AA18.1: Découverte du tidyverse

Visualiser avec `ggplot2`

`ggplot2` est une extension du *tidyverse* qui permet de générer des graphiques avec une syntaxe cohérente et puissante. Elle nécessite l'apprentissage d'un "mini-langage" supplémentaire, mais permet la construction de graphiques complexes de manière efficace.

Une des particularités de `ggplot2` est qu'elle part du principe que les données relatives à un graphique sont stockées dans un tableau de données (*data frame*, *tibble* ou autre).

Préparation

`ggplot2` fait partie du coeur du *tidyverse*, elle est donc chargée automatiquement avec :

```
library(tidyverse)
```

On peut également la charger explicitement avec :

```
library(ggplot2)
```

Dans ce qui suit on utilisera le jeu de données des scories de culot (culot_blignicourt.csv) Le plus simple est de cliquer dessus puis choisir Ìmport dataset`

```
library(readr)
co <- read_delim("culot_blignicourt.csv",
    ";", escape_double = FALSE, trim_ws = TRUE)
```

```
## Parsed with column specification:
## cols(
##   id = col_integer(),
##   st = col_integer(),
##   us = col_character(),
##   facies = col_character(),
##   forme = col_character(),
##   morphologie = col_character(),
##   aspect = col_character(),
##   longueur = col_integer(),
##   largeur = col_integer(),
##   epaisseur = col_integer(),
##   poids = col_integer(),
##   magnetisme = col_character()
## )
```

```
View(co)
```

Initialisation

Un graphique `ggplot2` s'initialise à l'aide de la fonction `ggplot()`. Les données représentées graphiquement sont toujours issues d'un tableau de données (*data frame* ou *tibble*), qu'on passe en argument `data` à la fonction :

```
ggplot(data = co)
## Ou, équivalent
ggplot(co)
```

On a défini la source de données, il faut maintenant ajouter des éléments de représentation graphique. Ces éléments sont appelés des `geom`, et on les ajoute à l'objet graphique de base avec l'opérateur `+`.

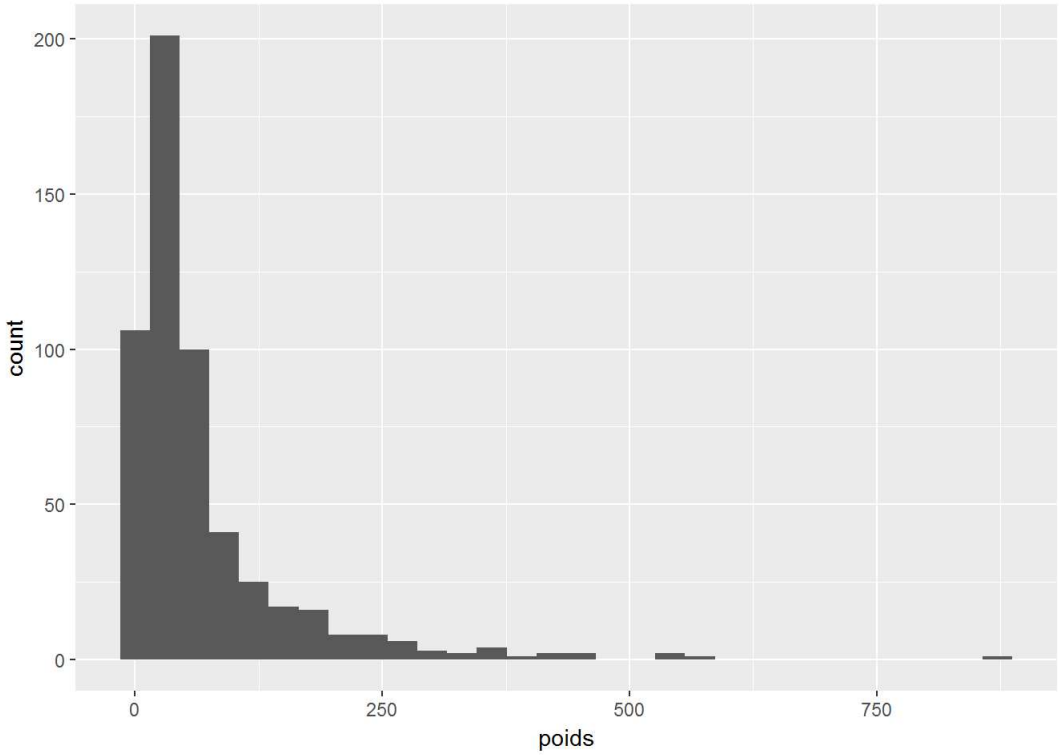
Un des `geom` les plus simples est `geom_histogram`. On peut l'ajouter de la manière suivante :

```
ggplot(co) + geom_histogram()
```

Reste à indiquer quelle donnée nous voulons représenter sous forme d'histogramme. Cela se fait à l'aide d'arguments passés via la fonction `aes()`. Ici nous avons un paramètre à renseigner, `x`, qui indique la variable à représenter sur l'axe des x (l'axe horizontal). Par exemple :

```
ggplot(co) + geom_histogram(aes(x = poids))
```

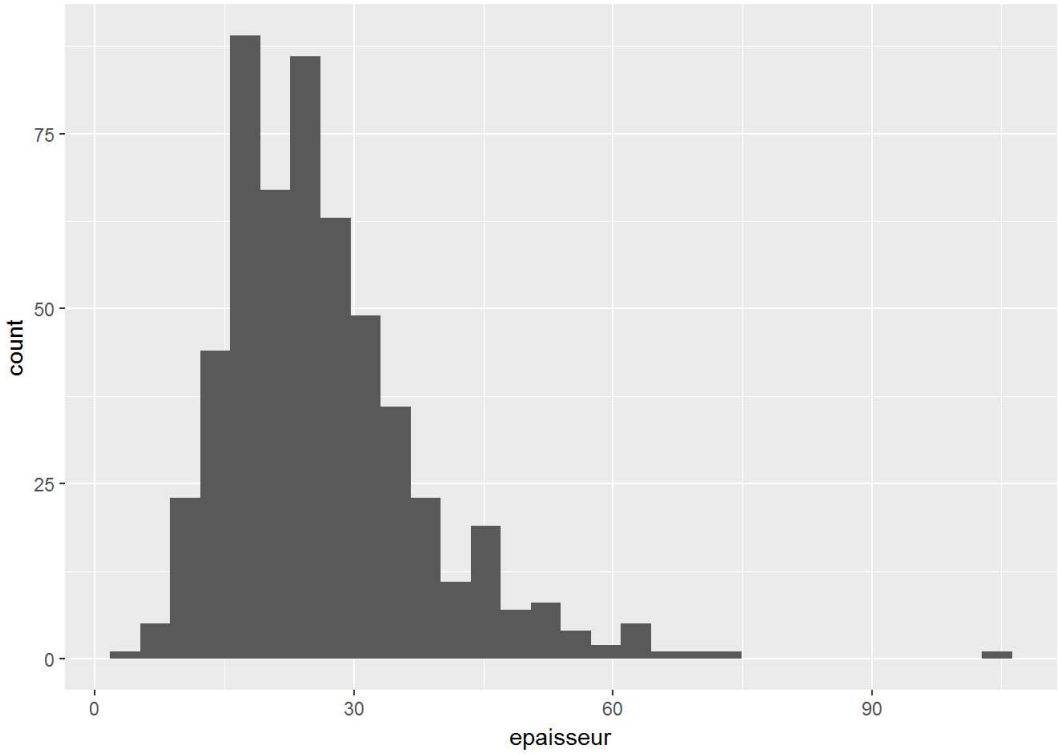
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Si on veut représenter une autre variable, il suffit de changer la valeur de `x` :

```
ggplot(co) + geom_histogram(aes(x = epaisseur))
```

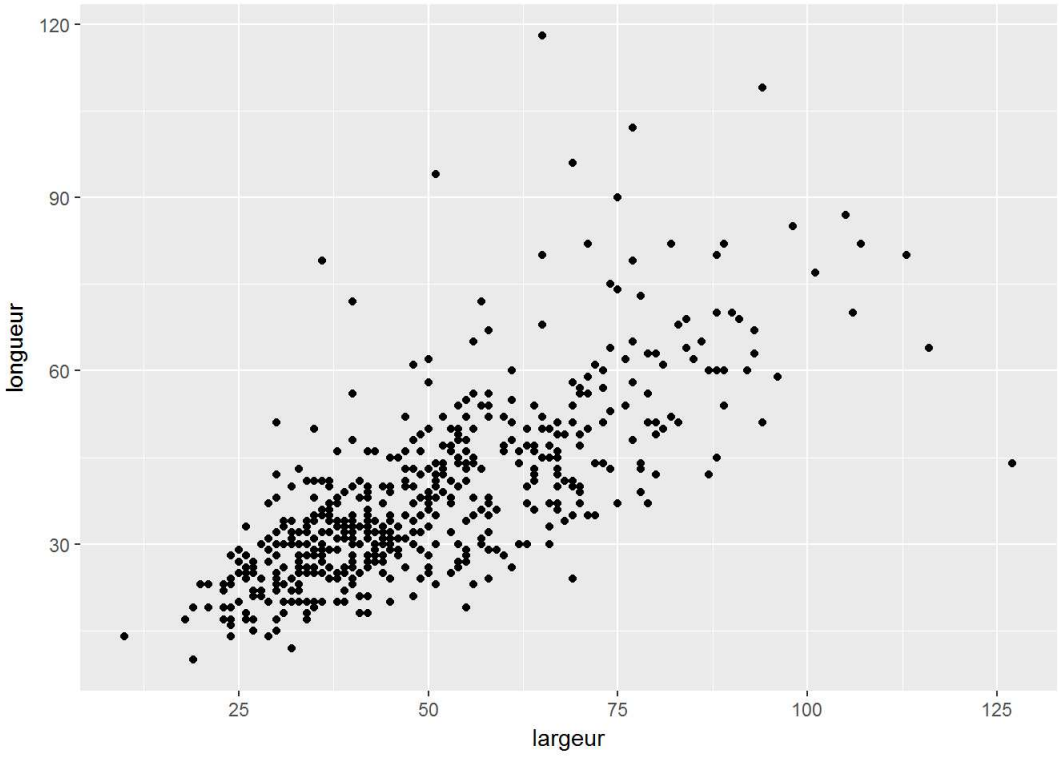
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Quand on spécifie une variable, inutile d'indiquer le nom du tableau de données sous la forme `co$epaisseur`, car `ggplot2` recherche automatiquement la variable dans le tableau de données indiqué avec le paramètre `data`. On peut donc se contenter de `epaisseur`.

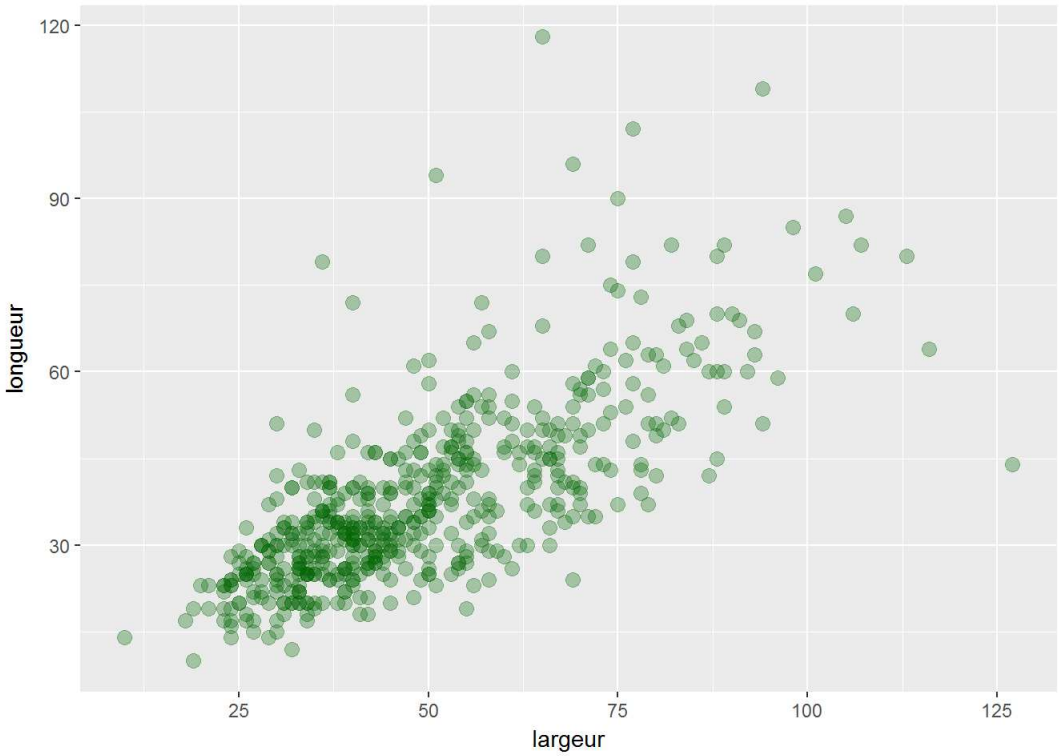
cotains `geom` prennent plusieurs paramètres. Ainsi, si on veut représenter un nuage de points, on peut le faire en ajoutant un `geom_point`. On doit alors indiquer à la fois la position en `x` et en `y` de ces points, il faut donc passer ces deux arguments à `aes()` :

```
ggplot(co) + geom_point(aes(x = largeur, y = longueur))
```



On peut modifier cotains attributs graphiques d'un `geom` en lui passant des arguments supplémentaires. Par exemple, pour un nuage de points, on peut modifier la couleur des points avec l'argument `color`, leur taille avec l'argument `size`, et leur transparence avec l'argument `alpha` :

```
ggplot(co) +
  geom_point(aes(x = largeur, y = longueur),
             color = "darkgreen", size = 3, alpha = 0.3)
```



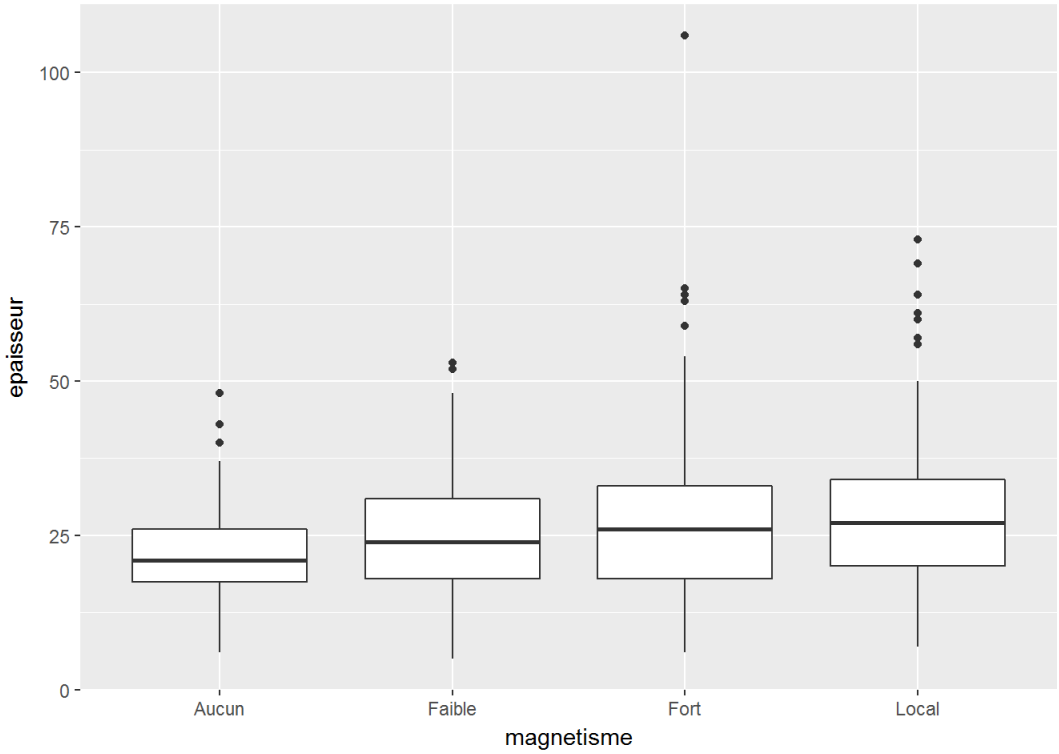
Exemples de `geom`

Il existe un grand nombre de `geom`, décrits en détail dans la documentation officielle (http://ggplot2.tidyverse.org/reference/). Outre `geom_histogram` et `geom_point`, on pourra noter les `geom` suivants.

`geom_boxplot`

`geom_boxplot` permet de représenter des boîtes à moustaches. On lui passe en `y` la variable dont on veut étudier la répartition, et en `x` la variable contenant les classes qu'on souhaite comparer. Ainsi, si on veut comparer la répartition des epaisseur en fonction de leur magnétisme, on pourra faire :

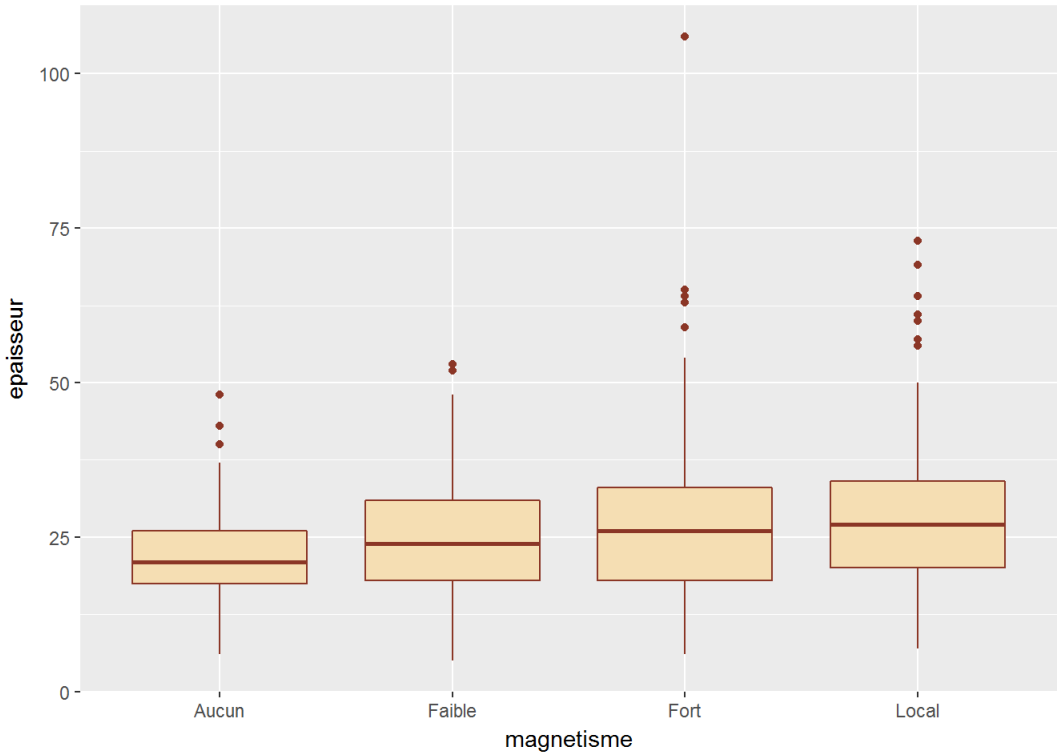
```
ggplot(co) + geom_boxplot(aes(x = magnetisme, y = epaisseur))
```



À noter que dans ce cas, `x` doit être une variable qualitative, et `y` une variable numérique.

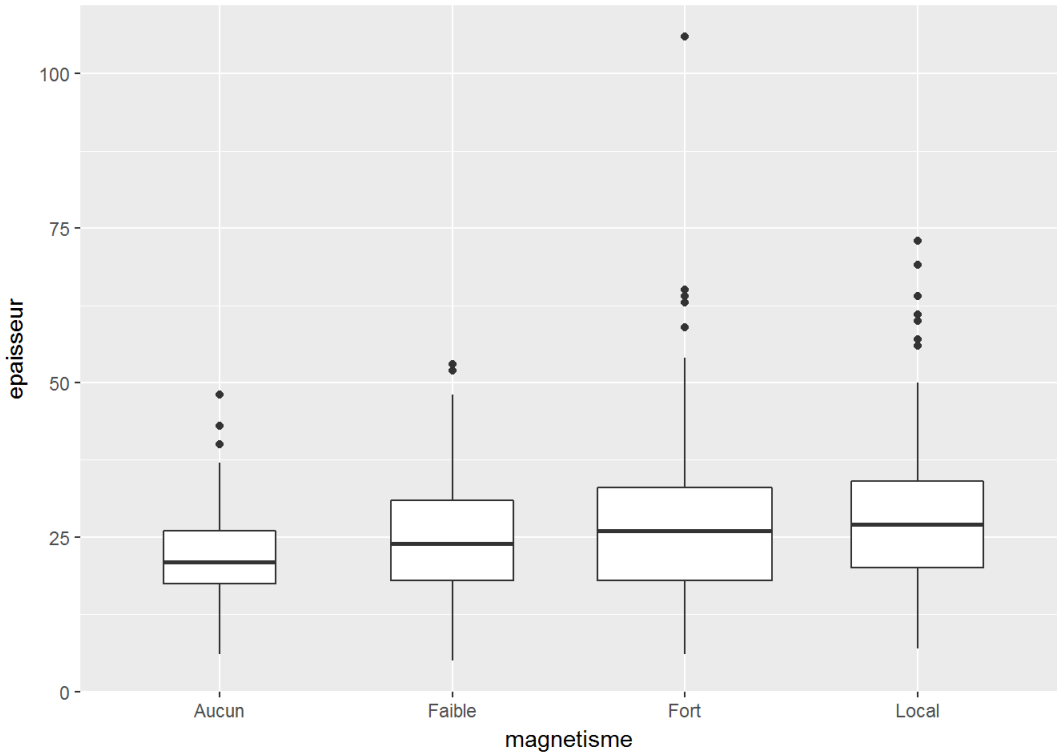
On peut personnaliser la présentation avec différents argument supplémentaires :

```
ggplot(co) +
  geom_boxplot(aes(x = magnetisme, y = epaisseur), fill = "wheat", color = "tomato4")
```



Un autre argument utile, `varwidth`, permet de faire varier la largeur des boîtes en fonction des effectifs de la classe (donc, ici, en fonction du nombre de scories de chaque niveau de magnétisme) :

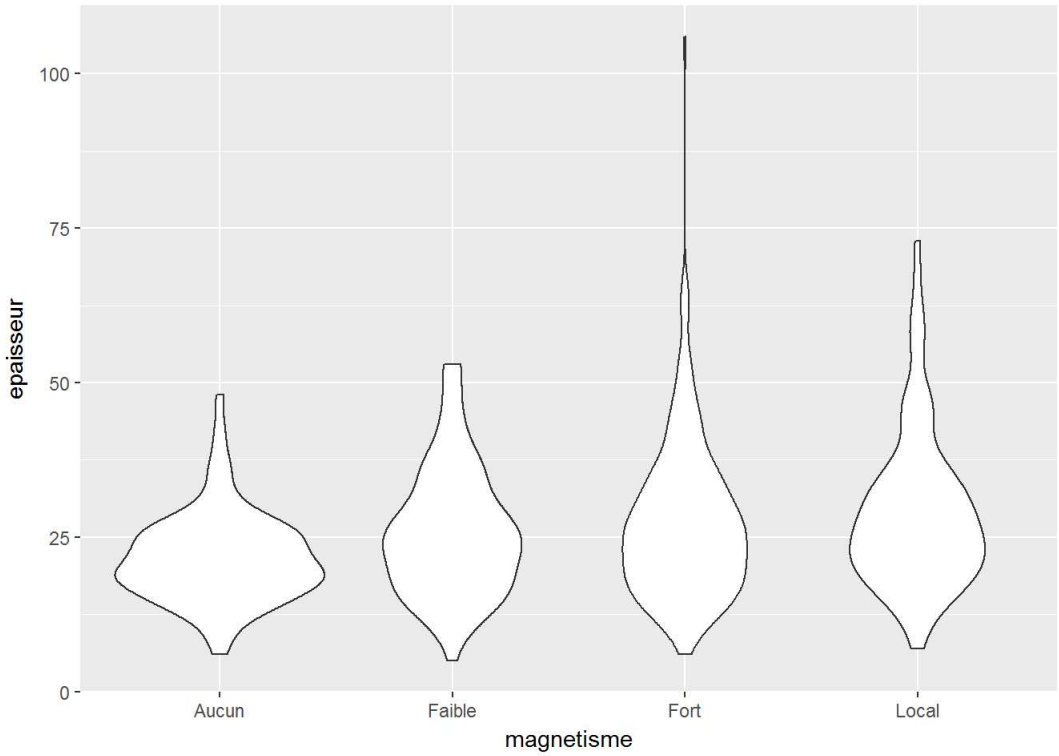
```
ggplot(co) +
  geom_boxplot(aes(x = magnetisme, y = epaisseur), varwidth = TRUE)
```



geom_violin

`geom_violin` est très semblable à `geom_boxplot` , mais utilise des graphes en violon à la place des boîtes à moustache.

```
ggplot(co) + geom_violin(aes(x = magnetisme, y = epaisseur))
```



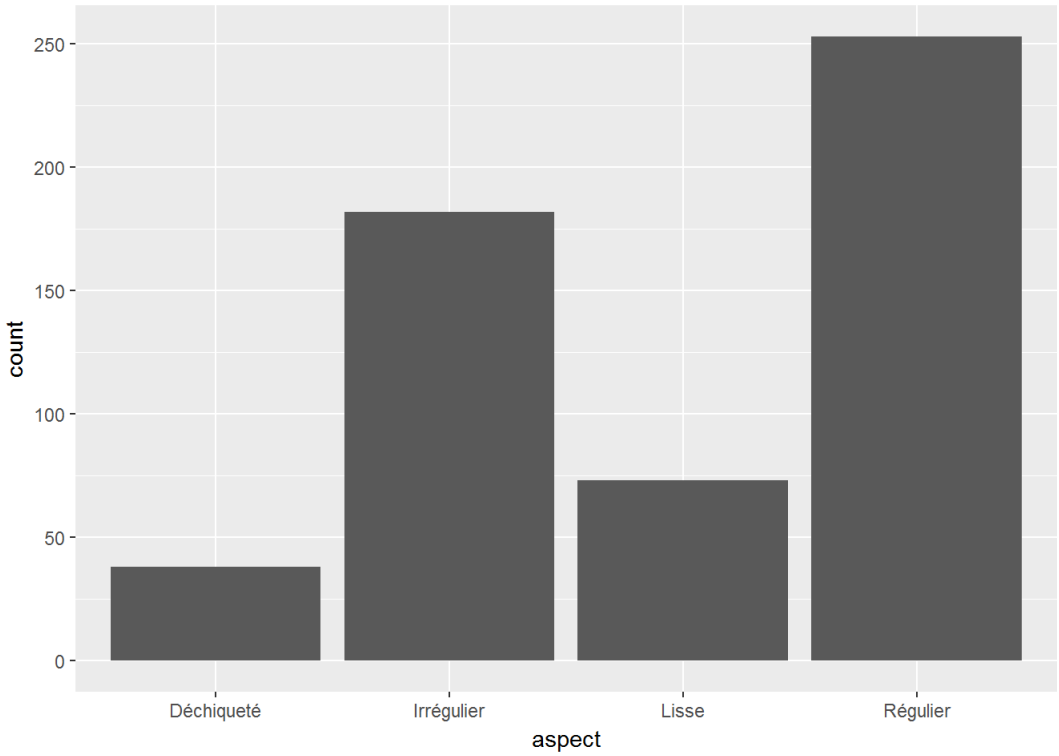
Les graphes en violon peuvent donner une lecture plus fine des différences de distribution selon les classes.

geom_bar

`geom_bar` permet de produire un graphique en bâtons (*bacolot*). On lui passe en `x` la variable qualitative dont on souhaite représenter l'effectif de chaque modalité.

Par exemple, si on veut afficher le nombre de scories de notre jeu de données pour chaque aspect :

```
ggplot(co) + geom_bar(aes(x = aspect))
```



Un cas assez fréquent mais un peu plus complexe survient quand on a déjà calculé le tri à plat de la variable à représenter. Dans ce cas on souhaite que `geom_bar` représente les effectifs sans les calculer : cela se fait en indiquant un mappage `y` pour la variable contenant les effectifs précalculés, et en ajoutant l'argument `stat = "identity"` .

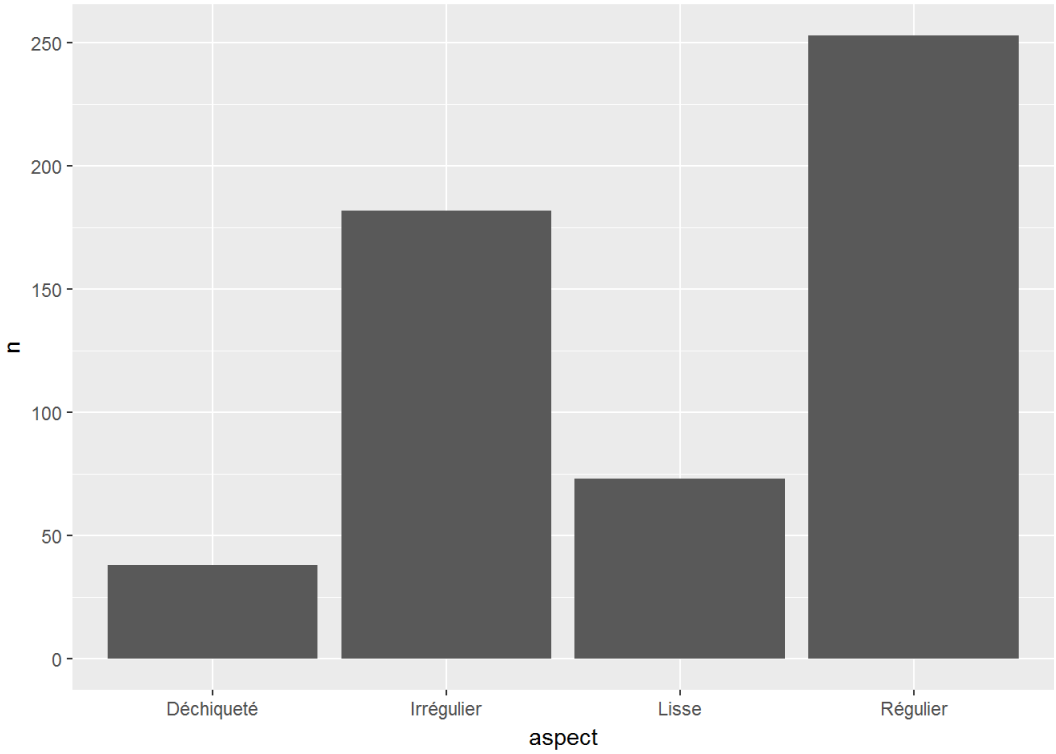
Par exemple, si on a les données sous cette forme :

aspect <fctr>	**n** <int>
Déchiqueté	38
Irrégulier	182
Lisse	73
Régulier	253

4 rows

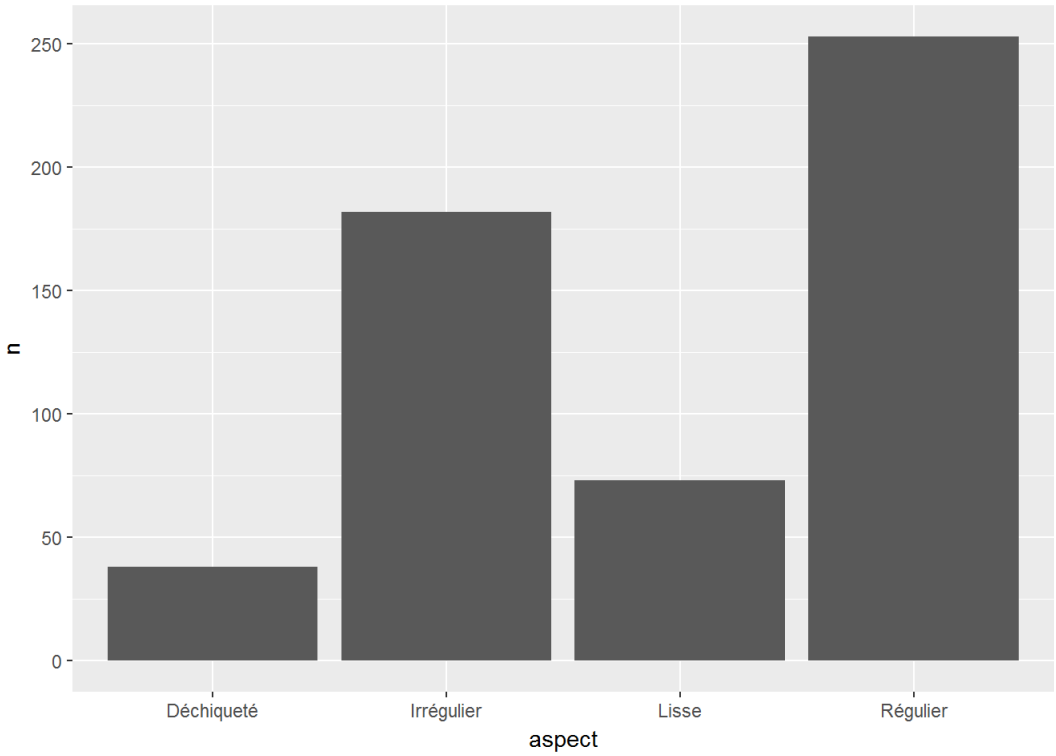
On peut obtenir le graphique souhaité ainsi :

```
ggplot(df) + geom_bar(aes(x = aspect, y = n), stat = "identity")
```



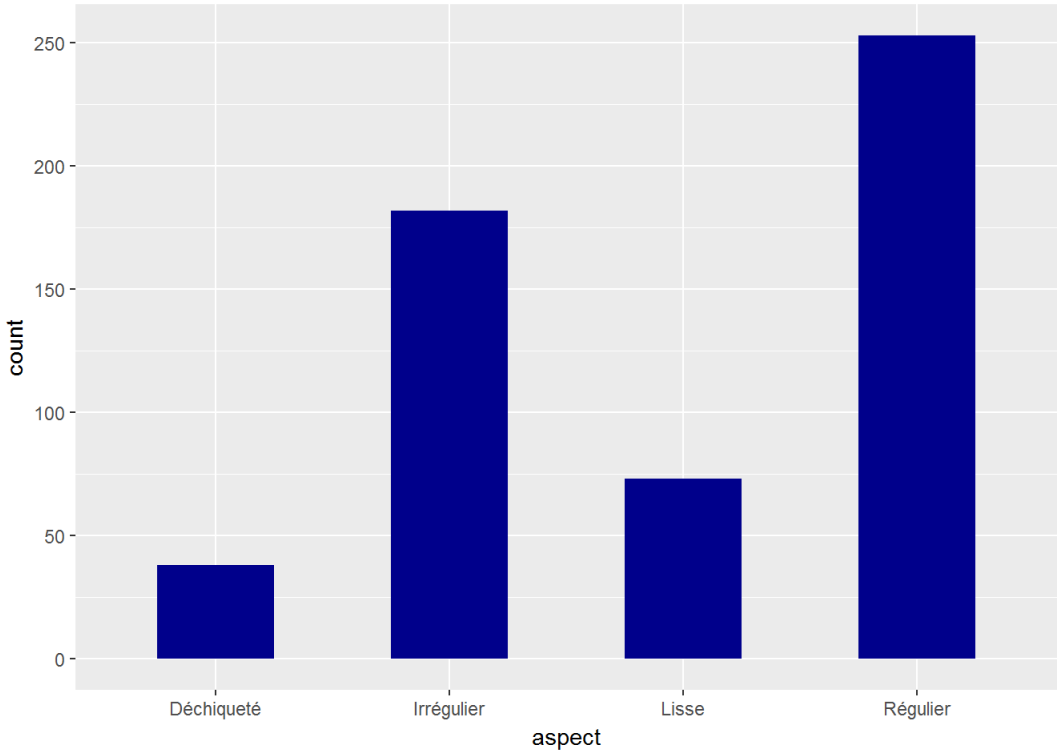
À noter qu'on peut aussi utiliser `geom_col` qui est un raccourci pour appliquer un `geom_bar` avec `stat = "identity"` . La commande précédente est donc équivalente à :

```
ggplot(df) + geom_col(aes(x = aspect, y = n))
```



On peut également modifier l'apparence du graphique avec des arguments supplémentaires comme `fill` (remplissage) ou `width` (largeur/contour):

```
ggplot(co) + geom_bar(aes(x = aspect), fill = "darkblue", width = .5)
```

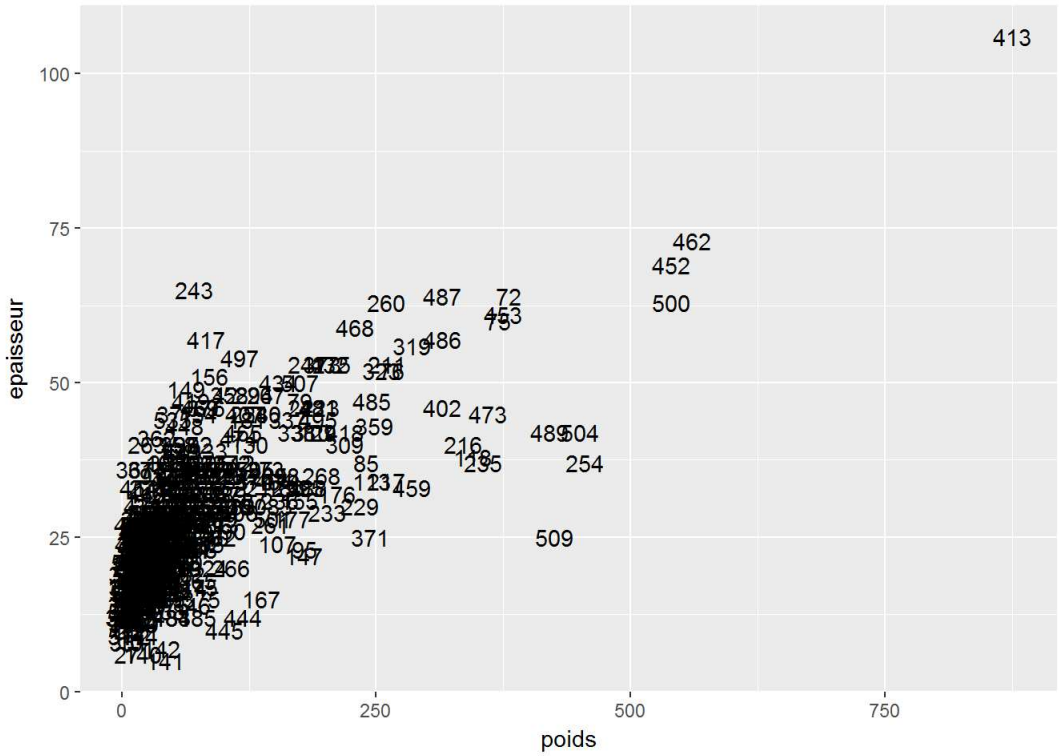


geom_text

`geom_text` représente des points identifiés par des labels. On doit lui passer `x` et `y` pour la position des points, et `label` pour le texte des étiquettes.

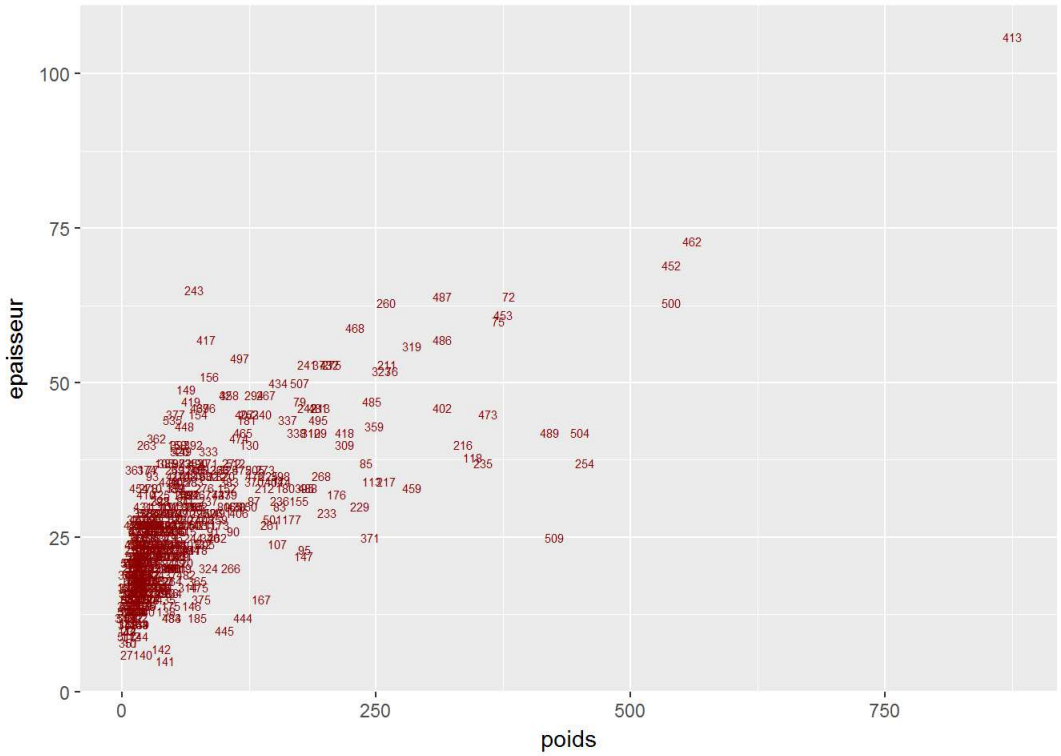
Par exemple, si on souhaite représenter le nuage croisant la part des diplômés du supérieur et la part de cadres, mais en affichant le nom de la commune plutôt qu'un simple point, on peut faire :

```
ggplot(co) + geom_text(aes(x = poids, y = epaisseur, label = id))
```



On peut personnaliser l'apparence et la position du texte avec des arguments comme `size`, `color`, etc.

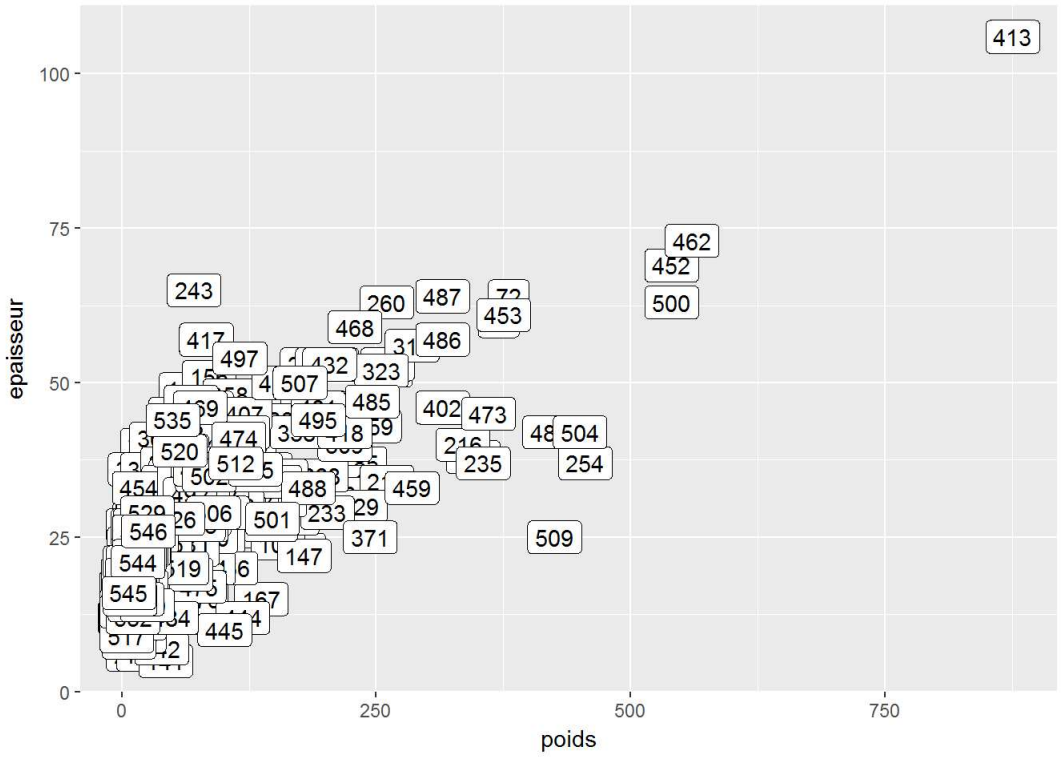
```
ggplot(co) +
  geom_text(aes(x = poids, y = epaisseur, label = id), color = "darkred", size = 2)
```



geom_label

`geom_label` est identique à `geom_text` , mais avec une présentation un peu différente.

```
ggplot(co) + geom_label(aes(x = poids, y = epaisseur, label = id))
```

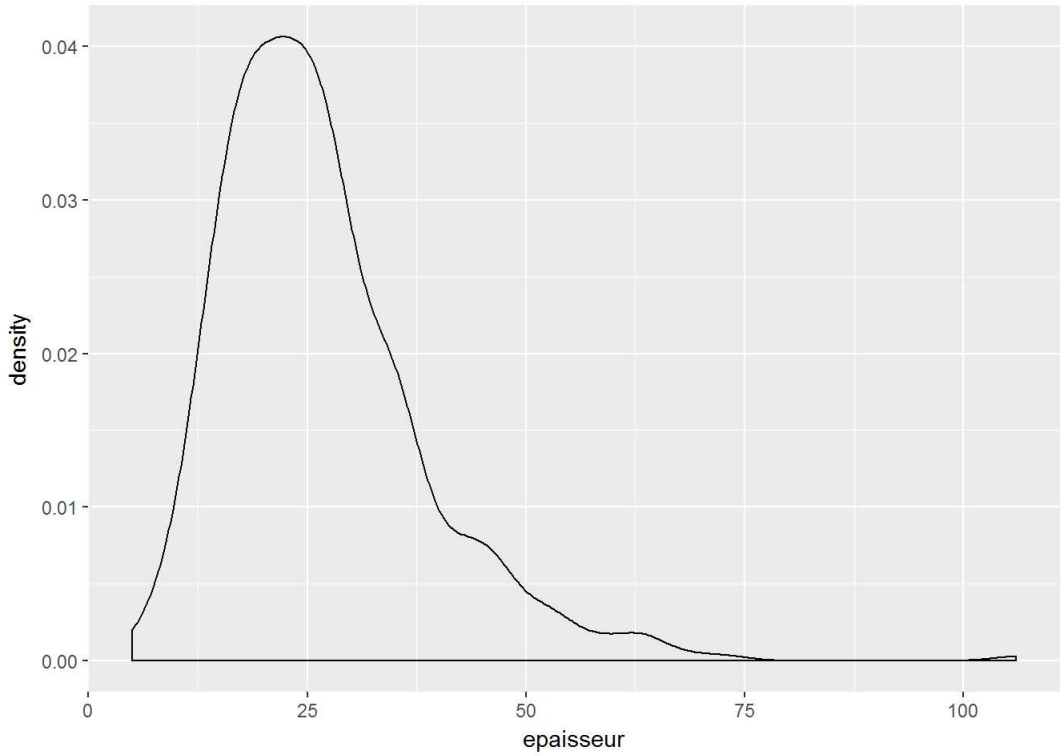


geom_density

`geom_density` permet d'afficher l'estimation de densité d'une variable numérique. Son usage est similaire à celui de `geom_histogram` .

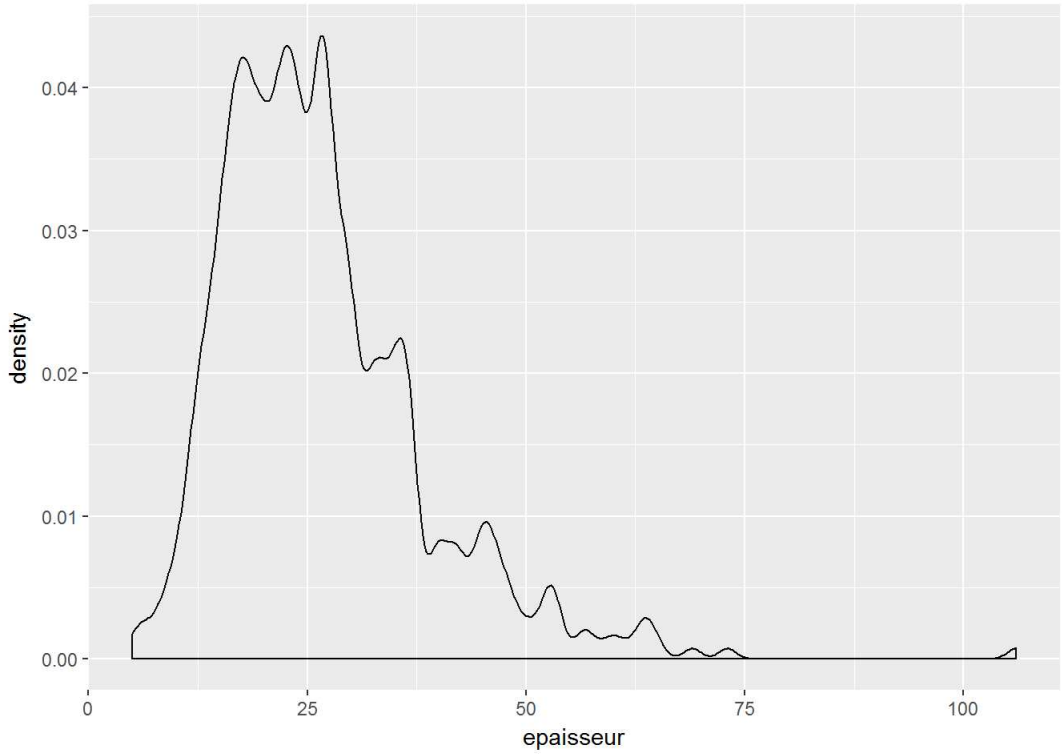
Ainsi, si on veut afficher la densité de la répartition de la part des cadres dans les communes de notre jeu de données :

```
ggplot(co) + geom_density(aes(x = epaisseur))
```



On peut utiliser différents arguments pour ajuster le calcul de l'estimation de densité, parmi lesquels `kernel` et `bw` (voir la page d'adide de la fonction `density` pour plus de détails) :

```
ggplot(co) + geom_density(aes(x = epaisseur), bw = 1)
```



geom_line

`geom_line` trace des lignes connectant les différentes observations entre elles. Il est notamment utilisé pour la représentation de séries temporelles. On passe à `geom_line` deux paramètres : `x` et `y`. Les observations sont alors connectées selon l'ordre des valeurs passées en `x`.

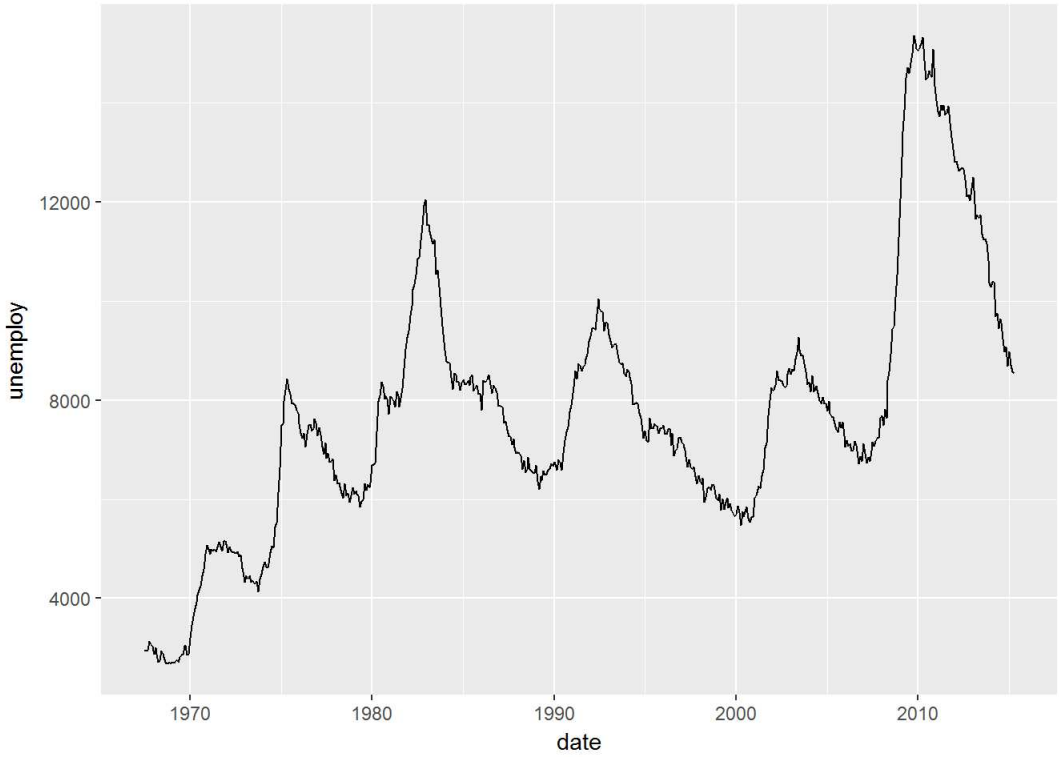
Comme il n'y a pas de données adaptées pour ce type de représentation dans notre jeu de données d'exemple, on va utiliser le jeu de données `economics` inclus dans `ggplot2` et représenter l'évolution du Poids aux États-Unis (variable `unemploy`) dans le temps (variable `date`) :

```
data("economics")
economics
```

date <date>	pce <dbl>	pop <int>	psavert <dbl>	uempmed <dbl>	unemploy <int>
1967-07-01	507.4	198712	12.5	4.5	2944
1967-08-01	510.5	198911	12.5	4.7	2945
1967-09-01	516.3	199113	11.7	4.6	2958
1967-10-01	512.9	199311	12.5	4.9	3143
1967-11-01	518.1	199498	12.5	4.7	3066
1967-12-01	525.8	199657	12.1	4.8	3018
1968-01-01	531.5	199808	11.7	5.1	2878
1968-02-01	534.2	199920	12.2	4.5	3001
1968-03-01	544.9	200056	11.6	4.1	2877
1968-04-01	544.6	200208	12.2	4.6	2709

1-10 of 574 rows Previous **1** 2 3 4 5 6 … 58 Next

```
ggplot(economics) + geom_line(aes(x = date, y = unemploy))
```



Mappages

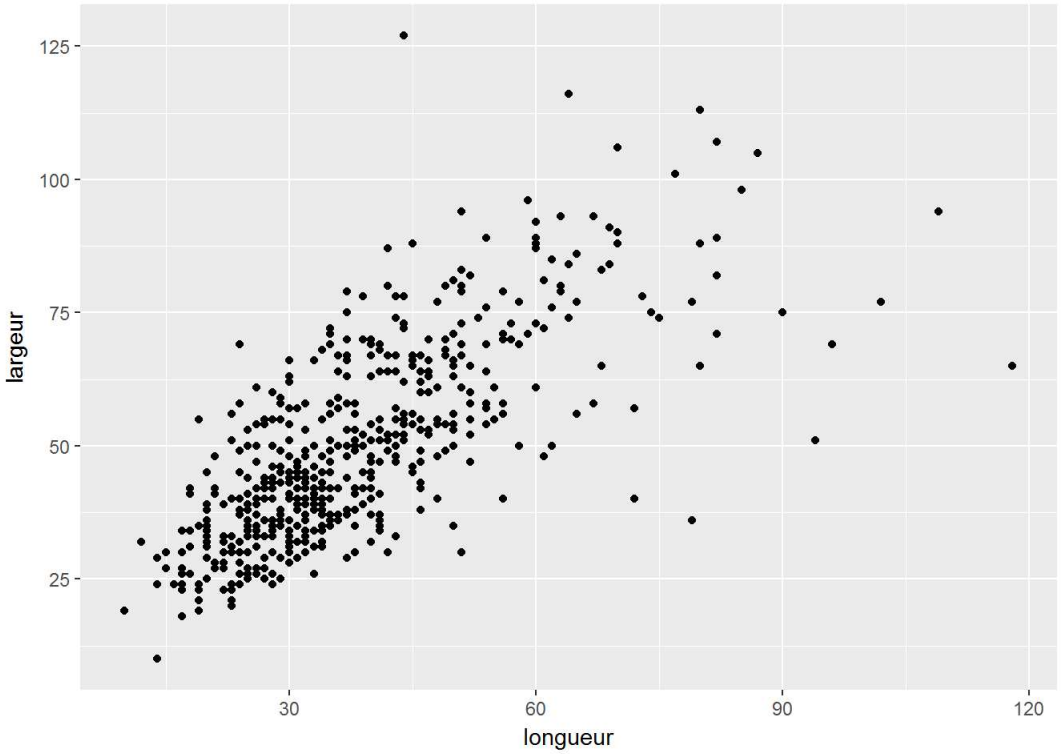
Un *mappage*, dans `ggplot2`, est une mise en relation entre **un attribut graphique** du `geom` (position, couleur, taille…) et **une variable** du tableau de données.

Ces mappages sont passés aux différents `geom` via la fonction `aes()` (abbréviation d'*aesthetic*).

Exemples de mappages

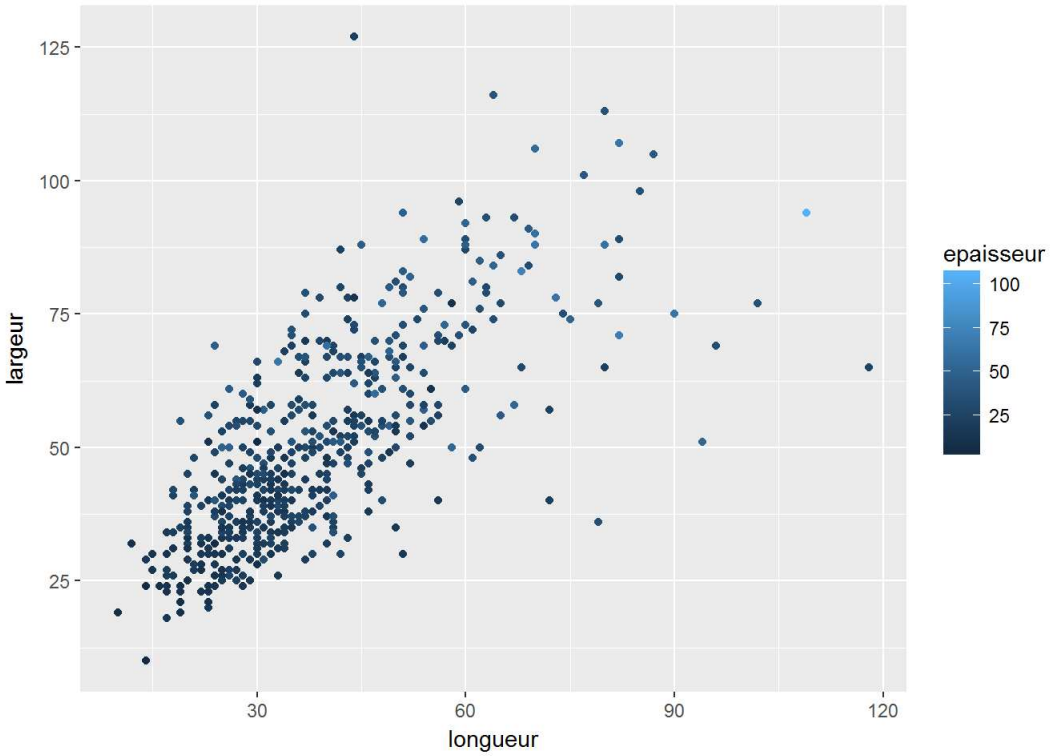
On a déjà vu les mappages `x` et `y` pour un nuage de points. Ceux-ci signifient que la position d'un point donné horizontalement (`x`) et verticalement (`y`) depend de la valeur des variables passées comme arguments `x` et `y` dans `aes()` :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur))
```



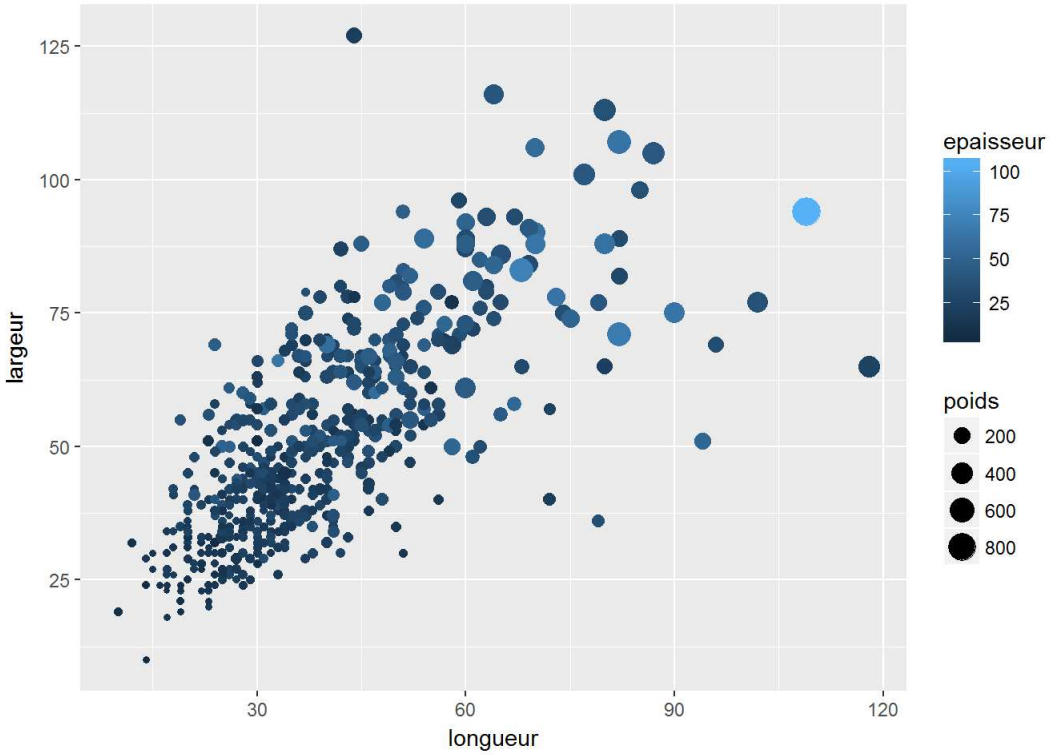
Mais on peut en ajouter d'autres. Par exemple, `color` permet de faire varier la couleur des points automatiquement en fonction des valeurs d'une troisième variable. Par exemple, on peut vouloir colorer les points selon le département de la commune correspondante :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = epaisseur))
```



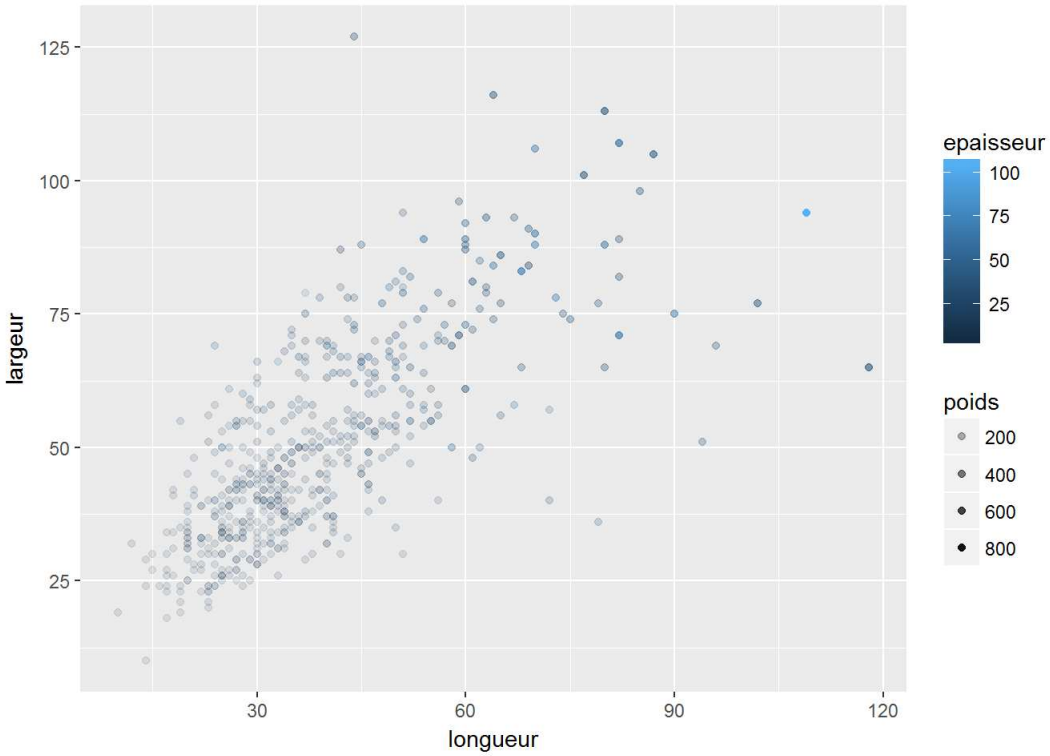
On peut aussi faire varier la taille des points avec `size` . Ici, la taille dépend de la population totale de la commune :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur,
                 color = epaisseur, size = poids))
```



On peut même associer la transparence des points à une variable avec `alpha` :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur,
                 color = epaisseur, alpha = poids))
```

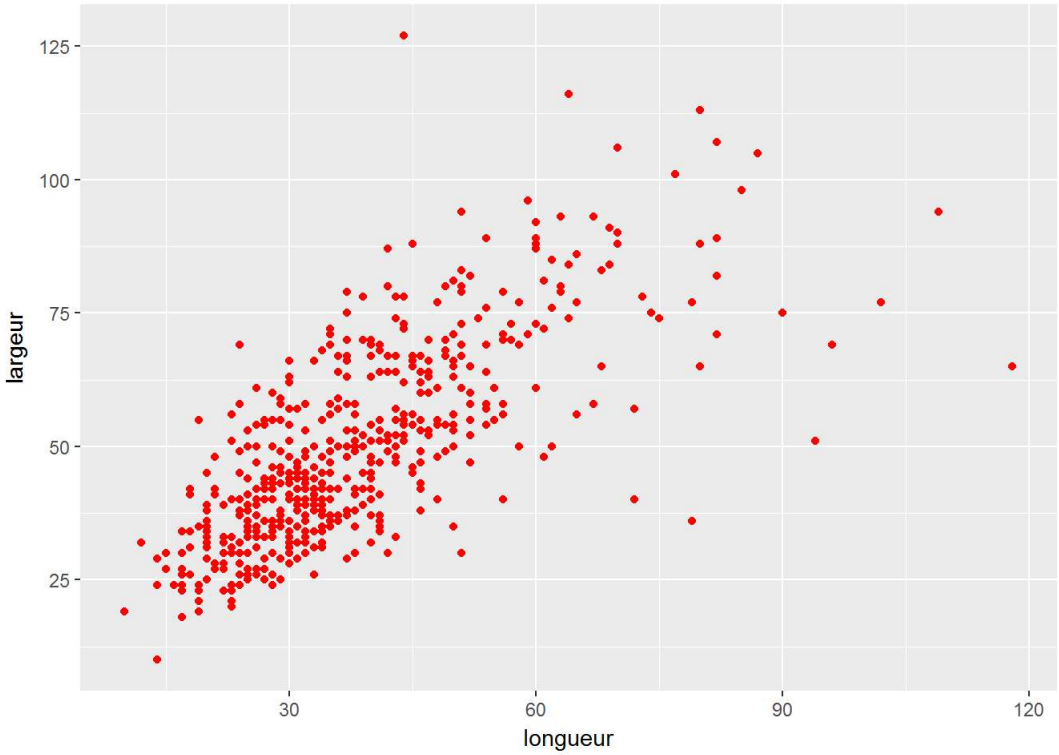


Chaque `geom` possède sa propre liste de mappages.

`aes()` or not `aes()` ?

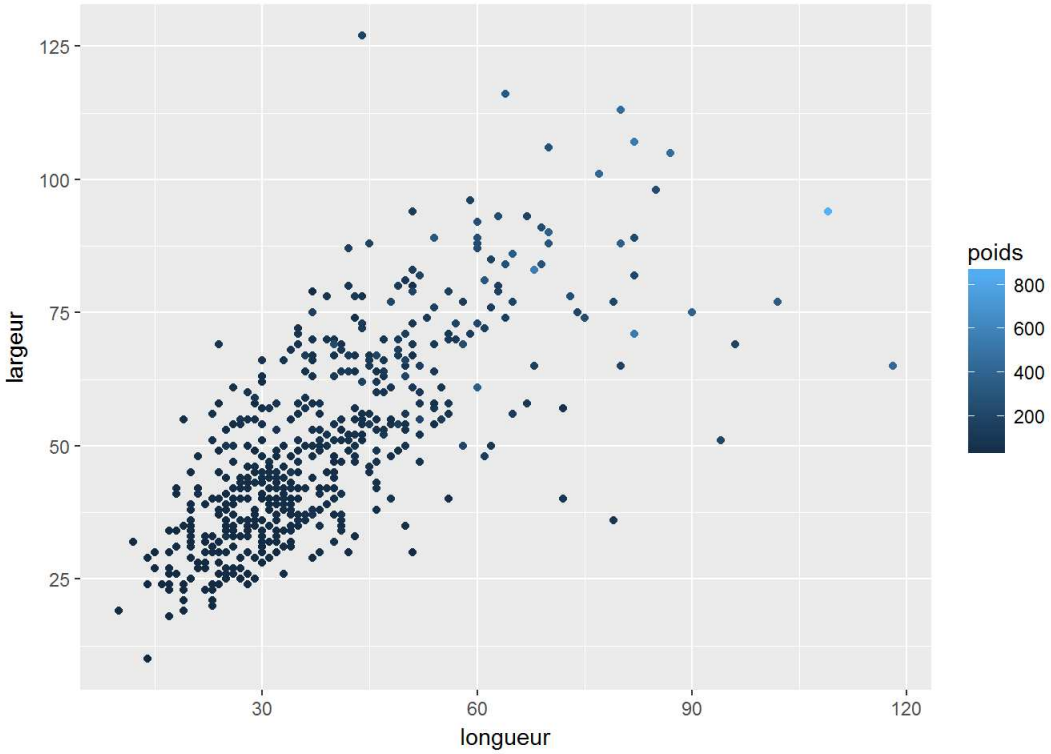
Comme on l'a déjà vu, parfois on souhaite changer un attribut sans le relier à une variable. Par exemple, on veut représenter tous les points en rouge. Dans ce cas on utilise toujours l'attribut `color` , mais comme il ne s'agit pas d'un mappage, on le définit **à l'extérieur** de la fonction `aes()` :

```
ggplot(co) + geom_point(aes(x = longueur, y = largeur), color = "red")
```



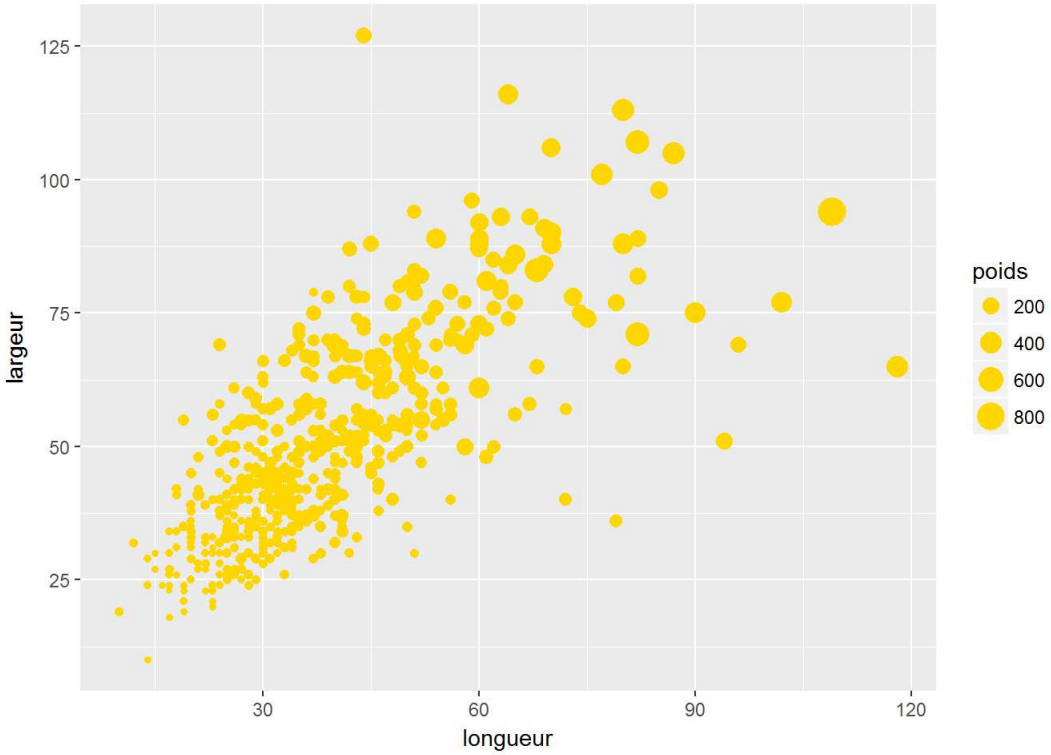
Par contre, si on veut faire varier la couleur en fonction des valeurs prises par une variable, on réalise un mappage, et on doit donc placo l'attribut `color` **à l'intérieur** de `aes()` .

```
ggplot(co) + geom_point(aes(x = longueur, y = largeur, color = poids))
```



On peut évidemment mélanger attributs liés à une variable (mappage, donc dans `aes()`) et attributs constants (donc à l'extérieur). Dans l'exemple suivant, la taille varie en fonction de la variable `pop_tot`, mais la couleur est constante pour tous les points :

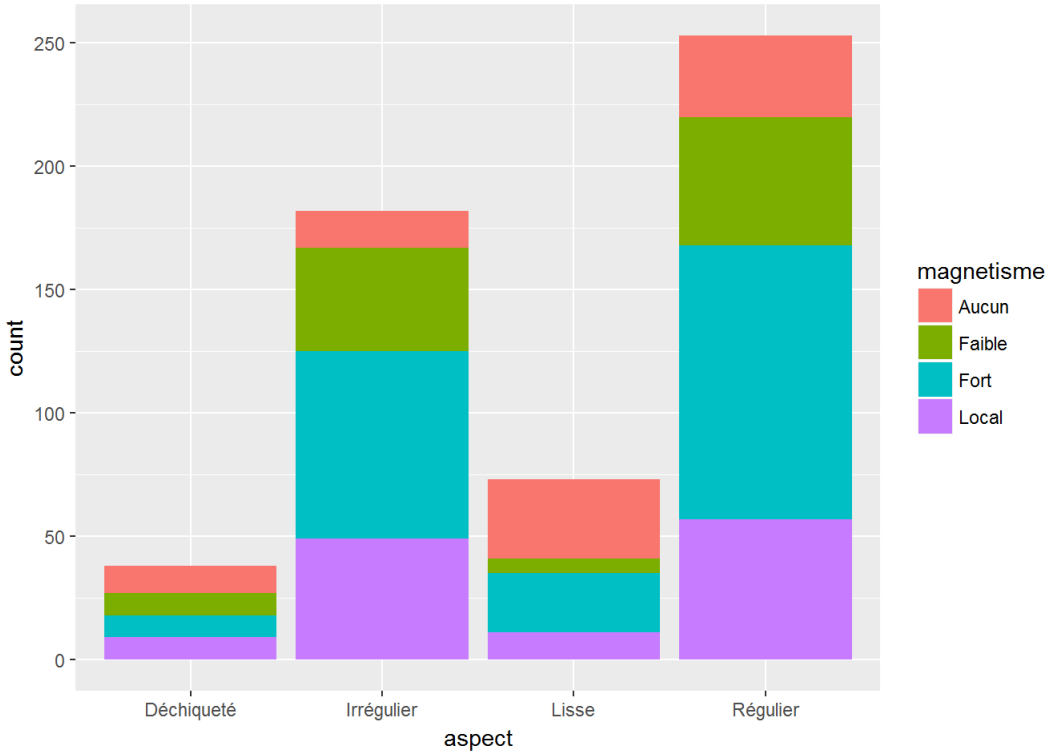
```
ggplot(co) + geom_point(aes(x = longueur, y = largeur, size = poids), color = "gold")
```



La règle est donc simple mais très importante : Si on établit un lien entre les valeurs d'une variable et un attribut graphique, on définit un mappage, et on le déclare dans `aes()`. Sinon, on modifie l'attribut de la même manière pour tous les points, et on le définit en-dehors de la fonction `aes()`.
`` ### geom_bar et position`

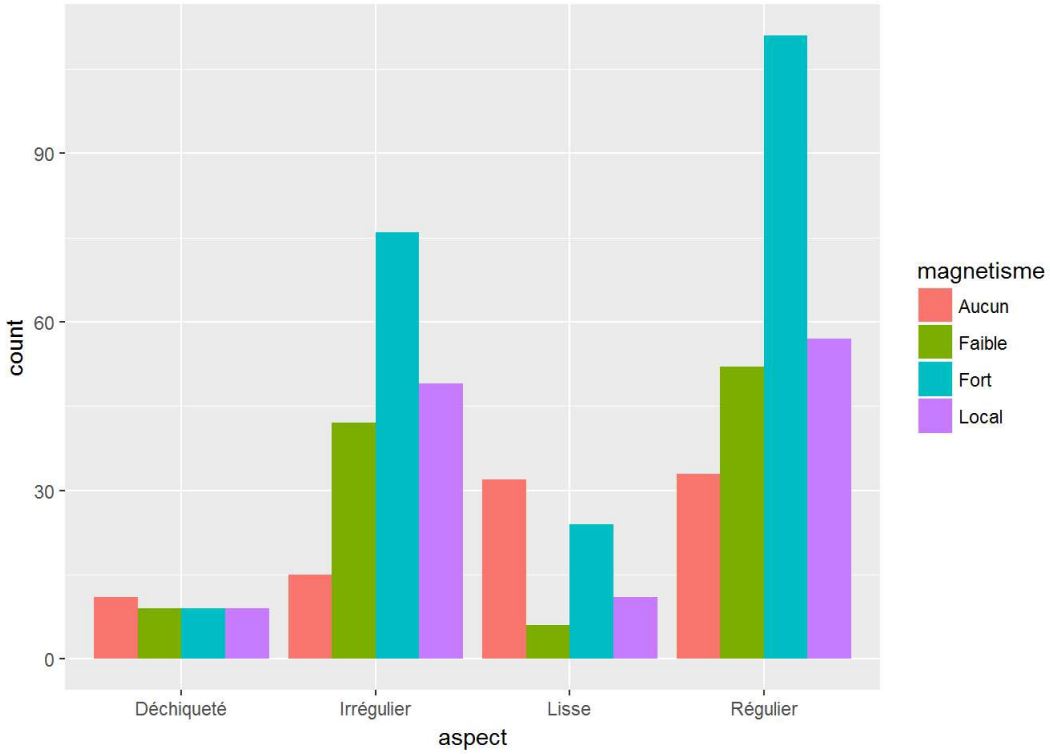
Un des mappages possibles de `geom_bar` est l'attribut `fill`, qui permet de traco des barres de couleur différentes selon les modalités d'une variable :

```
ggplot(co) + geom_bar(aes(x = aspect, fill = magnetisme))
```



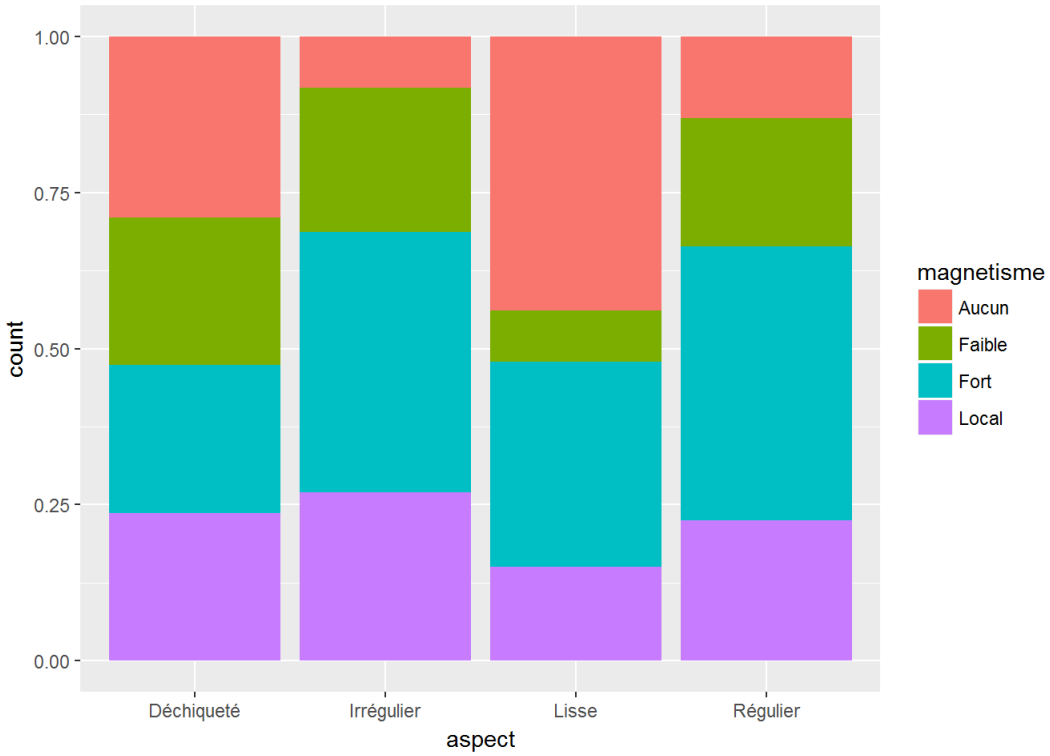
L'attribut `position` de `geom_bar` permet d'indiquer comment les différentes barres doivent être positionnées. Par défaut on a `position = "stack"` et elles sont donc "empilées". Mais on peut préciser `position = "dodge"` pour les mettre côte à côte :

```
ggplot(co) + geom_bar(aes(x = aspect, fill = magnetisme), position = "dodge")
```



Ou encore `position = "fill"` pour représenter non plus des effectifs, mais des proportions :

```
ggplot(co) + geom_bar(aes(x = aspect, fill = magnetisme), position = "fill")
```

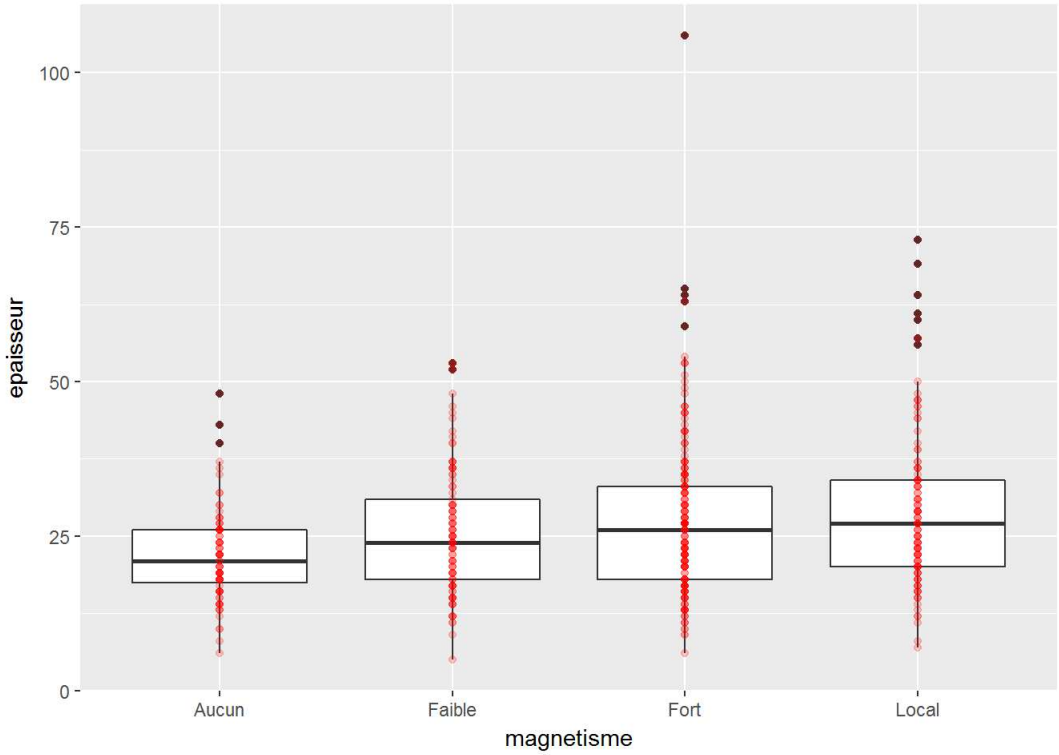


Représentation de plusieurs `geom`

On peut représenter plusieurs `geom` simultanément sur un même graphique, il suffit de les ajouter à tour de rôle avec l'opérateur `+` .

Par exemple, on peut supecooser la position des points au-dessus d'un graphique en boîtes à moustaches. On va pour cela ajouter un `geom_point` après avoir ajouté notre `geom_boxplot` :

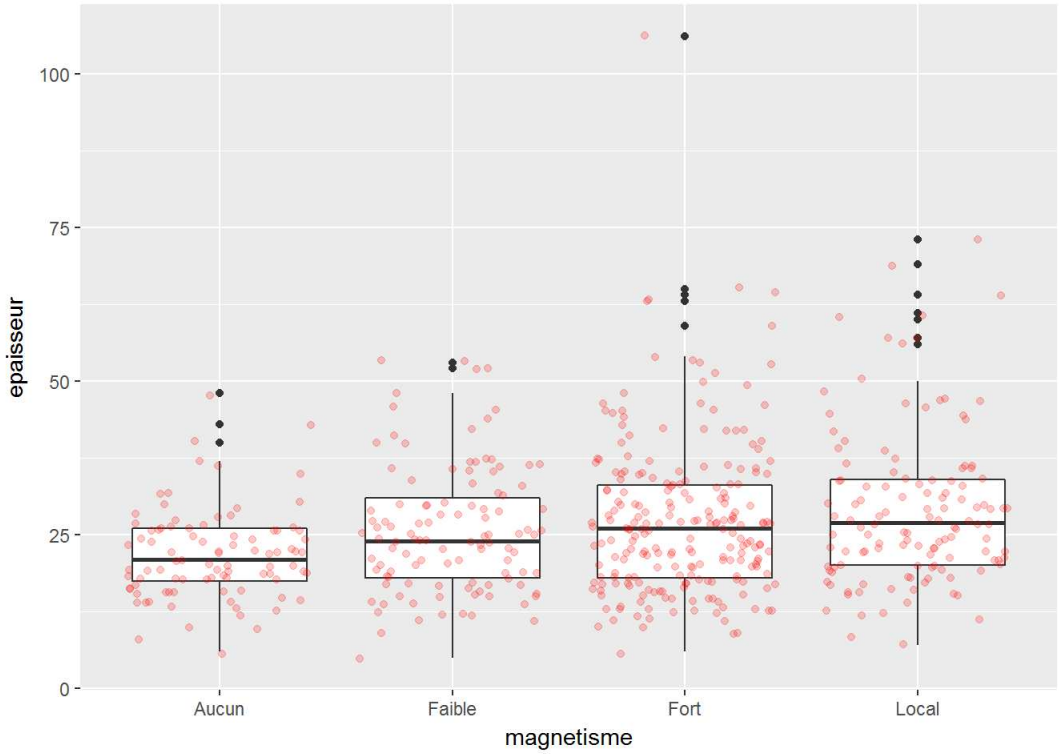
```
ggplot(co) +
  geom_boxplot(aes(x = magnetisme, y = epaisseur)) +
  geom_point(aes(x = magnetisme, y = epaisseur), col = "red", alpha = 0.2)
```



Quand une commande `ggplot2` devient longue, il peut être plus lisible de la répartir sur plusieurs lignes. Dans ce cas, il faut penser à placo l'opérateur `+` en fin de ligne, afin que R comprenne que la commande n'est pas complète et prenne en compte la suite.

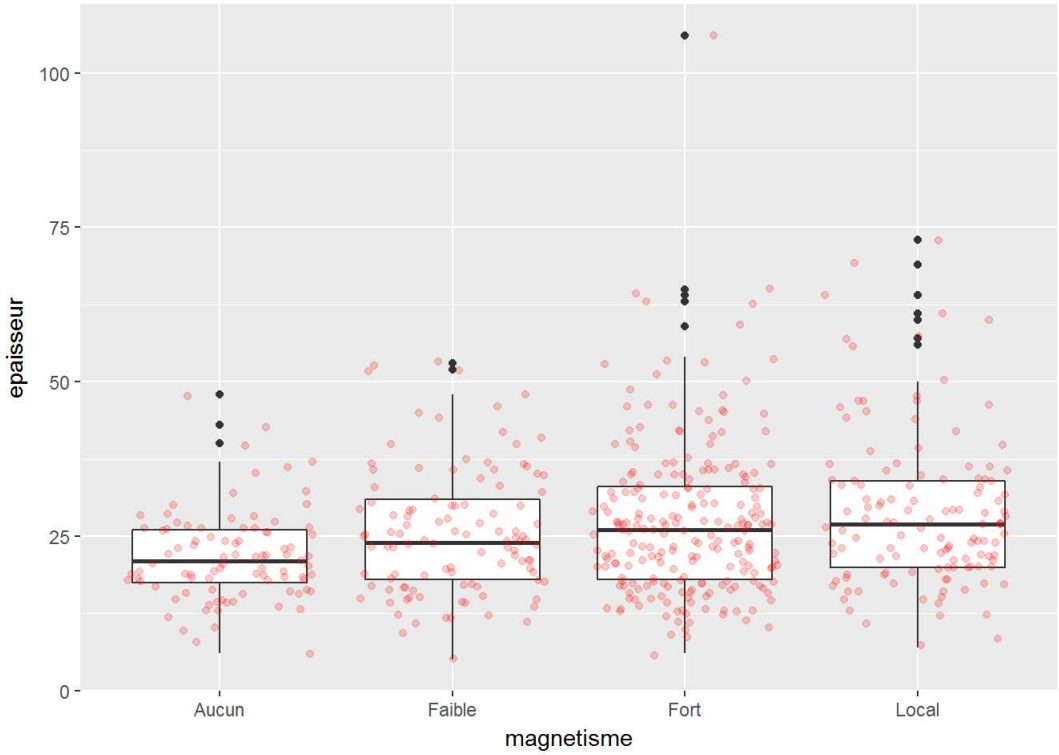
Pour un résultat un peu plus lisible, on peut remplaco `geom_point` par `geom_jitter` , qui disperse les points horizontalement et facilite leur visualisation :

```
ggplot(co) +
  geom_boxplot(aes(x = magnetisme, y = epaisseur)) +
  geom_jitter(aes(x = magnetisme, y = epaisseur), col = "red", alpha = 0.2)
```



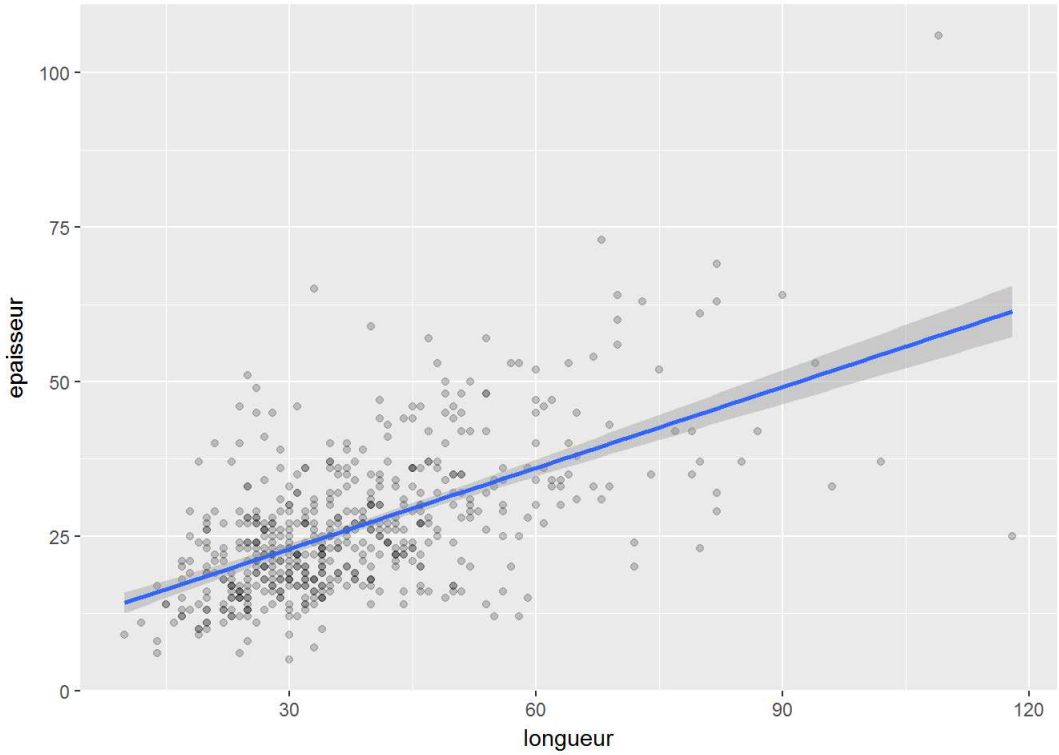
Dans ce cas, plutôt que de déclarer les mappages dans chaque `geom` , on peut les déclarer dans l'appel à `ggplot()` . Ils seront automatiquement "hérités" par les `geom` ajoutés (sauf s'ils redéfinissent les mêmes mappages) :

```
ggplot(co, aes(x = magnetisme, y = epaisseur)) +
  geom_boxplot() +
  geom_jitter(color = "red", alpha = 0.2)
```



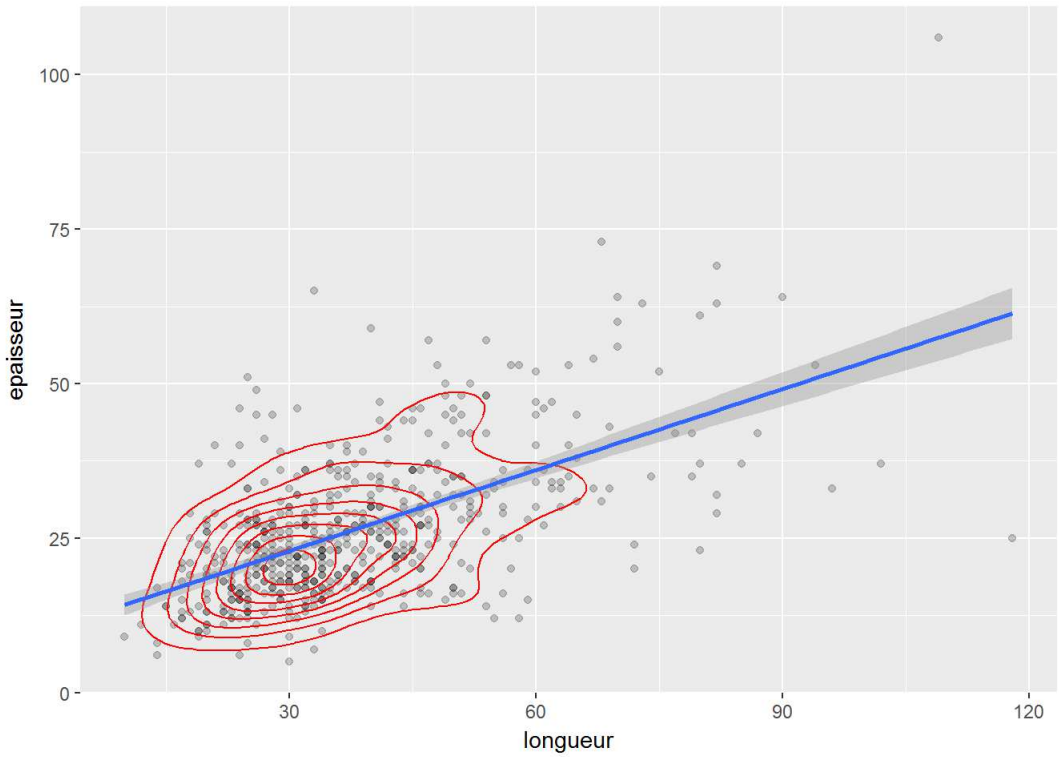
Autre exemple, on peut vouloir ajouter à un nuage de points une ligne de régression linéaire à l'aide de `geom_smooth` :

```
ggplot(co, aes(x = longueur, y = epaisseur)) +
  geom_point(alpha = 0.2) +
  geom_smooth(method = "lm")
```



Et on peut même supecooser une troisième visualisation de la répartition des points dans l'espace avec `geom_density2d` :

```
ggplot(co, aes(x = longueur, y = epaisseur)) +
  geom_point(alpha = 0.2) +
  geom_density2d(color = "red") +
  geom_smooth(method = "lm")
```



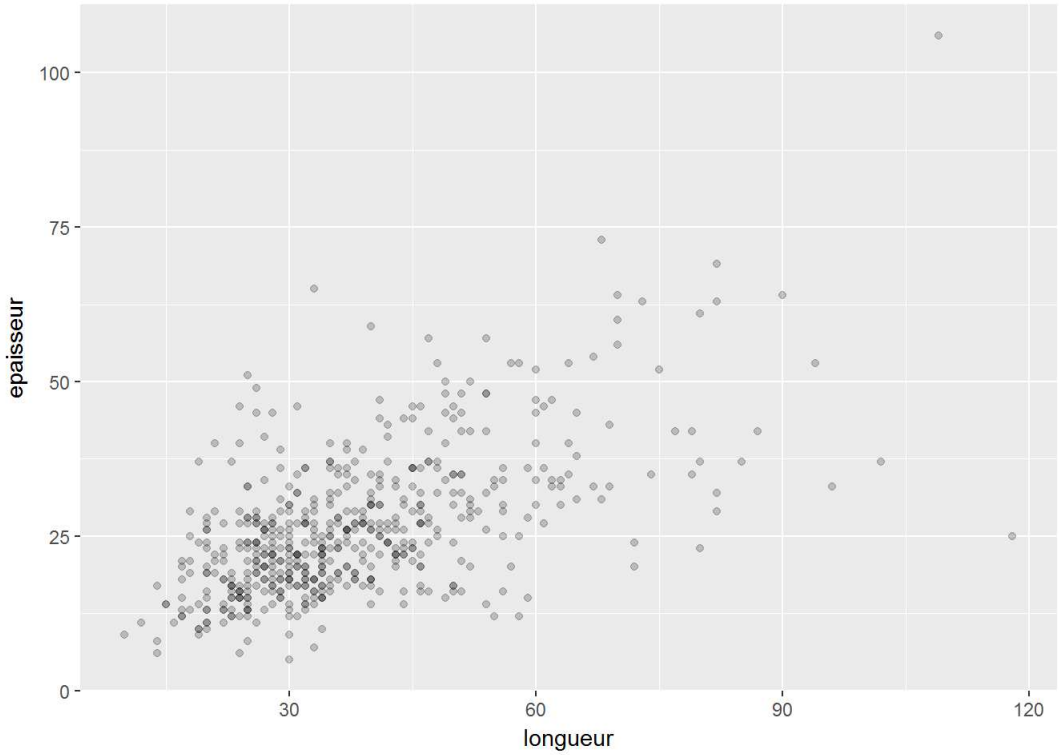
Plusieurs sources de données

On peut aussi associer à différents `geom` des sources de données différentes. Supposons qu'on souhaite afficher sur un nuage de points les id des scories d'aspect lisse. On peut commenco par créer un tableau de données avec seulement ces scories `filter` :

```
co50 <- filter(co, aspect == "Lisse")
```

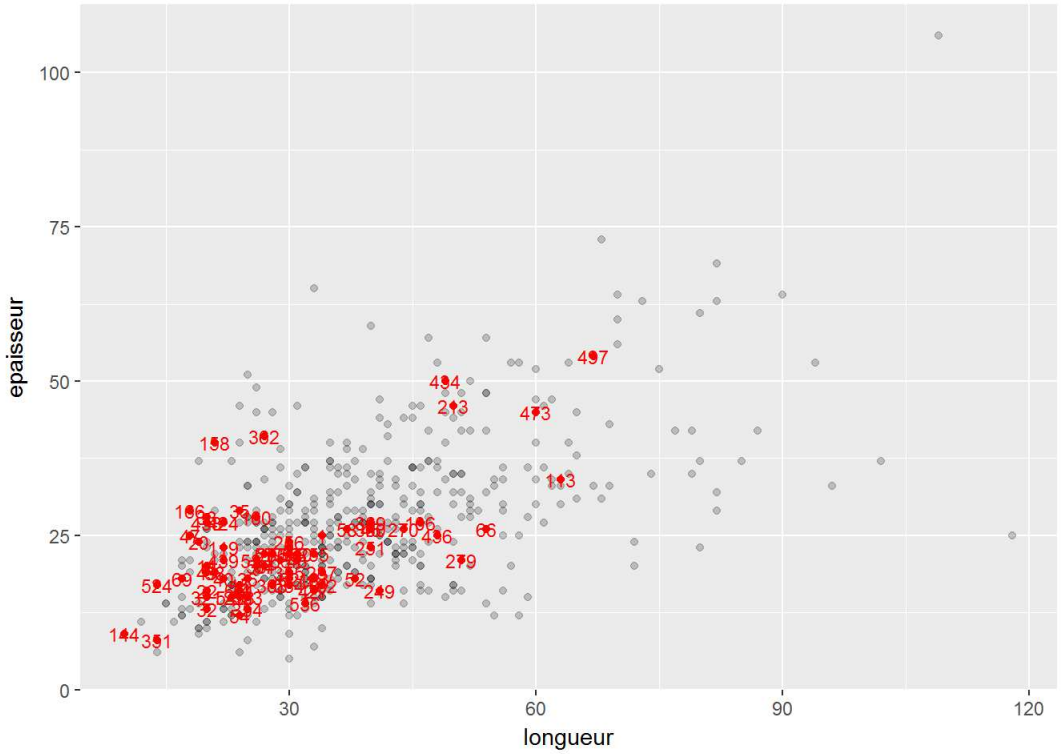
On fait ensuite le nuage de points comme précédemment :

```
ggplot(data = co, aes(x = longueur, y = epaisseur)) +
  geom_point(alpha = 0.2)
```



Pour supecooser les noms des cories d'aspect lisse, on peut ajouter un `geom_text` , mais en spécifiant que les données proviennent du nouveau tableau `com50` et non de notre tableau initial `co` . On le fait en passant un argument `data` spécifique à `geom_text` :

```
ggplot(data = co, aes(x = longueur, y = epaisseur)) +
  geom_point(alpha = 0.2) +
  geom_text(data = co50, aes(label = id),color = "red", size = 3) +
  geom_point(data = co50, color ="red")
```



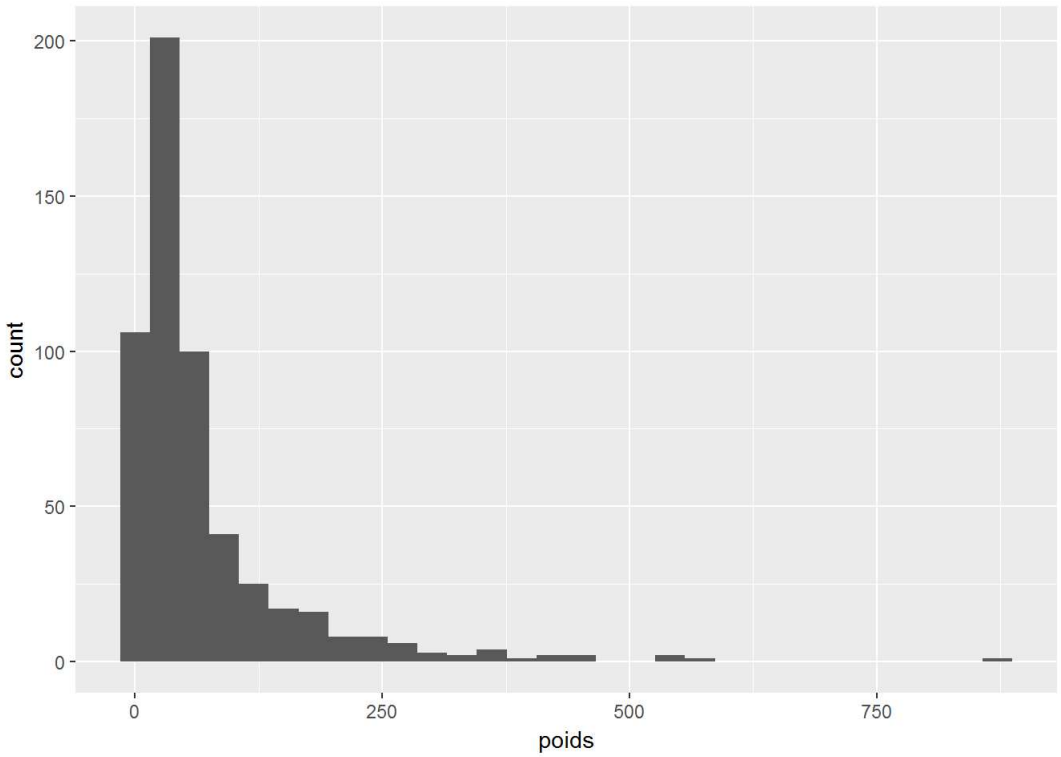
Ainsi, on a un graphique avec deux `geom` supecoosés, mais dont les données proviennent de deux tableaux différents.

Faceting

Le *faceting* permet d'effectuer plusieurs fois le même graphique selon les valeurs d'une ou plusieurs variables qualitatives.

Par exemple, on peut représenter l'histogramme des épaisseurs de scories de culot :

```
ggplot(data = co) +
  geom_histogram(aes(x = poids))
```

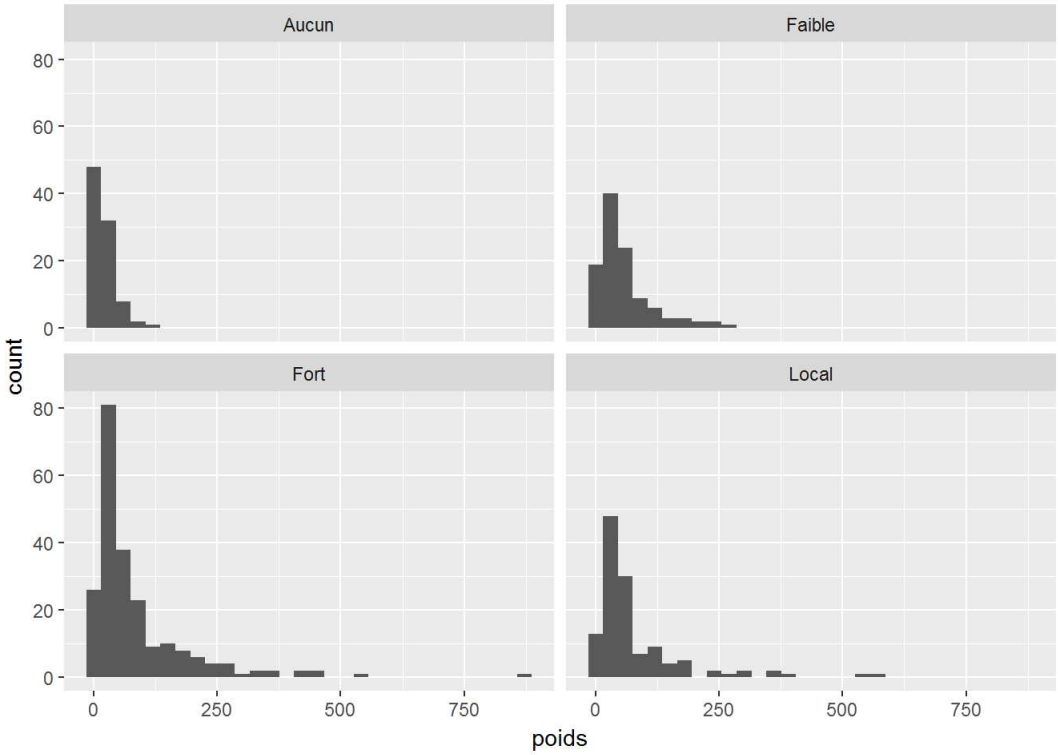


On peut vouloir comparer la répartition de la part des cadres selon le département, et donc faire un histogramme pour chacun de ces départements. On peut dans ce cas utiliser `facet_wrap` ou `facet_grid` .

Les deux fonctions prennent en paramètre une formule de la forme `~variable` , où `variable` est le nom de la variable en fonction de laquelle on souhaite faire les différents graphiques.

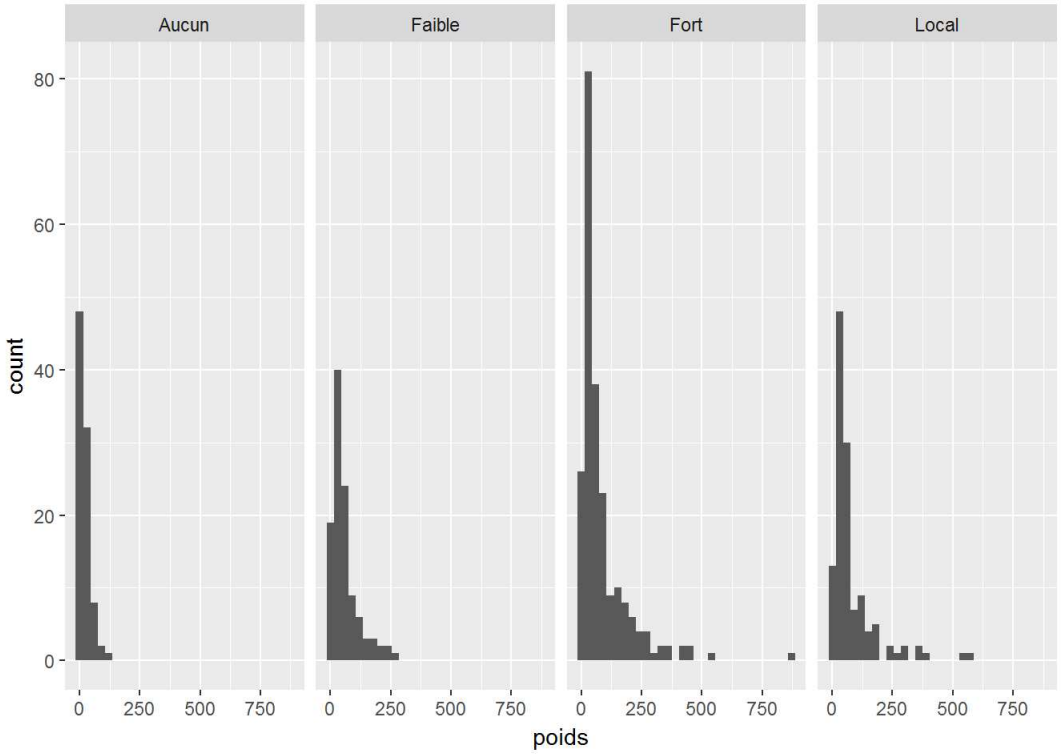
Avec `facet_wrap` , les différents graphiques sont affichés les uns à côté des autres et répartis automatiquement dans la page. Par exemple :

```
ggplot(data = co) +
  geom_histogram(aes(x = poids)) +
  facet_wrap(~magnetisme)
```



Pour `facet_grid` , les graphiques sont disposés selon une grille. La formule est alors de la forme `variable en ligne ~ variable en colonne` . Si on n'a pas de variable dans l'une des deux dimensions, on met un point (`.`) :

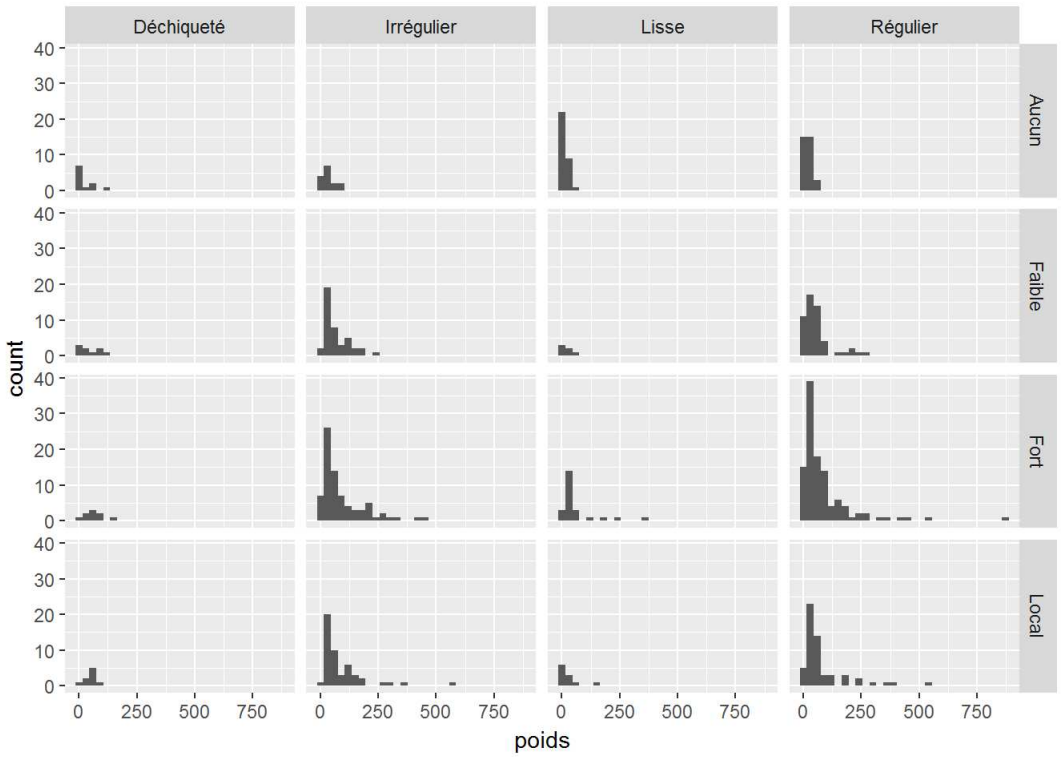
```
ggplot(data = co) +
  geom_histogram(aes(x = poids)) +
  facet_grid(.~magnetisme)
```



Un des intérêts du faceting dans `ggplot2` est que tous les graphiques générés ont les mêmes échelles, ce qui permet une comparaison directe.

Enfin, notons qu'on peut même faire du faceting sur plusieurs variables à la fois. Ici on fait des histogrammes de la répartition de la part des cadres pour chaque croisement des variables `departement` et `pop_cl` :

```
ggplot(data = co) +
  geom_histogram(aes(x = poids)) +
  facet_grid(magnetisme~aspect)
```



Scales

On a vu qu'avec `ggplot2` on définissait des mappages entre des attributs graphiques (position, taille, couleur, etc.) et des variables d'un tableau de données. Ces mappages sont définis, pour chaque `geom`, via la fonction `aes()`.

Les *scales* dans `ggplot2` permettent de modifier la manière dont un attribut graphique va être relié aux valeurs d'une variable, et dont la légende correspondante va être affichée. Par exemple, pour l'attribut `color`, on pourra définir la palette de couleur utilisée. Pour `size`, les tailles minimales et maximales, etc.

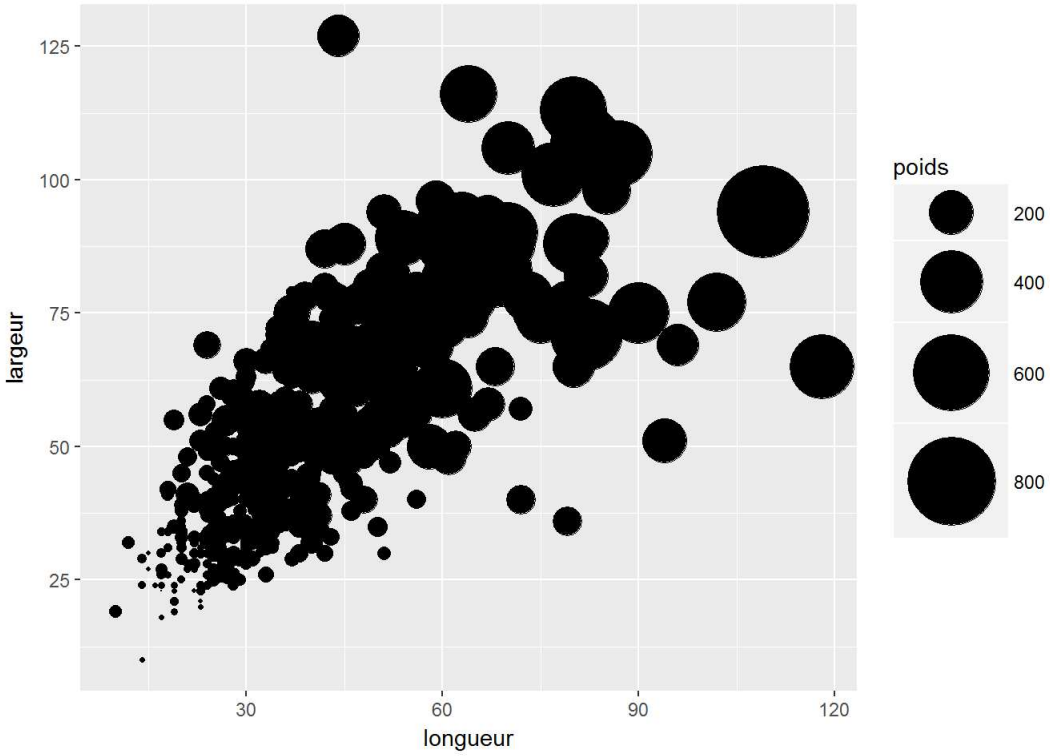
Pour modifier une *scale* existante, on ajoute un nouvel élément à notre objet `ggplot2` avec l'opérateur `+`. Cet élément prend la forme `scale_<attribut>_<type>`.

Voyons tout de suite quelques exemples.

`scale_size`

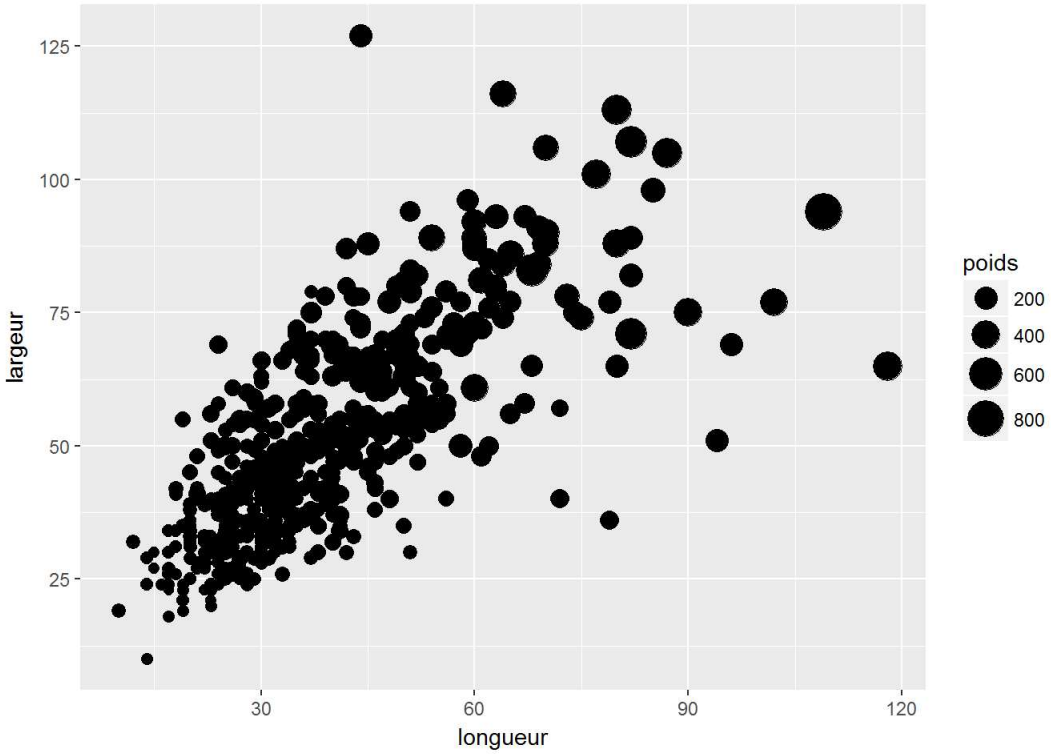
Si on souhaite modifier les tailles minimales et maximales des objets quand on a effectué un mappage de type `size`, on peut utiliser la fonction `scale_size` et son argument `range` :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, size = poids)) +
  scale_size(range = c(0,20))
```



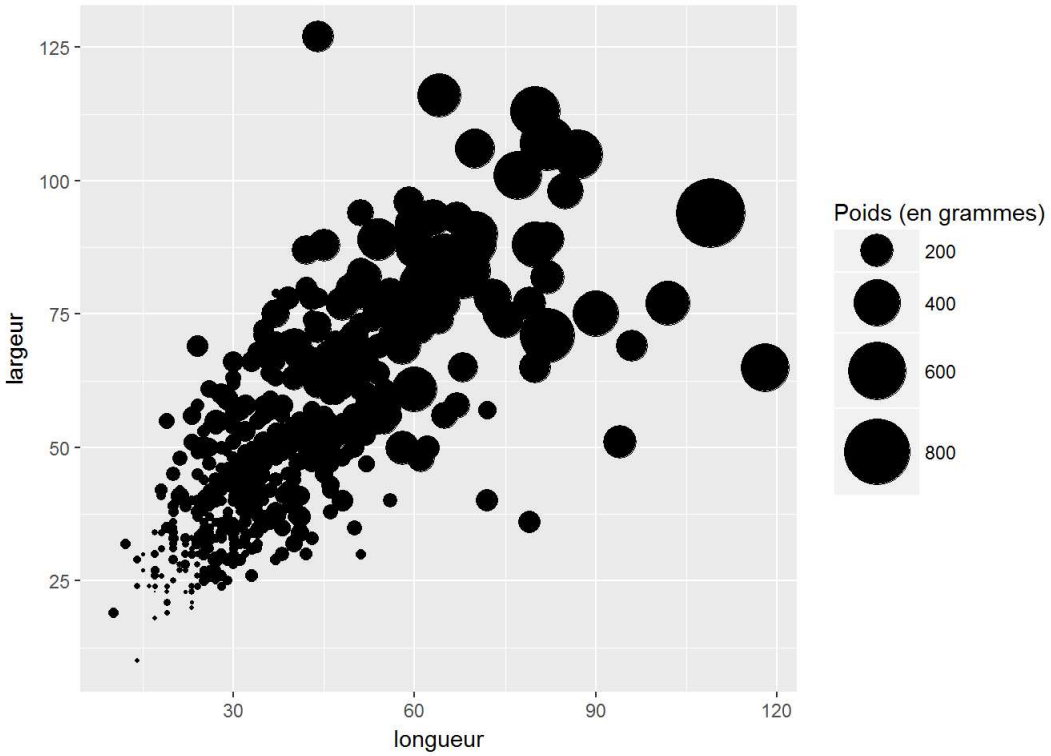
À comparer par exemple à :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, size = poids)) +
  scale_size(range = c(2,8))
```



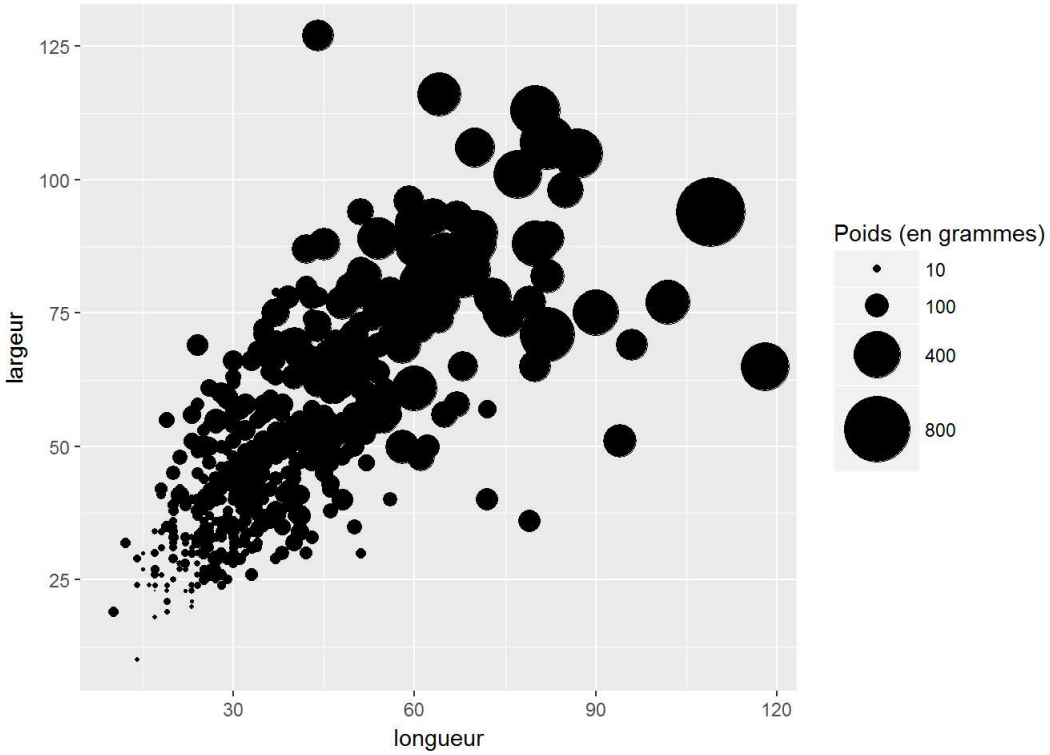
On peut ajouter d'autres paramètres à `scale_size` . Le premier argument est toujours le titre donné à la légende :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, size = poids)) +
  scale_size("Poids (en grammes)", range = c(0,15))
```



On peut aussi définir manuellement les éléments de légende représentés :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, size = poids)) +
  scale_size("Poids (en grammes)", range = c(0,15), breaks = c(10,100,400,800))
```



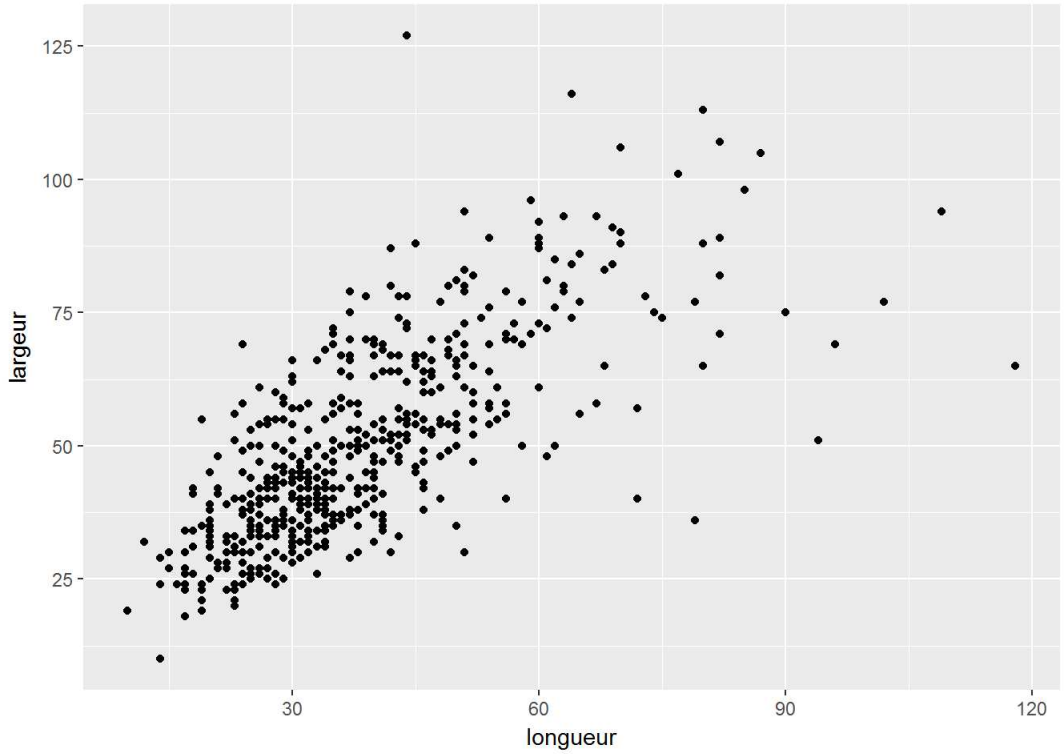
scale_x, scale_y

Les *scales* `scale_x` et `scale_y` modifient les axes `x` et `y` du graphique.

`scale_x_continuous` et `scale_y_continuous` s'appliquent lorsque la variable `x` ou `y` est numérique (quantitative).

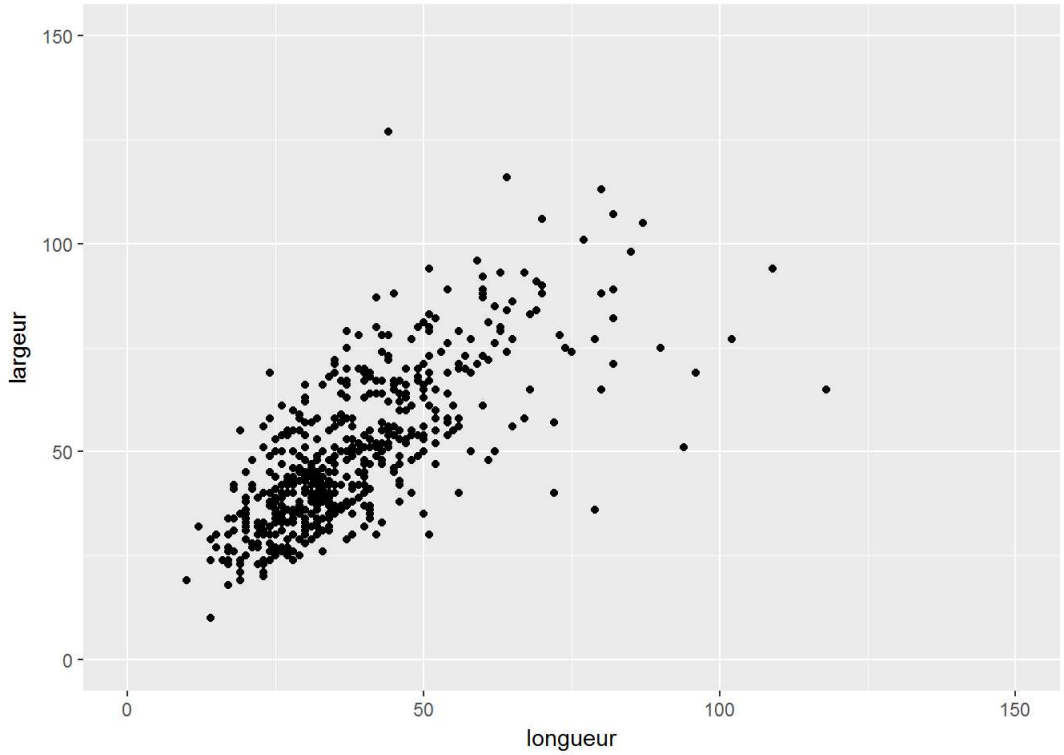
C'est le cas de notre nuage de points croisant longueur et largeurs des cories de culot :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur))
```



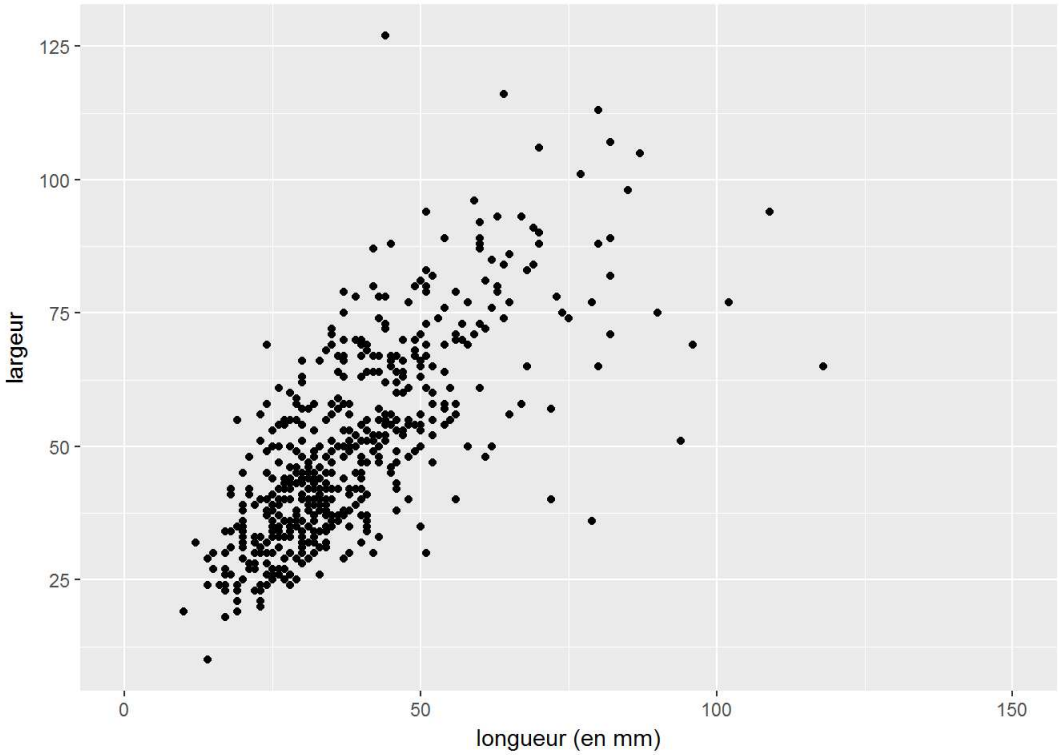
On peut vouloir forco les axes `x` et `y` à s'étendre des valeurs 0 à 150. On peut le faire en ajoutant un élément `scale_x_continuous` et un élément `scale_y_continuous` , et en utilisant leur argument `limits` :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur)) +
  scale_x_continuous(limits = c(0,150)) +
  scale_y_continuous(limits = c(0,150))
```



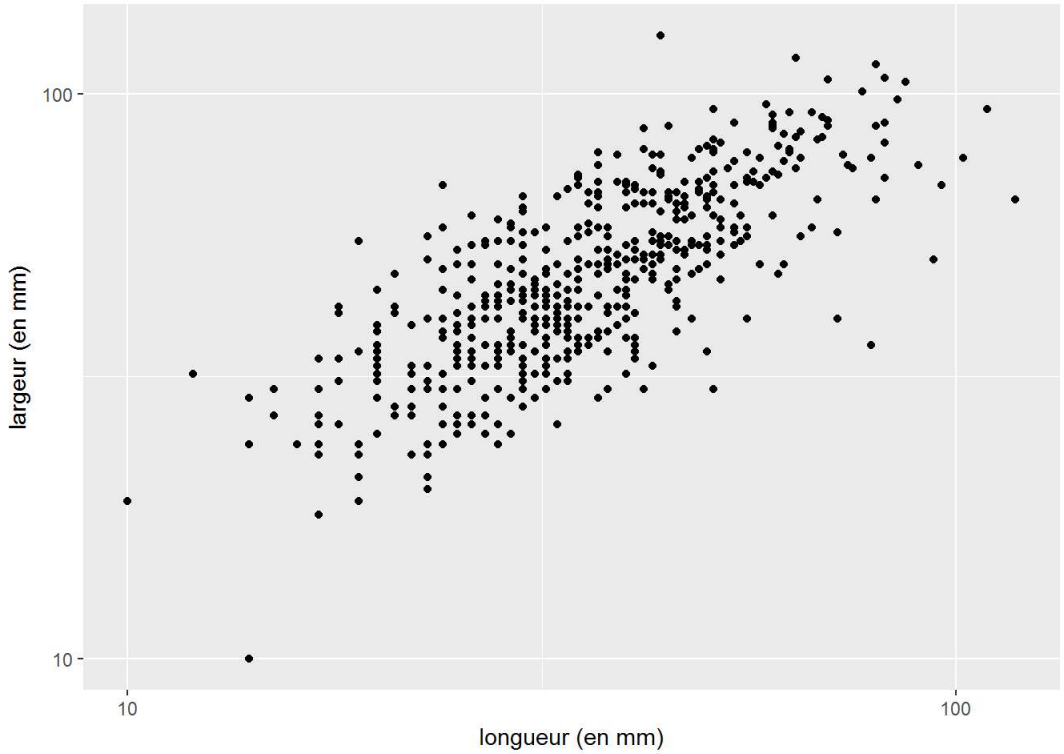
Là aussi, on peut modifier les étiquettes des axes en indiquant une chaîne de caractères en premier argument :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur)) +
  scale_x_continuous("longueur (en mm)", limits = c(0,150))
```



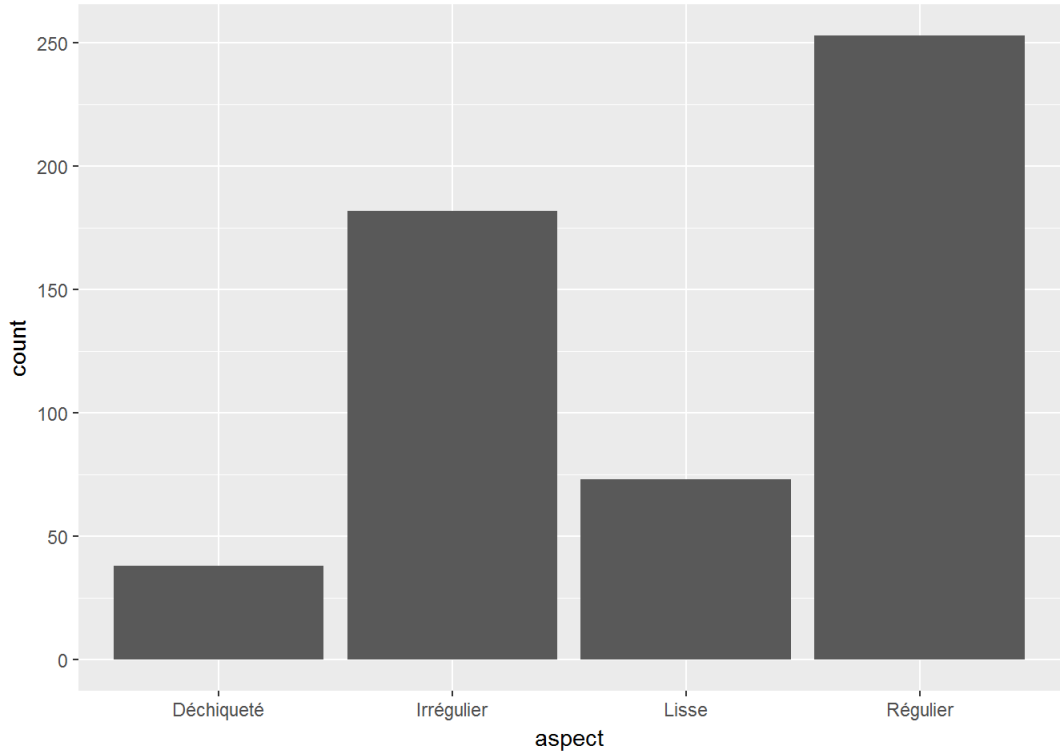
On peut utiliser `scale_x_log10` et `scale_y_log10` pour passer un axe à une échelle logarithmique : (Attention il n'y a pas d'interêt ici pour ce jeu de données..)

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur)) +
  scale_x_log10("longueur (en mm)")+
  scale_y_log10("largeur (en mm)")
```



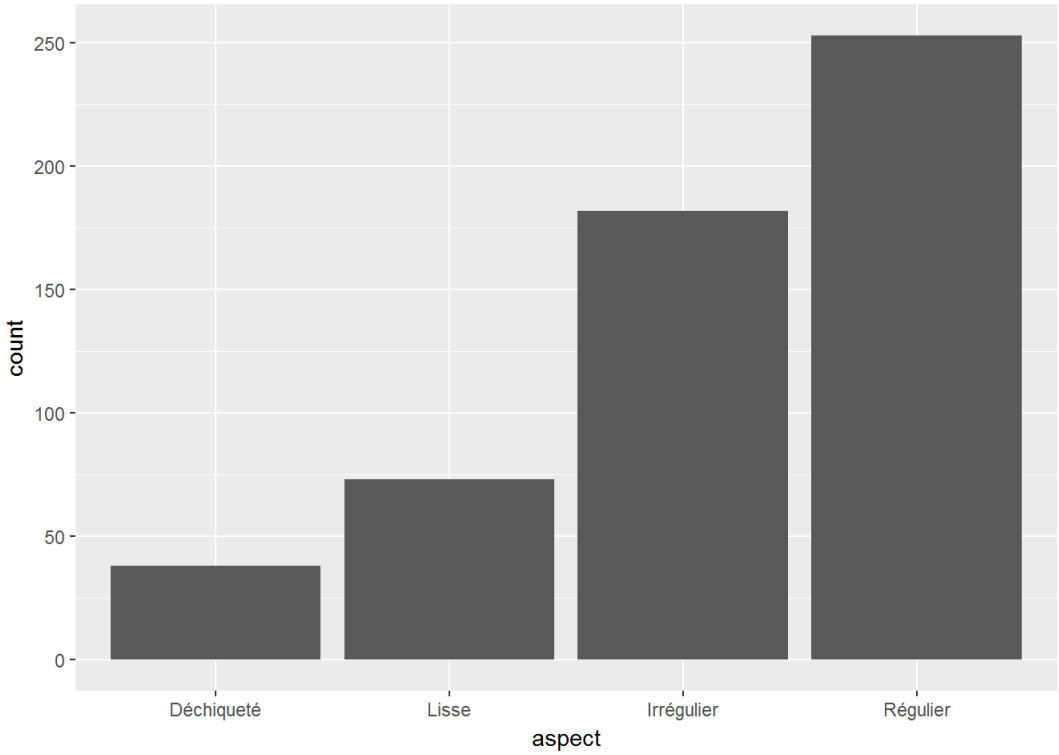
`scale_x_discrete` et `scale_y_discrete` s'appliquent quant à elles lorsque l'axe correspond à une variable discrète (qualitative). C'est le cas de l'axe des `x` dans un diagramme en bâtons :

```
ggplot(co) +
  geom_bar(aes(x = aspect)) +
  scale_x_discrete("aspect")
```



L'argument `limits` de `scale_x_discrete` permet d'indiquer quelles valeurs sont affichées et dans quel ordre.

```
ggplot(co) +
  geom_bar(aes(x = aspect)) +
  scale_x_discrete("aspect", limits = c("Déchiqueté", "Lisse", "Irrégulier", "Régulier"))
```



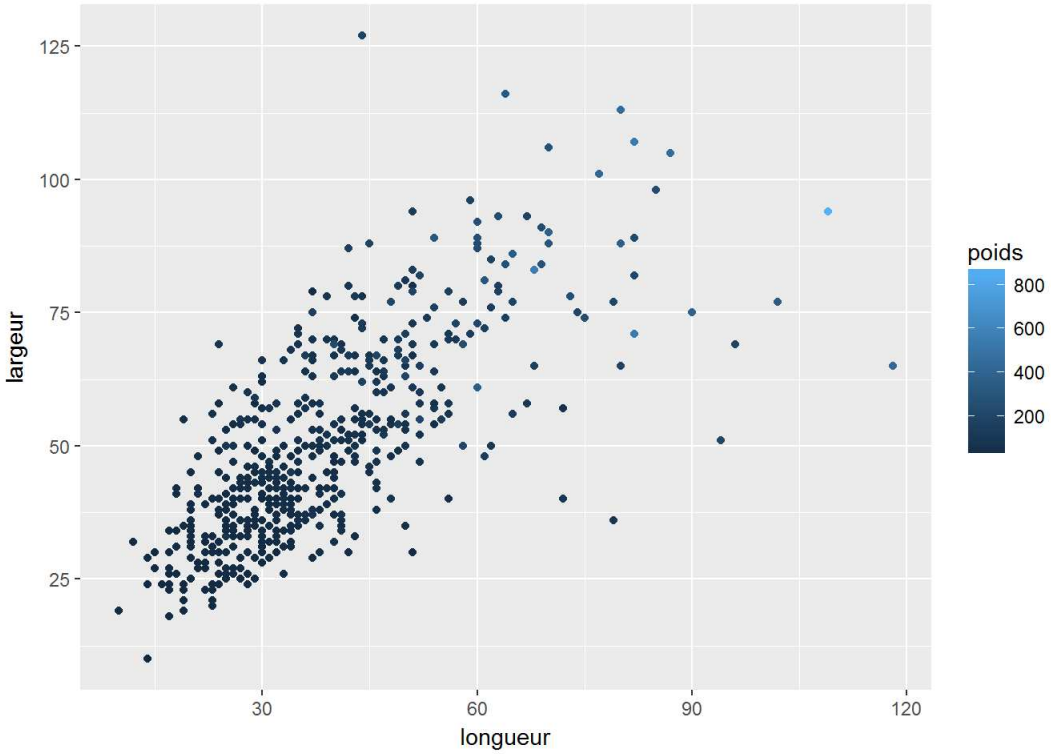
`scale_color`, `scale_fill`

Ces *scales* permettent, entre autre, de modifier les palettes de couleur utilisées pour le dessin (`color`) ou le remplissage (`fill`) des éléments graphiques. Dans ce qui suit, pour chaque fonction `scale_color` présentée il existe une fonction `scale_fill` équivalente et avec en général les mêmes arguments.

Variables quantitatives

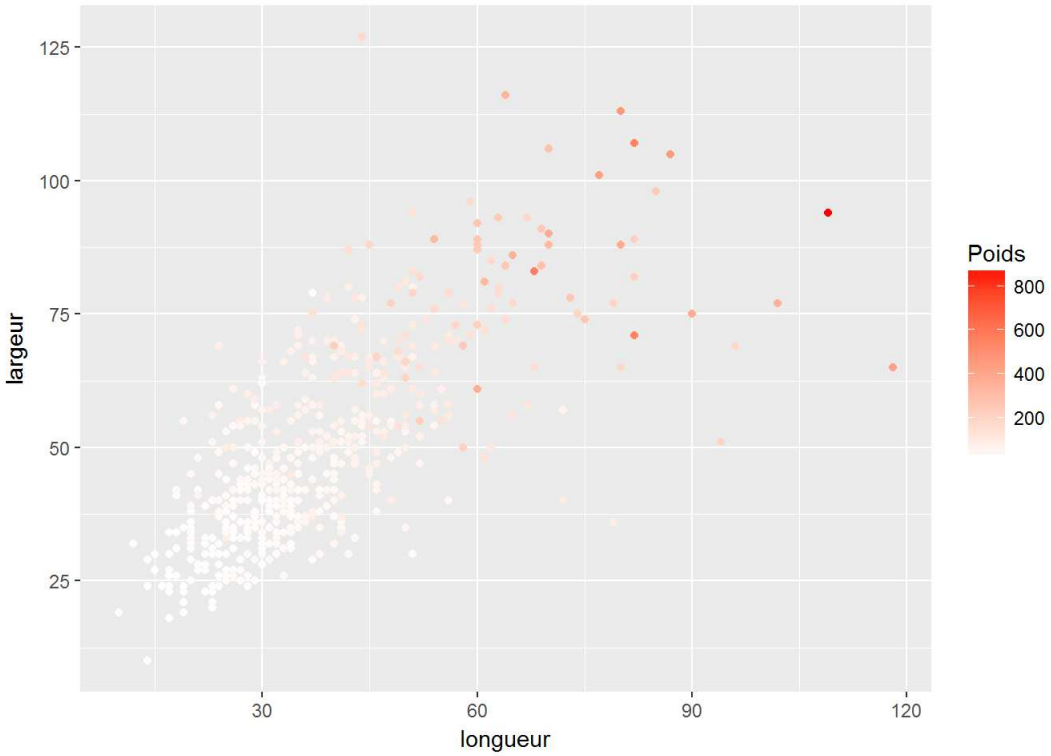
Le graphique suivant colore les points selon la valeur d'une variable numérique quantitative (ici le Poids) :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = poids))
```



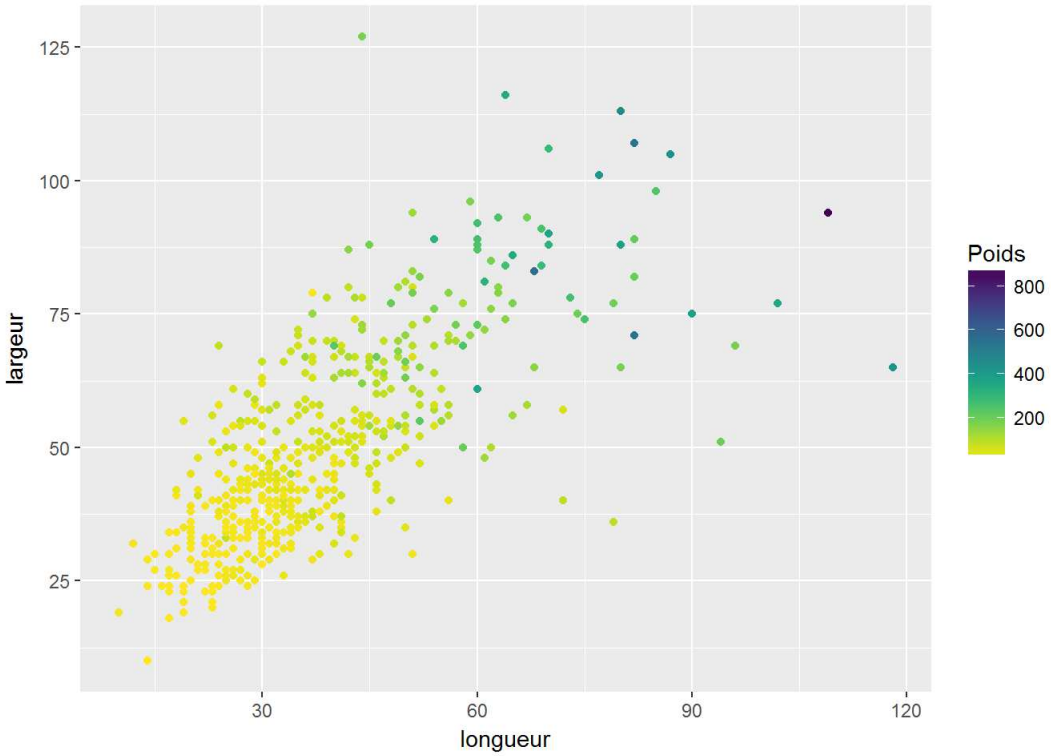
On peut modifier les couleurs utilisées avec les arguments `low` et `high` de la fonction `scale_color_gradient` . Ici on souhaite que la valeur la plus faible soit blanche, et la plus élevée rouge :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = poids)) +
  scale_color_gradient("Poids", low = "white", high = "red")
```



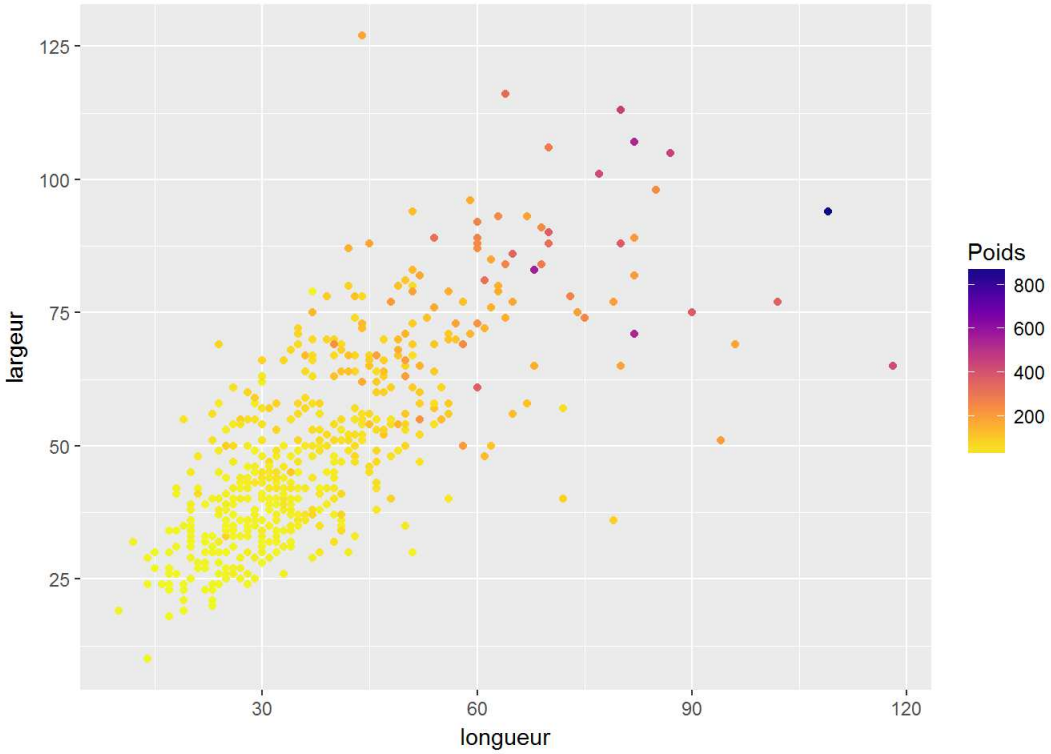
On peut aussi utiliser des palettes prédéfinies. L'une des plus populaires est la palette *viridis*, accessible depuis l'extension du même nom. On l'ajoute en utilisant `scale_color_viridis` : Attention: il faut d'abord télecharger le package "viridis" !

```r
# install.packages("viridis")
library(viridis)
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = poids)) +
  scale_color_viridis("Poids", direction = -1)
```



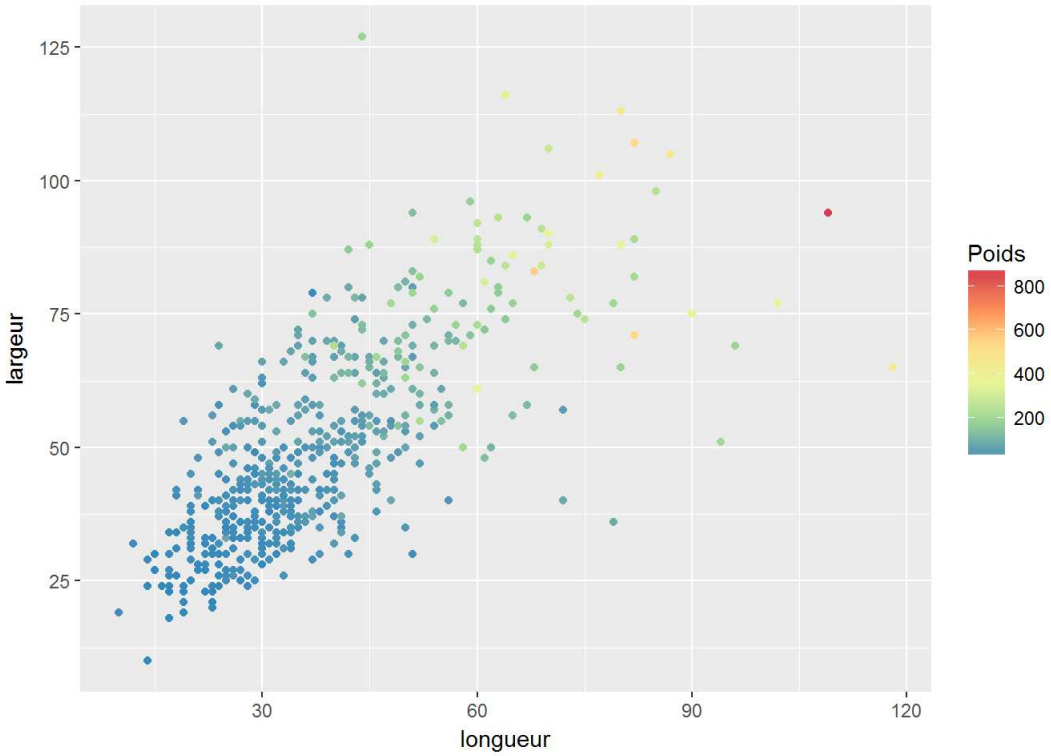
L'extension `viridis` propose également trois autres palettes, *magma*, *inferno* et *plasma*, accessibles via l'argument `option` :

```r
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = poids)) +
  scale_color_viridis("Poids", option = "plasma", direction = -1)
```



On peut aussi utiliser `scale_color_distiller` , qui transforme une des palettes pour variable qualitative de `scale_color_brewer` en palette continue pour variable numérique :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = poids)) +
  scale_color_distiller("Poids", palette = "Spectral", direction = -1)
```

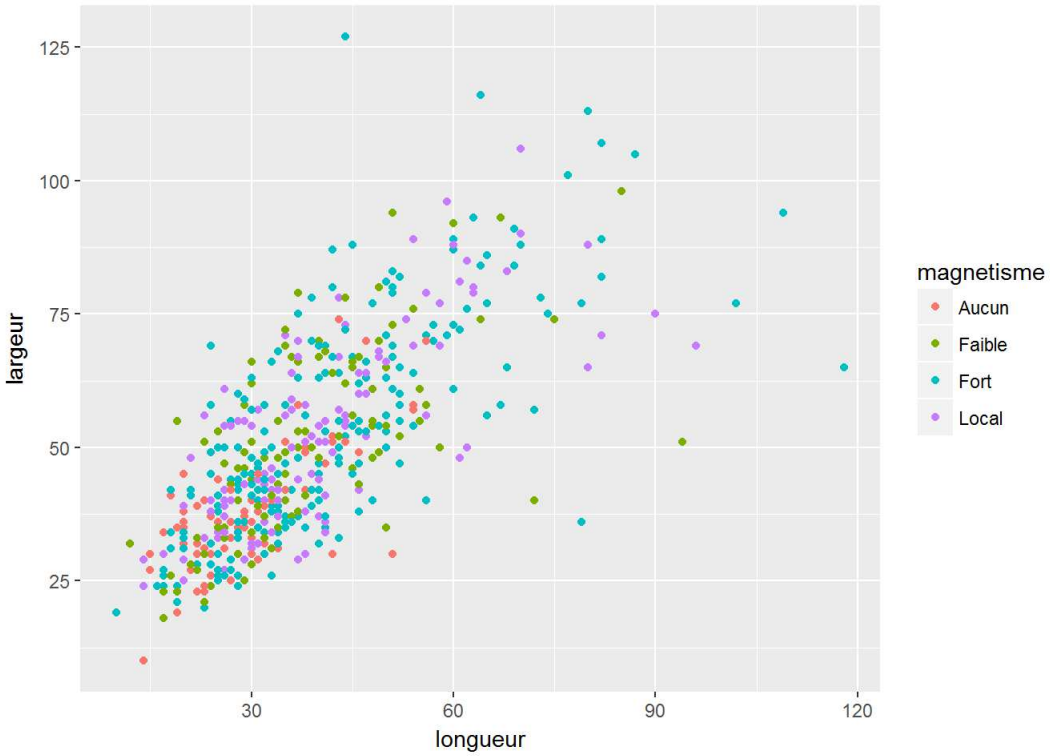


La liste des palettes de `scale_color_brewer` est indiquée en fin de section suivante.

Variables qualitatives

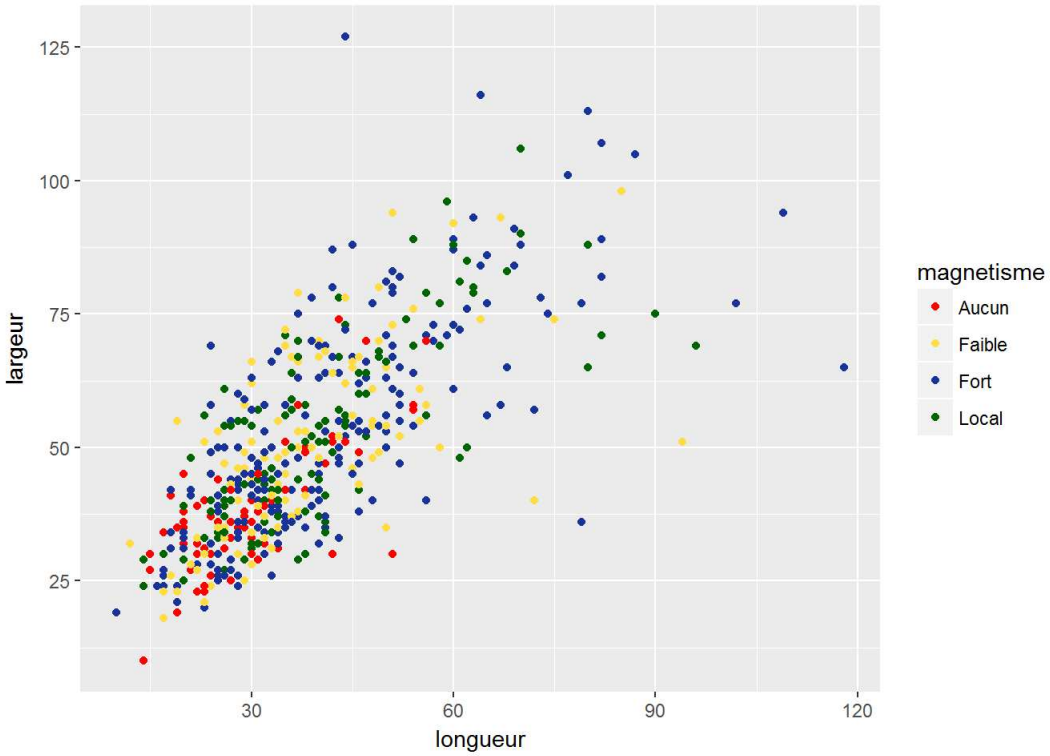
Si on a fait un mappage avec une variable discrète (qualitative), comme ici avec le magnétisme :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = magnetisme))
```



Une première possibilité est de modifier la palette manuellement avec `scale_color_manual` et son argument `values` :

```
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = magnetisme)) +
  scale_color_manual("magnetisme",
                     values = c("red", "#FFDD45", rgb(0.1,0.2,0.6), "darkgreen", "grey80"))
```



L'exemple précédent montre plusieurs manières de définir manuellement des couleurs dans R : - Par code hexadécimal : `"#FFDD45"` - En utilisant la fonction `rgb` et en spécifiant les composantes rouge, vert, bleu par des nombres entre 0 et 1 (et optionnellement une quatrième composante d'opacité, toujours entre 0 et 1) : `rgb(0.1,0.2,0.6)` - En donnant un nom de couleur : `"red"`, `"darkgreen"`

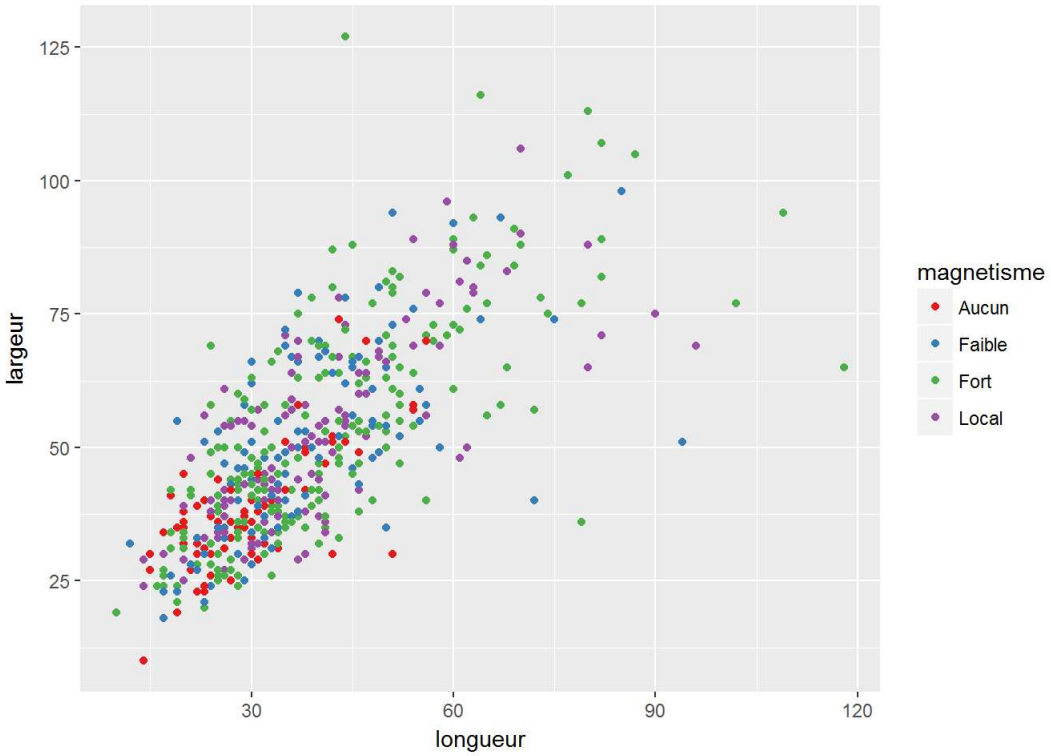
La liste complète des noms de couleurs connus par R peut être obtenu avec la fonction `colors()`. Vous pouvez aussi retrouver en ligne la liste des couleurs et leur nom (http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf) (PDF).

Il est cependant souvent plus pertinent d'utiliser des palettes prédéfinies. Celles du site Colorbrewer (http://colorbrewer2.org/), initialement prévues pour la cartographie, permettent une bonne lisibilité, et peuvent être adaptées pour cotains types de daltonisme.

Ces palettes s'utilisent via la fonction `scale_color_brewer`, en passant le nom de la palette via l'argument `palette`. Par exemple, si on veut utiliser la palette `Set1` :

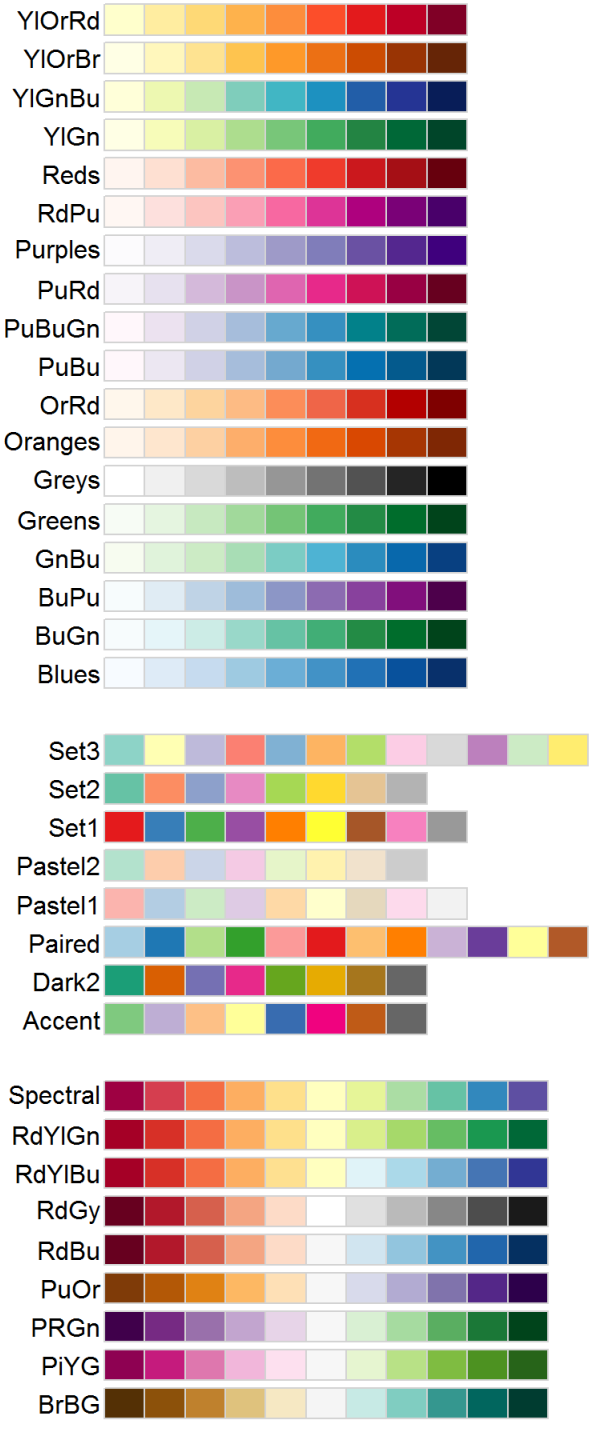
Attention: il faut installer préalablement le package "RColorBrewer""

```
# install.packages("RColorBrewer")
ggplot(co) +
  geom_point(aes(x = longueur, y = largeur, color = magnetisme)) +
  scale_color_brewer("magnetisme", palette = "Set1")
```



Le graphique suivant, accessible via la fonction `display.brewer.all()`, montre la liste de toutes les palettes disponibles via `scale_color_brewer`. Elles sont réparties en trois familles : les palettes séquentielles (pour une variable quantitative), les palettes qualitatives, et les palettes divergentes (typiquement pour une variable quantitative avec une valeur de référence, souvent 0, et deux palettes continues distinctes pour les valeurs inférieures et pour les valeurs supérieures).

```
RColorBrewer::display.brewer.all()
```



Il existe d'autres méthodes pour définir les couleurs : pour plus d'informations on pourra se reporter à l'article de la documentation officielle sur ce sujet (http://ggplot2.tidyverse.org/articles/ggplot2-specs.html#colour).

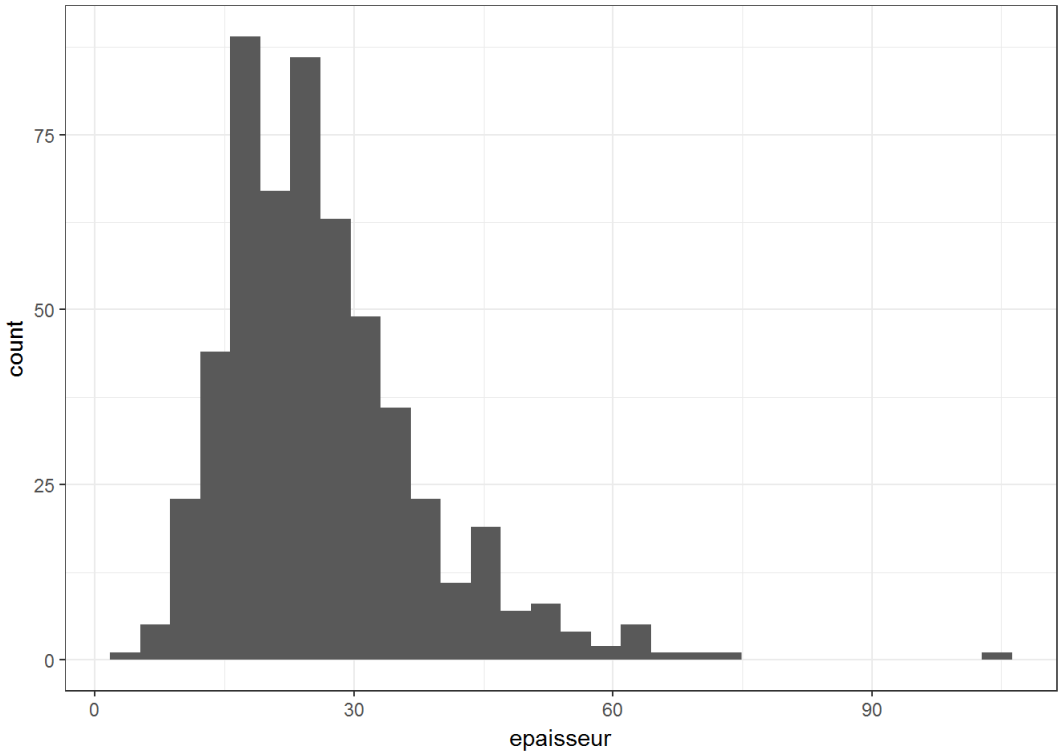
Thèmes

Les thèmes permettent de contrôler l'affichage de tous les éléments du graphique qui ne sont pas reliés aux données : titres, grilles, fonds, etc.

Il existe un cotain nombre de thèmes préexistants, par exemple le thème `theme_bw` :

```
ggplot(data = co) +
  geom_histogram(aes(x = epaisseur)) +
  theme_bw()
```

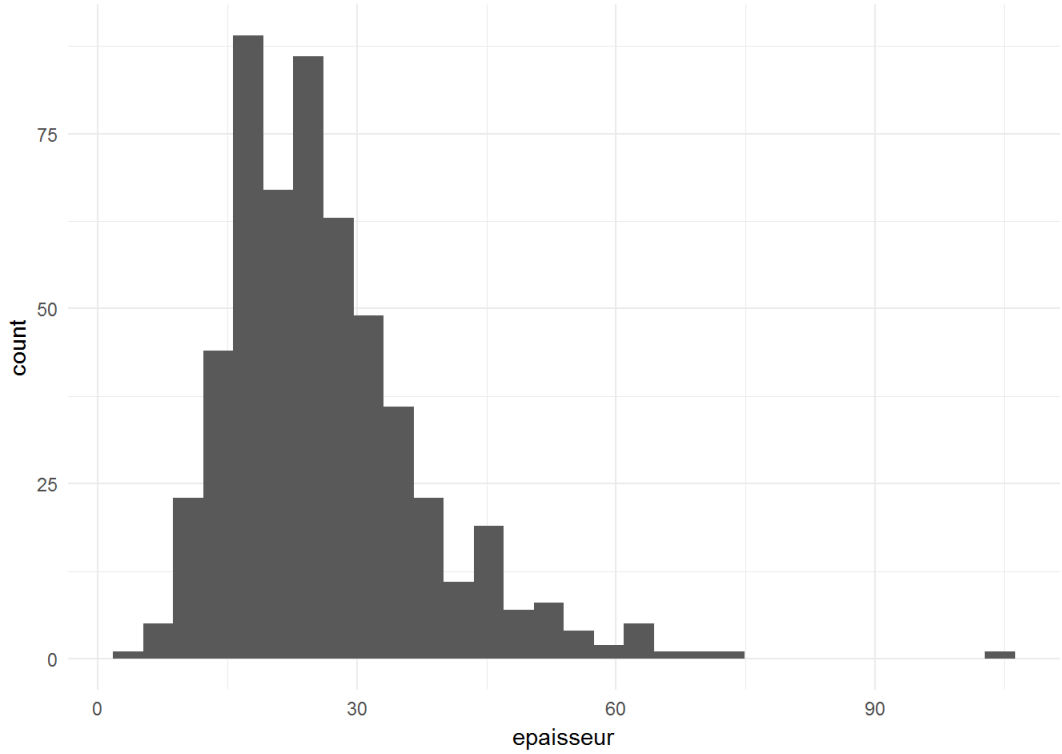
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Ou le thème `theme_minimal` :

```
ggplot(data = co) +
  geom_histogram(aes(x = epaisseur)) +
  theme_minimal()
```

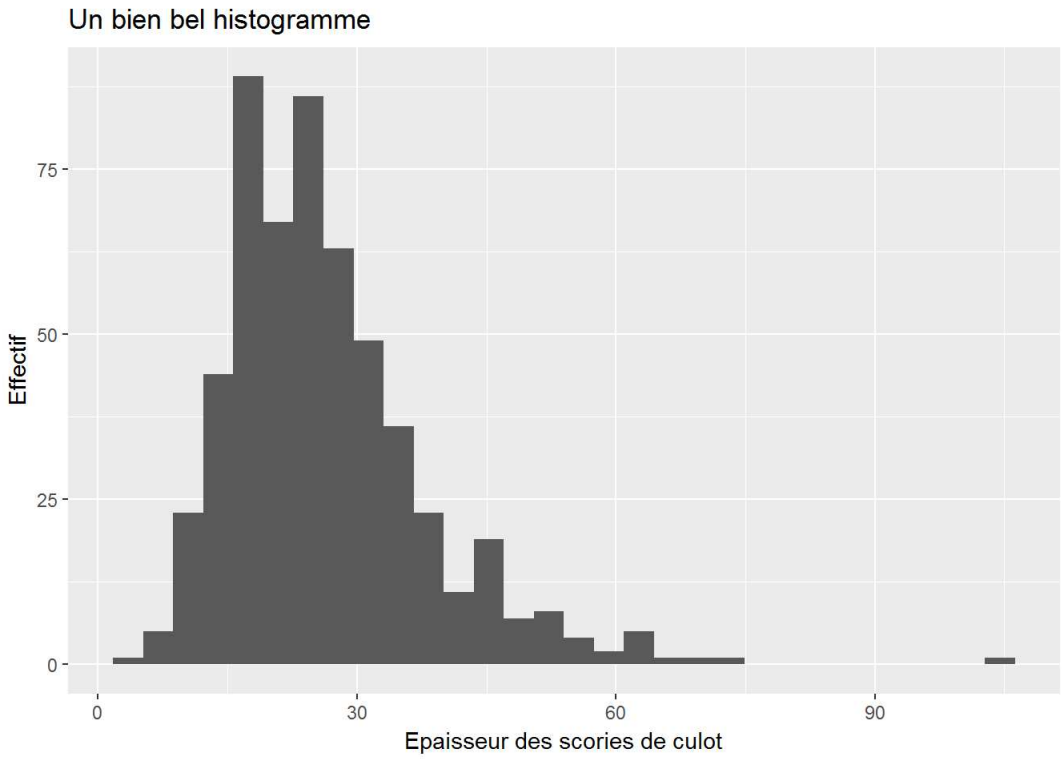
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



On peut cependant modifier manuellement les différents éléments. Par exemple, les fonctions `ggtitle`, `xlab` et `ylab` permettent d'ajouter ou de modifier le titre du graphique, ainsi que les étiquettes des axes `x` et `y` :

```
ggplot(data = co) +
  geom_histogram(aes(x = epaisseur)) +
  ggtitle("Un bien bel histogramme") +
  xlab("Epaisseur des scories de culot") +
  ylab("Effectif")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

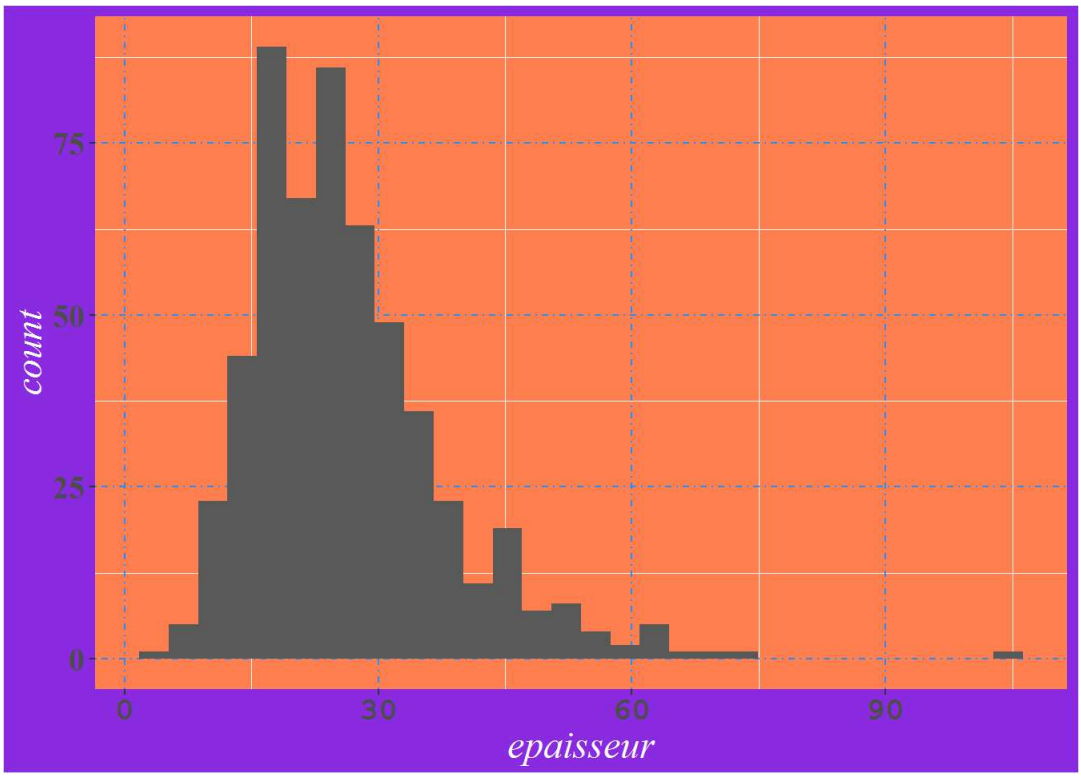


Les éléments personnalisables étant nombreux, un bon moyen de se familiariser avec tous les arguments est sans doute l'addin RStudio `ggThemeAssist`. Pour l'utiliser il suffit d'installer le package du même nom, de sélectionner dans son script RStudio le code correspondant à un graphique `ggplot2`, puis d'aller dans le menu *Addins* et choisir *ggplot Theme Assistant*. Une interface graphique s'affiche alors permettant de modifier les différents éléments. Si on clique sur *Done*, le code sélectionné dans le script est alors automatiquement mis à jour pour correspondre aux modifications effectuées.

Ce qui permet d'obtenir très facilement des résultats extrêmement moches :

```
ggplot(data = co) + geom_histogram(aes(x = epaisseur)) +
 theme(panel.grid.major = element_line(colour = "dodgerblue",
    size = 0.5, linetype = "dotdash"), axis.title = element_text(family = "serif",
    size = 18, face = "italic", colour = "white"),
    axis.text = element_text(family = "serif",
        size = 15, face = "bold"), axis.text.x = element_text(family = "mono"),
    plot.title = element_text(family = "serif"),
    legend.text = element_text(family = "serif"),
    legend.title = element_text(family = "serif"),
    panel.background = element_rect(fill = "coral"),
    plot.background = element_rect(fill = "blueviolet"))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Ressources

La documentation officielle (http://ggplot2.tidyverse.org/index.html) (en anglais) de `ggplot2` est très complète et accessible en ligne.

Une "antisèche" (en anglais) résumant en deux pages l'ensemble des fonctions et arguments et disponible soit directement depuis RStudio (menu *Help* > *Chetasheets* > *Data visualization with ggplot2*) ou directement en ligne (https://www.rstudio.com/resources/cheatsheets/)

Les parties Data visualisation (http://r4ds.had.co.nz/data-visualisation.html) et Graphics for communication (http://r4ds.had.co.nz/graphics-for-communication.html) de l'ouvrage en ligne *R for data science*, de Hadley Wickham, sont une très bonne introduction à `ggplot2`.

Plusieurs ouvrages, toujours en anglais, abordent en détail l'utilisation de `ggplot2`, en particulier ggplot2: Elegant Graphics for Data Analysis (http://www.amazon.fr/ggplot2-Elegant-Graphics-Data-Analysis/dp/0387981403/), toujours de Hadley Wickham, et le R Graphics Cookbook (http://www.amazon.fr/R-Graphics-Cookbook-Winston-Chang/dp/1449316956) de Winston Chang.

Le site associé (http://www.cookbook-r.com/Graphs/) à ce dernier ouvrage comporte aussi pas mal d'exemples et d'informations

intéressantes.

Enfin, si `ggplot2` présente déjà un très grand nombre de fonctionnalités, il existe aussi un système d'extensions permettant d'ajouter des `geom`, des thèmes, etc. Le site ggplot2 extensions (http://www.ggplot2-exts.org/) est une très bonne ressource pour les parcourir et les découvrir, notamment grâce à sa galerie (http://www.ggplot2-exts.org/gallery/).