

S05: Fonctionnalités graphiques avancées

Analyse de données quantitatives avec R

Samuel Coavoux

- 1 Personnaliser les sorties graphiques de R-base
- 2 ggplot
- 3 Calculs préalables aux graphs complexes
- 4 Personnalisation

Personnaliser les sorties graphiques de R-base

Données

```
x <- 1990:2010  
y <- rnorm(21)^2
```

Scatterplot

```
plot(x, y)
```

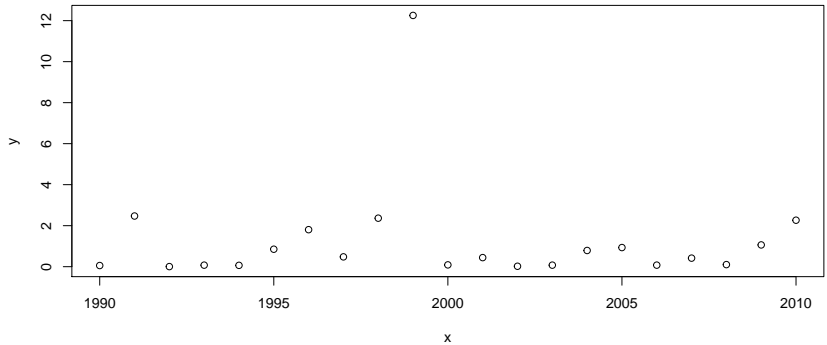
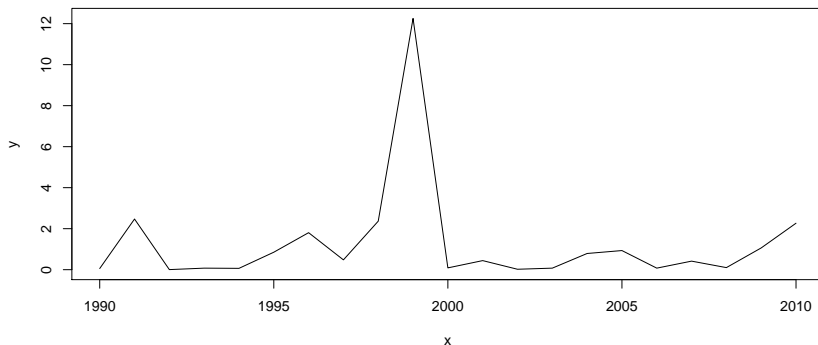


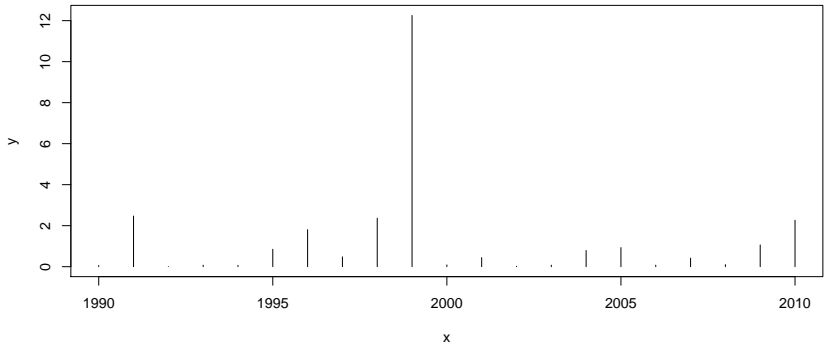
Diagramme en ligne

```
plot(x, y, type = "l")
```



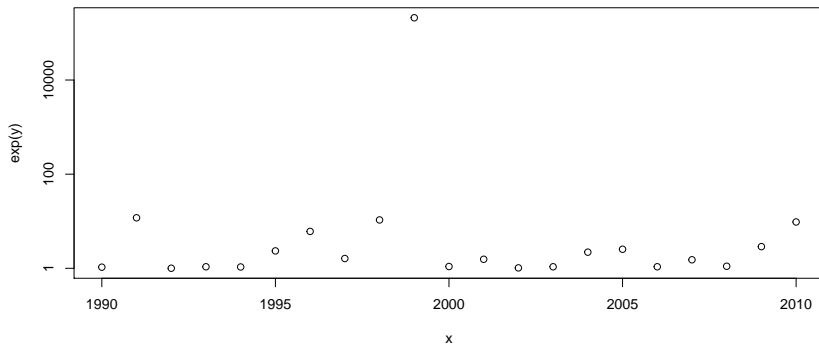
Histogram

```
plot(x, y, type = "h")
```



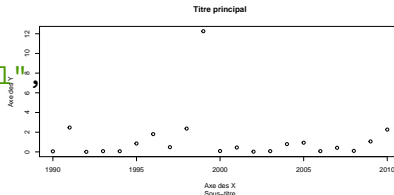
Échelle logarithmique

```
plot(x, exp(y), log = "y")
```



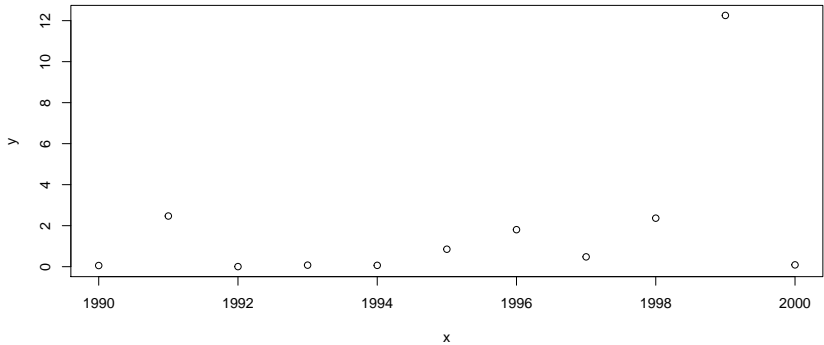
Titres, sous-titres, et titres des axes

```
plot(x, y,  
      main = "Titre principal",  
      sub = "Sous-titre",  
      xlab = "Axe des X",  
      ylab = "Axe des Y")
```



Zoom

```
plot(x, y, xlim = c(1990, 2000))
```



ggplot

Grammar of graphics

La théorie: Leland Wilkinson, *The Grammar of Graphics*, Springer, 1999.

La pratique: Hadley Wickham, package ggplot2.

La grammaire des graphiques est une théorie visant à formaliser la représentation visuelle des données.

Théorie

Un graphique sous ggplot est composé de différents éléments, qui sont autant de couches (layers)

- des données (dans un data.frame) (data);
- des “aesthetics mappings”: un rôle graphique donné à des vecteurs (aes);
- des objets géométriques (geom);
- des transformations statistiques (stat)
- des échelles (scale)
- des systèmes de coordonnées (coord)
- des subdivision (facet)

Un graphique se construit en empilant des couches. L'opérateur + permet cet empilement.

Premier exemple

```
library(ggplot2)
load("data/ESS7e02_1.stata/ess7.RData")
source("data/ess7_recodage.R")
library(dplyr)
d <- filter(d, cntry %in% c("AT", "BE", "FI", "FR", "GB", "IT", "NL", "PT", "SE", "UK"),
            eduyrs < 30)
d <- tbl_df(d)
```

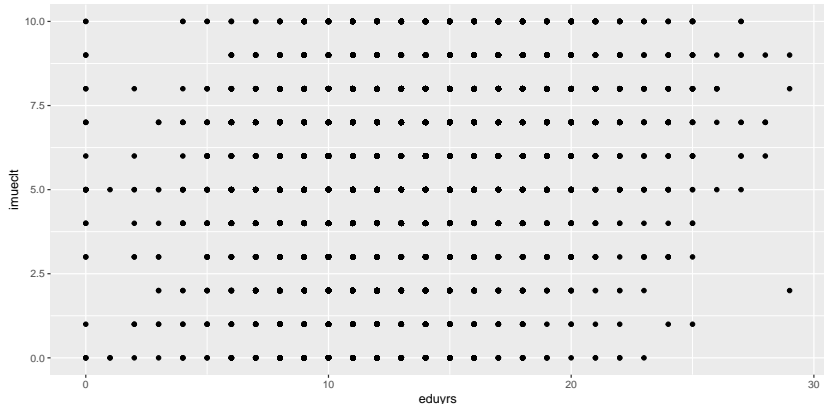
Conventions

Conventionnellement, on écrit chaque couche supplémentaire sur une ligne à part, avec une indentation de deux espaces.

```
ggplot(data, mapping) +  
  geom_*() +  
  stat_*() +  
  theme_*()
```

Premier exemple

```
ggplot(data = d, mapping = aes(x = eduyrs, y = imueclt)) +  
  geom_point()
```



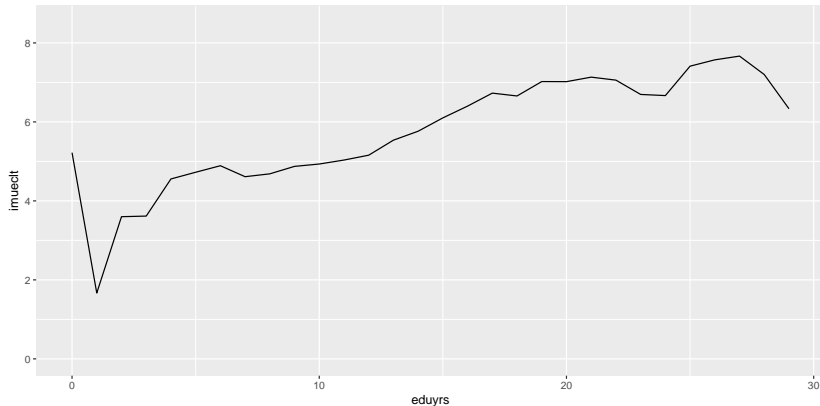
Geoms

Les geoms

- point : scatterplot
- line : diagramme en ligne
- bar : diagramme en bar
- histogram : histogramme
- boxplot : boîte à moustache
- smooth : lissage d'une courbe (estimation de la tendance et intervalle de confiance)
- text : écrire du texte
- errorbar: erreur-standard

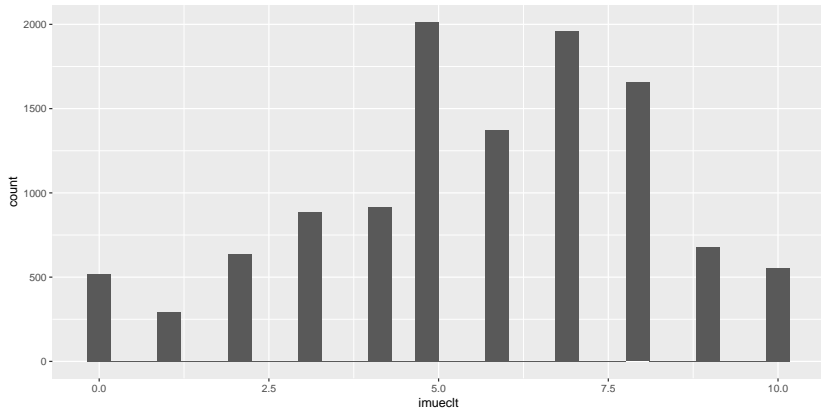
Ligne

```
ggplot(data = d, mapping = aes(x = eduyrs, y = imueclt)) +  
  geom_line(stat="summary")
```



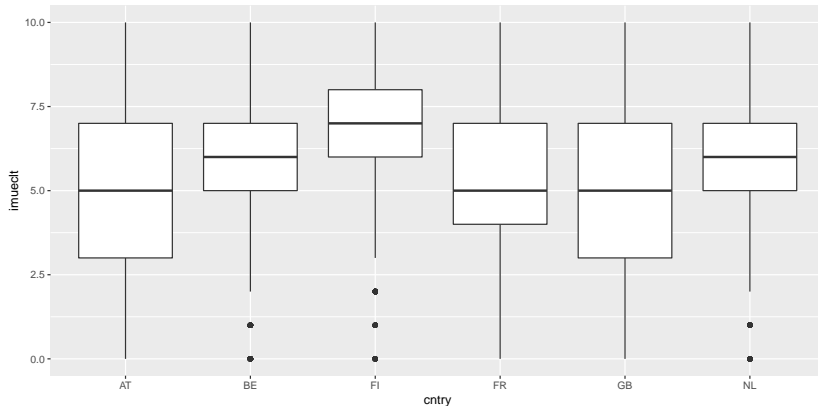
Histogram

```
ggplot(data = d, mapping = aes(x = imueclt)) +  
  geom_histogram()
```



Boxplot

```
ggplot(data = d, mapping = aes(x = cntry, y = imueclt)) +  
  geom_boxplot()
```



Mapping

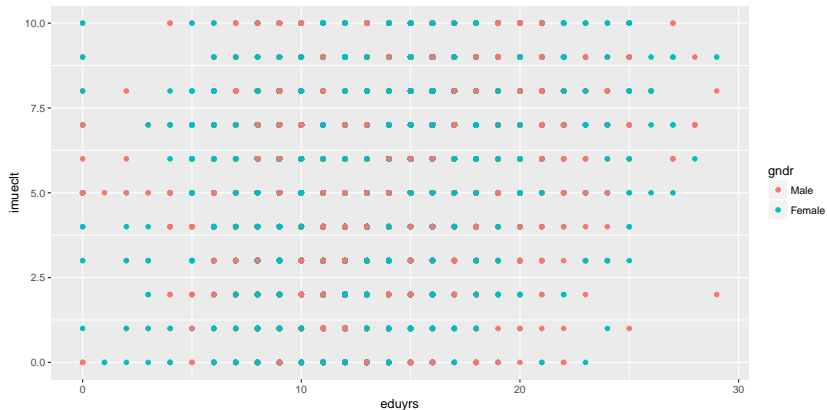
Ils dépendent des geom employés. Les plus fréquents sont:

- x, y : les vecteurs à représenter en abscisse/ordonnées;
- fill : le vecteur donnant sa couleur à un élément plein;
- color : le vecteur donnant sa couleur à un élément creux / au contour d'un élément plein;
- group : divise les données en groupes sans rien faire d'autre;
- linetype : le vecteur donnant un type de ligne;
- size : déterminer la taille des éléments.

Tous ces mapping sauf x et y groupent les données en autant de classes qu'il y a de modalités de la variable employée.

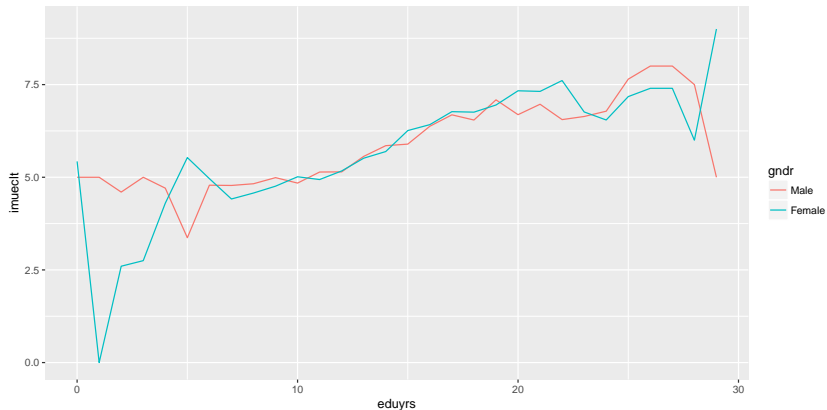
Groupement par mapping

```
ggplot(d, aes(x = eduyrs, y = imueclt, color = gndr)) +  
  geom_point()
```



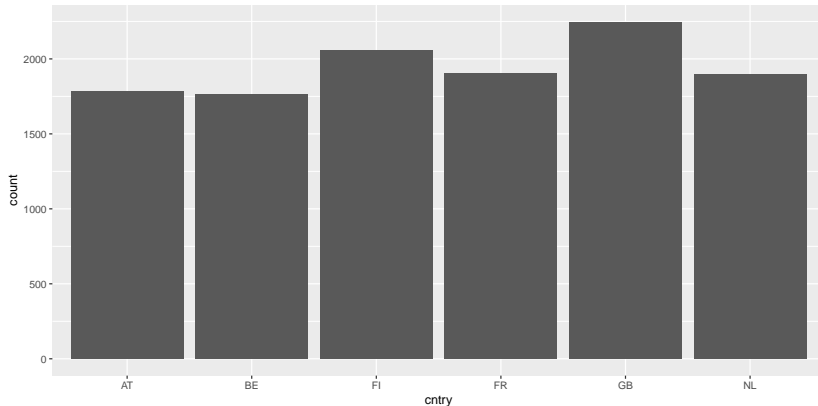
Groupement par mapping

```
ggplot(d, aes(x = eduyrs, y = imueclt, color = gndr)) +  
  geom_line(stat="summary")
```



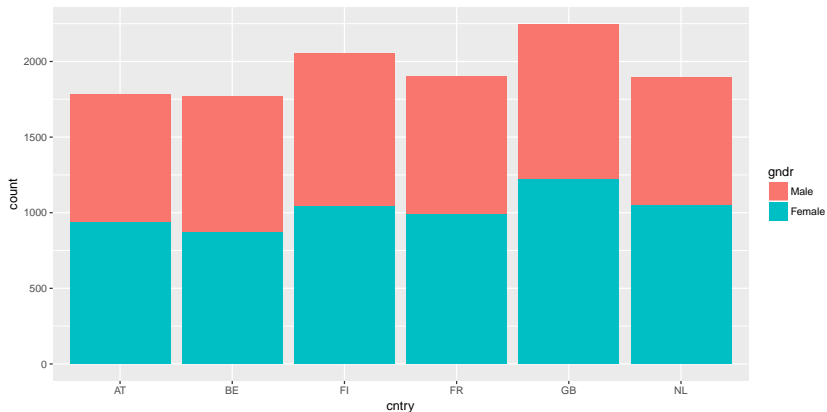
Groupement par mapping

```
ggplot(d, aes(x = cntry)) +  
  geom_bar()
```



Groupement par mapping

```
ggplot(d, aes(x = cntry, fill = gndr)) +  
  geom_bar()
```



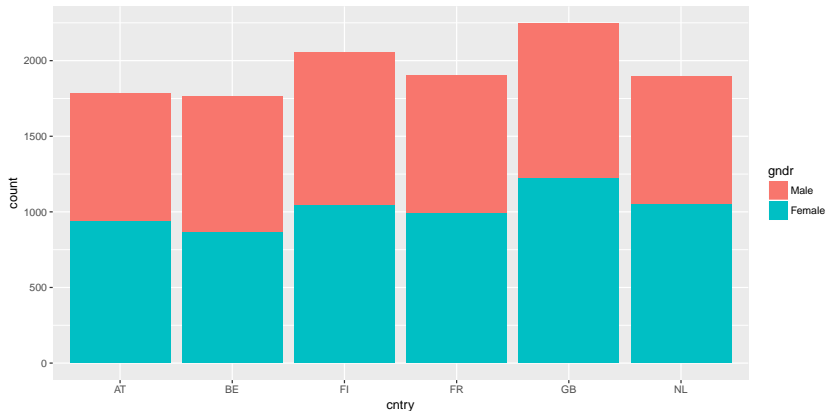
Positions

Pour `geom_bar` principalement, on peut ajuster la position des barres avec les fonctions `position_*`, lorsqu'il y a une variable mappée à `fill`, `colour`, ou `group`:

- `stack` : les barres sont empilées les unes sur les autres (par défaut);
- `fill` : les barres sont empilées, mais la proportion est représentée plutôt que l'effectif;
- `dodge` : les barres sont représentés côte à côte.

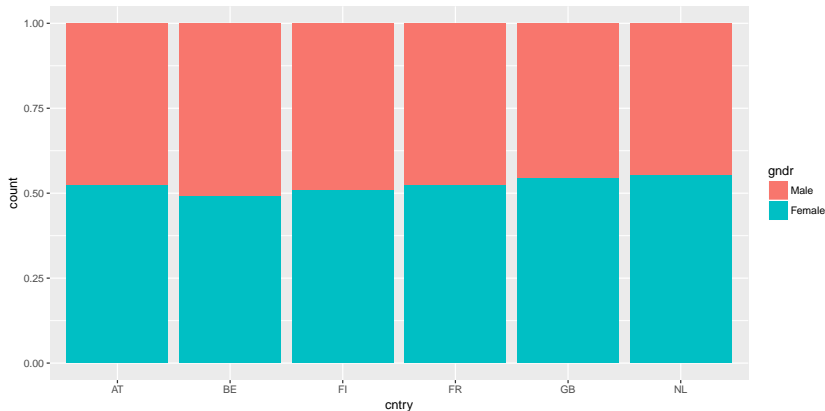
Positions

```
ggplot(d, aes(x = cntry, fill = gndr)) +  
  geom_bar(position = "stack")
```



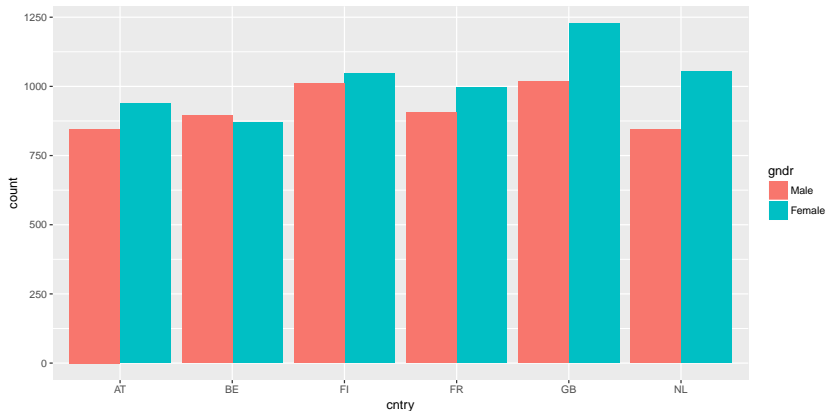
Positions

```
ggplot(d, aes(x = cntry, fill = gndr)) +  
  geom_bar(position = "fill")
```



Positions

```
ggplot(d, aes(x = cntry, fill = gndr)) +  
  geom_bar(position = "dodge")
```



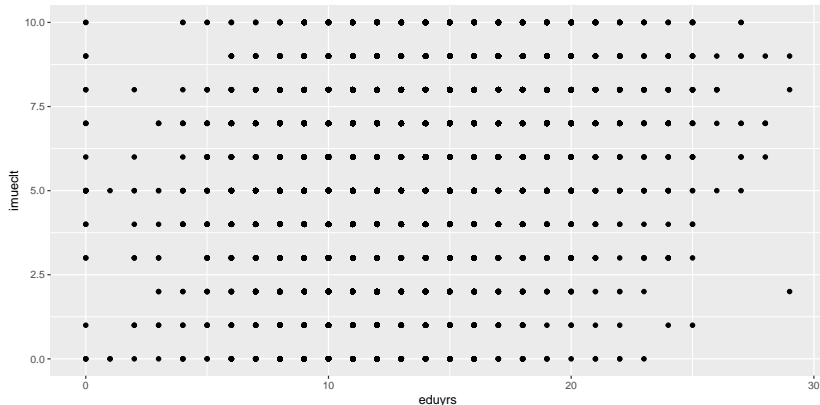
Positions

D'autres geom acceptent parfois des positions. En particulier, cela peut être utile lorsque l'on représente un scatterplot avec une très forte concentration des points autour de quelques valeurs. Avec la position par défaut (`position_identity`), les points qui ont les mêmes coordonnées se surajoutent les uns aux autres.

`position_jitter` les décale légèrement, au hasard, de leurs coordonnées

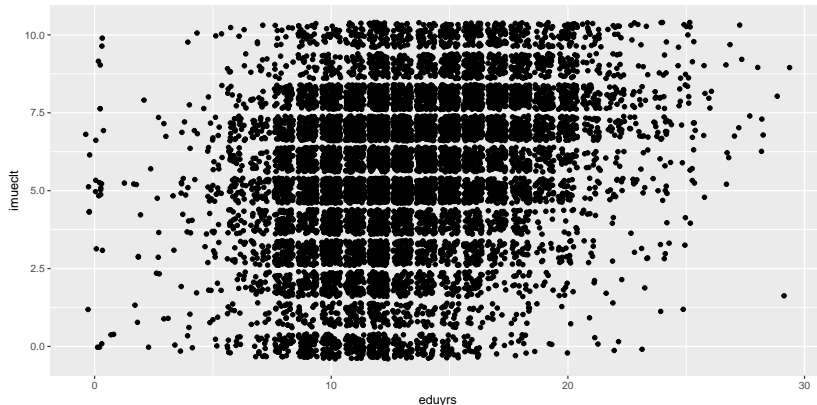
Positions: identity

```
ggplot(d, aes(x = eduyrs, y = imueclt)) +  
  geom_point()
```



Positions: jitter

```
ggplot(d, aes(x = eduyrs, y = imueclt)) +  
  geom_point(position = "jitter")
```



Stat

stat_*

Les fonctions `stat_*()` désignent les transformations statistiques réalisées sur les données avant qu'elles soient représentées sur le graph. Habituellement, on peut se passer de ces fonctions car chaque geom possède une valeur stat par défaut, qui est adaptée à son usage.

- `geom_histogram` => `stat_bin` (découper la variable en classe et faire le compte)
- `geom_bar` => `stat_count` (tri à plat de la variable)
- `geom_point`, `geom_line` => `stat_identity` (ne pas transformer les données)
- `geom_boxplot` => `stat_boxplot` (calculer médiane, quartile, etc.)

stat_*

Dans certaines situations, il peut être utile de changer de fonction `stat_`. Les cas les plus fréquents sont:

- lorsque l'on souhaite faire un barplot à partir de données agrégées et non brutes (donc à partir du tri à plat plutôt que du tableau croisé) => passe à `stat_identity`. Il y a alors deux façons de faire:
 - `geom_bar(stat="identity")` (classique)
 - `geom_col()` variante de `geom_bar` avec un `stat="identity"` par défaut
- lorsque l'on souhaite faire un graphique en ligne résumant des tendances centrales à partir d'un jeu de données brutes. Dans ce cas:
 - soit on transforme les données d'abord (le plus simple; cf. la section suivante)

Facettes

Facet

Les facettes permettent de découper un graph en sous-graph appliqués chacun à une population, d'après une variable de groupement. Il existe deux fonctions principales pour en créer:

- `facet_grid()`: définir une variable de groupement en ligne et/ou une en colonne ; le graphes comprendra autant de lignes et/ou colonnes que de modalités des variables en questions.
- `facet_wrap()`: définir une seule variable de groupement. Le graph final les alignera dans une grille, avec plusieurs lignes et colonnes, de façon optimale pour la lecture.

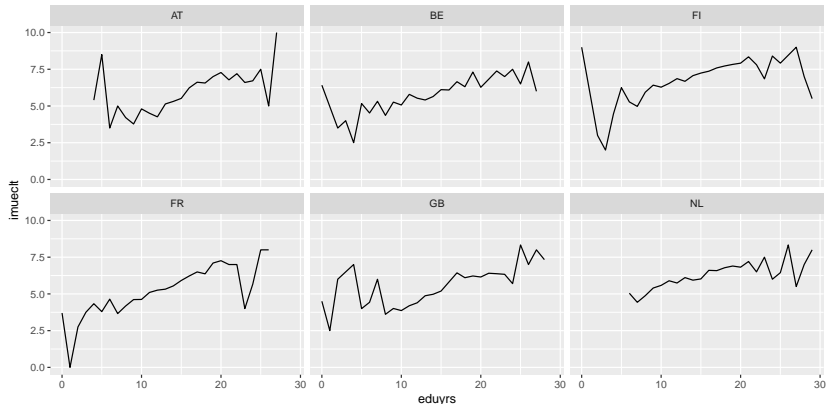
Facet: syntaxe

Les facettes emploient la notation en formule pour définir les variables de groupement. Pour des variables de groupement `var`, `var1` et `var2`

- `facet_wrap()`: on indique simplement `~ var`
- `facet_grid()`: on indique avant le tilde la variable en ligne et après le tilde la variable en colonne (`var1 ~ var2`) ; avec une seule variable, il faut ajouter `.` de l'autre côté du tilde : `var1 ~ .` groupe uniquement par `var1` en ligne

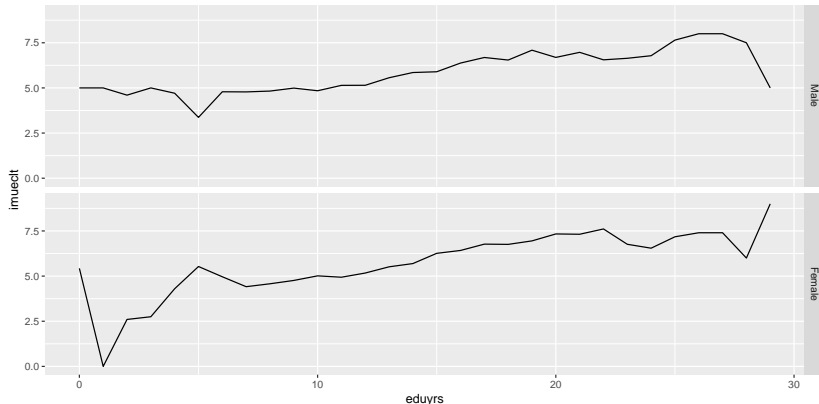
facet_wrap

```
ggplot(d, aes(x = eduyrs, y = imueclt)) +  
  geom_line(stat = "summary") + facet_wrap(~cntry)
```



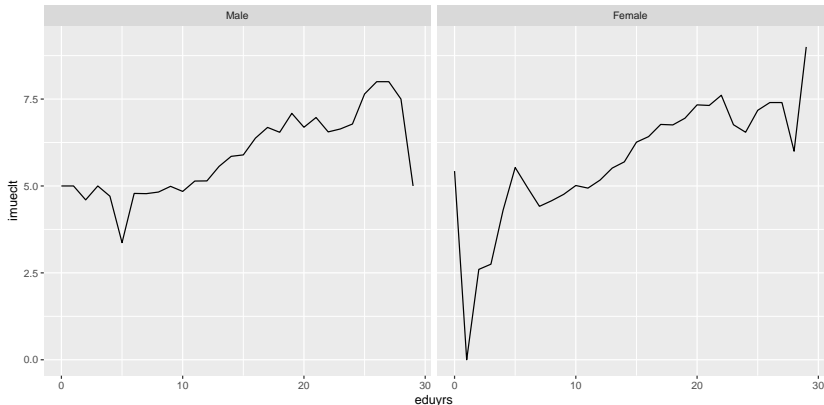
facet_grid: une seule variable en lignes

```
ggplot(d, aes(x = eduyrs, y = imueclt)) +  
  geom_line(stat = "summary") + facet_grid(gndr ~ .)
```



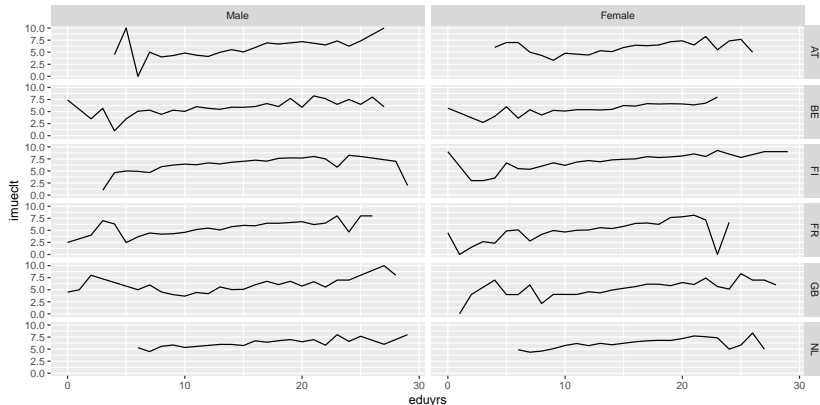
facet_grid: une seule variable en colonnes

```
ggplot(d, aes(x = eduyrs, y = imueclt)) +  
  geom_line(stat = "summary") + facet_grid(. ~ gndr)
```



facet_grid: deux variables

```
ggplot(d, aes(x = eduyrs, y = imueclt)) +  
  geom_line(stat = "summary") + facet_grid(cntry ~ gndr)
```



Calculs préalables aux graphs complexes

tidyr: principe

Pour les graphiques simples, croisant deux ou trois variables seulement, et utilisant les stats identity, bin, ou summary avec des fonctions simples, ggplot2 fonctionne très bien tout seul. Dès lors que l'on veut réaliser des choses un peu plus compliqué, il apparaît nécessaire de retravailler les données avant de les représenter.

Très souvent, pour ce faire, il faudra transformer le data.frame original d'un format large à un format long. Au début de ggplot2, ces opérations étaient souvent faites avec la fonction aggregate (en base-r) ou avec le package reshape; désormais, on emploie surtout tidyr, qui est l'héritier de reshape (comme dplyr hérite de plyr).

- format large: c'est le format classique d'une base de donnée : une ligne est un individu, une colonne est une variable
- format long : une ligne est **une observation**, c'est-à-dire un couple variable-valeur

Exemple de données larges et longues

table: base large

ID	Carac	Var2
1	A	X
2	B	Y

Exemple de données larges et longues

table: base longue

key	value
ID	1
ID	2
Carac	A
Carac	B

Exemple de données larges et longues

table: base longue

ID	key	value
1	Carac	A
2	Carac	B
1	Var2	X
2	Var2	Y

tidyr

Deux fonctions de tidyr permettent de passer d'un type de base à l'autre:

- `gather()` transforme une base large en base longue
- `spread()` transforme une base longue en base large

```
library(tidyr)
```

Gather

`gather()` prend comme premier argument le `data.frame` à remodeler, puis prend deux arguments `key` et `value` qui sont les noms que doivent prendre le vecteur des noms de variables d'une part, le vecteur des valeurs d'autre part dans le nouveau `data.frame`. Ensuite, on ajoute des arguments de sélection: à quelles variables du `data.frame` le passage de large à long doit-il être appliqué. La sélection peut être positive ou négative (quelles sont les variables qui doivent être épargnées).

Habituellement, lorsque l'on utilise `tidyr` pour préparer des données à leur représentation graphique, on commence par restreindre le `data.frame` aux seules variables d'intérêt et donc à seulement employer une sélection négative, pour préserver les variables *index*.

Gather

Les trois variables seront transformées. On ne peut plus faire le lien entre les valeurs d'un même individu.

```
select(d, imueclt, imbgeco, gndr) %>%  
  gather(var, val) %>%  
  group_by(var) %>% slice(1:3)
```

```
## Source: local data frame [9 x 2]
```

```
## Groups: var [3]
```

```
##
```

```
##      var      val
```

```
##    <chr>    <chr>
```

```
## 1    gndr      Male
```

```
## 2    gndr      Male
```

```
## 3    gndr    Female
```

Gather

On préserve une variable ; il devient possible de faire le lien entre cette variable et les autres.

```
select(d, imueclt, imbgeco, gndr) %>%  
  gather(var, val, -gndr) %>%  
  group_by(var) %>% slice(1:3)
```

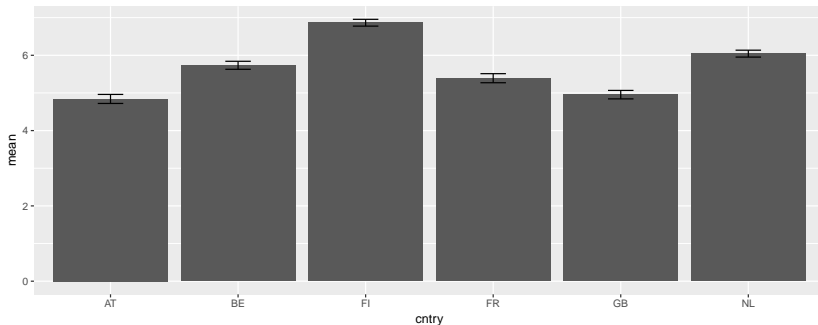
```
## Source: local data frame [6 x 3]  
## Groups: var [2]  
##  
##      gndr      var                val  
##   <fctr>   <chr>                <chr>  
## 1   Male imbgeco Bad for the economy  
## 2   Male imbgeco                    5  
## 3 Female imbgeco                    4
```

Exemple: calcul d'erreur standard

```
dt <- select(d, imueclt, cntry) %>%  
  gather(key, value, -cntry) %>%  
  group_by(cntry) %>%  
  filter(!is.na(value)) %>%  
  summarize(mean = mean(value),  
            se = 1.96 * sd(value) / sqrt(n()))
```

Exemple: calcul d'erreur standard

```
ggplot(dt, aes(x = cntry, y = mean)) +  
  geom_col() +  
  geom_errorbar(aes(ymin = mean-se, ymax = mean+se), width=
```

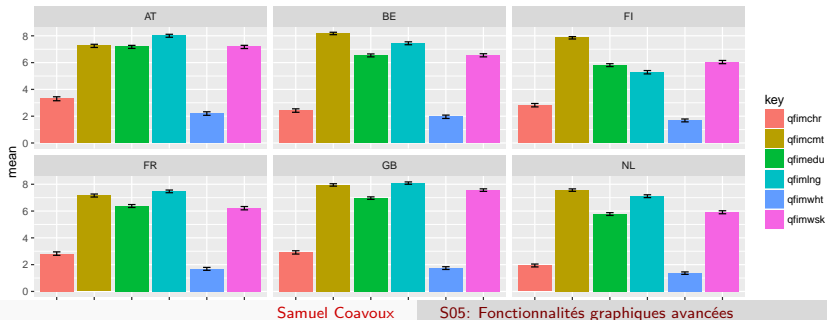


Exemple: produire un barplot unique avec plusieurs variables

```
dt <- select(d, starts_with("qfi"), cntry) %>%  
  gather(key, value, -cntry) %>%  
  group_by(cntry, key) %>%  
  filter(!is.na(value)) %>%  
  summarize(mean = mean(value),  
             se = 1.96 * sd(value) / sqrt(n()))
```

Exemple: produire un barplot unique avec plusieurs variables

```
ggplot(dt, aes(x = key, y = mean, fill = key)) +  
  geom_col() +  
  geom_errorbar(aes(ymin = mean-se, ymax = mean+se), width=1)  
  facet_wrap(~ cntry)
```



spread()

`spread()` est la fonction inverse de `gather()`. Elle permet de repasser d'un data.frame long à un data.frame large. Le premier argument est le data.frame concerné, le second, `key`, le nom du vecteur contenant les noms de variables, le troisième, `value`, le nom de la colonne contenant la valeur. Il est nécessaire d'avoir un identifiant, cad une autre variable qui identifie de façon unique chacun des individus.

On l'utilise beaucoup moins que `gather()`, qui doit être la fonction prioritaire.

spread()

```
df <- data.frame(id = c(1, 2, 1, 2),  
                  var = c("var1", "var1", "var2", "var2"),  
                  val = c(1, 2, 4, 7))  
spread(df, var, val)
```

```
##   id var1 var2  
## 1  1    1    4  
## 2  2    2    7
```

Personnalisation

Titres

Scales

Themes