Représentation des données avec les packages graphics et ggplot2 de **R**

M. L. Delignette-Muller
VetAgro Sup - LBBE

7 janvier 2024

Package `graphics`

Commençons par tenter de maîtriser le package `graphics` qui est le package graphique de base de **R** ?

Jeu de données exemple

Résultats d'une enquête réalisée sur un échantillon d'étudiants vétérinaires

```
d <- read.table("DATA/ENQ9697.txt", header = TRUE,
                stringsAsFactors = TRUE)
str(d)
```

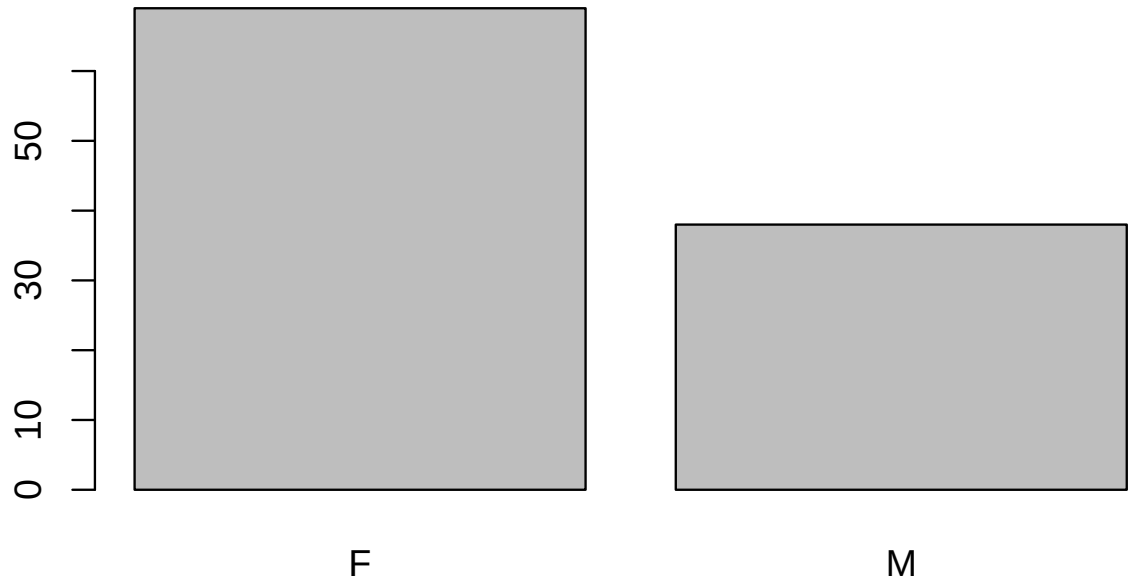
```
'data.frame':        107 obs. of  7 variables:
 $ SEXE    : Factor w/ 2 levels "F","M": 1 2 1 2 1 1 2 1 1 1 .
 $ AGE     : int  22 21 19 20 19 21 21 19 20 22 ...
 $ POIDS   : int  53 67 63 60 48 58 77 61 52 70 ...
 $ TAILLE  : int  175 175 172 175 167 171 187 170 161 168 ...
 $ CADRE   : Factor w/ 2 levels "C","V": 2 1 2 2 2 1 1 1 1 1 .
 $ DECISION: Factor w/ 3 levels "A","E","T": 2 1 1 1 1 2 1 1 2
 $ FILIERE : Factor w/ 7 levels "A","C","E","I",..: 6 6 3 2 6
```

La fonction `plot()` du package graphics

Une fonction qui permet de réaliser de nombreuses représentations graphiques.
Le graphe qu'elle propose dépend du ou des objets donnés en arguments de la fonction.

`plot(varqual)` : diagramme en bâtons

```
plot(d$SEXE)
```

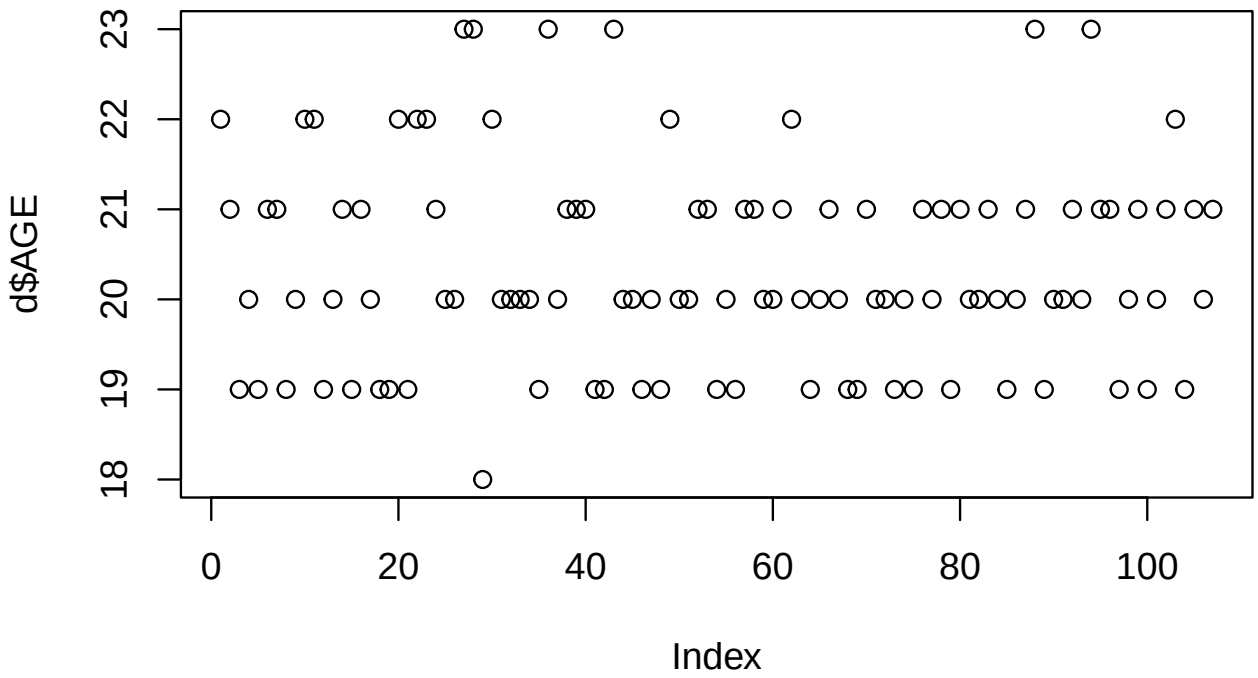


Ecriture équivalente :

```
barplot(table(d$SEXE))
```

`plot(varquant)` : séquence des valeurs

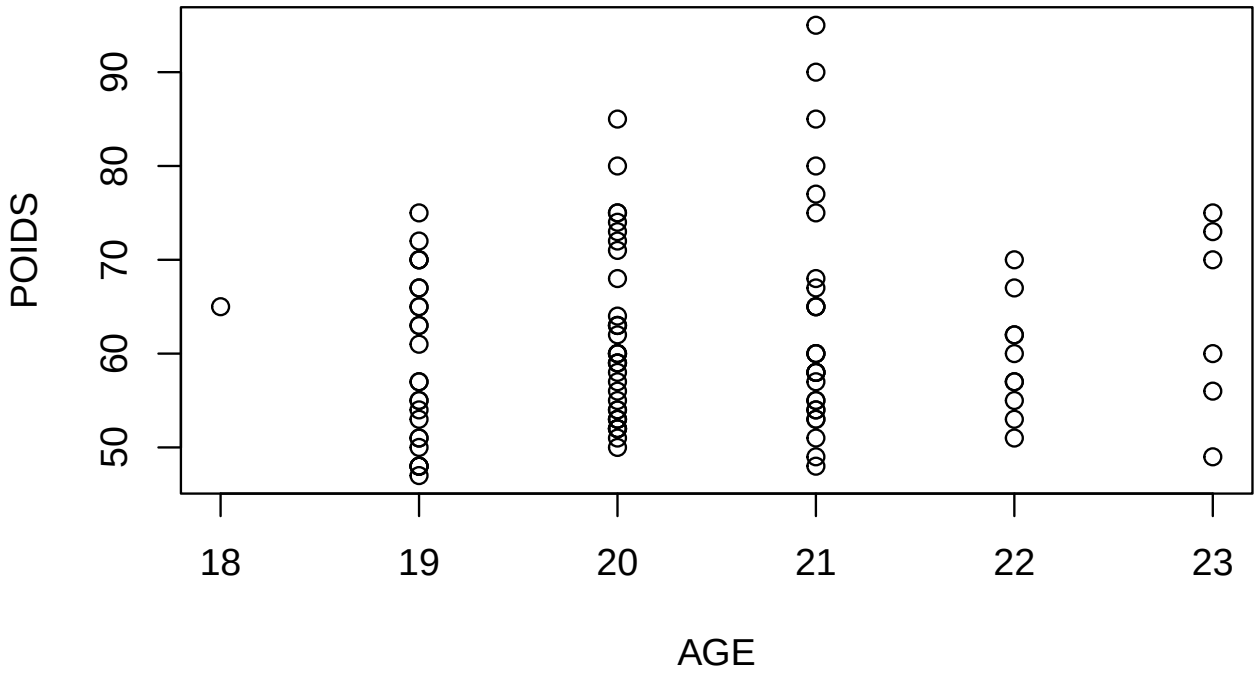
```
plot(d$AGE)
```



Tracé des valeurs observées en fonction du numéro de ligne.

`plot(varquant,varquant)` : nuage de points

```
plot(POIDS ~ AGE, data = d)
```



Ecriture équivalente :

```
plot(d$AGE, d$POIDS)
```

`plot(varqual,varquant)` : diagrammes en boîtes

```
plot(POIDS ~ SEXE, data = d)
```

Ecritures équivalentes :

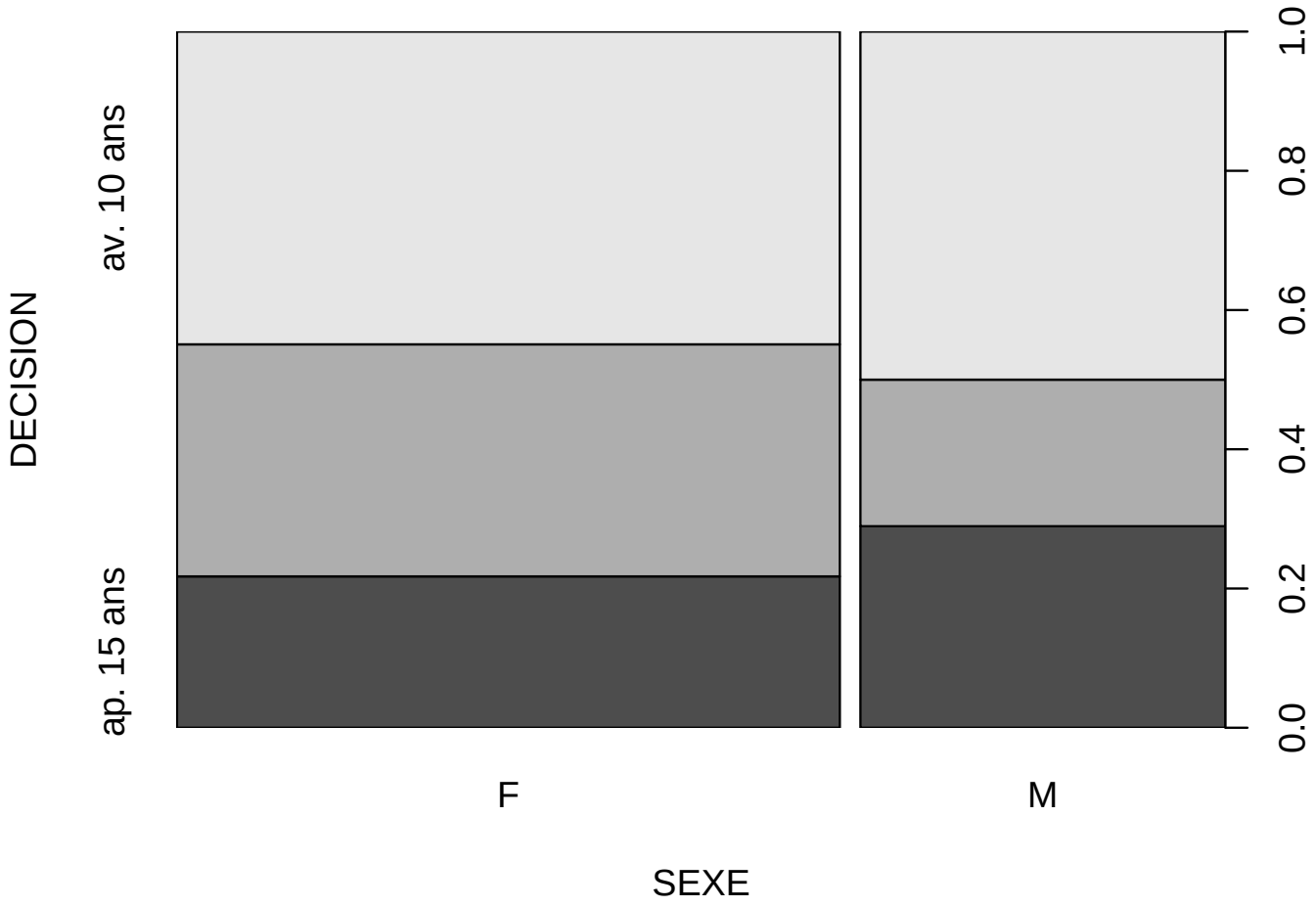
```
plot(d$SEXE, d$POIDS)
boxplot(POIDS ~ SEXE, data = d)
```

Représentation des variables qualitatives - travail préliminaire souvent nécessaire

```
 levels(d$DECISION)
[1] "A" "E" "T"
 ## Changement du nom des modalités ##
 ###################################
 levels(d$DECISION) <-
   c("10-15 ans", "av. 10 ans", "ap. 15 ans")
 ## Changement de l'ordre des modalités ##
 ########################################
 d$DECISION <- factor(d$DECISION,
   levels = c("av. 10 ans", "10-15 ans", "ap. 15 an
```

`plot(varqual,varqual)` : diagrammes en bandes

```
plot(DECISION ~ SEXE, data = d)
```

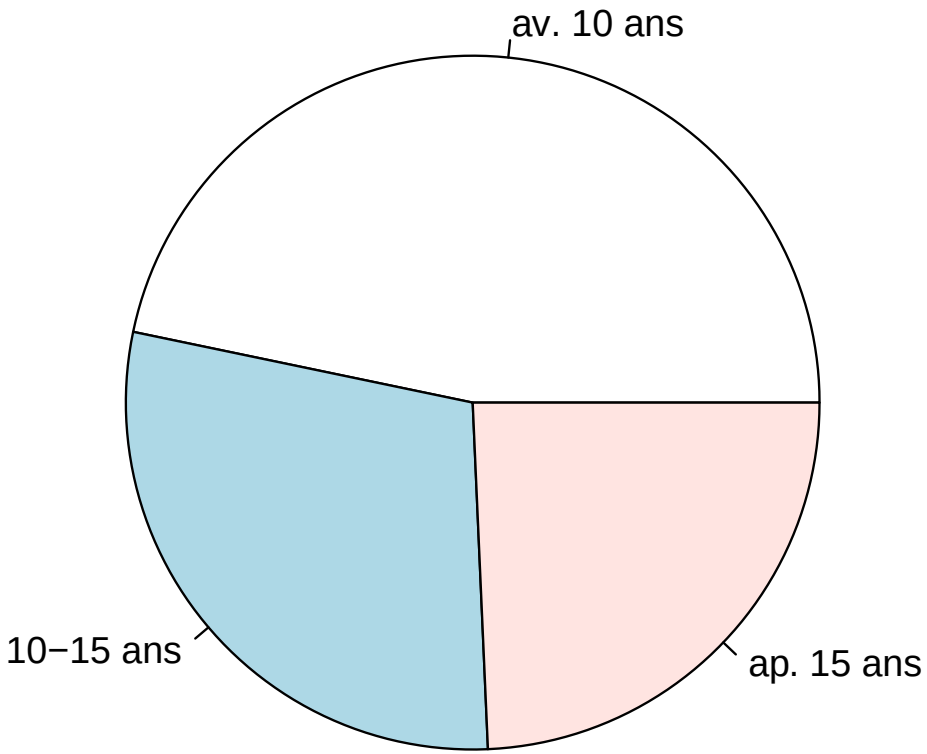


Autres fonctions du package graphics

Dans chacun des cas présentés, il existe des alternatives à la réalisation du graphe proposé par défaut par la fonction `plot()`.
Pour cela il suffit de faire appel à d'autres fonctions graphiques dont nous allons présenter les plus classiques.

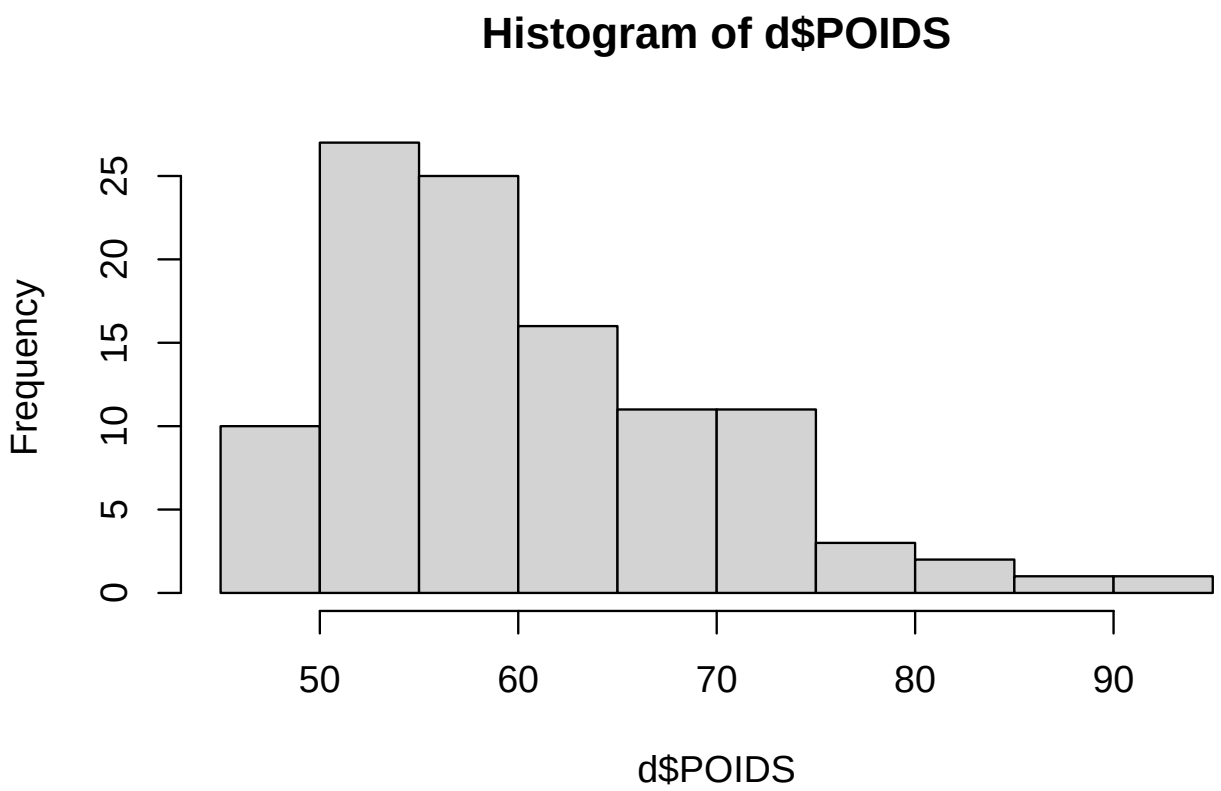
`pie(table(varqual))` : diagramme en secteurs

`pie(table(d$DECISION))`



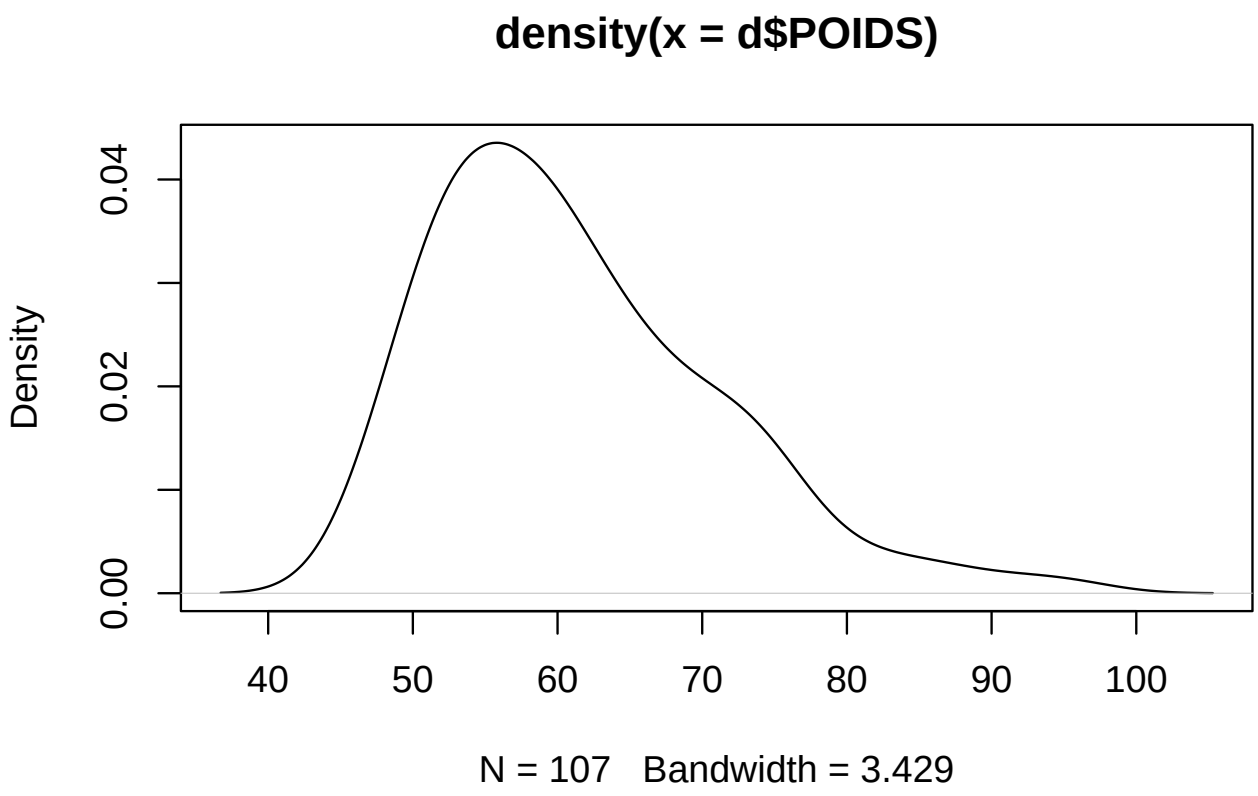
`hist(varquant)` : histogramme de fréquences

```
hist(d$POIDS)
```



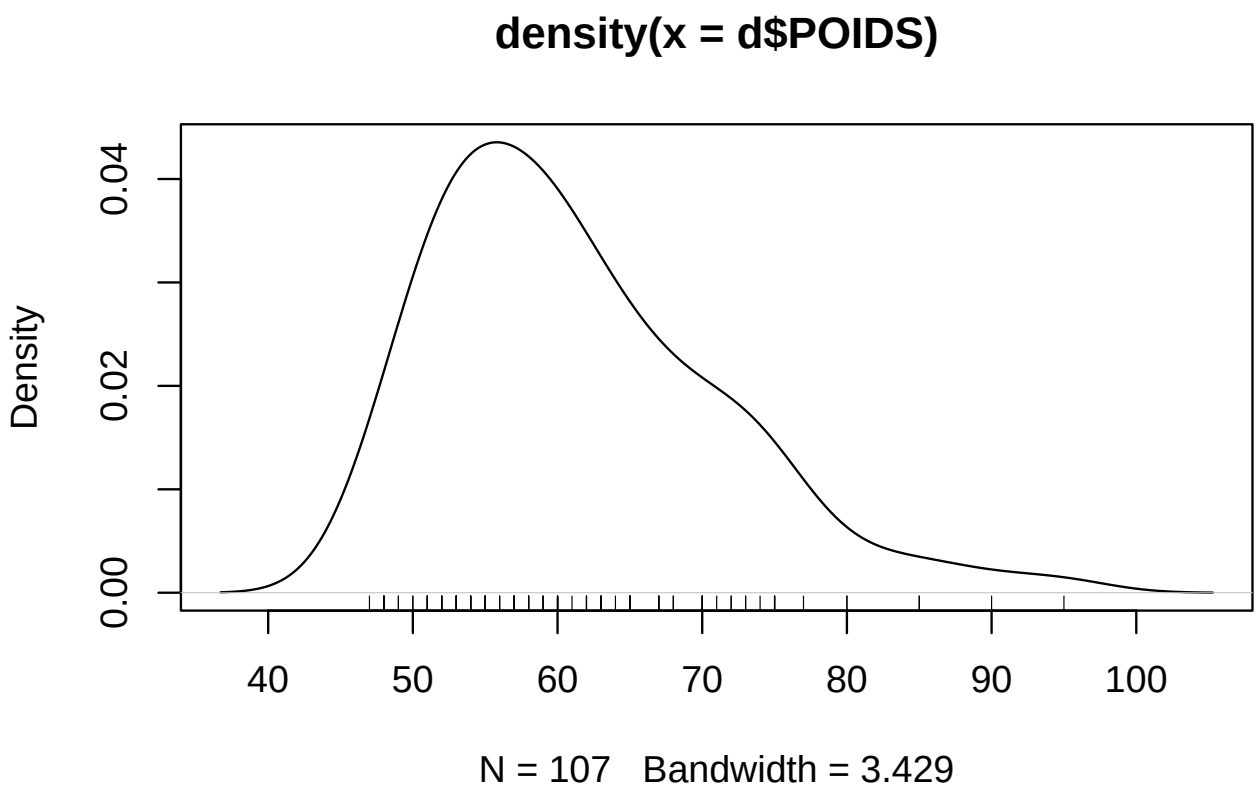
`plot(density(varquant))` : estimateur de la densité

```
plot(density(d$POIDS))
```



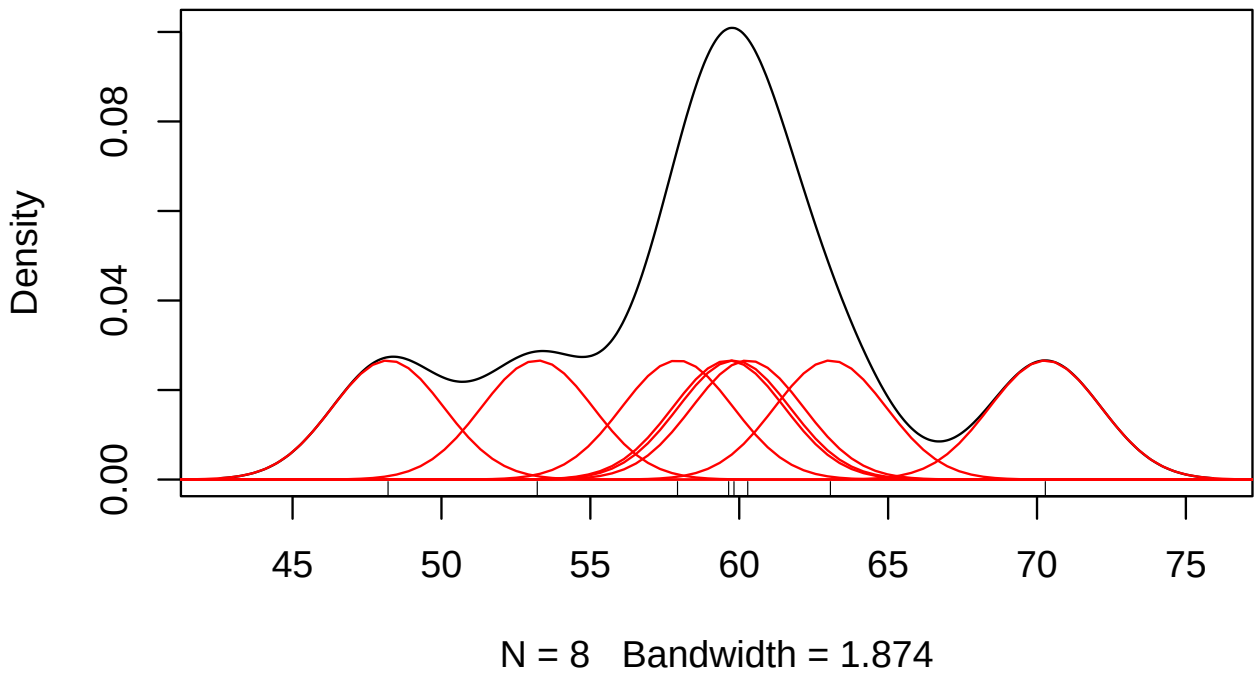
plot(density(varquant)) avec ajout des valeurs

```
plot(density(d$POIDS))
rug(d$POIDS)
```



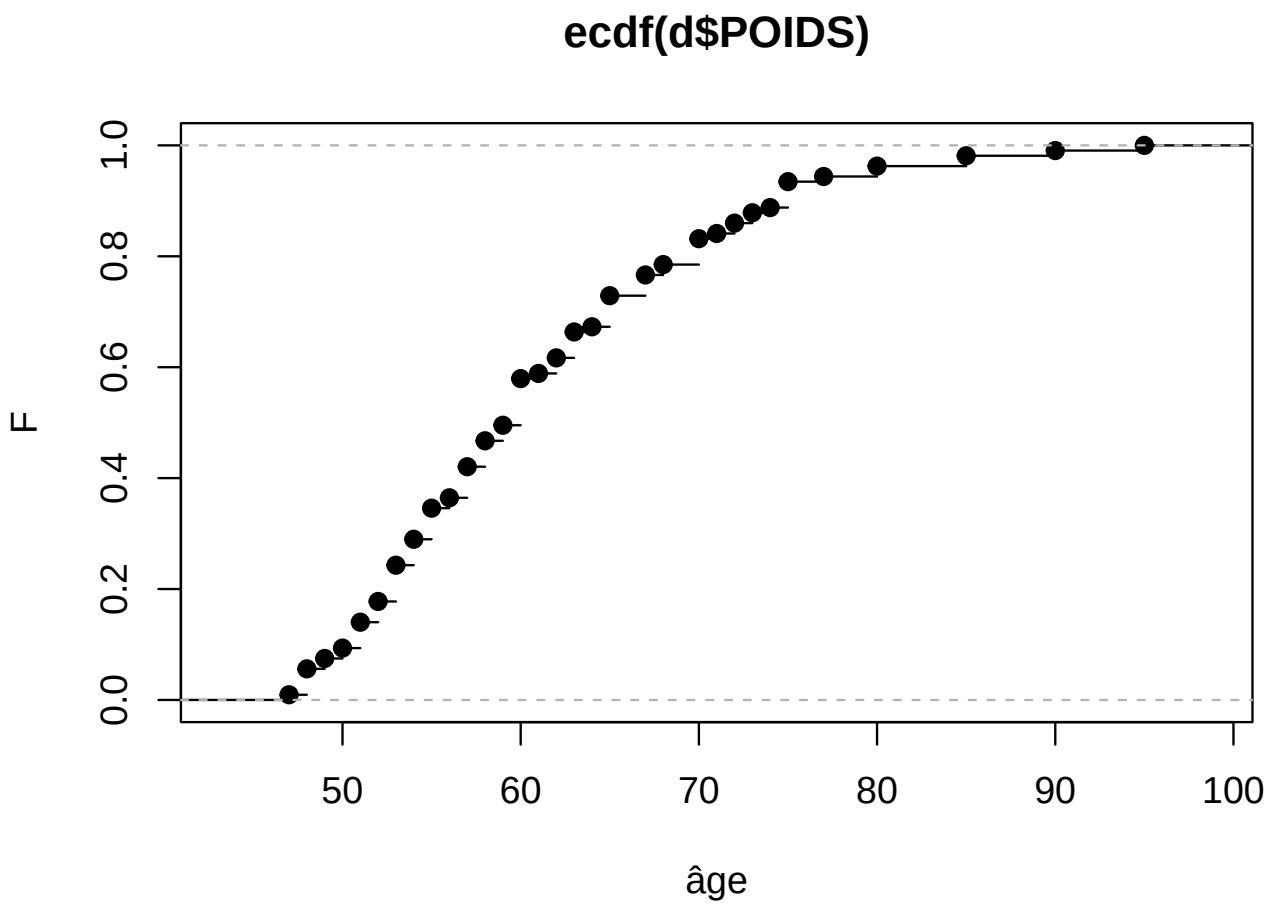
Principe de l'estimateur à noyau de la densité de probabilité illustré sur un sous-échantillon de 8 poids

Mélange de 8 distributions normales centrées sur les 8 observations et de même écart type ajustable (ici 5.32)



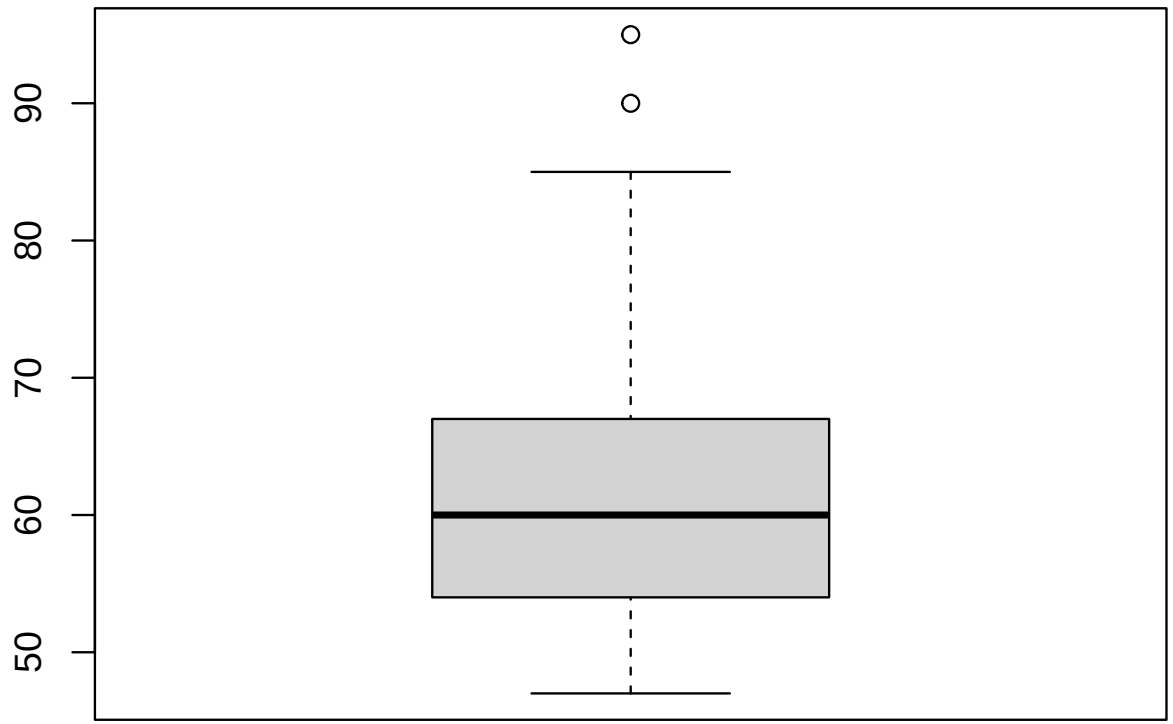
`plot(ecdf(varquant))` : diagramme des fréquences cumulées

```
plot(ecdf(d$POIDS),xlab = "âge", ylab = "F")
```



`boxplot(varquant)` : diagramme en boîte

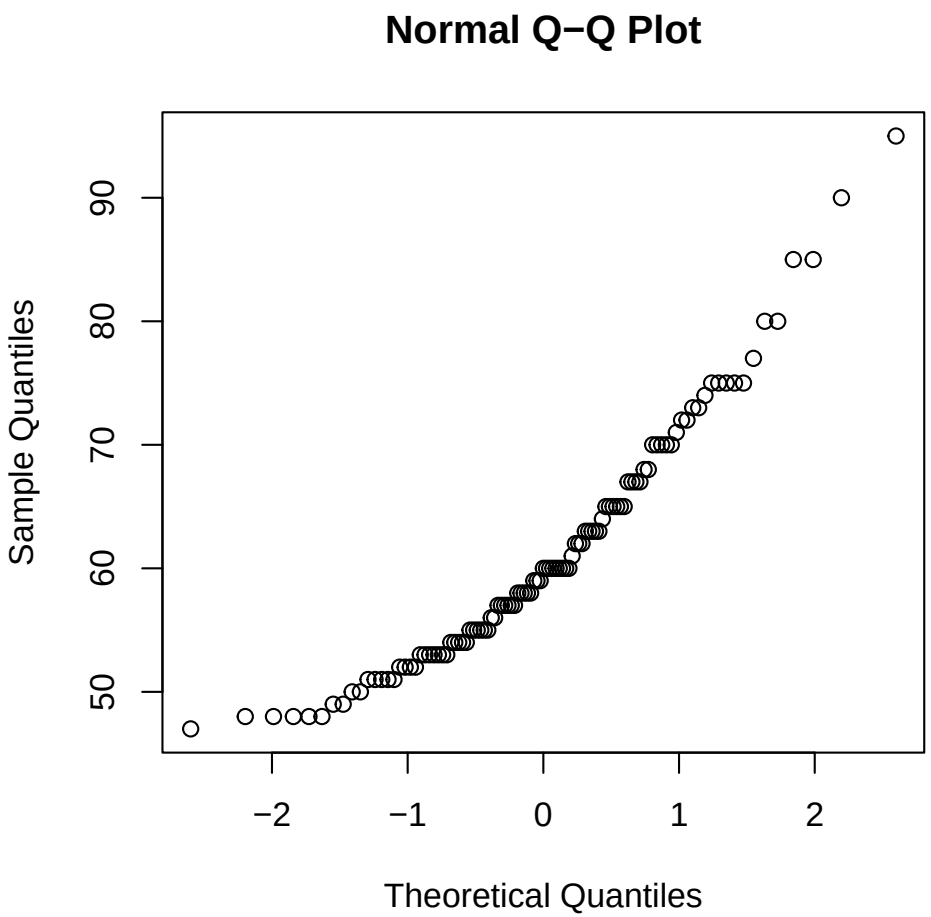
```
boxplot(d$POIDS)
```



qqnorm(varquant) : diagramme Quantile - Quantile

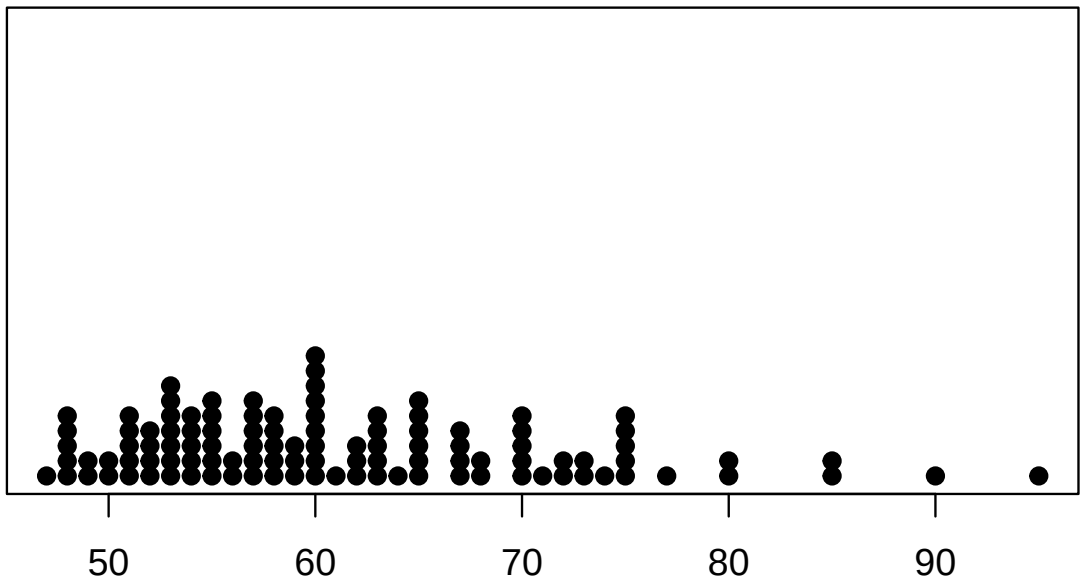
Quantiles observés en fonction de ceux d'une loi normale pour vérifier la normalité de la distribution (points alignés)

```
qqnorm(d$POIDS)
```



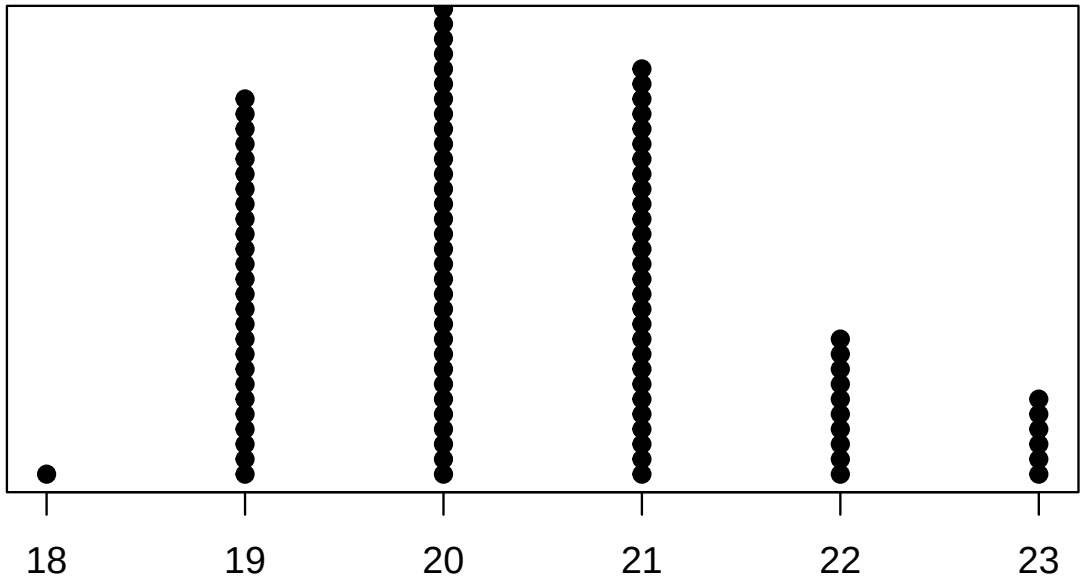
`stripchart(varquant)` : visualisation de tous les points

```
stripchart(d$POIDS, method = "stack",
           pch = 19, at = 0)
```



`stripchart(varquant)` : particulièrement adaptée pour une variable quantitative discrète ou de mesure discrète

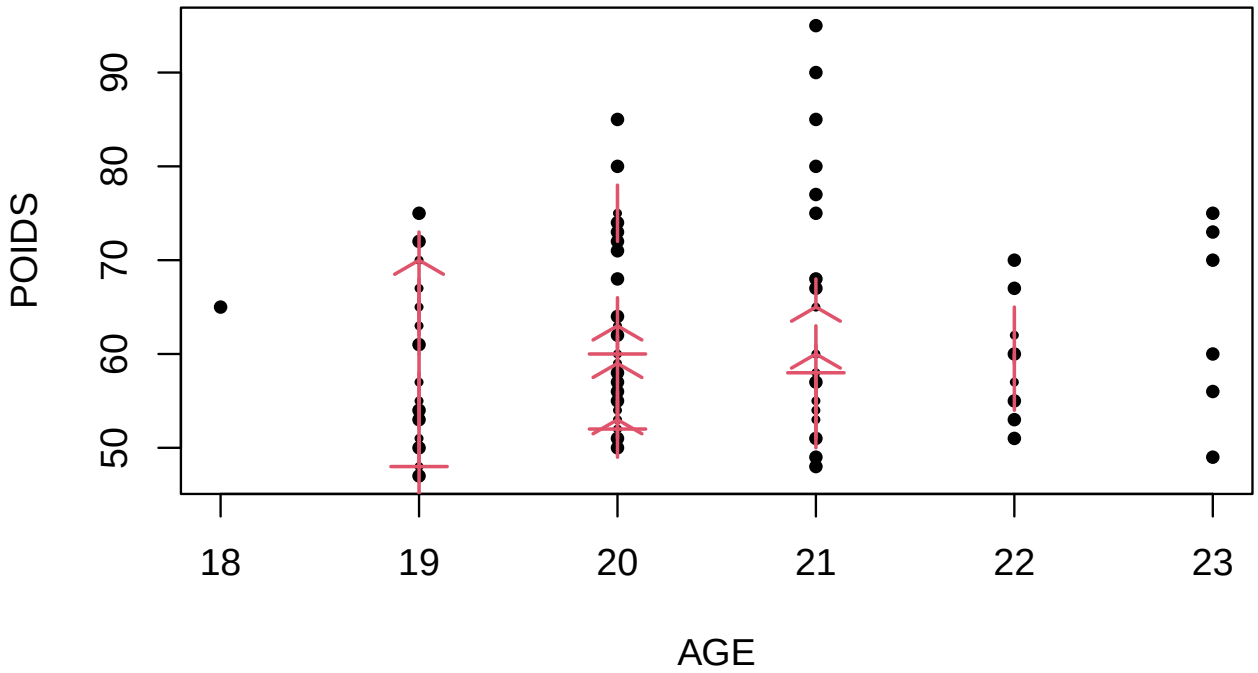
```
stripchart(d$AGE, method = "stack",
           pch = 19, at = 0)
```



`sunflowerplot(varquant,varquant)` : nuage de points avec tournesols

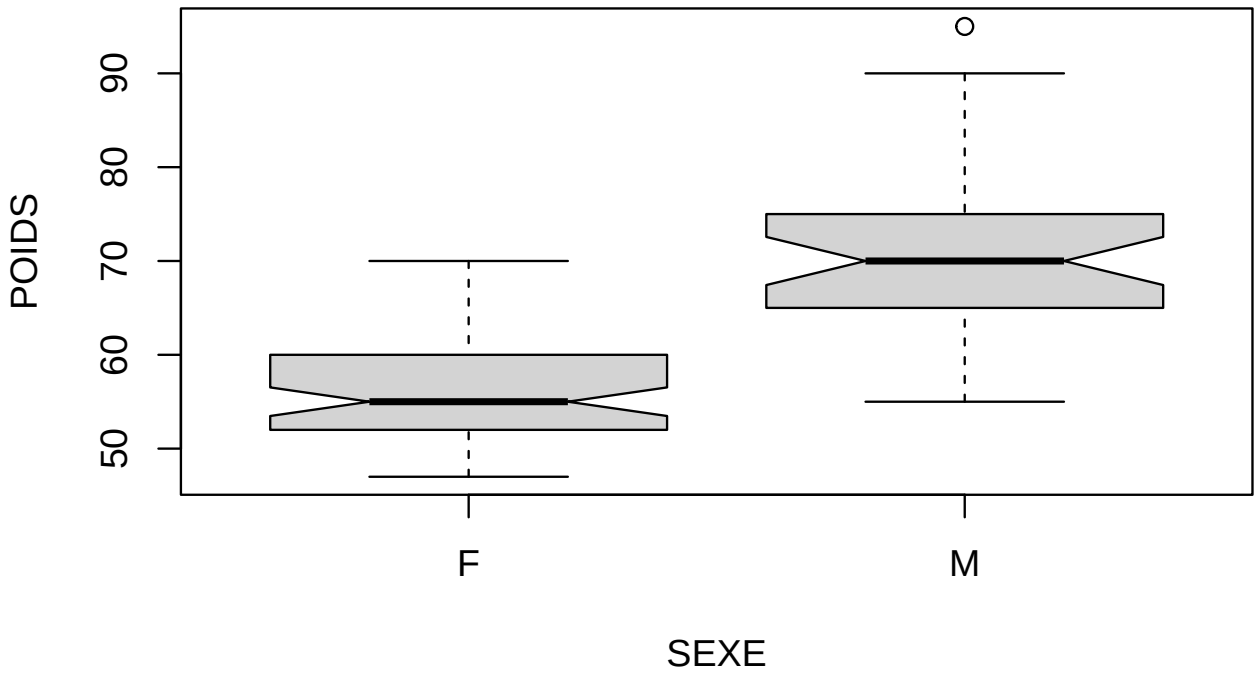
Visualisation des points superposés par des tournesols

```
sunflowerplot(POIDS ~ AGE, data = d)
```



`boxplot(vquant ~ vqual, notch = TRUE)` : variante des diagrammes en boîtes avec intervalles de confiance approchés sur les médianes

`boxplot(POIDS ~ SEXE, data = d, notch = TRUE)`



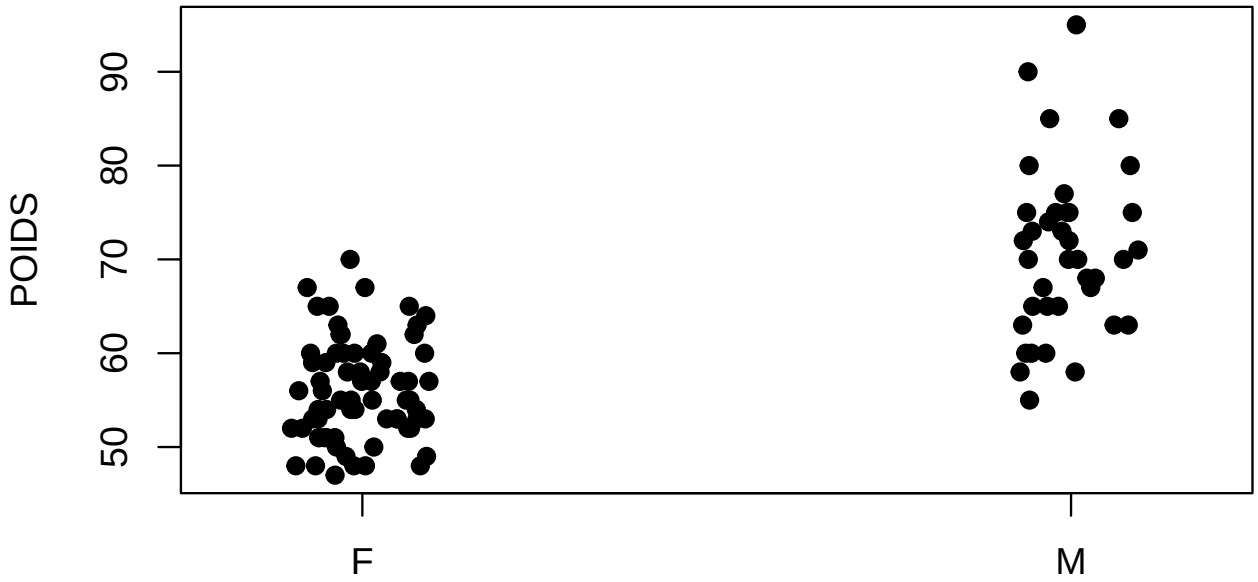
`stripchart(varquant ~ varqual)` : report des points par groupe

```
stripchart(POIDS ~ SEXE, data = d,
           vertical = TRUE)
```

Problème : on ne distingue pas les ex-aequos

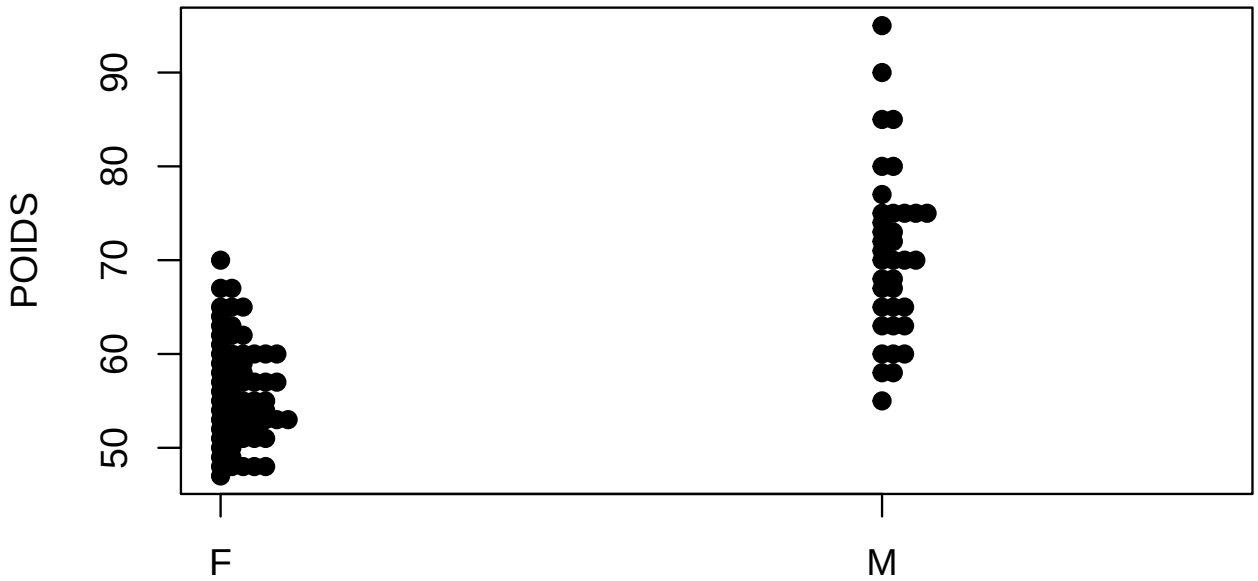
`stripchart(..., method = "jitter")` : bruitage des abscisses et changement du type de points

```
stripchart(POIDS ~ SEXE, data = d,
  vertical = TRUE, method = "jitter", pch = 19)
```



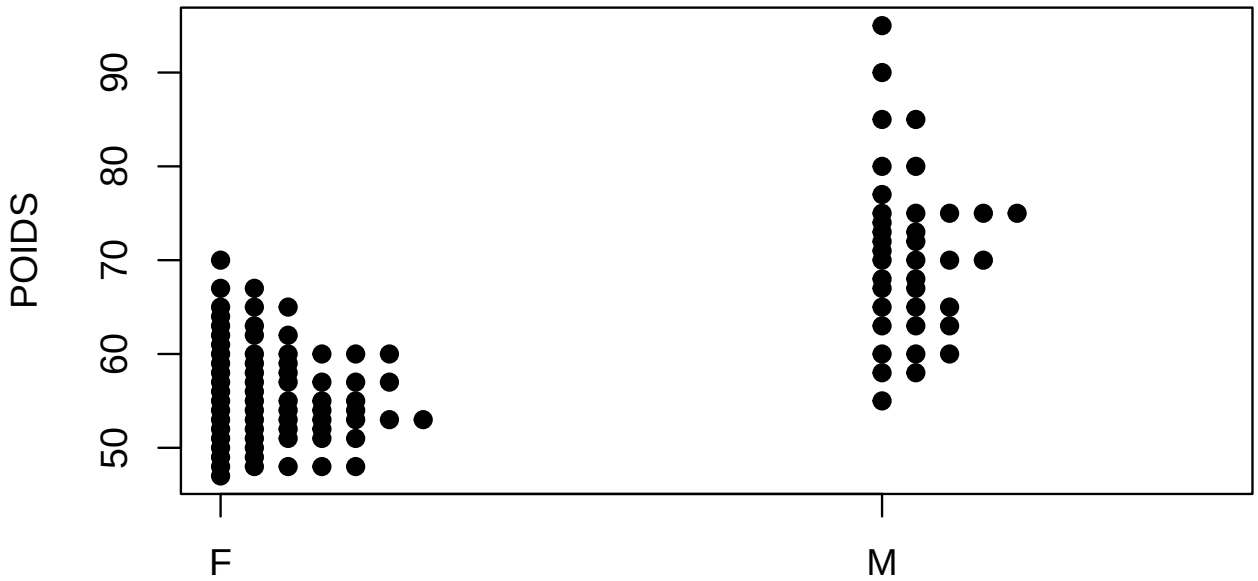
`stripchart(..., method = "stack")` : empilement des points ex-aequos

```
stripchart(POIDS ~ SEXE, data = d,
  vertical = TRUE, method = "stack", pch = 19)
```



`stripchart(..., method = "stack")` : variante en écartant les points

```
stripchart(POIDS ~ SEXE, data = d, vertical = TRUE
           method = "stack", pch = 19, offset = 1)
```



Diagrammes en bâtons accolés (1)

```
# Calcul de la table de contingence
(t <- table(d$SEXE, d$DECISION))
```

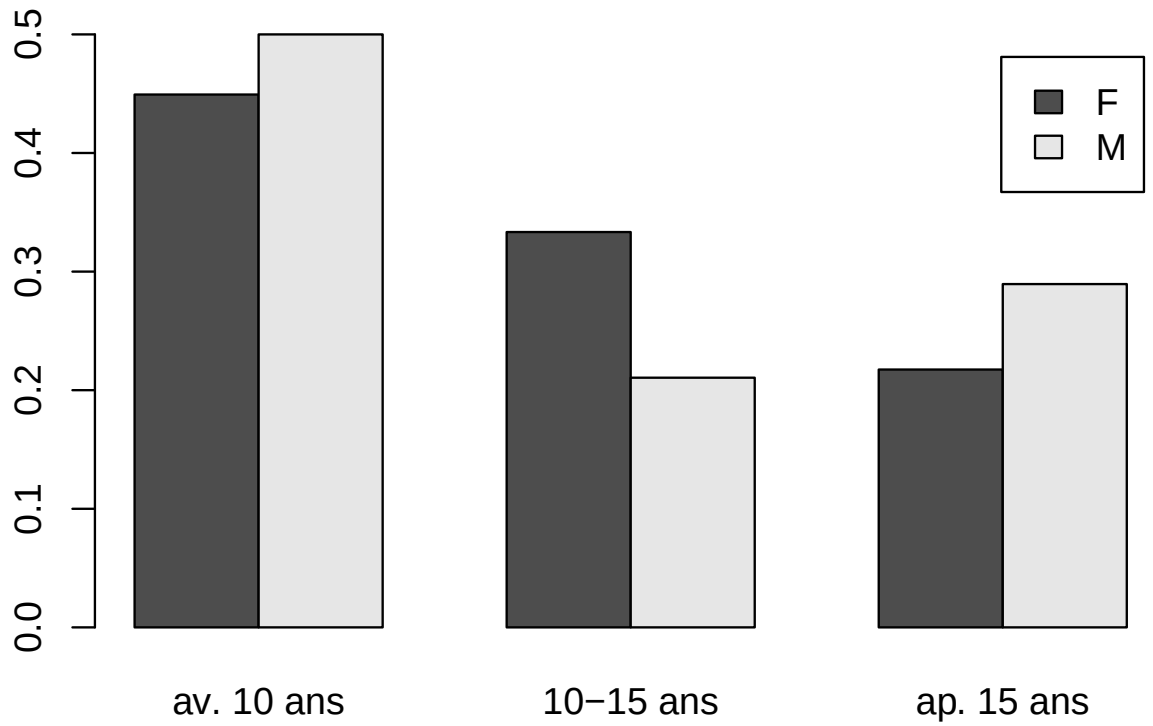
```
   av. 10 ans 10-15 ans ap. 15 ans
F          31         23          15
M          19          8          11
```

```
# Calcul des fréquences sur chaque ligne
(tf <- prop.table(t, margin = 1))
```

```
   av. 10 ans 10-15 ans ap. 15 ans
F      0.449      0.333      0.217
M      0.500      0.211      0.289
```

Diagrammes en bâtons accolés (2)

```
barplot(tf, beside = TRUE, legend.text = TRUE)
```



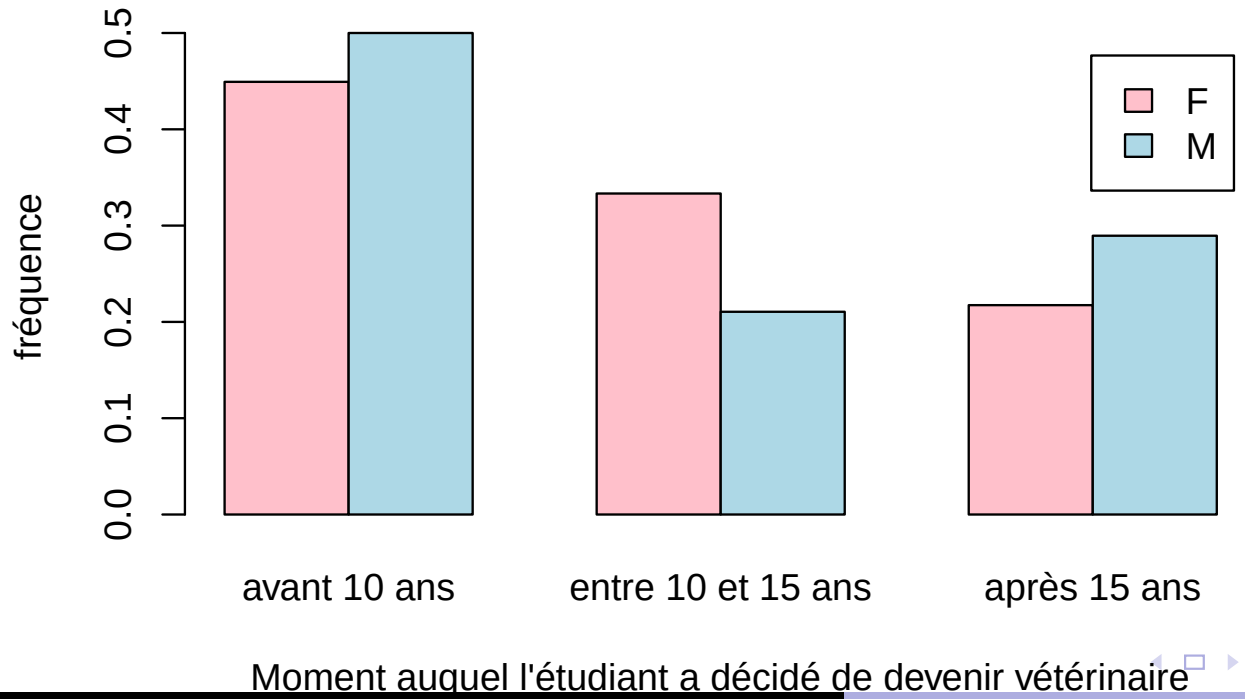
Personnalisation des graphes en utilisant les arguments de fonctions graphiques

Il est très simple d'accéder à l'aide de chaque fonction **R** qui décrit en particulier tous les arguments de la fonction.
Exemples :

```
?barplot
?stripchart
?boxplot
```

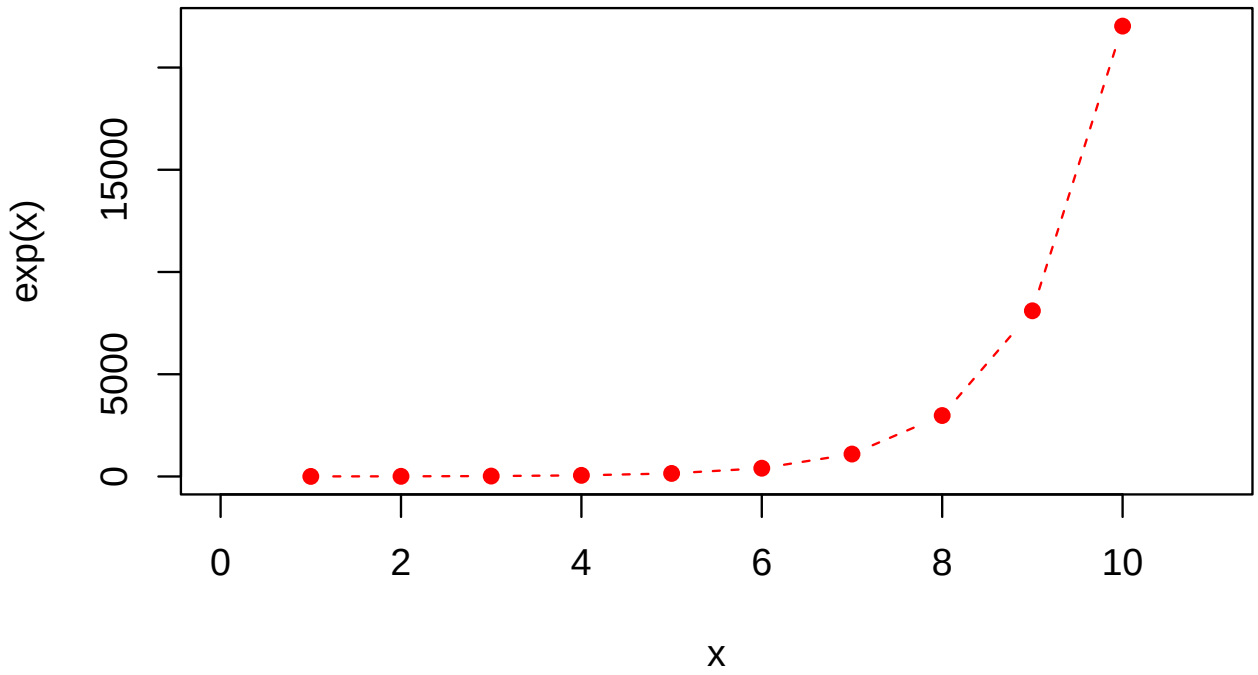
Exemple de diagramme en bâtons utilisant les arguments de la fonction `barplot()`

```
barplot(tf, beside = TRUE,
  names.arg = c("avant 10 ans","entre 10 et 15 ans", "après 15 ans"),
  ylab = "fréquence", legend.text = TRUE,
  xlab = "Moment auquel l'étudiant a décidé de devenir vétérinaire",
  col = c("pink", "lightblue"))
```



Exemple de nuage de points avec points reliés utilisant la fonction `plot()`

```
plot(1:10, exp(1:10), type = "b", pch = 16, lty = 2, col = "red",
xlab = "x", ylab = "exp(x)", xlim = c(0,11))
```



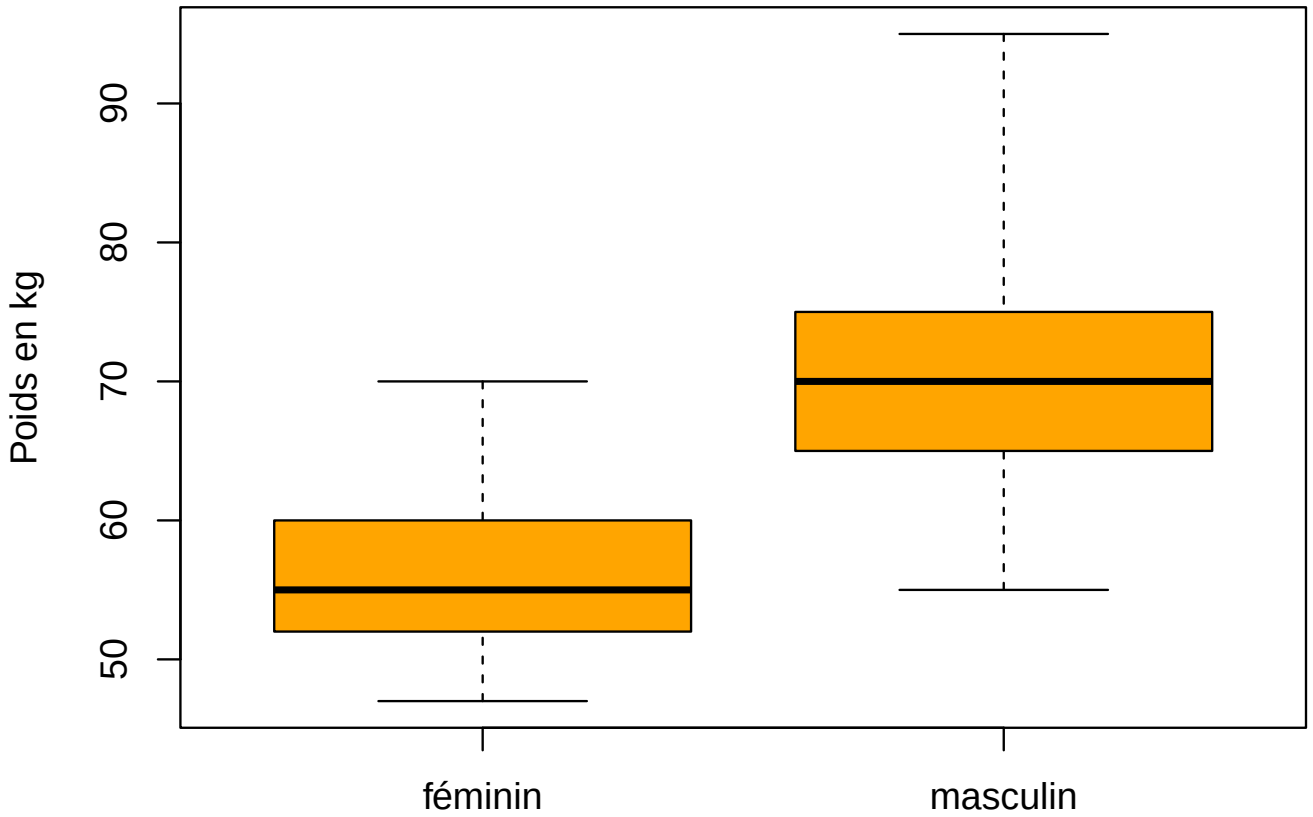
Quelques arguments classiques à connaître

- `type` (type de tracé)
- `pch` (type de point)
- `lty` (type de ligne)
- `lwd` (largeur de la ligne)
- `cex` (de combien le texte et les symboles de tracé doivent être agrandis par rapport à la valeur par défaut)
- `col` (couleur)
- `xlim, ylim` (limites sur x et y)
- `xlab,ylab` (noms des axes)
- `main` (titre du graphe)

A vous de jouer !

Consigne

Pour vous familiarisez avec les fonctions graphiques et leurs arguments, tentez de reproduire le graphe suivant.



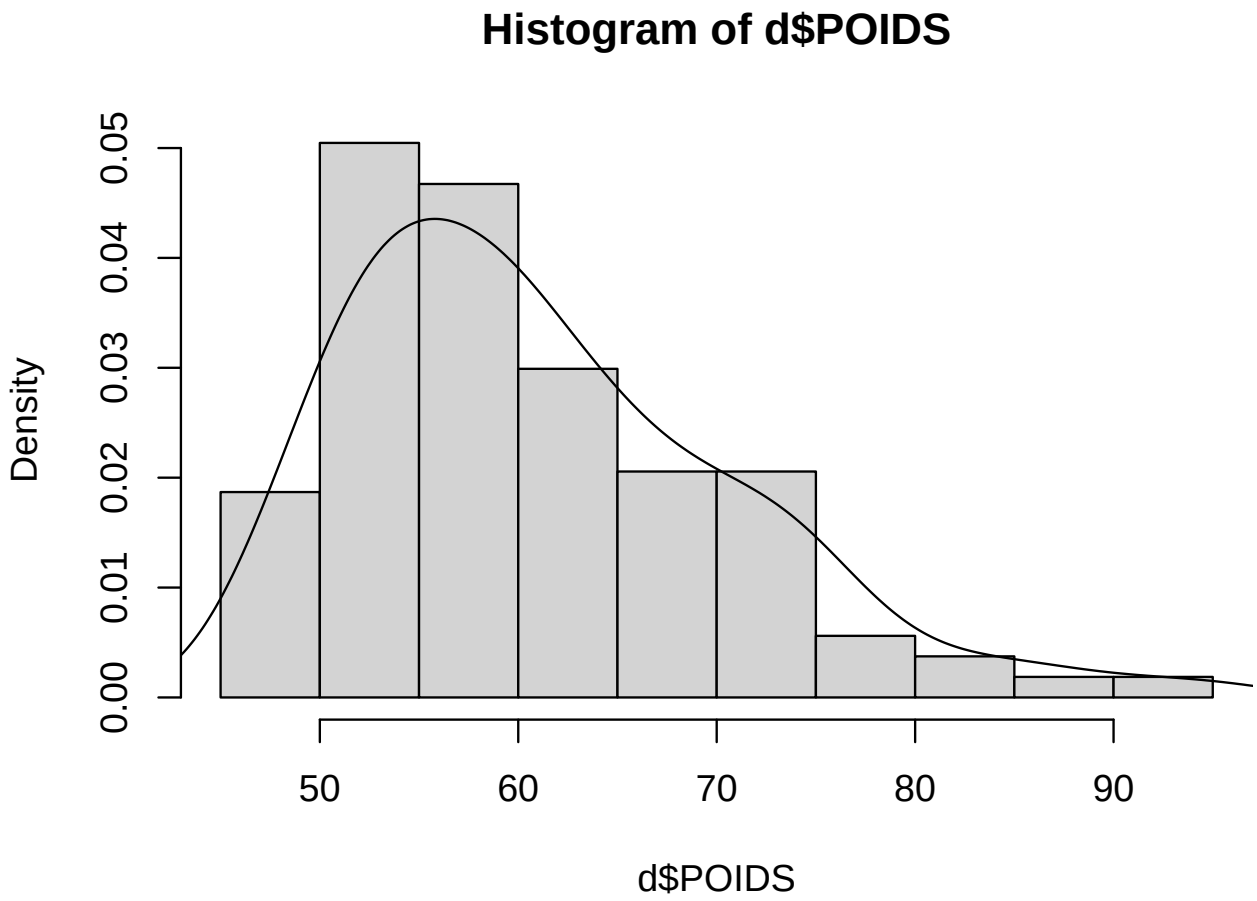
Autres fonctions

Afin de réaliser de belles figures publiables dans un article ou présentables dans un diaporama, quelques autres fonctionnalités graphiques de **R** sont à connaître :

- ajout d'un tracé (2^{eme} graphe) à un graphe existant
- ajout de texte à un graphe existant
- organisation des graphes sur une fenêtre
- divers paramètres d'une fenêtre graphique
- export d'une fenêtre graphique

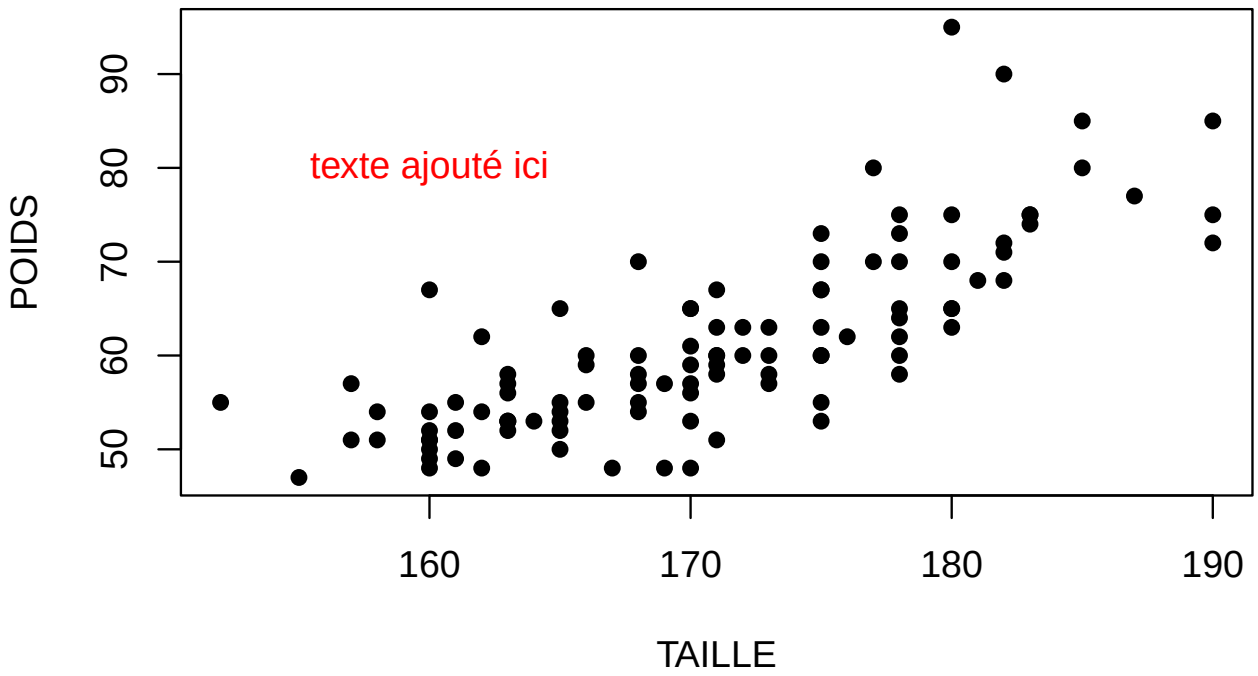
Ajout d'une ligne : fonction `lines()`

```
hist(d$POIDS, freq = FALSE)
lines(density(d$POIDS))
```



Ajout d'un texte : fonction `text()`

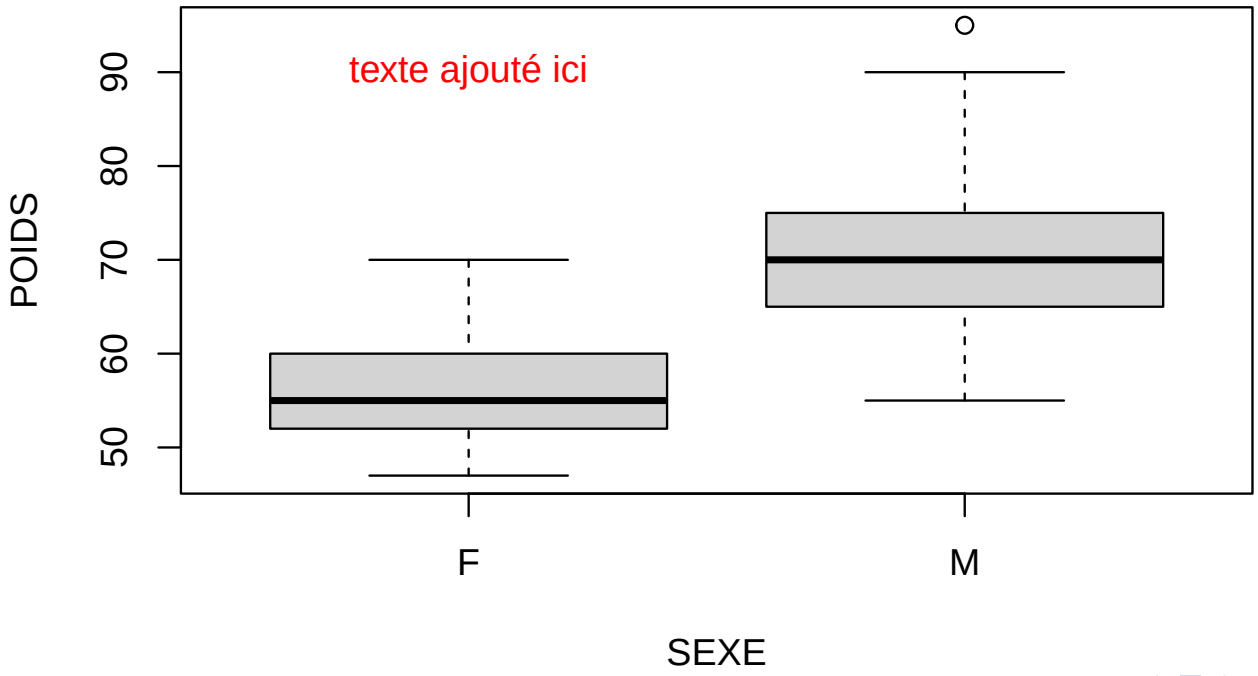
```
plot(POIDS ~ TAILLE, data = d, pch = 16)
text(x = 160, y = 80, labels = "texte ajouté ici", col = "red")
```



On ajoute ainsi un texte à l'abscisse et l'ordonnée indiquées.

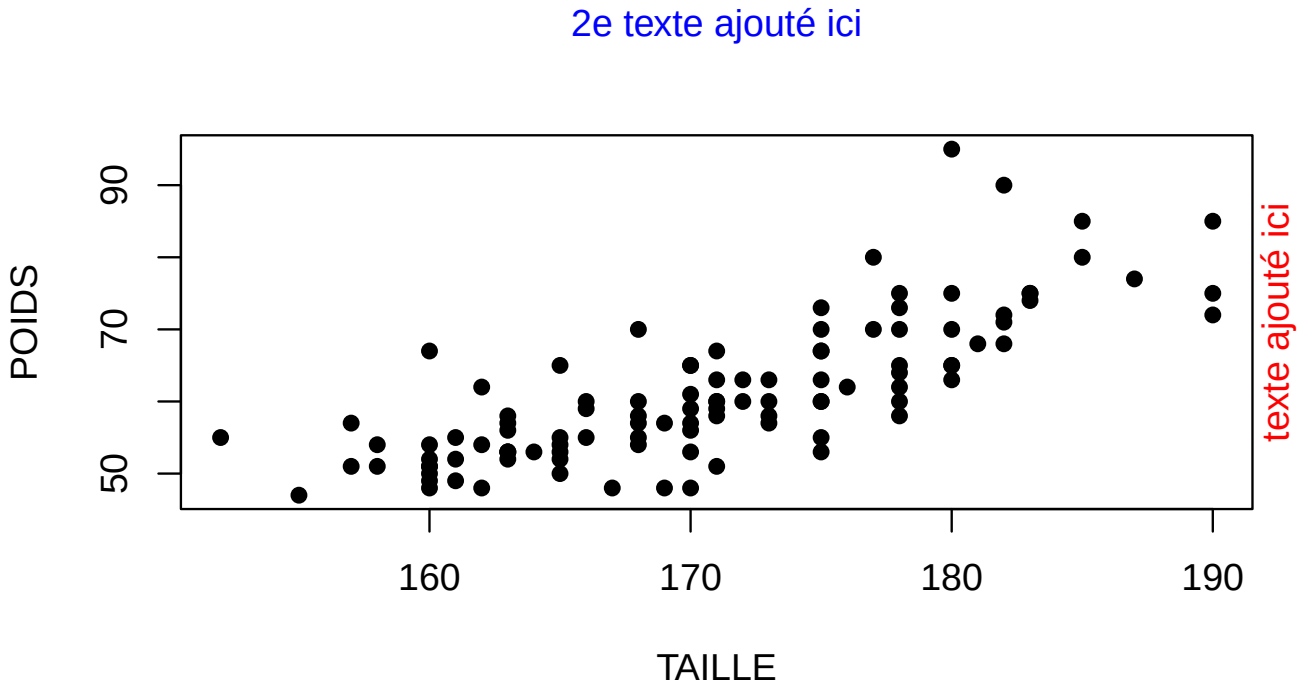
Utilisation du paramètre "usr" pour les variables qualitatives (coordonnées numériques des axes)

```
boxplot(POIDS ~ SEXE, data = d)
par("usr")
[1]  0.42  2.58 45.08 96.92
text(x = 1, y = 90, labels = "texte ajouté ici", col = "red")
```



Ajout d'un texte dans la marge : fonction `mtext()`

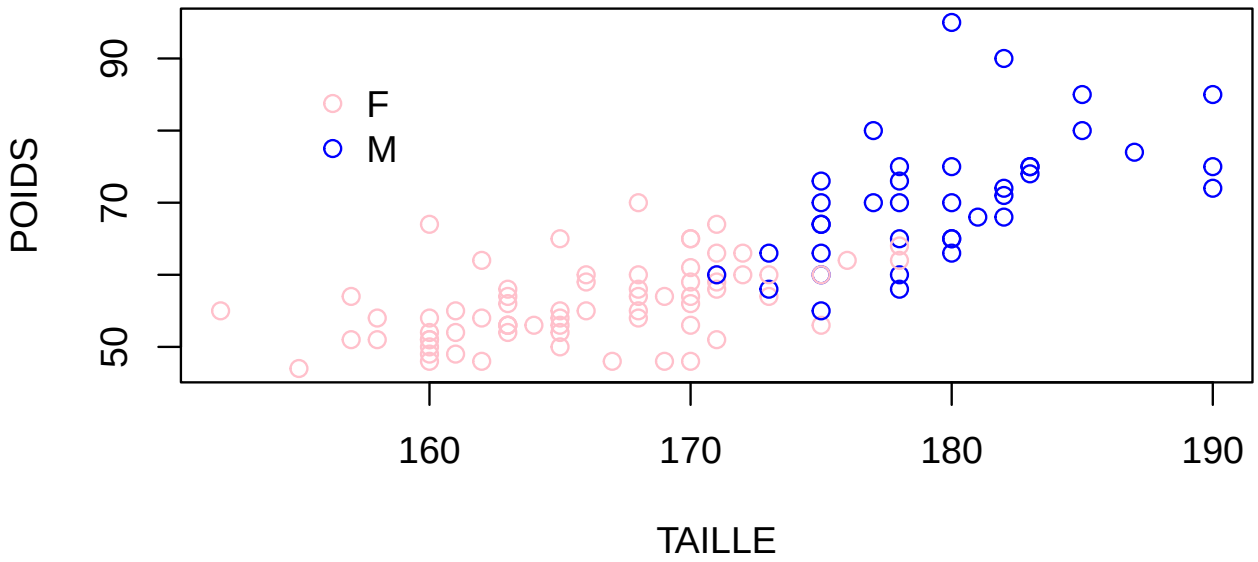
```
plot(POIDS ~ TAILLE, data = d, pch=16)
mtext(text = "texte ajouté ici", col = "red", side = 4, line = 0)
mtext(text = "2e texte ajouté ici", col = "blue", side = 3, line = 2)
```



On ajoute ainsi un texte dans la marge codée par `side` et la ligne codée par `line`.

Ajout d'une légende : fonction `legend()`

```
plot(POIDS ~ TAILLE, data = d,col = c("pink", "blue")[d$SEXE])
legend(x = 155, y = 90, legend=c("F","M"), col = c("pink","blue"),
       bty = "n", pch = 1)
```



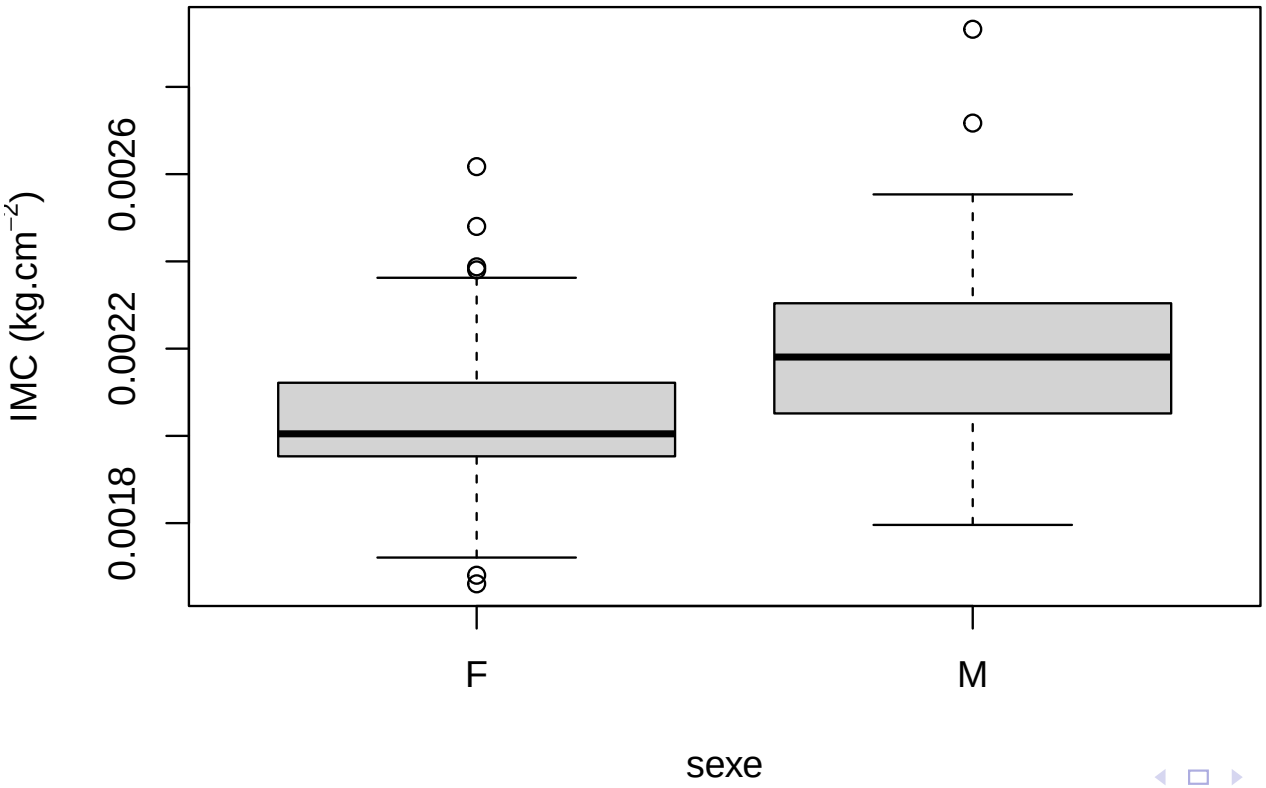
Le premier argument `x` de la fonction `legend()` peut aussi être fixé à

`"bottomleft", "topright", "center", ...`

Ecriture mathématique dans un texte ajouté

Comme insérer symboles, exposants, indices dans une légende, une étiquette, un titre ?

```
plot(POIDS / TAILLE^2 ~ SEXE, data = d, xlab="sexe",
     ylab = expression( paste( "IMC (kg.", cm^-2, ")")))
```



Codage des écritures mathématiques

Consigne

Explorez l'aide de plotmath et/ou tapez `demo(plotmath)`.

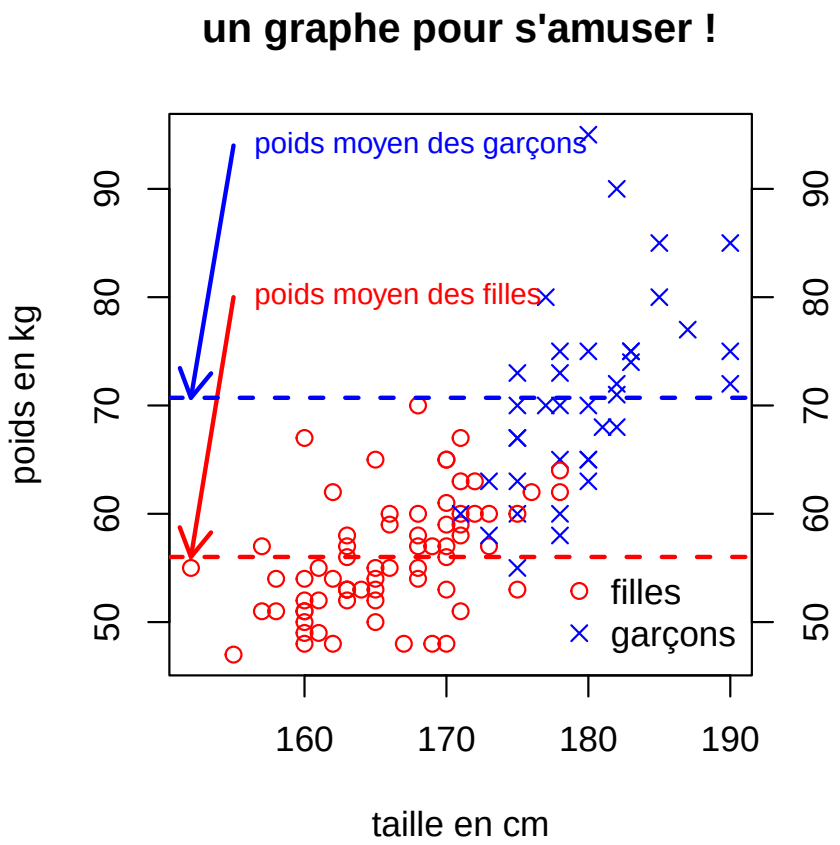
Autres ajouts

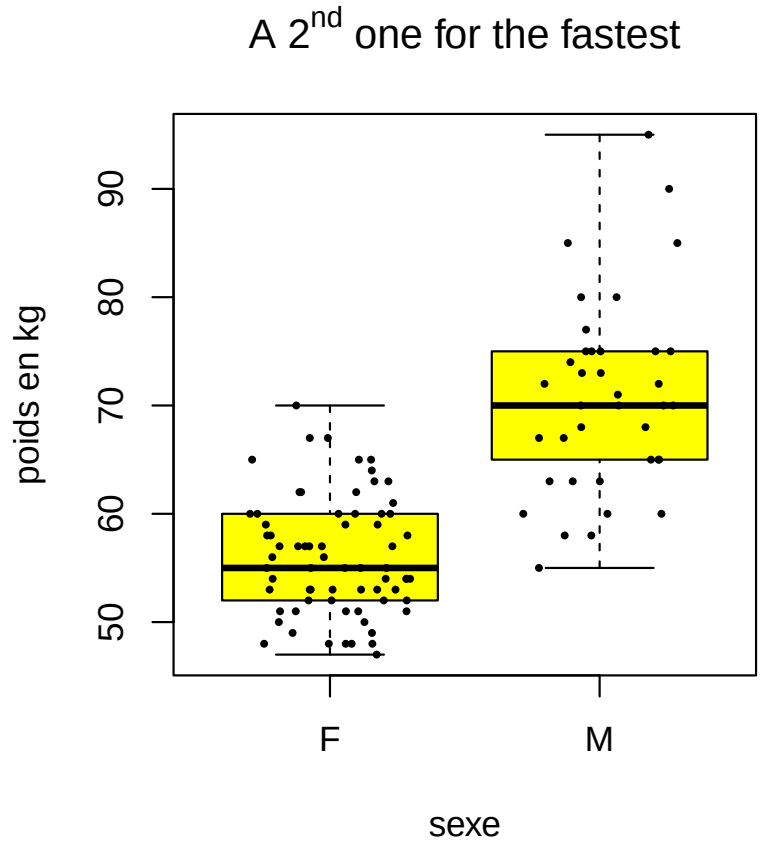
- ajout d'une flêche (fonction `arrows()`)
- ajout d'un segment (fonction `segments()`)
- ajout d'une ligne droite (fonction `abline()`)
- ajout d'un ou plusieurs points (fonction `points()`)
- ajout d'un axe (fonction `axis()`)

On peut aussi ajouter l'argument `add = TRUE` dans pas mal de fonctions graphiques (par exemple `hist()`, `stripchart()`, ...) pour faire un graphe par dessus un autre.

Consigne

En vous aidant de l'aide en ligne, tentez de refaire la figure de gauche, et si vous allez très vite celle de droite en prime





Organisation de plusieurs graphes sur une même fenêtre

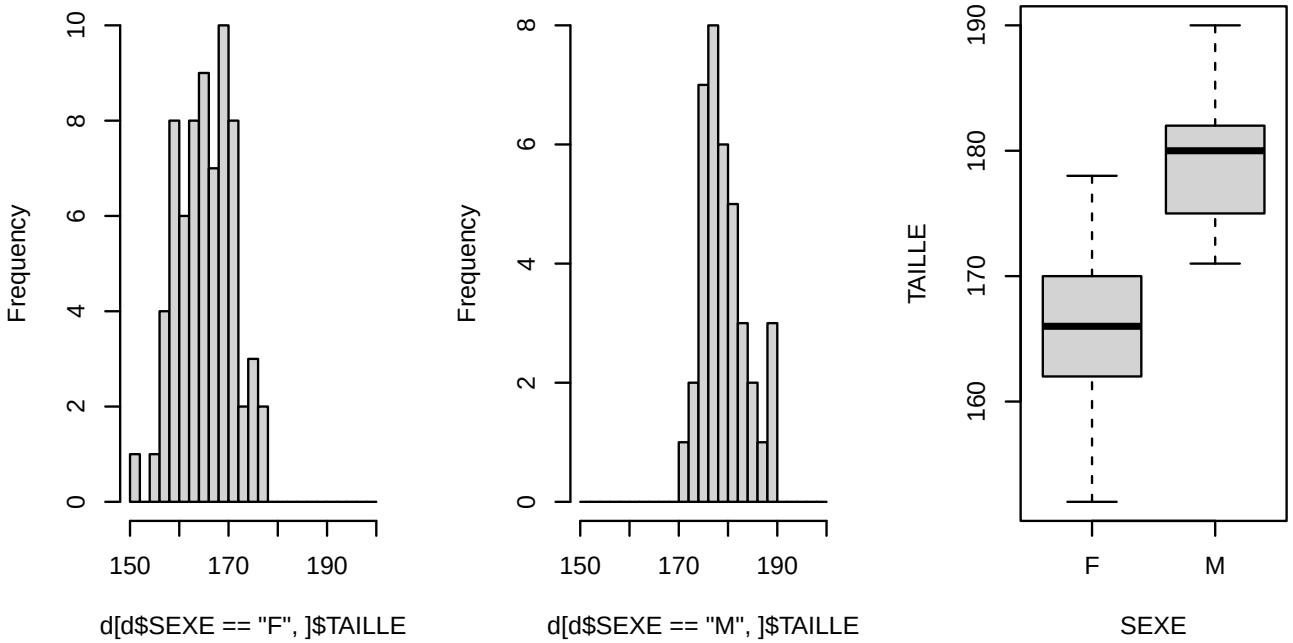
Avant le premier tracé, utilisation de l'une des deux fonctions suivantes :

- `par(mfrow=c(k,l))` avec `k` et `l` les nombres de lignes et de colonnes pour un découpage régulier de la fenêtre graphique en `k*l` cellules
- `layout(m)` pour un découpage régulier suivi d'un regroupement de certaines cellules, tel que spécifié dans la matrice `m`.
  Ex. pour deux graphes de même largeur en haut et un 2 fois plus large en bas :

```
m <- matrix(c(1,2,3,3), nrow = 2, byrow = TRUE)
```

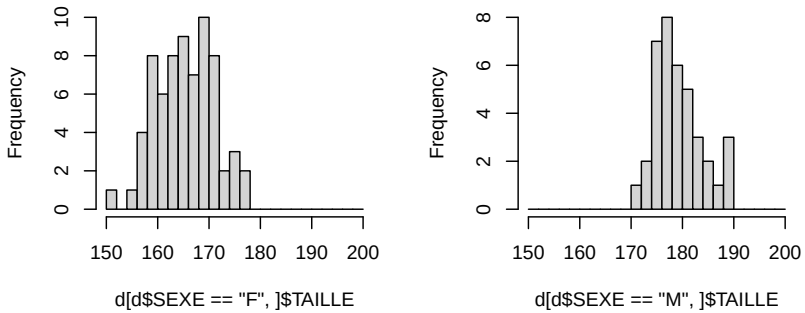
Exemple d'utilisation de `par(mfrow)`

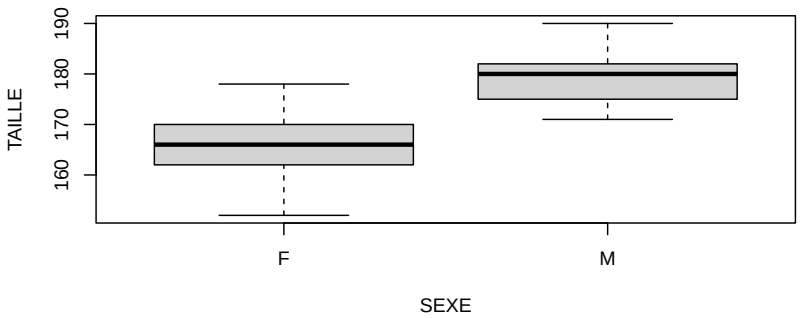
```
par(mfrow=c(1,3))
hist(d[d$SEXE=="F",]$TAILLE, xlim = c(150,200),
     breaks = seq(150,200,2), main = "")
hist(d[d$SEXE == "M",]$TAILLE, xlim = c(150,200),
     breaks = seq(150,200,2), main ="")
plot(TAILLE ~ SEXE, data = d)
```



Exemple d'utilisation de la fonction `layout()`

```
m <- matrix(c(1,2,3,3), nrow = 2, byrow = TRUE)
layout(m)
hist(d[d$SEXE == "F",]$TAILLE, xlim = c(150,200),
     breaks = seq(150,200,2), main = "")
hist(d[d$SEXE == "M",]$TAILLE, xlim = c(150,200),
     breaks = seq(150,200,2), main = "")
plot(TAILLE ~ SEXE, data = d)
```





Modification des paramètres graphiques par défaut - marges

De nombreux paramètres graphiques sont récupérales ou modifiables à l'aide de la fonction `par()` à appliquer sur une fenêtre graphique ouverte, **mais avant d'y réaliser le tracé**.
Il est par exemple souvent utile de modifier les marges avant un découpage de fenêtre graphique.
Pour cela on utilise la commande

```
par(mar = c(bottom, left, top, right))
```

Les quatre valeurs données indiquent les largeurs des marges respectivement
en bas, à gauche, en haut et à droite,
et il faut savoir que par défaut elles sont à
`c(5.1, 4.1, 4.1, 2.1)`.

Modification des paramètres graphiques par défaut - autres paramètres

Pour explorer quelques autres paramètres, faire :

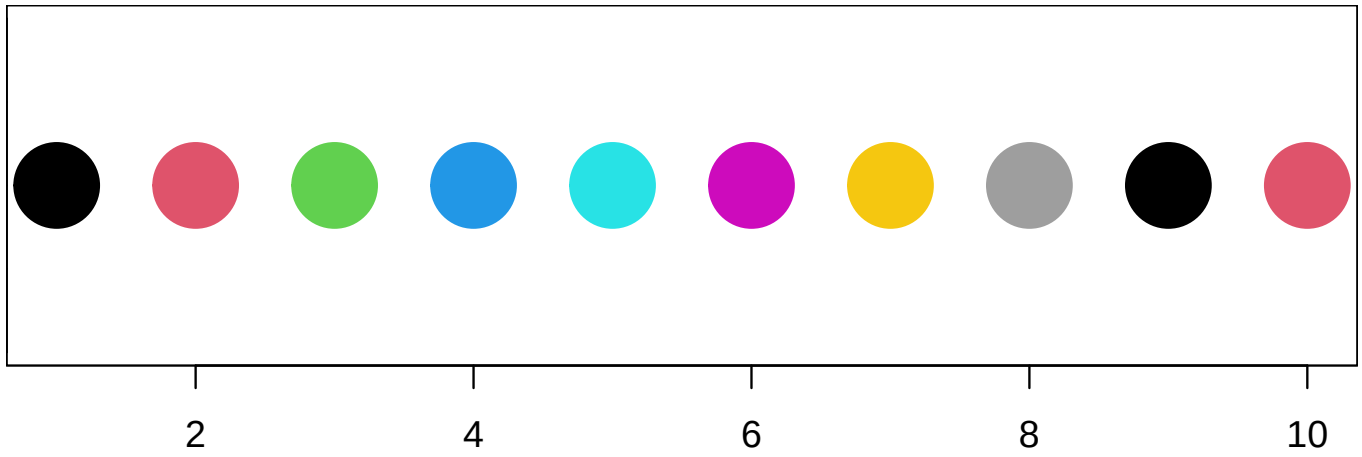
```
?par
```

Ce serait trop long et complexe de tout explorer mais regardez-en voire testez en quelques-uns : `bg`, `bty`, `las`, `mar`, `mfrow`, `usr`, `xaxt`, `col` et `yaxt`, `xlog` et `ylog`.

Gestion des couleurs de base

- On peut coder les 8 couleurs de base par leur numéros.

```
par(mar = c(2, 0, 0, 0))
plot(rep(1,10), col = 1:10, pch = 19, cex = 5)
```

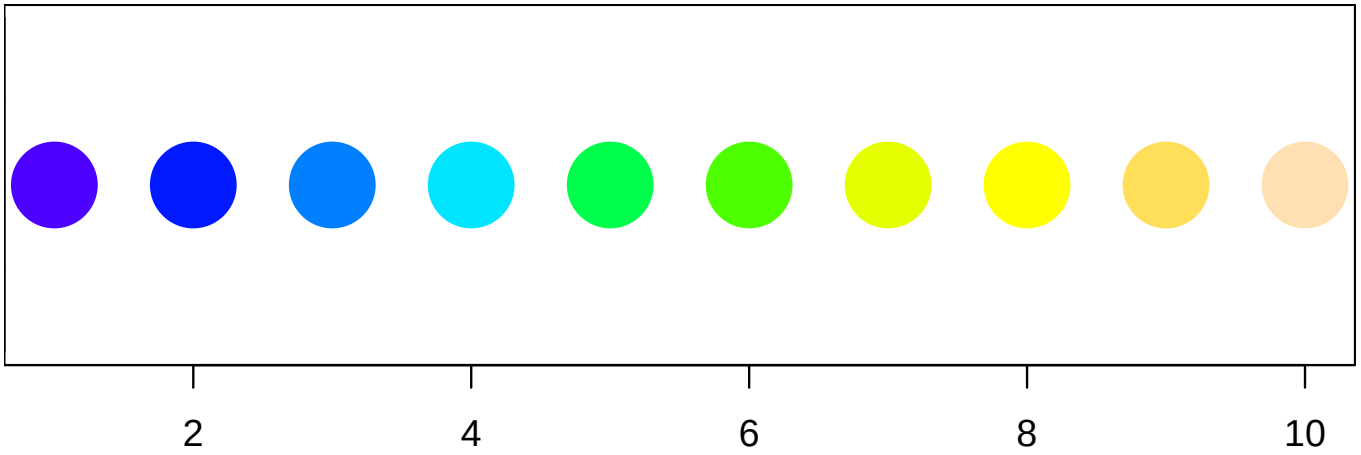


- Pour connaître toutes les couleurs disponibles par défaut on peut taper `colors()` et les voir à partir de `http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf`.

Création de palettes de couleur

Il est facile de créer des palettes de couleurs en définissant une nombre donné de couleurs dans une palette prédéfinie (par ex. avec `heat.colors()`, `terrain.colors()`, `topo.colors()`.

```
par(mar = c(2, 0, 0, 0))
coul <- topo.colors(n = 10)
plot(rep(1,10), col = coul[1:10], pch = 19, cex = 5)
```



Gestion de la transparence des couleurs (1)

- la fonction `col2rgb()` permet de récupérer le codage RGB d'une couleur :

```
(redinrgb <- col2rgb("red"))
```

```
      [,1]
red    255
green    0
blue     0
```

- la fonction `rgb()` permet de créer une couleur avec la transparence définie par l'argument `alpha` :

```
redT <- rgb(redinrgb[1], redinrgb[2],
            redinrgb[3],
            maxColorValue = 255, alpha = 100)
```

Gestion de la transparence des couleurs plus simple mais nécessitant le package ggplot 2 (1 bis)

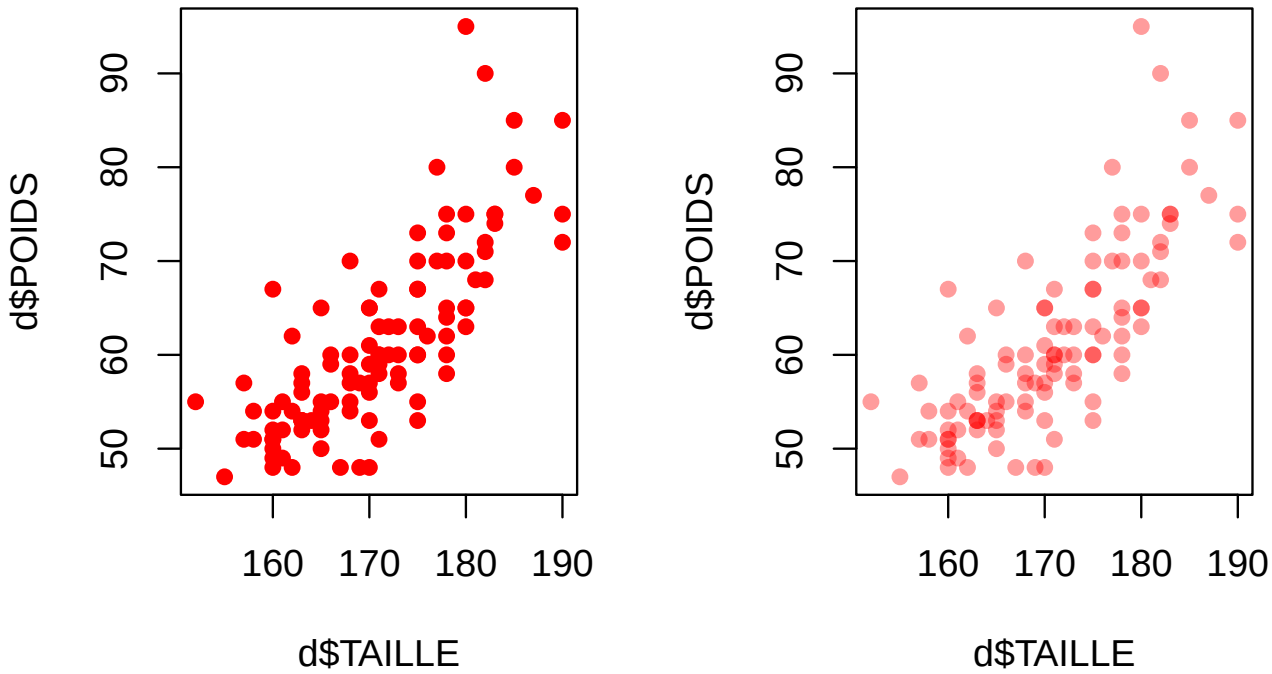
Même chose en une seule étape :

```
require(ggplot2)
redT.bis <- alpha("red", alpha = 0.4)
```

Gestion de la transparence des couleurs (2)

Exemple d'utilisation de la couleur avec transparence

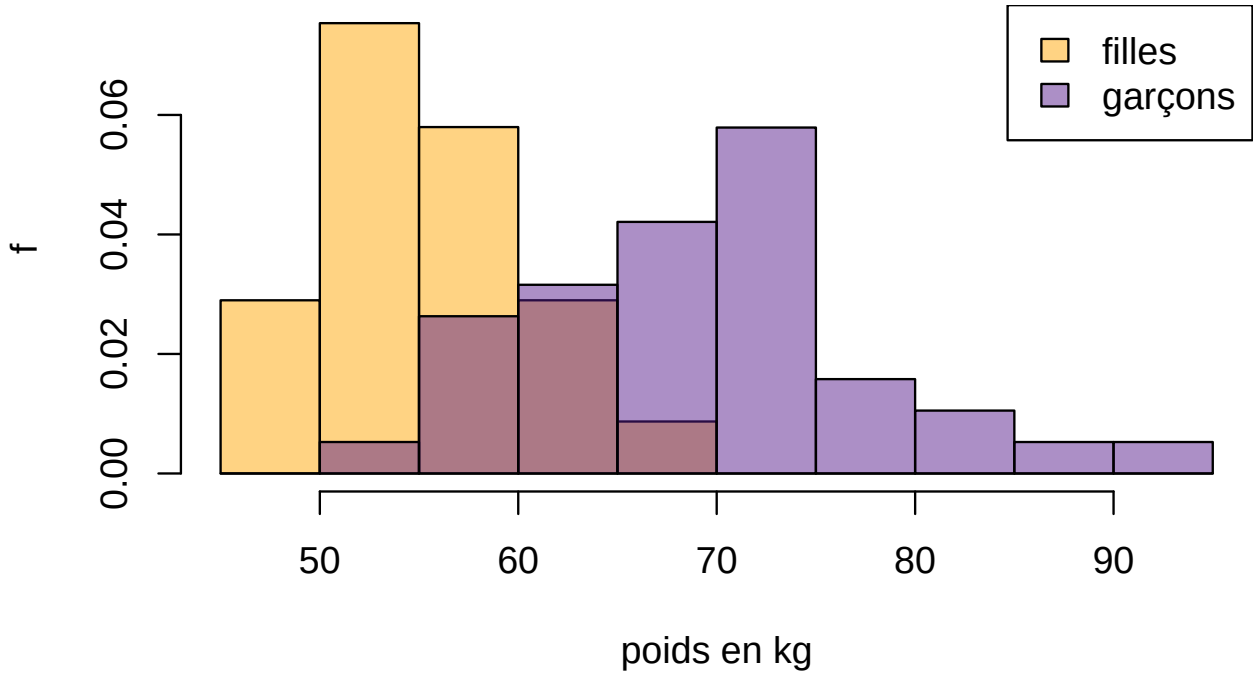
```
par(mfrow = c(1,2))
plot(d$TAILLE, d$POIDS, col = "red", pch = 16)
plot(d$TAILLE, d$POIDS, col = redT, pch = 16)
```



Exemple de figure utilisant la transparence

Consigne

En vous aidant de ce que l'on a vu, réalisez la figure suivante, en utilisant en particulier la transparence.



Exportation d'un graphique

Il existe une multitude de formats d'échange. Voici deux façons de procéder pour exporter une fenêtre graphique.

1. On ouvre une fenêtre graphique, on trace la figure, puis on l'exporte à l'aide de la fonction dev.copy() suivi de dev.off().

```
plot(.....)
dev.copy(device = pdf, file = "joligraphe.pdf")
dev.off()
```

2. On ouvre directement le fichier dans lequel on veut exporter la figure, à l'aide de l'une des fonctions `jpeg()`, `pdf()`, ..., on trace la figure et on ferme le fichier avec la fonction `dev.off()`.

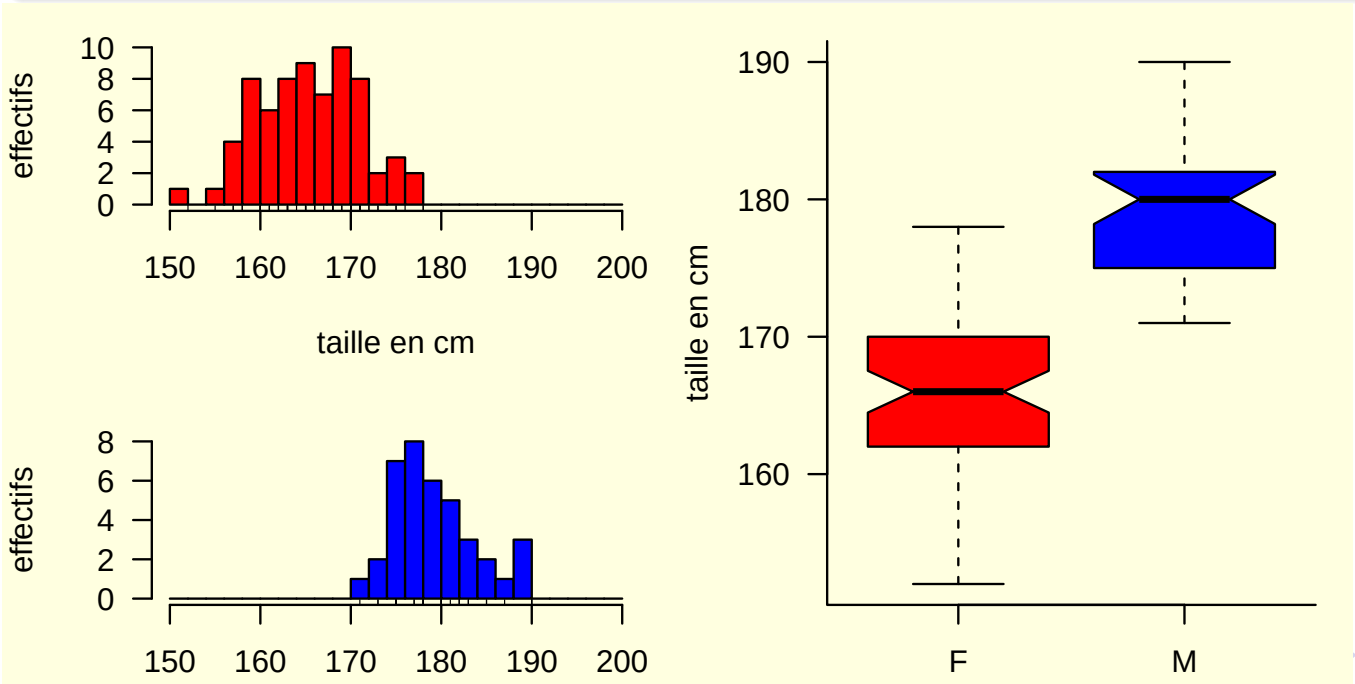
```
jpeg("toto.jpg", quality = 100, width = 15, height = 15,
  units = "cm", pointsize = 12, res = 300)
plot(.....)
dev.off()
```

Permet de gérer la taille, la résolution et la qualité de compression.

Réalisation et export d'une figure

Consigne

Réalisez la figure suivante (marges, couleur d'arrière plan, type de boîte de tracé, ...), puis exportez-la et insérez-la en bonne définition en jpeg dans un texte. Refaites la même chose en divisant par deux la hauteur et la longueur de la figure pour voir ce que cela donne.



Package `ggplot2`

Comment le package `ggplot2` peut vous simplifier la vie dans bien des cas classiques.

Introduction à l'utilisation du package `ggplot2`

- Le package `ggplot2` facilite grandement la visualisation par des couleurs, types de lignes ou de points, . . ., d'autres variables que celles représentées en x et y.
- Son utilisation requiert que les données soient correctement codées et toutes dans le même objet de type `data.frame`.
- `ggplot2` est un package qui a été créé en 2007 par Hadley Wickham sur une idée très originale de grammaire des graphes (gg pour "grammar of graphics"), qui permet de définir un graphe comme un objet R. Il est maintenant adopté par un très grand nombre d'utilisateurs de R.

Construction d'un graphe avec `ggplot()`

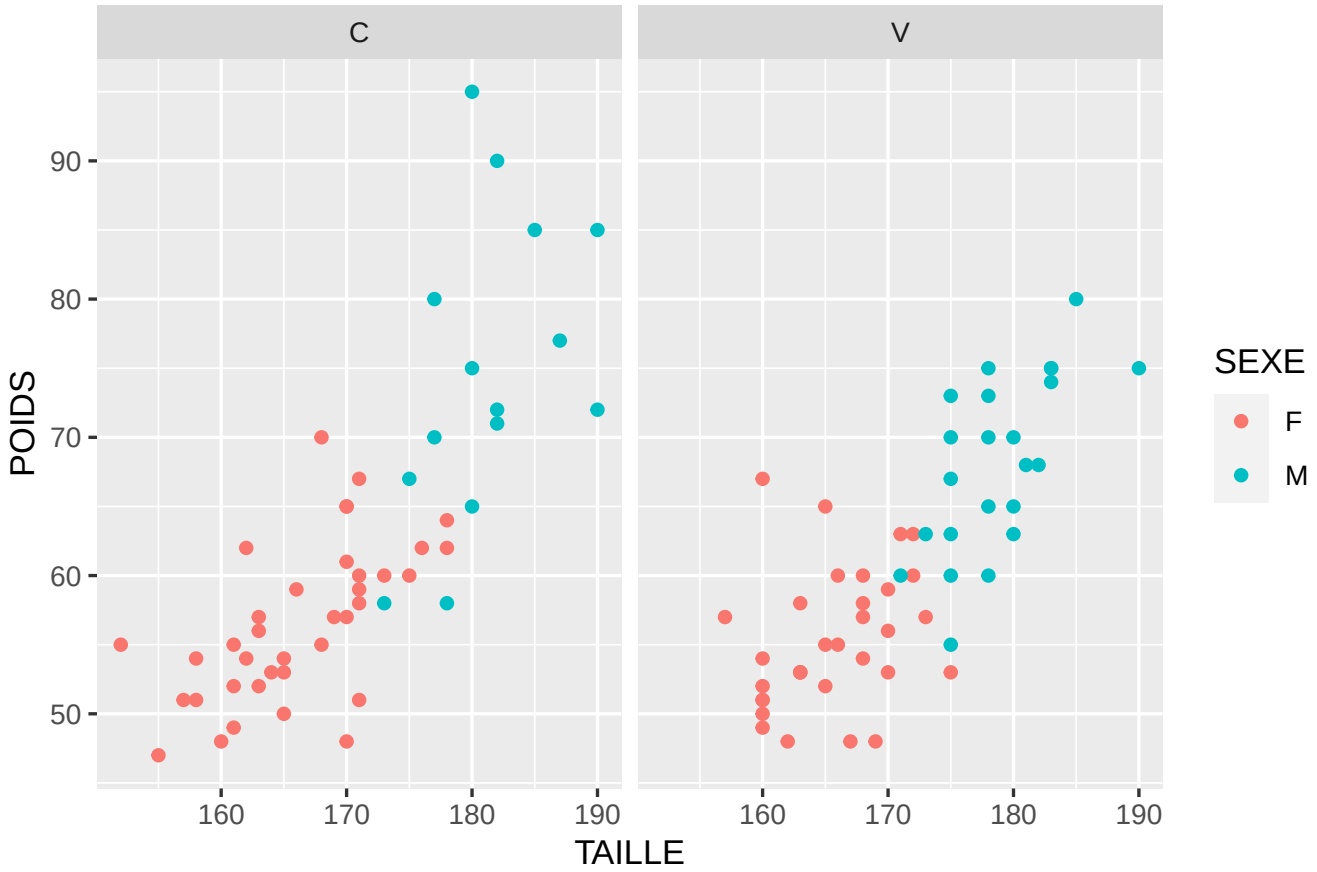
1. Un graphe "ggplot" est un objet **R** défini tout d'abord par un **jeu de données** (argument `data`) et une **esthétique** (argument `mapping` défini avec la fonction `aes()`. Cette esthétique permet de définir x, y, le codage de couleur, de forme, de groupe, ...).

```
g <- ggplot(data = d, mapping = aes(x = TAILLE,
        y = POIDS, colour = SEXE))
```

2. Le graphe est ensuite complété avec une ou plusieurs **géométries** (fonctions geom_point(), geom_line(), geom_boxplot(), ...) et un éventuel **découpage par groupe** (fonctions facet_wrap() en 1D ou facet_grid() en 2D).

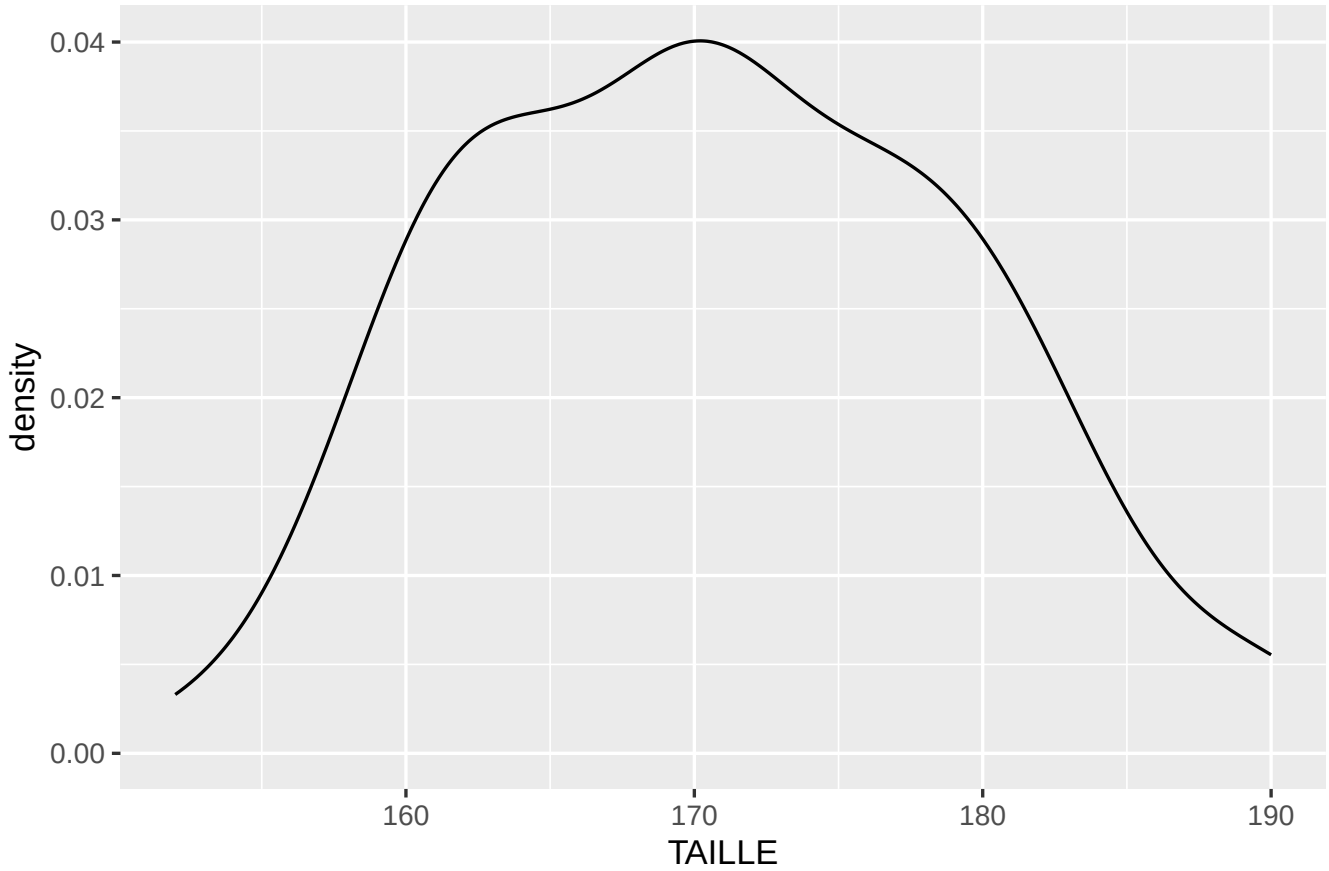
```
g + geom_point() + facet_wrap(~ CADRE)
```

Graphe obtenu



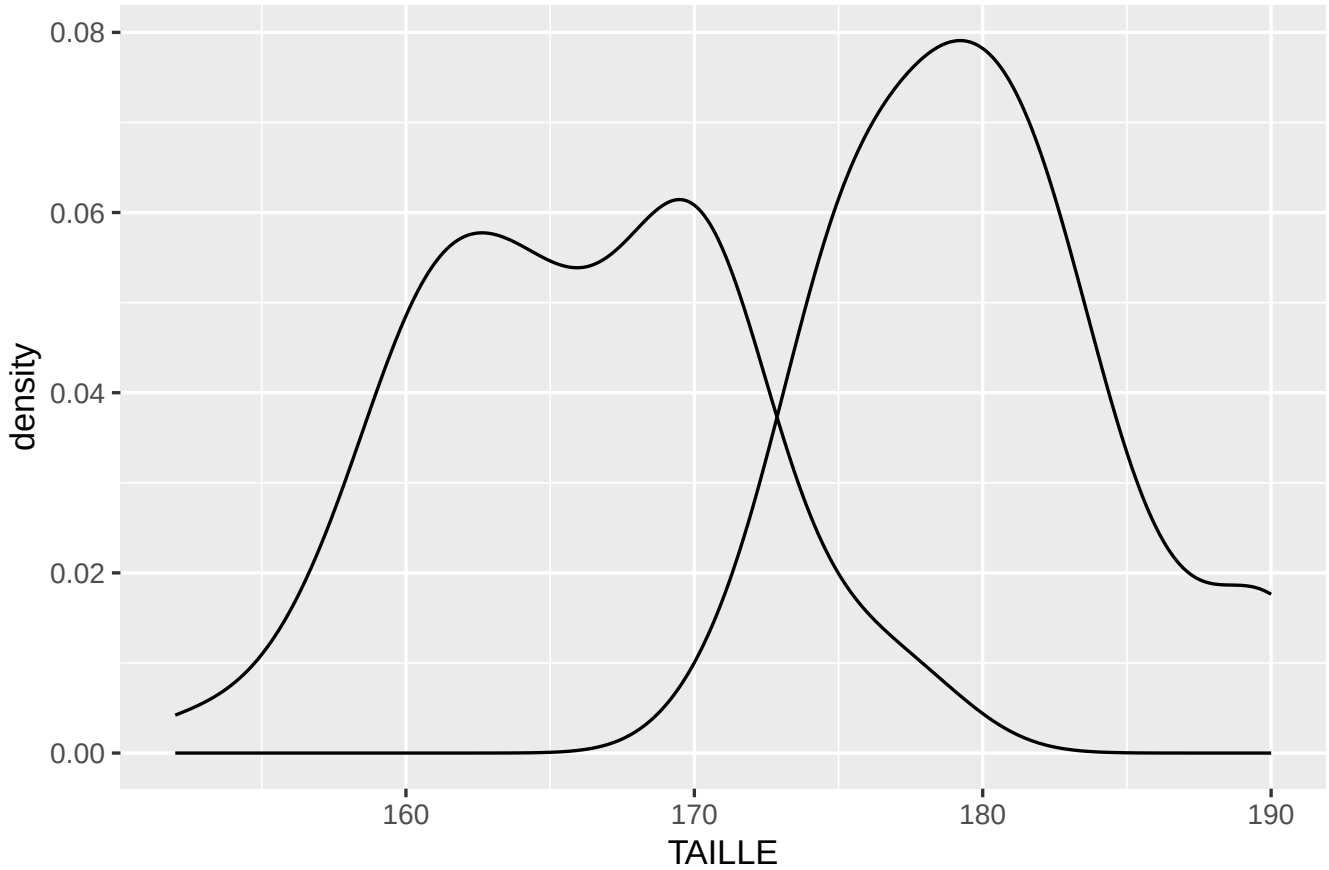
Exemple de base avec x **quantitatif** et pas de y

```
require(ggplot2)
ggplot(data = d, mapping = aes(x = TAILLE)) + geom_density()
```



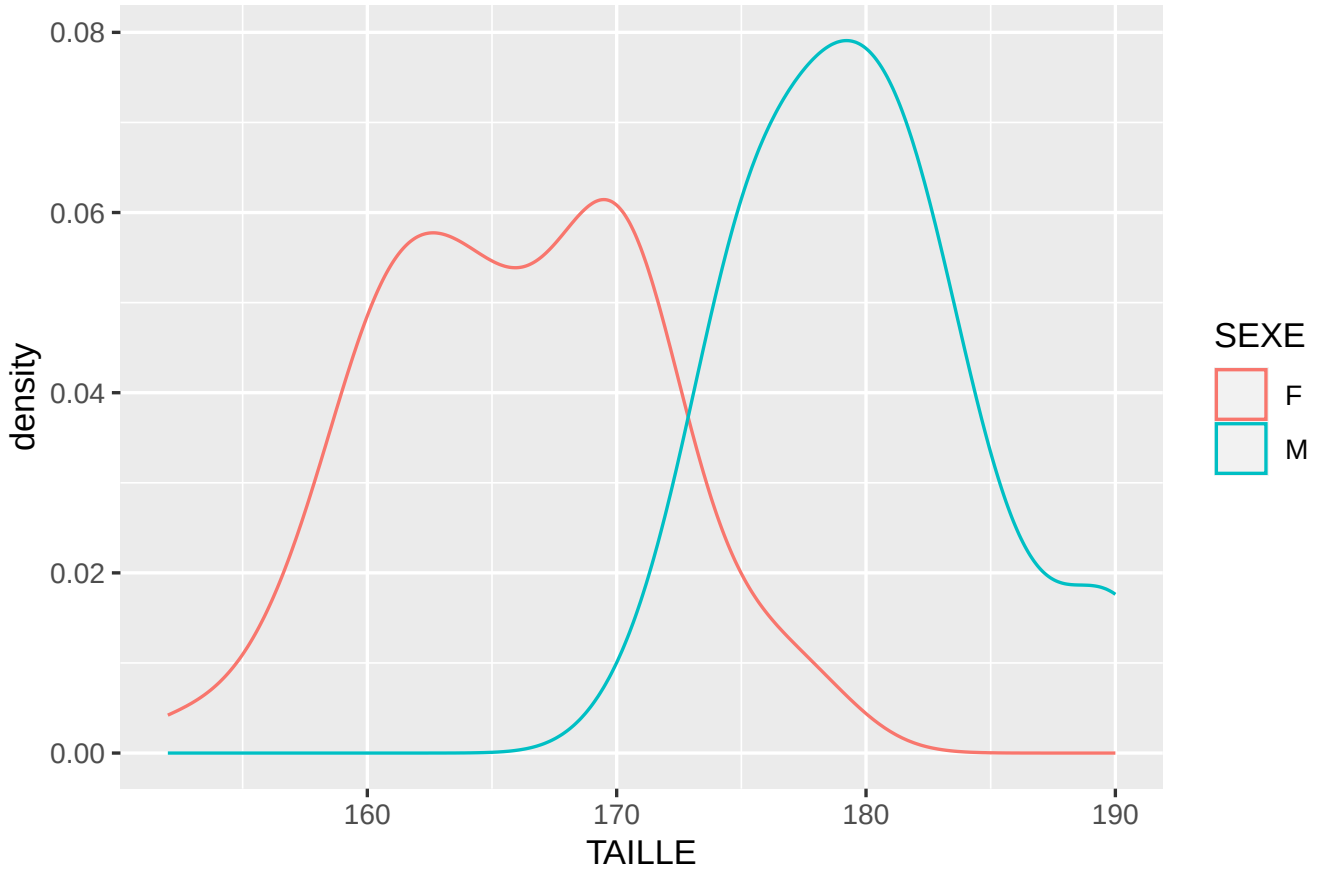
En dissociant les deux sexes

```
ggplot(d, aes(x = TAILLE, group = SEXE)) + geom_density()
```



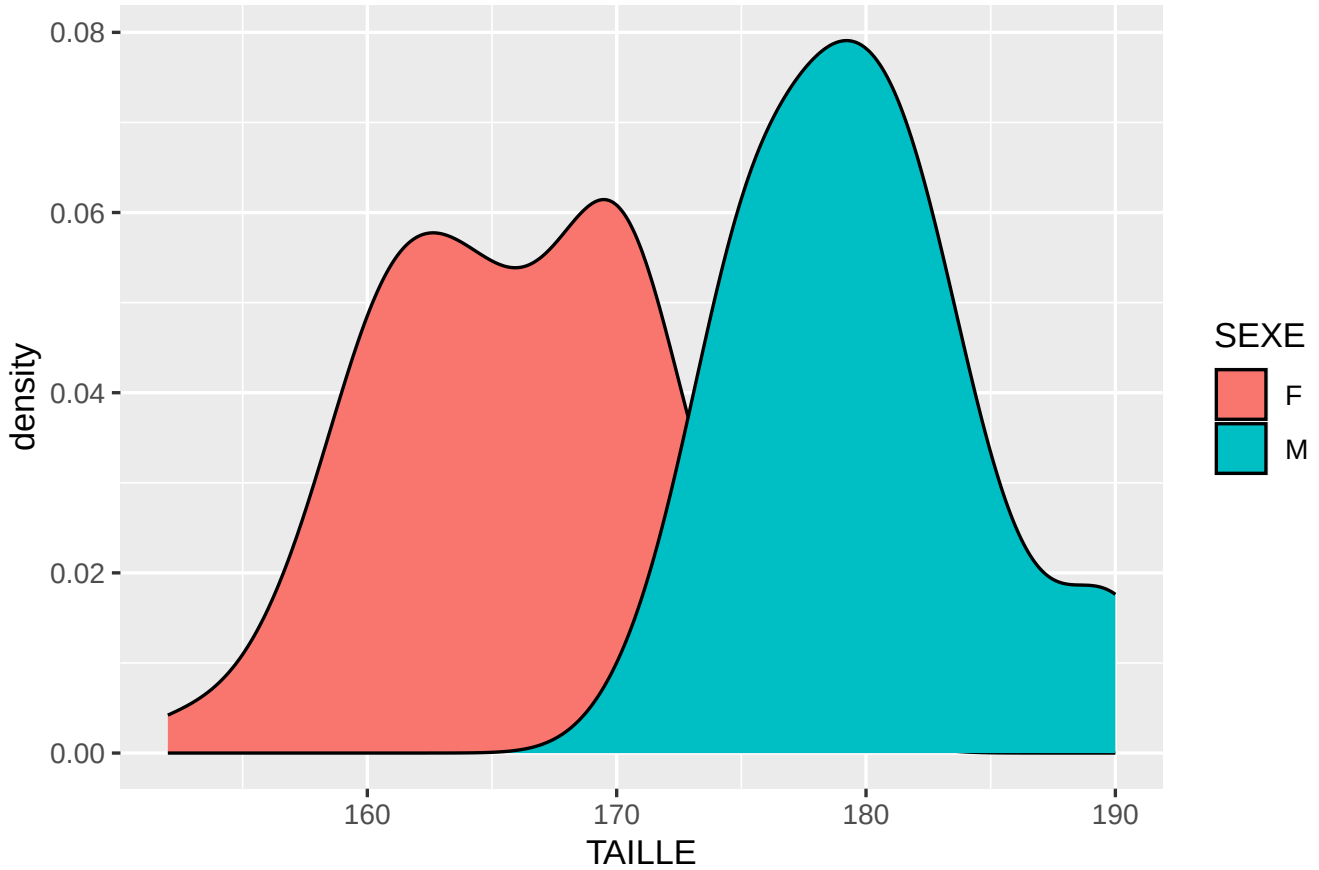
En associant une couleur par sexe

```
ggplot(d, aes(x = TAILLE, colour = SEXE)) + geom_density()
```



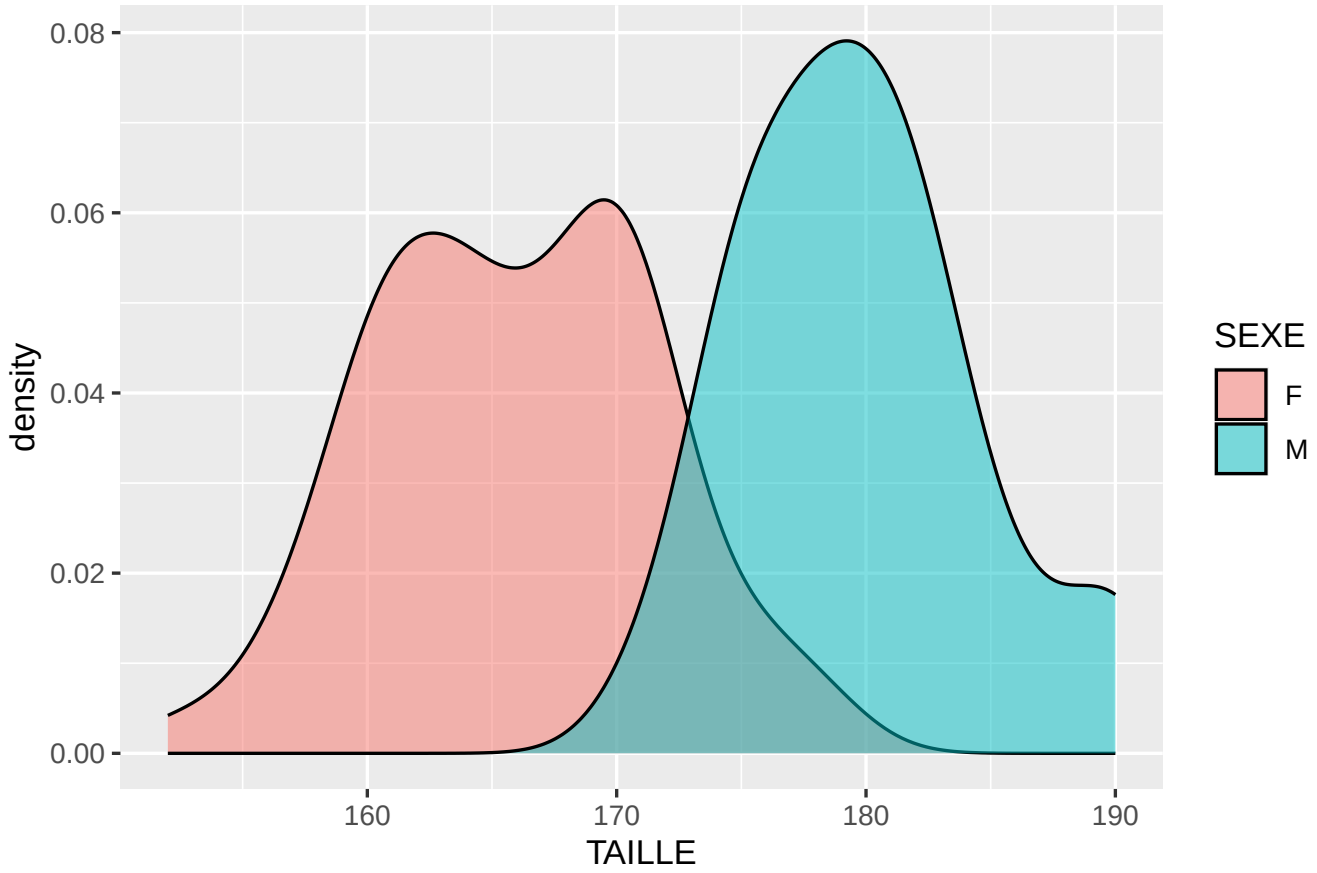
En coloriant sous la courbe de densité

```
ggplot(d, aes(x = TAILLE, fill = SEXE)) + geom_density()
```



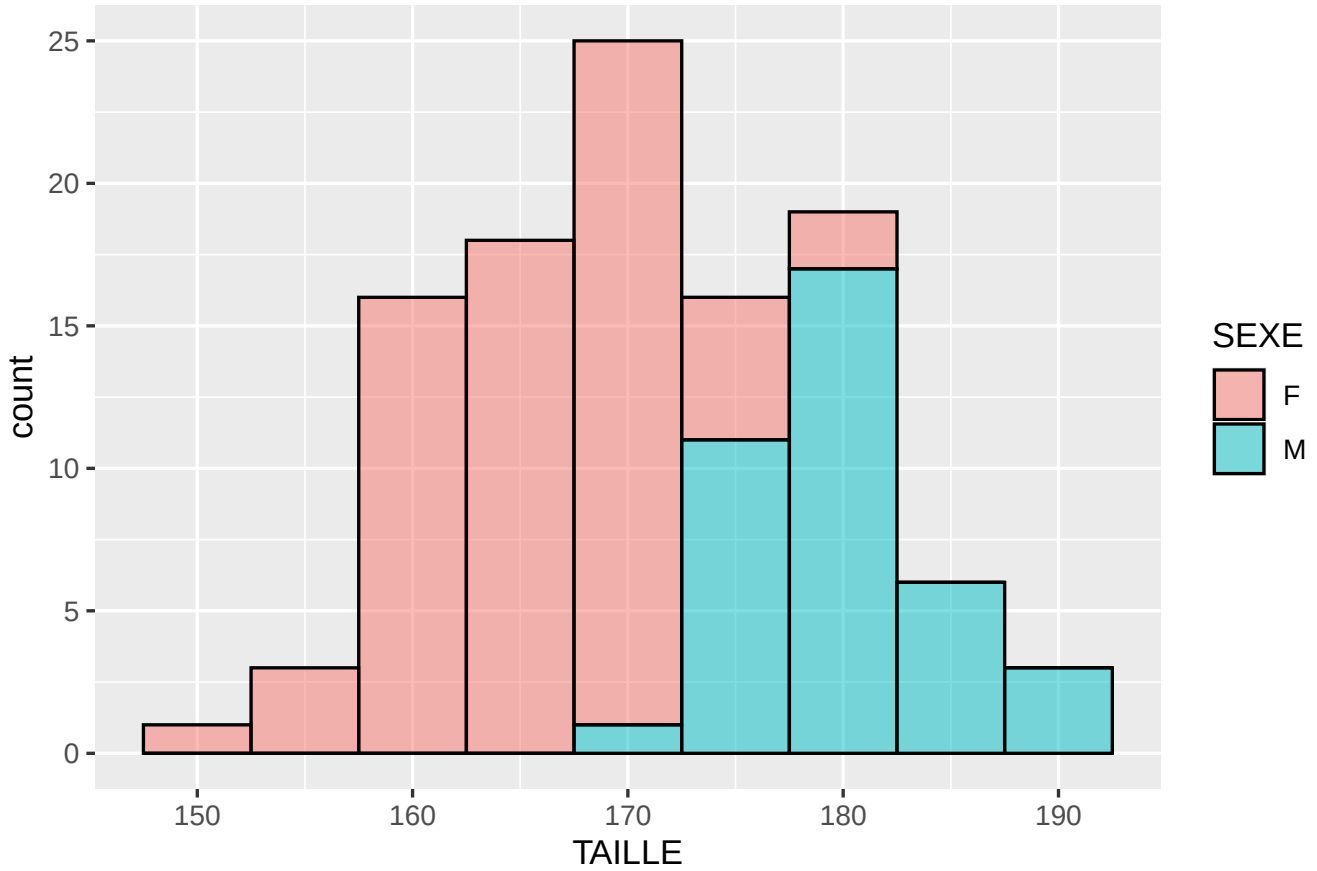
En ajoutant un degré de transparence

```
ggplot(d, aes(x = TAILLE, fill = SEXE)) +
  geom_density(alpha = 0.5)
```



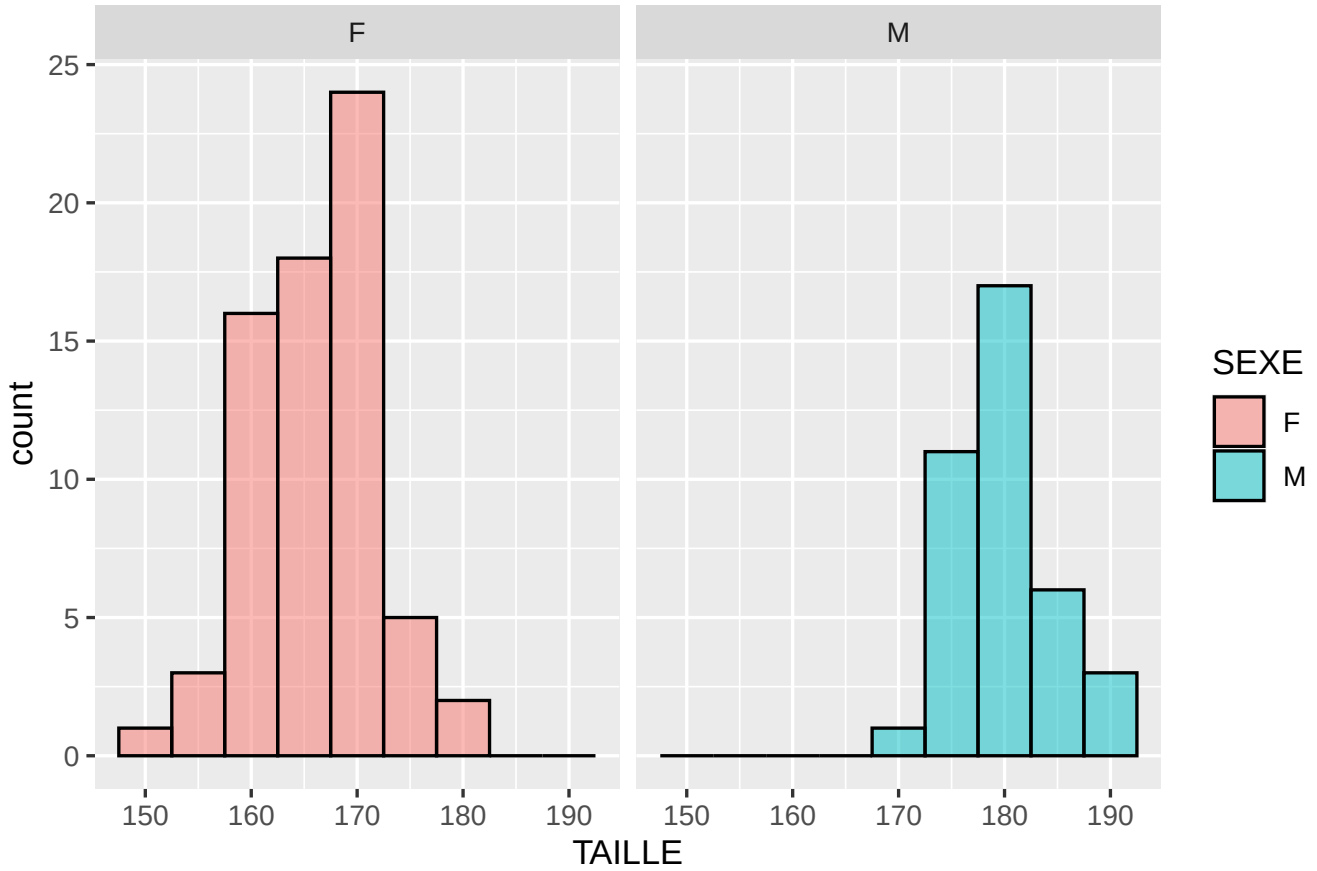
Autres géométries pour *x* quantitatif : histogramme

```
ggplot(d, aes(x = TAILLE, fill = SEXE)) +
  geom_histogram(binwidth = 5, alpha = 0.5, col = "black")
```



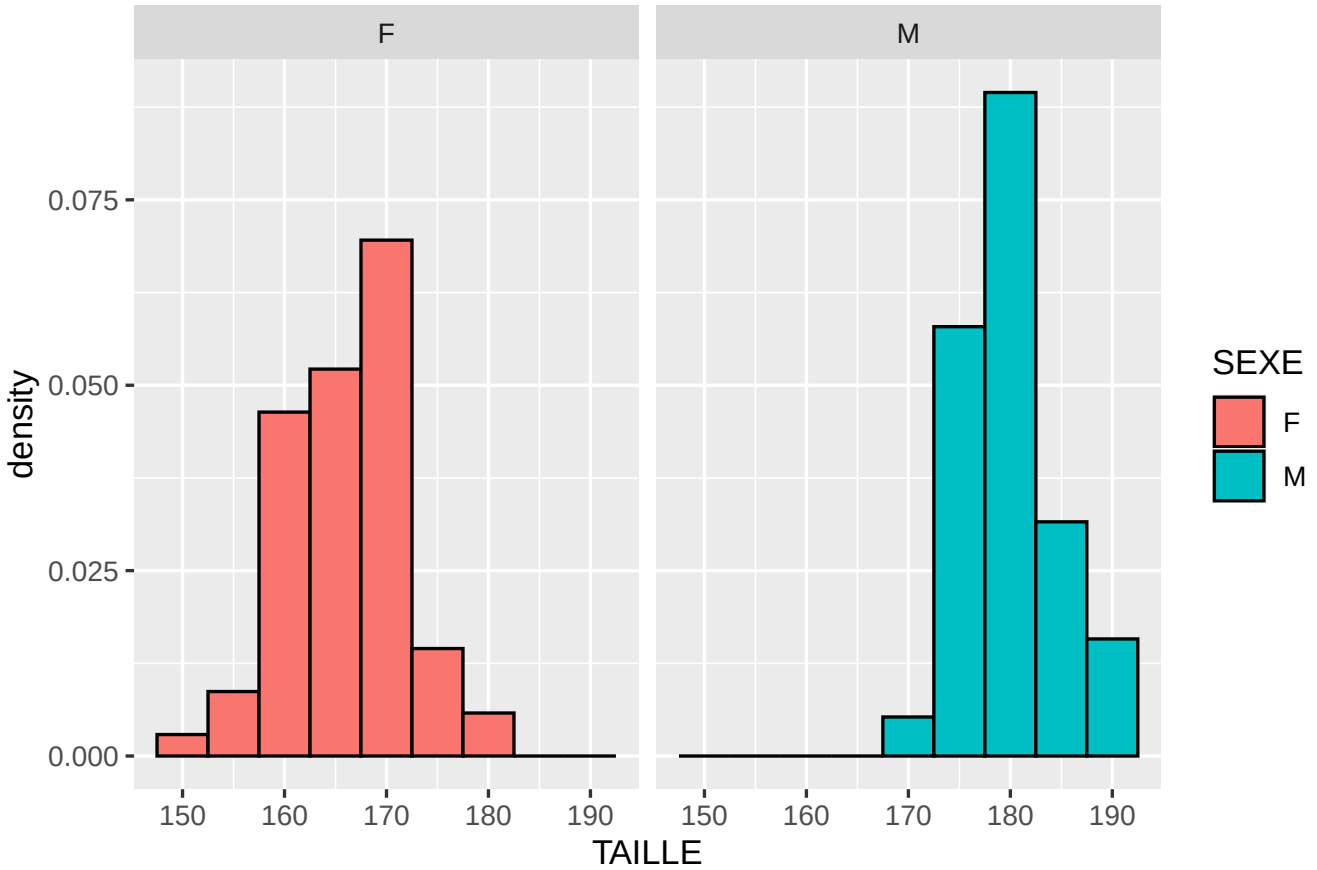
Autres géométries pour *x* quantitatif : histogramme

```
ggplot(d, aes(x = TAILLE, fill = SEXE)) + facet_wrap(~ SEXE)
  geom_histogram(binwidth = 5, alpha = 0.5, colour = "black")
```



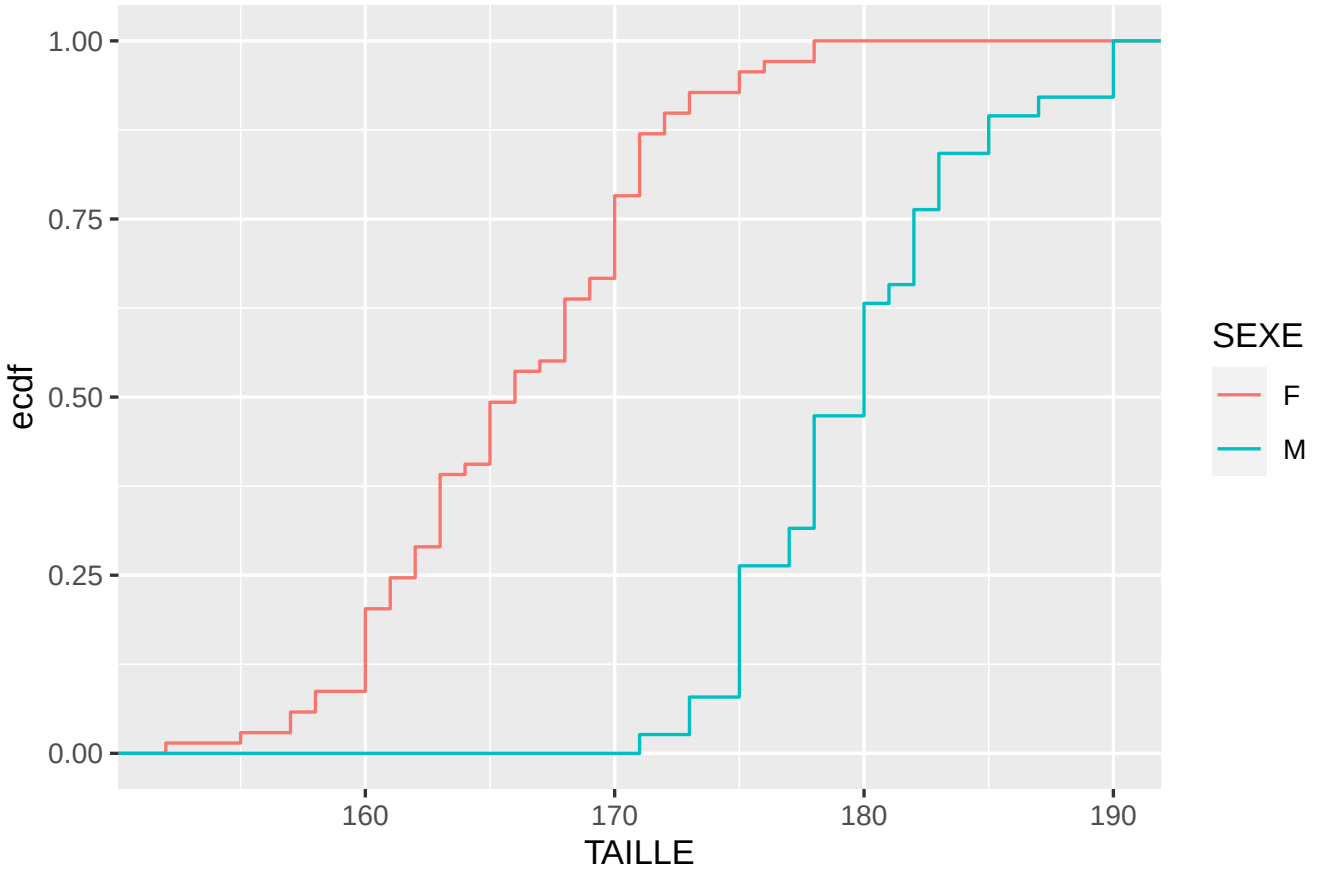
Autres géométries pour *x* quantitatif : histogramme

```
ggplot(d, aes(x = TAILLE, y = stat(density), fill = SEXE)) +
  facet_wrap(~ SEXE) +
  geom_histogram(binwidth = 5, colour = "black")
```



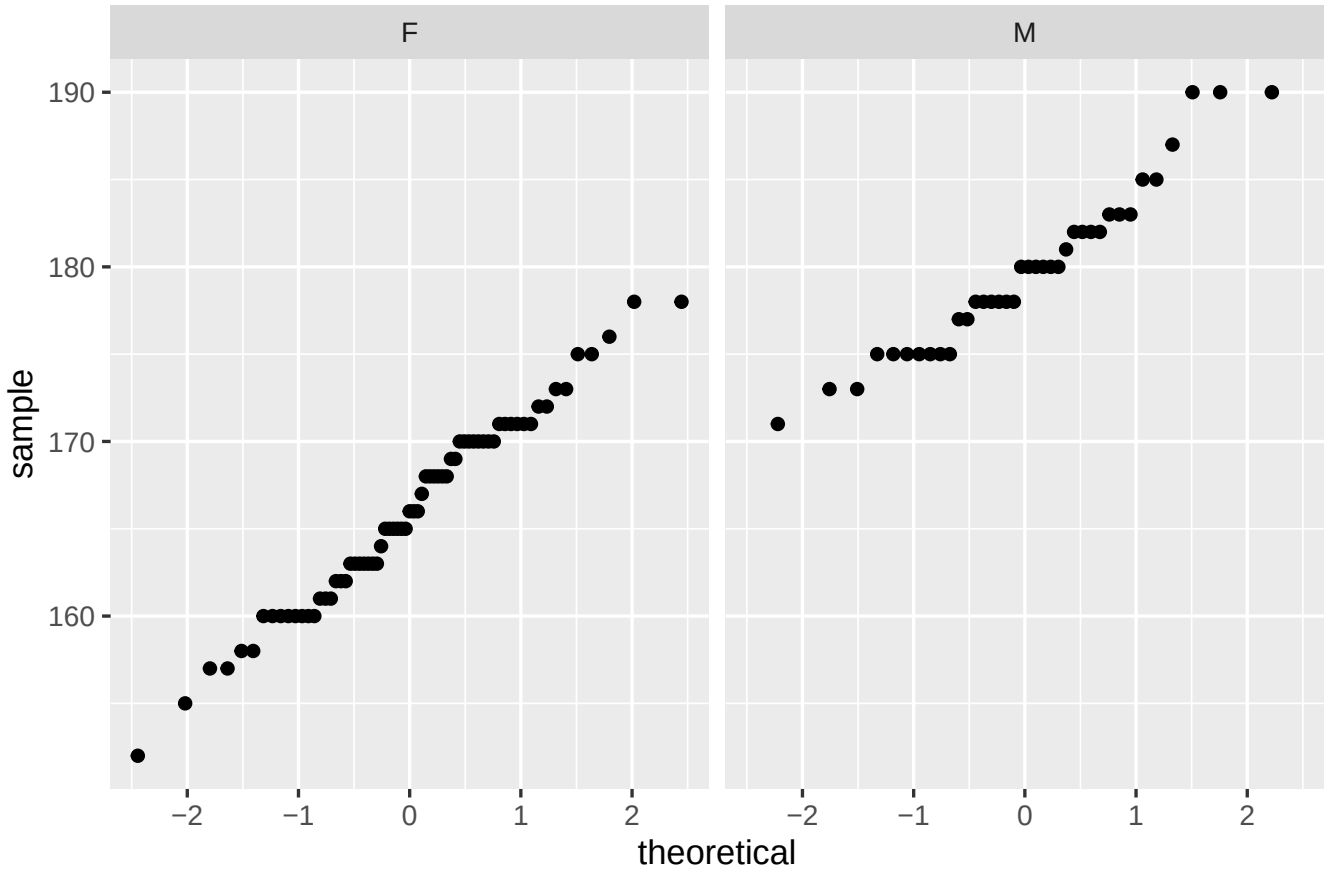
Autres géométries pour *x* quantitatif : ECDF plot

```
ggplot(d, aes(x = TAILLE, colour = SEXE)) +
  stat_ecdf(geom = "step")
```



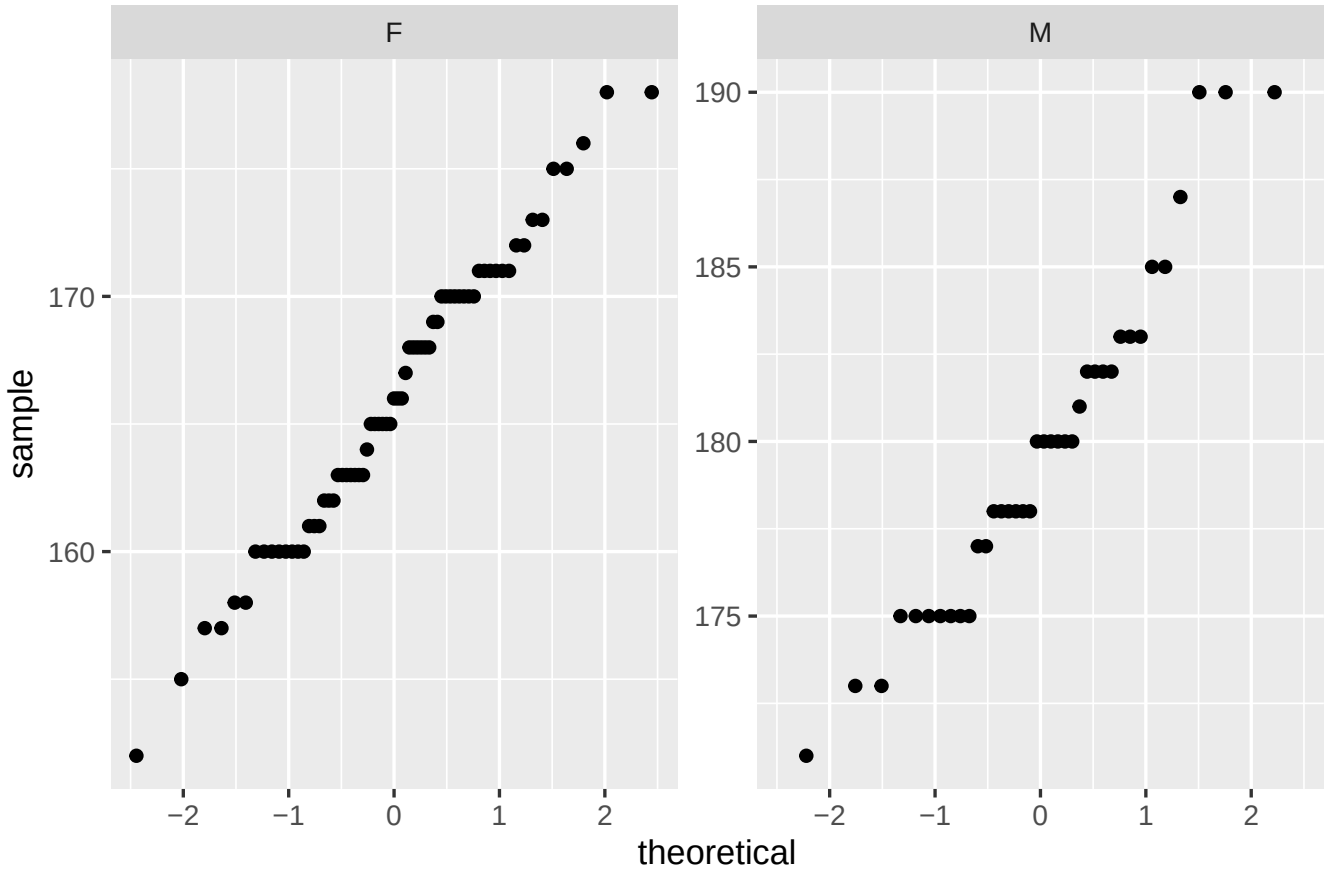
Autres géométries pour *x* quantitatif : QQplot

```
ggplot(d, aes(sample = TAILLE)) +
  facet_wrap(~ SEXE) + geom_qq()
```



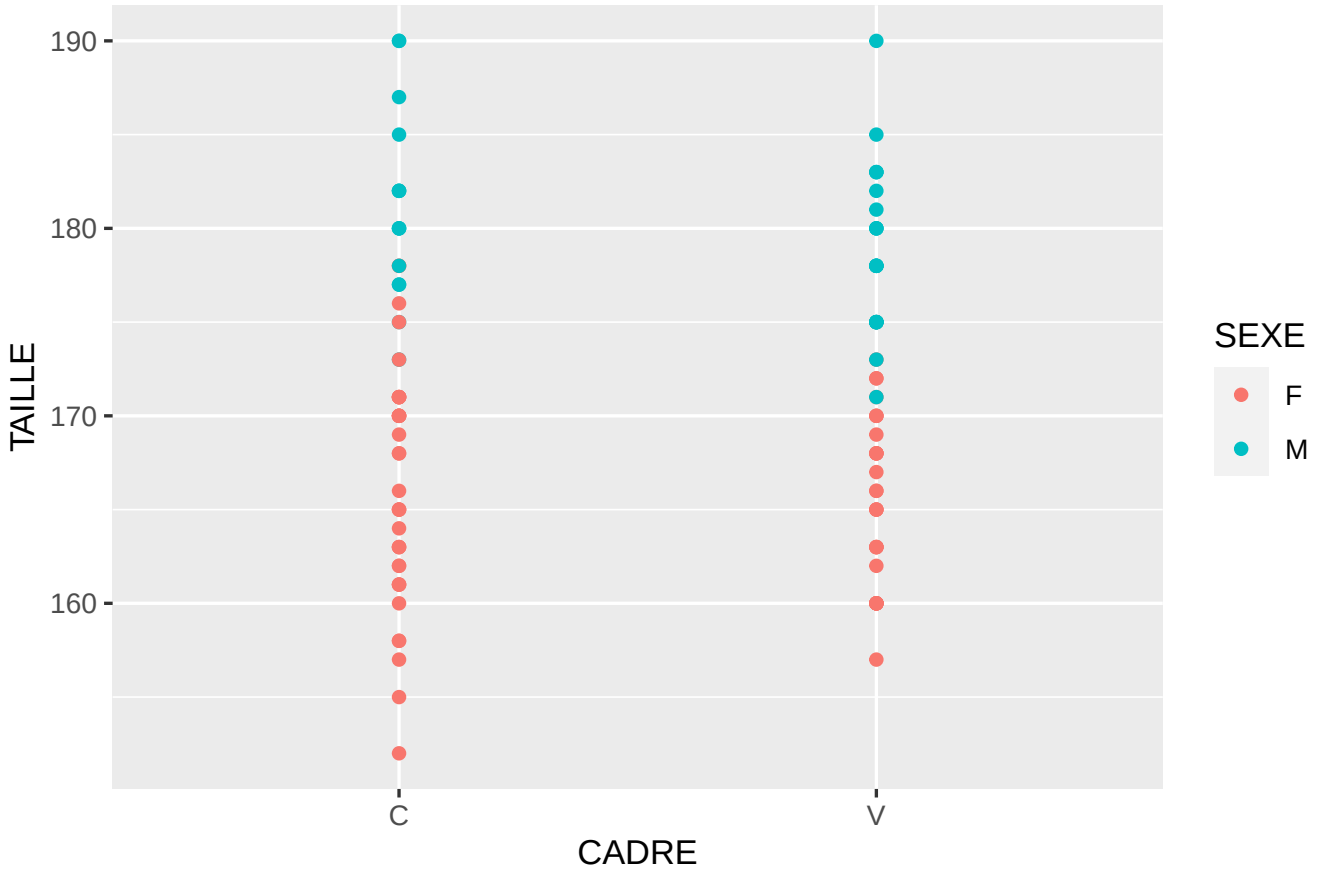
Autres géométries pour *x* quantitatif : QQplot

```
ggplot(d, aes(sample = TAILLE)) +
  facet_wrap(~ SEXE, scales = "free_y") + geom_qq()
```



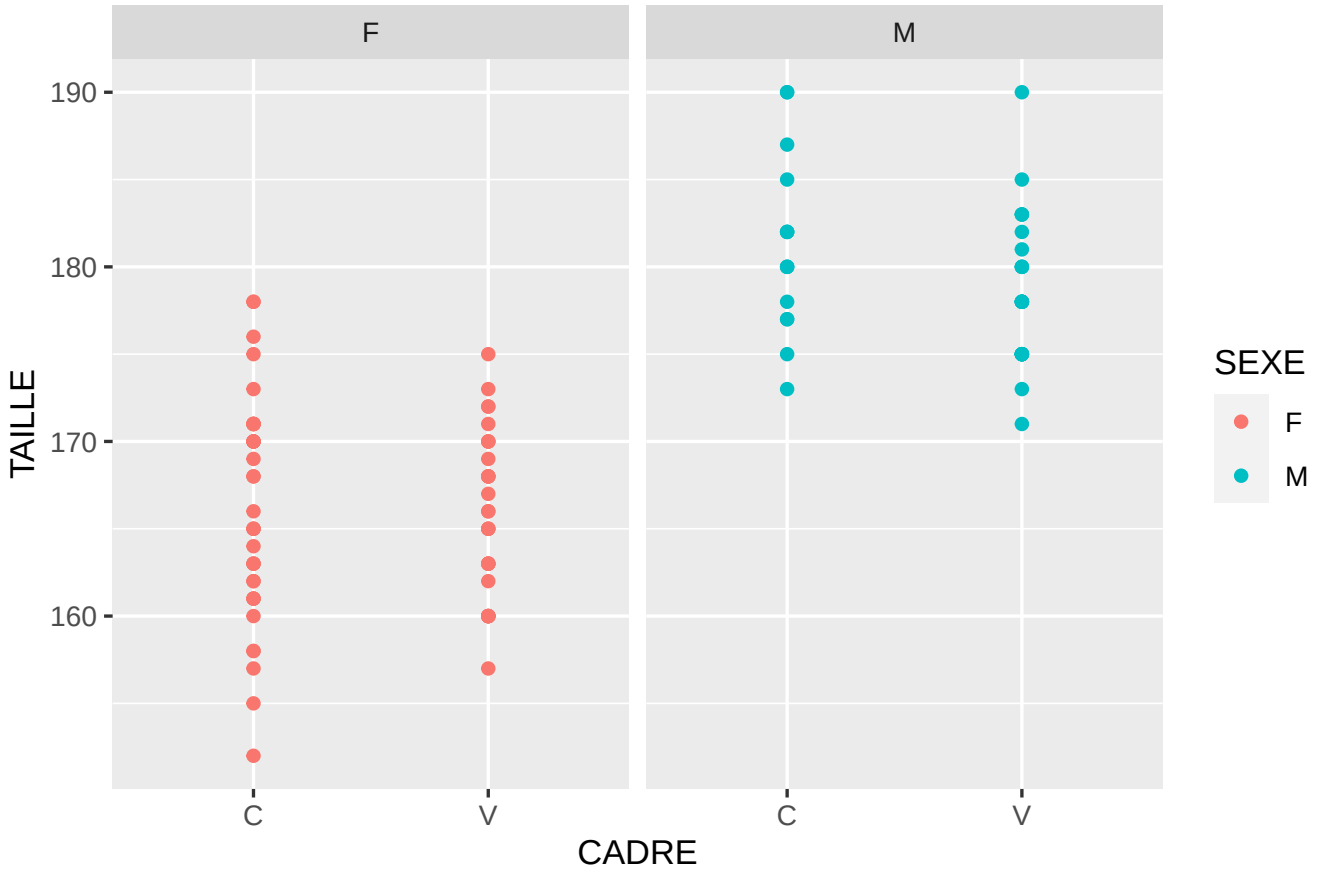
Exemple de base avec *x* **qualitatif et** *y* **quantitatif**

```
ggplot(d, aes(x = CADRE, y = TAILLE, colour = SEXE)) +
  geom_point()
```



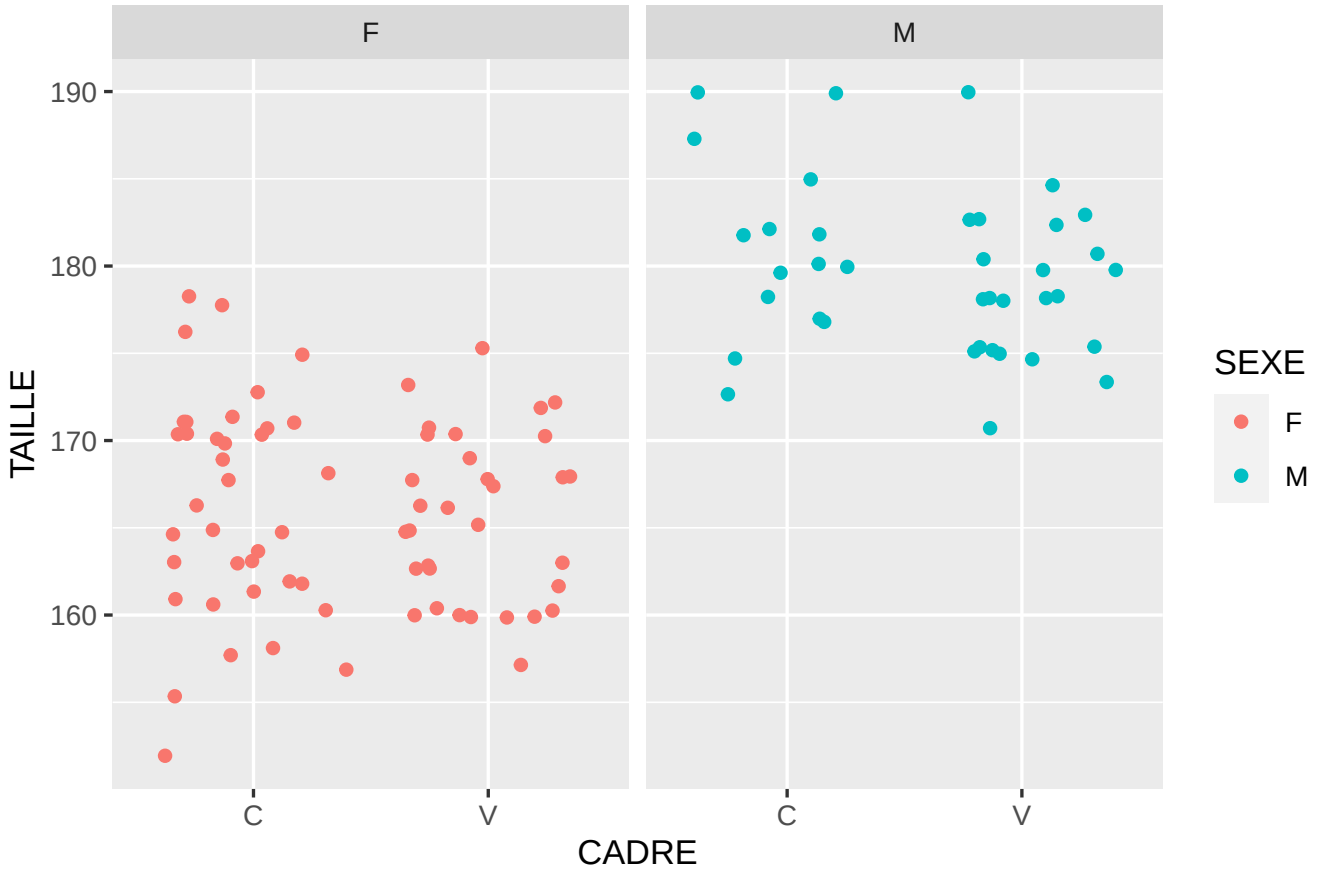
En divisant la figure (une par sexe)

```
ggplot(d, aes(x = CADRE, y = TAILLE, colour = SEXE)) +
  geom_point() + facet_wrap(~ SEXE)
```



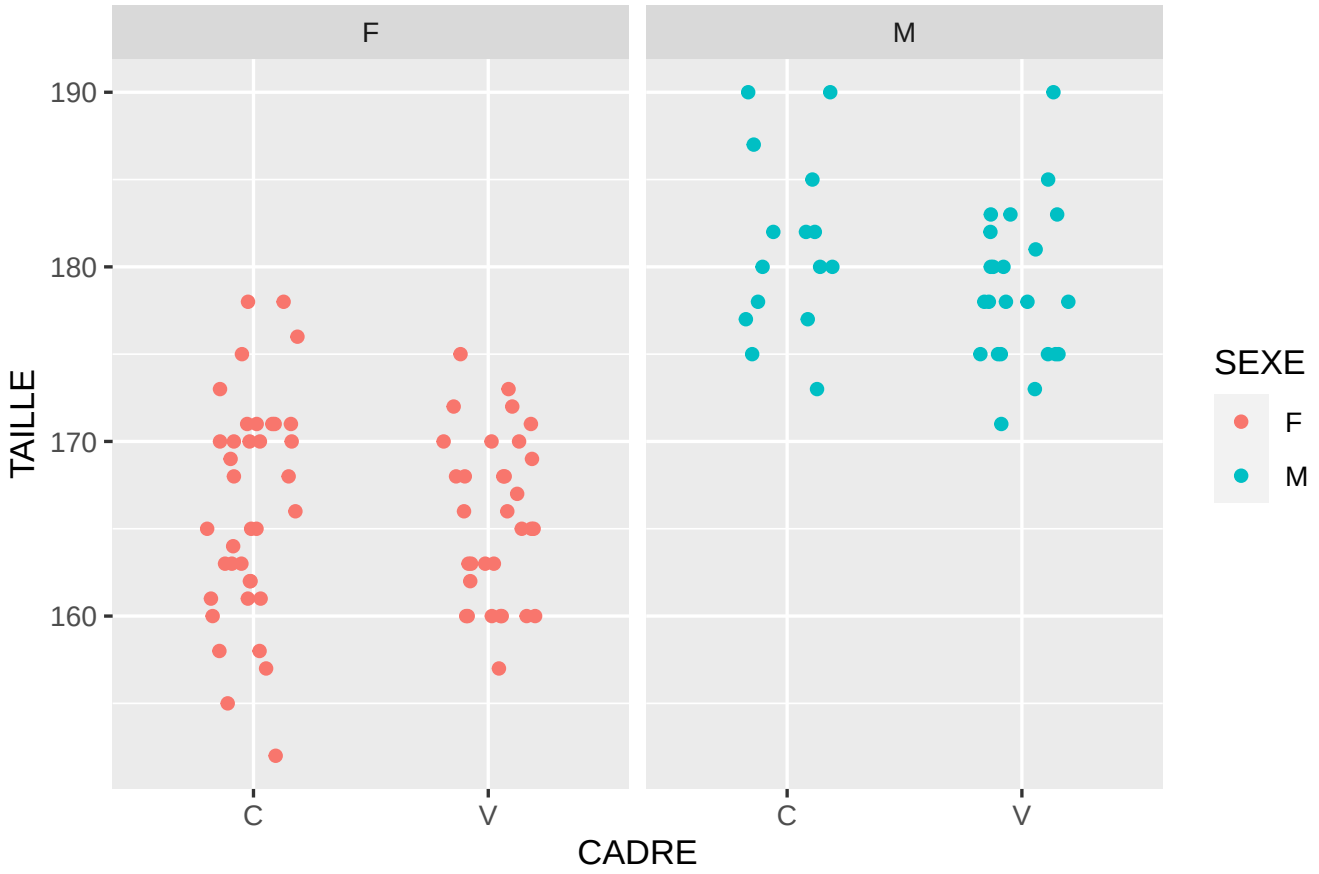
En bruitant les points pour voir les ex aequos

```
ggplot(d, aes(x = CADRE, y = TAILLE, colour = SEXE)) +
  geom_jitter() + facet_wrap(~ SEXE)
```



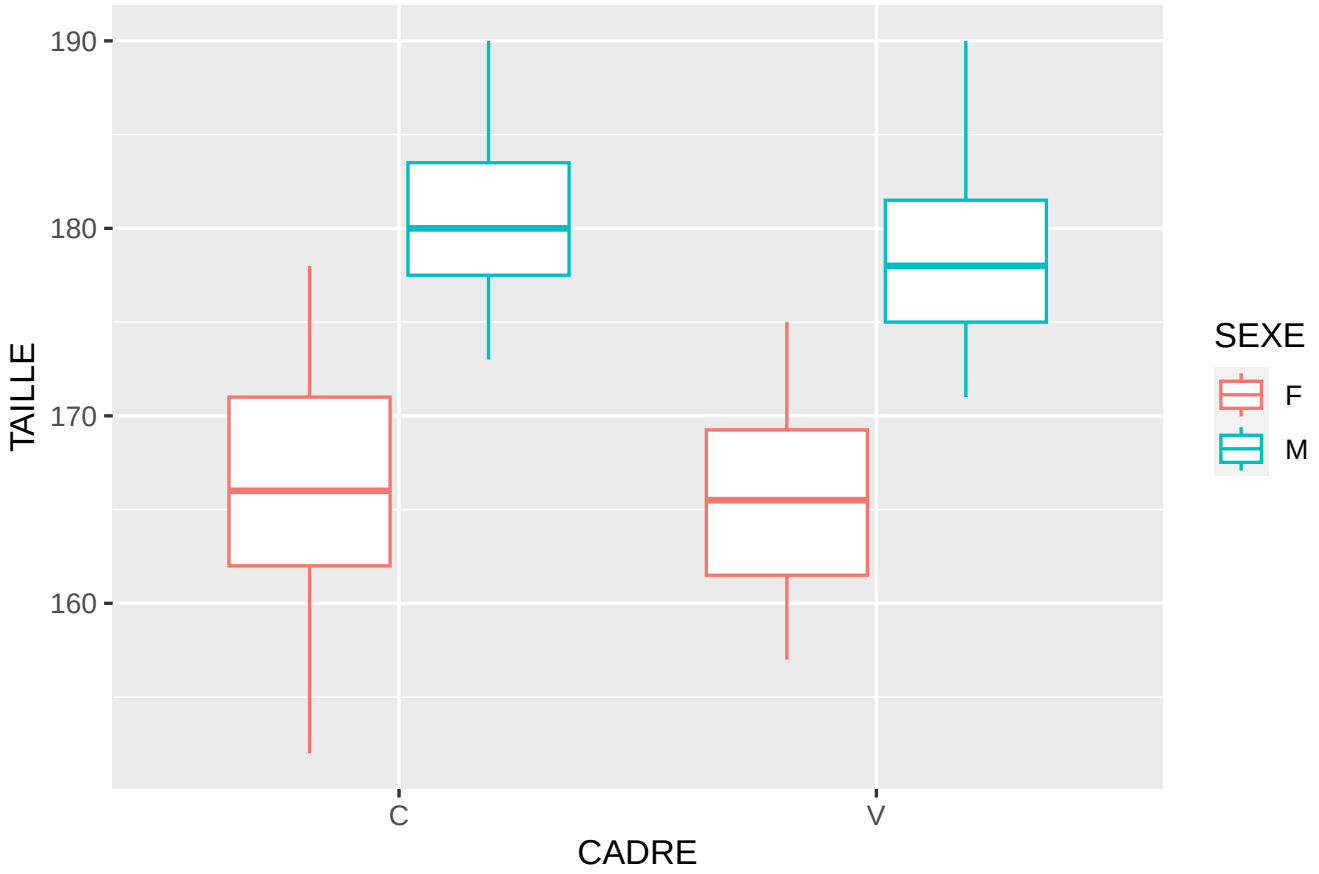
En contrôlant le bruit sur x et sur y

```
ggplot(d, aes(x = CADRE, y = TAILLE, colour = SEXE)) +
  geom_jitter(height = 0, width = 0.2) + facet_wrap(~ SEXE)
```



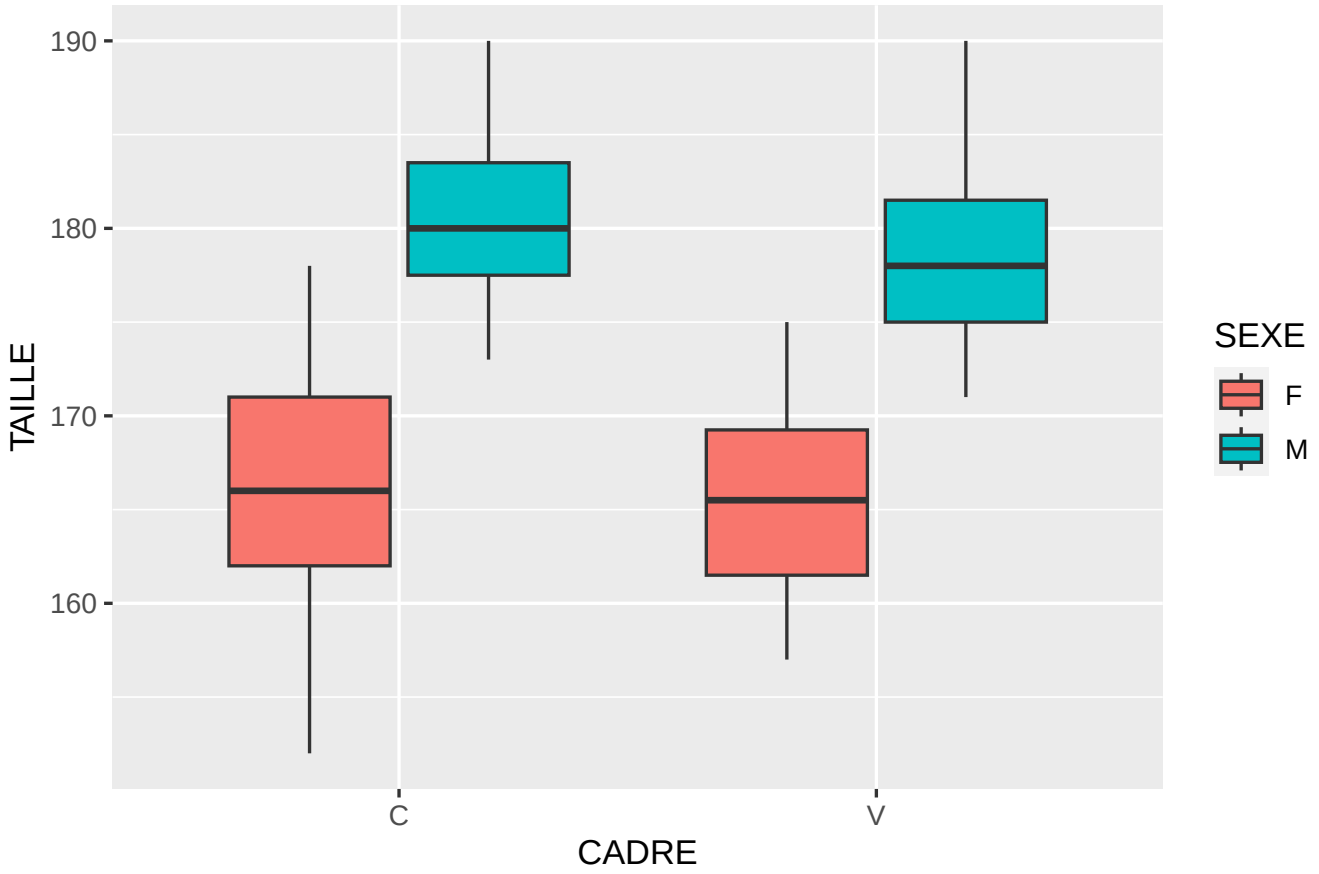
Avec des diagrammes en boîte - version 1

```
ggplot(d, aes(x = CADRE, y = TAILLE, colour = SEXE)) +
  geom_boxplot()
```



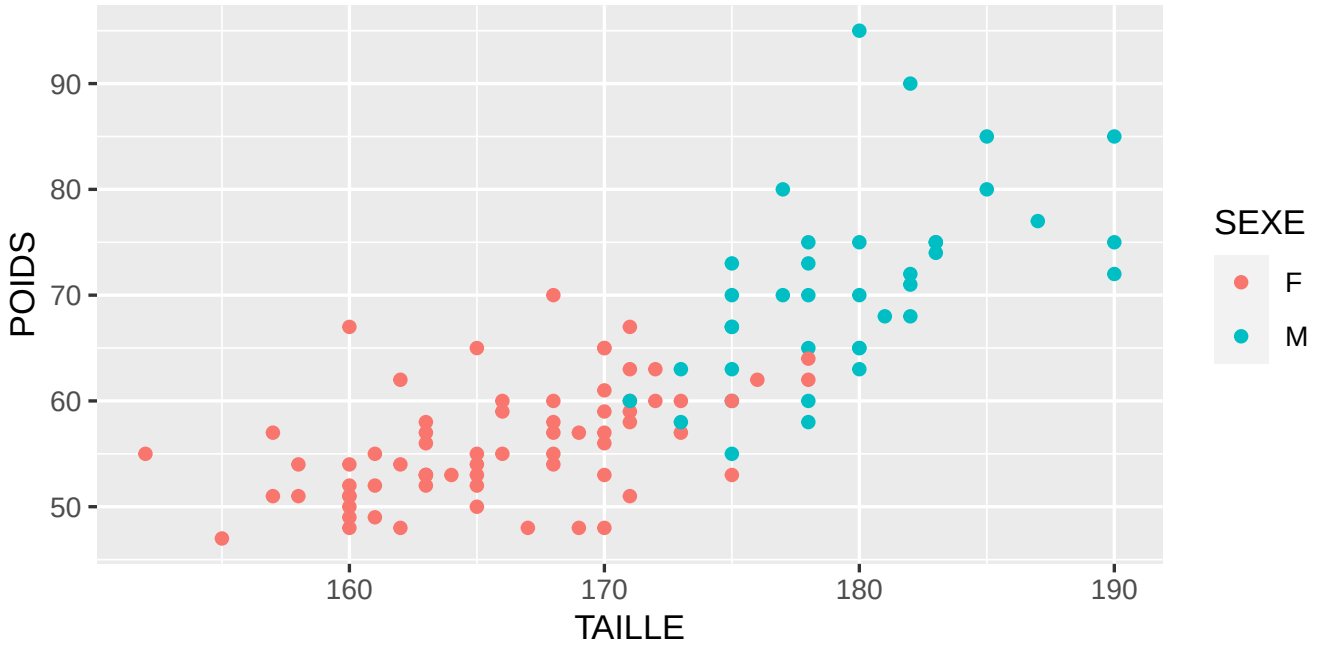
Avec des diagrammes en boîte - version 2

```
ggplot(d, aes(x = CADRE, y = TAILLE, fill = SEXE)) +
  geom_boxplot()
```



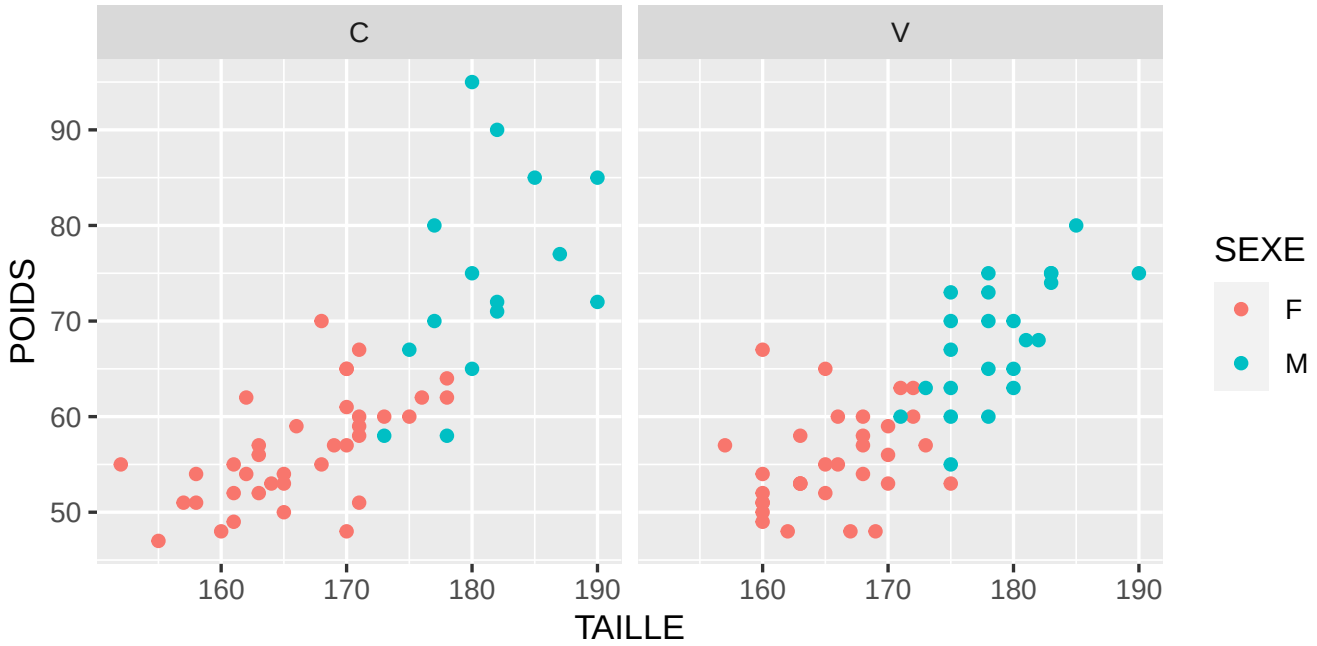
Exemple de base avec *x* **quantitatif et** *y* **quantitatif**

```
ggplot(d, aes(x = TAILLE, y = POIDS, colour = SEXE)) +
  geom_point()
```



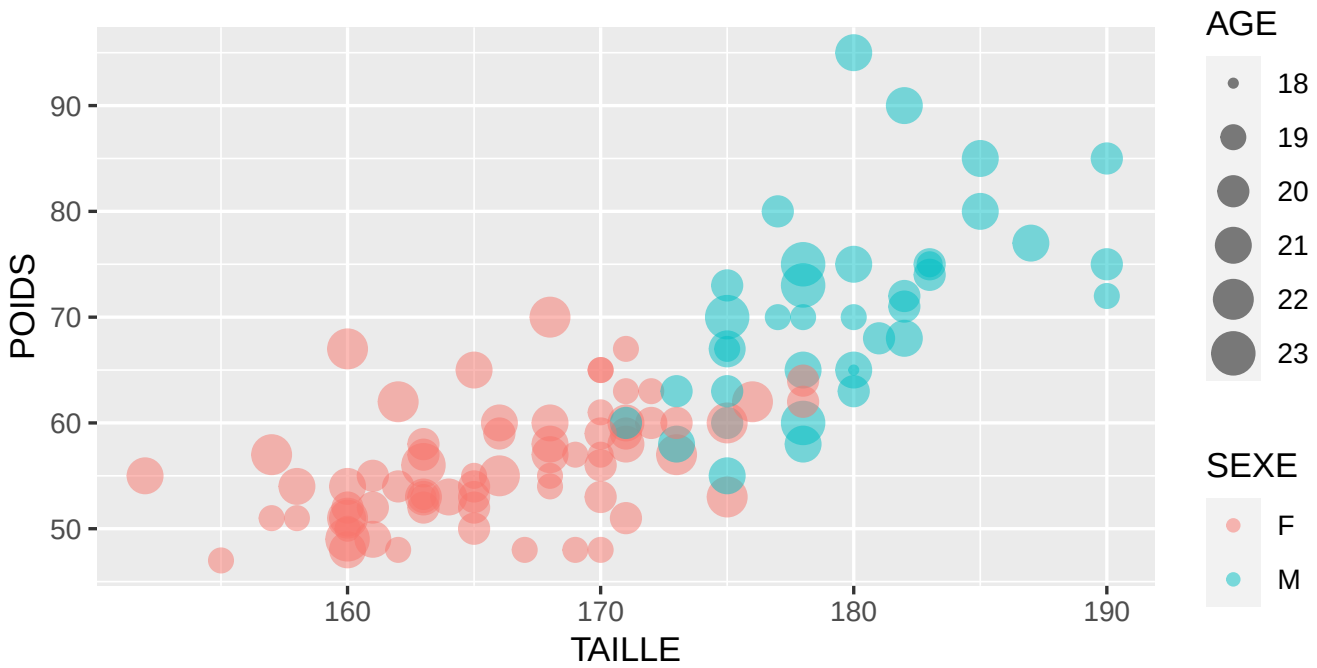
En divisant par le cadre de vie

```
ggplot(d, aes(x = TAILLE, y = POIDS, colour = SEXE)) +
  geom_point() +  facet_wrap(~ CADRE)
```



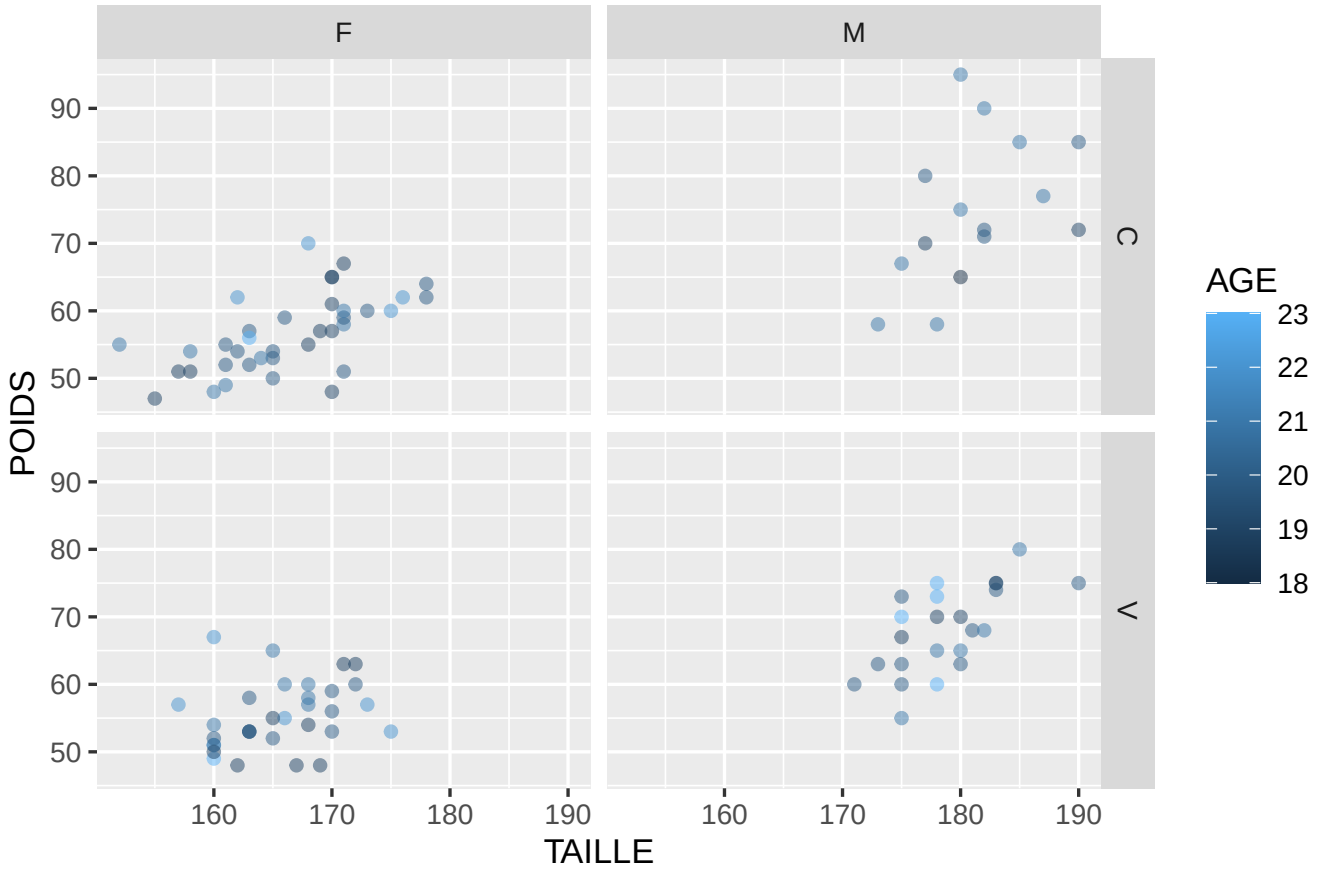
Avec l'âge codant pour la taille des points

```
ggplot(d, aes(x = TAILLE, y = POIDS, colour = SEXE,
              size = AGE)) + geom_point(alpha = 0.5)
```



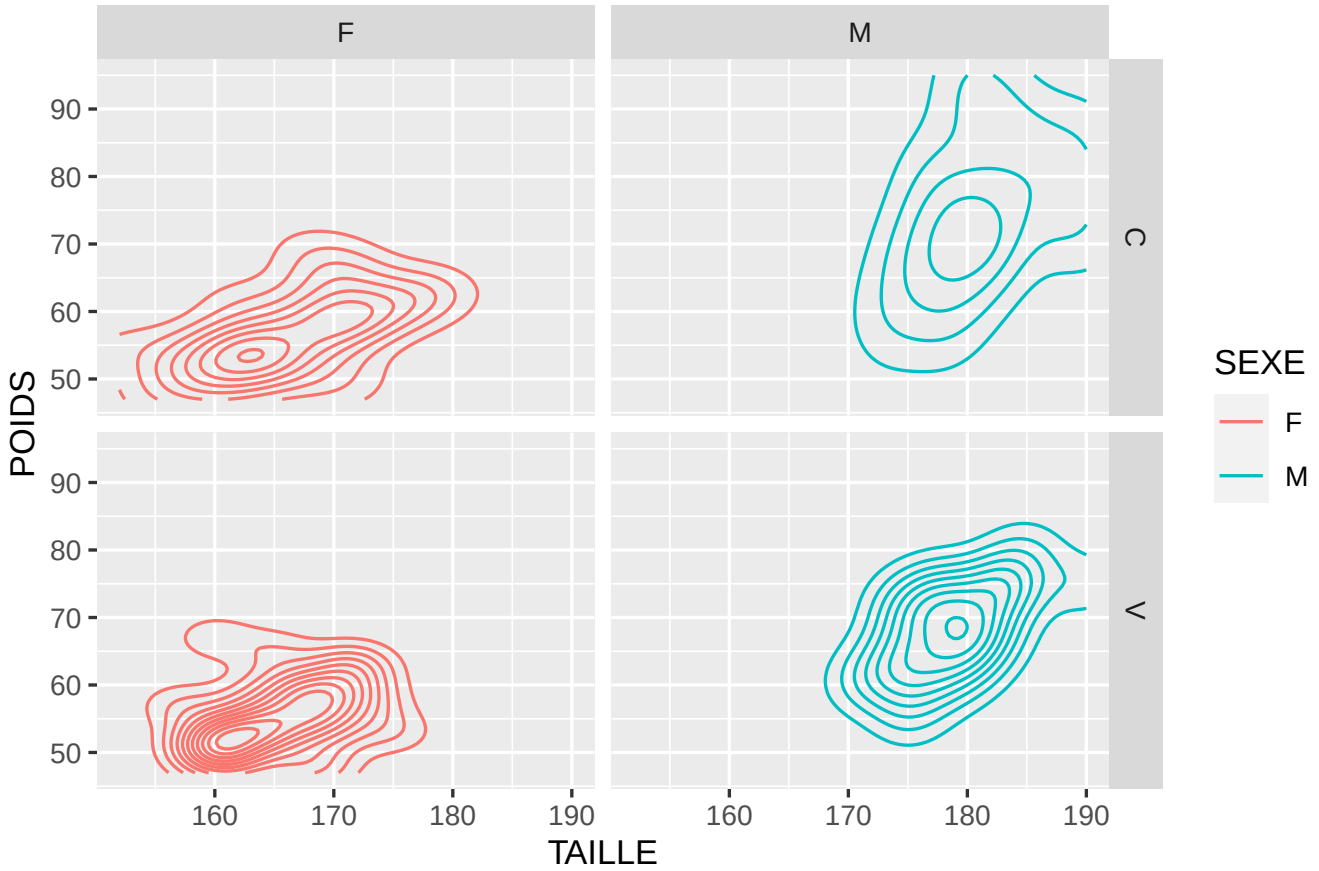
En séparant les figures par sexe et cadre de vie, avec l'âge codant pour la couleur

```
ggplot(d, aes(x = TAILLE, y = POIDS, colour = AGE)) +
    geom_point(alpha = 0.5) + facet_grid(CADRE ~ SEXE)
```



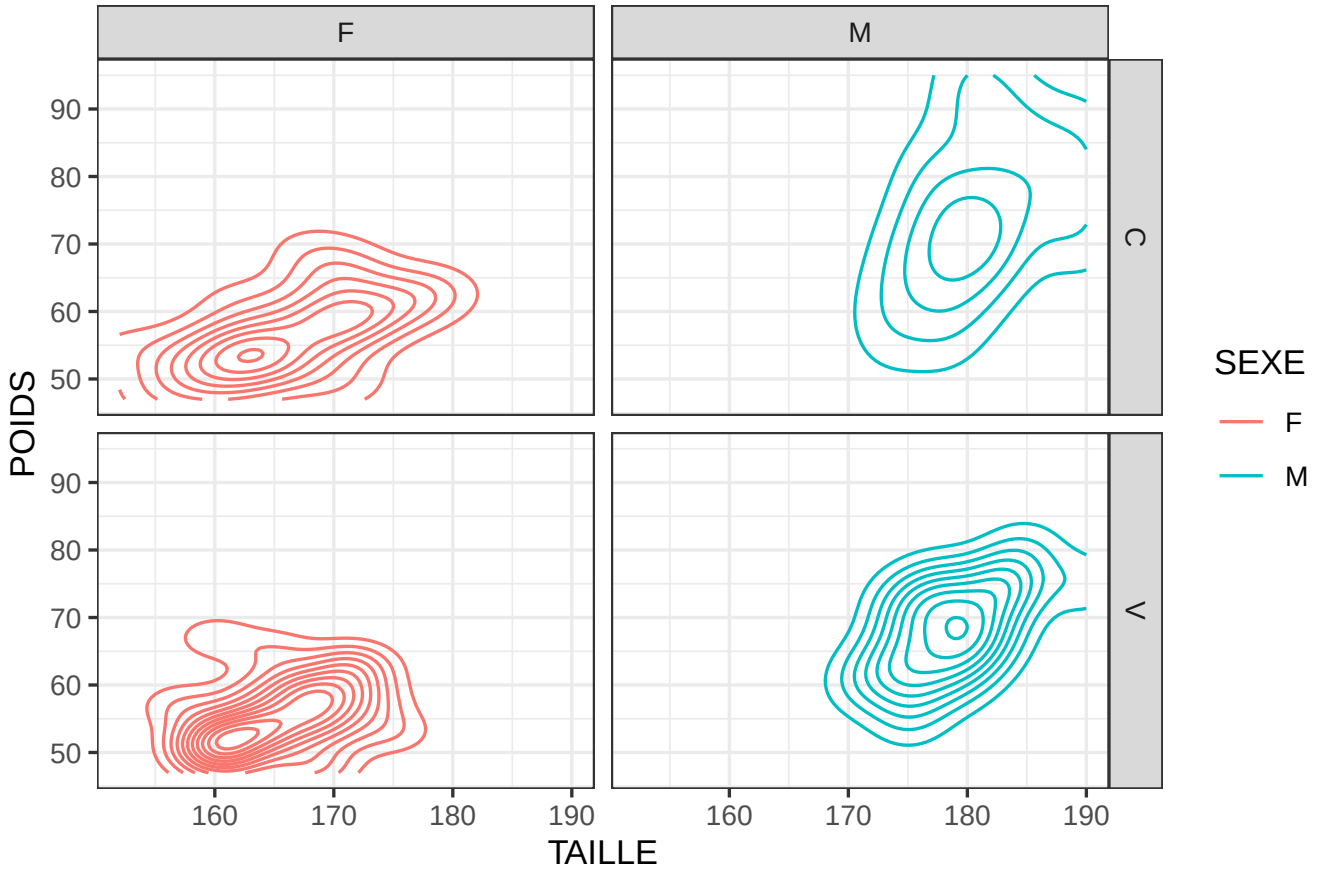
En contours de la densité de probabilité

```
ggplot(d, aes(x = TAILLE, y = POIDS, colour = SEXE)) +
  geom_density2d() + facet_grid(CADRE ~ SEXE)
```



Changement de thème en black and white par ex.

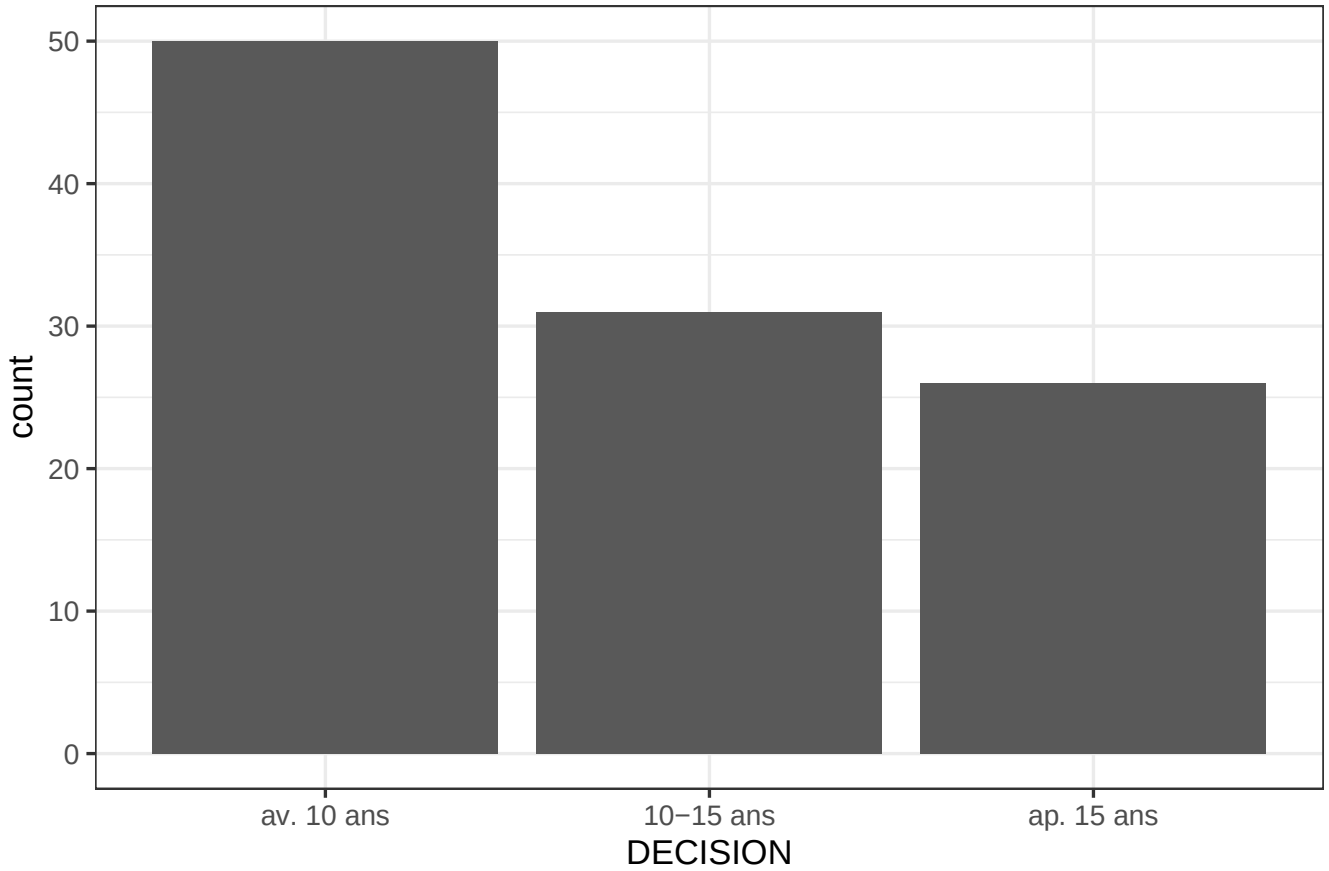
```
ggplot(d, aes(x = TAILLE, y = POIDS, colour = SEXE)) +
  geom_density2d() + facet_grid(CADRE ~ SEXE) + theme_bw()
```



Représentation d'une variable qualitative

Ex. : âge à la décision.

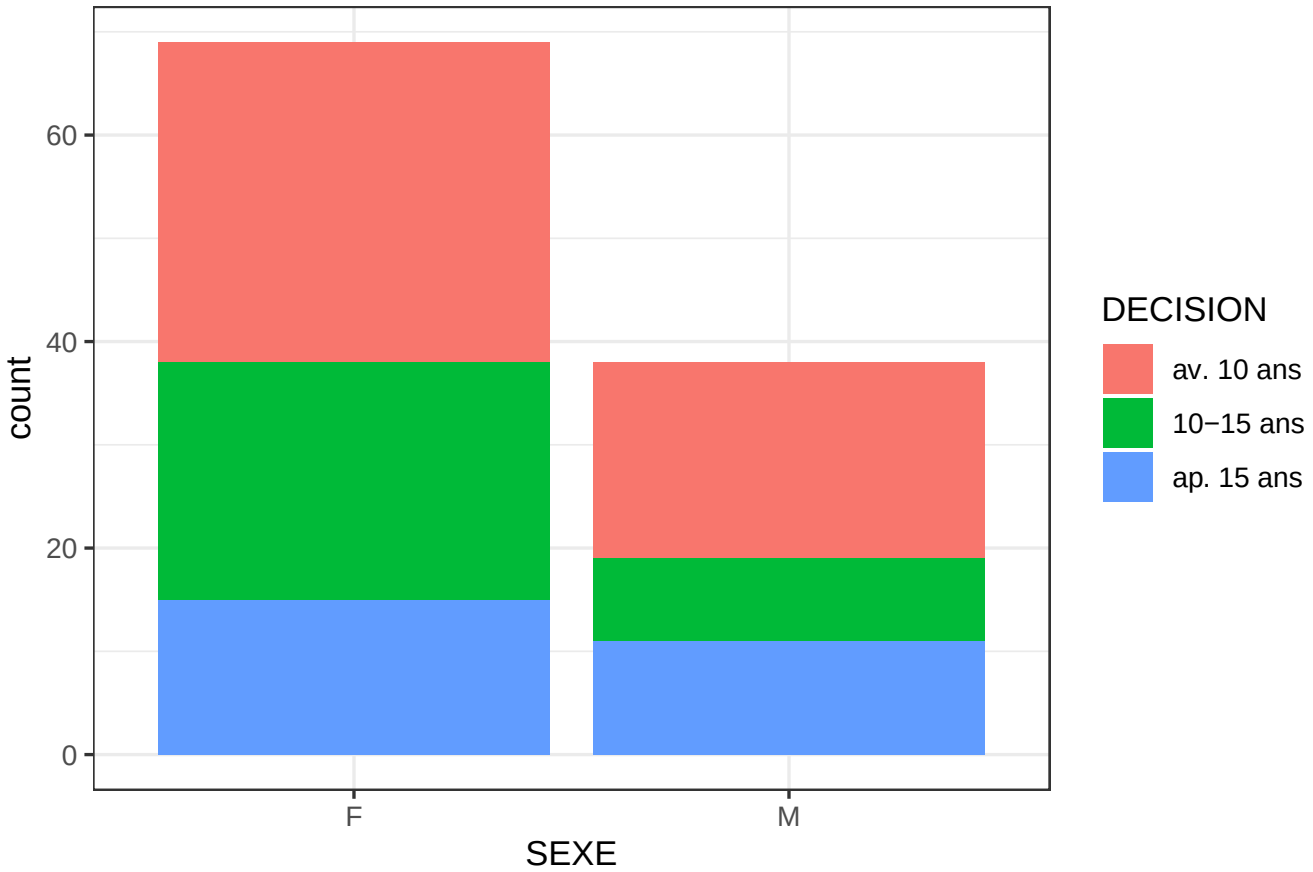
```
ggplot(d, aes(x = DECISION)) +
  geom_bar() + theme_bw()
```



Représentation de 2 variables qualitatives

Ex. : âge à la décision et sexe.

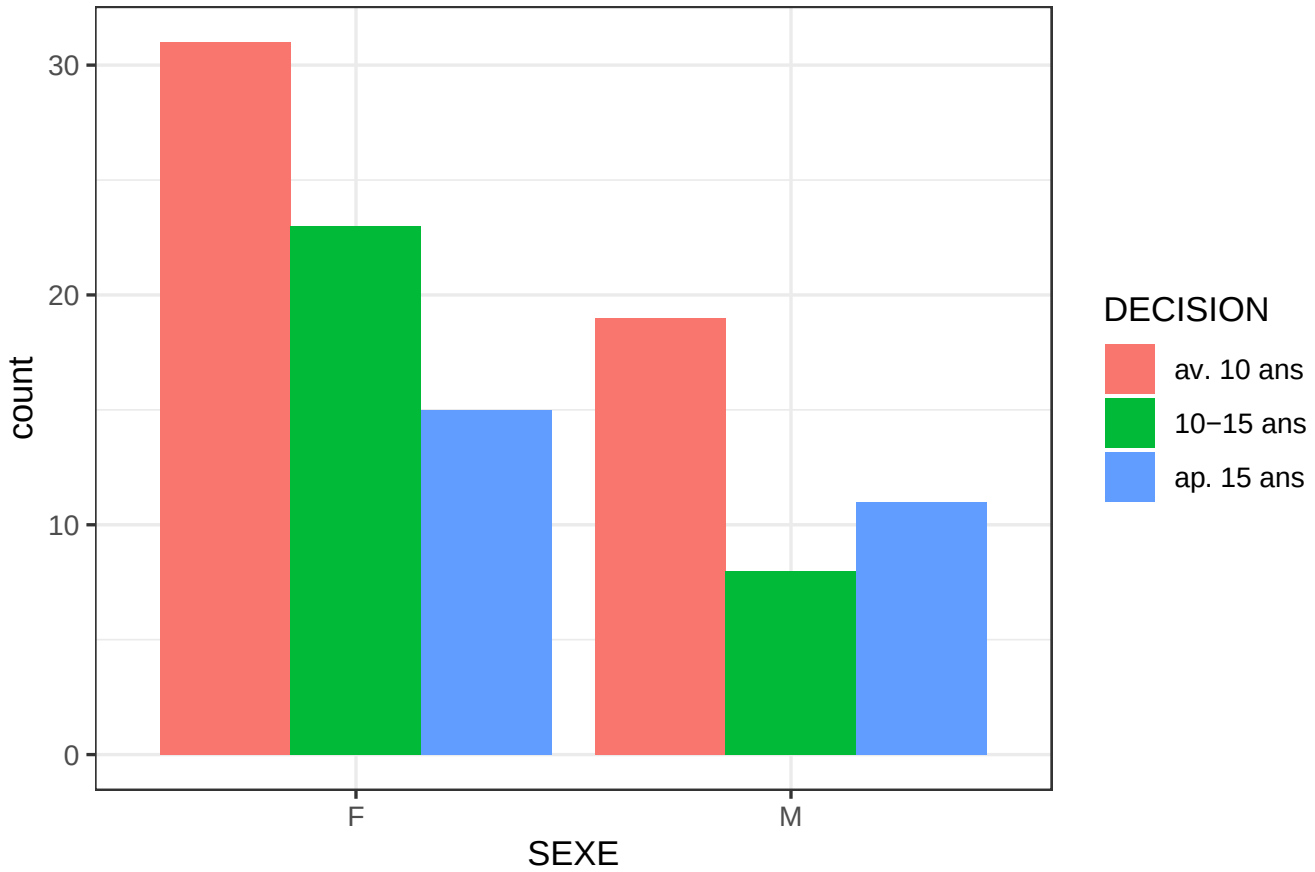
```
ggplot(d, aes(x = SEXE, fill = DECISION)) +
  geom_bar() + theme_bw()
```



Représentation de 2 variables qualitatives - variante 2

Ex. : âge à la décision et sexe.

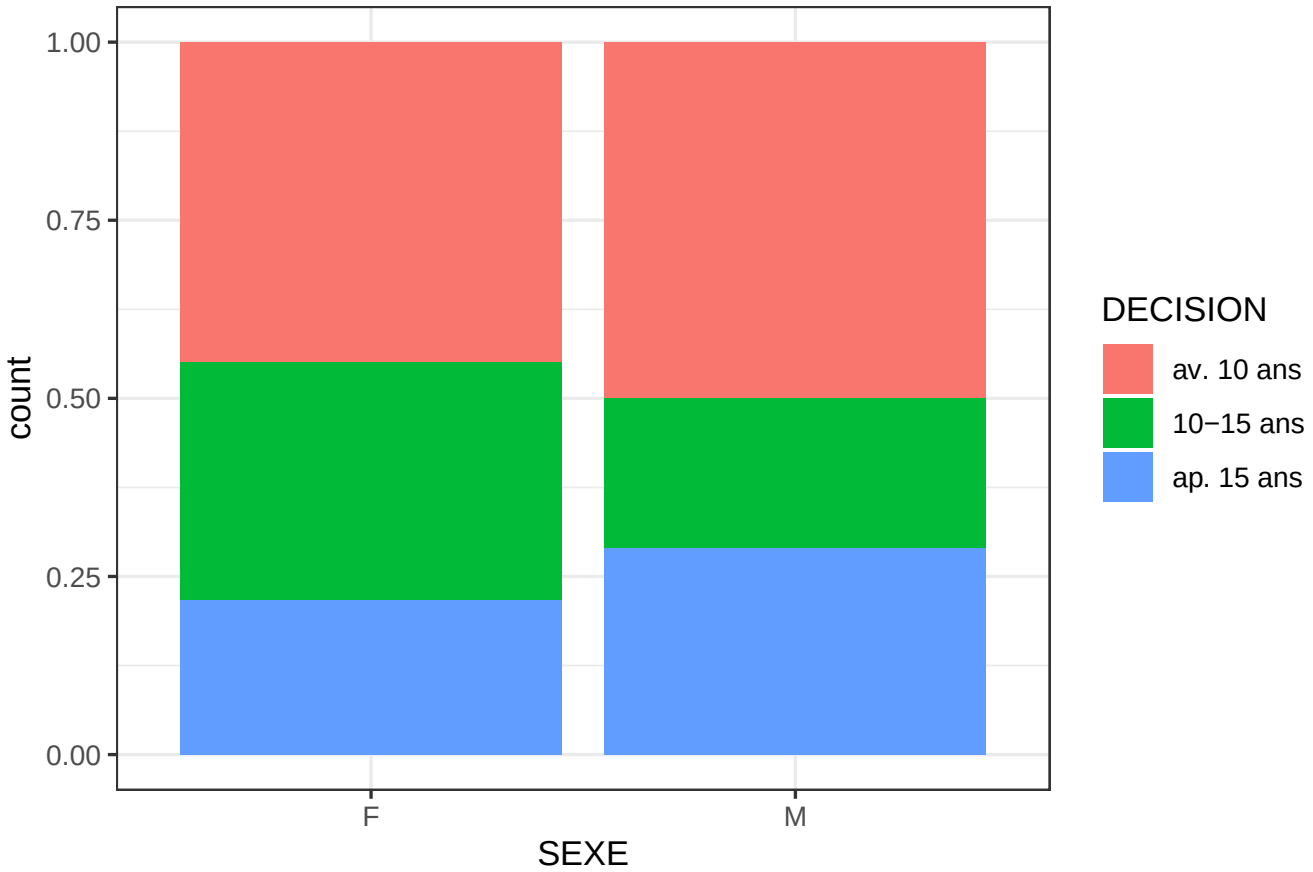
```
ggplot(d, aes(x = SEXE, fill = DECISION)) +
  geom_bar(position = "dodge") + theme_bw()
```



Représentation de 2 variables qualitatives - variante 3

Ex. : âge à la décision et sexe.

```
ggplot(d, aes(x = SEXE, fill = DECISION)) +
  geom_bar(position = "fill") + theme_bw()
```



Application à un exemple classique de **données longitudinales**

Suivi au cours du temps du poids de souris appartenant à deux groupes

```
dlong <- read.table("DATA/tramadol_poids.txt", header = TRUE,
                    dec = ",", stringsAsFactors = TRUE)
str(dlong)
```

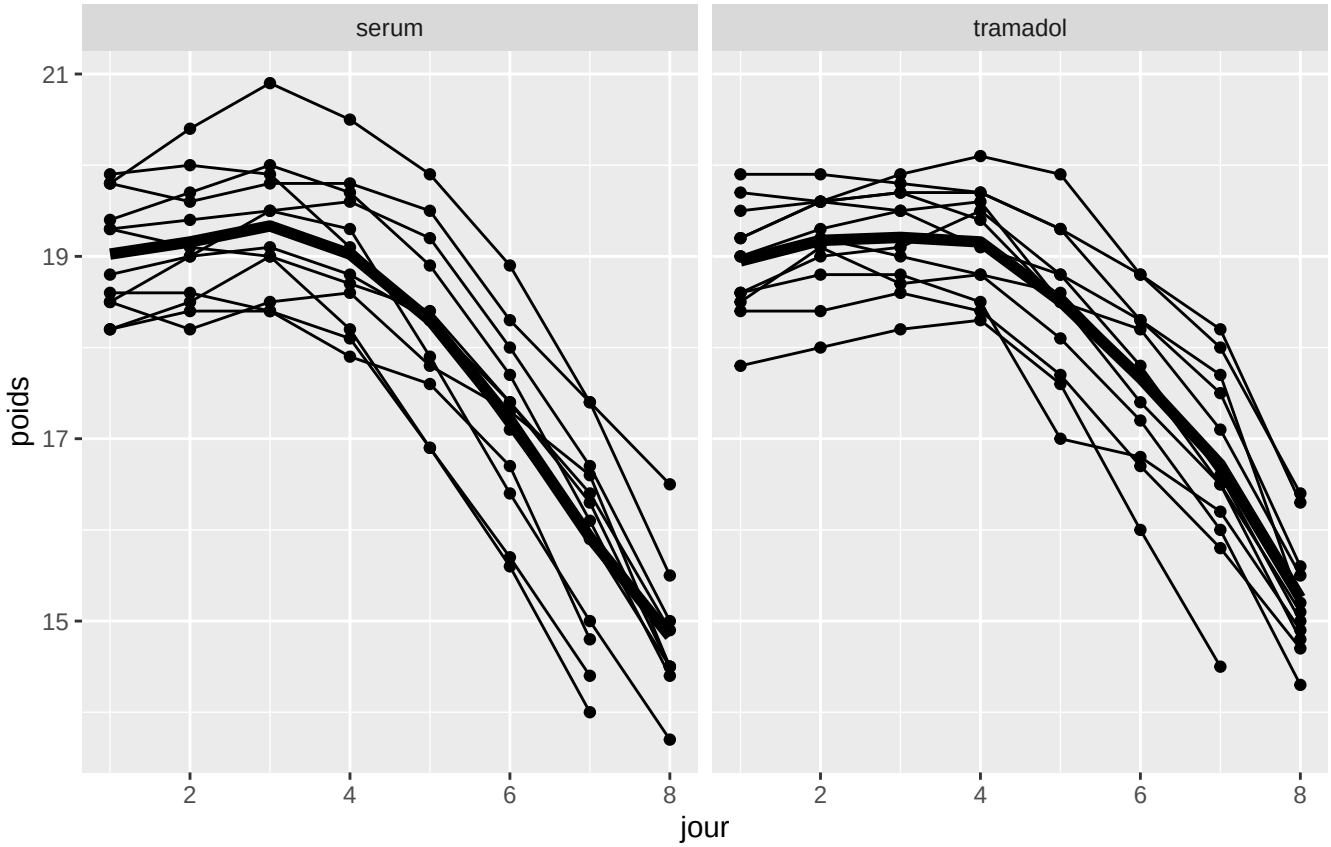
```
'data.frame':        192 obs. of  4 variables:
 $ souris    : Factor w/ 24 levels "B","BB","BBR",..: 1 14 21
 $ jour      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ traitement: Factor w/ 2 levels "serum","tramadol": 2 2 2 2
 $ poids     : num  19.2 17.8 19.2 18.4 18.6 19 19.5 18.5 18.5
```

Consigne

Proposez diverses façons de représenter les cinétiques de poids individuelles à l'aide de la fonction `ggplot()`.

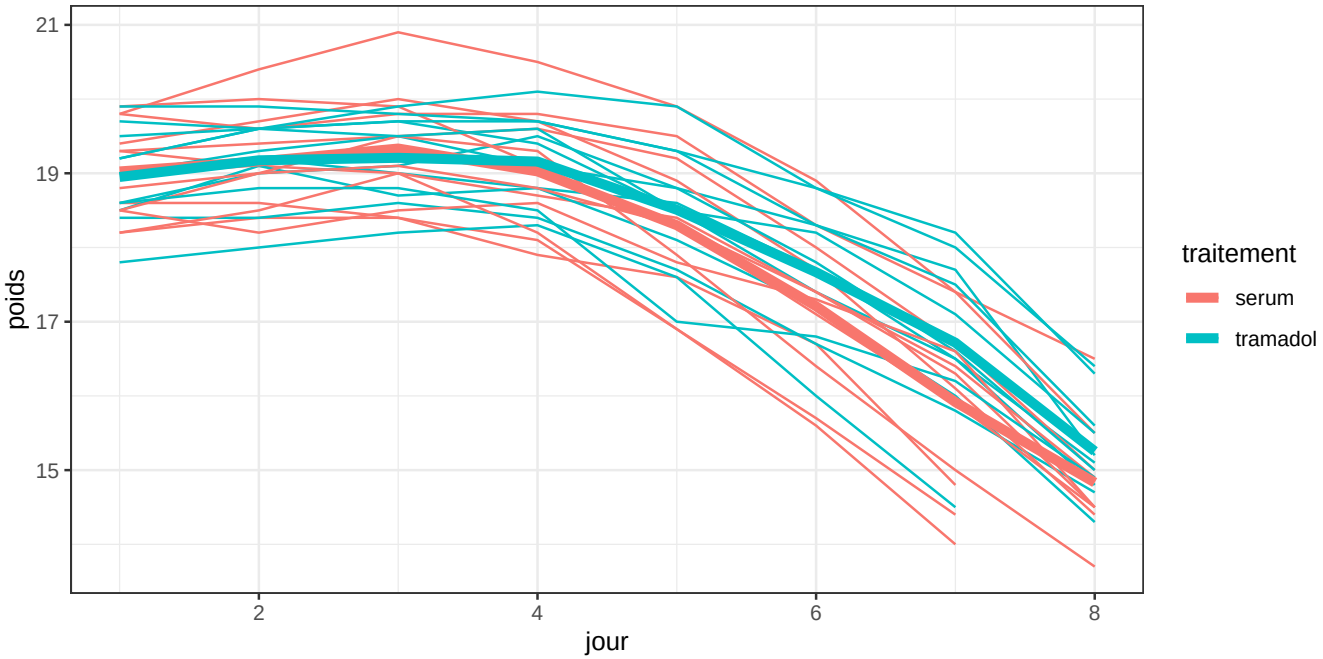
Ajout des courbes moyennes (version 1)

```
ggplot(dlong, aes(x = jour, y = poids, group = souris)) +
  geom_line() + geom_point() + facet_wrap(~ traitement) +
stat_summary(aes(group = traitement), fun = mean,
             geom = "line", lwd = 2)
```



Ajout des courbes moyennes (version 2)

```
ggplot(dlong, aes(x = jour, y = poids, group = souris,
                  colour = traitement)) + geom_line() +
stat_summary(aes(group = traitement), fun = mean,
             geom = "line", lwd = 2) + theme_bw()
```



Personnalisation des graphes avec ggplot2

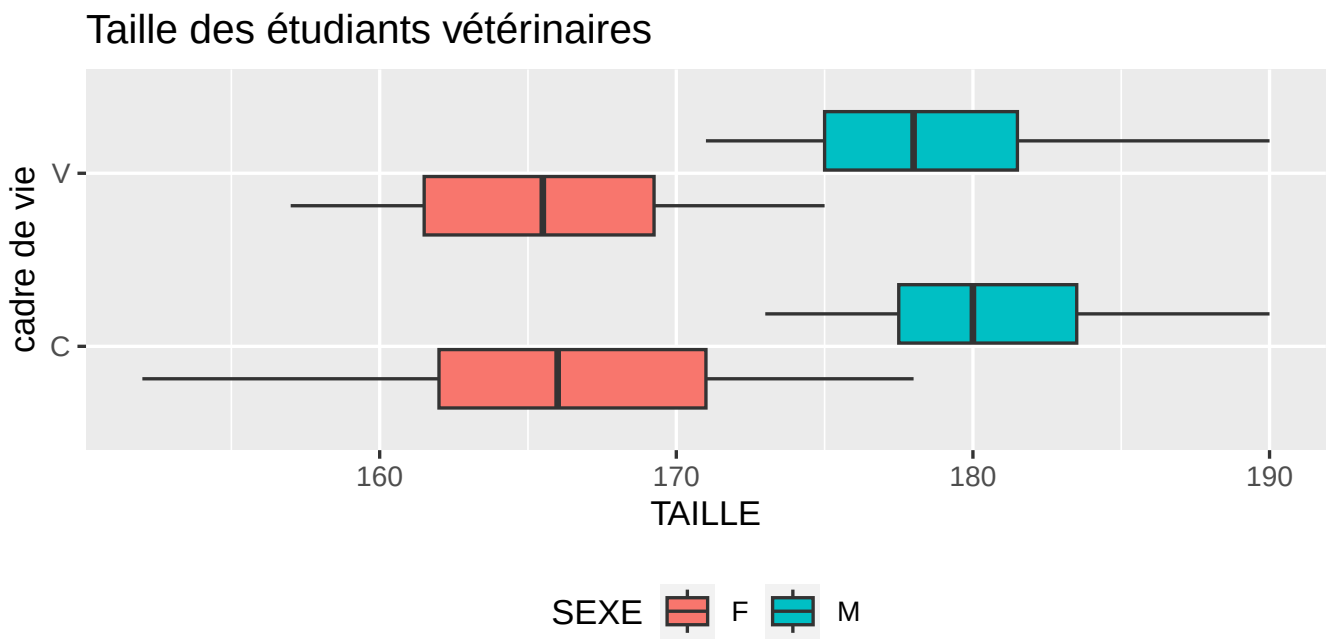
On peut personnaliser un graphe

- en utilisant les **arguments des fonctions** `geom` (pour accéder aux arguments personnalisables, voir l'aide de ces fonctions),
- en changeant le thème (on peut accéder à la description des**différents thèmes**, en consultant l'aide de l'un quelconque des thèmes).
- en ajoutant d'autres fonctions complémentaires :
  - `labs()` pour changer titres, labels des axes, ...,
  - `theme()` pour changer la position de la légende,
  - les fonctions de type `coord` pour modifier le système de coordonnées,
  - fonctions de type `scale` pour modifier les diverses échelles (quantitative, couleur, ...)

Et pour sauver le graphe en différents formats : `?ggsave()`

Ex. d'utilisation de quelques autres fonctions complémentaires

```
ggplot(data = d, aes(x = CADRE, y = TAILLE, fill = SEXE)) +
  geom_boxplot() + theme(legend.position = "bottom")  +
  labs(title = "Taille des étudiants vétérinaires",
       x = "cadre de vie") + coord_flip()
```



Pour aller plus loin . . .

On peut utiliser la fonction `ggplot()` et les multiples fonctions du package, dont je ne vous ai montré qu'une infime partie, pour créer et personnaliser des graphes bien plus sophistiqués. Cela nécessite un peu plus d'investissement mais de nombreuses informations sont disponibles en ligne (il suffit de taper quelques mots clefs en anglais dans un navigateur) et le "cookbook" de ggplot2 est bien utile :
`http://www.cookbook-r.com/Graphs/`

Très utile aussi, l'antisèche de ggplot2 :
`https://posit.co/wp-content/uploads/2022/10/data-visualization-1.pdf`

Le package `esquisse` pour vous aider à démarrer un nouveau graphe

Sa vignette :

`https://cran.r-project.org/web/packages/esquisse/vignettes/get-started.html`

Vous pouvez l'installer et l'essayer rapidement sur un exemple de graphe à partir de l'objet R de type data.frame dans lequel vous avez importé les données de "ENQ9697.txt" par exemple.

```
require(esquisse)
esquisser(d)
```

Très utile : à la fin on peut récupérer le code R associé, pour le copier dans un script.

Quand utiliser `graphics` ou `ggplot2` ?

- `ggplot2` permet de réaliser en un nombre réduit de lignes de code des graphes permettant de visualiser l'effet de nombreux facteurs.
- `ggplot2` gère automatiquement les légendes mais avec des choix par défaut (couleurs, types de points ...) qui sont un peu plus difficiles à modifier qu'avec `graphics`.
- Lorsque l'on veut réaliser un graphe très spécifique et très personnalisé il reste parfois plus simple de le réaliser avec `graphics`, enfin pour ceux qui en avaient l'habitude.

A voir si les deux packages restent donc complémentaires ou si on peut tout faire avec ggplot2 ?

Et si on part d'un jeu de données au format large ?

Prenons l'exemple d'un jeu de données donnant les notes obtenues dans diverses disciplines par des étudiants groupe.

```
str(dlarge)
```

```
'data.frame':        35 obs. of  7 variables:
 $ nom  : Factor w/ 35 levels "etu_1","etu_10",..: 1 12 23 30
 $ genre: Factor w/ 2 levels "femme","homme": 1 1 1 1 1 1 1 1
 $ UE1  : num  8.8 9.8 15.2 10.7 10.9 15.6 11.9 6.7 8.4 9.2 ..
 $ UE2  : num  13.2 13.1 12.4 12.2 12.1 11.8 12.3 11.2 14.7 13
 $ UE3  : num  14 10.4 17 13.6 13.6 17.1 14.4 12.6 15.4 14.7 .
 $ UE4  : num  8.9 7.8 6.5 8.4 10.4 8.1 9.9 6.6 8.9 9.8 ...
 $ UE5  : num  16.1 13.2 9.3 9.5 9.2 12.4 9.6 16.1 20 10.1 ...
```

Utilité de la fonction `stack()` pour passer du format large au format long

```
dlong <- stack(dlarge[, -c(1,2)])
str(dlong)
```

```
'data.frame':        175 obs. of  2 variables:
 $ values: num  8.8 9.8 15.2 10.7 10.9 15.6 11.9 6.7 8.4 9.2 .
 $ ind   : Factor w/ 5 levels "UE1","UE2","UE3",..: 1 1 1 1 1
```

```
head(dlong)
```

```
  values ind
1    8.8 UE1
2    9.8 UE1
3   15.2 UE1
4   10.7 UE1
5   10.9 UE1
6   15.6 UE1
```

Manipulations complémentaires pour créer les variables utiles (1)

```
# changement des noms de colonnes
colnames(dlong) <- c("note", "UE")
str(dlong)
```

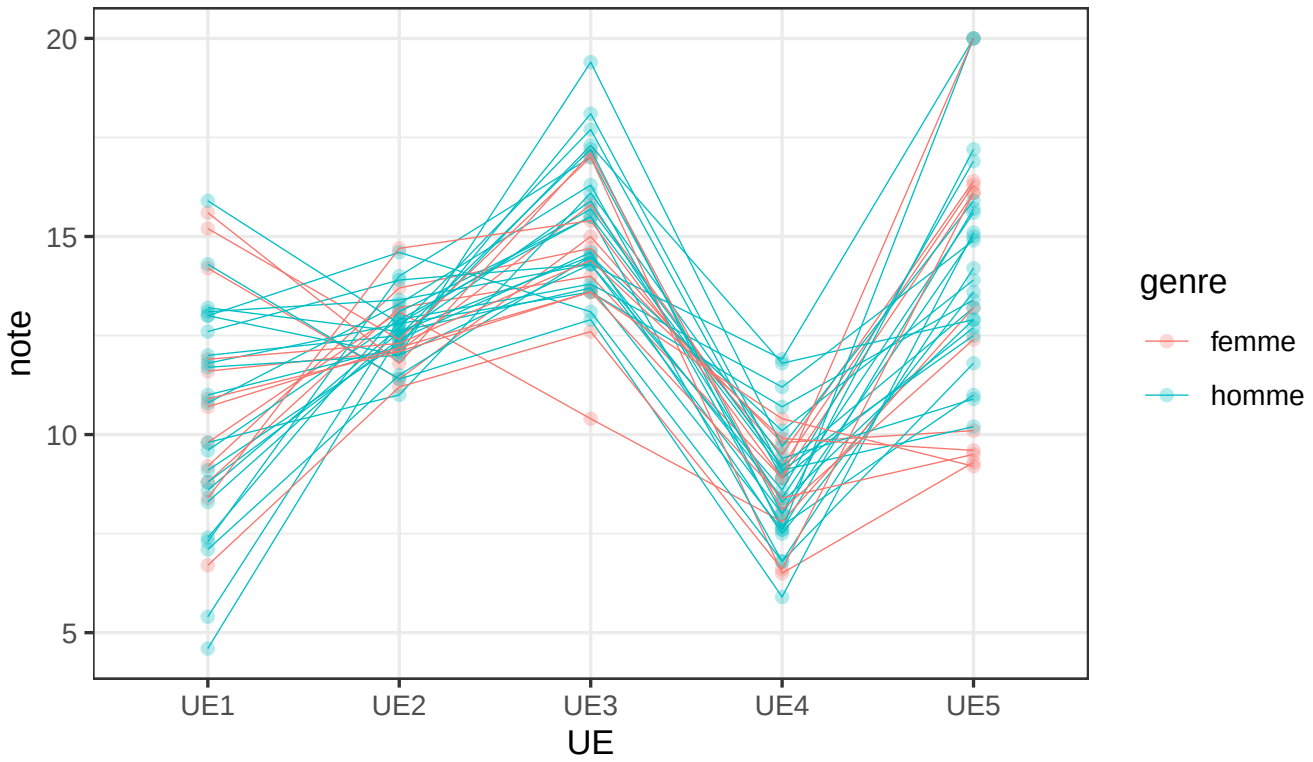
```
'data.frame':        175 obs. of  2 variables:
 $ note: num  8.8 9.8 15.2 10.7 10.9 15.6 11.9 6.7 8.4 9.2 ...
 $ UE  : Factor w/ 5 levels "UE1","UE2","UE3",..: 1 1 1 1 1 1
```

```
# récupération des nom et genre de chaque étudiant
dlong$nom <- rep(dlarge$nom, ncol(dlarge) - 2)
dlong$genre <- rep(dlarge$genre, ncol(dlarge) - 2)
str(dlong)
```

```
'data.frame':        175 obs. of  4 variables:
 $ note : num  8.8 9.8 15.2 10.7 10.9 15.6 11.9 6.7 8.4 9.2 ..
 $ UE   : Factor w/ 5 levels "UE1","UE2","UE3",..: 1 1 1 1 1 1
 $ nom  : Factor w/ 35 levels "etu_1","etu_10",..: 1 12 23 30
 $ genre: Factor w/ 2 levels "femme","homme": 1 1 1 1 1 1 1 1
```

Il n'y a plus qu'à utiliser `ggplot()` sur ce nouveau jeu de données reformaté

```
ggplot(dlong, aes(x = UE, y = note, group = nom,
        colour = genre)) + geom_point(alpha = 0.3) +
  geom_line(linewidth = 0.2) + theme_bw()
```



Et on peut en faire facilement une figure interactive

A tester si vous le souhaitez à partir du fichier FigureInteractive.Rmd.
Le résultat est dans FigureInteractive.html.

```
require(plotly)
g <- ggplot(dlong, aes(x = UE, y = note, group = nom,
        colour = genre)) + geom_point(alpha = 0.3) +
  geom_line(linewidth = 0.2) + theme_bw()
```

L'utilisation de fonctions peut être utile pour créer un graphe personnalisé

Pour vous montrer un exemple simple prenons le jeu de données suivant sur lequel nous allons, pour chaque vache, calculer son GMQ par régression linéaire, et le reporter sur chaque graphe de régression.

```
dGMQ <- read.table("DATA/GMQ_6vaches.txt",
                   header = TRUE, stringsAsFactors = TRUE)
str(dGMQ)
```

```
'data.frame':         42 obs. of  3 variables:
 $ nom       : Factor w/ 6 levels "Betty","Blondie",
 $ age_jours: int  182 212 243 273 304 334 365 182
 $ poids_kg : int  134 155 177 187 201 215 232 156
```

Comment écrire une fonction

Principe de codage d'une fonction **R**

```
nom_fonction <- function(arg1, arg2, ...){
    instruction1
    instruction2
    ...
    # si on veut sortir un objet qui n'est pas
    # défini dans la dernière instruction
    return(sortie)
}
```

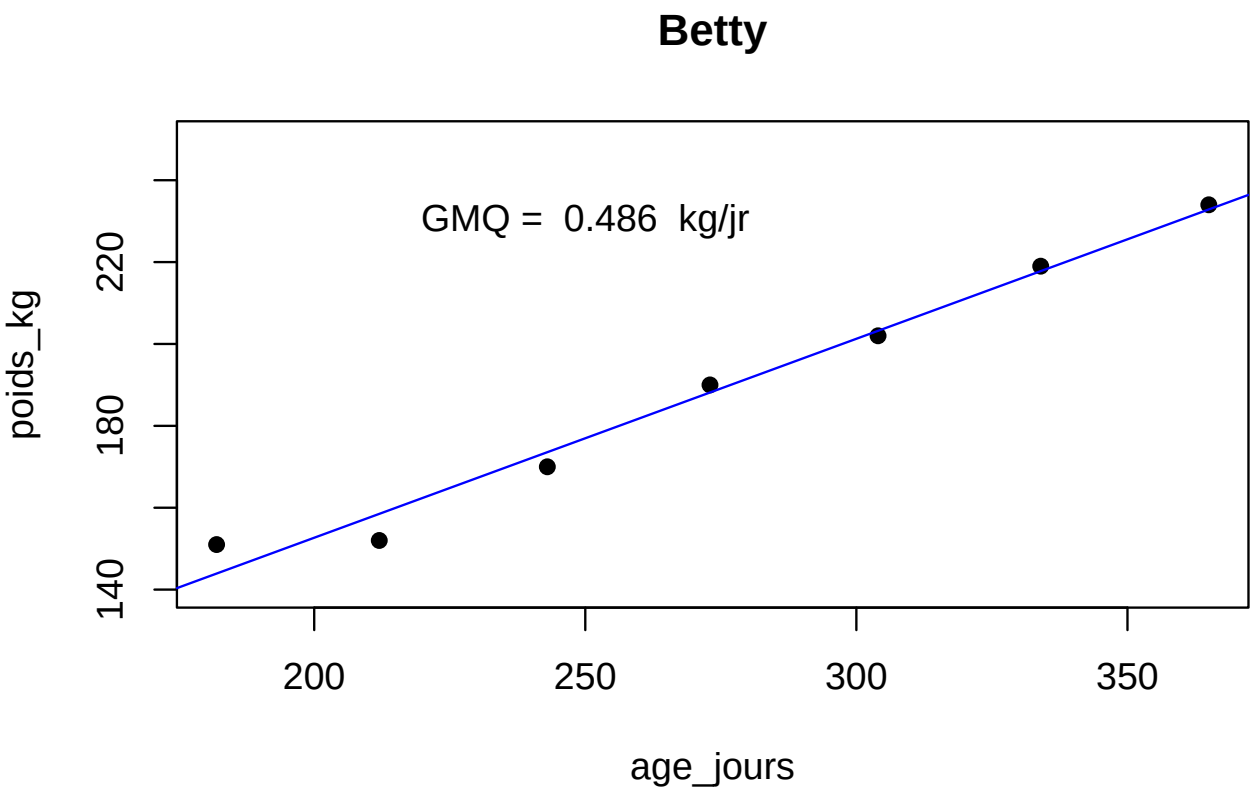
Exemple de fonction

Fonction qui, pour un sous-jeu de données `d1vache` correspondant à une vache, fait la régression et trace le graphe souhaité, en y reportant le GMQ

```
GMQ <- function(d1vache){
    reg <- lm(poids_kg ~ age_jours, data = d1vache)
    nomvache <- d1vache$nom[1]
    plot(poids_kg ~ age_jours, data = d1vache, pch = 16,
         main = nomvache, ylim = c(140, 250))
    abline(reg, col = "blue")
    gmq <- round(coef(reg)[2], 3)
    text(250, 230, paste("GMQ = ", gmq, " kg/jr"))
    return(gmq)
}
```

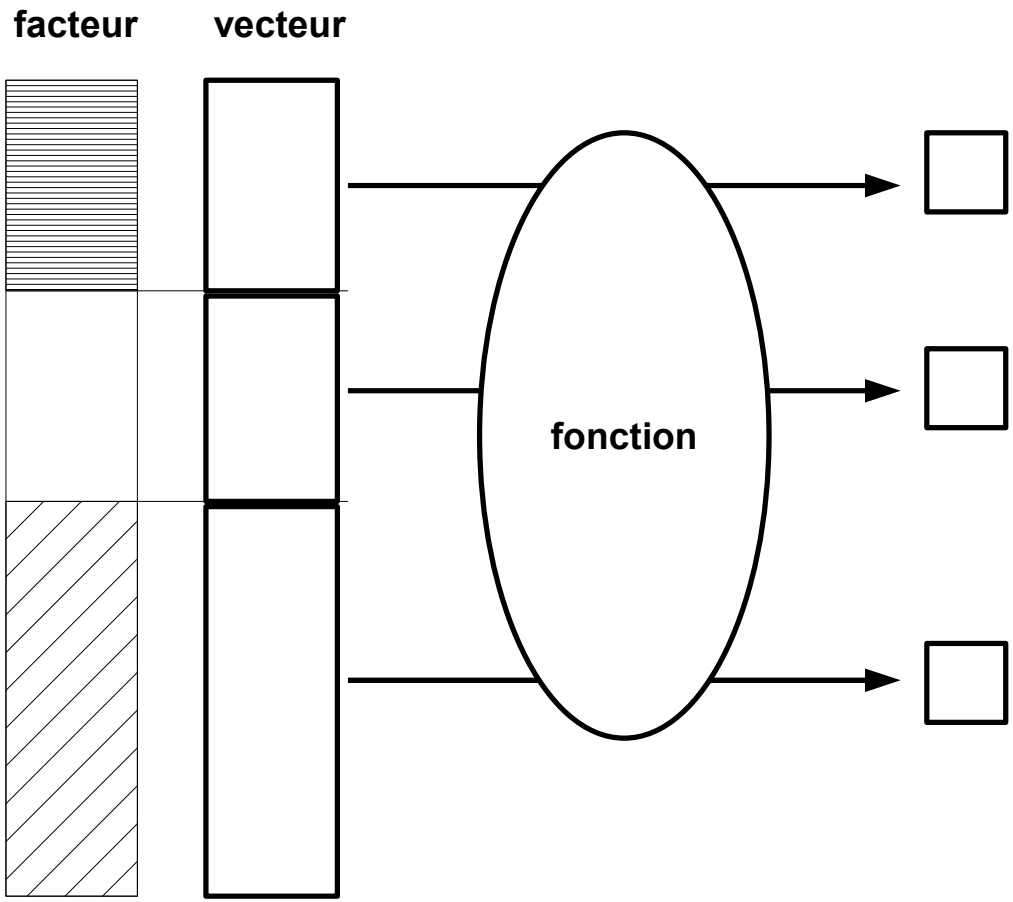
Exemple d'application de la fonction à une vache

```
dBetty <- subset(dGMQ, nom == "Betty")
GMQ(dBetty)
```



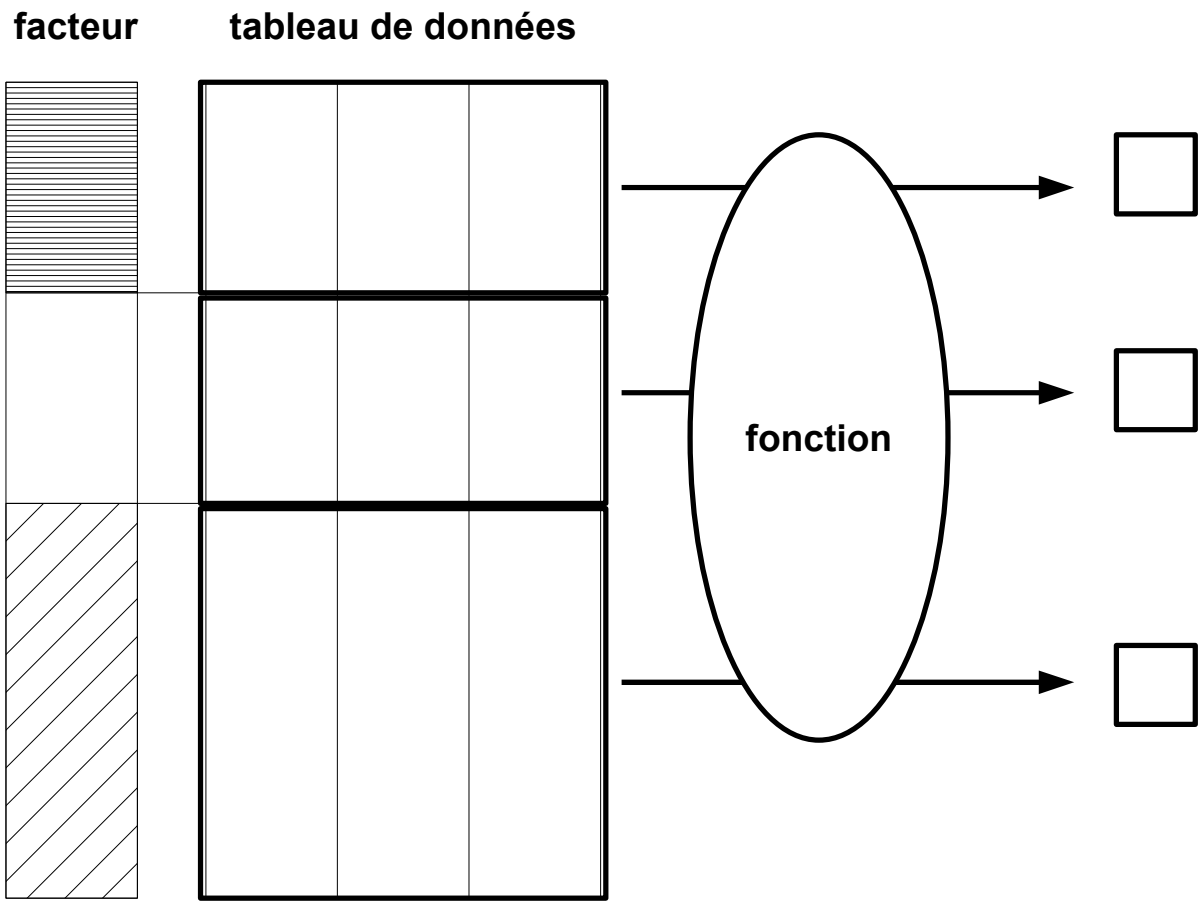
Rappel du principe de la fonction `tapply`

tapply(vecteur, facteur, fonction)



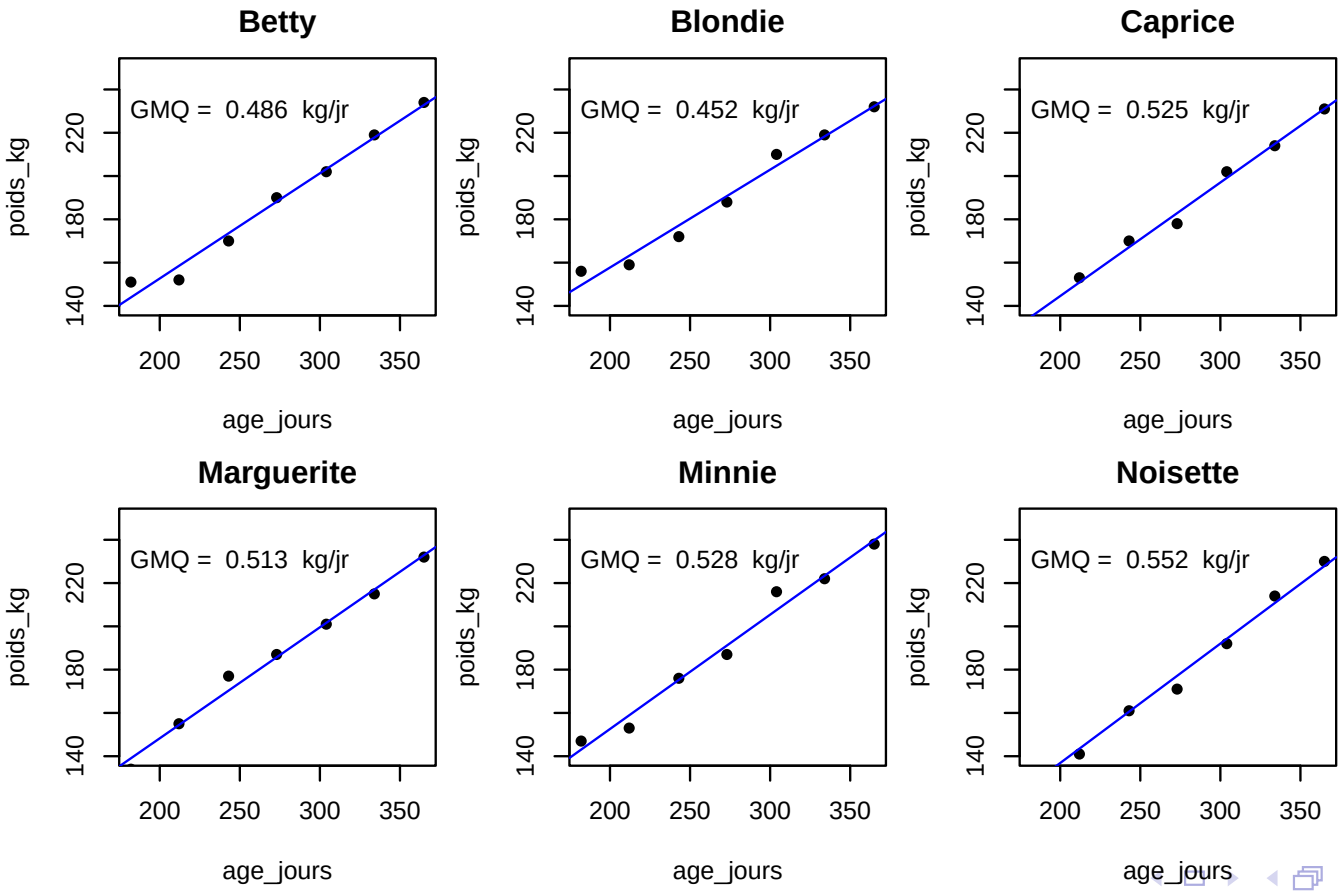
Principe de la fonction `by`

```
by(tableau_de_donnees, facteur, fonction)
```



Application de la fonction à toutes les vaches à l'aide de `by`

```
par(mfrow = c(2,3), mar = c(4, 4, 2.5, 0.5))
by(dGMQ, dGMQ$nom, GMQ)
```



A vous de jouer !

Consigne

Prenez vos propres jeux de données, ou le jeu de données de l'article de Sandoe *et al.* 2023, et tentez de créer des graphes intéressants. Vous pouvez aussi, au choix, reproduire le graphe ci-dessous.