Séance 3 : Graphiques

Introduction à la sociologie quantitative, niveau 2

Samuel Coavoux

R-base et ggplot

Trois systèmes graphiques principaux dans R :

- R-base : la fonction `plot()`
- lattice : le package `lattice`
- ggplot : le package `ggplot2` (qui est chargé par `library(tidyverse)` => pas besoin d'ajouter `library(ggplot2)`)

C'est ce dernier que nous allons utiliser.

Grammar of graphics

Le gg de ggplot signifie “Grammar of graphics”.

La théorie (Wilkinson 2005) ; la pratique (Wickham 2010)

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

Qu'est-ce qu'un graphique

Comment produire la représentation graphique sur un écran d'ordinateur d'un jeu de données ? Exemple tiré de Wickham (2010).

A	B	C
2	4	a
1	1	a
4	15	b
9	80	b

Qu'est-ce qu'un graphique

Partons d'un jeu de données de trois variables, deux numériques et une catégorielle. La façon la plus simple de représenter cela est un scatterplot. Concrétement, cela consiste à représenter A sur l'axe numérique x, B sur l'axe numérique y, et C par la forme des points. On appelle **aesthetic mapping** l'association d'une variable à une forme de représentation.

On peut vouloir **transformer** nos données. Par exemple, dans certains cas, nous aurons besoin d'agréger les données de base dans différents groupes (quand on fait un histogramme) ou calculer des valeurs résumées (quand on produit un boxplot). On appelle **transformation** ces opérations. Ici, nous appliquons la *identity transformation* = les données transformées sont identiques aux données brutes (pas de modifications).

Qu'est-ce qu'un graphique

Ensuite, on transforme les valeurs de ces variables en **coordonnées** graphiques. Pour un écran d'ordinateur, nous avons besoin de connaître les coordonnées du ou des pixels qui vont représenter chaque point. On transforme donc A et B en coordonnées sur l'écran. De la même façon, nous avons besoin de savoir quelles sont les formes associées à C (par exemple des cercles pour la valeur "a", des carrés pour la valeur "b"). Ces coordonnées sont appellée **scales**, échelles. On note que ces échelles peuvent être spéciales : par exemple, l'échelle logarithmique.

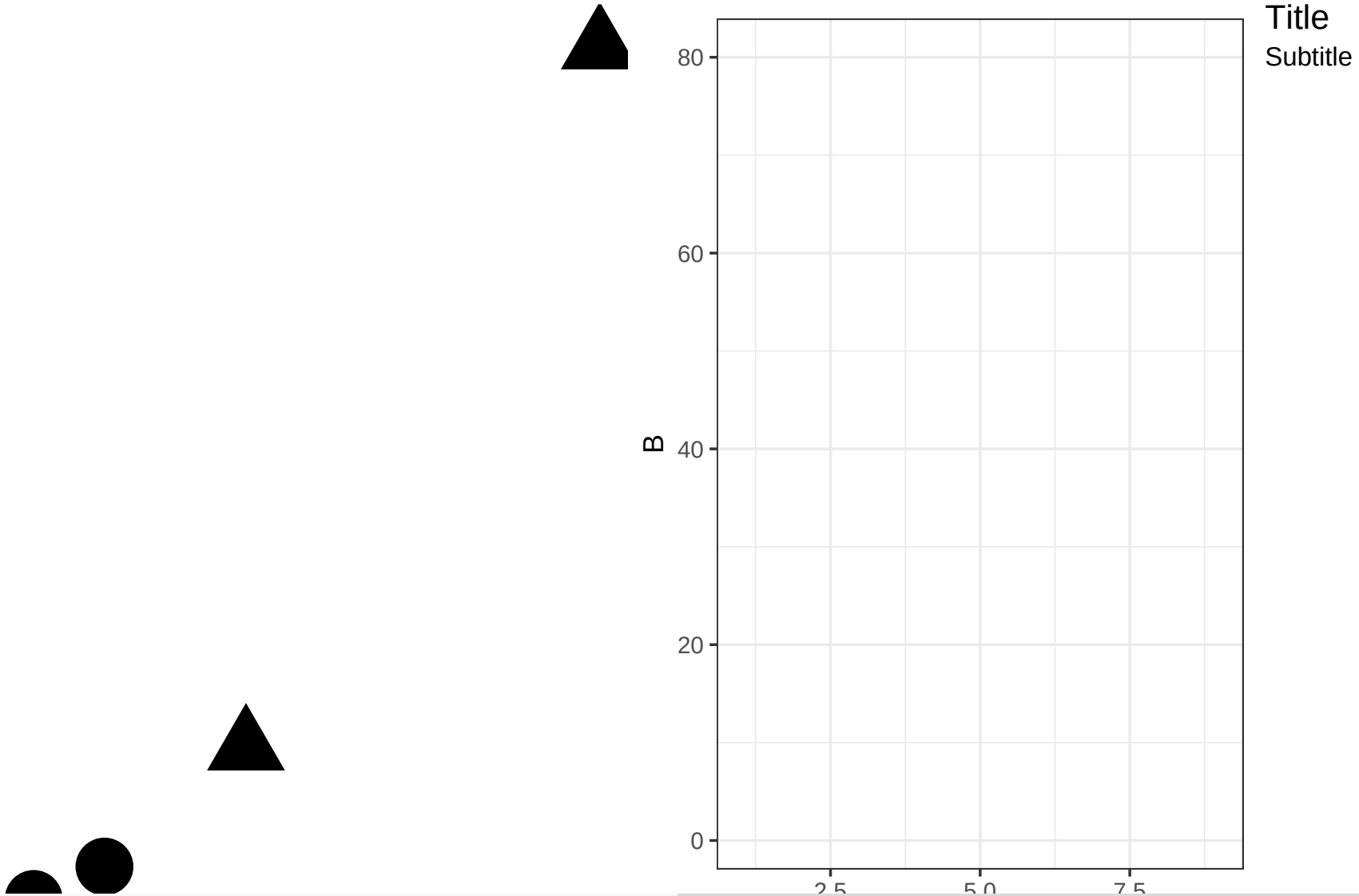
x	y	shape
25	11	circle
0	0	circle
75	53	square
200	300	square

Qu'est-ce qu'un graphique

La production du graphique proprement dit consiste à combiner trois éléments :

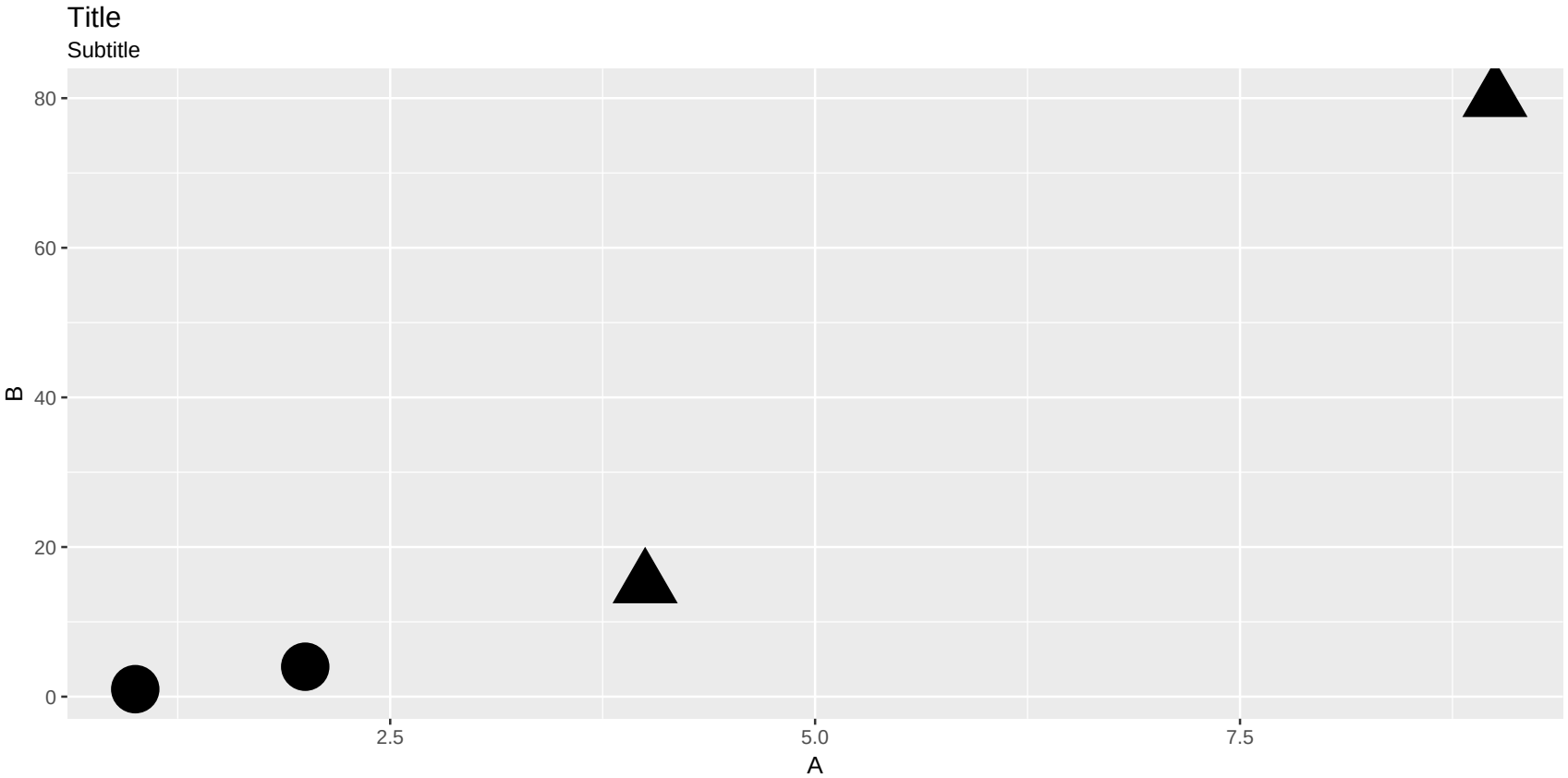
- des objets géométriques
- un système de coordonnées et d'échelles
- les annotations du graphique (titre, etc.)

Qu'est-ce qu'un graphique



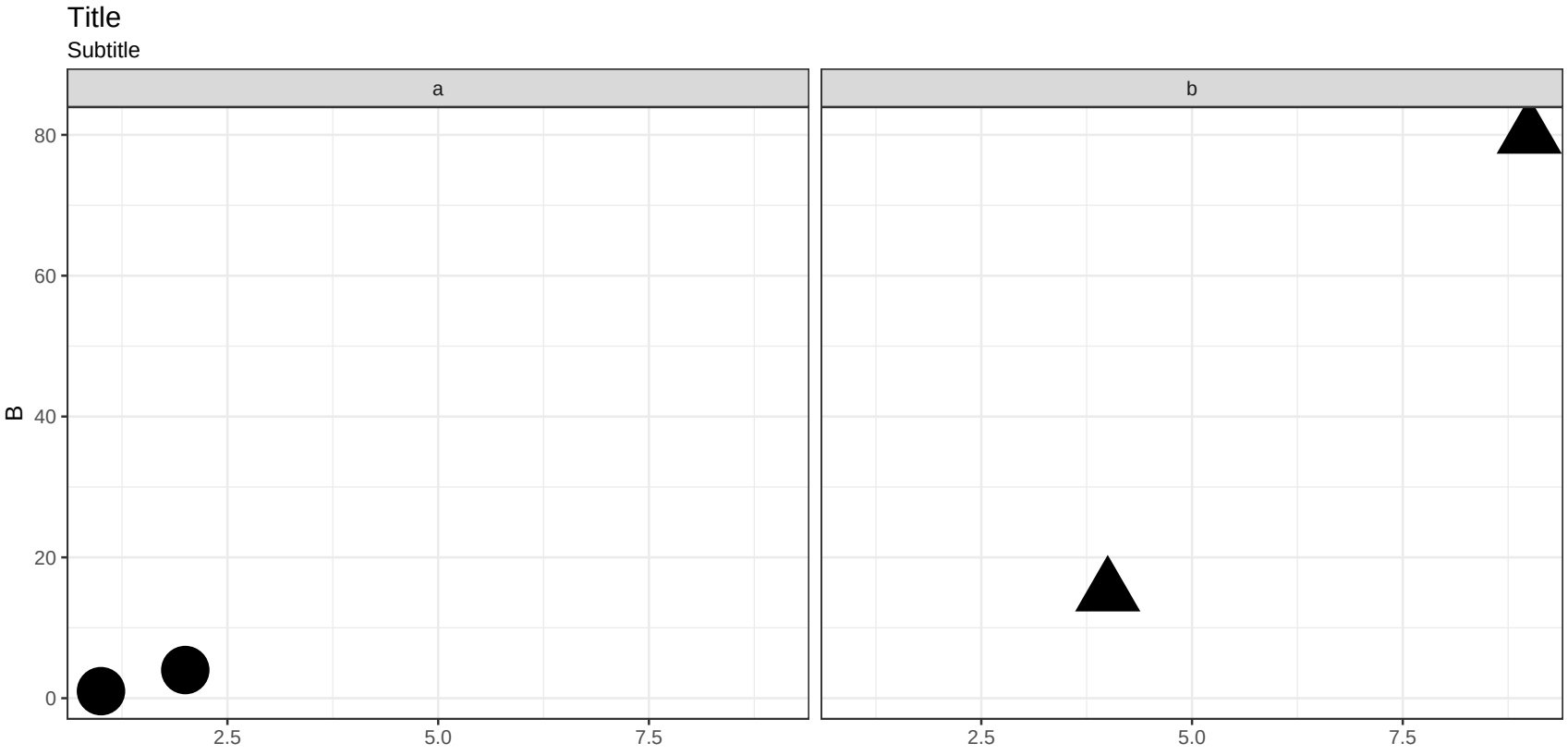
Qu'est-ce qu'un graphique

Le graphique final est donc



Qu'est-ce qu'un graphique

On peut le complexifier en ajoutant un dernier élément, les **facet**, qui consiste à découper les données en sous-échantillon à partir d'une variable catégorielle, et à représenter dans des graphiques différents ces deux échantillons.



Composants d'un graphique

On appelle **layer** (couche) une association de données + mappings + objet géométrique + transformation. Un graphique peut comprendre plusieurs couches (par exemple, on peut superposer un diagramme en points et un diagramme en ligne), qui souvent partagent les mêmes données. L'échelle, le système de coordonnées, et les facets sont communes à toutes les layers.

Sous ggplot, la construction d'un graph se fait de la manière suivante :

```
ggplot(data, mapping) +
  geom_*() +
  scale_*()...
```

`ggplot()` permet d'initialiser le diagramme (déclarer que l'on fabrique un nouveau graphique). Les données et mappings déclarés ici seront réutilisés par toutes les couches, mais peuvent être spécifiées dans chacune. Ensuite, on ajoute des éléments avec +. Conventionnellement, on présente chaque élément sur une nouvelle ligne.

Composants d'un graphique

Les éléments que l'on ajoute sont :

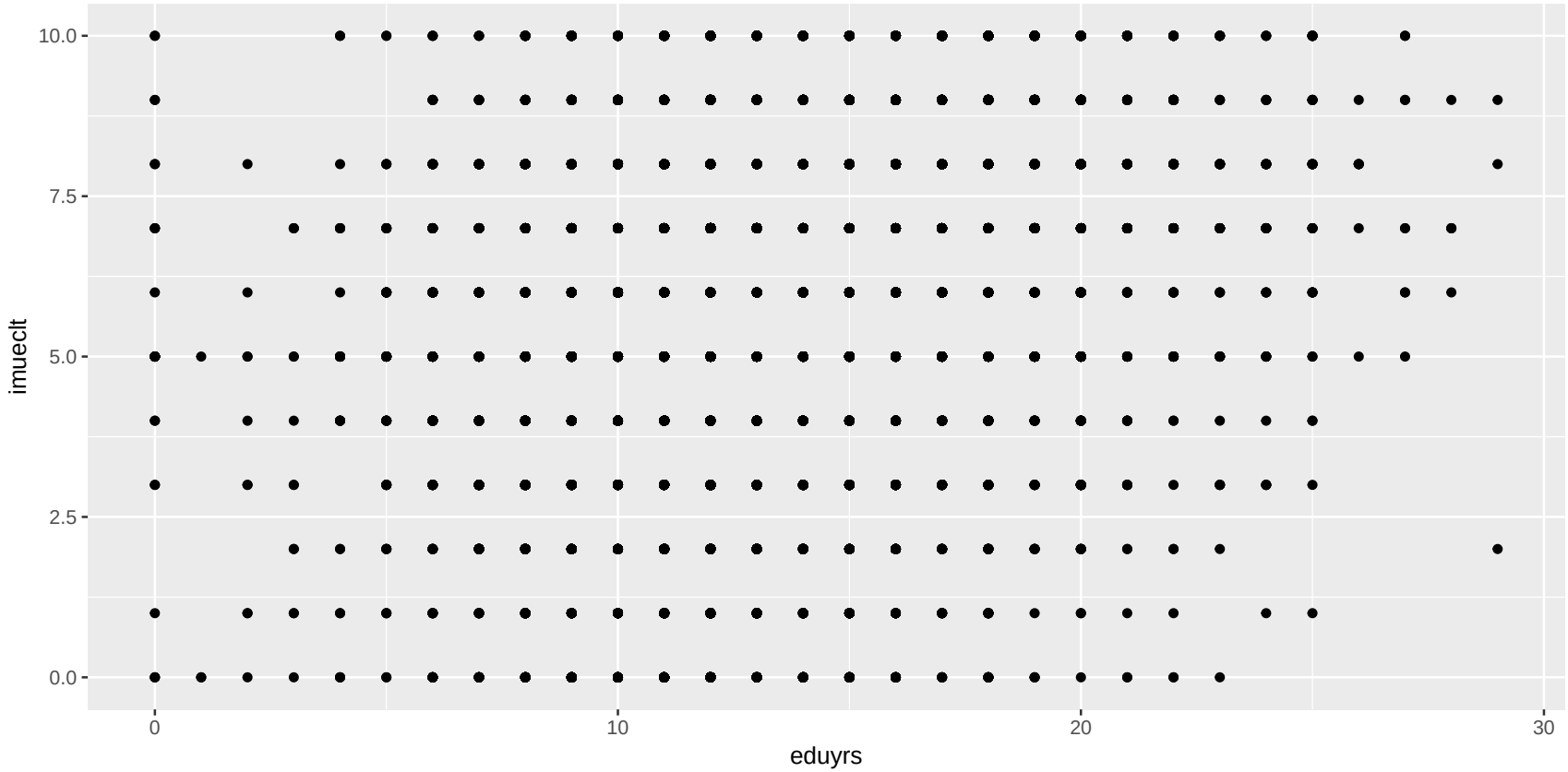
- des objets géométriques (`geom_*`) : déclarer que l'on veut tracer des lignes, des points, des barres, etc. ;
- des transformations statistiques (`stat_*`) : déclarer que l'on veut conserver les données telles quelles ou les transformer ;
- des échelles (`scale_*`) : déclarer et spécifier la façon de représenter chaque aesthetic mapping (`x`, `y`) ;
- des systèmes de coordonnées (`coord_*`) ;
- des subdivision (`facet_wrap()` et `facet_grid()`) : déclarer que l'on veut plusieurs sous-graphs ;
- des annotations graphiques (plusieurs fonctions, en particulier `theme()`)

Premier exemple : les données

```
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)
```

Premier exemple : scatterplot

```
ggplot(data = d, mapping = aes(x = eduyrs, y = imueclt)) +
  geom_point() # on pourrait mettre données et mapping ici
```

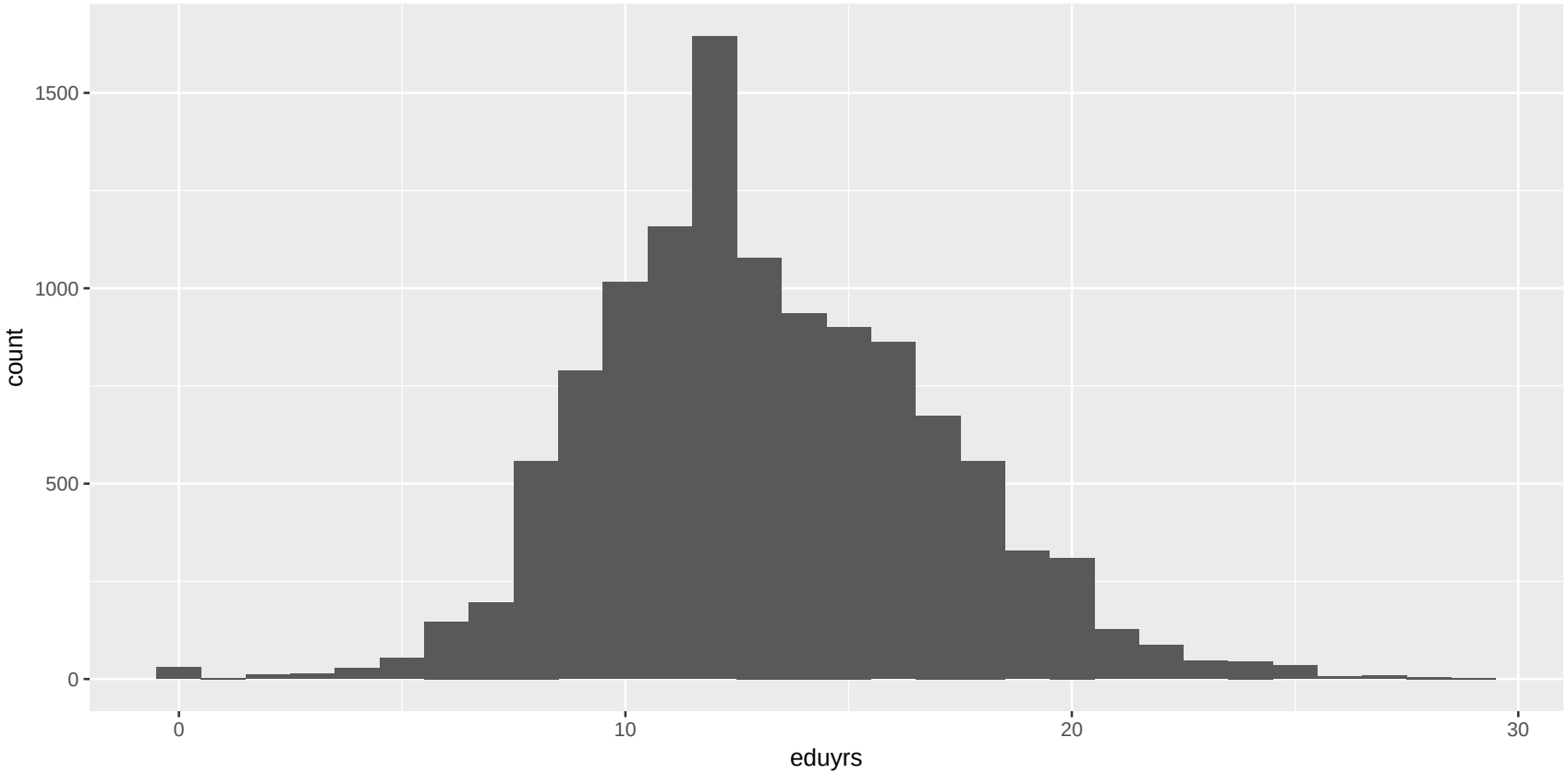


Graphiques basiques

Numérique univarié

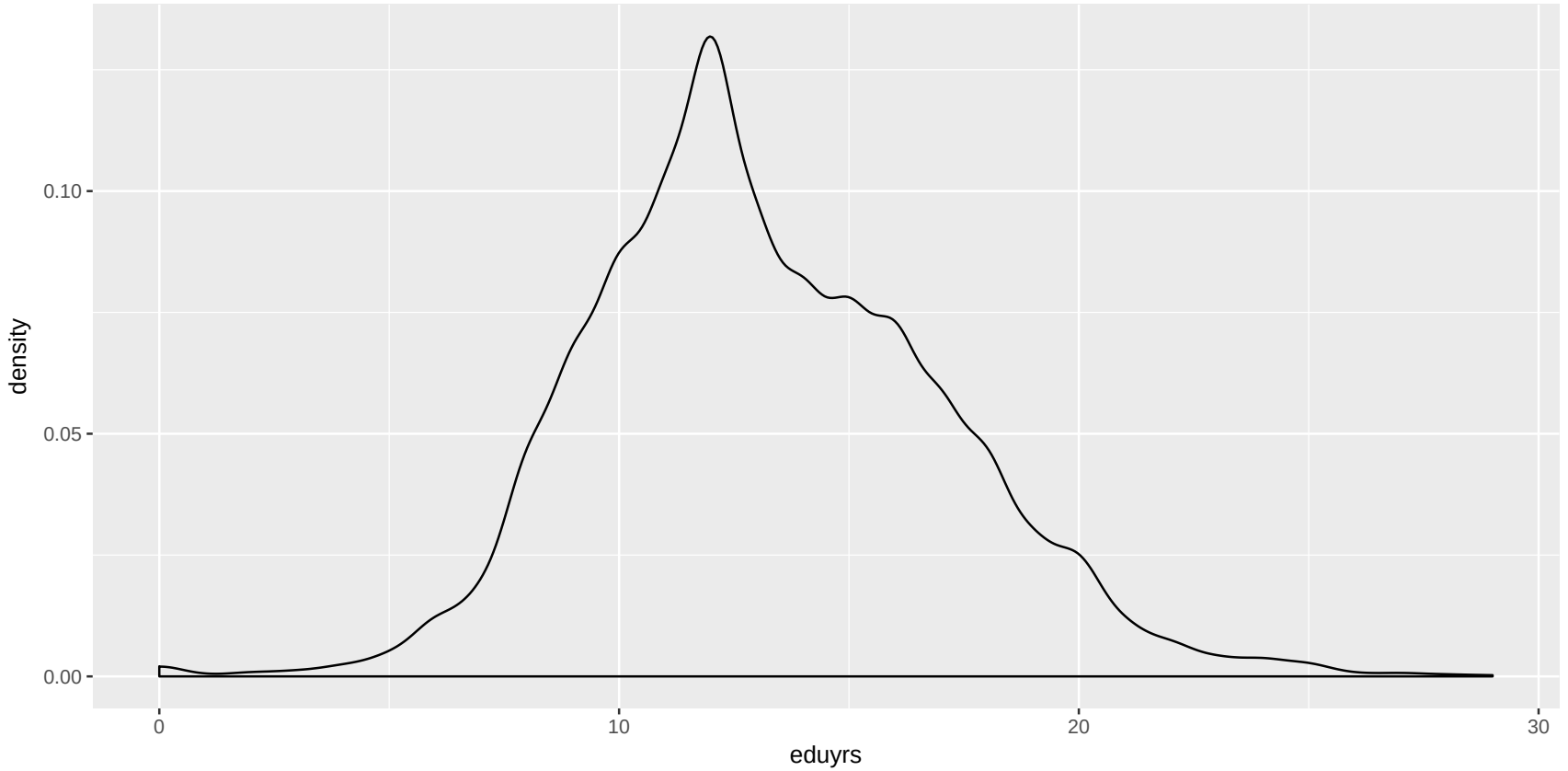
Histogramme

```
ggplot(d, aes(eduyrs)) +
  geom_histogram()
```



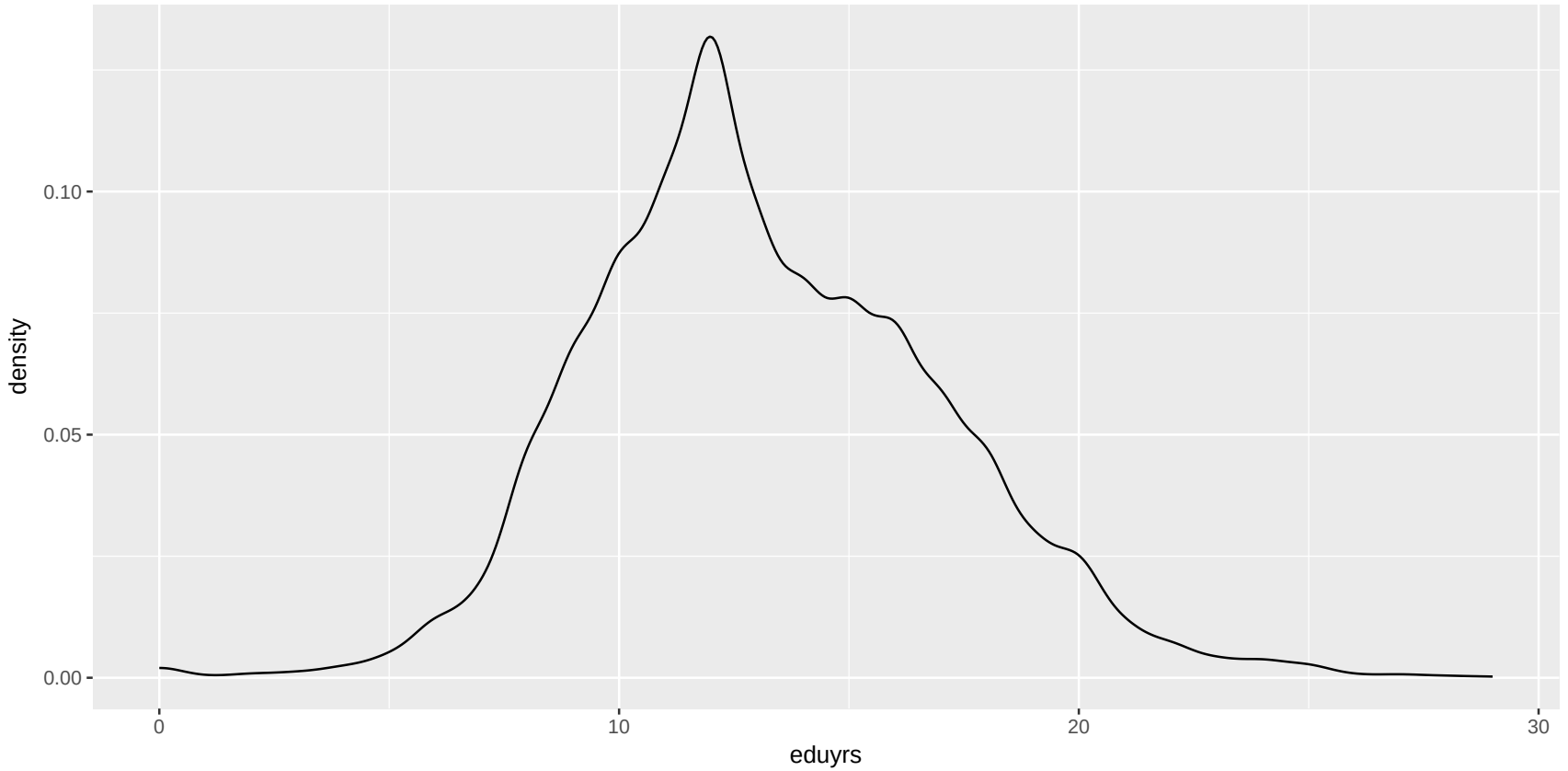
Densité

```
ggplot(d, aes(eduyrs)) +
  geom_density()
```



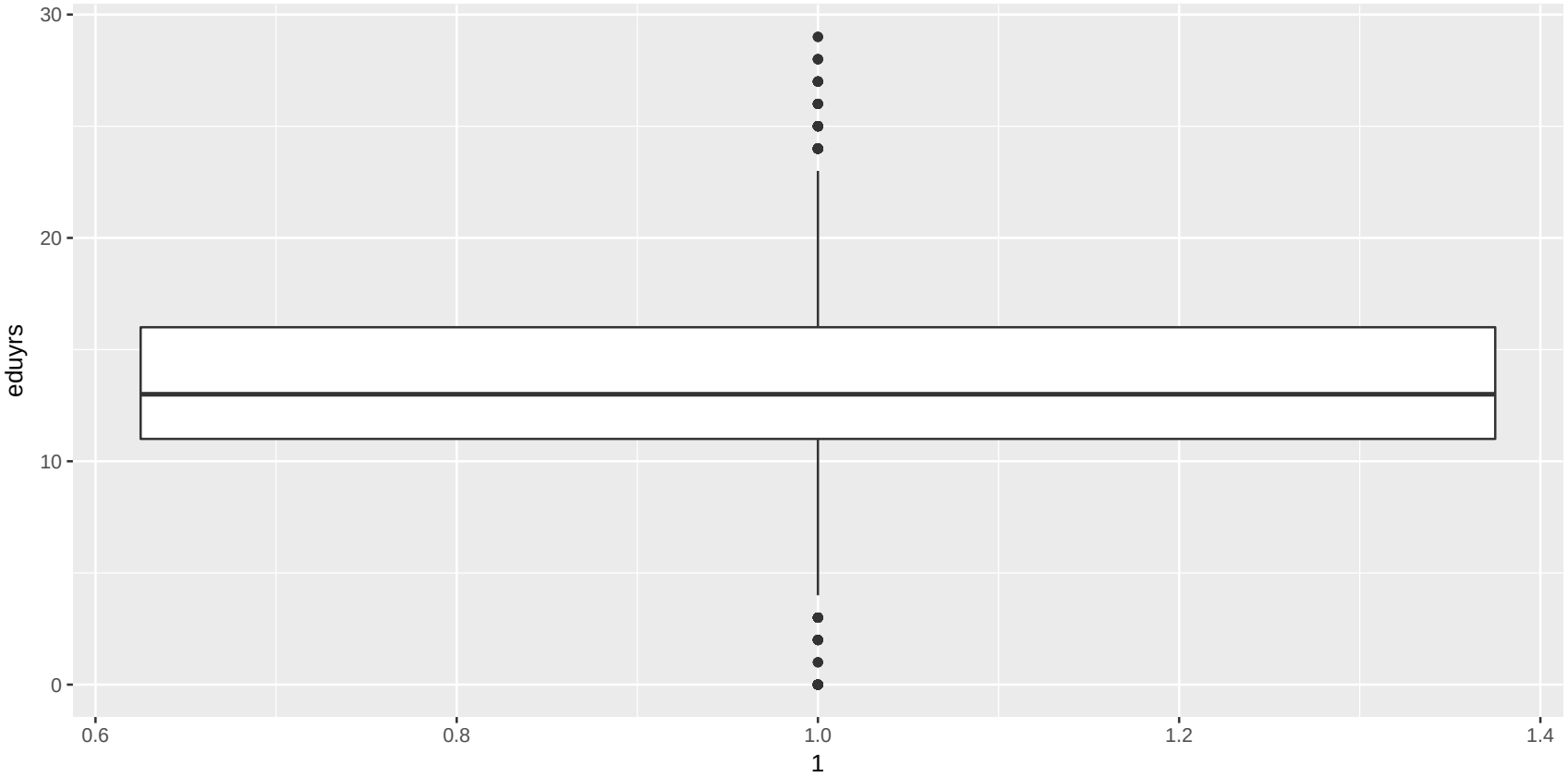
Densité (alternative)

```
ggplot(d, aes(eduyrs)) +
  geom_line(stat = "density")
```



Boxplot

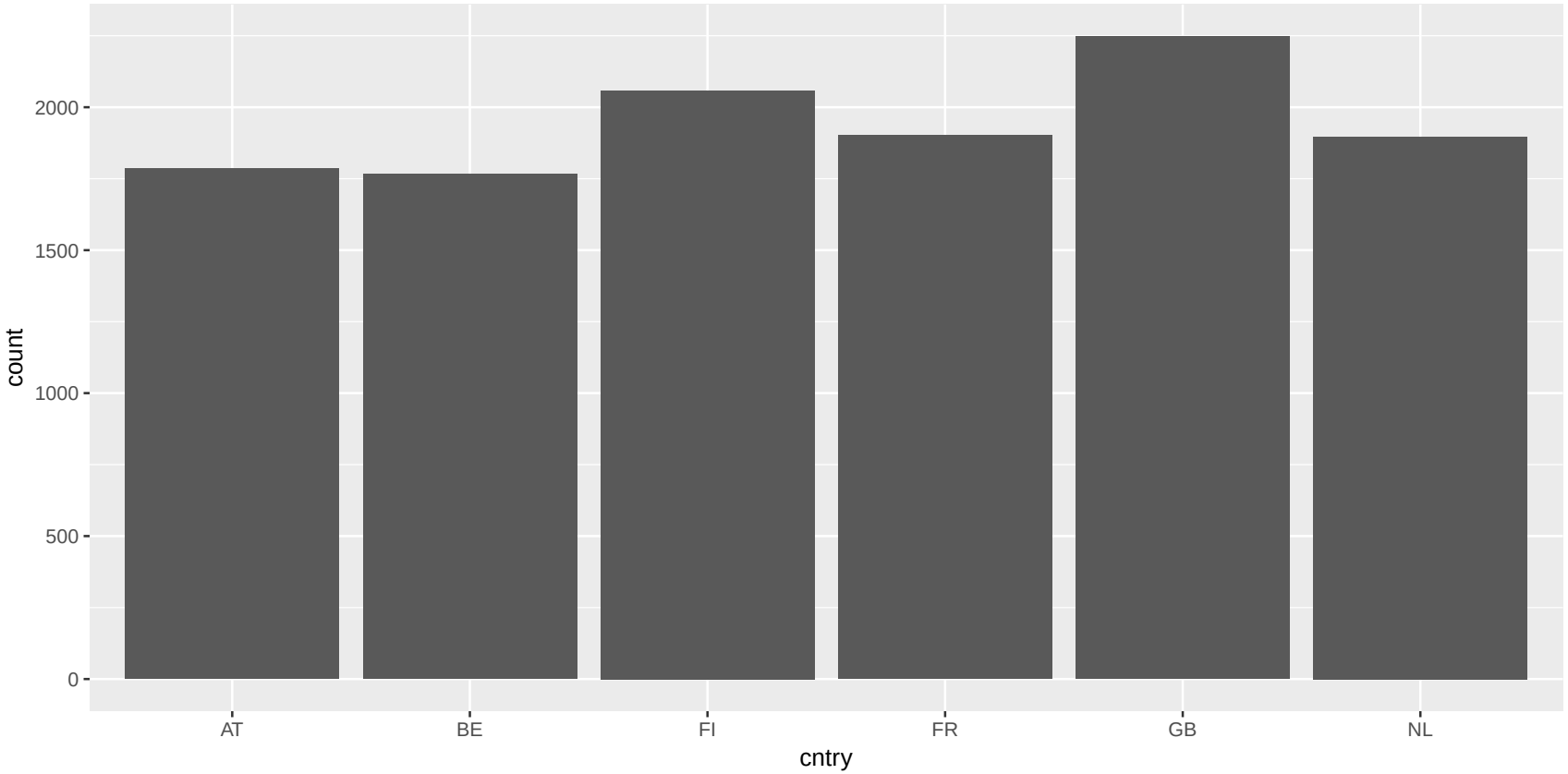
```
ggplot(d, aes(1, eduyrs)) +
  geom_boxplot()
```



Catégoriel univarié

Barplot

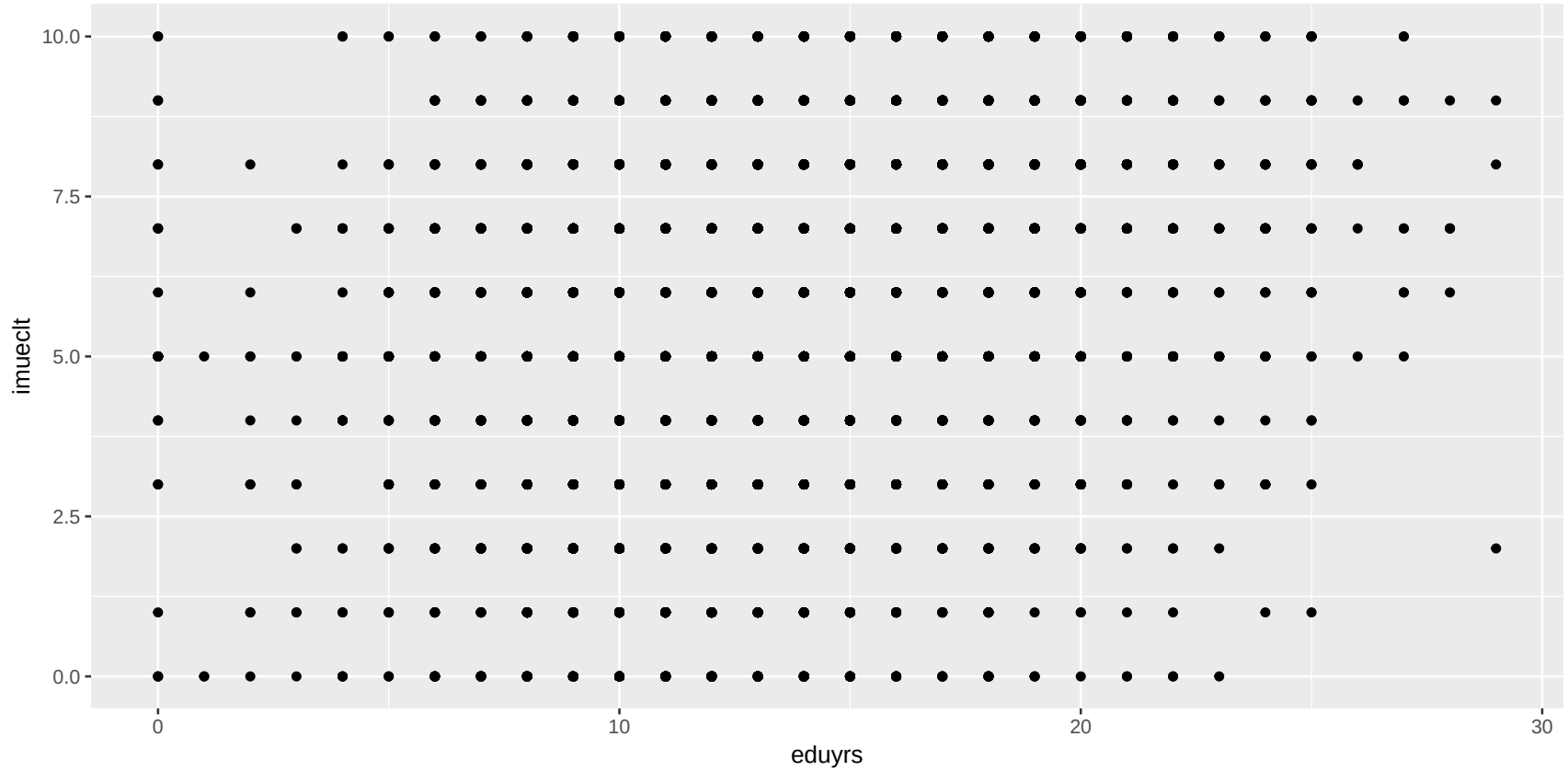
```
ggplot(d, aes(cntry)) +
  geom_bar()
```



Numérique bivarié

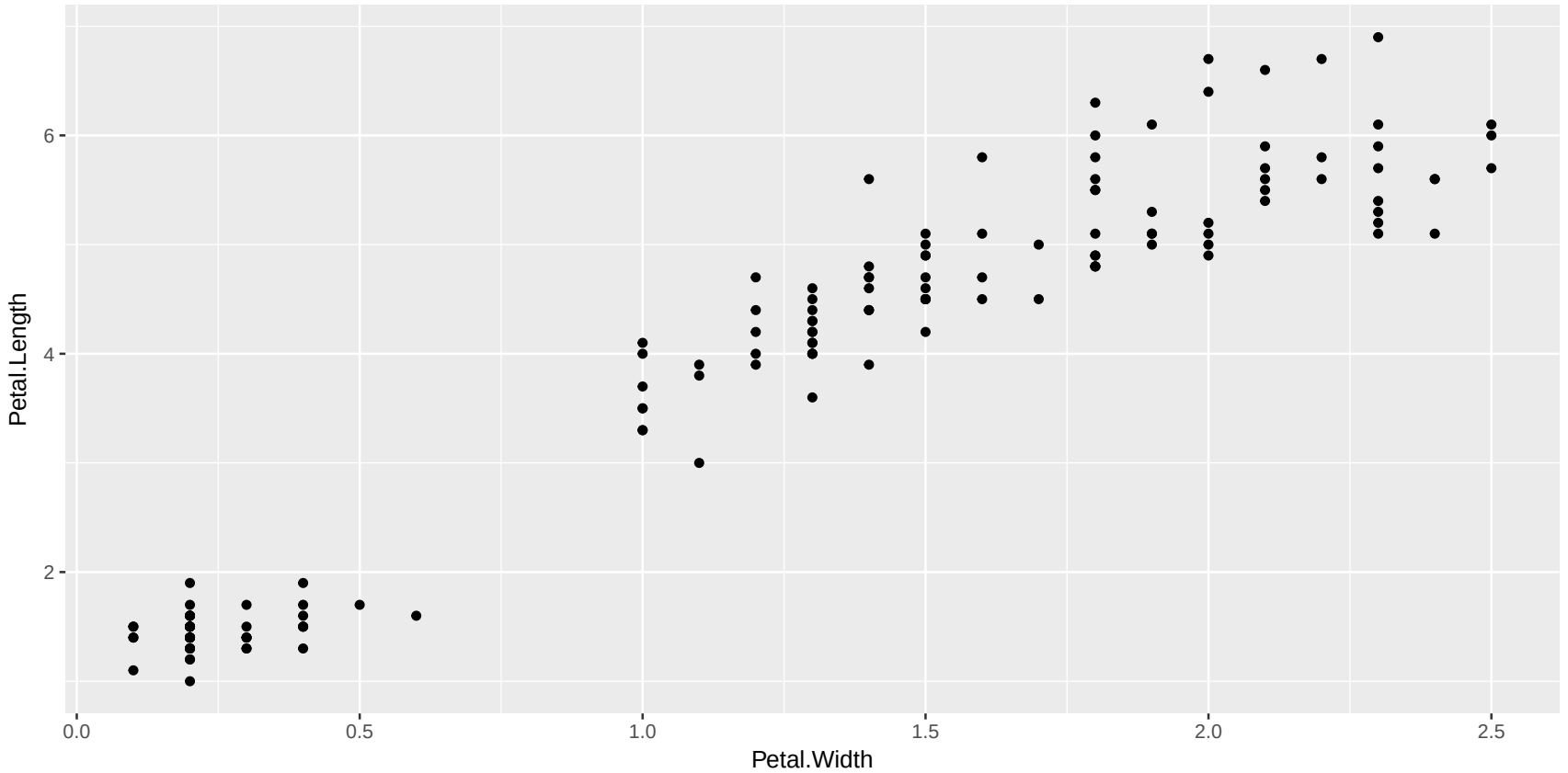
Scatterplot

```
ggplot(d, aes(eduyrs, imueclt)) +
  geom_point()
```



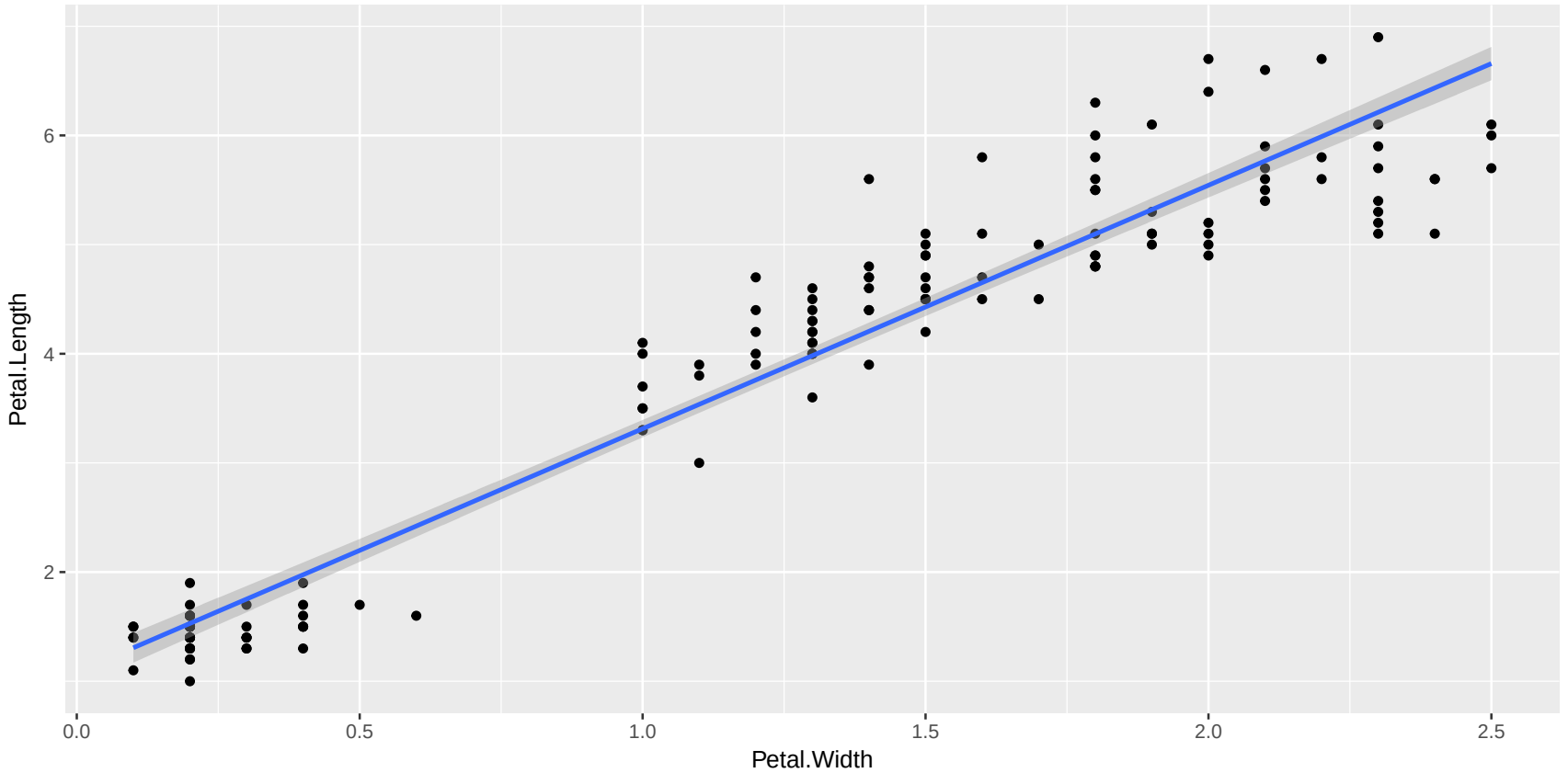
Ajouter une tendance

```
data(iris)
ggplot(iris, aes(Petal.Width, Petal.Length)) +
  geom_point()
```



Ajouter une tendance

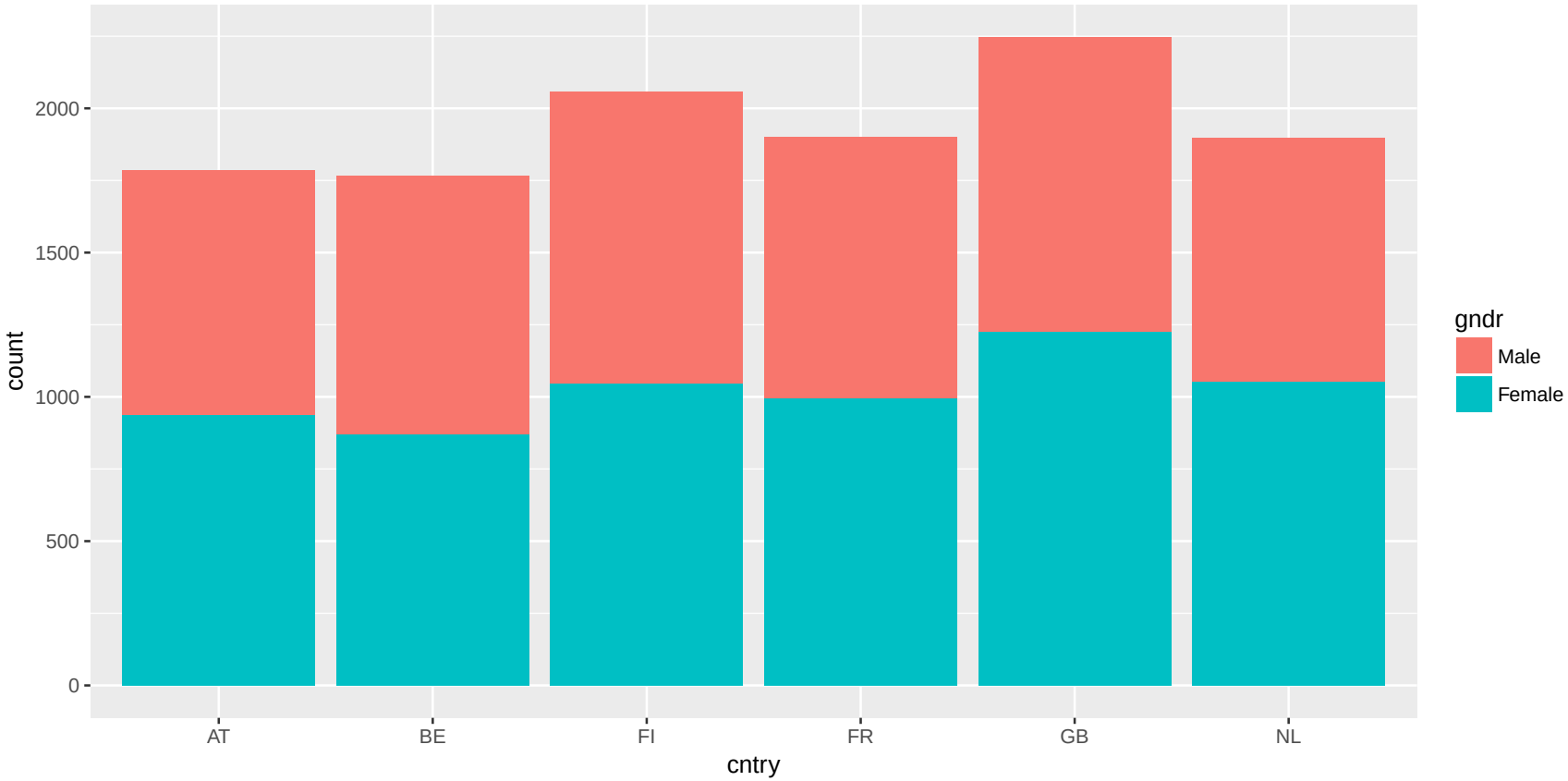
```
ggplot(iris, aes(Petal.Width, Petal.Length)) +
  geom_point() +
  geom_smooth(method = "lm")
```



Deux variables catégorielles

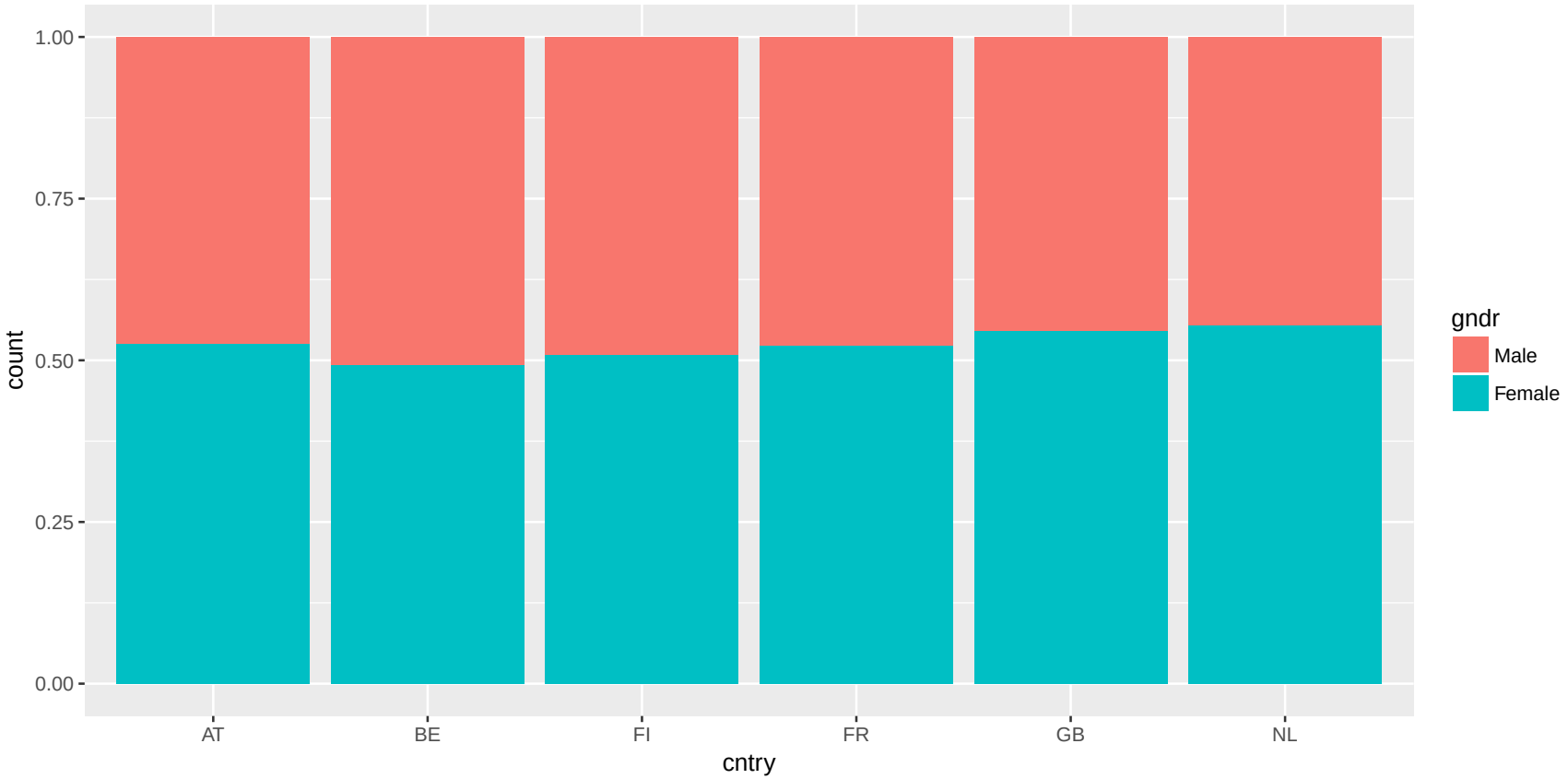
Barplot en effectifs

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



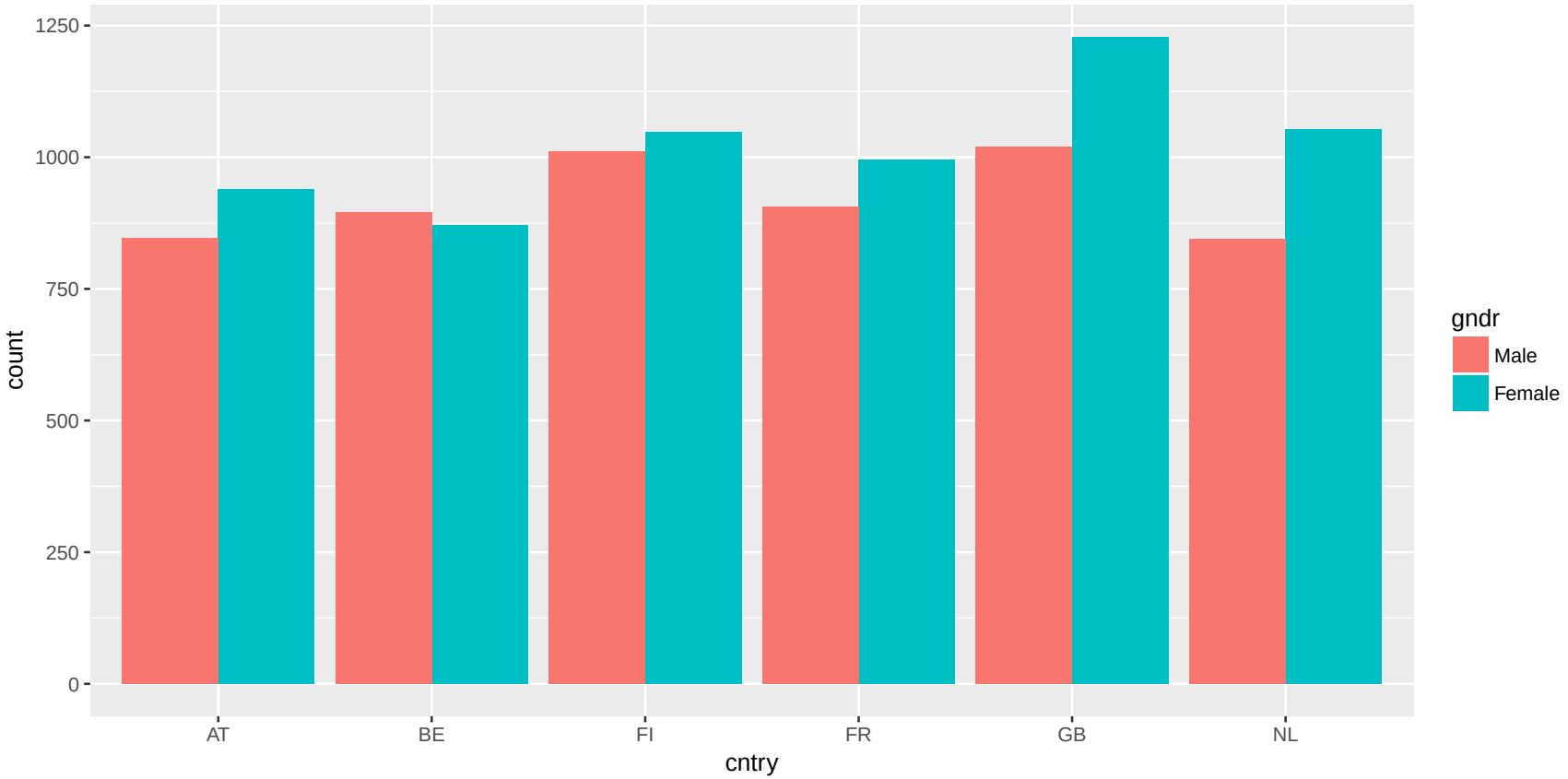
Barplot en fréquence

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



Barplot dodged

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

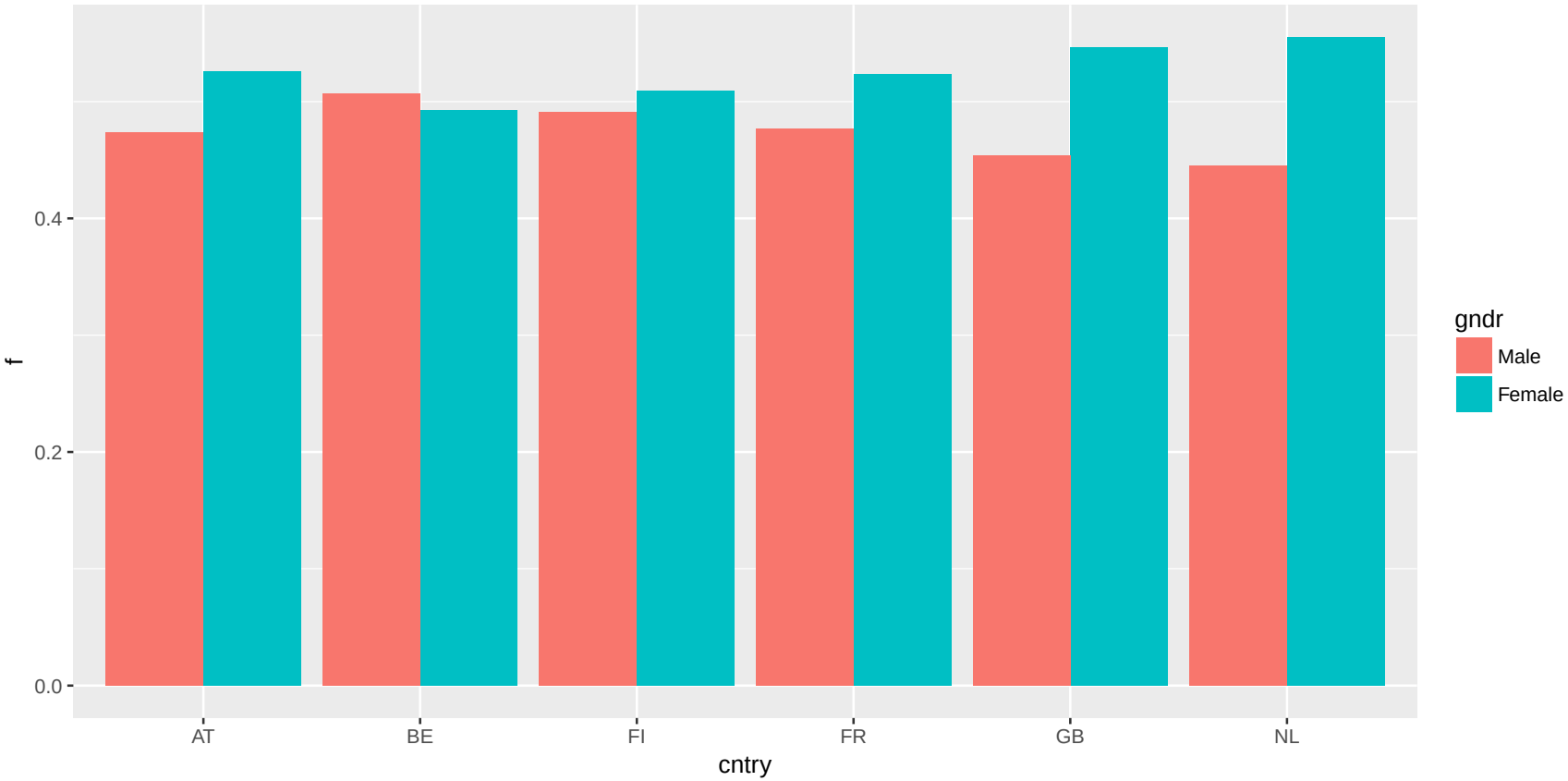


Barplot dodged – fréquence

```
bp <- group_by(d, cntry, gndr) %>%
  summarize(n = n()) %>%
  group_by(cntry) %>%
  mutate(f = n / sum(n))
```

Barplot dodged – fréquence

```
ggplot(bp, aes(x = cntry, y = f, fill = gndr)) +
  geom_bar(position = "dodge", stat = "identity")
```

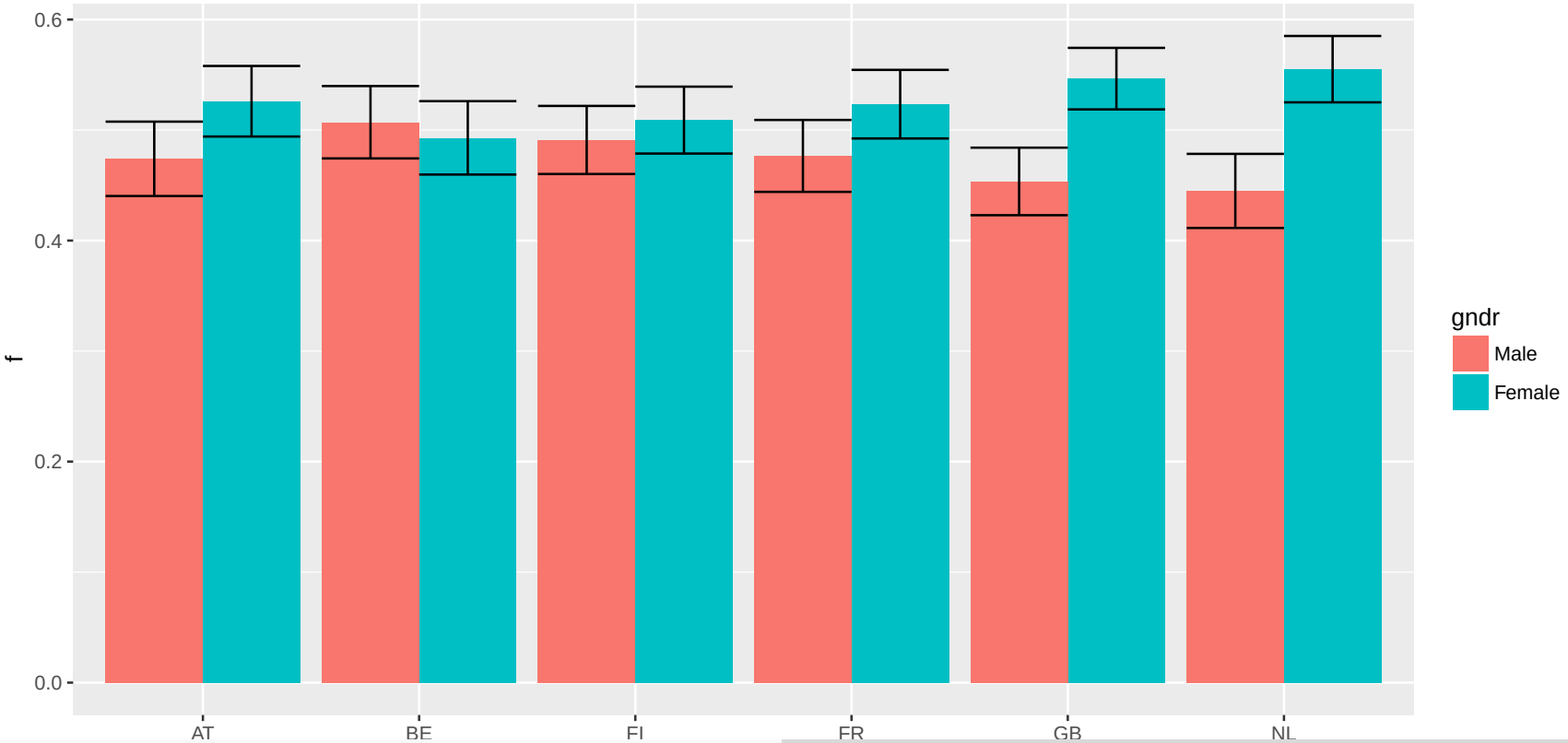


Ajouter des barres d'erreur

```
bp <- group_by(d, cntry, gndr) %>%
  summarize(n = n()) %>%
  group_by(cntry) %>%
  mutate(f = n / sum(n),
         se = 1.96 * sqrt((f * (1-f))/n))
```

Barplot dodged – fréquence + se

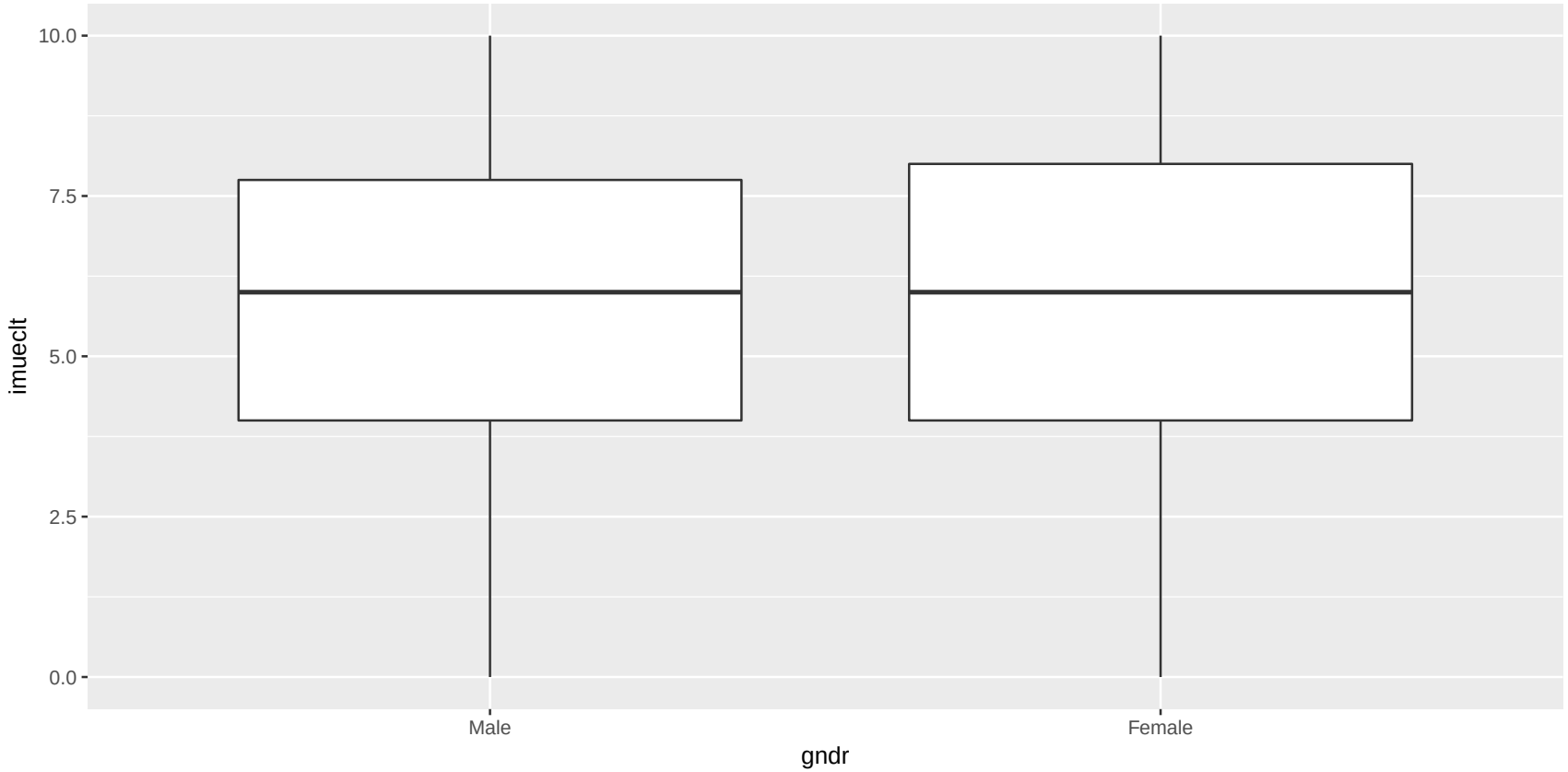
```
ggplot(bp, aes(x = cntry, y = f, fill = gndr)) +
  geom_bar(position = "dodge", stat = "identity") +
  geom_errorbar(aes(ymin = f - se, ymax = f + se),
                position = "dodge")
```



Une variable catégorielle, une variable numérique

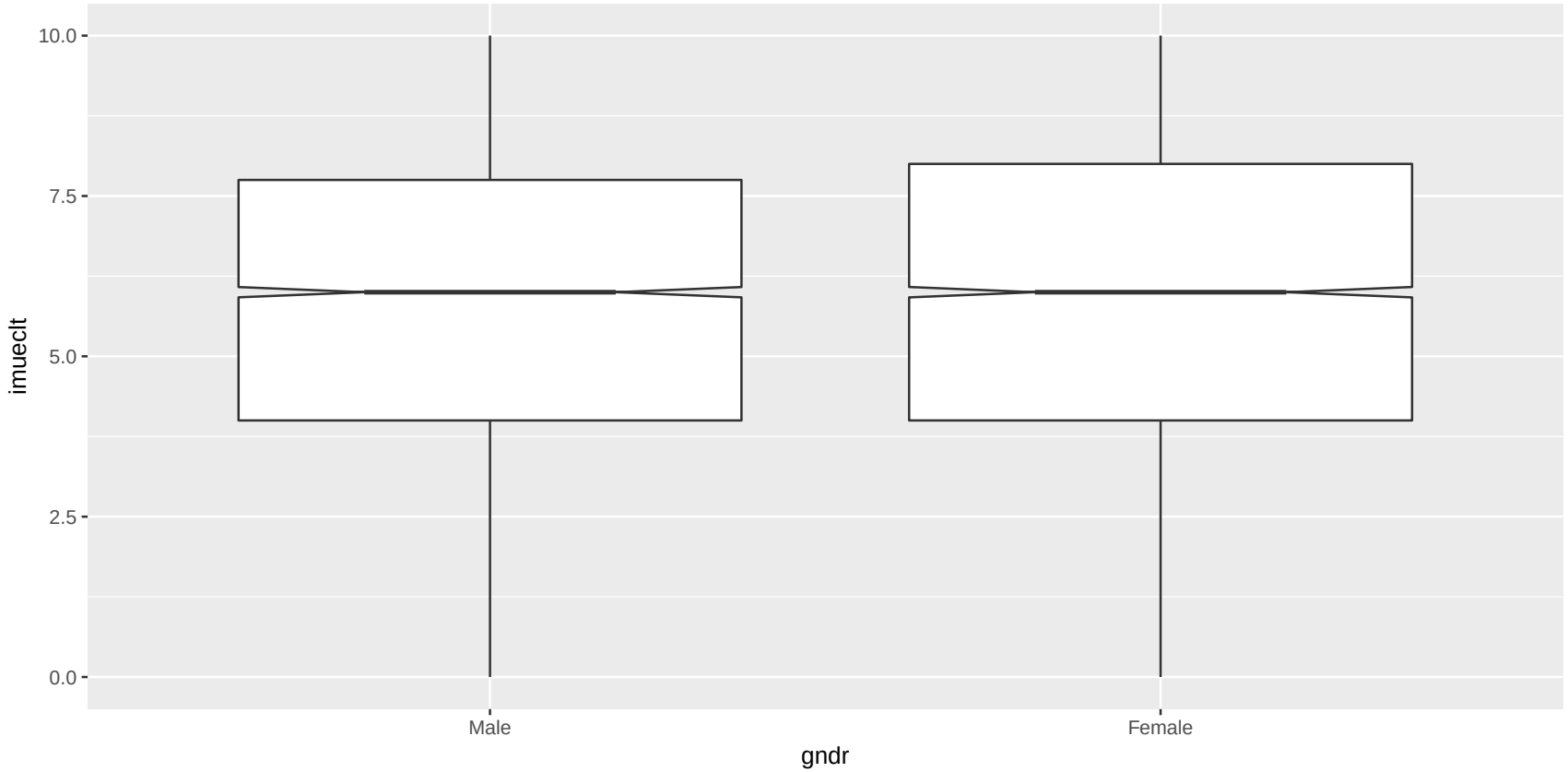
Boxplot

```
ggplot(d, aes(gndr, imueclt)) +
  geom_boxplot()
```



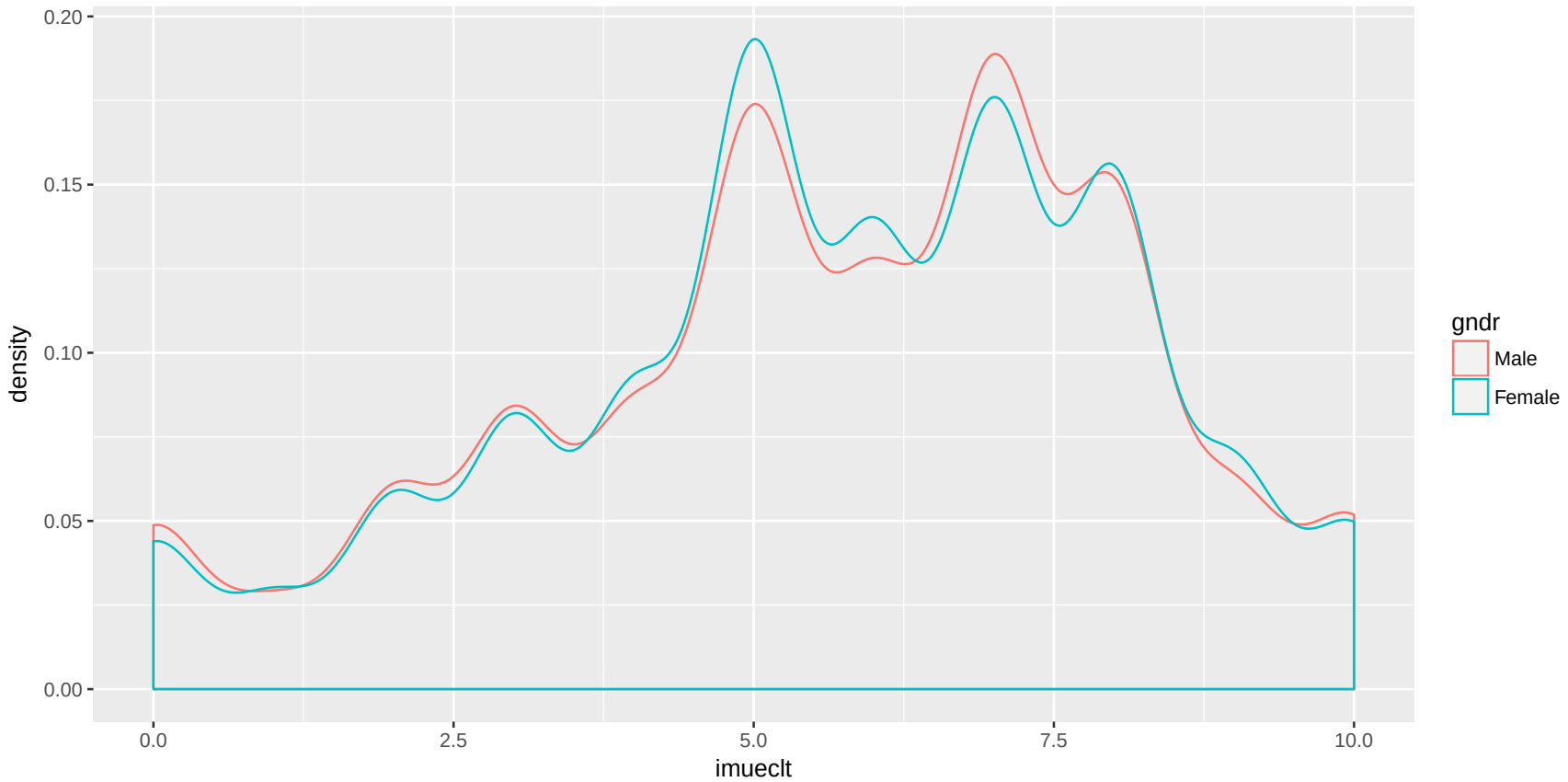
Boxplot

```
ggplot(d, aes(gndr, imueclt)) +
  geom_boxplot(notch = TRUE)
```



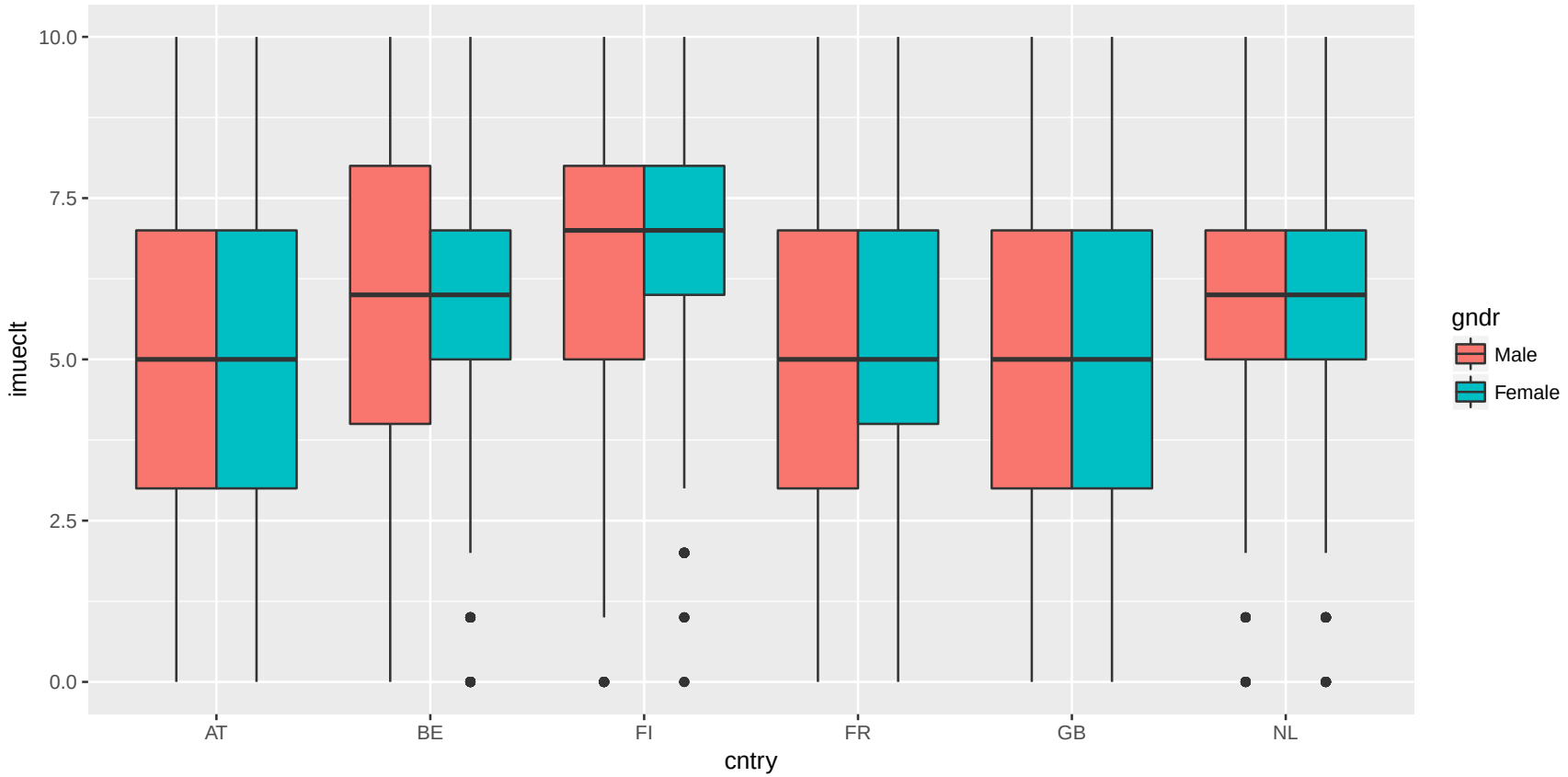
Densities

```
ggplot(d, aes(imueclt, color = gndr)) +
  geom_density()
```



Généralisation

```
ggplot(d, aes(cntry, imueclt, fill = gndr)) +
  geom_boxplot()
```



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.

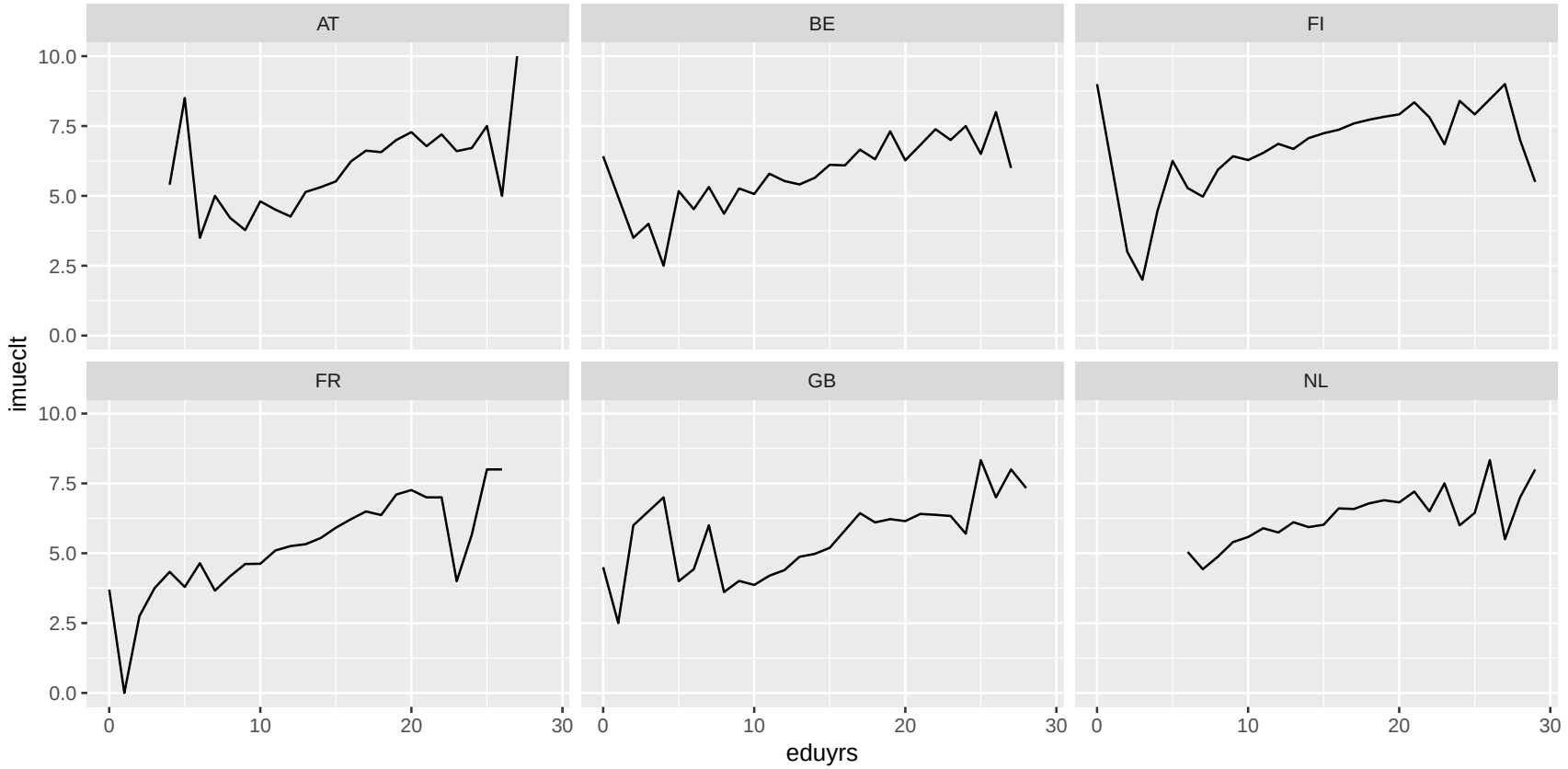
Facet : syntaxe

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

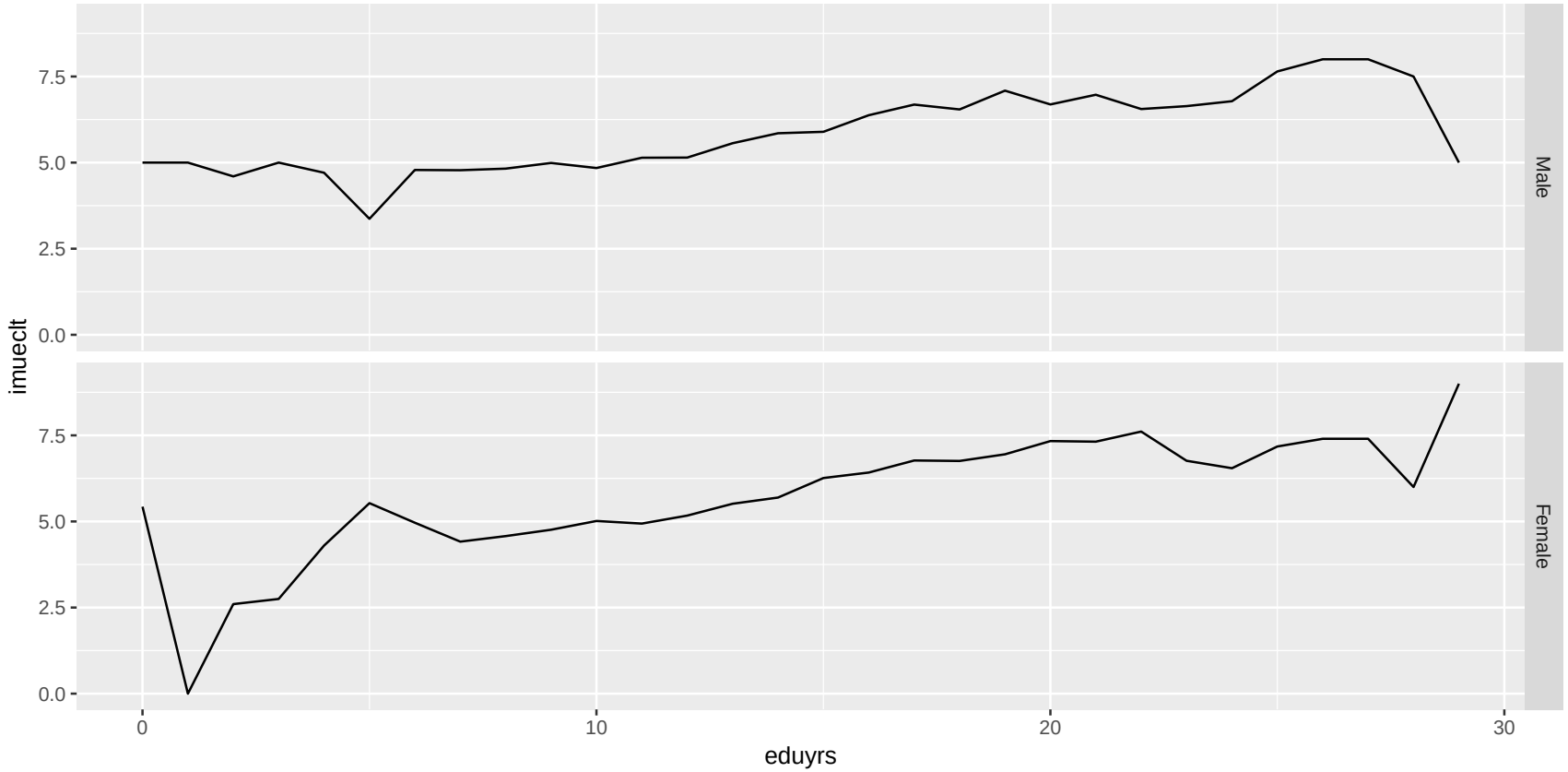
facet_wrap

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



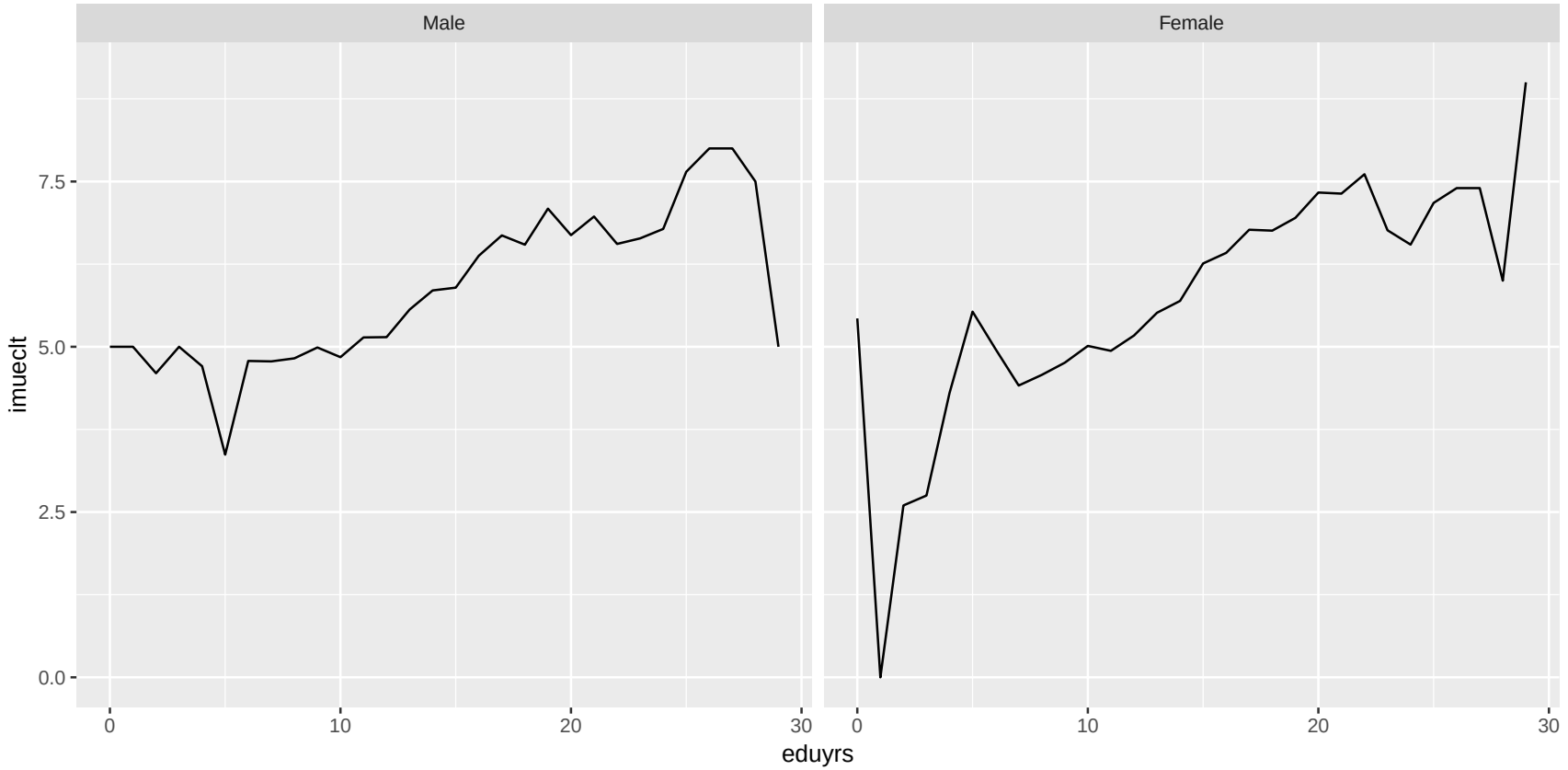
facet_grid : une seule variable en lignes

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



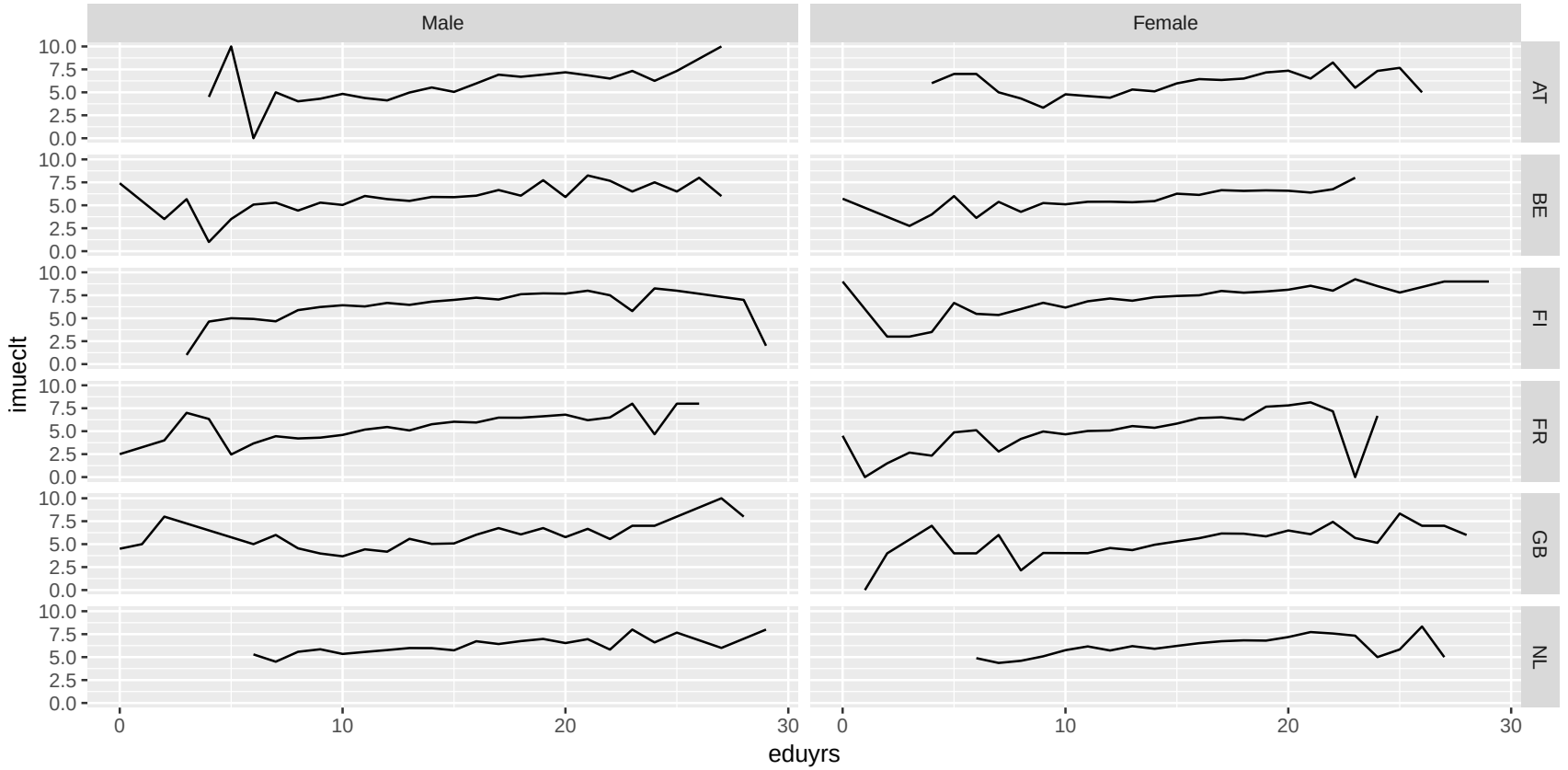
facet_grid : une seule variable en colonnes

```
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

Principe

ggplot et le tidyverse permette de faire rapidement des graphs d'un grand nombre de variables différentes. Cela nécessite un peu de manipulation.

La situation classique est la suivante : nous avons un grand nombre de variables d'une même série que nous souhaitons représenter chacune dans une facette.

Or, pour cela, nous devons transformer notre base de départ. En effet, nous avons besoin pour produire des facette d'un jeu de donnée dans lequelle une variable contient les catégories employée pour les facettes, et les autres les valeurs à représenter.

Données longues et larges

Nous cherchons donc à passer d'un format large

X	Y	Z
x1	y1	z1
x2	y2	z2
x3	y3	z3

A un format long.

variable	value
X	x1
X	x2
X	x3
Y	y1
Y	y2
Y	y3
Z	z1
Z	z2
Z	z3

Données longues et larges

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

Dans le tidyverse, nous avons deux fonctions qui permettent de passer d'un format à l'autre

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

Attention : ces fonctions sont en train de changer pour être remplacée par `pivot_long()` et `pivot_wide()`. Cf. https://tidyr.tidyverse.org/dev/articles/pivot.html

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).

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)
```

```
## # A tibble : 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 avec une indexation négative (-) ; 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)
```

```
## # A tibble : 2 x 3
## # Groups :   var [2]
##   gndr  var      val
##   <fct> <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().

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 x 3
##    cntry var       val
##    <fct> <chr>   <dbl>
##  1 AT    qfimedu    10
##  2 AT    qfimedu    10
```

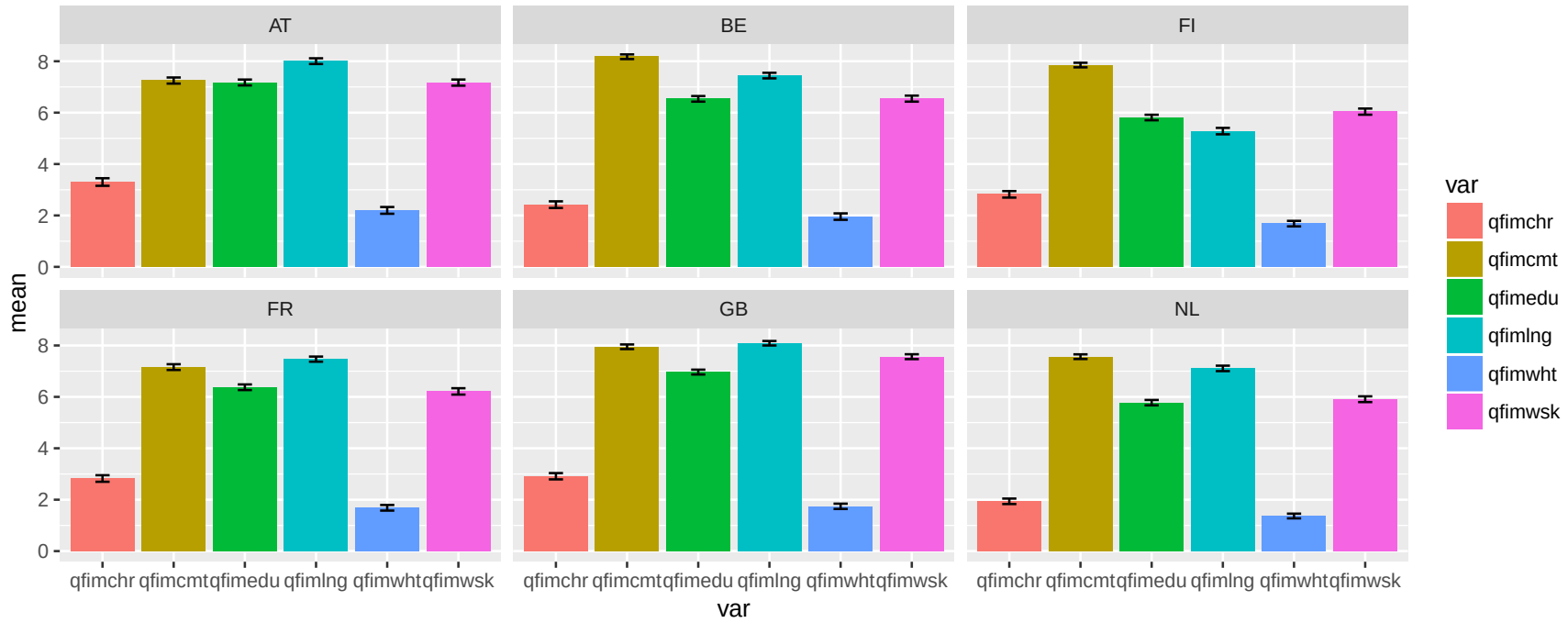
Exemple

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()))
```

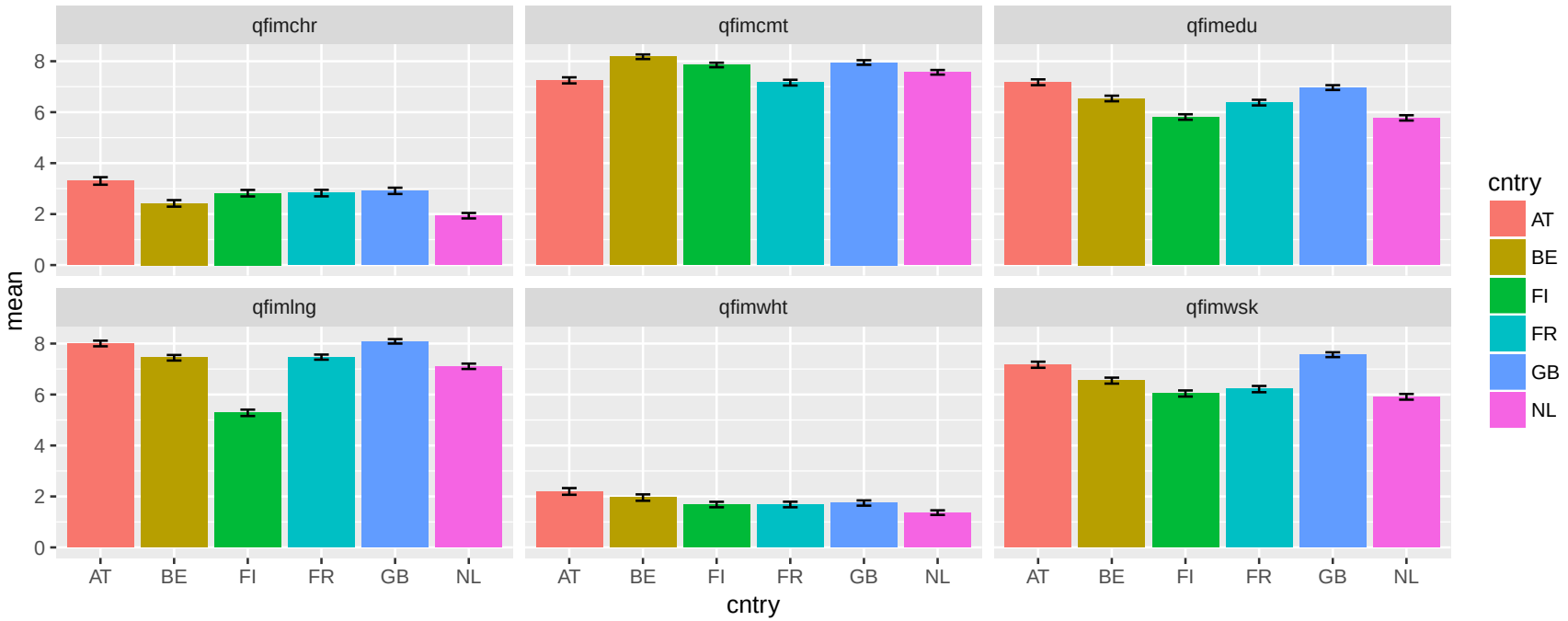
Exemple : facet par pays

```
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)
```



Exemple : facet par variable

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

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

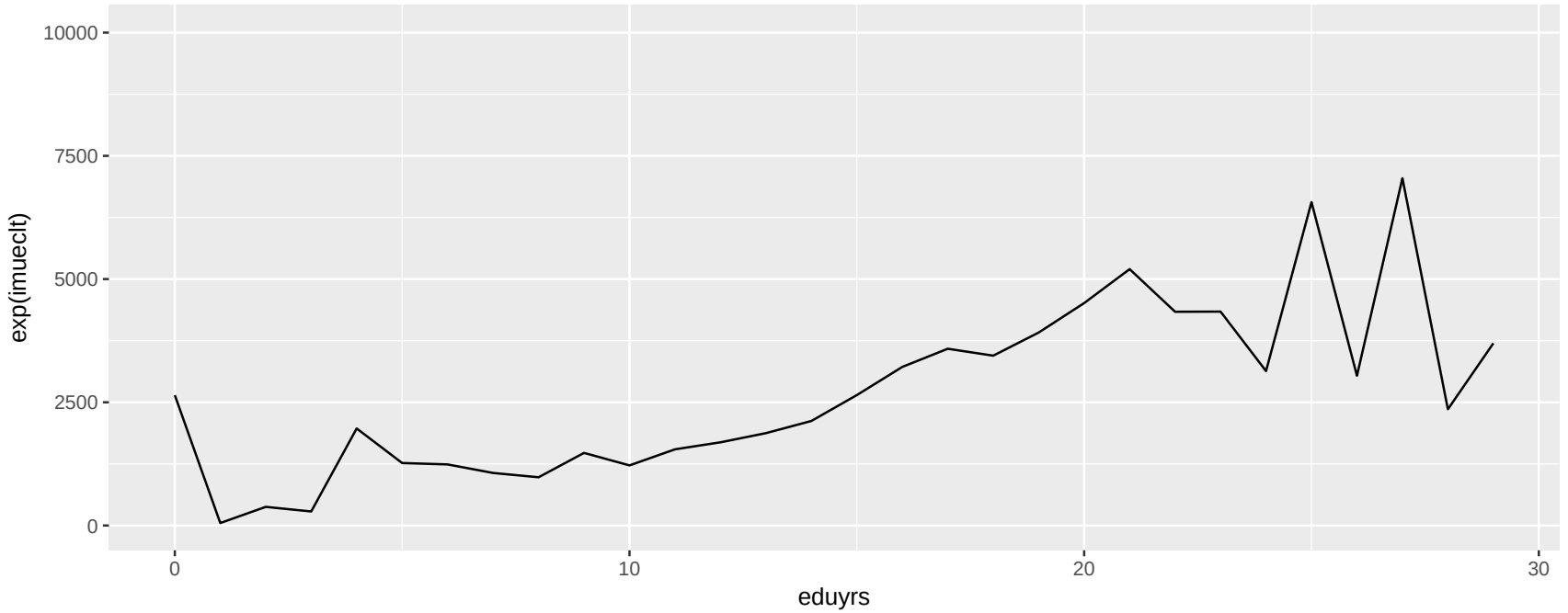
Les échelles (scales) sont des couches de ggplot qui précisent les transformations à appliquer aux axes, ainsi que les valeurs à indiquer. de ce fait, on les emploie à la fois :

- 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.)

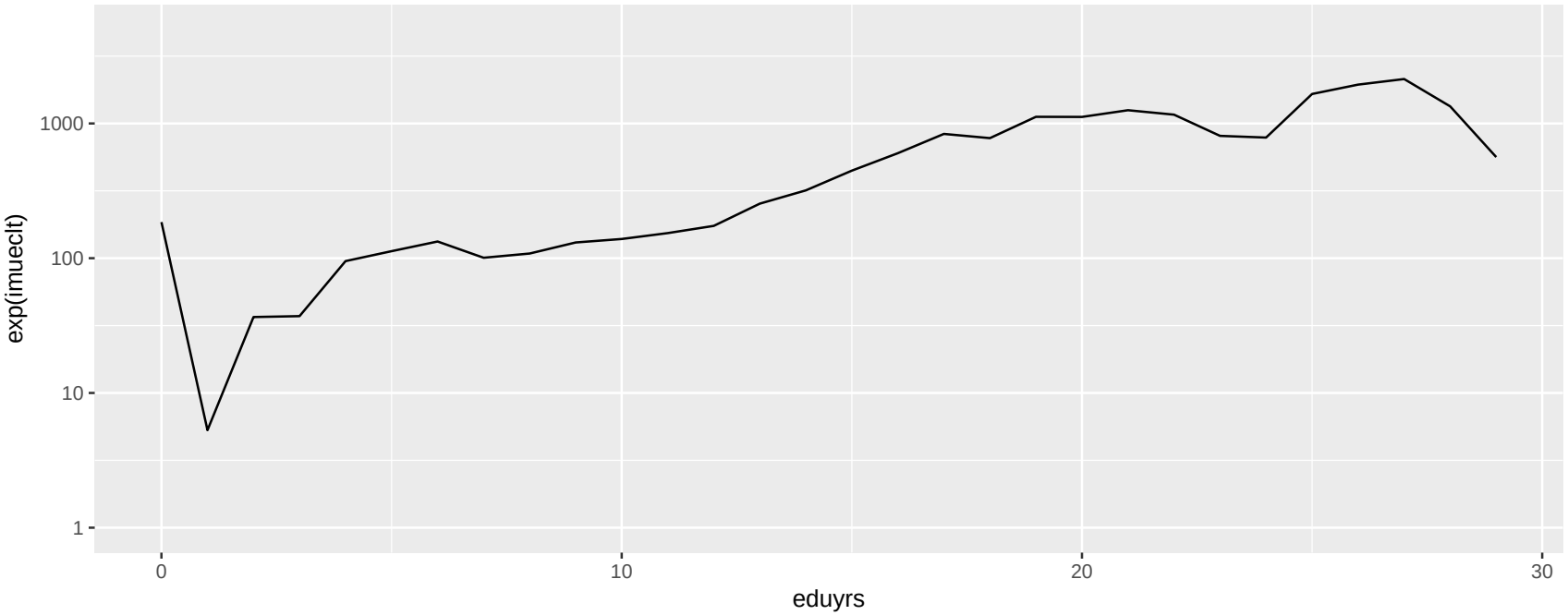
Transformer une échelle

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



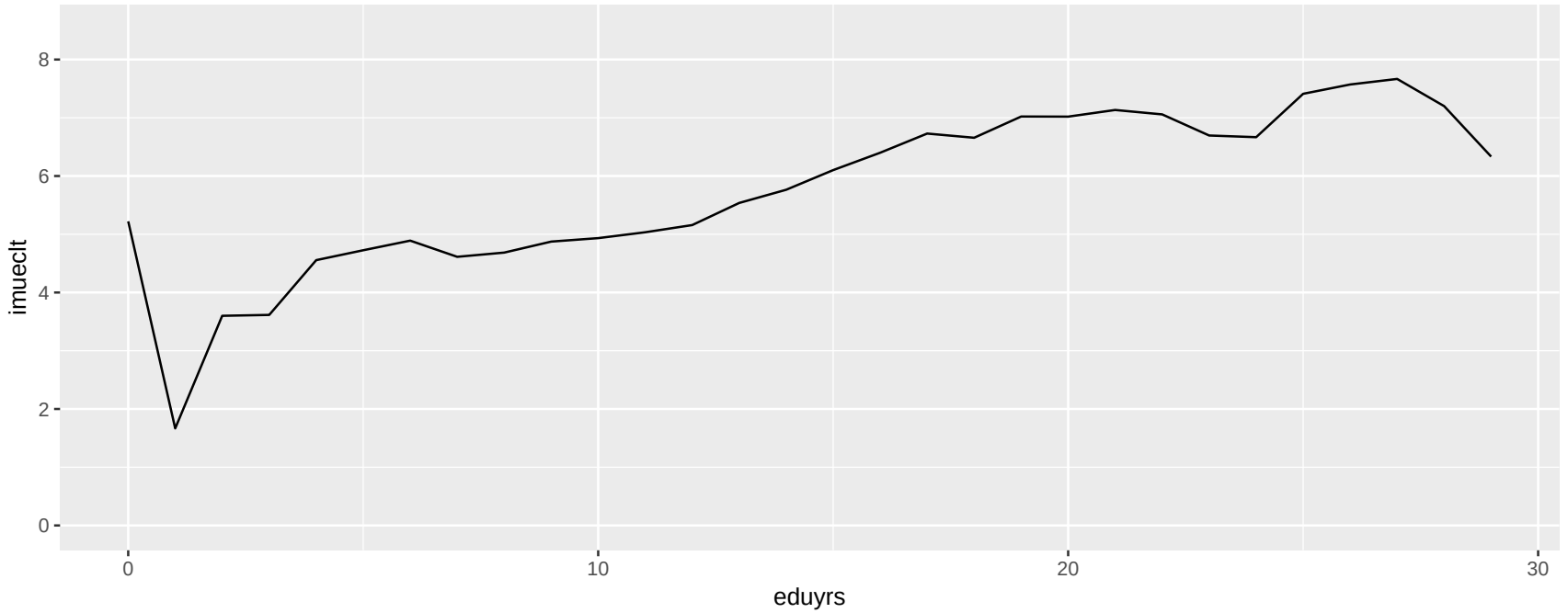
Transformer une échelle : y logarithmique

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



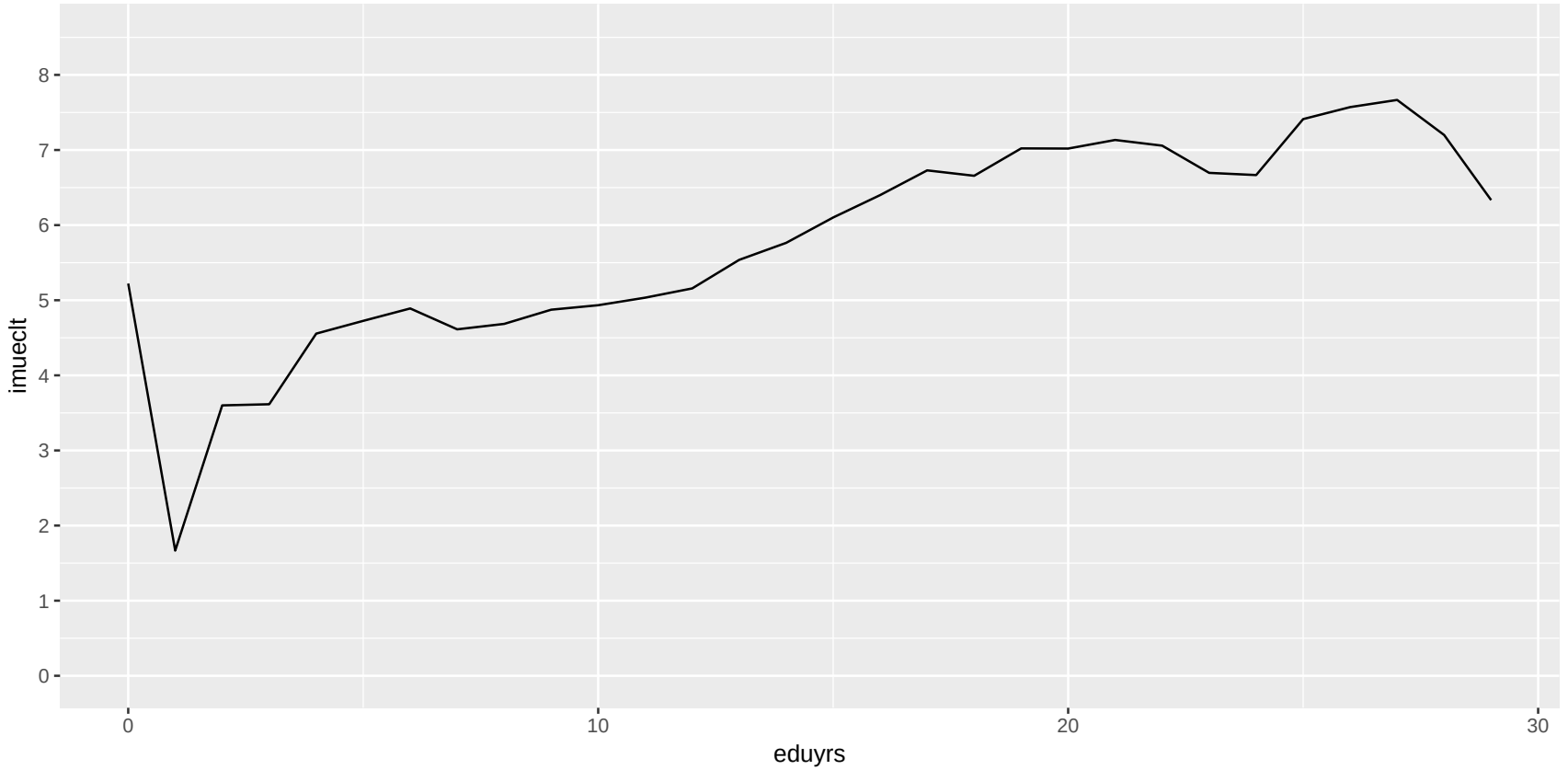
Renuméroter des axes

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



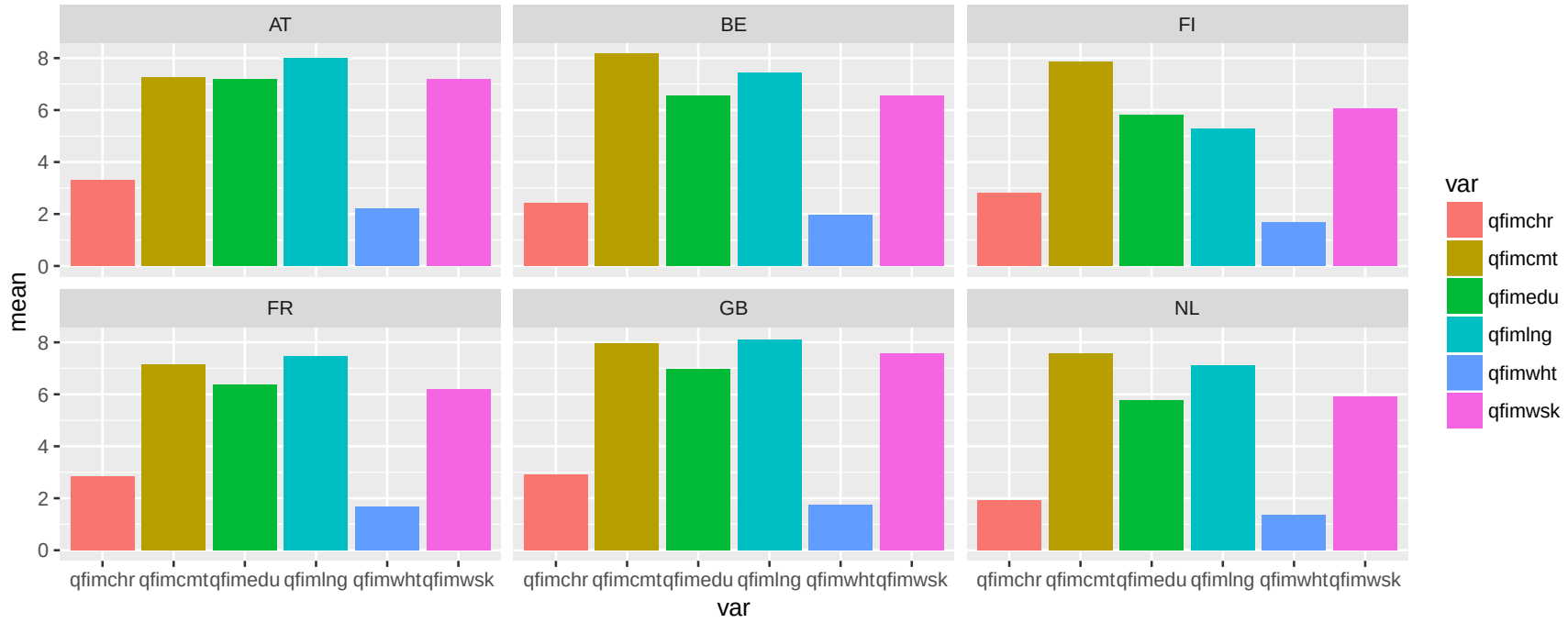
Renuméroter des axes

```
p + scale_y_continuous(breaks = seq(0, 8))
```



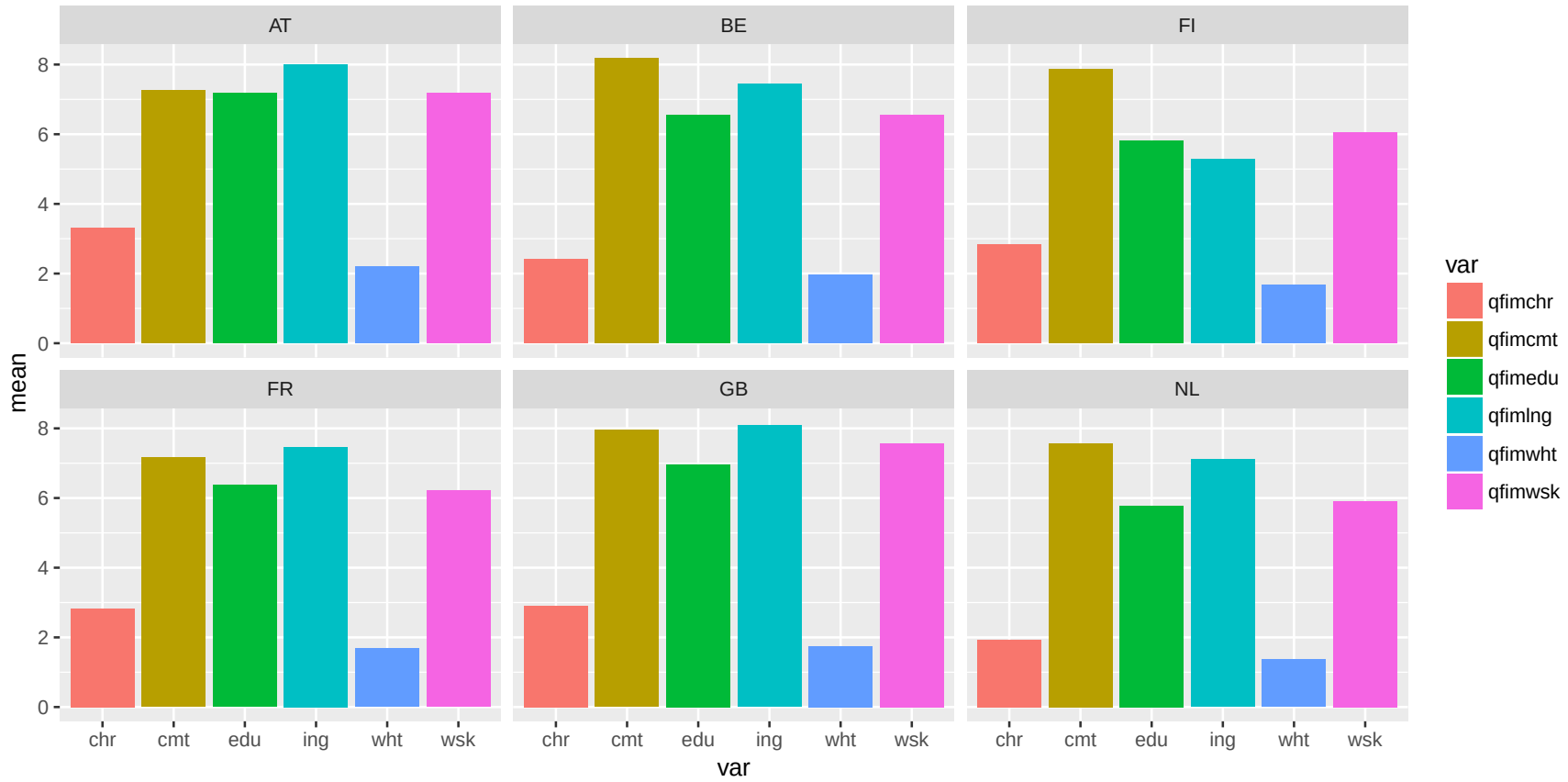
Changer des valeurs sur des axes discrets

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



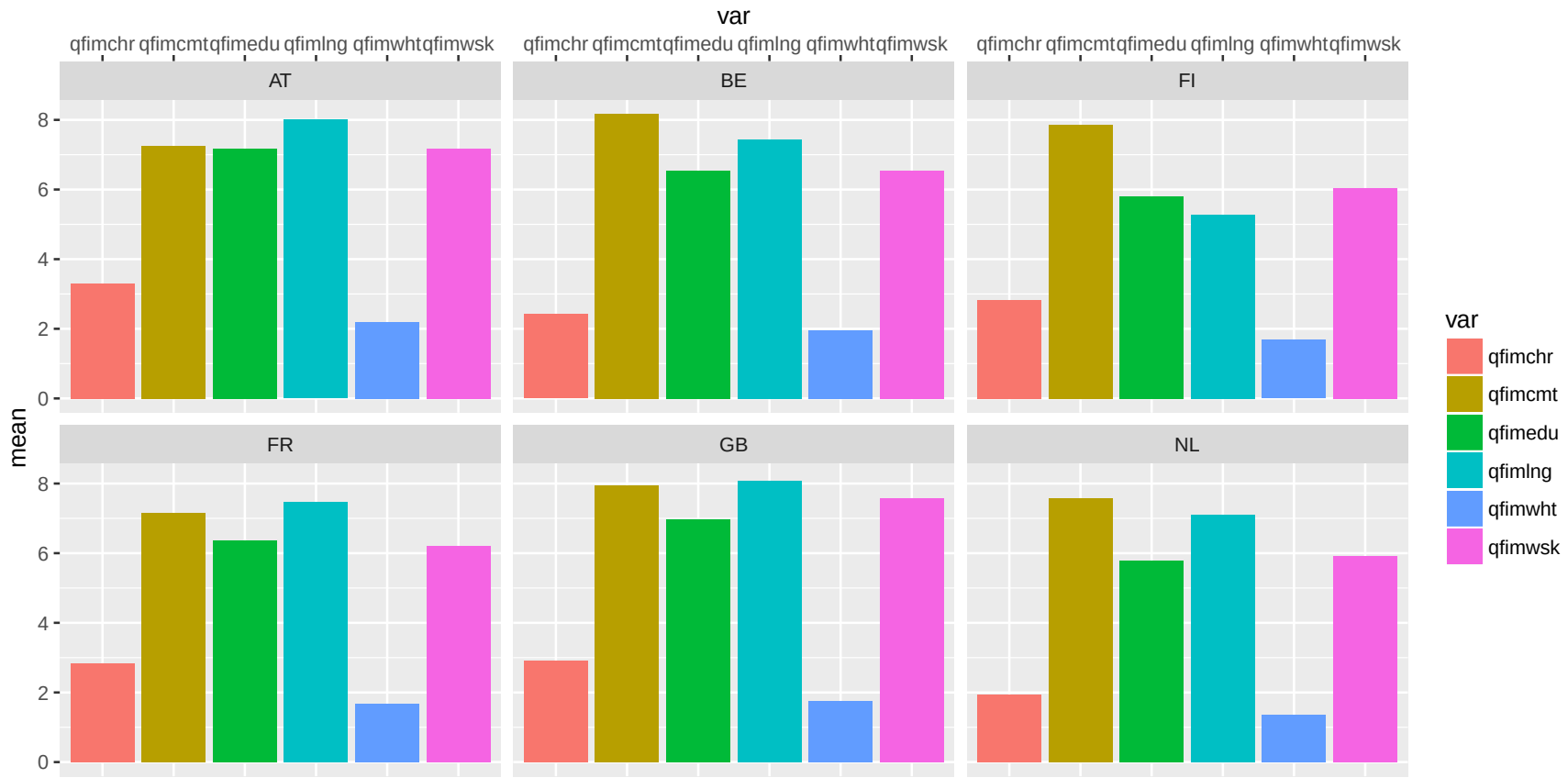
Changer des valeurs sur des axes discrets

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



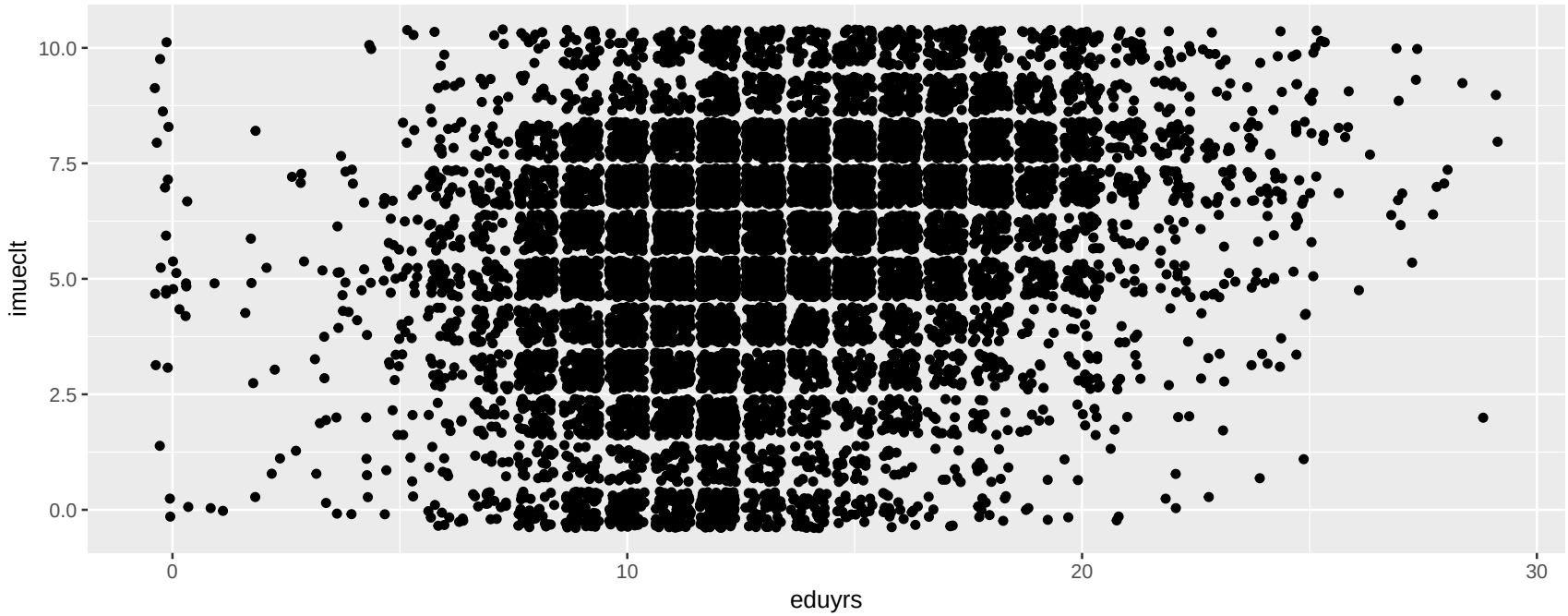
Modifier le placement d'un axe

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



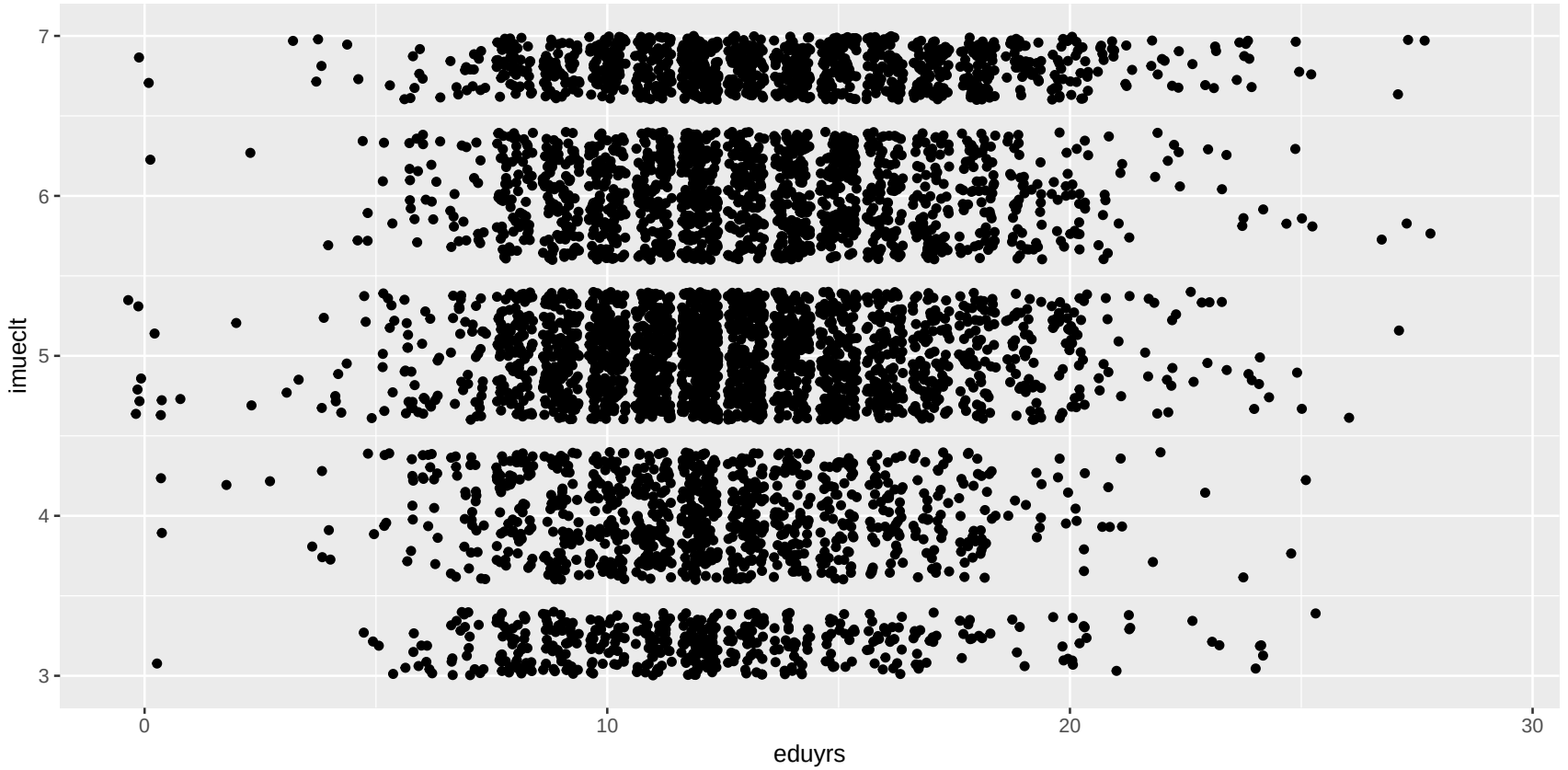
Zoom : limiter un axe

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



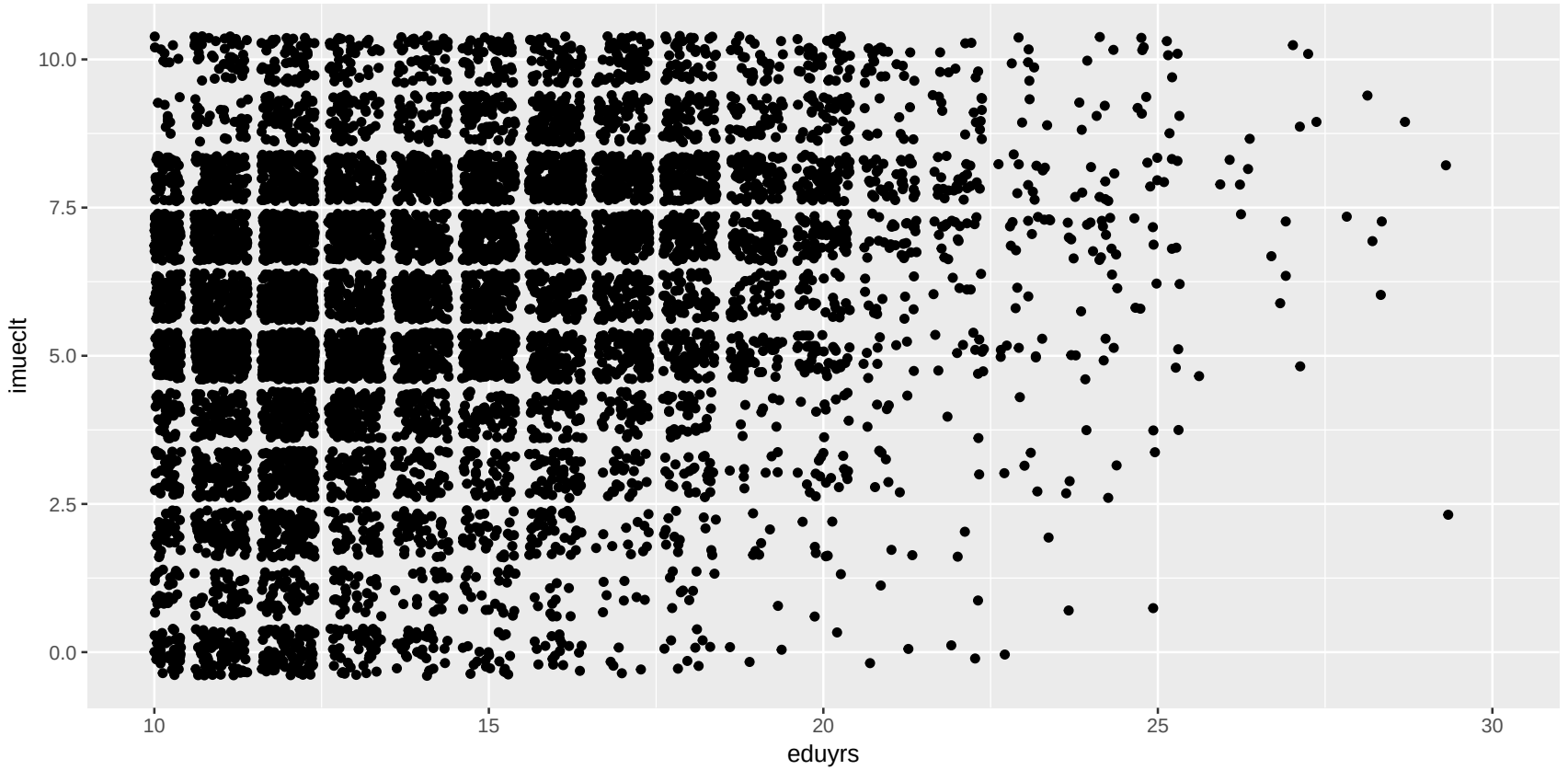
Zoom : limiter un axe

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



Zoom : limiter un axe

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



Changer les couleurs

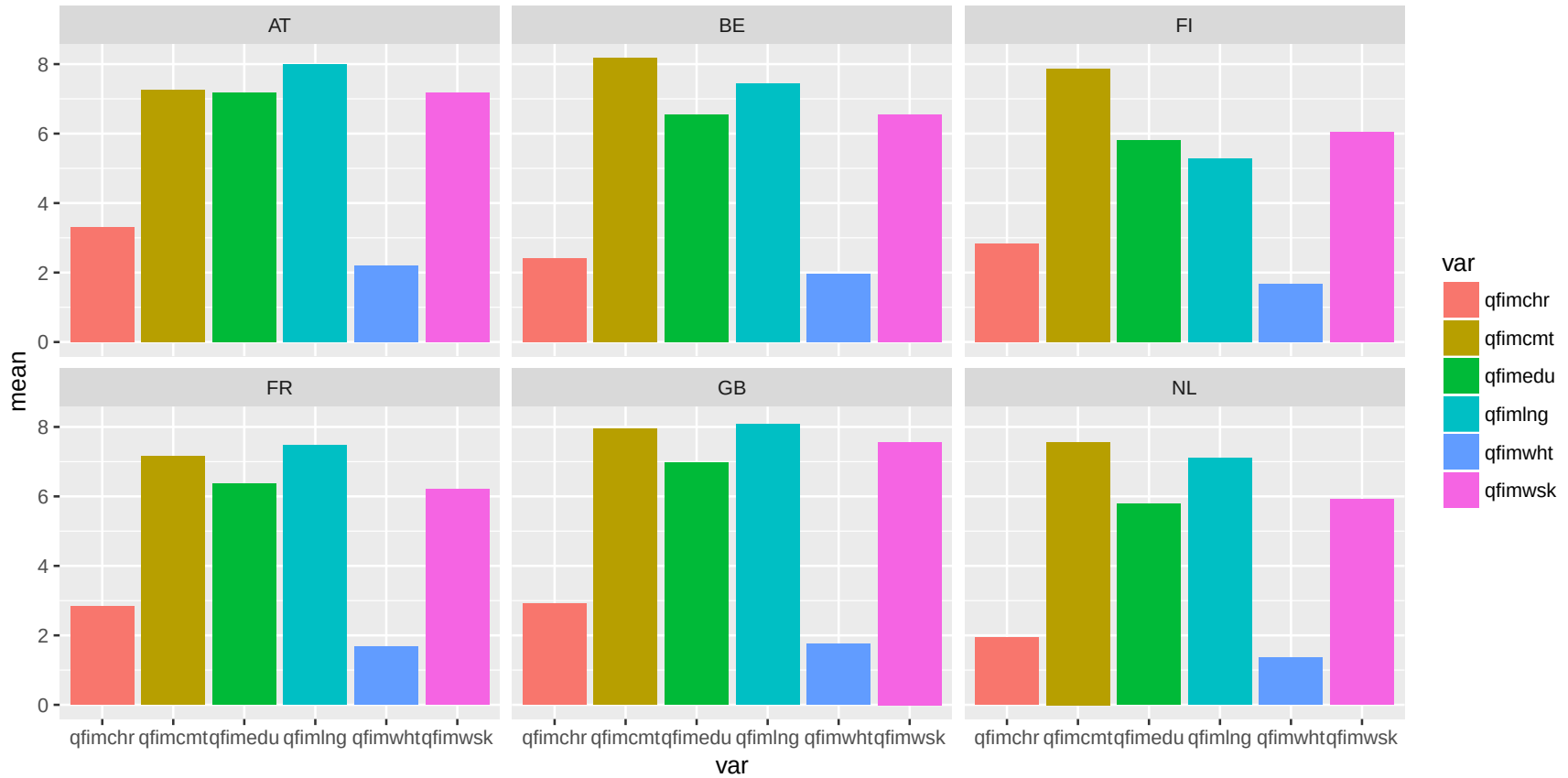
Il est possible de spécifier des couleurs à la main, mais la solution préférable est d'utiliser colorbrewer et son implémentation R.

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.

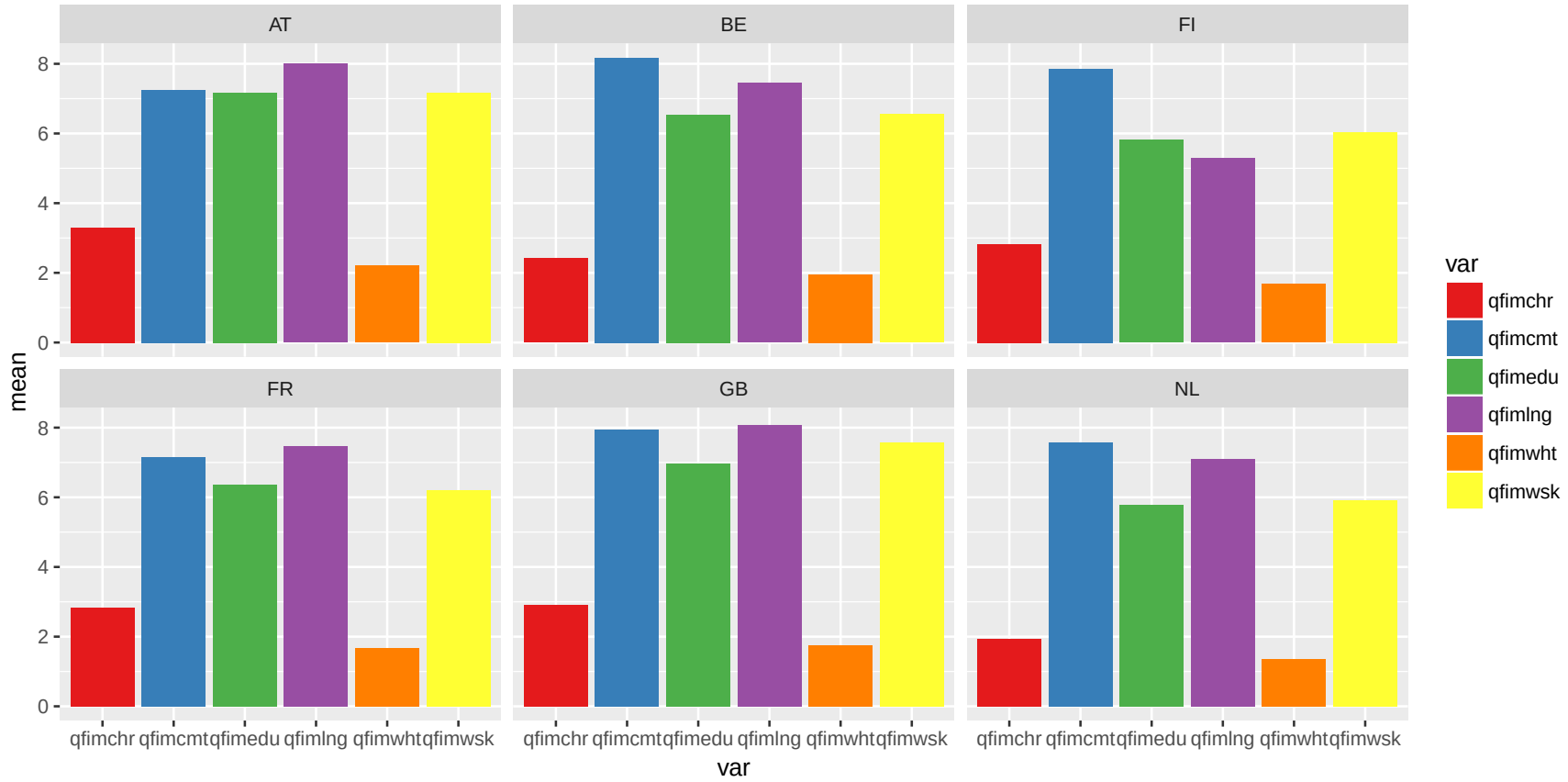
Changer les couleurs

```
p
```



Changer les couleurs

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



Themes et titres

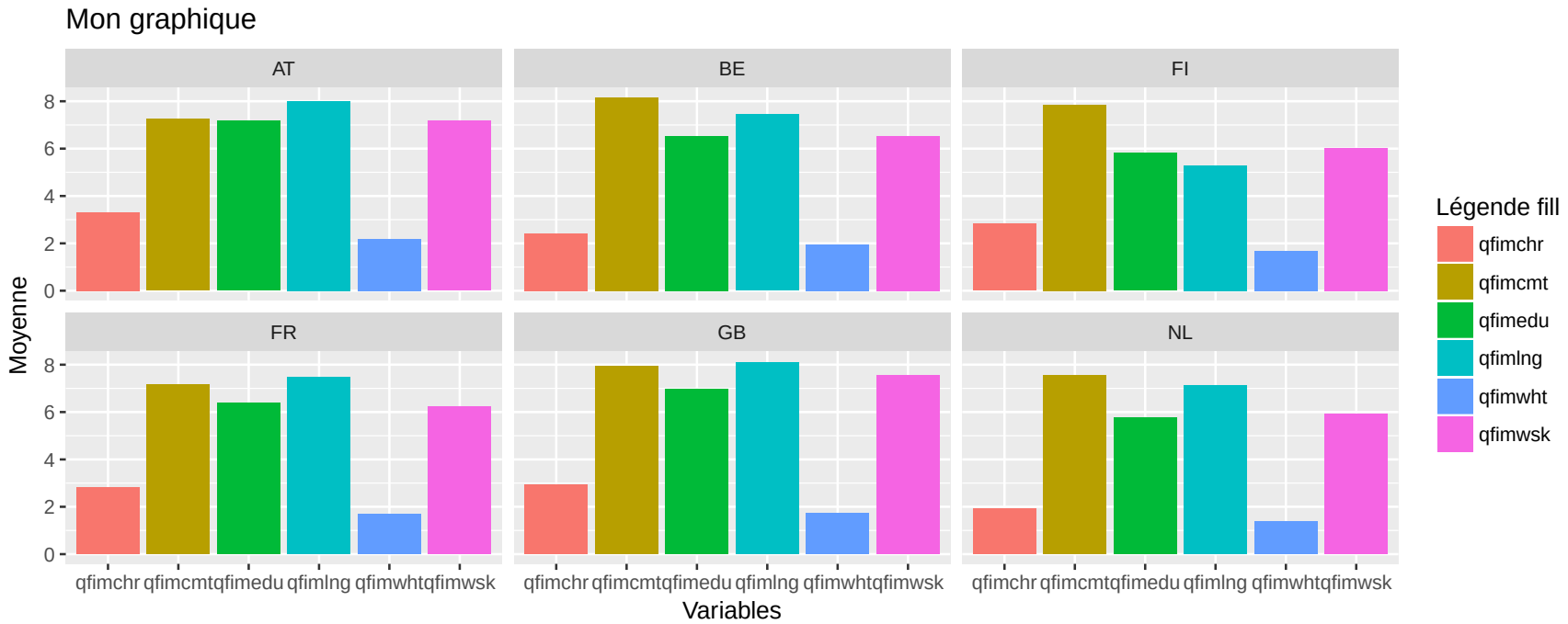
Titres

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 fill",
         title = "Mon graphique")
```



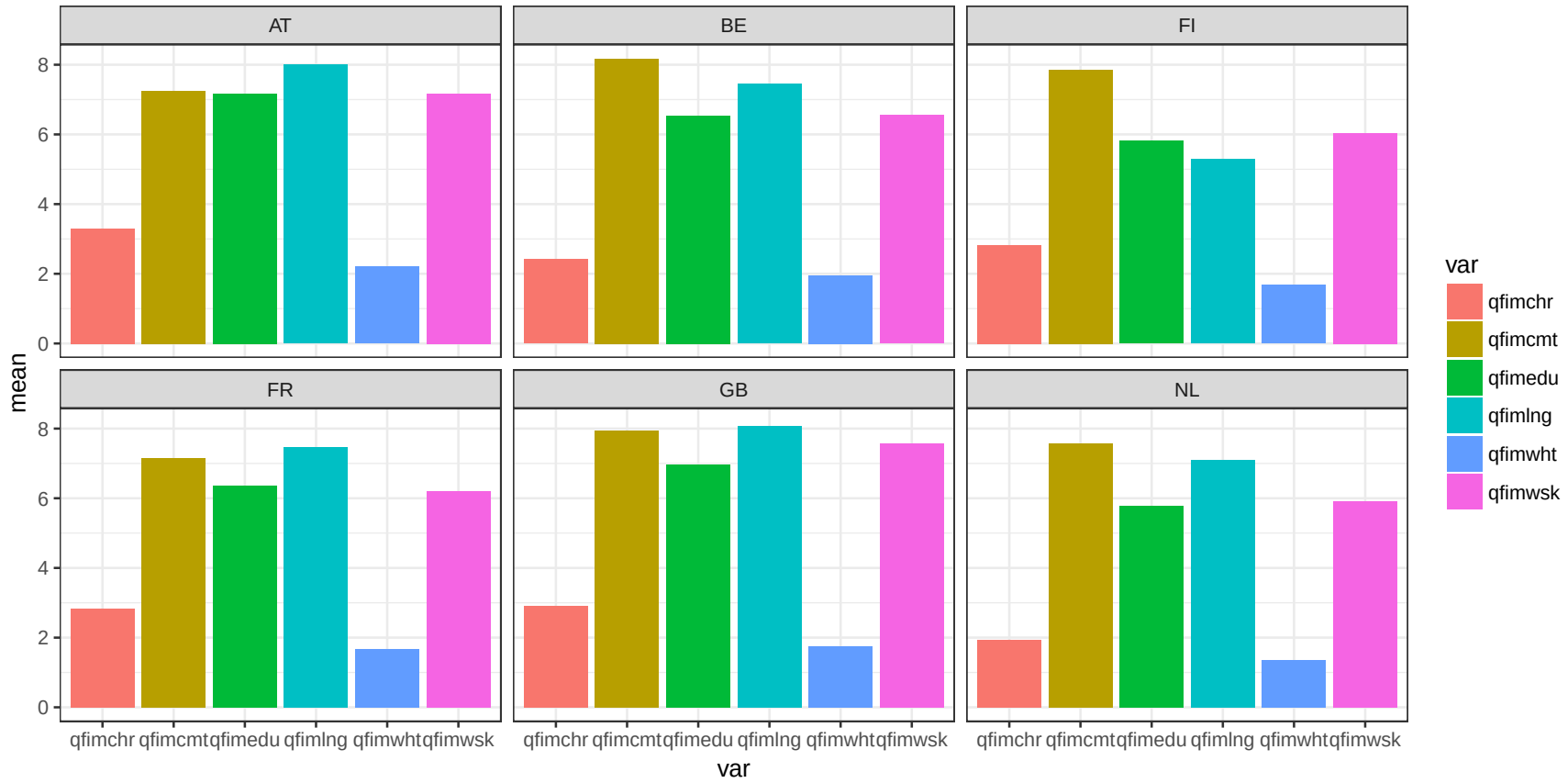
Thèmes

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()
```

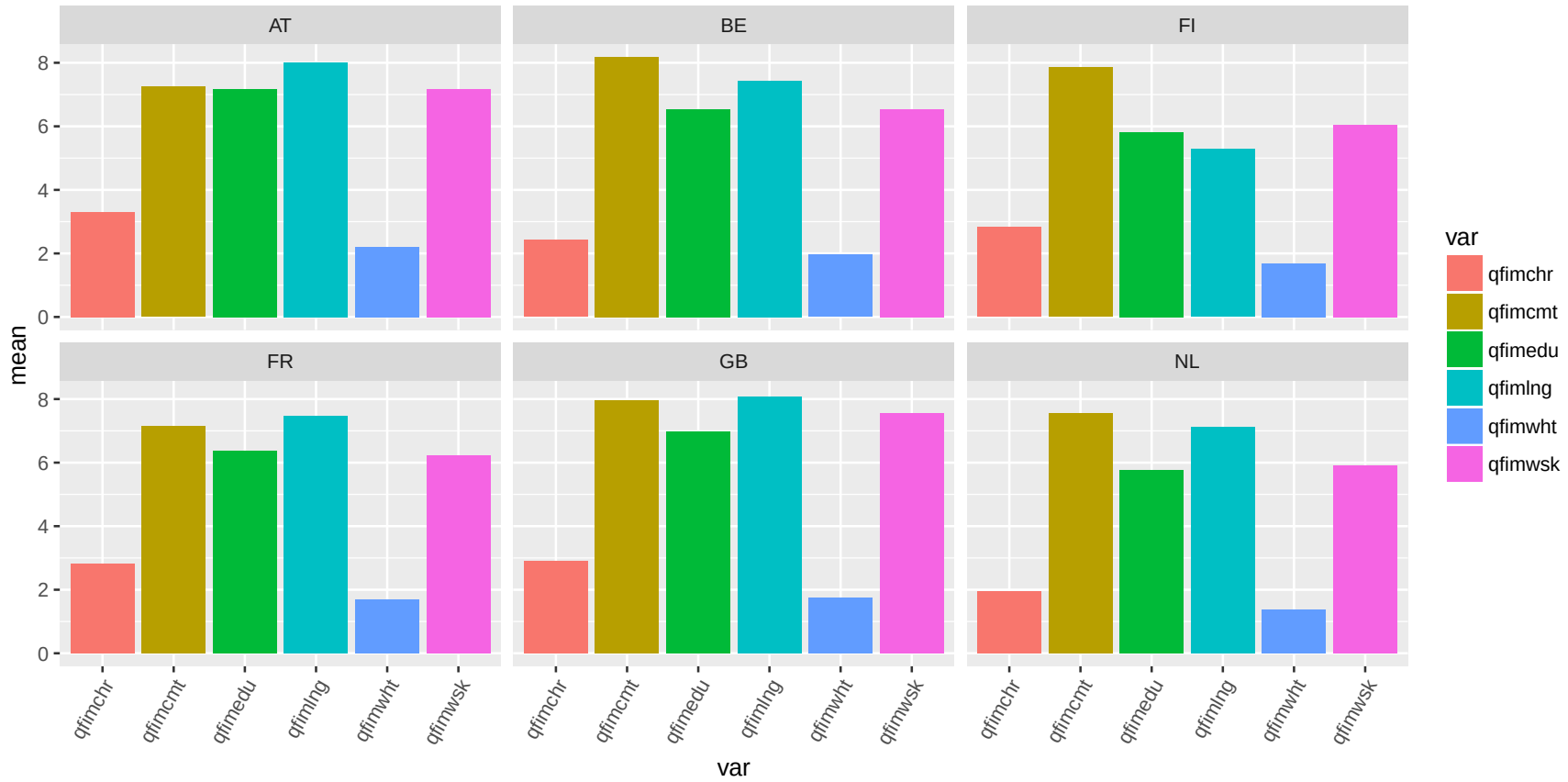


Theme : personnaliser

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.

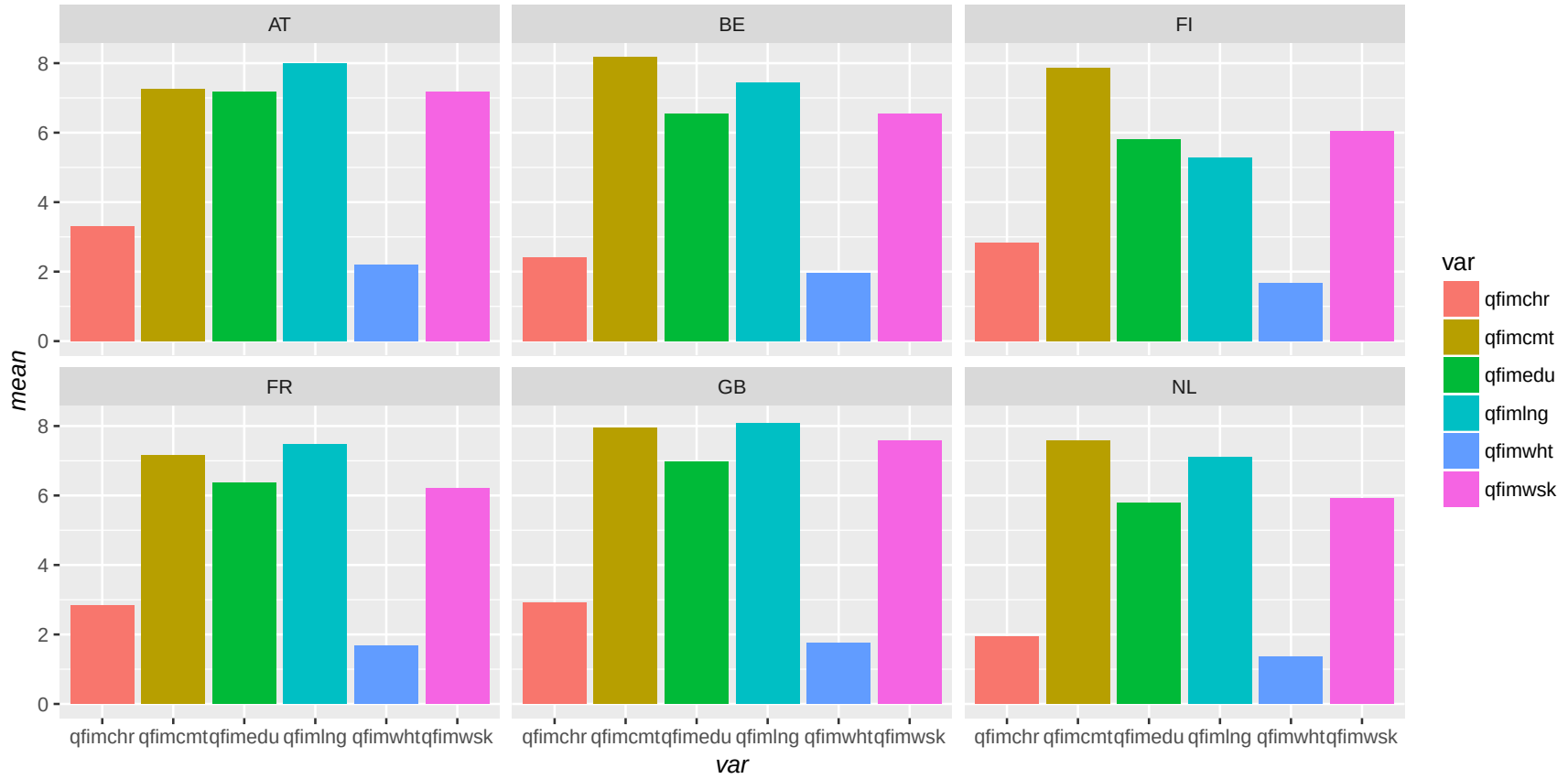
Labels des axes

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



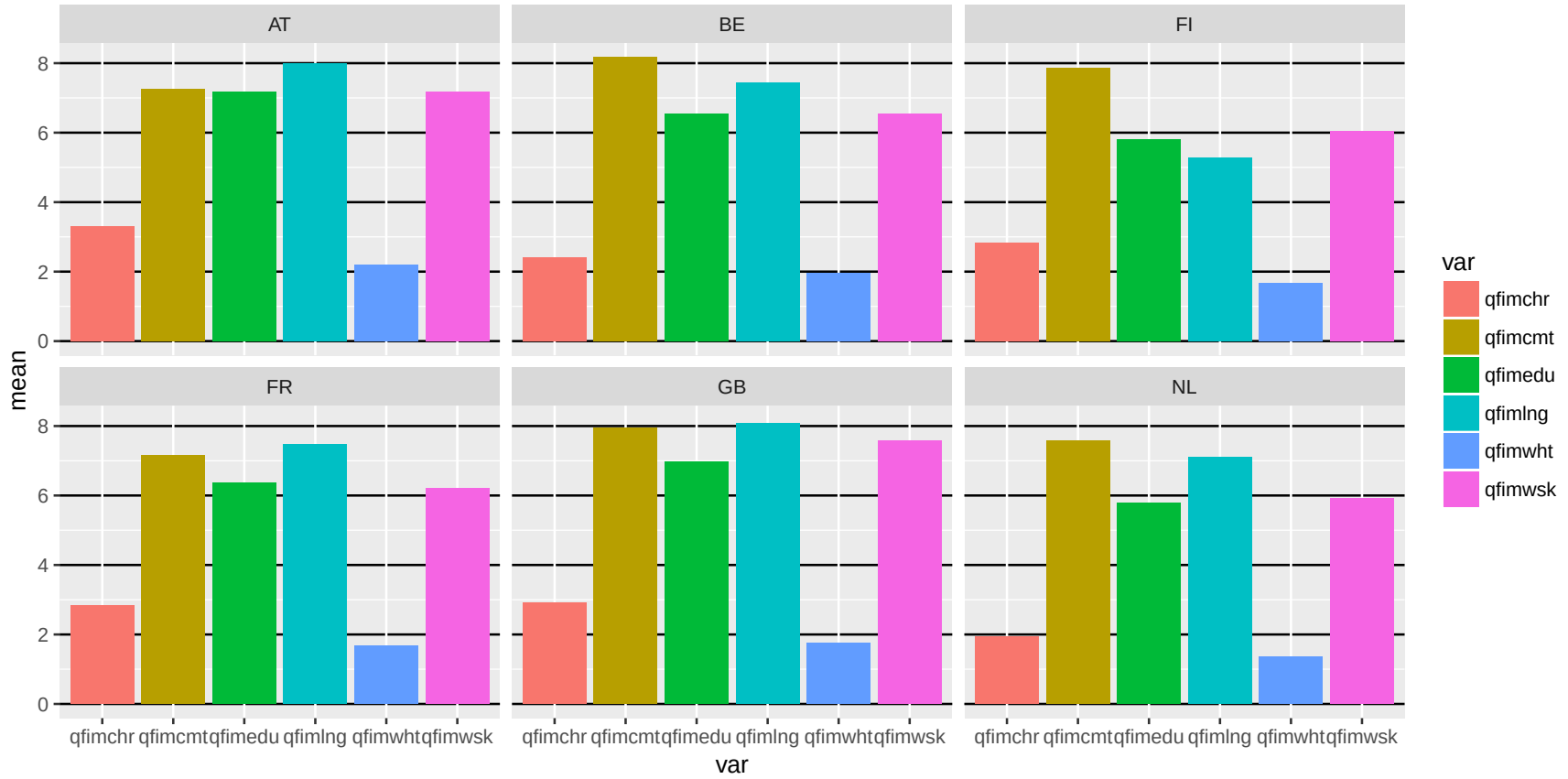
Labels des axes

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



Grille d'arrière plan

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

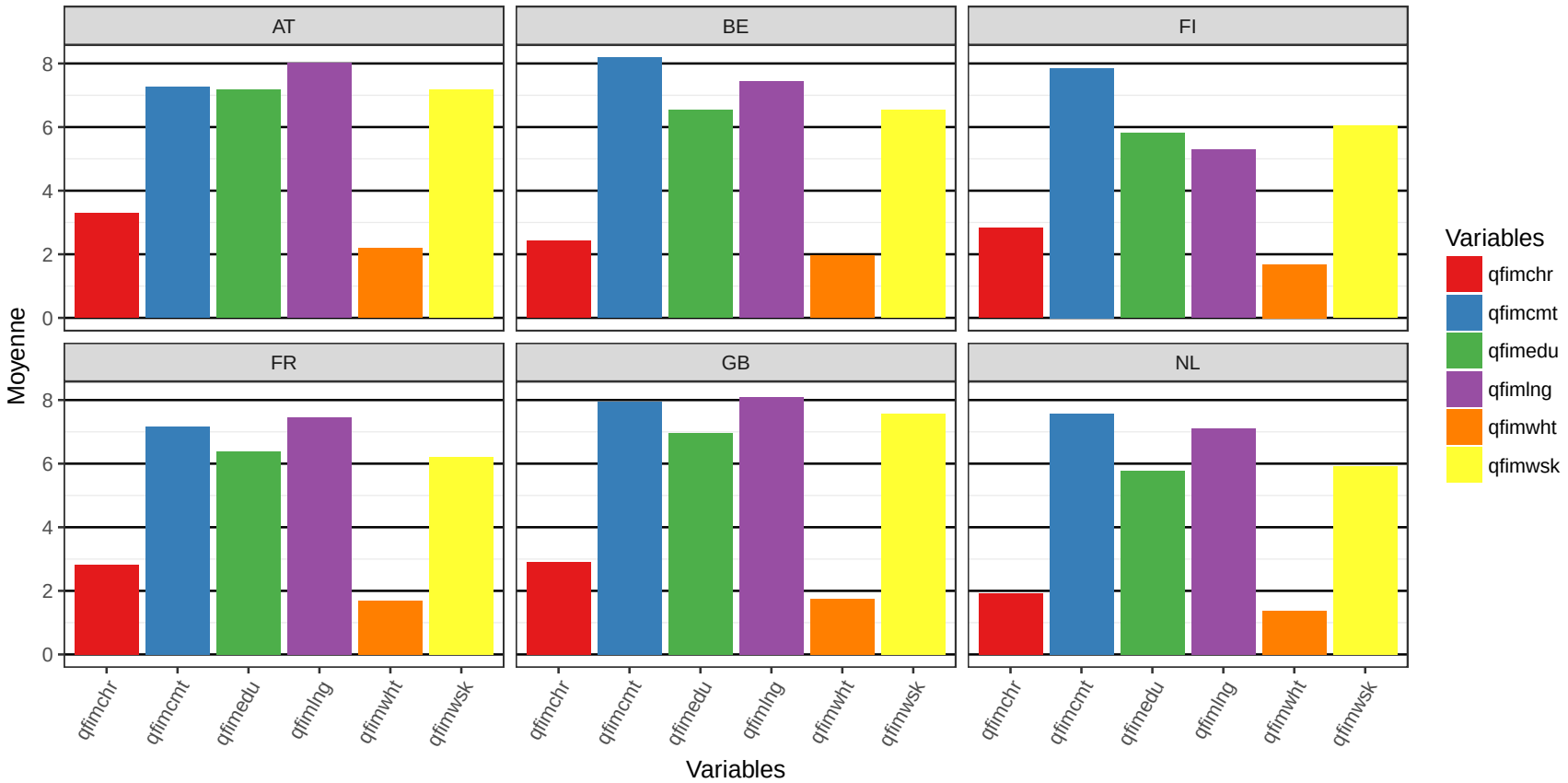


Un plot complet

```
# 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



Bibliographie I

Wickham, Hadley. 2010. « A Layered Grammar of Graphics ». *Journal of Computational and Graphical Statistics* 19 (1). Informa UK Limited : 3-28. doi :10.1198/jcgs.2009.07098.

Wilkinson, Leland. 2005. *The grammar of graphics*. New York : Springer.