

# Formation R Perfectionnement

Drees – 16-17 avril 2018



Martin CHEVALIER (Insee)

Réaliser des graphiques avec R

## 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.

# Réaliser des graphiques avec R

## 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.

# Réaliser des graphiques avec R

## Base R ou ggplot2 ?

Il existe aujourd'hui trois 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

Réaliser des graphiques avec `graphics`

Réaliser des graphiques avec `ggplot2`

## Réaliser des graphiques avec R

### 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"
## [4] "year" "cyl" "trans"
## [7] "drv" "cty" "hwy"
## [10] "fl" "class"
```

## Réaliser des graphiques avec R

### 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"
## [4] "year" "cyl" "trans"
## [7] "drv" "cty" "hwy"
## [10] "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.

## Réaliser des graphiques avec graphics

### 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.

## Réaliser des graphiques avec graphics

### 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**.



## Réaliser des graphiques avec graphics

### 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](http://statmethods.net) recense et illustre la plupart des fonctions du *package* `graphics`.

Réaliser des graphiques avec `graphics`

## Histogrammes et densités

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

### 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.

## Réaliser des graphiques avec `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.

## Réaliser des graphiques avec `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()` :

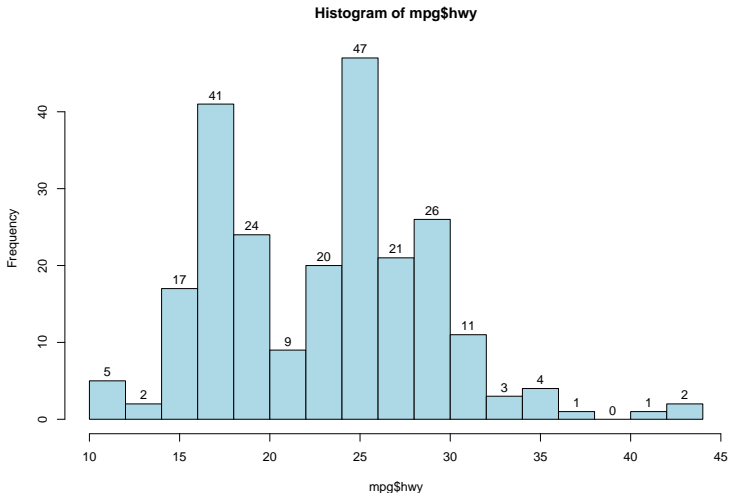
- ▶ `bw` : largeur de la fenêtre utilisée par la fonction de lissage ;
- ▶ `kernel` : fonction de lissage utilisée.

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

# Réaliser des graphiques avec graphics

## Histogrammes et densités

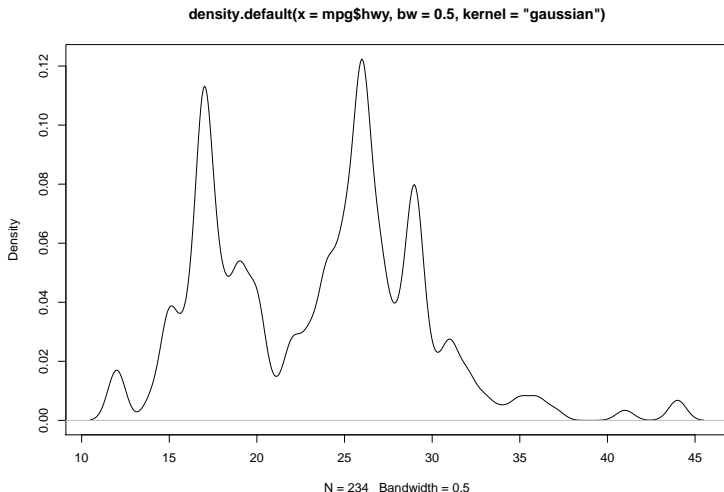
```
hist(mpg$hwy, breaks = seq(10, 44, by = 2),  
     col = "lightblue", labels = TRUE)
```



# Réaliser des graphiques avec graphics

## Histogrammes et densités

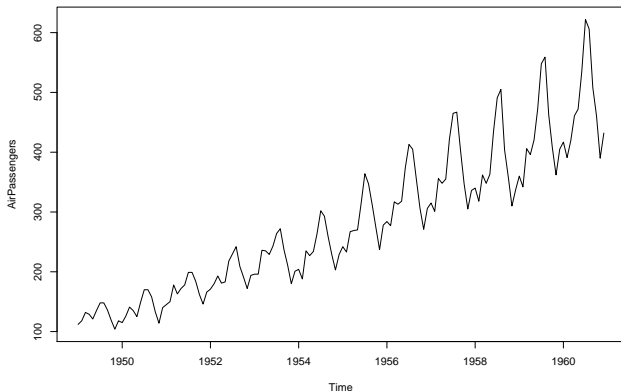
```
plot(density(mpg$hwy, bw = 0.5, kernel = "gaussian"))
```



# Réaliser des graphiques avec graphics

## Séries chronologiques avec plot(ts)

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

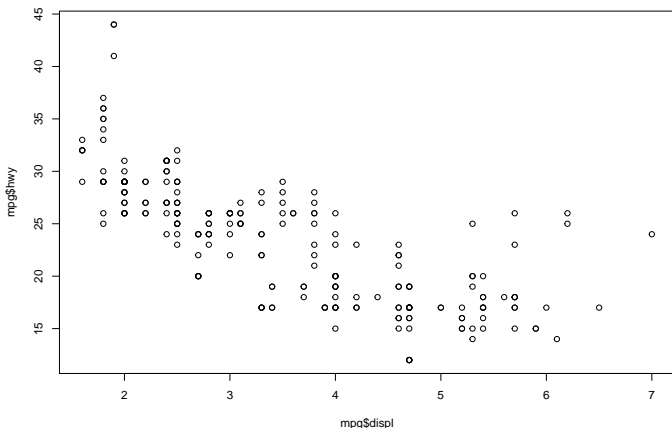




Réaliser des graphiques avec graphics

## Nuages de points avec `plot(x, y)`

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



Réaliser des graphiques avec `graphics`

## Diagrammes en bâtons et circulaires

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

Réaliser des graphiques avec `graphics`

## 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.

## Réaliser des graphiques avec `graphics`

### 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.

## Réaliser des graphiques avec `graphics`

### 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.

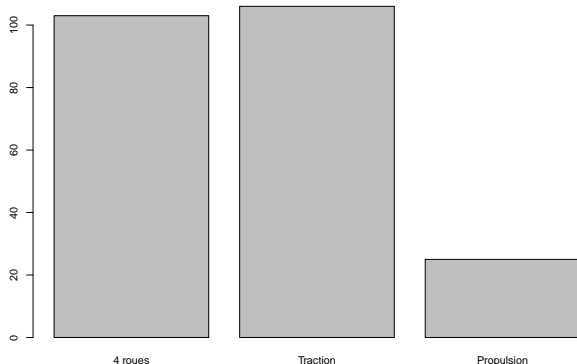
**Remarque** Quand `barplot()` est appliqué à un tri croisé, la couleur des barres varie et les paramètres deviennent utiles :

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

## Réaliser des graphiques avec graphics

# Diagrammes en bâtons et circulaires

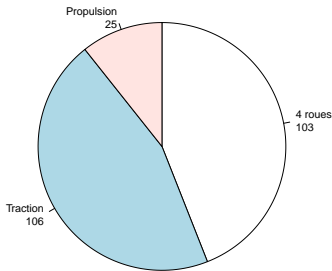
```
uni <- table(mpg$drv)
lab <- c("4 roues", "Traction", "Propulsion")
barplot(uni, names.arg = lab)
```



# Réaliser des graphiques avec graphics

## Diagrammes en bâtons et circulaires

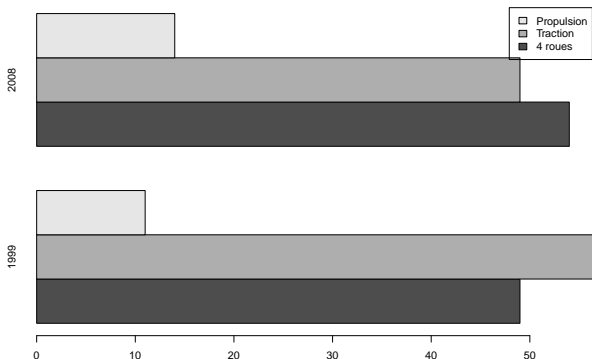
```
pie(uni, labels = paste0(lab, "\n", uni)  
    , init.angle = 90, clockwise = TRUE)
```



# Réaliser des graphiques avec graphics

## Diagrammes en bâtons et circulaires

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





Réaliser des graphiques avec `graphics`

## Couleur, forme et taille des objets

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

## Réaliser des graphiques avec graphics

### 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.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
○	△	+	×	◇	▽	⊠	✱	⊕	⊗	⊞	⊠	⊞	⊠	■	●	▲	◆	●	●	○	□	◇	△	▽

## Réaliser des graphiques avec graphics

### 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.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
○	△	+	×	◇	▽	⊠	✱	⊕	⊗	⊞	⊠	⊞	⊠	■	●	▲	◆	●	●	○	□	◇	△	▽

## Réaliser des graphiques avec graphics

### 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"
```

## Réaliser des graphiques avec graphics

### 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.

Réaliser des graphiques avec `graphics`

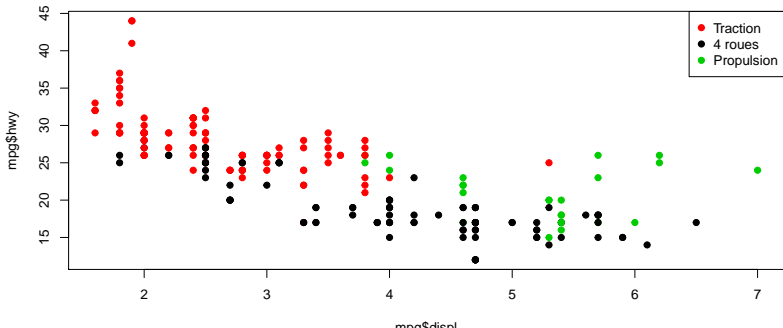
## Couleur, forme et taille des objets

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

# Réaliser des graphiques avec graphics

## Couleur, forme et taille des objets

```
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))
```



## Réaliser des graphiques avec `graphics`

### 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.



## Réaliser des graphiques avec `graphics`

### 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).

## Réaliser des graphiques avec `graphics`

### 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.

## Réaliser des graphiques avec `graphics`

### 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()`.

Réaliser des graphiques avec `graphics`

## Combinaison de plusieurs graphiques

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

### 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.

### 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 :

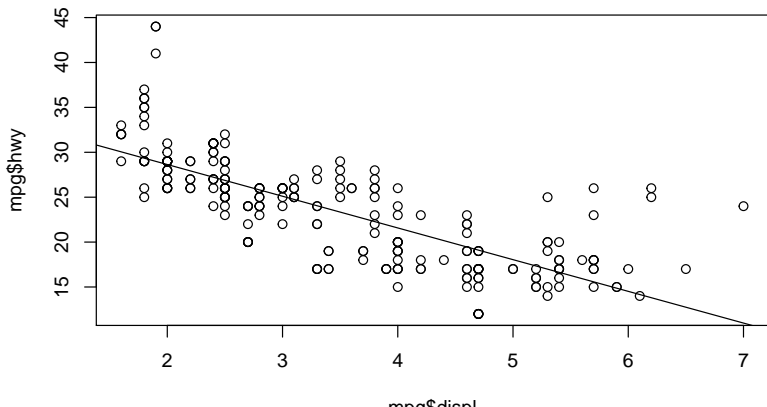
- ▶ avec `points()` pour ajouter des points ;
- ▶ avec `lines()` pour ajouter des lignes ;
- ▶ avec `abline()` pour ajouter des lignes d'après une équation ;
- ▶ avec `curve()` pour ajouter des courbes d'après une équation.

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

## Réaliser des graphiques avec graphics

### Combinaison de plusieurs graphiques

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



Réaliser des graphiques avec `graphics`

## Paramètres généraux et disposition (1)

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



## Réaliser des graphiques avec graphics

### 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).

## Réaliser des graphiques avec graphics

### 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
```

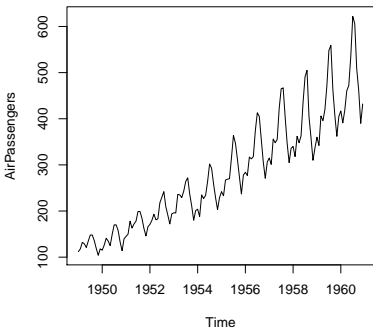
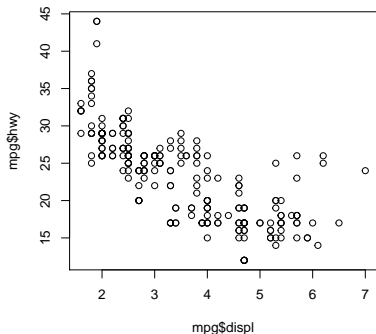
- ▶ `cex` : coefficient multiplicatif pour modifier la taille de l'ensemble des textes et symboles utilisés dans les graphiques (1 par défaut).

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

# Réaliser des graphiques avec graphics

## Paramètres généraux et disposition (2)

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



## Réaliser des graphiques avec `graphics`

### Exportation

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

## Réaliser des graphiques avec graphics

### Exportation

Pour exporter des graphiques depuis R, la démarche consiste à rediriger le flux de production du graphique 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()
```

## Réaliser des graphiques avec graphics

### Exportation

Pour exporter des graphiques depuis R, la démarche consiste à rediriger le flux de production du graphique 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 produits par R.

# Réaliser des graphiques avec graphics

## Exportation

Pour exporter des graphiques depuis R, la démarche consiste à rediriger le flux de production du graphique 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 produits par R.

**Remarque** Les graphiques peuvent également facilement être exportés depuis RStudio en utilisant les menus prévus à cet effet.

## Réaliser des graphiques avec ggplot2

### 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.



## Réaliser des graphiques avec ggplot2

### 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**.

## Réaliser des graphiques avec ggplot2

### 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](#)

# Réaliser des graphiques avec ggplot2

## 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.

# Réaliser des graphiques avec ggplot2

## 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.

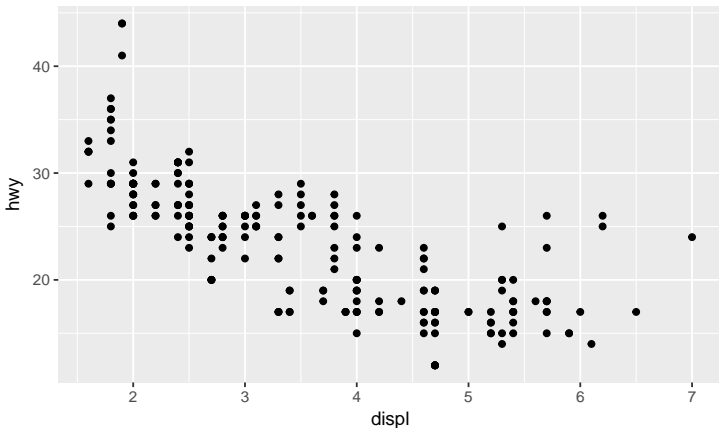
**Exemple** *Miles per gallon* sur l'autoroute en fonction de la cylindrée.

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

# Réaliser des graphiques avec ggplot2

## Les trois composants essentiels d'un graphique

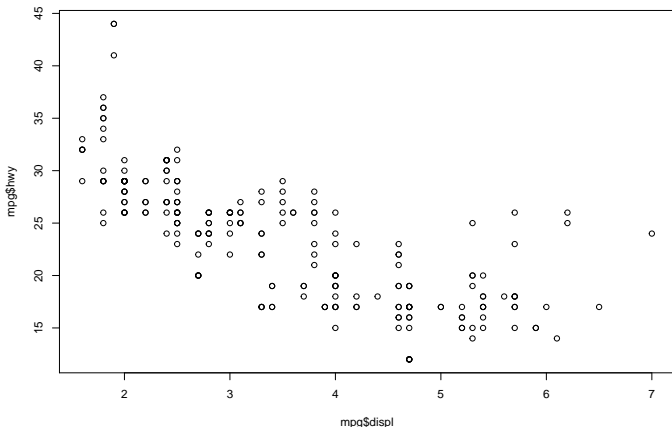
```
ggplot(data = mpg, mapping = aes(x = displ, y = hwy)) +  
  geom_point()
```



# Réaliser des graphiques avec ggplot2

## Rappel : le même graphique avec base R

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



Réaliser des graphiques avec ggplot2

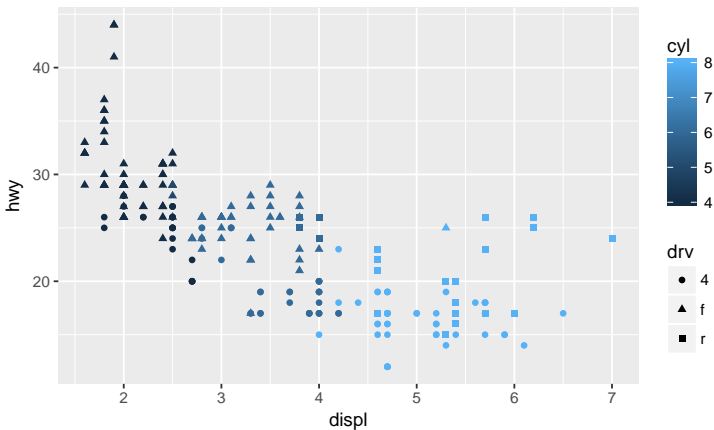
## Couleur, forme et taille des objets

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()`.

# Réaliser des graphiques avec ggplot2

## Couleur, forme et taille des objets

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





## Réaliser des graphiques avec ggplot2

### Couleur, forme et taille des objets

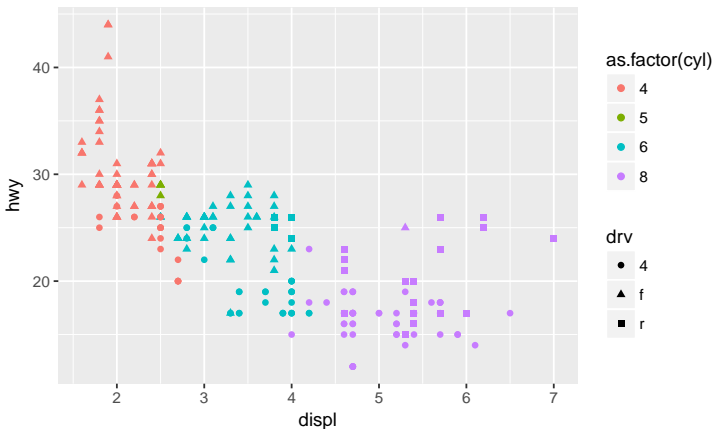
Pour faire varier l'aspect visuel des éléments représentés en fonction de données, il suffit d'**associer une variable à l'attribut de couleur, de taille ou de forme** dans la fonction `aes()`.

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

# Réaliser des graphiques avec ggplot2

## Couleur, forme et taille des objets

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



## Réaliser des graphiques avec ggplot2

### 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()`.

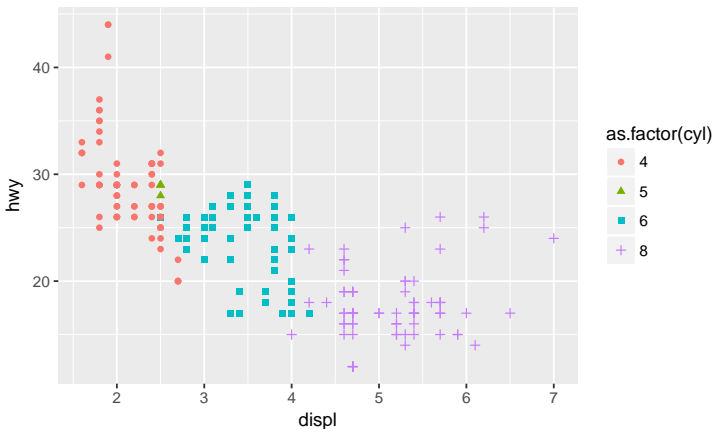
Selon le type des variables utilisées pour les correspondances esthétiques, **les échelles sont continues ou discrètes**.

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

# Réaliser des graphiques avec ggplot2

## Couleur, forme et taille des objets

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



## Réaliser des graphiques avec ggplot2

### 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**.

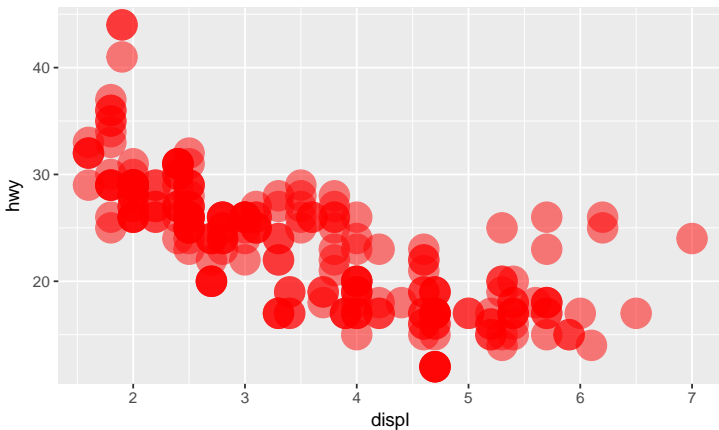
Quand la même variable est utilisée dans plusieurs correspondances esthétiques, **les échelles qui lui correspondent sont fusionnées**.

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

# Réaliser des graphiques avec ggplot2

## Couleur, forme et taille des objets

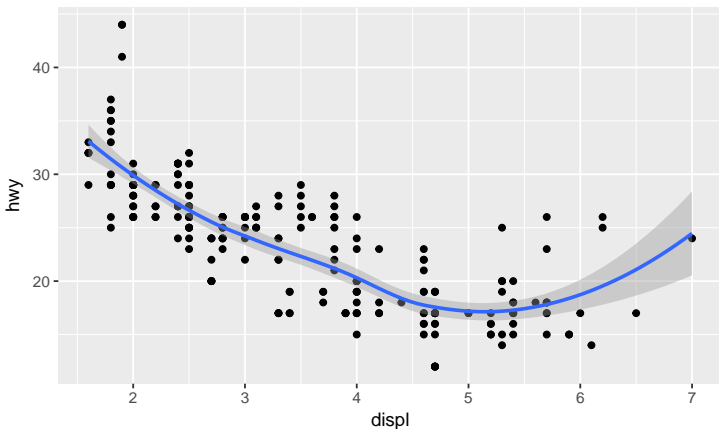
```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point(colour = "red", size = 8, alpha = 0.5)
```



# Réaliser des graphiques avec ggplot2

## Combinaison de plusieurs graphiques

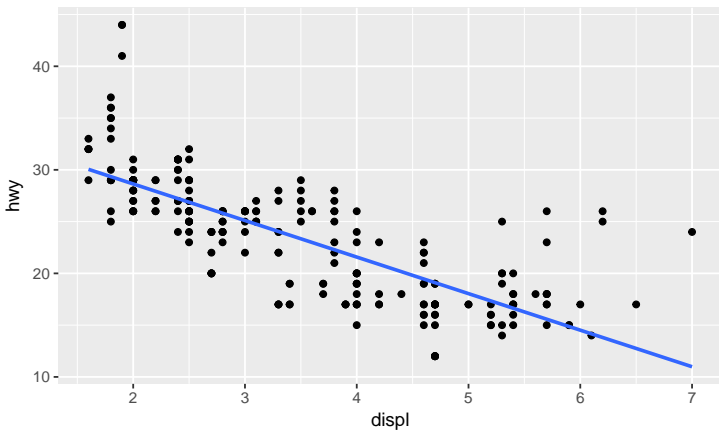
```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point() + geom_smooth()  
## `geom_smooth()` using method = 'loess'
```



# Réaliser des graphiques avec ggplot2

## Combinaison de plusieurs graphiques

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

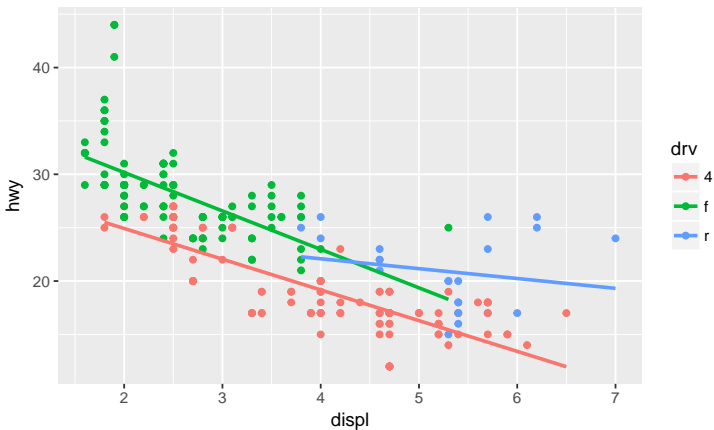




# Réaliser des graphiques avec ggplot2

## Combinaison de plusieurs graphiques

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



# Réaliser des graphiques avec ggplot2

## 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**.

# Réaliser des graphiques avec ggplot2

## 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.

# Réaliser des graphiques avec ggplot2

## 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.

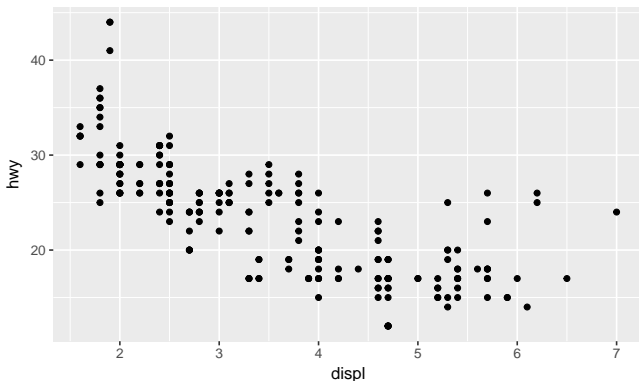
**Les fonctions `geom_*` vues précédemment sont des appels pré-paramétrés de `layer()`.**

# Réaliser des graphiques avec ggplot2

## 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"  
)
```



# Réaliser des graphiques avec ggplot2

## Le fonctionnement en « couches » de ggplot2

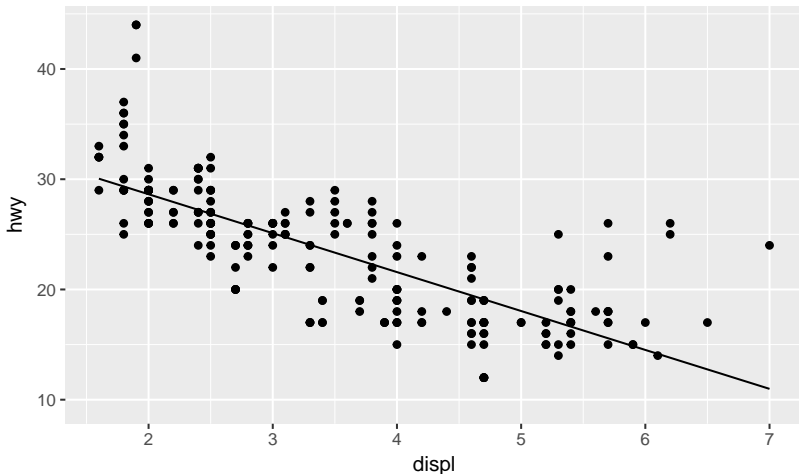
### *Un graphique à deux couches*

```
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)  
)
```

# Réaliser des graphiques avec ggplot2

## Le fonctionnement en « couches » de ggplot2

*Un graphique à deux couches*



# Réaliser des graphiques avec ggplot2

## Le fonctionnement en « couches » de ggplot2

### *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)
```

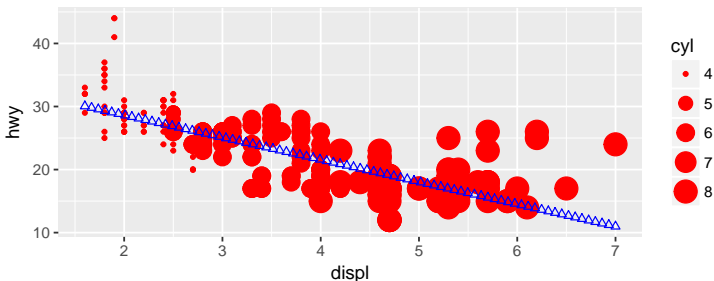


# Réaliser des graphiques avec ggplot2

## 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.

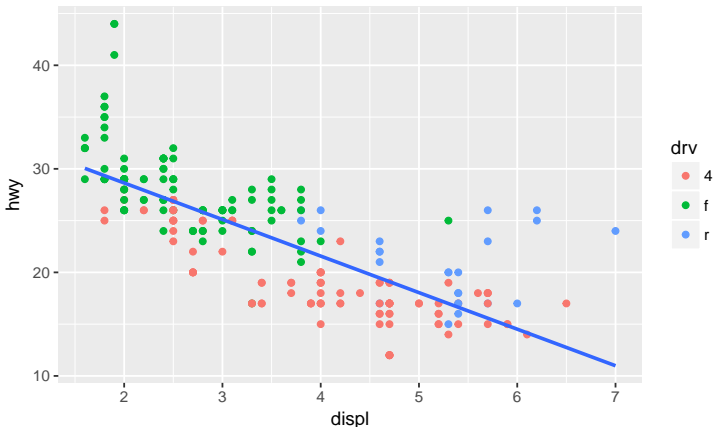
```
ggplot(data = mpg, mapping = aes(displ, hwy)) +  
  geom_point(colour = "red", aes(size = cyl)) +  
  stat_smooth(geom = "point", method = "lm", se = FALSE  
    , colour = "blue", shape = 2)
```



# Réaliser des graphiques avec ggplot2

## Le fonctionnement en « couches » de ggplot2

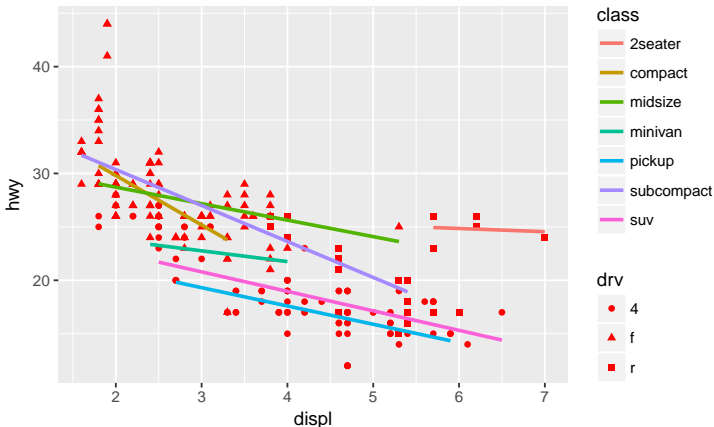
```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point(aes(colour = drv)) +  
  stat_smooth(method = "lm", se = FALSE)
```



# Réaliser des graphiques avec ggplot2

## Le fonctionnement en « couches » de ggplot2

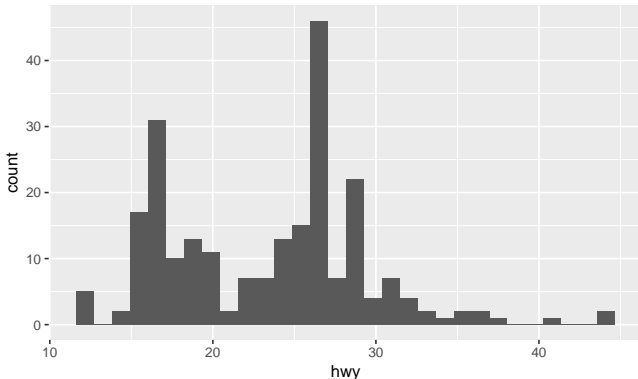
```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point(aes(shape = drv), colour = "red") +  
  stat_smooth(aes(colour = class), method = "lm", se = FALSE)
```



# Réaliser des graphiques avec ggplot2

## Histogrammes et densités

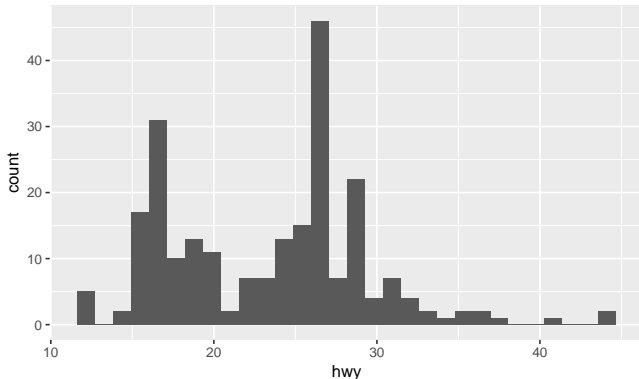
```
ggplot(mpg, aes(hwy)) + geom_histogram()
```



# Réaliser des graphiques avec ggplot2

## Histogrammes et densités

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

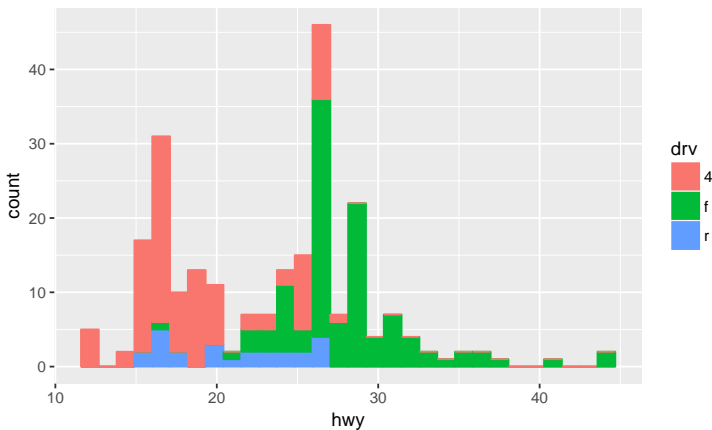


**Remarque** Le positionnement des classes des histogrammes semble perturbé dans les dernières versions de ggplot2 : le paramètre `boundary` permet de corriger ce problème (*cf.* [cette discussion](#)).

# Réaliser des graphiques avec ggplot2

## Histogrammes et densités

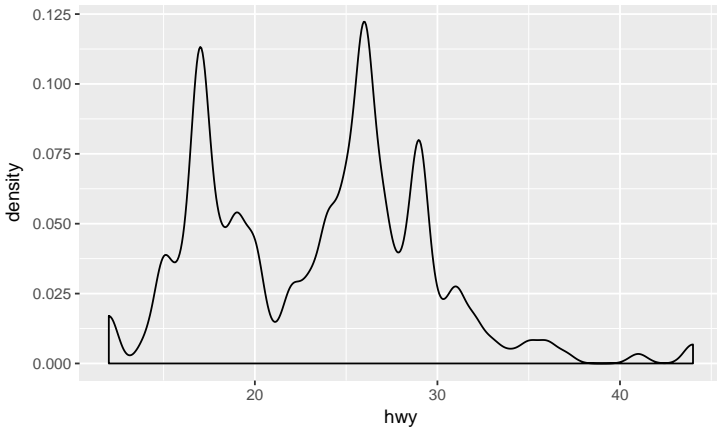
```
ggplot(mpg, aes(hwy, colour = drv, fill = drv)) +  
  geom_histogram()
```



# Réaliser des graphiques avec ggplot2

## Histogrammes et densités

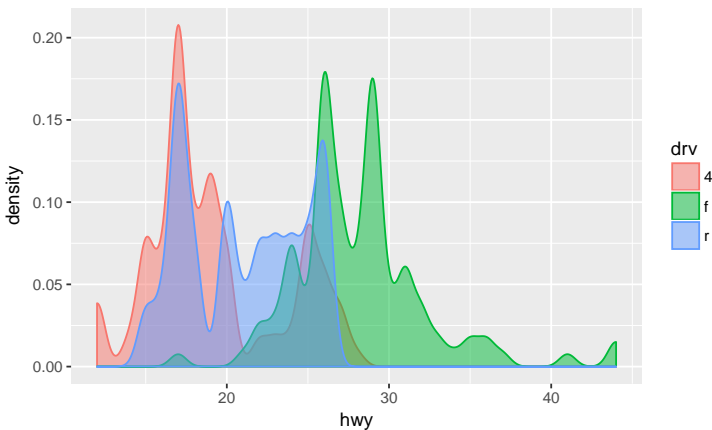
```
ggplot(mpg, aes(hwy)) + geom_density(bw = 0.5)
```



# Réaliser des graphiques avec ggplot2

## Histogrammes et densités

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

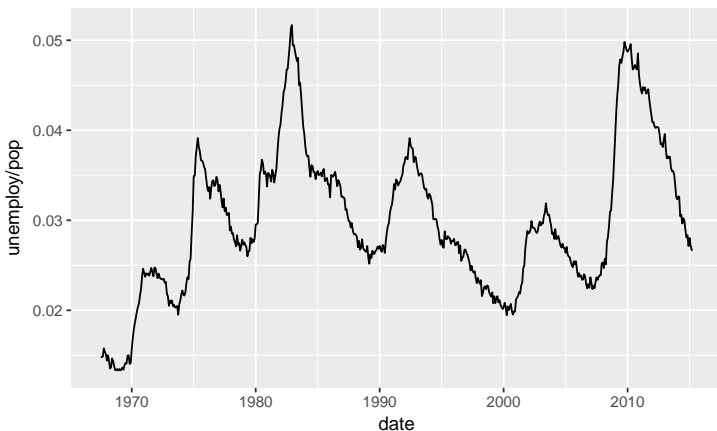




# Réaliser des graphiques avec ggplot2

## Séries temporelles

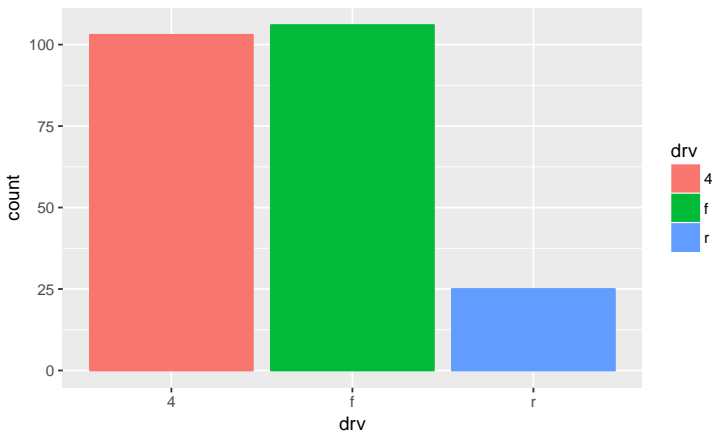
```
ggplot(economics, aes(date, unemploy / pop)) +  
  geom_line()
```



# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

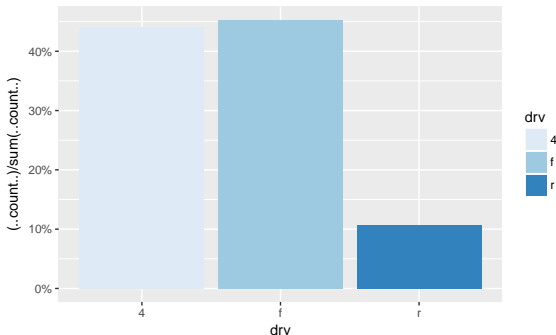
```
ggplot(mpg, aes(drv, colour = drv, fill = drv)) +  
  geom_bar()
```



# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

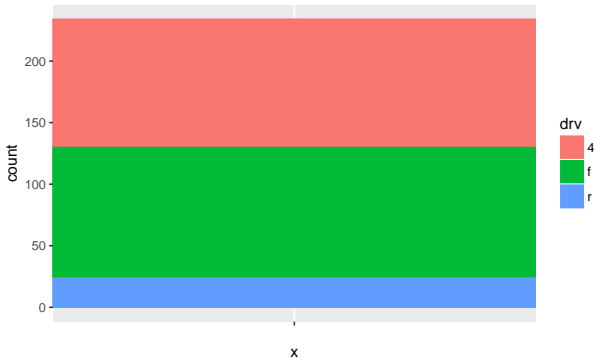
```
library(scales)
ggplot(mpg, aes(drv, fill = drv)) +
  geom_bar(aes(y = (..count..)/sum(..count..))) +
  scale_y_continuous(labels=percent) +
  scale_fill_brewer(palette="Blues")
```



# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

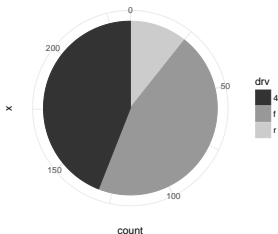
```
g <- ggplot(mpg, aes(x = "", fill = drv, colour = drv)) +  
  geom_bar(width = 1)  
g
```



# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

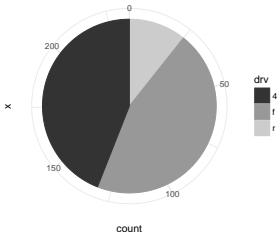
```
g + coord_polar(theta = "y") + theme_minimal() +  
  scale_fill_grey() + scale_colour_grey()
```



# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

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

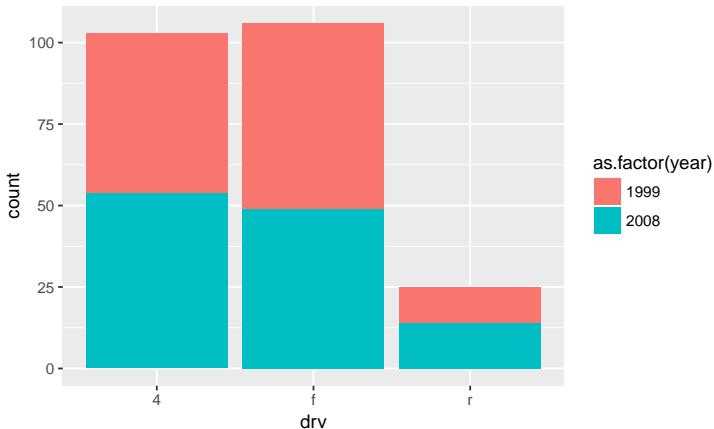


**Pour aller plus loin** Une page du site [sthda.com](http://sthda.com) explique (en français) comment produire un diagramme circulaire complet avec ggplot2.

# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

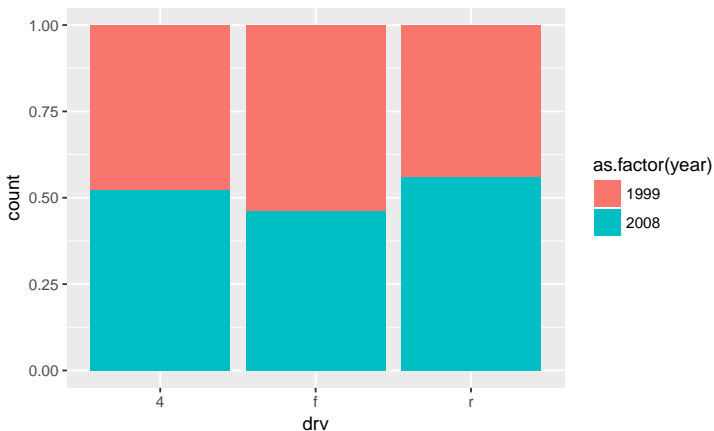
```
ggplot(mpg, aes(drv, fill = as.factor(year))) +  
  geom_bar()
```



# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

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

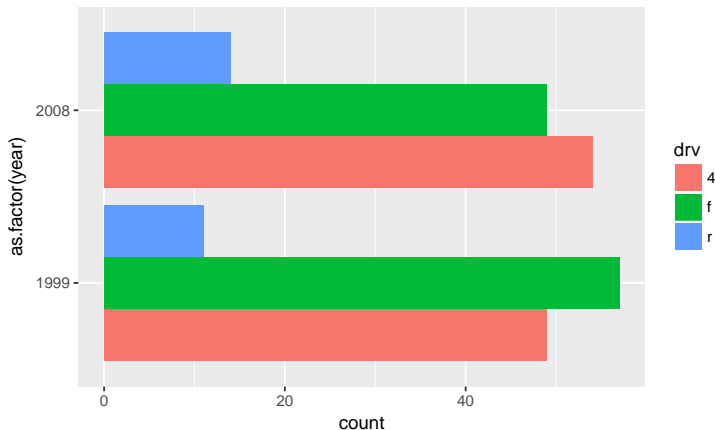




# Réaliser des graphiques avec ggplot2

## Diagrammes en bâtons et circulaires

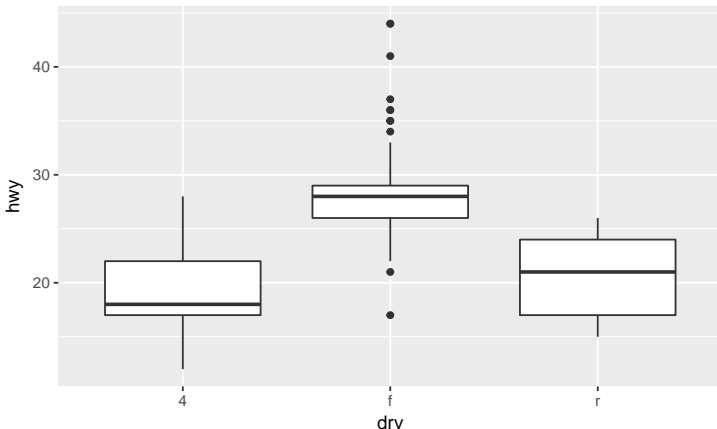
```
ggplot(mpg, aes(as.factor(year), fill = drv)) +  
  geom_bar(position = "dodge") +  
  coord_flip()
```



# Réaliser des graphiques avec ggplot2

## Boîtes à moustaches et assimilés

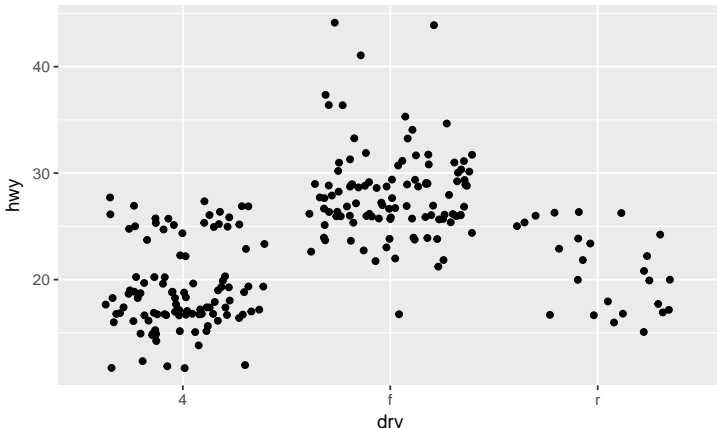
```
ggplot(mpg, aes(x = drv, y = hwy)) +  
  geom_boxplot(coef = 1.5)
```



# Réaliser des graphiques avec ggplot2

## Boîtes à moustaches et assimilés

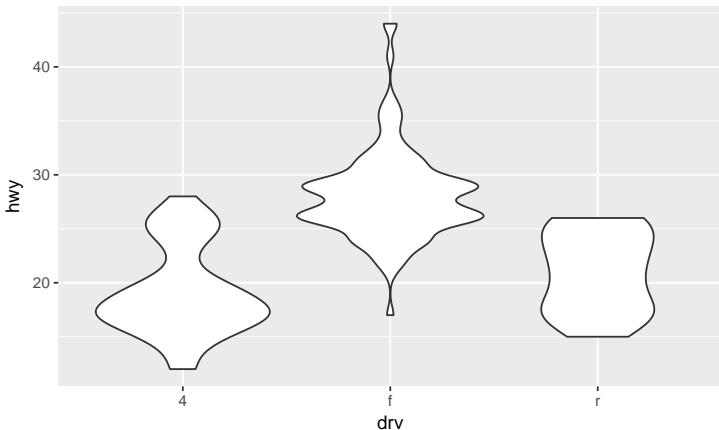
```
ggplot(mpg, aes(x = drv, y = hwy)) +  
  geom_jitter()
```



# Réaliser des graphiques avec ggplot2

## Boîtes à moustaches et assimilés

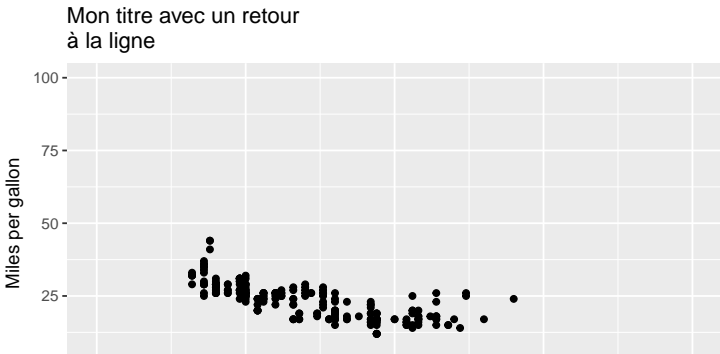
```
ggplot(mpg, aes(x = drv, y = hwy)) +  
  geom_violin()
```



# Réaliser des graphiques avec ggplot2

## Titres et axes

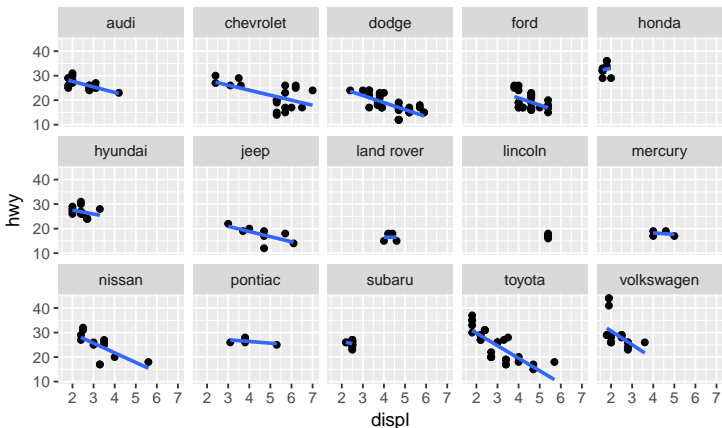
```
ggplot(mpg, aes(displ, hwy)) + geom_point() +  
  ggtitle("Mon titre avec un retour \nà la ligne") +  
  xlab("Cylindrée") + ylab("Miles per gallon") +  
  coord_cartesian(xlim = c(0,10), ylim = c(0, 100))  
## Warning in grid.Call(L_stringMetric, as.graphicsAnnot(x  
## $label)): taille de police inconnue pour le caractère  
## 0x17
```



# Réaliser des graphiques avec ggplot2

## Disposition : le *facetting*

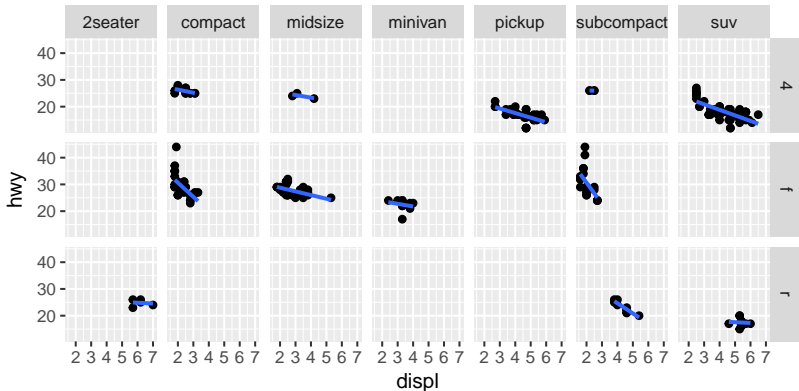
```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point() + geom_smooth(method = "lm", se = FALSE) +  
  facet_wrap(~manufacturer, nrow = 3)
```



# Réaliser des graphiques avec ggplot2

## Disposition : le *facetting*

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



## Réaliser des graphiques avec ggplot2

### 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")
```



## Réaliser des graphiques avec ggplot2

### 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")
```