Visualisation avec ggplot2

Malick SENE

2025-04-14

```
library(ggplot2)
library(haven)
wf <- read_dta("ehcvm_welfare_sen2021.dta")
```

```
library(tidyverse)
```

```
## Warning: le package 'tidyverse' a été compilé avec la version R 4.3.3

## Warning: le package 'readr' a été compilé avec la version R 4.3.3

## Warning: le package 'dplyr' a été compilé avec la version R 4.3.3

## Warning: le package 'lubridate' a été compilé avec la version R 4.3.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v lubridate 1.9.3     v tibble    3.2.1
## v purrr     1.0.2     v tidyr     1.3.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
wf <- wf %>%
  mutate(across(where(~ is.character(.) || haven::is.labelled(.)), as_factor))
```

4. Visualisation avec ggplot2

La visualisation des données est une étape essentielle dans l'analyse statistique. Elle permet non seulement d'explorer les données de manière intuitive, mais aussi de communiquer efficacement les résultats. Le package ggplot2, composant clé du tidyverse, offre un cadre puissant, cohérent et extensible pour créer des graphiques en R. Fondé sur la "Grammaire des graphiques" (The Grammar of Graphics), ggplot2 repose sur une logique de couches successives où chaque élément graphique (les points, les lignes, les axes, les couleurs, etc.) peut être construit indépendamment puis combiné harmonieusement.

Cette section propose une exploration structurée de ggplot2, depuis l'initialisation d'un graphique jusqu'à sa personnalisation avancée. Nous y présenterons divers types de géométries, les principes de mappage esthétique, l'utilisation simultanée de plusieurs couches, le faceting pour des comparaisons multi-groupes, la gestion fine des échelles (scale_*), ainsi que les thèmes pour un rendu esthétique.

4.1 Initialisation

Un graphique ggplot2 commence toujours par l'appel à la fonction ggplot(), à laquelle on fournit une source de données (data =).

```
ggplot(data= wf)
```



```
 ##Ou,équivalent
ggplot(wf)
```

Une fois la **source de données définie** dans un graphique `ggplot`, il convient d'ajouter des **éléments de représentation visuelle**, appelés `geom`.

Ces éléments graphiques sont ajoutés à l'objet `ggplot` de base à l'aide de l'opérateur `+`.

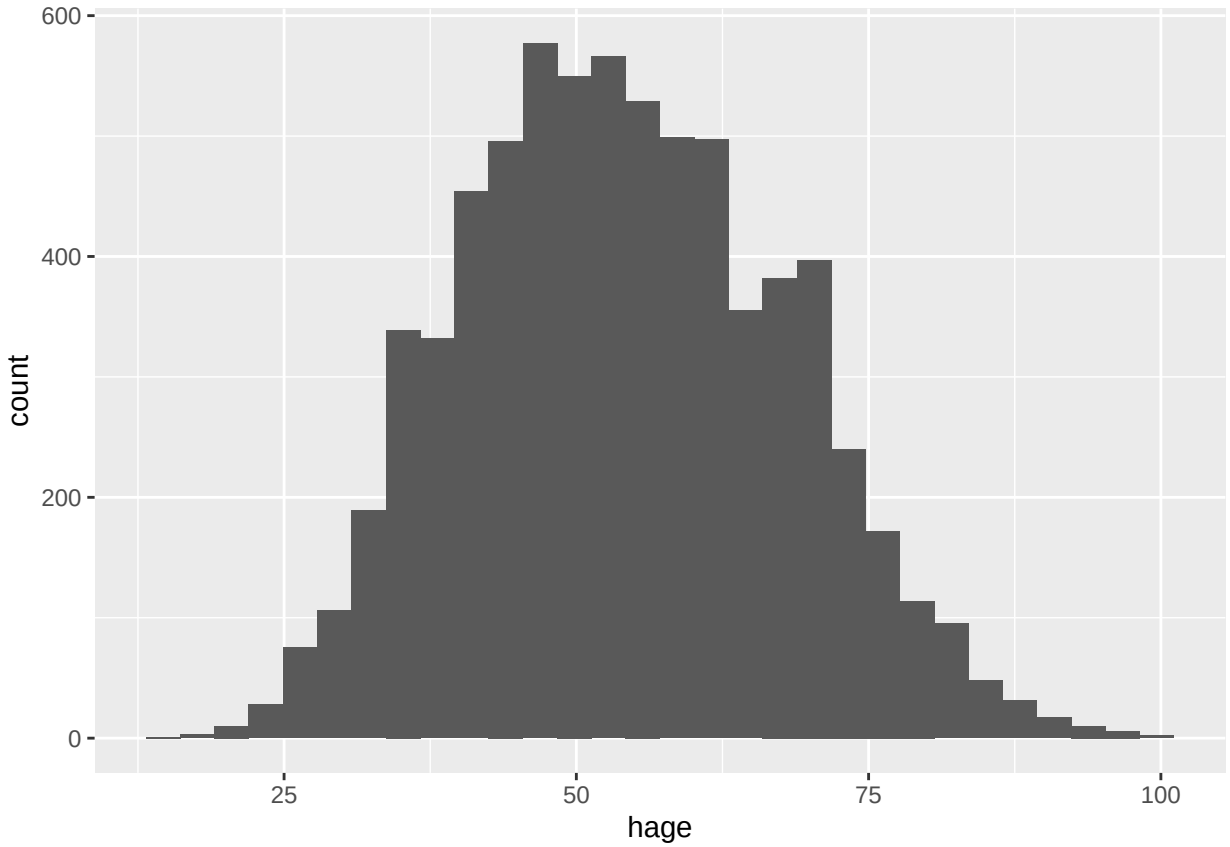
L'un des `geom` les plus simples est `geom_histogram()`, qui permet de représenter la **distribution d'une variable numérique**.

```
ggplot(wf) +
  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(wf) +
  geom_histogram(aes(x = hage))
```

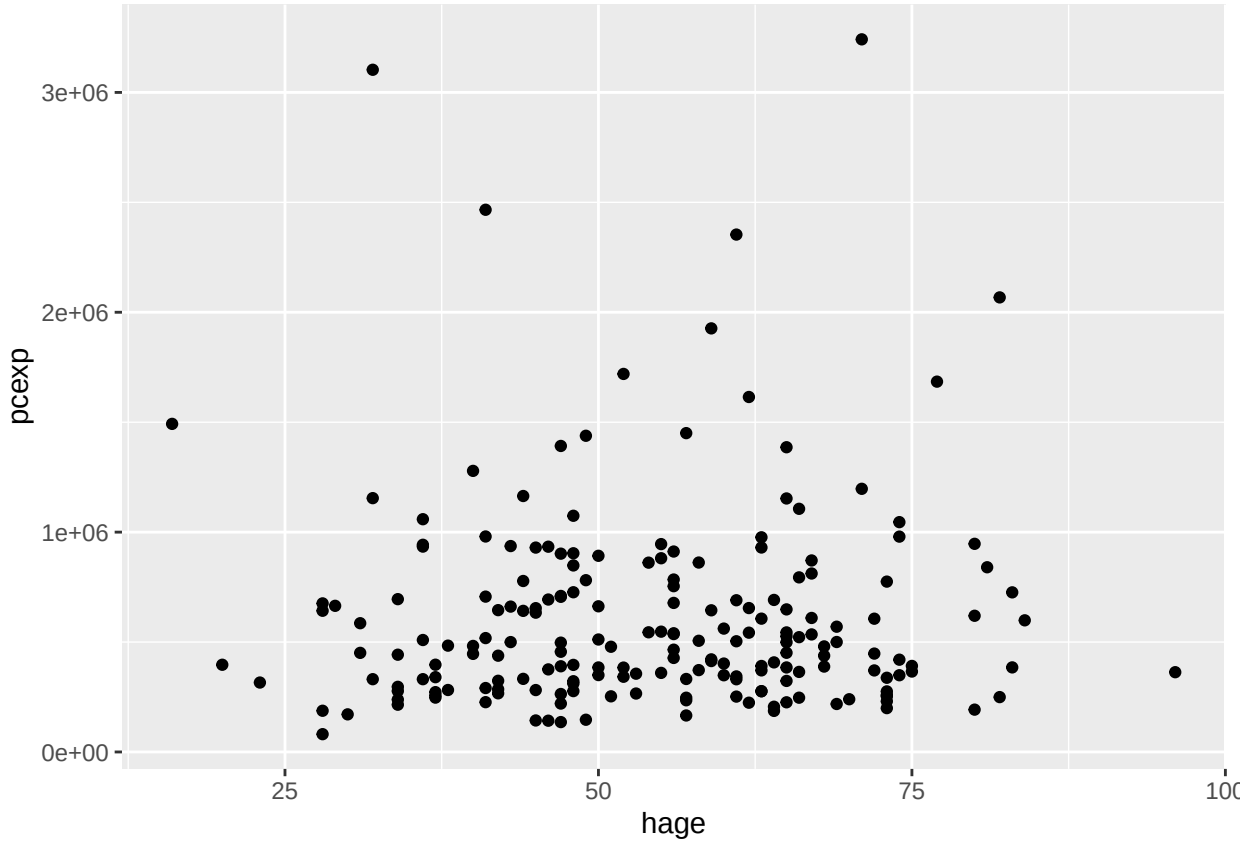
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Si on veut représenter une autre variable, il suffit de changer la valeurde x. Quand on spécifie une variable, il est inutile d'indiquer le nom du tableau de données sous la forme `df$var`,
car `ggplot2` recherche automatiquement la variable dans le tableau de données indiqué avec le paramètre `data`.

Certains `geom` prennent plusieurs paramètres. Ainsi, si l'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()` :

```
# Prendre un échantillon aléatoire de 200 observations pour ne pas surcharger la visualisation
wf_sample <- wf %>%
  sample_n(200)
ggplot() +
  geom_point(data = wf_sample, aes(x = hage, y = pcexp))
```

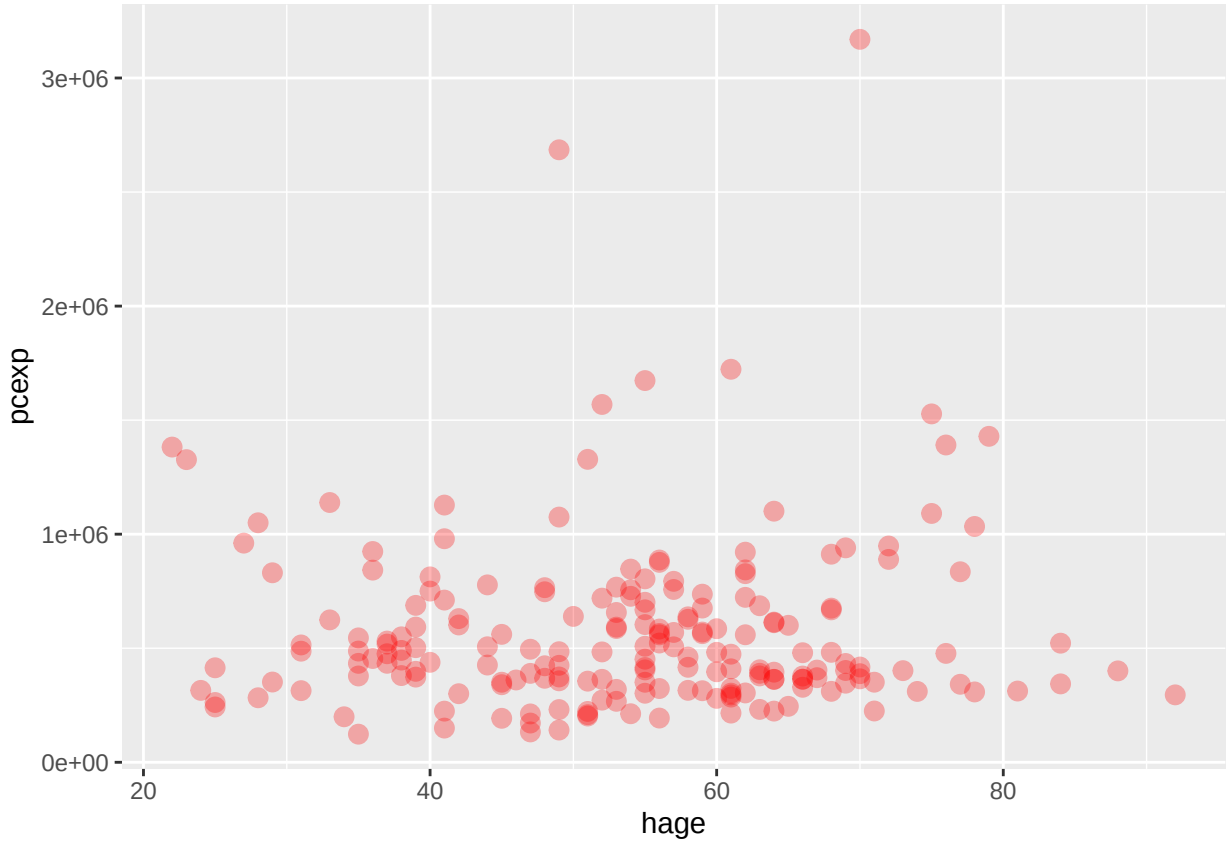


Ici, chaque point représente un ménage, positionné selon l'âge du chef de ménage (hage) et le niveau de bien-être (pcexp).

On peut modifier certains 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`,
- leur **transparence** avec l'argument `alpha` :

```
# Prendre un échantillon aléatoire de 200 observations pour ne pas surcharger la visualisation
wf_sample <- wf %>%
  sample_n(200)
ggplot(data = wf_sample) +
  geom_point( aes(x = hage, y = pcexp), color= "red",size= 3,alpha= 0.3)
```



4.2 Exemples de geom

Dans ggplot2, les éléments visuels d'un graphique sont appelés géométries (abrégées geom_*). Chaque type de graphique correspond à une géométrie spécifique. On les ajoute comme des couches avec le symbole + après l'appel à ggplot().

Nous présentons ici quelques-unes des géométries les plus courantes : Outre geom_histogram et geom_point, on pourra noter les geom suivants.

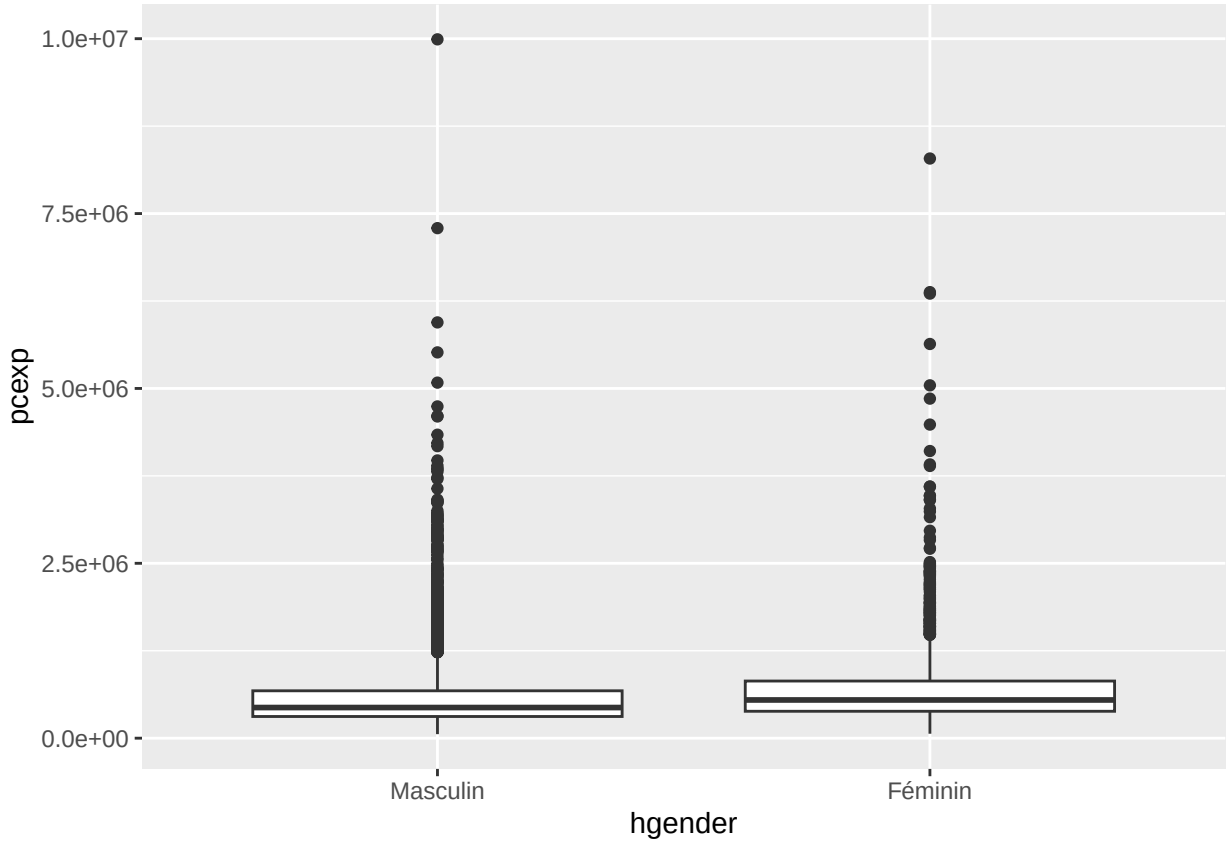
4.2.1 geom_boxplot

La fonction `geom_boxplot()` permet de représenter des **boîtes à moustaches** (*boxplots*), utiles pour visualiser la **distribution** d'une variable selon différentes **catégories**.

On lui passe :

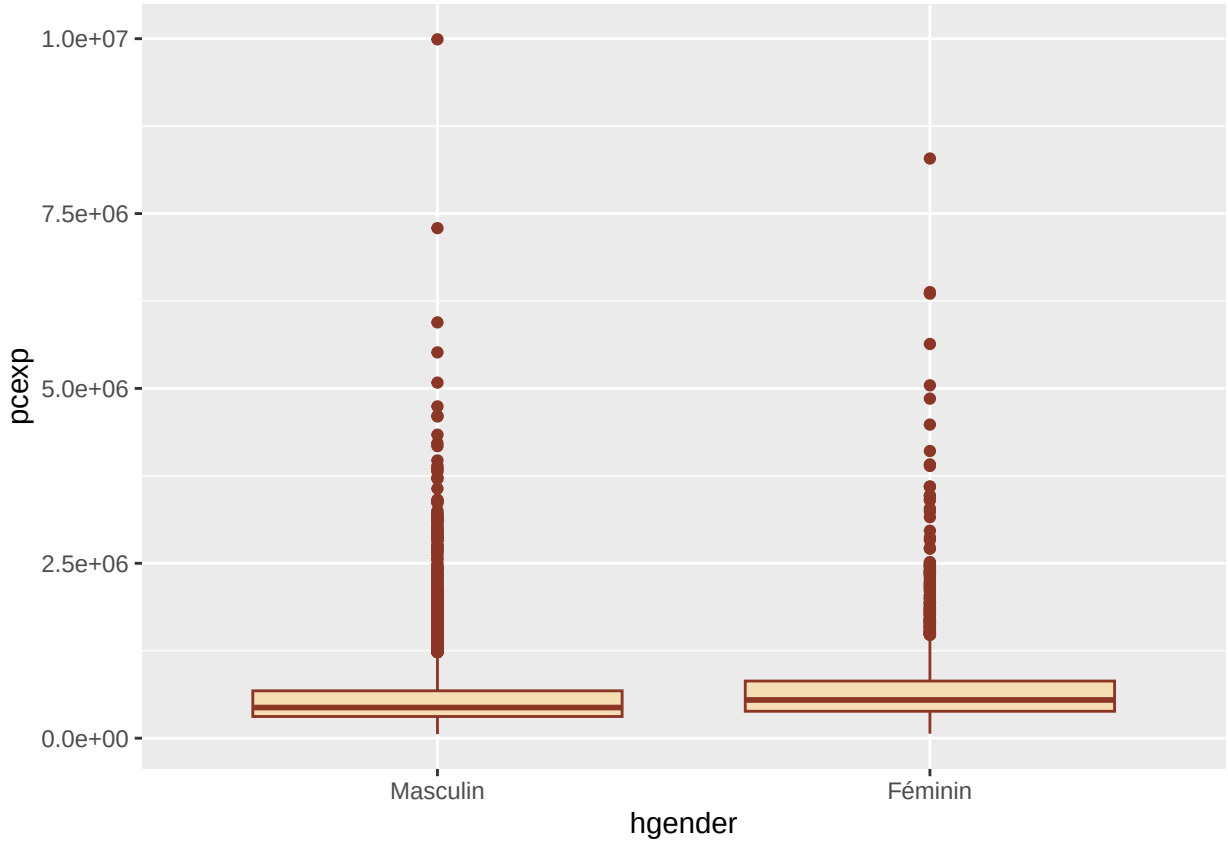
- en `y` : la variable quantitative dont on souhaite étudier la répartition,
- en `x` : la variable catégorielle contenant les classes à comparer.

```
ggplot(data = wf) +
  geom_boxplot(aes(x = hgender, y = pcexp))
```



On peut personnaliser la présentation avec différents argument supplémentaires :

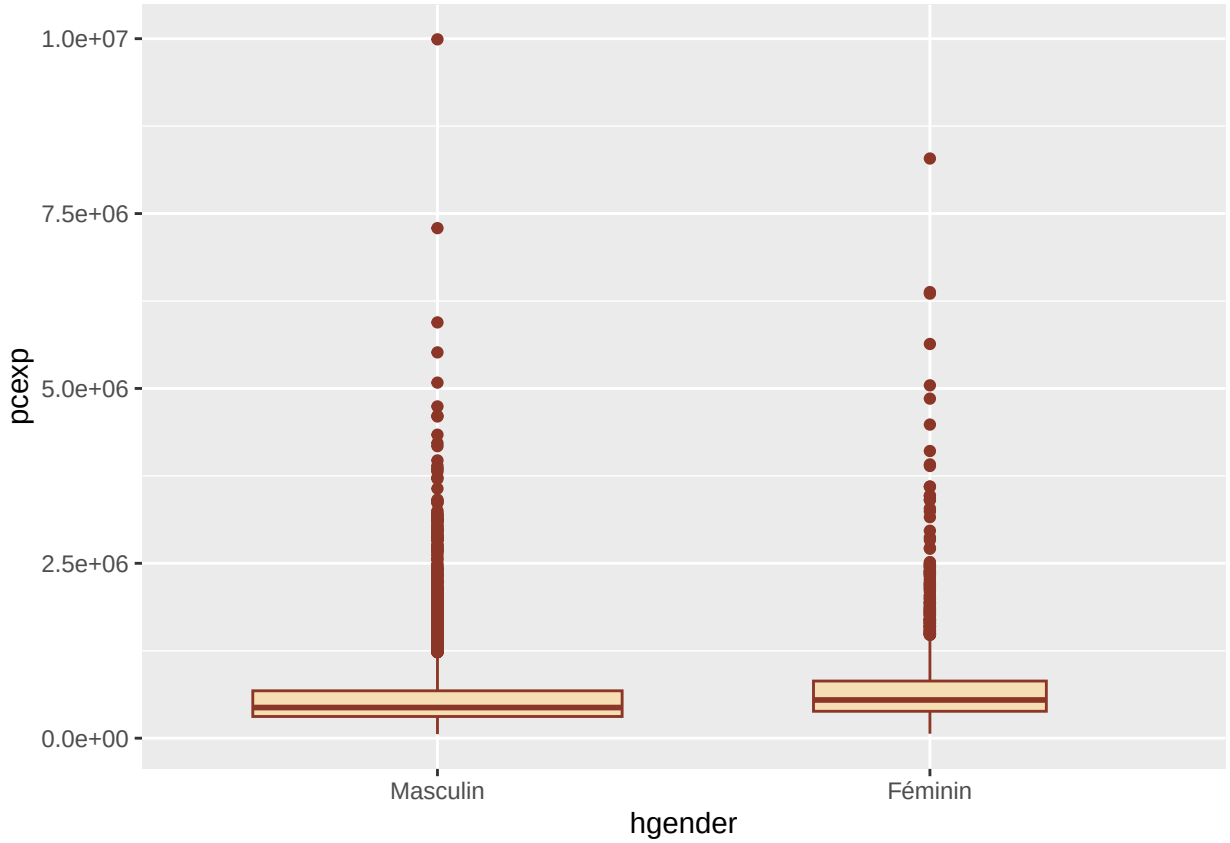
```
ggplot(data = wf) +
  geom_boxplot(aes(x = hgender, y = pcexp),  fill = "wheat", color = "tomato4")
```



Un autre argument utile dans `geom_boxplot()` est **`varwidth`**,
qui permet de **faire varier la largeur des boîtes** en fonction des **effectifs des classes**.

Cela signifie que chaque boîte sera plus ou moins large selon le **nombre d'observations** dans chaque modalité de la variable `x`.

```
ggplot(data = wf) +
geom_boxplot(aes(x = hgender, y = pcexp),  fill = "wheat", color = "tomato4", varwidth = TRUE)
```

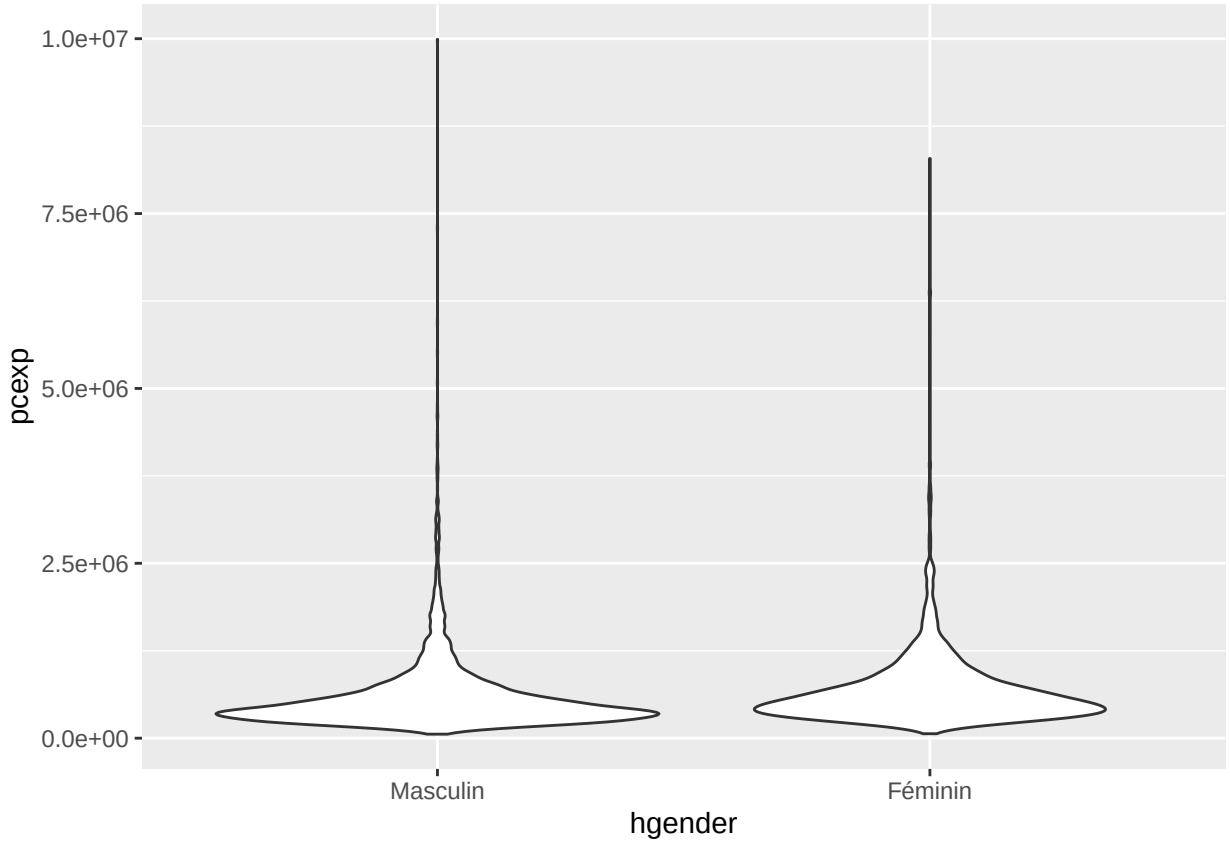


4.2.2 geom_violin

Semblable au `geom_boxplot()`, la fonction `geom_violin()` permet de visualiser la **distribution** d'une variable,
mais avec une représentation **plus détaillée** de sa densité.

Elle combine les avantages du **boxplot** et d'un **graphique de densité**, offrant ainsi une vue plus riche de la répartition des données.

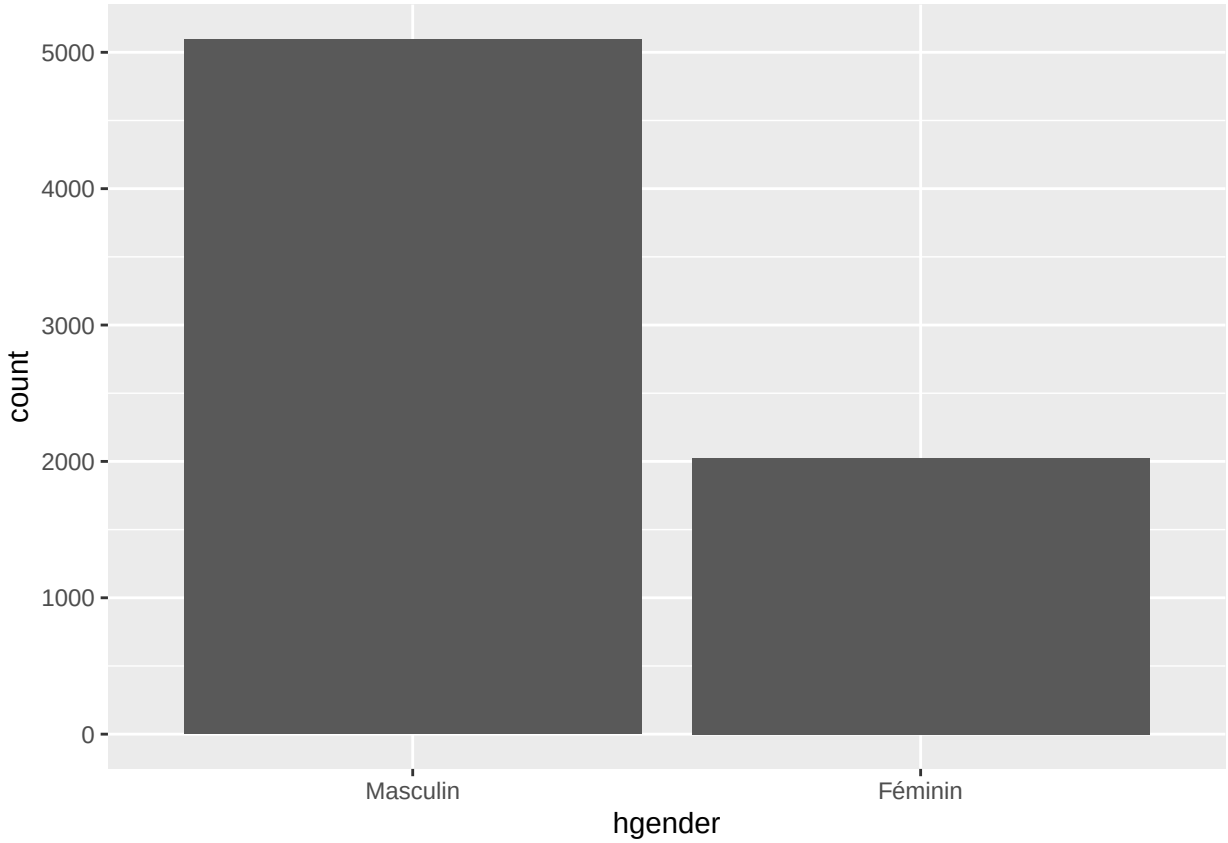
```
ggplot(wf) + geom_violin(aes(x = hgender, y = pcexp))
```



4.2.3 geom_bar

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

```
ggplot(data = wf ) +
  geom_bar(aes(x = hgender))
```



4.2.4 geom_text

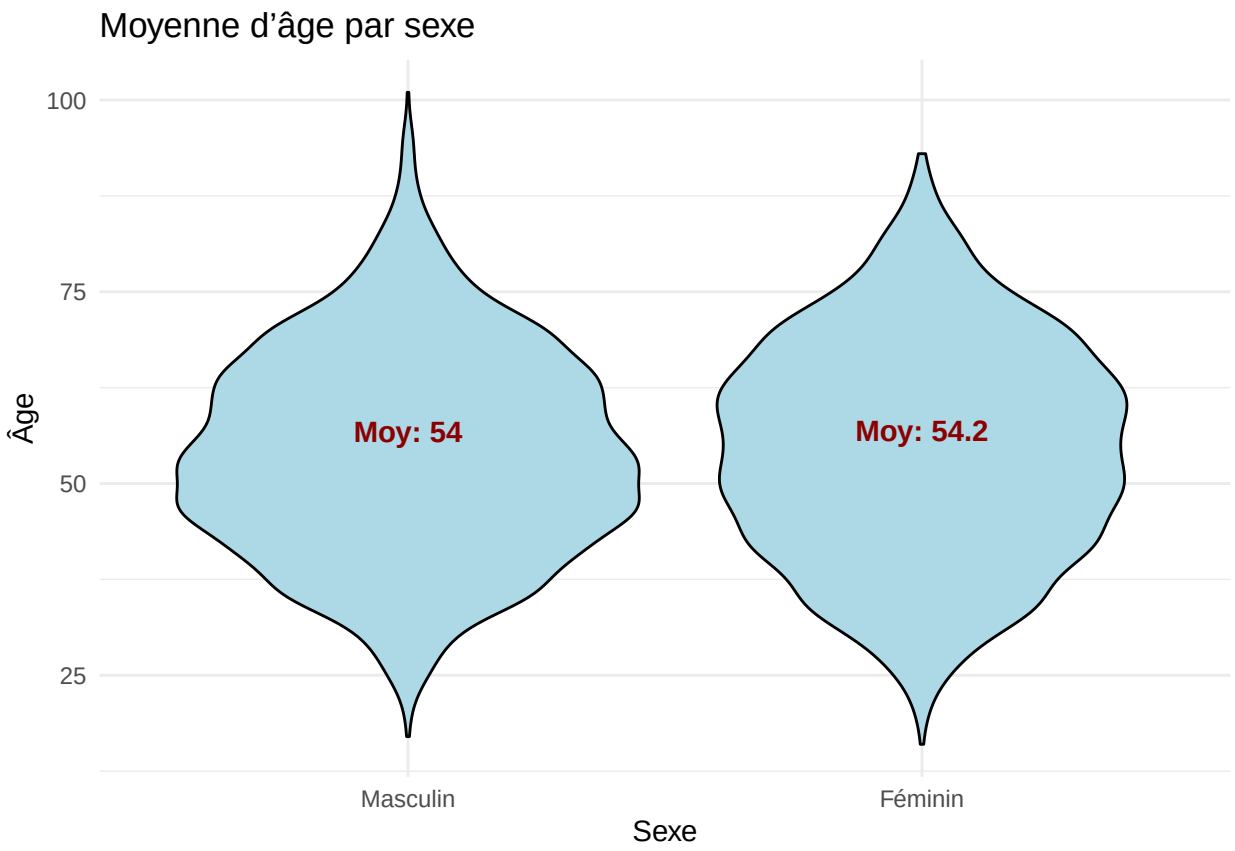
La fonction `geom_text()` permet de représenter des **points identifiés par des labels**.
Elle est utile pour **annoter un graphique** avec du texte à des coordonnées précises.

On doit lui passer :

- `x` : la position horizontale du point,
- `y` : la position verticale du point,
- `label` : le texte à afficher pour chaque point.

Exemple: Annoter la moyenne d'âge par sexe avec geom_text()

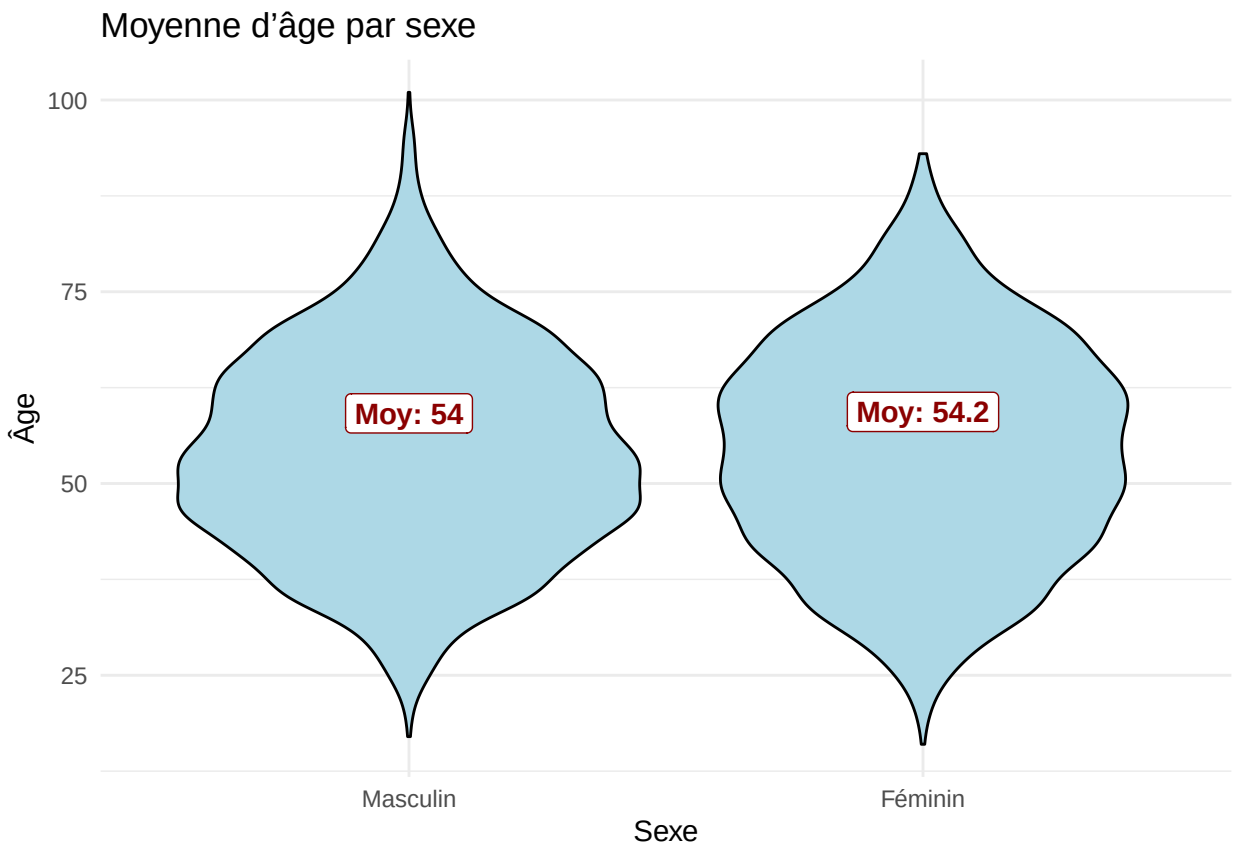
```
moyenne_age <- wf %>%
  group_by(hgender) %>%
  summarise(moy = mean(hage, na.rm = TRUE))

# Graphique avec geom_violin + geom_text

ggplot(data = wf, aes(x = hgender, y = hage)) +
  geom_violin(fill = "lightblue", color = "black") +
  geom_text(data = moyenne_age,
            aes(x = as_factor(hgender), y = moy, label = paste0("Moy: ", round(moy, 1))),
            color = "darkred", fontface = "bold", size = 4, vjust = -0.5) +
  labs(x = "Sexe", y = "Âge", title = "Moyenne d'âge par sexe") +
  theme_minimal()
```



4.2.5 geom_label

Similaire à geom_text(), mais le texte est entouré d’un cadre.

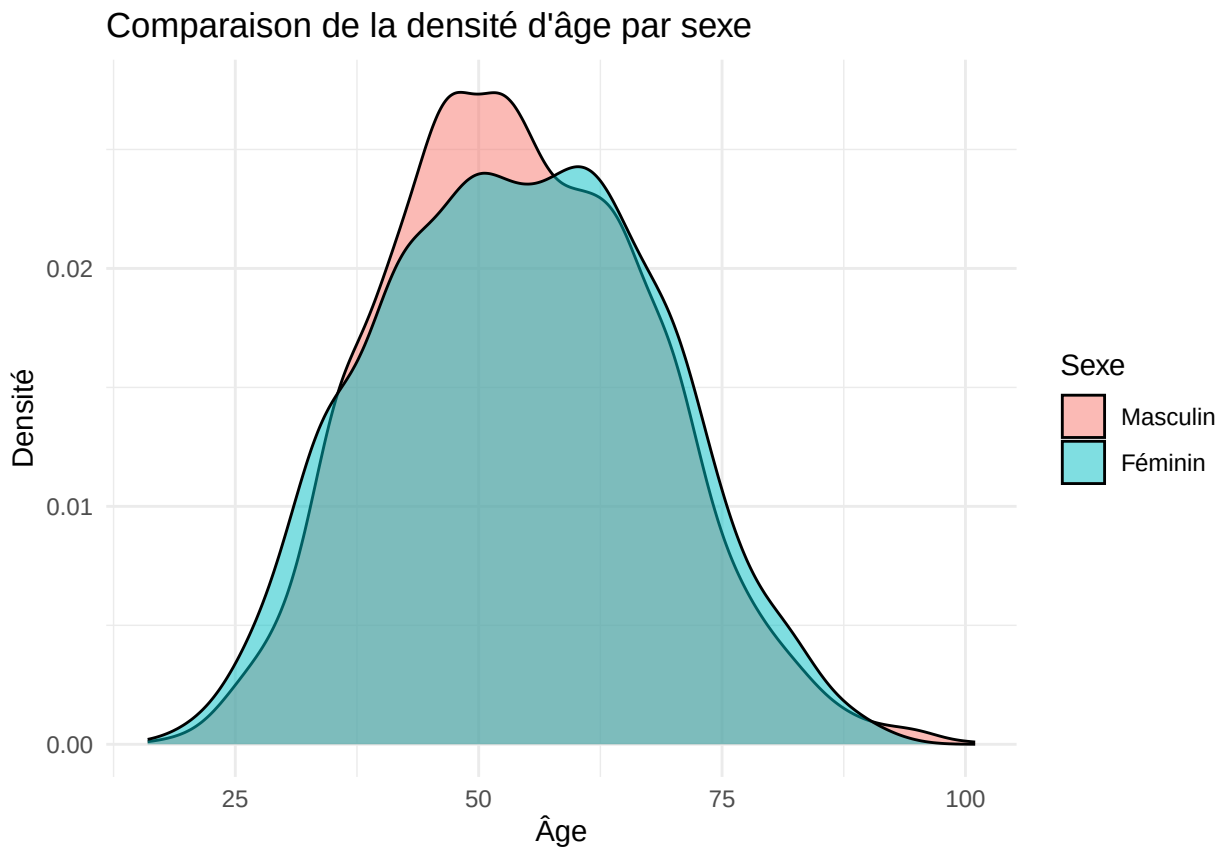
```
moyenne_age <- wf %>%
  group_by(hgender) %>%
  summarise(moy = mean(hage, na.rm = TRUE))

# Graphique avec geom_violin + geom_text

ggplot(data = wf, aes(x = hgender, y = hage)) +
  geom_violin(fill = "lightblue", color = "black") +
  geom_label(data = moyenne_age,
             aes(x = as_factor(hgender), y = moy, label = paste0("Moy: ", round(moy, 1))),
             color = "darkred", fontface = "bold", size = 4, vjust = -0.5) +
  labs(x = "Sexe", y = "Âge", title = "Moyenne d’âge par sexe") +
  theme_minimal()
```



4.2.6 geom_density

La fonction `geom_density()` permet d'afficher l'**estimation de densité** d'une variable numérique.
Son usage est **similaire à celui de `geom_histogram()`**, mais la densité est représentée **sous forme de courbe lissée**,
plutôt qu'en barres.

```
# Tracer la densité de l'âge par sexe
ggplot(wf, aes(x = hage, fill = hgender)) +
  geom_density(alpha = 0.5) +
  labs(title = "Comparaison de la densité d'âge par sexe",
       x = "Âge", y = "Densité", fill = "Sexe") +
  theme_minimal()
```



4.2.7 geom_line

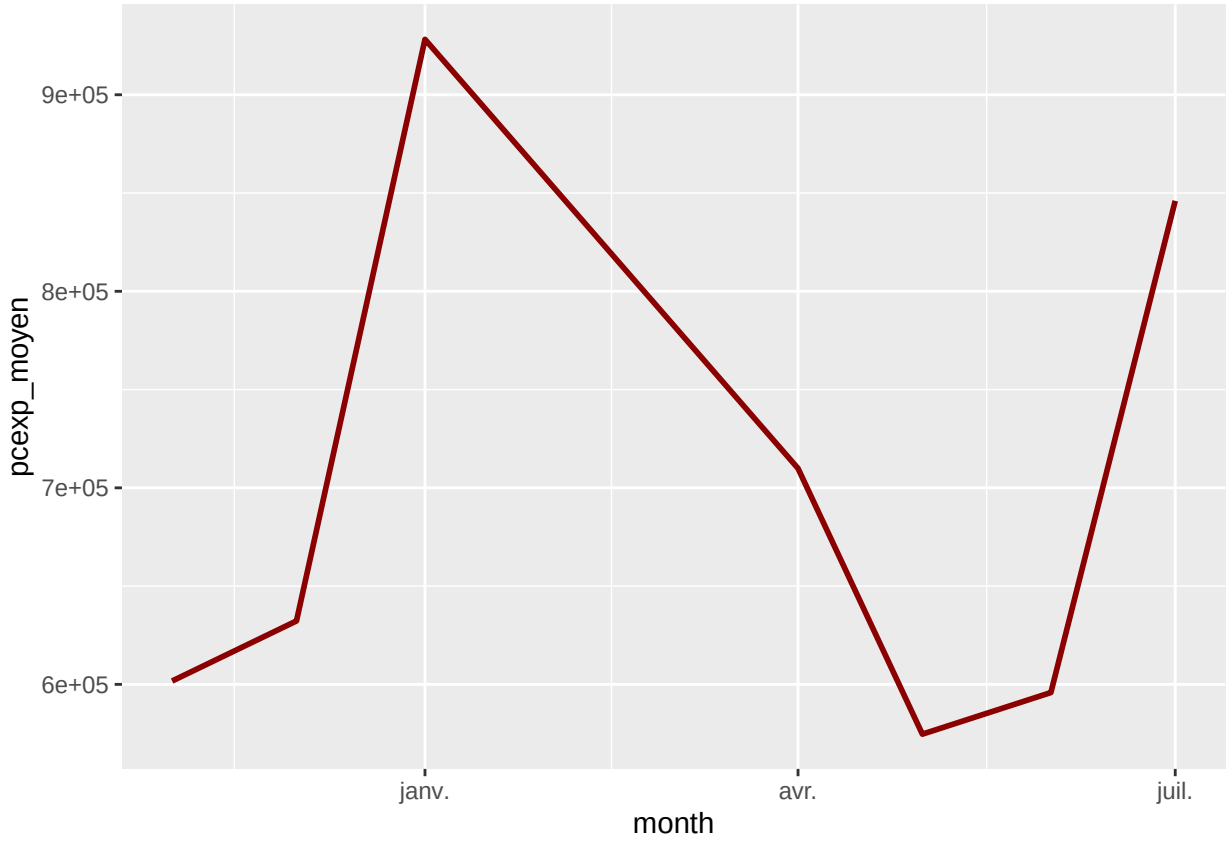
La fonction `geom_line()` trace des **lignes connectant les différentes observations entre elles**. Elle est notamment utilisée pour la **visualisation de séries temporelles**.

On lui passe deux paramètres principaux : - `x` : variable représentant le temps (ou l'ordre), - `y` : variable représentant la valeur à suivre.

Les points sont **connectés selon l'ordre des valeurs de `x`**.

```
# Calcul du niveau moyen de bien-être par mois
evolution <- wf %>%
  group_by(month) %>%
  summarise(pcexp_moyen = mean(pcexp, na.rm = TRUE))

# Graphique en ligne
ggplot(evolution, aes(x = month, y = pcexp_moyen)) +
  geom_line(color = "darkred", size = 1)
```

```
## Warning: Using ‘size‘ aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use ‘linewidth‘ instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```



4.3 Mappage

Le mappage permet d'associer des variables de la base de données à des éléments visuels du graphique : position (x, y), couleur (color, fill), taille (size), forme (shape), transparence (alpha), etc.

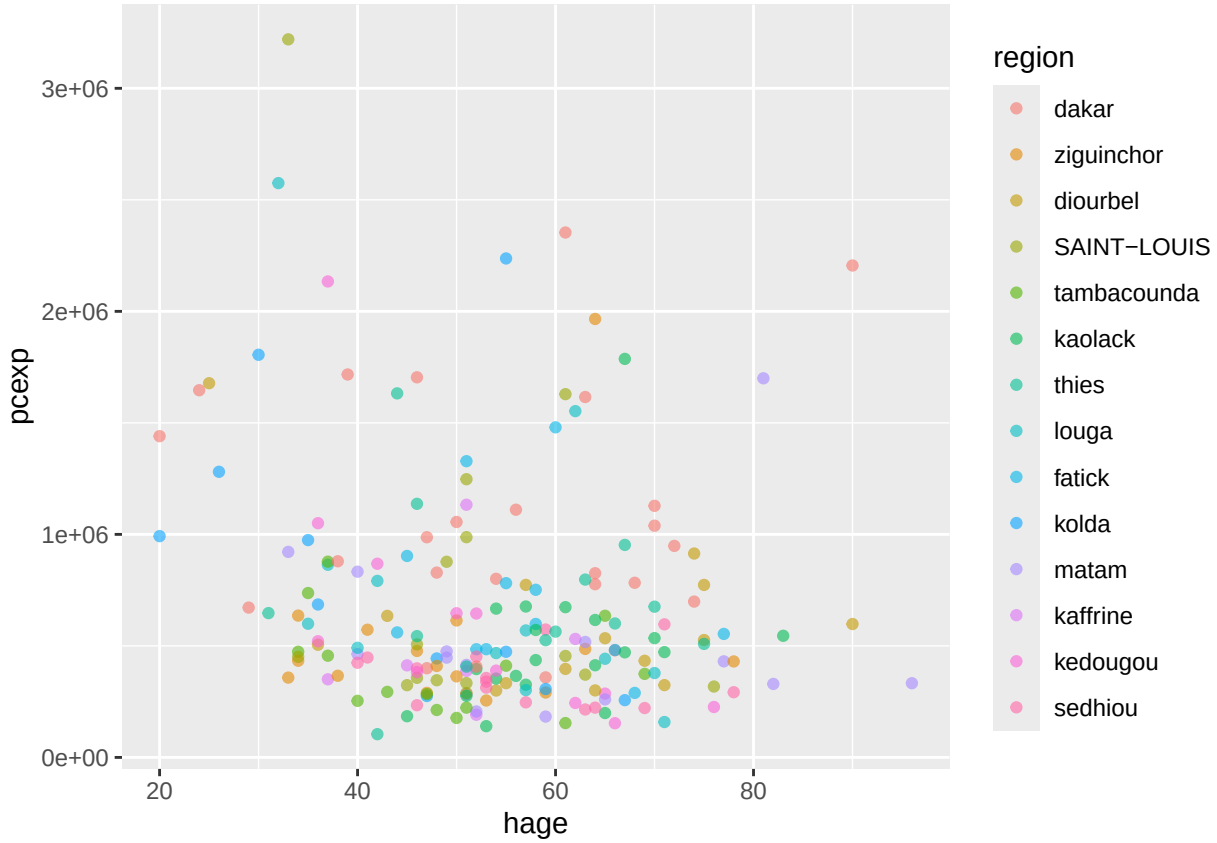
Cette association s'effectue principalement dans la fonction aes().

4.3.1 Exemples de mappage

On a déjà vu les mappages `x` et `y` pour un nuage de points.
Ceux-ci signifient que la **position d'un point**, horizontalement (`x`) et verticalement (`y`), dépend des valeurs des variables passées dans `aes()`.

Mais on peut aller plus loin en ajoutant d'autres **mappages esthétiques**.
Par exemple, `color` permet de faire varier la **couleur des points automatiquement** selon les valeurs d'une **troisième variable**.

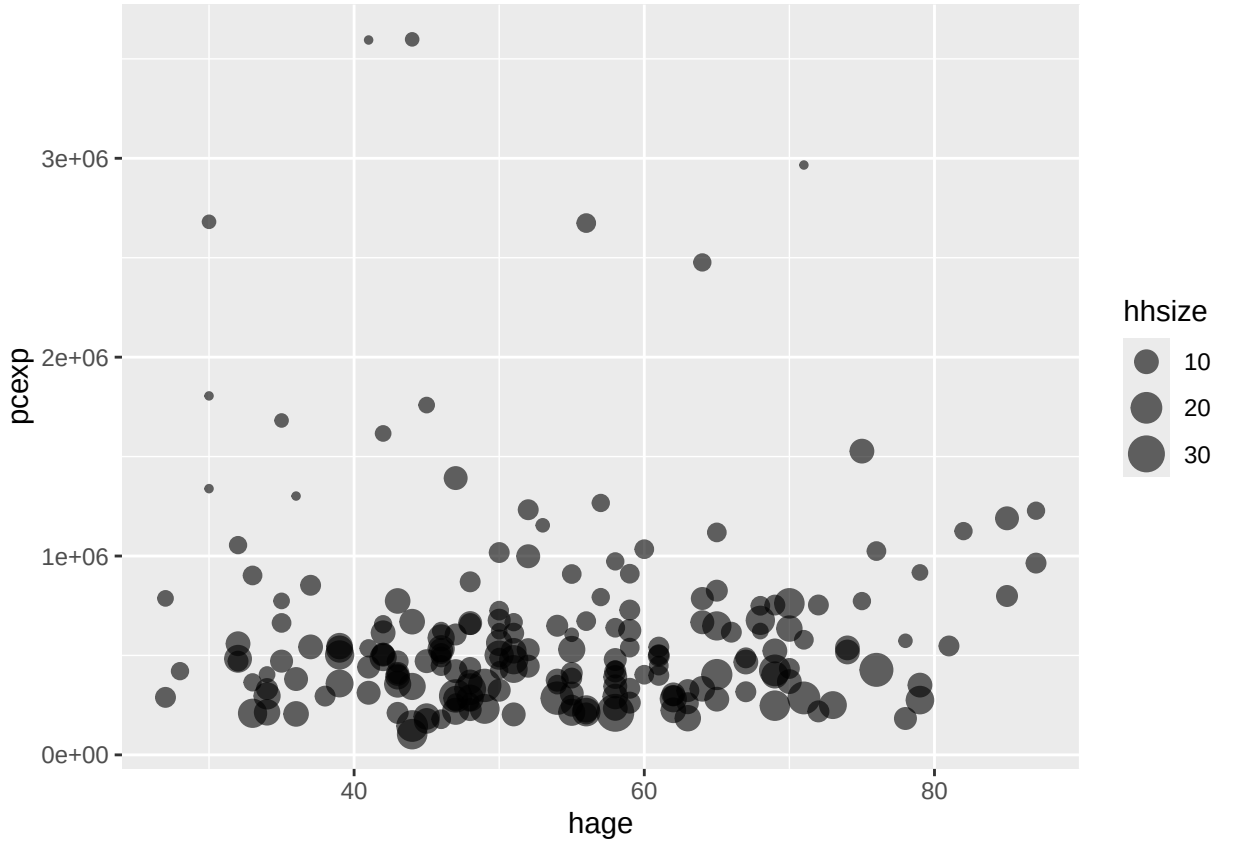
```
# Prendre un échantillon aléatoire de 200 observations pour ne pas surcharger la visualisation
wf_sample <- wf %>%
  sample_n(200)

ggplot(wf_sample, aes(x = hage, y = pcexp, color = region)) +
  geom_point(alpha = 0.6)
```



Ici, la couleur des points varie selon les régions.

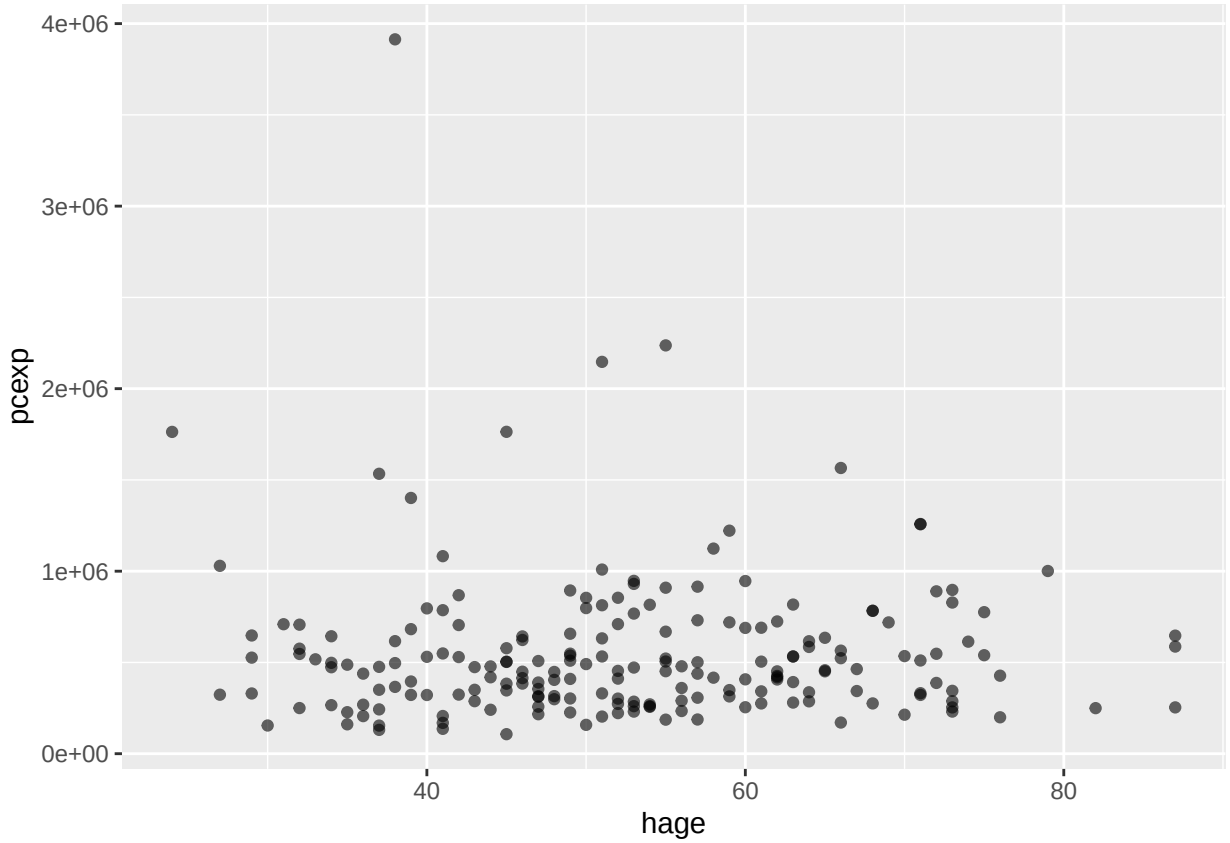
On peut également faire varier la **taille des points** en fonction d'une variable quantitative, grâce à l'argument `size` dans `aes()`. On peut par exemple faire dépendre la taille des points de la **taille du ménage** (`hhsize`),
ce qui revient à représenter l'importance relative de chaque ménage dans le nuage de points.

```
# Prendre un échantillon aléatoire de 200 observations pour ne pas surcharger la visualisation
wf_sample <- wf %>%
  sample_n(200)

ggplot(wf_sample, aes(x = hage, y = pcexp, size = hhsize)) +
  geom_point(alpha = 0.6)
```



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

```
# Prendre un échantillon aléatoire de 200 observations pour ne pas surcharger la visualisation
wf_sample <- wf %>%
  sample_n(200)

ggplot(wf_sample, aes(x = hage, y = pcexp, alpha = hhsize)) +
  geom_point(alpha = 0.6)
```



Chaque geom() possède sa propre liste de mappage.

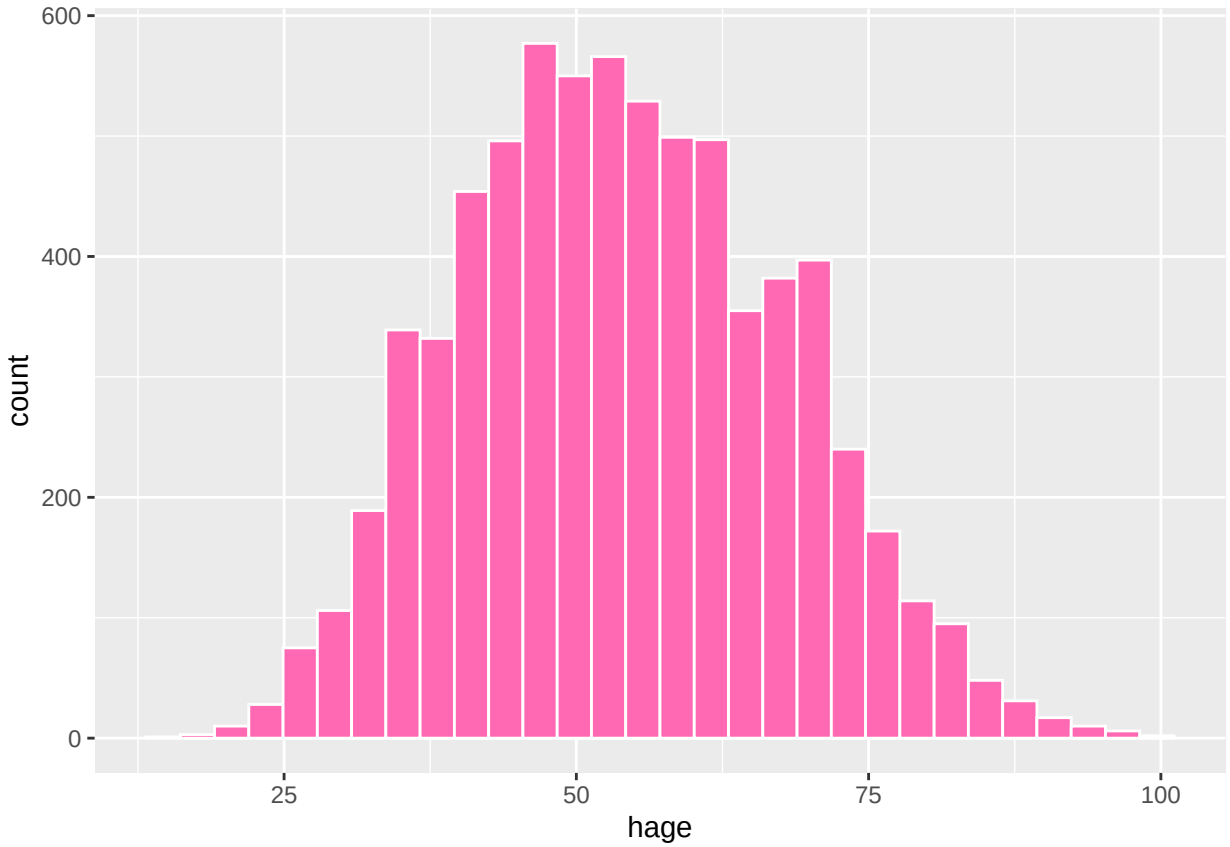
4.3.2 aes() ou non aes()

Comme on l'a déjà vu, parfois on souhaite **changer un attribut graphique** 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()`** :

Exemple 1 : Couleur fixée manuellement (hors aes())

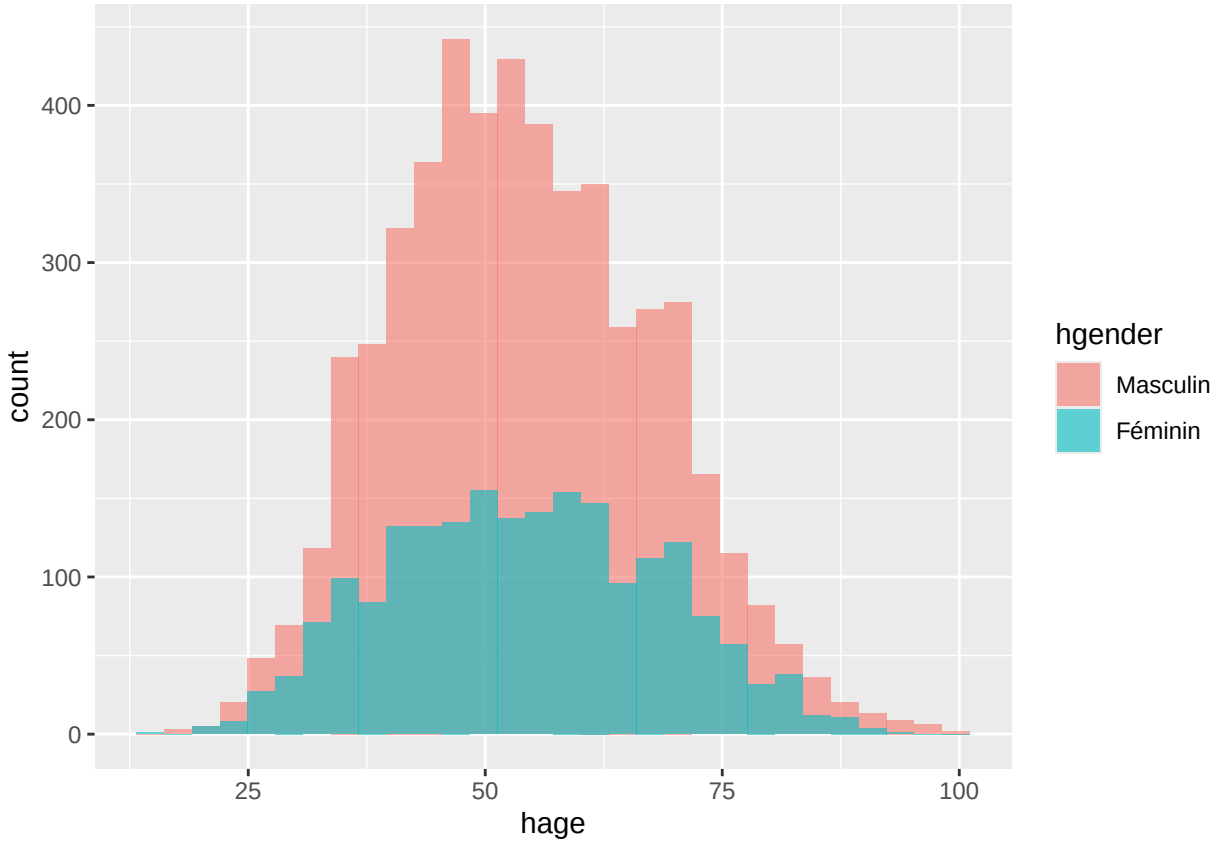
```
ggplot(wf, aes(x = hage)) +
  geom_histogram(fill = "hotpink", color = "white", bins = 30)
```



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

Exemple 2 : Couleur dépendante d'une variable (dans aes())

```
ggplot(wf, aes(x = hage, fill = hgender)) +
  geom_histogram(position = "identity", bins = 30, alpha = 0.6)
```



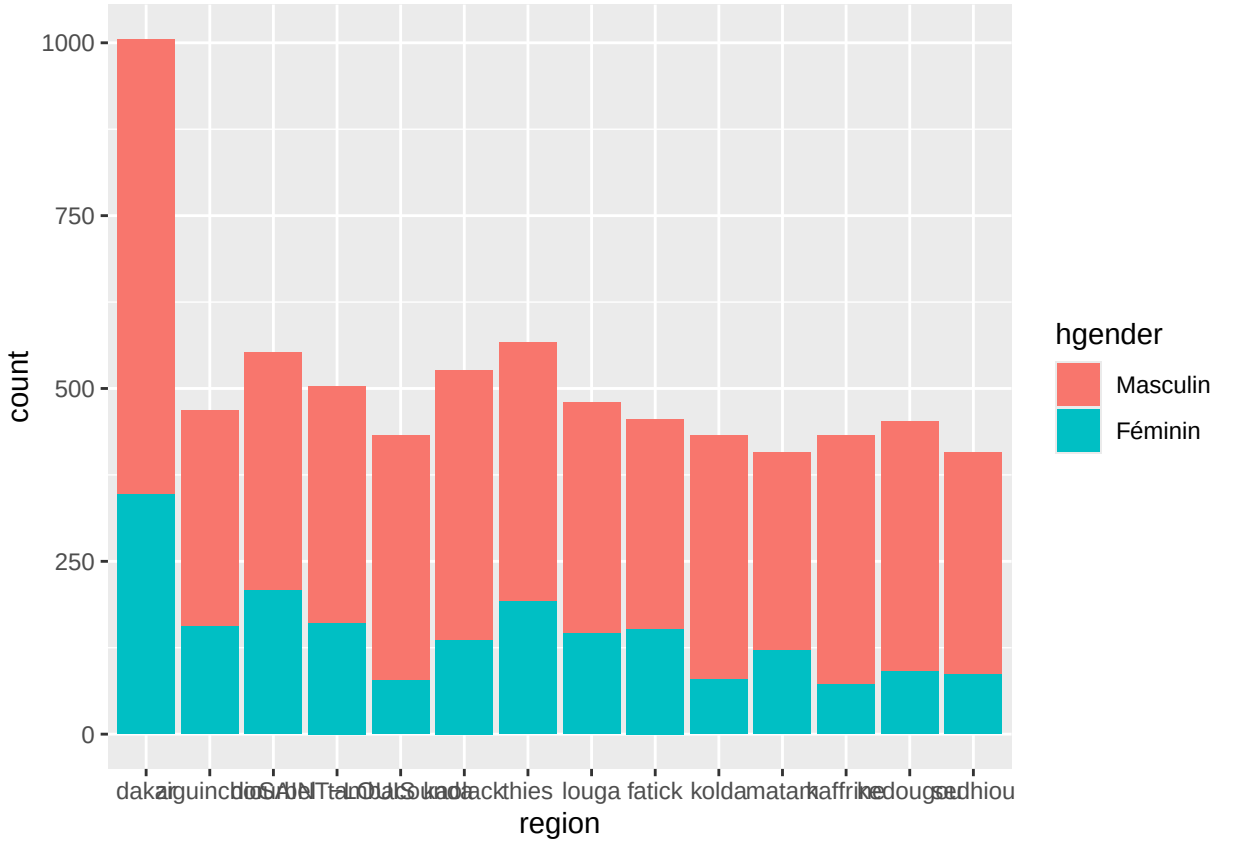
La règle est donc simple mais très importante :

- Si l'on **établit un lien entre les valeurs d'une variable et un attribut graphique**, on définit un **mappage**, et on le déclare **dans la fonction `aes()`**.
- Sinon, si l'on **modifie l'attribut de la même manière pour tous les points**, on le définit **en dehors de `aes()`**.

4.3.3 geom_bar et position

Un des mappages possibles de `geom_bar()` est l'attribut `fill`,
qui permet de **colorer les barres selon les modalités d'une variable catégorielle**.

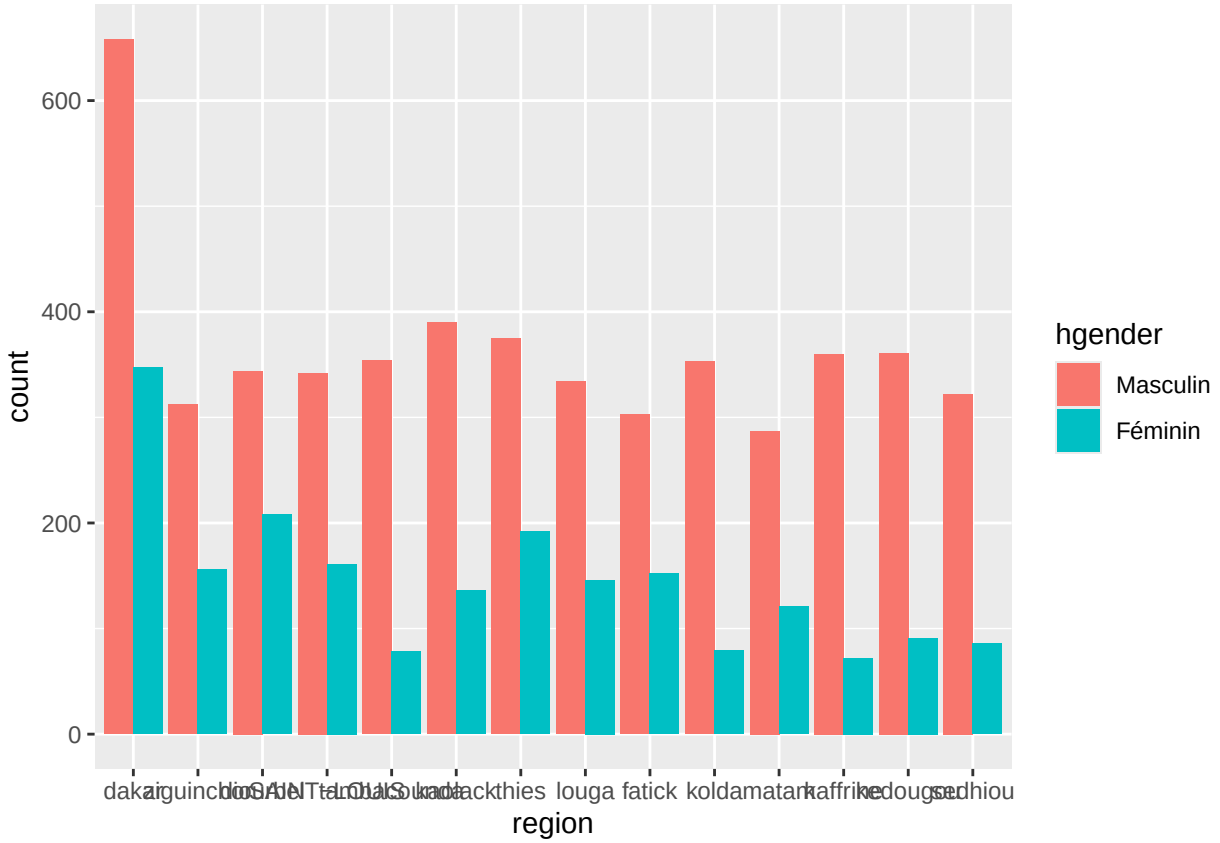
```
ggplot(wf, aes(x = region, fill = hgender)) +
  geom_bar()
```



L'attribut `position` de `geom_bar()` permet d'indiquer **comment les différentes barres doivent être positionnées**.

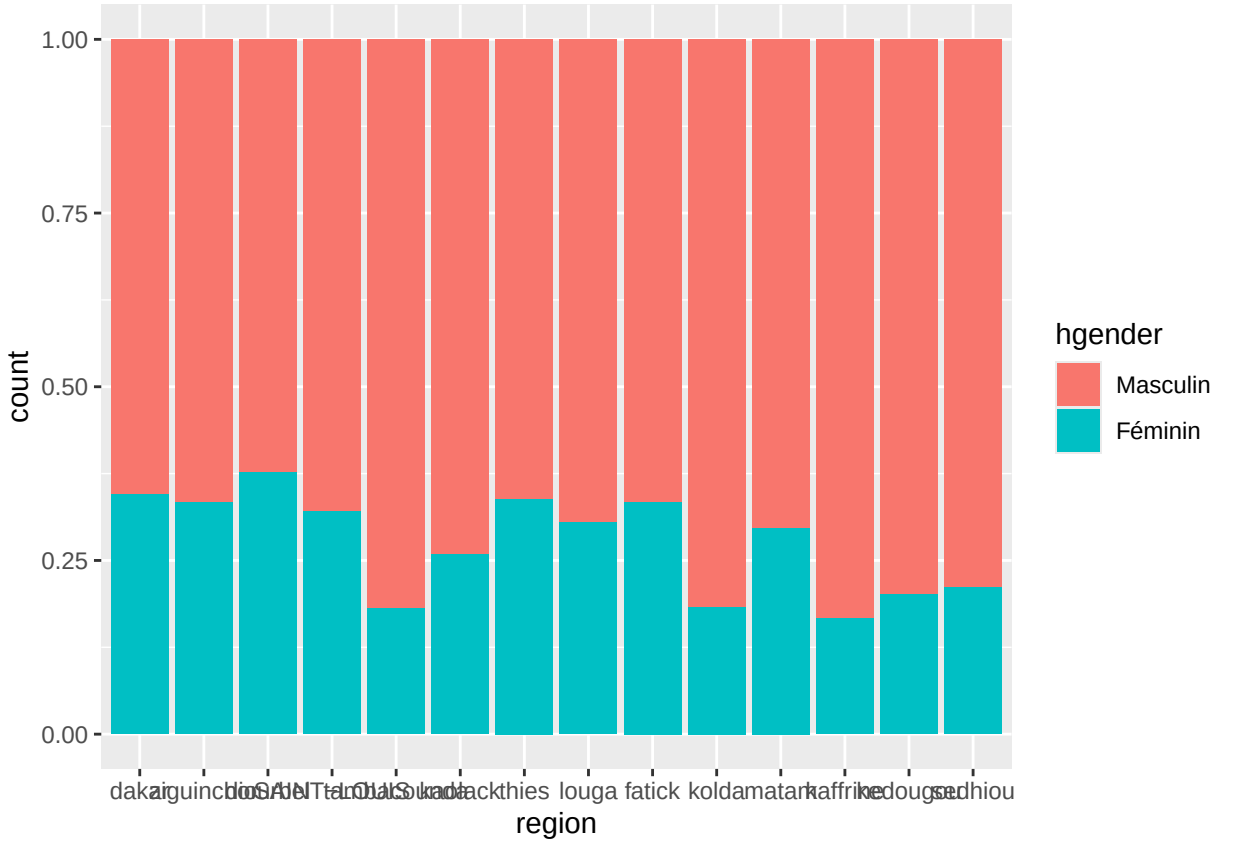
- Par défaut, on a `position = "stack"` : les barres sont **empilées** les unes au-dessus des autres.
- On peut aussi préciser `position = "dodge"` : les barres sont alors **placées côte à côte**, ce qui facilite la comparaison entre modalités.

```
ggplot(wf, aes(x = region, fill = hgender)) +
  geom_bar(position = "dodge")
```



Ou encore, on peut utiliser `position = "fill"` pour représenter **des proportions** plutôt que des effectifs. Cela permet de visualiser la **composition relative de chaque groupe**, normalisée à 100 %.

```
ggplot(wf, aes(x = region, fill = hgender)) +
  geom_bar(position = "fill")
```

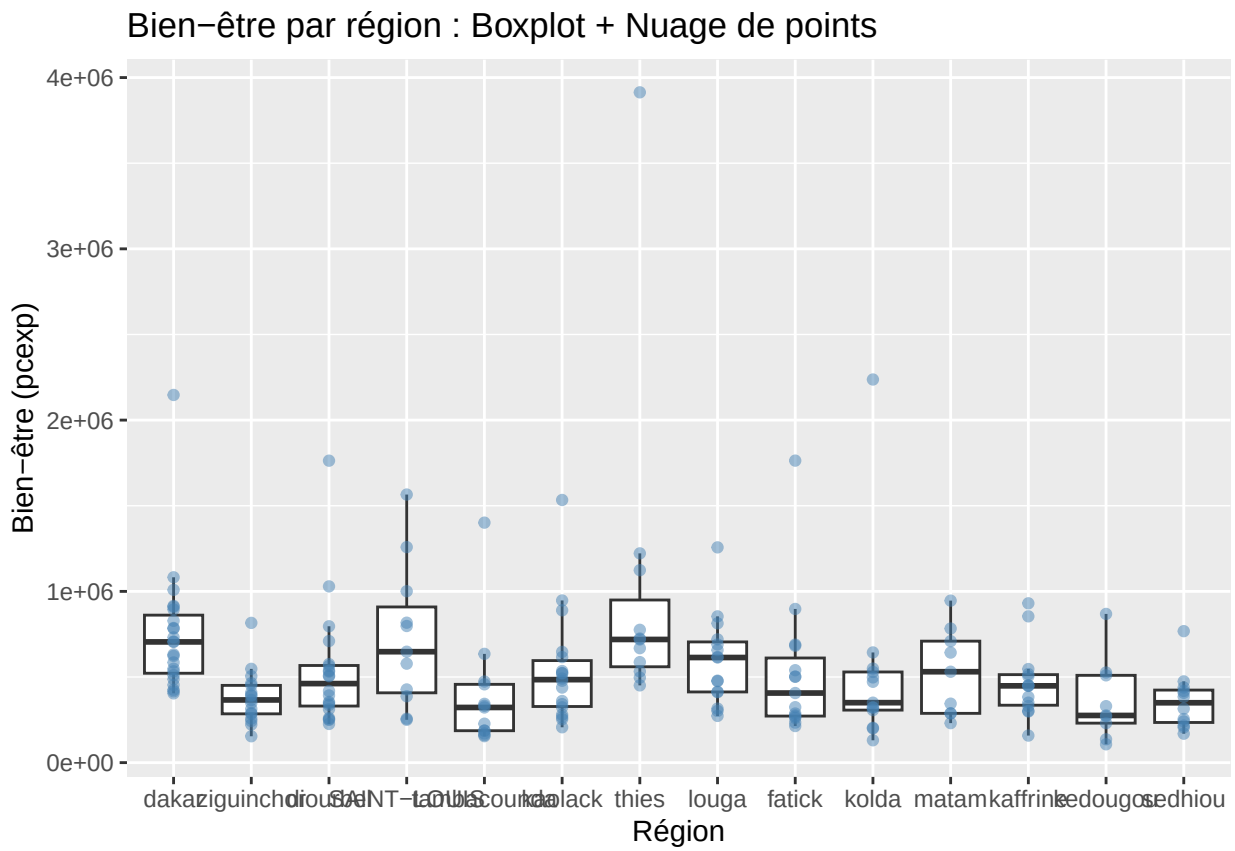


4.4 Représentation de plusieurs geom

L'un des atouts majeurs de ggplot2 est la possibilité de superposer plusieurs couches de géométries dans un même graphique. Cela permet de comparer différentes formes de visualisation (points, lignes, barres...),d'enrichir un graphique avec des éléments informatifs (lignes de tendance, annotations, etc.), ou encore intégrer des données issues de plusieurs sources dans une seule visualisation cohérente.

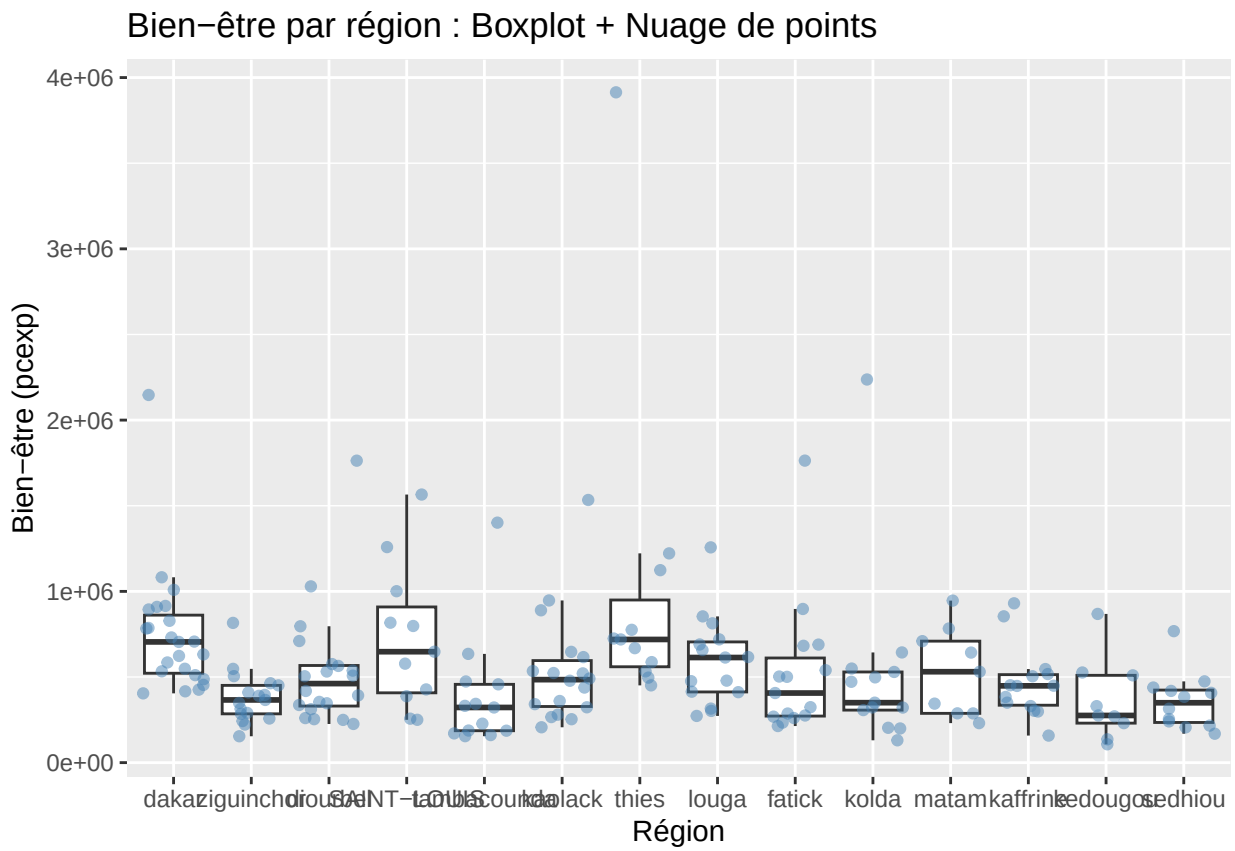
Par exemple, on peut **superposer un nuage de points** à un **graphique en boîtes à moustaches**, afin d'avoir à la fois une **vue globale (boxplot)** et une **vue individuelle (points)** des données.

```
ggplot(wf_sample, aes(x = region, y = pcexp)) +
  geom_boxplot(outlier.shape = NA) +  # éviter de doubler les points extrêmes
  geom_point( alpha = 0.5, color = "steelblue") +
  labs(title = "Bien-être par région : Boxplot + Nuage de points",
       x = "Région", y = "Bien-être (pcexp)")
```



Pour un résultat un peu plus lisible, on peut remplacer `geom_point()` par `geom_jitter()`, qui **disperse légèrement les points horizontalement**.
Cela permet d'éviter la superposition excessive des points et de **mieux visualiser la distribution individuelle**.

```
ggplot(wf_sample, aes(x = region, y = pcexp)) +
  geom_boxplot(outlier.shape = NA) +  # éviter de doubler les points extrêmes
  geom_jitter( alpha = 0.5, color = "steelblue") +
  labs(title = "Bien-être par région : Boxplot + Nuage de points",
       x = "Région", y = "Bien-être (pcexp)")
```

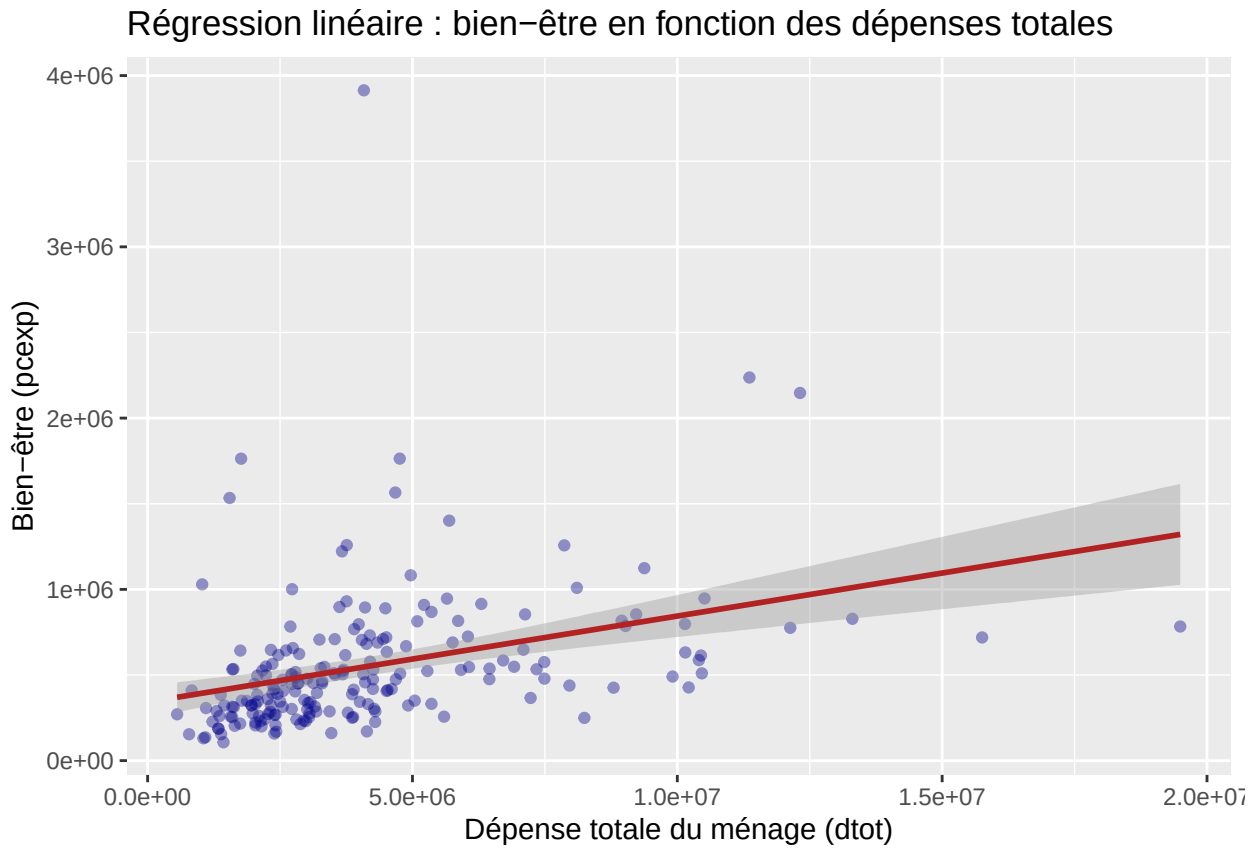


Autre exemple : on peut vouloir **ajouter à un nuage de points une ligne de régression linéaire**, à l'aide de la fonction `geom_smooth()`.

Par défaut, `geom_smooth()` ajuste une **courbe de tendance** (loess), mais il est possible de forcer un modèle **linéaire** en précisant `method = "lm"`.

```
ggplot(wf_sample, aes(x = dtot, y = pcexp)) +
  geom_point(alpha = 0.4, color = "darkblue") +
  geom_smooth(method = "lm", se = TRUE, color = "firebrick") +
  labs(title = "Régression linéaire : bien-être en fonction des dépenses totales",
       x = "Dépense totale du ménage (dtot)",
       y = "Bien-être (pcexp)")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

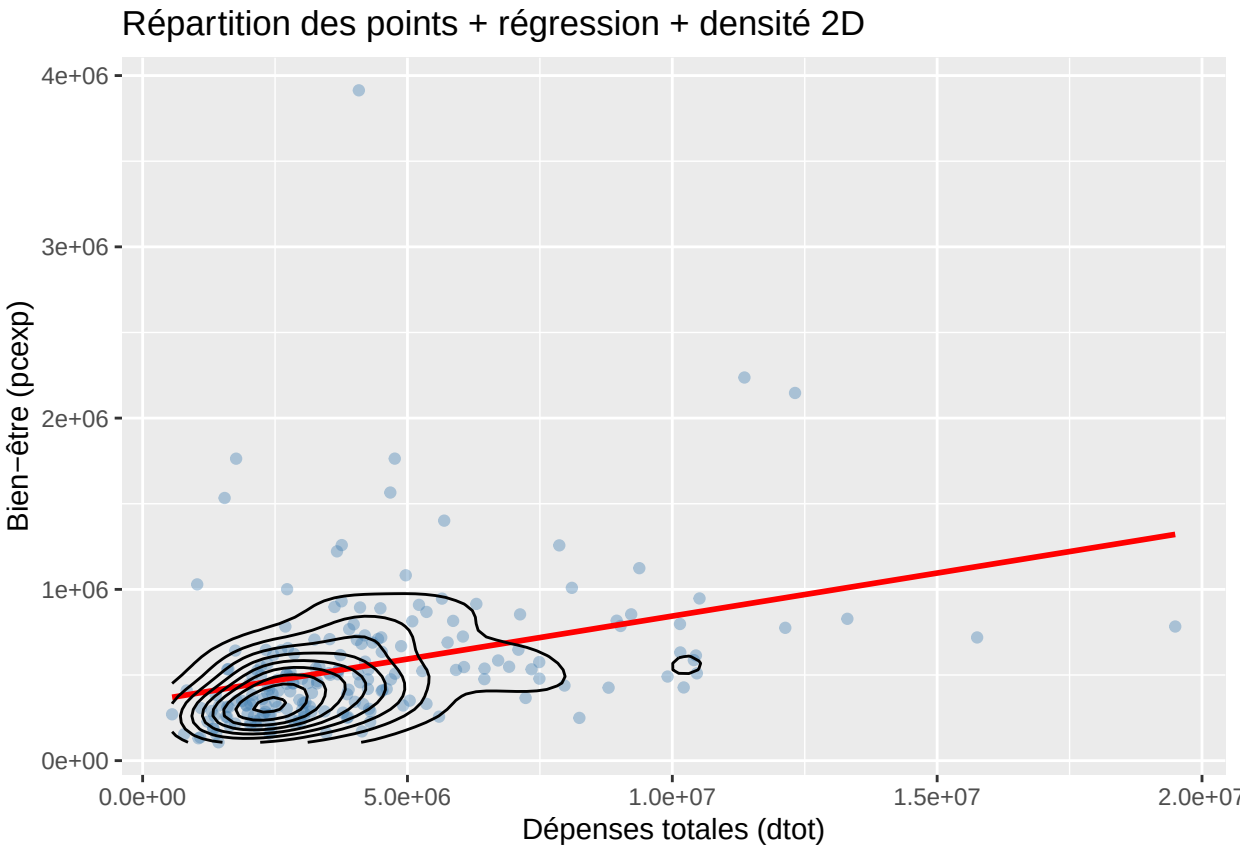


Et on peut même **superposer une troisième visualisation** de la répartition des points dans l'espace à l'aide de la fonction `geom_density2d()`.

Cette fonction trace des **courbes de densité en deux dimensions**, similaires à des courbes de niveau, qui indiquent **où les points sont les plus concentrés** dans le nuage.

```
ggplot(wf_sample, aes(x = dtot, y = pcexp)) +
  geom_point(alpha = 0.4, color = "steelblue") +
  geom_smooth(method = "lm", color = "red", se = FALSE) +
  geom_density2d(color = "black") +
  labs(title = "Répartition des points + régression + densité 2D",
       x = "Dépenses totales (dtot)",
       y = "Bien-être (pcexp)")
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



4.5 Faceting : Diviser pour mieux comparer

Le faceting est une fonctionnalité puissante de ggplot2 qui permet de créer plusieurs sous-graphiques à partir d'une même visualisation, selon une ou plusieurs variables de regroupement. Cela rend les comparaisons plus lisibles entre différents groupes, sans changer d'échelle ni de structure graphique.

Pourquoi utiliser le faceting ? Pour comparer des groupes (hommes vs femmes, régions, milieux urbain/rural. . .)

Pour repérer des tendances spécifiques à certains sous-groupes

Pour analyser des effets croisés (ex. : l'effet de l'âge selon le sexe et le milieu)

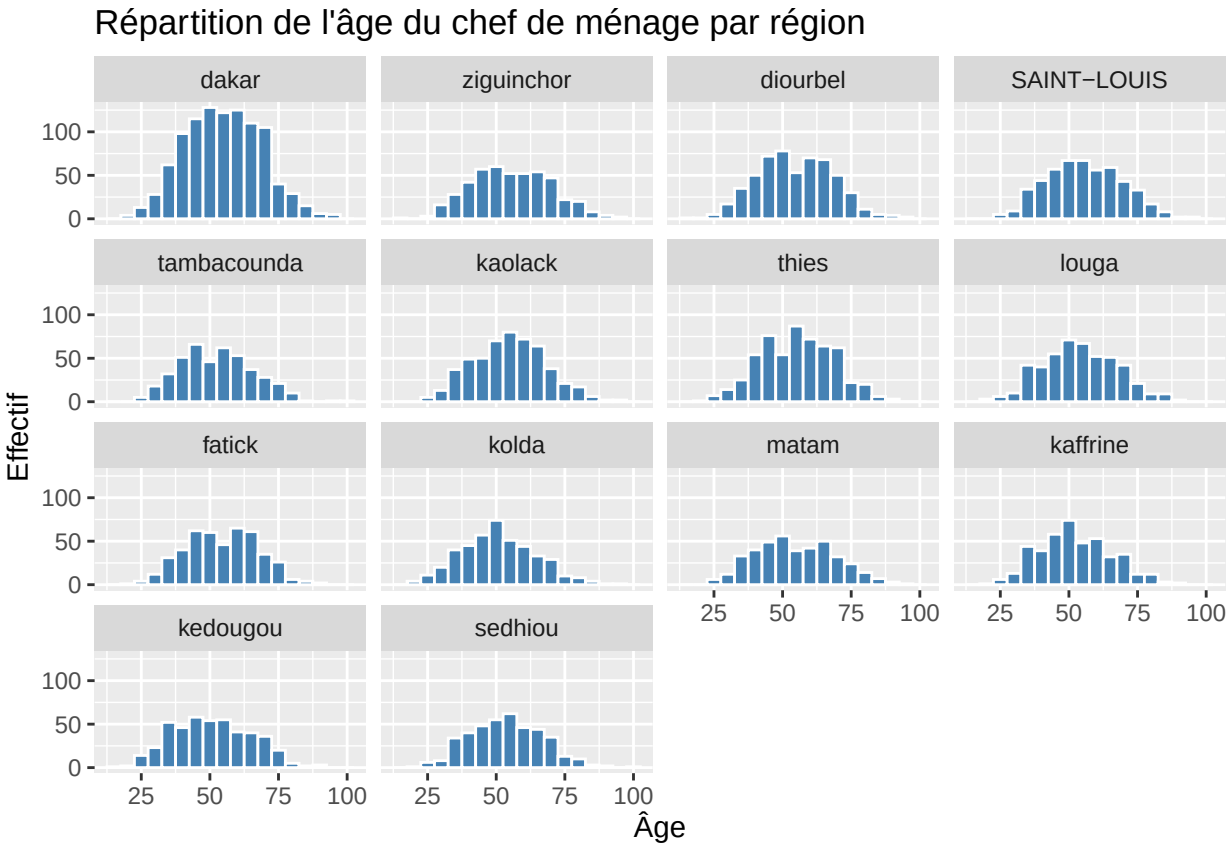
4.5.1 facet_wrap(~ variable) – 'facet_grid(ligne ~ colonne)

Par exemple, on peut vouloir comparer la **répartition de la part des cadres** (`hcsp == "Cadre supérieur"`) selon le **département** (`region`),
et donc tracer **un histogramme pour chaque département**.

Les deux fonctions utilisées pour cela sont :

- `facet_wrap(~ variable)` : les graphiques sont **répartis automatiquement dans la page**
- `facet_grid(ligne ~ colonne)` : les graphiques sont organisés en **grille selon deux variables**

```r
ggplot(wf, aes(x = hage)) +
  geom_histogram(binwidth = 5, fill = "steelblue", color = "white") +
  facet_wrap(~ region) +
  labs(title = "Répartition de l'âge du chef de ménage par région",
       x = "Âge", y = "Effectif")
```



Avec `facet_grid()`, les graphiques sont **disposés selon une grille**,
en croisant **deux variables qualitatives** : l'une pour les lignes, l'autre pour les colonnes.

La formule à passer à `facet_grid()` est de la forme :
`variable_en_ligne ~ variable_en_colonne`

- Si l'on ne souhaite pas utiliser de variable dans une des deux dimensions,
  on remplace par un **point (`.`)**.

Un des intérêts du **faceting** dans `ggplot2` est que **tous les graphiques générés ont les mêmes échelles** (par défaut), que ce soit pour l'axe des abscisses ou celui des ordonnées.

4.6 Scales : Personnaliser les axes, couleurs et tailles

Les échelles (scales) permettent de contrôler la manière dont les données sont traduites en éléments visuels dans ggplot2 : taille, couleur, position, forme, etc. Elles rendent les graphes plus lisibles, cohérents et adaptés aux objectifs d'analyse.

4.6.1 scale_size() – Ajuster la taille selon une variable

L'esthétique size dans ggplot2 permet de représenter une troisième variable continue en modifiant la taille des points d'un graphique. Cela permet d'ajouter une couche d'information supplémentaire dans une visualisation en deux dimensions.

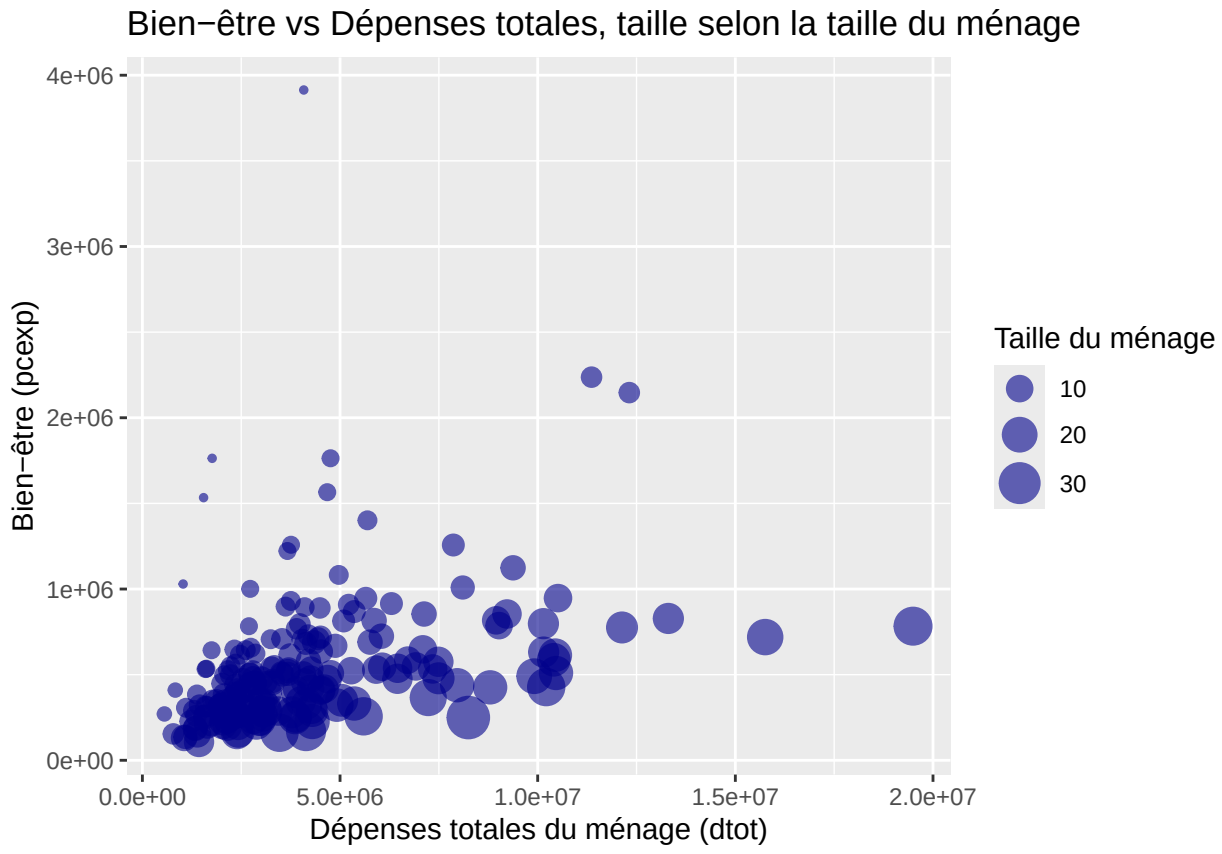
On utilise souvent scale_size() pour :

Mieux contrôler l'échelle visuelle (plage de taille) des points ;

Adapter l'interprétation à la portée des données (petits points pour les faibles valeurs, gros points pour les fortes valeurs).

Exemple :

```
ggplot(wf_sample, aes(x = dtot, y = pcexp, size = hhsize)) +
  geom_point(alpha = 0.6, color = "darkblue") +
  scale_size(range = c(1, 7)) +
  labs(
    title = "Bien-être vs Dépenses totales, taille selon la taille du ménage",
    x = "Dépenses totales du ménage (dtot)",
    y = "Bien-être (pcexp)",
    size = "Taille du ménage"
  )
```



4.6.2 scale_x_() et scale_y_() – Personnaliser les axes

Pourquoi personnaliser les axes ?

Quand on crée un graphique avec ggplot2, les axes sont générés automatiquement à partir des données. Mais souvent, on souhaite aller plus loin :

-Modifier le titre des axes pour le rendre plus explicite ;

-Définir où placer les graduations (ex. tous les 10 ans pour l'âge) ;

-Limiter l'échelle (par exemple, ne montrer que de 0 à 50 jours pour une durée).

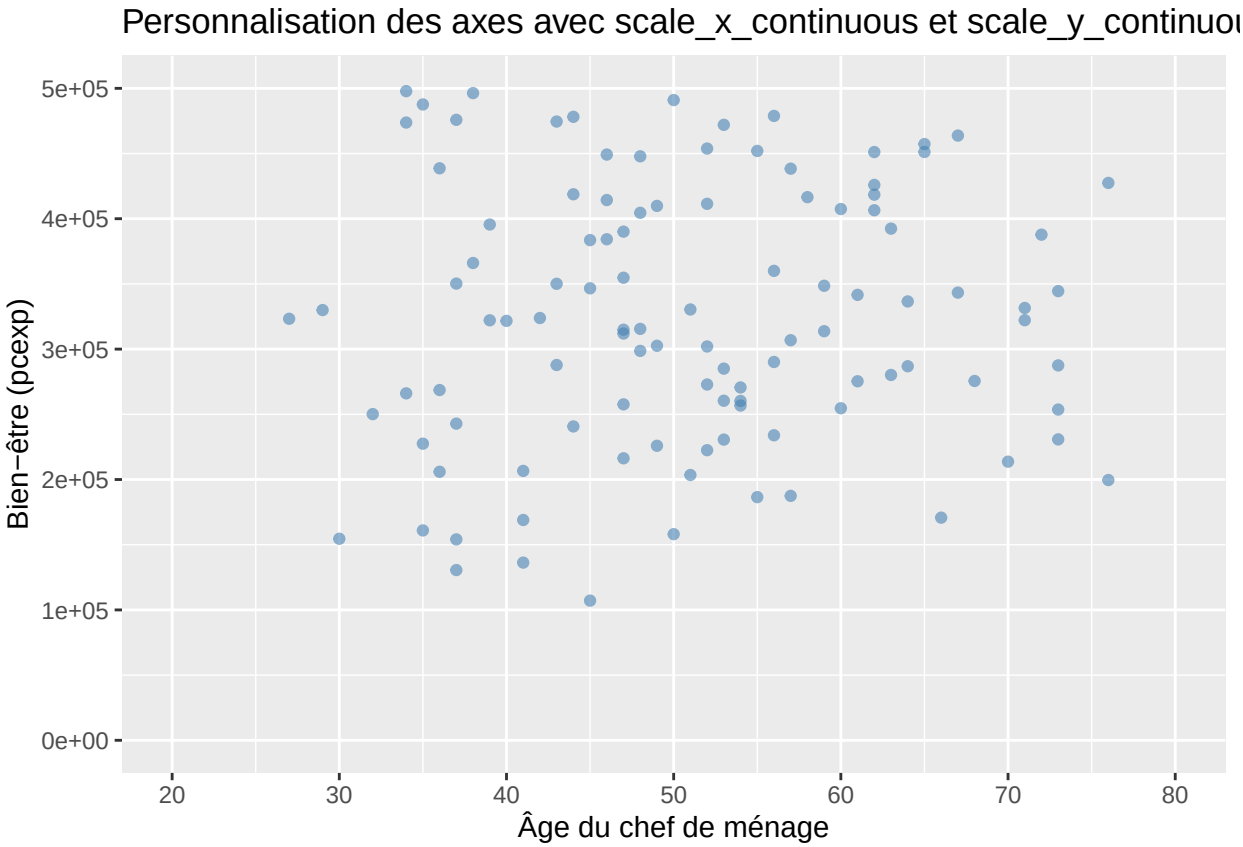
C'est là qu'interviennent les fonctions scale_x_*() et scale_y_*().

Exemple : Ces deux fonctions (scale_x_*() et scale_y_*()) sont utilisées que pour les variables numériques continues :

Exemple : personnaliser l'échelle de l'axe des x (`hage`) et des y (`pcexp`)

```
ggplot(wf_sample, aes(x = hage, y = pcexp)) +
  geom_point(color = "steelblue", alpha = 0.6) +
  scale_x_continuous(
    name = "Âge du chef de ménage",
    breaks = seq(20, 80, by = 10), ## Personnalisation des axes
    limits = c(20, 80)
  ) +
  scale_y_continuous(
    name = "Bien-être (pcexp)",
    limits = c(0, 500000)
  ) +
  labs(title = "Personnalisation des axes avec scale_x_continuous et scale_y_continuous")
```

```
## Warning: Removed 92 rows containing missing values or values outside the scale range
## (`geom_point()`).
```



4.6.3 scale_color_() et scale_fill_() – Travailler les couleurs

4.6.3.1 Variables quantitatives : scale_color_gradient() La fonction `scale_color_gradient()` de `ggplot2` permet d'afficher une **variable numérique continue** sous forme de **dégradé de couleurs**.

C'est très utile pour **explorer visuellement l'intensité, les tendances ou les variations** d'une variable dans un graphique.

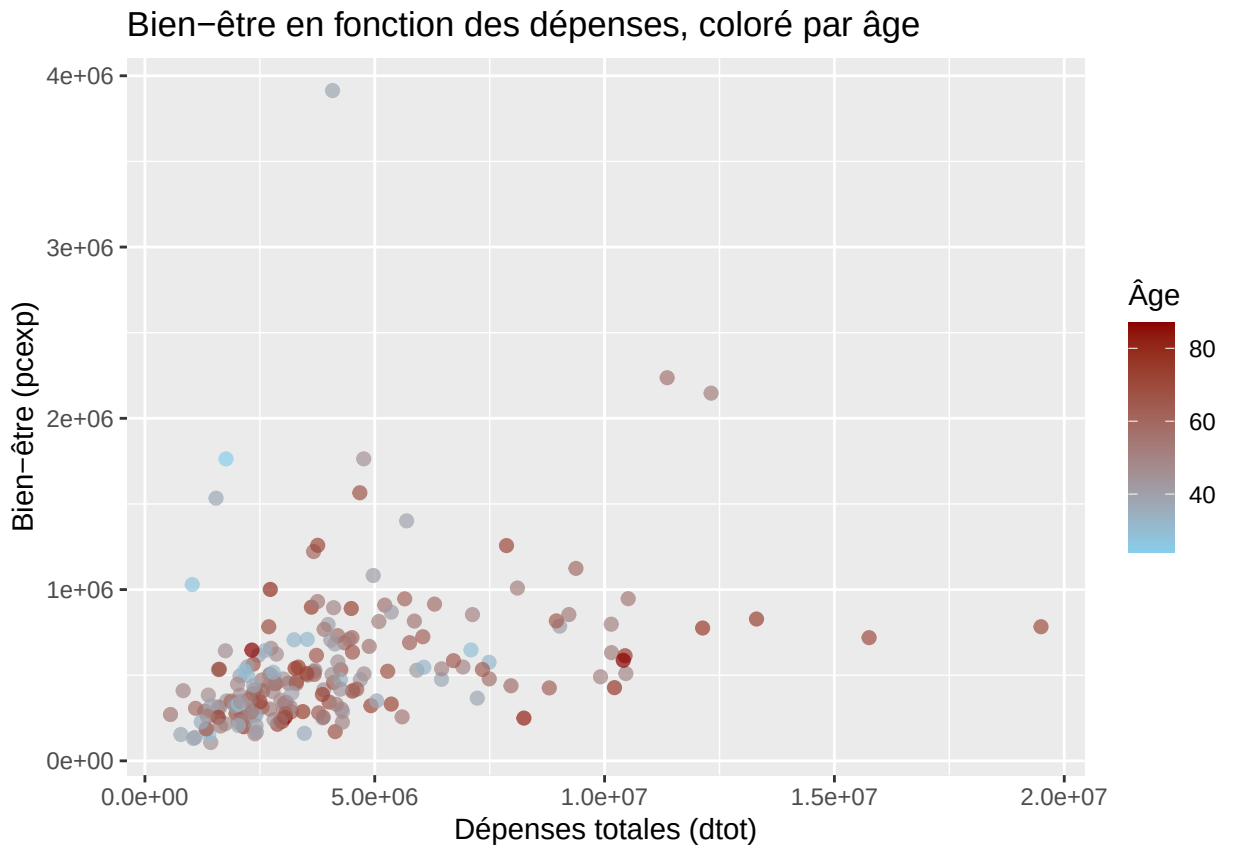
Quand l'utiliser ?

On l'utilise lorsque l'on souhaite que la **couleur d'un point, d'une ligne ou d'un objet graphique** reflète la valeur d'une **variable continue**, par exemple :

- L'âge (`hage`)
- Le revenu ou bien-être (`pcexp`)
- La durée d'un événement, etc.

```
ggplot(wf_sample, aes(x = dtot, y = pcexp, color = hage)) +
  geom_point(size = 2, alpha = 0.7) +

  # Appliquer un dégradé de bleu à rouge selon l'âge
  scale_color_gradient(
    low = "skyblue",      # Couleur pour les petites valeurs
    high = "darkred"      # Couleur pour les grandes valeurs
  ) +
```

```
  labs(
    title = "Bien-être en fonction des dépenses, coloré par âge",
    x = "Dépenses totales (dtot)",
    y = "Bien-être (pcexp)",
    color = "Âge"
  )
```



4.6.3.2 Variables qualitatives : scale_color_manual() et scale_fill_manual()

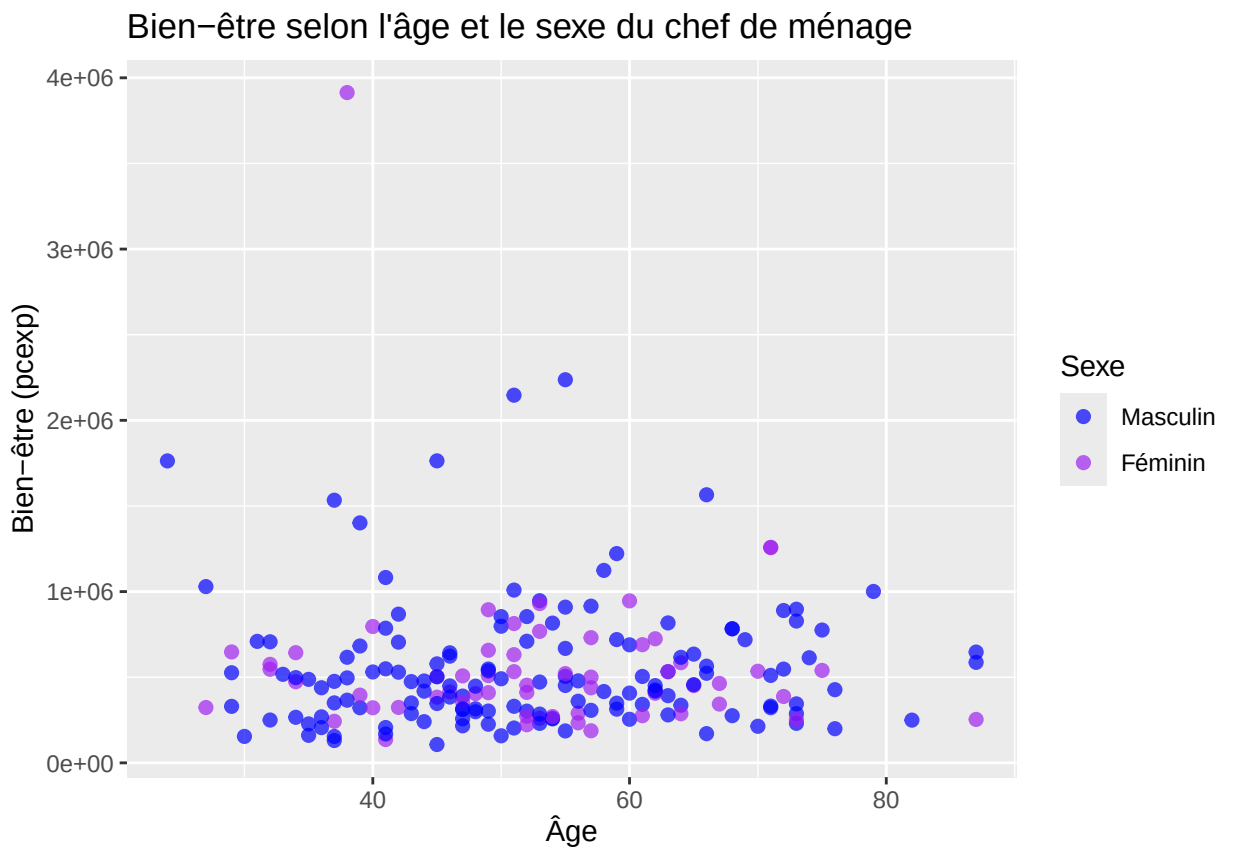
La fonction `scale_color_manual()` de `ggplot2` est utilisée pour **attribuer manuellement des couleurs fixes**
à des **modalités d'une variable catégorielle**, comme :

- le **sexe** (`hgender`),
- le **milieu de résidence** (`milieu`),
- ou la **région** (`region`).

Cela permet de **contrôler précisément les couleurs utilisées** dans un graphique, au lieu de laisser `ggplot2` choisir automatiquement.

```
ggplot(wf_sample, aes(x = hage, y = pcexp, color = hgender)) +
  geom_point(size = 2, alpha = 0.7) +

  # Attribution manuelle des couleurs
  scale_color_manual(
    values = c("Masculin" = "blue", "Féminin" = "purple")
```

```
) +

labs(
  title = "Bien-être selon l'âge et le sexe du chef de ménage",
  x = "Âge",
  y = "Bien-être (pcexp)",
  color = "Sexe"
)
```



4.7 Thèmes

Les thèmes dans ggplot2 permettent de contrôler l'apparence globale d'un graphique sans altérer les données. Ils sont essentiels pour rendre une visualisation plus claire, professionnelle et agréable à lire, que ce soit pour une publication académique, une présentation ou un rapport.

Ils agissent sur :

le fond du graphique,

les axes et leurs étiquettes,

la police de caractère (taille, style, couleur),

la légende (position, apparence),

les marges,

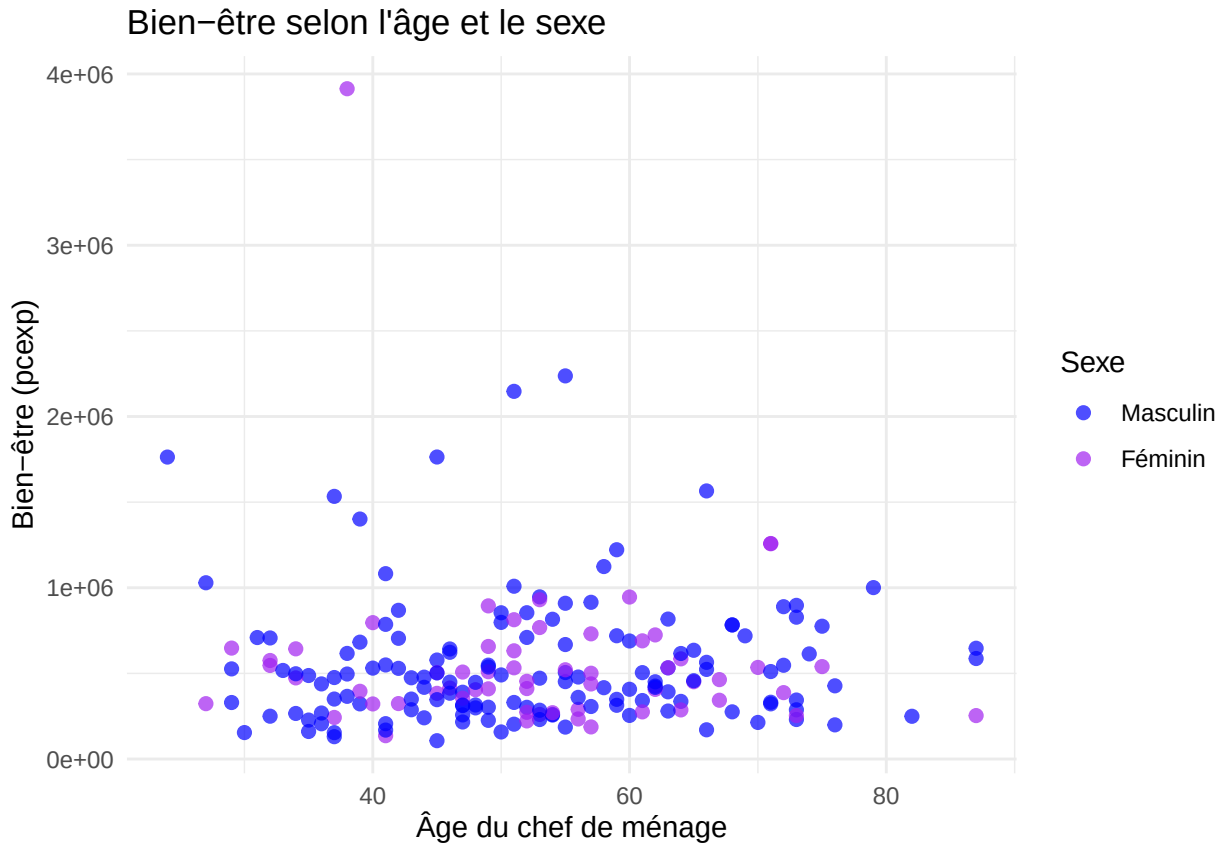
les titres et sous-titres, etc.

4.7.1 Thèmes prédéfinis

`ggplot2` fournit plusieurs **thèmes intégrés** permettant de **changer rapidement l'aspect visuel** d'un graphique
(en modifiant la grille, la police, le fond, etc.).

L'un des plus courants est le **`theme_minimal()`**, qui applique un **style épuré** et professionnel,
idéal pour les publications ou rapports.

```
ggplot(wf_sample, aes(x = hage, y = pcexp, color = hgender)) +
  geom_point(alpha = 0.7, size = 2) +
  scale_color_manual(values = c("Masculin" = "blue", "Féminin" = "purple")) +
  labs(
    title = "Bien-être selon l'âge et le sexe",
    x = "Âge du chef de ménage",
    y = "Bien-être (pcexp)",
    color = "Sexe"
  ) +
  theme_minimal()  # Application du thème épuré
```



Autres thèmes disponibles :

theme_bw() : fond blanc, cadre noir – bon pour l'impression.

theme_classic() : look épuré, sans quadrillage – très lisible.

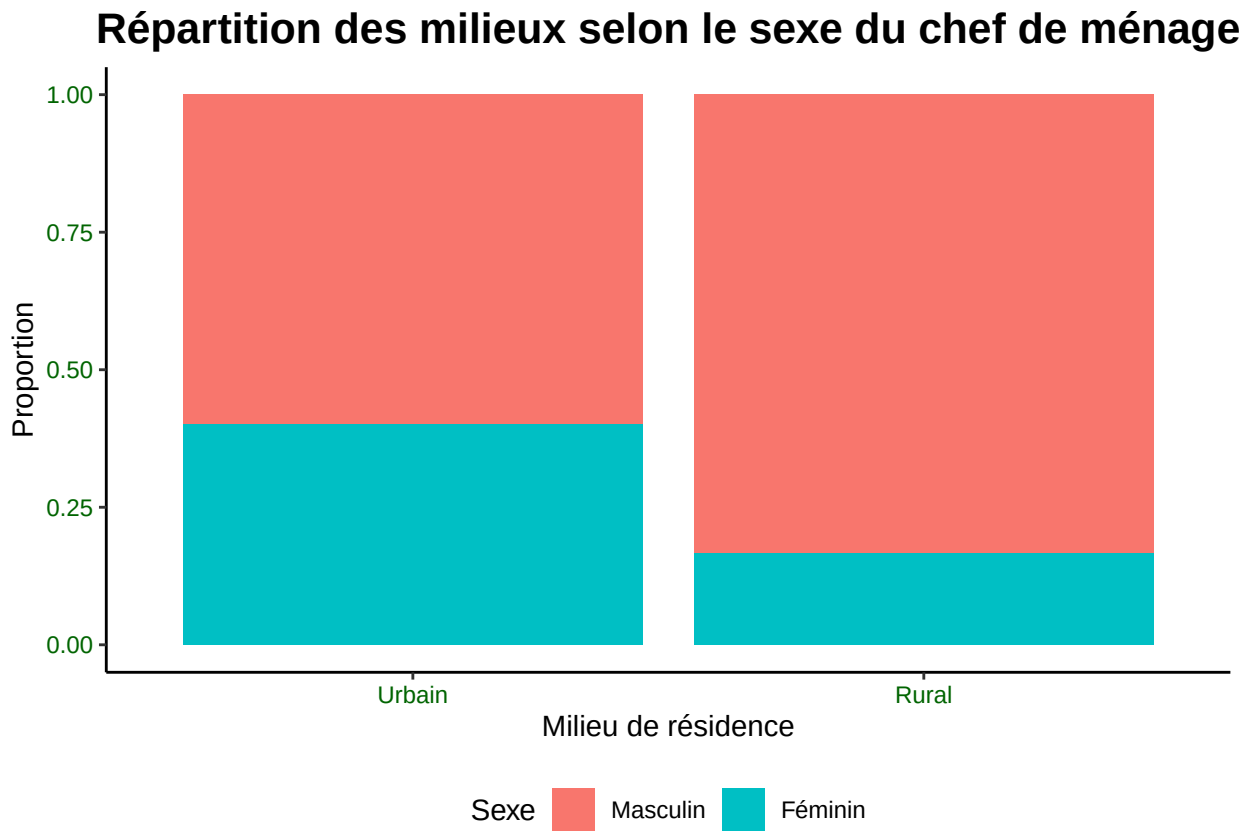
theme_light() : version plus douce que theme_bw().

theme_void() : supprime tous les éléments visuels – parfait pour des cartes.

4.7.2 Personnalisation fine avec theme()

Pour aller plus loin, la fonction theme() permet de personnaliser précisément chaque composant graphique.

```
ggplot(wf_sample, aes(x = milieu, fill = hgender)) +
  geom_bar(position = "fill") +                # Barres empilées en proportion
  theme_classic() +                            # Style de base épuré
  theme(
    plot.title = element_text(                 # Titre en gras, centré, plus grand
      size = 16,
      face = "bold",
      hjust = 0.5
    ),
    axis.text = element_text(                  # Couleur du texte des axes
      color = "darkgreen"
    ),
    legend.position = "bottom"                 # Position de la légende en bas
  ) +
  labs(
    title = "Répartition des milieux selon le sexe du chef de ménage",
    x = "Milieu de résidence",
    y = "Proportion",
    fill = "Sexe"
  )
```



Ce code modifie :

le titre (taille, gras, centrage),

la couleur du texte des axes,

la position de la légende (placée en bas).

Chaque élément est personnalisable : marges, fond du panel, quadrillage, etc.