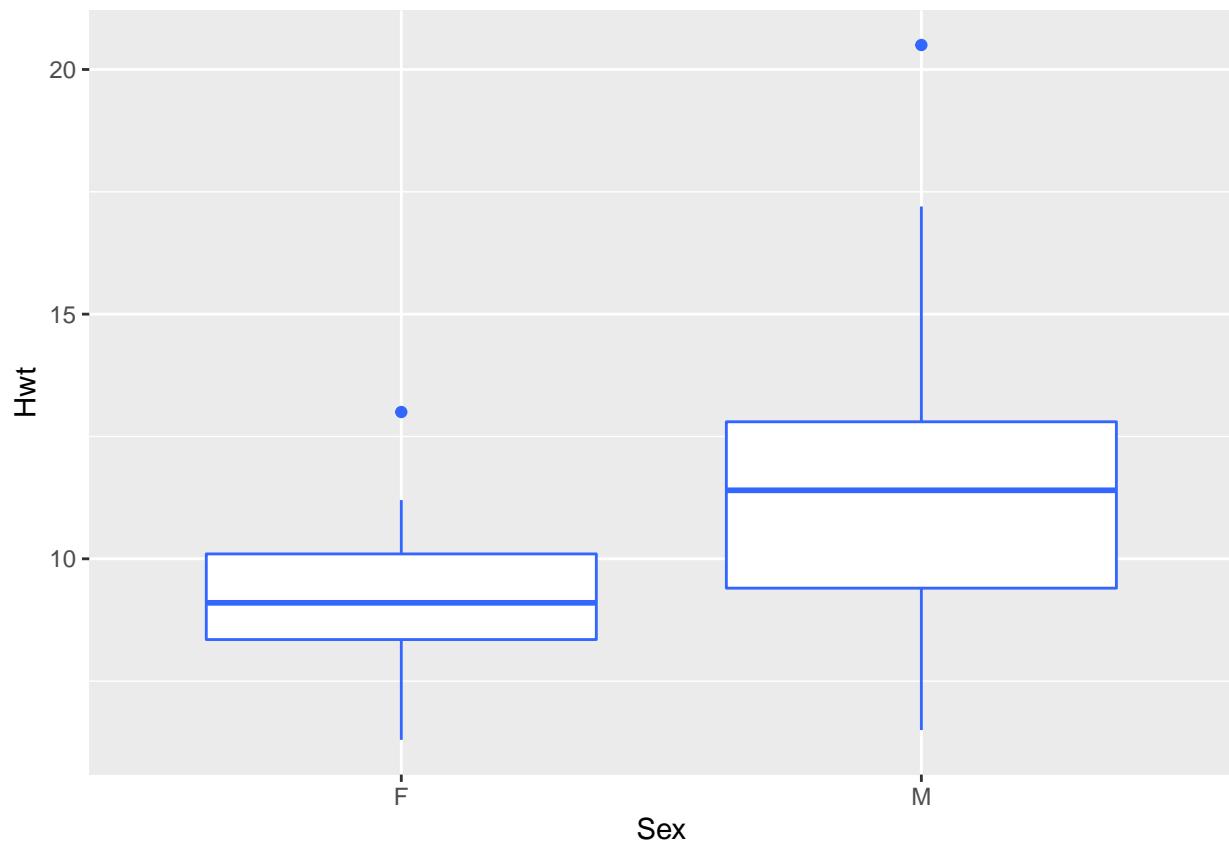
On peut également changer la couleur à l'aide de la palette de couleurs déjà présente dans R. L'instruction `alpha` permet de spécifier la transparence.

```
box + geom_boxplot(fill="slateblue", alpha=0.2) + xlab("class")
```



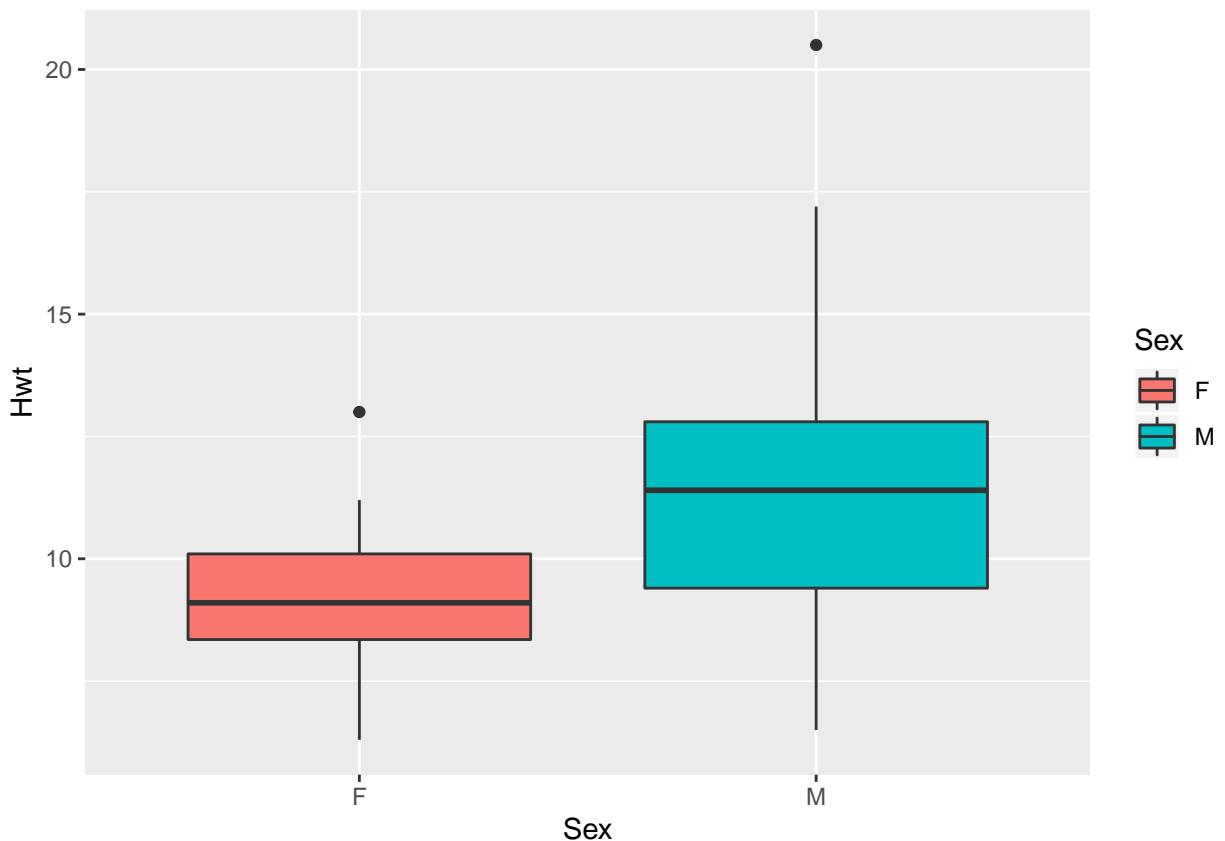
On peut spécifier la couleur des boîtes à moustaches : contour et intérieur.

```
box + geom_boxplot(fill = "white", colour = "#3366FF")
```



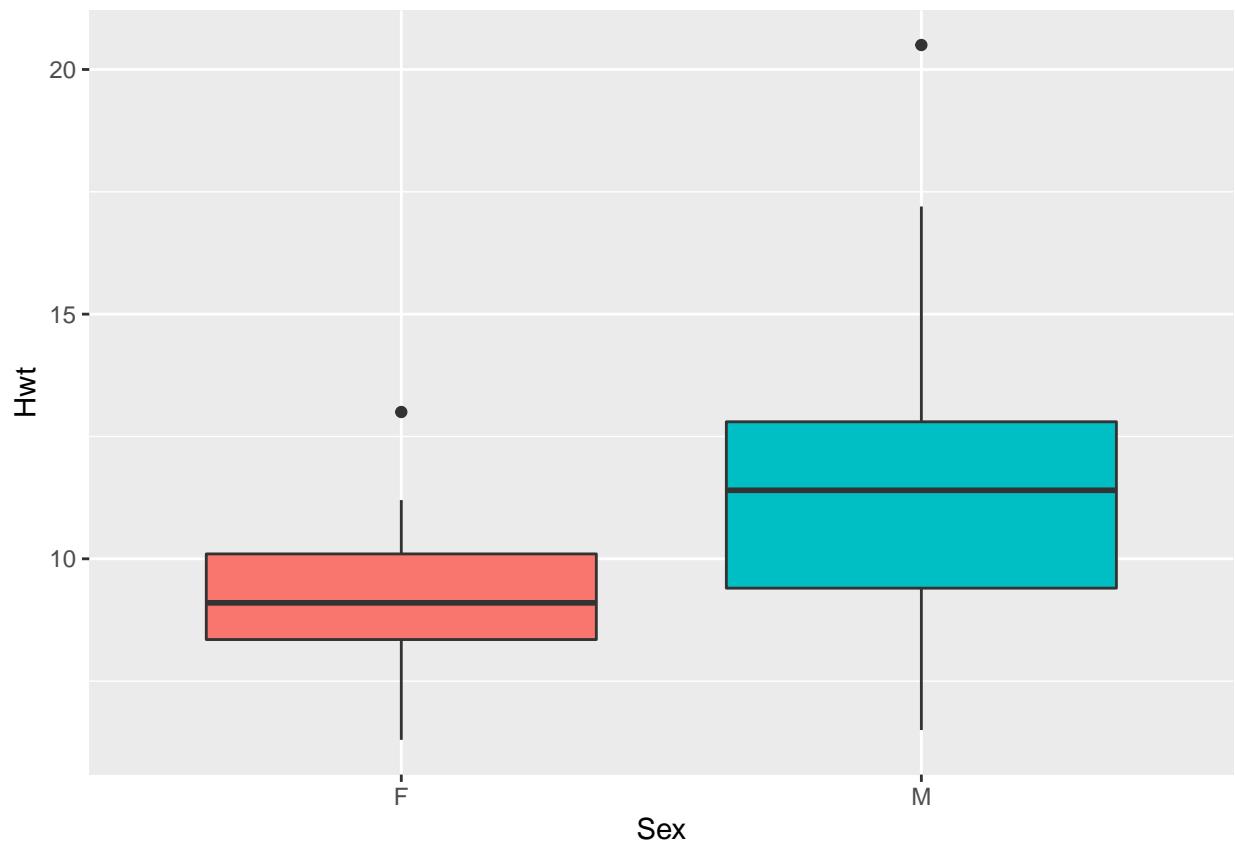
Si l'on souhaite ne rien spécifier dans `geom_boxplot()` on peut conditionner les couleurs en fonction de la variable `class` à l'aide de l'instruction `fill` dans `ggplot`.

```
ggplot(data=cats, aes(x=Sex, y=Hwt, fill=Sex)) + geom_boxplot()
```



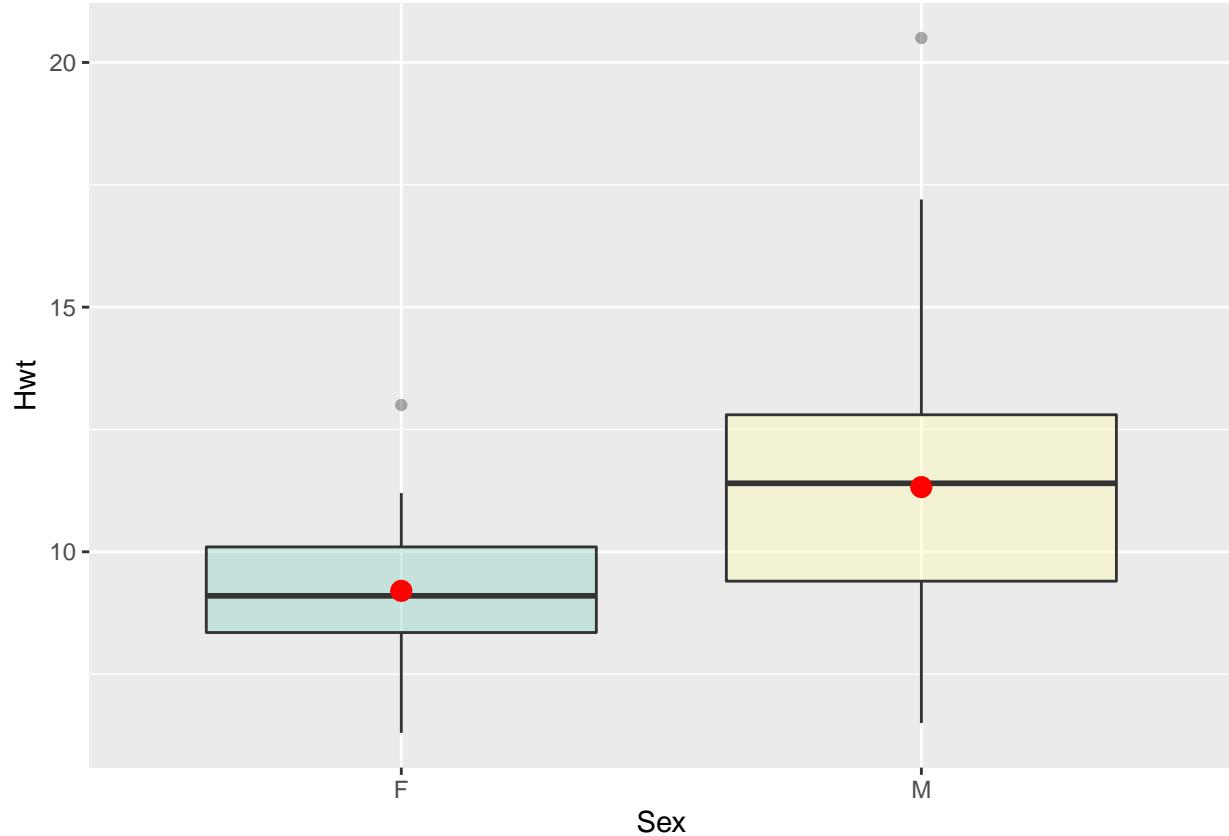
Afin d'avoir un graphique le plus sobre possible on supprime la légende.

```
ggplot(data=cats, aes(x=Sex, y=Hwt, fill=Sex)) + geom_boxplot() + theme(legend.position="none")
```



Il est un peu plus compliqué de rajouter la moyenne à chacun de nos boxplots.

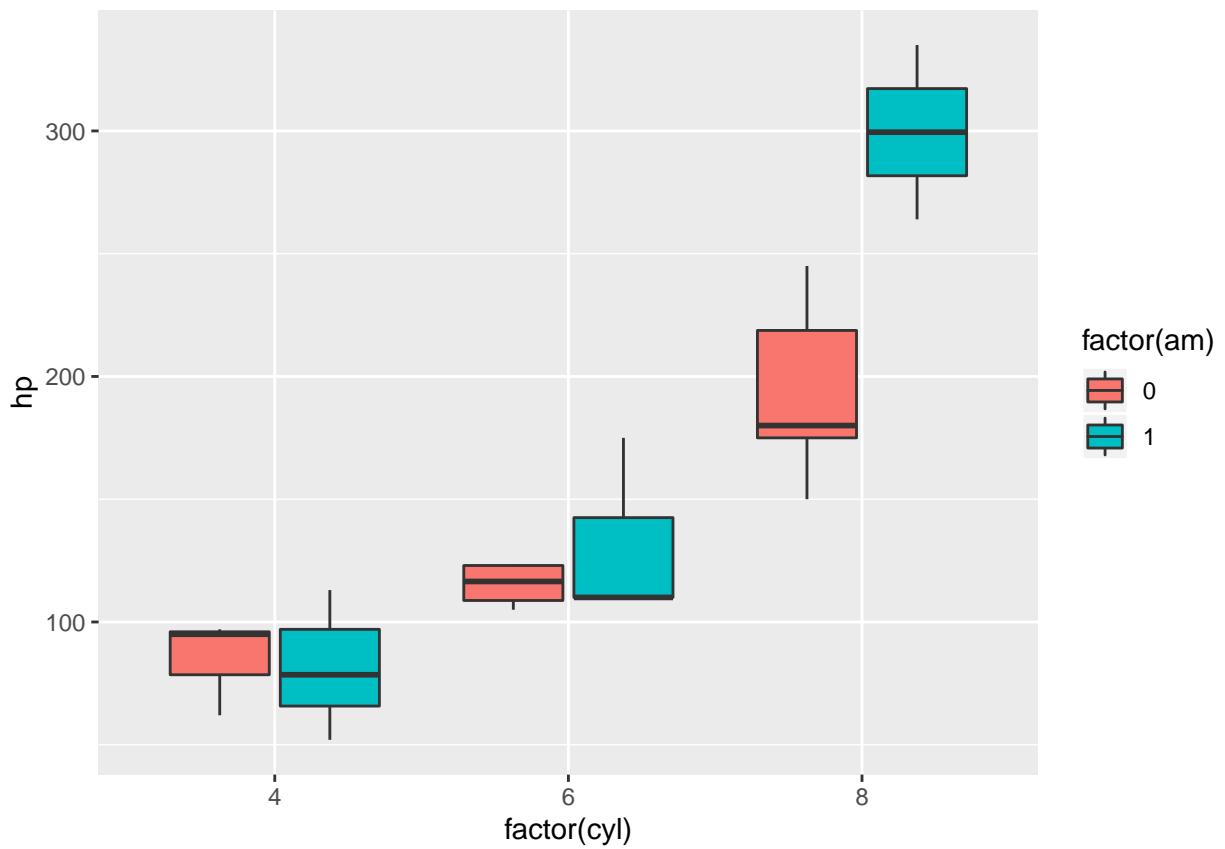
```
ggplot(data=cats, aes(x=Sex, y=Hwt, fill=Sex)) +geom_boxplot(alpha=0.4) +stat_summary(fun.y=mean, geom=
```



L'instruction `scale_fill_brewer` signifie : Sequential, diverging and qualitative colour scales from colorbrewer.org. Cette instruction permet de mettre des couleurs qui “vont bien ensemble” du package RColorBrewer. Voir <http://colorbrewer2.org> pour plus d'informations.

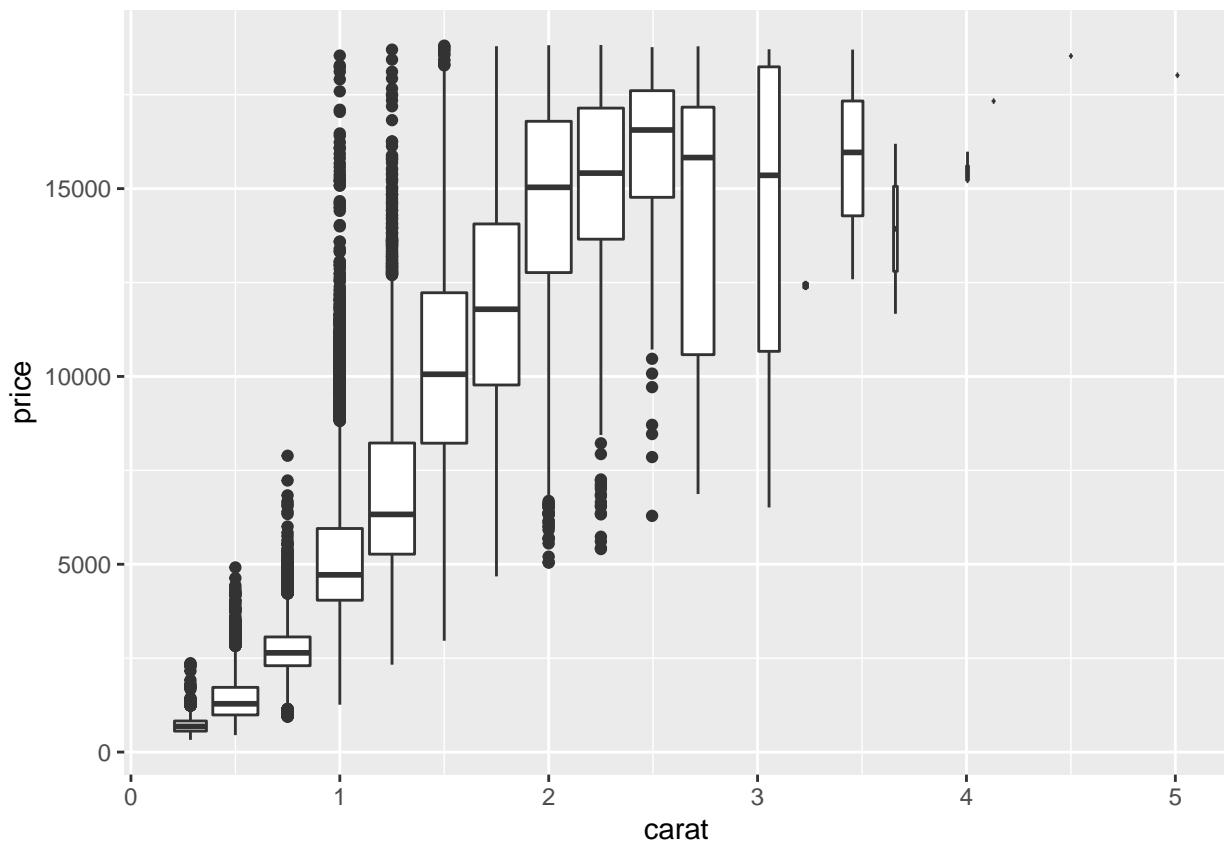
Il est possible de mettre des couleurs qui sont fonction des valeurs d'une variable qualitative ici `am`

```
ggplot(data = mtcars, aes(x=factor(cyl),y=hp,fill=factor(am)))+ geom_boxplot()
```



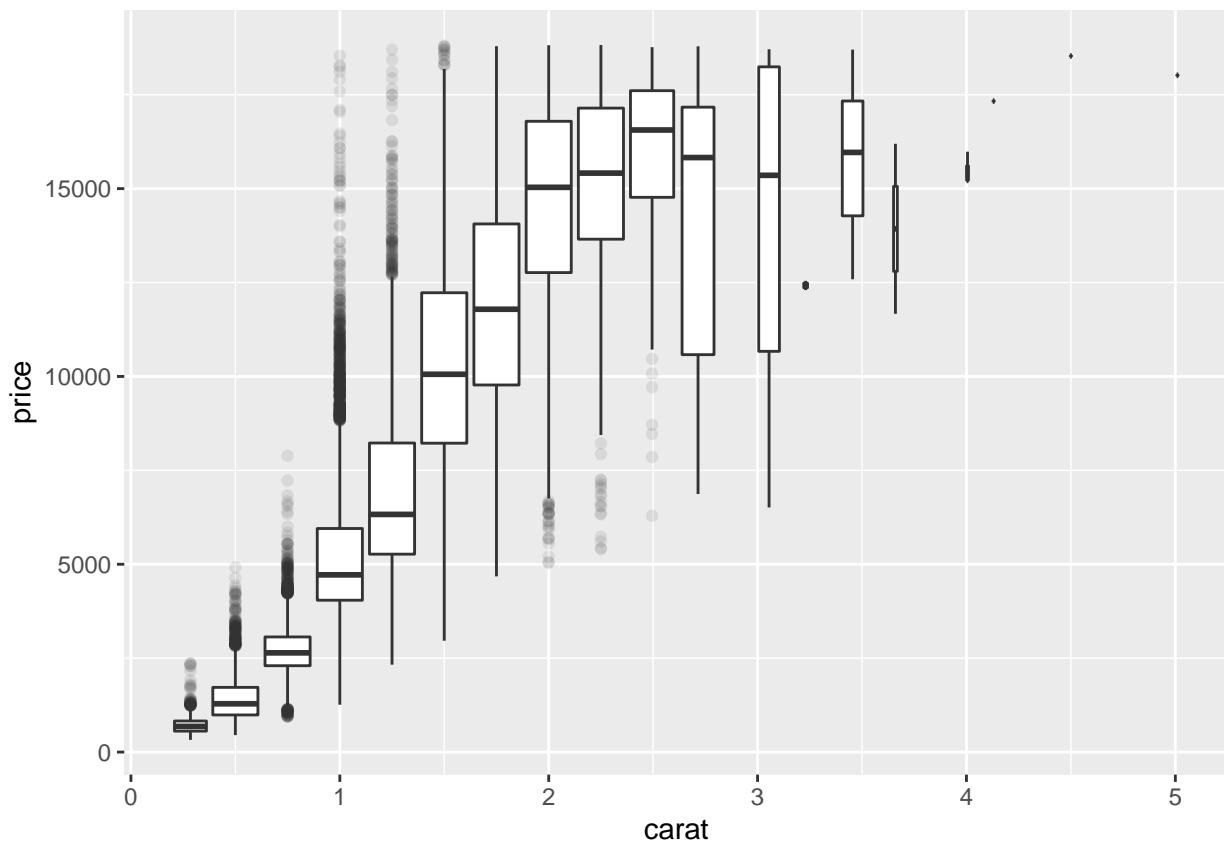
Revenons au jeu de données `diamonds` et traçons les boxplots de `price` en fonction de `carat` (variable quantitative ici coupée en classes).

```
ggplot(diamonds, aes(carat, price)) + geom_boxplot(aes(group = cut_width(carat, 0.25)))
```



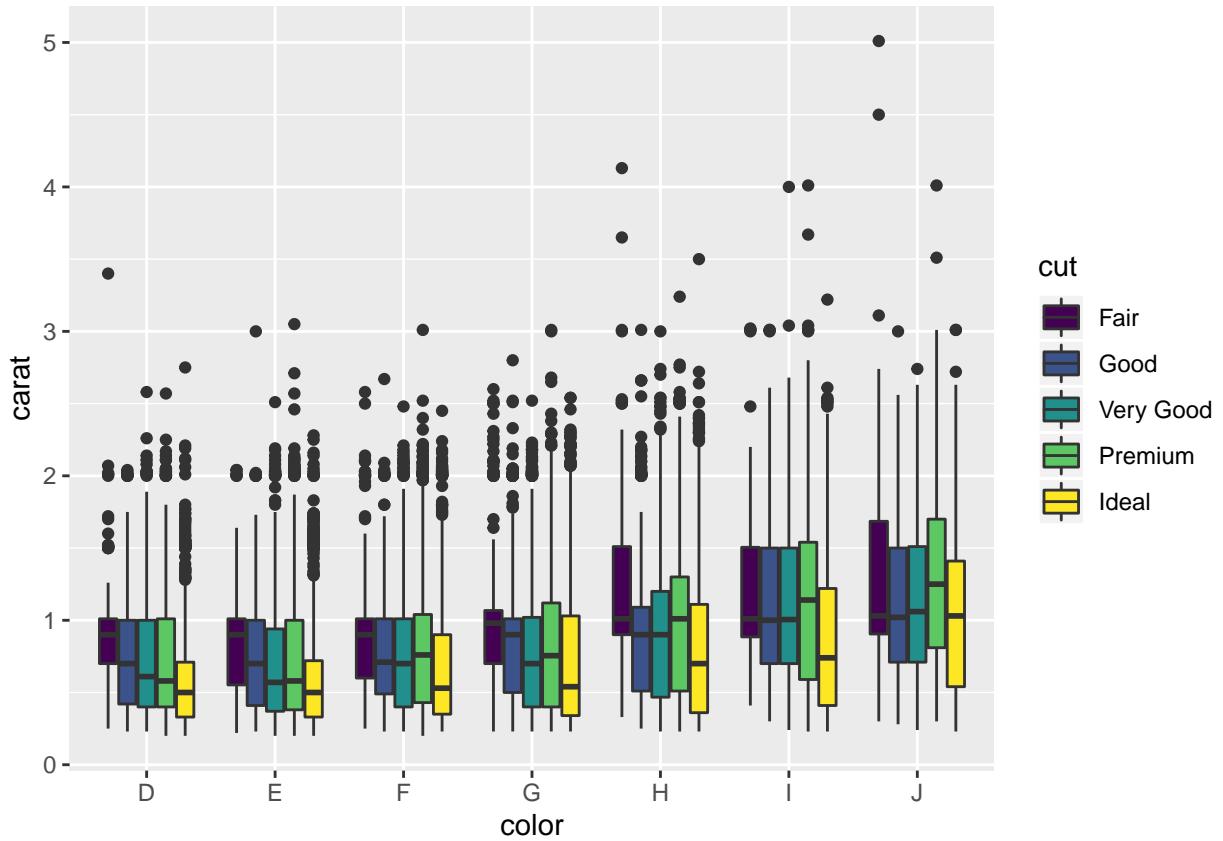
On peut jouer sur la transparence des outliers avec `outlier.alpha`.

```
ggplot(diamonds, aes(carat, price)) +  
  geom_boxplot(aes(group = cut_width(carat, 0.25)), outlier.alpha = 0.1)
```



Ici on représente les boxplots de `carat` en fonction de `color` mais également de `cut`. On a plusieurs niveaux de lecture.

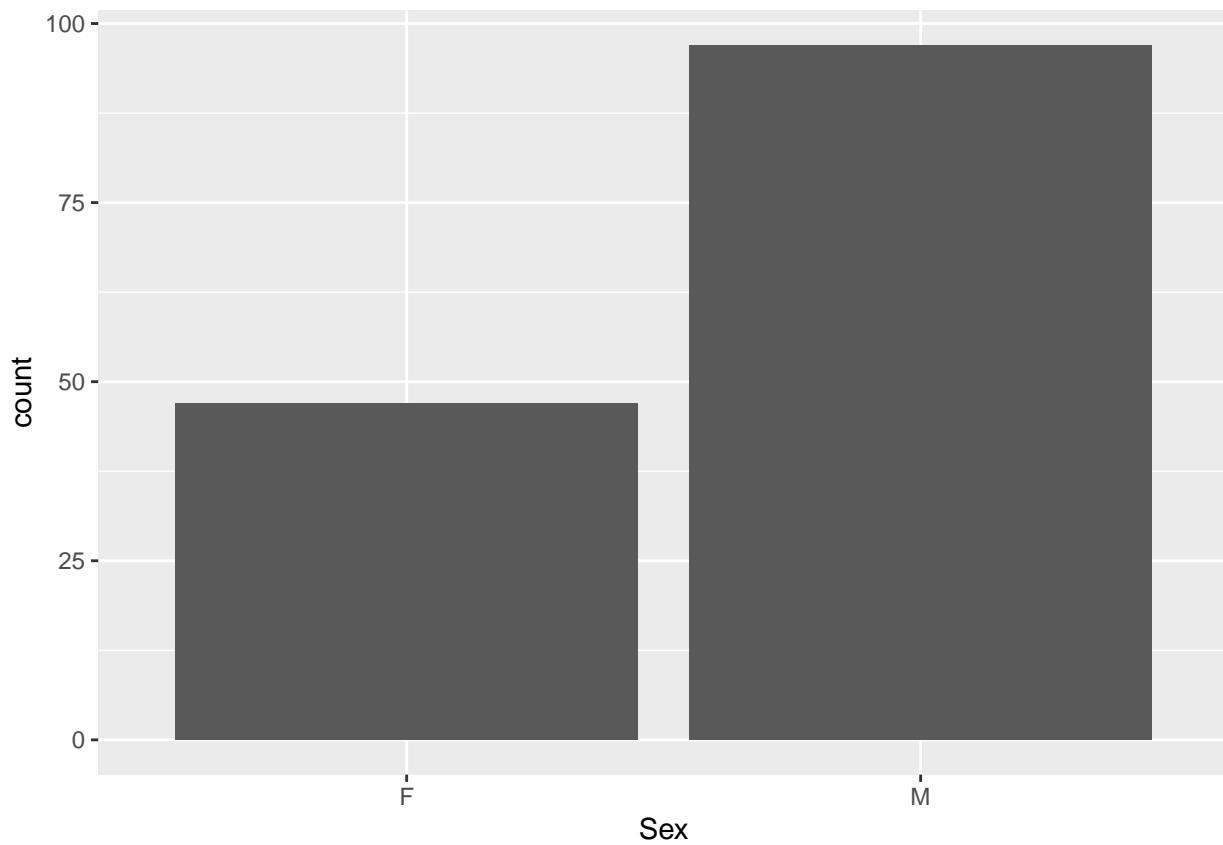
```
ggplot(diamonds, aes(x=color, y=carat, fill=cut)) + geom_boxplot()
```



## Barplot

Il est très simple de tracer un barplot ou diagramme en barres à l'aide de l'instruction `geom_bar()`

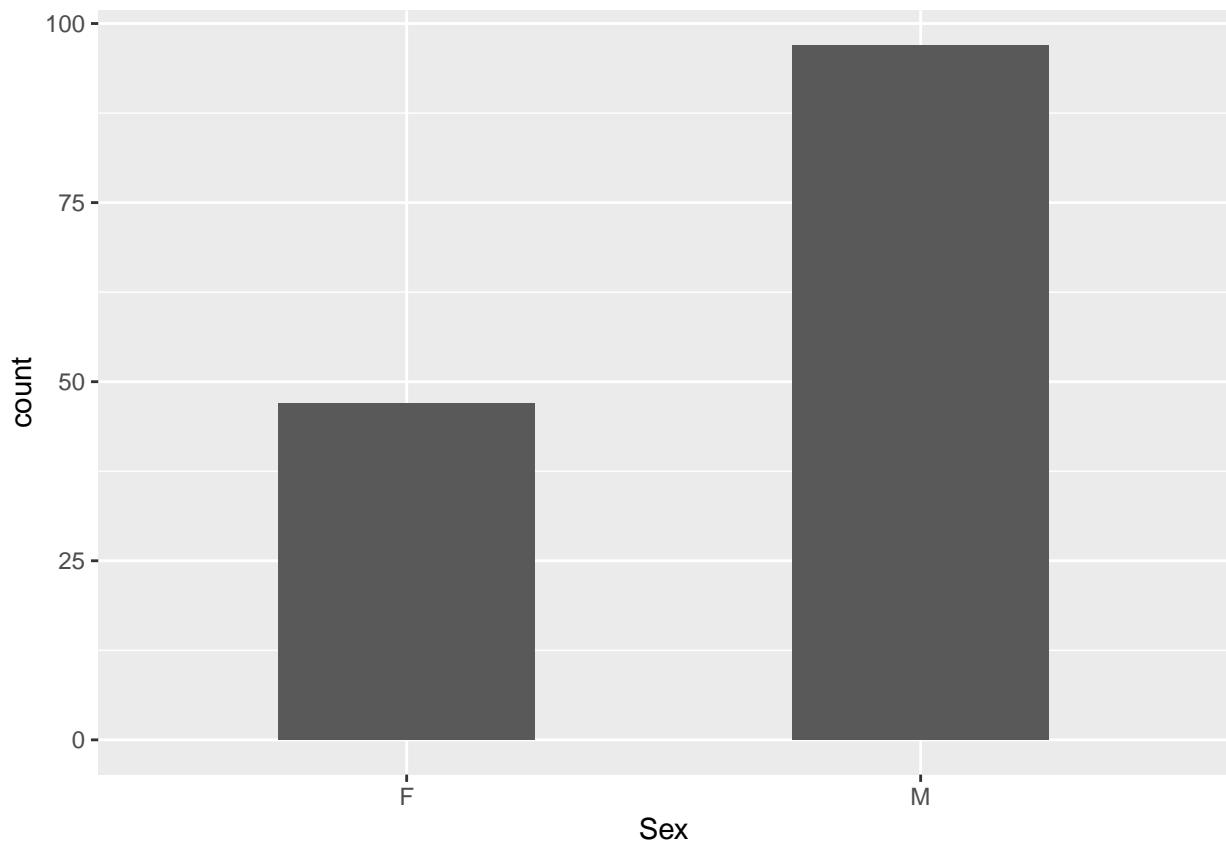
```
ggplot(data=cats, aes(x=Sex)) + geom_bar()
```



Calculer les effectifs en fonction de `Sex` à l'aide de `table(Sex)`.

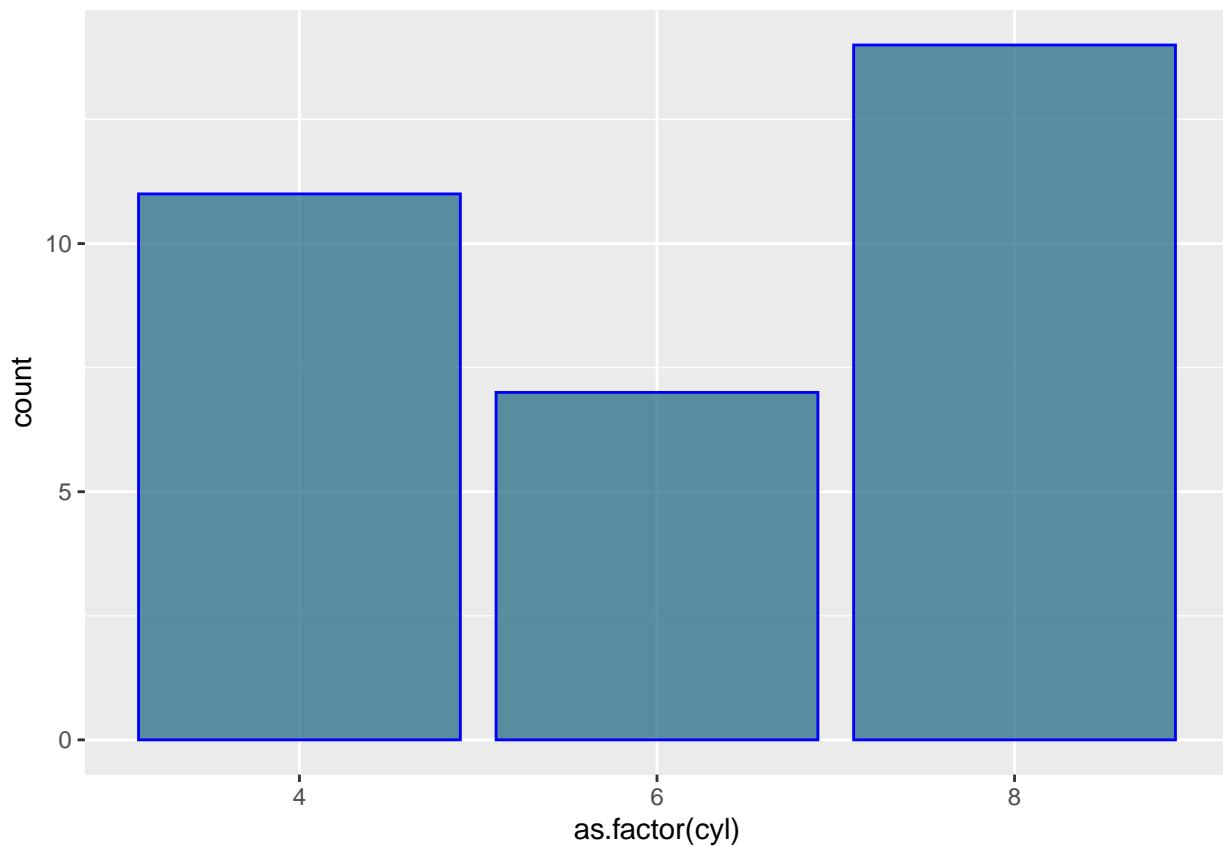
On peut réduire la largeur des batons avec `width`.

```
ggplot(data=cats, aes(x=Sex)) + geom_bar(width=0.5)
```



Comme pour les boxplots il est possible de modifier les couleurs intérieures et extérieures.

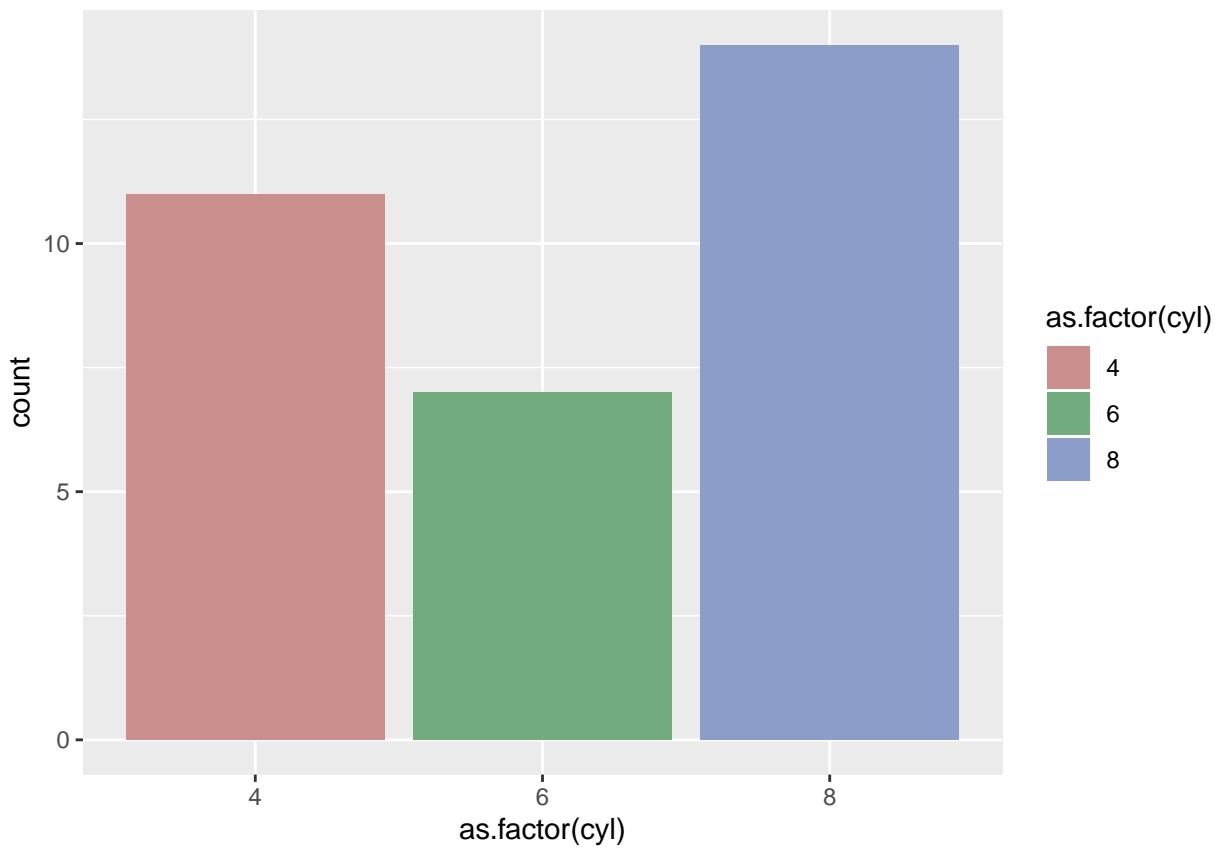
```
ggplot(data=mtcars, aes(x=as.factor(cyl)))+geom_bar(color="blue", fill=rgb(0.1,0.4,0.5,0.7) )
```



Où les couleurs et cela en fonction de la variable qualitative cyl. Toujours en utilisant l'instruction `scale_fill` :

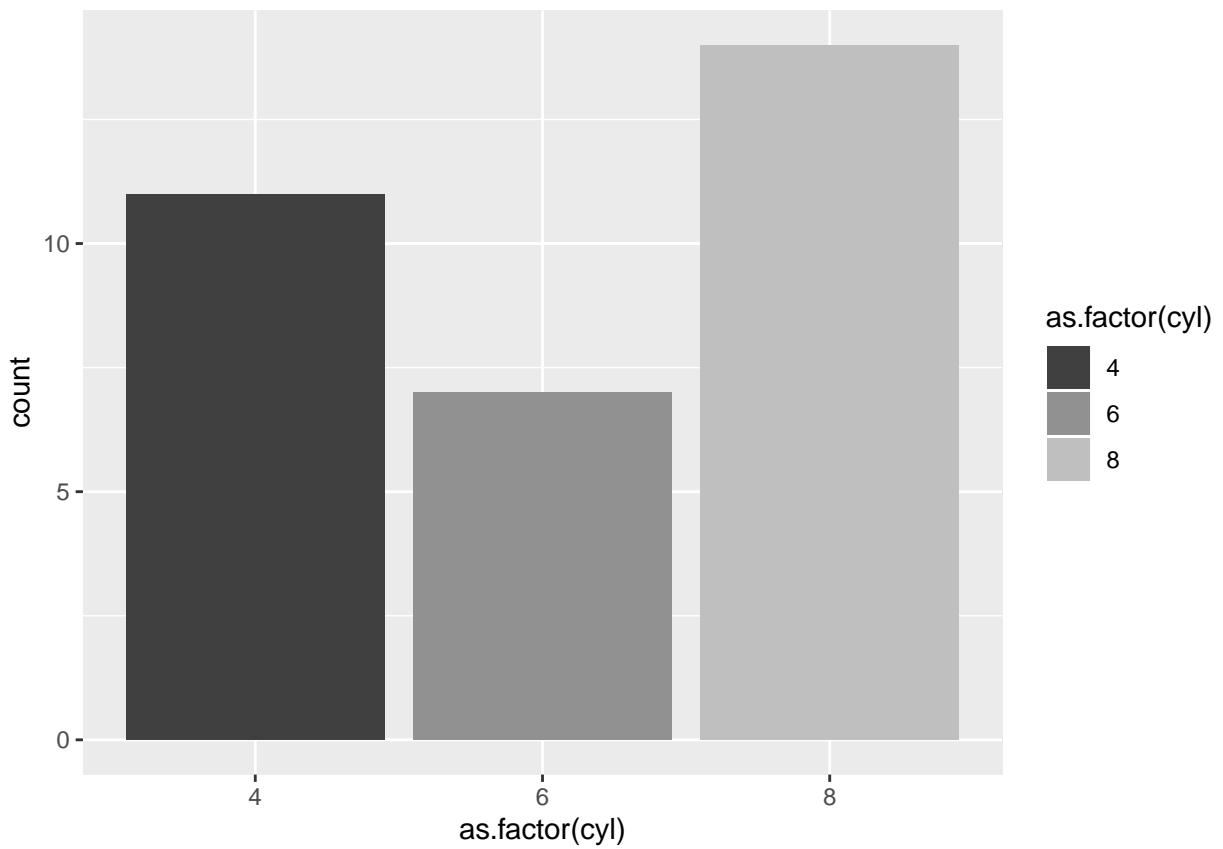
- `scale_fill_manual()` : pour utiliser des couleurs personnalisées.
- `scale_fill_brewer()` : pour utiliser les palettes de couleurs du package RColorBrewer.
- `scale_fill_grey()` : pour utiliser la palette de couleurs grises.
- `scale_fill_hue()` : pour utiliser différentes teintes de couleurs.

```
ggplot(data=mtcars, aes(x=as.factor(cyl), fill=as.factor(cyl) )) + geom_bar( ) +scale_fill_hue(c = 40)
```



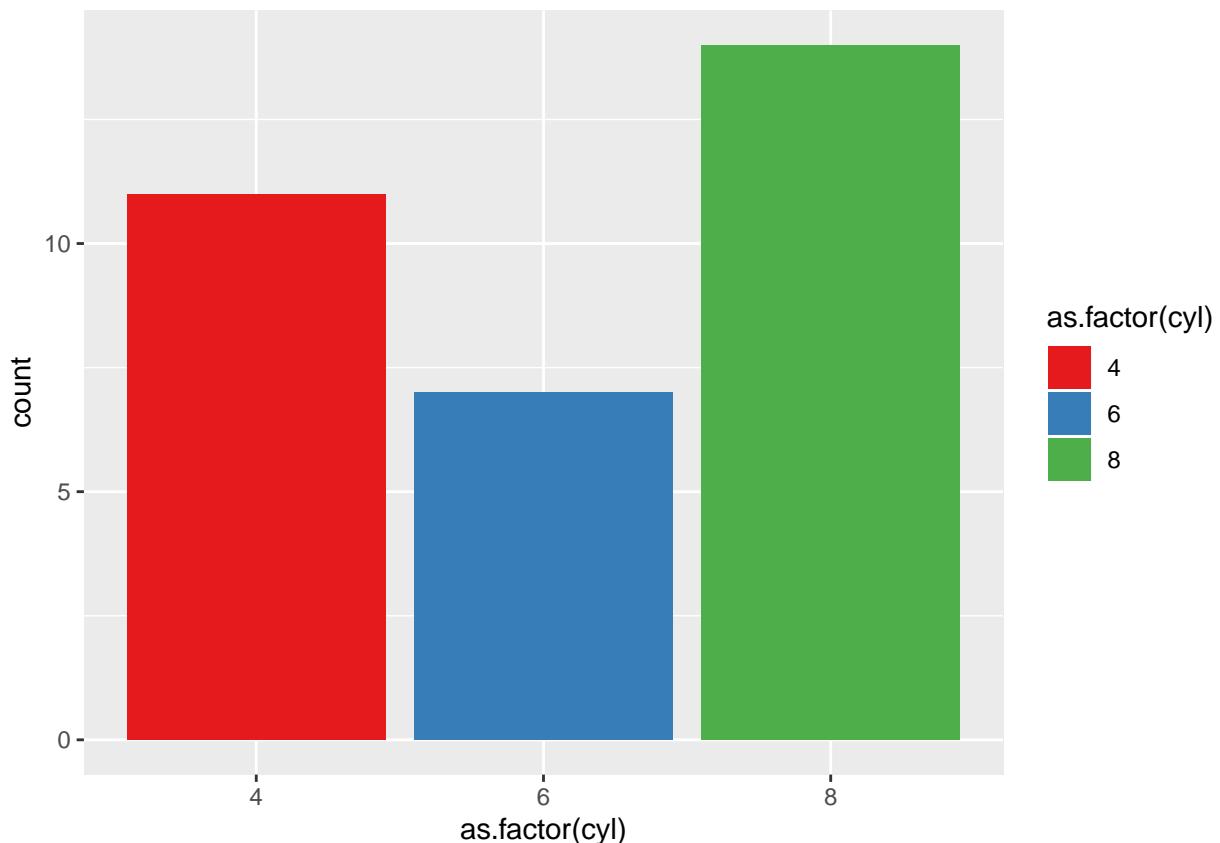
Si l'on veut obtenir une nuance de gris.

```
ggplot(data=mtcars, aes(x=as.factor(cyl), fill=as.factor(cyl) )) + geom_bar( ) +  
  scale_fill_grey(start = 0.25, end = 0.75)
```



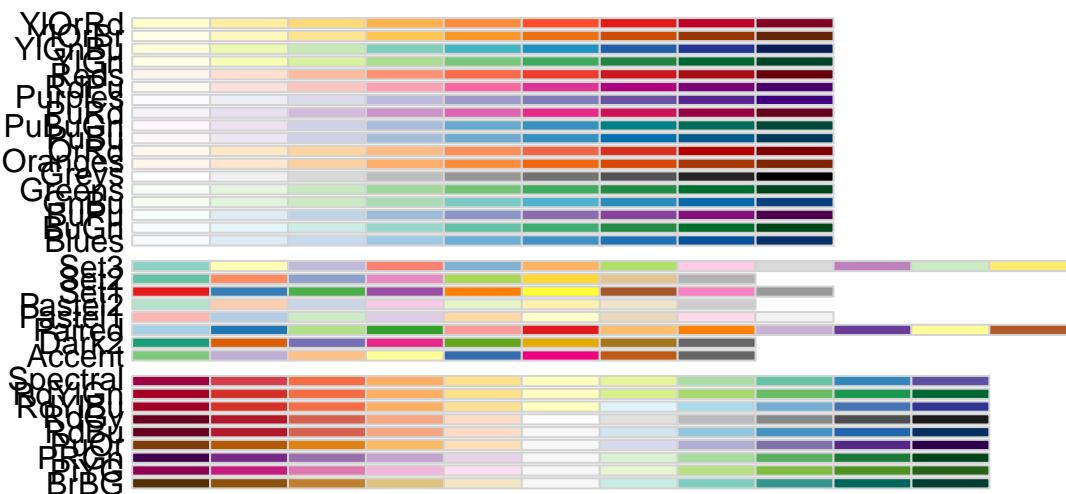
Si l'on souhaite utiliser la palette de R.

```
ggplot(data=mtcars, aes(x=as.factor(cyl), fill=as.factor(cyl) )) + geom_bar() +scale_fill_brewer(palette="Greys")
```



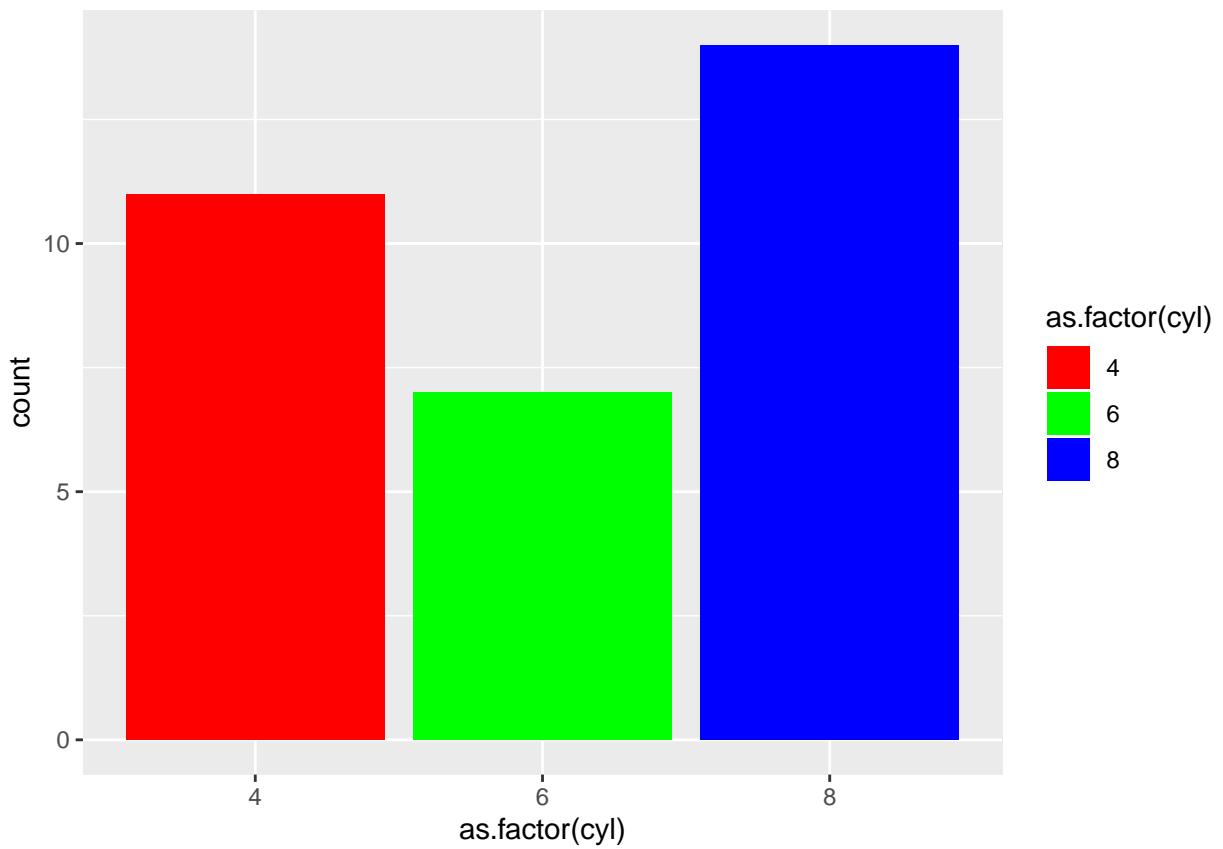
Pour afficher les palettes dont dispose R.

```
display.brewer.all()
```



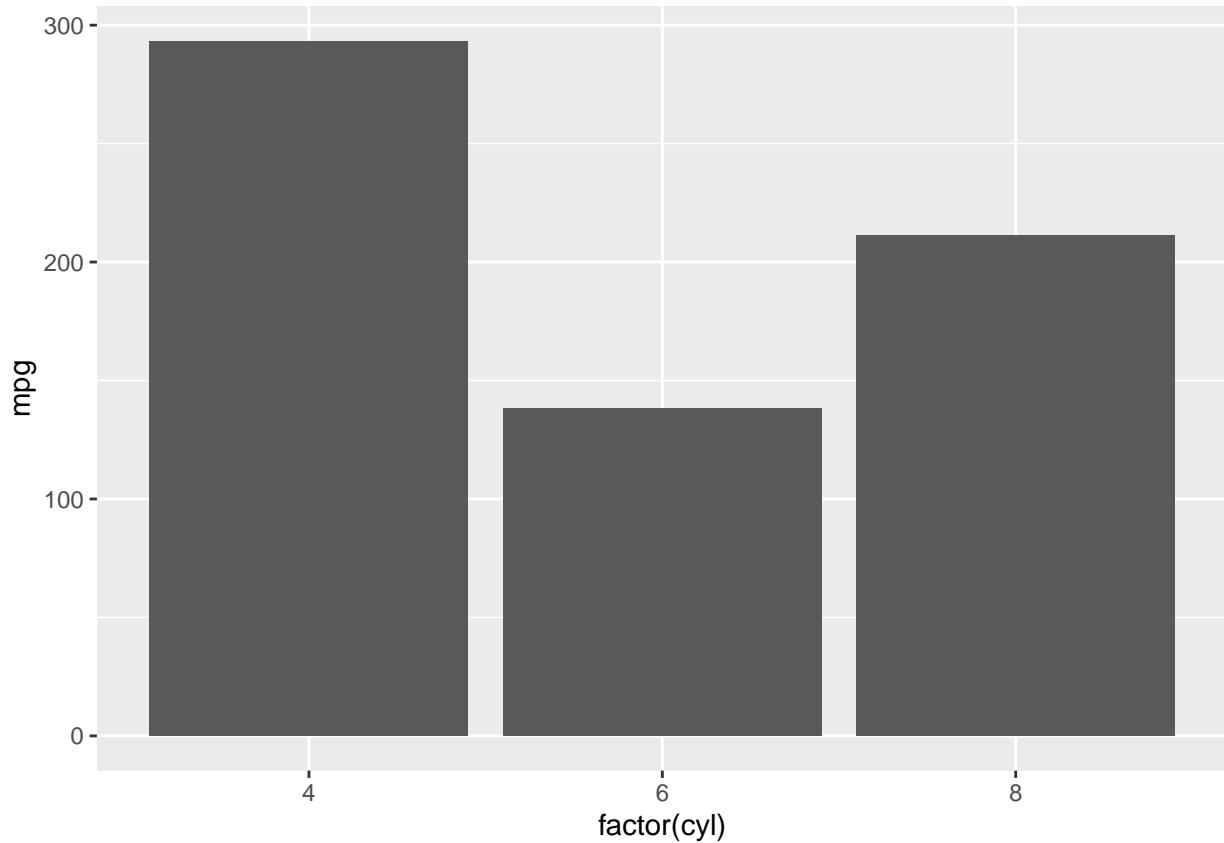
Si l'on souhaite utiliser la palette de R en spécifiant chaque couleur.

```
ggplot(data=mtcars, aes(x=as.factor(cyl), fill=as.factor(cyl) ) ) + geom_bar( ) +  
  scale_fill_manual(values = c("red", "green", "blue") )
```



Si l'on veut comparer la somme des valeurs de `mpg` selon les modalités de `cyl`.

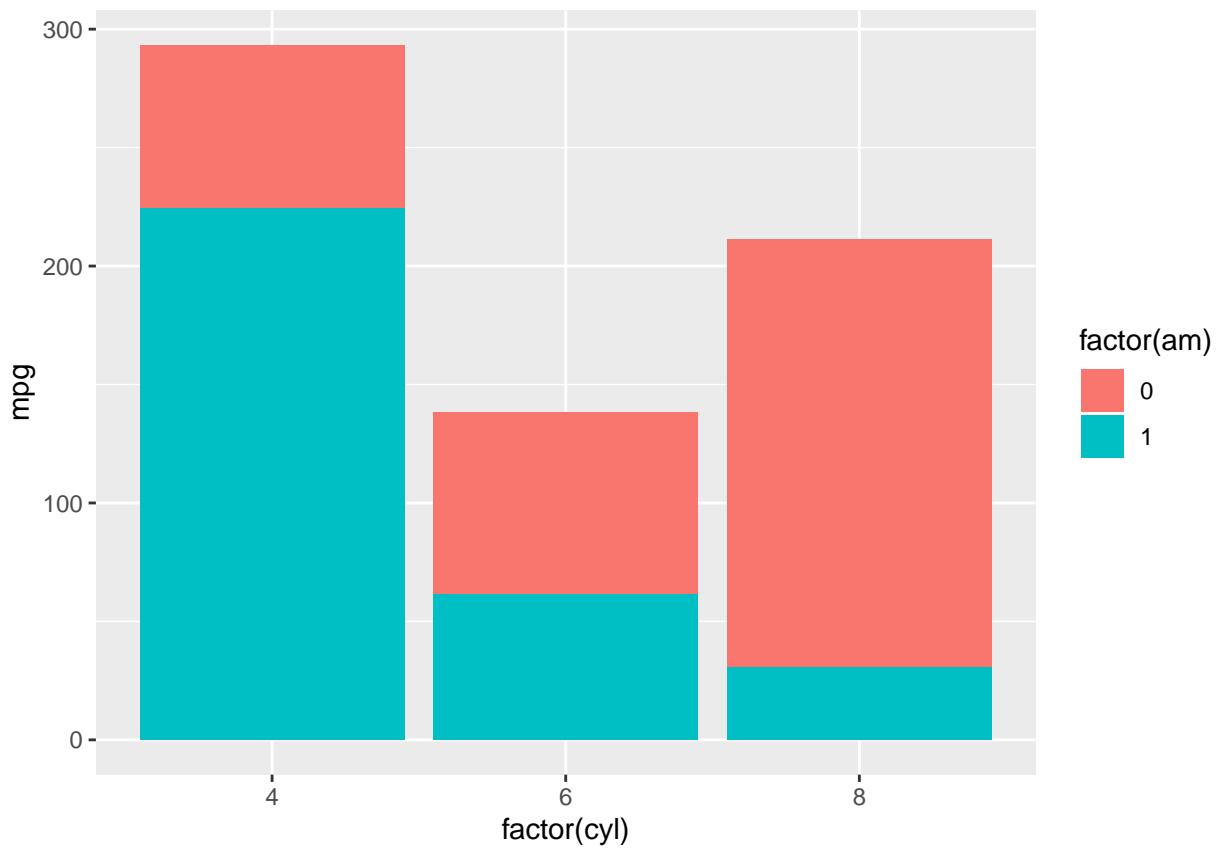
```
ggplot(data=mtcars, aes(x=factor(cyl), y=mpg)) + geom_bar(stat="identity")
```



Cela correspond à la somme de `mpg` dans les croisements `tapply(mtcars$mpg, mtcars$cyl, sum)`.

Si l'on veut comparer la somme des valeurs de `mpg` selon les modalités du croisement entre les variables `am` et `cyl` sous forme empilée. On parle de stacked bar.

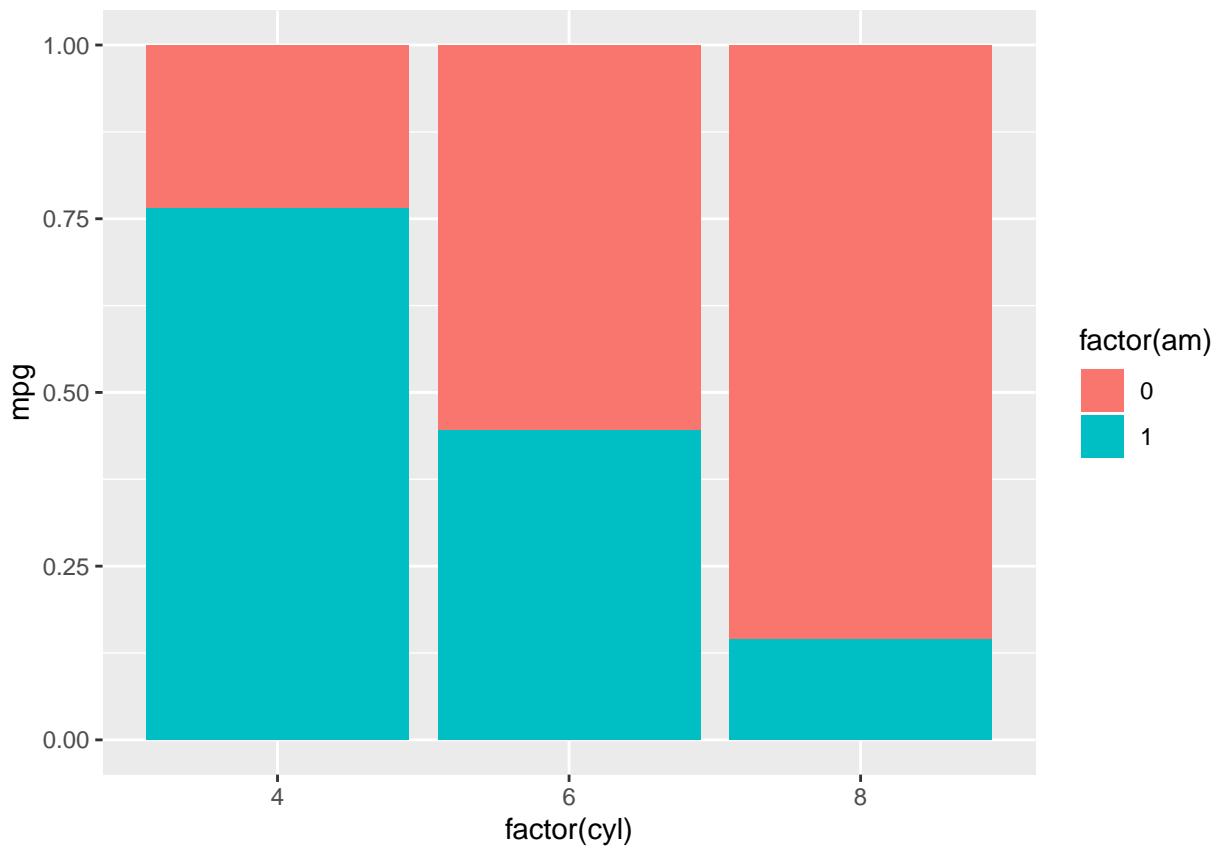
```
ggplot(data=mtcars, aes(fill=factor(am), y=mpg, x=factor(cyl)))+geom_bar(stat="identity")
```



Cela correspond à la somme de mpg dans les croisements `tapply(mtcars$mpg, list(mtcars$cyl, mtcars$am), sum)`.

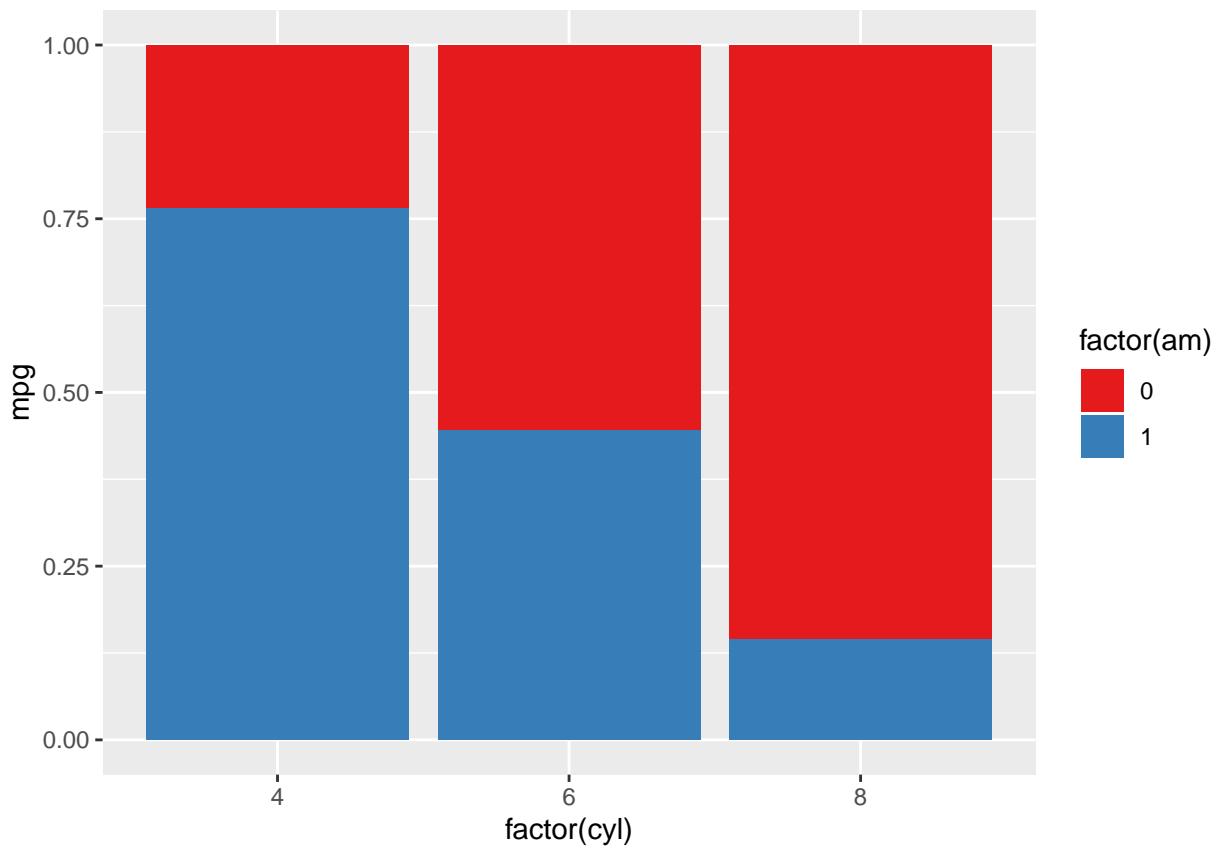
En pourcentage à l'aide de `position="fill"`

```
ggplot(data=mtcars, aes(fill=factor(am), y=mpg, x=factor(cyl))) +geom_bar(stat="identity", position="fill")
```



Si l'on souhaite utiliser la palette de R.

```
ggplot(data=mtcars, aes(fill=factor(am), y=mpg, x=factor(cyl))) +geom_bar(stat="identity",position="fill")
```

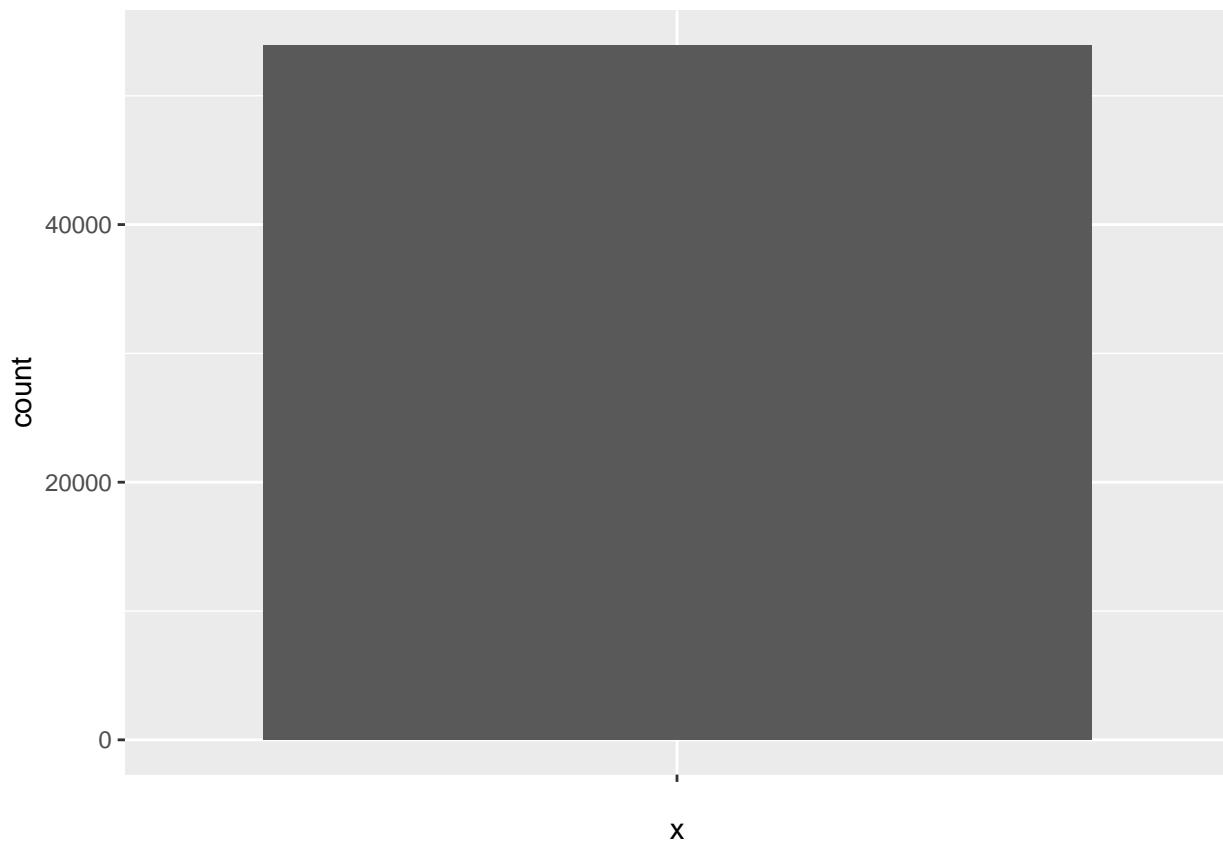


### Aspect en fonction d'une variable

L'avantage de cette grammaire est de définir les paramètres du graphique en fonction des variables. Ceci est assez évident pour les coordonnées  $x$  ou  $y$ , mais il est aussi possible de définir la couleur, la forme, la taille, ... en fonction de variables.

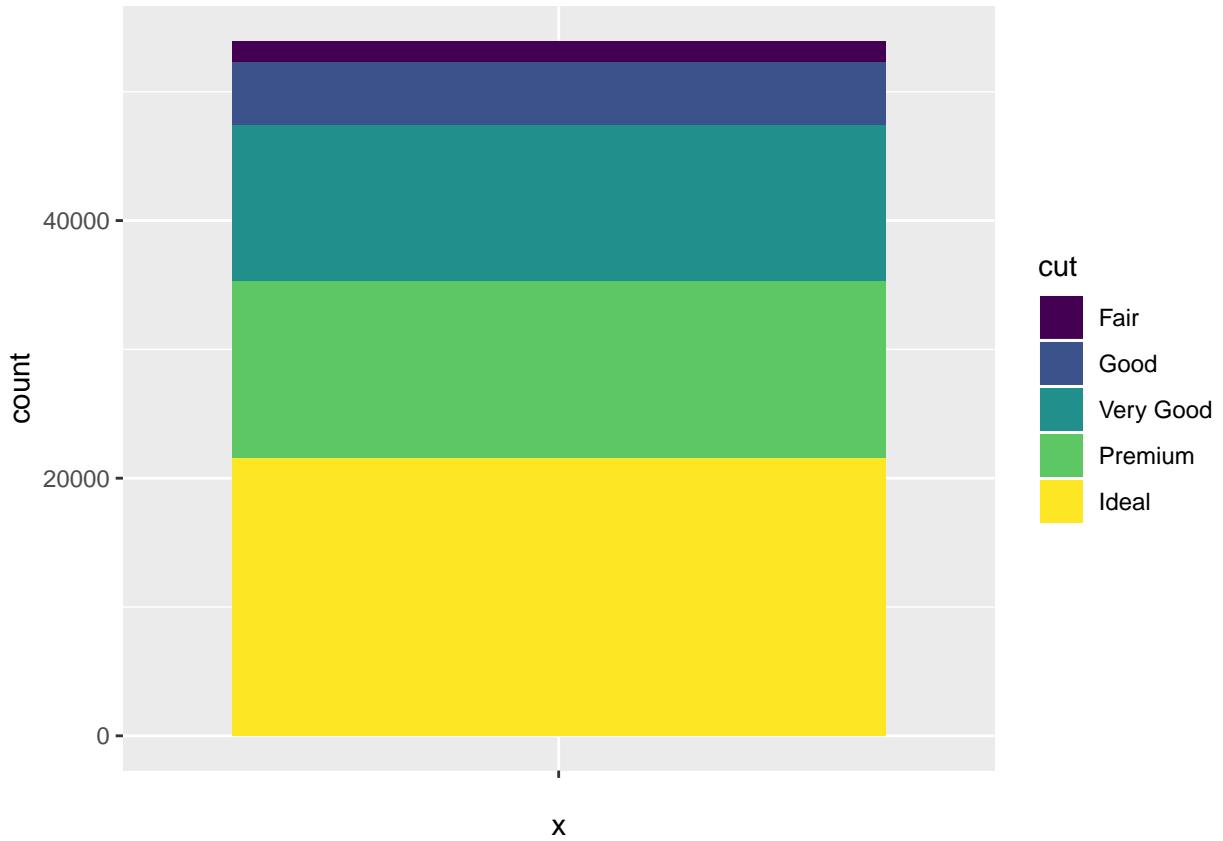
Ci-dessous, nous créons un diagramme en barres (avec `geom_bar()`) sur une constante (`x = ""`). Cela a pour le moment peu d'intérêt, sauf à montrer qu'il y a plus de 50000 diamants dans le jeu de données.

```
ggplot(diamonds, aes(x = "")) + geom_bar()
```



Dans le dataframe `diamonds`, nous disposons de la variable qualitative `cut`. Dans la suite, nous définissons une couleur en fonction de celle-ci (avec le paramètre esthétique `fill`). Ceci nous permet d'avoir un diagramme en barres empilées.

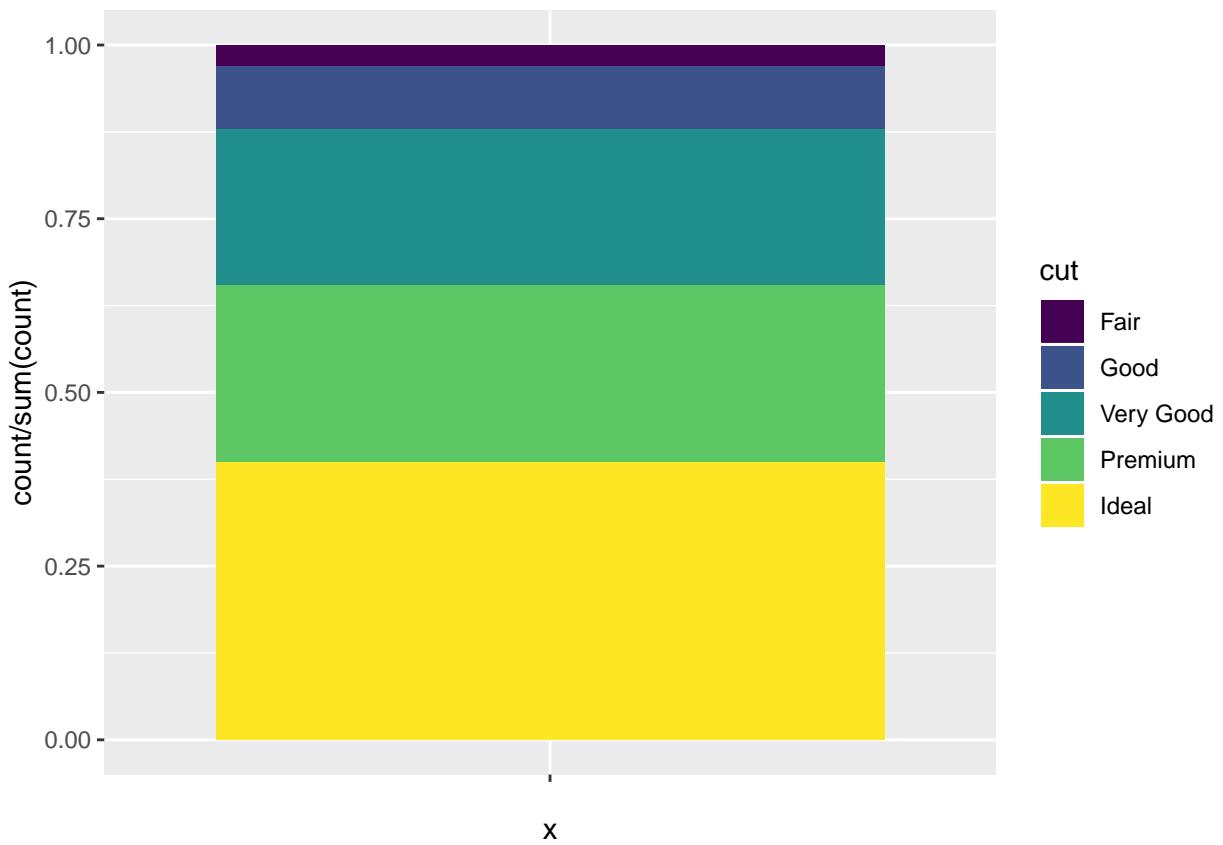
```
ggplot(diamonds, aes(x = "", fill = cut)) + geom_bar()
```



### Variable spécifique (.xxx.)

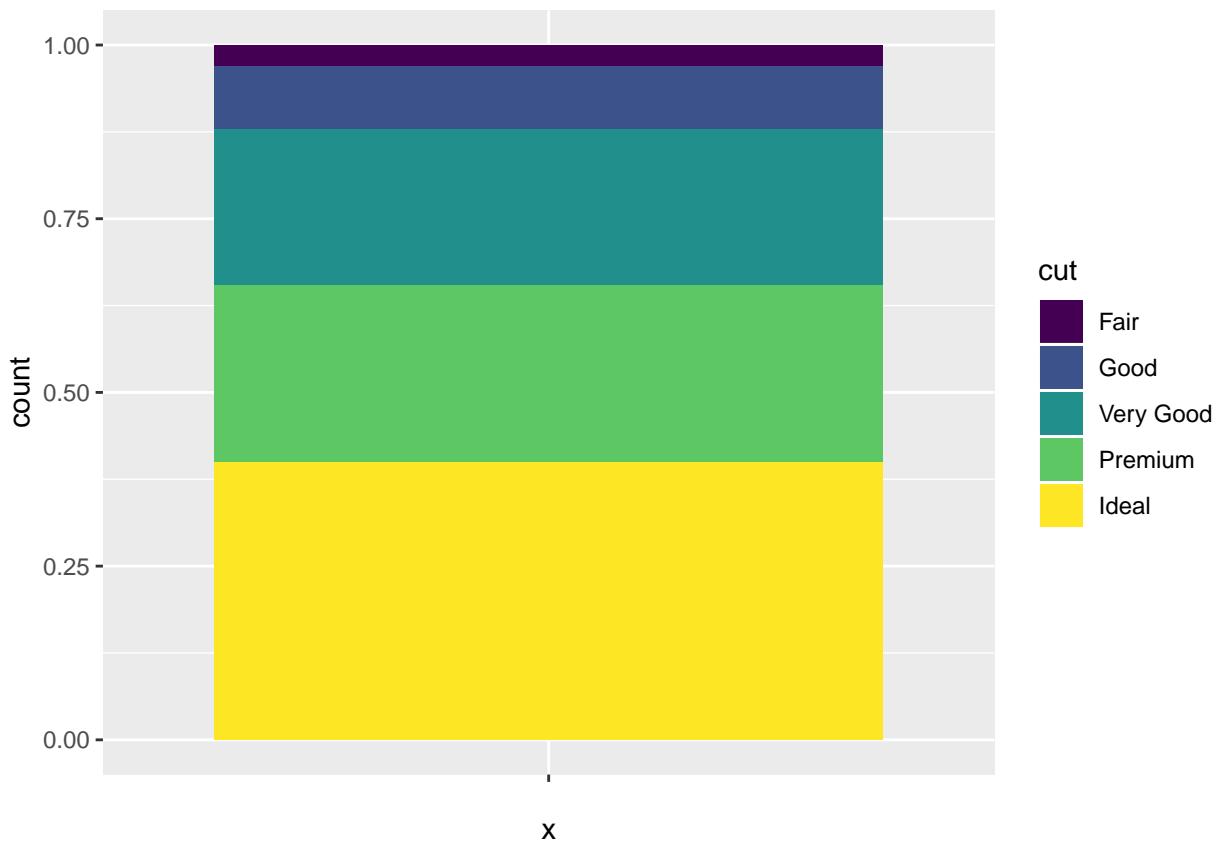
Dans le diagramme ci-dessous est présenté le dénombrement de chaque modalité de la variable. Ceci peut ne pas être particulièrement parlant, on peut préférer vouloir avoir des proportions (valeurs entre 0 et 1). C'est une bonne occasion pour utiliser les variables spécifiques créées par les fonctions de type `geom_xxx()`, qui sont de type `..xxx...`. Ici, `geom_bar()` crée la variable `..count..` qui représente le nombre de lignes pour chaque modalité. Donc, en redéfinissant les coordonnées `y` avec le calcul  $\frac{\text{..count..}}{\sum \text{..count..}}$ , nous réajustons les valeurs entre 0 et 1 (le graphique reste bien évidemment le même finalement).

```
ggplot(diamonds, aes(x = "", fill = cut)) +
  geom_bar(aes(y = ..count../sum(..count..)))
```



ou tout simplement avec l'instruction `position="fill"`.

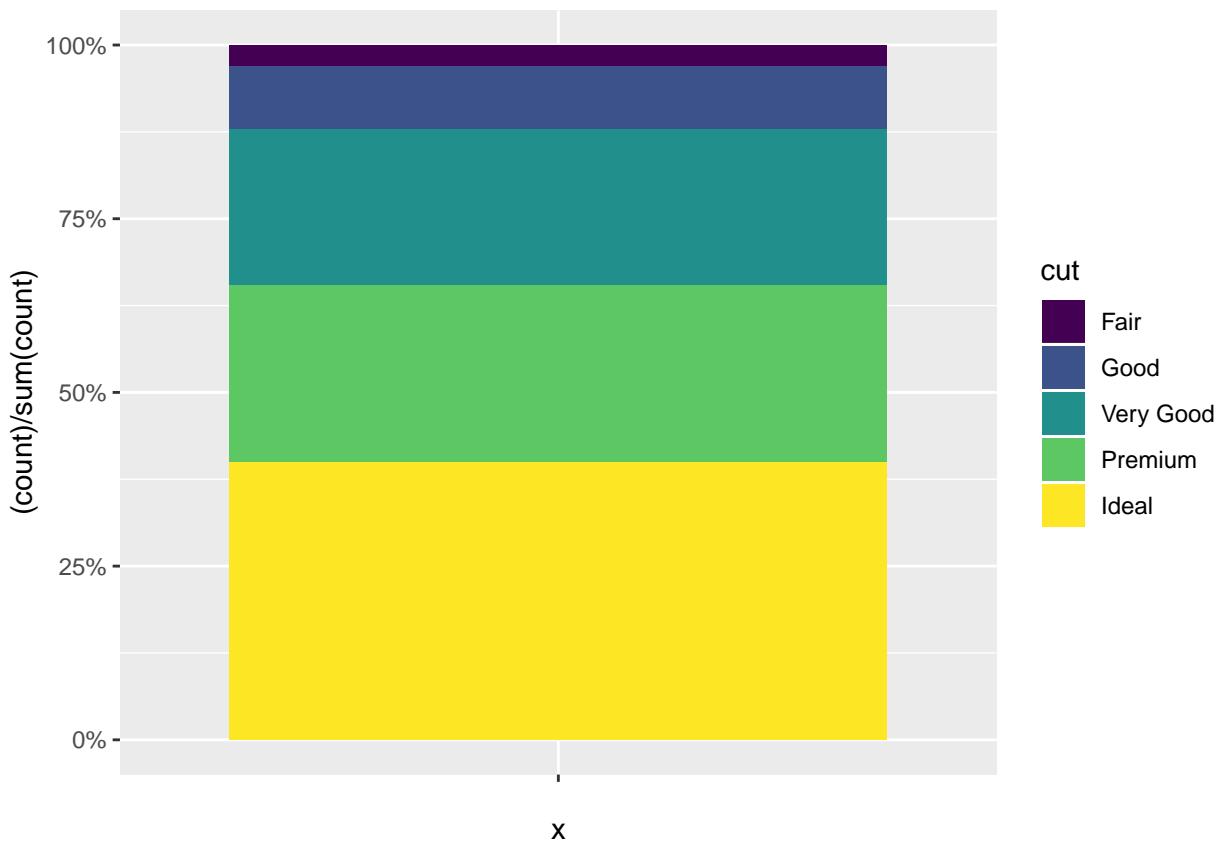
```
ggplot(diamonds, aes(x = "", fill = cut)) + geom_bar(position="fill")
```



### Changement d'échelle

Maintenant que nous avons des valeurs entre 0 et 1, il peut être intéressant de changer ces valeurs en pourcentage (entre 0 et 100 donc, et avec le sigle %). C'est possible avec ce qu'on va appeler un changement d'échelle, réalisable avec les fonctions de type `scale_xxx_yyy()` (où le `xxx` représente l'aspect esthétique à modifier et `yyy` le type de cet aspect). Dans notre cas, nous allons utiliser la fonction `percent()` du package `scales` pour afficher donc des valeurs en pourcentage, plutôt qu'en proportion.

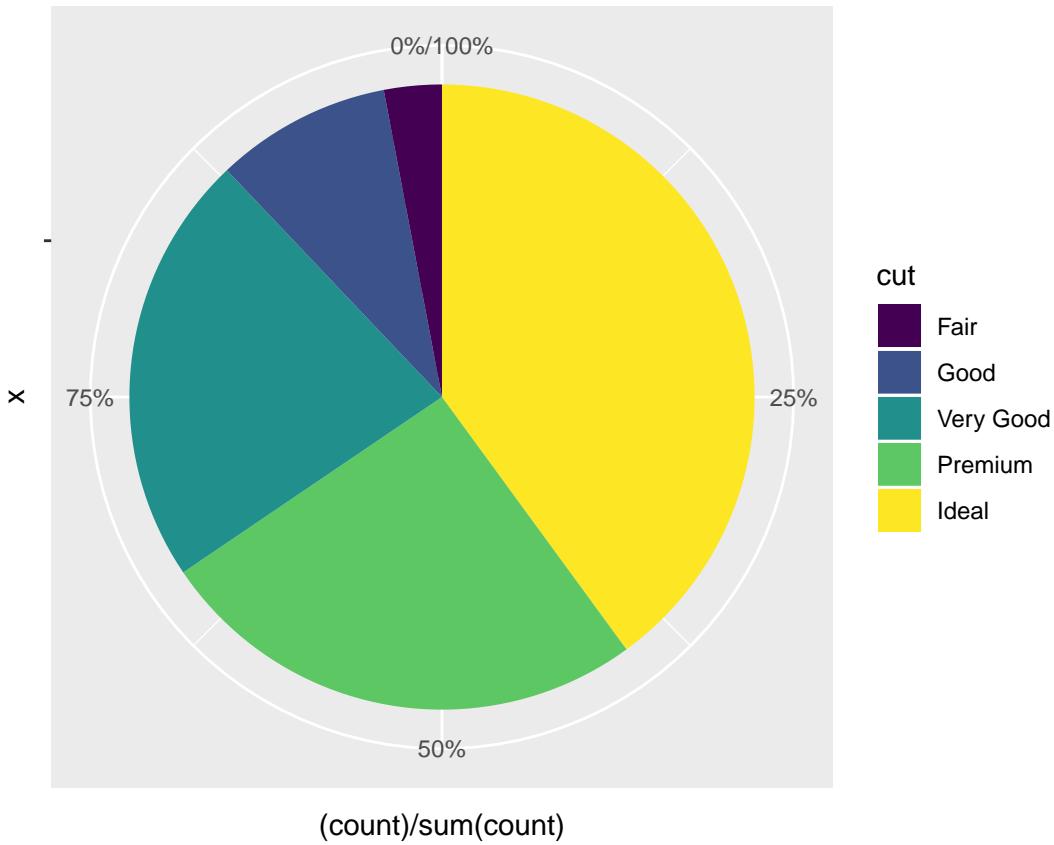
```
ggplot(diamonds, aes(x = "", fill = cut)) +
  geom_bar(aes(y = (..count..)/sum(..count..))) +
  scale_y_continuous(labels = percent)
```



### Changement de système de coordonnées

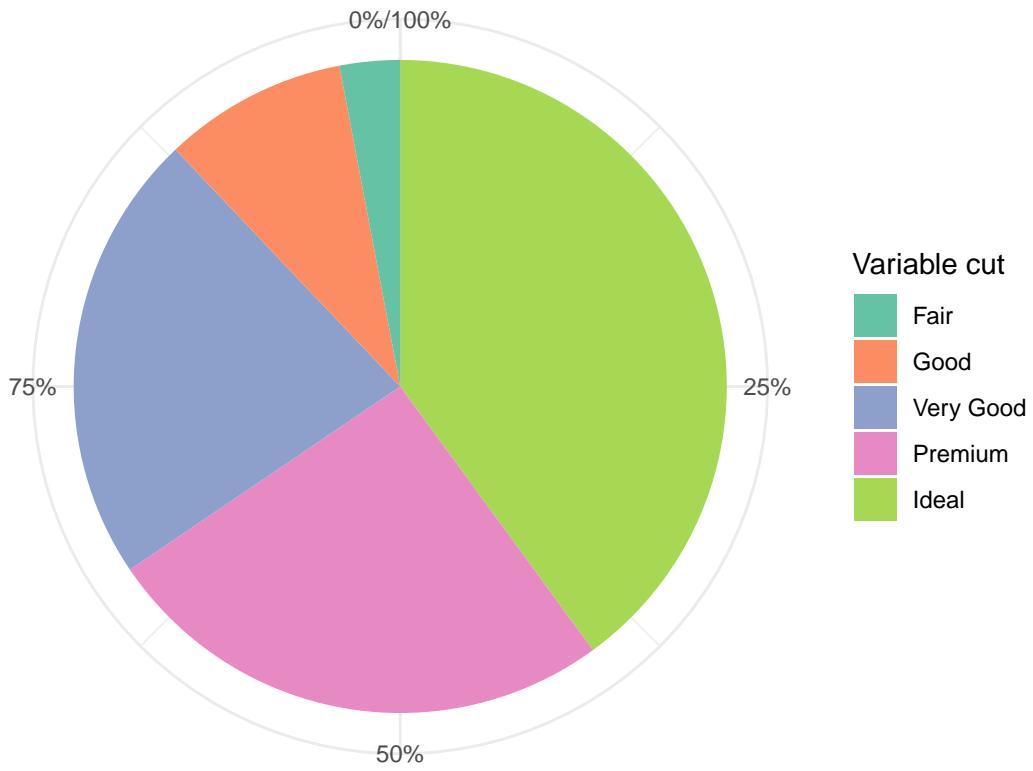
Pour continuer le tour des différentes possibilités de cette grammaire, nous allons créer maintenant un diagramme circulaire. Celui-ci peut être vu comme un diagramme en barres empilées (comme ci-dessous), avec un changement de système de coordonnées. En effet, si on considère les données en coordonnées polaires, avec la variable  $y$  définissant l'angle, nous obtenons ce que nous souhaitons. Ceci est réalisable avec les fonctions de type `coord_xxx()` (ici `coord_polar()` donc, en indiquant que l'angle `theta` dépend de  $y$ ). Nous ajoutons l'option `width = 1` dans `geom_bar()` pour éviter d'avoir un trou au milieu du cercle.

```
ggplot(diamonds, aes(x = "", fill = cut)) +
  geom_bar(aes(y = (..count..)/sum(..count..)), width = 1) +
  scale_y_continuous(labels = percent) +
  coord_polar(theta = "y")
```



Suite à ce qui a été vu précédemment, on peut donc avoir un graphique circulaire propre avec le code suivant.

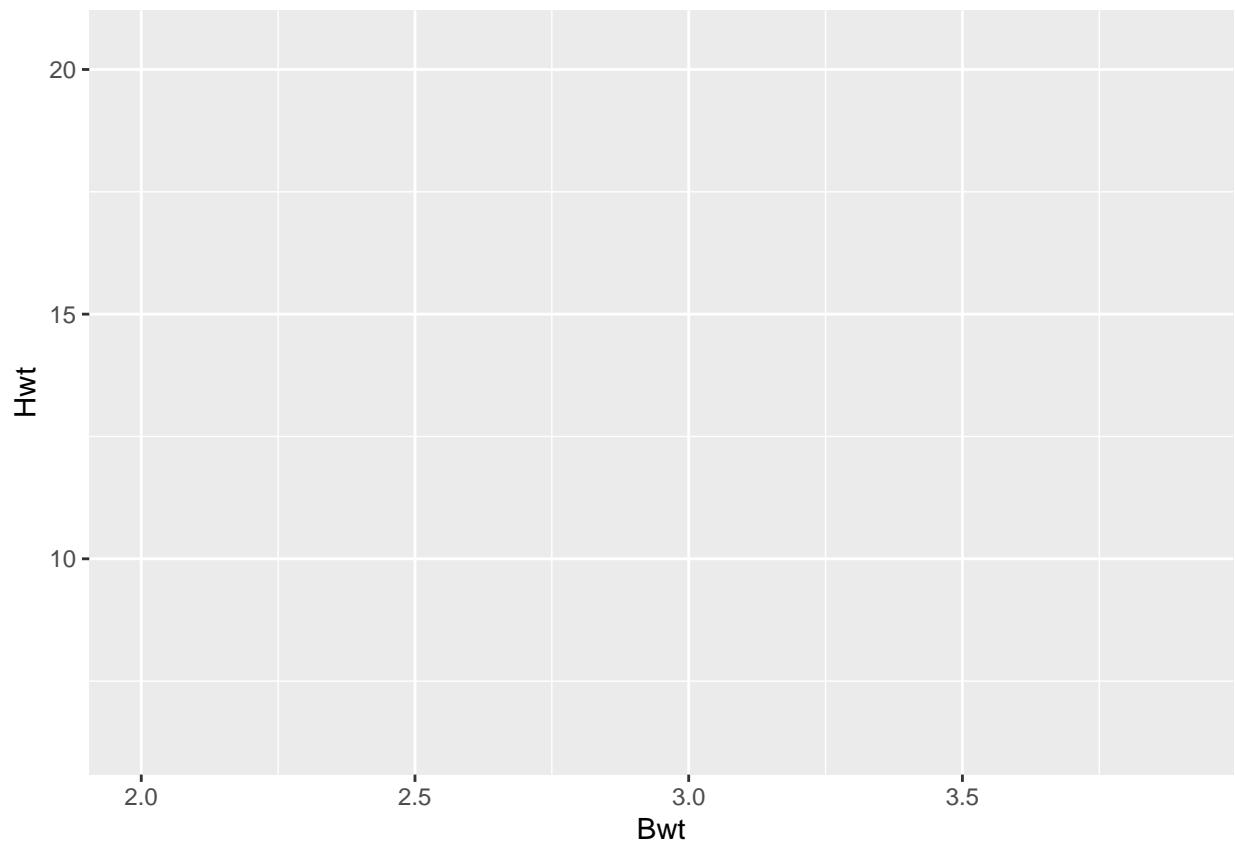
```
ggplot(diamonds, aes(x = "", fill = cut)) +
  geom_bar(aes(y = (..count..)/sum(..count..)), width = 1) +
  scale_fill_brewer(palette = "Set2") +
  scale_y_continuous(labels = percent) +
  coord_polar(theta = "y") +
  theme_minimal() +
  theme(axis.title = element_blank()) +
  labs(fill = "Variable cut")
```



### Superposition de graphique (geom\_point, geom\_smooth)

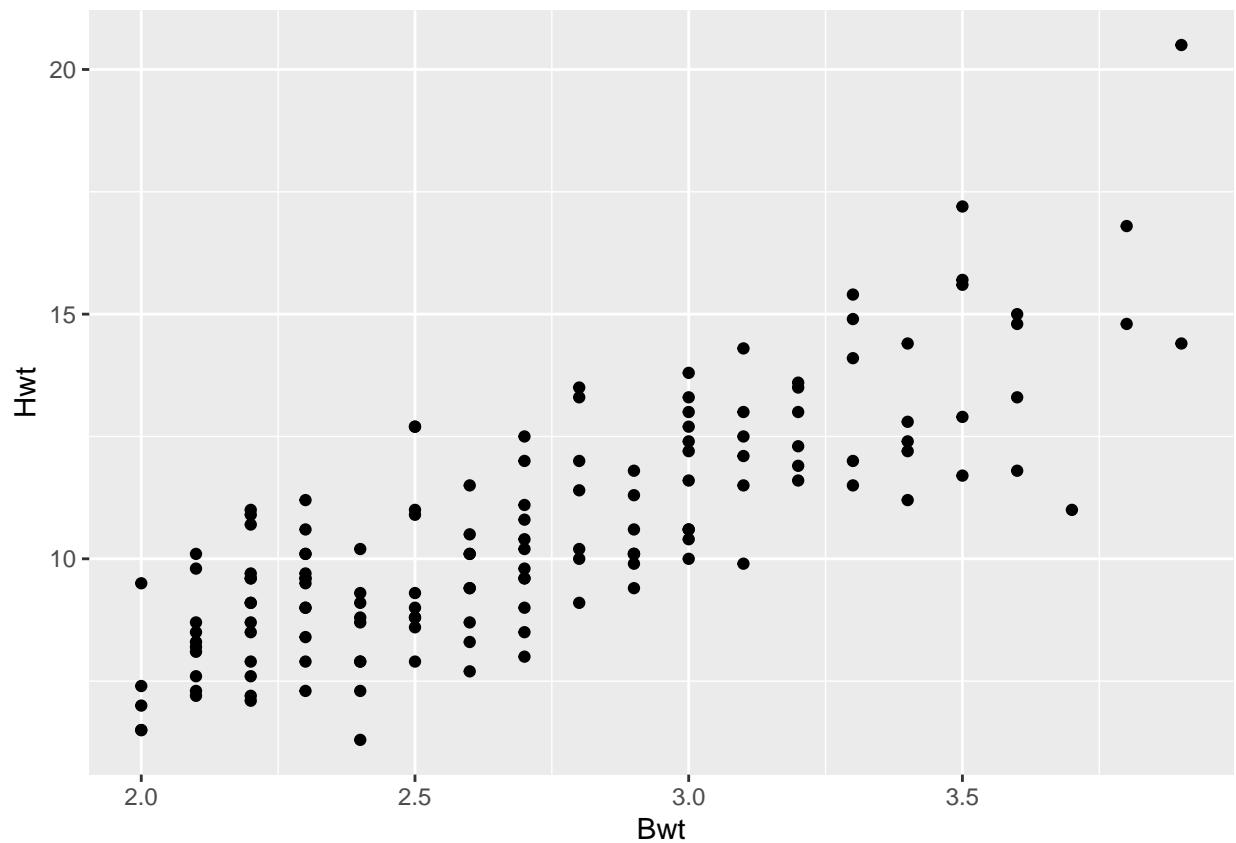
Grace à la logique de construction des graphiques, via la grammaire, il est ais  de superposer diff entes repr sentations, simplement en ajoutant les couches. Ci-dessous, nous dessinons le graphique entre les variables `hp` et `mpg`.

```
nuage=ggplot(data=cats, aes(x = Bwt, y =Hwt))  
nuage
```



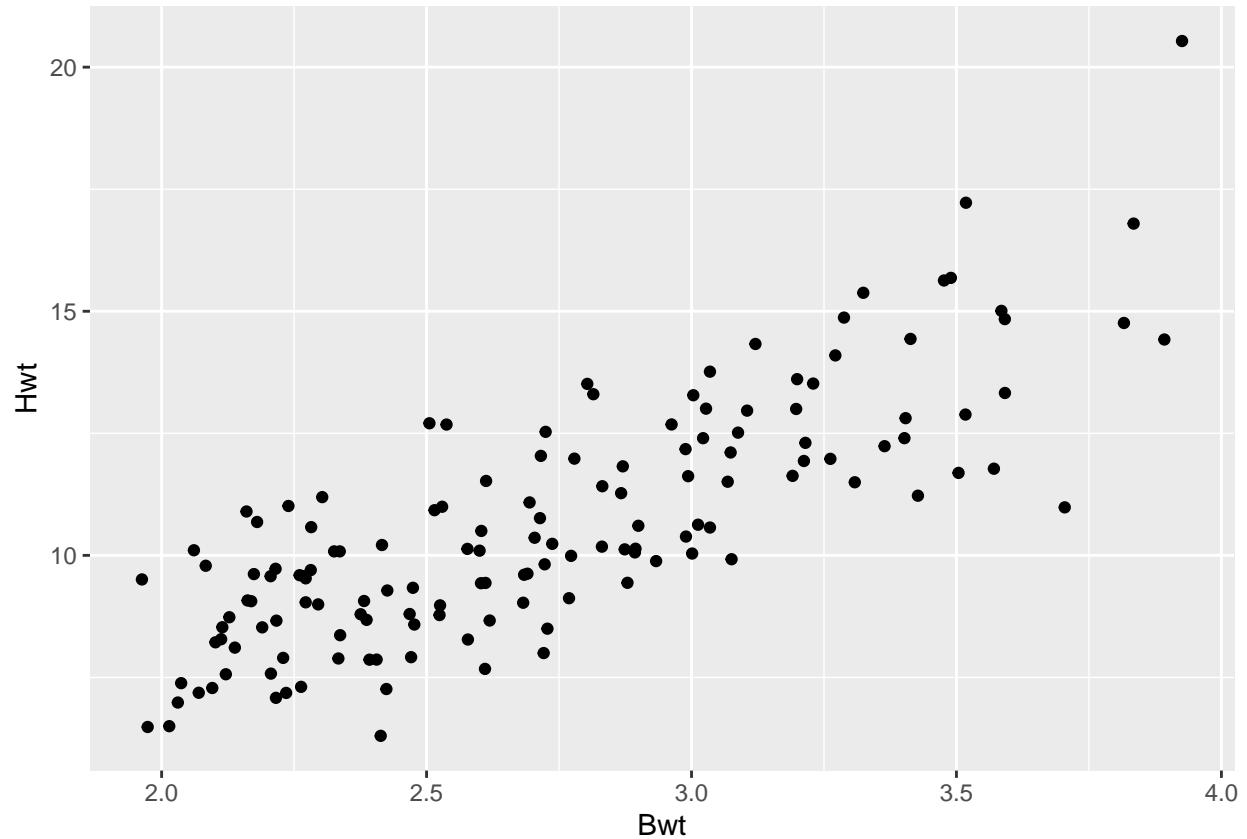
On peut dès lors rajouter le nuage de points à l'aide de `geom_point()`.

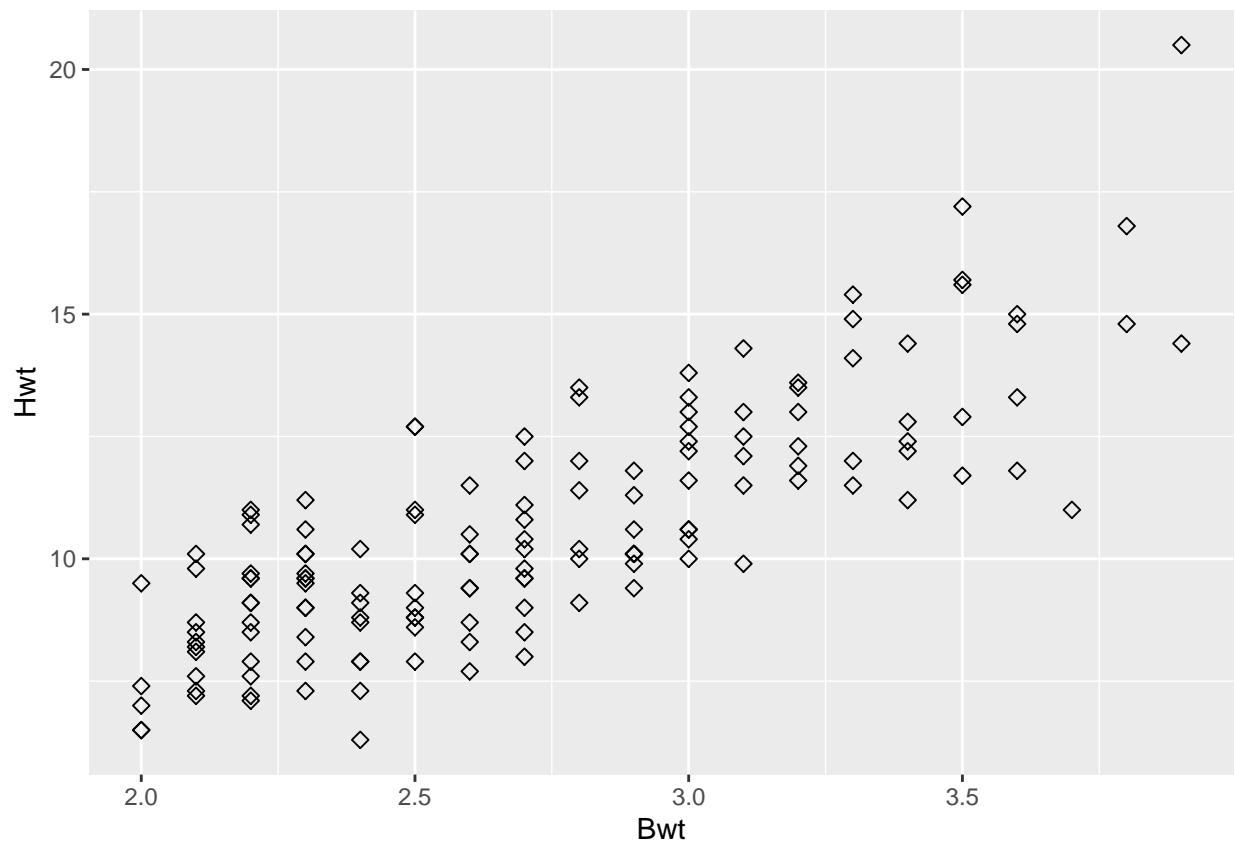
```
nuage+geom_point()
```



Si l'on veut décaler les points superposés.

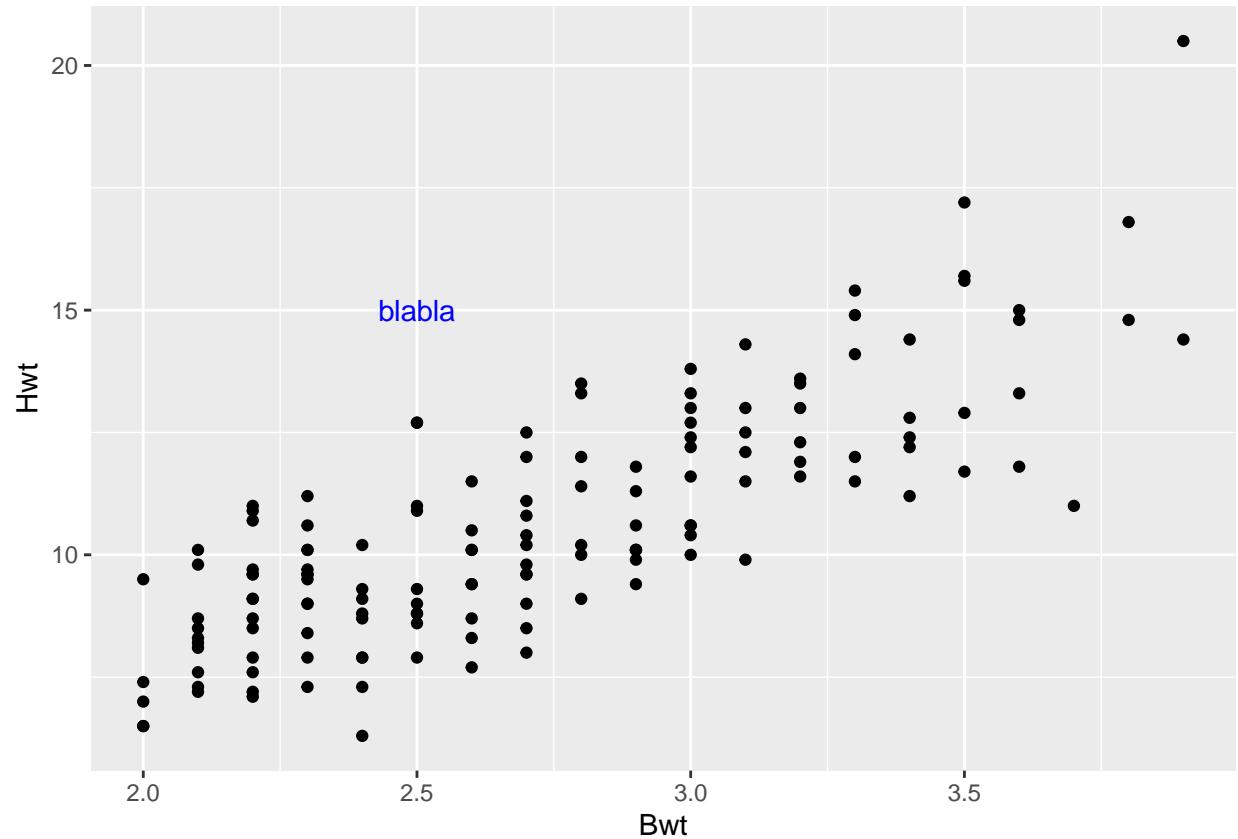
```
nuage+geom_point(position="jitter")
```





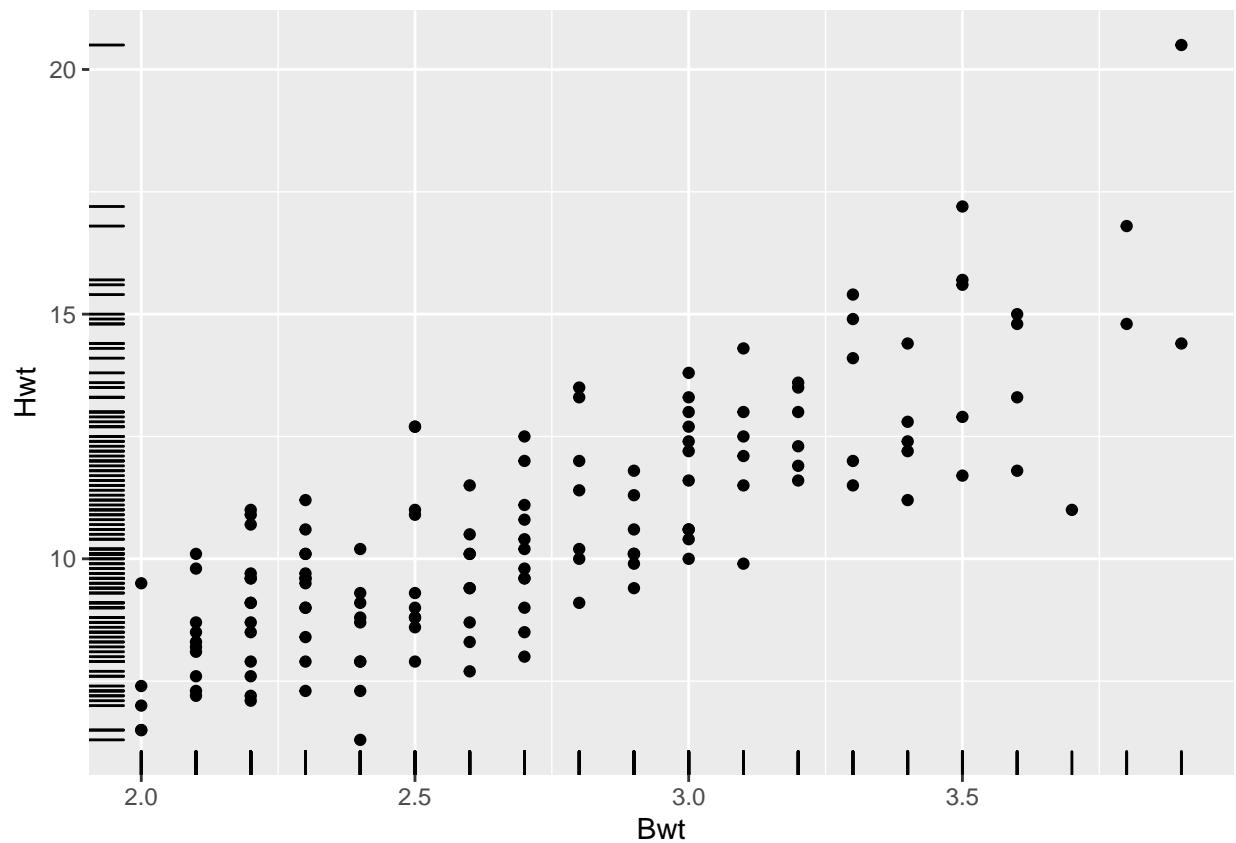
Il est facile d'ajouter du texte sur un graphique avec `annotate()`.

```
nuage+geom_point() +annotate("text", x=2.5, y=15, label="blabla", colour="blue")
```



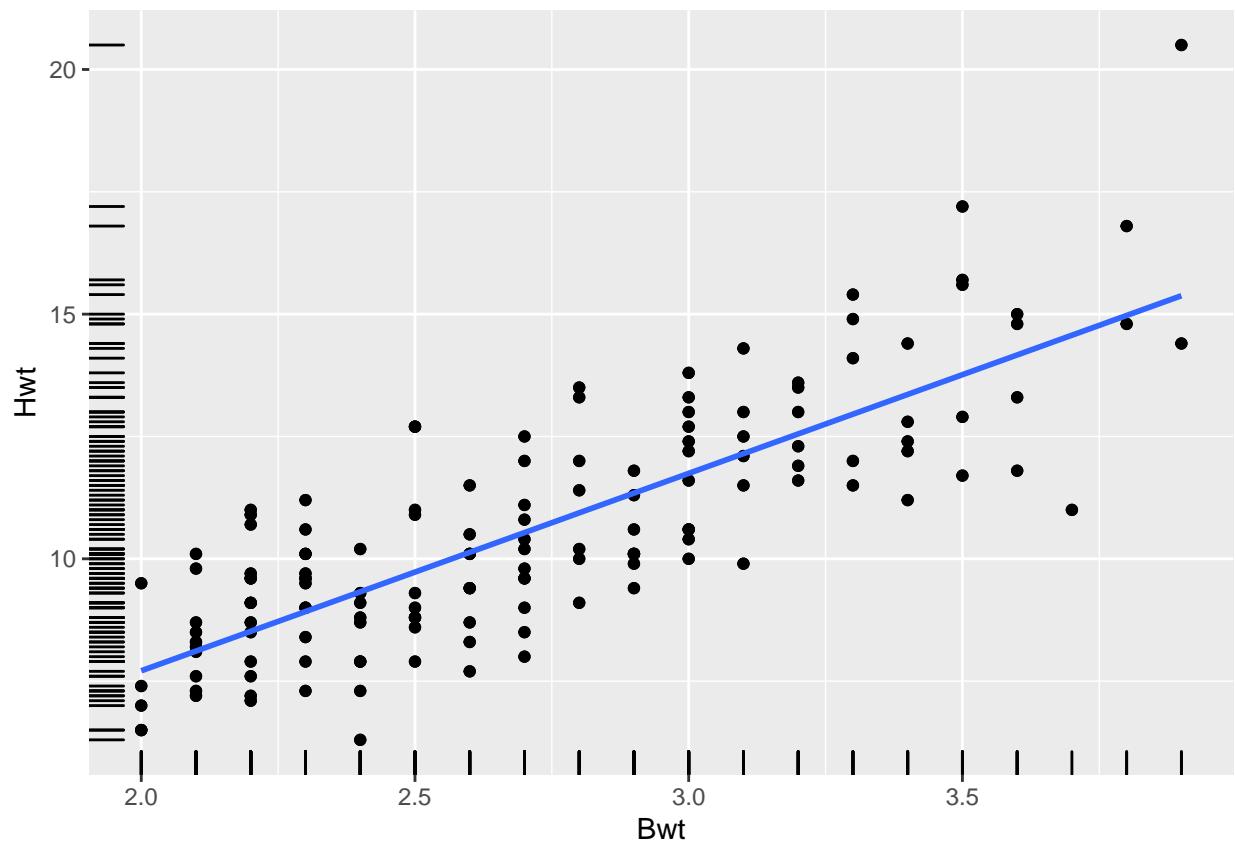
Il est facile à l'aide de `geom_rug` d'indiquer sur les axes les différentes valeurs prises par les variables.

`nuage+geom_point() +geom_rug()`



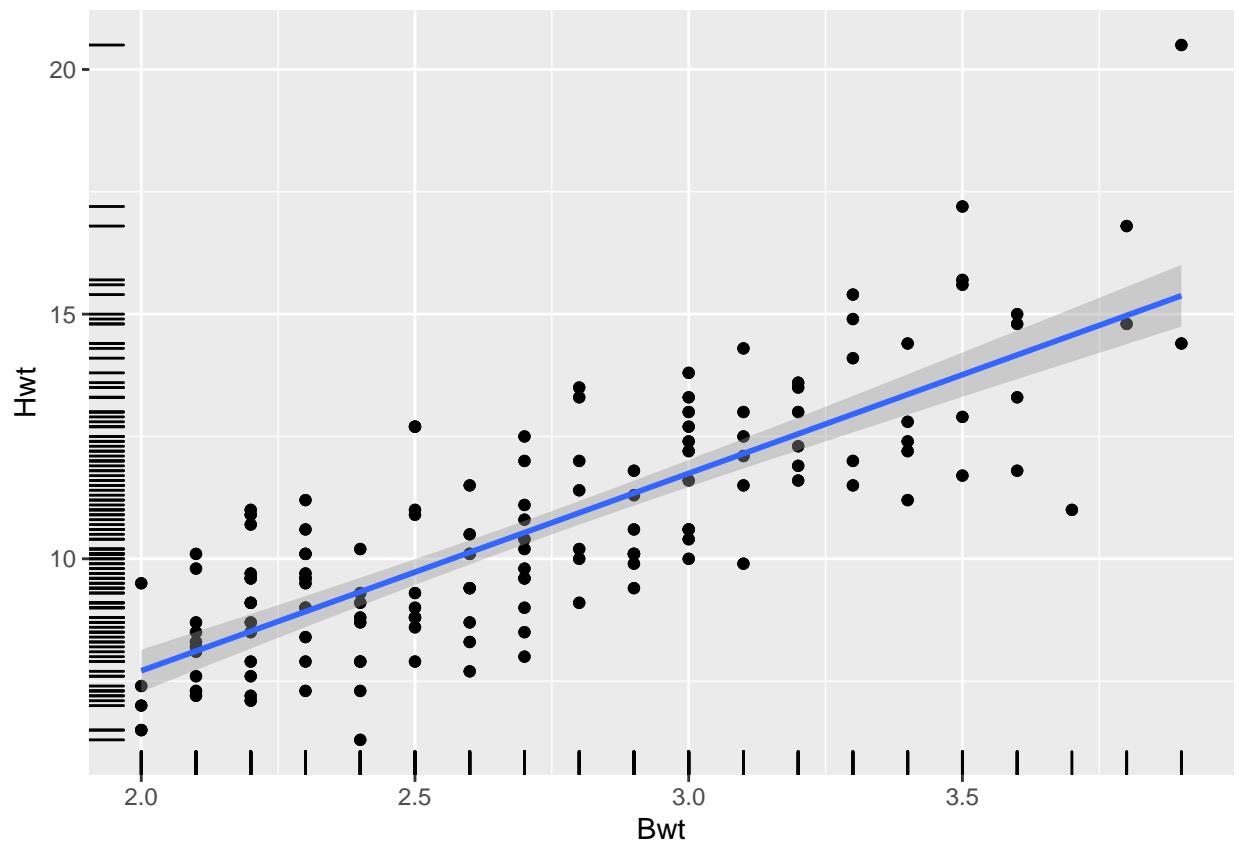
On peut très facilement y ajouter la droite des moindres carrés à l'aide de `geom_smooth`.

```
nuage+geom_point()+geom_point()+geom_rug() +geom_smooth(method = "lm", se = FALSE)
```



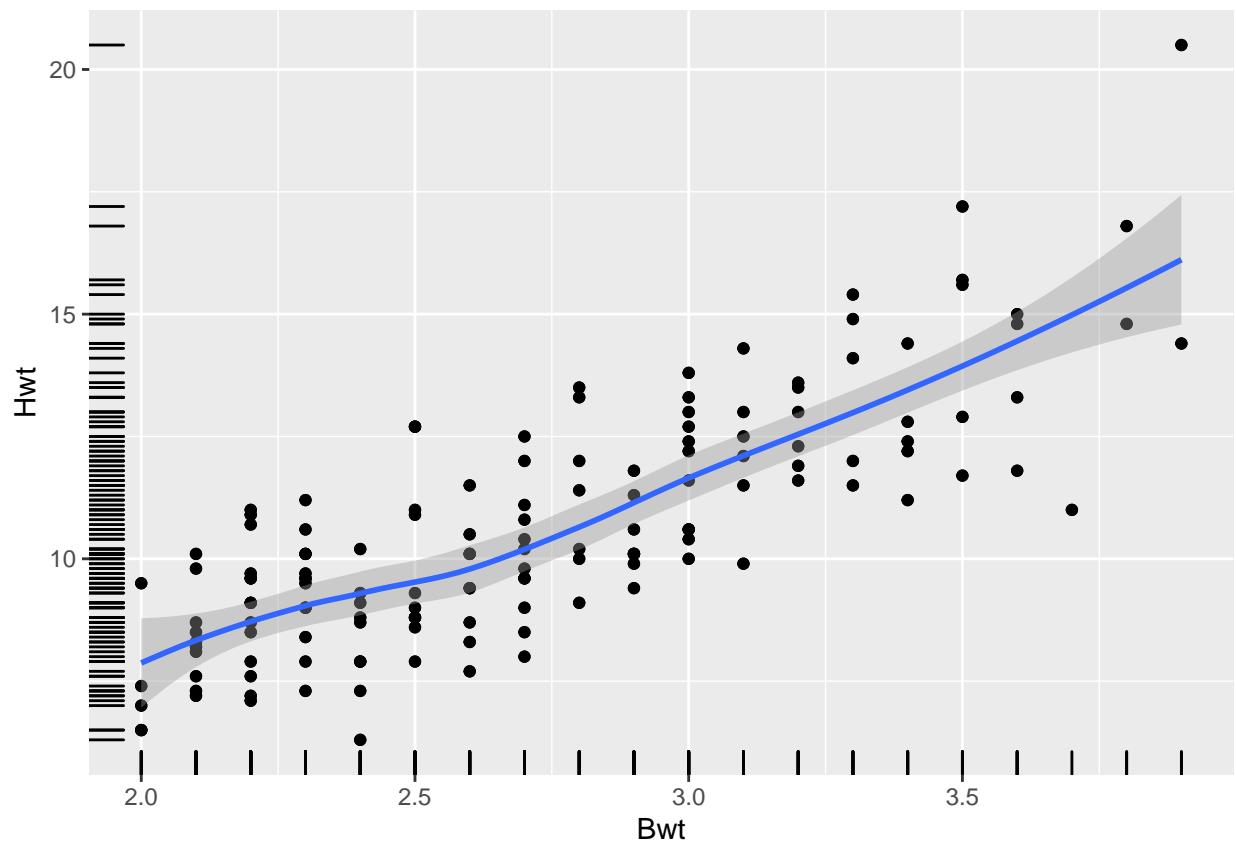
L'argument `se` de `geom_smooth` permet de faire apparaître ou non les intervalles de confiance.

```
nuage+geom_point()+geom_point()+geom_rug()+geom_smooth(method = "lm", se = TRUE)
```



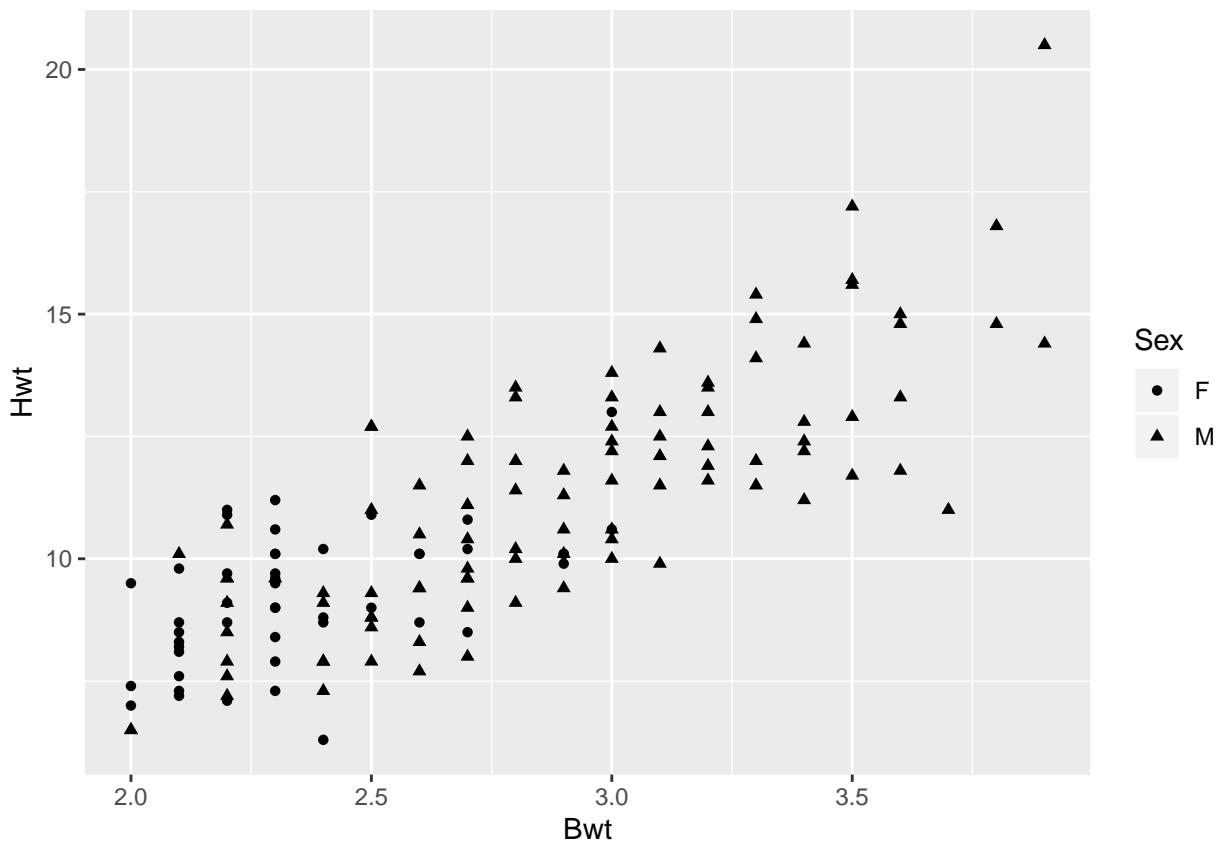
On peut effectuer un autre type d'ajustement qu'un ajustement linéaire qui fait apparaitre la tendance. C'est l'argument de base si on ne spécifie pas `method`.

```
nuage+geom_point()+geom_point()+geom_rug()+geom_smooth()
```



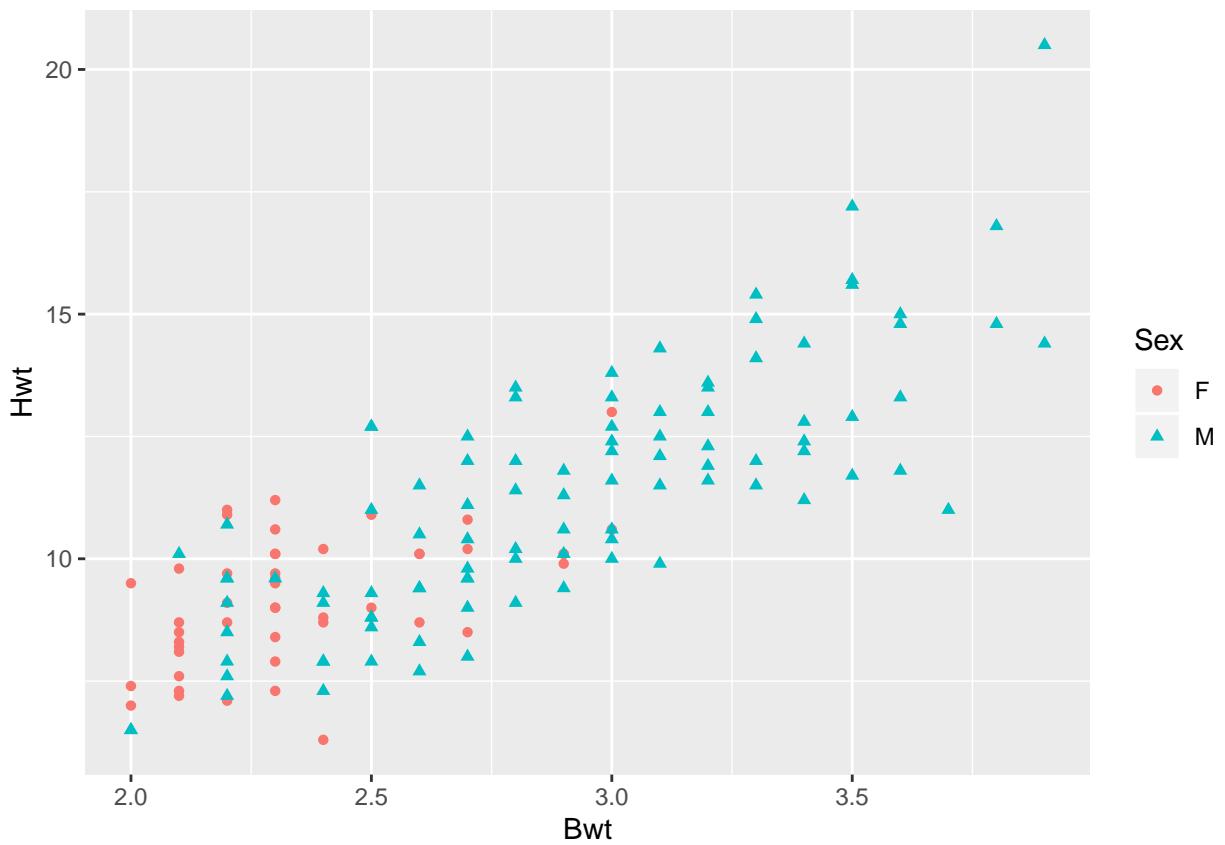
Il est possible de changer le type de points en fonction des niveaux de Sex

```
ggplot(data=cats, aes(x=Bwt, y=Hwt, shape=Sex))+geom_point()
```



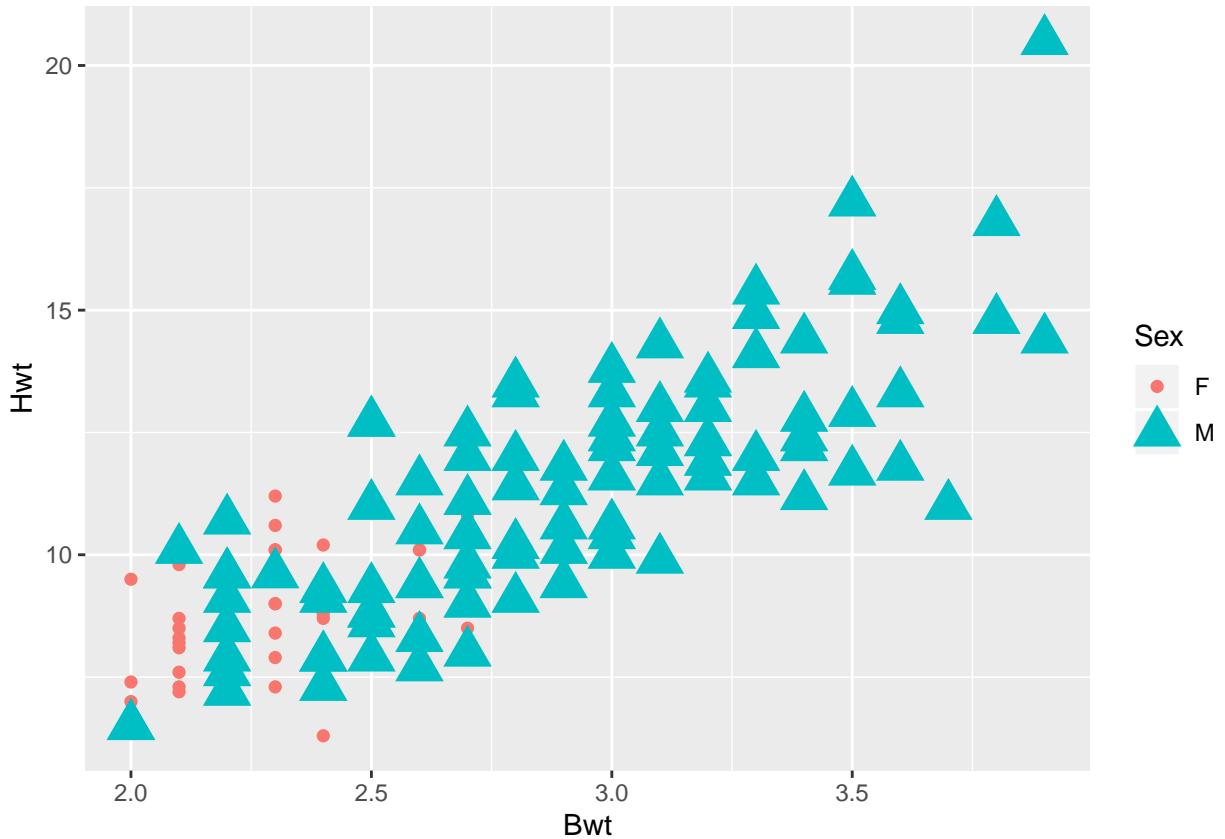
Il est possible de changer le type de points ainsi que leur couleur en fonction des niveaux de `Sex`

```
ggplot(data=cats, aes(x=Bwt, y=Hwt, color=Sex, shape=Sex))+geom_point()
```



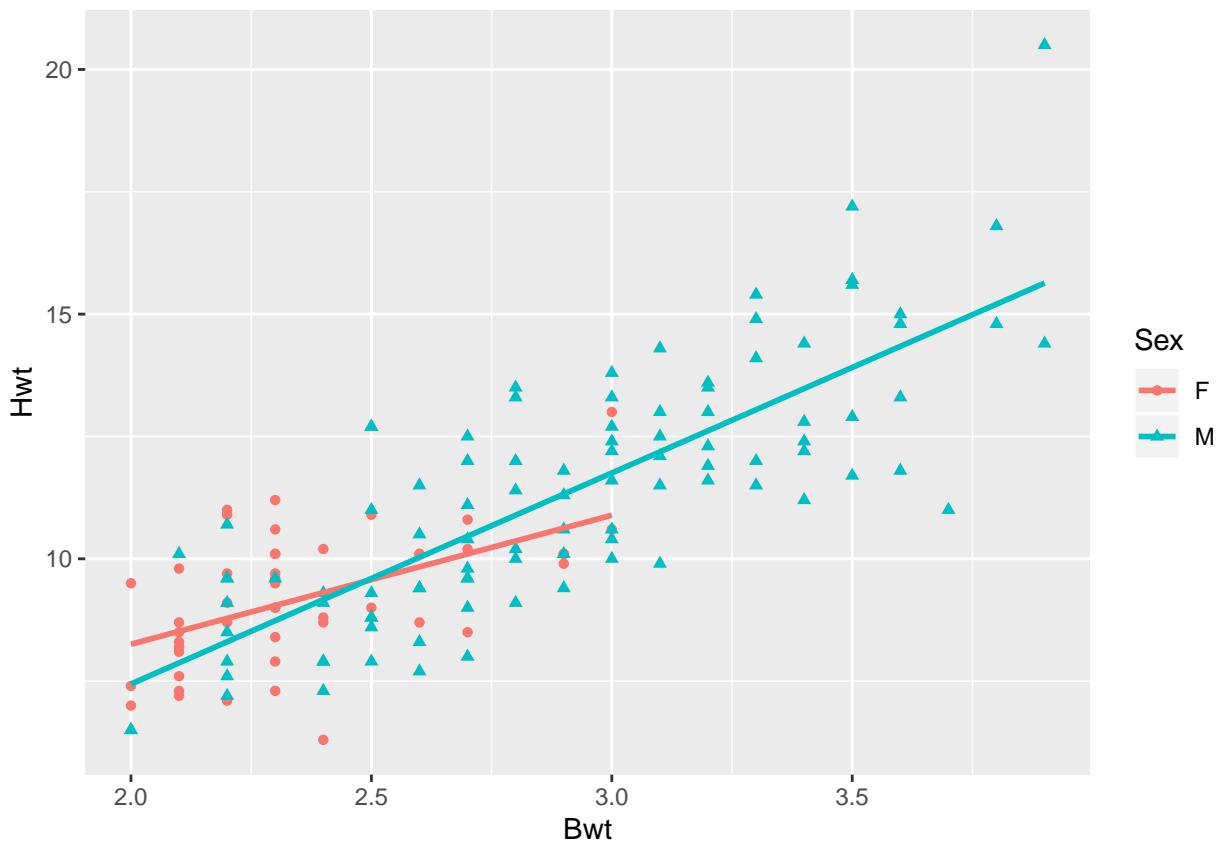
Il est possible de changer le type de points, leur couleur ainsi que leur taille en fonction des niveaux de `Sex`

```
ggplot(data=cats, aes(x=Bwt, y=Hwt, color=Sex, shape=Sex, size=Sex)) + geom_point()
```



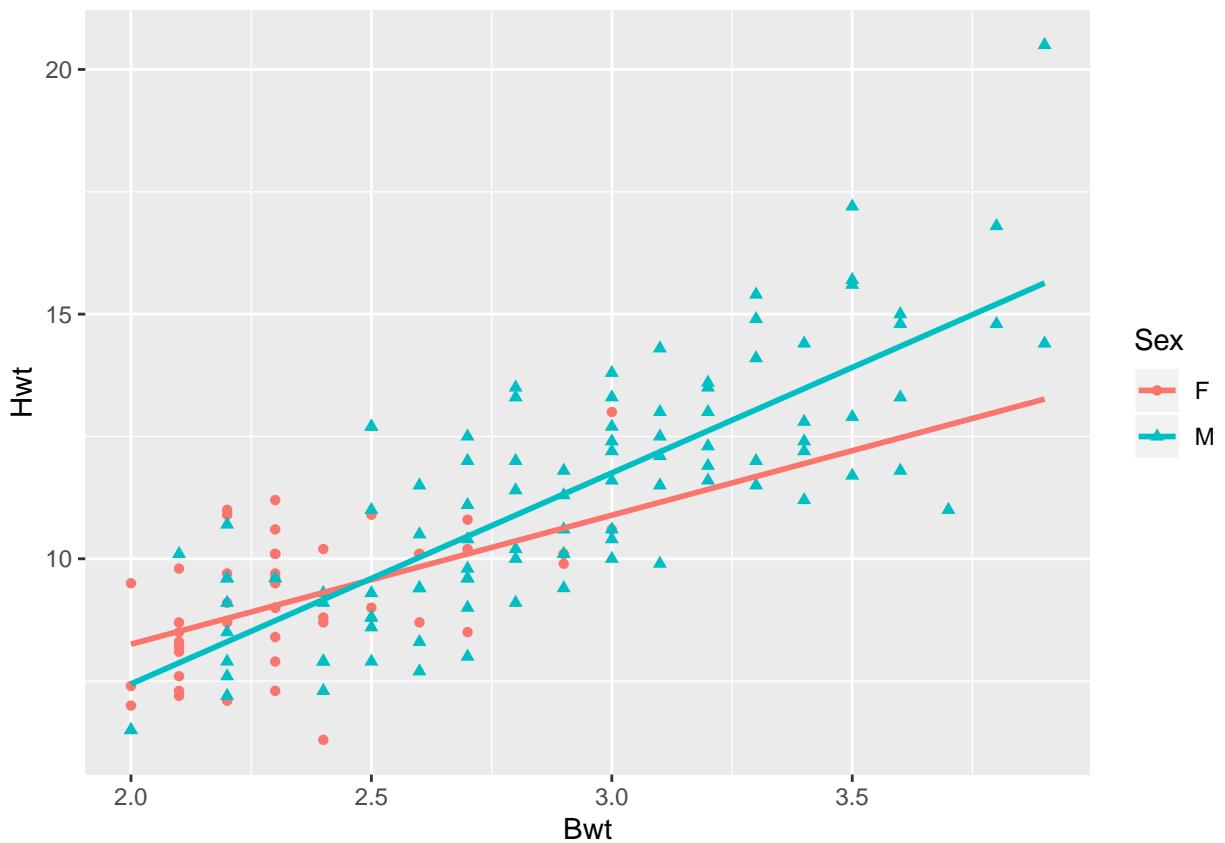
On peut y ajouter les différentes droites de régression dans les groupes formés par les modalités de Sex.

```
ggplot(data=cats, aes(x=Bwt, y=Hwt, color=Sex, shape=Sex)) +
  geom_point() + geom_smooth(method=lm, se=FALSE,)
```



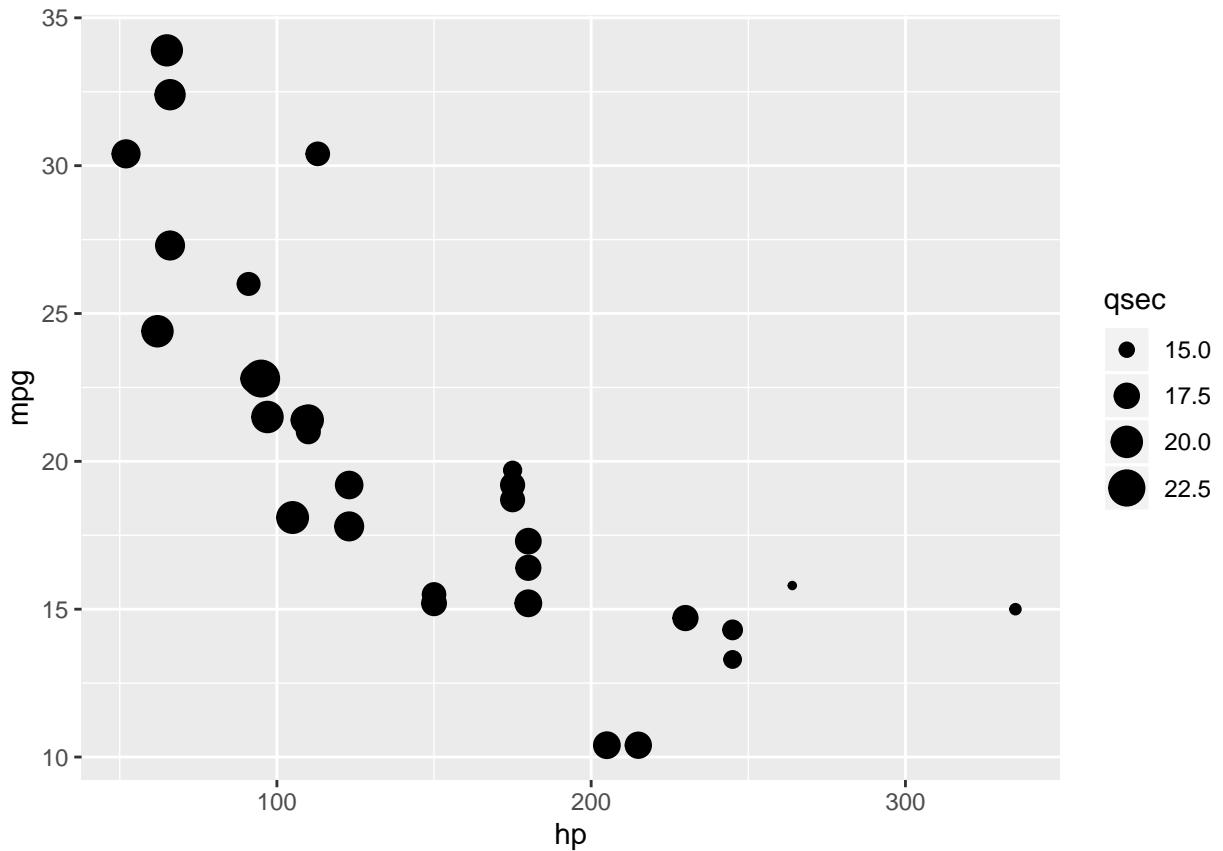
Il est dès lors facile d'étendre les droites de régression.

```
ggplot(data=cats, aes(x=Bwt, y=Hwt, color=Sex, shape=Sex)) +
  geom_point() + geom_smooth(method=lm, se=FALSE, fullrange=TRUE)
```



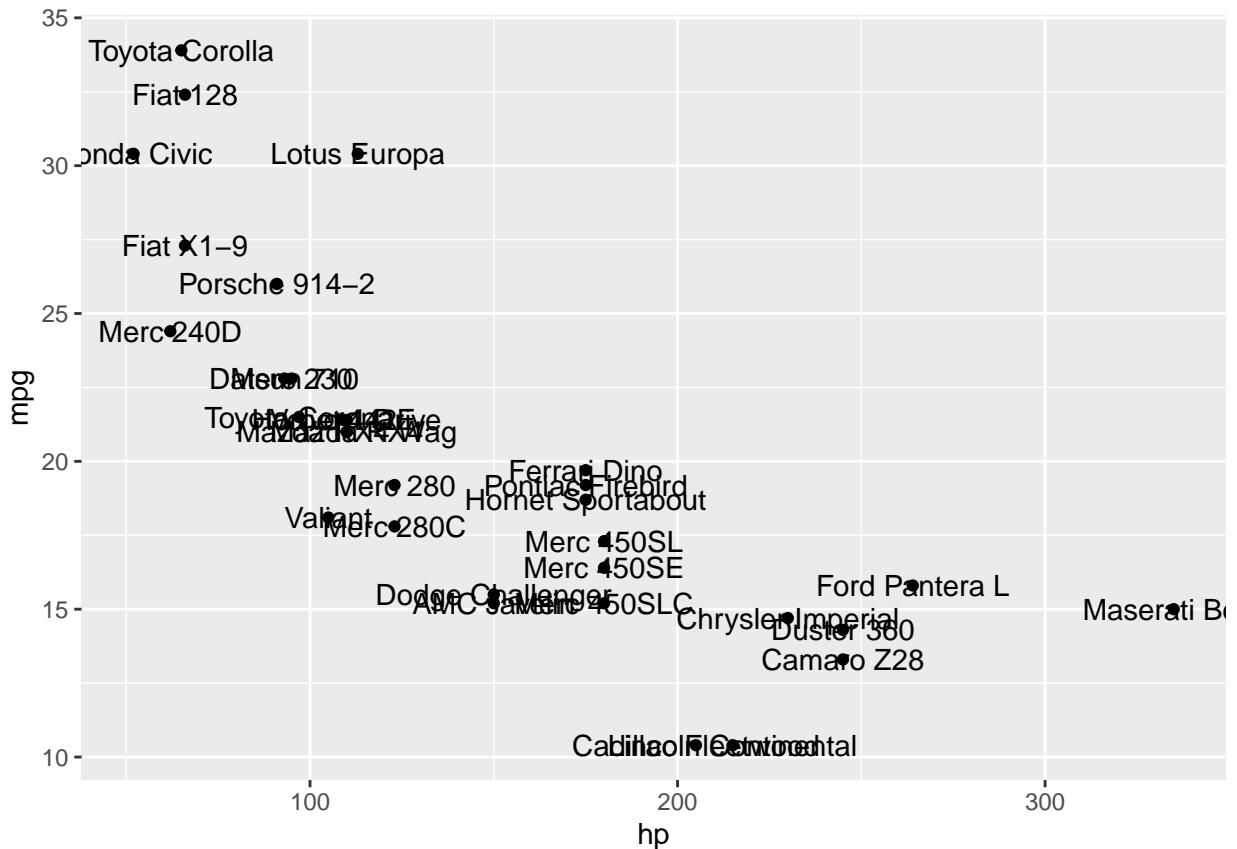
Revenons au jeu de données `mtcars` car il est plus riche en variables. On peut ainsi donner une taille aux points fonction d'une variable quantitative ici `qsec`.

```
nuage2=ggplot(data=mtcars, aes(x=hp,y=mpg))
nuage2+geom_point(aes(size=qsec))
```



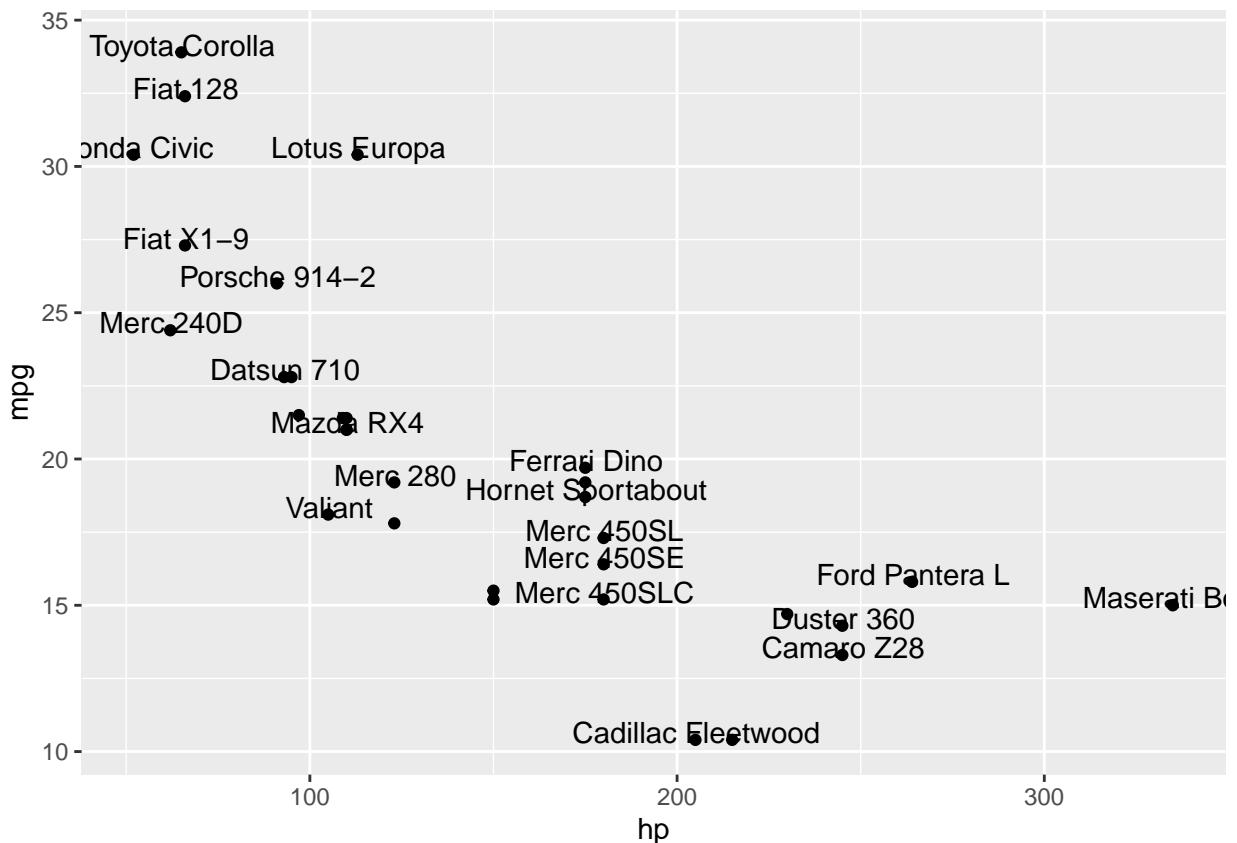
On peut rajouter à chaque point le nom qui lui est attribué dans le datafram avec `geom_text()`. Cela peut etre vite illisible.

```
nuage2+geom_point() +geom_text(label=rownames(mtcars))
```



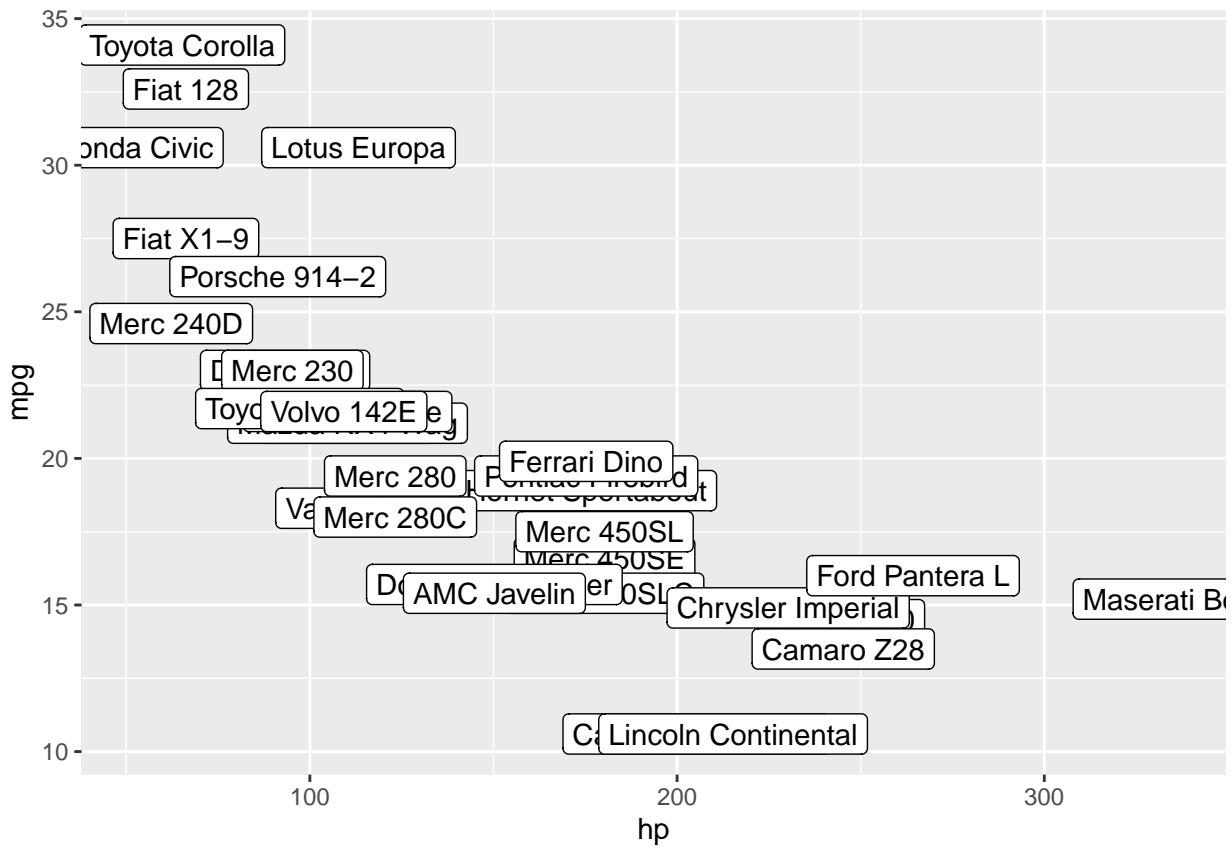
Avec `check_overlap` on peut faire en sorte que les noms ne se superposent pas. `nudge_x` et `nudge_y` servent à ajuster la position du texte par rapport à celle des points.

```
nuage2+geom_point() +geom_text(label=rownames(mtcars), nudge_x = 0.25, nudge_y = 0.25, check_overlap = T)
```



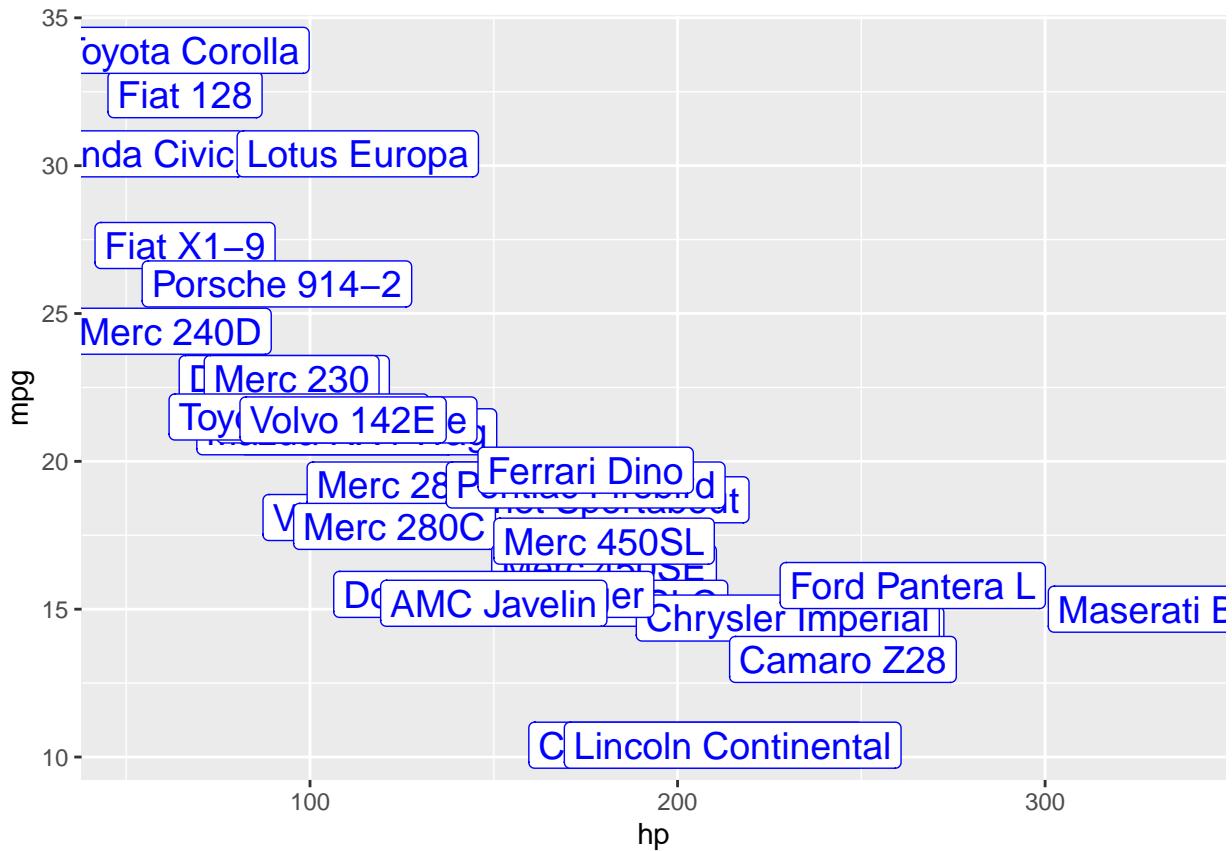
Il peut etre préférable d'utiliser `geom_label()` au lieu de `geom_text`.

```
nuage2+geom_point()+geom_label(label=rownames(mtcars), nudge_x = 0.25, nudge_y = 0.2)
```



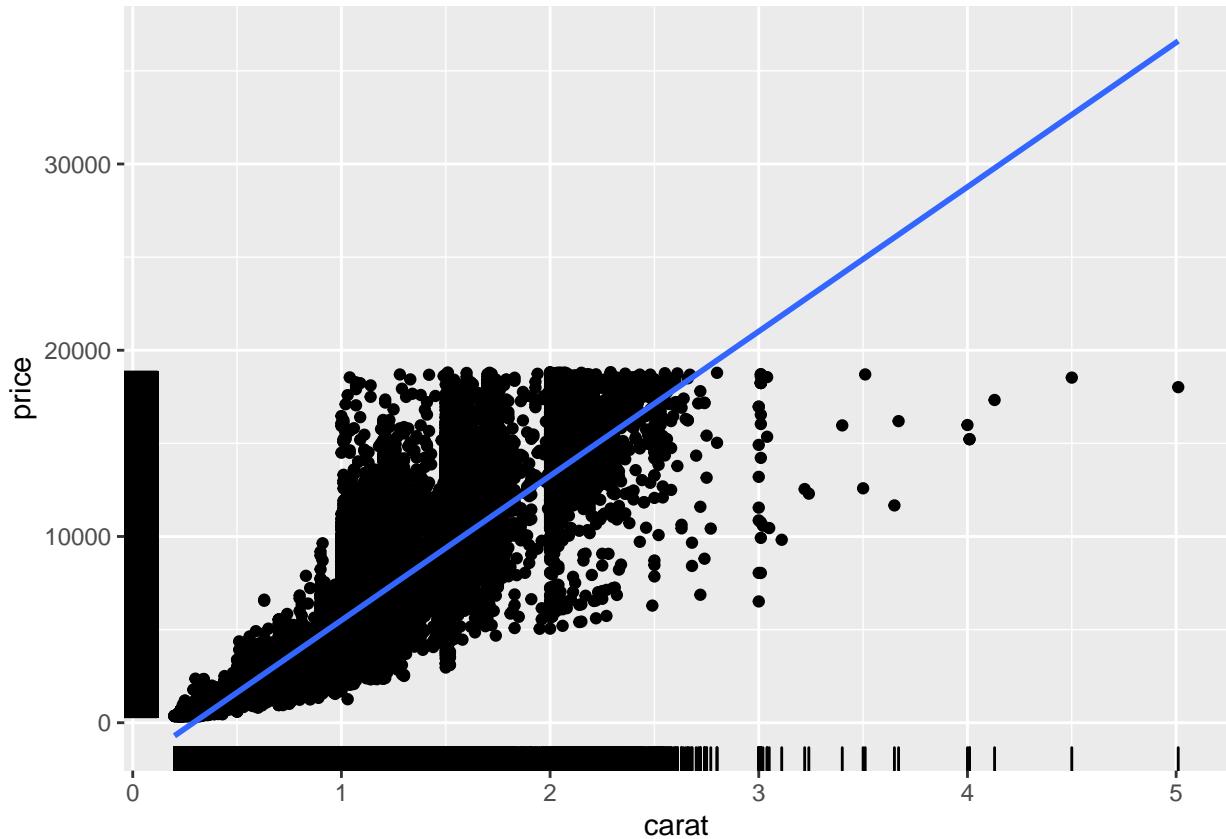
Avec un peu de couleur.

```
nuage2+geom_point() + geom_label(label=rownames(mtcars), color="blue", size=5)
```



Ci-dessous, nous dessinons le nuage de points (via `geom_point()`) entre les variables `carat` et `price`. Nous ajoutons ensuite des indicateurs de valeurs sur chaque axe (avec `geom_rug()`), et la droite de régression (via `geom_smooth()` avec l'option `method = "lm"`).

```
ggplot(diamonds, aes(x = carat, y = price)) +geom_point() +geom_rug() +geom_smooth(method = "lm", se = T)
```



### Instructions supplémentaires

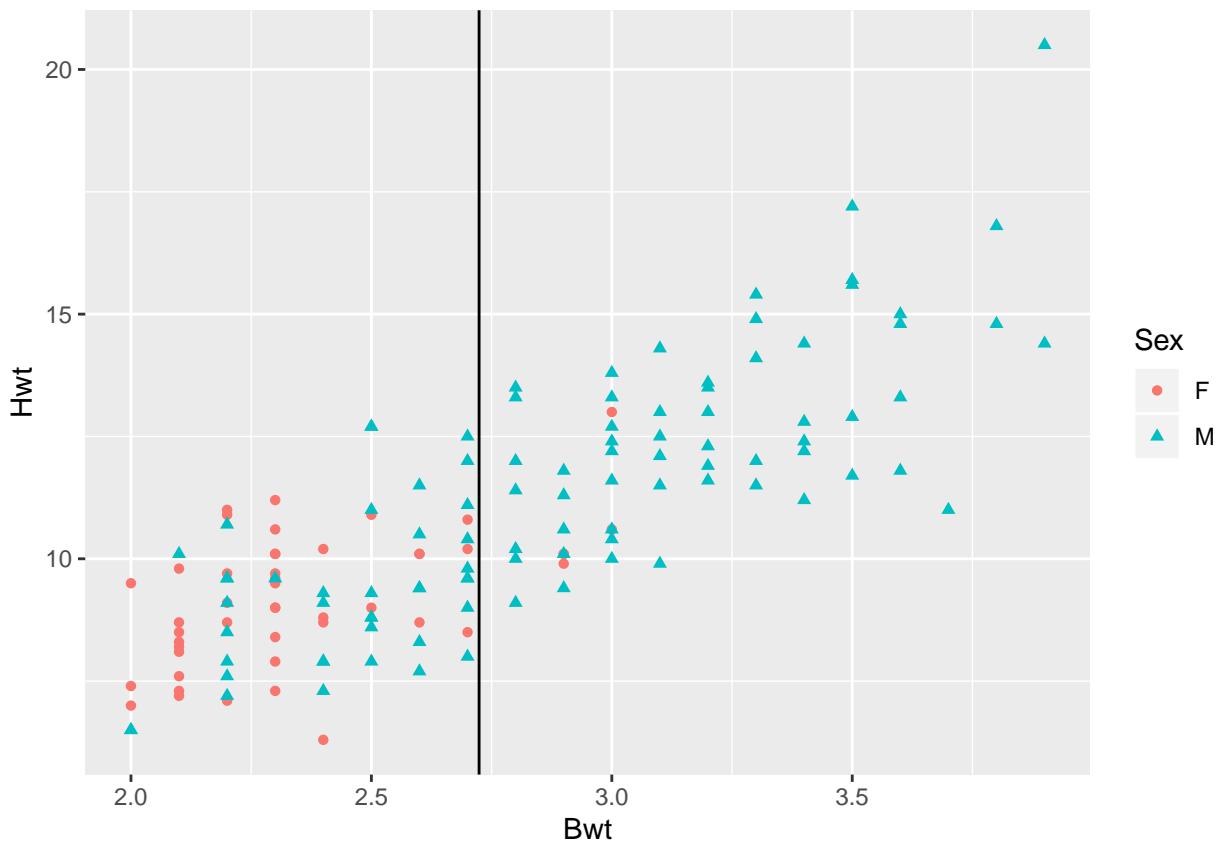
Il est possible d'ajouter des lignes sur un graphique :

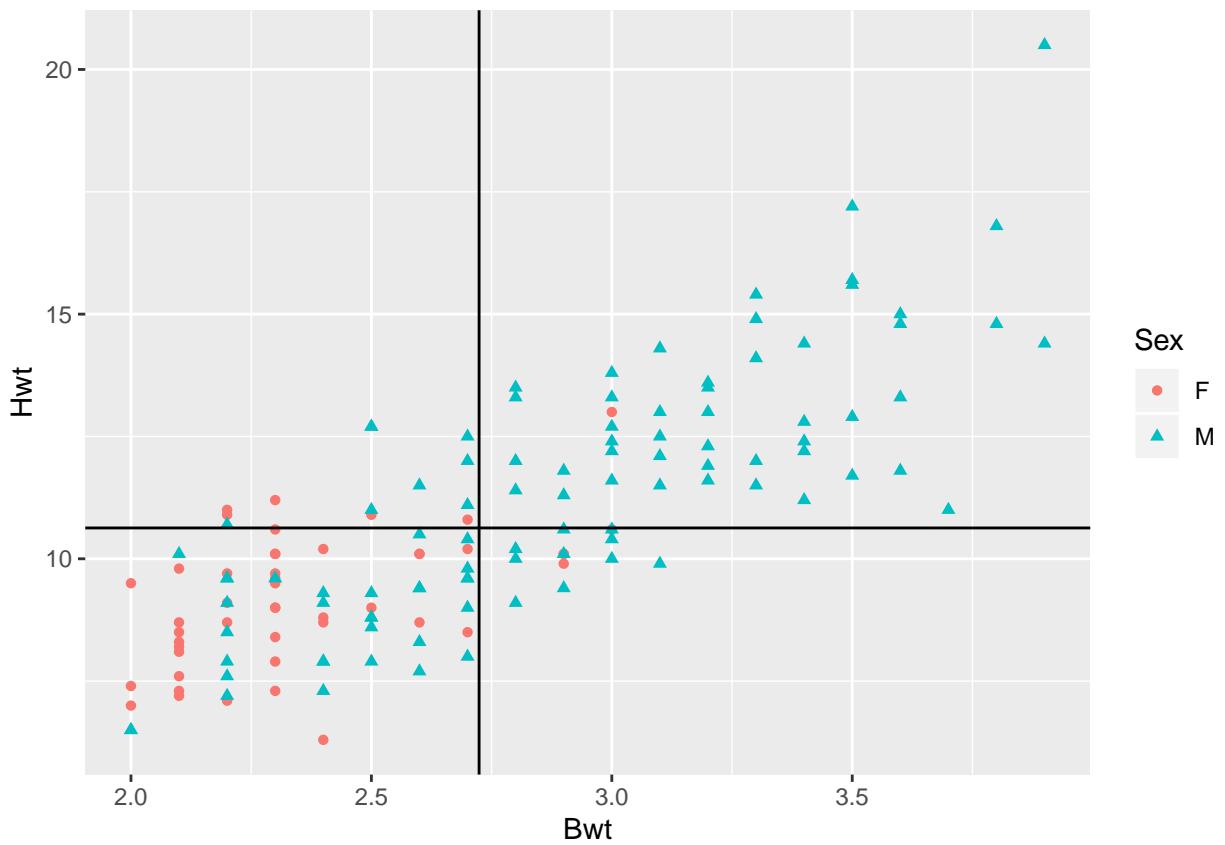
- `geom_vline()` : ligne verticale
- `geom_hline()` : ligne horizontale
- `geom_abline()` : ligne spécifiée par sa pente et son ordonnée à l'origine avec `intercept` et `slope`
- `geom_segment()`: segment (ou flèche en utilisant `arrow`).

```
summary(cats)
```

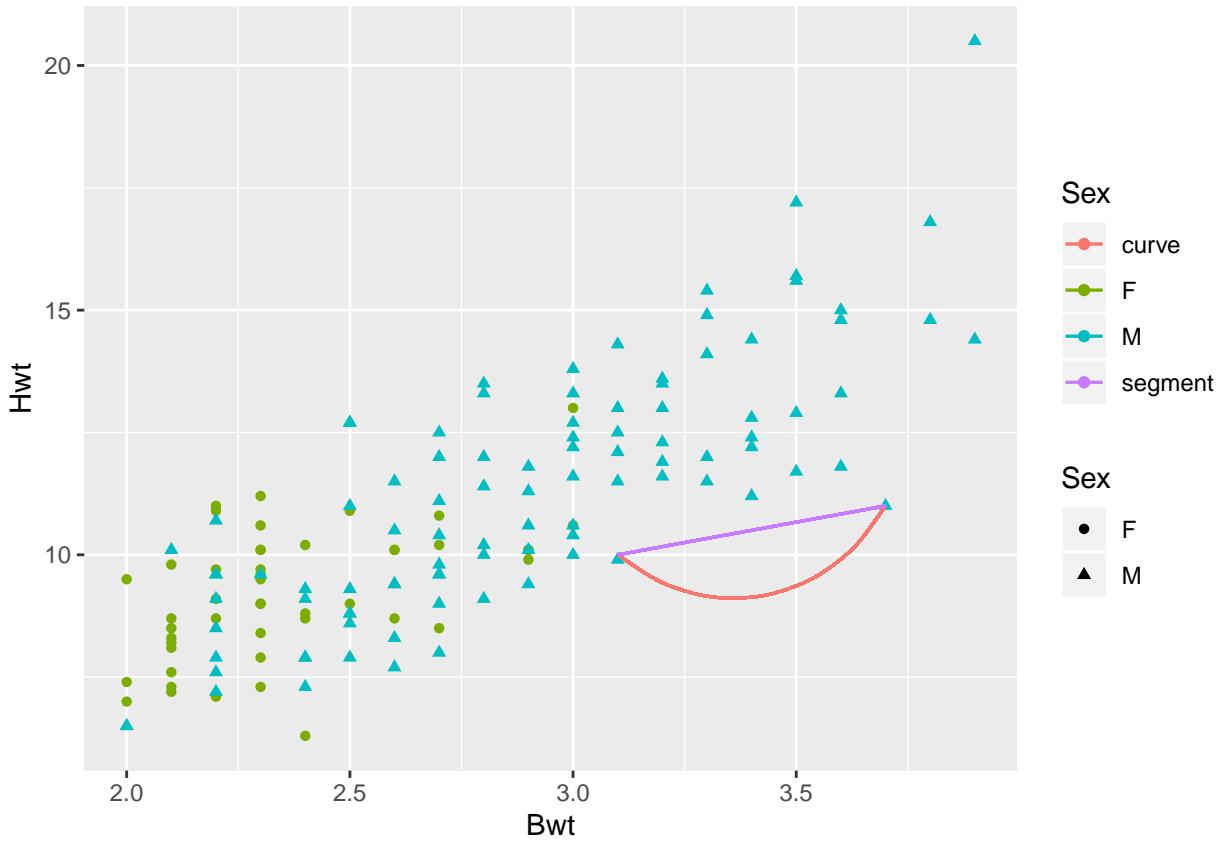
```
##   Sex          Bwt          Hwt
## F:47  Min.   :2.000  Min.   : 6.30
## M:97  1st Qu.:2.300  1st Qu.: 8.95
##           Median :2.700  Median :10.10
##           Mean   :2.724  Mean   :10.63
##           3rd Qu.:3.025  3rd Qu.:12.12
##           Max.   :3.900  Max.   :20.50
```

```
ggplot(data=cats, aes(x=Bwt, y=Hwt, color=Sex, shape=Sex)) +
  geom_point() + geom_vline(xintercept=2.724)
```





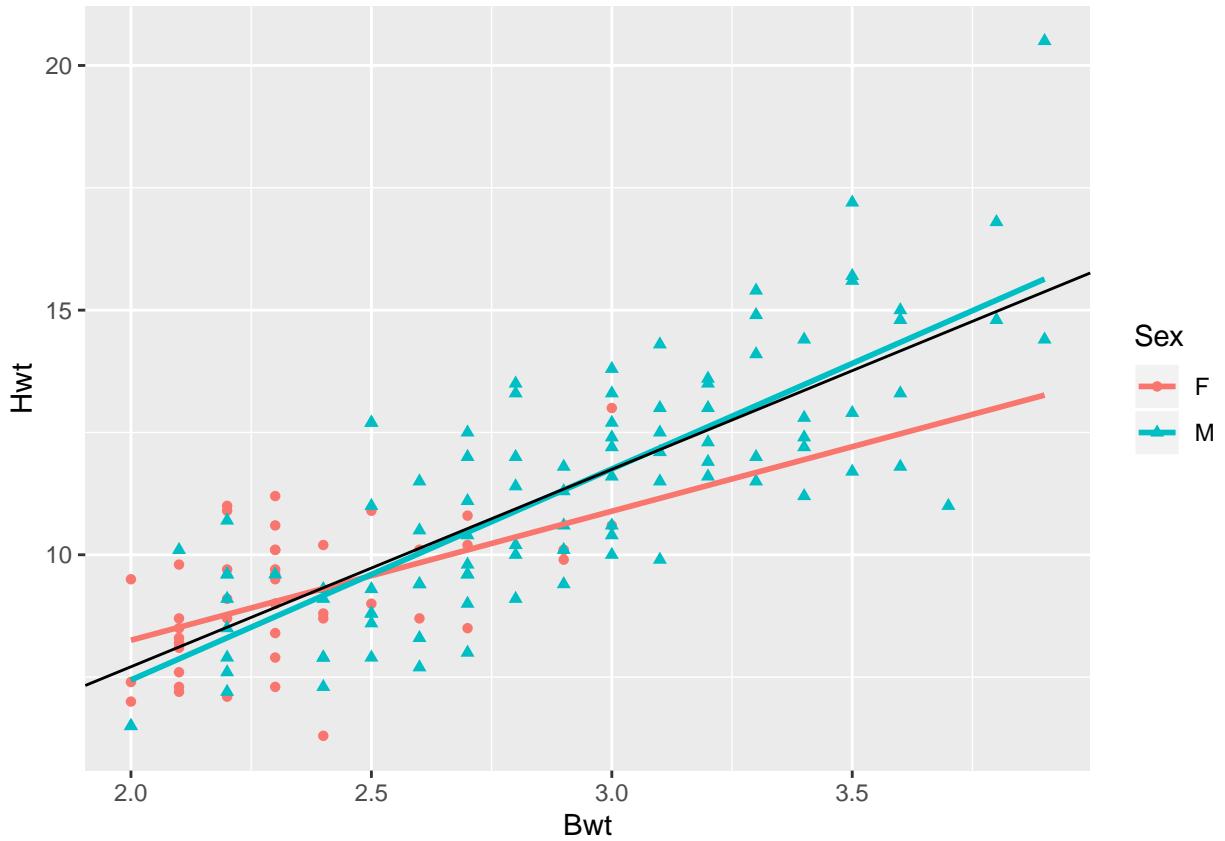
```
ggplot(data=cats, aes(x=Bwt, y=Hwt, color=Sex, shape=Sex)) +
  geom_point() +
  geom_curve(aes(x = 3.1, y = 10, xend = 3.7, yend = 11, colour = "curve")) +
  geom_segment(aes(x = 3.1, y = 10, xend = 3.7, yend = 11, colour = "segment"))
```



```
summary(lm(cats$Hwt ~ cats$Bwt))

##
## Call:
## lm(formula = cats$Hwt ~ cats$Bwt)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.5694 -0.9634 -0.0921  1.0426  5.1238
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.3567     0.6923  -0.515   0.607
## cats$Bwt     4.0341     0.2503 16.119 <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.452 on 142 degrees of freedom
## Multiple R-squared:  0.6466, Adjusted R-squared:  0.6441
## F-statistic: 259.8 on 1 and 142 DF,  p-value: < 2.2e-16

ggplot(data=cats, aes(x=Bwt, y=Hwt, color=Sex, shape=Sex)) +
  geom_point() +
  geom_smooth(method=lm, se=FALSE, fullrange=TRUE) +
  geom_abline(intercept=-0.3567, slope=4.0341)
```



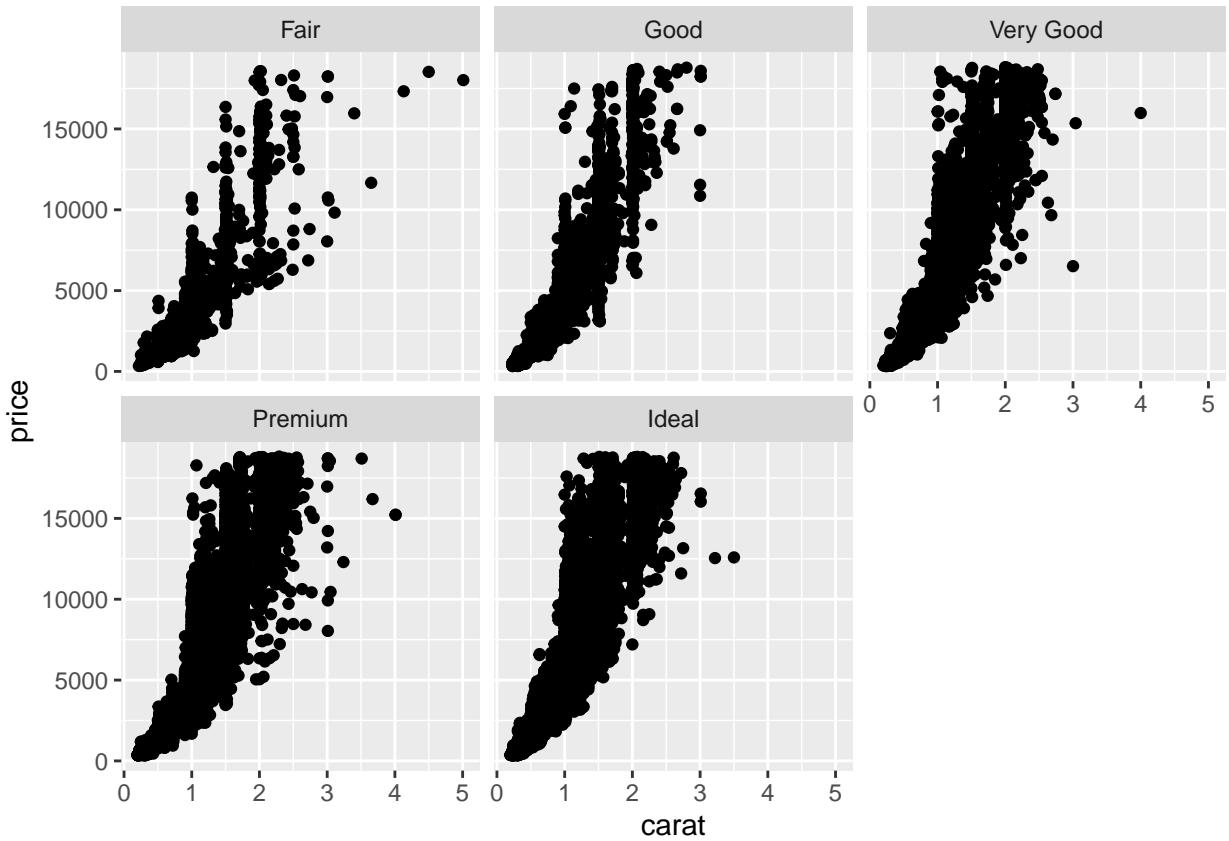
## Faceting

Pour comparer des données, il est très intéressant de faire des graphiques de type *small multiples*. L'idée est de faire un même graphique, mais pour chaque modalité d'une variable. Pour faire cela facilement avec `ggplot2`, nous disposons de deux fonctions : `facet_wrap()` et `facet_grid()`.

### avec `facet_wrap()`

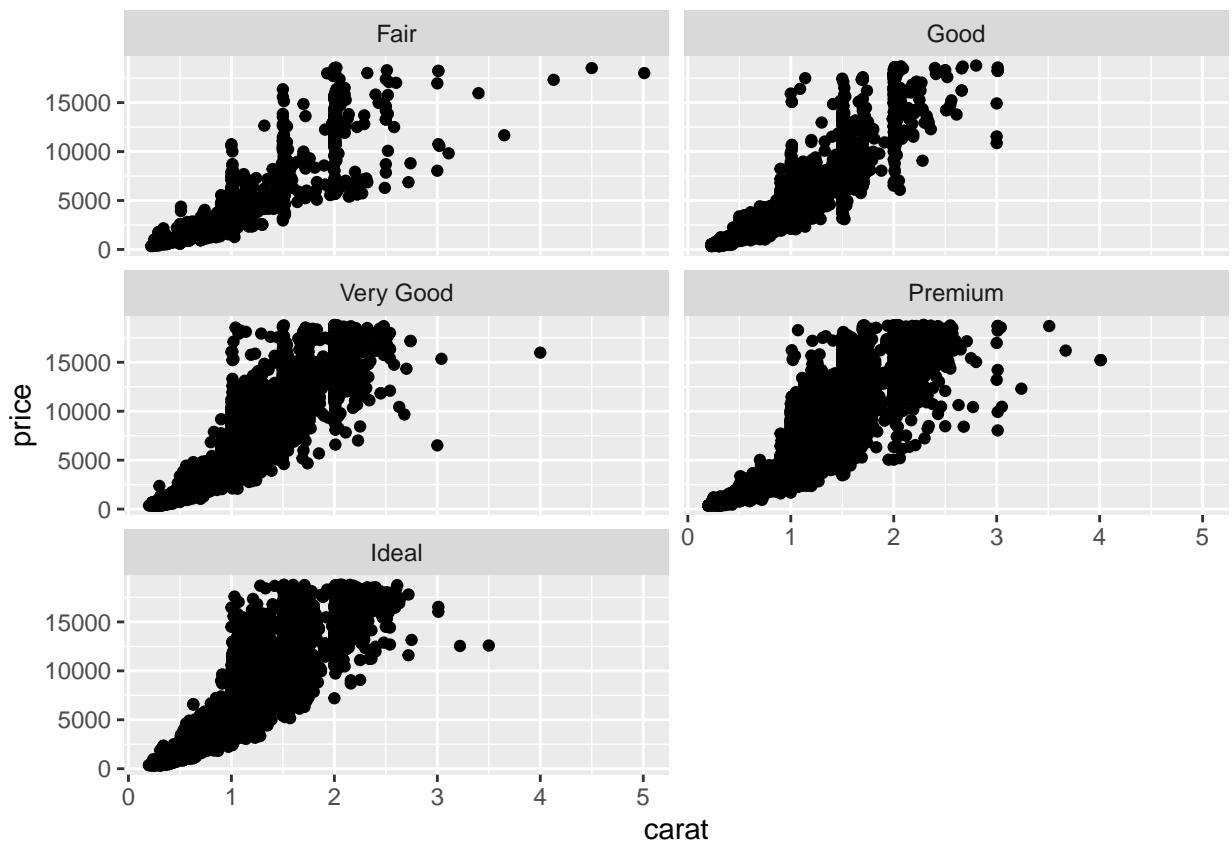
Cette première fonction permet de s'affranchir de la réflexion du nombre de modalités dans la variable en question. Le programme choisit lui-même ce qu'il lui semble approprié. Ci-dessous, nous représentons le nuage de points entre `carat` et `price`, mais pour chaque valeur de `cut`.

```
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  facet_wrap(facets = ~ cut)
```



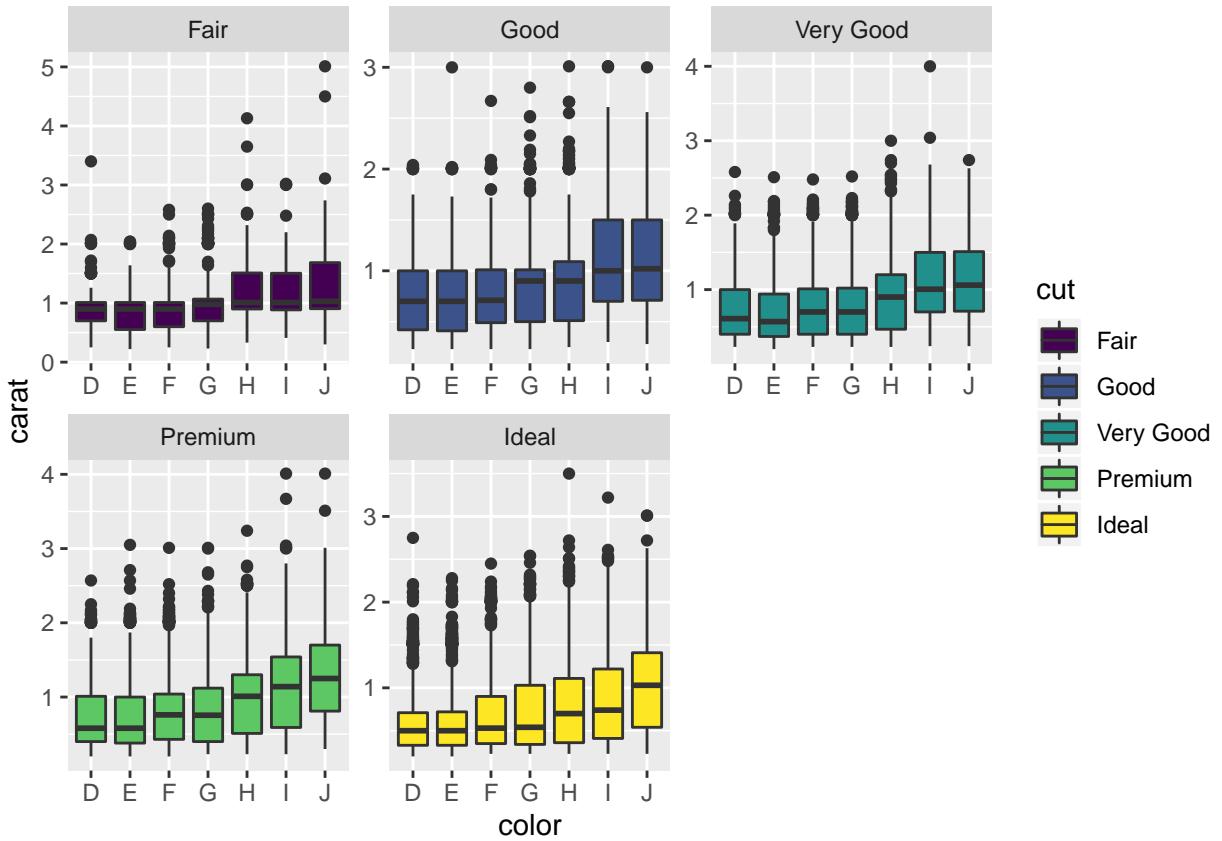
Si le choix ne convient pas, nous pouvons forcer le nombre de colonnes avec l'option `ncol` (ou le nombre de lignes avec `nrow`), comme ci-dessous.

```
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  facet_wrap(facets = ~cut, ncol = 2)
```



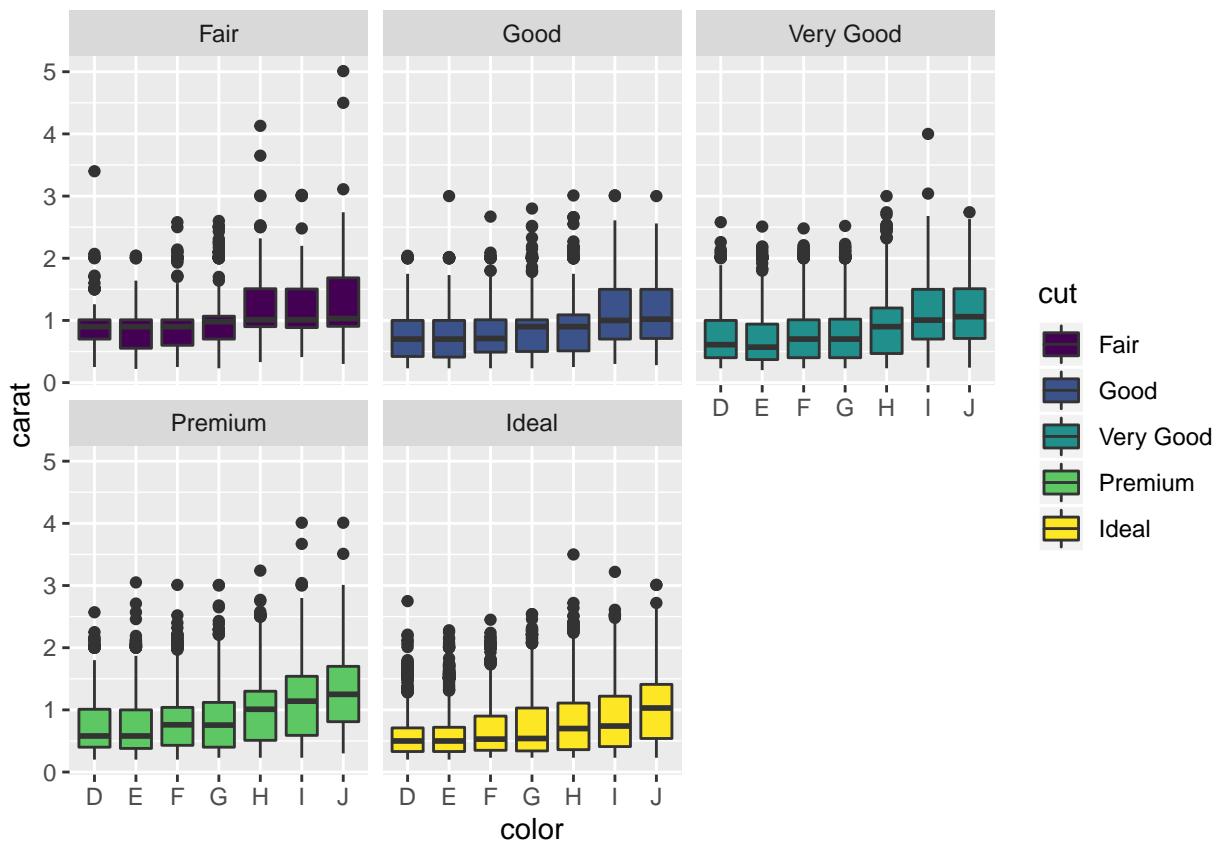
Pour des boxplots au lieu de nuages de points.

```
ggplot(diamonds, aes(x=color, y=carat, fill=cut)) + geom_boxplot() + facet_wrap(~cut, scale="free")
```



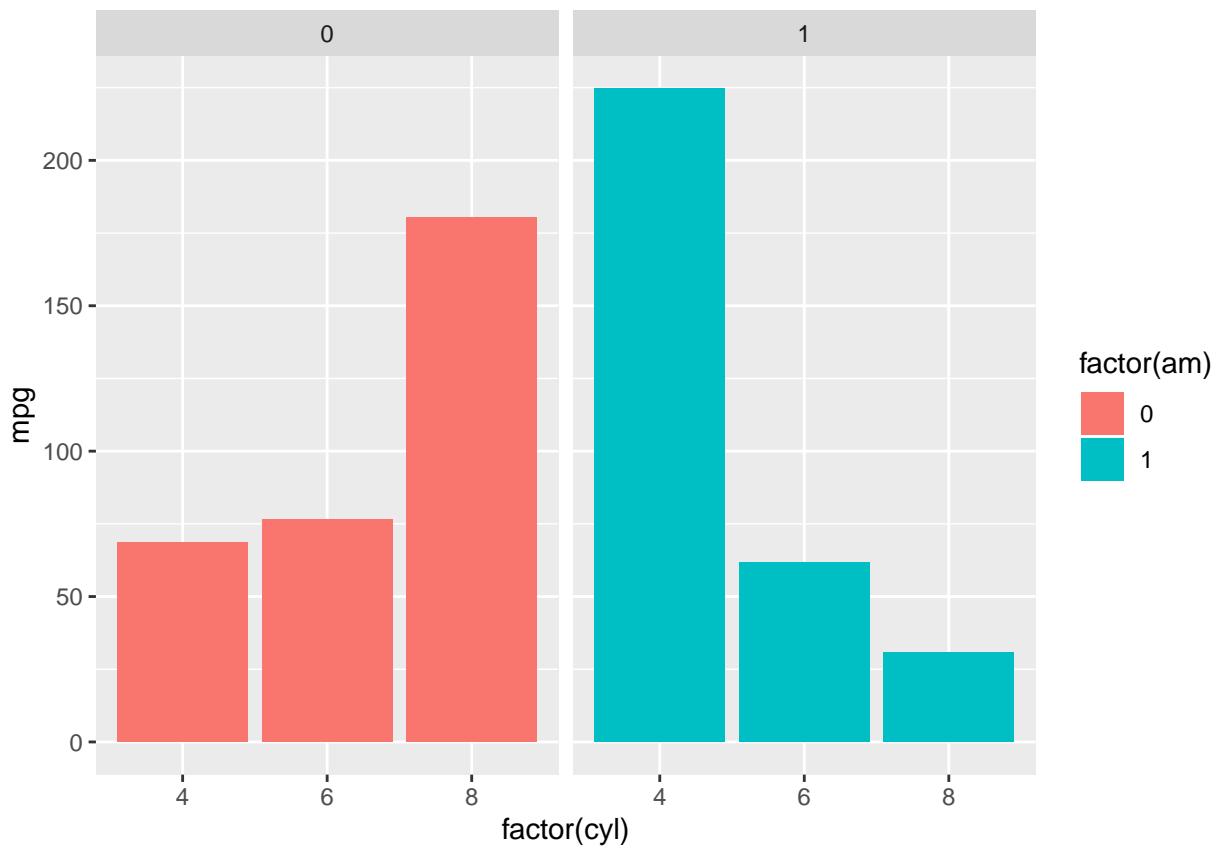
Attention à l'échelle des ordonnées !

```
ggplot(diamonds, aes(x=color, y=carat, fill=cut)) + geom_boxplot() + facet_wrap(~cut)
```



Pour des diagrammes en barres, cela permet de comparer la somme des valeurs de `mpg` selon les modalités du croisement entre les variables `am` et `cyl`.

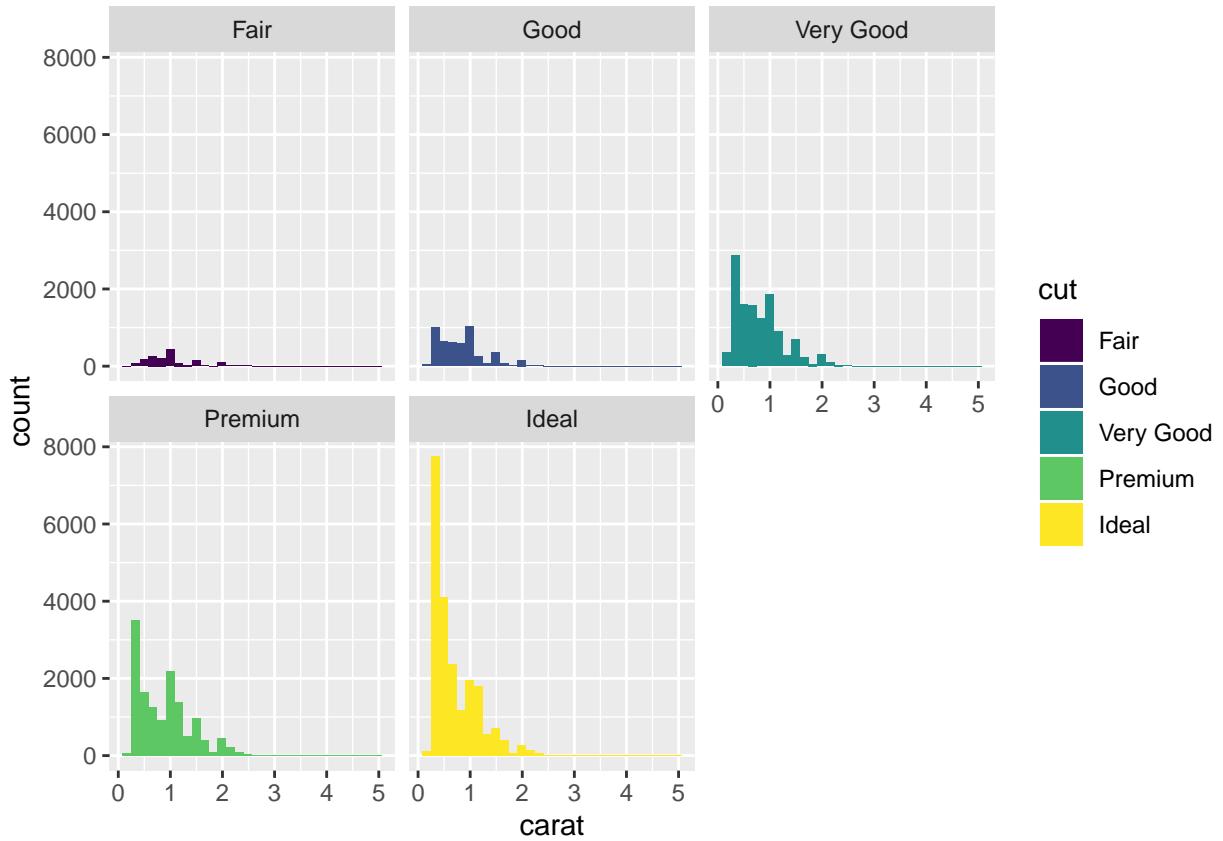
```
ggplot(data=mtcars, aes(fill=factor(am), y=mpg, x=factor(cyl))) +geom_bar(stat="identity")+facet_wrap(~
```



Cela correspond à la somme de mpg dans les croisements `tapply(mtcars$mpg, list(mtcars$cyl, mtcars$am), sum)`.

Pour des histogrammes.

```
ggplot(data=diamonds, aes(x=carat, fill=cut)) + geom_histogram() + facet_wrap(~cut)
```

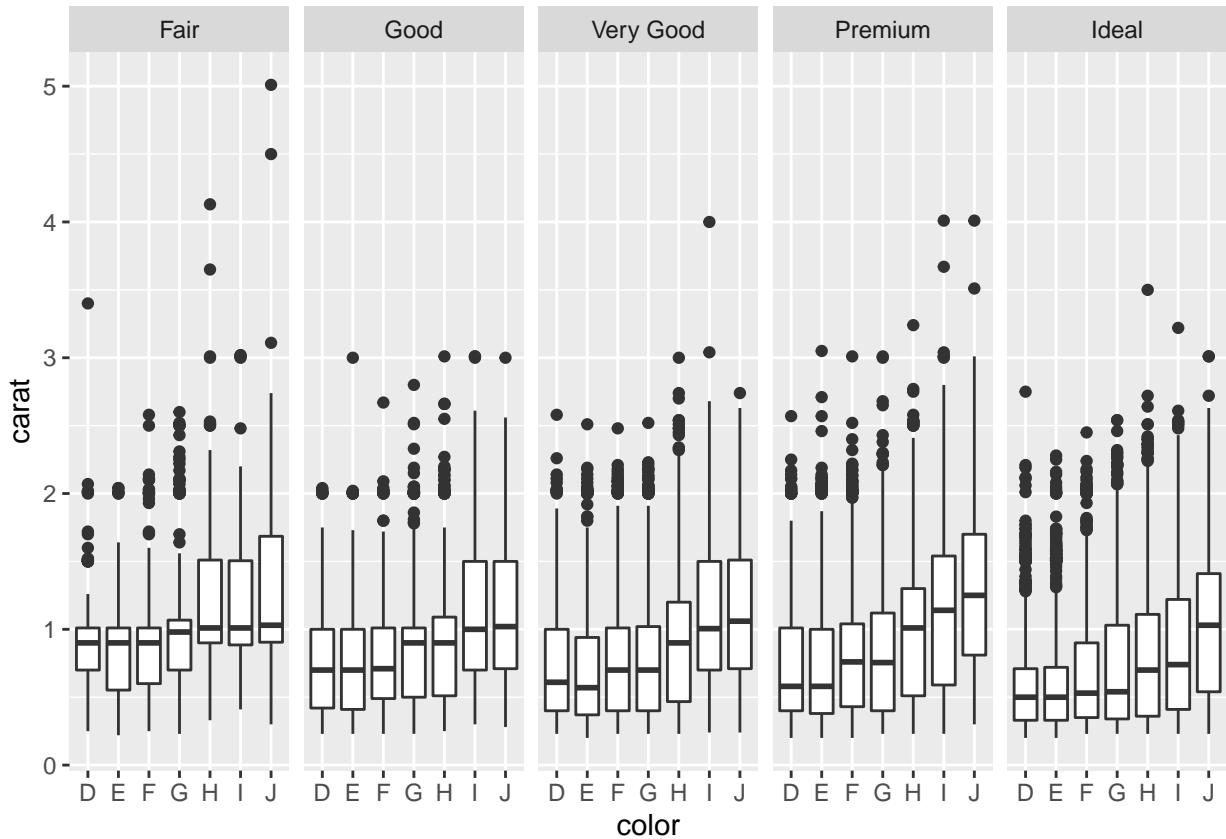


```
avec facet_grid()
```

Mais il est utile de forcer soit une répartition sur une même ligne de ces différents graphiques, soit sur une même colonne. Dans ce cas, nous utilisons la fonction `facet_grid()`, qui permet de déterminer explicitement comment on veut répartir les différentes facettes.

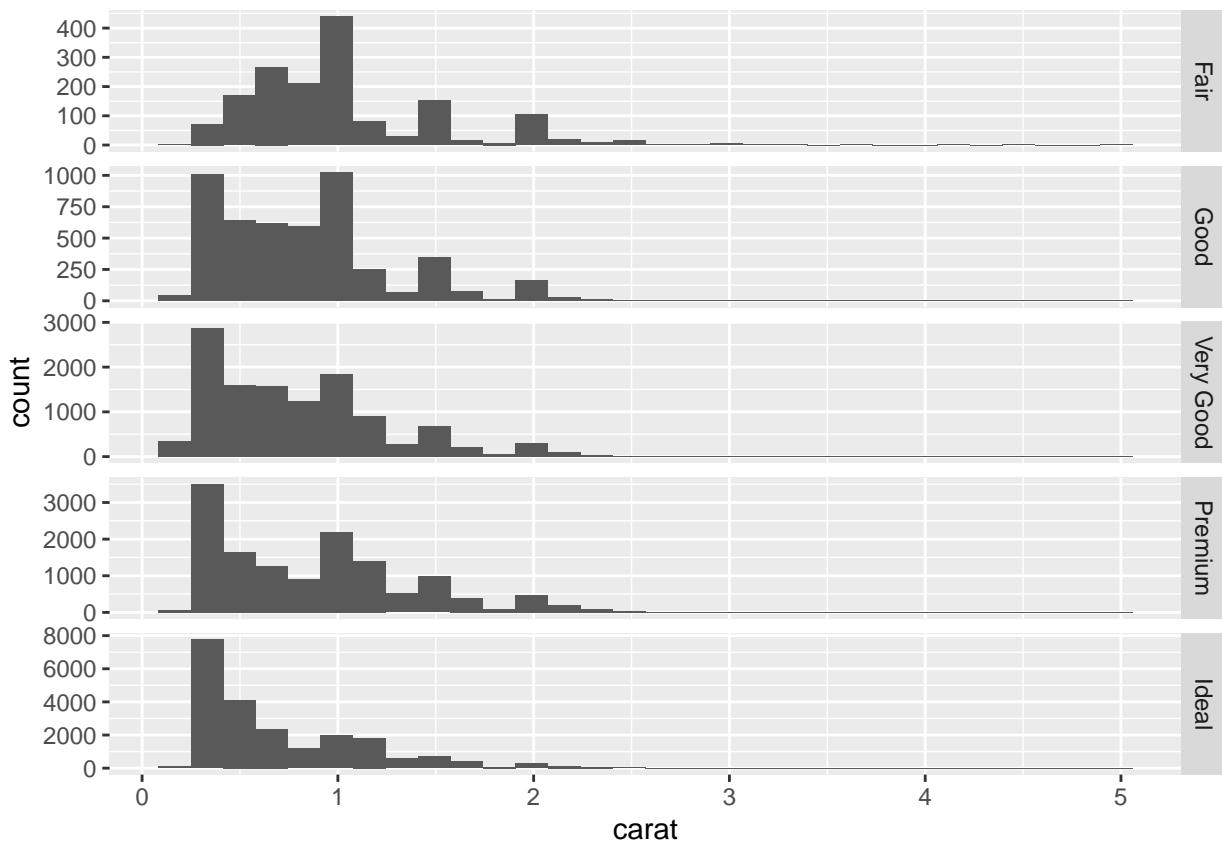
Ci-dessous, nous créons les boîtes à moustaches de `carat` pour chaque modalité de `color`, et ceci pour chaque modalité de `cut`. Tout est représenté sur une même ligne.

```
ggplot(diamonds, aes(color, carat)) +
  geom_boxplot() +
  facet_grid(facets = ~ cut)
```



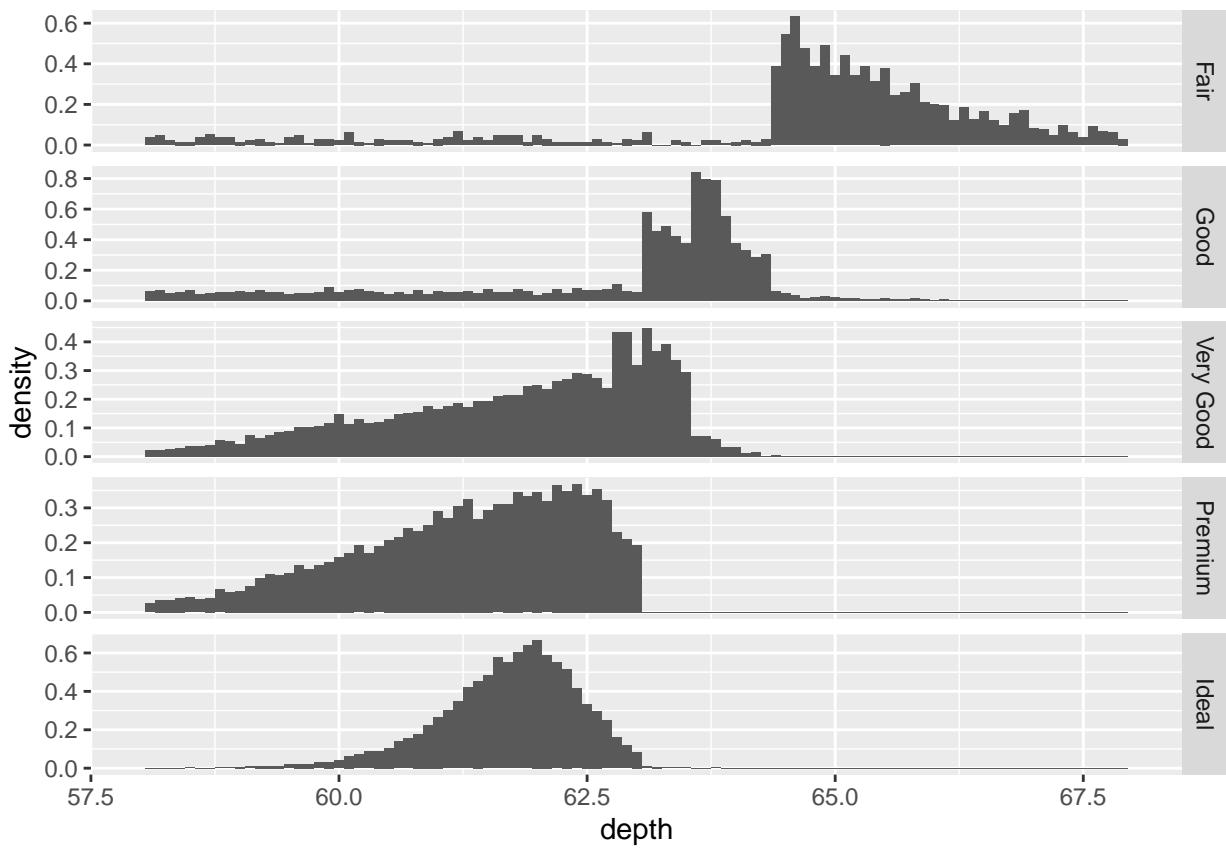
Pour forcer la répartition en colonne, nous inversons la formule. Ici, nous représentons l'histogramme de `carat`, pour chaque valeur de `cut` (cela permet donc de bien les comparer). Comme les effectifs ne sont pas les mêmes entre les modalités, nous décidons de laisser libre l'axe des  $y$  pour mieux visualiser les répartitions, grâce à l'option `scales`.

```
ggplot(diamonds, aes(x = carat)) +
  geom_histogram() +
  facet_grid(facets = cut ~ ., scales = "free_y")
```



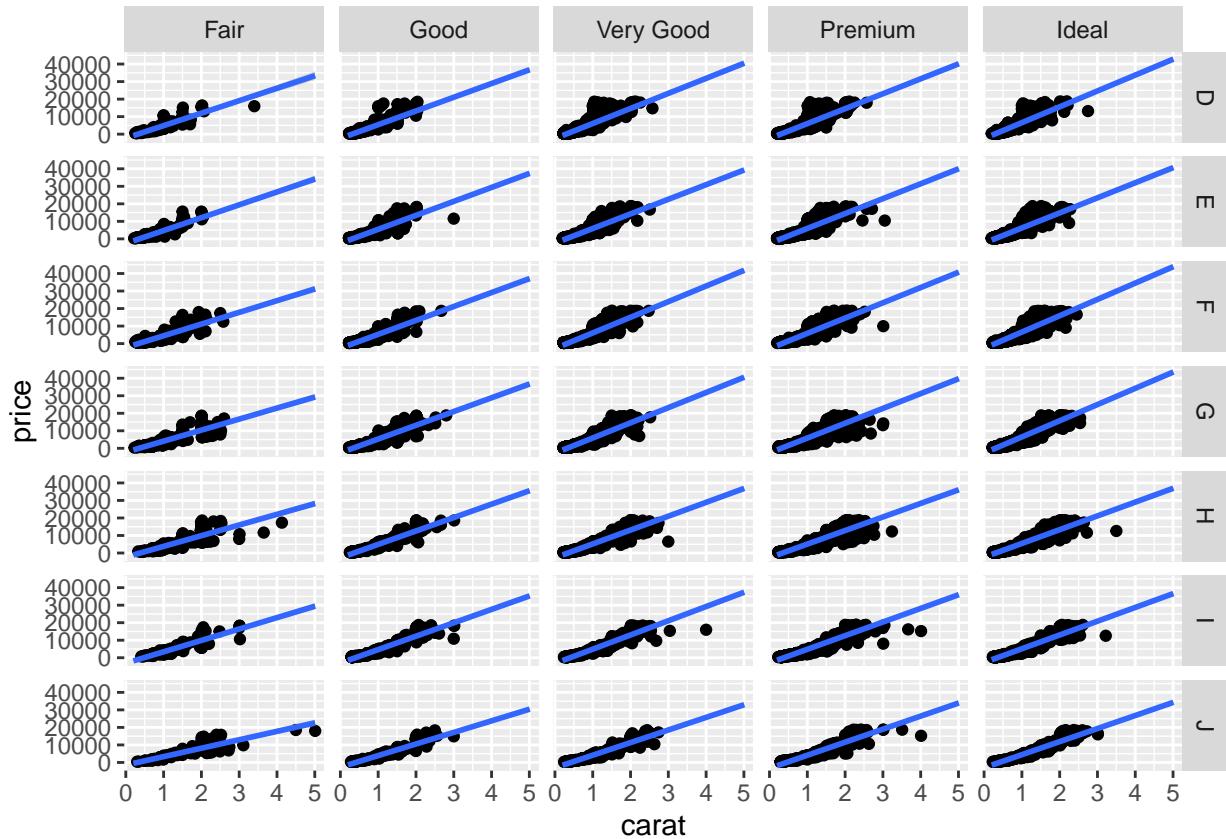
De même que précédemment on peut représenter les histogrammes (cette fois en densité) de la variable `depth` en fonction de `cut` à l'aide `facet_grid()`.

```
ggplot(diamonds, aes(x = depth)) +
  geom_histogram(aes(y=..density..), binwidth=0.1) +
  facet_grid(facets = cut ~ ., scales = "free_y") + xlim(58,68)
```



Bien évidemment, il est possible de combiner les deux possibilités. Par exemple, ici, nous créons le nuage de point entre `carat` et `price`, pour chaque couple de modalités des variables `color` et `cut`.

```
ggplot(diamonds, aes(carat, price)) +
  geom_point() +
  geom_smooth(method = "lm", fullrange = T) +
  facet_grid(facets = color ~ cut)
```



### Graphiques de corrélation

Pour aller plus loin on peut utiliser le package `GGally` qui est une extension de `ggplot2`. Commençons par extraire les données quantitatives du jeu de données `mtcars`:

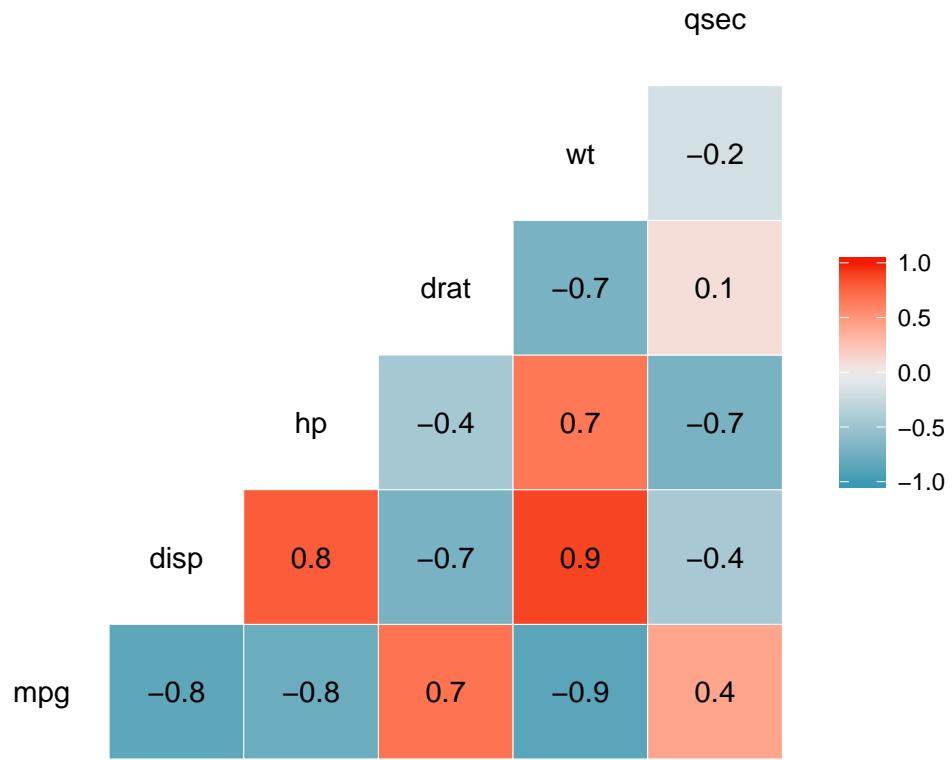
```
mtcarsquant <- mtcars[, c(1,3,4,5,6,7)]
```

```
head (mtcarsquant)
```

```
##          mpg disp  hp drat    wt  qsec
## Mazda RX4     21.0 160 110 3.90 2.620 16.46
## Mazda RX4 Wag 21.0 160 110 3.90 2.875 17.02
## Datsun 710    22.8 108  93 3.85 2.320 18.61
## Hornet 4 Drive 21.4 258 110 3.08 3.215 19.44
## Hornet Sportabout 18.7 360 175 3.15 3.440 17.02
## Valiant       18.1 225 105 2.76 3.460 20.22
```

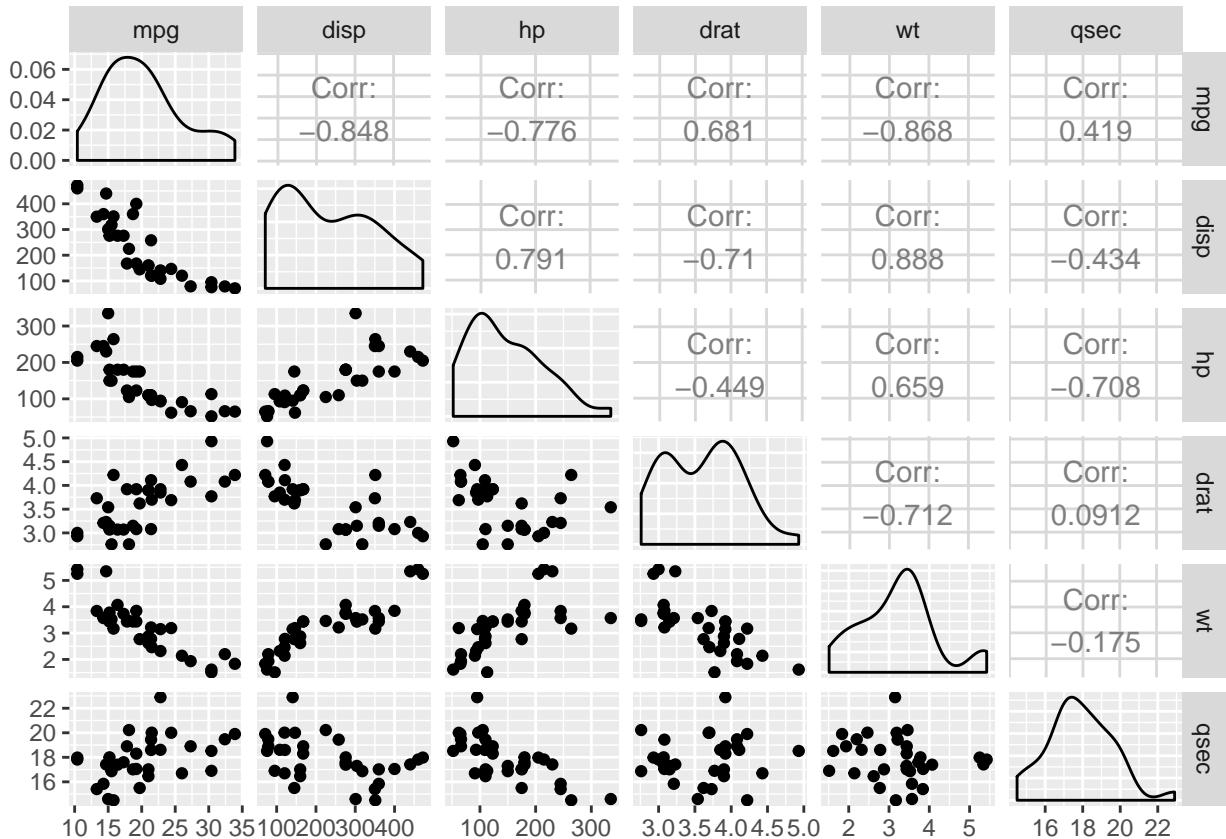
A partir de ce nouveau jeu de données on peut obtenir le graphique ci-dessous qui met en avant les corrélations 2 à 2 à l'aide d'une échelle de couleurs.

```
ggcorr(mtcarsquant, palette = "RdBu", label = TRUE)
```



On peut enrichir ces résultats à l'aide du graphique ci-après qui nous renseigne en plus sur la densité des variables et les nuages de points correspondant au croisement des variables 2 à 2 et donc aux coefficients de corrélation.

```
ggpairs(mtcarsquant)
```



### Effet du placement de data et aes()

Dans le graphique ci-dessous, nous ne redéfinissons jamais `x` ou `y`. Ceux-ci ayant été définis dans la fonction `ggplot()`, les fonctions ajoutées reprennent leur définition. Mais il est possible de les définir dans chaque fonction.

Pour détailler ce comportement, voici trois commandes permettant de faire strictement le même graphique (le premier produit dans le paragraphe ci-dessous).

```
ggplot(diamonds, aes(x = price)) + geom_histogram()
ggplot(diamonds) + geom_histogram(mapping = aes(x = price))
ggplot() + geom_histogram(data = diamonds, mapping = aes(x = price))
```

Voici ce qui diffère entre ces trois versions :

- Dans la première, les données seront `diamonds` pour l'ensemble des commandes ajoutées, et `x` sera la variable `price` (sauf spécification ultérieure) ;
- Dans la seconde, on utilisera toujours `diamonds` comme données, mais `x` n'est défini que pour l'histogramme. On devra définir `x` pour les fonctions ultérieures si besoin ;
- Dans la dernière, il n'y a aucune spécification de base, et chaque fonction devra déterminer quelles données prendre, ainsi que les aspects esthétiques à utiliser dans celles-ci.

Ce mécanisme est particulièrement intéressant lorsque nous souhaitons utiliser plusieurs jeux de données ensemble. Dans le graphique ci-dessous, nous allons afficher trois informations différentes :

- les différentes valeurs de `price` en fonction de la variable `cut`, avec un aléa sur l'axe des `x` (avec la fonction `geom_jitter()`) ;
- les boîtes à moustaches ;
- un indicateur de moyenne (avec la fonction `geom_point()`) et d'écart-type (avec `geom_errorbar()`).

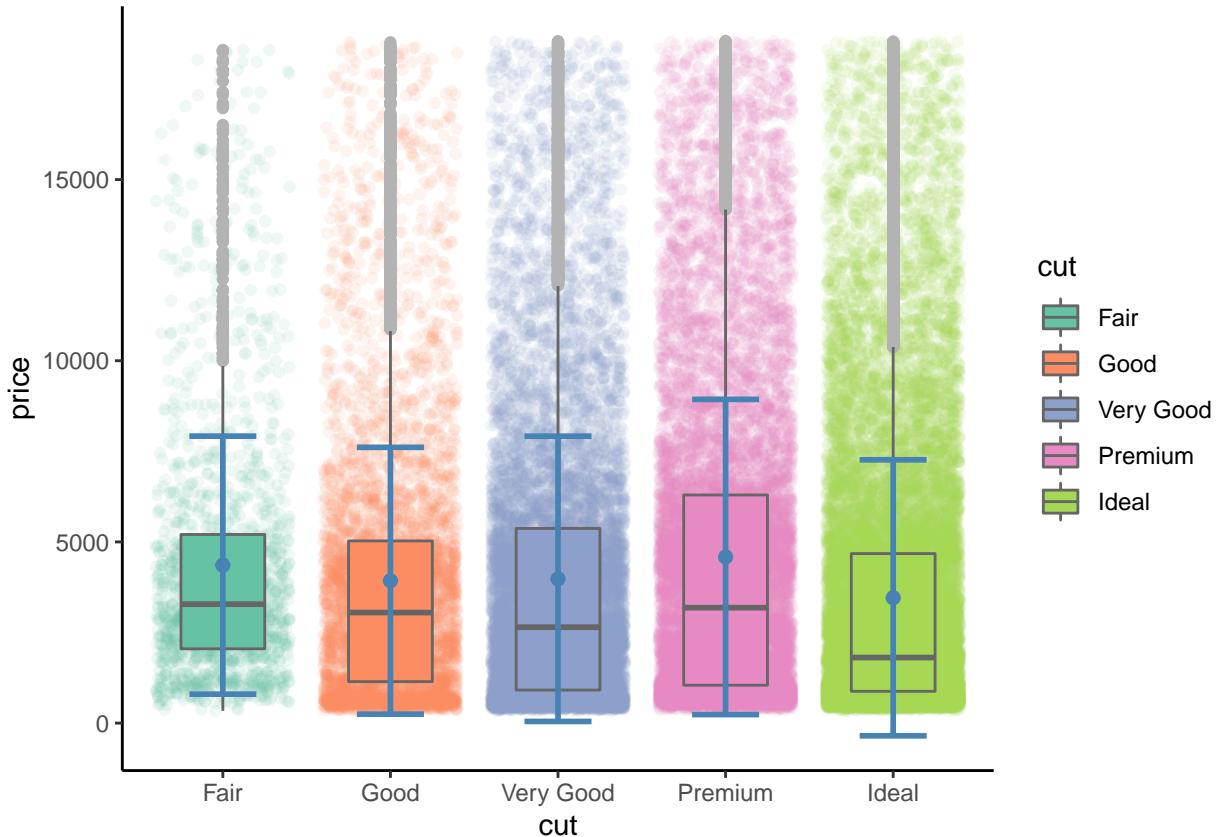
Pour cela, nous calculons en premier lieu la moyenne et l'écart-type de `price` pour chaque modalité de `cut`, stocké dans `df`.

```
df = diamonds %>%
  group_by(cut) %>%
  summarise(
    mean = mean(price, na.rm = T),
    sd = sd(price, na.rm = T)
  )
```

cut	mean	sd
Fair	4358.758	3560.387
Good	3928.864	3681.590
Very Good	3981.760	3935.862
Premium	4584.258	4349.205
Ideal	3457.542	3808.401

Nous avons donc le graphique ci-dessous, produit par un code très complet.

```
ggplot(data = diamonds, aes(x = cut, y = price, color = cut)) +
  scale_color_brewer(palette = "Set2") +
  scale_fill_brewer(palette = "Set2") +
  geom_jitter(alpha = .1) +
  geom_boxplot(mapping = aes(fill = cut),
               color = "gray40", width = .5,
               outlier.color = "gray70") +
  geom_errorbar(data = df,
                mapping = aes(y = mean,
                               ymin = mean - sd,
                               ymax = mean + sd),
                col = "steelblue", width = .4, size = 1) +
  geom_point(data = df,
             mapping = aes(y = mean),
             col = "steelblue", size = 2) +
  theme_classic()
```



### Enregistrer un graphique

Afin d'enregistrer un graphique réalisé avec `ggplot()` on utilise `ggsave()`.

- `filename` : nom du fichier, ou chemin et nom du fichier
- `plot` : graphique à sauvegarder (par défaut, le dernier)
- `scale` : facteur d'échelle
- `width` : largeur
- `height` : hauteur
- `dpi` : nombre de points par pouce, uniquement pour les images matricielles

`ggsave()` reconnaît automatiquement les extensions suivantes :

- `eps/ps`
- `pdf`
- `jpeg`
- `tiff`
- `png`
- `bmp`
- `svg`

### Exemple version `ggplot`

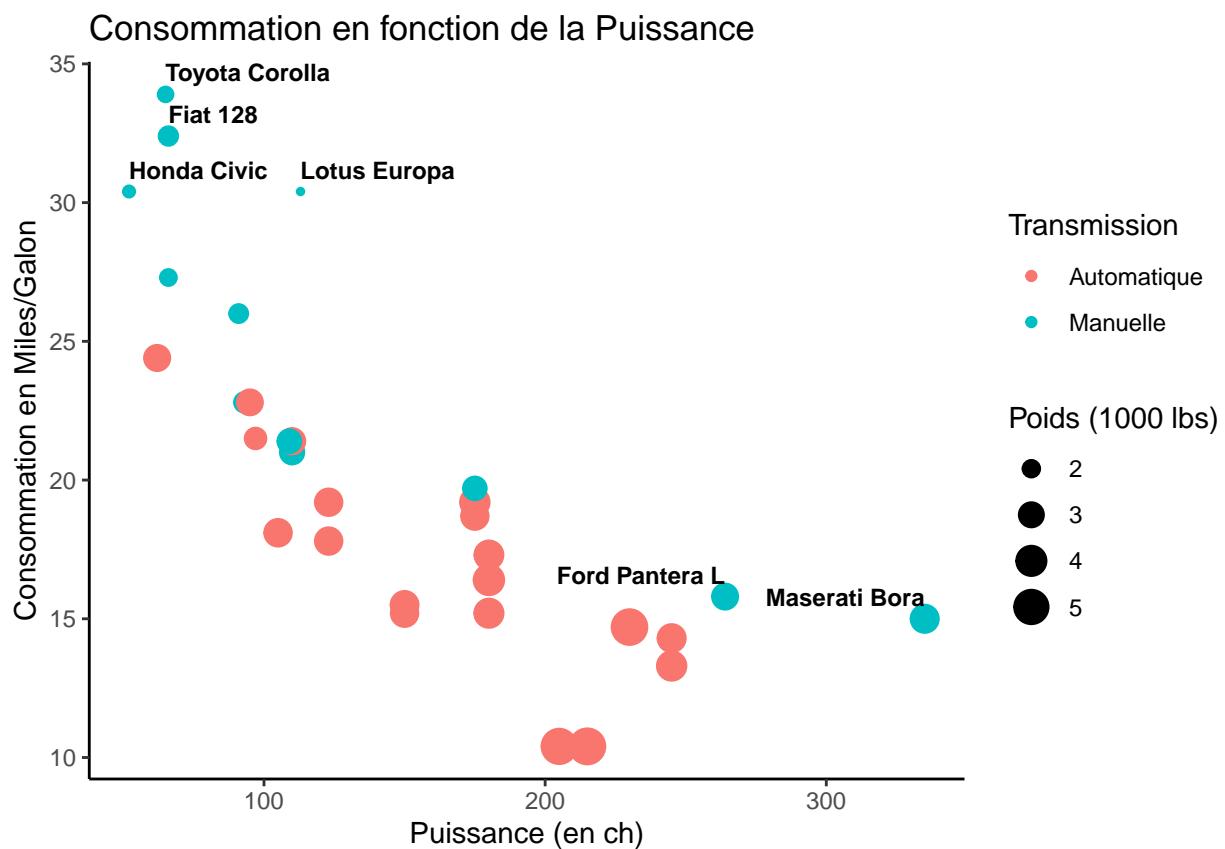
Nous pouvons maintenant reprendre le graphique fait au début, pour le recréer avec la fonction `ggplot()`. Ici, tout est automatisé (couleur, légende, labels) et le code sera beaucoup plus robuste à de nouvelles données.

```
ggplot(mtcars %>% rownames_to_column(),
       aes(x = hp, y = mpg,
```

```

color = factor(am, labels = c("Automatique", "Manuelle")),
size = wt,
label = ifelse(hp <= 250 & mpg <=30, "", rowname)) +
geom_point() +
geom_text(size = 3, color = "black",
vjust = -.75, fontface = "bold",
hjust="inward") +
ggtitle("Consommation en fonction de la Puissance") +
labs(x = "Puissance (en ch)",
y = "Consommation en Miles/Galon",
color = "Transmission",
size = "Poids (1000 lbs)") +
theme_classic()

```



Vous pouvez maintenant le stocker et l'enregistrer !

## Applications

A partir du jeu de données `txhousing` effectuez des représentations graphiques jolies et pertinentes à l'aide de la fonction `ggplot()`. Vous commenterez succinctement chaque représentation graphique afin d'expliquer ce que vous avez voulu mettre en avant à l'aide de cette dernière.

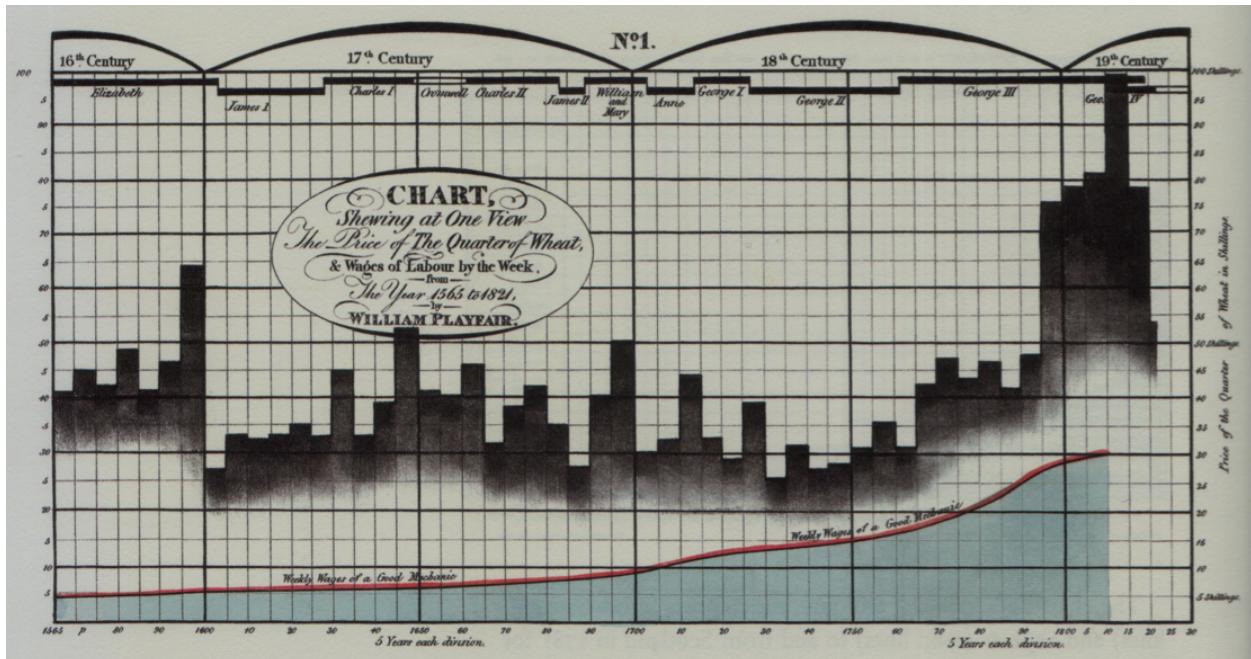
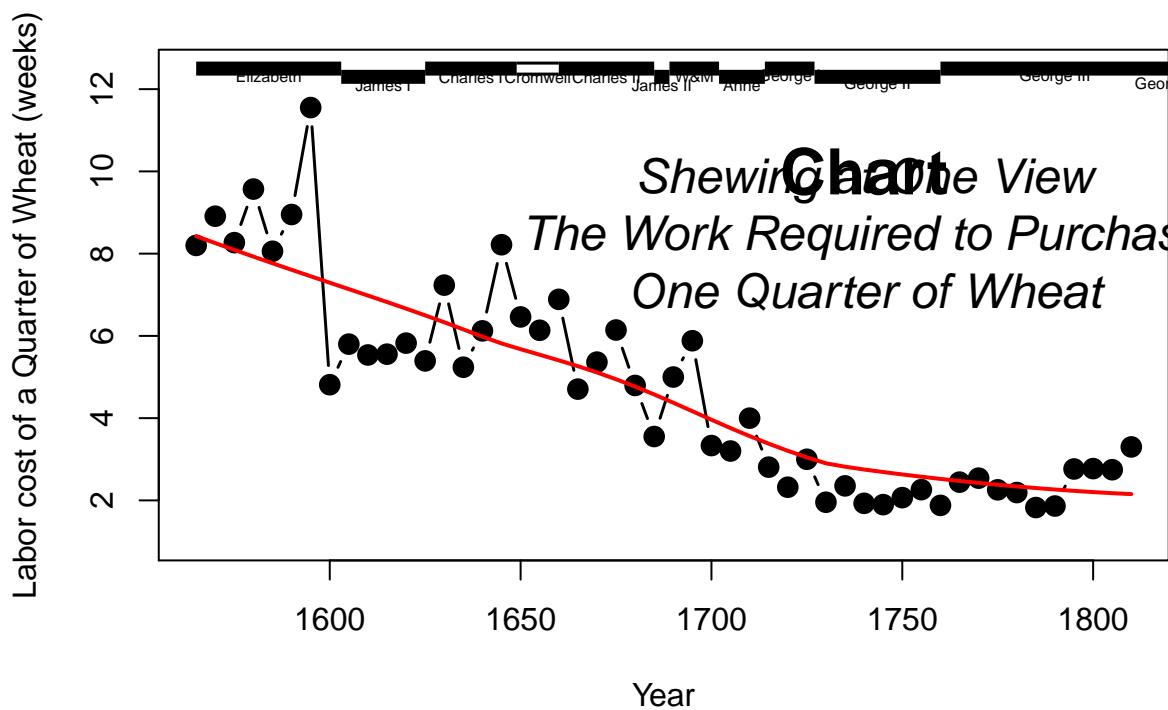
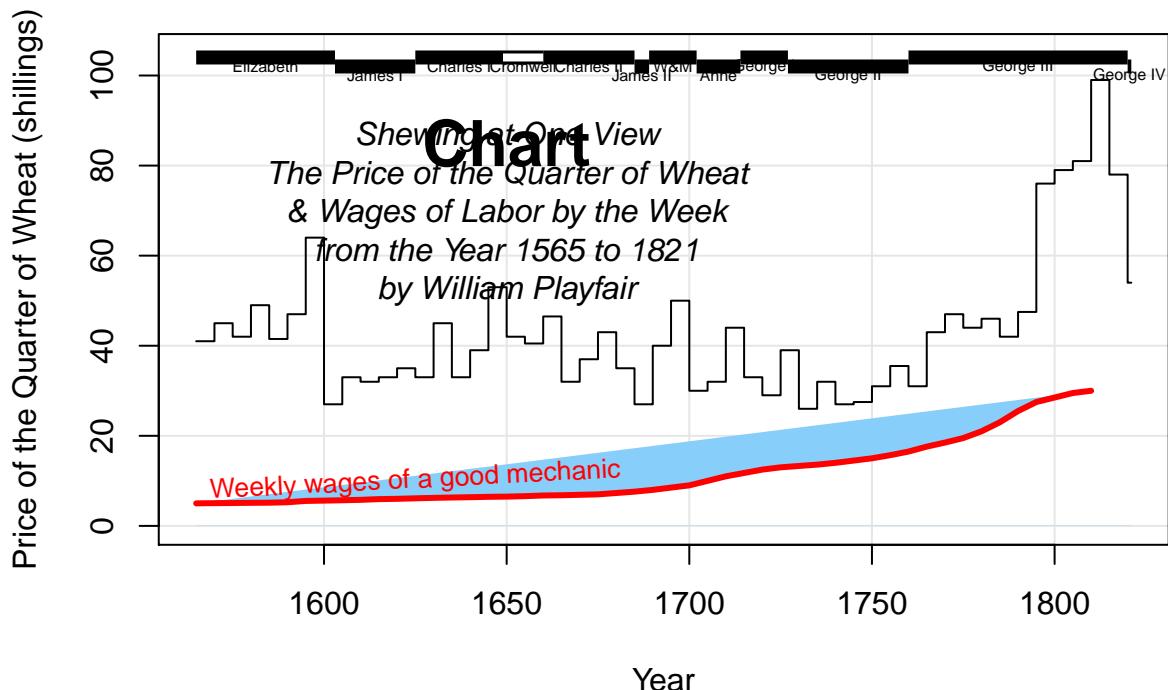


Figure 1: Le graphique original de William Playfair (1759-1823)

## Retour sur Hist Data

```
# Package contenant des jeux de données sur l'histoire de la statistique et de la data visualisation
library(HistData)
library(Guerry) # spécifique au jeu de données Guerry
```

William Playfair (1759-1823)



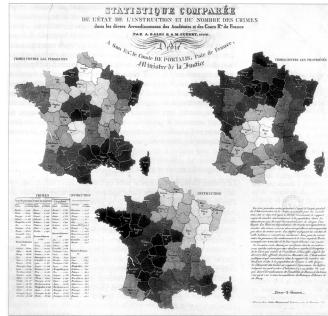
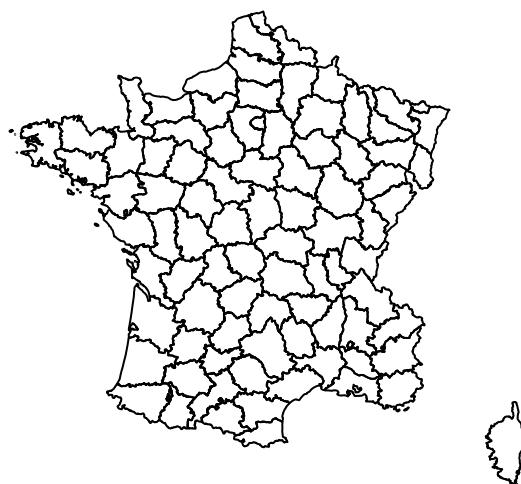
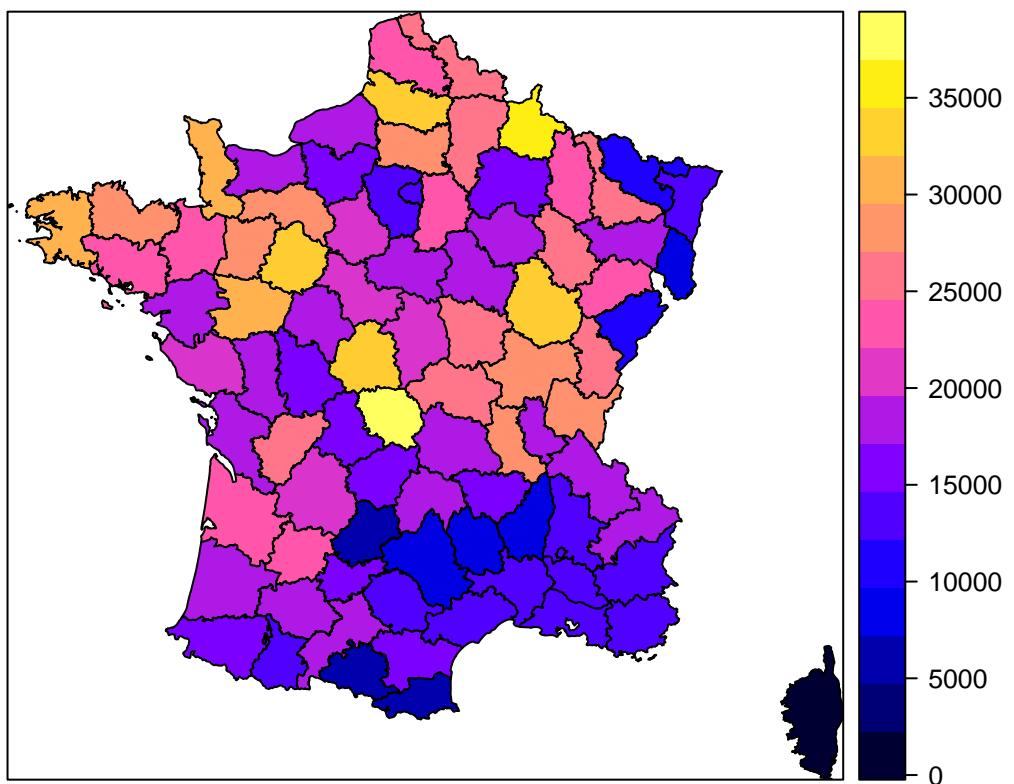


Figure 2: Le graphique original de André-Michel Guerry (1802-1866)

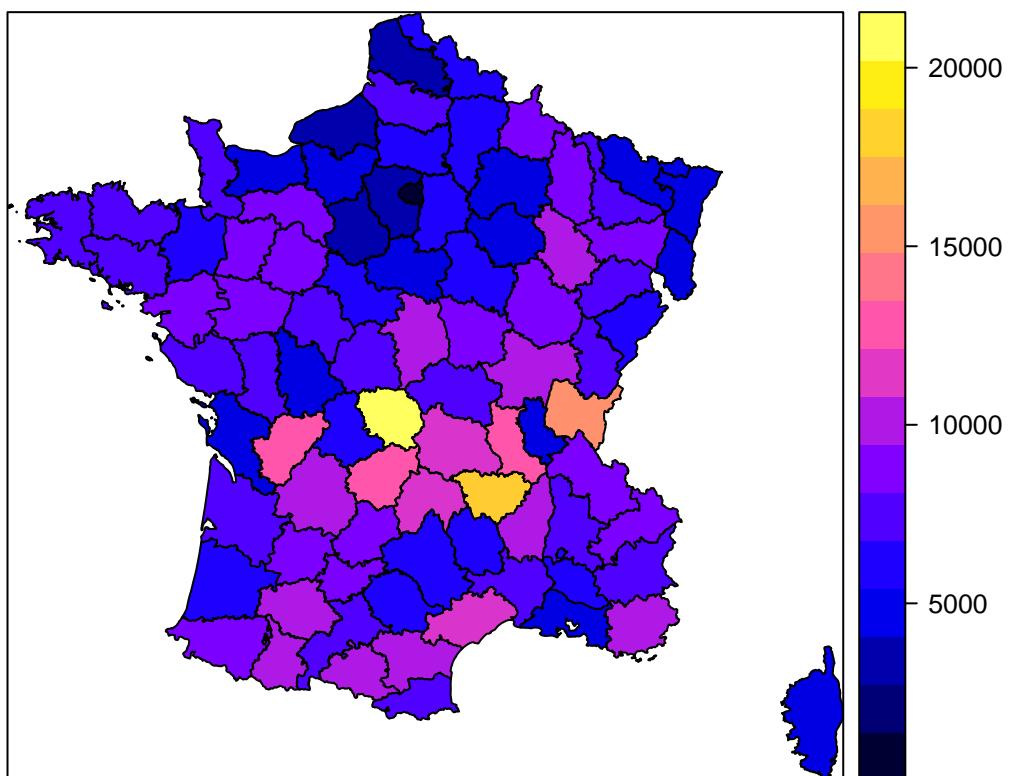
### André-Michel Guerry (1802-1866)



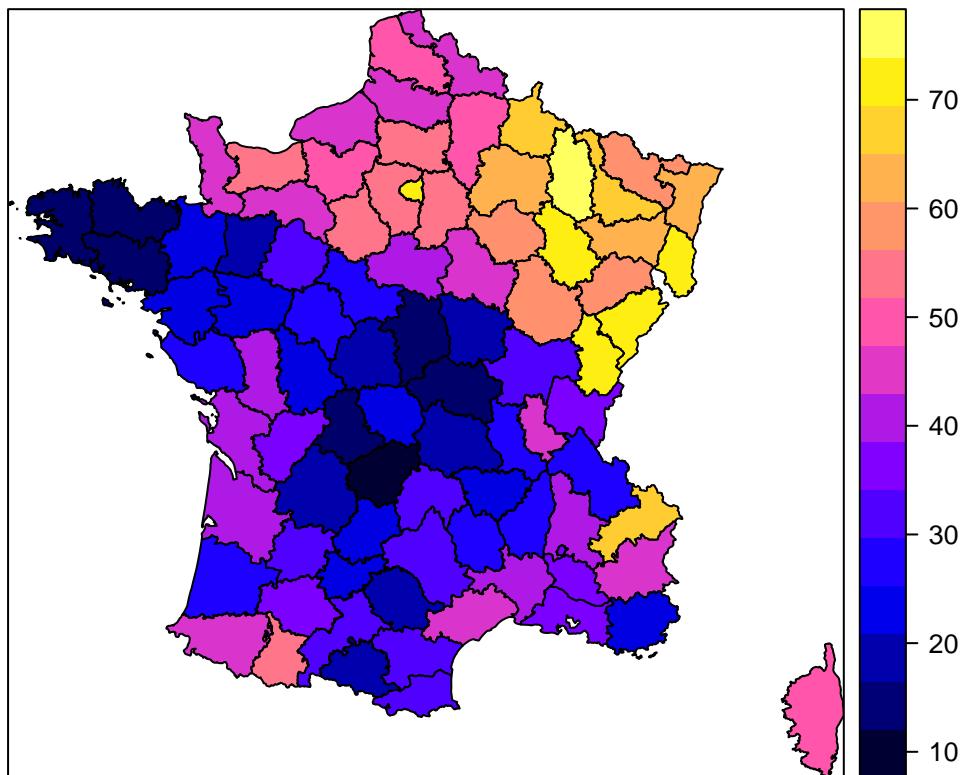
```
spplot(gfrance, "Crime_pers")
```



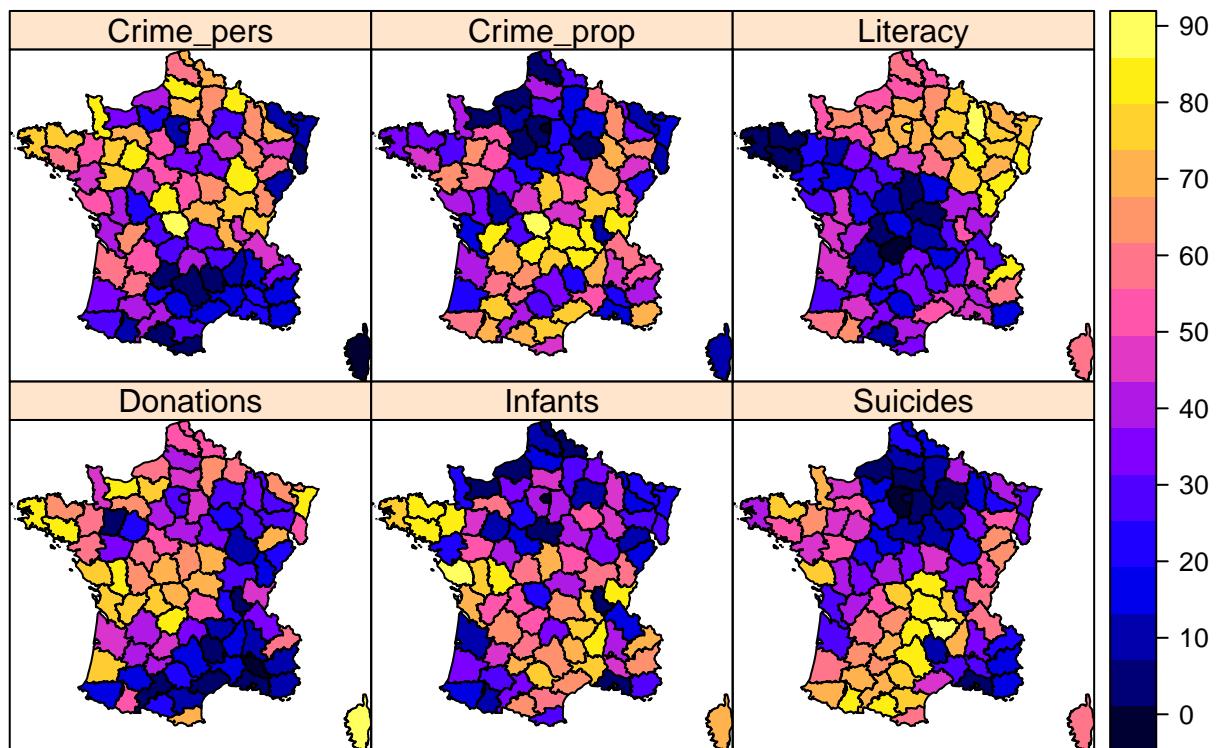
```
spplot(gfrance, "Crime_prop")
```



```
splot(gfrance, "Literacy")
```



**Guerry's main moral variables**



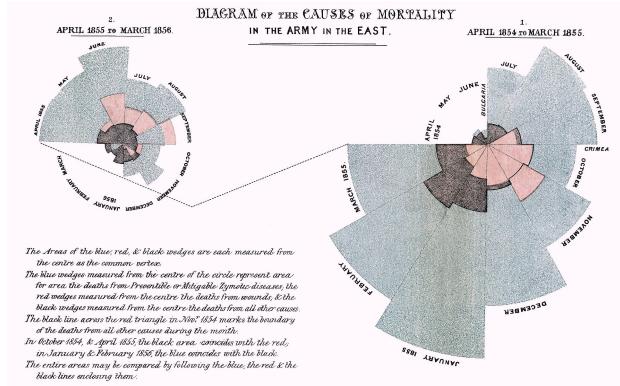
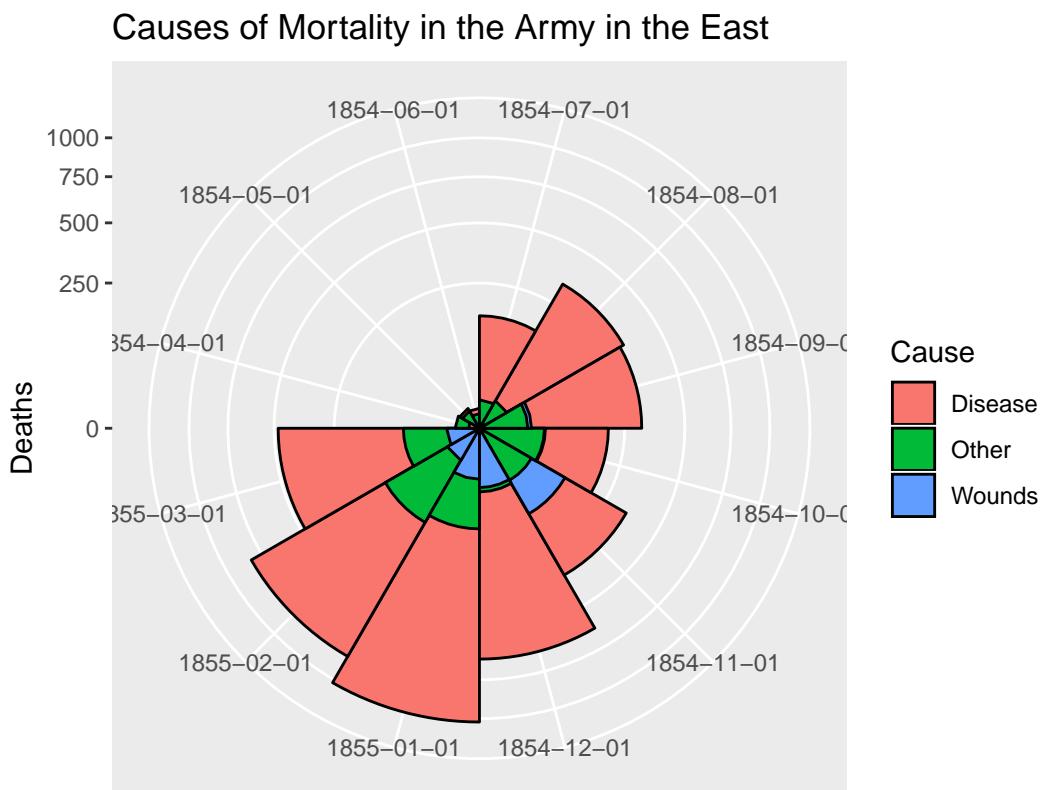
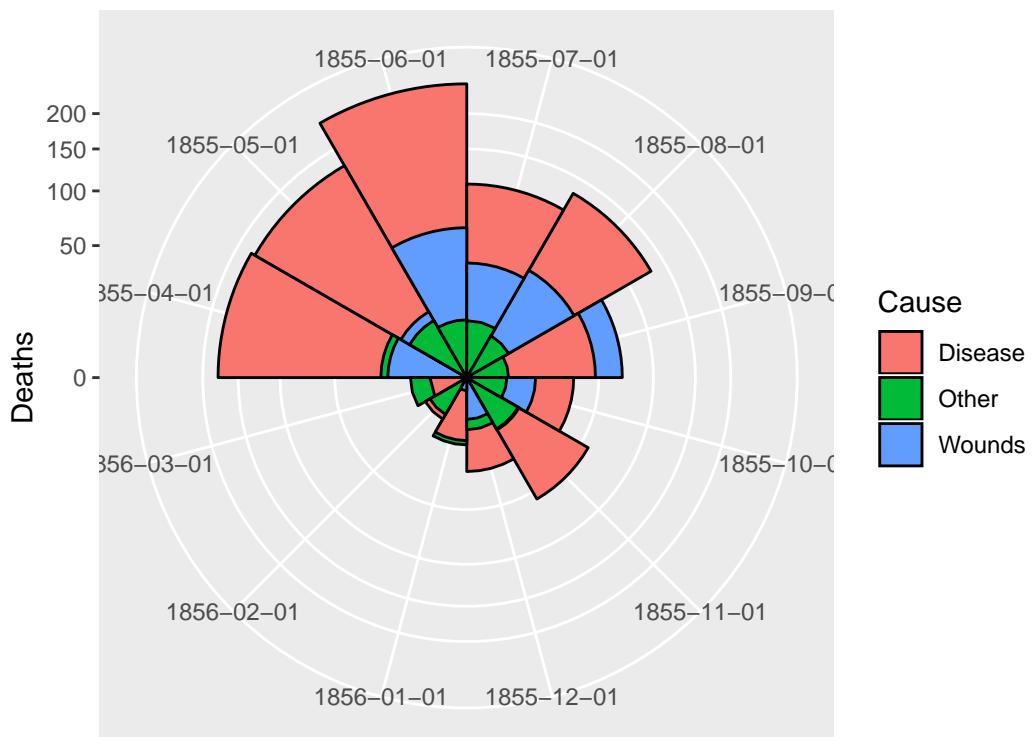


Figure 3: Le graphique original de

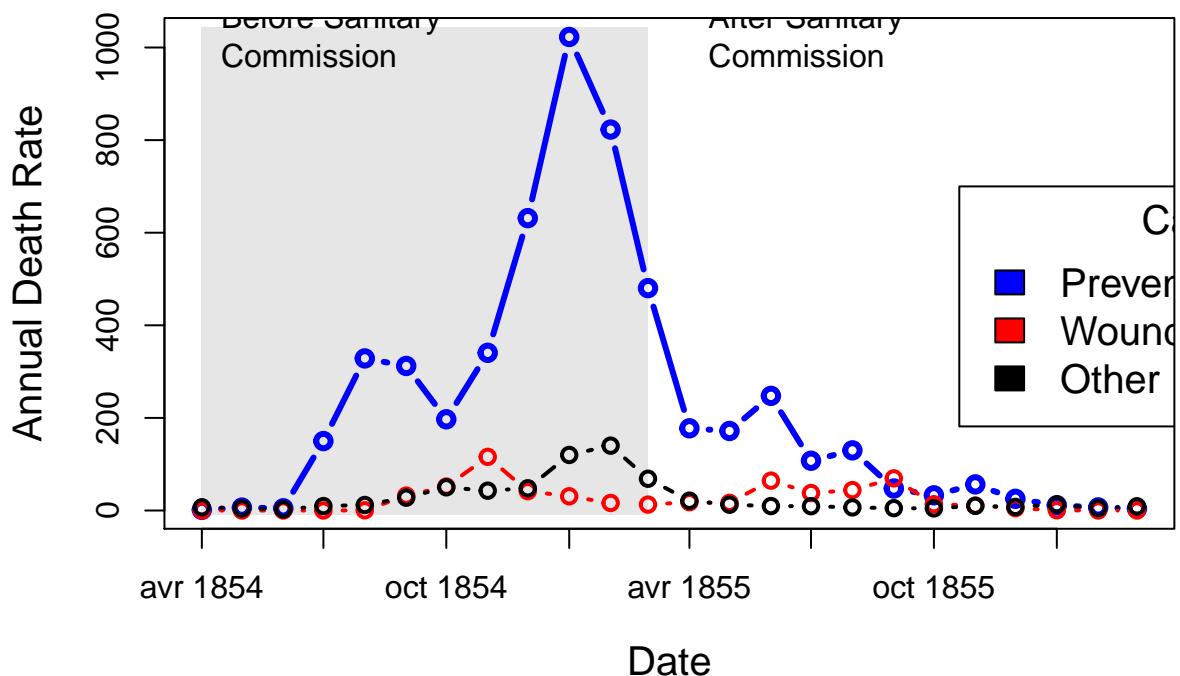
Florence Nightingale (1820-1910)

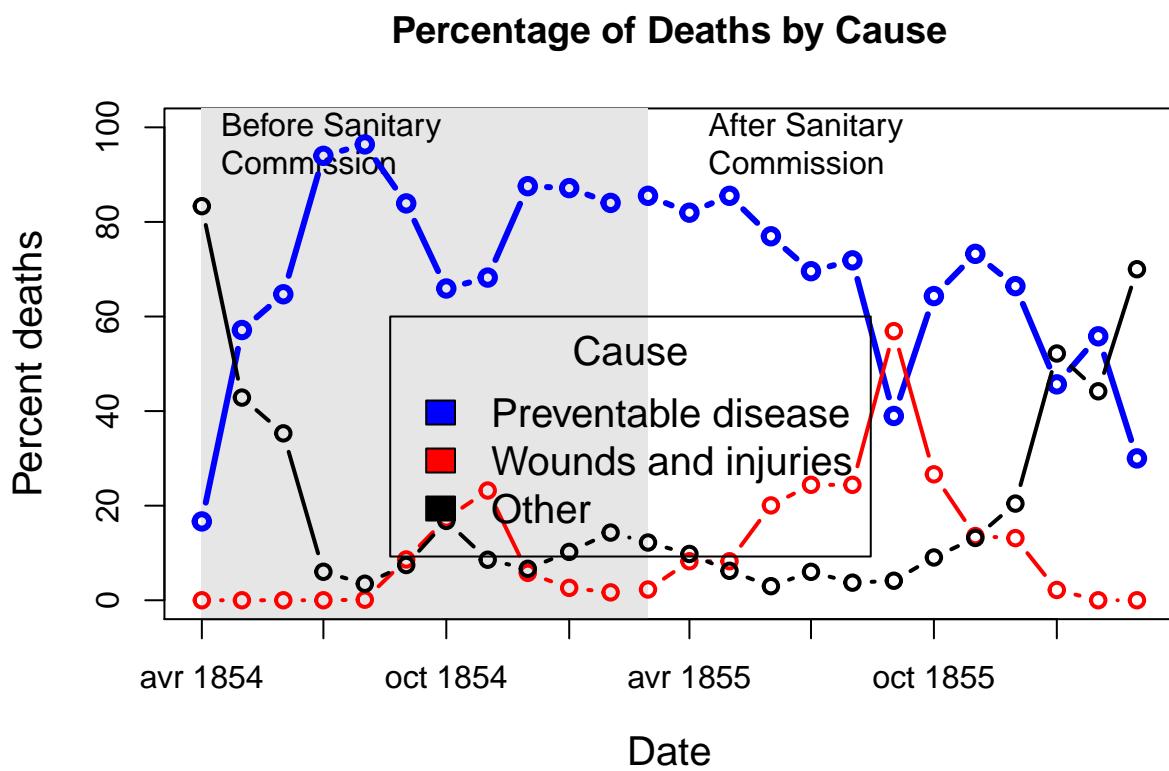


### Causes of Mortality in the Army in the East



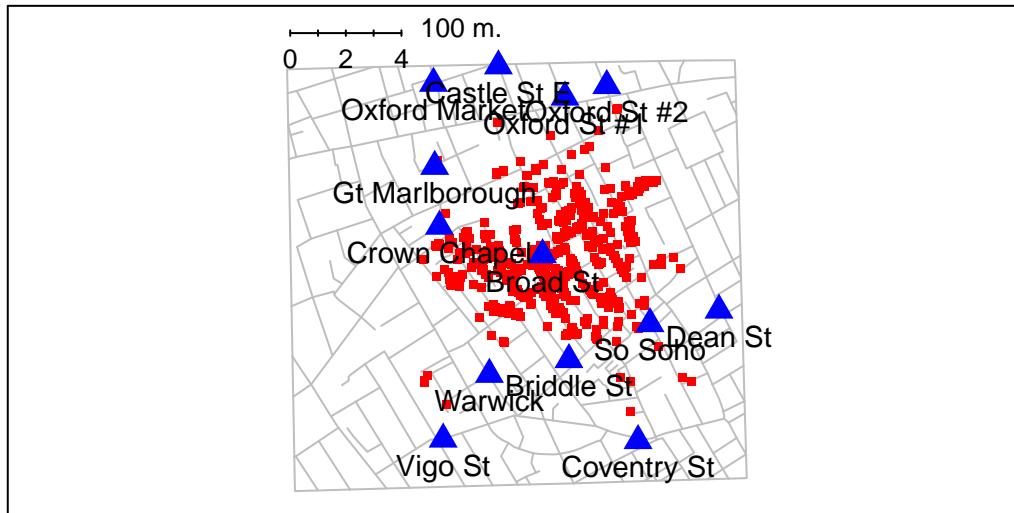
### Causes of Mortality of the British Army in the East





John Snow (1813-1858)

### Snow's Cholera Map of London



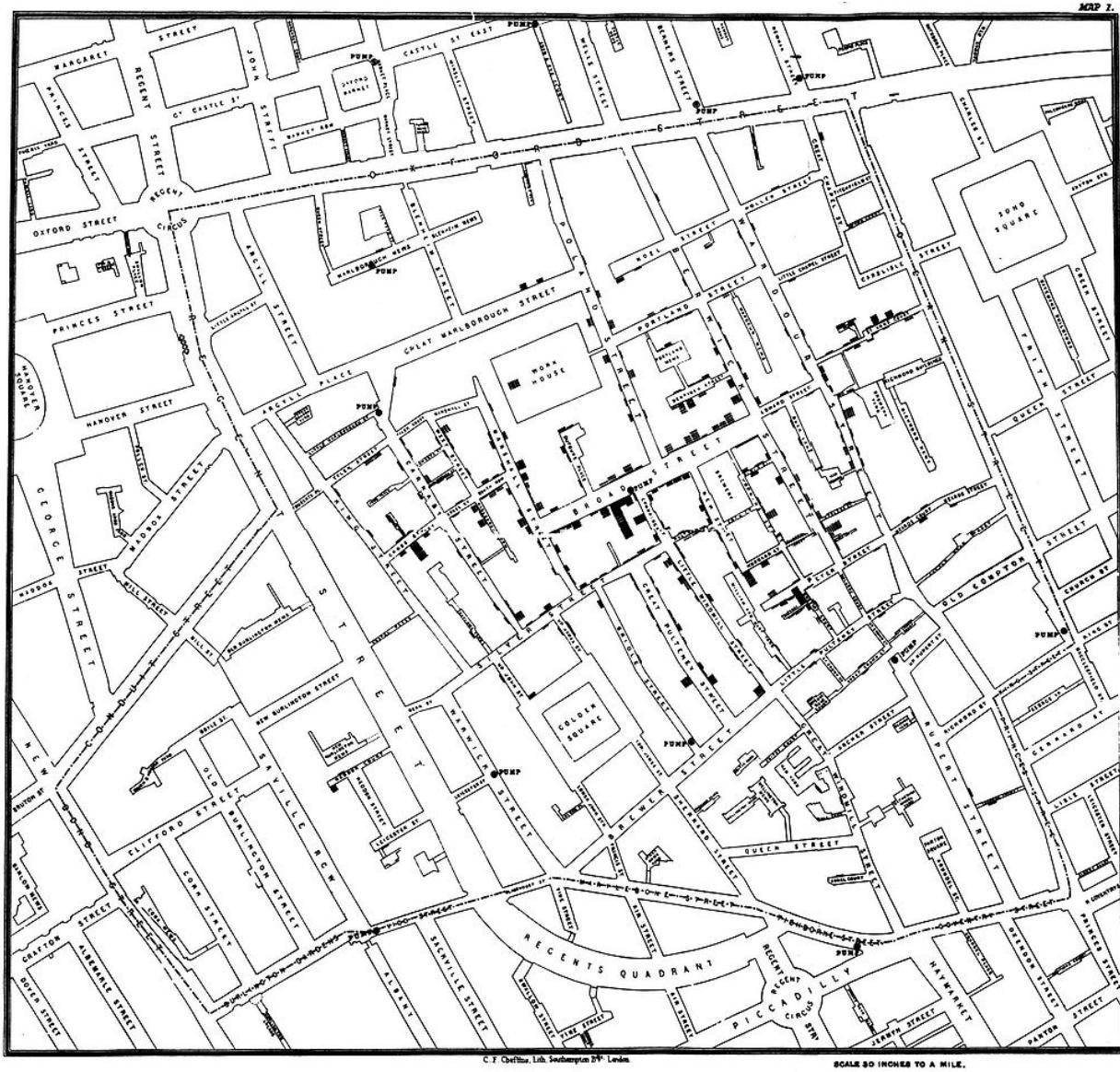


Figure 4: Le graphique original de John Snow (1813-1858)

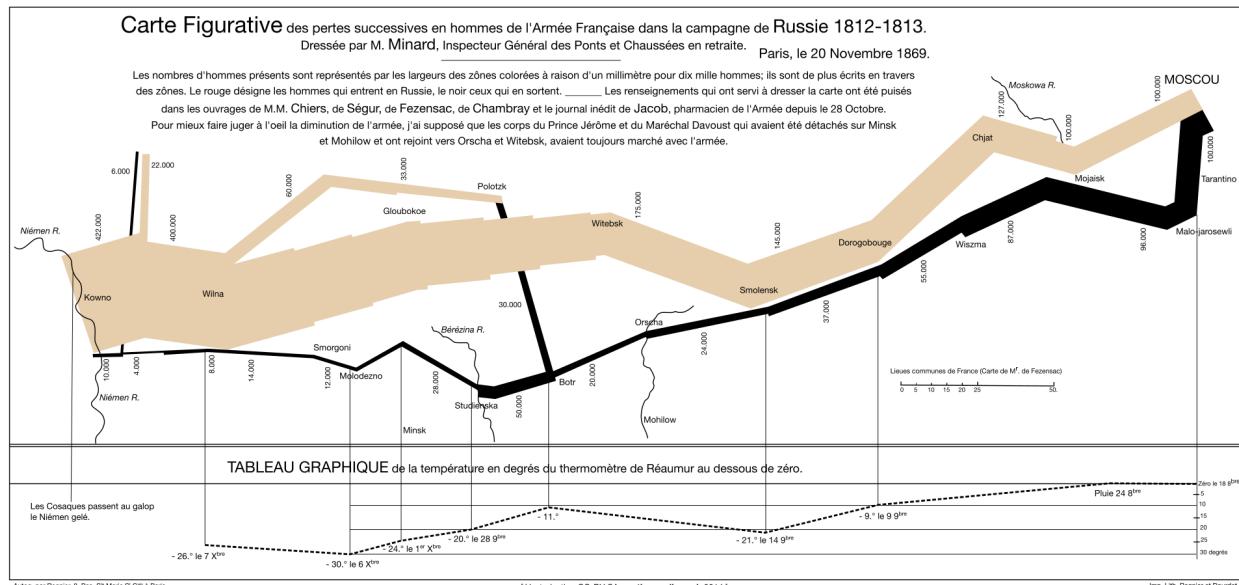
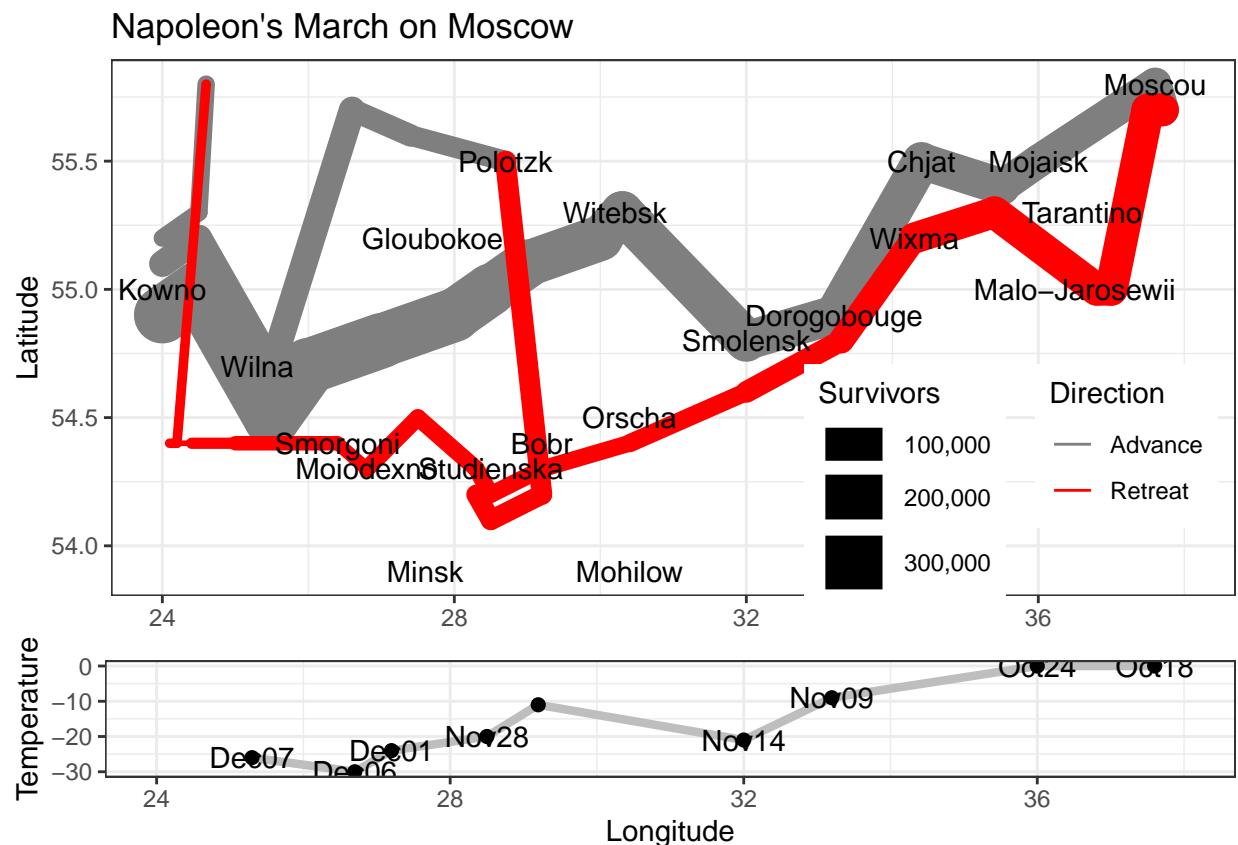


Figure 5: Le graphique original de Charles Joseph Minard (1781-1870)

Charles Joseph Minard (1781-1870)



<https://www.yvesago.net/pourquoi/2014/03/r-la-campagne-de-russie-par-minard.html>

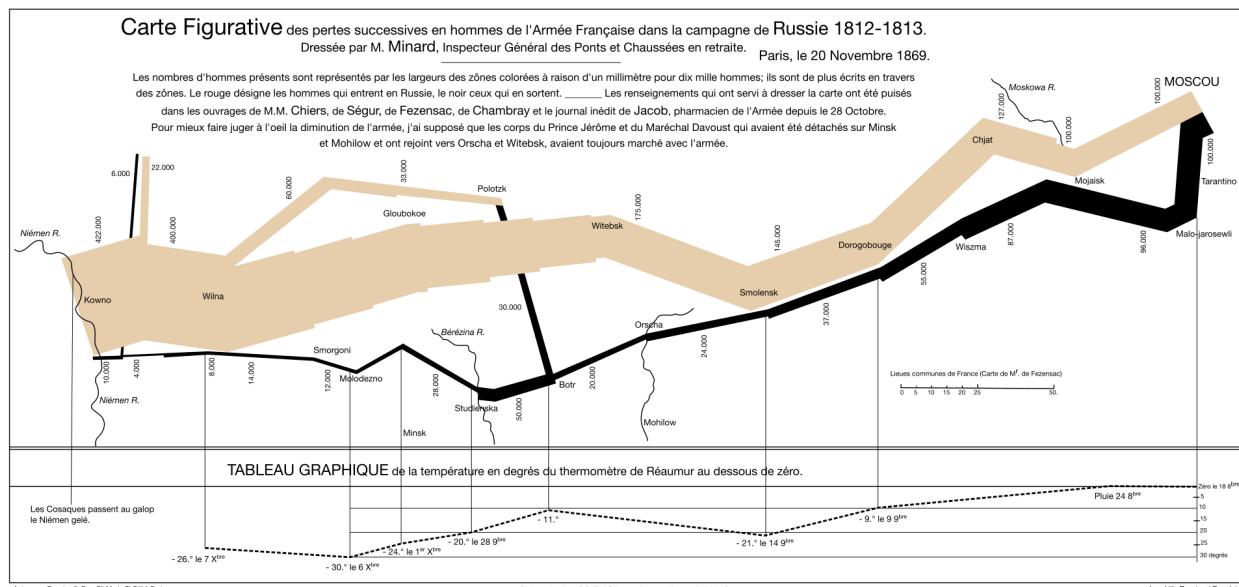


Figure 6: Le graphique revisitée de Charles Joseph Minard (1781-1870)