S05: Fonctionnalités graphiques avancées

Analyse de données quantitatives avec R

Samuel Coavoux

Personnaliser les sorties graphiques de R-base

Données

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

Scatterplot

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

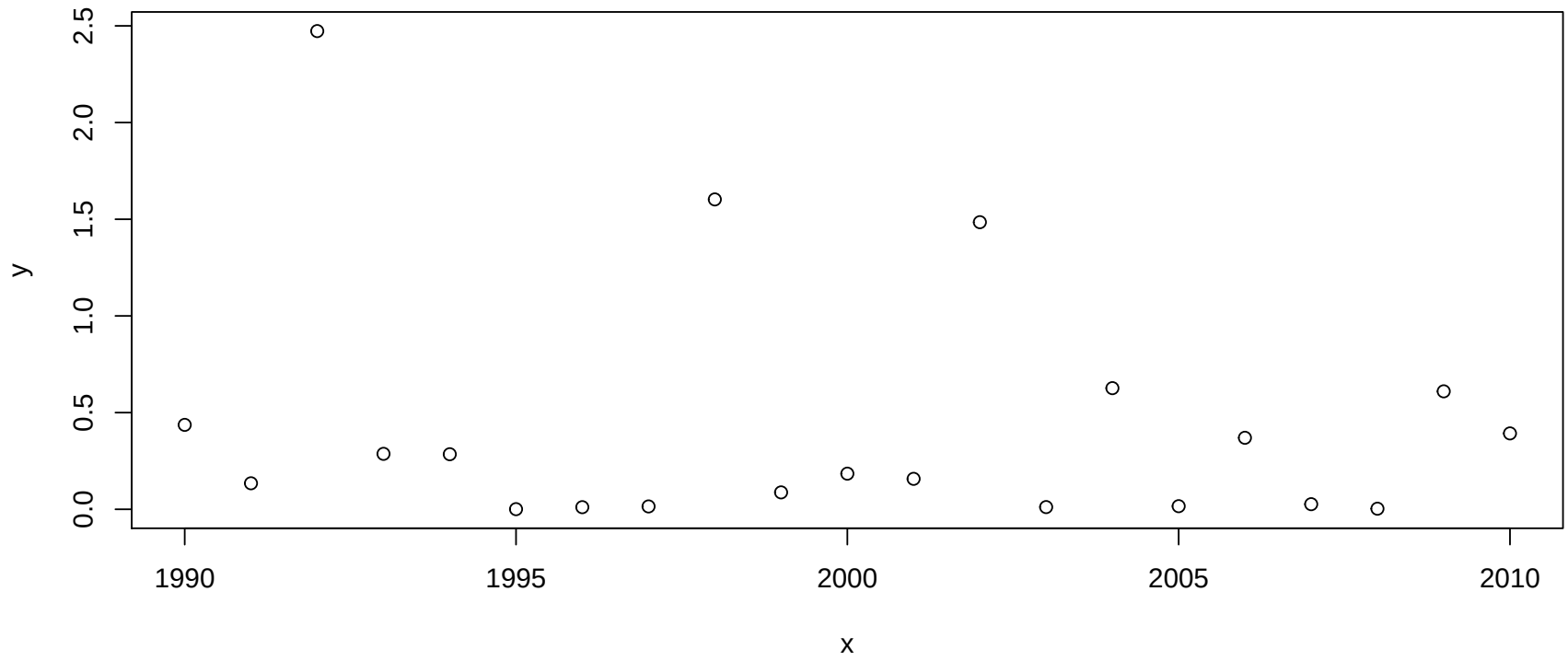
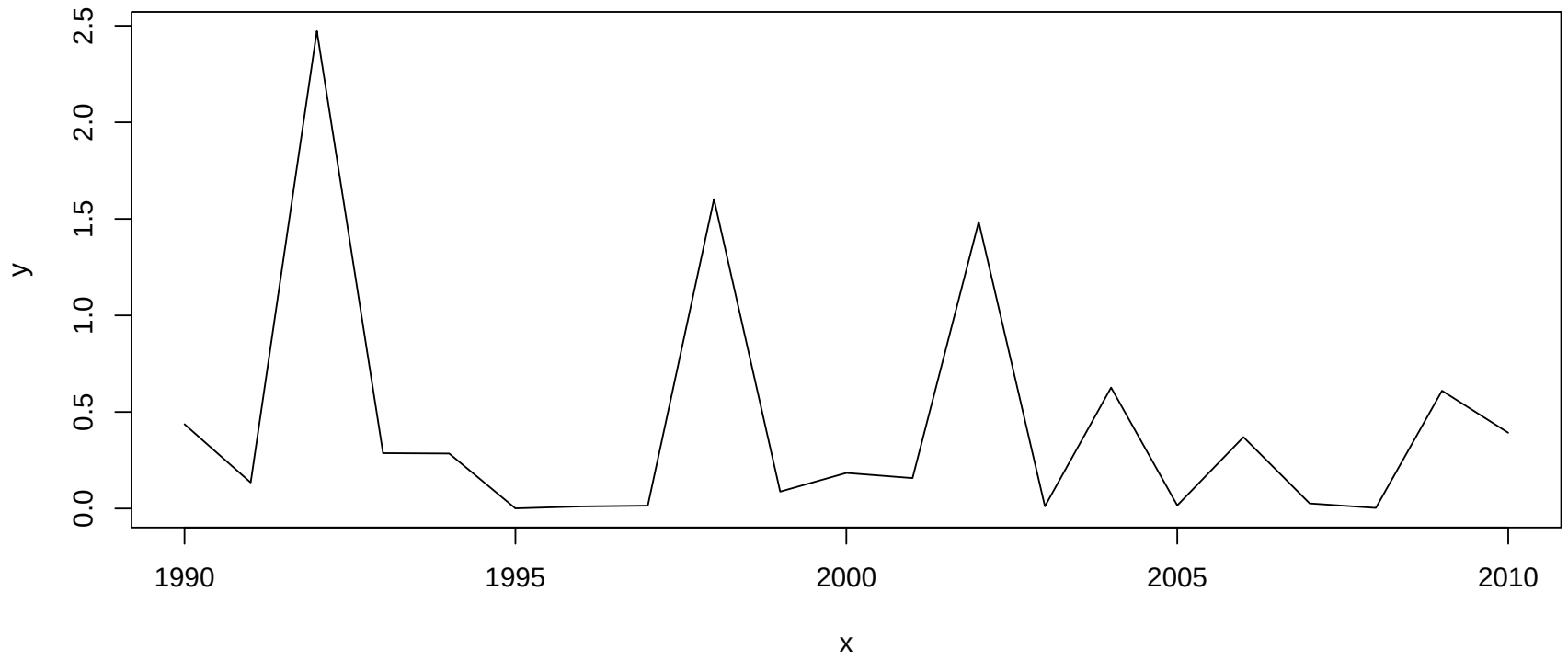


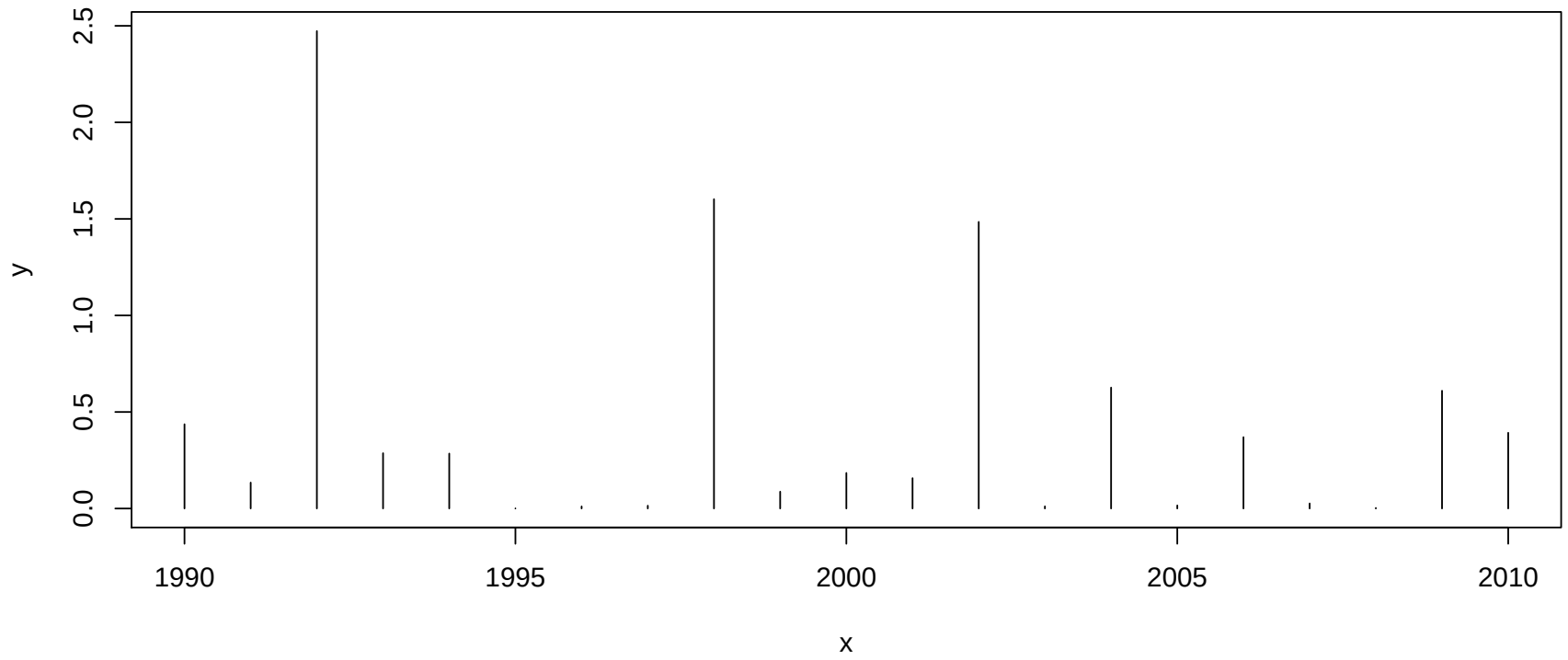
Diagramme en ligne

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



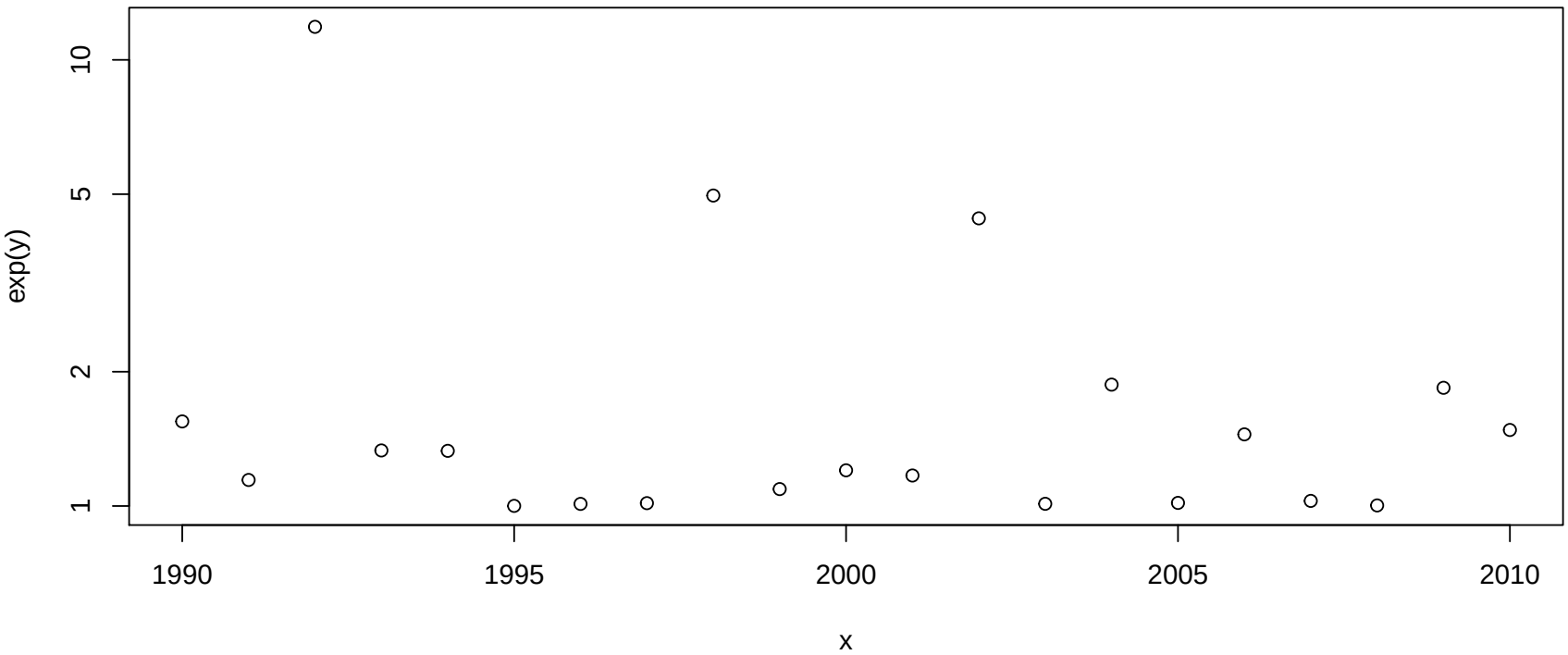
Histogram

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



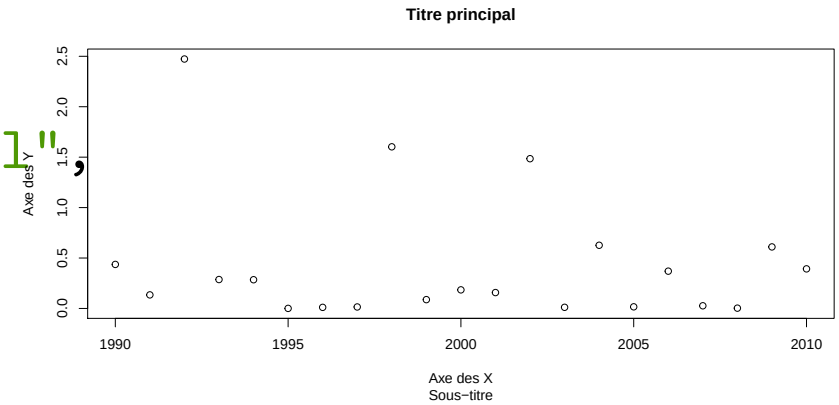
Échelle logarithmique

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



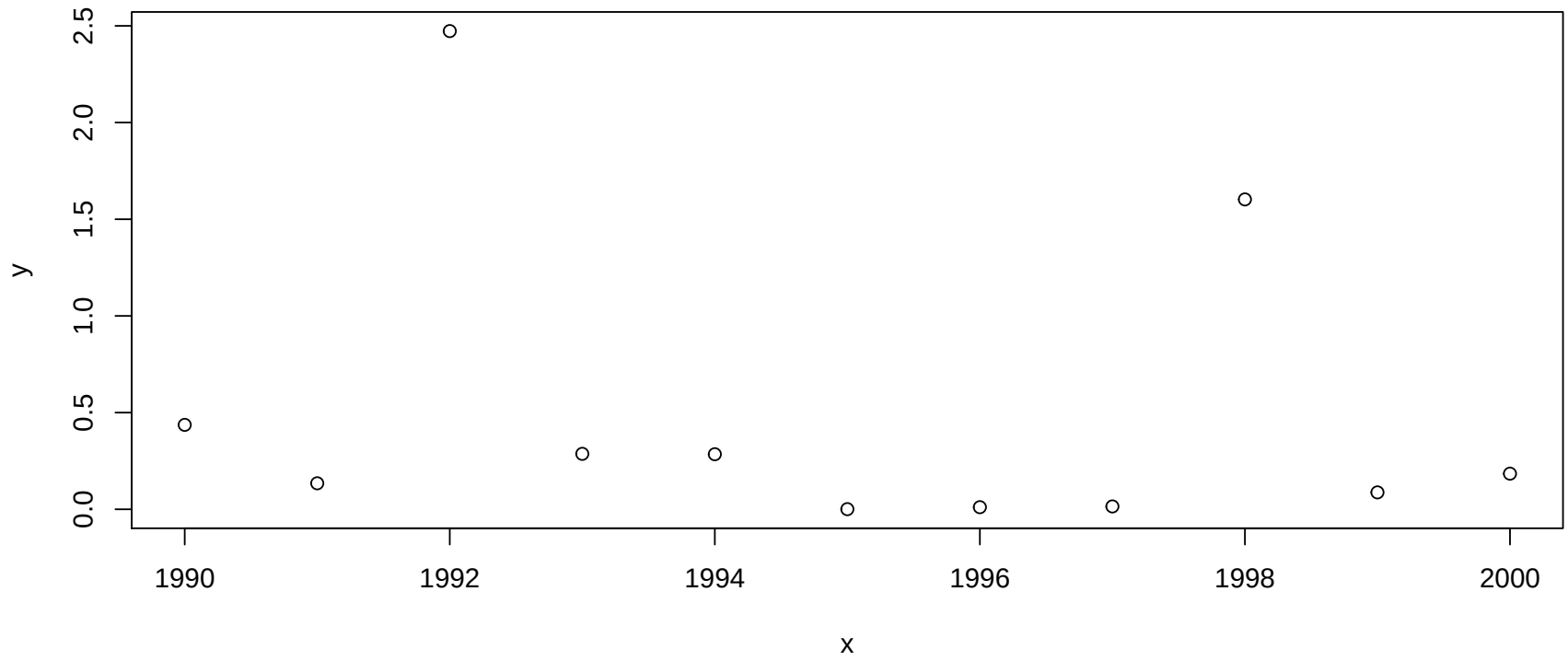
Titres, sous-titres, et titres des axes

```
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 Whickam, 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 ce 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", "NL"),
            eduyrs < 30)
d <- tbl_df(d)
```

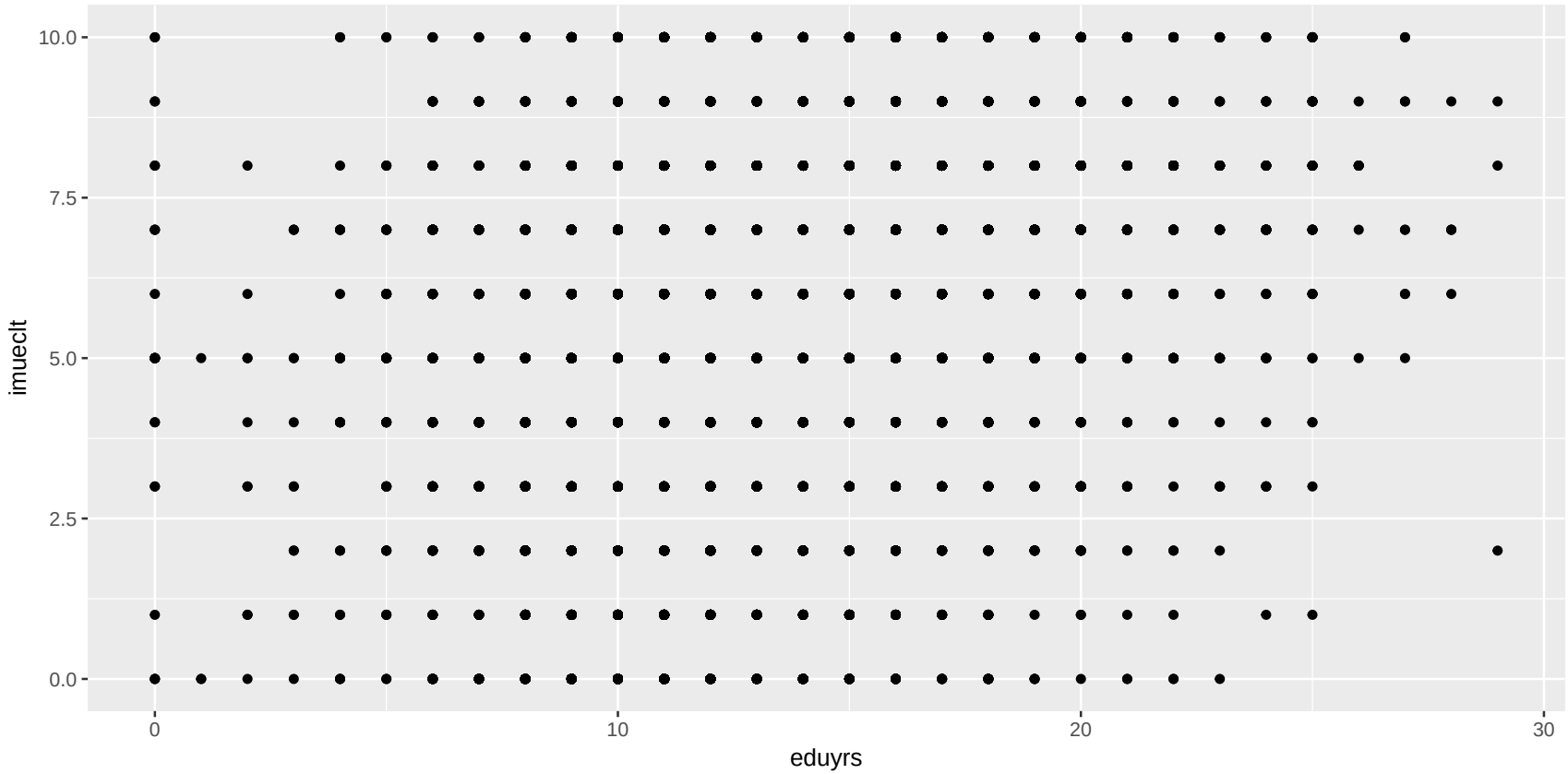
Conventions

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()
```



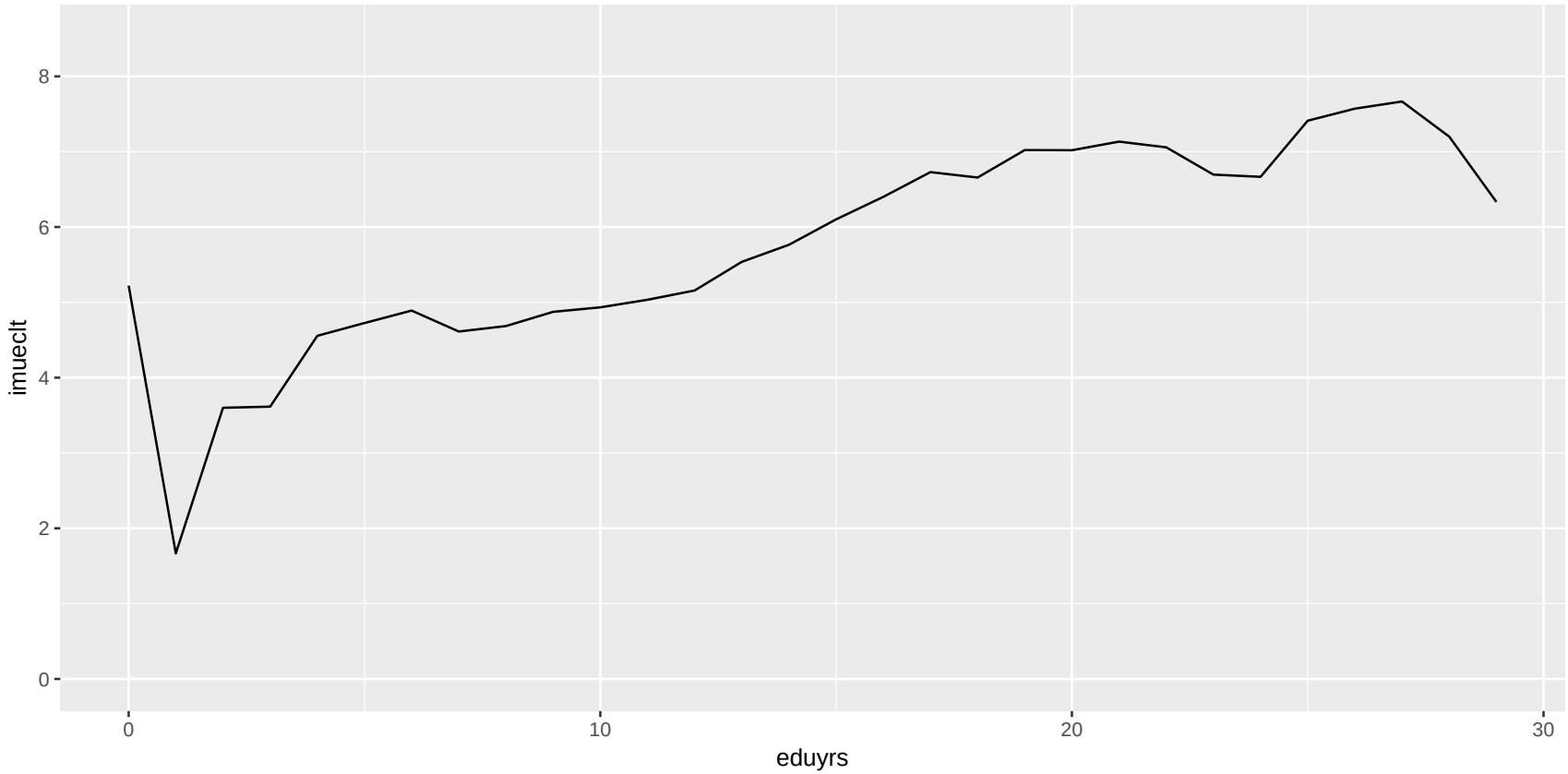
Geoms

Les geoms

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

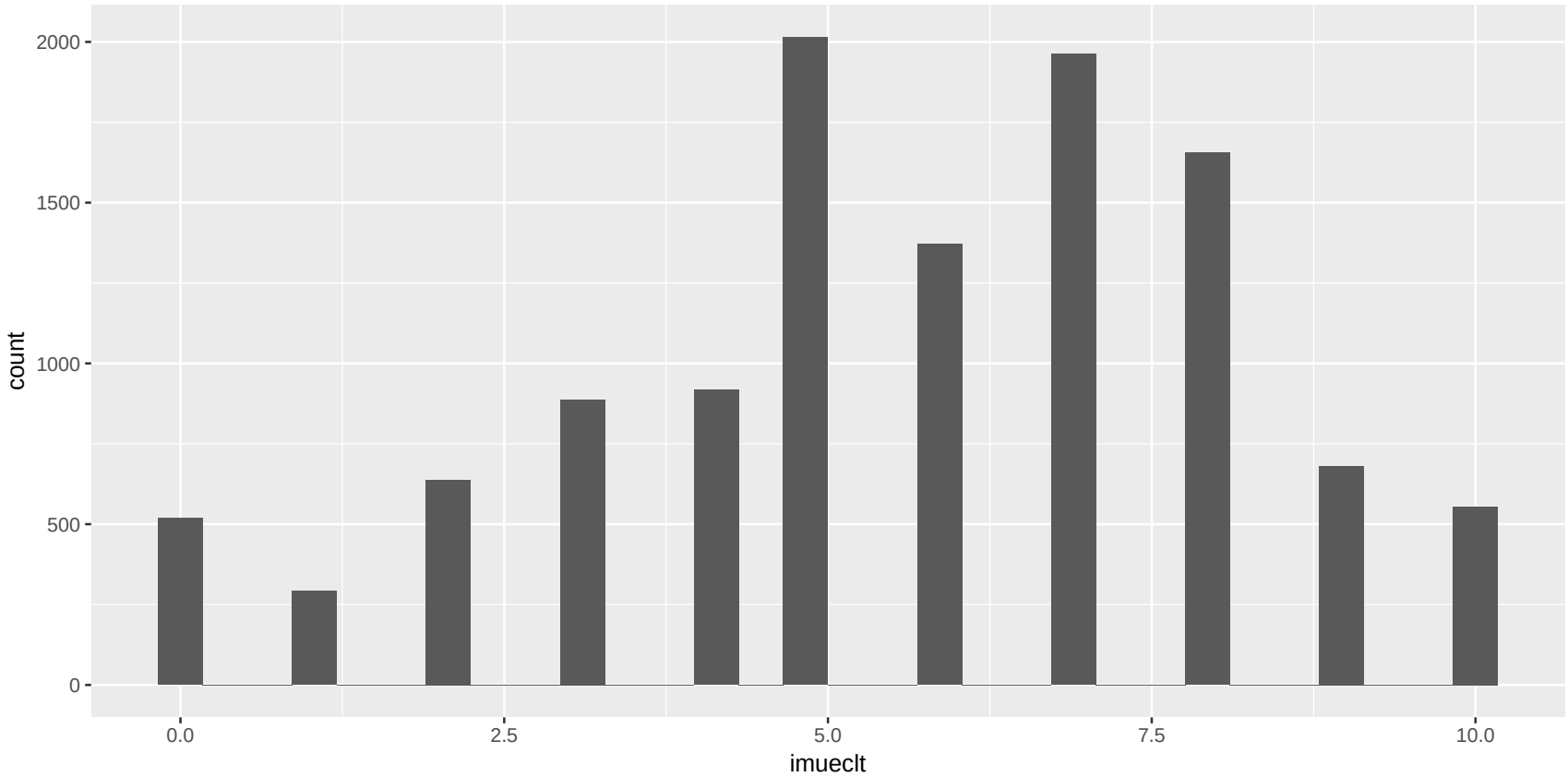
Ligne

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



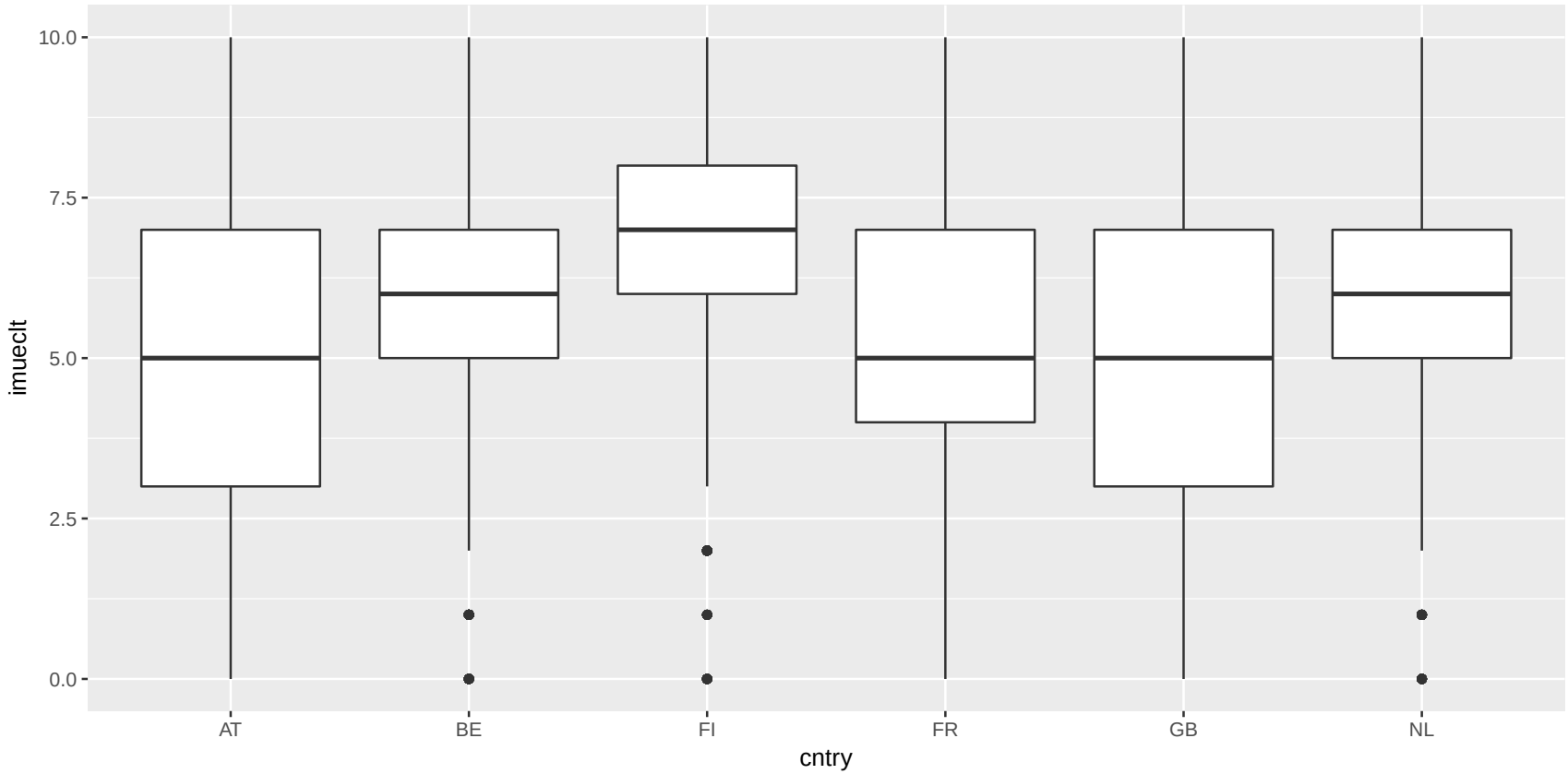
Histogram

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



Boxplot

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



Mapping

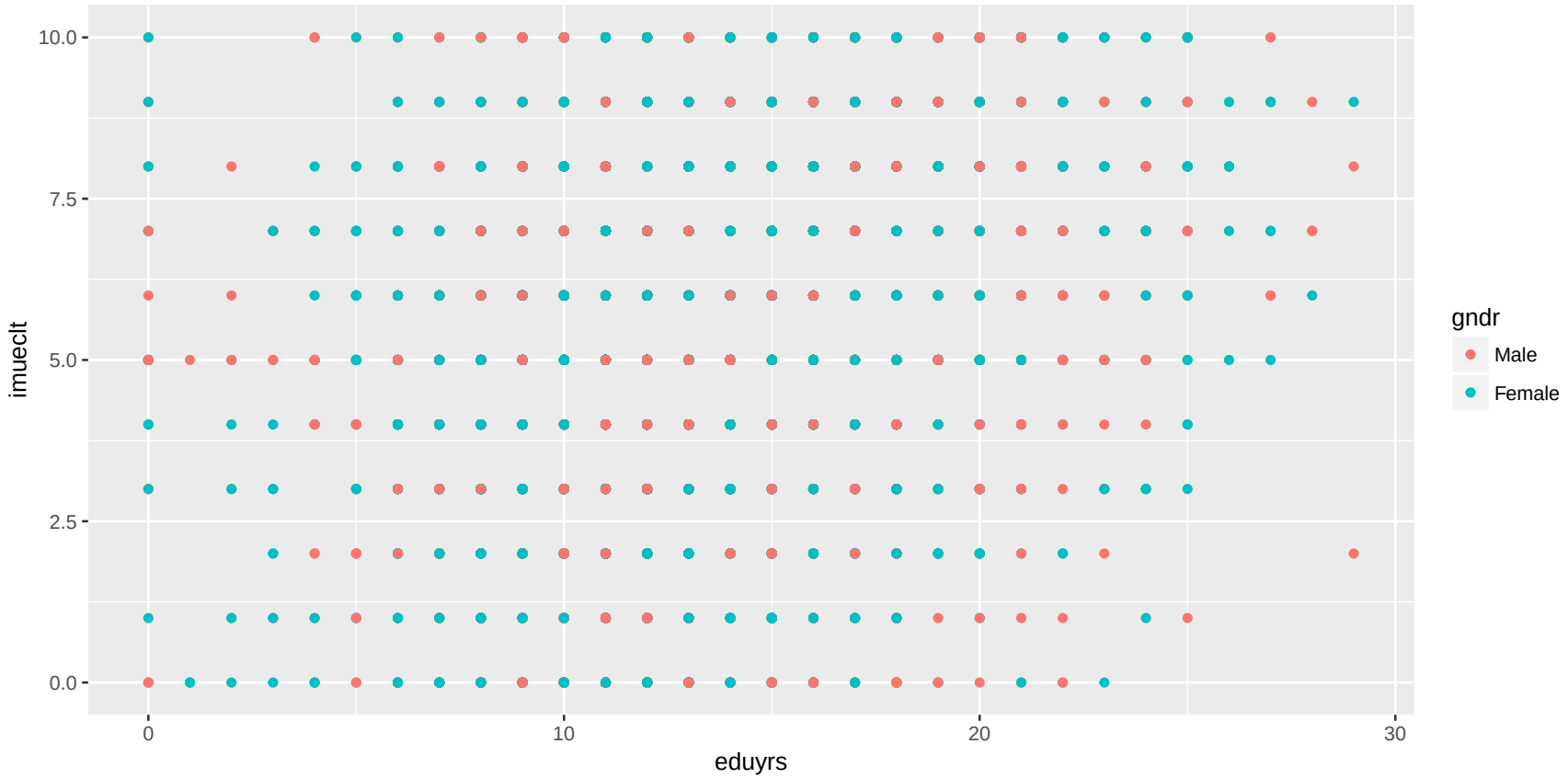
Ils dépendent des geom employés. Les plus fréquents sont:

- x, y : les vecteurs à représenter en abcisse/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.

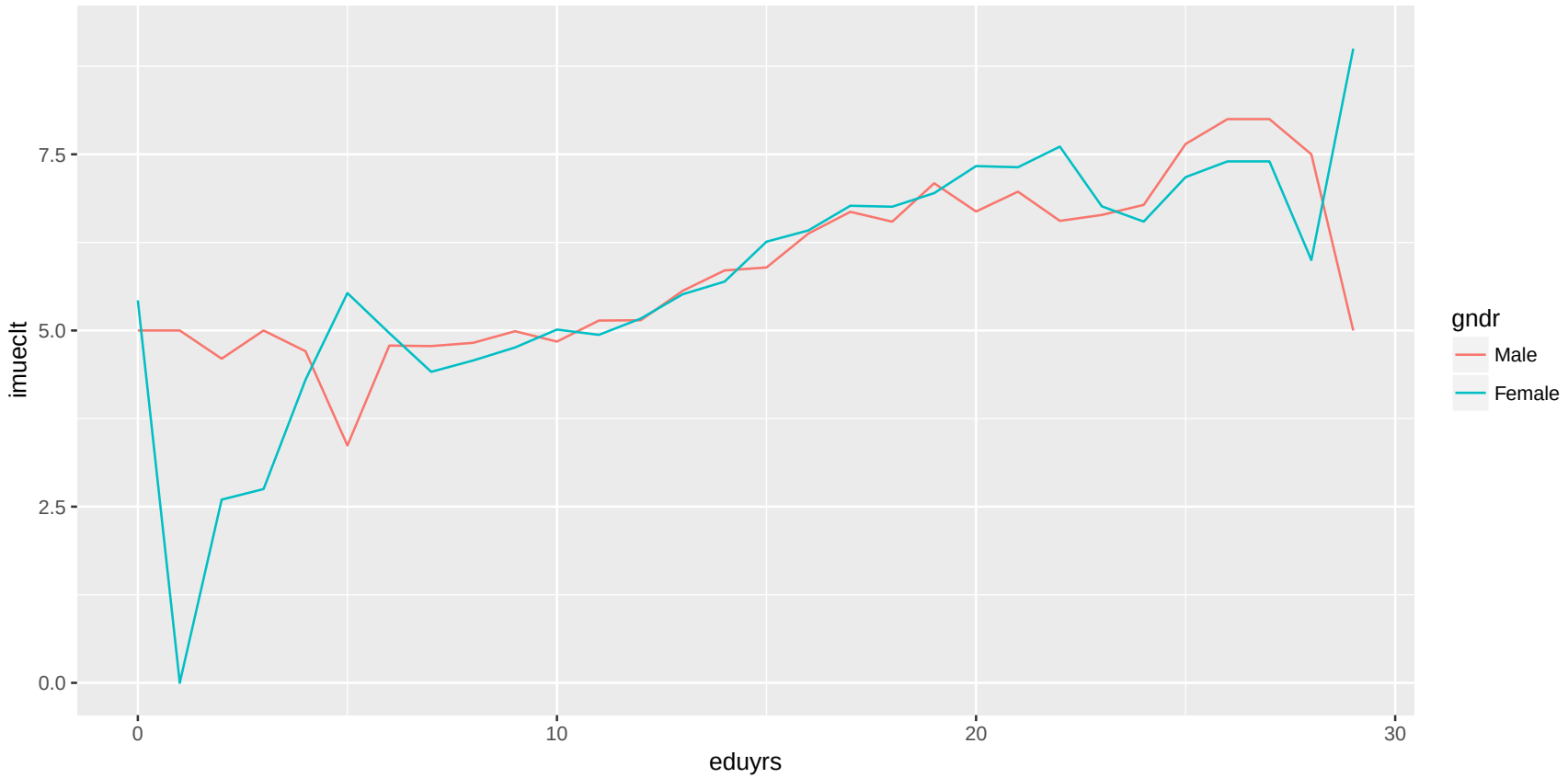
Groupement par mapping

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



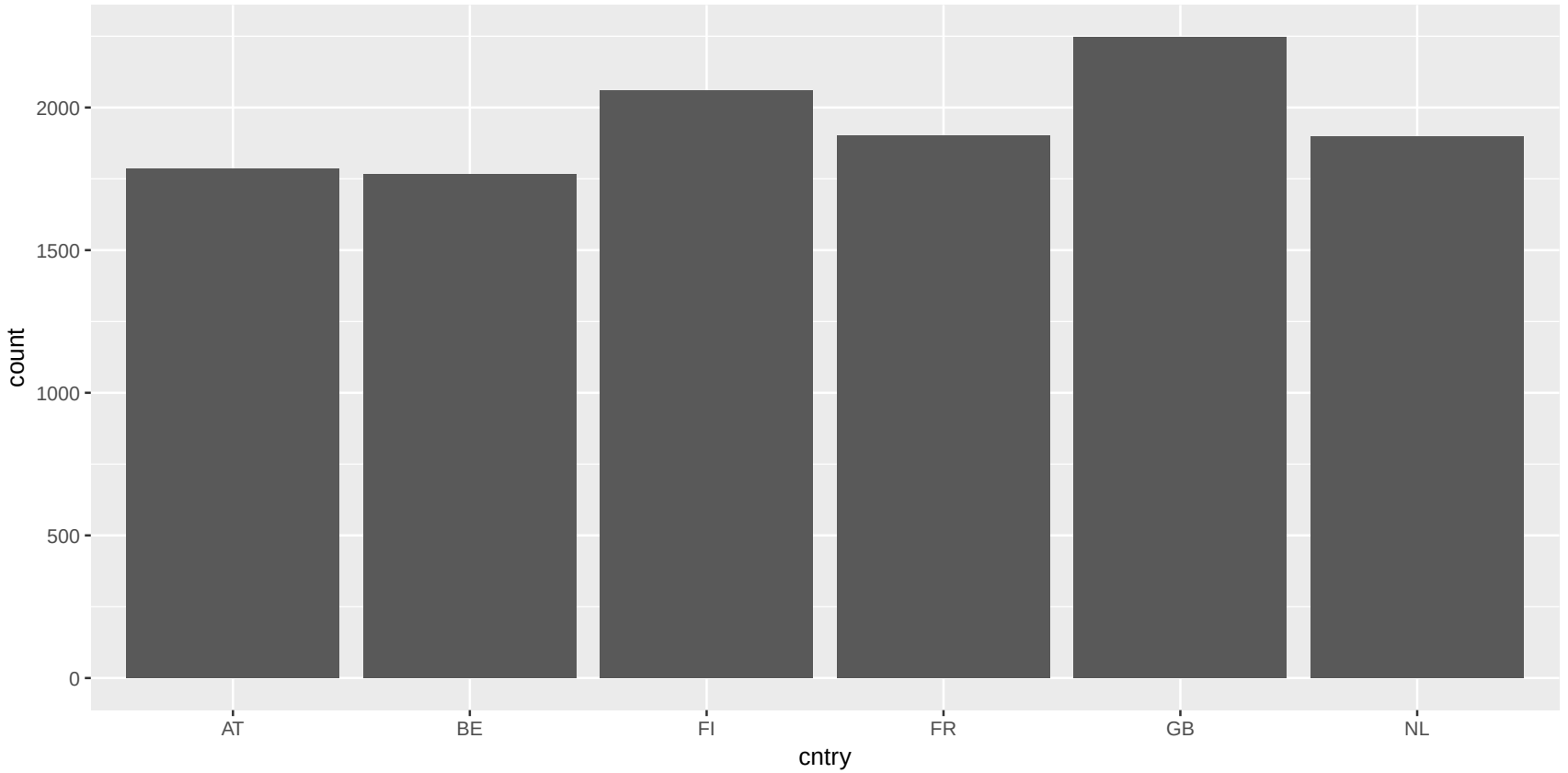
Groupement par mapping

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



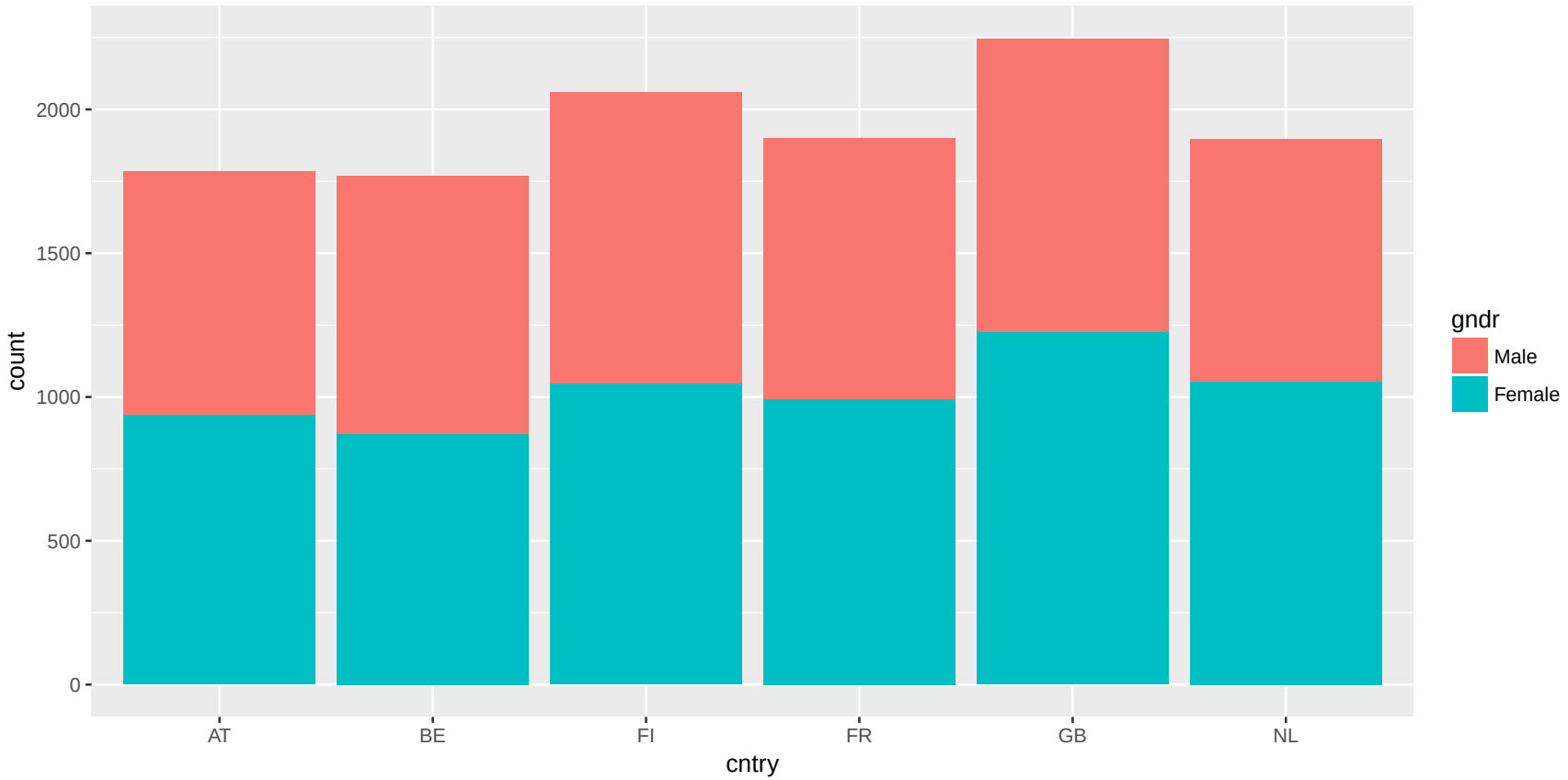
Groupement par mapping

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



Groupement par mapping

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



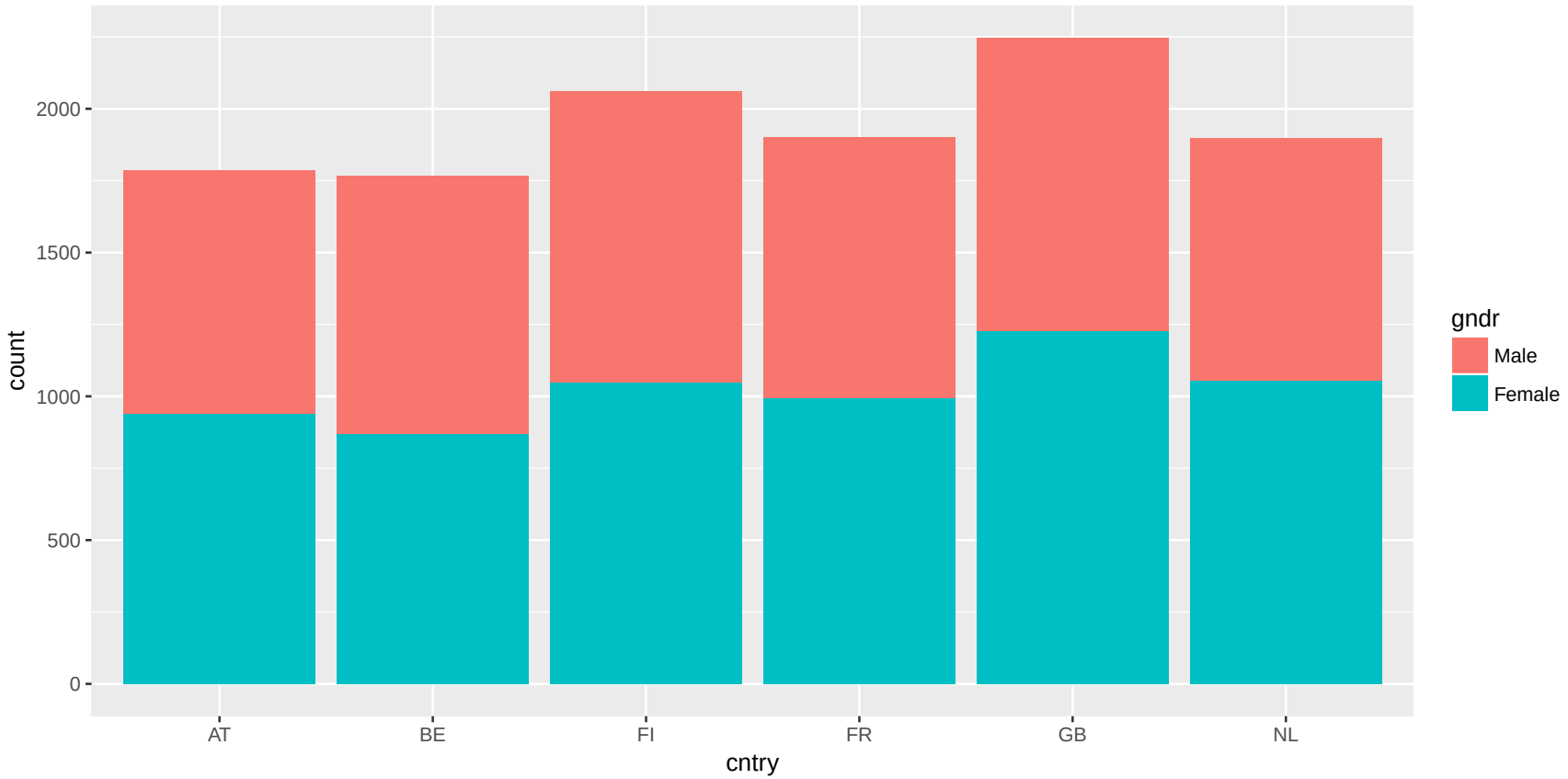
Positions

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.

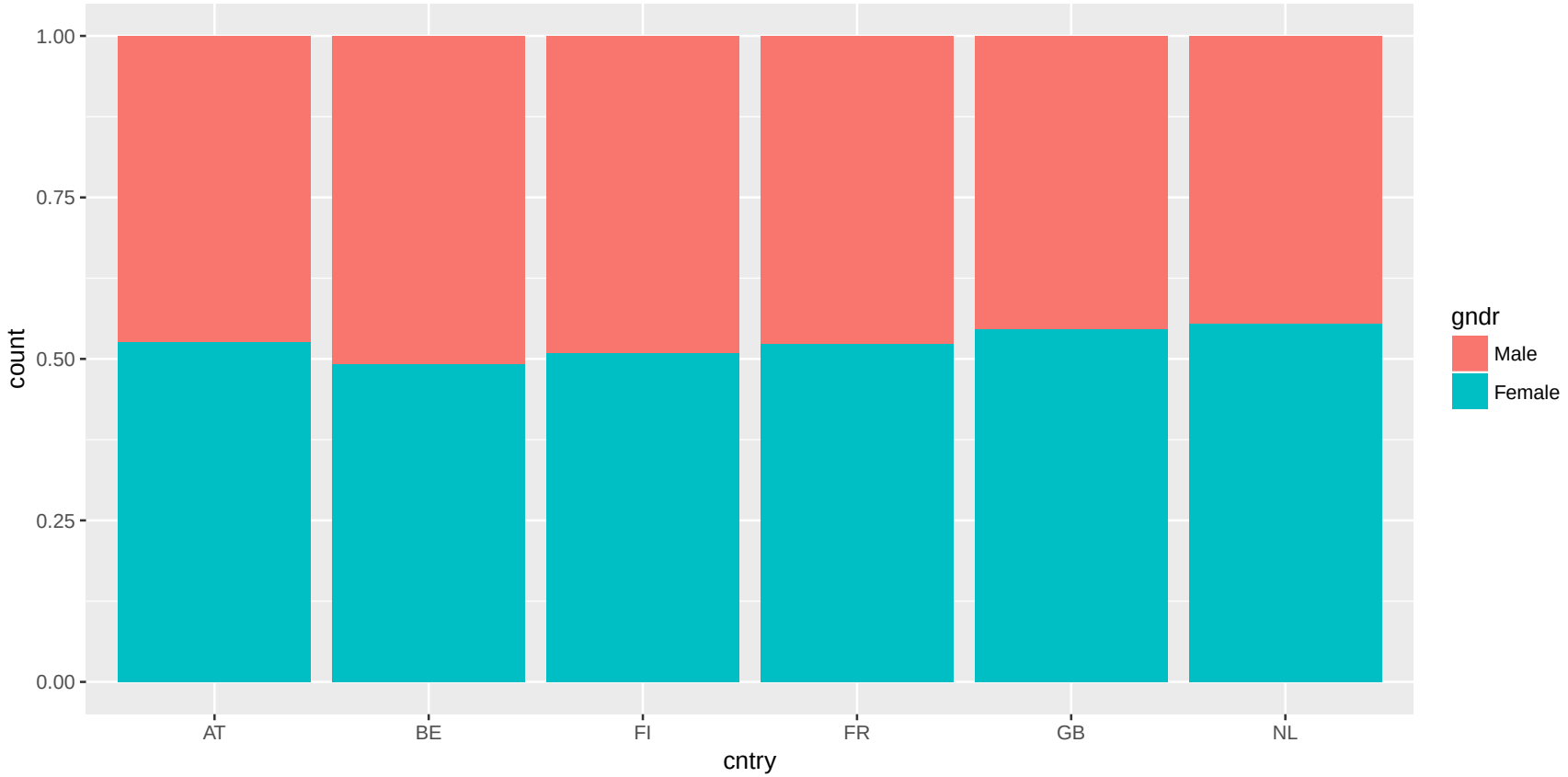
Positions

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



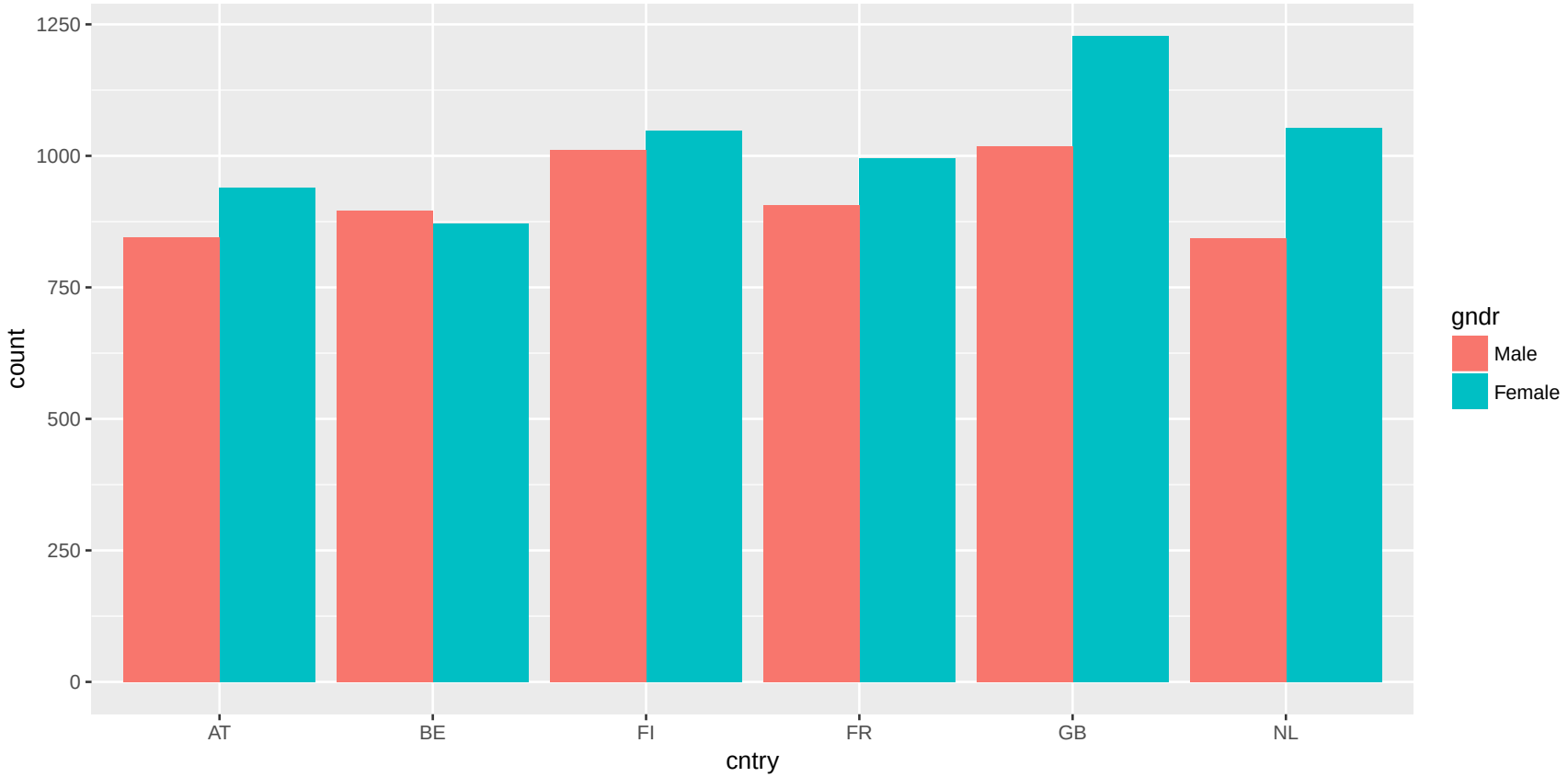
Positions

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



Positions

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

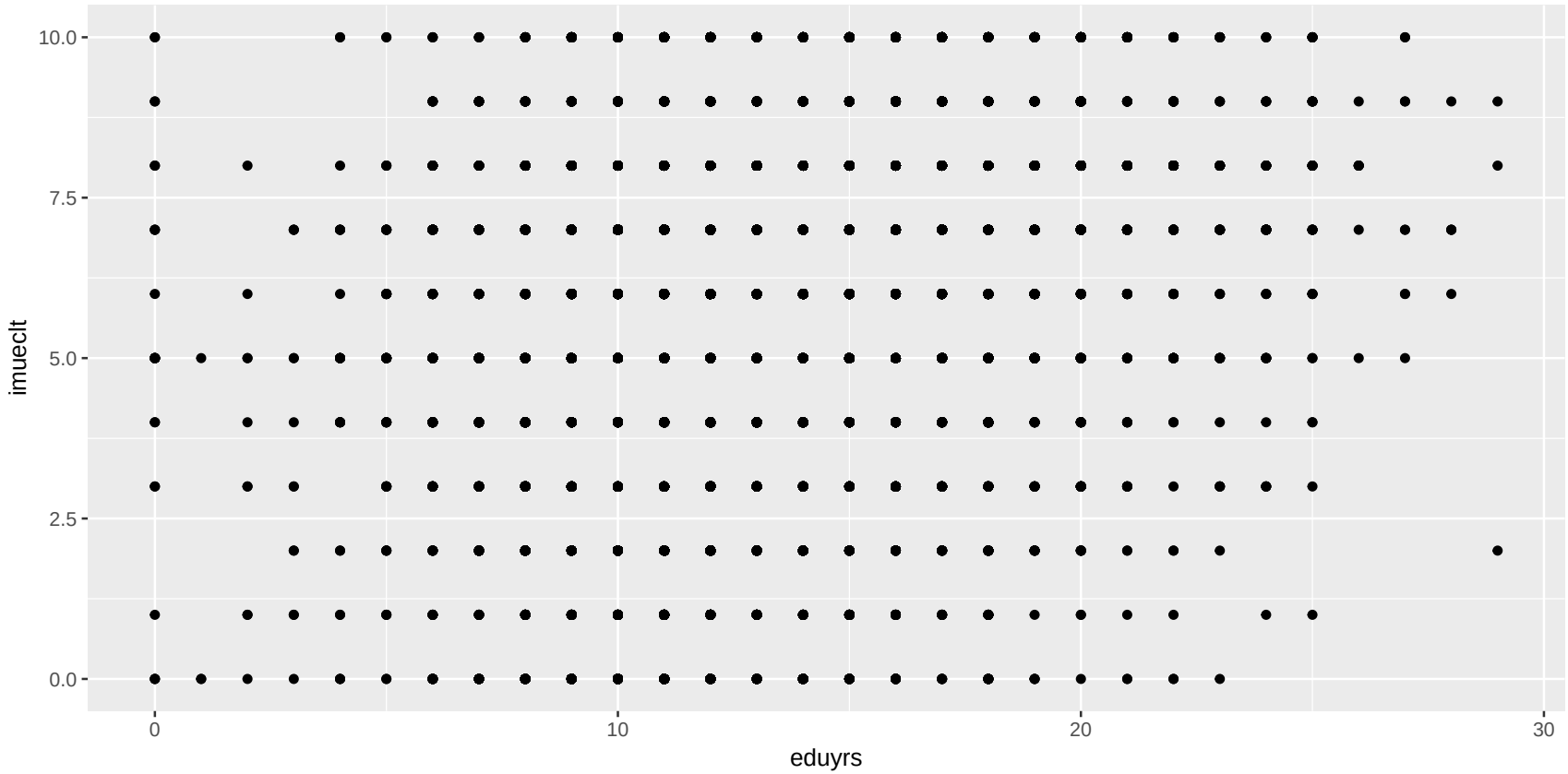


Positions

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

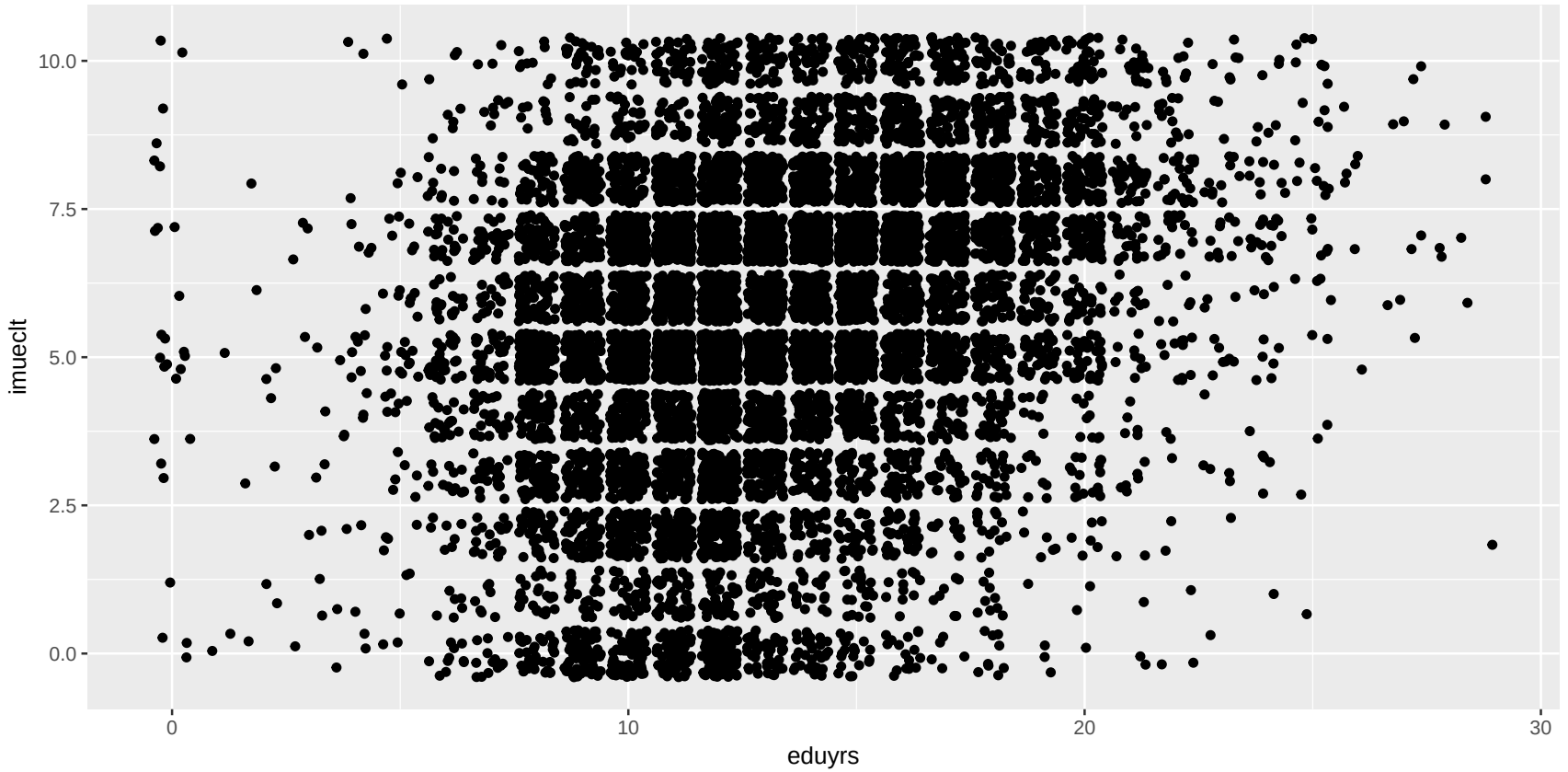
Positions: identity

```
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

stat_*

- 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)
  - soit on emploie `geom_line(stat=stat_summary())`

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.

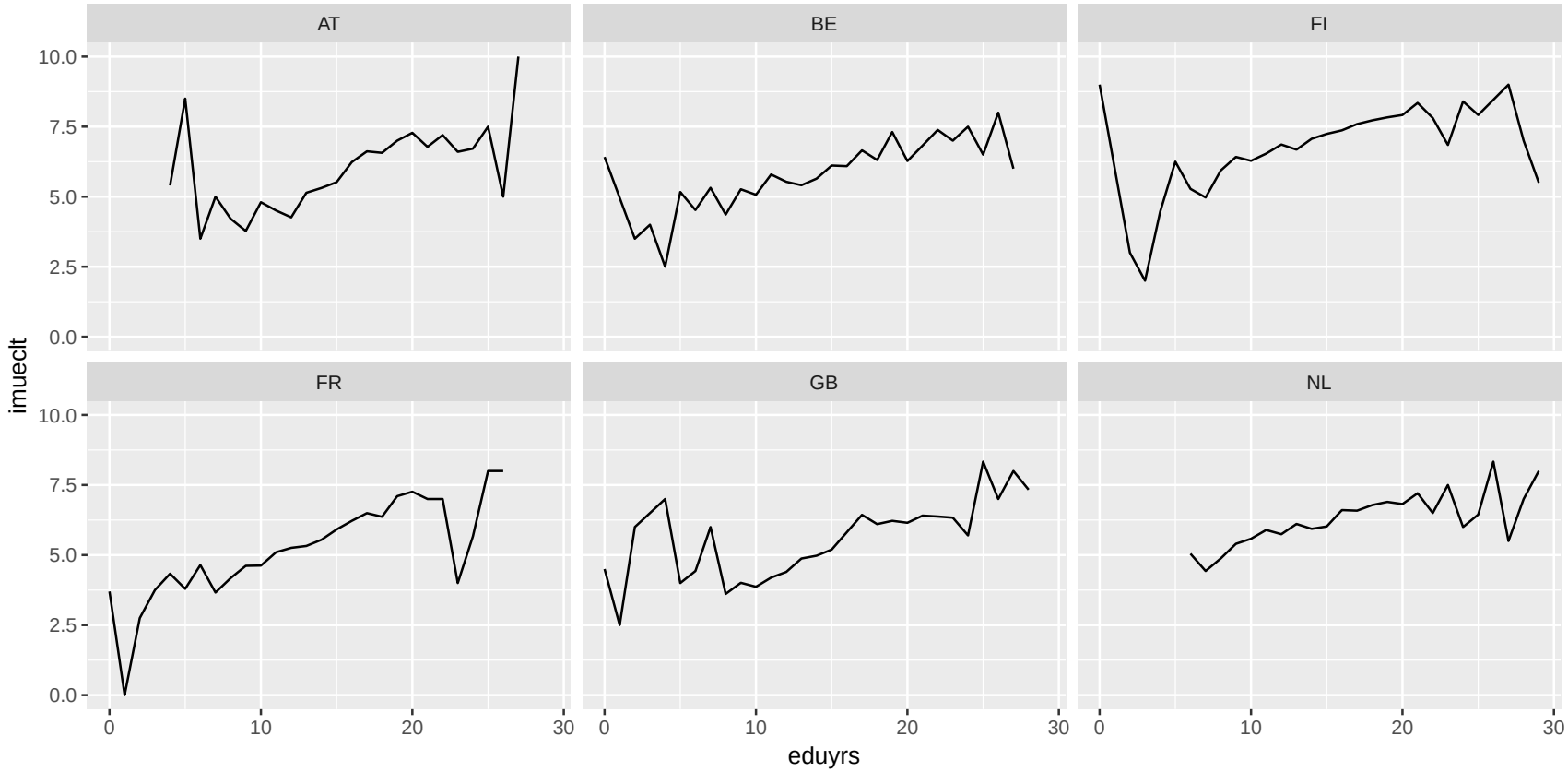
Facet: syntaxe

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

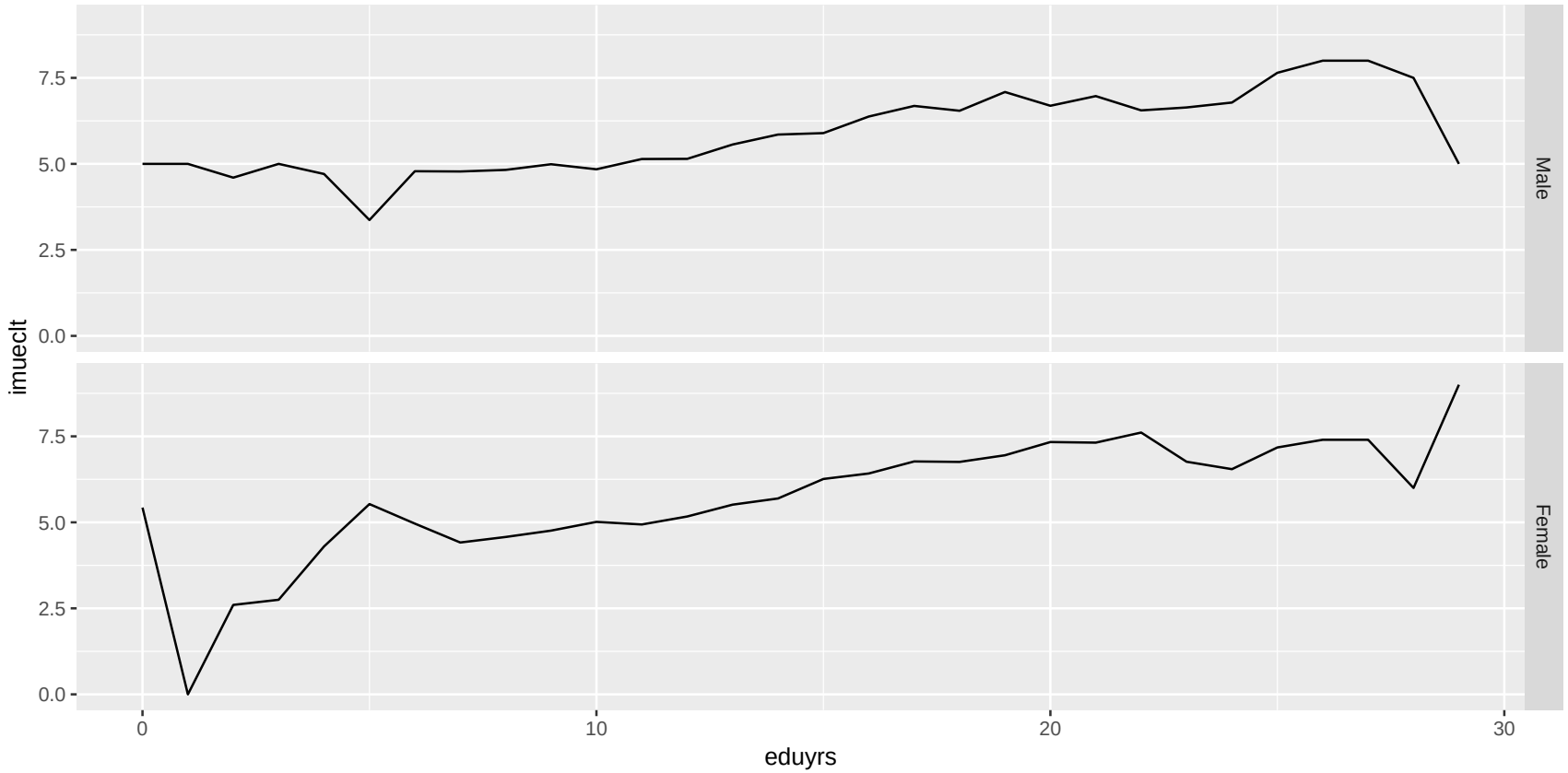
facet_wrap

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



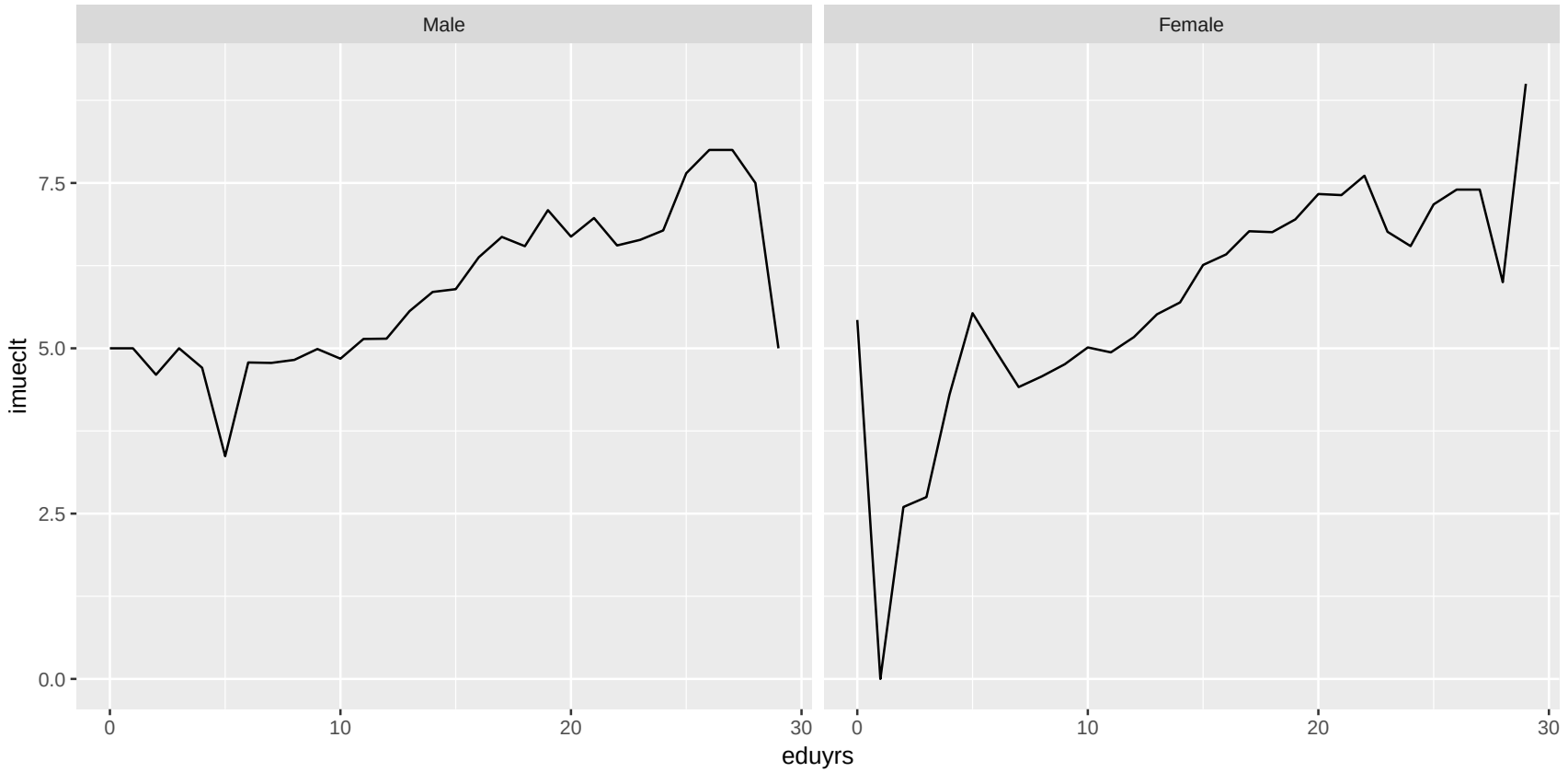
facet_grid: une seule variable en lignes

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



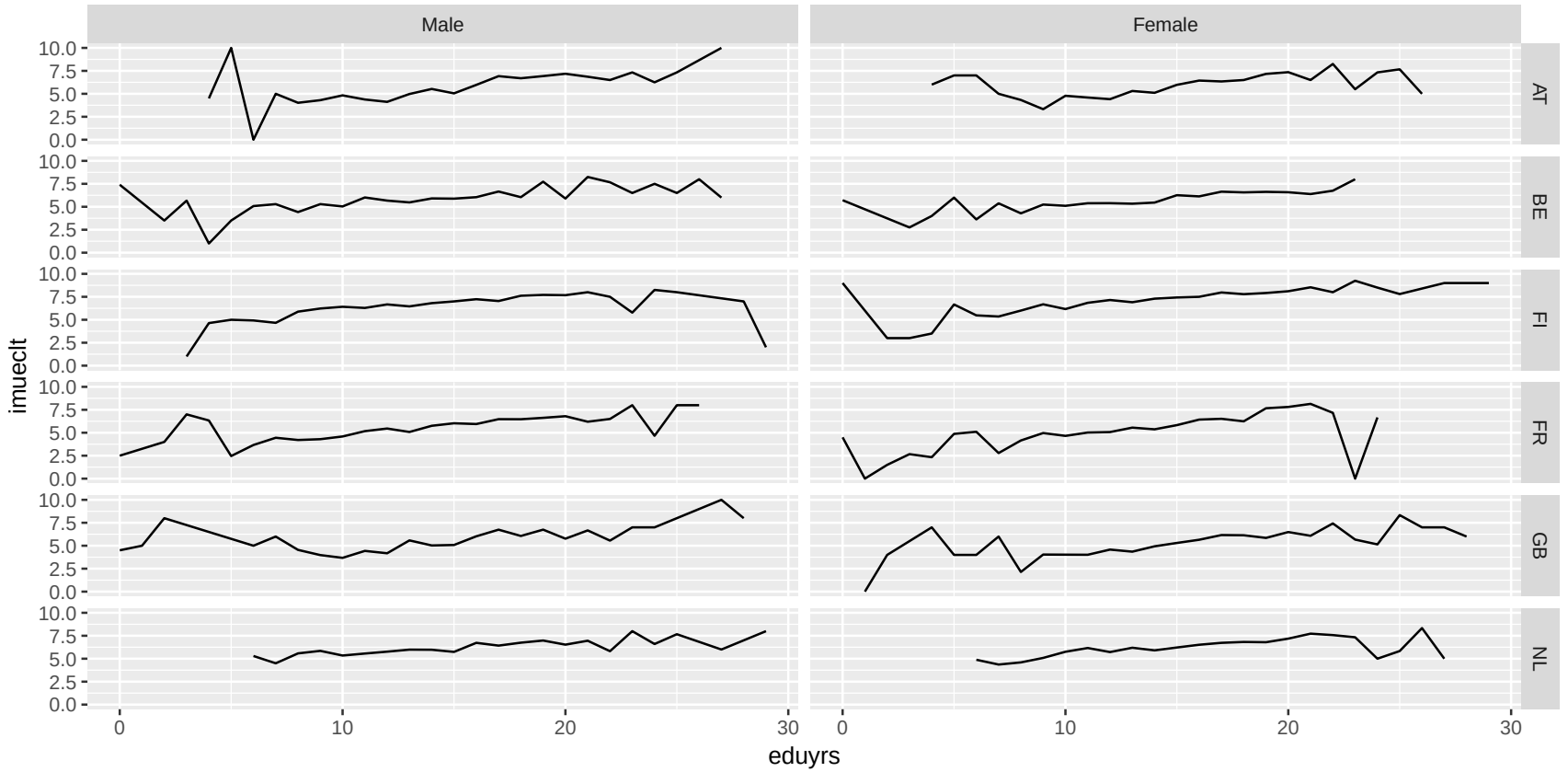
facet_grid: une seule variable en colonnes

```
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`).

tidyr: principe

- 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 base large

ID	Carac	Var2
1	A	X
2	B	Y

Exemple de base longue

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

Exemple de base longue avec identifiant

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)
```

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

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)
```

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

Exemple

Dans cet exemple, on souhaite préparer les données pour faire un graphique de toutes les variables qfi par cntry. Faire un graphique par variable et par pays revient à produire 36 graphiques. Comme on souhaite représenter, pour chaque variable, uniquement la moyenne et l'erreur standard, de façon à pouvoir les comparer dans chaque pays, on peut réduire cela à six graphiques, un par pays.

Exemple

On commence par sélectionner les seules variables d'intérêt avec `select()`. Nous allons avoir besoin de faire des groupes pays+variable (des calculs différents par chaque association de pays+variable). Pour pouvoir faire cela, il nous faut une colonne pays et une colonne variable. On emploie donc d'abord gather().

```
select(d, starts_with("qfi"), cntry) %>%
  gather(var, val, -cntry) %>%
  group_by(cntry, val)
```

```
## Source: local data frame [69,936 x 3]
## Groups: cntry, val [72]
##
##      cntry      var   val
##     <fctr>    <chr> <dbl>
```

Exemple

Le data.frame résultant a des valeurs manquantes dans val (les valeurs non renseignés des variables qfi). On les supprime avec filter

```
select(d, starts_with("qfi"), cntry) %>%
  gather(var, val, -cntry) %>%
  filter(!is.na(val))
```

```
## # A tibble: 69,496 × 3
##     cntry     var   val
##    <fctr>   <chr> <dbl>
## 1      AT qfimedu    10
## 2      AT qfimedu    10
## 3      AT qfimedu     5
## 4      AT qfimedu     5
## 5      AT qfimedu     7
## 6      AT qfimedu     6
```

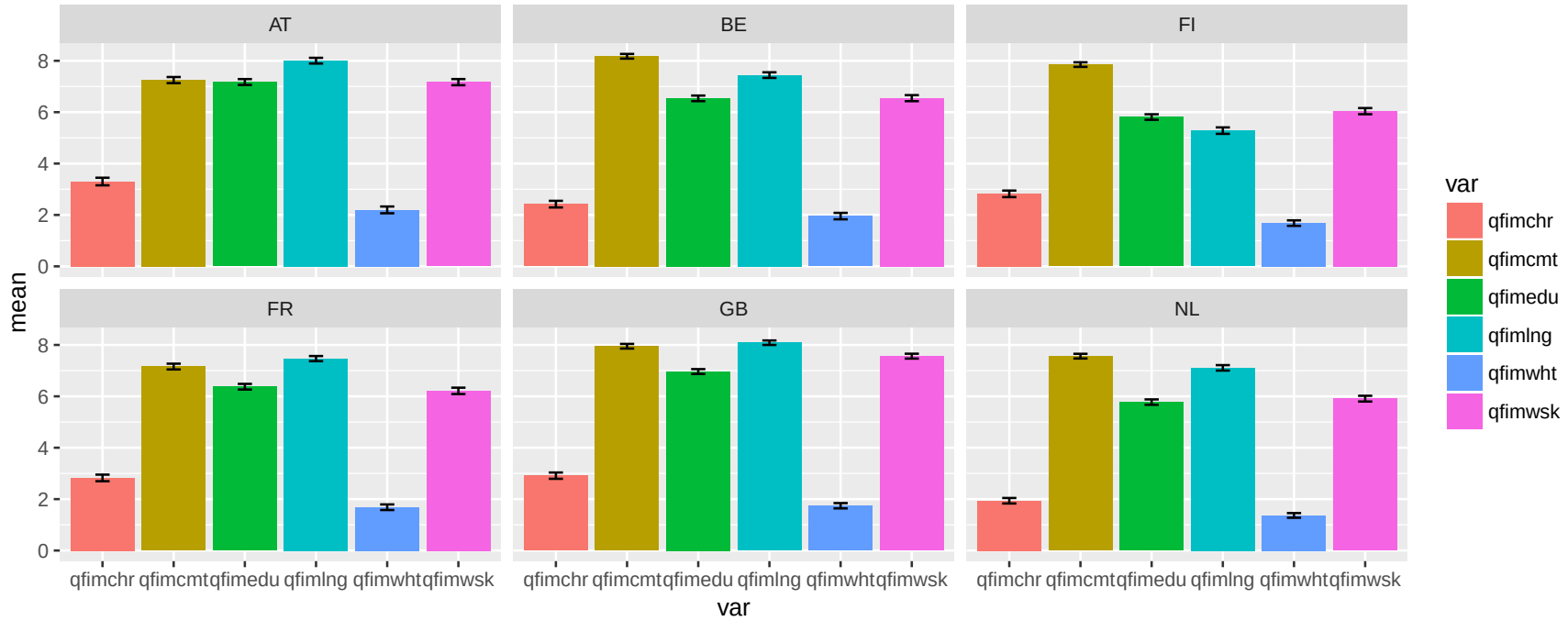
Exemple

On peut désormais grouper les observation par cntry et var et calculer à chaque fois moyenne et erreur-type (pour rappel, pour un intervalle de confiance à .95, $1.96 * \frac{\sigma_x}{\sqrt{n_x}}$). On stocke le résultat dans un nouvel objet.

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

Exemple

```
ggplot(dt, aes(x = var, y = mean, fill = var)) +
  geom_col() + facet_wrap(~ cntry) +
  geom_errorbar(aes(ymin = mean-se, ymax = mean+se),
                width=.2)
```



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

Scales

Scales

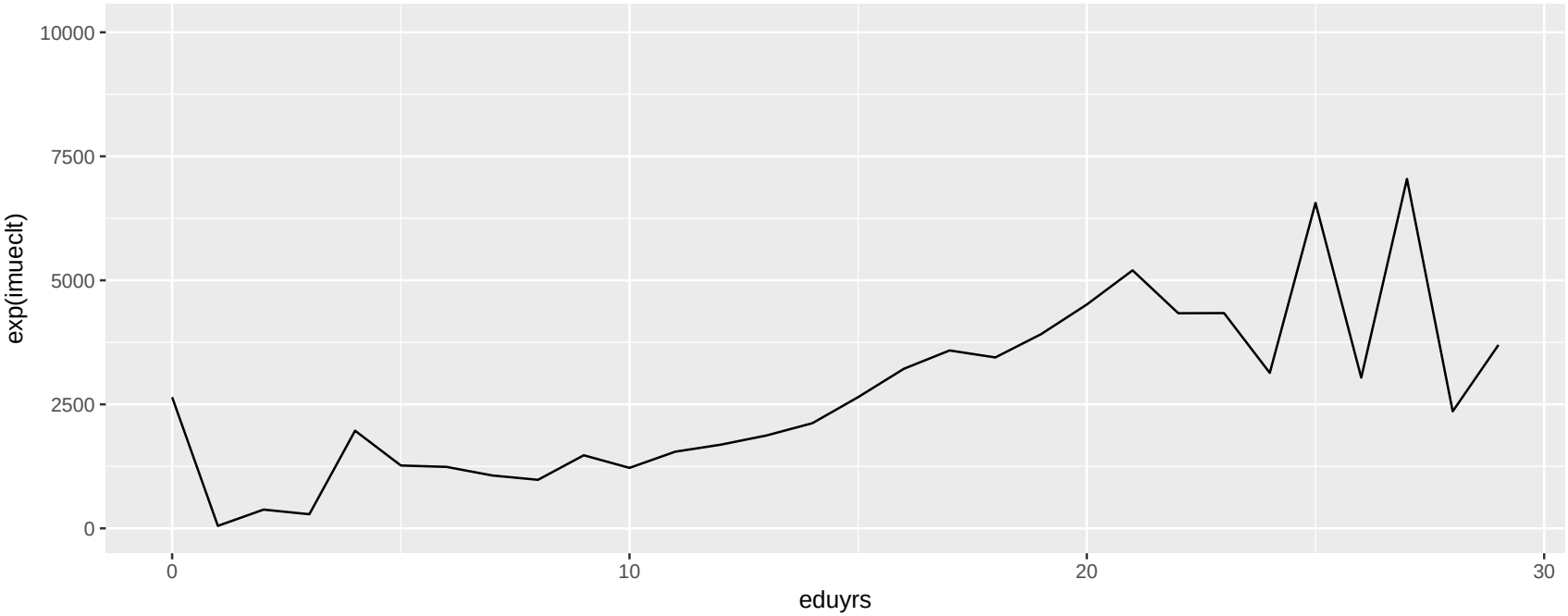
Les échelles (scales) sont des couches de ggplot qui précisent les transformations à appliquer aux axes, ainsi que les valeurs à indiquer. de ce fait, on les emploie à la fois :

- pour opérer une transformation (échelle logarithmique par exemple)
- pour choisir la numérotation d'axes continus
- pour choisir les couleurs

Toutes les fonctions de scales suivent le même modèle: scale_mapping_trans où mapping désigne la place de la variable (x, y, fill, color, etc.) et trans le type d'échelle à appliquer (continuous, discrete, log, etc.)

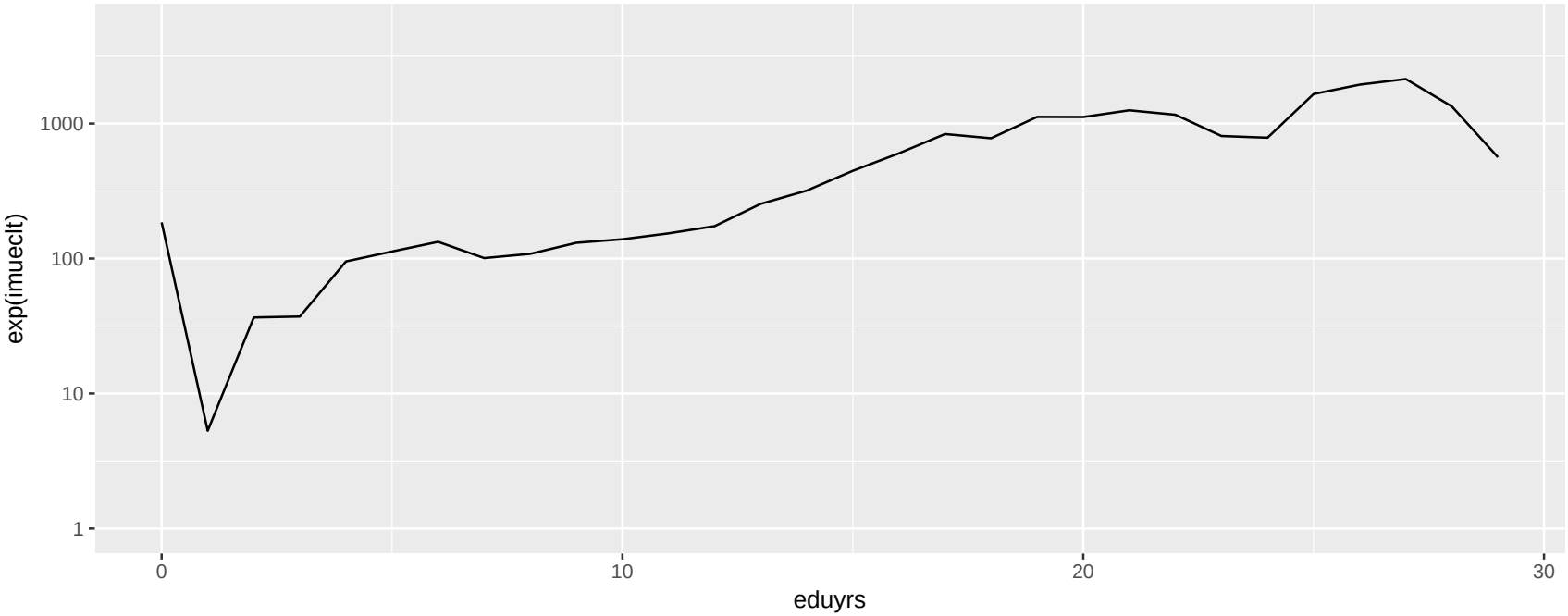
Transformer une échelle

```
p <- ggplot(d, aes(y = exp(imueclt), x = eduyrs)) +
  geom_line(stat="summary")
p
```



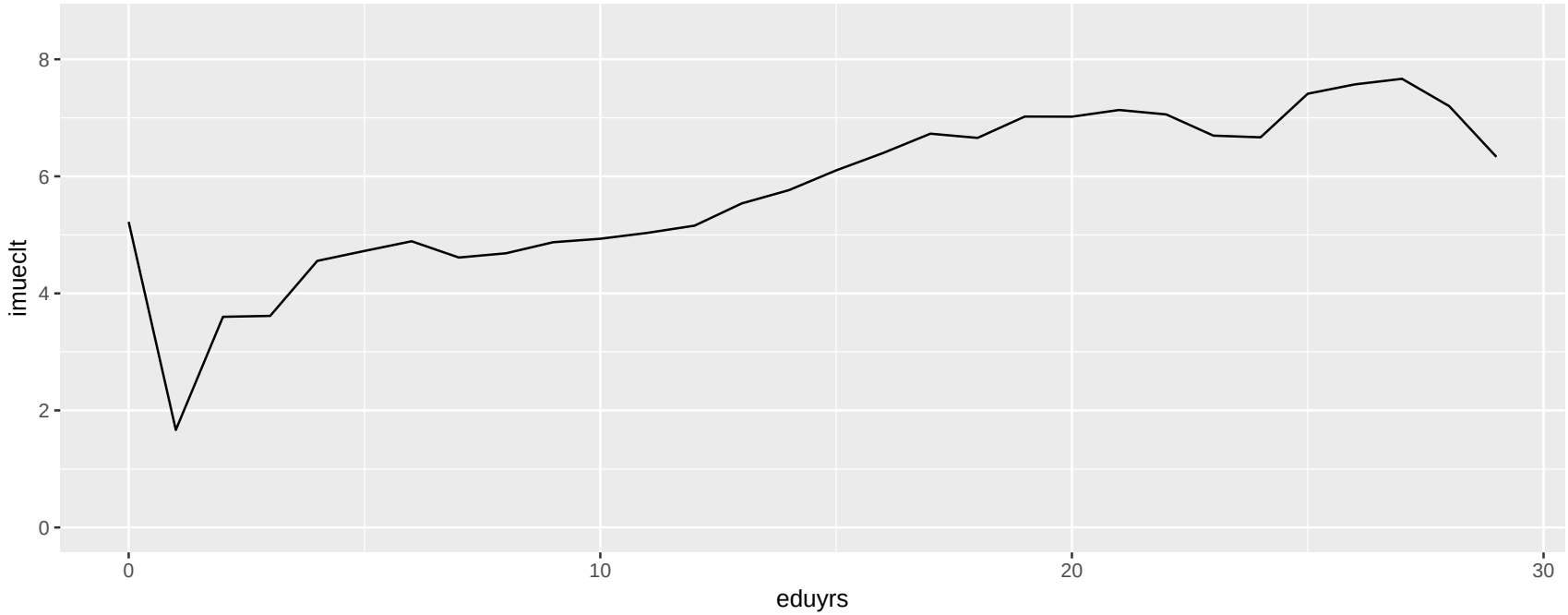
Transformer une échelle

```
p + scale_y_continuous(trans = "log",
                       breaks = c(1, 10, 100, 1000))
```



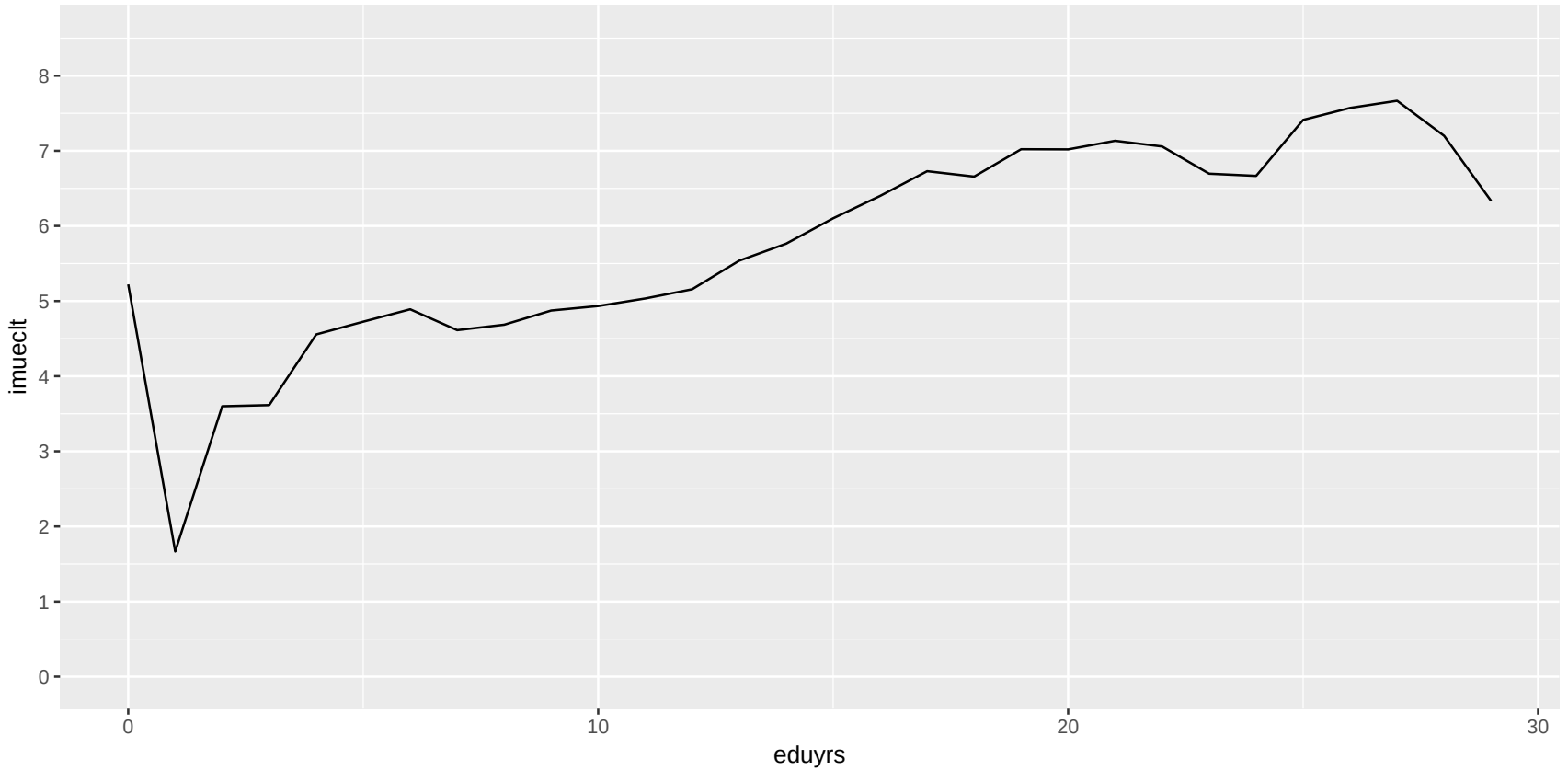
Renuméroter des axes

```
p <- ggplot(d, aes(y = imueclt, x = eduyrs)) +
  geom_line(stat="summary")
p
```



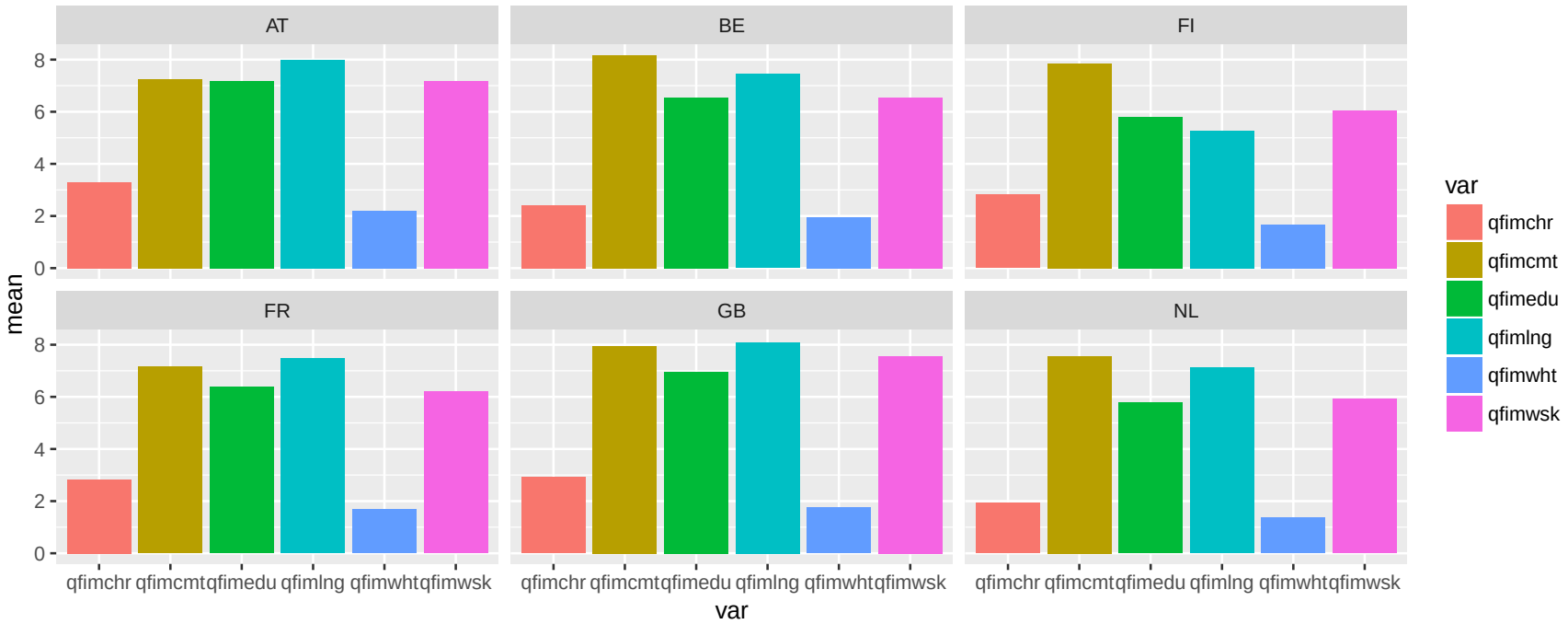
Renuméroter des axes

```
p + scale_y_continuous(breaks = 0:8)
```



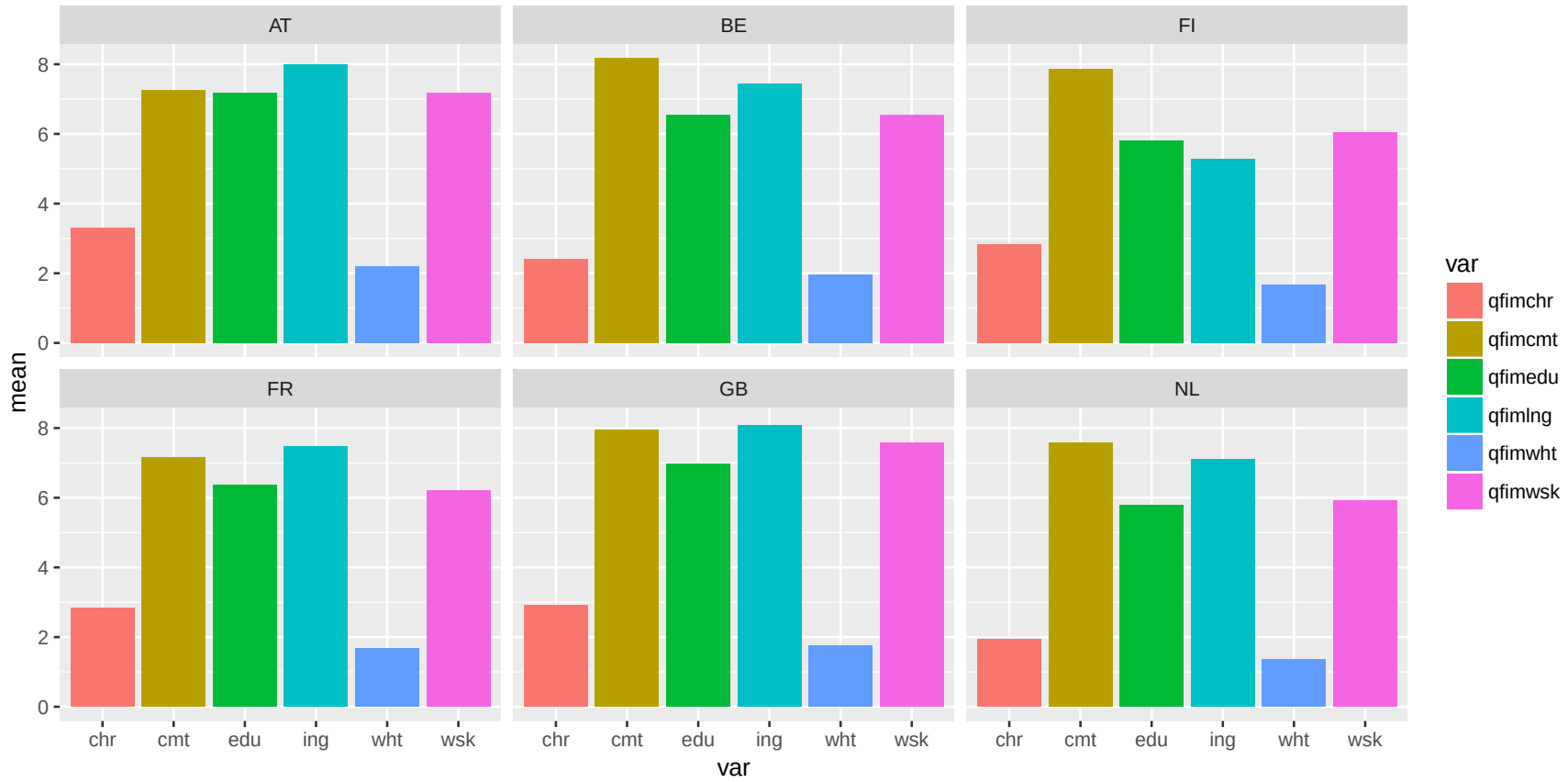
Changer des valeurs sur des axes discrets

```
p <- ggplot(dt, aes(x = var, y = mean, fill = var)) +
  geom_col() + facet_wrap(~ cntry)
p
```



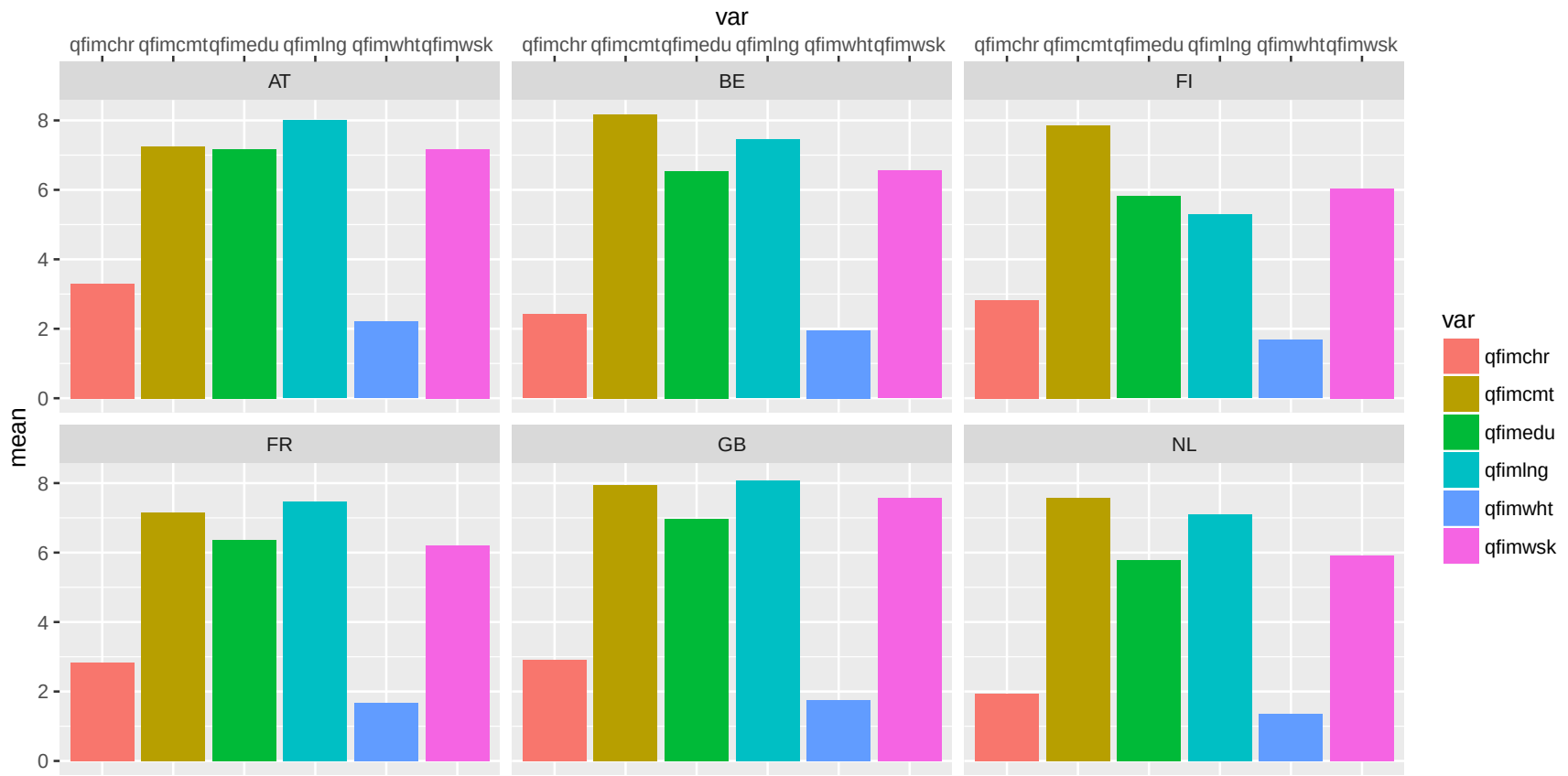
Changer des valeurs sur des axes discrets

```
p + scale_x_discrete(labels = c("chr", "cmt", "edu",
                                "ing", "wht", "wsk"))
```



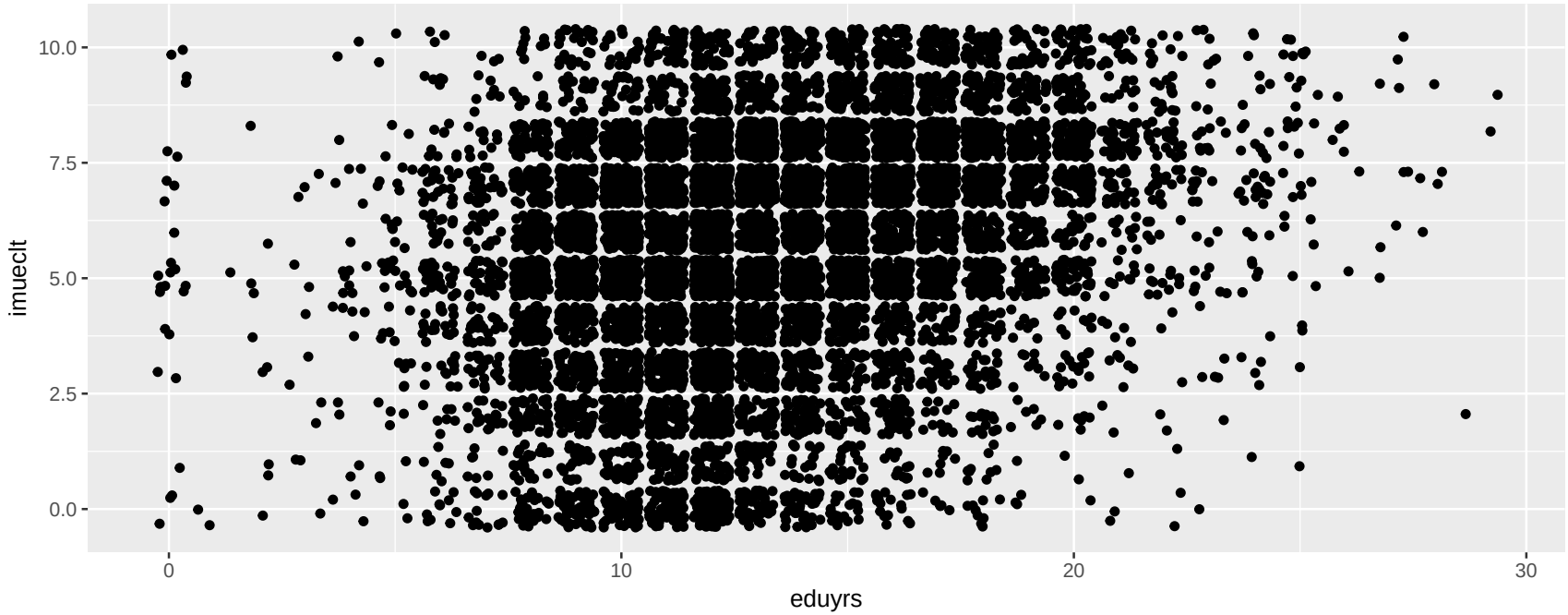
Modifier le placement d'un axe

```
p + scale_x_discrete(position = "top")
```



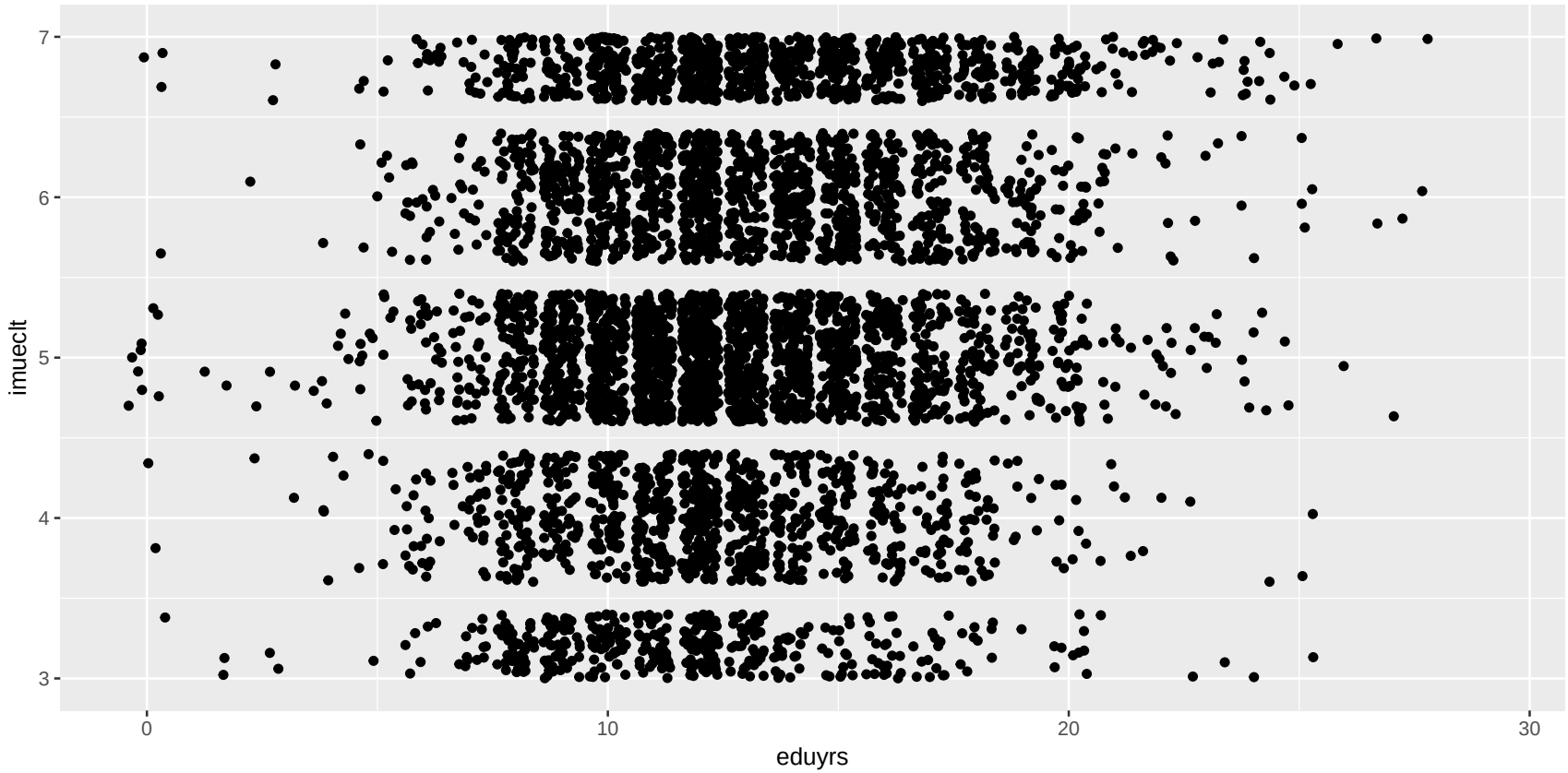
Zoom: limiter un axe

```
pp <- ggplot(data = d, mapping = aes(x = eduyrs, y = imuecl
  geom_point(position = "jitter")
pp
```



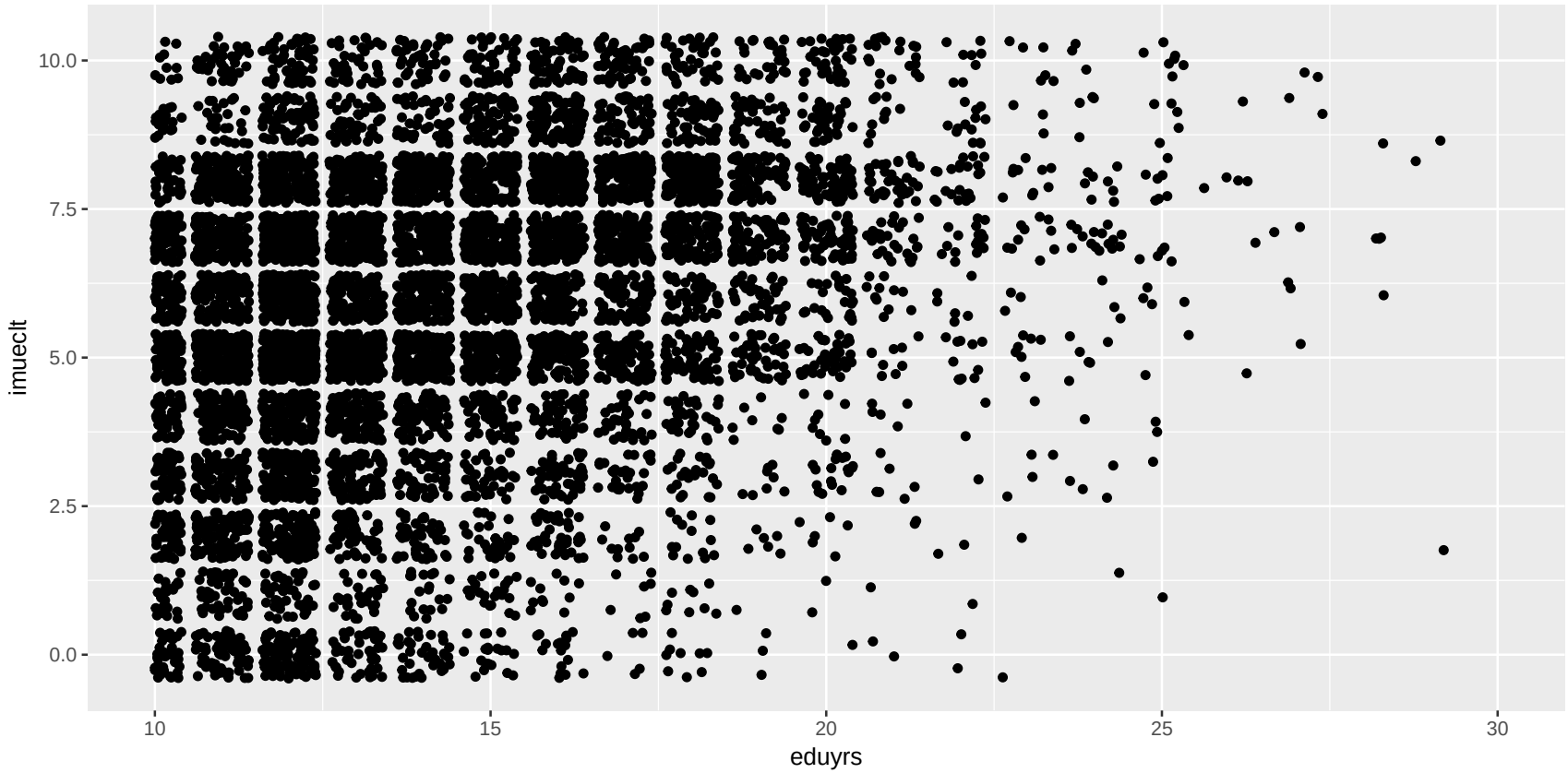
Zoom: limiter un axe

```
pp  + ylim(3, 7)
```



Zoom: limiter un axe

```
pp  + xlim(10, 30)
```



Changer les couleurs

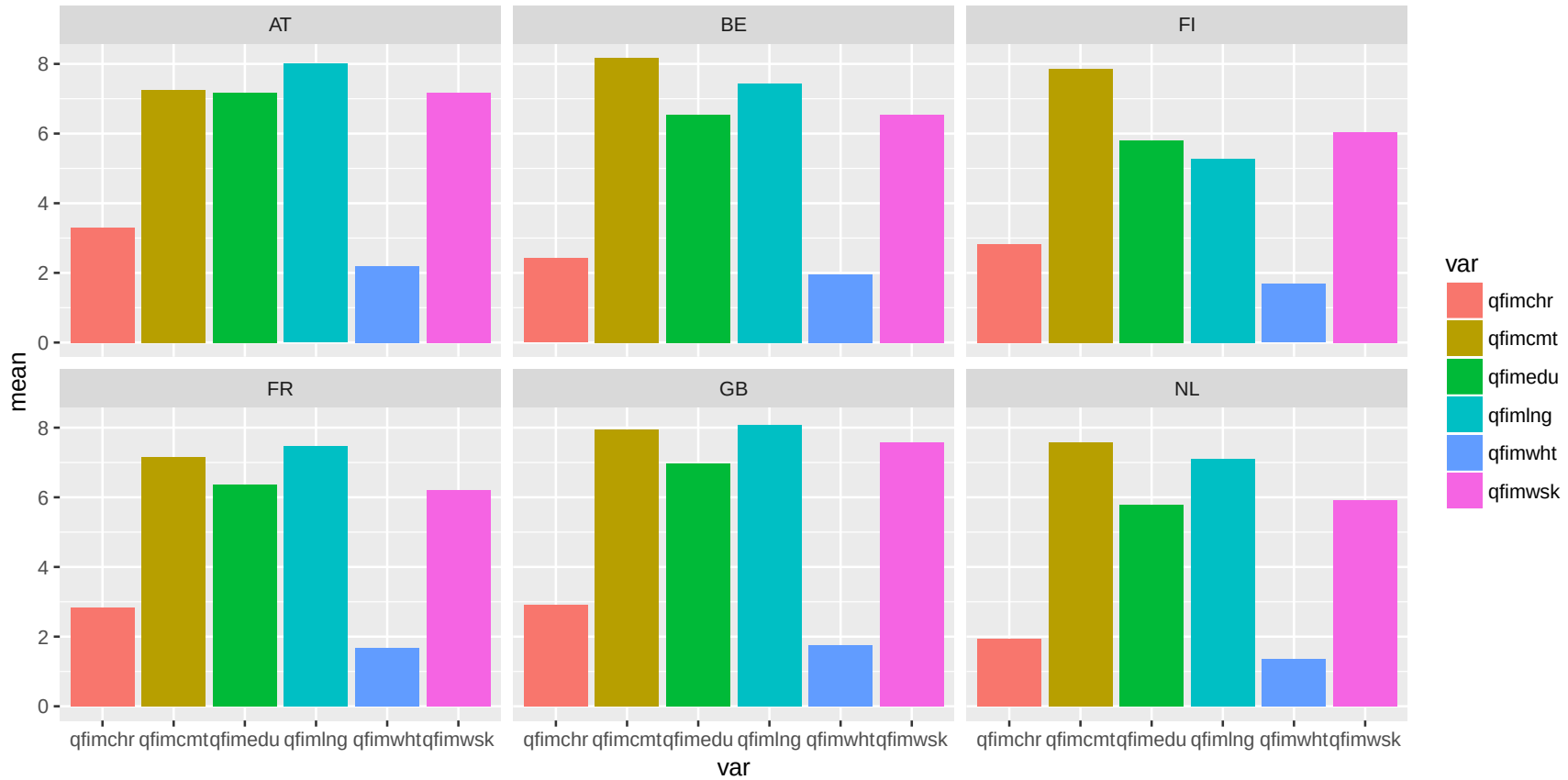
Il est possible de spécifier des couleurs à la main, mais la solution préférable est d'utiliser colorbrewer et son implémentation R.

Pour choisir un set de couleurs, aller sur `http://colorbrewer2.org`.

Adapter votre set: continu pour une variable quantitative ou ordonnée, qualitative pour une variable catégorielle.

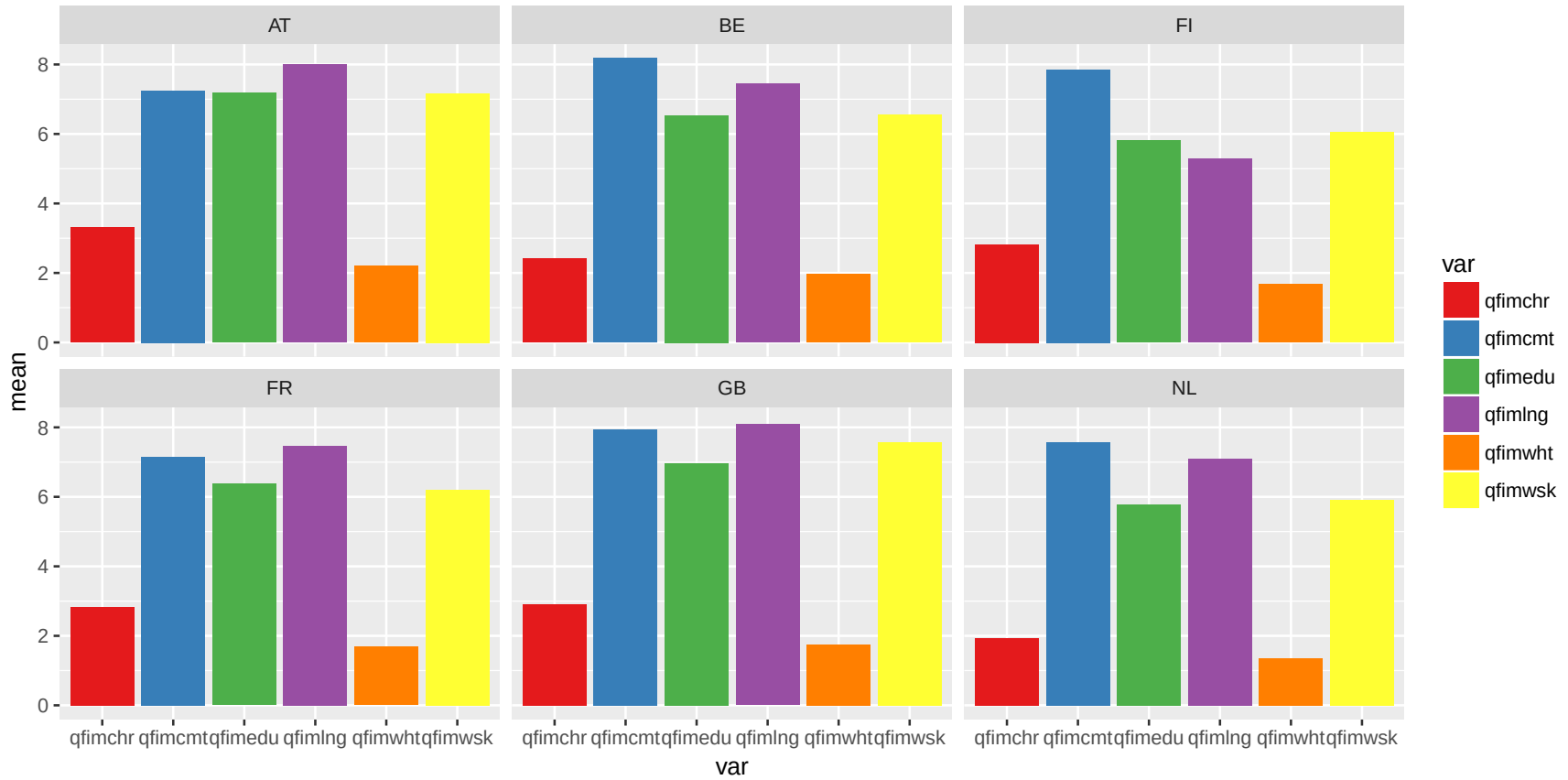
Changer les couleurs

```
p
```



Changer les couleurs

```
p + scale_fill_brewer(palette = "Set1")
```



Themes et titres

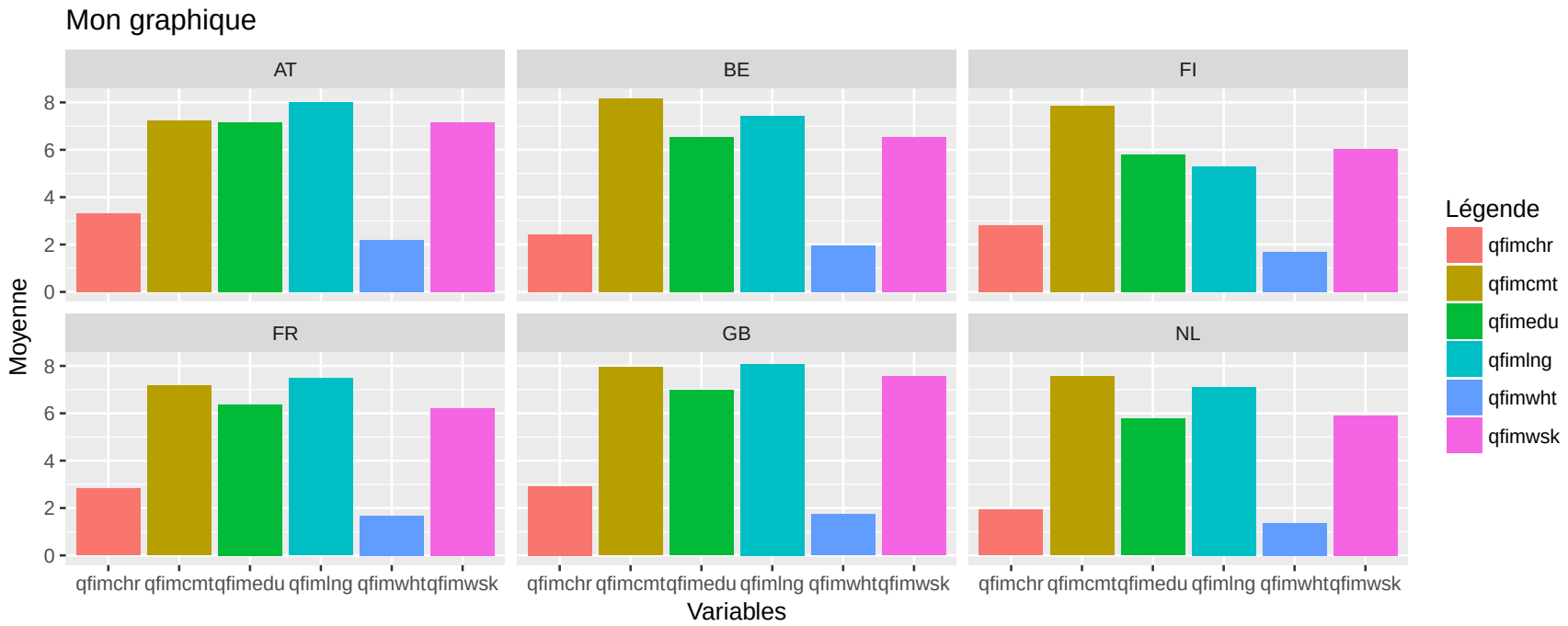
Titres

La fonction `labs()` permet de donner des titres au graphique et aux axes. Elle prend pour arguments les noms des différents mappings: `x` pour donner le label de l'axe des abcisses, `y` pour celui de l'axe des ordonnées, `colour` pour le label de la variable déterminant les groupes de couleurs, etc.

On peut également préciser le titre (`title`) et le sous-titre (`subtitle`) du graphique. Il est déconseillé de le faire lorsque l'on produit un document écrit (le titre inclu directement dans le graph n'est pas cherchable $=>$ mieux vaut une légende spécifiée dans le langage employé pour la rédaction), mais cela peut-être utile pour des images isolées et/ou dans des slides.

Titres

```
p + labs(x = "Variables", y = "Moyenne",
         fill = "Légende",
         title = "Mon graphique")
```



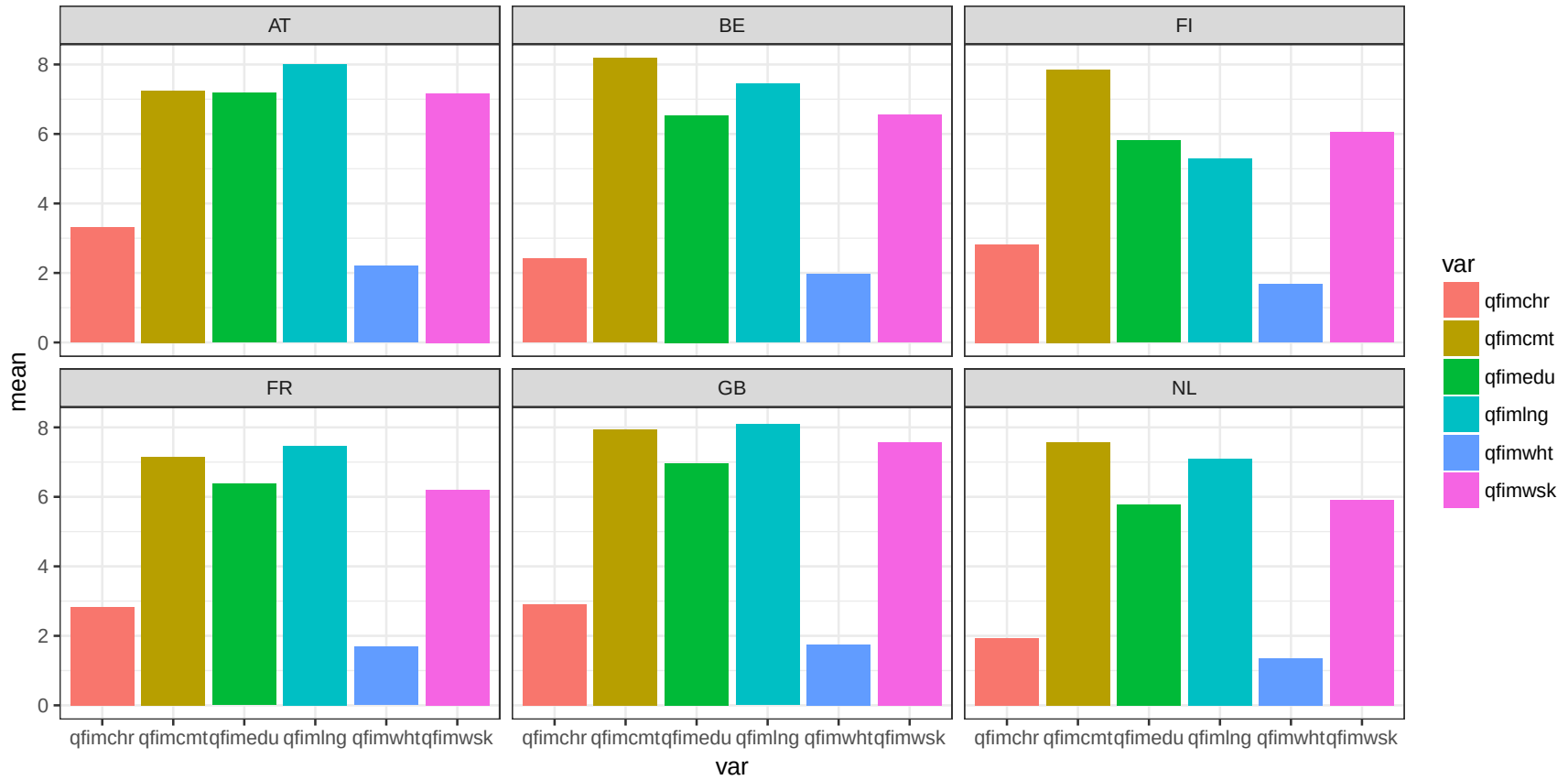
Thèmes

Les thèmes contiennent un ensemble d'instructions permettant de contrôler finement l'apparence du graphique. ggplot2 contient un ensemble de thèmes définis par défaut, la famille `theme_*()`.

Le thème par défaut, employé si vous n'en spécifiez aucun autre, est `theme_grey()`. Il possède un fond gris avec des traits blancs pour les séparateurs sur les deux axes. Un autre thème populaire est `theme_bw()`, avec un fond blanc, mais toutes les autres avancées de ggplot.

Black & white

```
p + theme_bw()
```

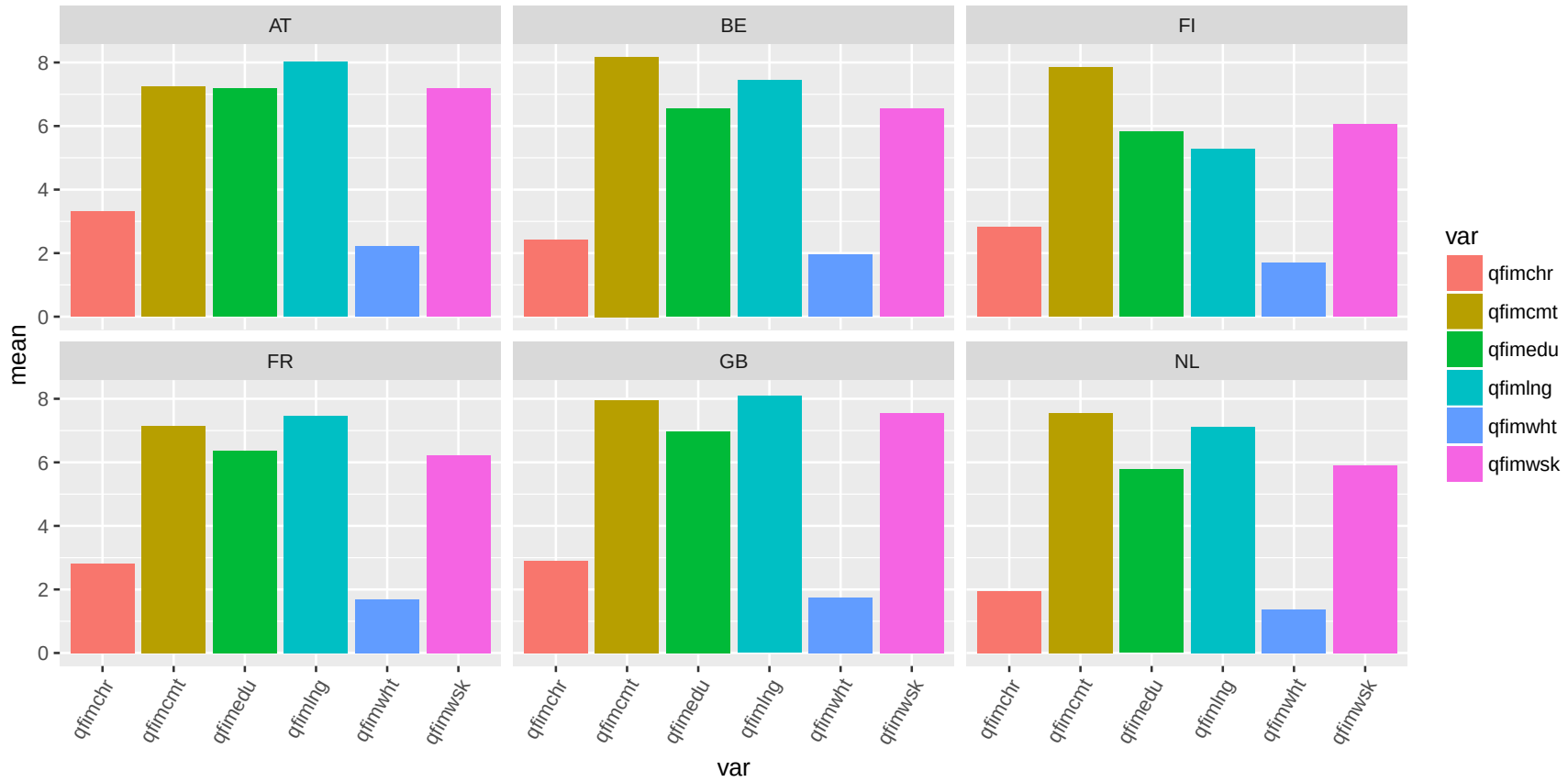


Theme: personnaliser

On peut par ailleurs personnaliser ces thèmes. On emploie pour cela la fonction `theme()`. `?theme` donne la liste de tout ce qu'il est possible de personnaliser.

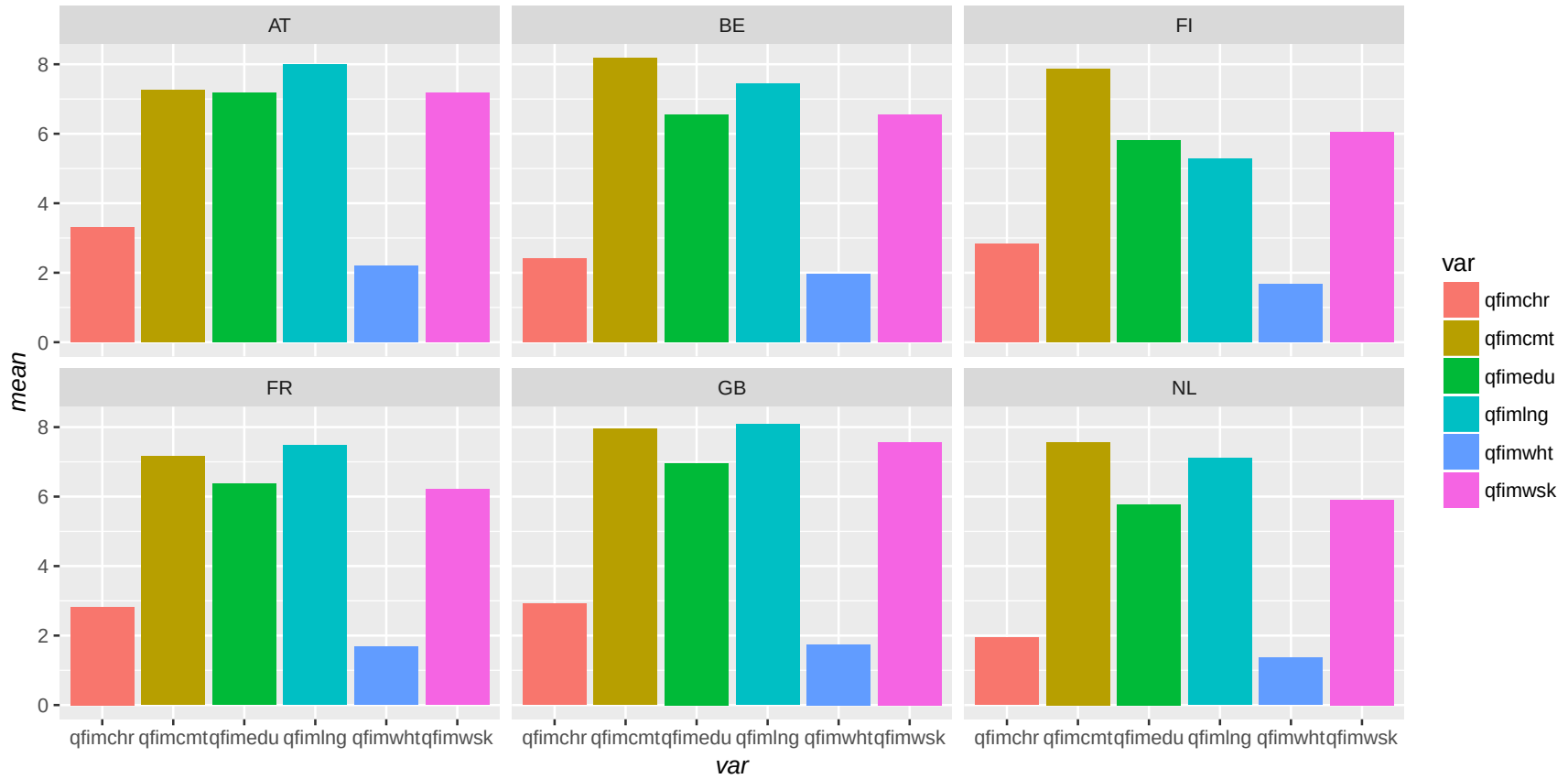
Labels des axes

```
p + theme(axis.text.x =
            element_text(angle = 60, hjust = 1))
```



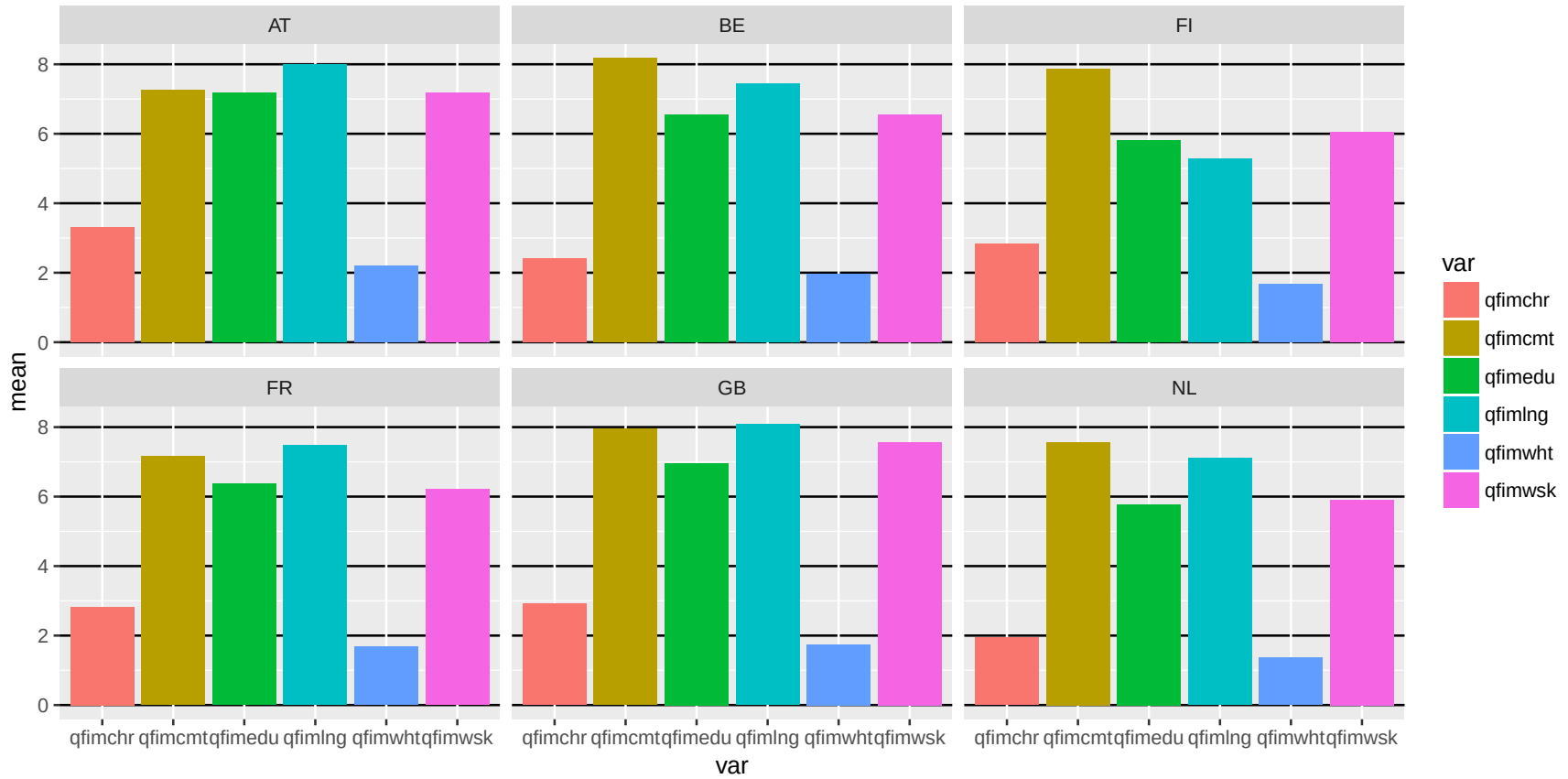
Labels des axes

```
p + theme(axis.title =
            element_text(face = "italic"))
```



Grille d'arrière plan

```
p + theme(panel.grid.major.y =
            element_line(color="black"))
```



Un plot complet

```
# on part des données transformées, dt
p <- ggplot(dt, aes(x = var, y = mean, fill = var)) +
  geom_col() +
  facet_wrap(~ cntry) +
  scale_fill_brewer(palette = "Set1") +
  theme_bw() +
  theme(axis.text.x = element_text(angle=60, hjust=1),
        panel.grid.major.y = element_line(color="black"),
        panel.grid.major.x = element_blank()) +
  labs(y = "Moyenne", fill = "Variables", x = "Variables")
```

Un plot complet

```
p
```