Compléments sur R

Martin Chevalier (Insee)

Année universitaire 2017-2018

R et la réalisation de graphiques

La réalisation de graphiques dans un logiciel statistique est une opération souvent longue et complexe.

Dans la plupart des cas, l'ajustement fin des paramètres par le biais de lignes de code relève de la gageure.

R dispose néanmoins de plusieurs caractéristiques qui facilitent la réalisation de graphiques :

- **souplesse** : la très grande variété des types d'objets simplifie les paramétrages ;
- **rigueur** : la dimension fonctionnelle du langage aide à systématiser l'utilisation des paramètres graphiques ;
- **adaptabilité** : la liberté de développement de modules complémentaires rend possible de profondes innovations dans la conception des graphiques.

Base R ou ggplot2 ?

Il existe aujourd'hui troix principaux paradigmes pour produire des graphiques avec R :

- les fonctionnalités de base du logiciel du *package* `graphics` ;
- les fonctionnalités plus élaborées des *packages* `grid` et `lattice` (non-abordées dans cette formation) ;
- la « grammaire des graphiques » du *package* `ggplot2`.

Plan de la partie

Données d'exemple : table mpg de ggplot2

La plupart des exemples de cette partie sont produits à partir de la table mpg du *package* ggplot2.

```
library(ggplot2)
dim(mpg)
## [1] 234  11
names(mpg)
##  [1] "manufacturer" "model"        "displ"        "year"
##  [5] "cyl"          "trans"        "drv"          "cty"
##  [9] "hwy"          "fl"           "class"
```

- displ : cylindrée ;
- drv : transmission (f traction, r propulsion, 4 quatre roues motrices) ;
- cty et hwy : nombre de *miles* parcourus par *gallon* d'essence en ville et sur autoroute respectivement.

Beaucoup de fonctions, des paramètres communs

La création de graphiques avec le *package* de base `graphics` s'appuie sur la **fonction `plot()`** ainsi que sur des **fonctions spécifiques** :

- `plot(hist(x))`, `plot(density(x))` : histogrammes et densités ;
- `plot(ts)` : représentation de séries chronologiques ;
- `plot(x, y)` : nuages de points ;
- `barplot(table(x))` et `pie(table(x))` : diagrammes en bâtons et circulaires.

Si ce n'est quelques **arguments spécifiques**, ces fonctions partagent un ensemble de **paramètres graphiques communs**.

Pour en savoir plus Le site statmethods.net recense et illustre la plupart des fonctions du *package* `graphics`.

Histogrammes et densités

Les fonctions `histogram()` et `density()` calculent les statistiques ensuite utilisées par la fonction `plot()` pour construire les graphiques.

Arguments spécifiques à `hist()` :

- `breaks` : méthode pour déterminer les limites des classes ;
- `labels = TRUE` : ajoute l'effectif de chaque classe.

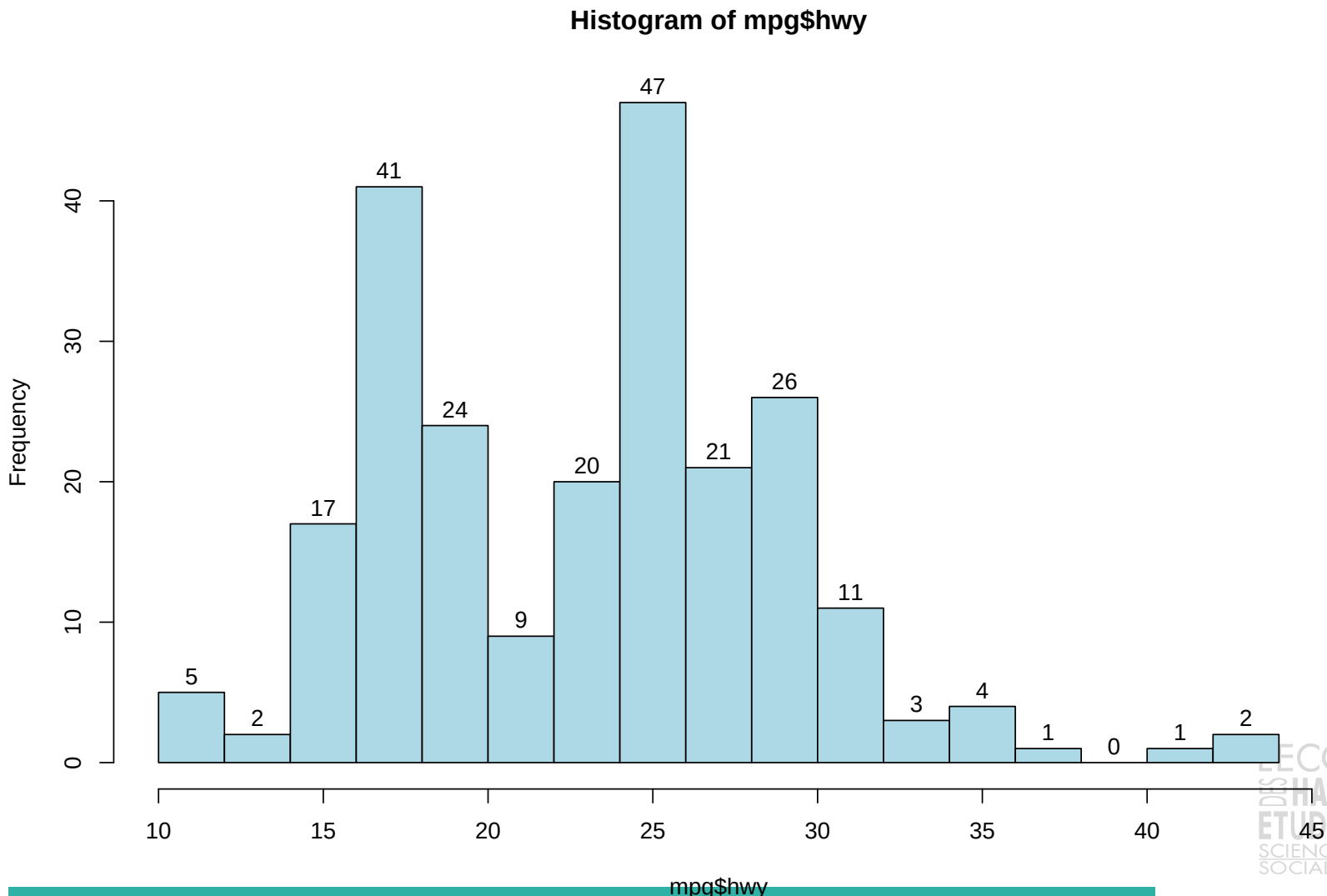
Arguments spécifiques à `density()` :

- `bw` : largeur de la fenêtre utilisée par la fonction de lissage ;
- `kernel` : fonction de lissage utilisée.

Remarque L'argument `plot` de la fonction `hist()` (`TRUE` par défaut) affiche automatiquement un graphique, sans avoir à appeler explicitement la fonction `plot()`.

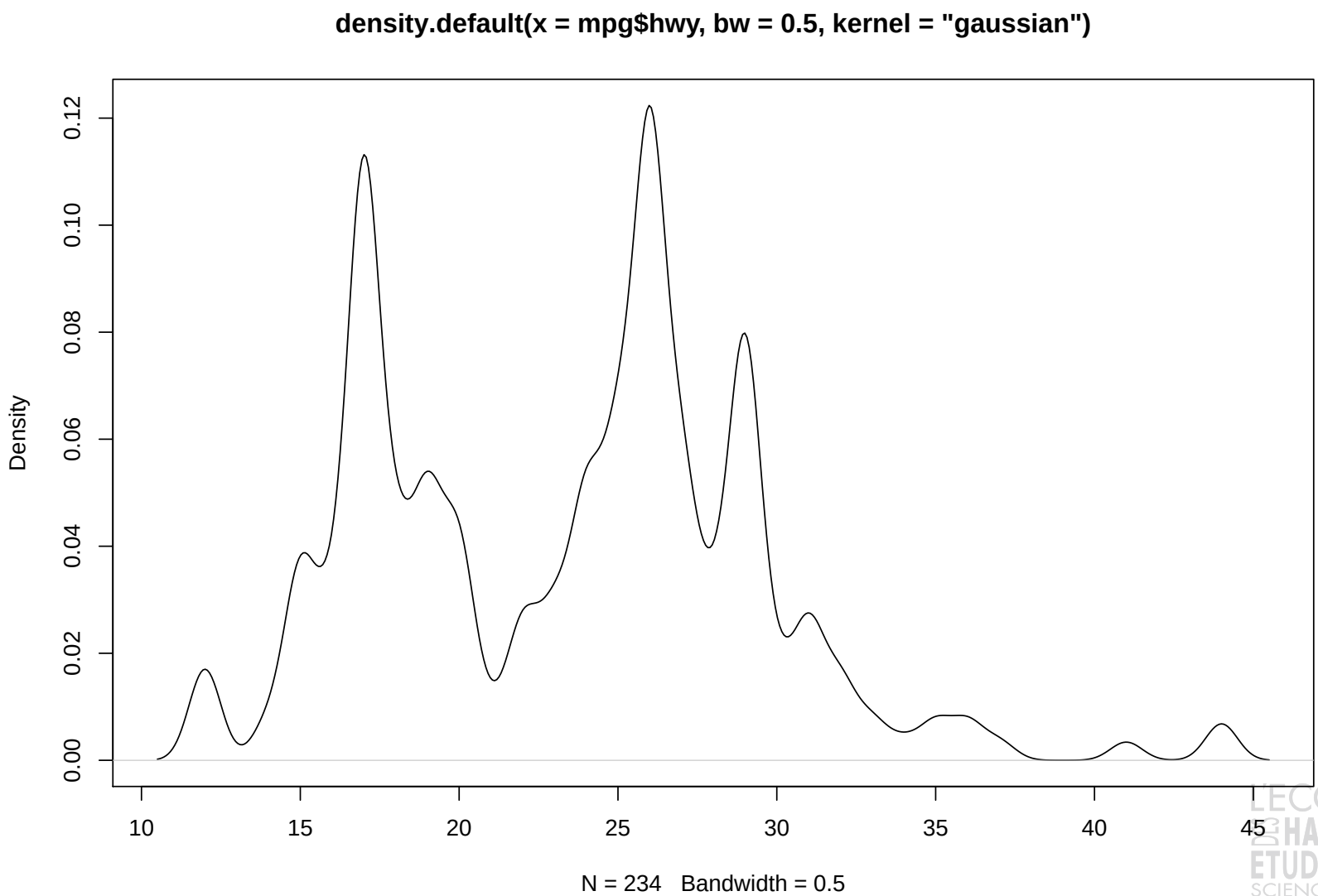
Histogrammes et densités

```
hist(mpg$hwy, breaks = seq(10, 44, by = 2),
     col = "lightblue", labels = TRUE)
```



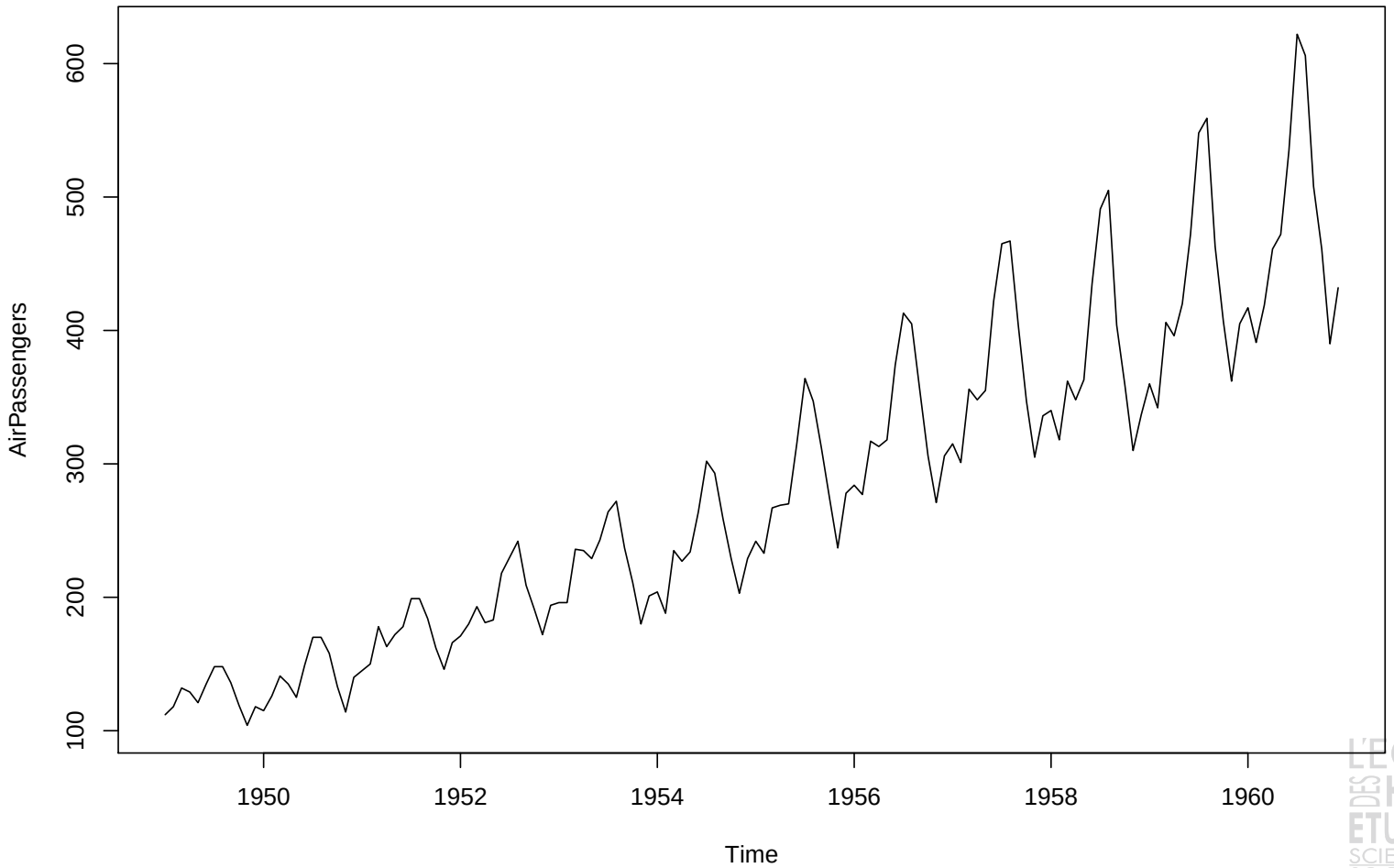
Histogrammes et densités

```
plot(density(mpg$hwy, bw = 0.5, kernel = "gaussian"))
```



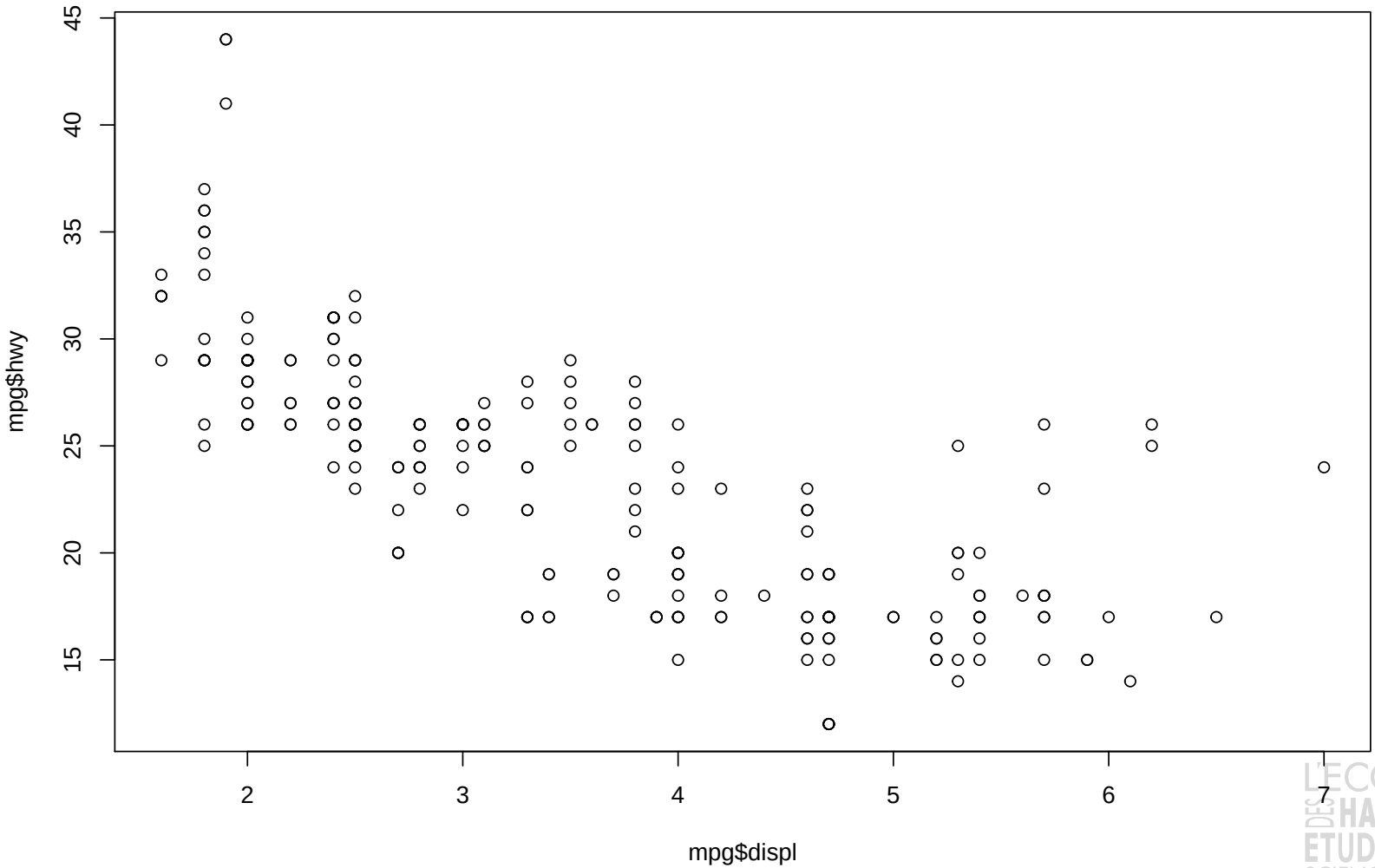
Séries chronologiques avec plot(ts)

```
class(AirPassengers)
## [1] "ts"
plot(AirPassengers)
```



Nuages de points avec plot(x, y)

```
plot(mpg$displ, mpg$hwy)
```



Diagrammes en bâtons et circulaires

La fonction `table()` permet de calculer les statistiques utilisées ensuite par `barplot()` et `pie()` pour construire les graphiques.

Arguments spécifiques à `barplot()` :

- `horiz` : construit le graphique horizontalement ;
- `names.arg` : nom à afficher près des barres.

Arguments spécifiques à `pie()` :

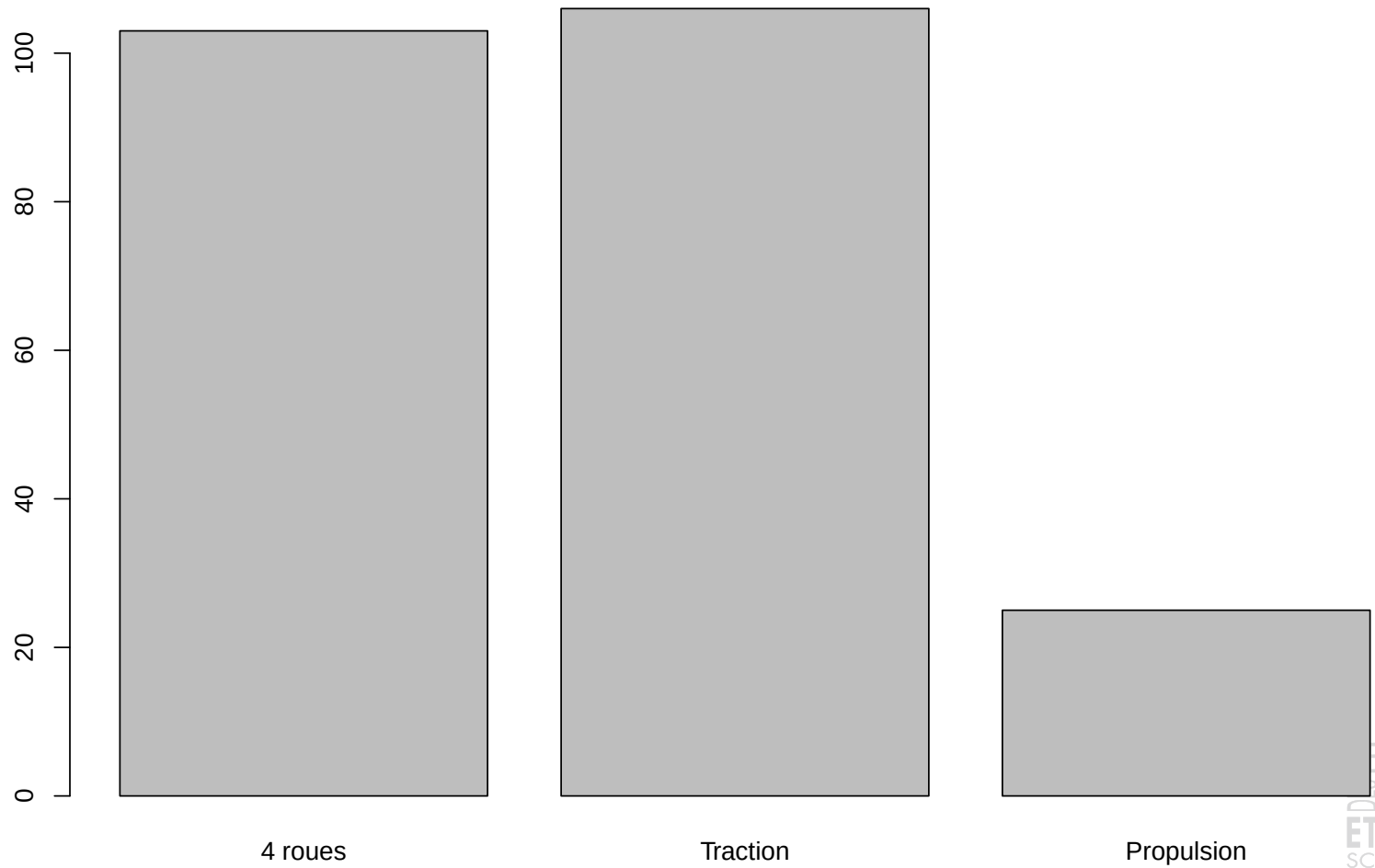
- `labels` : noms à afficher à côté des portions de disque ;
- `clockwise` : sens dans lequel sont représentées les modalités ;
- `init.angle` : point de départ en degrés.

Remarque Quand `barplot()` est appliqué à un tri croisé, la couleur des barres varie et les paramètres deviennent utiles :

- `beside` : position des barres ;
- `legend.text` : ajoute une légende avec le texte indiqué.

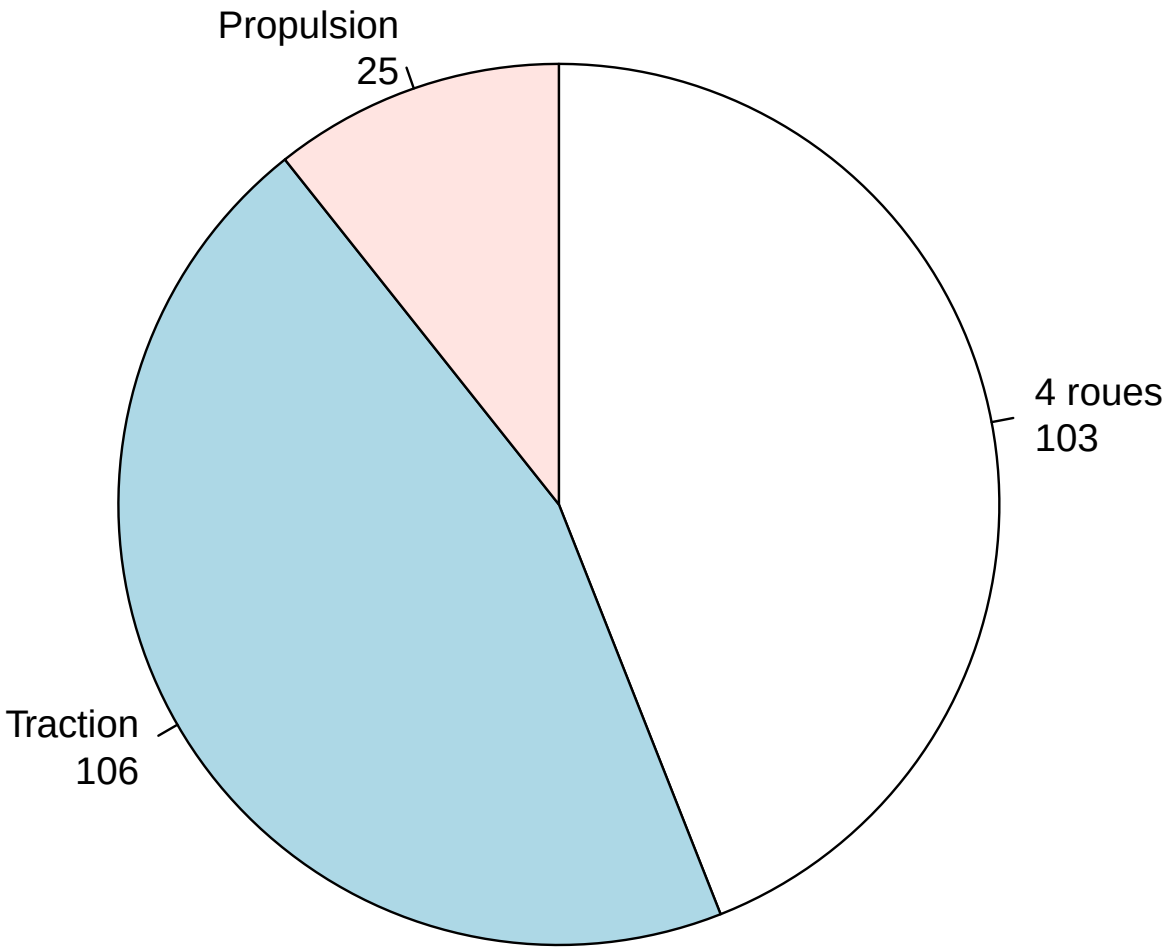
Diagrammes en bâtons et circulaires

```
uni <- table(mpg$drv)
lab <- c("4 roues", "Traction", "Propulsion")
barplot(uni, names.arg = lab)
```



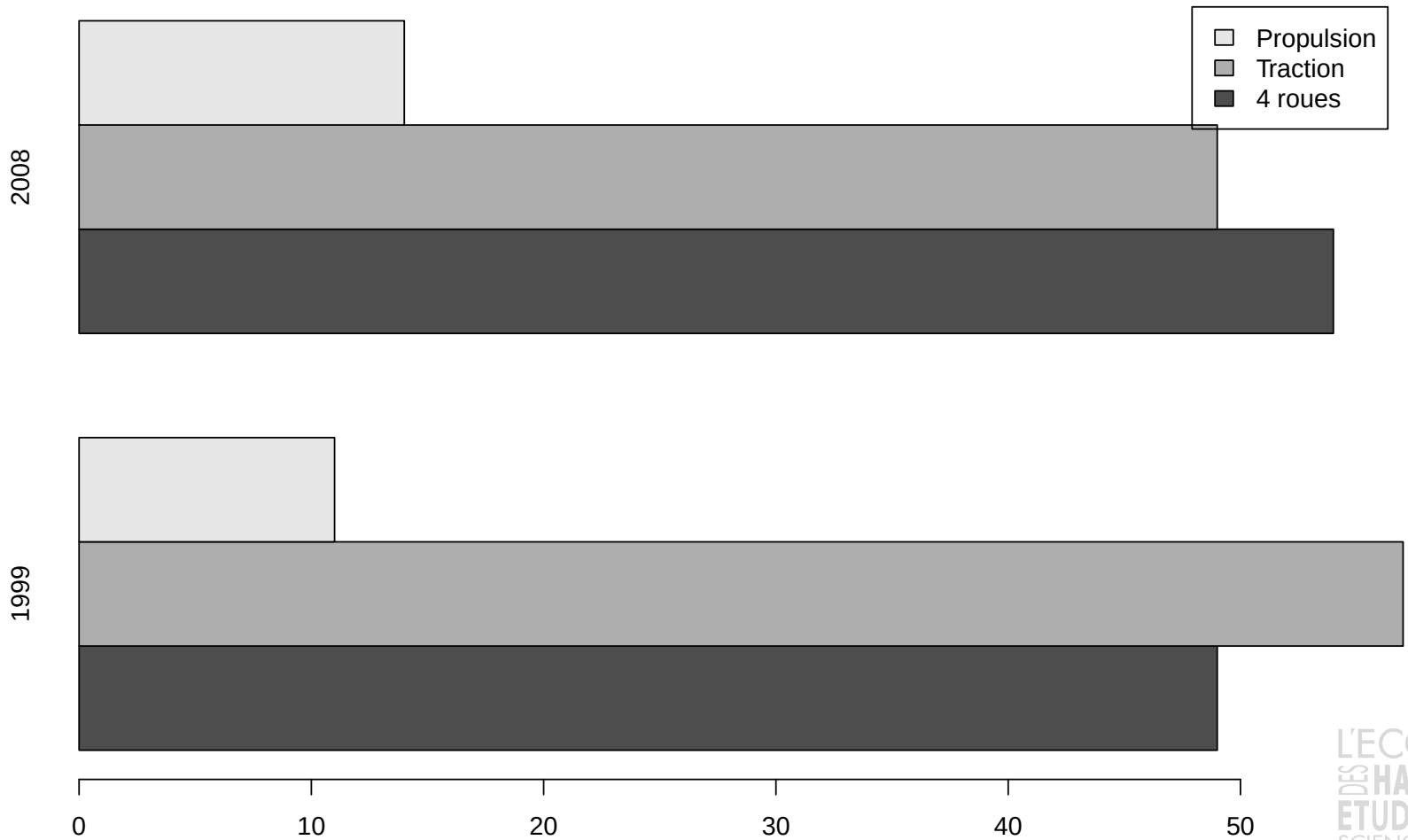
Diagrammes en bâtons et circulaires

```
pie(uni, labels = paste0(lab, "\n", uni)
    , init.angle = 90, clockwise = TRUE)
```



Diagrammes en bâtons et circulaires

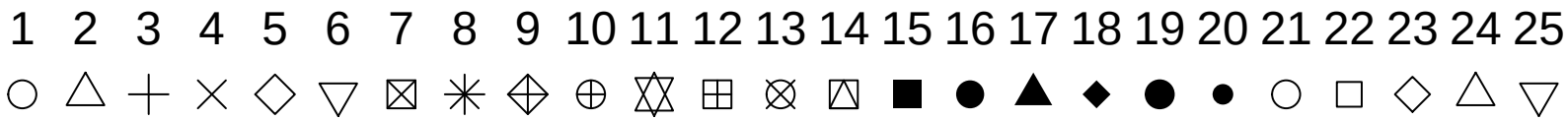
```
bi <- table(mpg$drv, mpg$year)
barplot(bi, horiz = TRUE, beside = TRUE, legend.text = lab)
```



Couleur, forme et taille des objets

Plusieurs paramètres permettent de modifier la couleur, la forme ou la taille des éléments qui composent un graphique :

- `pch` : entier ou caractère spécial indiquant la forme des points à représenter.



- `col` : valeur indiquant la couleur du contour des formes utilisées. Peut être un entier (recyclé au-delà de 8), un nom ou un code RGB hexadécimal (du type `"#FF1111"`).

Pour certaines formes (`pch` entre 21 et 25), il est également possible de modifier la couleur de remplissage avec `bg`.

Couleur, forme et taille des objets

Remarque : la palette de couleurs accessibles en utilisant des entiers est réduite. Il est possible de l'étendre considérablement *via* la fonction `colors()`.

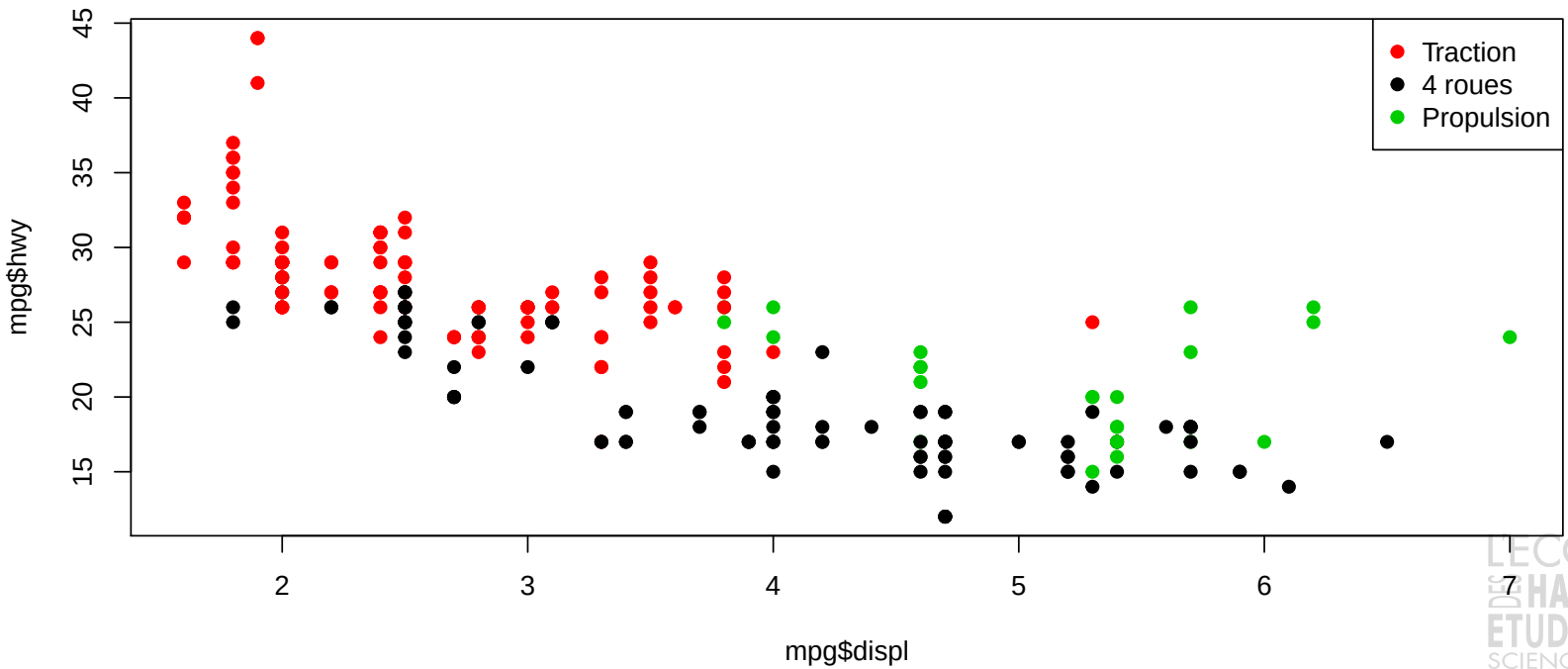
```
colors()[1:3]
## [1] "white"         "aliceblue"     "antiquewhite"
length(colors())
## [1] 657
grep("blue", colors(), value = TRUE)[1:3]
## [1] "aliceblue" "blue"      "blue1"
```

- `cex` : utilisé dans une fonction `plot()`, `cex` permet d'ajuster la taille des points qui le composent.

Couleur, forme et taille des objets

La fonction `legend()` permet d'ajouter une légende.

```
t <- factor(mpg$drv
  , labels = c("4 roues", "Traction", "Propulsion"))
plot(mpg$displ, mpg$hwy, pch = 21, col = t, bg = t)
legend("topright", legend = unique(t), pch = 21
  , col = unique(t), pt.bg = unique(t))
```



Titres, texte et axes

Les titres sont paramétrés à l'aide des fonctions suivantes :
- `main` pour ajouter le titre principal ; - `xlab` et `ylab` pour ajouter des titres aux axes.

La fonction `text()` permet d'ajouter du texte sur le graphique en le positionnant par ses coordonnées, éventuellement avec un décalage (pour nommer des points par exemple).

Il est également possible de paramétrer les axes :
- `xlim` et `ylim` spécifient les valeurs minimales et maximales de chaque axe ; - `axis()` est une fonction qui permet d'ajouter un axe personnalisé.

Remarque Pour produire un graphique sans axe et les rajouter après, utiliser l'option `axes = FALSE` de la fonction `plot()`.

Combinaison de plusieurs graphiques

Par défaut l'utilisation de la fonction `plot()` produit un nouveau graphique.

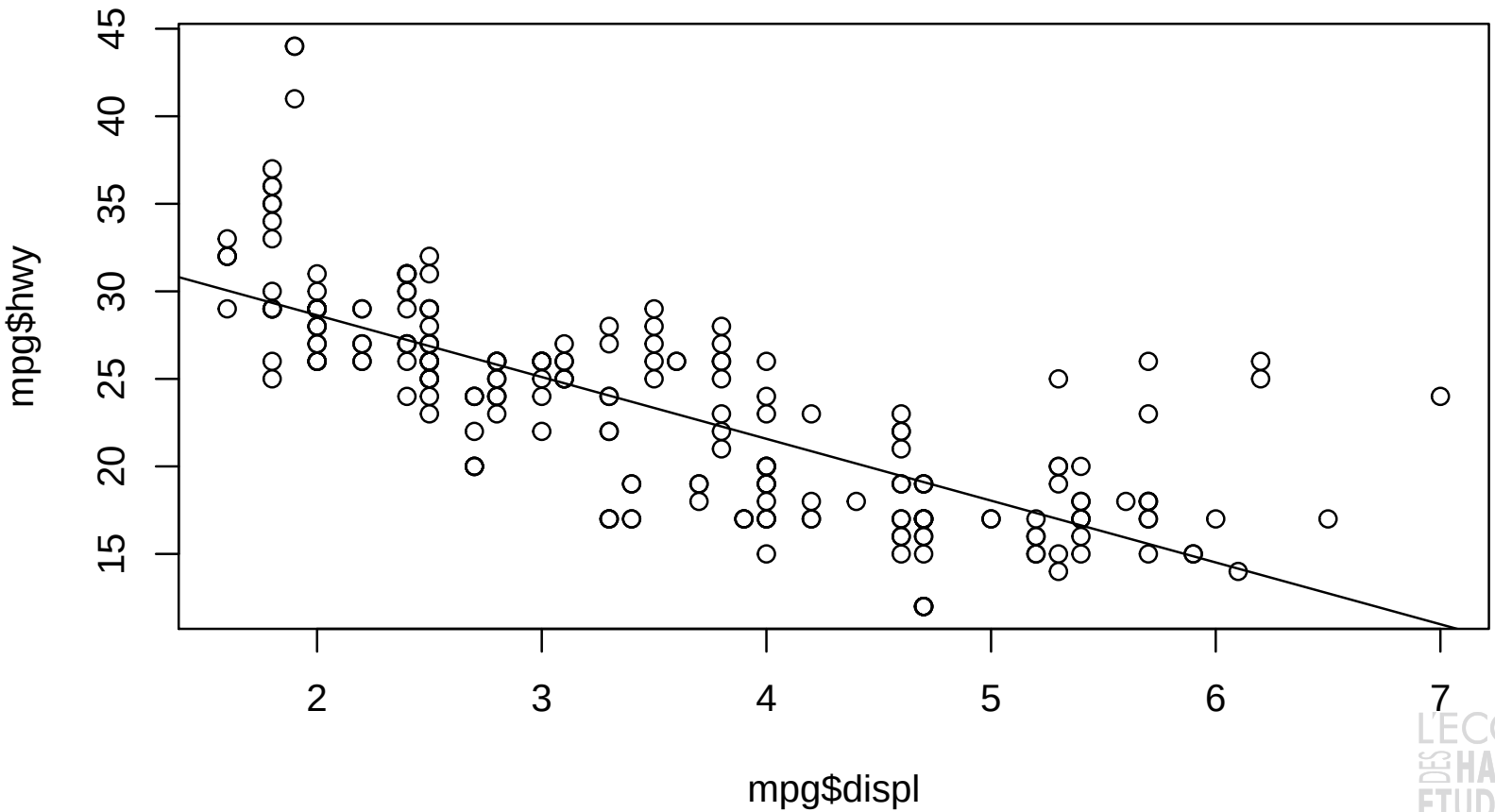
Pour superposer différents graphiques, le plus simple est de commencer par une instruction `plot()` puis de la compléter :

- avec `points()` pour ajouter des points ;
- avec `lines()` pour ajouter des lignes ;
- avec `abline()` pour ajouter des lignes d'après une équation ;
- avec `curve()` pour ajouter des courbes d'après une équation.

Exemple Ajout d'une droite de régression au graphique de `hwy` par `displ`.

Combinaison de plusieurs graphiques

```
reg <- lm(hwy ~ displ, data = mpg)
plot(mpg$displ, mpg$hwy)
abline(a = reg$coefficients[1], b = reg$coefficients[2])
```



Paramètres généraux et disposition (1)

Utilisée en dehors de la fonction `plot()`, la fonction `par()` permet de définir l'ensemble des paramètres graphiques globaux.

Ses mots-clés les plus importants sont :

- `mfrow` : permet de disposer plusieurs graphiques côte-à-côte.

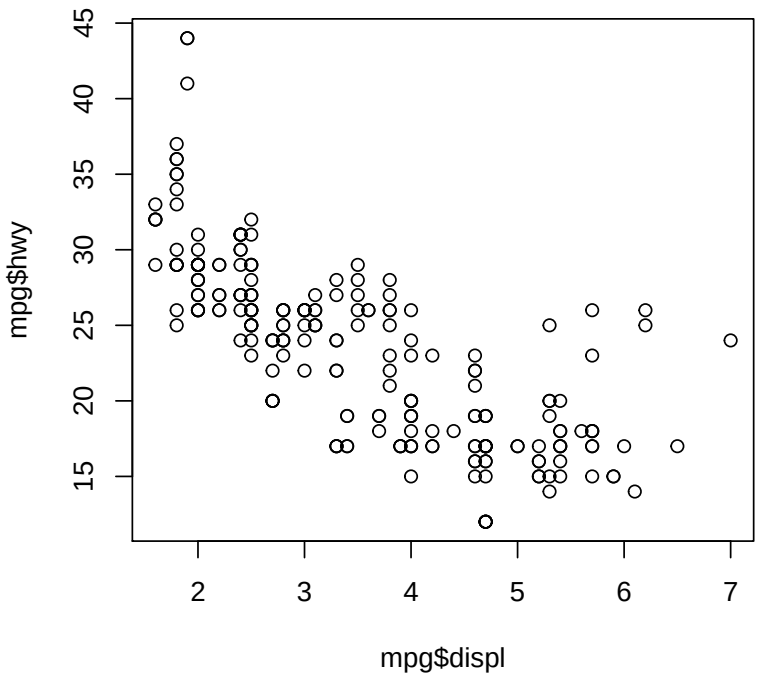
```
par(mfrow = c(1, 2)) # 1 ligne et 2 colonnes
par(mfrow = c(3, 2)) # 3 lignes et 2 colonnes
par(mfrow = c(1, 1)) # 1 ligne et 1 colonne
```

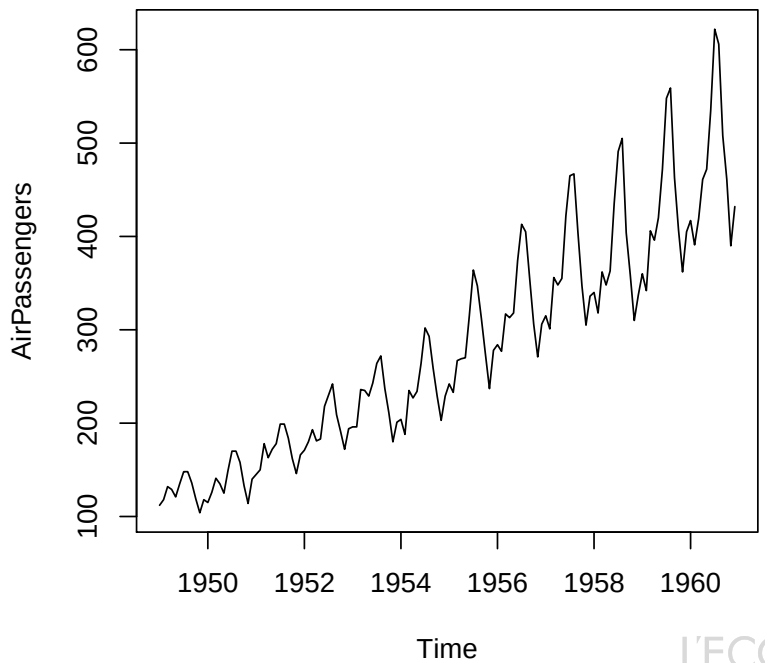
- `cex` : coefficient multiplicatif pour modifier la taille de l'ensemble des textes et symboles utilisés dans les graphiques (1 par défaut).

Pour en savoir plus La page d'aide de la fonction `par()` détaille toutes ces options.

Paramètres généraux et disposition (2)

```
par(mfrow = c(1, 2))
plot(mpg$displ, mpg$hwy)
plot(AirPassengers)
```





Exportation

Pour exporter des graphiques depuis R, la démarche consiste à rediriger le flux de production du graphiques vers un fichier à l'aide d'une fonction du *package* `grDevices`. Par exemple :

```
png("monGraphique.png", width = 10, height = 8
    , unit = "cm", res = 600)
plot(mpg$displ, mpg$hwy)
dev.off()
```

Dans ce contexte, les fonctions les plus utiles sont : `png()`, `jpeg()` et `pdf()`. En particulier, `pdf()` permet de conserver le caractère vectoriel des graphiques dans R.

Remarque Les graphiques peuvent également facilement être exportés depuis Rstudio en utilisant les menus spécialement conçus à cet effet.

L'implémentation d'une grammaire des graphiques

Le *package* `graphics` permet de réaliser une grande quantité de graphiques mais présente deux limites importantes :

- les fonctions qui le composent forment une casuistique complexe ;
- il n'est pas possible d'inventer de nouvelles représentations à partir des fonctions existantes.

Ce sont ces limites que tente de dépasser le *package* `ggplot2` en implémentant une **grammaire des graphiques**

Comme les éléments du langage, les **composants élémentaires** d'un graphique doivent pouvoir être **réassemblés** pour produire de **nouvelles représentations.**

Pour aller plus loin Wilkinson L. (2005) *The Grammar of Graphics*, Springer, ggplot2: elegant graphics for data analysis

Les trois composants essentiels d'un graphique

La construction d'un graphique avec ggplot2 fait intervenir trois composants essentiels (d'après Wickham, *ibid.*, 2.3) :

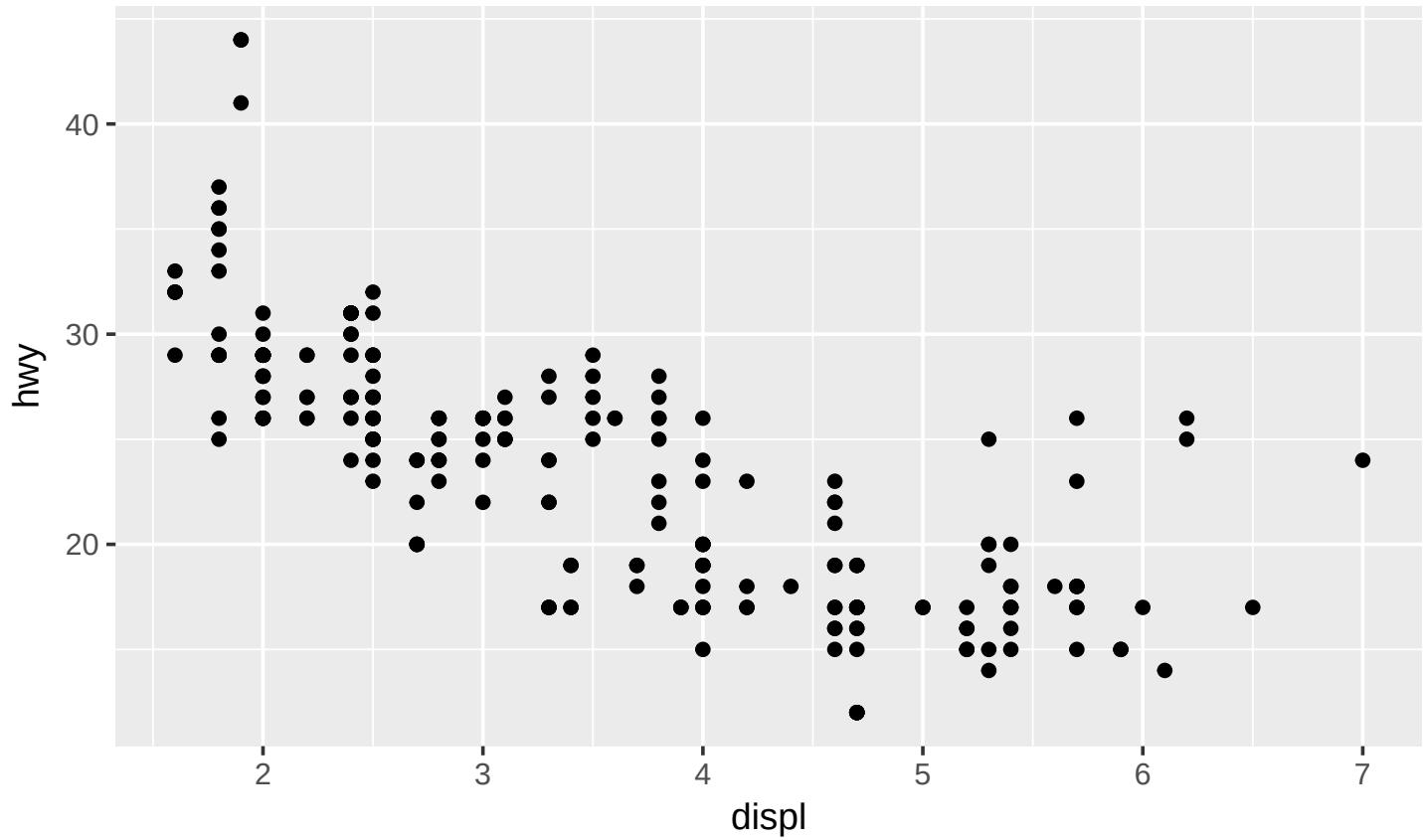
- le `data.frame` dans lequel sont stockées les données à représenter ;
- des correspondances esthétiques (*aesthetic mappings*) entre des variables et des propriétés visuelles ;
- au moins une couche (*layer*) décrivant comment représenter les observations.

Exemple *Miles per gallon* sur l'autoroute en fonction de la cylindrée.

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point()
```

Les trois composants essentiels d'un graphique

```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +
  geom_point()
```

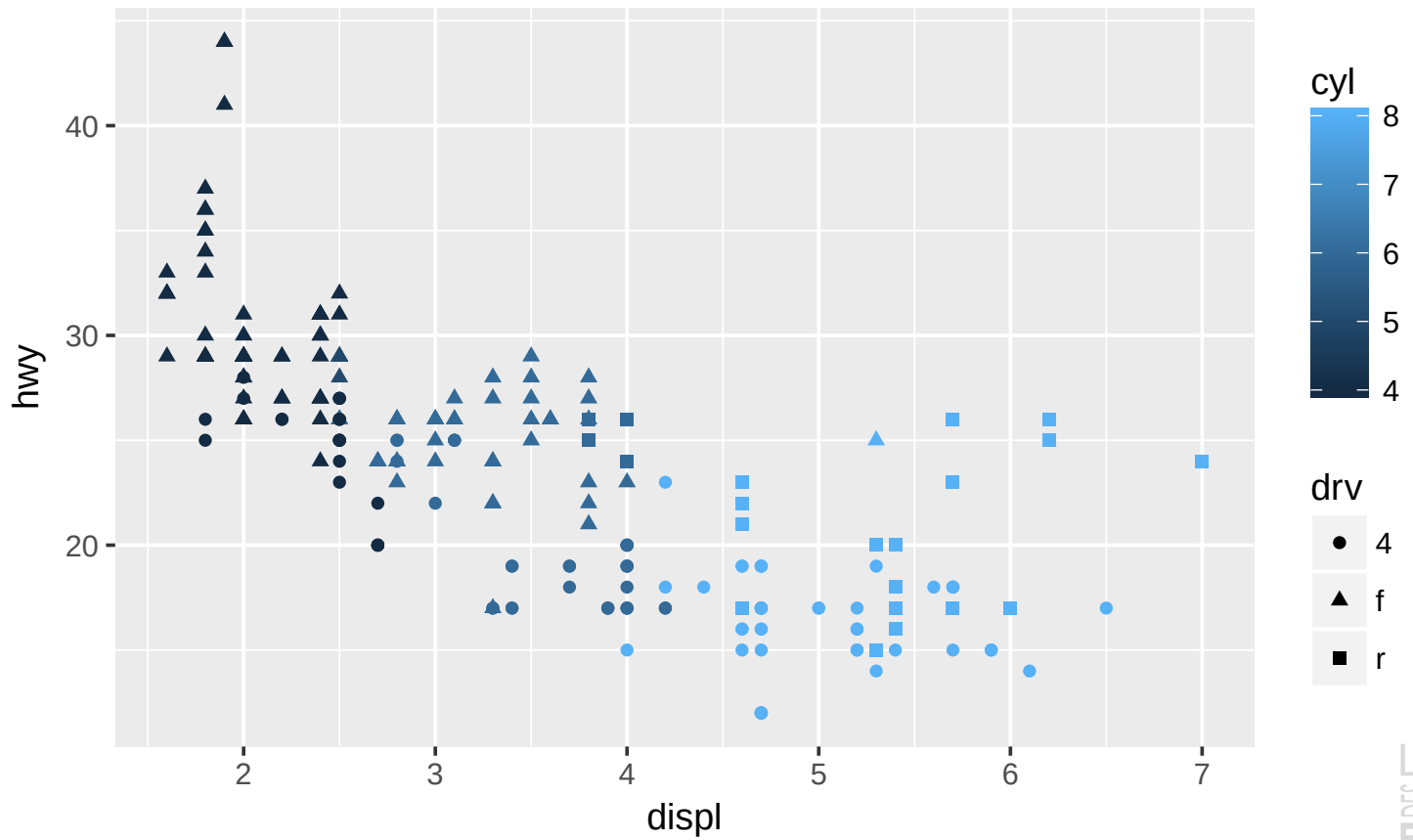


Couleur, forme et taille des objets

Pour faire varier l'aspect visuel des éléments représentés en fonction de données, il suffit d'**associer une variable à l'attribut de couleur, de taille ou de forme** dans la fonction `aes()`.

Couleur, forme et taille des objets

```
ggplot(mpg, aes(displ, hwy, colour = cyl, shape = drv)) +
  geom_point()
```



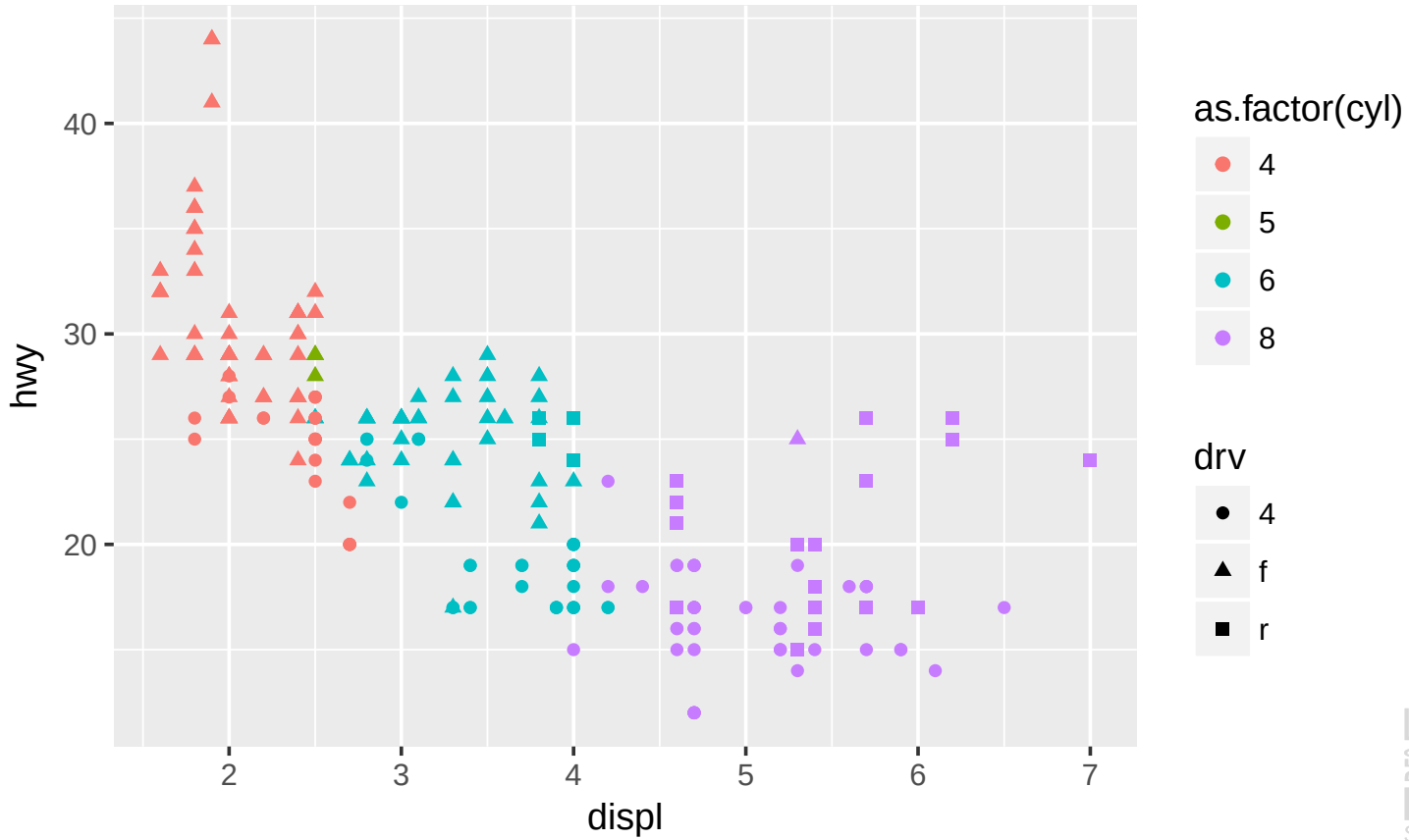
Couleur, forme et taille des objets

Pour faire varier l'aspect visuel des éléments représentés en fonction de données, il suffit d'**associer une variable à l'attribut de couleur, de taille ou de forme** dans la fonction `aes()`.

Selon le type des variables utilisées pour les correspondances esthétiques, **les échelles sont continues ou discrètes**.

Couleur, forme et taille des objets

```
ggplot(mpg, aes(displ, hwy, colour = as.factor(cyl)
  , shape = drv)) +
  geom_point()
```



Couleur, forme et taille des objets

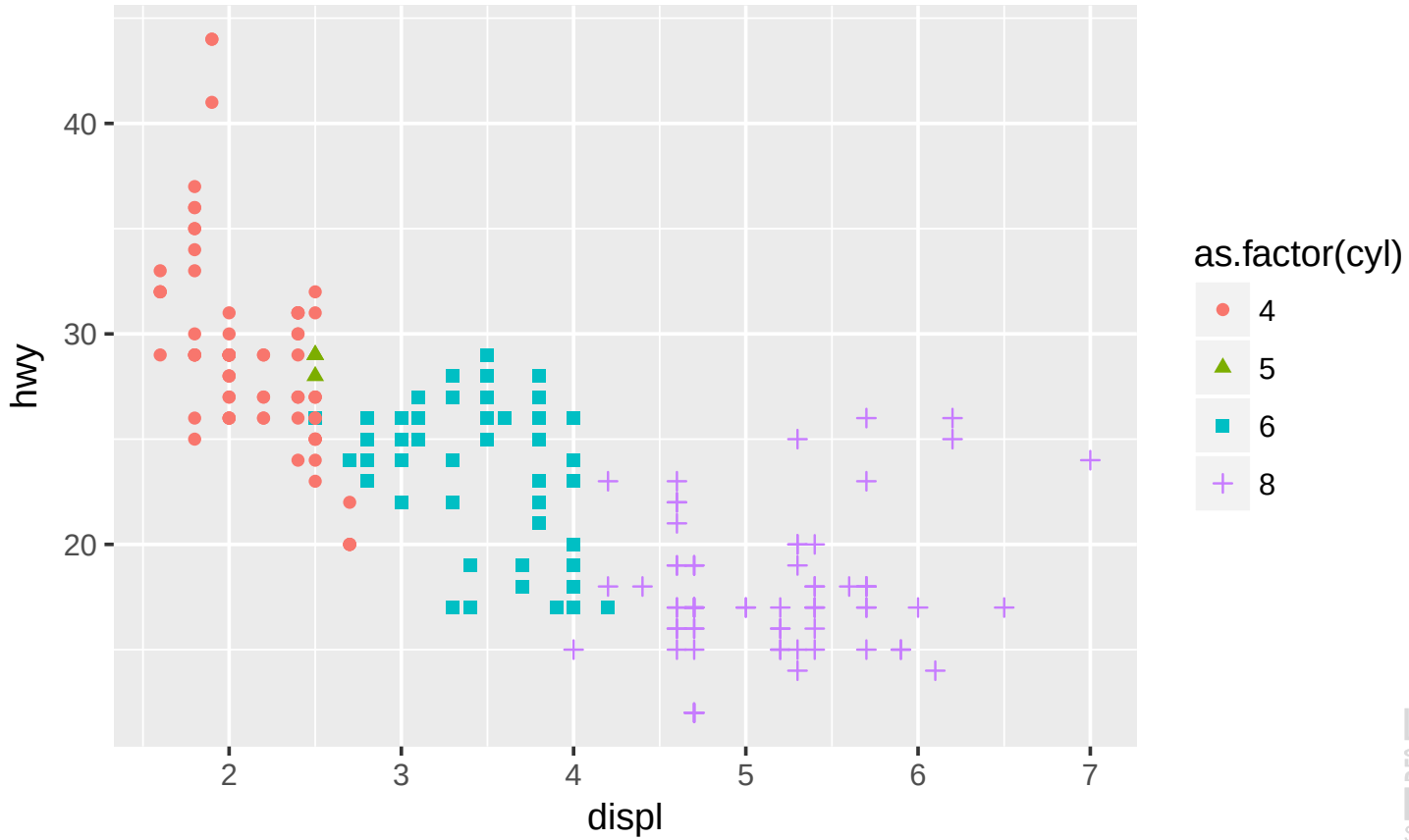
Pour faire varier l'aspect visuel des éléments représentés en fonction de données, il suffit d'**associer une variable à l'attribut de couleur, de taille ou de forme** dans la fonction `aes()`.

Selon le type des variables utilisées pour les correspondances esthétiques, **les échelles sont continues ou discrètes**.

Quand la même variable est utilisée dans plusieurs correspondances esthétiques, **les échelles qui lui correspondent sont fusionnées**.

Couleur, forme et taille des objets

```
ggplot(mpg, aes(displ, hwy, colour = as.factor(cyl)
  , shape = as.factor(cyl))) +
  geom_point()
```



Couleur, forme et taille des objets

Pour faire varier l'aspect visuel des éléments représentés en fonction de données, il suffit d'**associer une variable à l'attribut de couleur, de taille ou de forme** dans la fonction `aes()`.

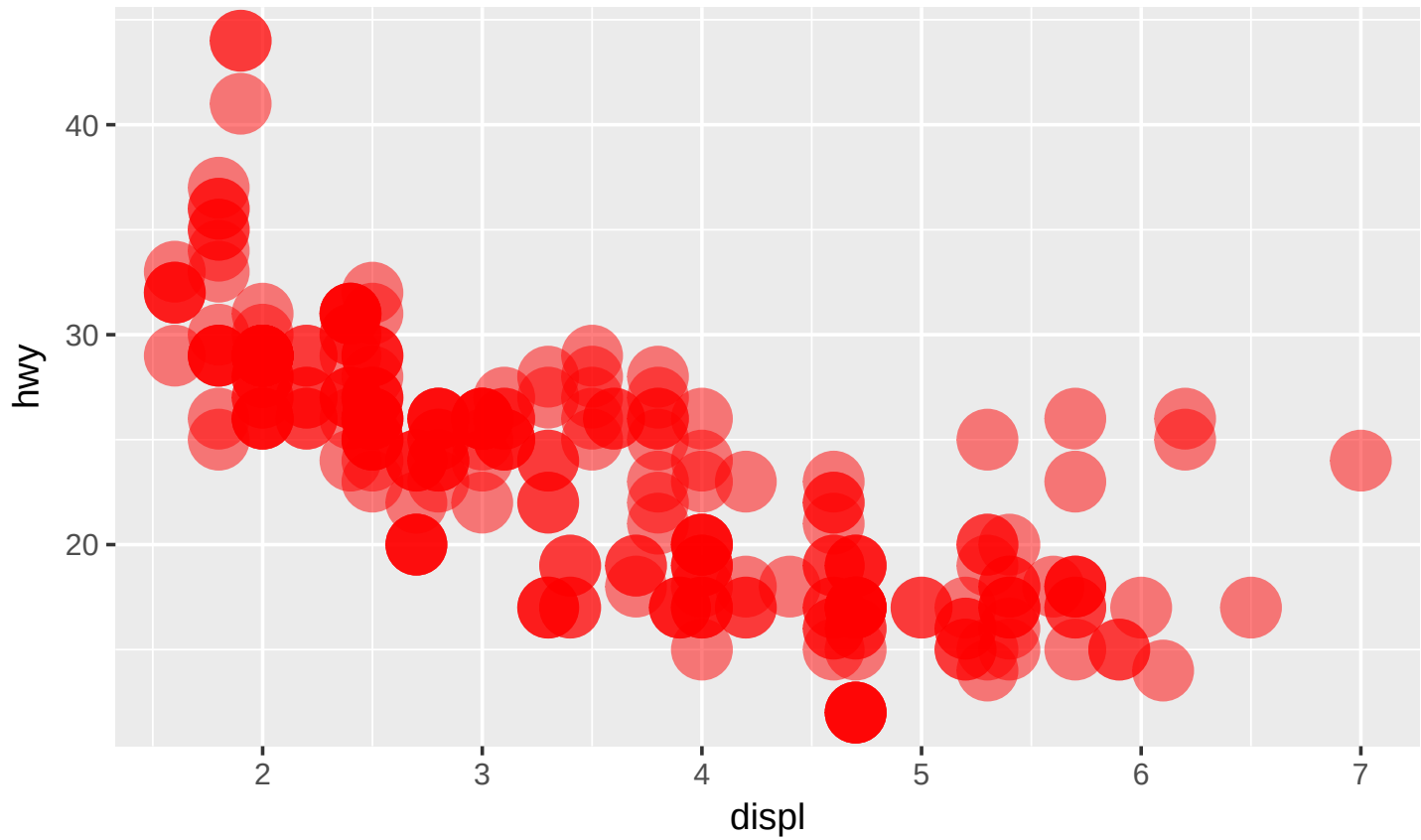
Selon le type des variables utilisées pour les correspondances esthétiques, **les échelles sont continues ou discrètes**.

Quand la même variable est utilisée dans plusieurs correspondances esthétiques, **les échelles qui lui correspondent sont fusionnées**.

Au-delà des correspondances esthétiques dans la fonction `aes()`, **l'aspect visuel peut être ajusté directement dans la fonction `geom_*`**.

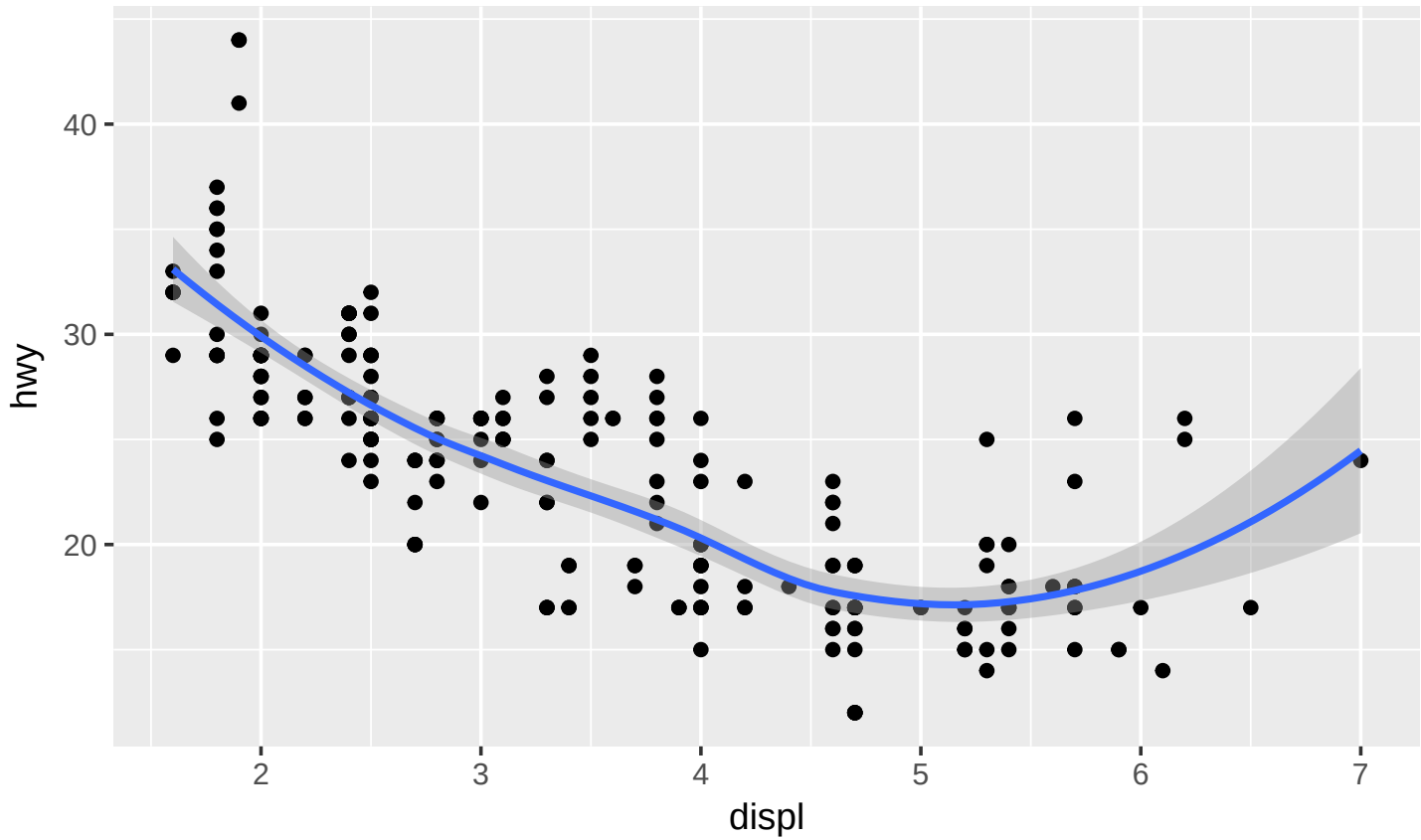
Couleur, forme et taille des objets

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(colour = "red", size = 8, alpha = 0.5)
```



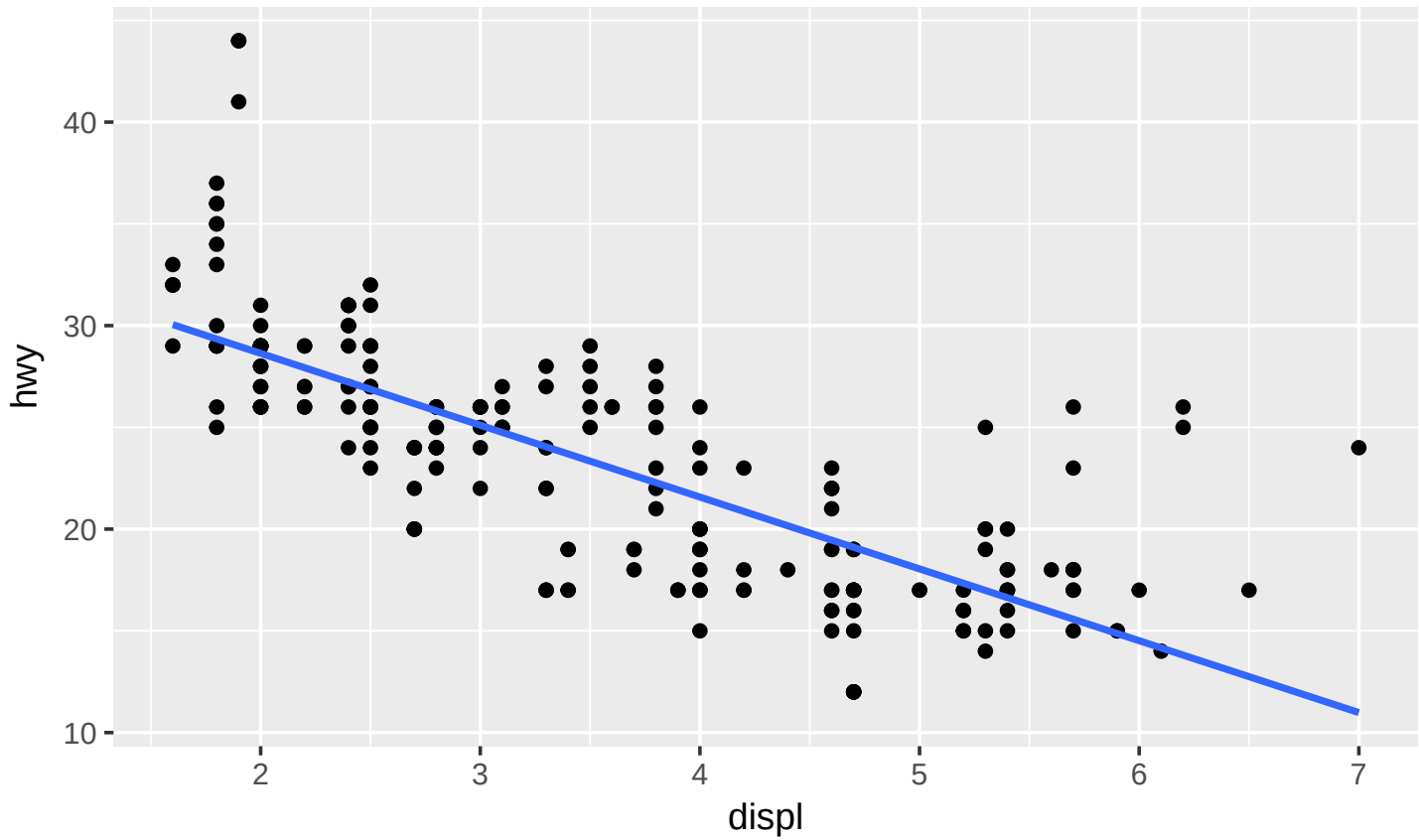
Combinaison de plusieurs graphiques

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() + geom_smooth()
## `geom_smooth()` using method = 'loess'
```



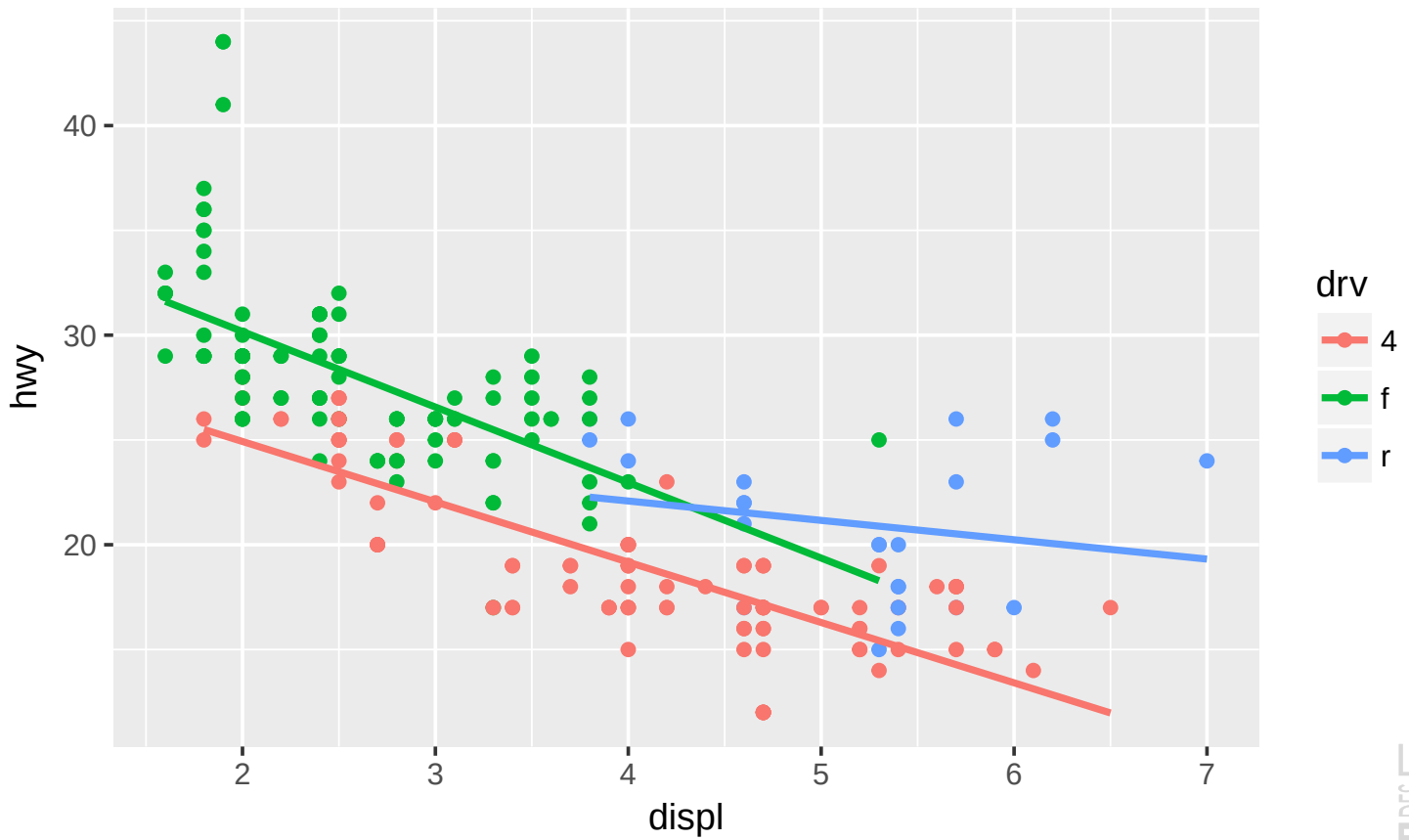
Combinaison de plusieurs graphiques

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() + geom_smooth(method = "lm", se = FALSE)
```



Combinaison de plusieurs graphiques

```
ggplot(mpg, aes(displ, hwy, colour = drv)) +
  geom_point() + geom_smooth(method = "lm", se = FALSE)
```



Le fonctionnement en « couches » de ggplot2

La construction d'un graphique dans ggplot2 repose sur la superposition de couches (*layer*) **conçues indépendamment** mais **réconciliées en fin d'opération**.

Chaque couche est composée de cinq éléments :

- un `data.frame` (`data`) ;
- une ou plusieurs correspondances esthétiques (`mapping`) ;
- une transformation statistique (`stat`) ;
- un objet géométrique (`geom`) ;
- un paramètre d'ajustement de la position (`position`).

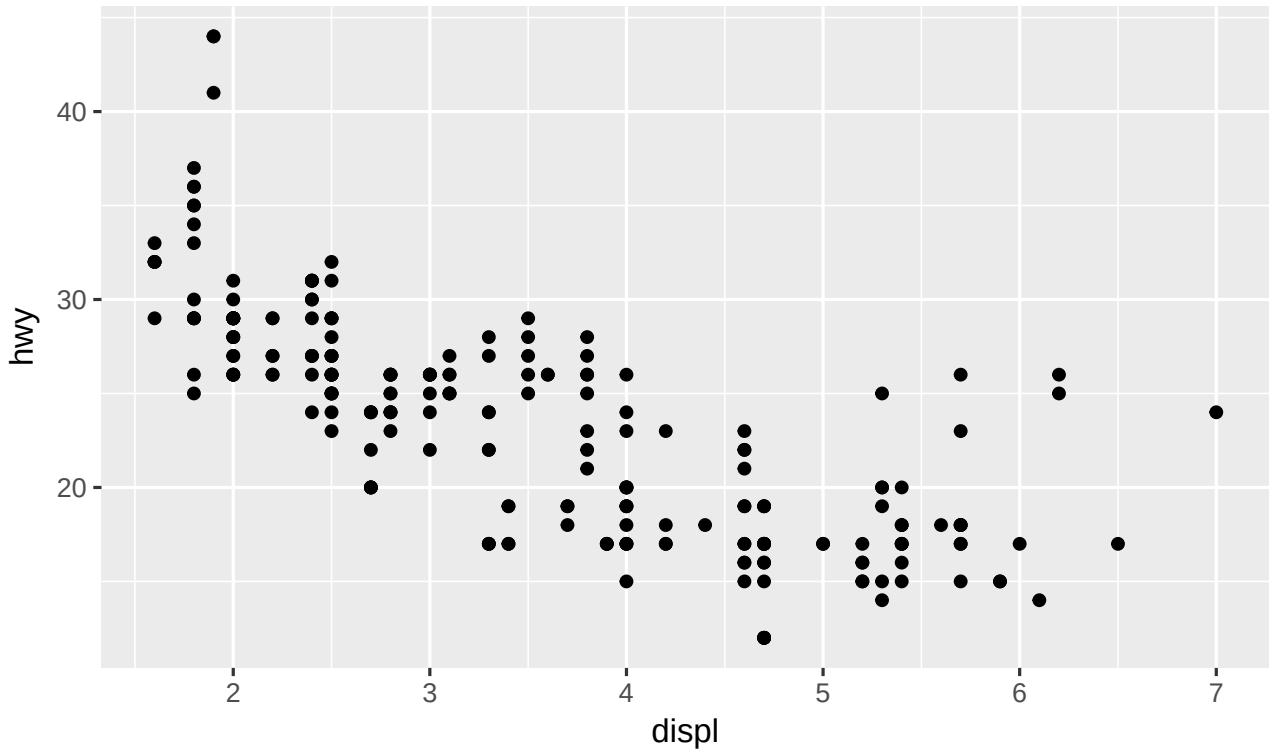
C'est la **fonction `layer()`** qui articule ces cinq éléments.

Les fonctions `geom_*` vues précédemment sont des appels pré-paramétrées de `layer()`.

Le fonctionnement en « couches » de ggplot2

Un graphique à une couche

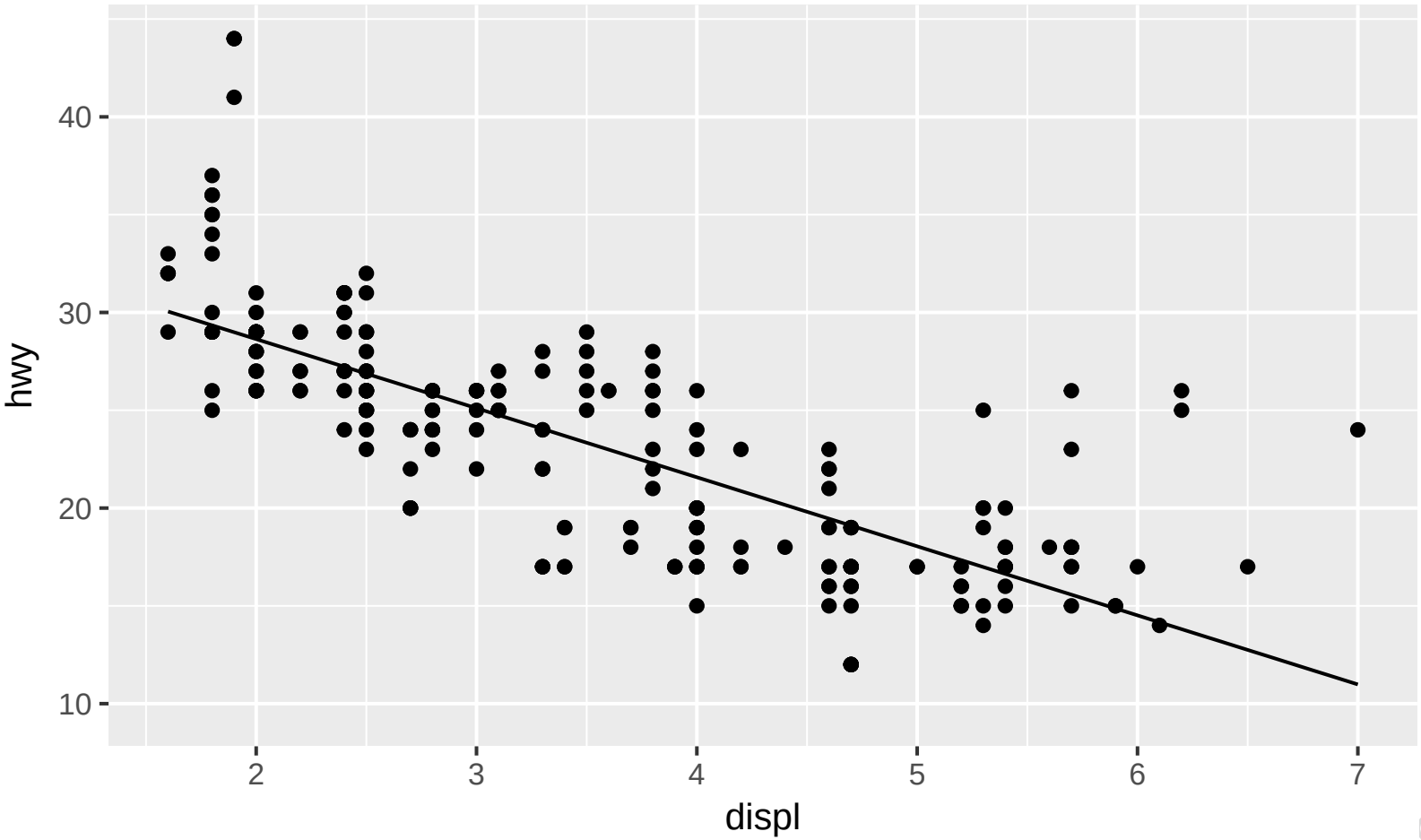
```
ggplot() + layer(
  data = mpg, mapping = aes(displ, hwy), stat = "identity"
  , geom = "point", position = "identity"
)
```



Un graphique à deux couches

```
ggplot() + layer(
  data = mpg, mapping = aes(displ, hwy), stat = "identity"
  , geom = "point", position = "identity"
) + layer(
  data = mpg, mapping = aes(displ, hwy), stat = "smooth"
  , geom = "line", position = "identity"
  , params = list(method = "lm", formula = y ~ x)
)
```

Un graphique à deux couches



Mise en facteur dans `ggplot()` de `data` et `mapping`

```
ggplot(data = mpg, mapping = aes(displ, hwy)) + layer(
  stat = "identity", geom = "point", position = "identity"
) + layer(
  stat = "smooth", geom = "line", position = "identity"
  , params = list(method = "lm", formula = y ~ x)
)
```

Remplacement de `layer()` par des alias pré-paramétrés

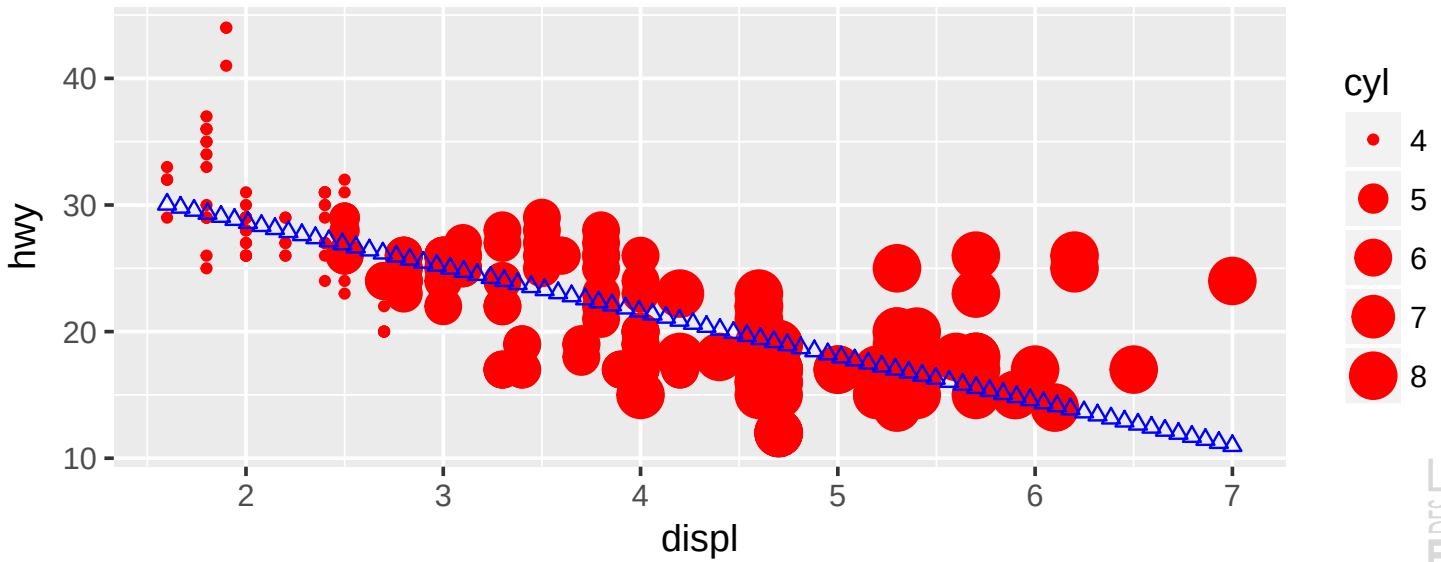
```
ggplot(data = mpg, mapping = aes(displ, hwy)) +
  geom_point() + geom_smooth(method = "lm", se = FALSE)
```

```
ggplot(data = mpg, mapping = aes(displ, hwy)) +
  geom_point() + stat_smooth(method = "lm", se = FALSE)
```

Le fonctionnement en « couches » de ggplot2

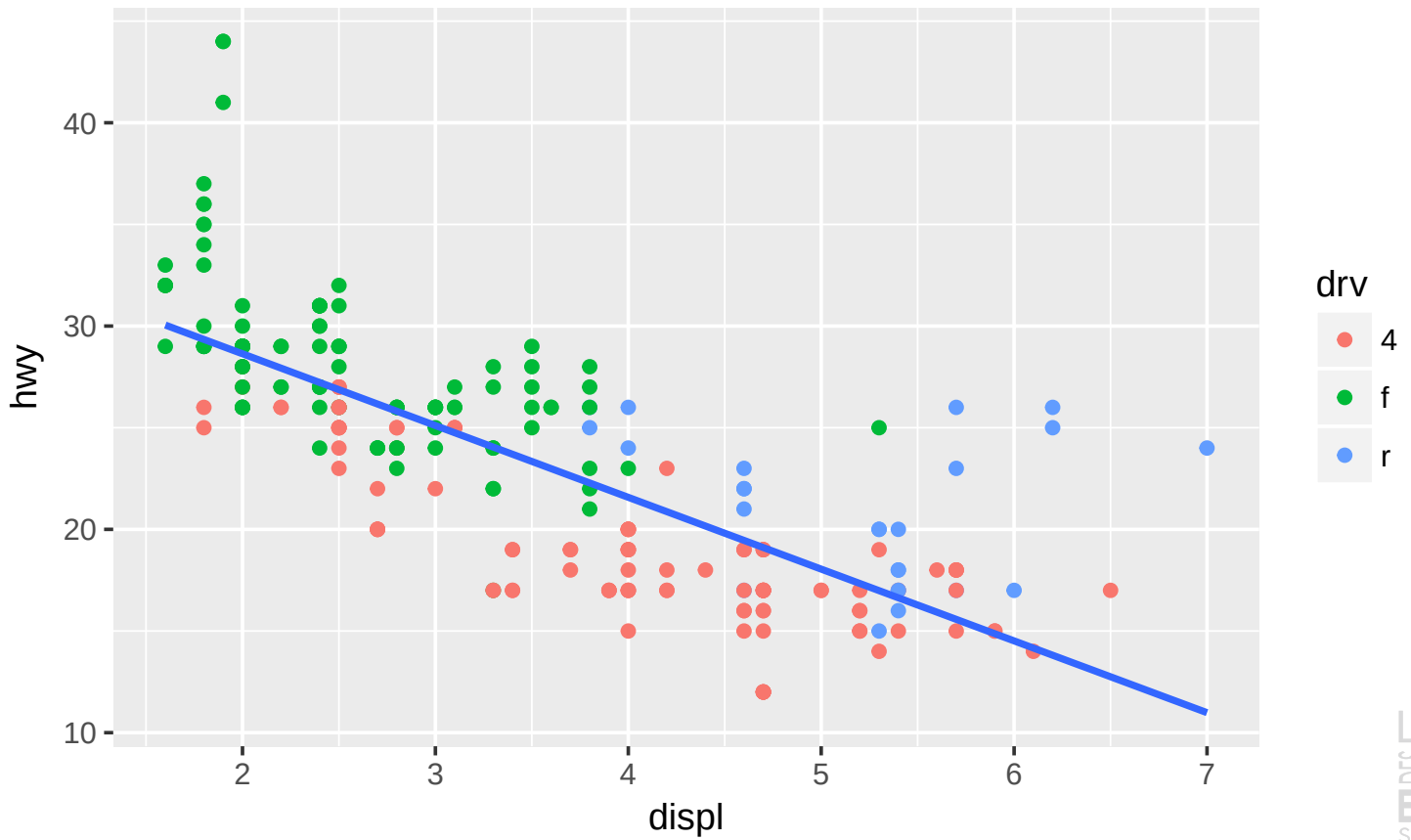
À chaque fonction `geom_*()` est associée un paramètre `stat` par défaut, et à chaque fonction `stat_*()` un `geom` par défaut.

```
ggplot(data = mpg, mapping = aes(displ, hwy)) +
  geom_point(colour = "red", aes(size = cyl)) +
  stat_smooth(geom = "point", method = "lm", se = FALSE
    , colour = "blue", shape = 2)
```



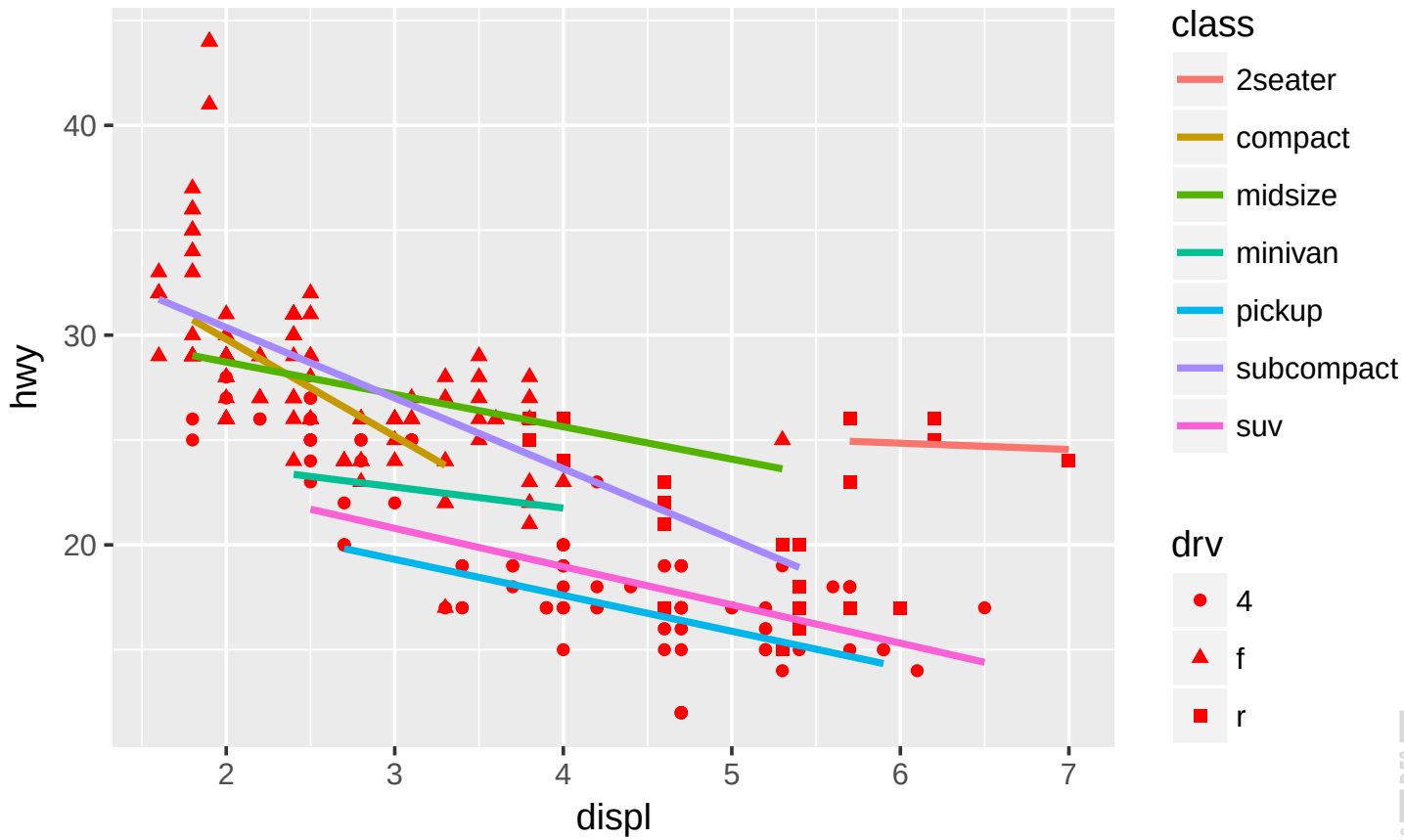
Le fonctionnement en « couches » de ggplot2

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(colour = drv)) +
  stat_smooth(method = "lm", se = FALSE)
```



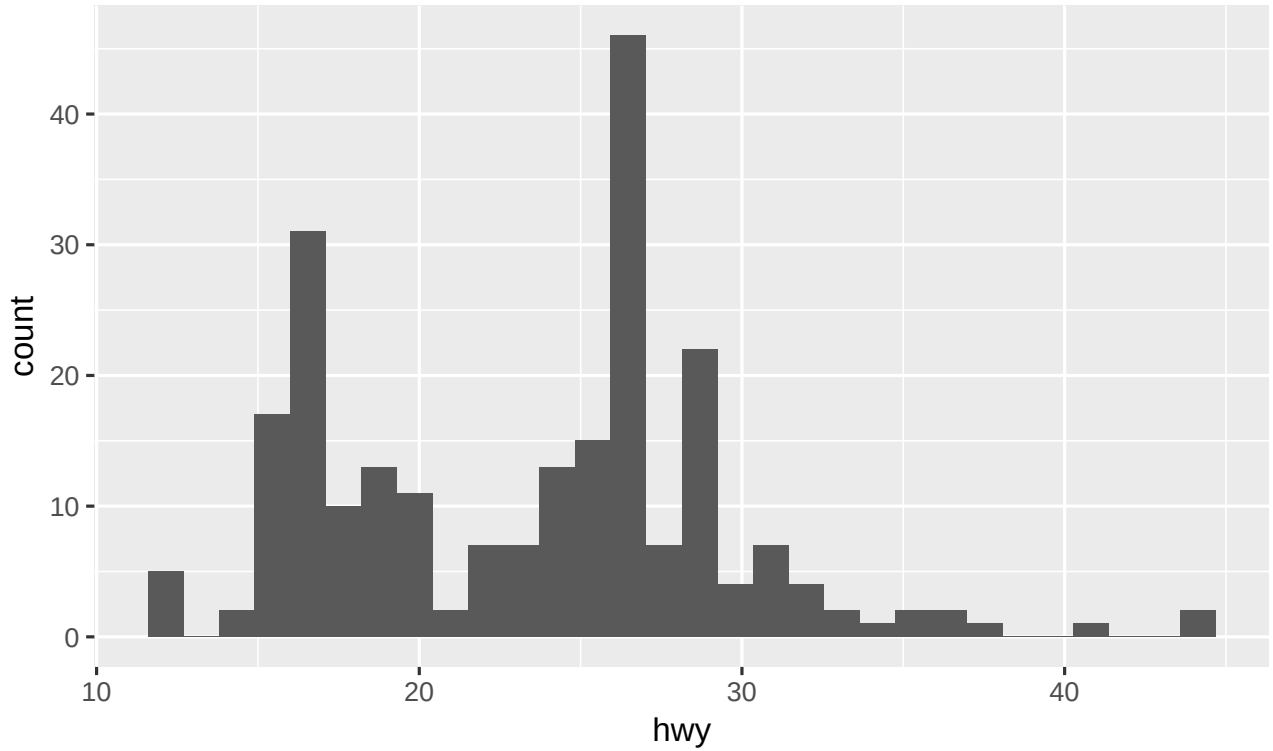
Le fonctionnement en « couches » de ggplot2

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(shape = drv), colour = "red") +
  stat_smooth(aes(colour = class), method = "lm", se = FALSE)
```



Histogrammes et densités

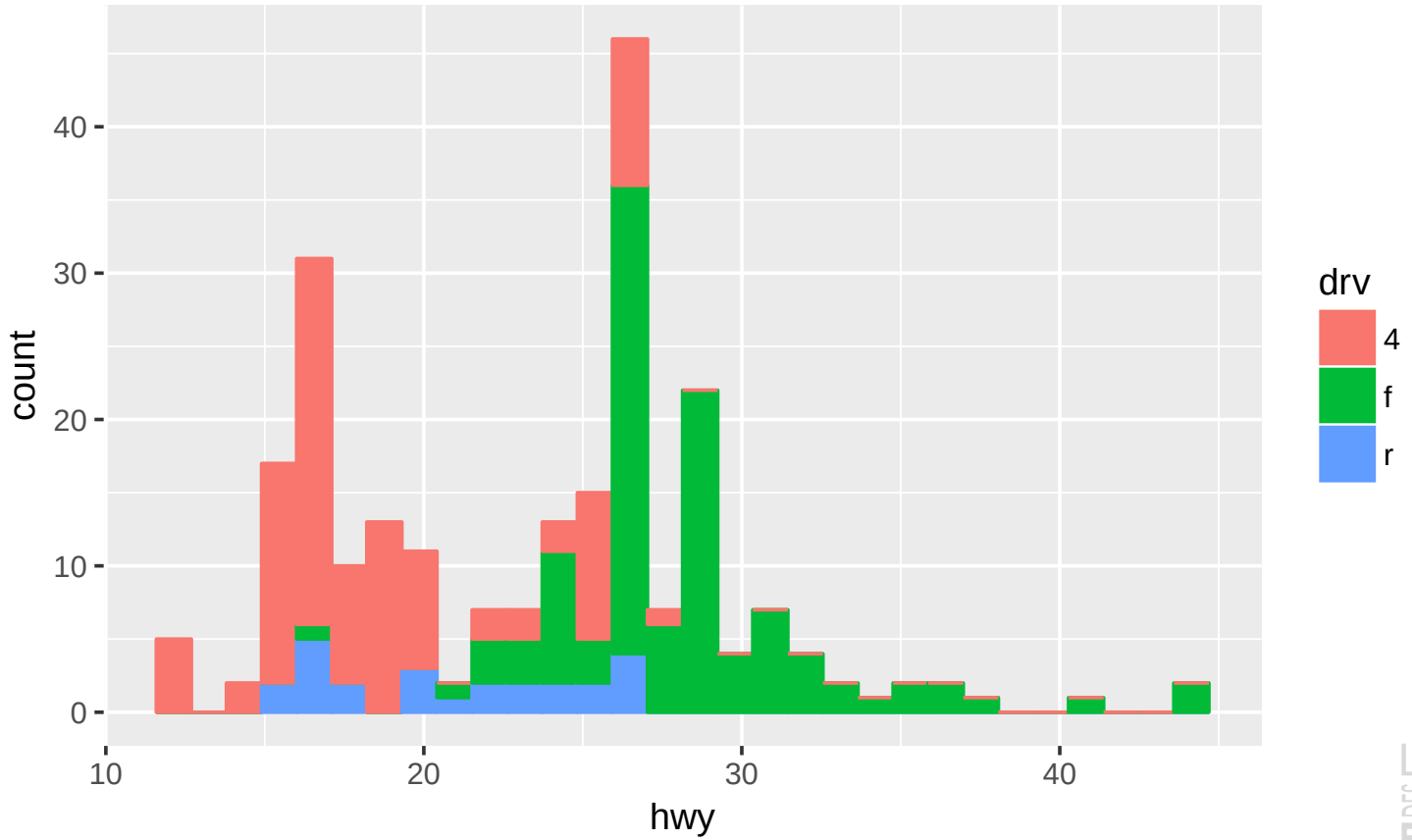
```
ggplot(mpg, aes(hwy)) + geom_histogram()
```



Remarque Le positionnement des classes des histogrammes semble perturbé dans les dernières versions de `ggplot2` : le paramètre `boundary` permet de corriger ce problème (*cf.* cette discussion).

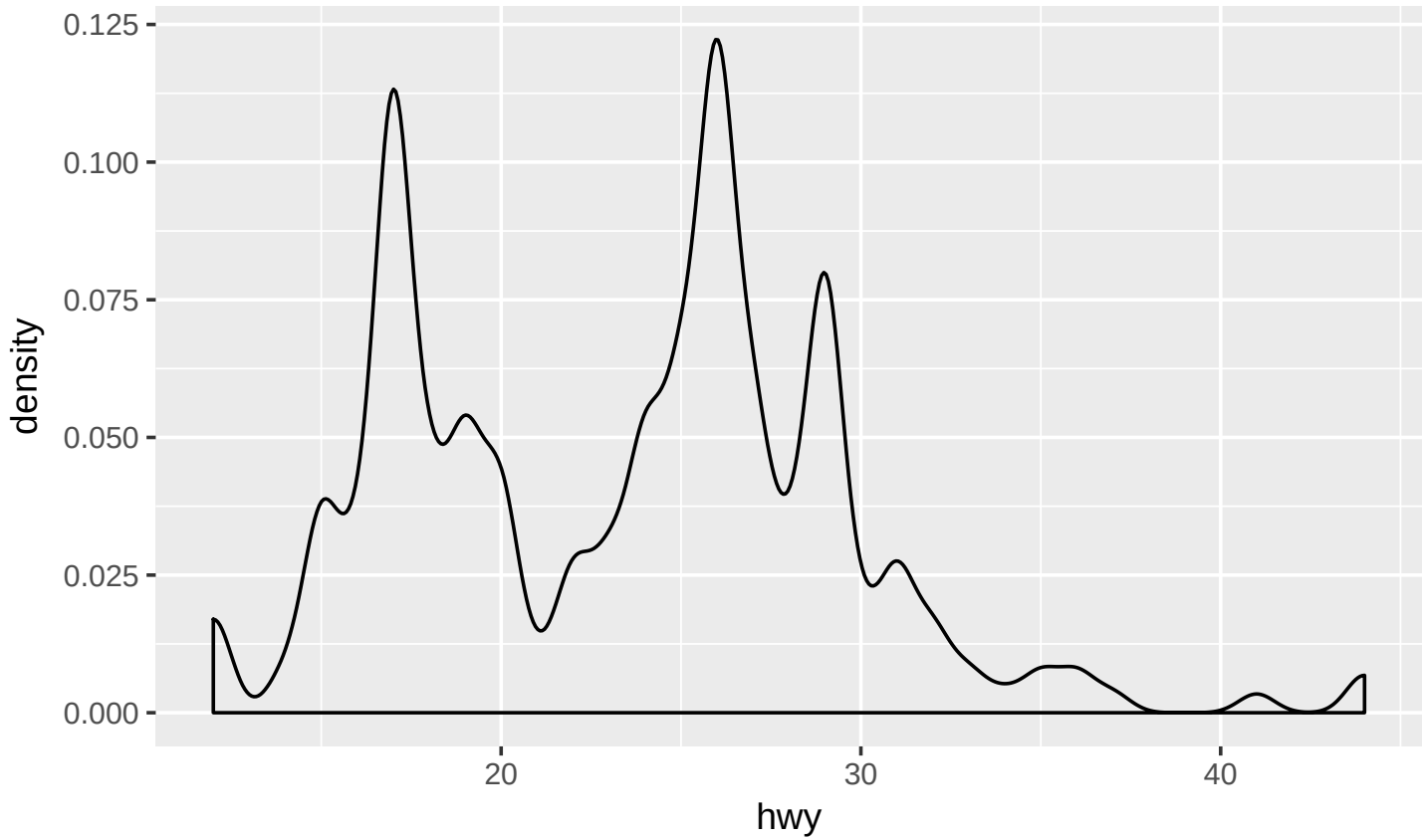
Histogrammes et densités

```
ggplot(mpg, aes(hwy, colour = drv, fill = drv)) +
  geom_histogram()
```



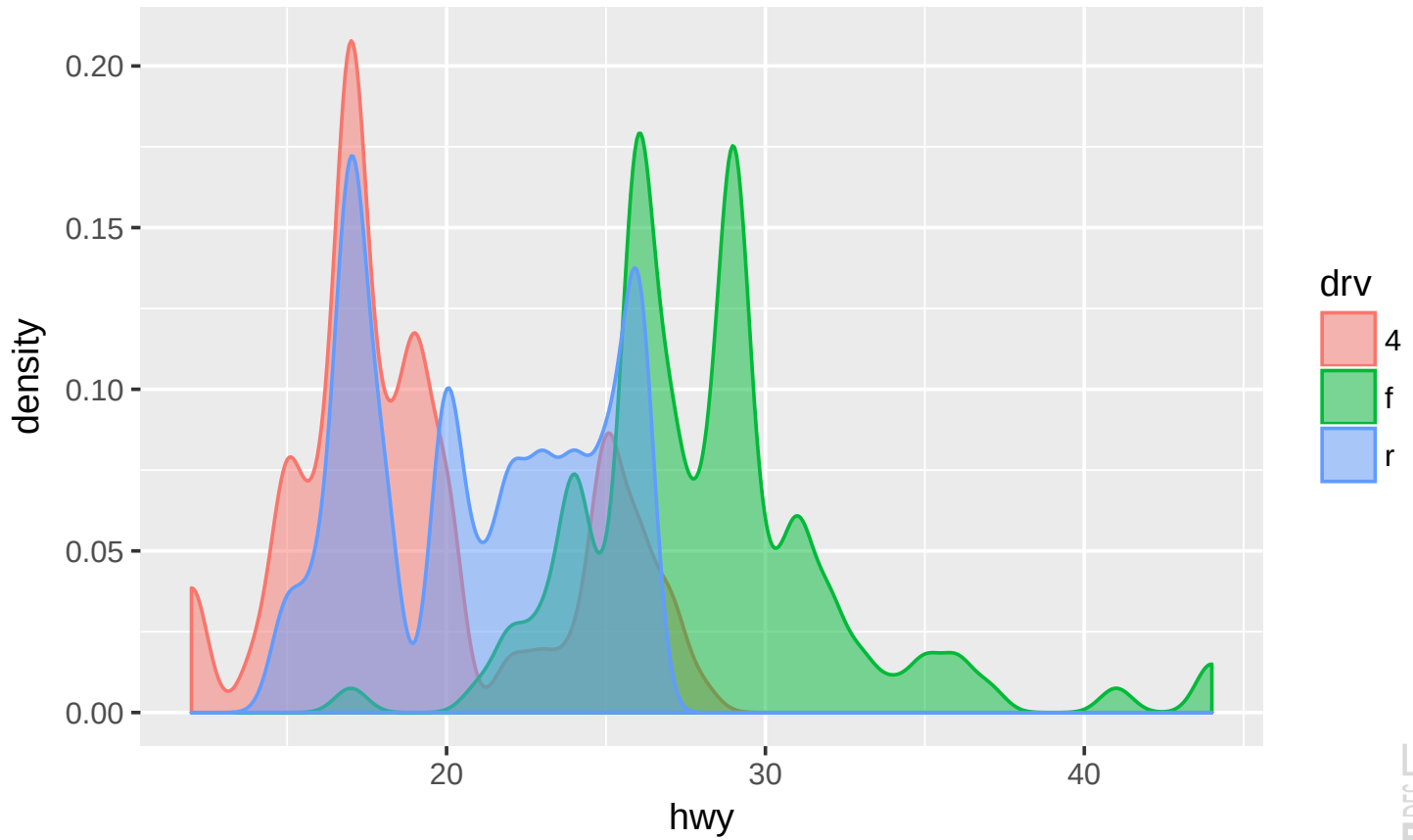
Histogrammes et densités

```
ggplot(mpg, aes(hwy)) + geom_density(bw = 0.5)
```



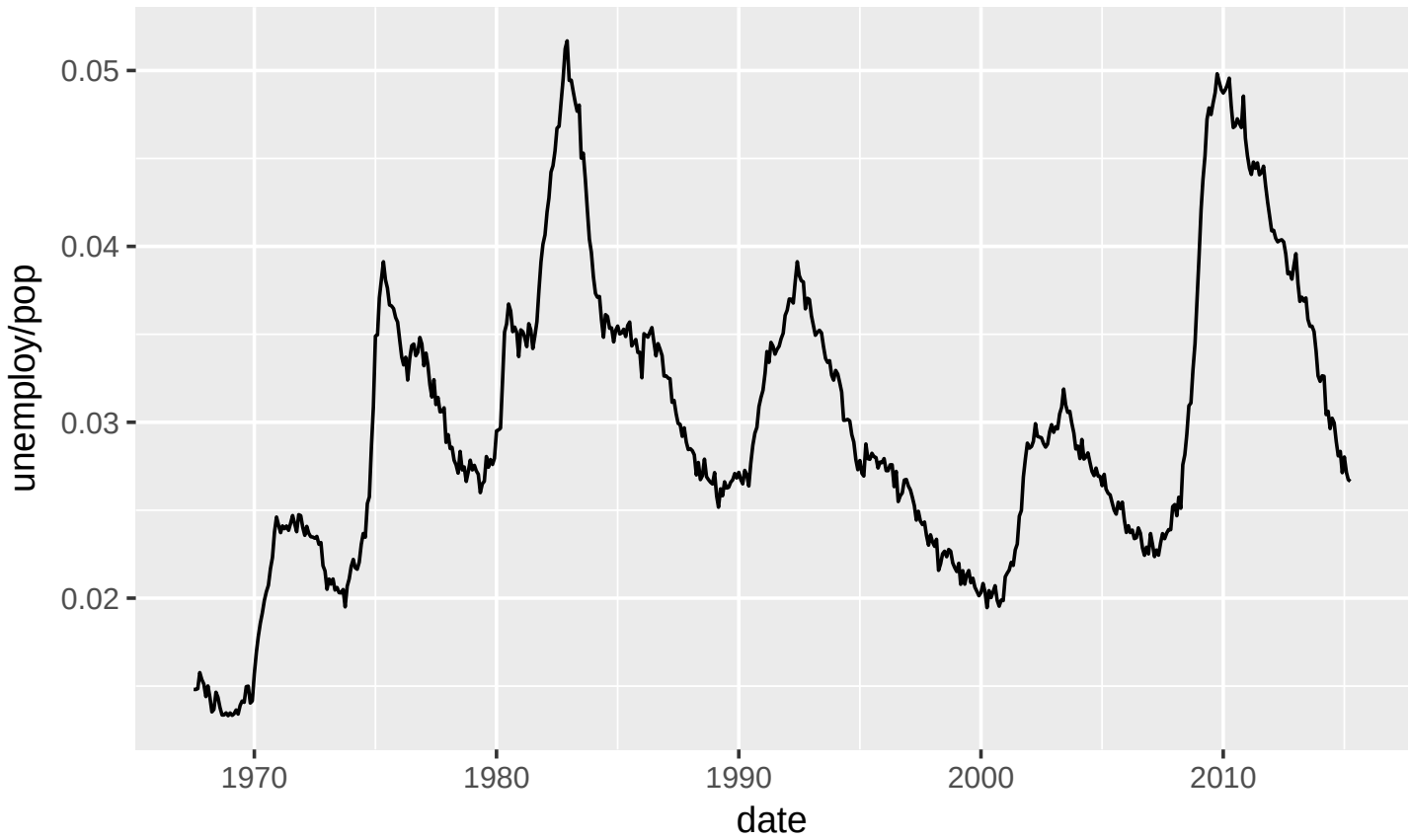
Histogrammes et densités

```
ggplot(mpg, aes(hwy, colour = drv, fill = drv)) +
  geom_density(bw = 0.5, alpha = 0.5)
```



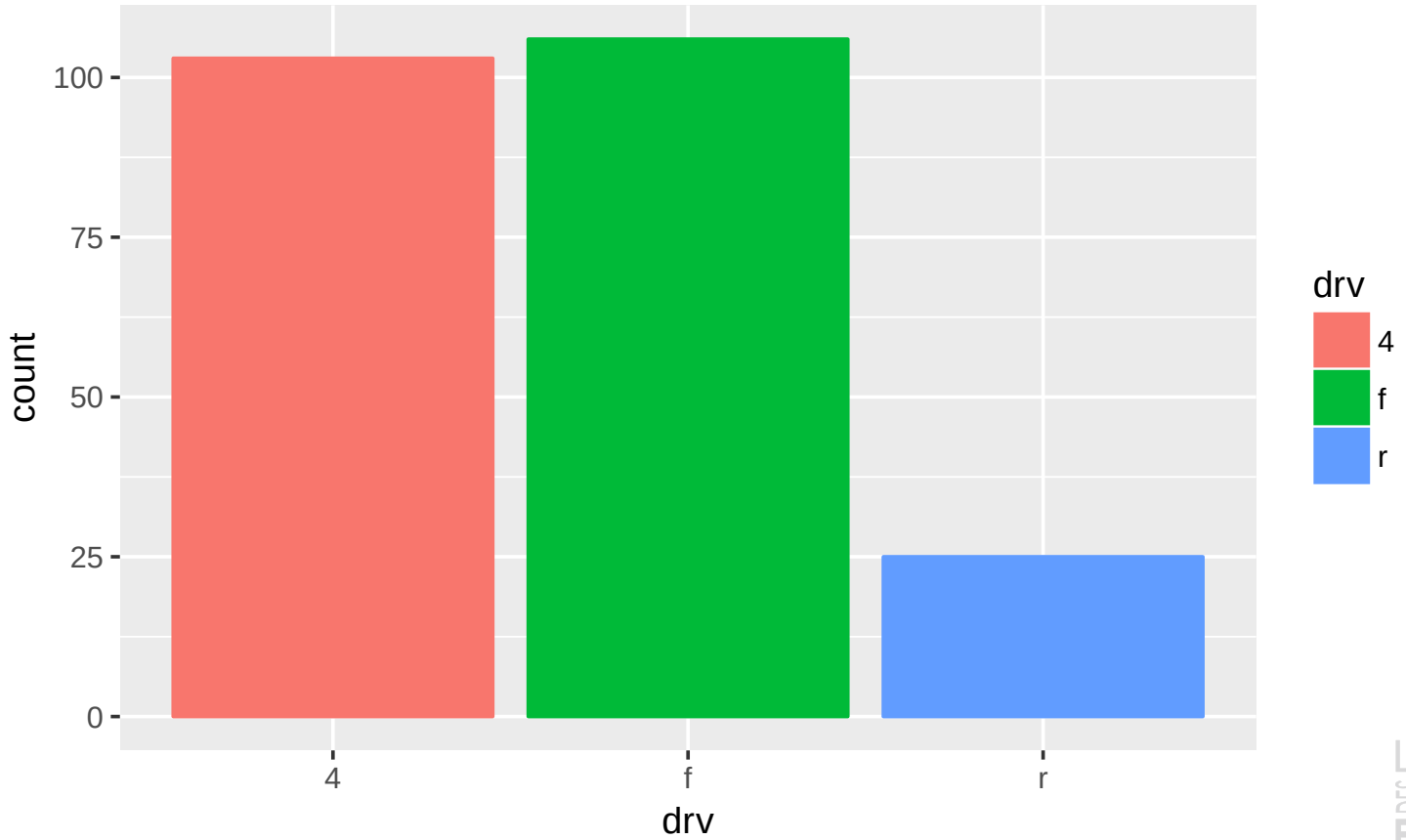
Séries temporelles

```
ggplot(economics, aes(date, unemploy / pop)) +
  geom_line()
```



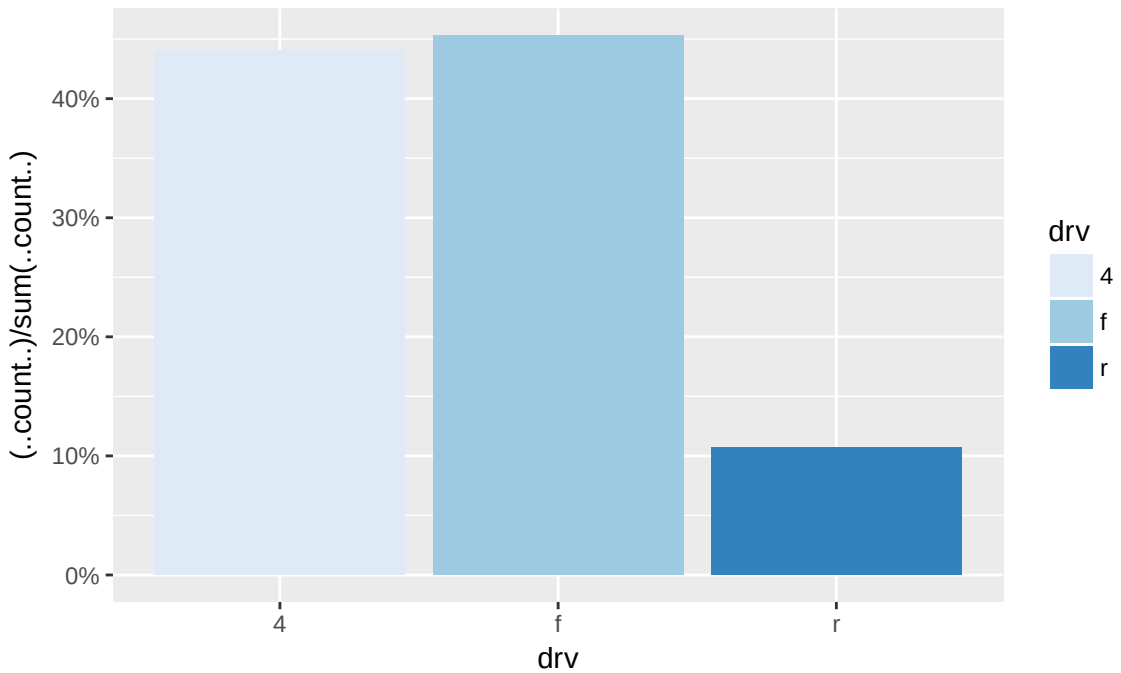
Diagrammes en bâtons et circulaires

```
ggplot(mpg, aes(drv, colour = drv, fill = drv)) +
  geom_bar()
```



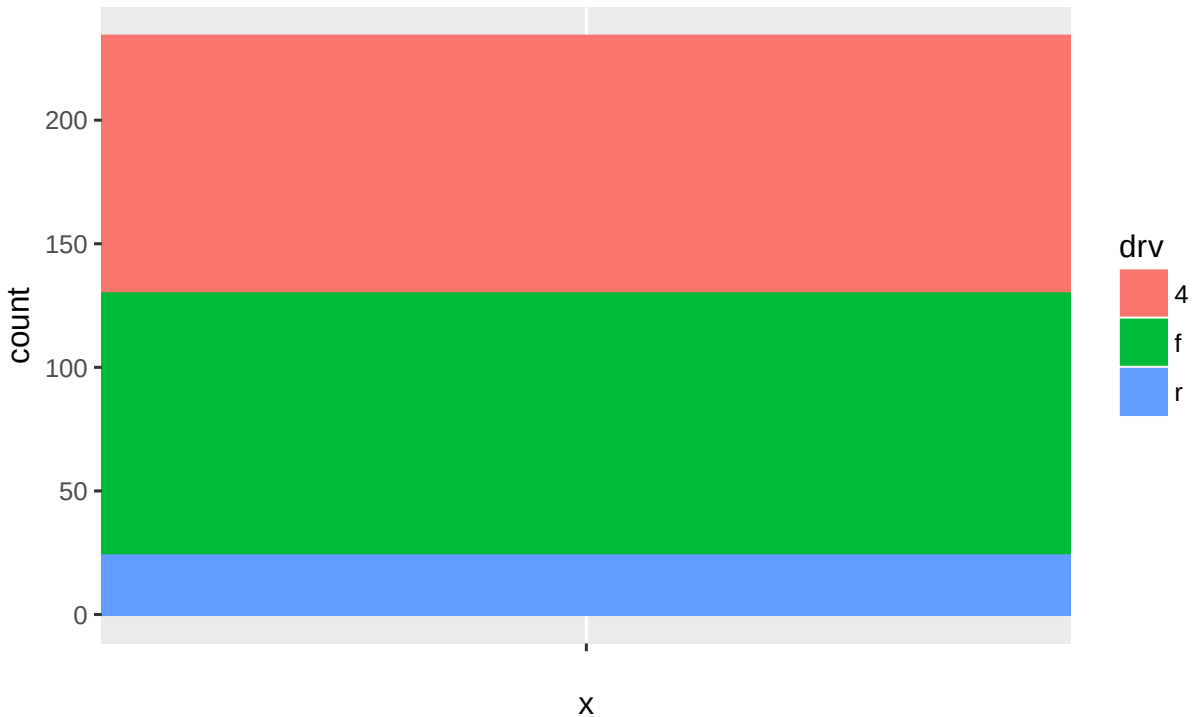
Diagrammes en bâtons et circulaires

```
library(scales)
ggplot(mpg, aes(drv, fill = drv)) +
  geom_bar(aes(y = (..count..)/sum(..count..))) +
  scale_y_continuous(labels=percent) +
  scale_fill_brewer(palette="Blues")
```



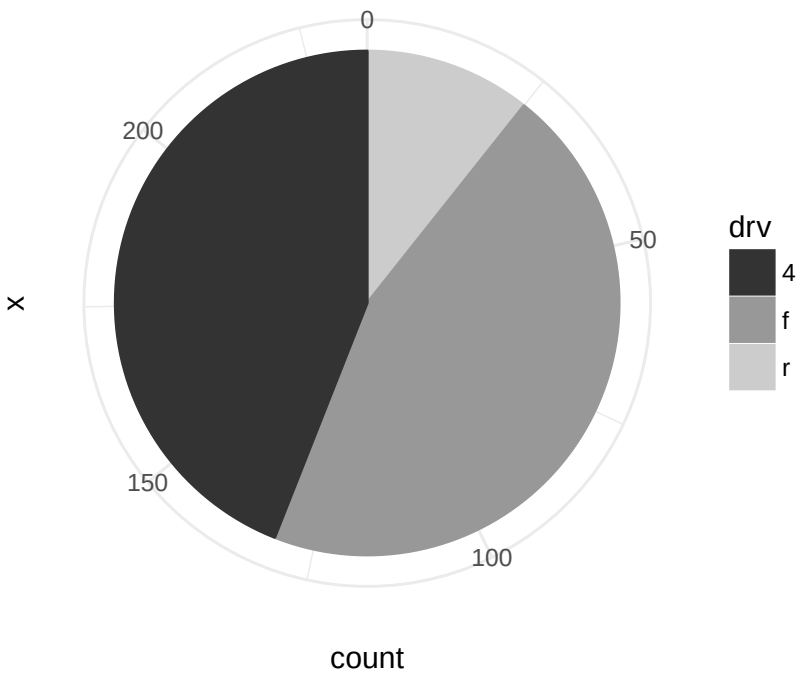
Diagrammes en bâtons et circulaires

```
g <- ggplot(mpg, aes(x = "", fill = drv, colour = drv)) +
  geom_bar(width = 1)
g
```



Diagrammes en bâtons et circulaires

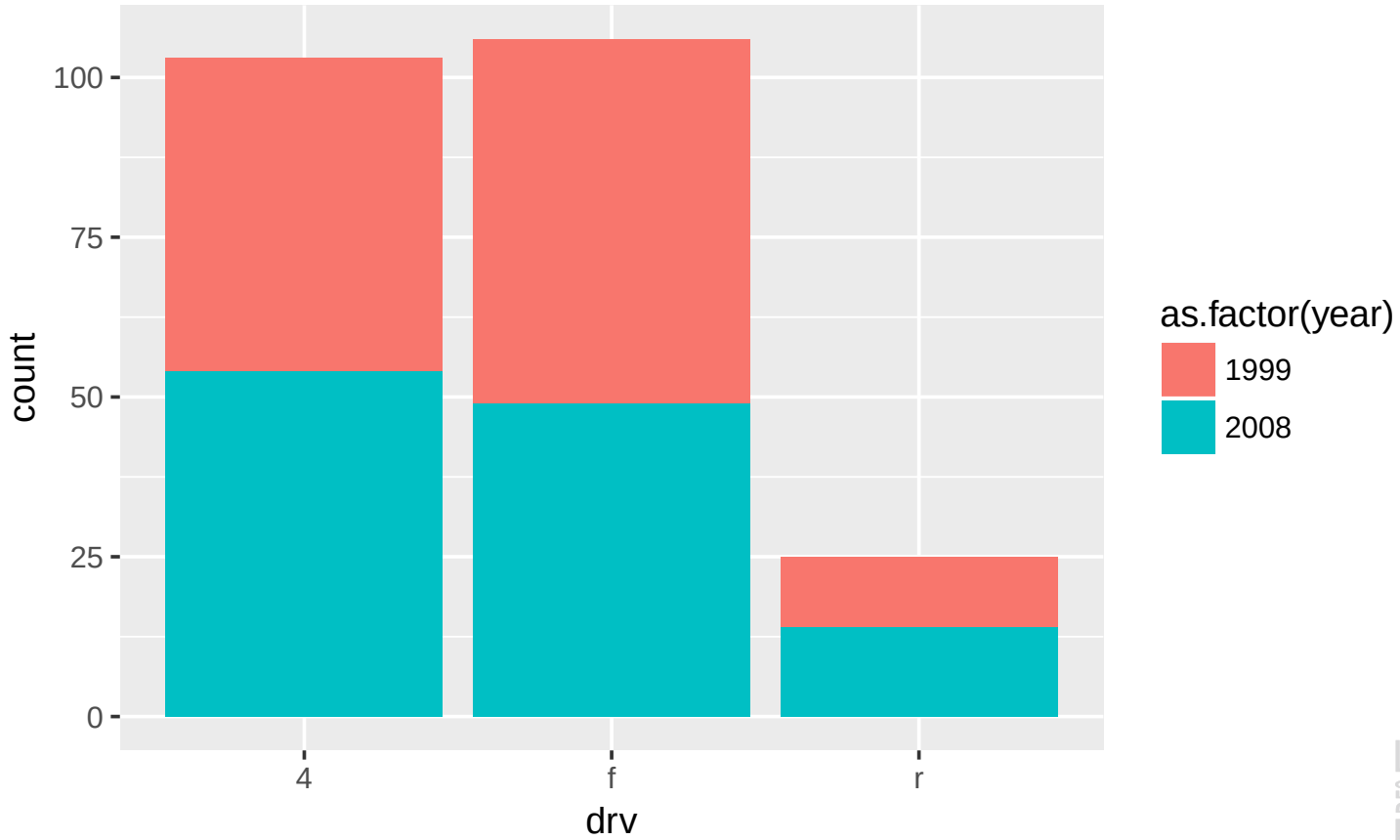
```
g + coord_polar(theta = "y") + theme_minimal() +
  scale_fill_grey() + scale_colour_grey()
```



Pour aller plus loin Une page du site sthda.com explique (en français) comment produire un diagramme circulaire complet avec ggplot2.

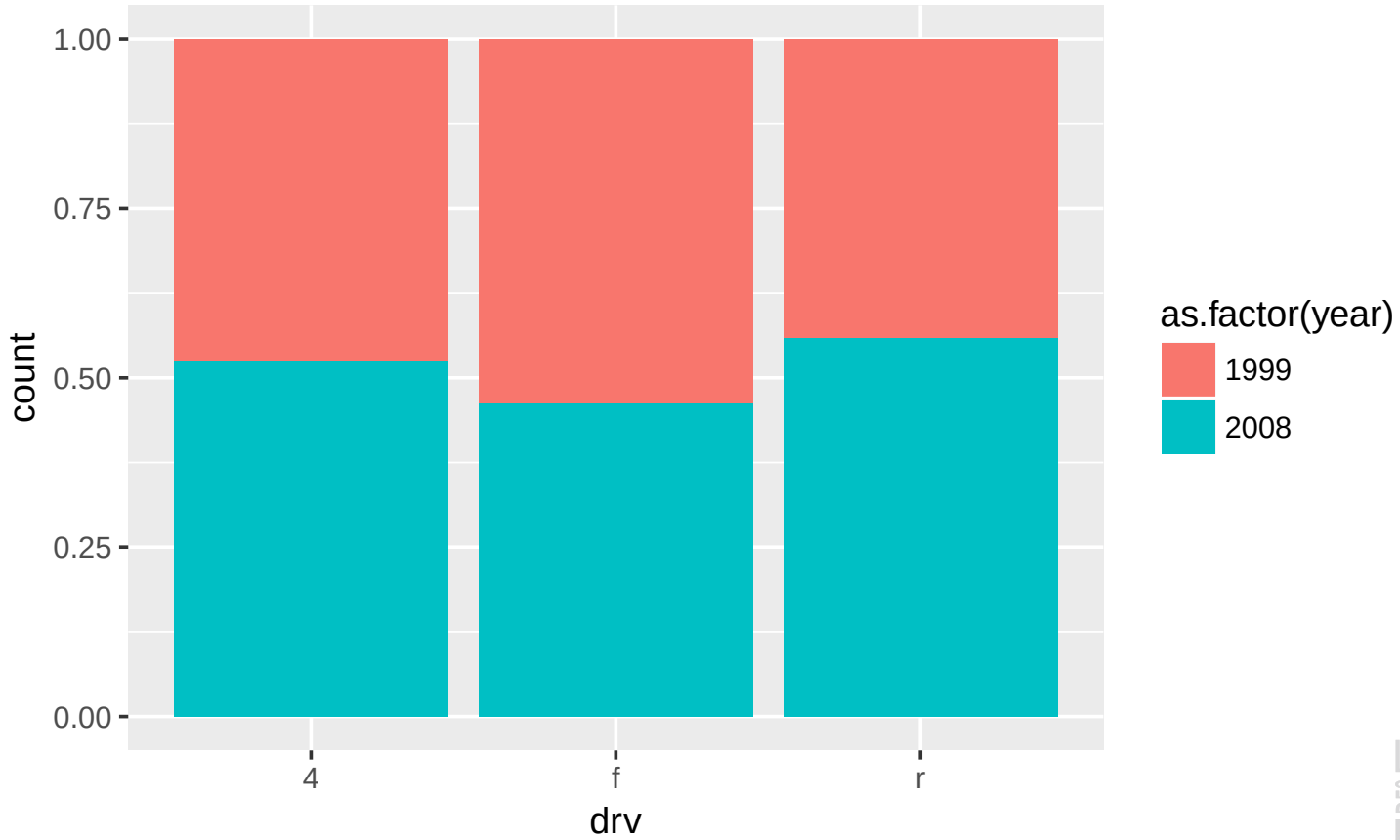
Diagrammes en bâtons et circulaires

```
ggplot(mpg, aes(drv, fill = as.factor(year))) +
  geom_bar()
```



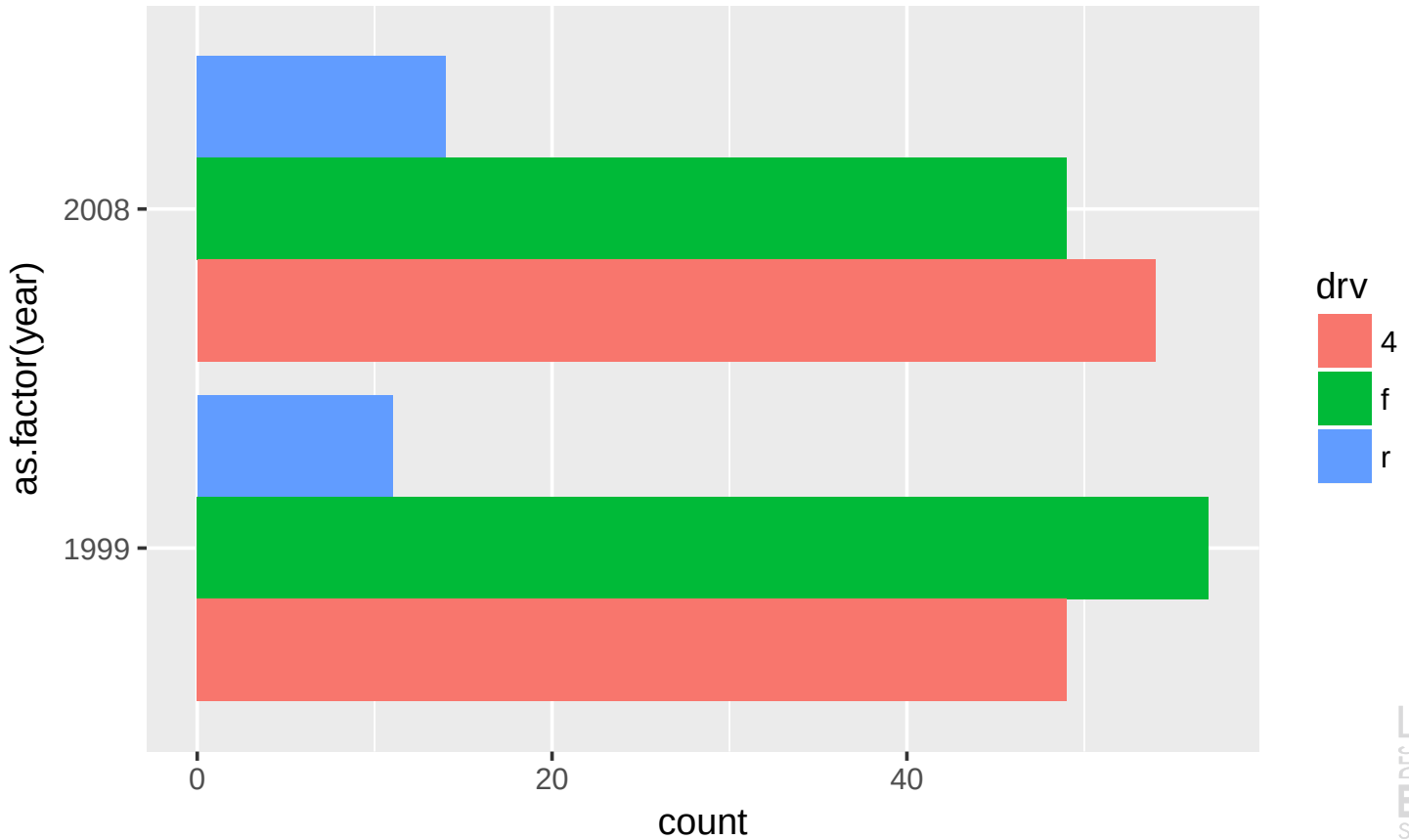
Diagrammes en bâtons et circulaires

```
ggplot(mpg, aes(drv, fill = as.factor(year))) +
  geom_bar(position = "fill")
```



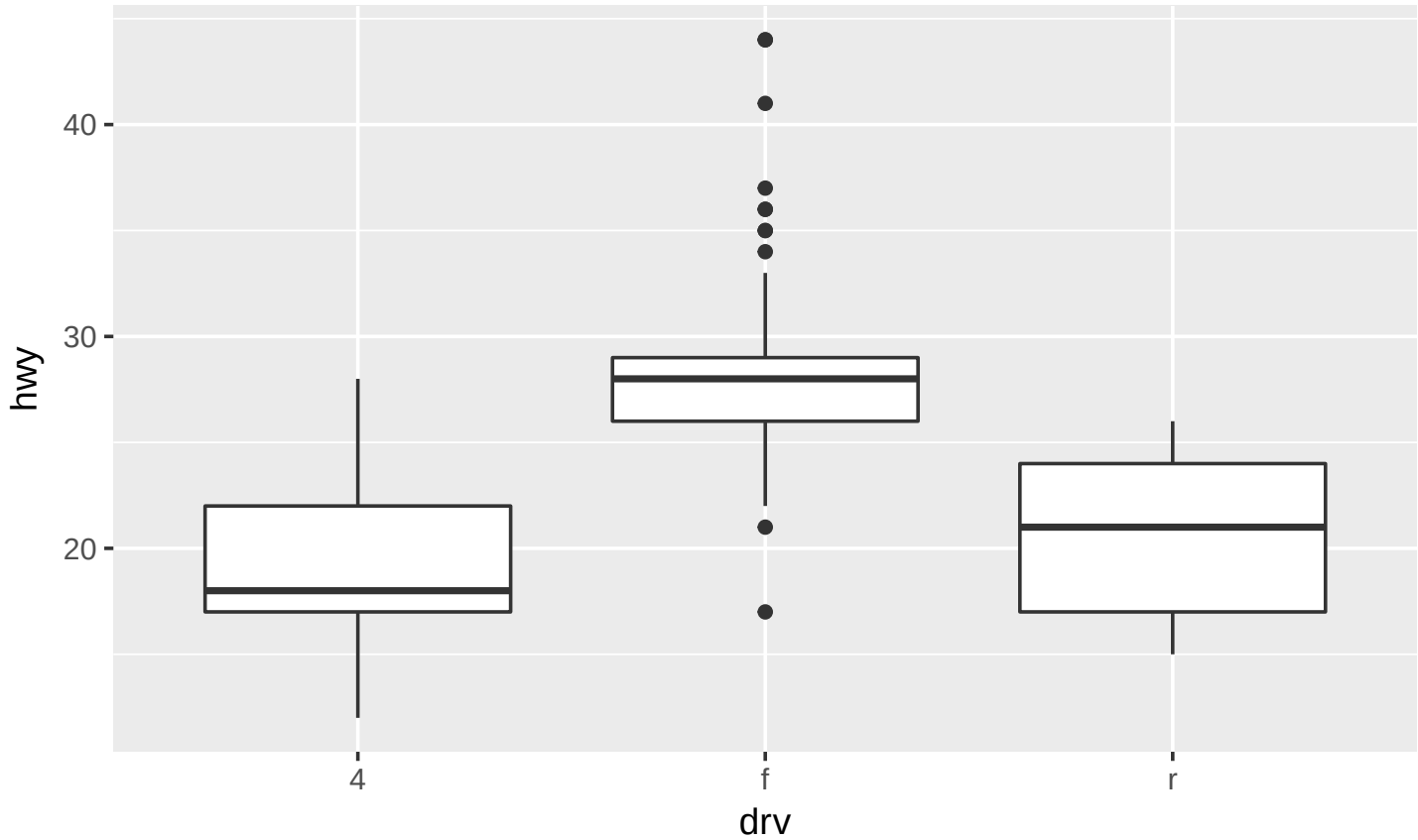
Diagrammes en bâtons et circulaires

```
ggplot(mpg, aes(as.factor(year), fill = drv)) +
  geom_bar(position = "dodge") +
  coord_flip()
```



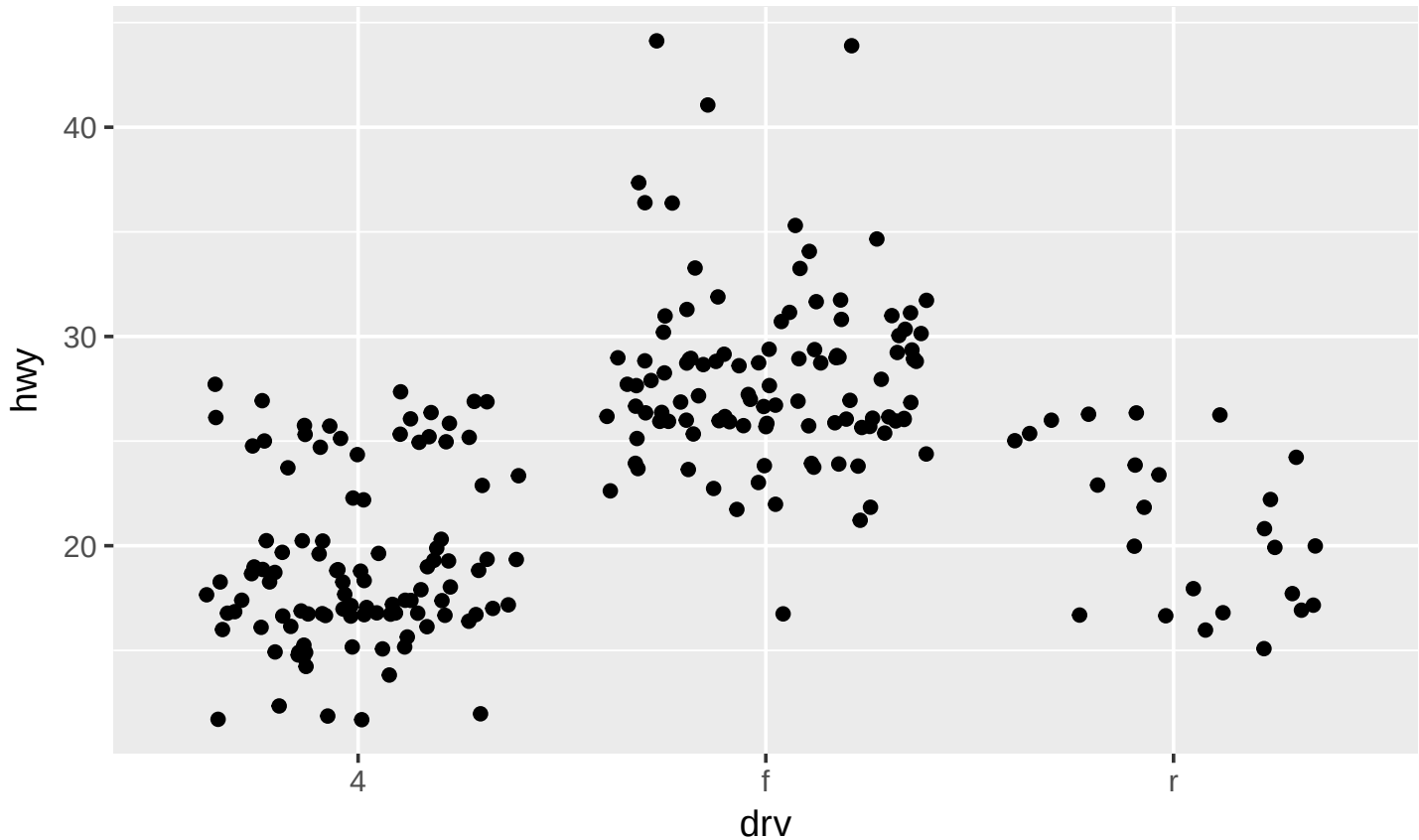
Boîtes à moustaches et assimilés

```
ggplot(mpg, aes(x = drv, y = hwy)) +
  geom_boxplot(coef = 1.5)
```



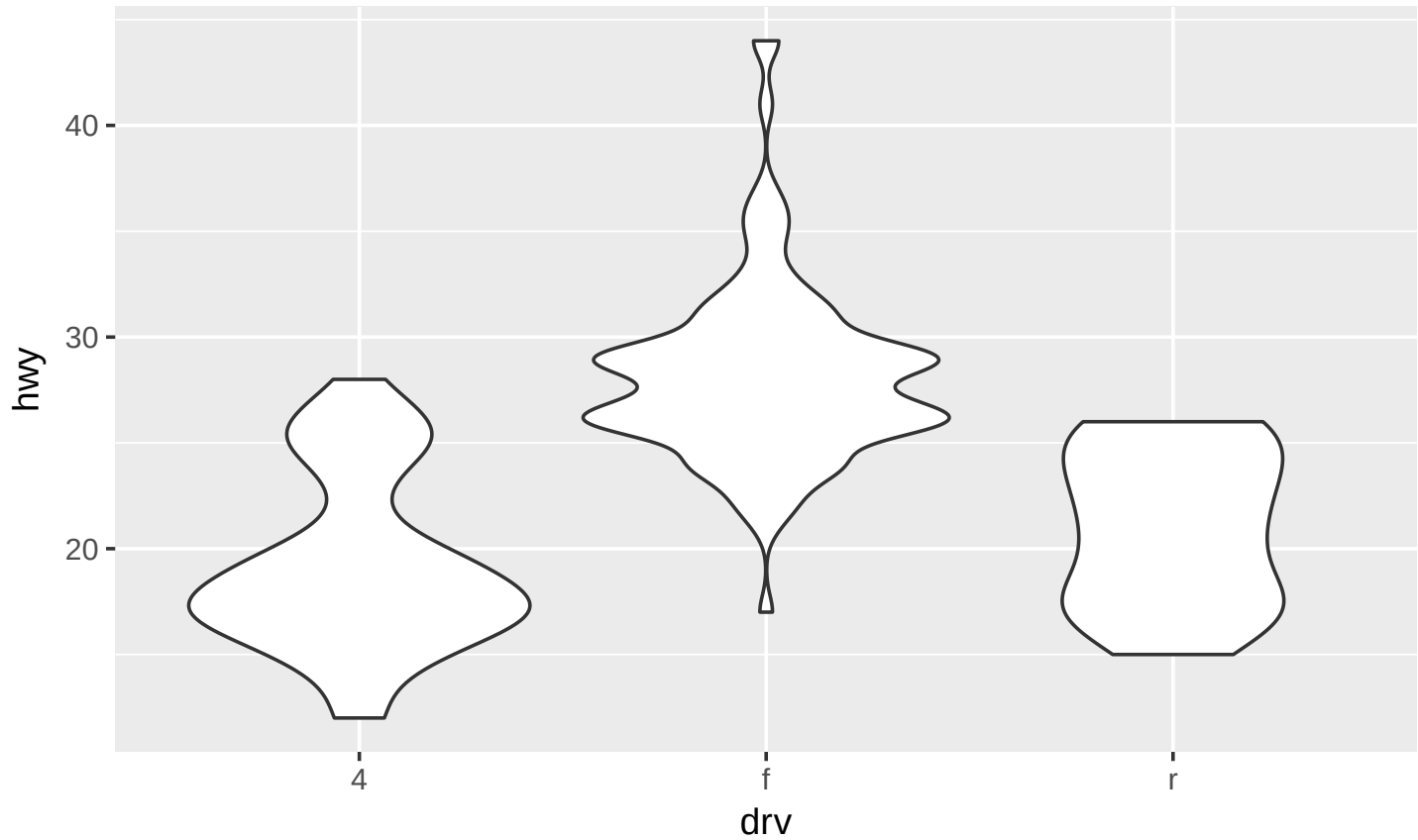
Boîtes à moustaches et assimilés

```
ggplot(mpg, aes(x = drv, y = hwy)) +
  geom_jitter()
```



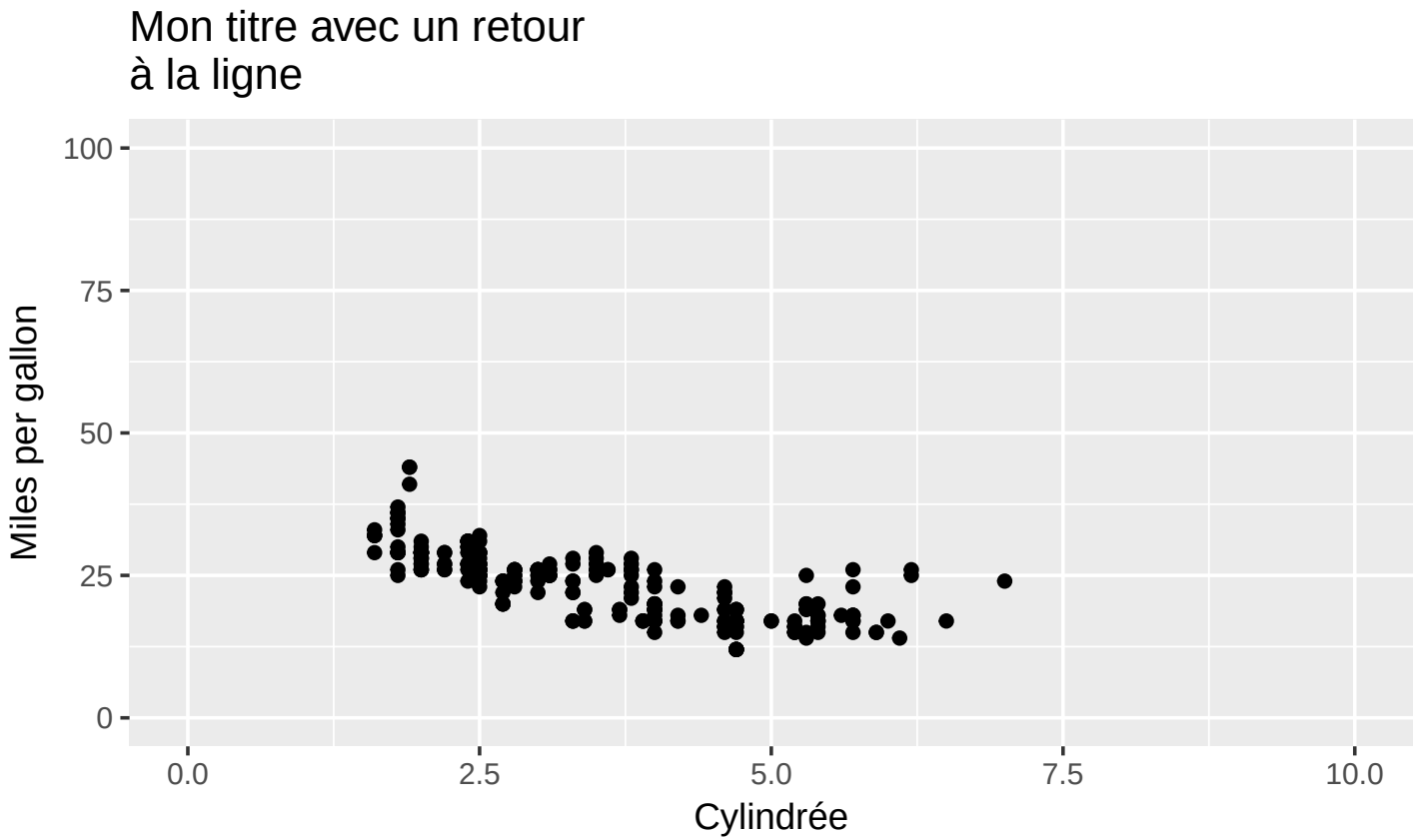
Boîtes à moustaches et assimilés

```
ggplot(mpg, aes(x = drv, y = hwy)) +
  geom_violin()
```



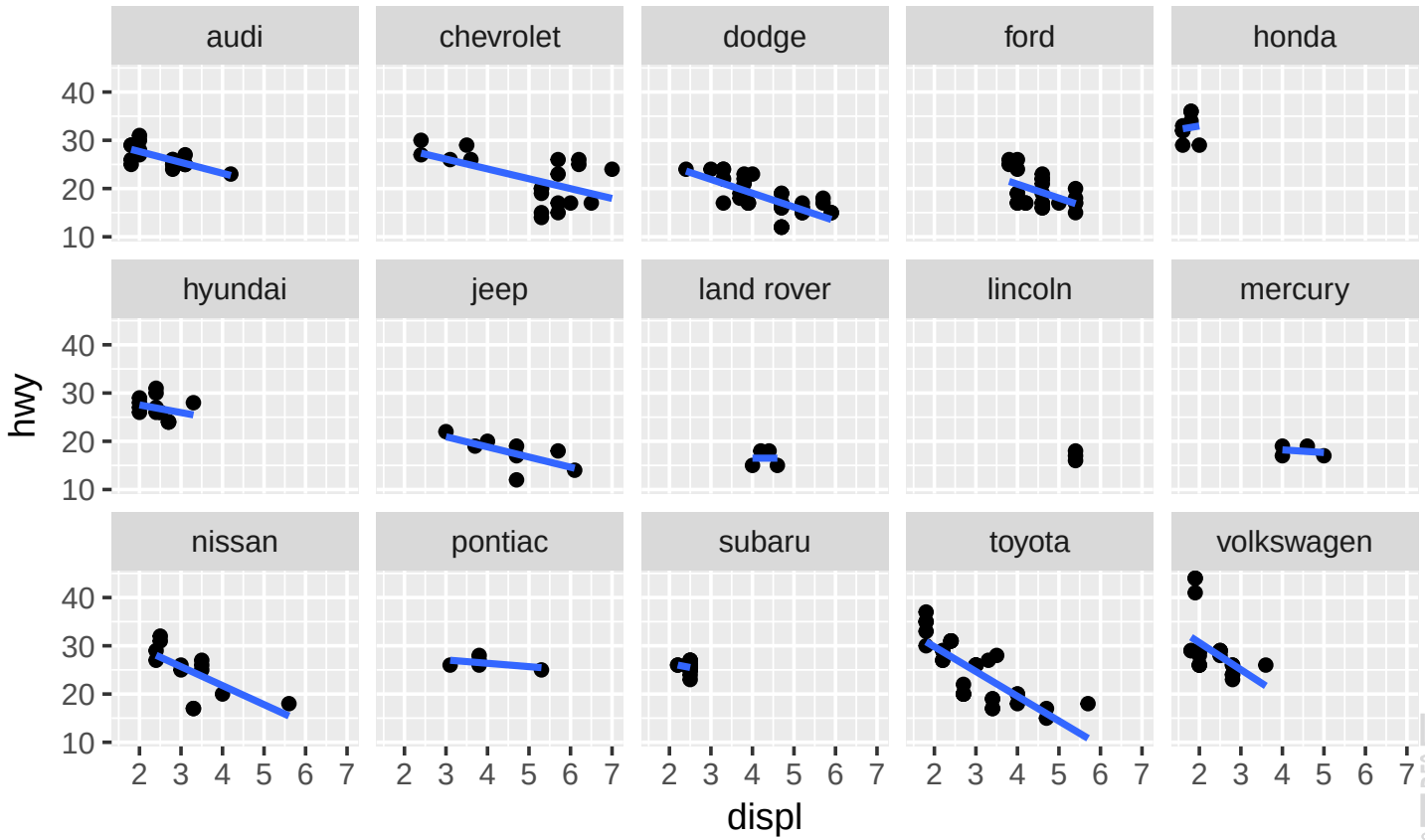
Titres et axes

```
ggplot(mpg, aes(displ, hwy)) + geom_point() +
  ggtitle("Mon titre avec un retour \nà la ligne") +
  xlab("Cylindrée") + ylab("Miles per gallon") +
  coord_cartesian(xlim = c(0,10), ylim = c(0, 100))
```



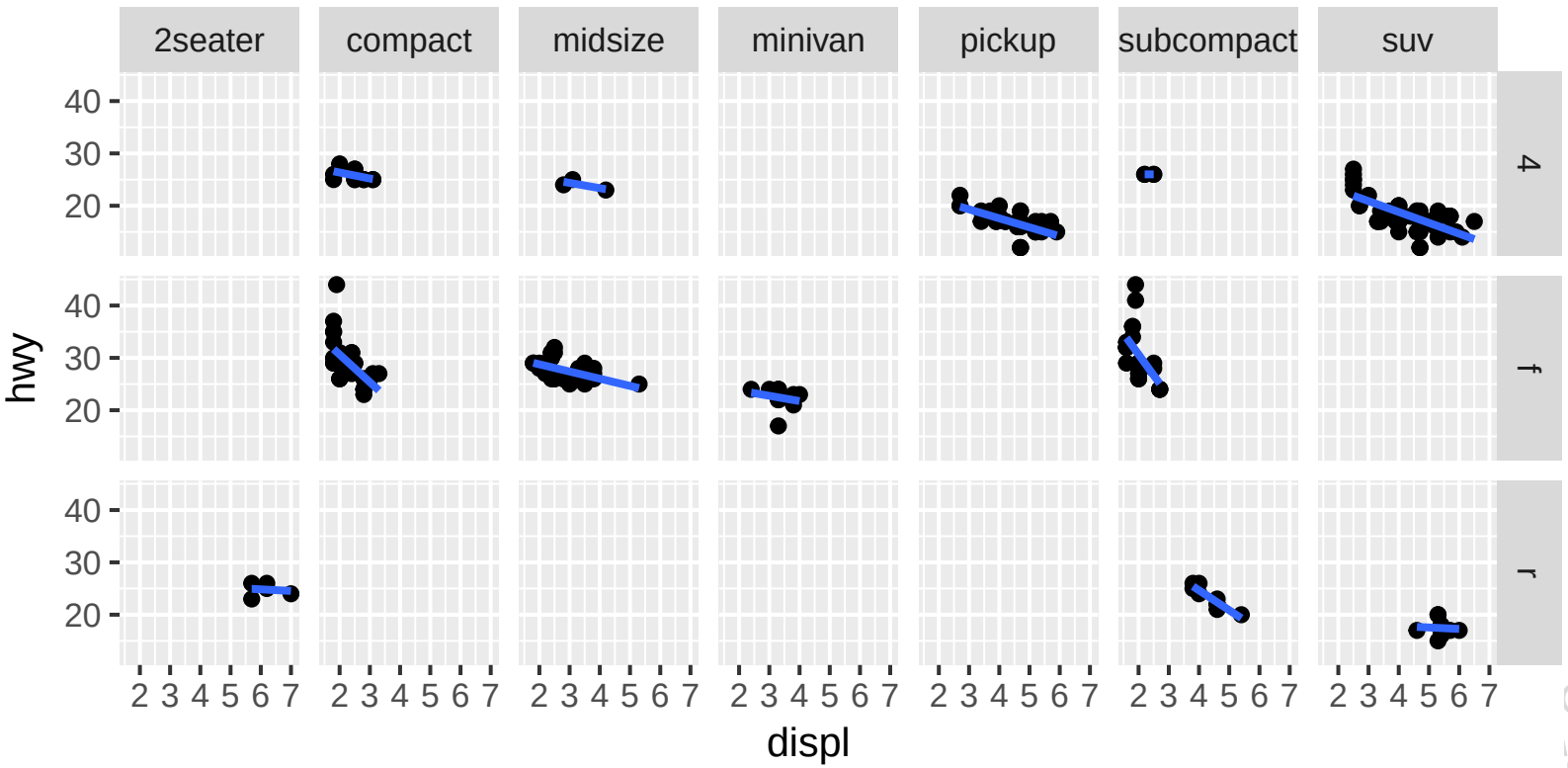
Disposition : le *facetting*

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() + geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~manufacturer, nrow = 3)
```



Disposition : le *facetting*

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point() + geom_smooth(method = "lm", se = FALSE) +
  facet_grid(drv~class)
```



Sauvegarde et exportation

Le résultat de la fonction `ggplot()` pouvant être stocké dans un objet R, il est possible de le sauvegarder tel quel avec `save()` ou `saveRDS()` et de le réutiliser par la suite dans R.

```
g <- ggplot(mpg, aes(displ, hwy)) + geom_point()
saveRDS(g, file = "g.rds")
```

La fonction `ggsave()` simplifie l'export de graphiques en dehors de R. Par défaut, elle sauvegarde le dernier graphique produit.

```
g + geom_smooth(method = "lm", se = FALSE)
ggsave("monGraphique.pdf")
ggsave("monGraphique.png")
```

Pourquoi générer automatiquement des documents ?

- Exporter et documenter des **traitements** en vue d'une réutilisation future : statistiques pour une étude, traitements réalisés lors d'une réunion de travail, etc.

  Remarque Utilisation analogue à celle permise par les instructions `ODS RTF` ou `ODS PDF` de SAS.

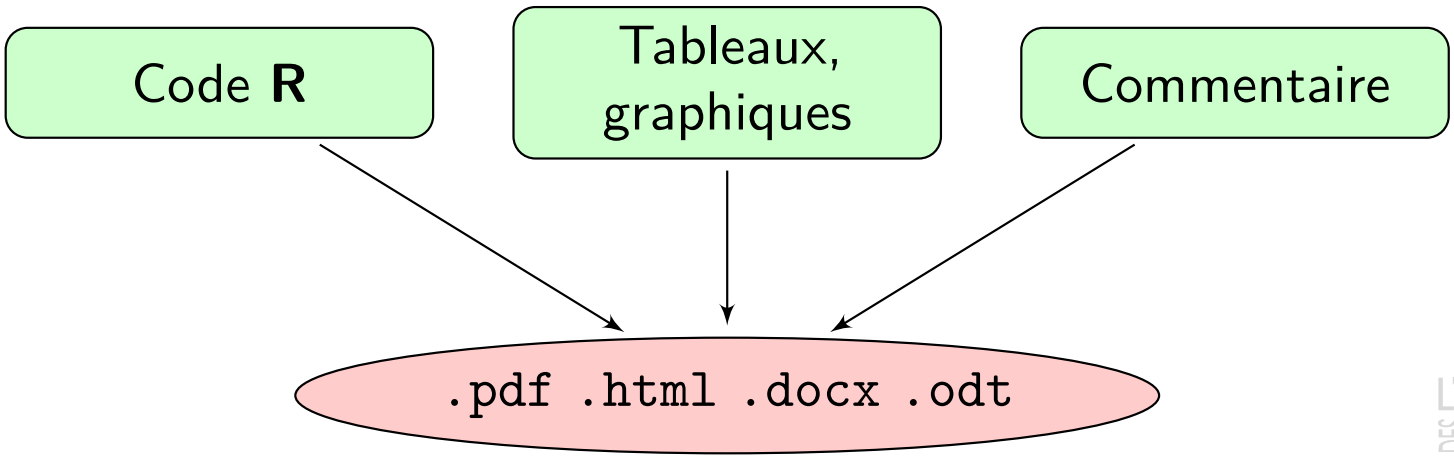
- Construire des **rapports complets et automatisés** pour des tâches répétitives : rapports d'utilisation, tests de la cohérence ou de la qualité de nouvelles données, etc.

- Produire des publications **reproductibles** sur différents supports : notes, documentation, articles de revues, etc.

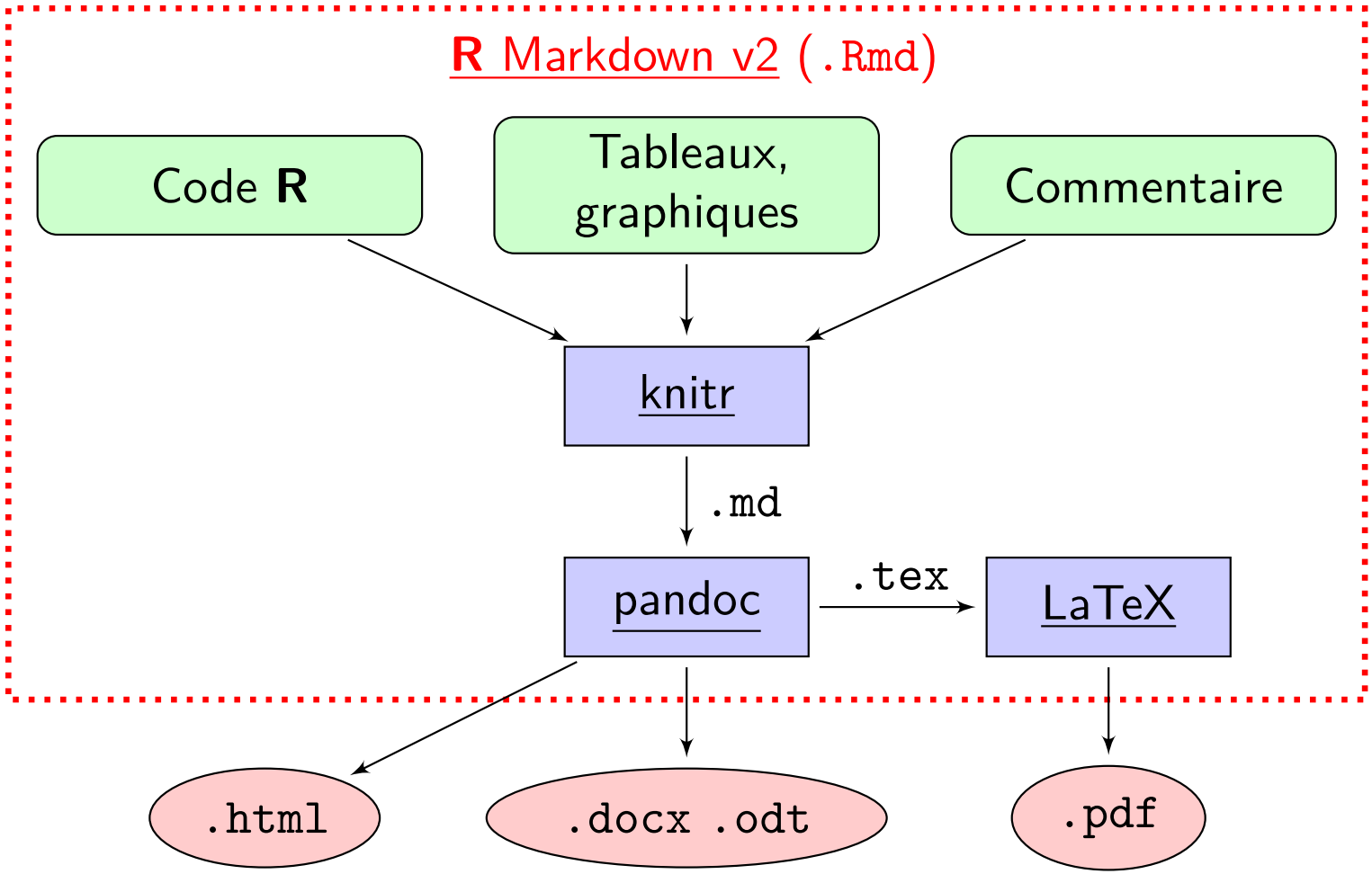
Principe de la génération automatique de documents

La génération automatique de documents complets repose sur deux éléments :

1. Articuler le code, les résultats et le commentaire dans un **même document** : garantir la cohérence et faciliter les mises à jour ;
2. Formater de façon standardisée le document vers **plusieurs sorties** : `.html`, `.pdf`, `.docx`, `.odt`.



Etapes de la génération automatique de documents



Note `rmarkdown` et `knitr` sont des *packages* R (avec plusieurs dépendances) ; pandoc et LaTeX sont des programmes autonomes.

Préparer et tester l'environnement de travail

1. Travailler sous RStudio
   - RStudio facilite l'édition et la compilation de fichier `.Rmd` ;
   - pandoc est embarqué par défaut dans RStudio.

2. Installer les *packages* nécessaires
   - installer le *package* `rmarkdown` et ses dépendances ;
   - installer le *package* `knitr` et ses dépendances.

3. Pour produire des fichiers `.pdf`, installer LaTeX (MiKTeX sous Windows) et s'assurer que ses programmes figurent dans le *path* de Windows.

4. Créer un nouveau fichier R Markdown (`.Rmd`), installer les *packages* complémentaires demandés, choisir le type de document et compiler le fichier d'exemple (`Ctrl + K`).

Ecrire du texte dans R Markdown

Pour écrire du texte dans un document R Markdown, il suffit de le **taper dans le fichier** `.Rmd` (sans le commenter ni l'échapper d'aucune manière).

Des **balises** spéciales permettent de mettre en forme le document :

- les signes `*` et `_` permettent de mettre des mots en `*italique*` ou en `**gras**` ;
- les six niveaux de titres sont préfixés par les signes `#` (premier niveau), `##` (deuxième niveau), etc.
- des listes sont automatiquement créées à partir de successions de `-` ou de séquences de nombres ou de lettres séparées par un retour à la ligne.

Note Pour une présentation synthétique de R Markdown, se référer à l'aide-mémoire (*cheat sheet*) sur le site de RStudio.

Ecrire du code dans R Markdown

Les blocs de code R sont intégrés dans R Markdown de la façon suivante :

````
```{r}
2 + 2
```
````

Par défaut **le code est évalué**, et **lui-même ainsi que ses résultats sont affichés** dans le document en sortie :

```
2 + 2
```

```
## [1] 4
```

Ecrire du code dans R Markdown

Les **options** saisies en début de bloc permettent de préciser à `knitr` la manière de le prendre en compte, par exemple :

- `eval=FALSE` : le bloc n'est pas évalué ;
- `echo=FALSE` : le bloc n'est pas affiché ;
- `collapse=TRUE` : code et résultats sont affichés à la suite.

````
```{r, echo=FALSE}
2 + 2
```
````

```
## [1] 4
```

Note Toutes les options de `knitr` relatives aux blocs de code (*chunk options*) sont présentées sur la page du créateur du *package*, Yihui Xie.

Ecrire du code dans R Markdown

Il est également possible d'intégrer le résultat d'un traitement R dans le corps d'un paragraphe avec la syntaxe :

```
`r   `
```

Exemple Pour intégrer dans le texte la date de compilation du document, utiliser

```
Document compilé le `r Sys.Date()`.
```

Document compilé le 2018-03-08.

Intégrer des graphiques dans R Markdown

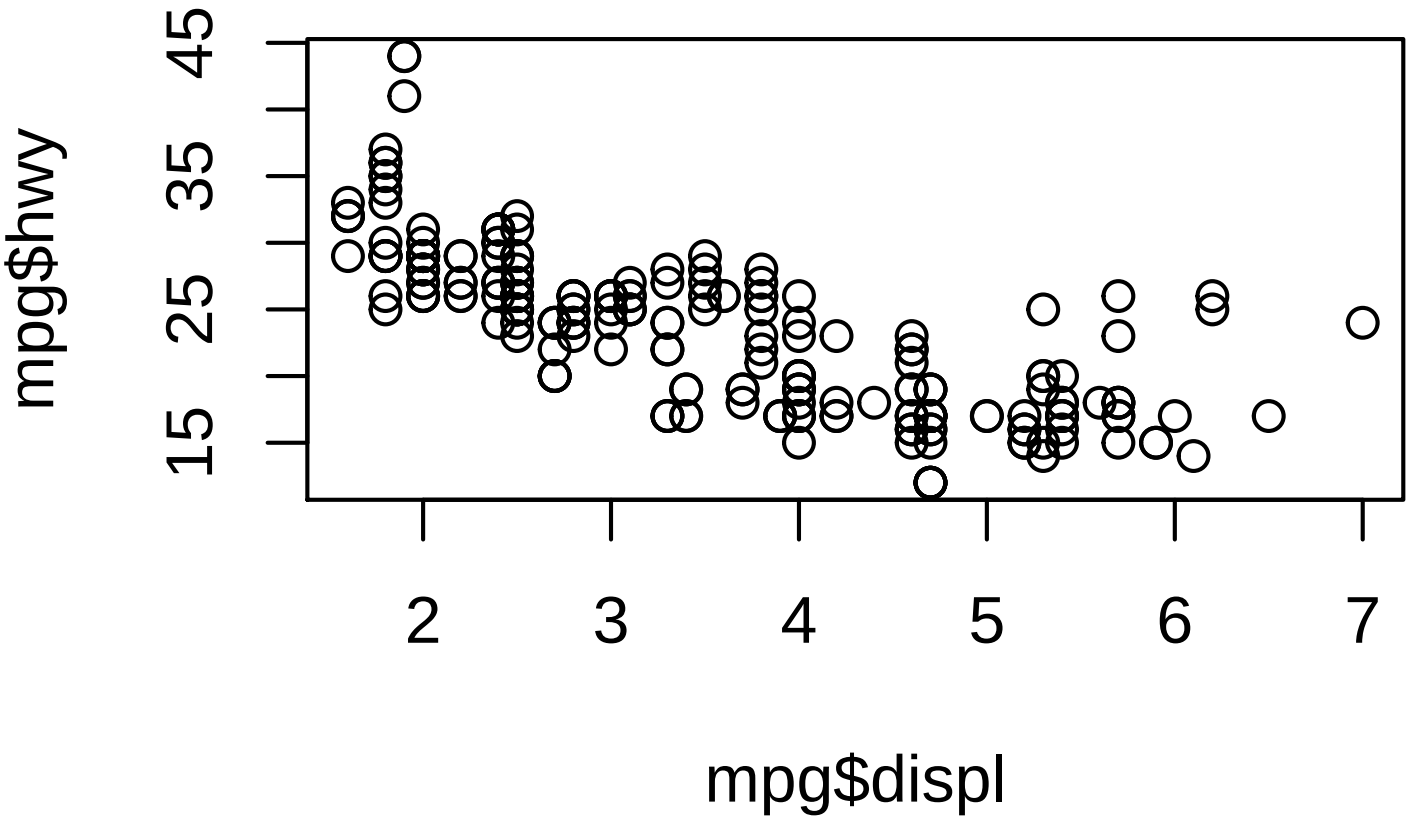
Tous les graphiques produits par les blocs de code sont **automatiquement intégrés au fichier final**.

Un **grand nombre d'options** sont consacrées au paramétrage des graphiques, notamment :

- `fig.width`, `fig.height` : largeur et hauteur utilisées pour produire le graphique, en pouces ;
- `fig.asp` : rapport hauteur/largeur (`fig.height` est neutralisé quand `fig.asp` est renseigné) ;
- `out.width`, `out.height` : largeur et hauteur du graphique dans la sortie finale ;
- `fig.align` : alignement du grahique (`"left"`, `"right"` ou `"center"`) ;
- `dpi` (72 par défaut) : résolution (utile uniquement pour HTML).

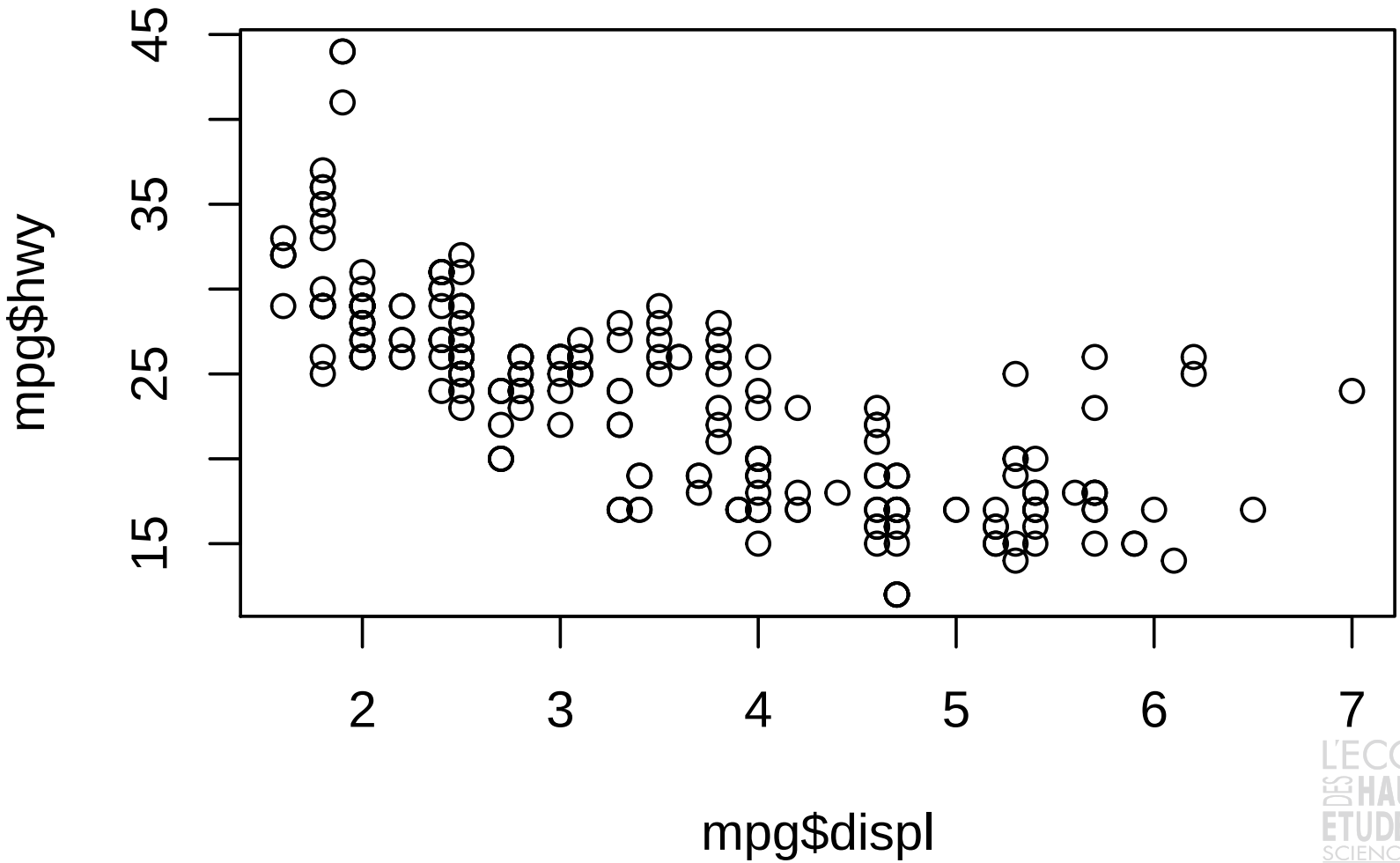
Intégrer des graphiques dans R Markdown

```
```{r, fig.asp = 3/4, fig.width = 4}
plot(mpg$displ, mpg$hwy)
```
```



Intégrer des graphiques dans R Markdown

```
```{r, fig.asp = 3/4, fig.width = 6, out.width = "4in"}
plot(mpg$displ, mpg$hwy)
```
```



Intégrer des tableaux dans R Markdown

Pour construire un tableau dans R Markdown, il suffit de le « dessiner » avec les signes - et | :

```
Colonne 1 | Colonne 2 | Colonne 3
--------: | :-------- | :-------:
1         | a         | `TRUE`
2         | b         | `FALSE`
```

Colonne 1	Colonne 2	Colonne 3
1	a	`TRUE`
2	b	`FALSE`

Les : permettent de spécifier l'alignement des colonnes.

Intégrer des tableaux dans R Markdown

En règle générale cependant, les tableaux à intégrer sont générés automatiquement à partir des données.

````
```{r}
resultat <- data.table(mpg)[
    , list(hwy=mean(hwy), cty=mean(cty)), by = drv
]
resultat
```
````

```
##    drv      hwy     cty
## 1:   f 28.16038 19.9717
## 2:   4 19.17476 14.3301
## 3:   r 21.00000 14.0800
```

La fonction `knitr::kable()` permet de **transformer un objet R en tableau formaté pour R Markdown**.

Intégrer des tableaux dans R Markdown

````
```{r, results = "asis"}
knitr::kable(resultat)
```
````

```
|drv |      hwy|      cty|
|:---|--------:|--------:|
|f   | 28.16038| 19.9717|
|4   | 19.17476| 14.3301|
|r   | 21.00000| 14.0800|
```

Ce qui donne une fois formaté par R Markdown :

drv	hwy	cty
f	28.16038	19.9717
4	19.17476	14.3301
r	21.00000	14.0800

Paramétrer un document R Markdown

La plupart des paramètres généraux du documents sont à indiquer dans son en-tête (désigné par l'acronyme YAML) :

```
---
title: "Formation R Perfectionnement"
author: "Martin Chevalier (Insee)"
output:
  html_document:
    highlight: haddock
    toc: yes
    toc_depth: 2
    toc_float: yes
---
```

Pour en savoir plus Le site de RStudio documente le paramétrage de l'en-tête YAML selon les formats de sortie souhaités (`html`, `pdf`).