Compléments de R

Programmation et Logiciels Statistiques

Références

- *R pour les Data Sciences*, H. Wickham et G. Gorlemund, Editions Eyrolles 2018.
- https://ggplot2.tidyverse.org
- https://thinkr.fr/pdf/ggplot2-french-cheatsheet.pdf
- https://www.r-graph-gallery.com

Le tidyverse

- Rappelons qu'un « package » est une collection de fonctions, données et documentation permettant d'étendre les capacités de R.

- La plupart des packages qu'on utilisera dans la suite font partie d'un ensemble appelé *tidyverse*.

- Ils partagent la même approche des données et de la programmation en R et sont conçus pour fonctionner les uns avec les autres.

Le tidyverse

- On peut installer l'ensemble de ces packages à l'aide d'une seule ligne de code :

```
install.packages(« tidyverse »)
```

- Rappelons qu'on ne peut utiliser les fonctions, objets et fichiers d'aide d'un package avant les avoir chargés avec la commande `library()`.

Le tidyverse

```
> library(tidyverse)
```

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1 ──
✔ ggplot2 3.1.0     ✔ purrr   0.2.5
✔ tibble  1.4.2     ✔ dplyr   0.7.8
✔ tidyr   0.8.2     ✔ stringr 1.3.1
✔ readr   1.2.1     ✔ forcats 0.3.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ .GlobalEnv::alpha() masks ggplot2::alpha()
✖ dplyr::filter()     masks stats::filter()
✖ dplyr::lag()        masks stats::lag()
```

Le tidyverse

- Cela indique que le tidyverse charge les packages ggplot2, tibble, tidyr, readr, purrr et dplyr.

- Il sont considérés comme le coeur (ang. core) du tidyverse.

- Les packages du tidyverse changent assez fréquemment.

- On peut vérifier si des mises à jour sont disponibles et les installer en exécutant `tidyverse_update()`.

ggplot2

Graphiques avec ggplot2

- R dispose de plusieurs systèmes pour la production de graphiques mais ggplot2 est l'un des plus élégants et des plus polyvalents.

- Il implémente la *grammaire de graphique*, décrite dans l'article « A Layered Grammar of Graphics » de Hadley Wickham, disponible sur le forum.

Graphiques avec ggplot2

- Pour accéder aux jeux de données, pages d'aide et fonctions que nous utiliserons, on commence par charger le tidyverse à l'aide du code :

```
> library(tidyverse)
```

- Lorsqu'il sera nécessaire de préciser la provenance d'une fonction ou d'un jeu de données, nous utiliserons la syntaxe package::fonction, par exemple ggplot2::ggplot.

Premiers pas

- Commençons par construire un graphe visant à répondre à la question suivante : les voitures ayant de plus gros moteurs consomment-elles plus d'essence ?

- A cet effet, on va utiliser le jeu de données mpg, inclus dans ggplot2.

- mpg contient des observations collectées par l'Agence de Protection Environnementale des Etats-Unis sur 38 modèles de voitures.

Premiers pas

```
> mpg
# A tibble: 234 x 11
   manufacturer model      displ  year   cyl trans      drv      cty   hwy fl    class
   <chr>        <chr>      <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
 1 audi         a4           1.8  1999     4 auto(l5)   f        18    29 p     compa…
 2 audi         a4           1.8  1999     4 manual(m…  f        21    29 p     compa…
 3 audi         a4           2    2008     4 manual(m…  f        20    31 p     compa…
 4 audi         a4           2    2008     4 auto(av)   f        21    30 p     compa…
 5 audi         a4           2.8  1999     6 auto(l5)   f        16    26 p     compa…
 6 audi         a4           2.8  1999     6 manual(m…  f        18    26 p     compa…
 7 audi         a4           3.1  2008     6 auto(av)   f        18    27 p     compa…
 8 audi         a4 quatt…    1.8  1999     4 manual(m…  4        18    26 p     compa…
 9 audi         a4 quatt…    1.8  1999     4 auto(l5)   4        16    25 p     compa…
10 audi         a4 quatt…    2    2008     4 manual(m…  4        20    28 p     compa…
# ... with 224 more rows
```

Premiers pas

- Les variables comprennent notamment :

- `displ` : la taille du moteur d'une voiture, en litres

- `hwy` : l'efficacité énergétique de la voiture dur autoroute, en miles per gallon (mpg).

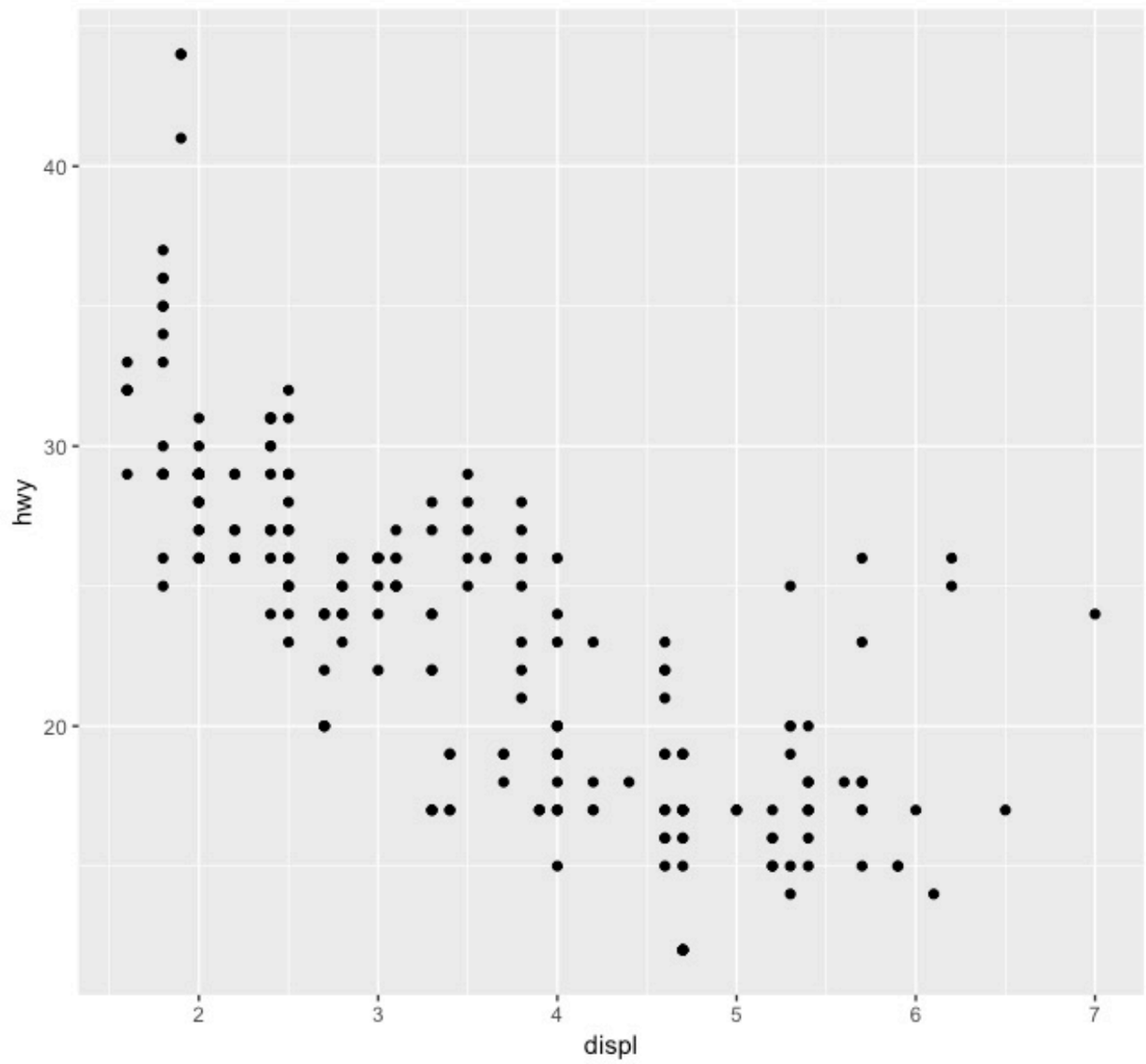
- Plus l'efficacité énergétique d'une voiture est élevée, moins elle consomme de carburant pour une distance donnée.

Premiers pas

- Commençons par représenter le nuage de points qui place displ en abscisse et hwy en ordonnée :

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

Premiers pas



Premiers pas

- Le graphique met en évidence une relation négative entre la taille du moteur (`displ`) et l'efficacité énergétique (`hwy`).

- Les voitures ayant de plus gros moteurs consomment plus de carburant.

Premiers pas

- Pour initier un graphique avec ggplot2, on utilise la fonction ggplot, qui crée un système de coordonnées sur lequel des couches vont pouvoir être ajoutées.

- Son premier argument est le jeu de données à utiliser.

- `ggplot(data=mpg)` crée en fait un graphique vide.

Premiers pas

- Pour compléter le graphique, on doit ajouter une ou plusieurs couches dans l'appel à ggplot.

- La fonction geom_point ajoute une couche de points.

- ggplot2 inclut de nombreuses *fonctions de géomes*, qui ajoutent chacune une couche d'un type distinct au graphique.

Premiers pas

- Chaque fonction de géome de ggplot2 utilise un *argument de liaison* qui définit des relations entre les variables du jeu de données et des propriétés visuelles.

- Il est intégré à `aes()`.

- Les arguments `x` et `y` de `aes()` indiquent les variables correspondant aux axes x et y.

Premiers pas

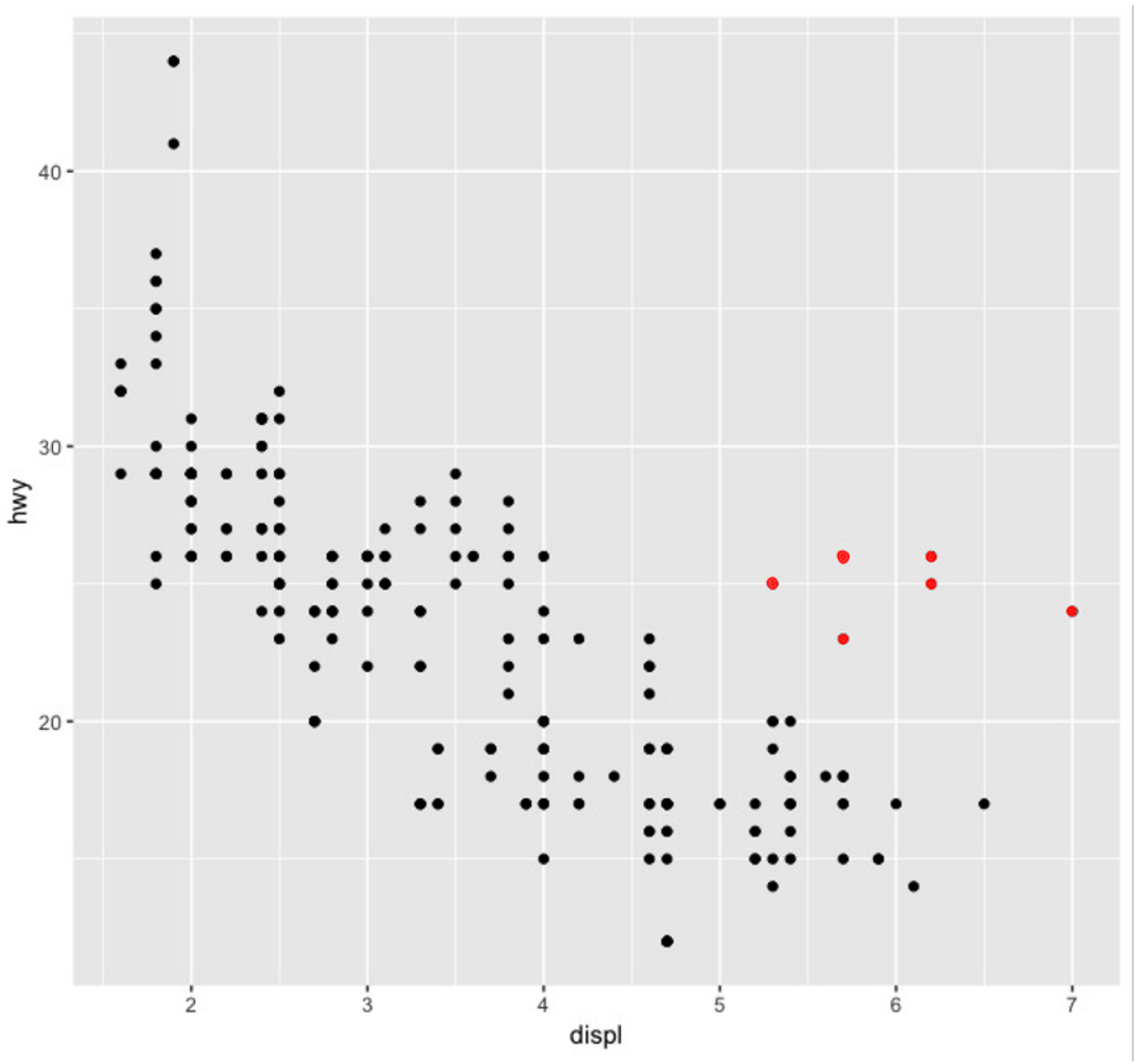
- Une syntaxe générale permettant d'obtenir un graphique avec ggplot2 est la suivante :

```
ggplot(data= < DONNEES >) +
  <FONCTION_GEOME> (mapping = aes(<LIAISONS>))
```

- On verra comment compléter et étendre ce modèle de syntaxe pour créer différents types de graphiques avec ggplot2.

- Commençons par reprendre le graphe précédent.

Liaisons



Liaisons

- Sur le graphique, le groupe de points en rouge semble s’écarter de la tendance linéaire : ces voitures sont plus économes que la normale.

- Comment l’expliquer ?

- On émet l’hypothèse que ces voitures sont **hybrides.**

- Pour la tester, on étudie la classe des voitures, donnée par la variable `class` du jeu de données `mpg`.

Liaisons

- On peut ajouter une troisième variable à un nuage de points - dans notre cas, la classe du véhicule - en lui faisant correspondre une *esthétique*.

- Une *esthétique* est une **propriété visuelle** des objets du graphe : taille, forme ou couleur des points par exemple.

- Un point peut être représenté différemment selon les valeurs de ses propriétés esthétiques.

- Le terme de « valeur » étant utilisé pour décrire les données, on utilise celui de « niveau » pour les propriétés esthétiques.

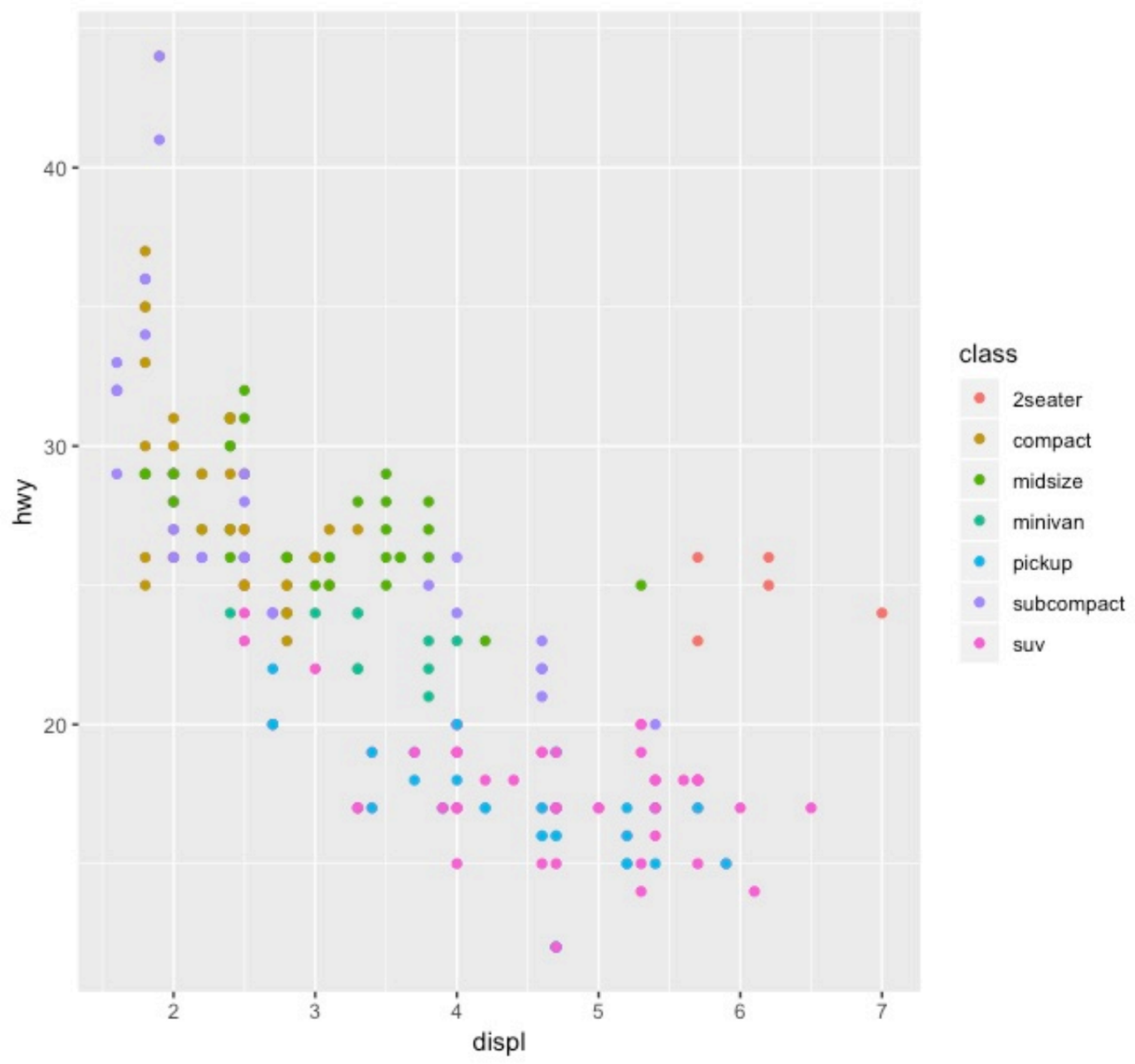
Liaisons

- On peut par exemple utiliser la couleur des points pour révéler la classe de chaque voiture :

```
ggplot(data=mpg) +

  geom_point(mapping = aes(x=displ,y=hwy, color=class))
```

- Le graphique obtenu est le suivant :

Liaisons


40
30
20
hwy
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

Liaisons

- Pour relier une esthétique à une variable, on associe dans `aes()` le nom de l'esthétique à celui de la variable.

- ggplot2 affecte automatiquement à chaque valeur différente de la variable un niveau distinct de l'esthétique (ici, une couleur), par un procédé nommé échelonnage (ang. scaling).

- Il ajoute en plus une légende pour indiquer à quelle valeur correspond chaque niveau.

Liaisons

- Les couleurs nous révèlent que la plupart des points anormaux représentent des voitures à deux places.

- Ce ne sont donc probablement pas hybrides, mais plutôt des voitures de sport !

- Celles-ci sont de taille similaire aux voitures compactes et moyennes, mais elles possèdent des moteurs de grand volume, comparables à ceux des SUV et des pick-ups, ce qui améliore leur efficacité relative.

- Rétrospectivement, on réalise que notre hypothèse de départ n’était pas très pertinente car les voitures hybrides ont rarement de gros moteurs.

Liaisons

- Dans l'exemple précédent, on a choisi comme esthétique la couleur des points, mais ont aurait également pu choisir leur taille. Dans ce cas c'est la taille de chaque point qui indique la classe à laquelle il appartient.

- Cela déclenche toutefois un avertissement, car on essaie de faire correspondre une variable discrète (`class`) à une esthétique continue (`size`), ce qui n'est en général pas une bonne idée.

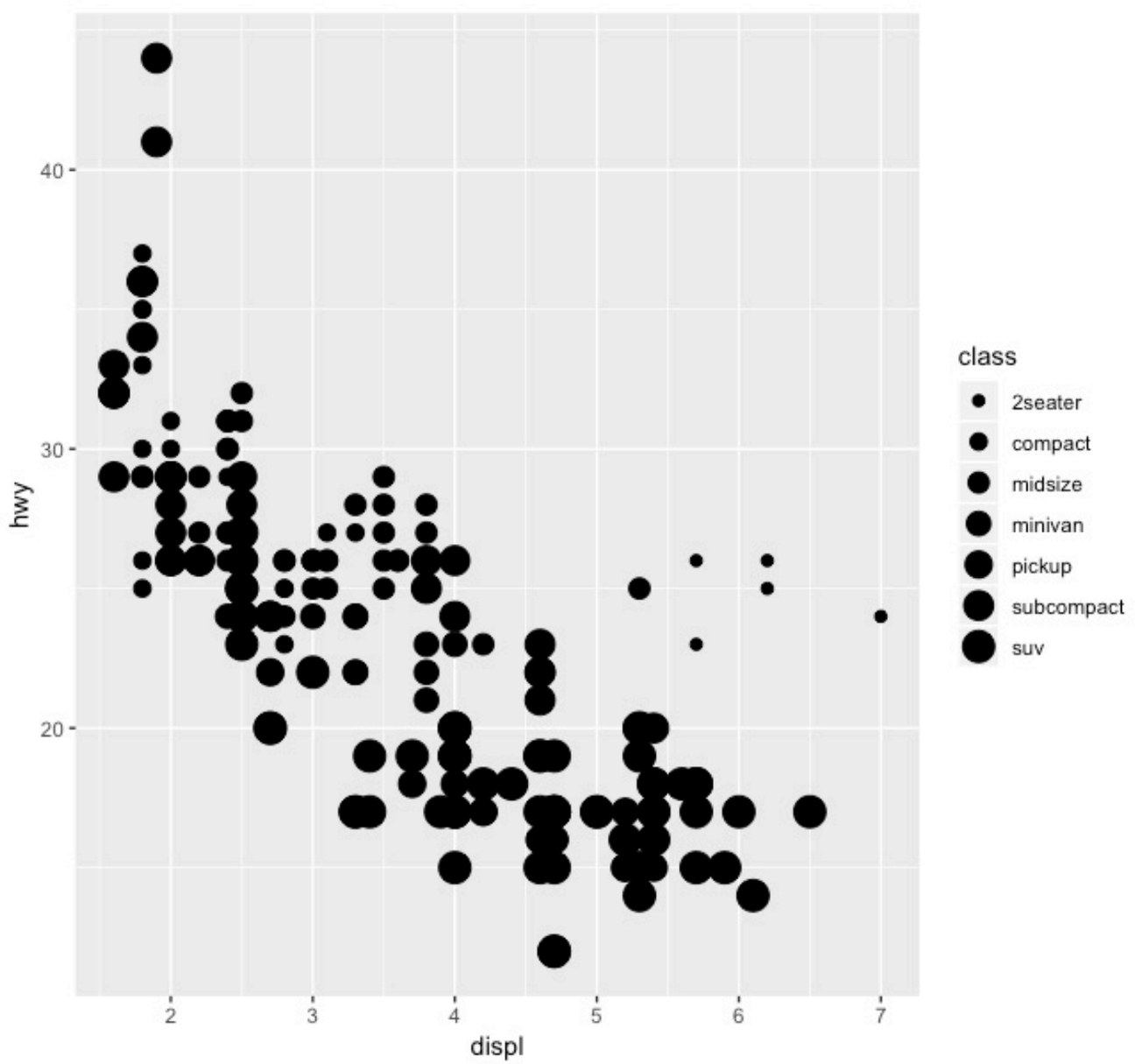
Liaisons

- Le code correspondant est le suivant :

```
ggplot(data=mpg) +

  geom_point(mapping = aes(x=displ,y=hwy, size=class))
```

- On obtient le graphique suivant :

Liaisons


40
30
20
hwy
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

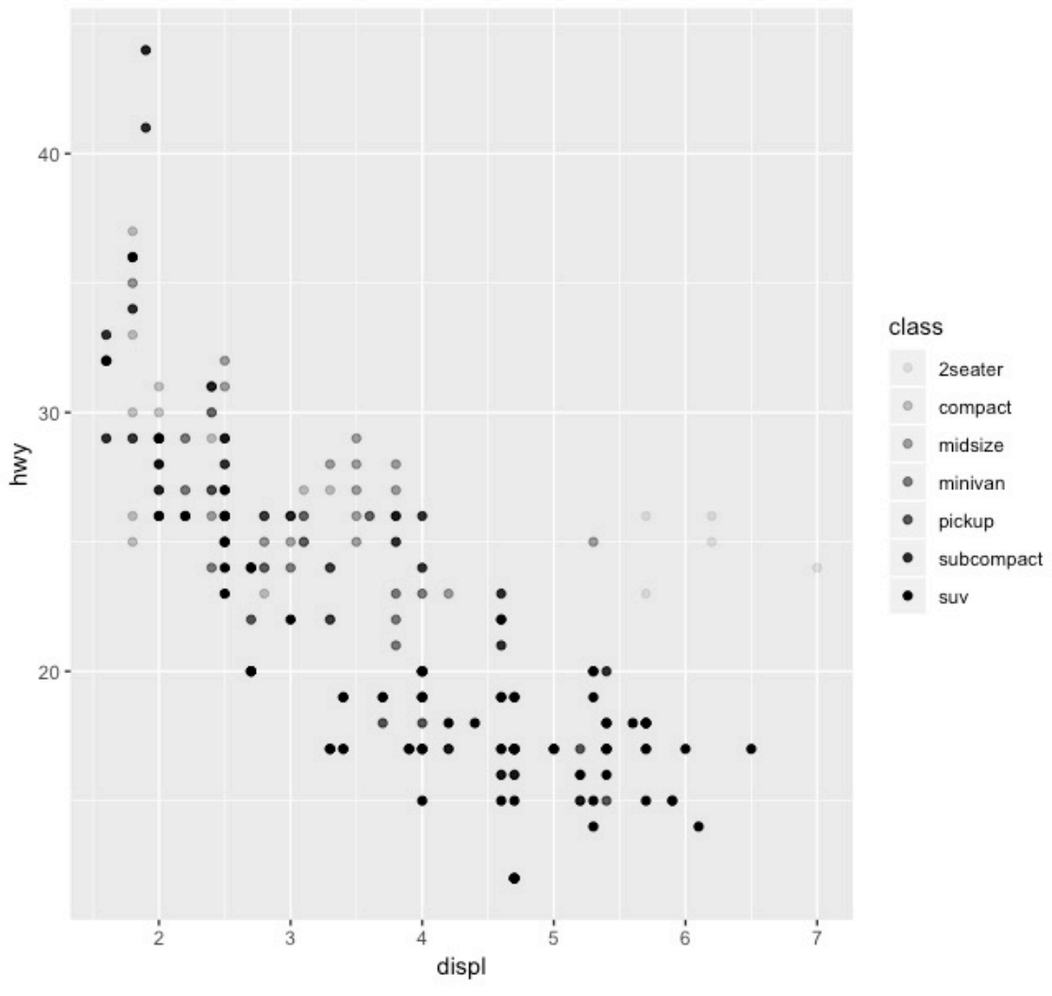
Liaisons

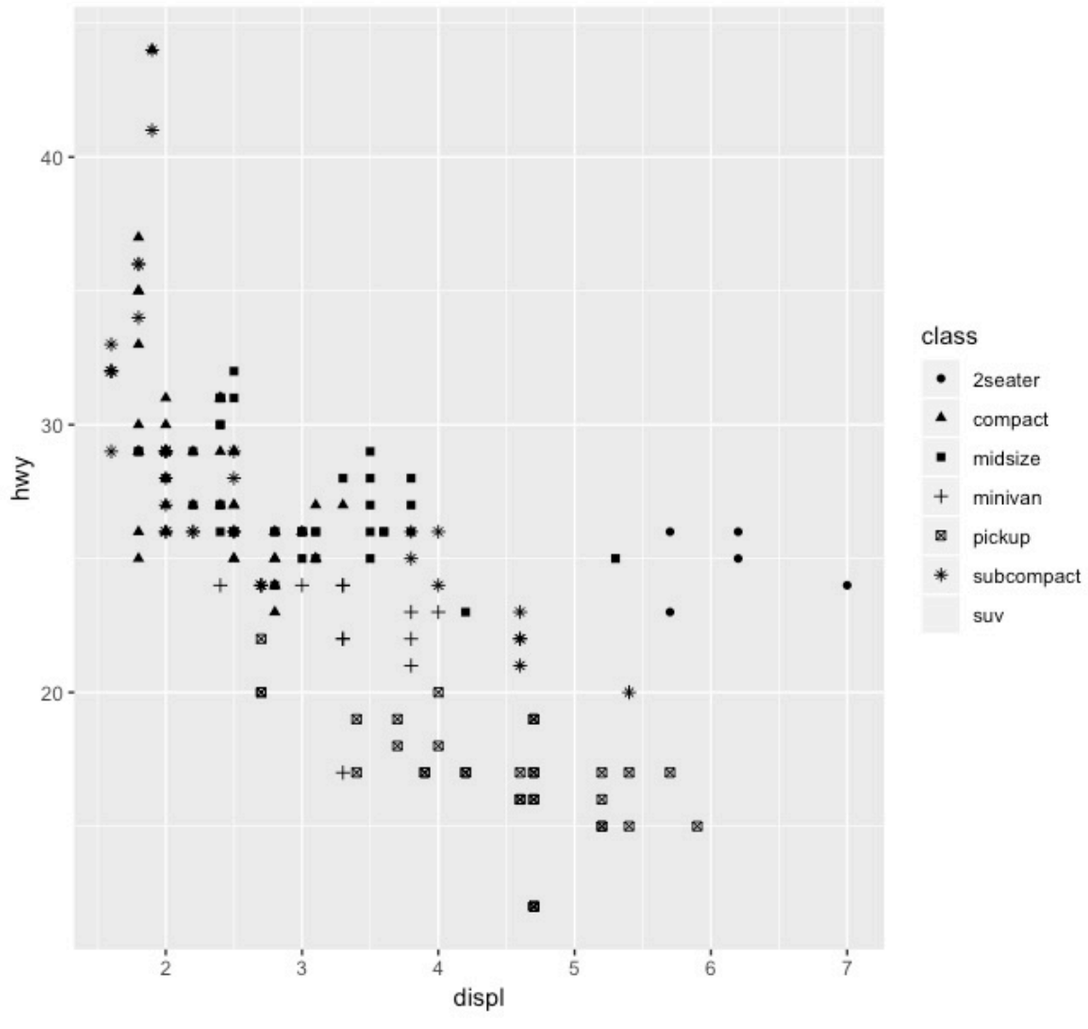
- On aurait pu également utiliser des esthétiques déterminant la forme des points (**`shape`**) ou leur transparence (**`alpha`**) :

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

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

Liaisons


40
30
20
hwy
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv


40
30
20
hwy
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

Liaisons

- Dans le deuxième graphique (esthétique : `shape`), les SUV n'apparaissent pas.

- En effet, ggplot2 n'utilise au plus que 6 formes différentes sur un même graphique. Par défaut les groupes supplémentaires ne sont pas affichés lorsqu'on utilise cette esthétique.

Liaisons

- Le nom de chaque esthétique qu'on souhaite utiliser est associé dans `aes()` à une variable à représenter.

- La fonction `aes()` rassemble toutes les liaisons impliquant les esthétiques utilisées par une couche et les lui transmet en tant qu'argument de liaison.

- La syntaxe utilisée met en évidence le fait que les coordonnées `x` et `y` sont aussi des esthétiques, c'est-à-dire des propriétés visuelles reliées à des variables afin de représenter des informations sur les données.

Liaisons

- Lorsqu'on établit une liaison pour une esthétique, ggplot2 se charge des détails : il sélectionne une échelle raisonnable et construit une légende présentant la relation entre les niveaux et les valeurs.

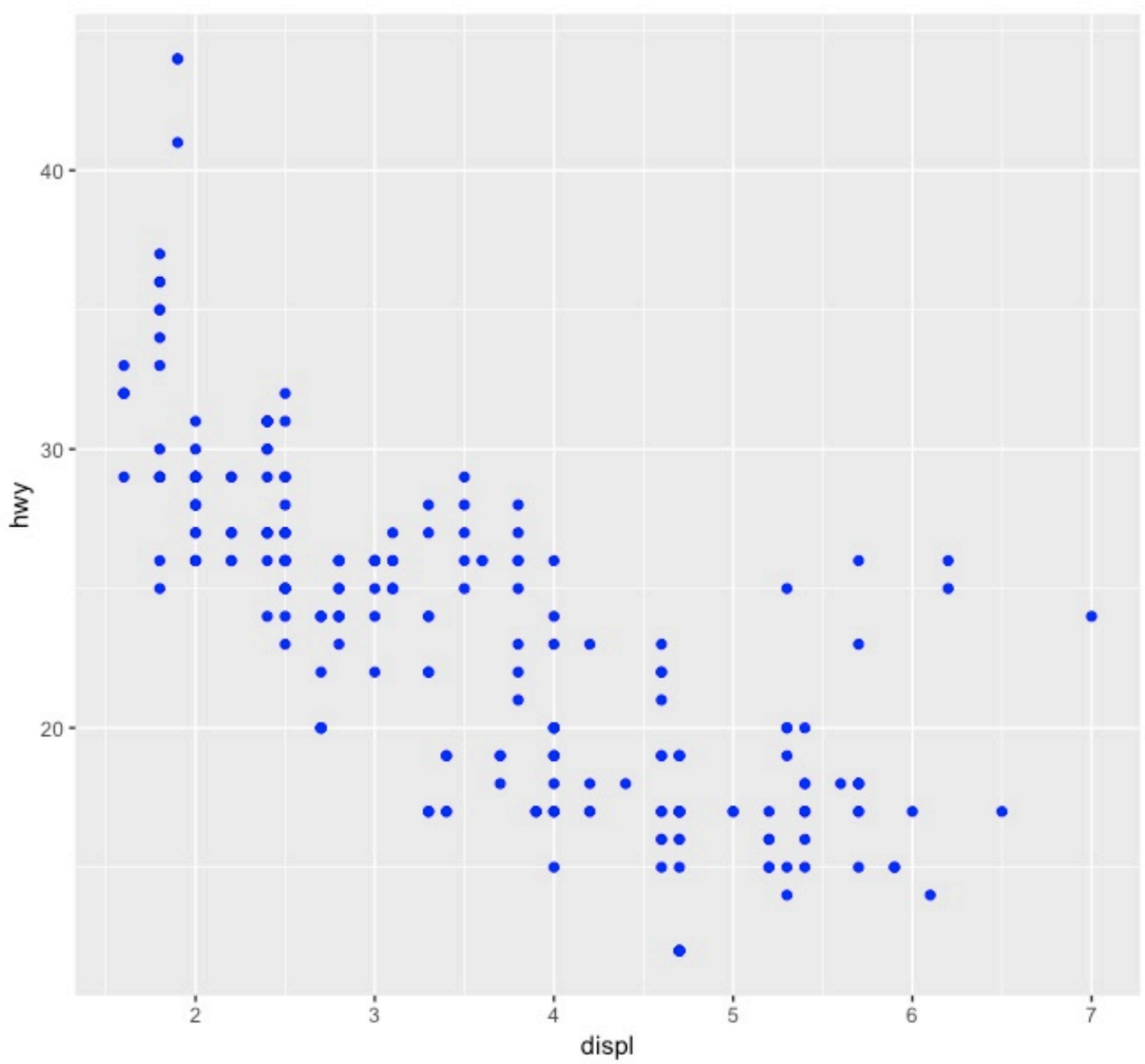
- Pour les esthétiques `x` et `y`, ggplot2 ne crée pas de légende mais un axe doté d'un nom et de graduations, qui remplit le même rôle : il expose la relation entre les emplacements des points et les valeurs.

Liaisons

- Il est possible de définir *manuellement* les propriétés esthétique des géomes.
- On peut par exemple afficher tous les points en bleu :

```
ggplot(data=mpg) +

  geom_point(mapping = aes(x=displ,y=hwy), color="blue")
```

Liaisons



Liaisons

- Ici la couleur *ne transmet pas d'information sur une variable* ; elle ne fait que *changer l'apparence* du graphique.

- Pour définir une esthétique *manuellement*, on doit la passer en argument de la fonction de géome, donc *en dehors* de `aes()`.

- On doit choisir une valeur pertinente pour l'esthétique en question :

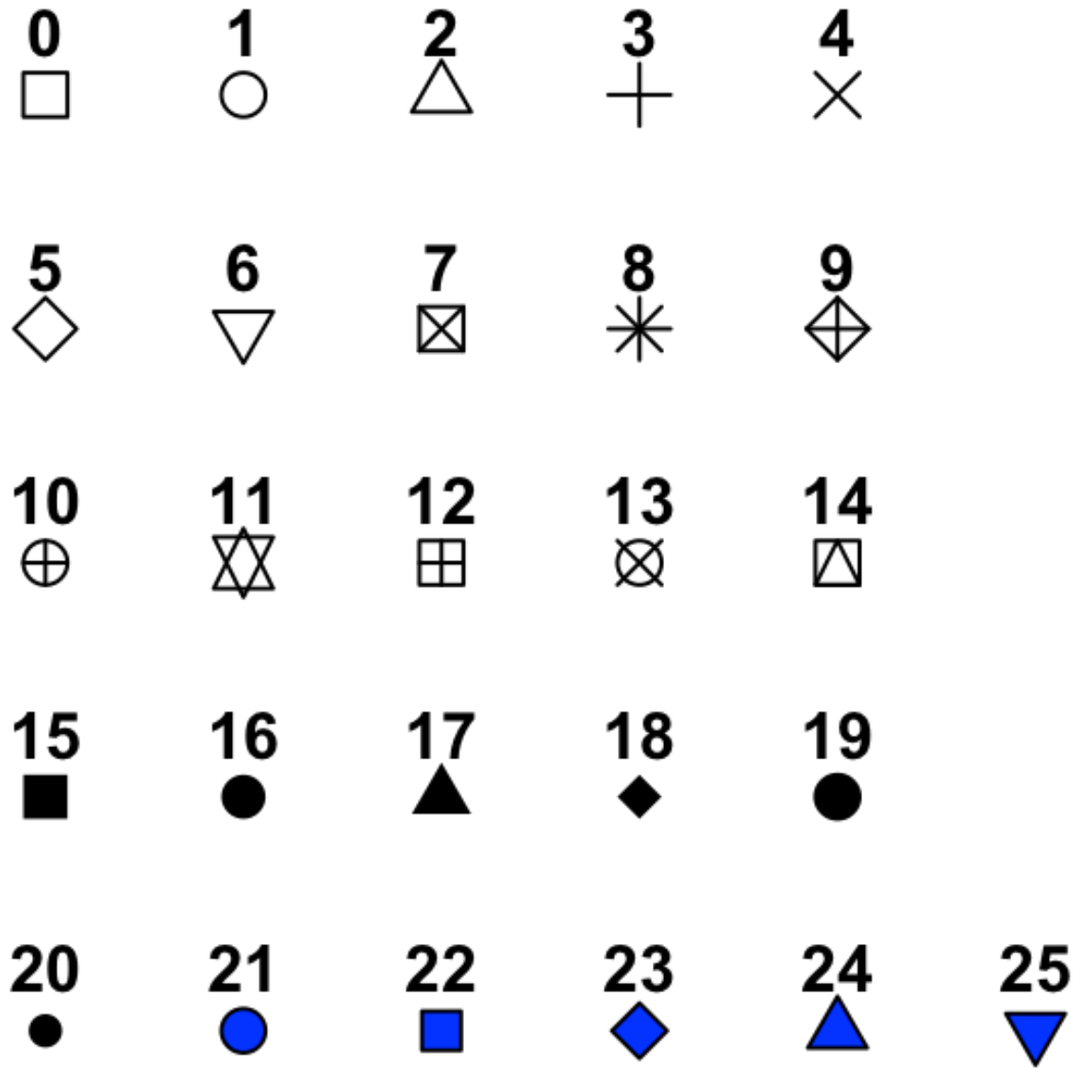
Liaisons

- Le nom d’une couleur, sous la forme d’une chaîne de caractères.

- La taille d’un point en mm.

- La forme d’un point sous la forme d’un numéro, comme illustré dans la figure suivante :

Liaisons



Liaisons

- Certaines formes semblent identiques ; par exemple les formes 0, 15 et 22 sont toutes des carrés.

- La différence réside dans l'interaction avec les esthétiques de couleur (`color`) et de remplissage (`fill`).

- Les formes vides (0-14) ont une bordure en couleur ; les formes solides (15-20) sont entièrement coloriées ; les formes pleines (21-25) ont la bordure d'une couleur et l'intérieur d'une autre.

- Dans ce dernier cas, le deuxième couleur est déterminée par l'esthétique `fill`.

Questions

- Que se passe-t-il si on relie une même variable à plusieurs esthétiques ?

- Que fait l'esthétique `stroke` ?

- Que se passe-t-il si on relie une esthétique avec autre chose que le nom d'une variable ? Par exemple,

```
aes(color = displ < 5 )
```

Facettes

- Une alternative aux esthétiques, notamment pour les variables catégorielles, est fourni par les facettes.

- Les facettes sont des sous-graphiques affichant chacun un sous-ensemble des données.

- Pour facetter un graphiques selon une seule variable, on utilise `facet_wrap()`. Son premier argument doit être une formule.

- La variable passée à `facet_wrap()` doit être discrète.

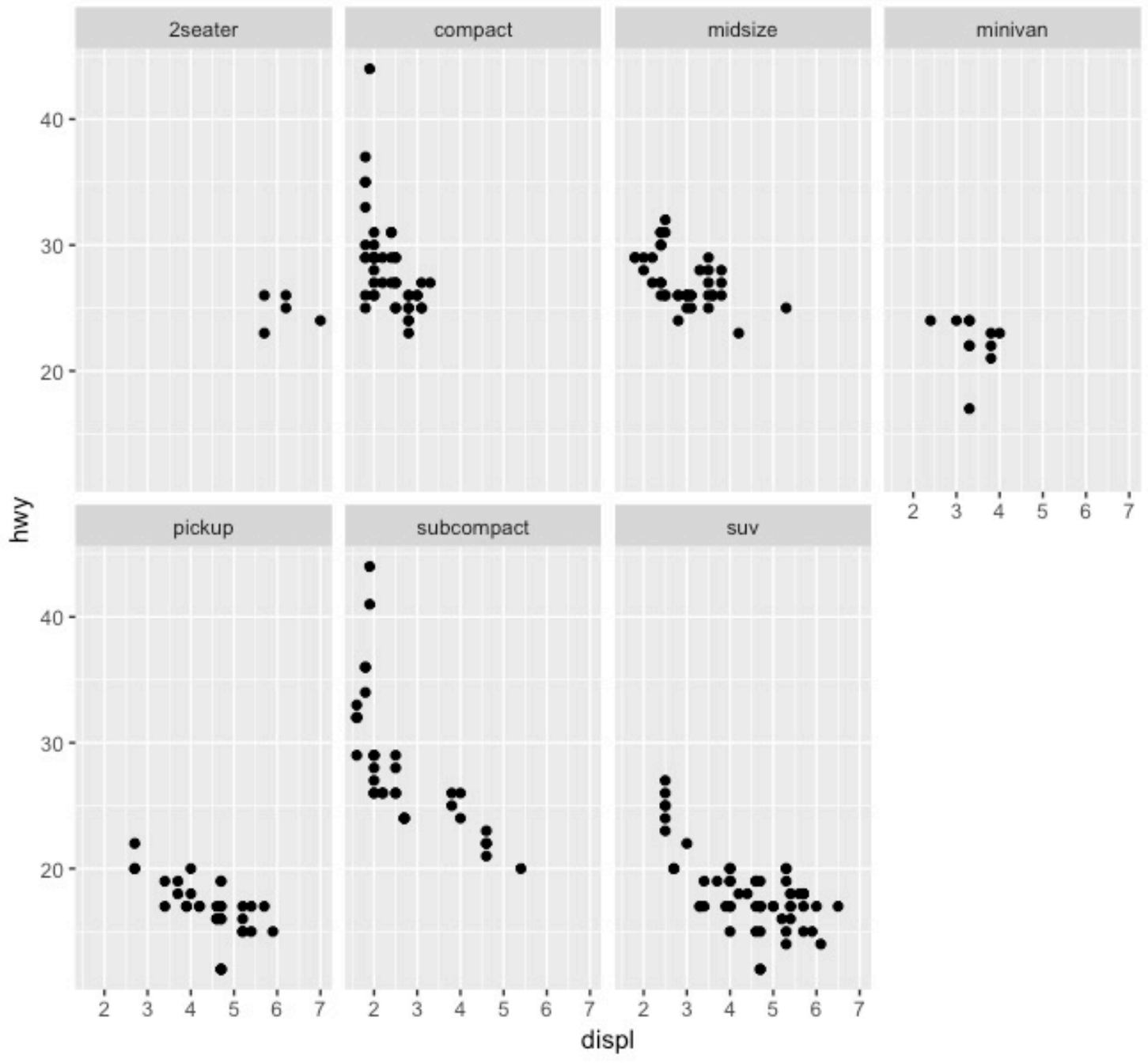
Facettes

- Ainsi, le code :

```
ggplot(data=mpg) +
  geom_point(mapping = aes(x=displ,y=hwy)) +
  facet_wrap(~class, nrow = 2)
```

génère le graphique suivant :

Facettes


2seater
compact
midsize
minivan
pickup
subcompact
suv
hwy
displ
40
30
20
2
3
4
5
6
7

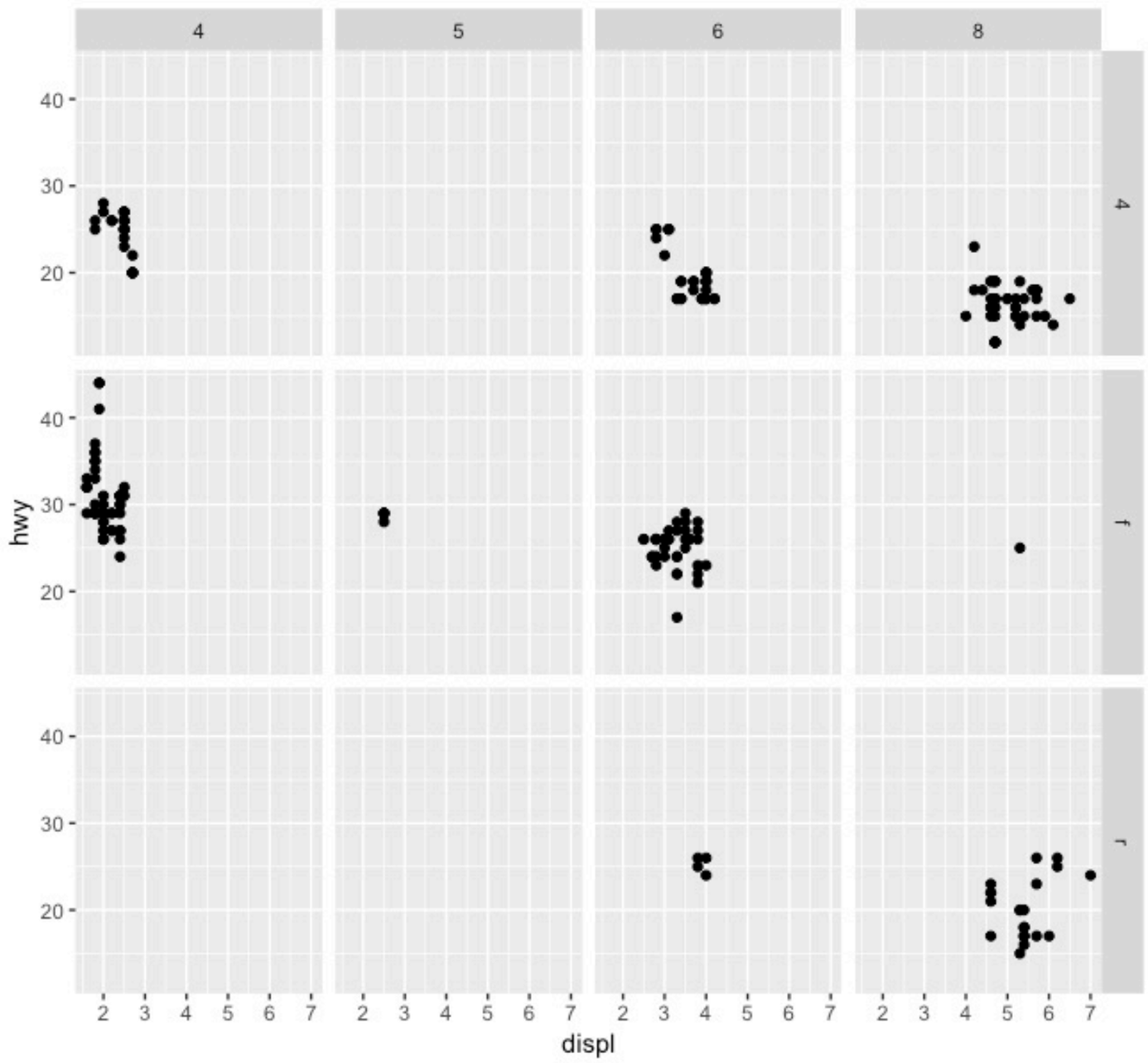
Facettes

- Pour facetter un graphique selon une combinaison de deux variables, on utilise facet_grid().

- Son premier argument est aussi une formule, qui doit contenir deux noms de variables séparés par ~ :

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

Facettes



Facettes

- Pour que le graphique ne soit facetté qu'en une dimension (ligne ou colonne), on utilise un point (.) à la place d'un des deux noms de variables :

```
+ facet_grid(. ~ cyl)
```

Questions

- Que se passe-t-il si on facette selon une variable continue ?
- Que signifient les cellules vides dans un graphe avec `facet_grid(drv~cyl)` ? A quoi correspondent-elles dans ce graphique ?

```
ggplot(data=mpg) +

  geom_point(mapping = aes(x=drv,y=cyl))
```

Objets géométriques

- Un *géome* est l'objet géométrique utilisé par un graphe pour représenter les données.

- Les graphiques sont souvent décrits par le type de géome qu'ils utilisent.

- Par exemple, les diagrammes en barres utilisent le géome `bar`, les diagramme linéaires le géome `line`, les diagrammes en boîte le géome `boxplot`, etc.

- Les nuages de points, par contre, utilisent le géome *point*.

Objets géométriques

- Les mêmes données peuvent être représentées par des géomes différents.

- Pour changer le géome d'un graphe, il suffit de modifier la fonction de géome ajoutée à `ggplot()`.

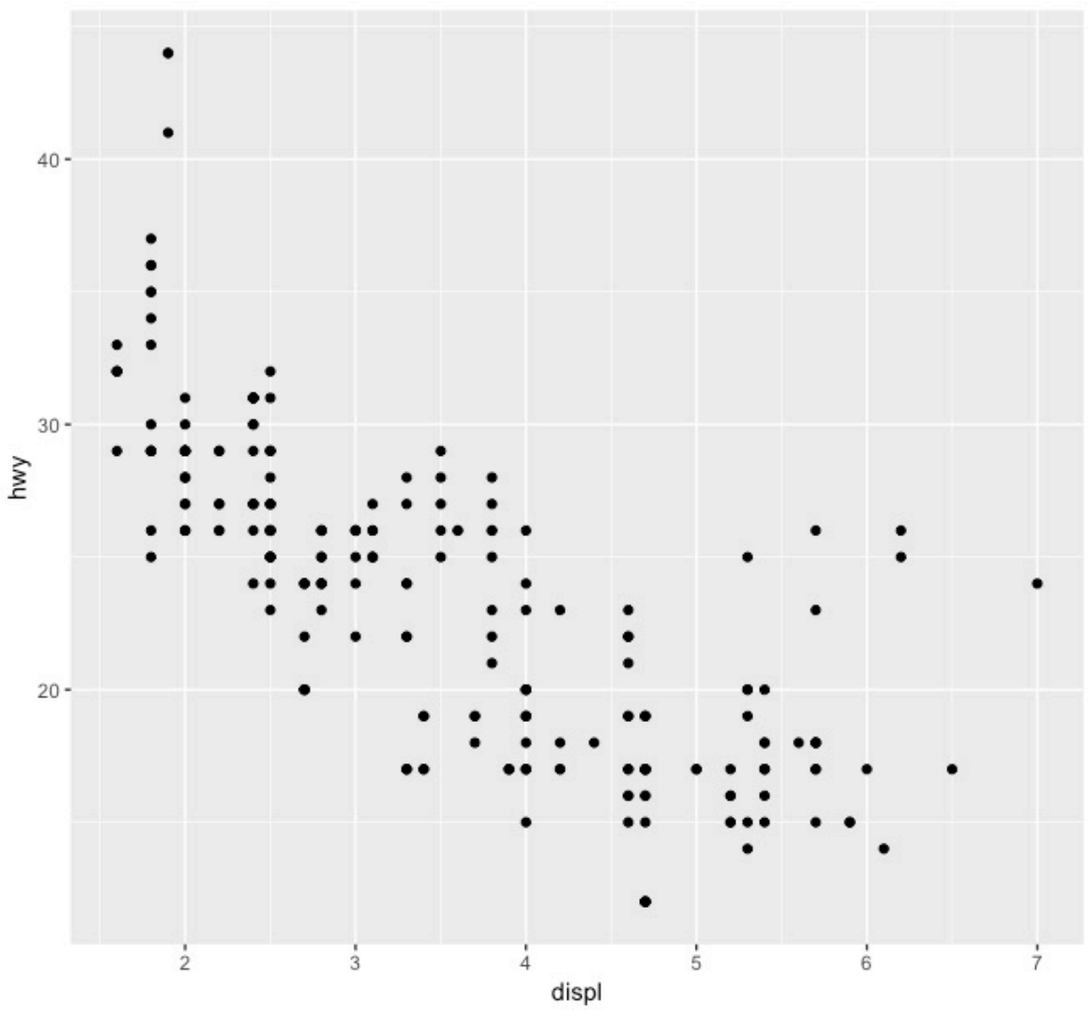
- Par exemple, les deux codes suivants construisent deux graphes différents à partir des mêmes données.

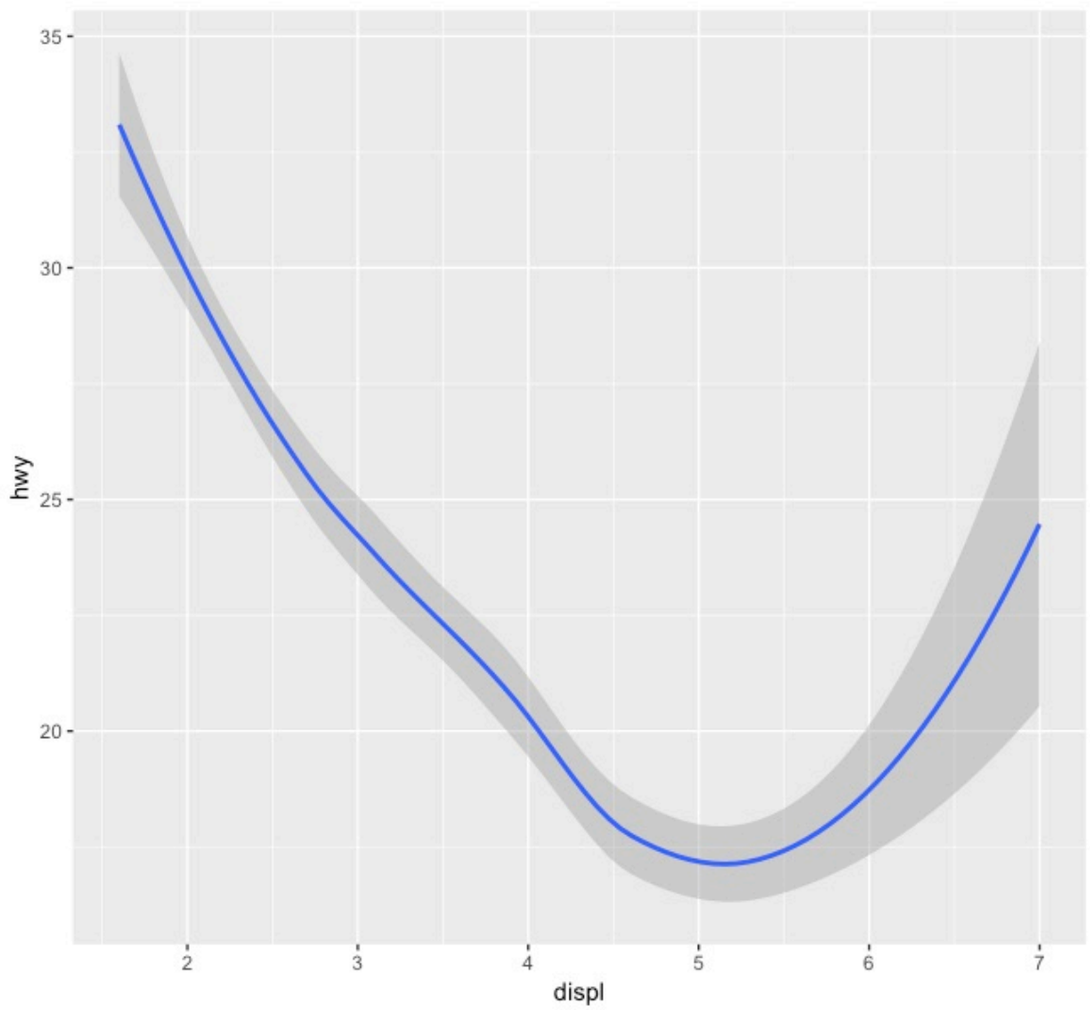
Objets géométriques

```
ggplot(data=mpg) +

  geom_point(mapping = aes(x=displ,y=hwy))
```

```
ggplot(data=mpg) +

  geom_smooth(mapping = aes(x=displ,y=hwy))
```

Objets géométriques





Objets géométriques

- Chaque fonction de géome de ggplot2 utilise un argument de liaison, mais toutes les esthétiques ne sont pas compatibles avec tous les géomes.

- Par exemple, il est possible de modifier la forme d'un point mais pas celle d'une ligne.

- Par contre, il est possible de modifier le type d'une ligne par l'esthétique `linetype`.

- Ainsi, `geom_smooth()` peut dessiner une courbe différente avec un type distinct pour chaque valeur de la variable liée à `linetype`.

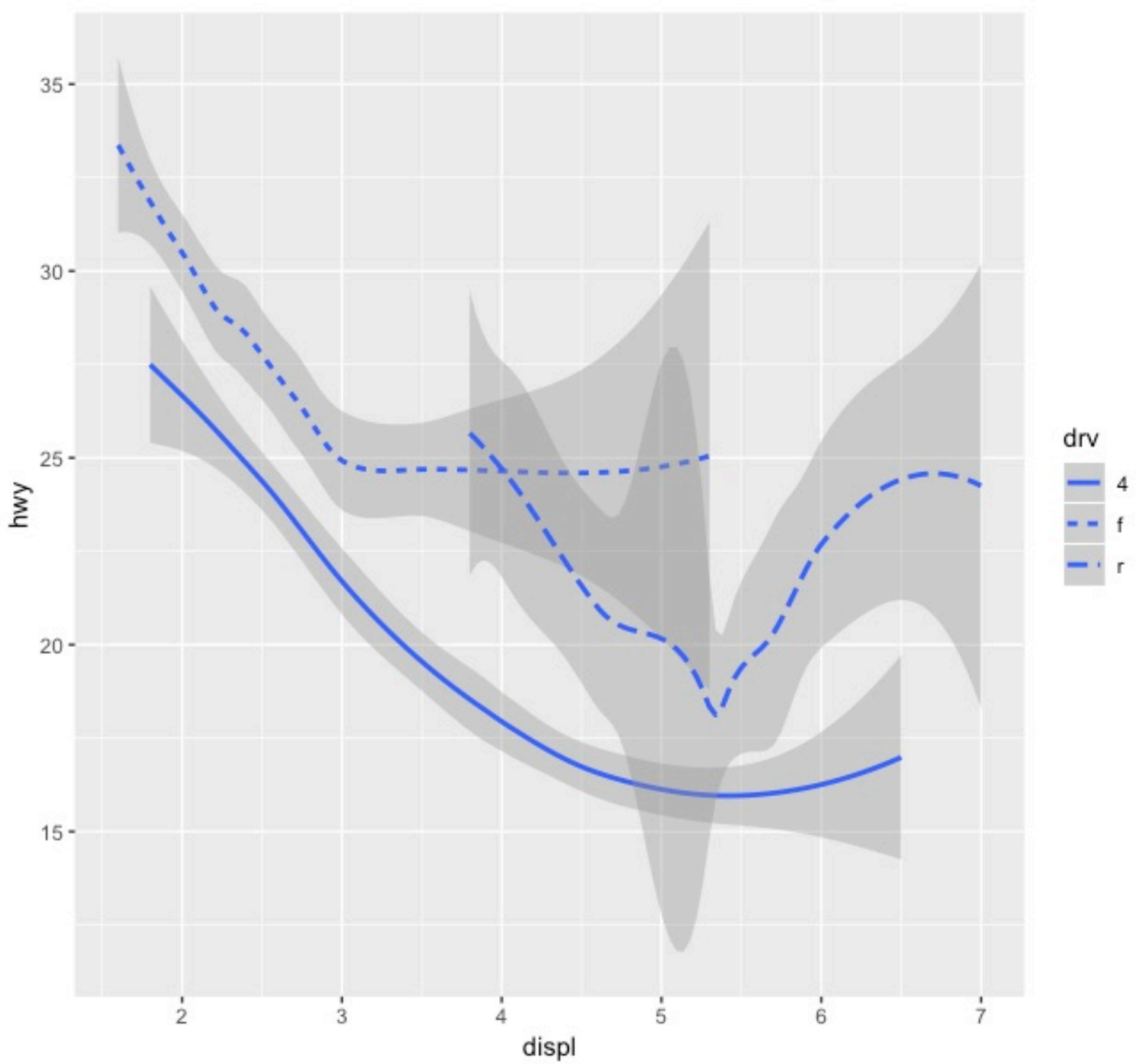
Objets géométriques

- Par exemple, le code :

```
ggplot(data=mpg) +

  geom_smooth(mapping = aes(x=displ, y=hwy, linetype=drv))
```

- produit le graphe suivant :

Objets géométriques



Objets géométriques

- Dans ce graphique, `geom_smooth()` répartit les voitures sur trois courbes, selon la valeur de la variable `drv`, qui décrit le type d'entraînement (modalité « 4 » : 4 roues motrices, « f »: traction avant (*front*), « r » : traction arrière (*rear*)).

Objets géométriques

- Le package ggplot2 fournit plus de 30 géomes et de nombreux autres sont disponibles dans des packages d'extension (http://www.ggplot2-exts.org).

- Une fiche récapitulative de ggplot2 est disponible sur la page https://thinkr.fr/pdf/ggplot2-french-cheatsheet.pdf.

- Pour plus de détails sur un géome donné, utiliser l'aide (par exemple : `?geom_smooth`).

Objets géométriques

- De nombreux géomes, notamment `geom_smooth()`, utilisent un seul objet géométrique pour représenter plusieurs lignes de données.

- Avec ces géomes, on peut relier l'esthétique `group` à une variable catégorielle pour afficher plusieurs objets.

- ggplot2 affiche un objet séparé pour chaque modalité de la variable choisie.

- ggplot2 groupe aussi les données automatiquement lorsqu'on relie une esthétique à une variable discrète, comme on l'a vu pour `linetype`.

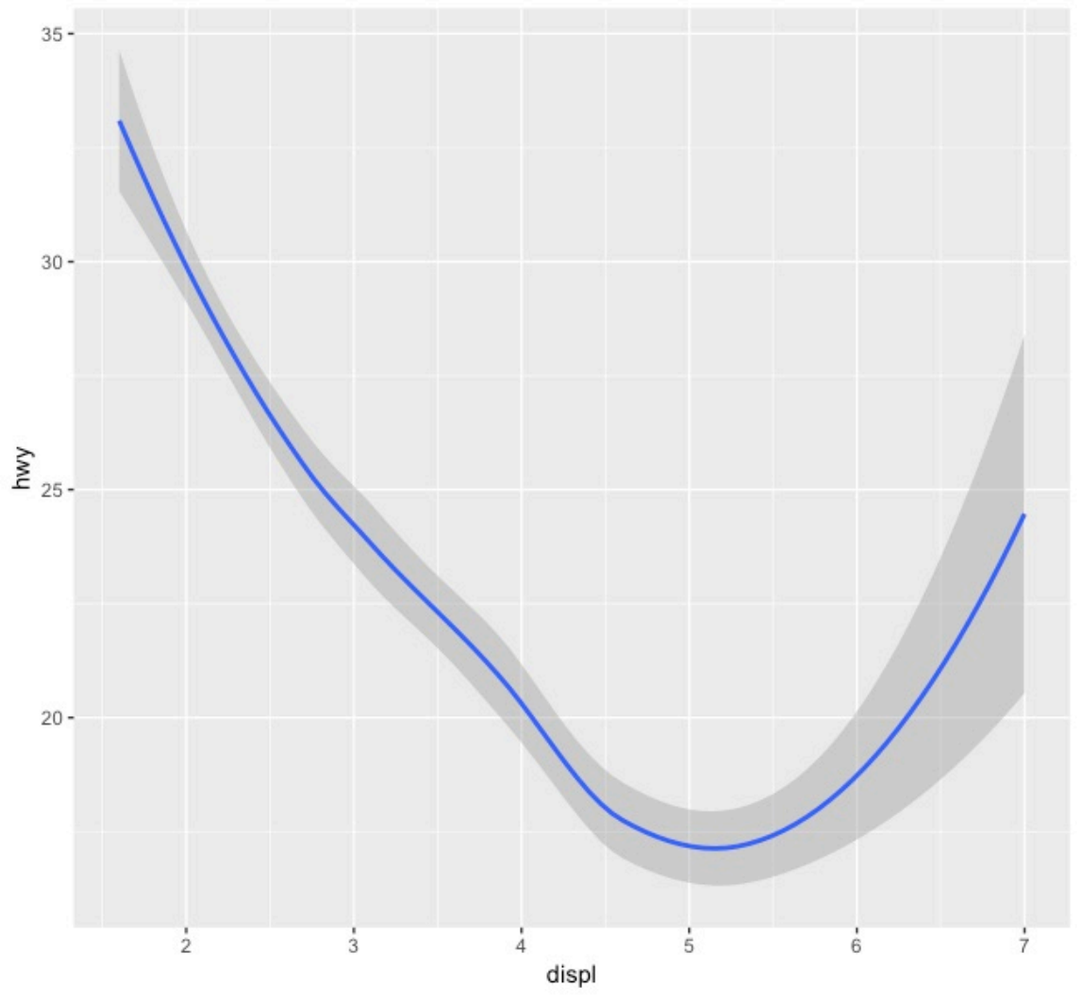
Objets géométriques

- S’appuyer sur cette dernière fonctionnalité peut s’avérer plus pratique car l’esthétique de groupe n’ajoute pas par elle-même de légende ou de caractéristiques distinctives aux géomes.

- Exemples :

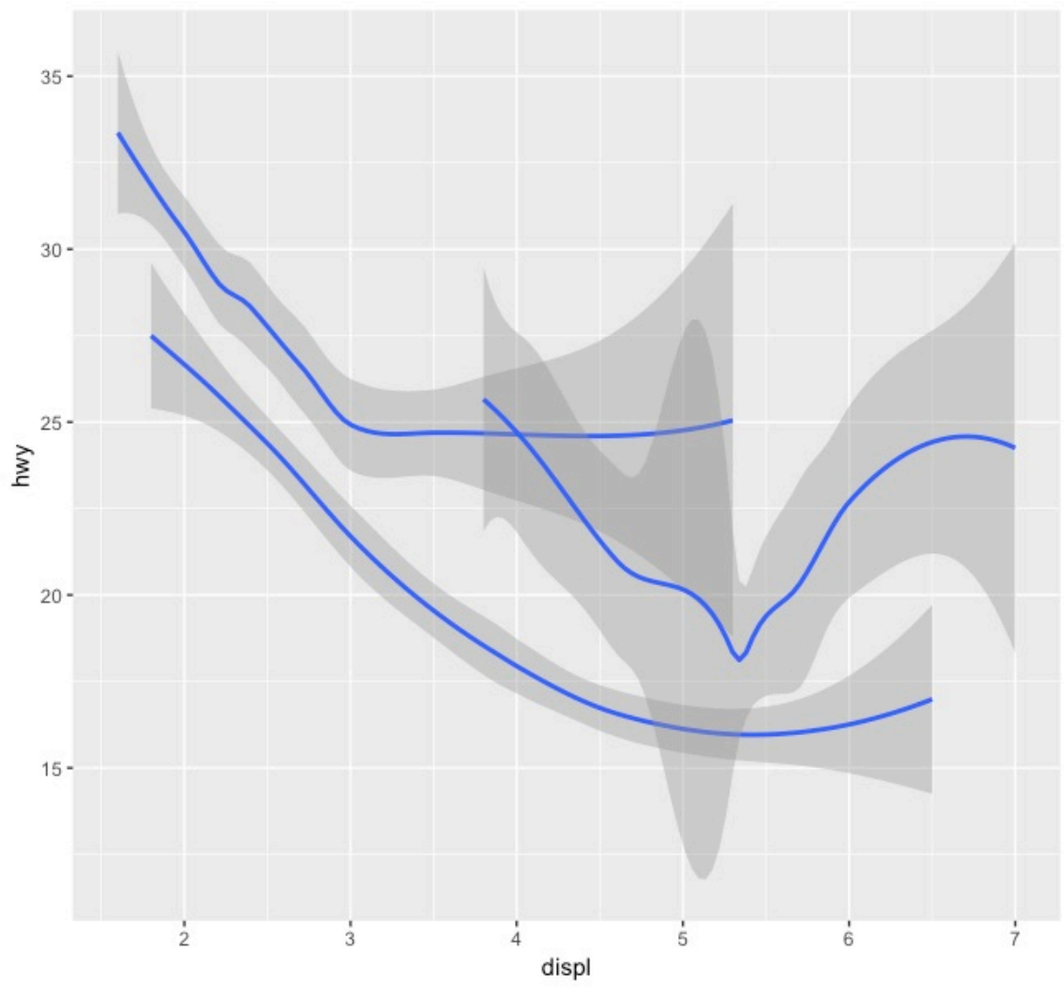
Objets géométriques

```
ggplot(data=mpg) +

  geom_smooth(mapping = aes(x=displ, y=hwy))
```



Objets géométriques

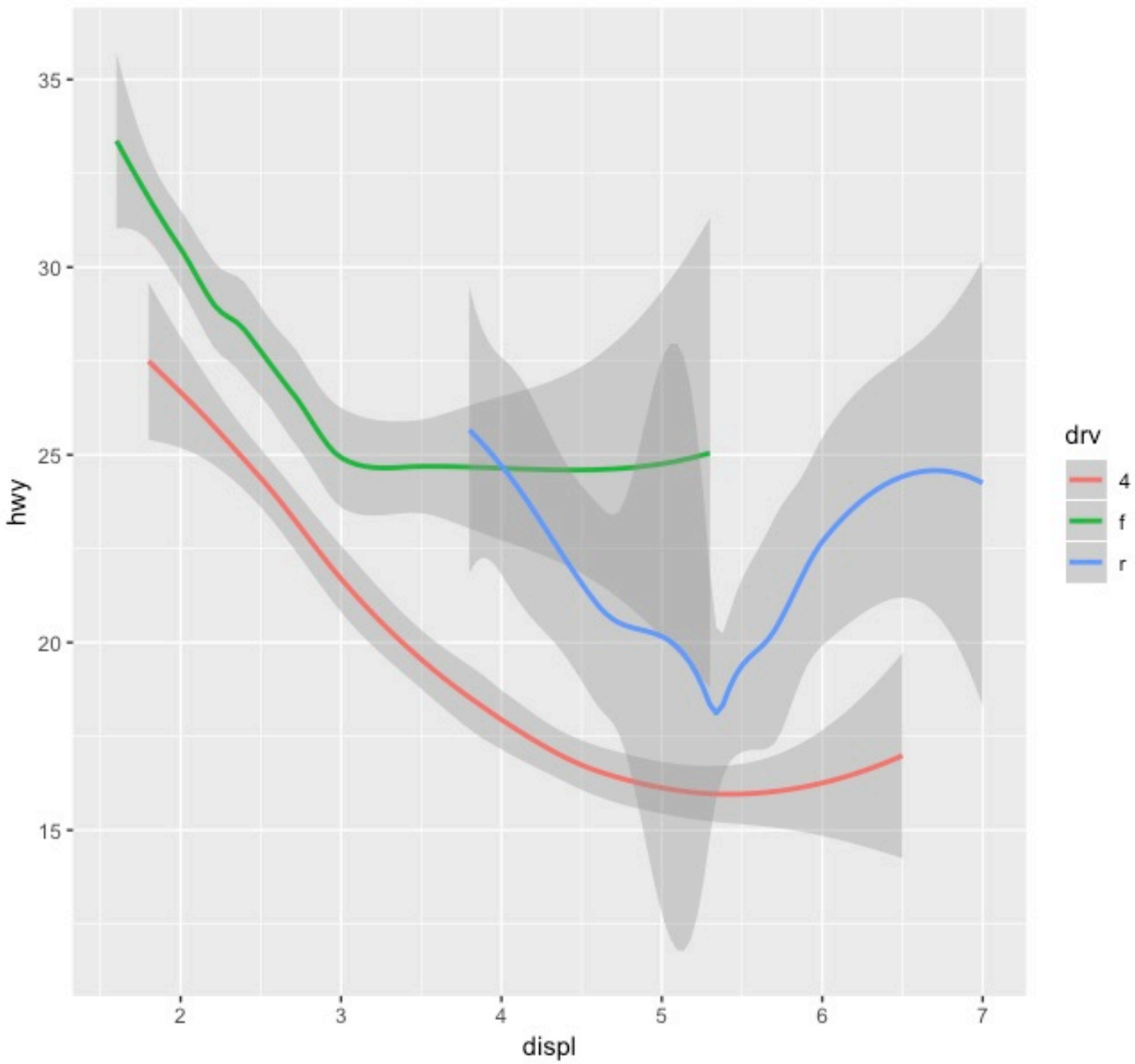
```
ggplot(data=mpg) +

  geom_smooth(mapping = aes(x=displ, y=hwy, group=drv))
```



Objets géométriques

```
ggplot(data=mpg) +

   geom_smooth(

     mapping = aes(x=displ, y=hwy, color=drv),

     show.legend = TRUE

   )
```

Objets géométriques

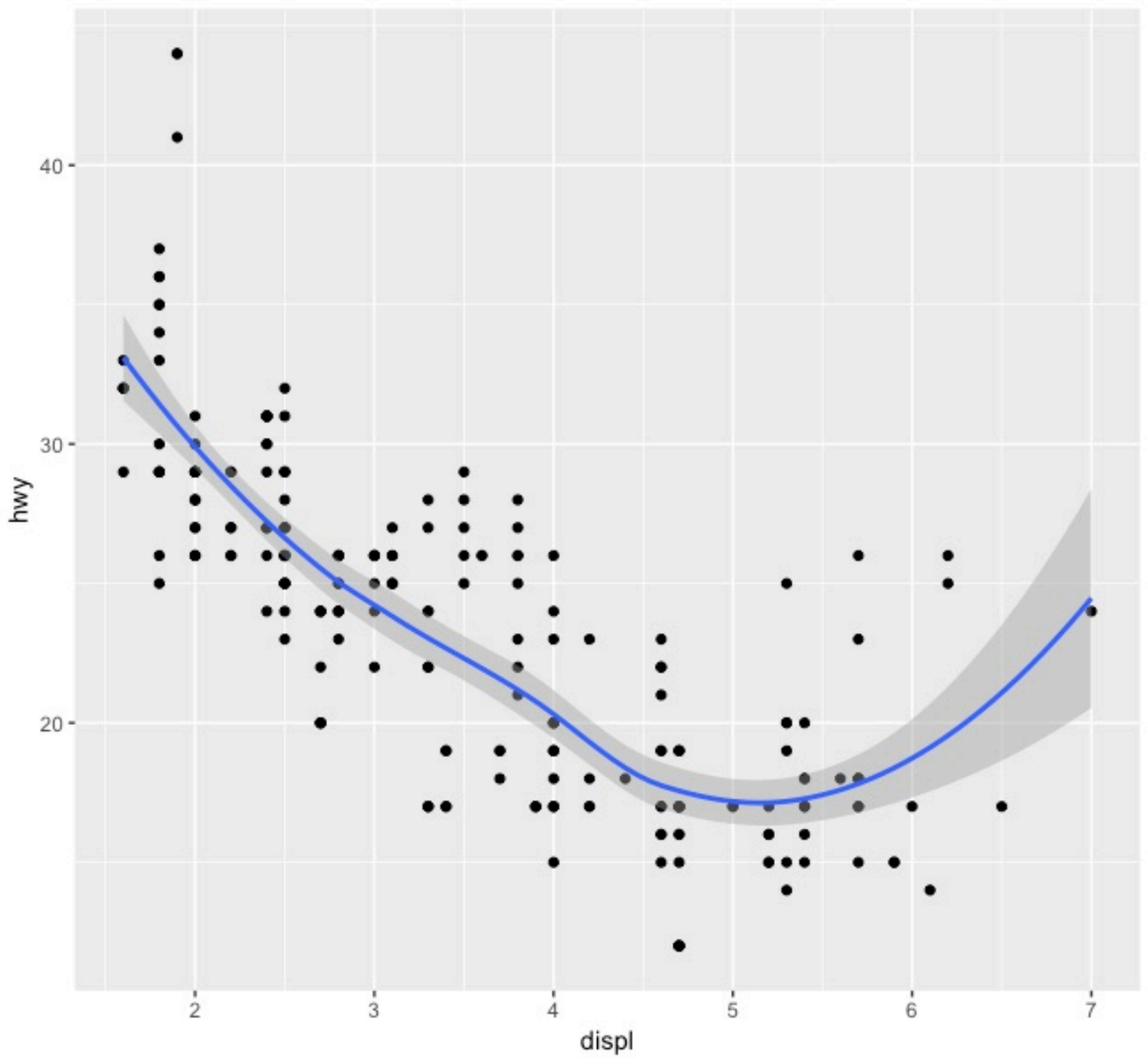


Objets géométriques

- Pour afficher plusieurs géomes sur un même graphique, il suffit d'ajouter plusieurs fonctions de géome à ggplot() :

```
ggplot(data=mpg) +

  geom_point(mapping = aes(x=displ, y=hwy)) +

  geom_smooth(mapping = aes(x=displ, y=hwy))
```

Objets géométriques



Objets géométriques

- Tel quel, cela introduit toutefois une duplication du code.

- Pour éviter ce type de répétition, on peut passer un ensemble de liaisons à `ggplot()`. Elles seront interprétées comme des liaisons globales et appliquées à chaque géome du graphique.

- Le code ci-après produit le même graphique que le précédent :

Objets géométriques

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

geom_point() +

geom_smooth()
```

Objets géométriques

- Lorsqu'on place des liaisons à l'intérieur d'une fonction de géome, ggplot2 les interprète comme des liaisons locales et les utilise pour étendre ou remplacer les liaisons globales uniquement pour cette couche.

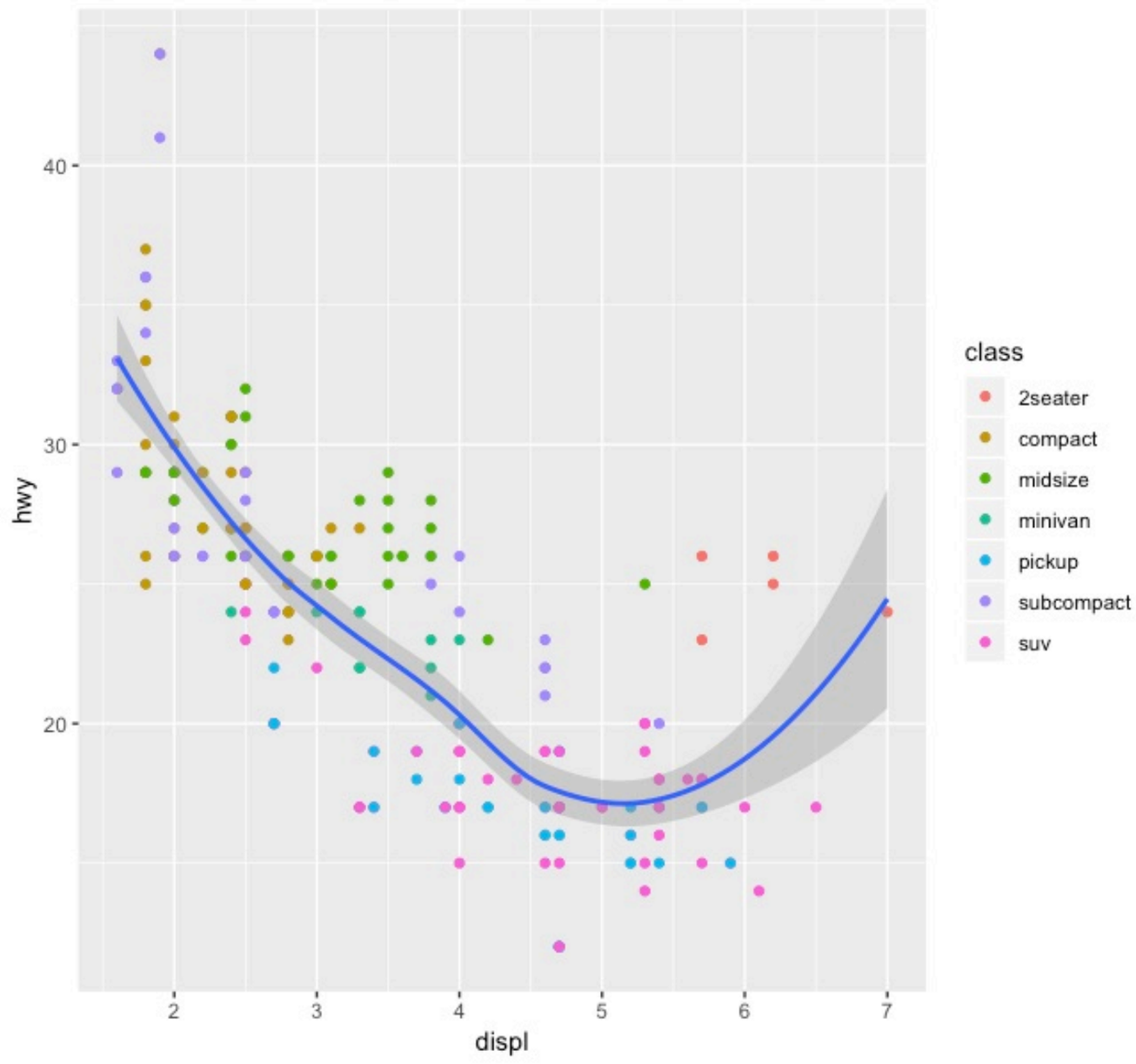
- Cela permet d'afficher différentes esthétiques sur différentes couches.

- Exemple :

Objets géométriques

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

geom_point(mapping = aes(color = class)) +

geom_smooth()
```

Objets géométriques



Objets géométriques

- Le même principe peut être utilisé pour spécifier des données différentes pour chaque couche.

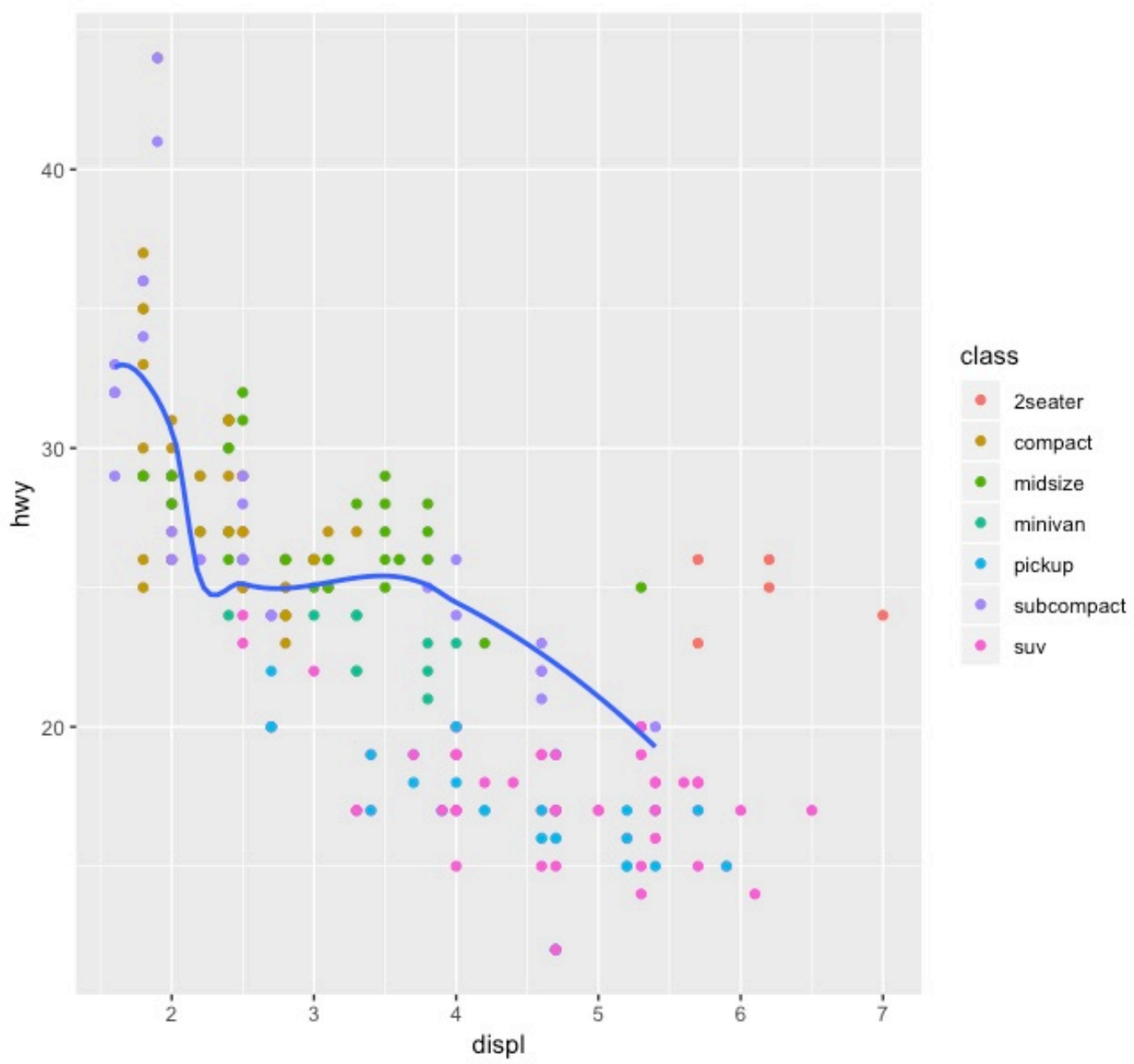
- Dans l'exemple ci-dessous, on n'affiche que le sous-ensemble du jeu de données `mpg` constitué des très petites voitures.

- L'argument local de `geom_smooth()` remplace l'argument global de `ggplot()` uniquement pour cette couche.

Objets géométriques

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

  geom_point(mapping = aes(color = class)) +

  geom_smooth(

    data = filter(mpg, class == "subcompact"),

    se = FALSE

  )
```

Objets géométriques



Questions

- Quel géome utiliser pour obtenir un diagramme en boîte ? Un histogramme ? Un diagramme en barres ? Un diagramme en aires ?

- Que fait l'argument `se` de `geom_smooth()` ?

Transformations

- On va maintenant s'intéresser aux diagrammes en barres.
- Ceux-ci semblent plutôt simples mais ils révèlent des informations intéressantes sur les graphiques.
- Considérons un diagramme basique, tel que produit par `geom_bar()`.
- Celui-ci affiche le nombre total de diamants du jeu de données `diamonds`, groupés par coupe (`cut`).

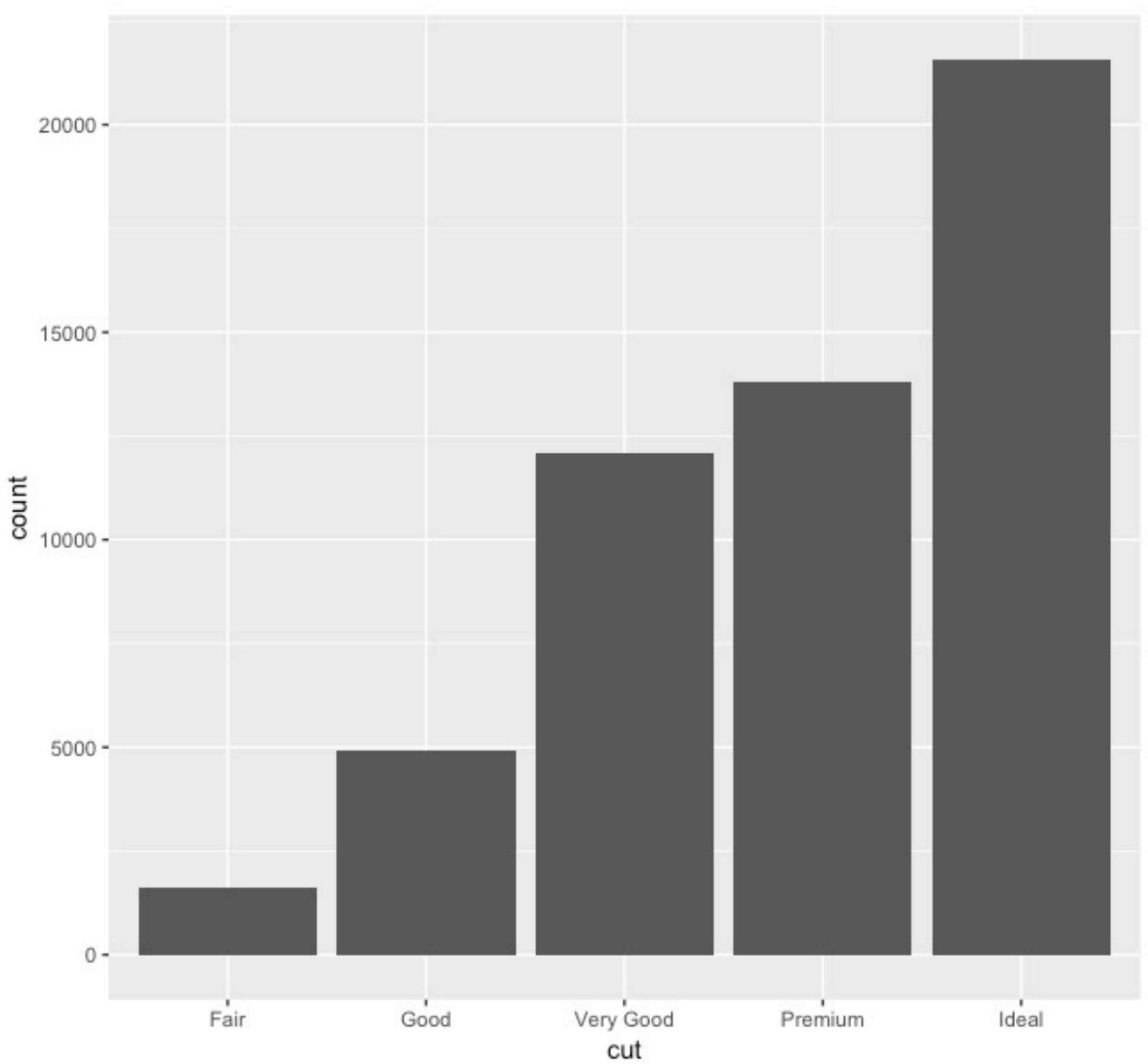
Transformations

- Ce jeu de données est inclus dans ggplot2 et contient des informations sur environ 54 000 diamants, dont : prix (`price`), carat, couleur (`color`), clarté (`clarity`) et coupe (`cut`).

- Le diagramme montre que les diamants ayant une coupe de haute qualité sont plus nombreux que ceux ayant une coupe de faible qualité.

```
ggplot(data=diamonds) +

  geom_bar(mapping =  aes(x=cut))
```

Transformations



Transformations

- L’axe des abscisses correspond à la coupe, une des variables de `diamonds`. L’axe des ordonnées affiche le compte. Or `diamonds` ne contient pas cette variable. D’où la variable `count` provient-elle donc ?

- De nombreux graphiques, par exemple les nuages de points, affichent les valeurs brutes du jeu de données, mais d’autres, comme les diagrammes en barres, calculent de nouvelles valeurs à afficher.

Transformations

- Les *diagrammes en barres*, *histogrammes* et *polygones de fréquence* répartissent les données en groupes puis affichent l'effectif total de chaque groupe.

- Les *lisseurs* ajustent un modèle aux donnée puis affichent les prédictions du modèle.

- Les *diagrammes en boîte* calculent un aperçu robuste de la distribution et affichent une boîte spécialement formatée.

Transformations

- Les algorithmes servant à calculer de nouvelles valeurs pour un graphique sont appelés *stats*, une abréviation pour « transformations statistiques ».

- On peut se renseigner sur la stat utilisée par un géome en inspectant la valeur par défaut de son argument `stat`.

- Ainsi `?geom_bar` nous apprend que la stat par défaut est « count », ce qui signifie que `geom_bar()` utilise `stat_count()`.

- `stat_count()` est documentée sur la même page que `geom_bar()`.

- En faisant défiler, on trouve une section intitulée « Variables calculées » qui nous apprend que deux nouvelles variables sont calculées : `count` et `prop`.

Transformations

- Les géomes et les stats peuvent généralement être utilisés de manière interchangeable.
- On peut par exemple recréer le graphique précédent en utilisant `stat_count` au lieu de `geom_bar` :

```
ggplot(data=diamonds) +

  stat_count(mapping =  aes(x=cut))
```

Transformations

- En effet, chaque géome a une stat par défaut et chaque stat a un géome par défaut.

- On peut donc généralement utiliser les géomes sans se soucier des transformations statistiques sous-jacentes.

- Toutefois, on peut également utiliser une stat explicitement, notamment pour trois raisons :

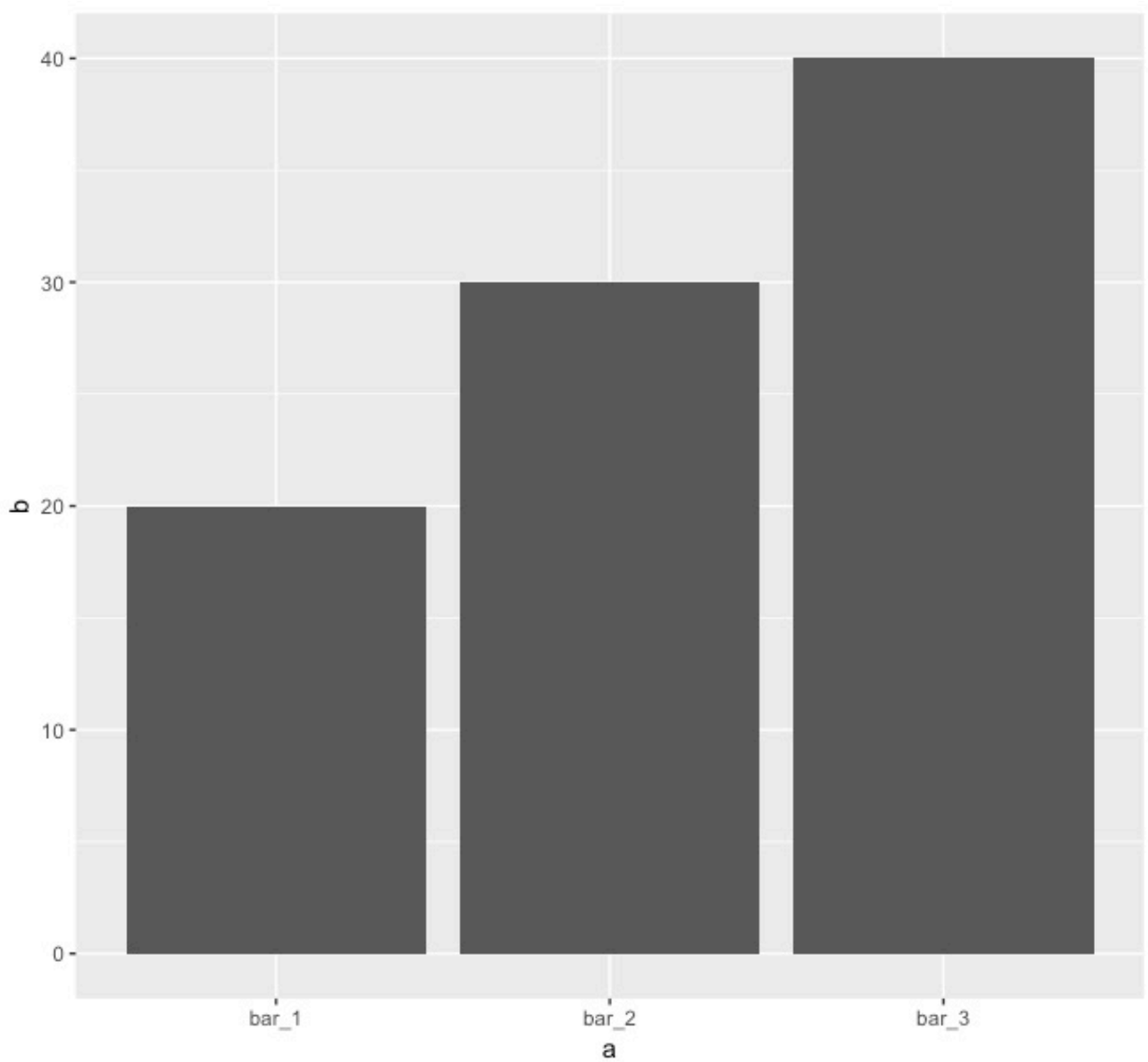
Transformations

1. *On peut souhaiter remplacer la stat par défaut*. Dans le code suivant, la stat de `geom_bar()`, normalement `count`, est changée en `identity` afin d'attribuer comme hauteur aux barres les valeurs brutes de la variable en y.

Transformations

```
demo <- tribble(
  ~a, ~b,
  "bar_1", 20,
  "bar_2", 30,
  "bar_3", 40
)


ggplot(data=demo) +
  geom_bar(
    mapping = aes(x=a,y=b),
    stat = "identity"
  )
```

Transformations



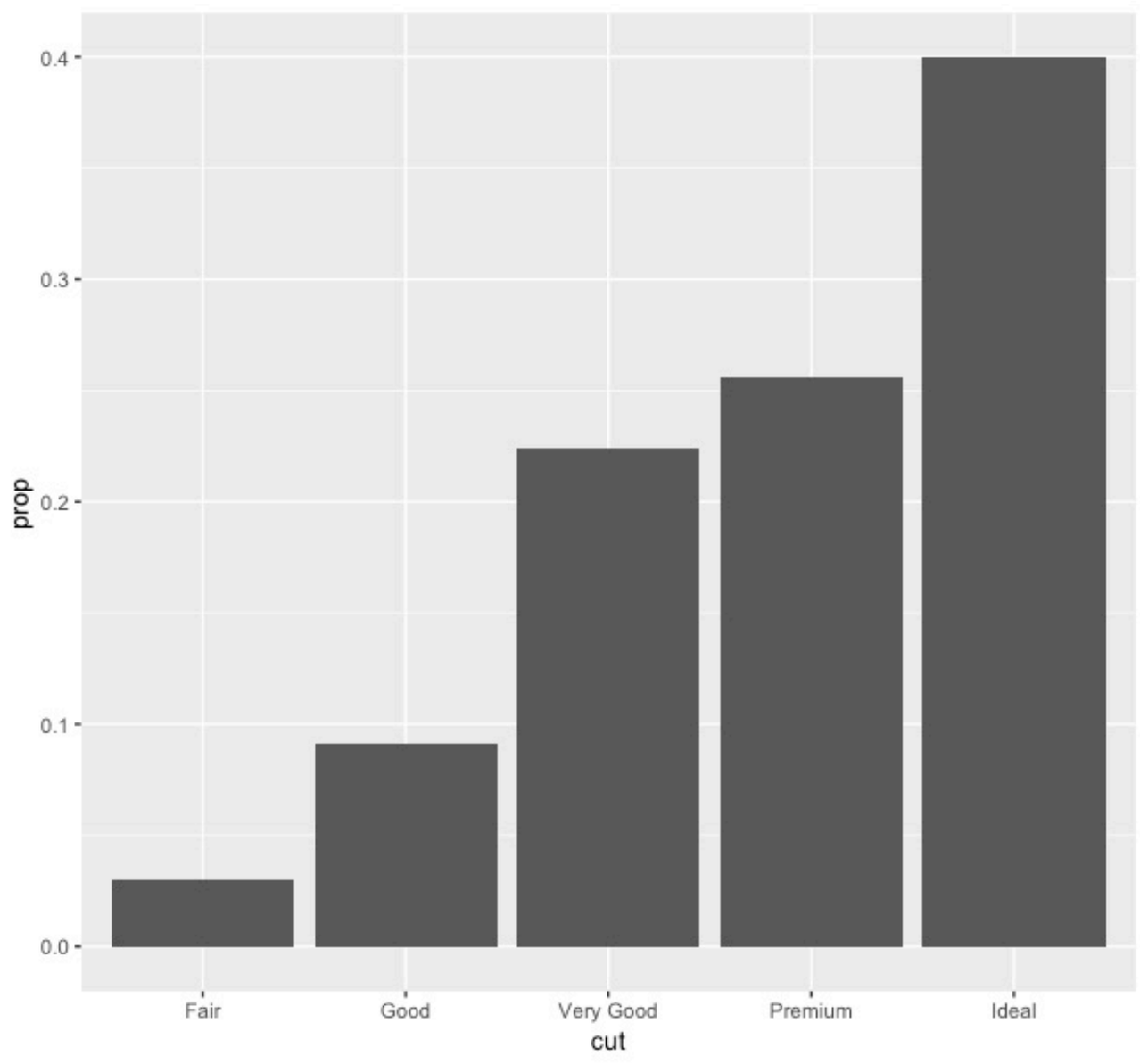
Transformations

2. *On peut vouloir remplacer la correspondance par défaut entre les variables transformées et les esthétiques*.

Par exemple, pour afficher un diagramme en barres représentant la proportion au lieu du compte :

```
ggplot(data=diamonds) +
  geom_bar(
    mapping = aes(x = cut, y =..prop.., group = 1)
  )
```

Transformations



Transformations

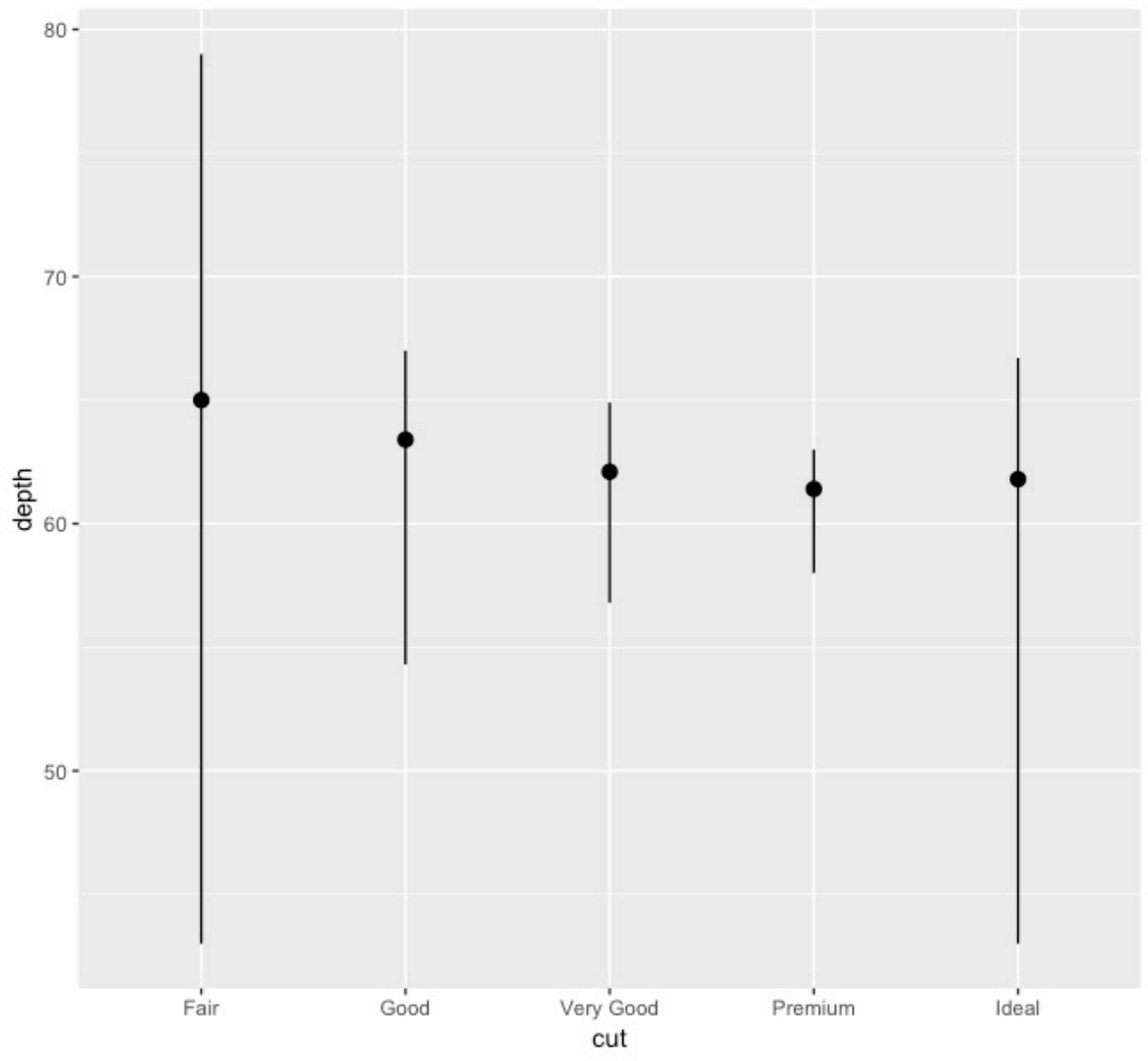
- Pour connaître les variables calculées par la stat, se référer à la section d’aide intitulée « Variables calculées ».

Transformations

3. *On veut mettre l'accent sur la transformation statistique*. On peut alors utiliser par exemple `stat_summary()` qui récapitule les valeurs y pour chaque valeur x distincte :

```
ggplot(data=diamonds) +
  stat_summary(
    mapping = aes(x=cut, y=depth),
    fun.ymin = base::min,
    fun.ymax = base::max,
    fun.y = median
  )
```

Transformations



Transformations

- ggplot2 fournit plus de 20 stats à utiliser. Chacune est une fonction sur laquelle on peut obtenir de l'aide de la façon usuelle.

- La fiche récapitulative de ggplot2 contient une liste complète des stats.

Questions

- Quel est le géome associé par défaut à `stat_summary()` ? Comment réécrire le code précédent en utilisant cette fonction de géome au lieu de la fonction de stat ?

- Que fait `geom_col()` ? En quoi diffère-t-il de `geom_bar()` ?

- Quelles sont les variable calculées par `stat_smooth()` ? Quels sont les paramètres qui contrôlent son comportement ?

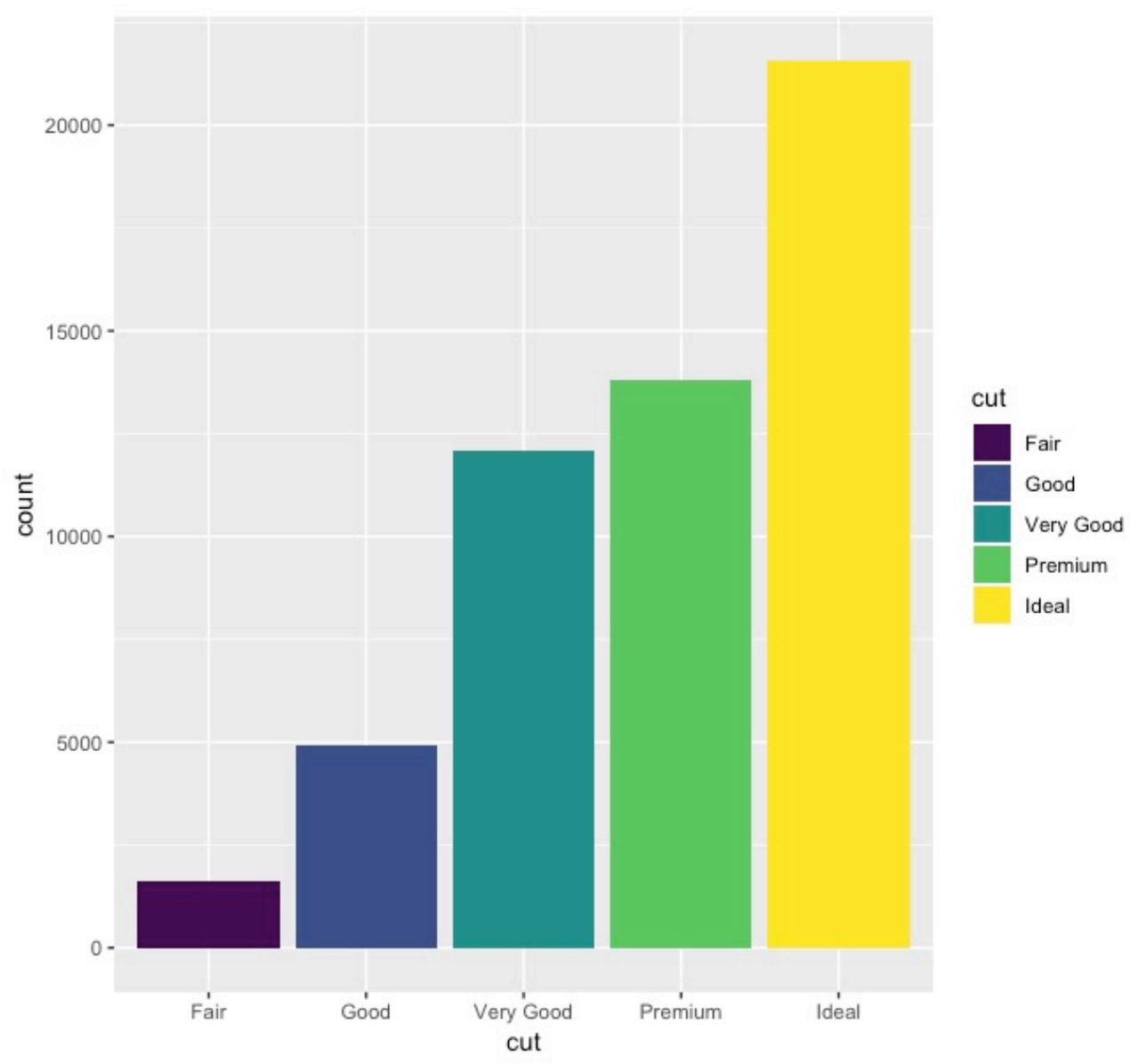
- Dans le diagramme de proportion précédent, on a du spécifier `group = 1`. Pourquoi ?

Positionnement

- Les diagrammes en barres peuvent être coloriés à l'aide des esthétiques de couleur et de remplissage. Le remplissage est généralement plus utile :

```
ggplot(data=diamonds) +

  geom_bar(mapping = aes(x=cut, fill=cut))
```

Positionnement

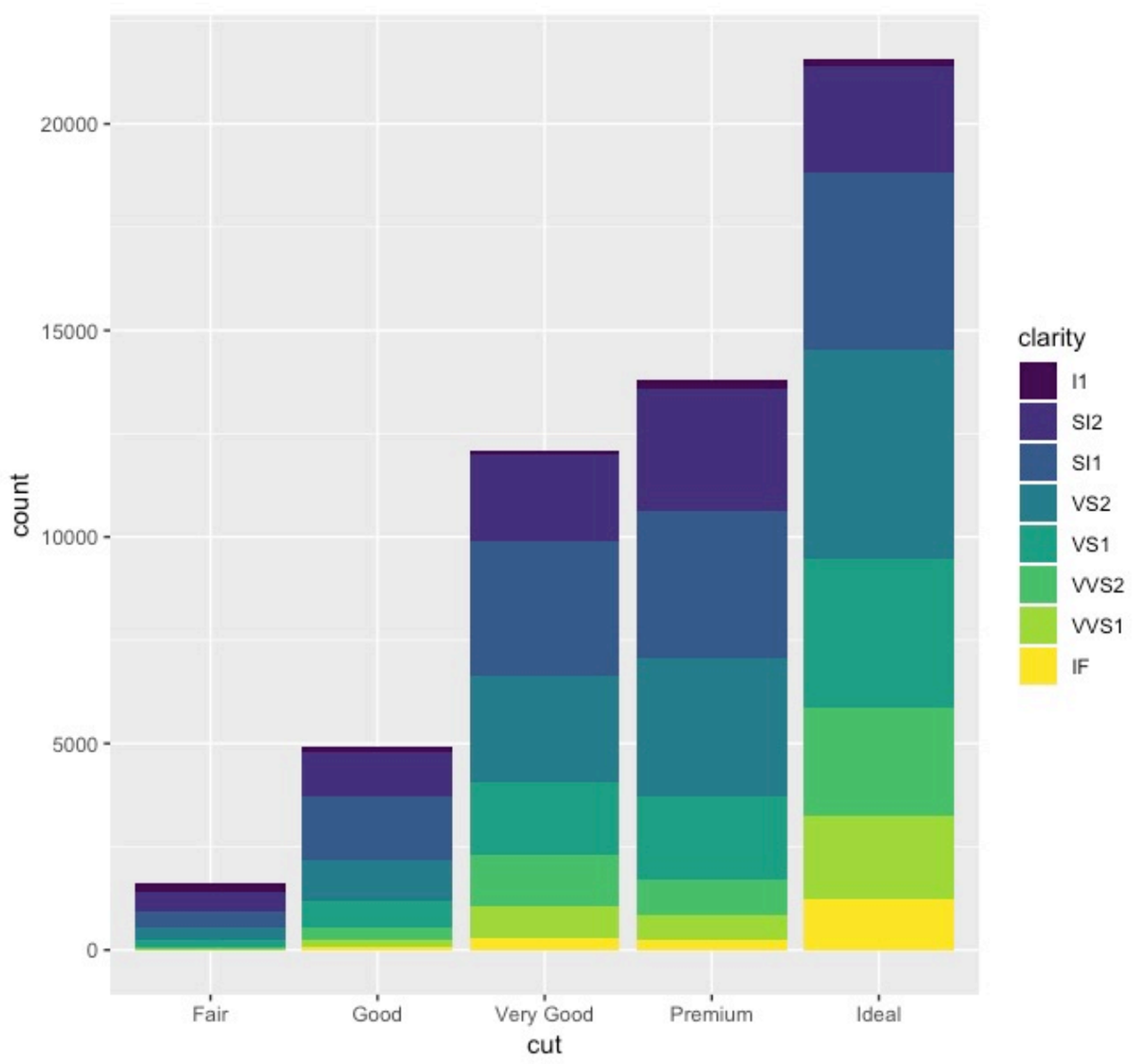


Positionnement

- Si on relie l'esthétique de remplissage à une variable, par exemple la clarté, les barres sont automatiquement empilées. Chaque rectangle de couleur représente une combinaison des deux variables :

```
ggplot(data=diamonds) +

  geom_bar(mapping = aes(x=cut, fill=clarity))
```

Positionnement



Positionnement

- Cet empilement est effectué automatiquement et dépend du *positionnement*, spécifié par l'argument `position`.

- Au lieu de l'empilement (`stack`), on peut utiliser une des trois autres options : `identity`, `dodge` ou `fill`.

Positionnement

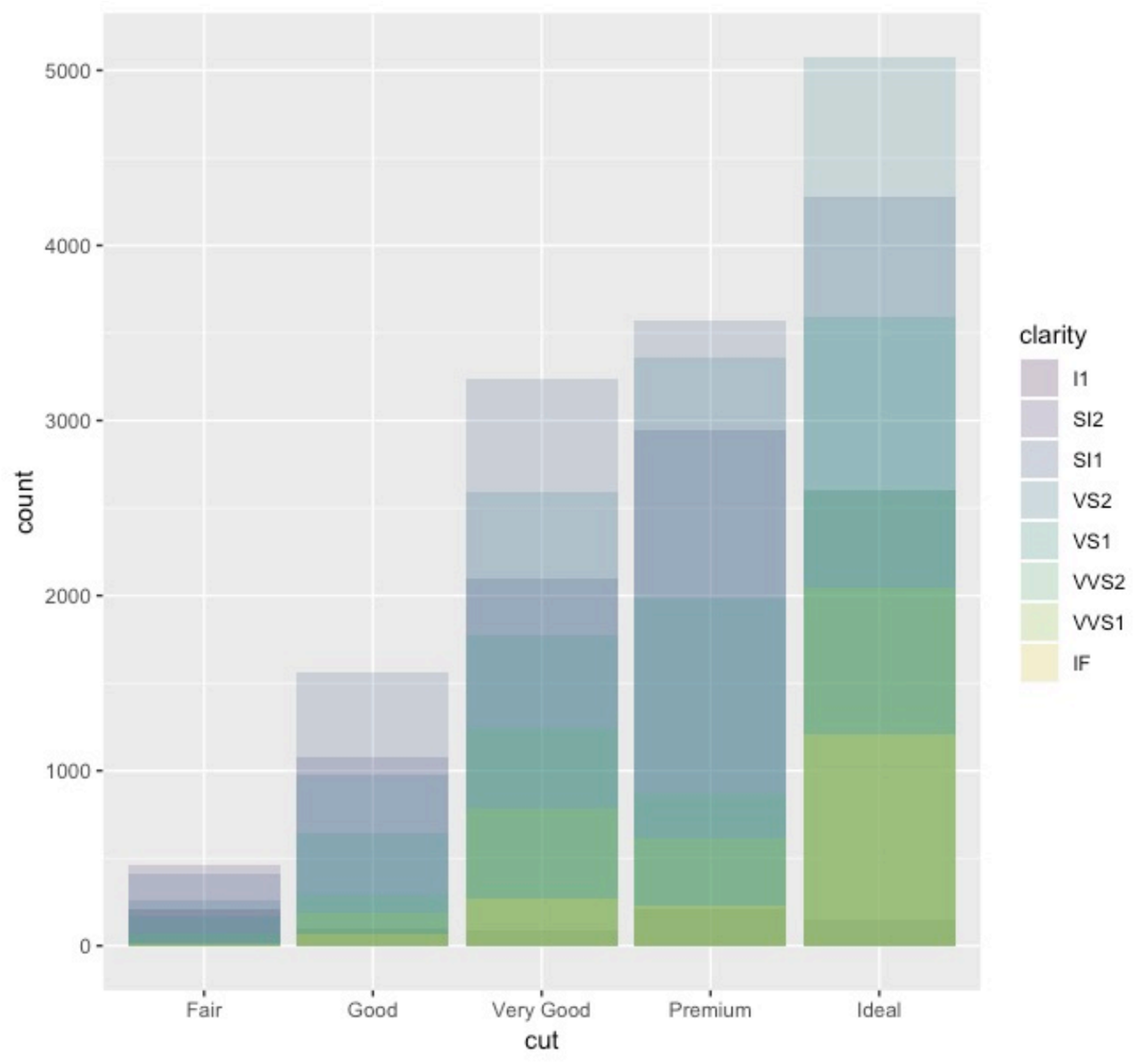
- `position = « identity »` place chaque objet à sa position exacte dans le contexte du graphique. Cela n'est pas très efficace pour les barres car elles se chevauchent.

- Pour pouvoir toutes les visualiser, il faut les rendre soit partiellement transparentes en attribuant une valeur faible à l'esthétique `alpha`, soit entièrement transparentes en configurant `fill = NA` :

Positionnement

```
ggplot(

   data = diamonds,

mapping = aes(x=cut, fill=clarity)

) +

geom_bar(alpha=1/5, position="identity")
```

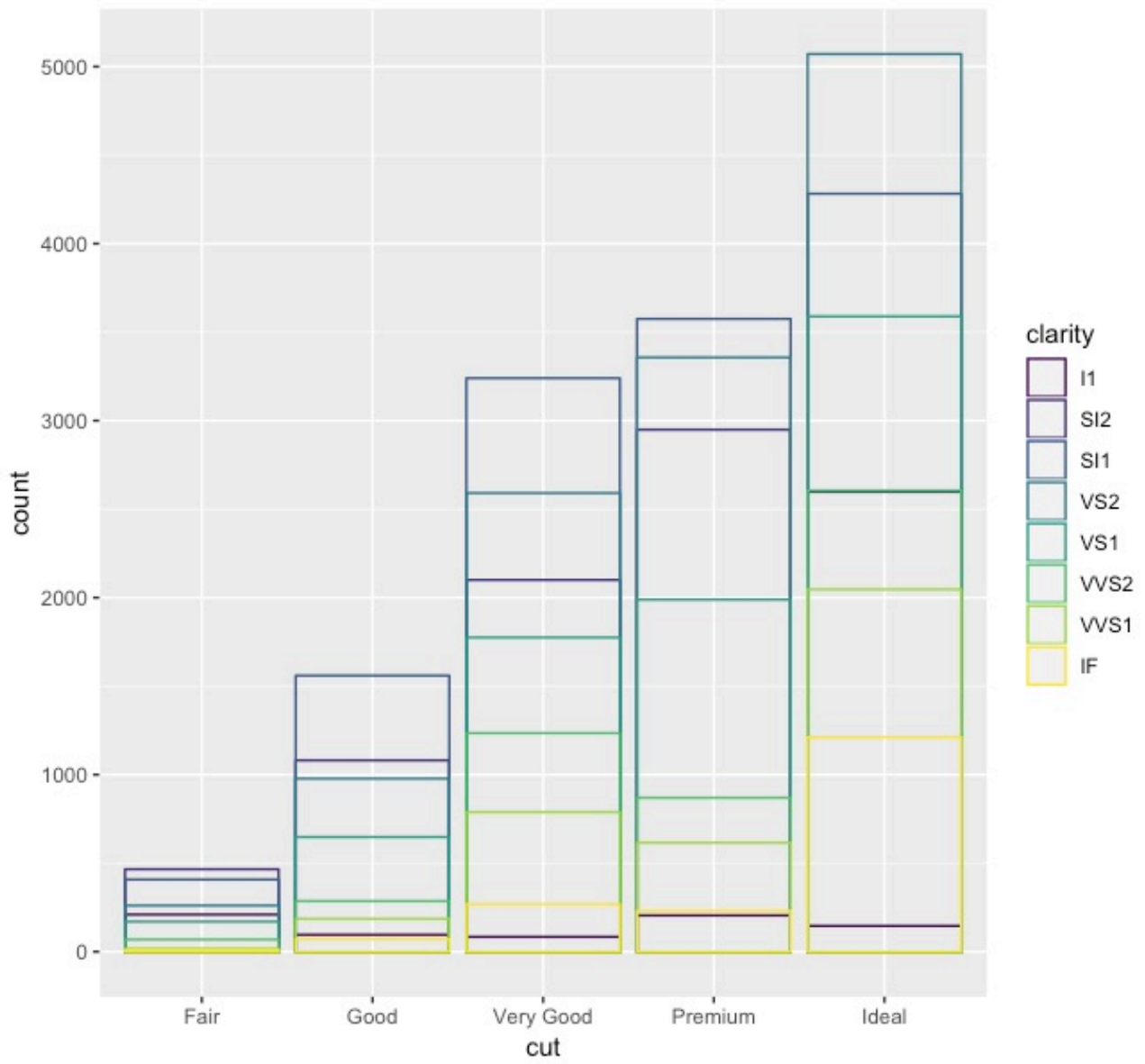
Positionnement



Positionnement

```
ggplot(

   data = diamonds,
   mapping = aes(x=cut, color=clarity)

) +

   geom_bar(fill=NA, position="identity")
```

Positionnement



Positionnement

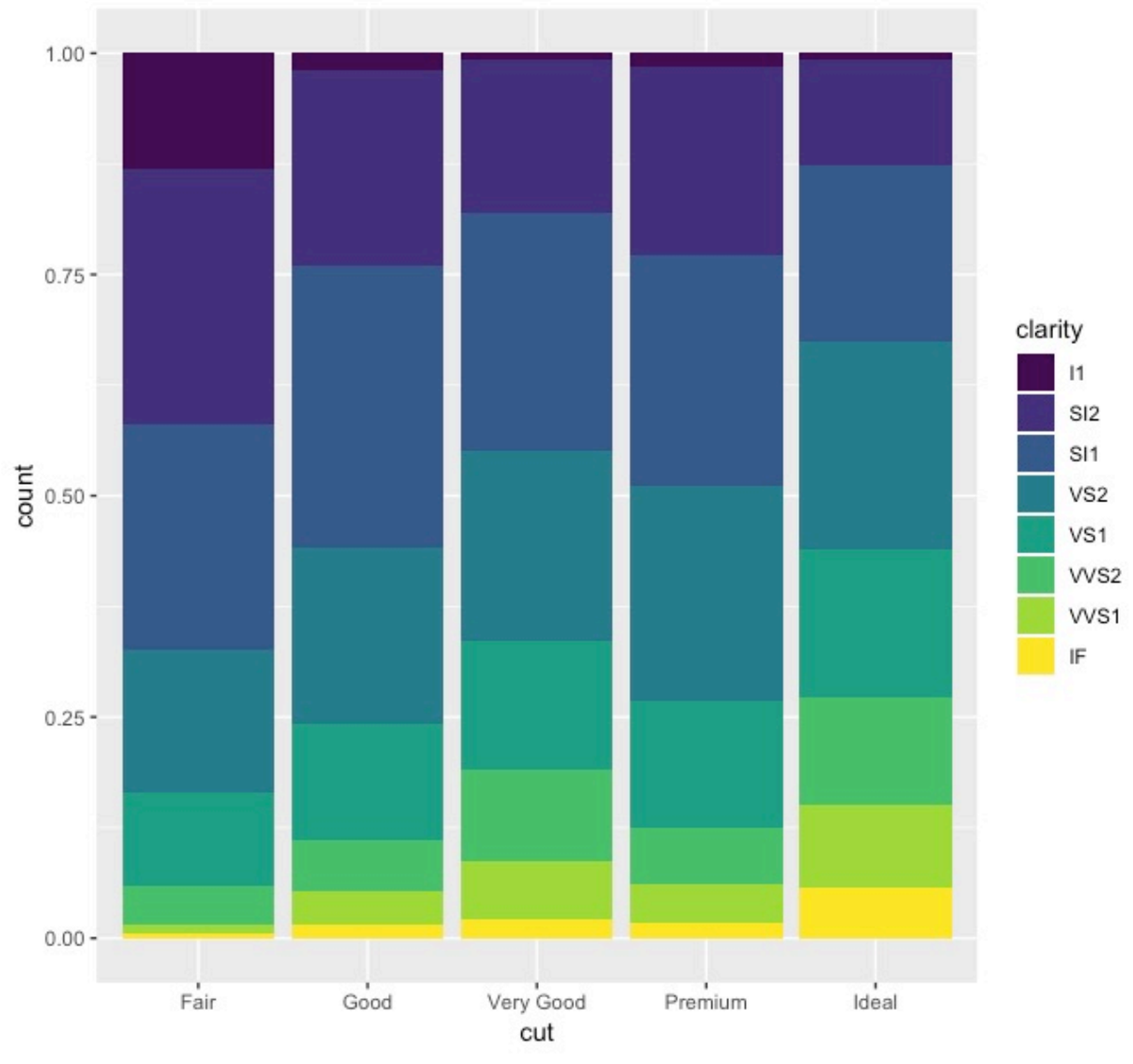
- Le positionnement identité est plus utile pour les géomes en 2D, comme les points, pour lesquels il est défini par défaut.

- `position = « fill »` fonctionne comme l'empilement, mais assigne la *même hauteur* à chaque ensemble de barres empilées, ce qui facilite la *comparaison de proportions* entre groupes :

Positionnement

```
ggplot(data=diamonds) +

  geom_bar(
    mapping = aes(x=cut, fill=clarity),
    position = "fill"
  )
```

Positionnement

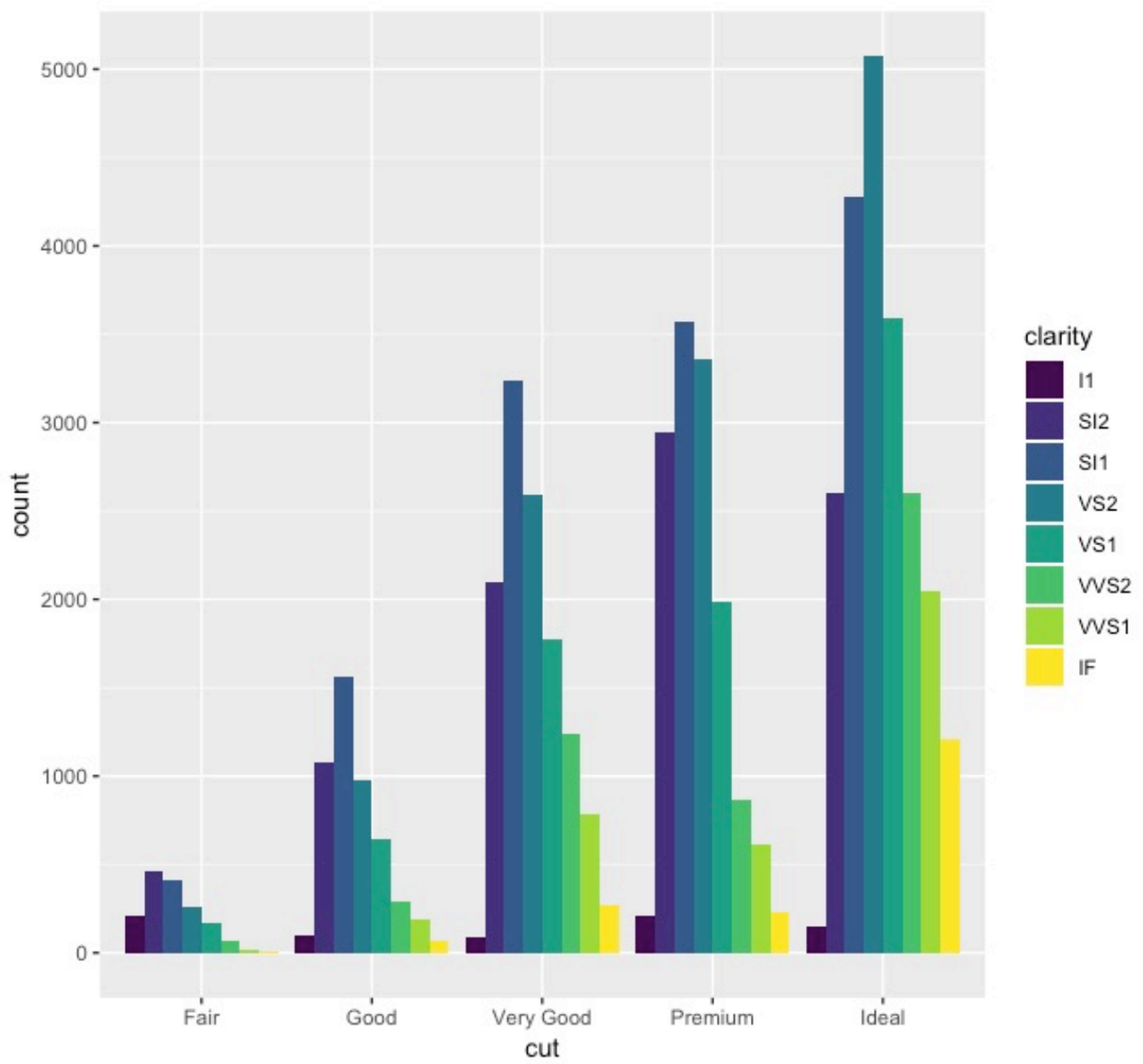


Positionnement

- `position = « dodge »` place les objets *les uns à côté des autres*, ce qui facilite la *comparaison des valeurs individuelles* :

```
ggplot(data=diamonds) +
  geom_bar(
    mapping = aes(x=cut, fill=clarity),
    position = "dodge"
  )
```

Positionnement

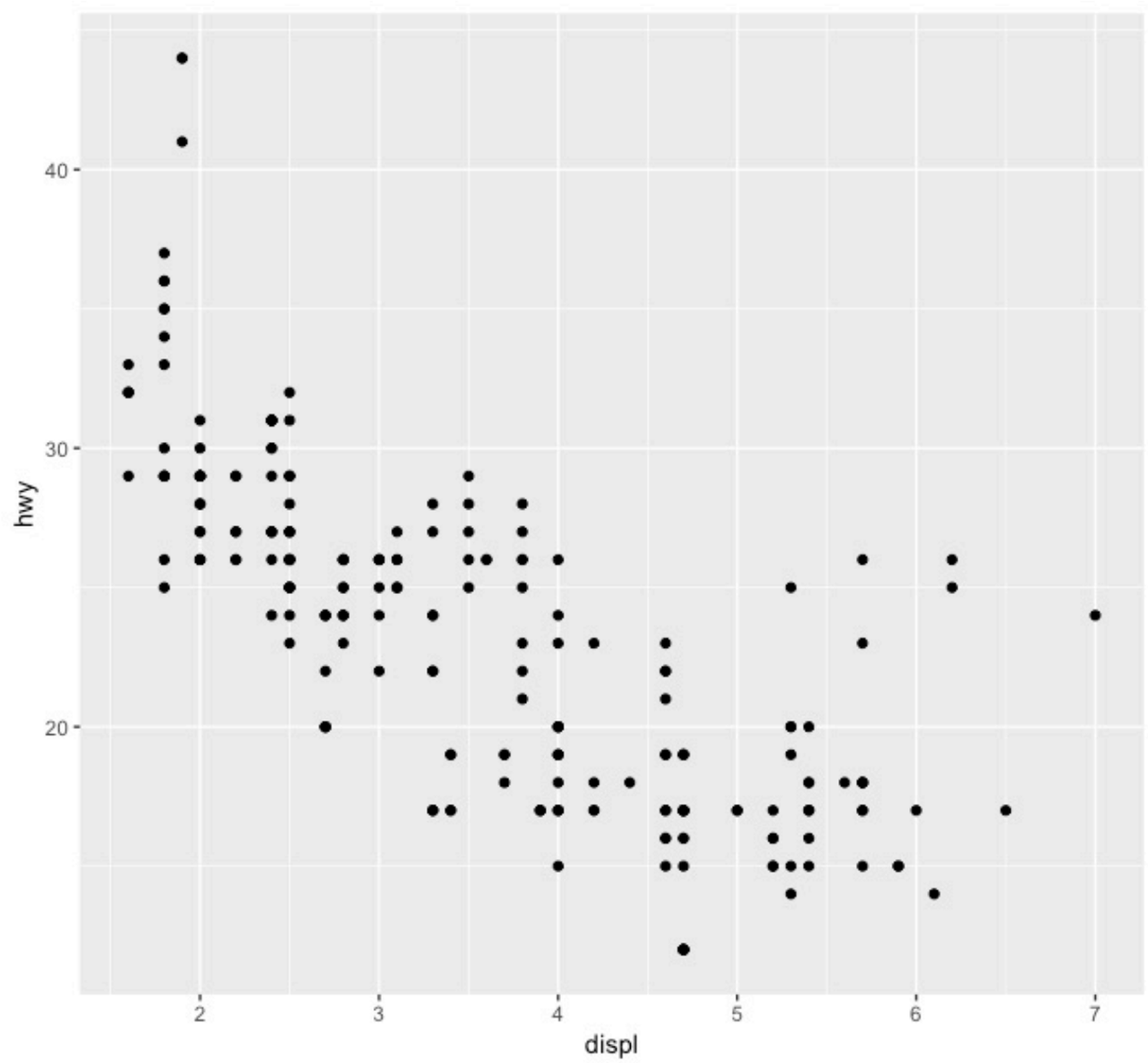


Positionnement

- Il existe un autre type de positionnement, qui ne sert pas pour les barres, mais qui est particulièrement utile pour les nuages de points.

- Reprenons notre premier nuage de points :

Positionnement



Positionnement

- Le graphique n’affiche que 126 points alors que le jeu de données contient 234 observations.

- C’est du au fait que les valeurs de `hwy` et `displ` sont arrondies. Par conséquent, les points sont placés sur une grille et de nombreux points sont au même endroit.

- Ce problème, appelé *suraffichage* (ang. *overplotting*), empêche de percevoir correctement l’ensemble des données.

Positionnement

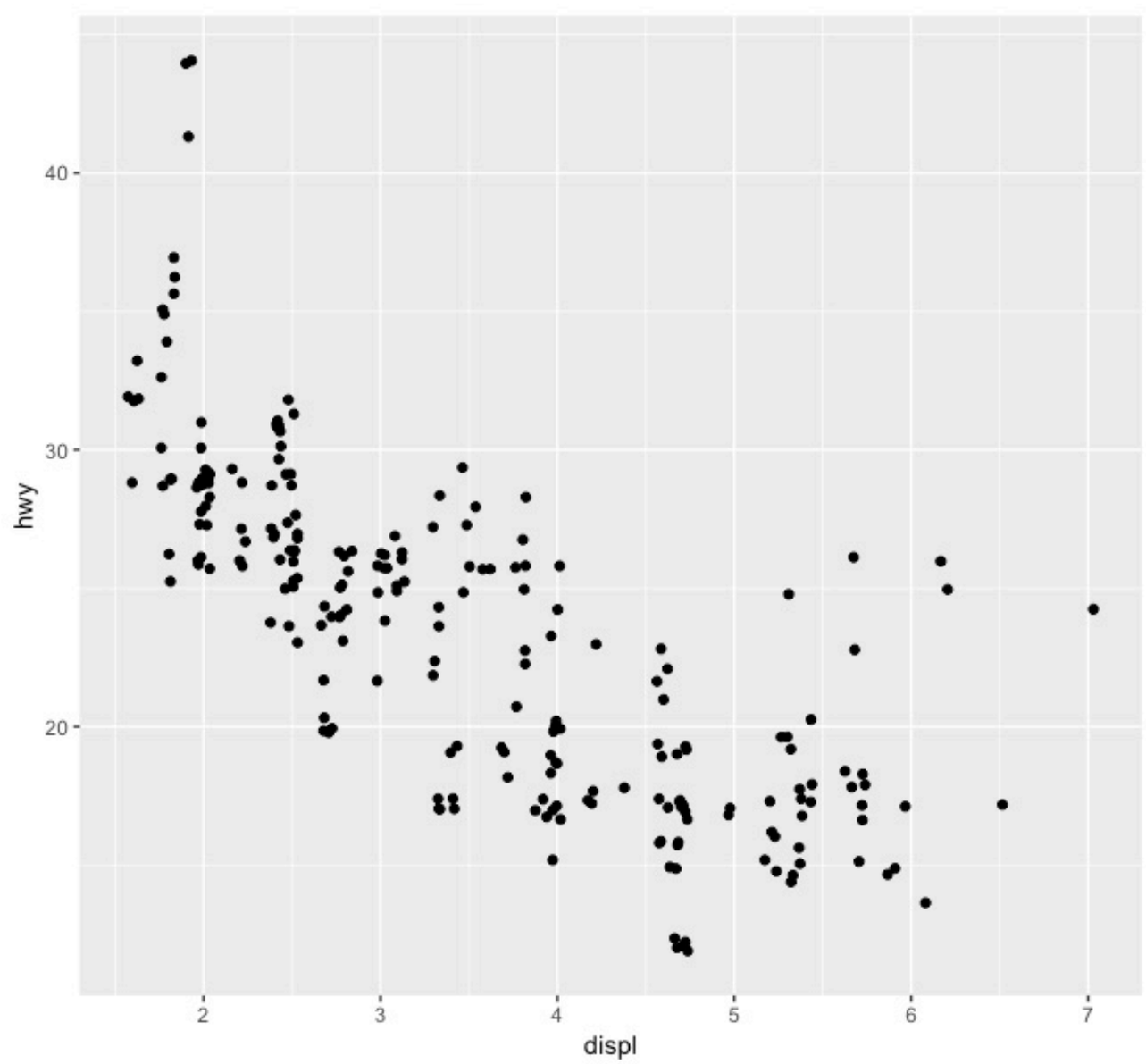
- Pour éviter ce problème, on peut configurer le positionnement avec `position = « jitter »`, qui ajoute à chaque point un léger déplacement aléatoire.

- La probabilité que deux points se voient affectés la même quantité étant très faible, les points sont séparés les uns des autres.

Positionnement

```
ggplot(data=mpg) +

  geom_point(
    mapping = aes(x=displ, y=hwy),

    position = "jitter"

  )
```

Positionnement



Positionnement

- Ajouter du bruit à un graphique peut sembler aberrant, et cela peut effectivement le rendre moins précis si l'échelle est petite, mais, à grande échelle, cela le rend au contraire plus efficace.

- Cette opération est tellement utile que ggplot2 fournit une commande raccourcie pour `geom_point(position = « jitter »)`. Il s'agit de `geom_jitter()`.

- Pour en savoir plus sur les positionnement, consulter la page d'aide associée à chacun d'eux ; par exemple :

`?position_dodge`, `?position_stack`, `?position_jitter`.

Questions

- Quels sont les paramètres de `geom_jitter()` qui contrôlent la quantité de bruit ?

- Comparer `geom_jitter()` et `geom_count()` et exposer les différences trouvées.

- Quel est le positionnement par défaut pour `geom_boxplot()` ? Créer une visualisation qui le montre.

Systèmes de coordonnées

- Les systèmes de coordonnées sont la partie la plus complexe de ggplot2.

- Le système par défaut est le système cartésien, pour lequel les positions en abscisses et en ordonnées déterminent la position de chaque point.

- D'autres systèmes de coordonnées peuvent être utiles dans certains cas.

Systèmes de coordonnées

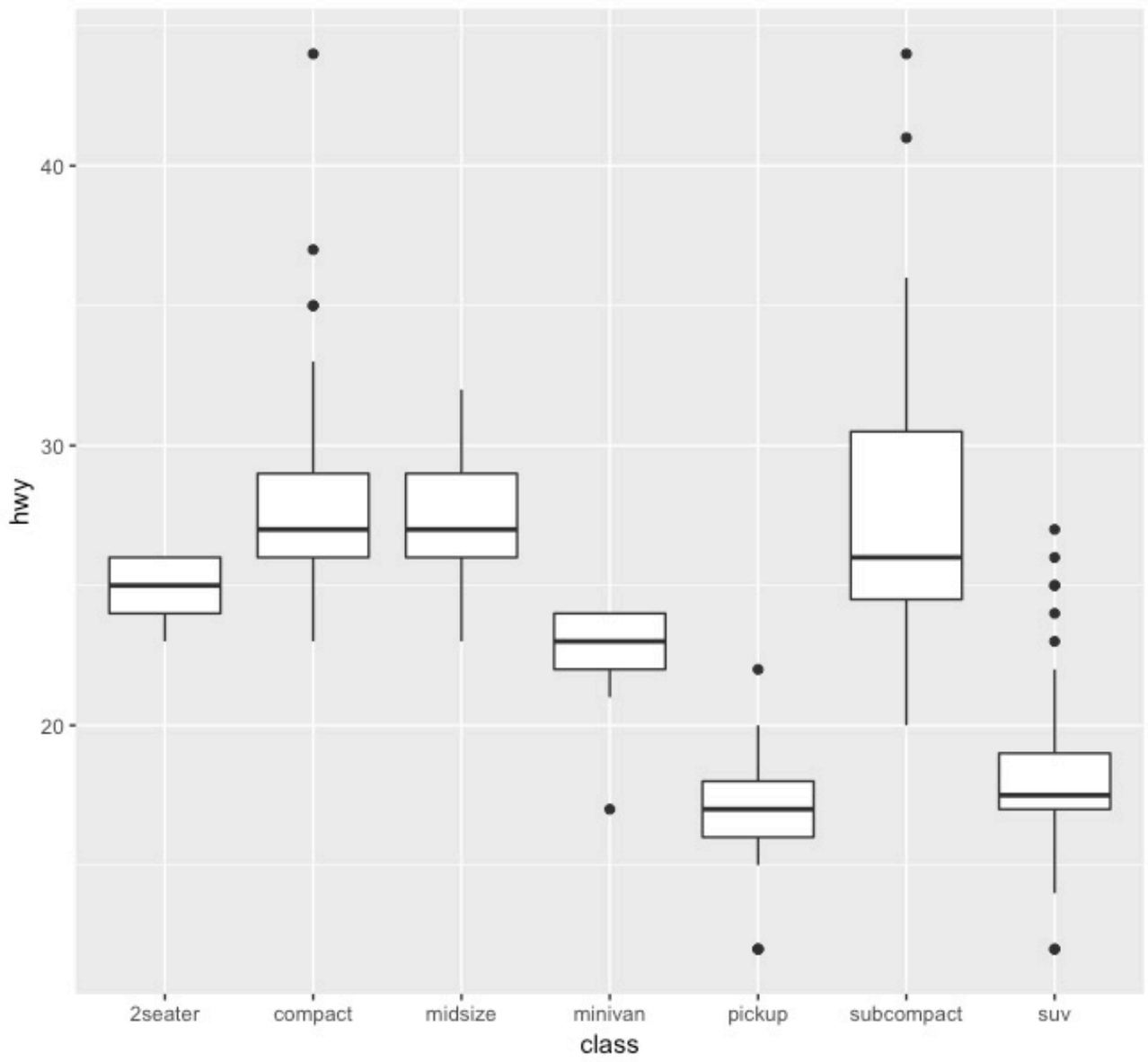
- `coord_flip()` échange les axes `x` et `y`. Cela peut servir pour créer des diagramme en boîte horizontaux, par exemple.

- C’est aussi une option intéressante lorsque les libellés sont trop longs pour être placés sur l’axe des abscisses sans chevauchement :

Systèmes de coordonnées

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

  geom_boxplot()
```

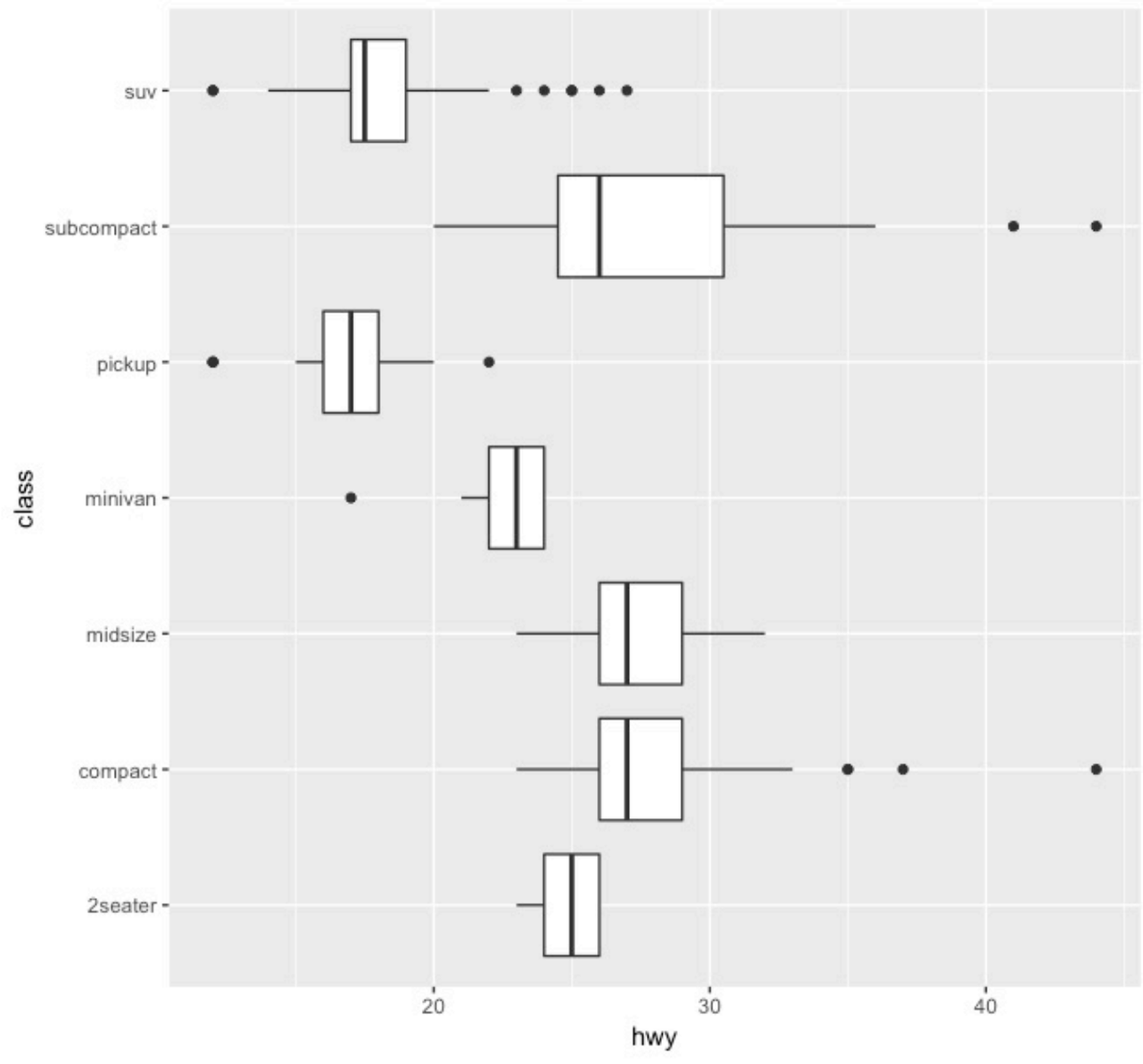
Systèmes de coordonnées



Systèmes de coordonnées

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

geom_boxplot() +

coord_flip()
```

Systèmes de coordonnées



Systèmes de coordonnées

- `coord_quickmap()` définit le rapport hauteur/largeur adapté aux cartes, ce qui est très important lorsqu'on utilise ggplot2 pour afficher des données spatiales (ce que nous ne couvrirons pas dans ce cours) :

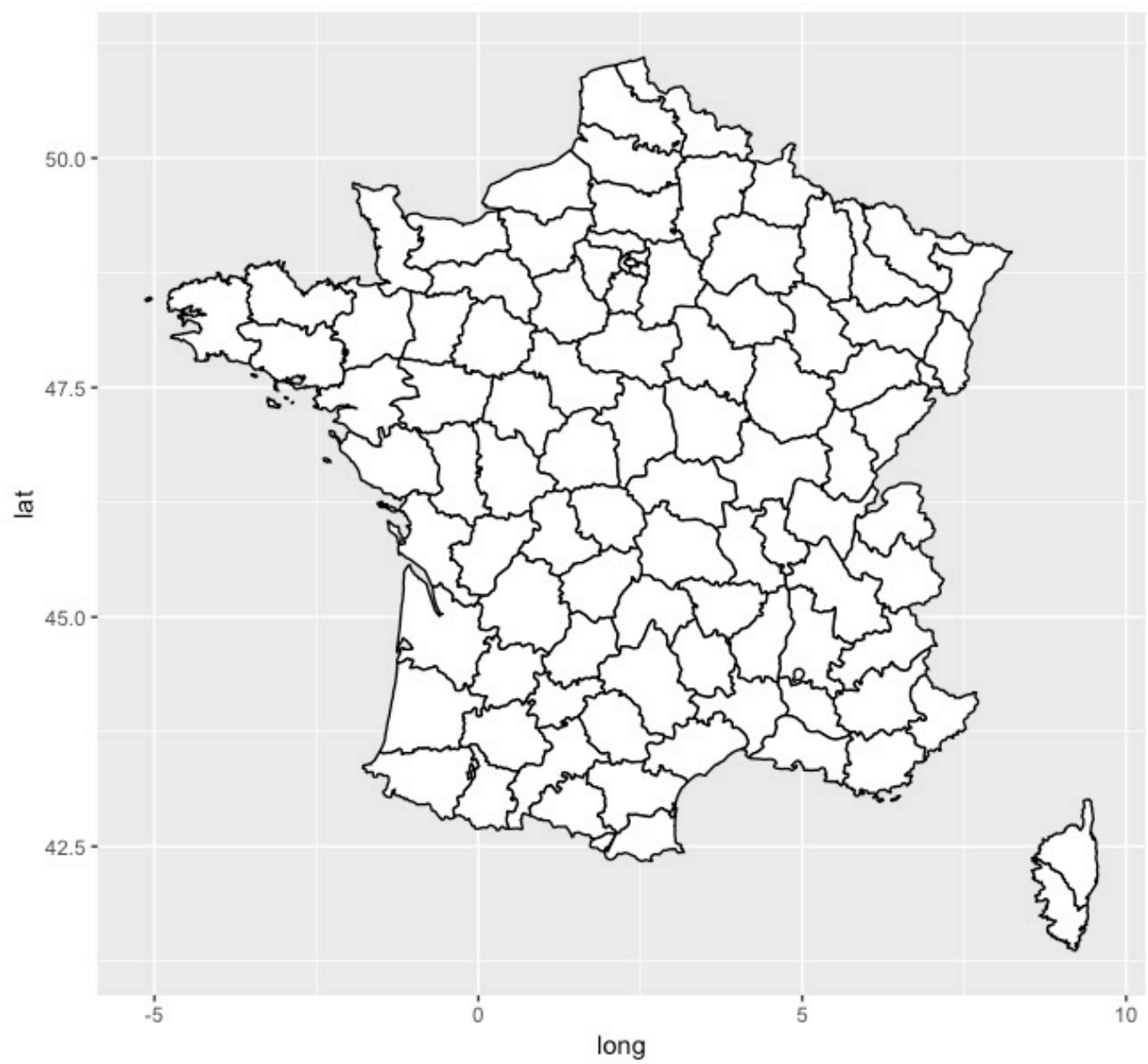
Systèmes de coordonnées

```
install.packages("maps")

library(maps)
```

```
fr <- map_data("france")
```

```
ggplot(fr, aes(long,lat, group=group)) +
  geom_polygon(fill = "white", color="black")
```

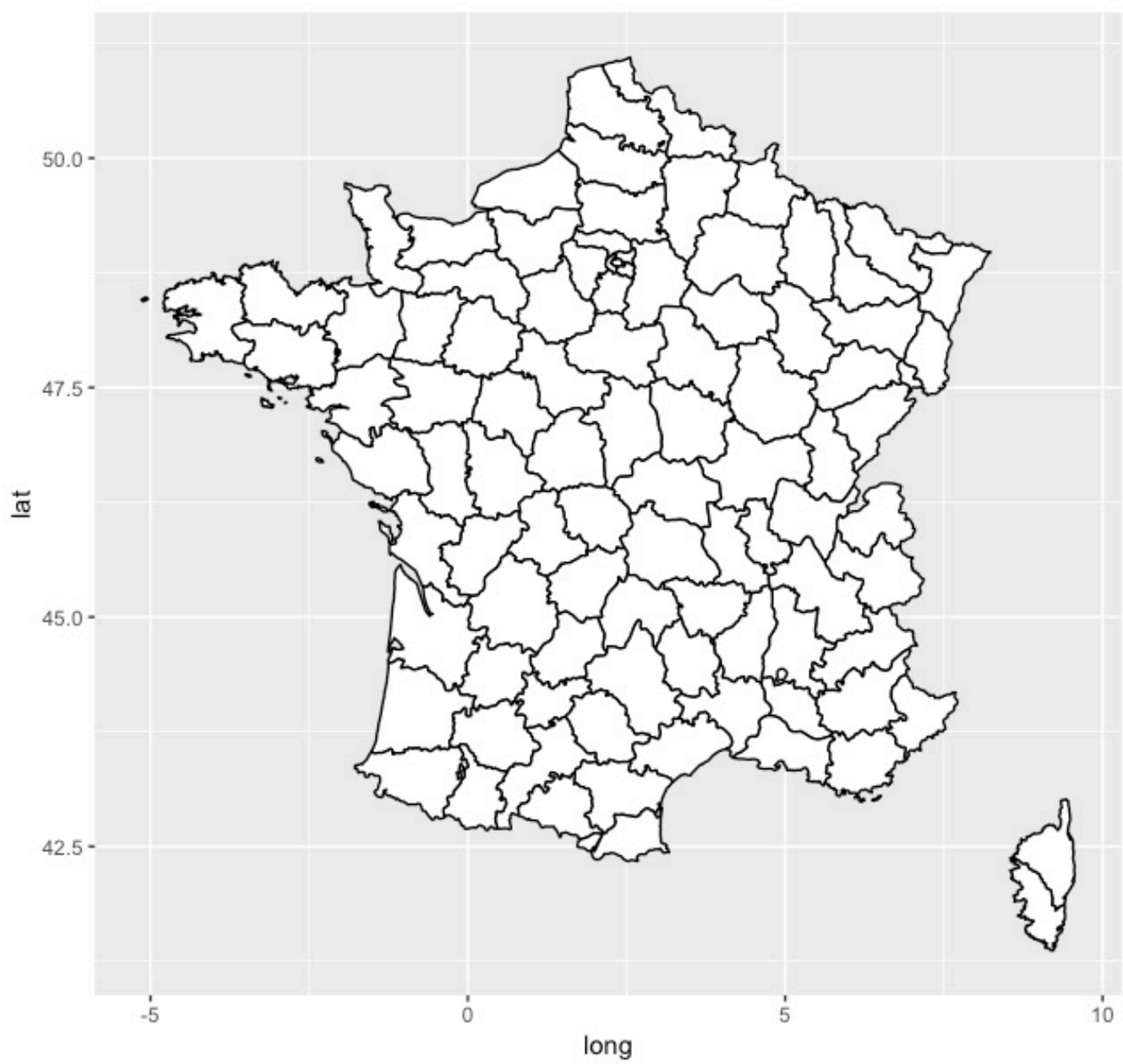
Systèmes de coordonnées



Systèmes de coordonnées

```
ggplot(fr, aes(long,lat, group=group)) +

geom_polygon(fill = "white", color="black") +

coord_quickmap()
```

Systèmes de coordonnées



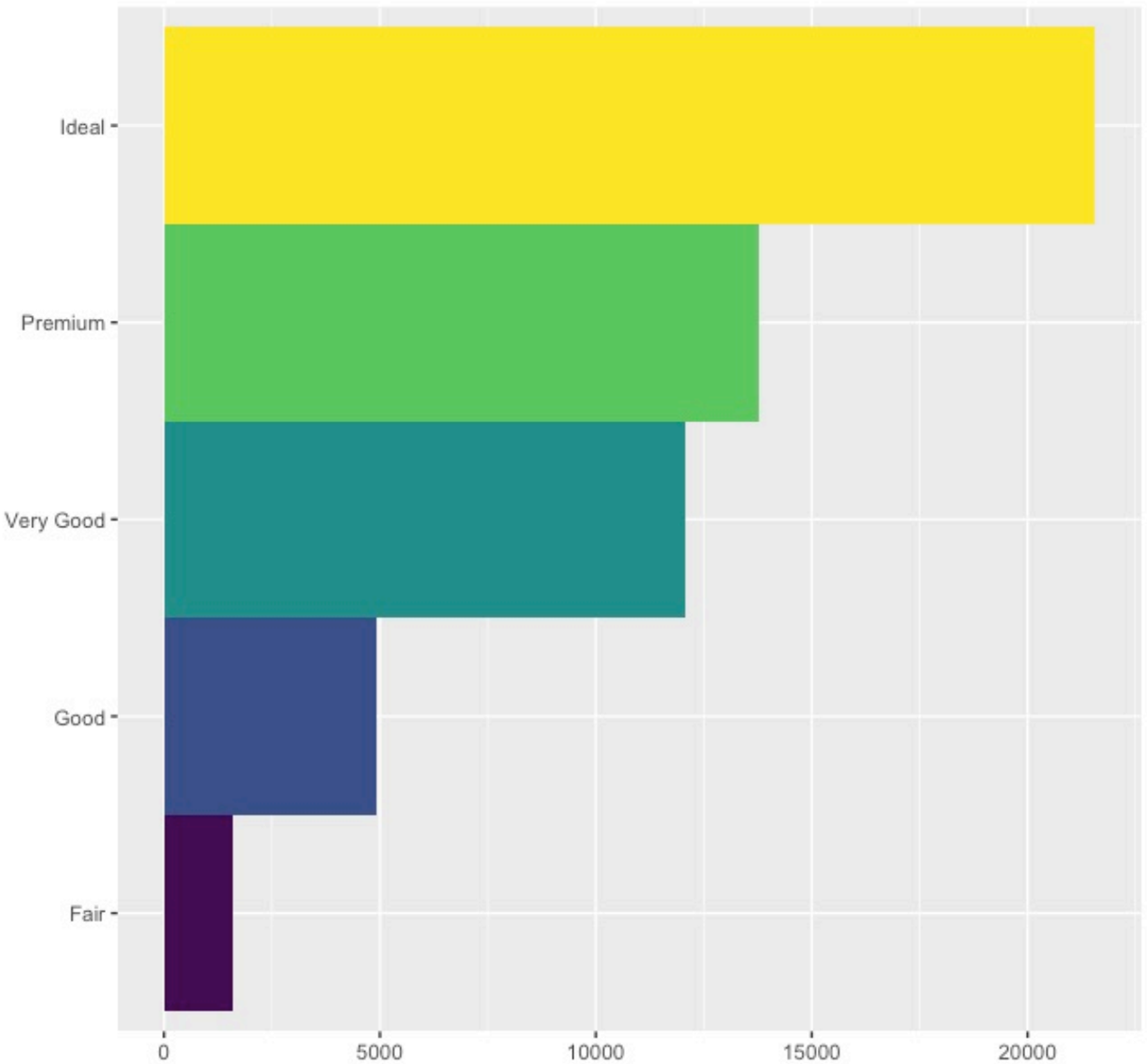
Systèmes de coordonnées

- `coord_polar()` utilise les coordonnées polaires. Cela permet de révéler une connexion intéressante entre diagramme en barres et diagramme de Coxcomb :

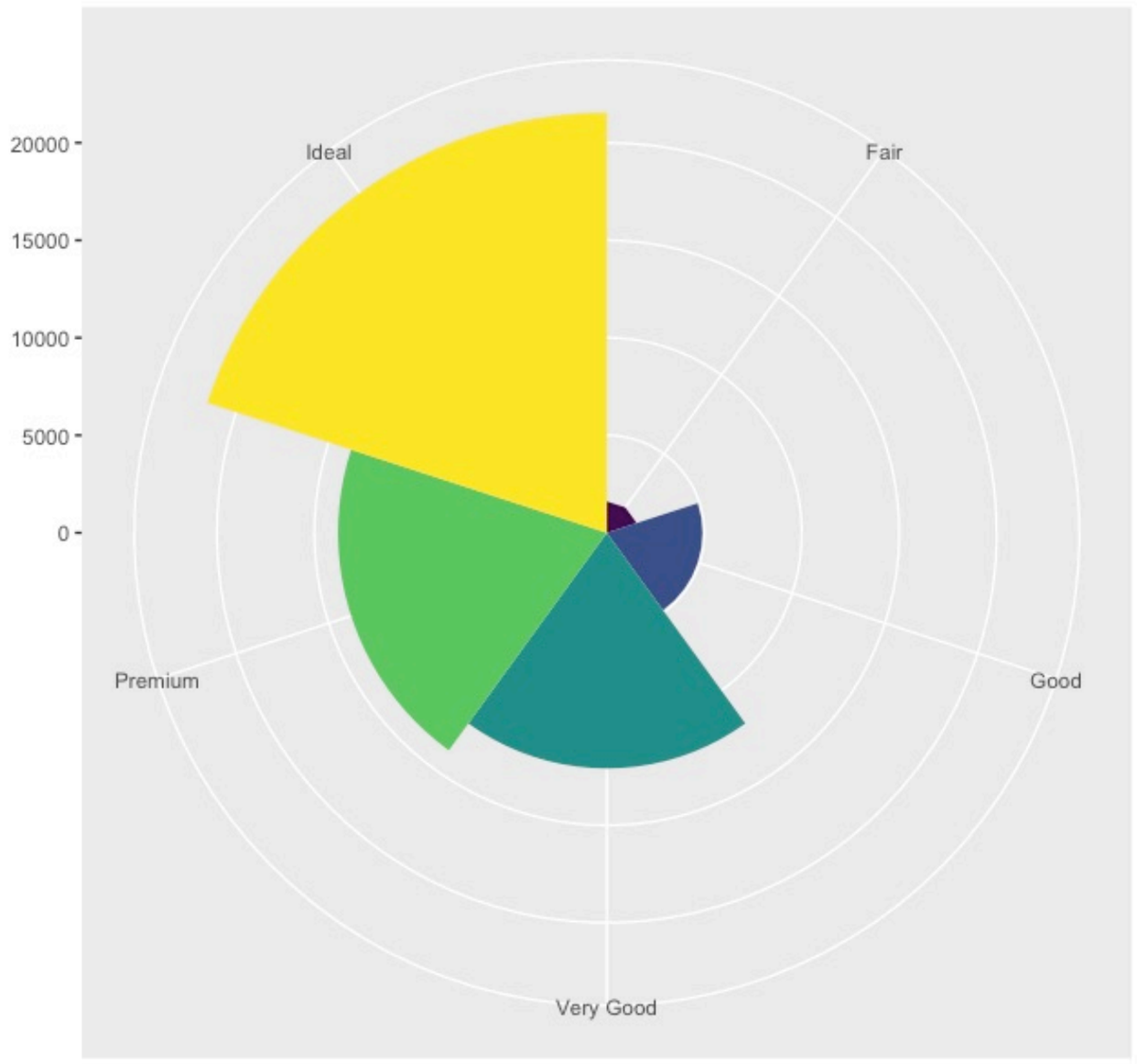
Systèmes de coordonnées

```
bar <- ggplot(data=diamonds) +

  geom_bar(

    mappin = aes(x=cut, fill=cut),

    show.legend = FALSE,

    width =1

  ) +

  theme(aspect.ratio = 1) +

  labs(x = NULL, y = NULL)


bar + coord_flip()

bar + coord_polar()
```

Systèmes de coordonnées



Systèmes de coordonnées



Questions

- Transformer un diagramme en barres empilées en un diagramme circulaire en utilisant `coord_polar()`.

- Que fait `labs()` ?

- Quelle est la différence entre `coord_quickmap()` et `coord_map()` ?

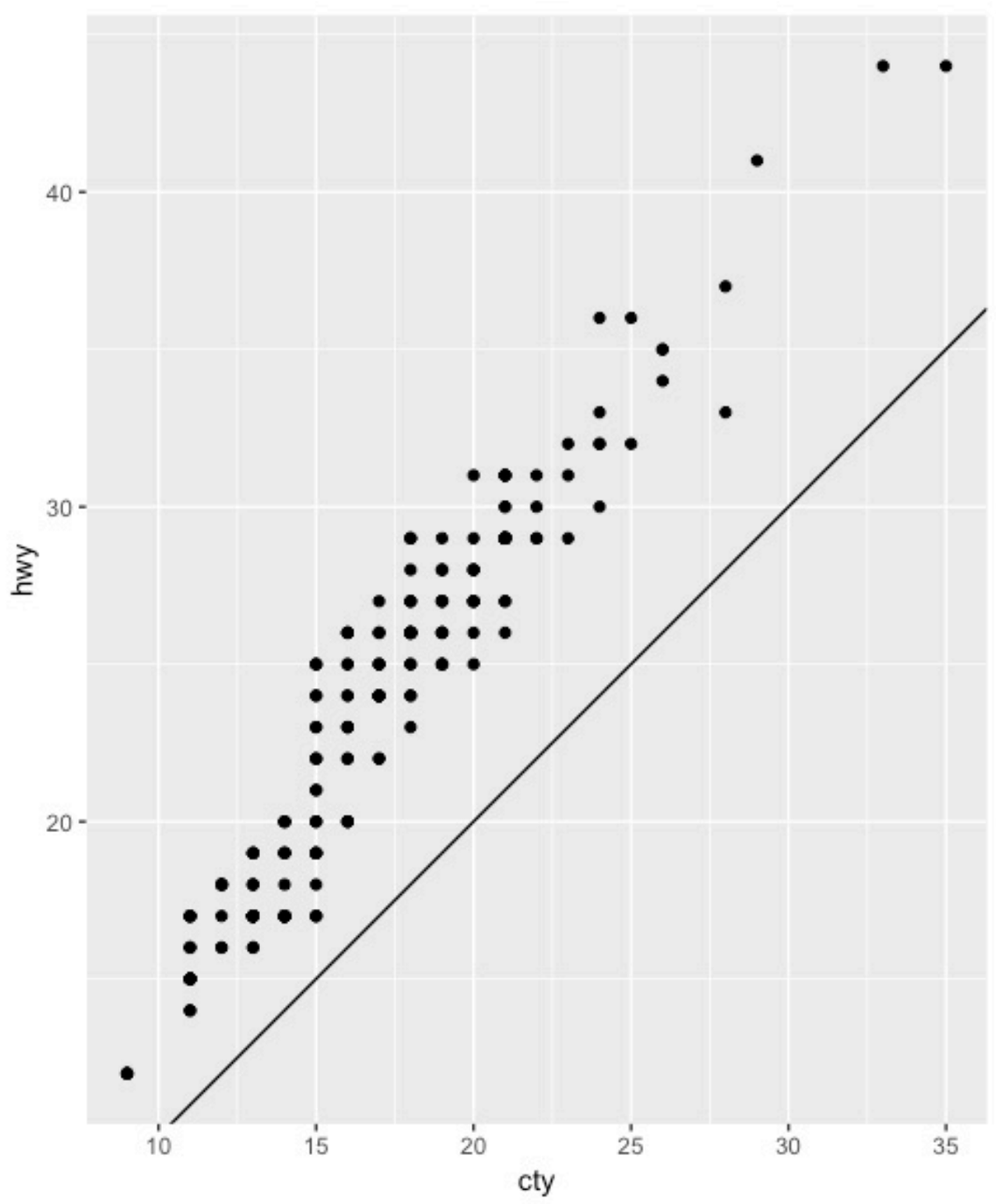
Questions

- Que peut-on apprend du graphique suivant sur la relation entre efficacité énergétique en ville et sur autoroute ? Pourquoi est-il important d'utiliser `coord_fixed()` ? Que fait `geom_abline()` ?

Questions

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

geom_point() +

geom_abline() +

coord_fixed()
```

Questions



Production de graphiques pour diffusion avec ggplot2

Introduction

- Les graphiques sont d'abord utilisés comme outil d'*exploration*.

- Lorsqu'on crée des graphiques d'exploration, on sait à l'avance quelles variables seront affichées.

- Chaque graphique est créé avec un objectif et on peut les observer rapidement et passer au suivant.

- Dans la plupart des analyses, on produit des dizaines voire des centaines de graphiques, dont la plupart ne seront examinés qu'une fois.

Introduction

- Une fois atteint le stade où l'on comprend les données, on doit *communiquer* cette compréhension à un public qui n'a en général pas la même connaissance du sujet et des données.

- Pour aider ce public à développer un bon modèle mental, on doit investir des efforts non négligeables pour rendre les graphiques aussi clairs et explicites que possible.

Prérequis

- On utilisera dans ce qui suit les packages suivants : ggplot2, bien sur, mais également dplyr pour des manipulations de données et des packages d'extension de ggplot2, notamment ggrepel et viridis.

- Plutôt que de charger ces extensions, nous nous référerons à leurs fonctions explicitement avec la notation ::, ce qui permettra de distinguer facilement les fonctions intégrées à ggplot2 et celles provenant d'autres packages.

Labels

- La façon la plus évidente de transformer un graphique d'exploration en graphique d'exposition est de lui adjoindre de bons labels.

- On utilise la fonction `labs()` à cet effet.

- L'exemple suivant ajoute un titre au graphique.

Labels

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  labs(
    title = paste("L'efficacité énergétique tend à diminuer
    avec la taille du moteur")
  )
```

Labels


L'efficacité énergétique tend à diminuer avec la taille du moteur
hwy
40
30
20
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

Labels

- **L’objectif du titre d’un graphique est de résumer ce qu’il révèle.**

- Eviter les titres qui se contentent de décrire ce qu’est le graphique, comme « nuage de points de l’efficacité énergétique en fonction de la taille du moteur ».

- Si on a besoin d’ajouter plus de texte, deux autres légendes peuvent être utiles :

Labels

- `subtitle` (« sous-titre ») ajoute des éléments additionnels sous le titre, dans une police plus petite

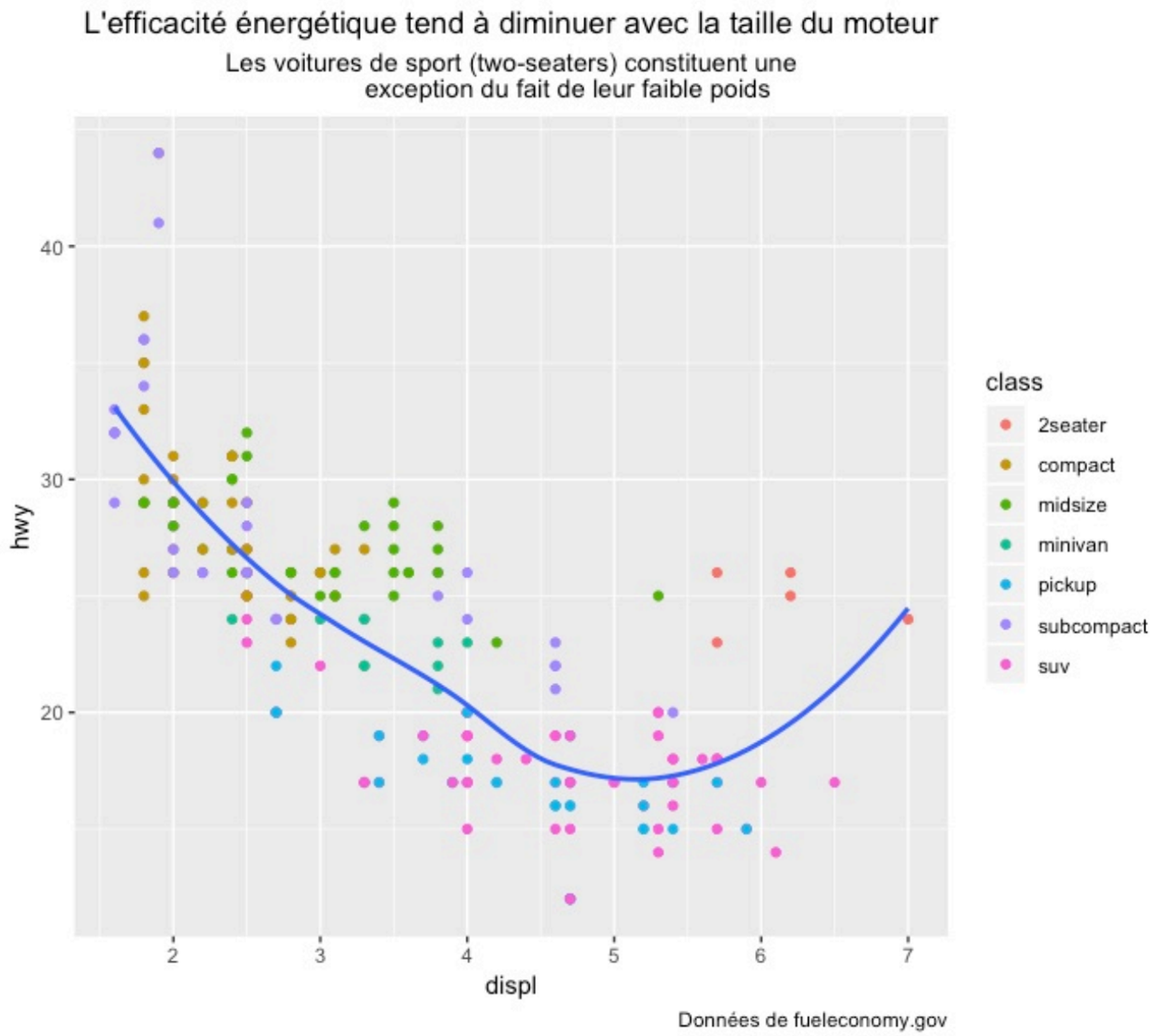
- `caption` (« notice ») ajoute du texte en bas à droite du graphique. C’est souvent utilisé pour décrire la source des données.

- Exemple :

Labels

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5)) +
labs(
  title = paste("L'efficacité énergétique tend à diminuer",
                "avec la taille du moteur"),
  subtitle = "Les voitures de sport (two-seaters) constituent une
              exception du fait de leur faible poids",
  caption = "Données de fueleconomy.gov"
)
```

Labels


L'efficacité énergétique tend à diminuer avec la taille du moteur
Les voitures de sport (two-seaters) constituent une exception du fait de leur faible poids
hwy
displ
40
30
20
2
3
4
5
6
7
class
2seater
compact
midsize
minivan
pickup
subcompact
suv
Données de fueleconomy.gov

Labels

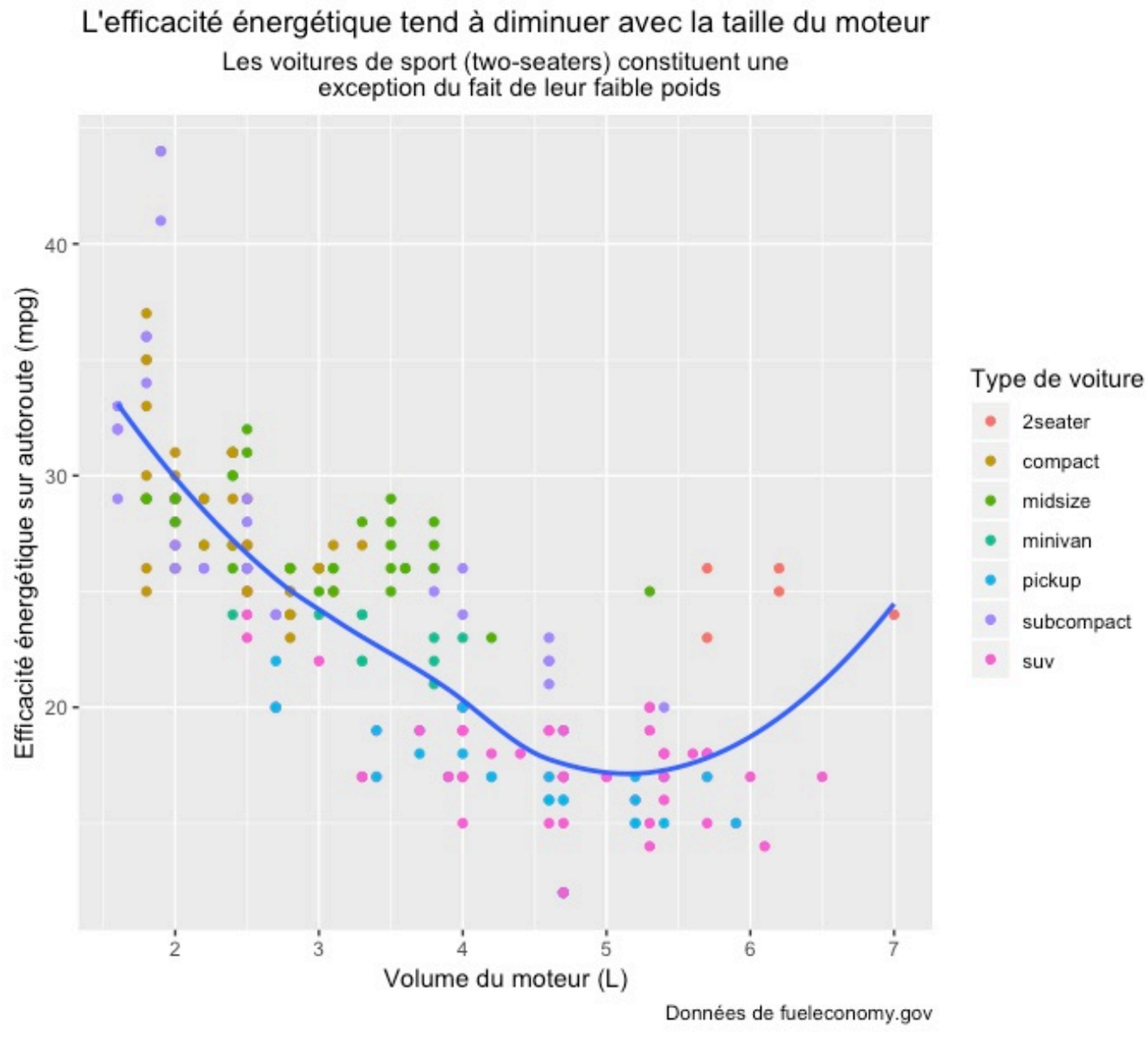
- On peut aussi utiliser `labs()` pour remplacer les titres des axes et légendes.

- Il est souvent judicieux de remplacer les noms de variables brefs par des descriptions plus détaillées et d'inclure les unités :

Labels

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme(plot.title = element_text(hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5)) +
  labs(
    title = paste("L'efficacité énergétique tend à diminuer",
                  "avec la taille du moteur"),
    subtitle = "Les voitures de sport (two-seaters) constituent une
    exception du fait de leur faible poids",
    caption = "Données de fueleconomy.gov",
    x = "Volume du moteur (L)",
    y = "Efficacité énergétique sur autoroute (mpg)",
    colour = "Type de voiture"
  )
```

Labels


L'efficacité énergétique tend à diminuer avec la taille du moteur
Les voitures de sport (two-seaters) constituent une exception du fait de leur faible poids
Efficacité énergétique sur autoroute (mpg)
40
30
20
2
3
4
5
6
7
Volume du moteur (L)
Données de fueleconomy.gov
Type de voiture
2seater
compact
midsize
minivan
pickup
subcompact
suv

Question

- La courbe de `geom_smooth()` peut être trompeuse, car `hwy` est dévié vers le haut pour les gros moteurs, à cause de l’inclusion des voitures de sport légères. Adapter et afficher un meilleur modèle.

Annotations

- En plus de légender les principaux composants d'un graphique, il est souvent utile *d'annoter* des observations individuelles ou des groupes d'observations.

- Le premier outil dont on dispose pour cela est `geom_text()`.

- `geom_text()` fonctionne de façon analogue à `geom_point()`, mais avec une esthétique supplémentaire : `label`.

- Cela permet d'ajouter des annotations textuelles aux graphiques.

Annotations

- Ces annotations peuvent provenir de deux sources.
- La première consiste à avoir un « tibble » qui les fournit (on reviendra sur cette notion dans la suite du cours).

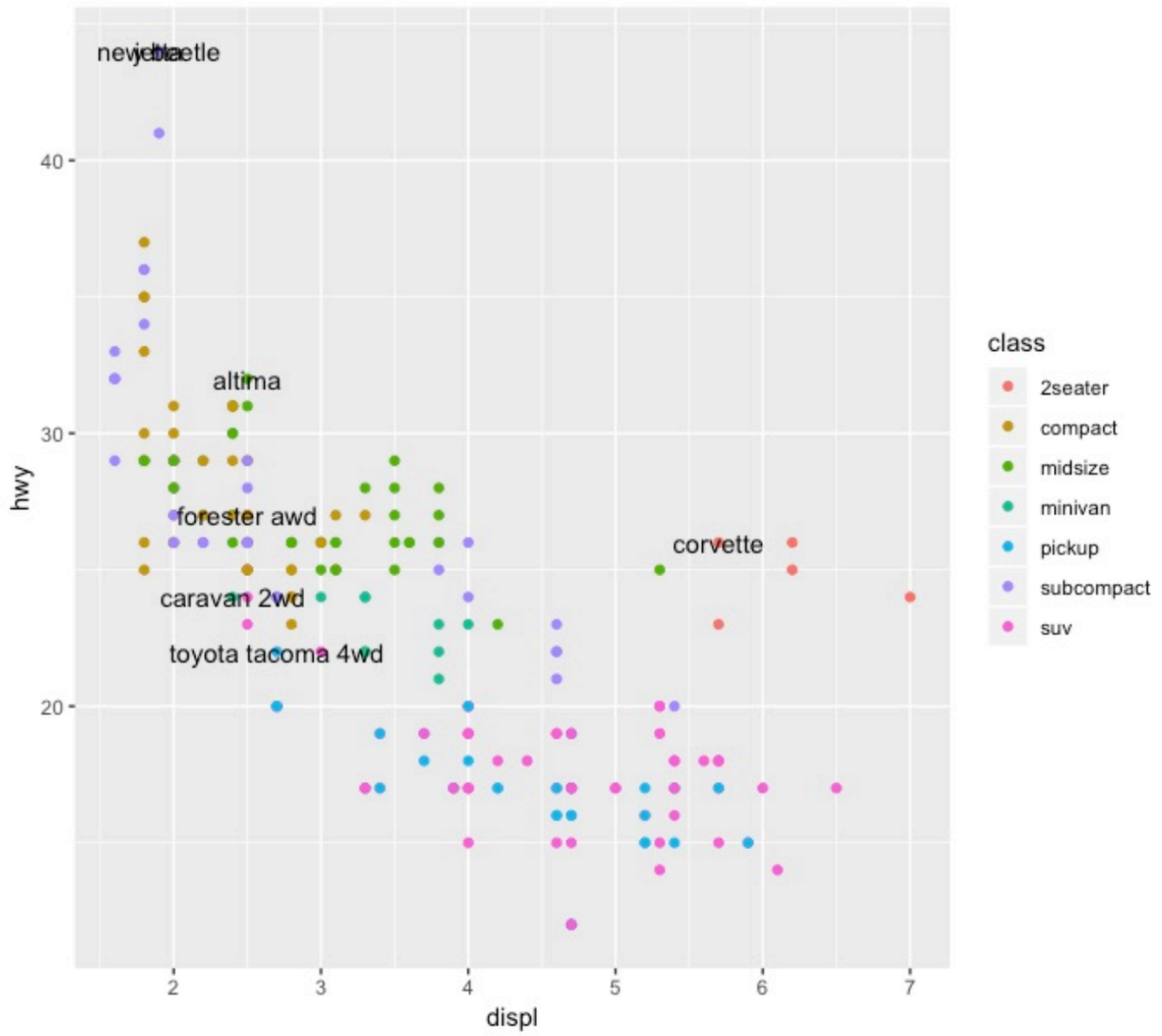
- Le graphique suivant illustre une approche utile : extraire la voiture la plus efficace de chaque catégorie avec dplyr, puis l'annoter sur le graphique.

Annotations

```
best_in_class <- mpg %>%

  group_by(class) %>%

  filter(row_number(desc(hwy)) == 1)
```

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

  geom_point(aes(color = class)) +

  geom_text(aes(label=model), data=best_in_class)
```

Annotations



Annotations

- La lecture est difficile car les annotations se superposent les unes sur les autres et sur les points.

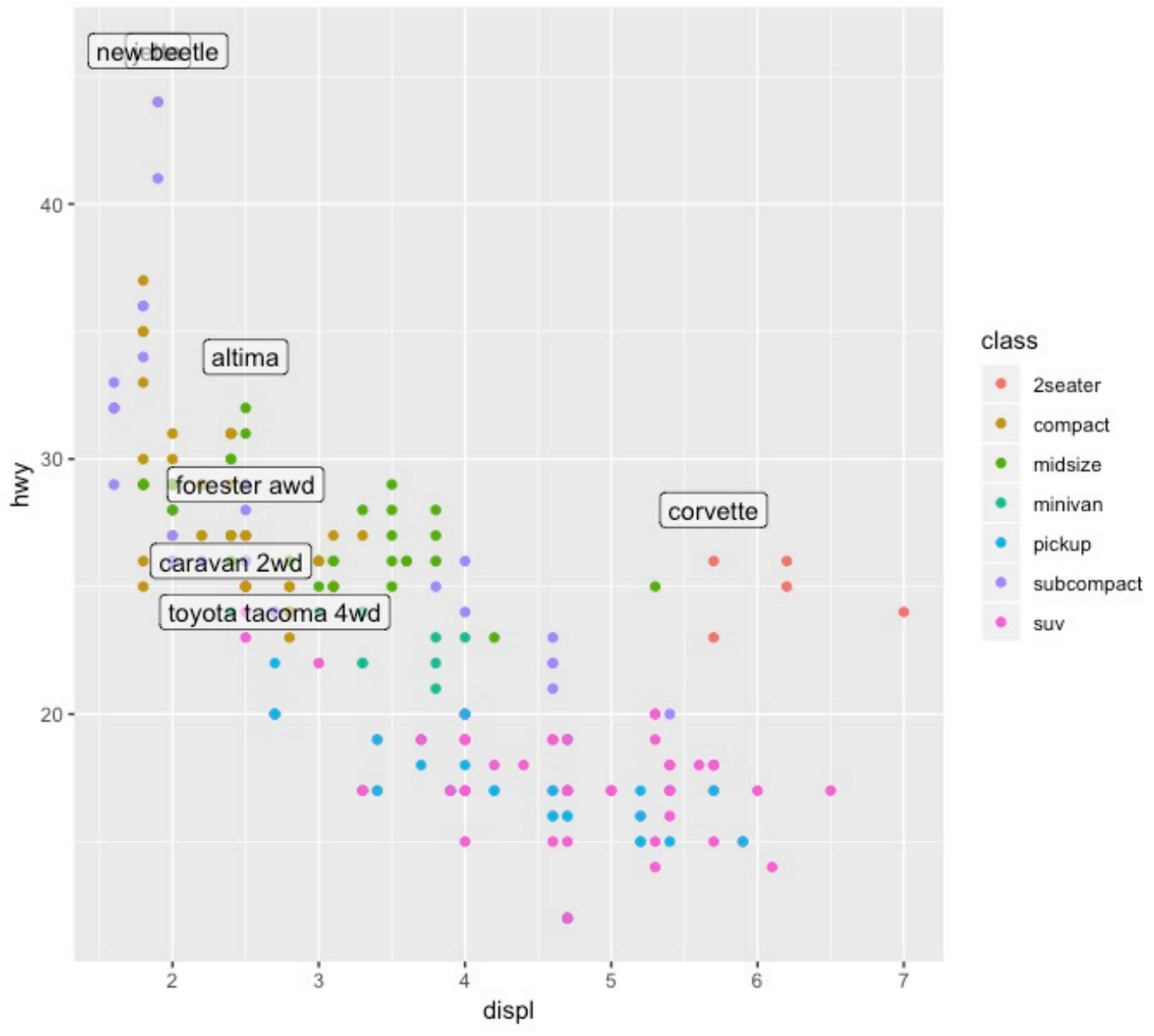
- Il est possible d'améliorer légèrement les choses en utilisant `geom_label()`, qui dessine un rectangle sous le texte.

- On peut également utiliser le paramètre `nudge_y` pour déplacer les annotations au-dessus des points correspondants.

Annotations

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_label(aes(label = model), data = best_in_class,
             nudge_y = 2, alpha = 0.5)
```

Annotations



Annotations

- C’est un peu mieux, mais les deux annotations du coin supérieur gauche sont pratiquement l’une sur l’autre.

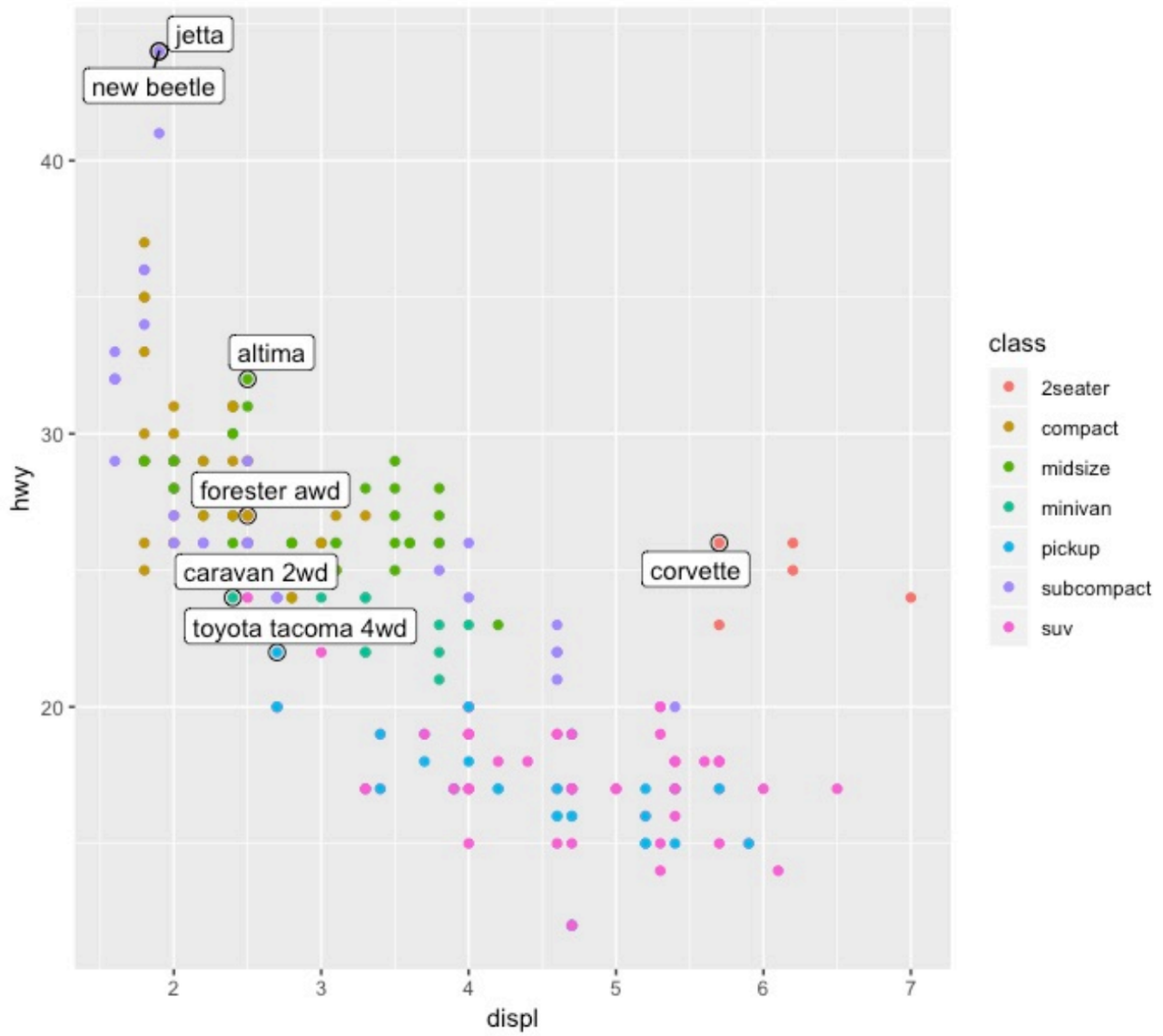
- Pour régler ce problème, on peut utiliser le package ggrepel : avec celui-ci, les annotations sont ajustées automatiquement de manière à éviter toute superposition.

- Exemple :

Annotations

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

  geom_point(aes(color = class)) +

  geom_point(size = 3, shape = 1, data = best_in_class) +

  ggrepel::geom_label_repel(aes(label = model), data = best_in_class)
```

Annotations



Annotations

- On a utilisé ici une autre technique intéressante : l'ajout d'une seconde couche de points larges et vides pour mettre en évidence les points annotés.

- On peut également utiliser des annotations directement placées sur le graphique pour remplacer la légende :

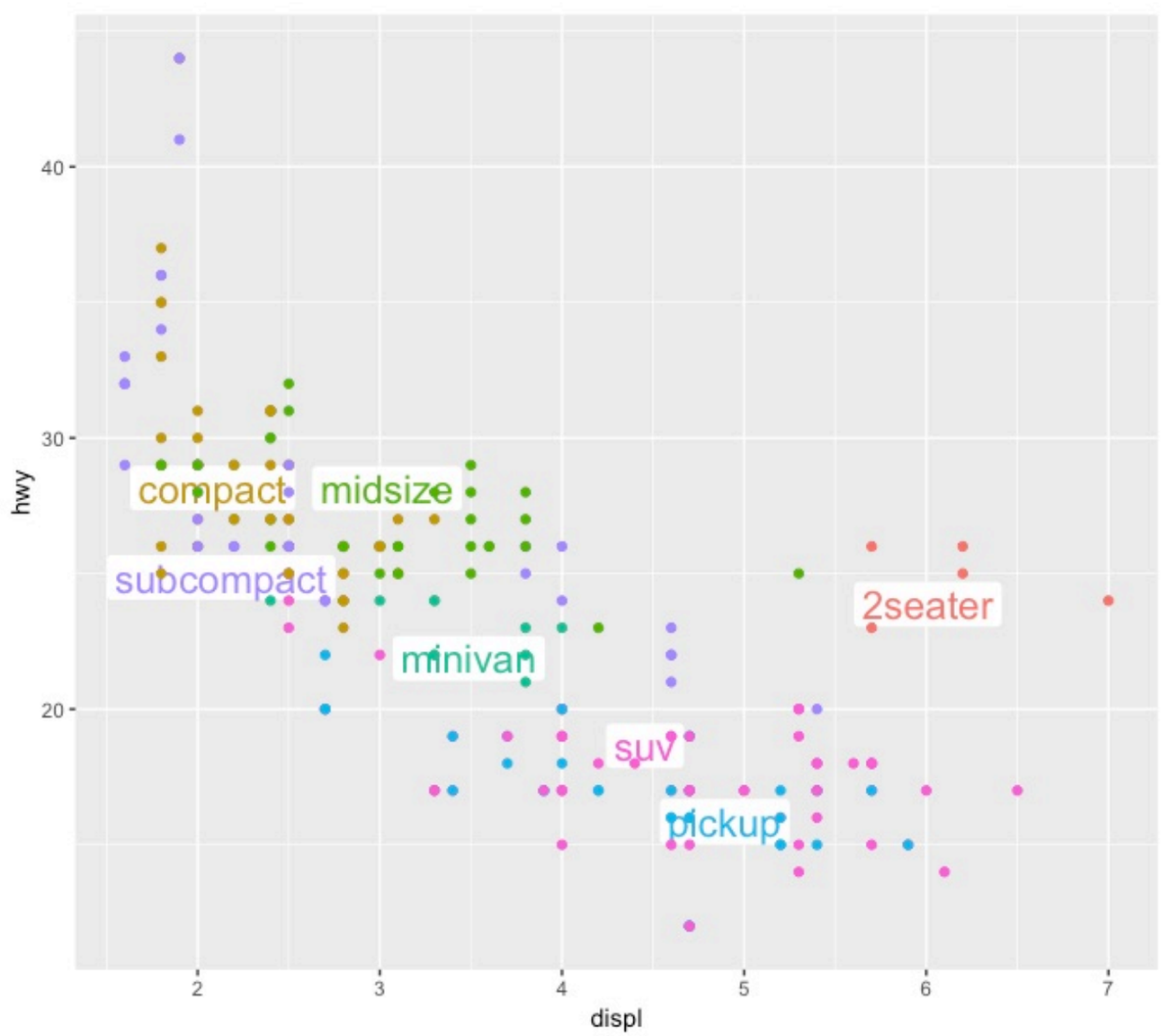
Annotations

```
class_avg <- mpg %>%
  group_by(class) %>%
  summarize(
    displ = median(displ),
    hwy = median(hwy)
  )
```

Annotations

```
ggplot(mpg, aes(displ, hwy, color = class)) +

  ggrepel::geom_label_repel(aes(label = class),

                            data = class_avg,

                            size = 6,

                            label.size = 0,

                            segment.color = NA) +

geom_point() +

theme(legend.position = "none")
```

Annotations



Annotations

- On peut également souhaiter ajouter une seule annotation au graphe.

- Il faut pour cela créer un cadre de données.

- Par exemple, pour la placer dans un coin du graphe, on peut utiliser `summarize()` pour calculer les valeurs maximales de `x` et `y` :

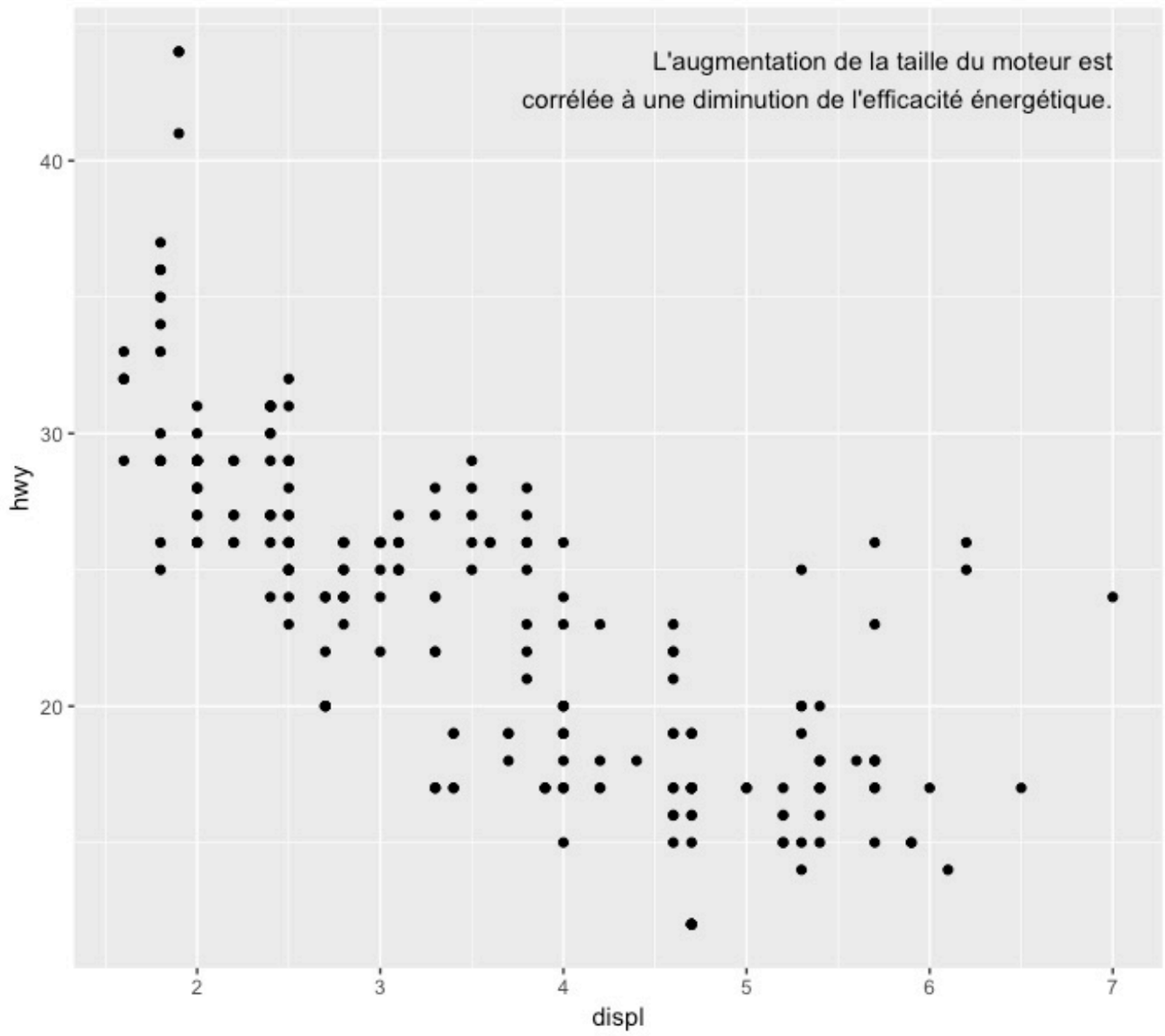
Annotations

```
label <- mpg %>%
  summarize(
    displ = max(displ),
    hwy = max(hwy),
    label = paste("L'augmentation de la taille du moteur est\ncorrélée",
    "à une diminution de l'efficacité énergétique.")
  )
```

Annotations

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

  geom_point() +

  geom_text(

    aes(label = label),

    data = label,

    vjust ="top",

    hjust = "right"

)
```

Annotations



Annotations

- Si on souhaite placer le texte exactement sur les limites du graphique, o peut utiliser `+Inf` et `-Inf`.

- Dans ce cas, il n'est plus nécessaire de calculer les positions à partir de mpg, et on peut utiliser `tibble()` pour créer le cadre de données :

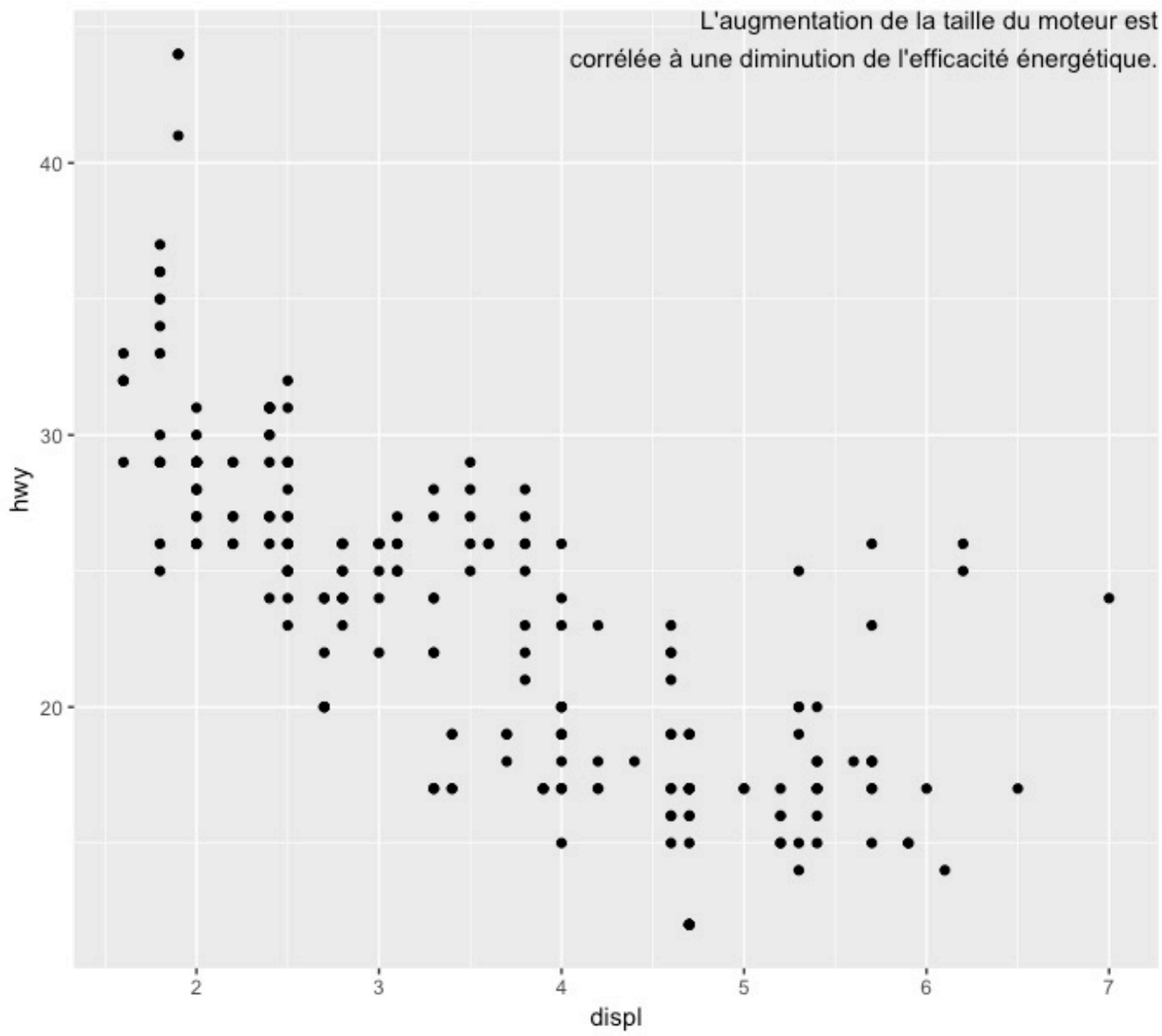
Annotations

```
label <- tibble(
  displ = Inf,
  hwy = Inf,
  label = paste("L'augmentation de la taille du moteur est\ncorrélée",
                "à une diminution de l'efficacité énergétique.")
)
```

Annotations

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

  geom_point() +

  geom_text(

    aes(label = label),

    data = label,

    vjust ="top",

    hjust = "right"

  )
```

Annotations



Annotations

- Chacun des paramètres **`hjust`** et **`vjust`** peut prendre trois valeurs : « left », « center » et « right » pour **`hjust`**, et « top », « center » et « bottom » pour **`vjust`**, ce qui génère 9 combinaisons possibles.

- Notons que d’autres géomes, non textuels, sont disponibles pour améliorer les annotations des graphiques :

Annotations

- `geom_hline()` et `geom_vline()` peuvent être utilisés pour rajouter des lignes de référence. Il est souvent préférable de les rendre épaisses (`size = 2`) et blanches (`color = white`) et les dessiner sous la couche de données principale, de façon à ce qu'elles soient faciles à voir sans détourner l'attention des données.

- `geom_rect()` permet de dessiner un rectangle autour de points particuliers. Les coordonnées du rectangle sont définies par les esthétiques `xmin`, `xmax`, `ymin` et `ymax`.

Annotations

- `geom_segment()` avec un argument `arrow` peut attirer l'attention sur un point à l'aide d'une flèche. Les esthétiques `x`, `y`, `xend` et `yend` définissent respectivement les points de départ et d'arrivée.

Questions

- Utiliser `geom_text()` avec des positions infinies pour placer du texte au quatre coins du graphique précédent.

- Lire la documentation pour `annotate()`. Comment peut-on l'utiliser pour annoter un graphique sans avoir à créer de tibble ?

- Comment les annotations créées avec `geom_text()` interagissent-elles avec le facettage ? Comment ajouter une annotation à une seule facette ? Comment ajouter une annotation à toutes les facettes ? (Indication : A quelles données correspondent les facettes ?)

Questions

- Quels arguments de `geom_label()` contrôlent l'apparence de la boîte d'arrière-plan ?

- Quels sont les quatre arguments de `arrow()` ? Comment fonctionnent-ils ? Créer une série de graphiques pour illustrer les options les plus importantes.

Echelles

- La troisième façon d’améliorer un graphique est d’ajuster ses échelles.

- Celles-ci contrôlent la correspondance entre les valeurs des données et les éléments visuels.

- Normalement, ggplot2 les ajoute automatiquement.

- Par exemple, si on soumet :

Echelles

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

  geom_point(aes(color = class))
```

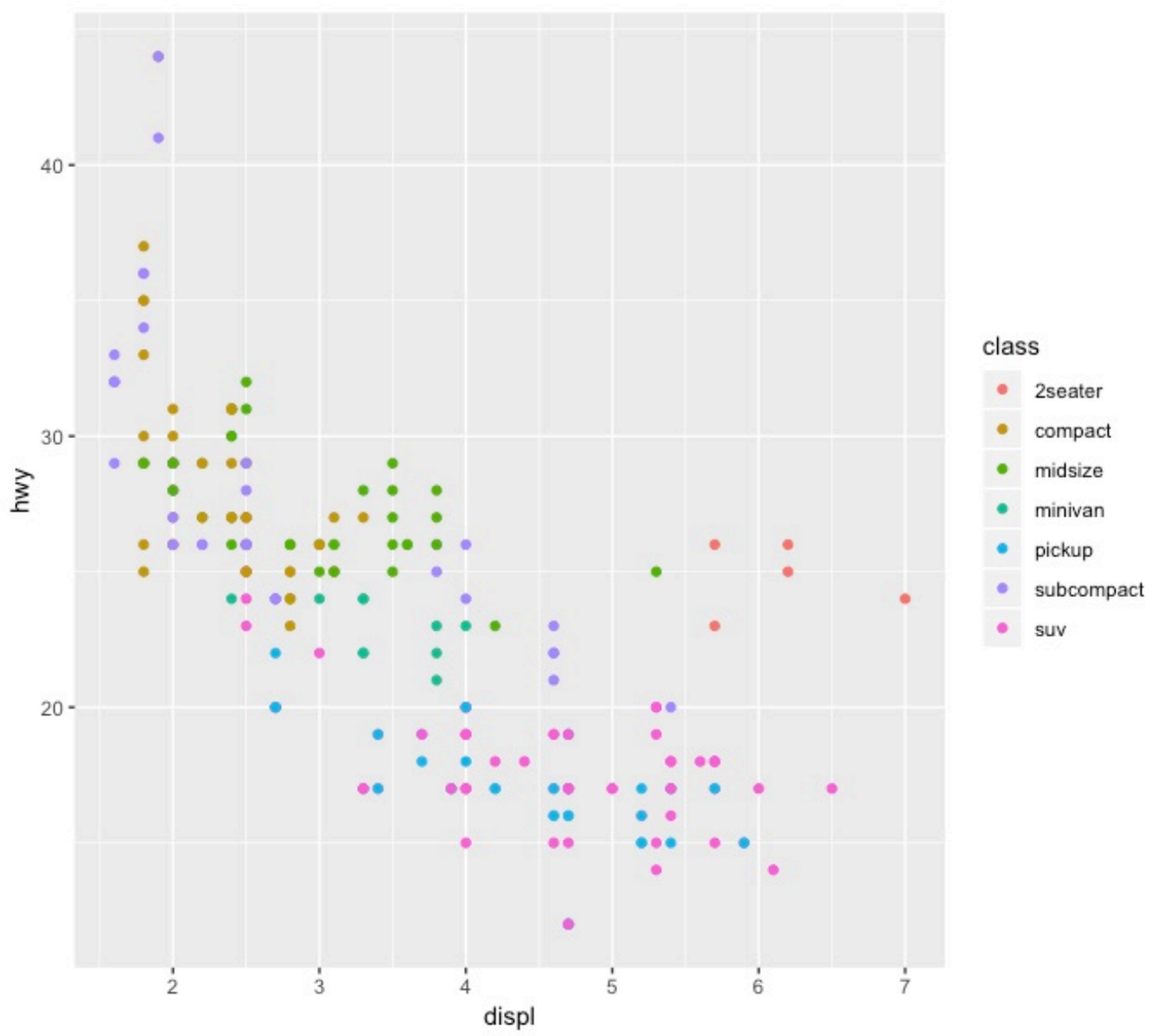
Echelles

- On obtient le même résultat qu'en soumettant :

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

  geom_point(aes(color = class)) +

  scale_x_continuous() +

  scale_y_continuous() +

  scale_color_discrete()
```

- C'est-à-dire :

Echelles


40
30
20
hwy
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

Echelles

- On remarque que les échelles ont le même schéma de désignation : `scale_`, suivi du nom de l'esthétique, du tiret bas `_` et du nom de l'échelle.

- Les échelles par défaut sont nommées selon le type de la variable à laquelle elles correspondent : continue (`continuous`), discrète (`discrete`), date-heure (`datetime`).

- Il en existe de nombreuses autres.

Echelles

- On peut souhaiter remplacer les échelles par défaut pour deux raisons :

1. Ajuster certains paramètres, par exemple modifier les graduations sur les axes, ou les légendes.

2. Remplacer entièrement l’échelle par une autre.

- On en verra des exemples dans la suite.

Graduations et légendes

- Les graduations des axes et le texte qui leur est associé dépend principalement de deux arguments : `breaks` et `labels`.

- `breaks` contrôle la position des graduations.

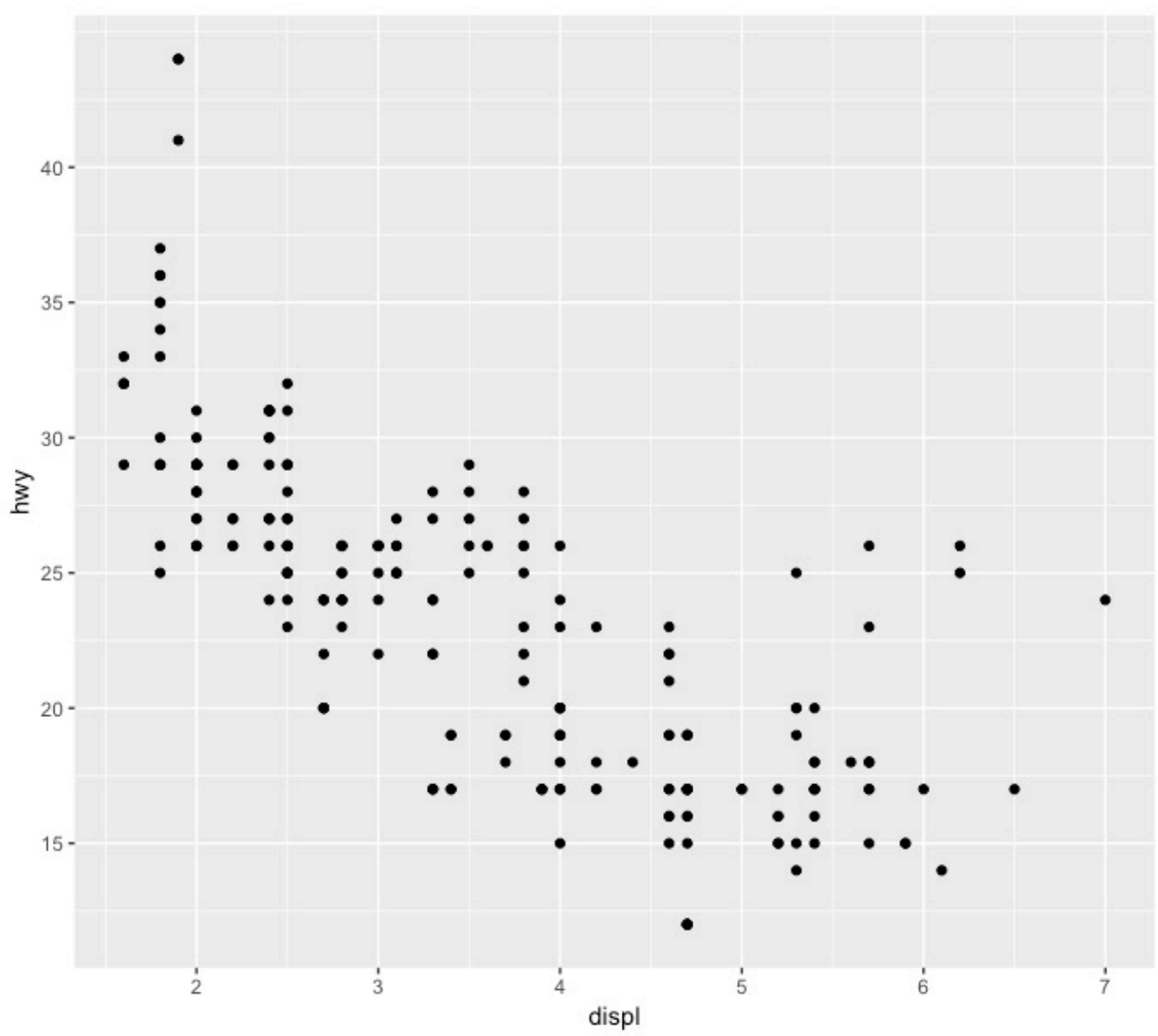
- `labels` contrôle le texte associé.

- L'usage le plus courant de `breaks` consiste à remplacer le choix par défaut :

Graduations et légendes

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

geom_point() +

scale_y_continuous(breaks=seq(15, 40, by = 5))
```

Graduations et légendes



Graduations et légendes

- On peut utiliser `labels` de façon similaire, avec un vecteur caractère de même longueur que breaks.

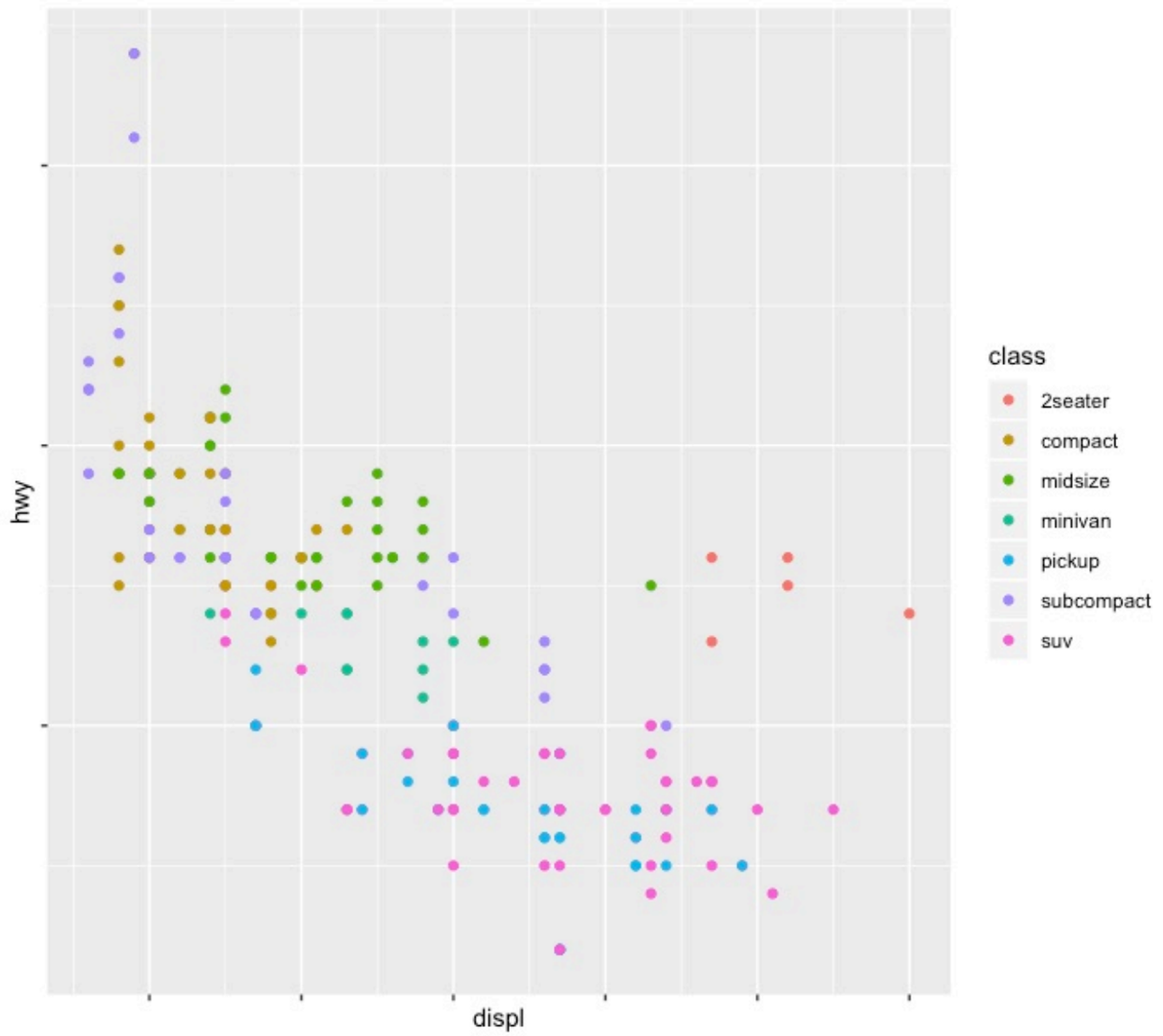
- On peut également utiliser `NULL` pour supprimer entièrement le texte des graduations.

- C’est notamment utile pour les cartes ou pour publier des graphiques qui ne doivent pas divulguer de nombres exacts (clause de confidentialité) :

Graduations et légendes

```
ggplot(mpg,aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  scale_x_continuous(labels = NULL) +
  scale_y_continuous(labels = NULL)
```

Graduations et légendes



Graduations et légendes

- On peut aussi utiliser `breaks` et `labels` pour contrôler l'apparence des légendes.

- Collectivement, les axes et légendes sont appelés *guides*.

- Les axes sont utilisés pour les esthétiques `x` et `y`, et les légendes pour tout le reste.

Graduations et légendes

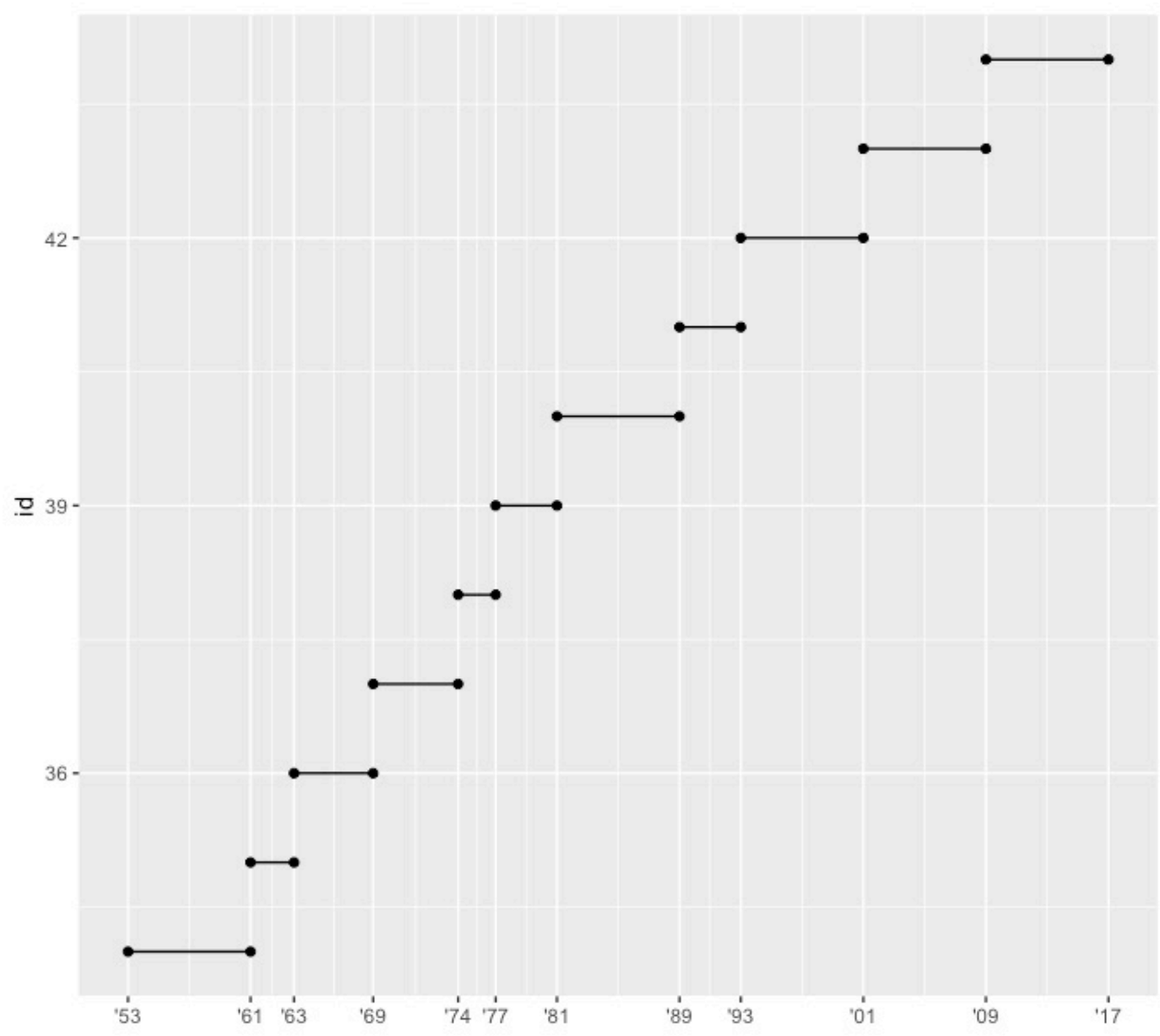
- `breaks` peut aussi être utile quand on n'a que peu de points de données et qu'on veut que leurs coordonnées soient indiquées explicitement.

- Le graphique suivant montre les années de début et de fin de mandat des présidents américains :

Graduations et légendes

```
presidential %>%
  mutate(id = 33 + row_number()) %>%
  ggplot(aes(start, id)) +
  geom_point() +
geom_point(aes(end, id)) +
geom_segment(aes(xend = end, yend = id)) +
scale_x_date(
  NULL,
  breaks = c(presidential$start,max(presidential$end)),
  date_labels = "'%y"
)
```

Graduations et légendes



Graduations et légendes

- Les spécifications des graduations et des légendes pour les dates et dates-heures sont légèrement différentes :

- `date_labels` accepte une spécification de format du même type que `parse_datetime()`.

- `date_breaks` accepte une chaîne comme « 2 days » ou « 1 month ».

Positionnement des légendes

- L'utilisation la plus courante de `breaks` et `labels` consiste à améliorer les axes.

- Ils peuvent être utilisés également pour les légendes, mais on utilisera le plus souvent d'autres techniques pour obtenir le même résultat.

- Pour contrôler la position globale d'une légende, on utilisera une configuration de `theme()`.

Positionnement des légendes

- Les thèmes contrôlent les portions d'un graphique qui ne dépendent pas des données.

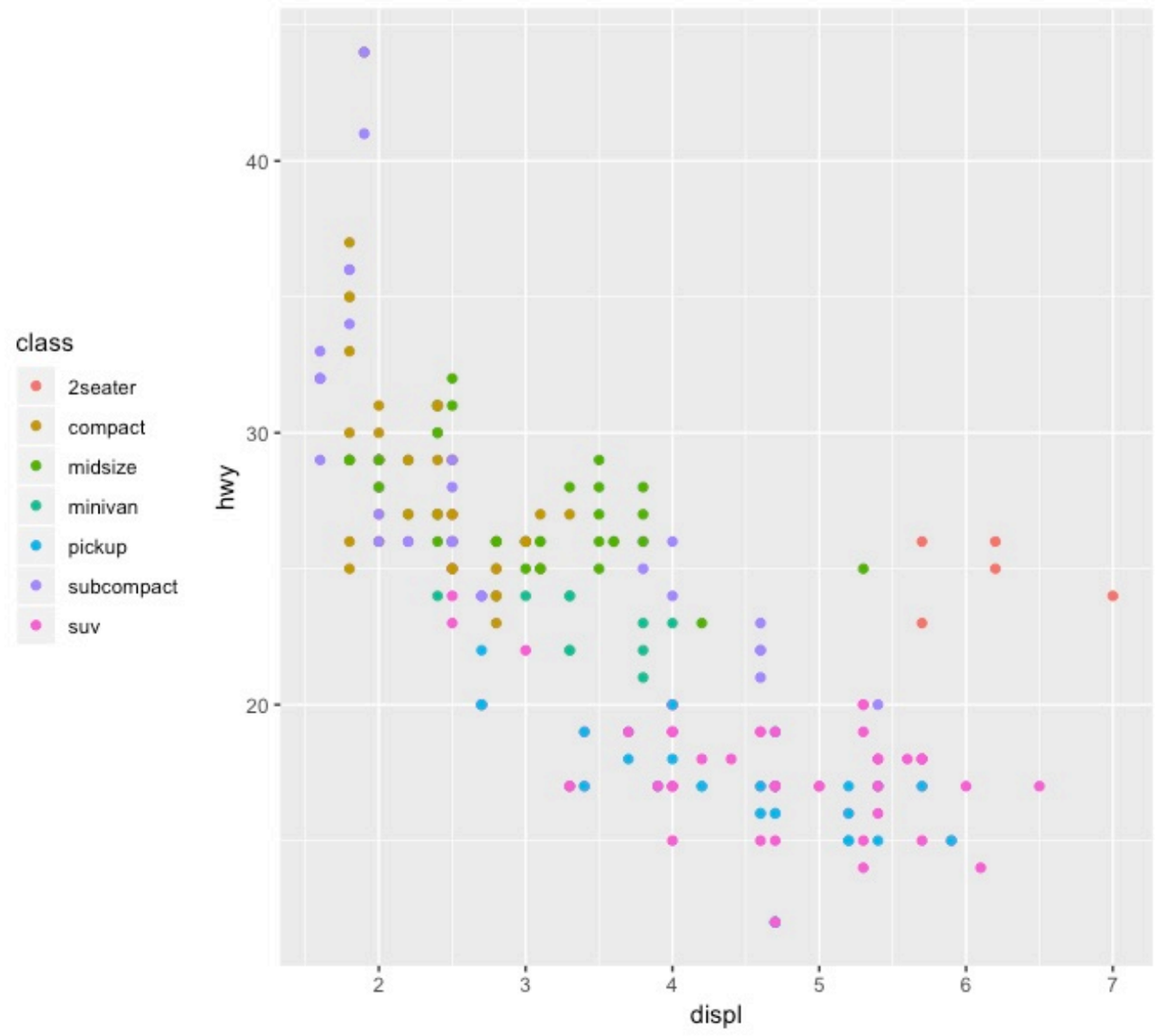
- Le paramètre de thème `legend.position` contrôle l'emplacement de la légende :

Positionnement des légendes

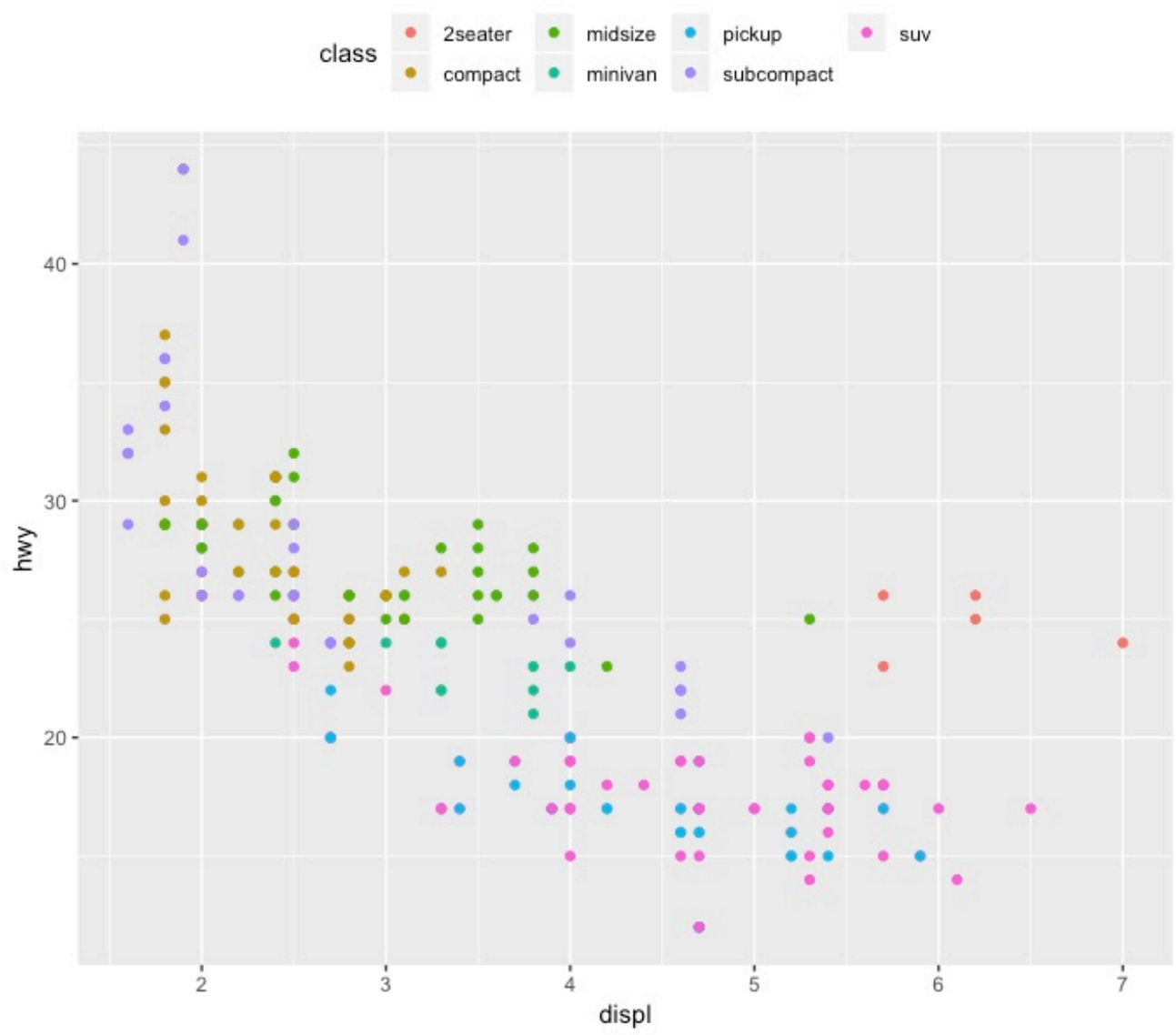
```
base <- ggplot(mpg,aes(displ, hwy)) +

   geom_point(aes(color = class))
```

```
base + theme(legend.position = "left")

base + theme(legend.position = "top")

base + theme(legend.position = "bottom")

base + theme(legend.position = "right") # par défaut
```

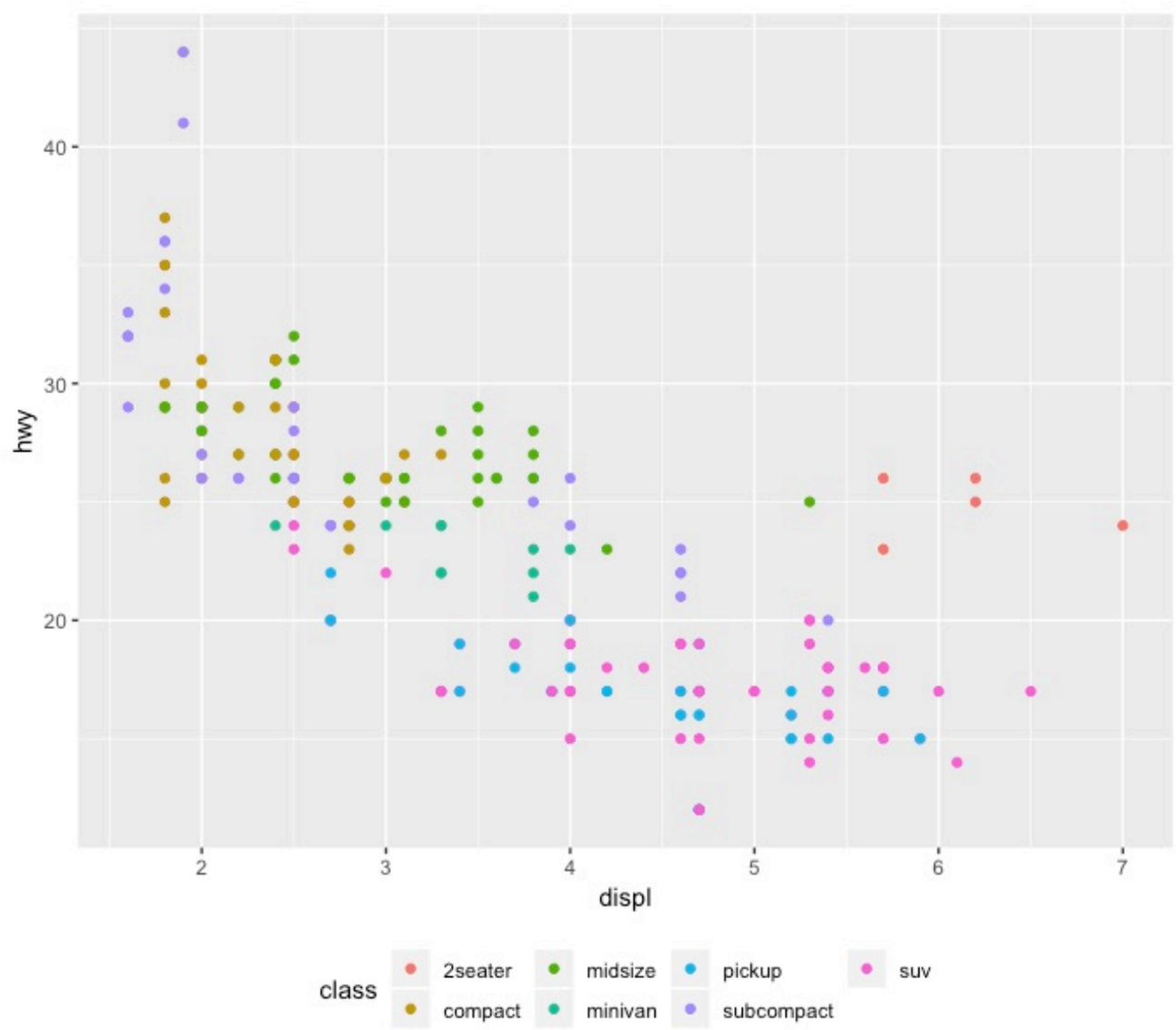
Positionnement des légendes



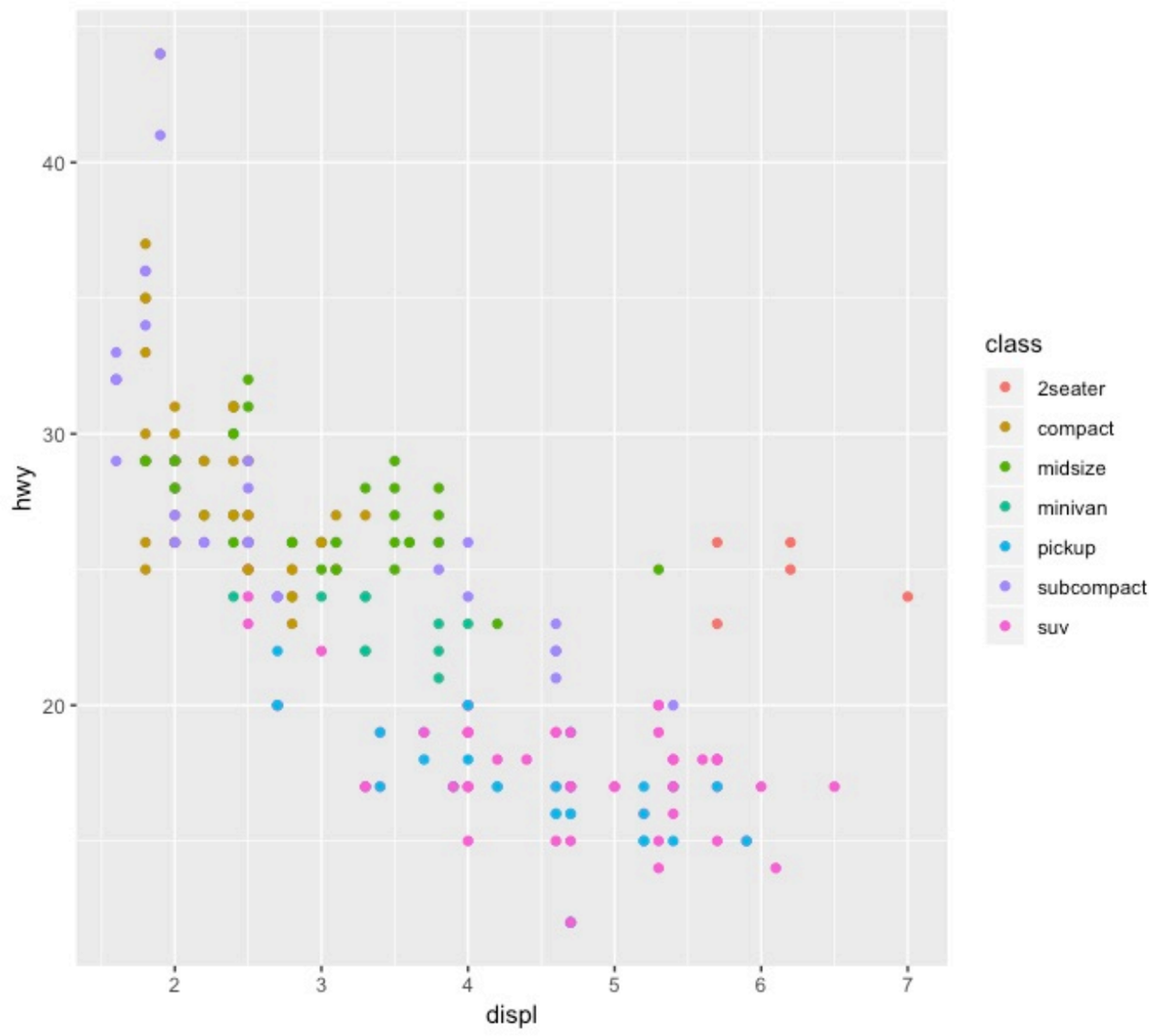
Positionnement des légendes



Positionnement des légendes



Positionnement des légendes



Positionnement des légendes

- On peut utiliser `legend.position = « none »` pour supprimer son affichage.

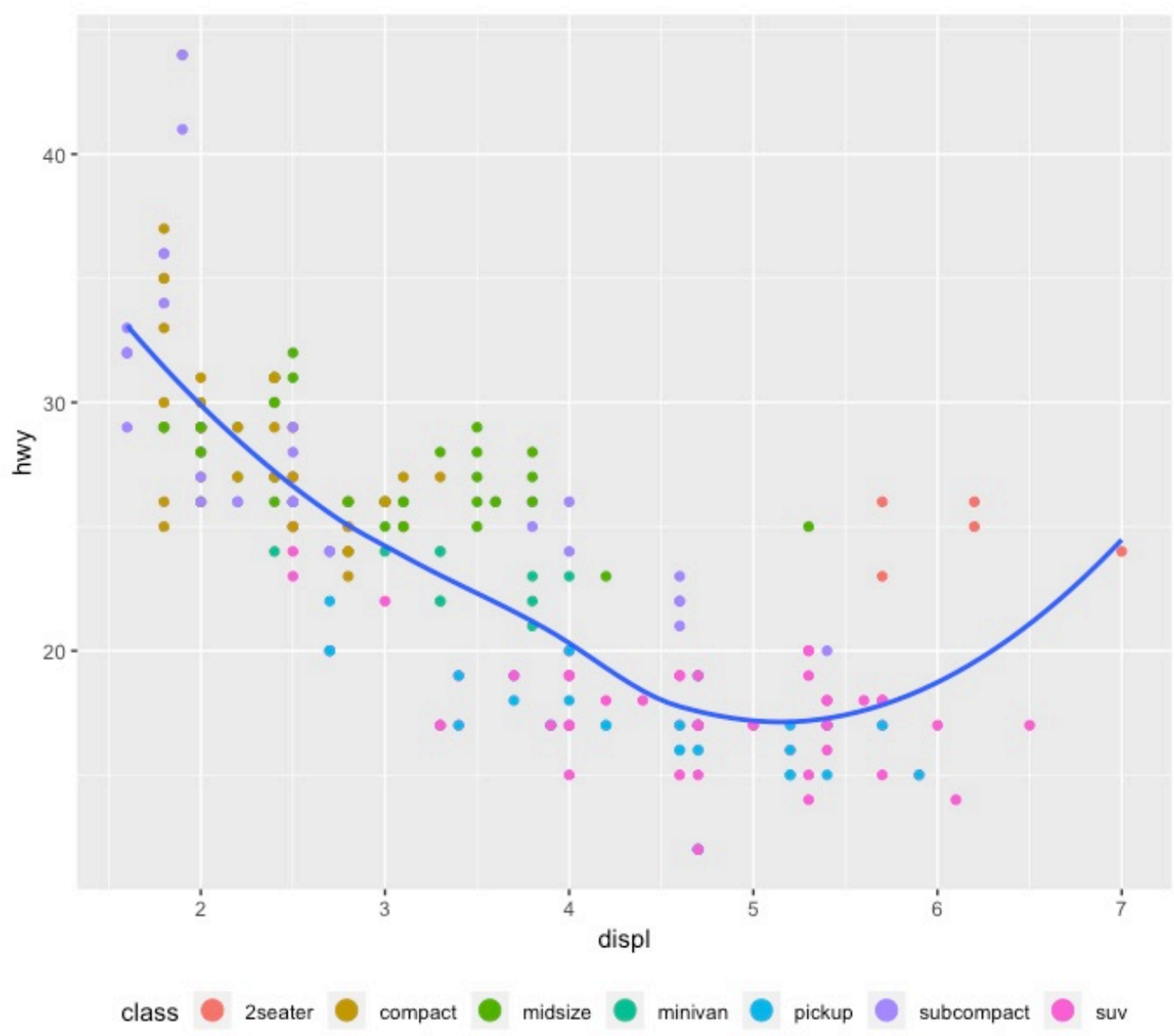
- Pour contrôler l'affichage des légendes individuelles, on utilise `guides()` conjointement à `guide_legend()` ou `guide_colorbar()`.

- L'exemple suivant met en évidence deux aspects importants des légendes : le nombre de rangées sur lesquelles apparaît la légende (`nrow`) et le remplacement d'une esthétique pour rendre les points plus épais.

Positionnement des légendes

```
ggplot(mpg,aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth(se = FALSE) +
  theme(legend.position = "bottom") +
  guides(
    color = guide_legend(
      nrow = 1,
      override.aes = list(size = 4)
    )
  )
```

Positionnement des légendes



Remplacement d’une échelle

- Plutôt que d’ajuster légèrement les détails, on peut remplacer entièrement l’échelle.
- Deux types d’échelles sont particulièrement susceptibles de justifier un remplacement :

1. Les échelles de positions continues
2. Les échelles de couleur

Remplacement d'une échelle

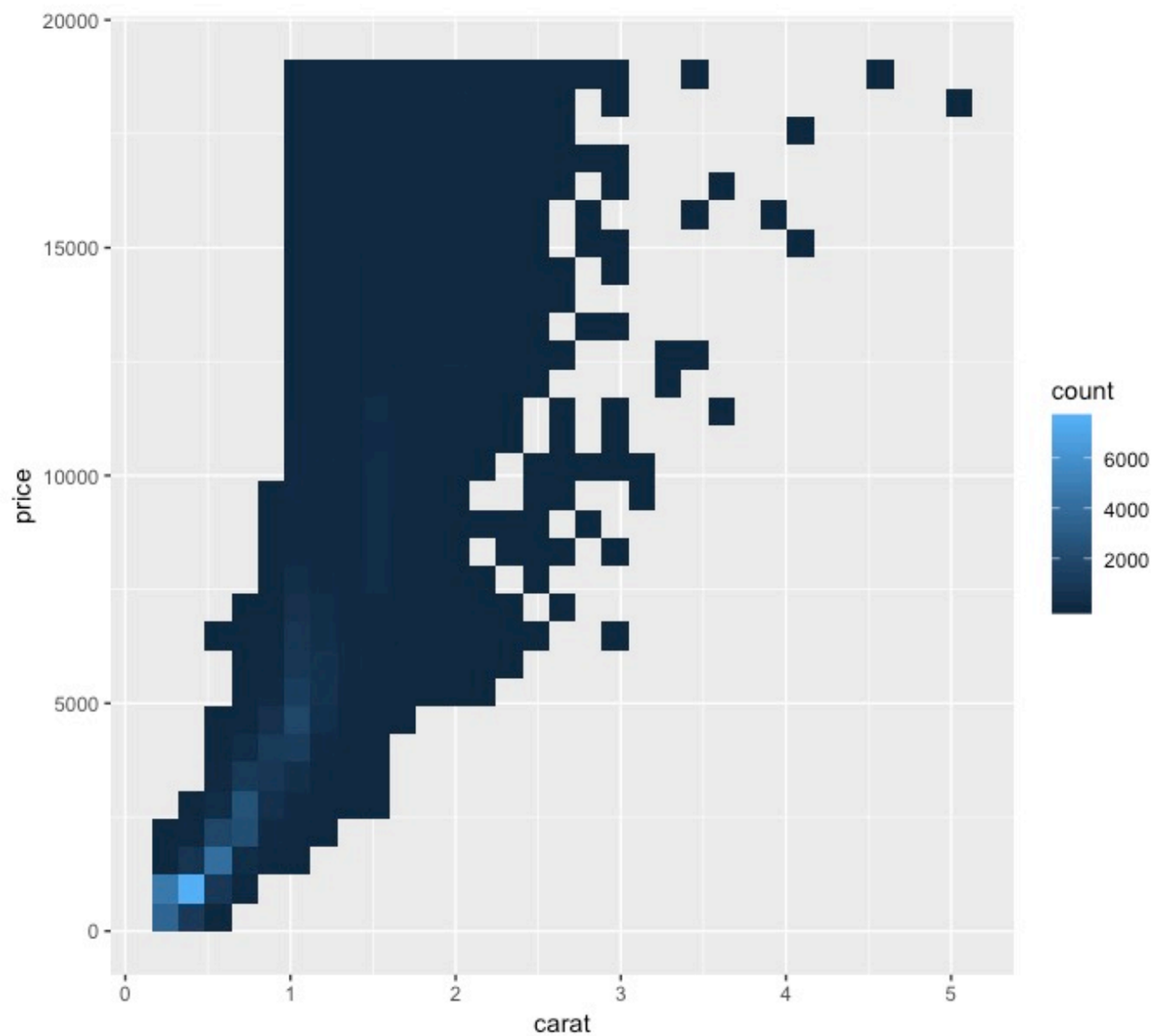
- Par exemple, afficher une transformation de la variable peut parfois être très utile.

- Par exemple, pour les diamants, il est plus facile de visualiser la relation entre carat et prix si nous leur appliquons une transformation logarithmique :

Remplacement d’une échelle

```
ggplot(diamonds, aes(carat, price)) +

  geom_bin2d()
```

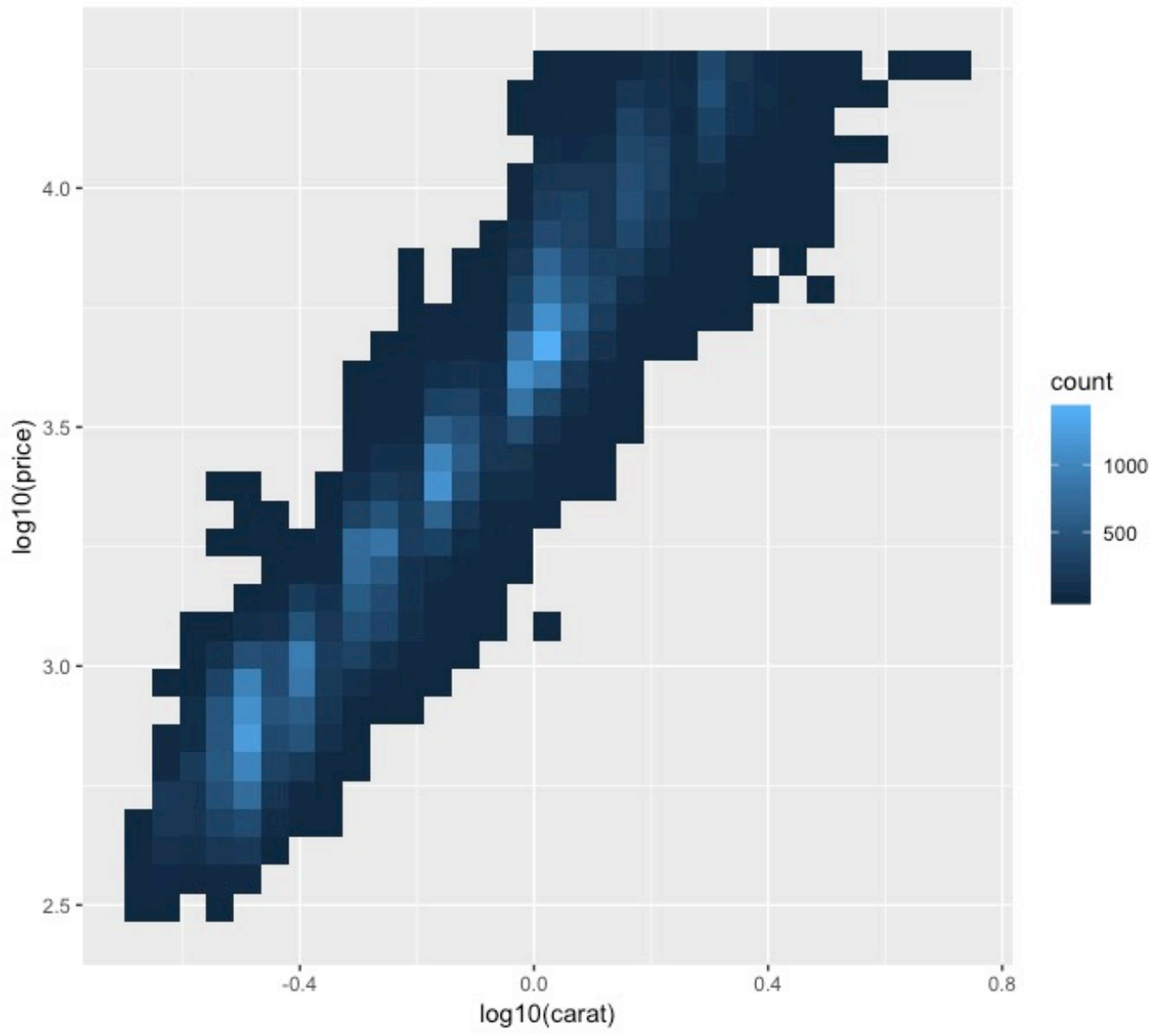
Remplacement d’une échelle



Remplacement d'une échelle

```
ggplot(diamonds, aes(log10(carat),
log10(price))) +

geom_bin2d()
```

Remplacement d’une échelle



Remplacement d’une échelle

- Cela présente toutefois un inconvénient : les légendes des axes affichent maintenant les valeurs transformées, ce qui complique l’interprétation.

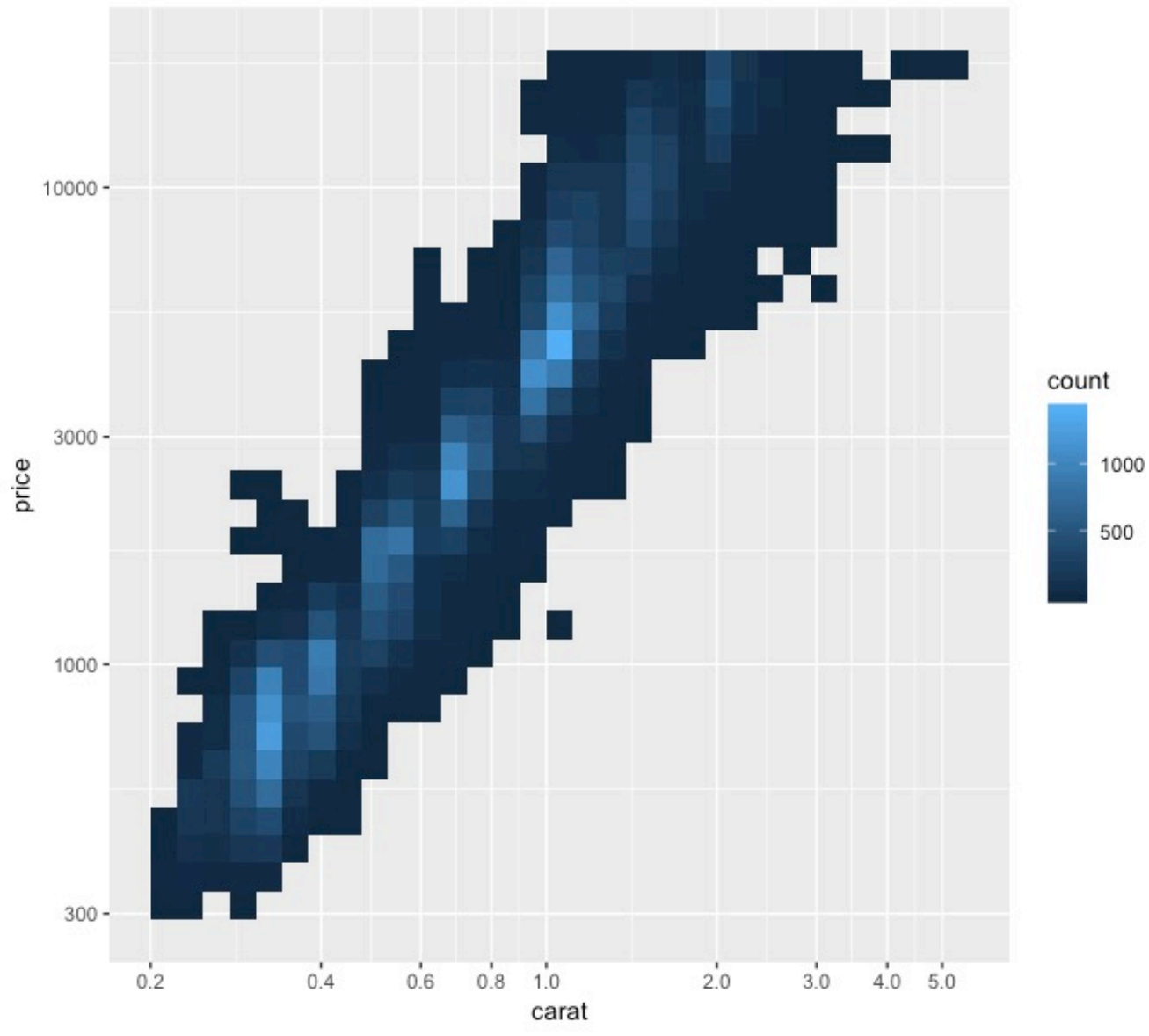
- Pour éviter cela, on peut effectuer la transformation avec l’échelle au lieu de l’effectuer dans la correspondance d’esthétique.

- L’aspect visuel sera identique mais les légendes des axes conserveront l’échelle des données initiales :

Remplacement d'une échelle

```
ggplot(diamonds, aes(carat, price)) +
  geom_bin2d() +
  scale_x_log10(breaks=c(seq(0,1,by=0.2),2:5)) +
  scale_y_log10()
```

Remplacement d'une échelle



Remplacement d’une échelle

- Les échelles de couleurs sont elles aussi fréquemment personnalisées.

- L’échelle par défaut choisit des couleur équitablement réparties sur le cercle chromatique.

- Des alternatives sont disponibles, comme les échelles ColorBrewer, ajustées pour être confortables pour les personnes présentant les types de daltonisme les plus courants.

Remplacement d’une échelle

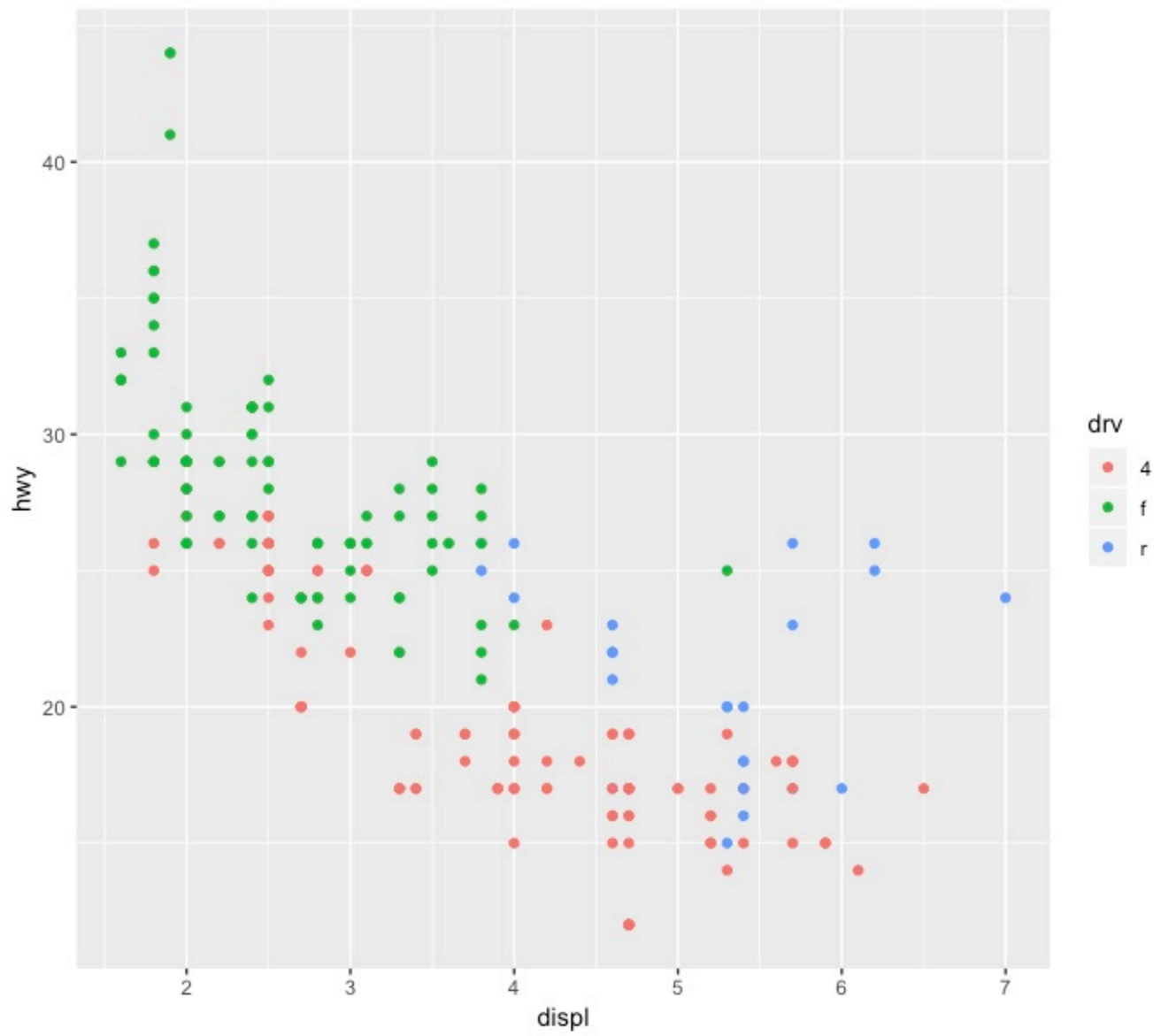
- Les deux graphiques suivants semblent similaires, mais les nuances de rouge et de vert du deuxième graphique sont suffisantes pour que les personnes affectées du daltonisme rouge-vert puissent les différencier :

Remplacement d'une échelle

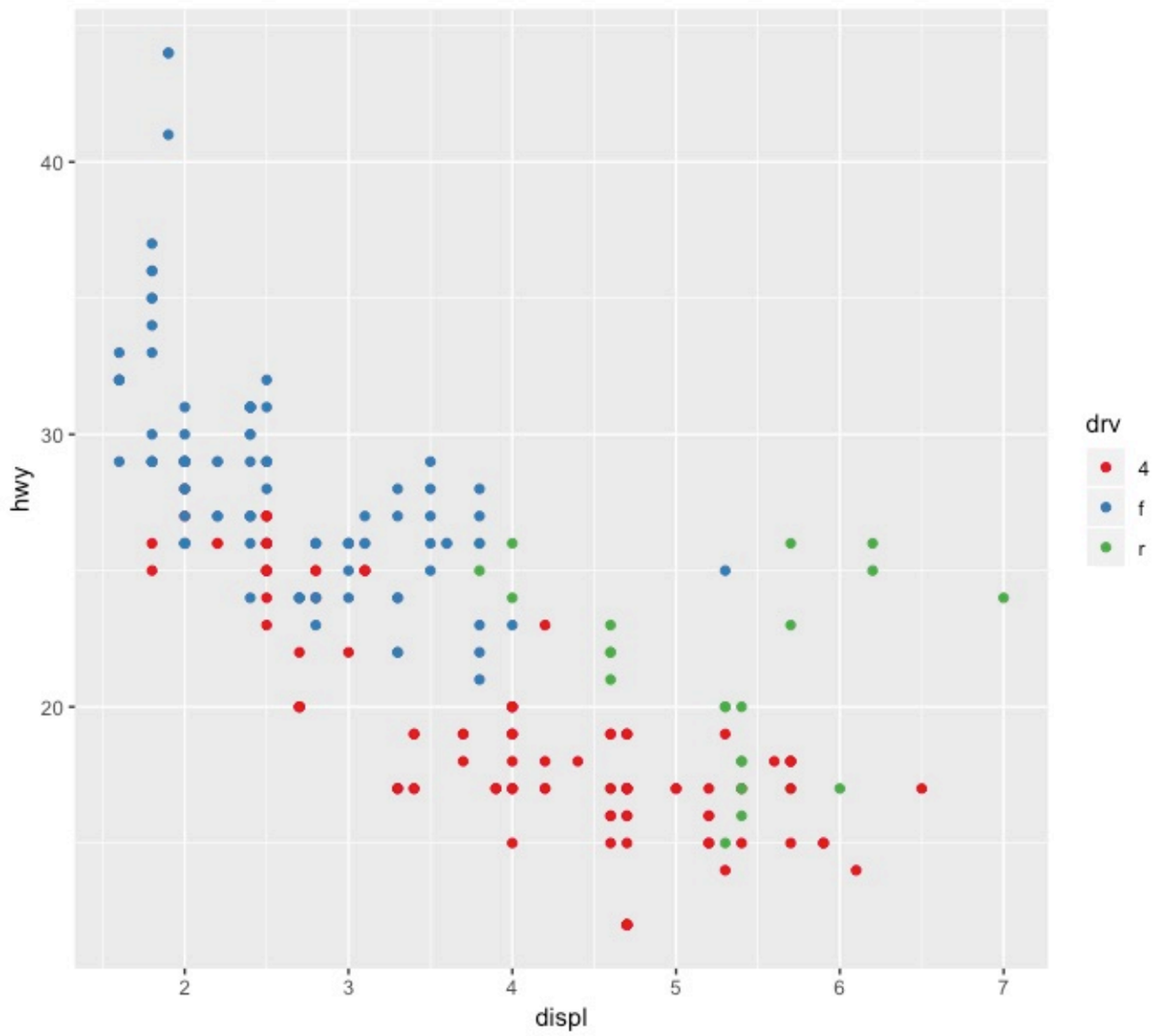
```
ggplot(mpg, aes(displ, hwy)) +

  geom_point(aes(color = drv))
```

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

  geom_point(aes(color = drv)) +

  scale_color_brewer(palette = "Set1")
```

Remplacement d'une échelle



Remplacement d'une échelle



Remplacement d’une échelle

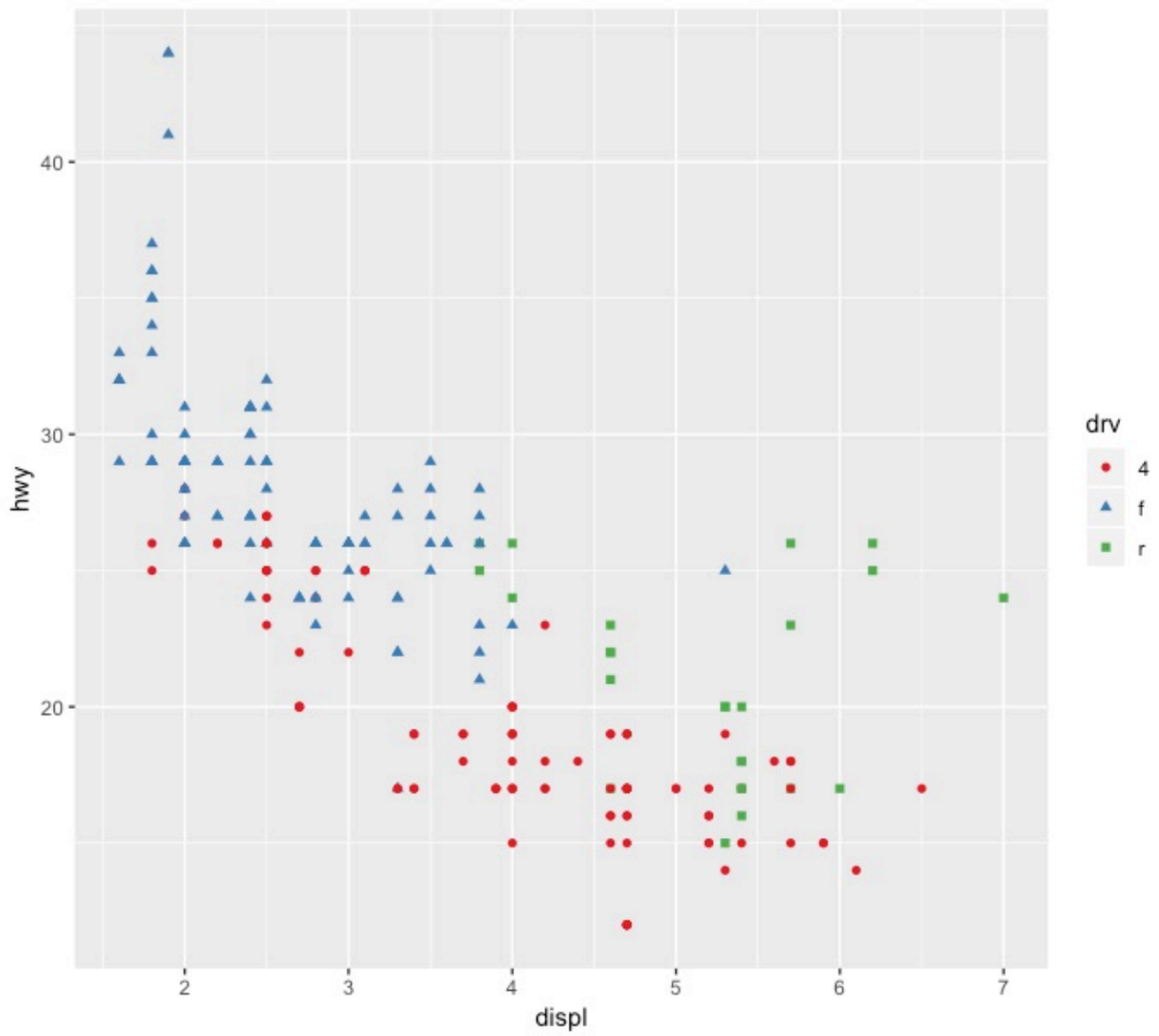
- Notons que des techniques plus simples restent disponibles.

- Si on n’a que quelques couleurs, on peut cumuler la couleur avec une esthétique de forme, qui permet de plus au graphique de rester interprétable en noir et blanc :

Remplacement d'une échelle

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = drv, shape = drv)) +
  scale_color_brewer(palette = "Set1")
```

Remplacement d'une échelle



Remplacement d'une échelle

- Les échelles ColorBrewer sont documentées en ligne sur la page http://colorbrewer2.org et disponibles dans R grâce au package RColorBrewer.

Remplacement d'une échelle

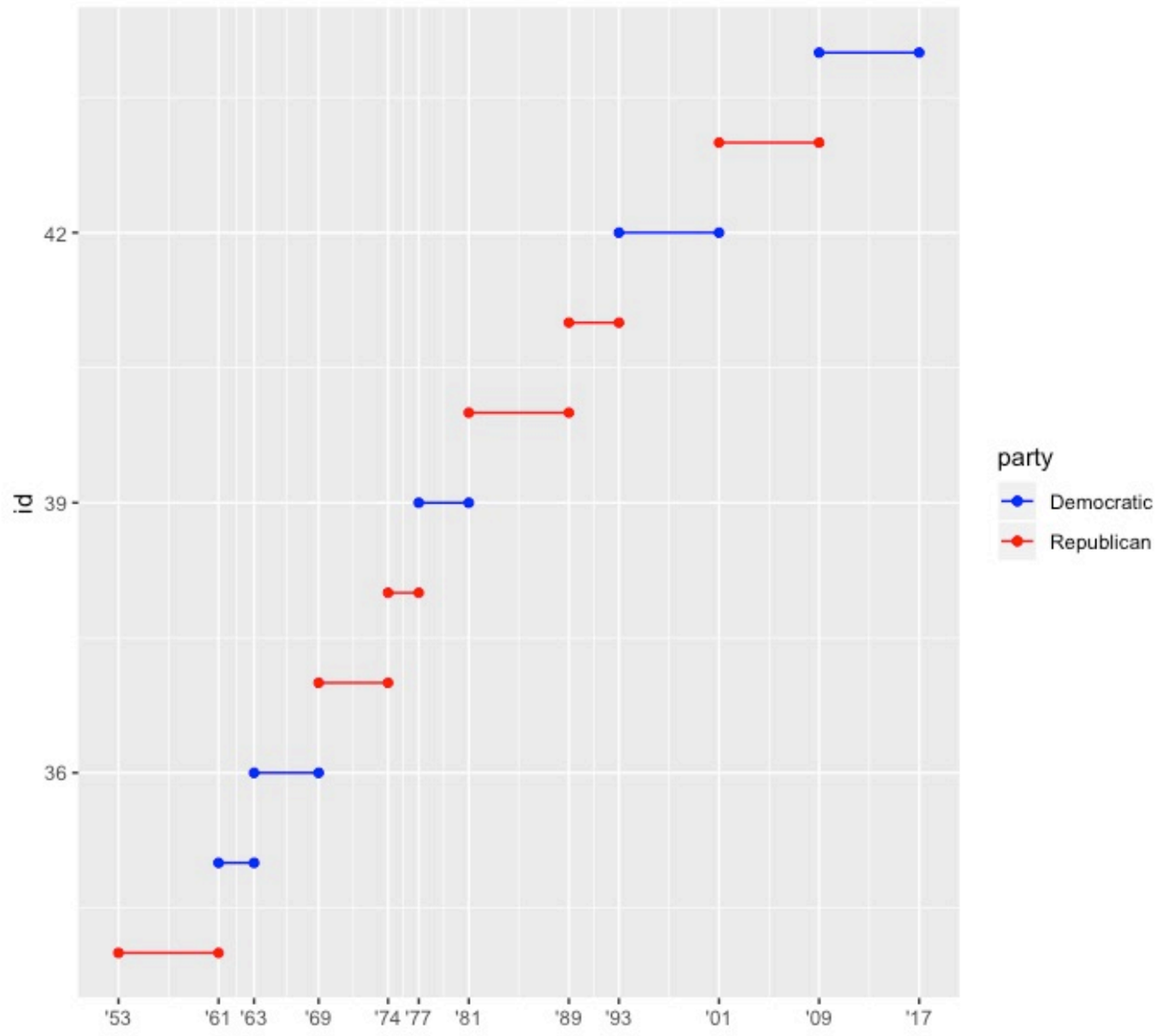
- Lorsqu'on dispose d'une correspondance prédéfinie entre valeurs et couleurs, on peut utiliser `scale_color_manual()`.

- Par exemple, pour les présidents américains, rouge est la couleur des républicains et bleu celle des démocrates :

Remplacement d'une échelle

```
presidential %>%
  mutate(id = 33 + row_number()) %>%
  ggplot(aes(start, id, color = party)) +
  geom_point() +
  geom_point(aes(end, id)) +
  geom_segment(aes(xend = end, yend = id)) +
  scale_x_date(
    NULL,
    breaks = c(presidential$start,max(presidential$end)),
    date_labels = "'%y"
  ) +
  scale_color_manual(
    values = c(Republican = "red", Democratic = "blue")
  )
```

Remplacement d’une échelle



Remplacement d'une échelle

- Pour une couleur continue, on peut utiliser les fonctions `scale_color_gradient()` ou `scale_fill_gradient()`, incluses avec ggplot2.

- `scale_color_gradient2()` permet, quant à elle, de donner des couleurs différentes aux valeurs positives et négatives. Elle peut aussi d'avérer utile pour distinguer les points au-dessus ou en dessous de la moyenne.

Remplacement d'une échelle

- Il existe une autre possibilité : `scale_color_viridis()`, fournie par le package viridis.

- C'est un analogue continu des échelles catégorielles ColorBrewer.

- Exemple :

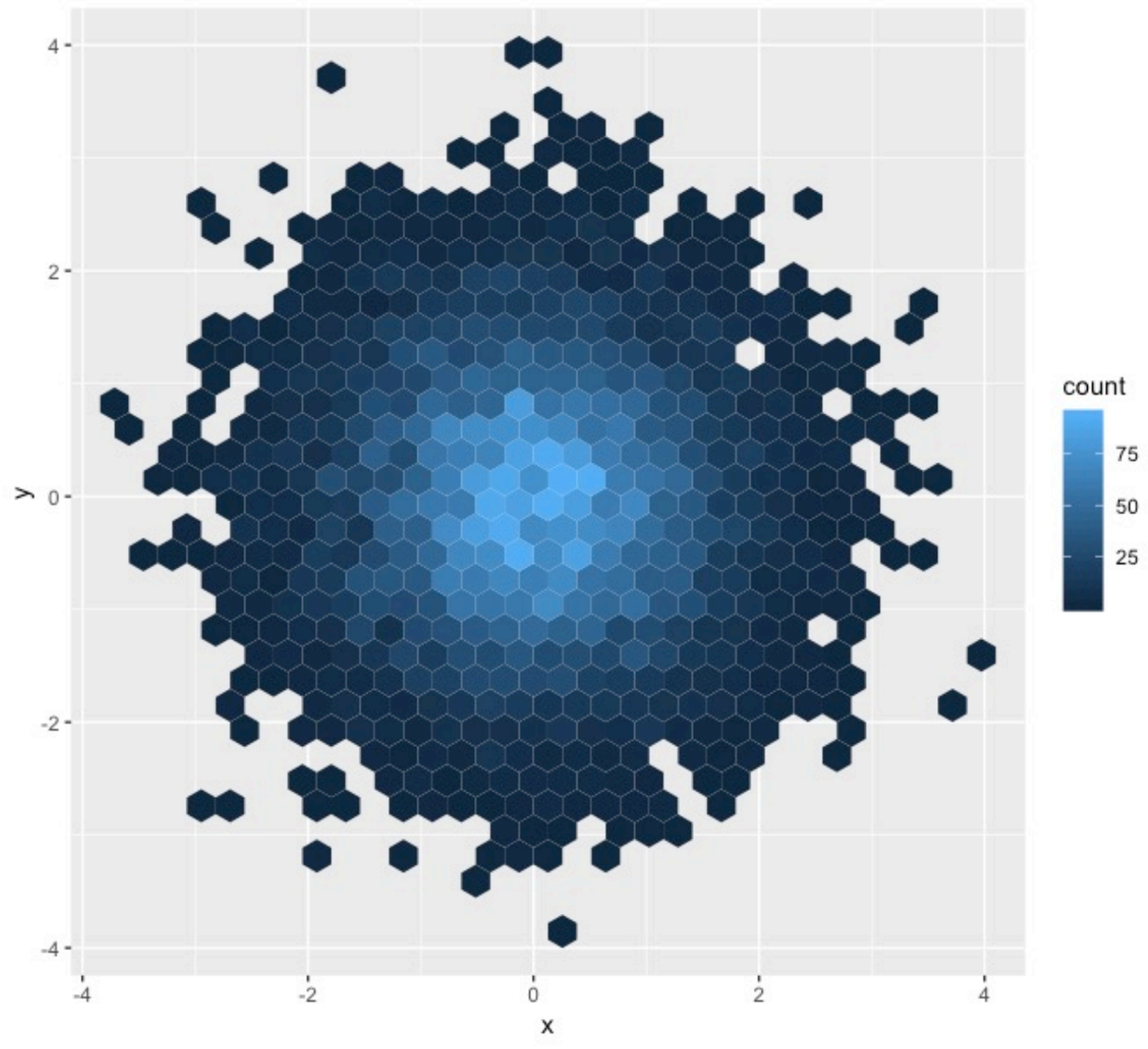
Remplacement d'une échelle

```
df <- tibble(
  x = rnorm(10000),
  y = rnorm(10000)
)

ggplot(df, aes(x, y)) +
  geom_hex() +
  coord_fixed()
```

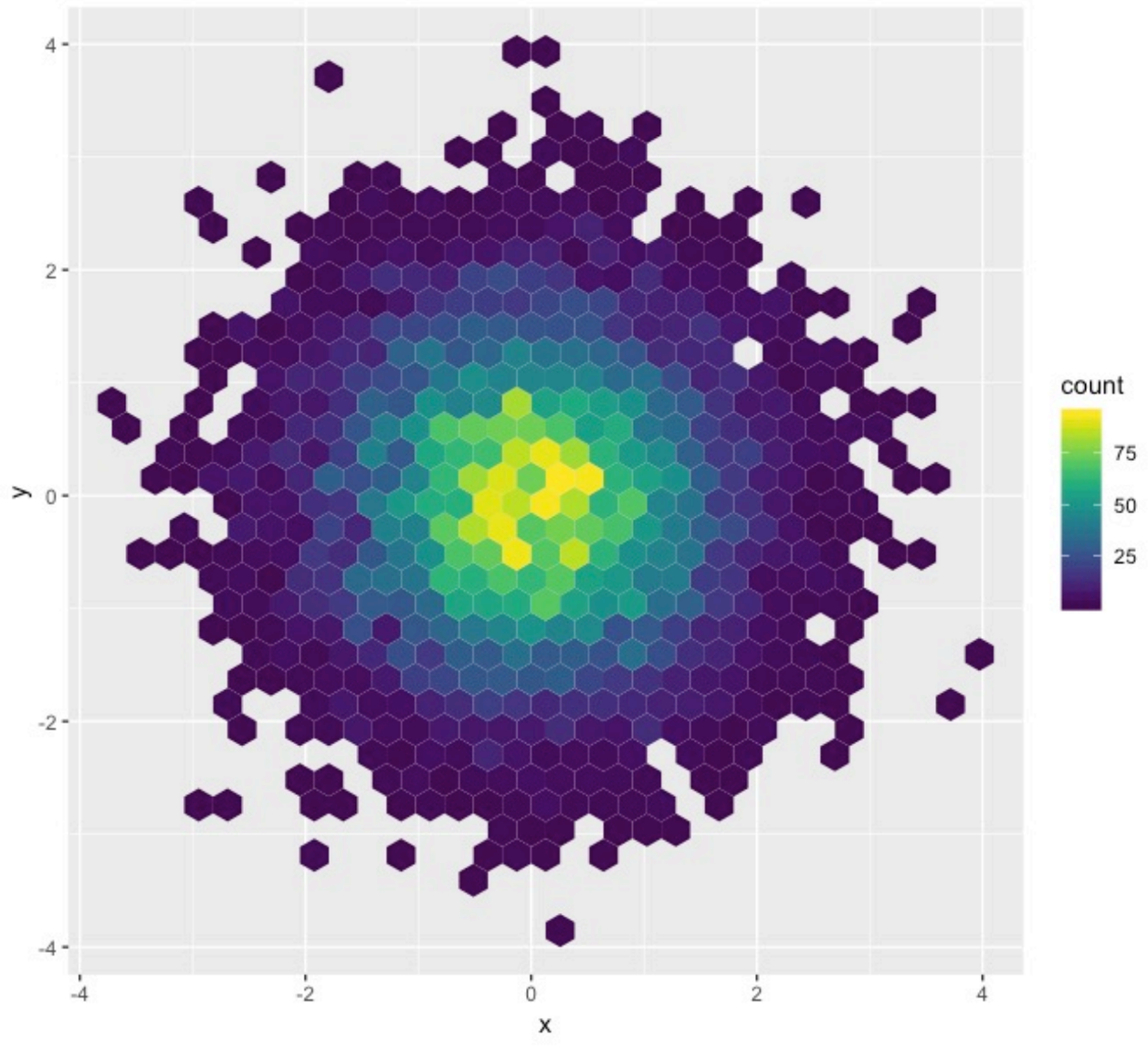
Remplacement d'une échelle

```
ggplot(df, aes(x, y)) +

  geom_hex() +
  viridis::scale_fill_viridis() +
  coord_fixed()
```

Remplacement d’une échelle



Remplacement d’une échelle



Remplacement d'une échelle

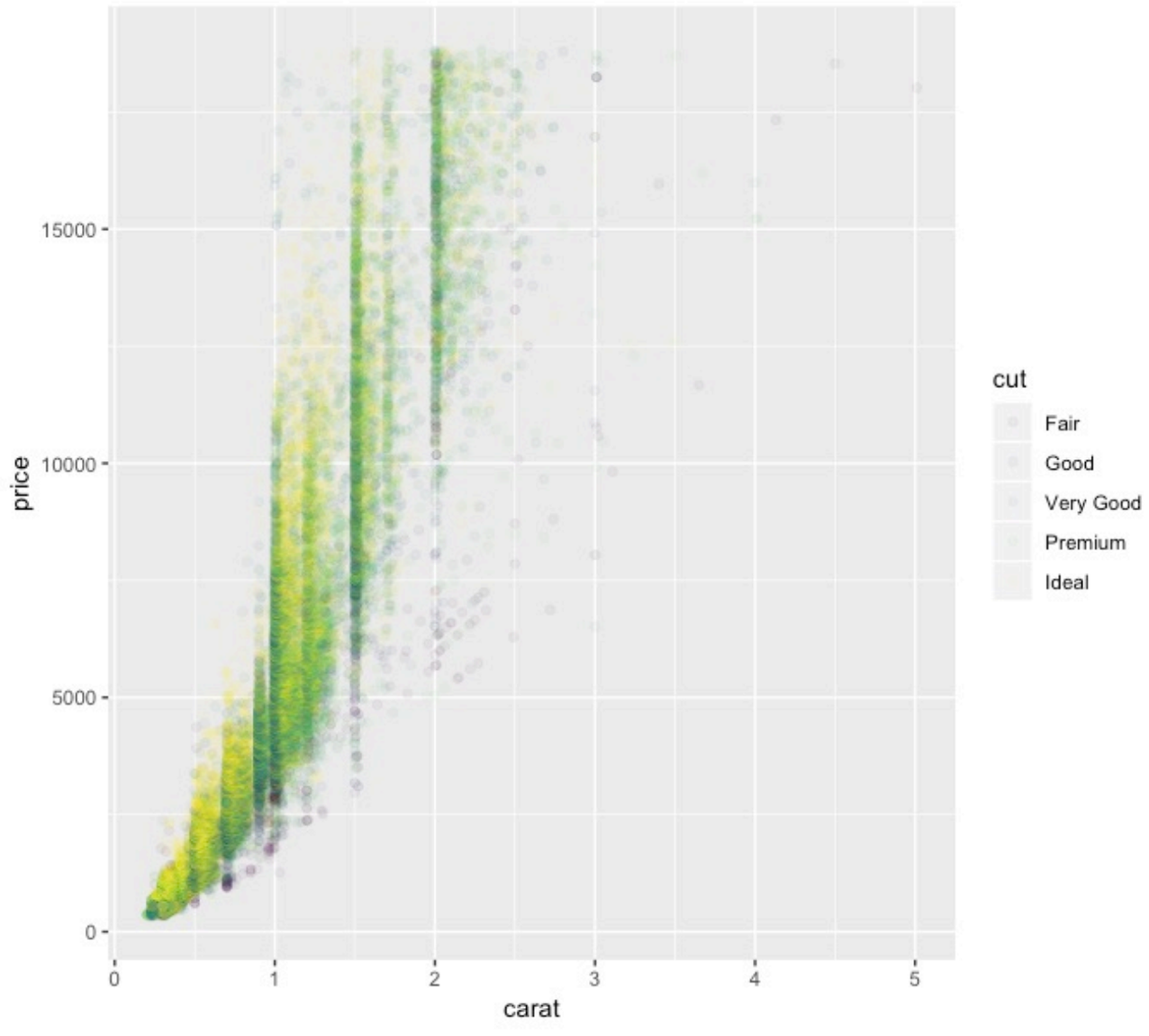
- Notons que toutes les échelles de couleur proposent deux variétés : `scale_color_x()` et `scale_fill_x()`, respectivement pour les esthétiques `color` et `fill`.

Question

- Comment utiliser `override.aes` pour rendre la légende du graphique suivant plus facile à voir ?

```
ggplot(diamonds, aes(carat, price)) +

  geom_point(aes(color = cut), alpha = 1/20)
```

Question


15000
10000
5000
0
price
0
1
2
3
4
5
carat
cut
Fair
Good
Very Good
Premium
Ideal

Zoom

- Les limites d'un graphique peuvent être contrôlées de trois manières :

1. en ajustant les données représentées ;

2. en définissant des limites pour chaque échelle ;

3. en définissant `xlim` et `ylim` avec `coord_cartesian()`.

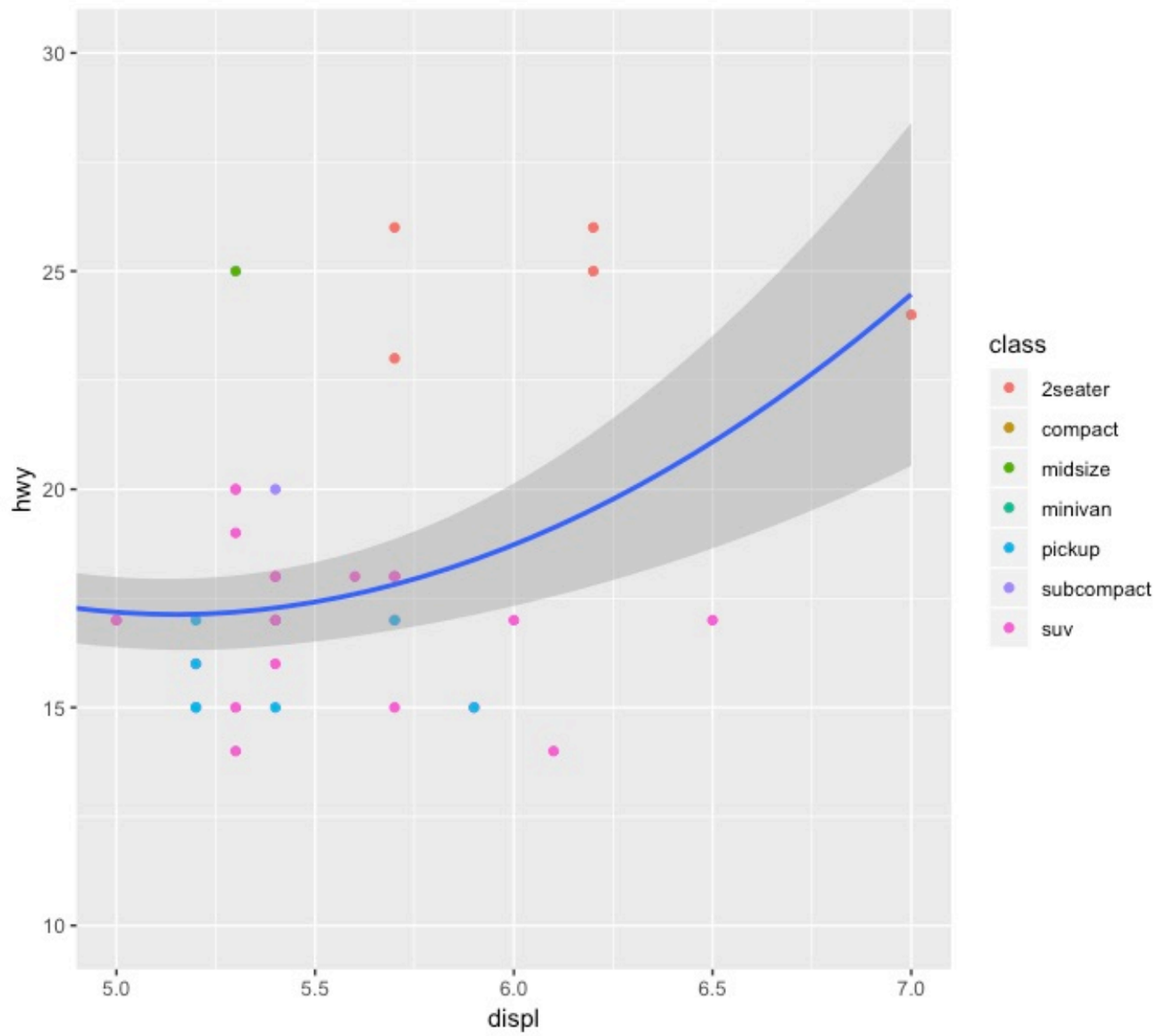
- Pour zoomer sur une région d'un graphique, il est en général préférable d'utiliser la troisième option :

Zoom

- Comparer les deux graphiques suivants :

```
ggplot(mpg, aes(displ, hwy)) +
  geom_point(aes(color = class)) +
  geom_smooth() +
  coord_cartesian(xlim = c(5, 7), ylim = c(10,30))
```

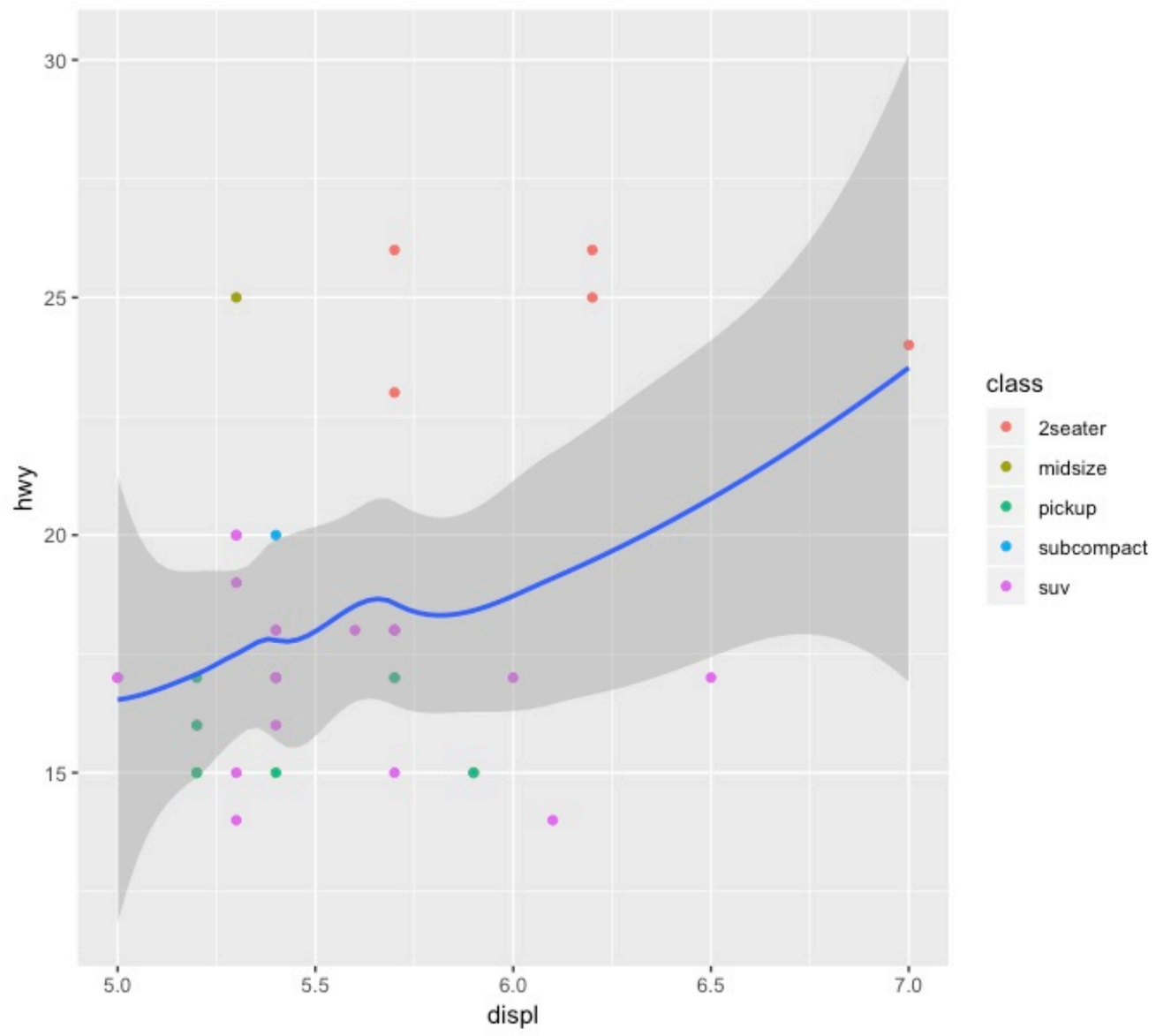
Zoom


30
25
20
15
10
hwy
5.0
5.5
6.0
6.5
7.0
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

Zoom

```
mpg %>%

  filter(displ >= 5, displ <= 7, hwy >= 10, hwy <= 30) %>%

  ggplot(aes(displ, hwy)) +

  geom_point(aes(color = class)) +

  geom_smooth()
```

Zoom


30
25
20
15
hwy
5.0
5.5
6.0
6.5
7.0
displ
class
2seater
midsize
pickup
subcompact
suv

Zoom

- On peut aussi définir des limites sur les échelles individuelles.

- Réduire les limites a en pratique le même effet que filtrer les données.

- En revanche, agrandir les limites est parfois plus utile, notamment pour harmoniser les échelles entre plusieurs graphiques.

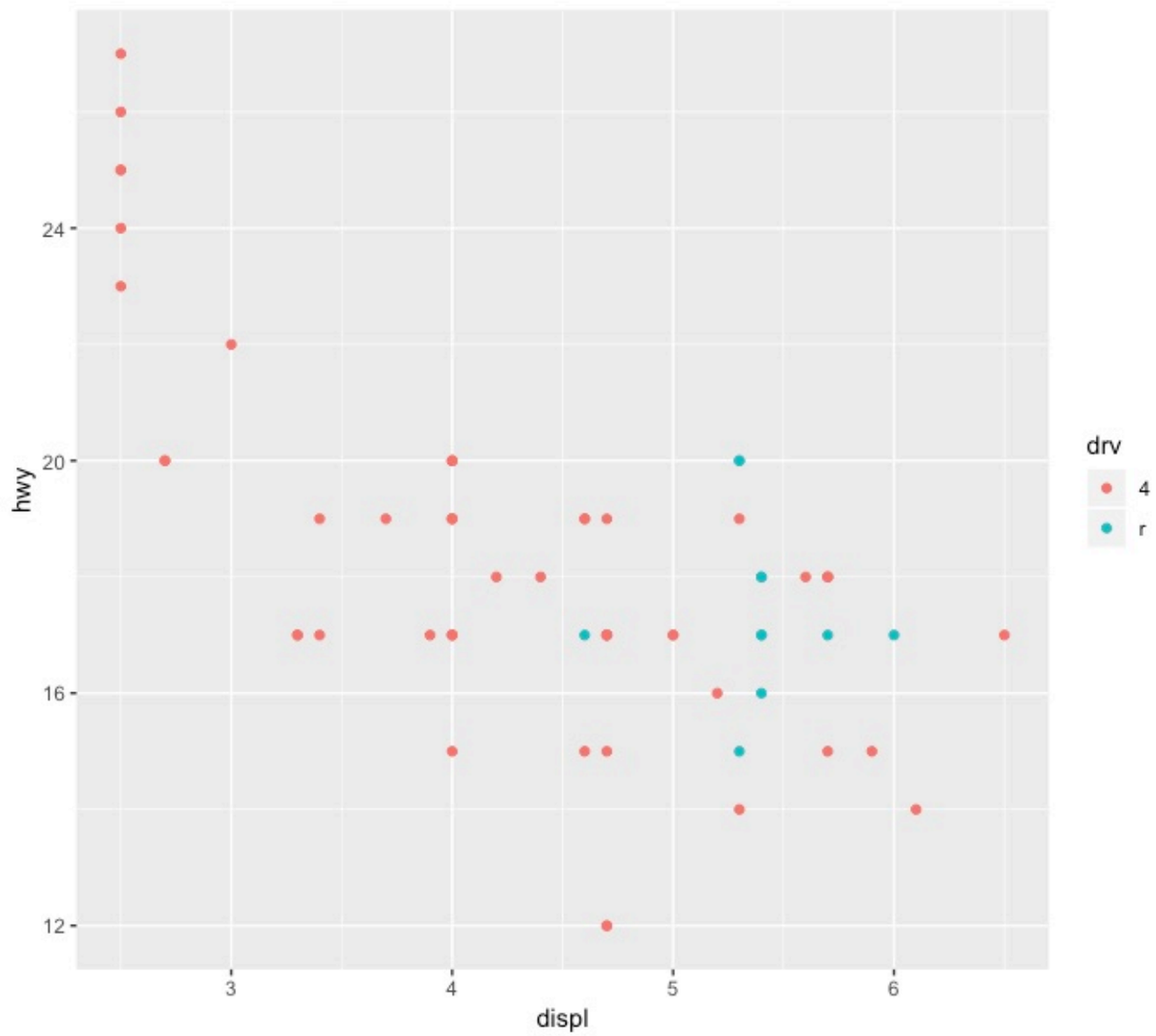
- Si par exemple, on extrait deux catégories de voitures pour les afficher séparément, la comparaison des deux graphiques sera desservie par le fait que les trois échelles (l'axe x, l'axe y, l'esthétique de couleur) ont des plages différentes :

Zoom

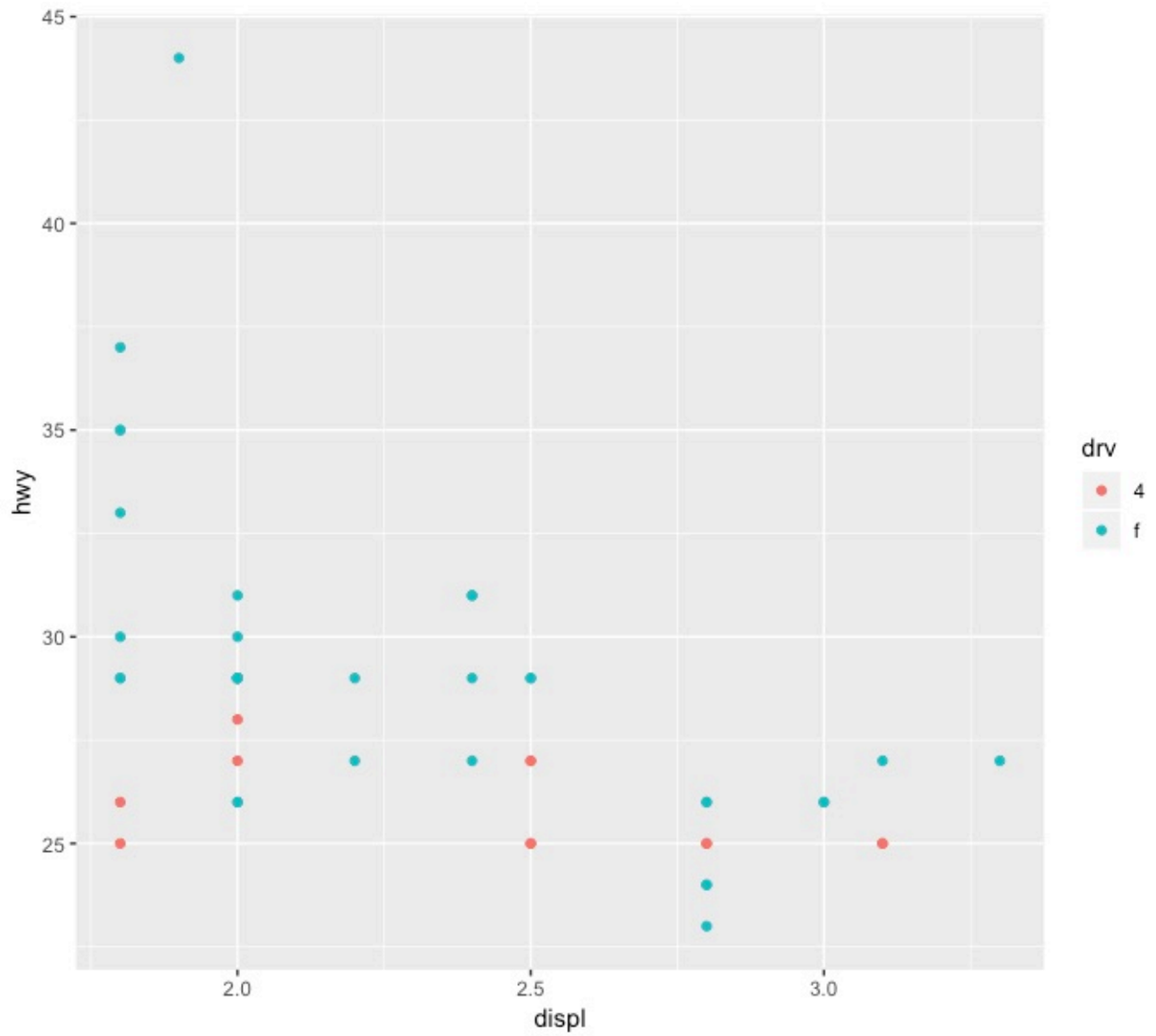
```
suv <- mpg %>% filter(class == "suv")

compact <- mpg %>% filter(class == "compact")
```

```
ggplot(suv, aes(displ, hwy, color = drv)) +
  geom_point()


ggplot(compact, aes(displ, hwy, color = drv)) +
  geom_point()
```

Zoom



Zoom



Zoom

- Pour éviter ce problème, il est possible de partager les échelles entre plusieurs graphiques, en les configurant avec les limites du jeu de données complet :

Zoom

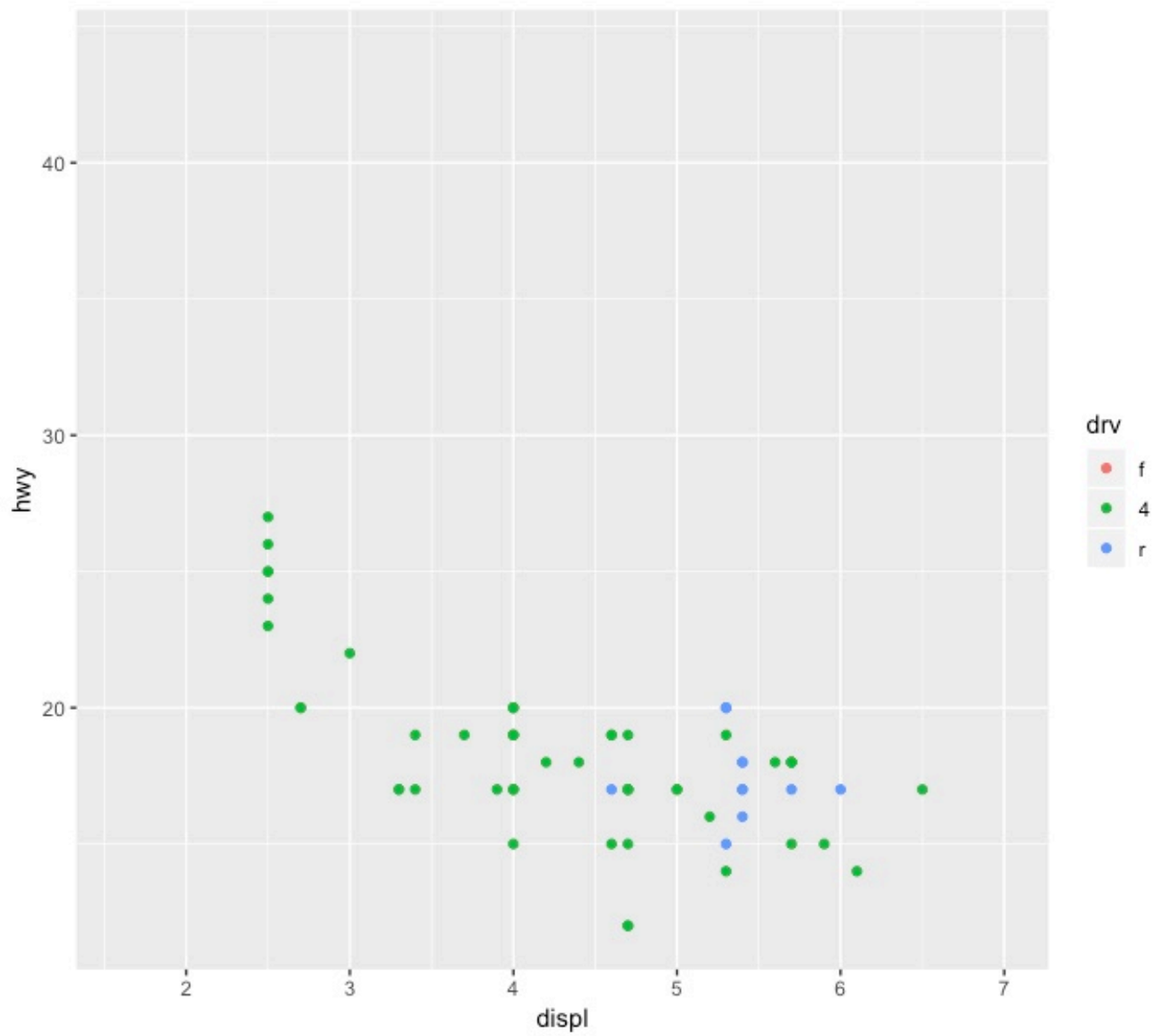
```
x_scale <- scale_x_continuous(limits = range(mpg$displ))

y_scale <-  scale_y_continuous(limits = range(mpg$hwy))

col_scale <- scale_color_discrete(limits = unique(mpg$drv))
```

```
ggplot(suv, aes(displ, hwy, color = drv)) +
  geom_point() +
  x_scale +
  y_scale +
  col_scale
```

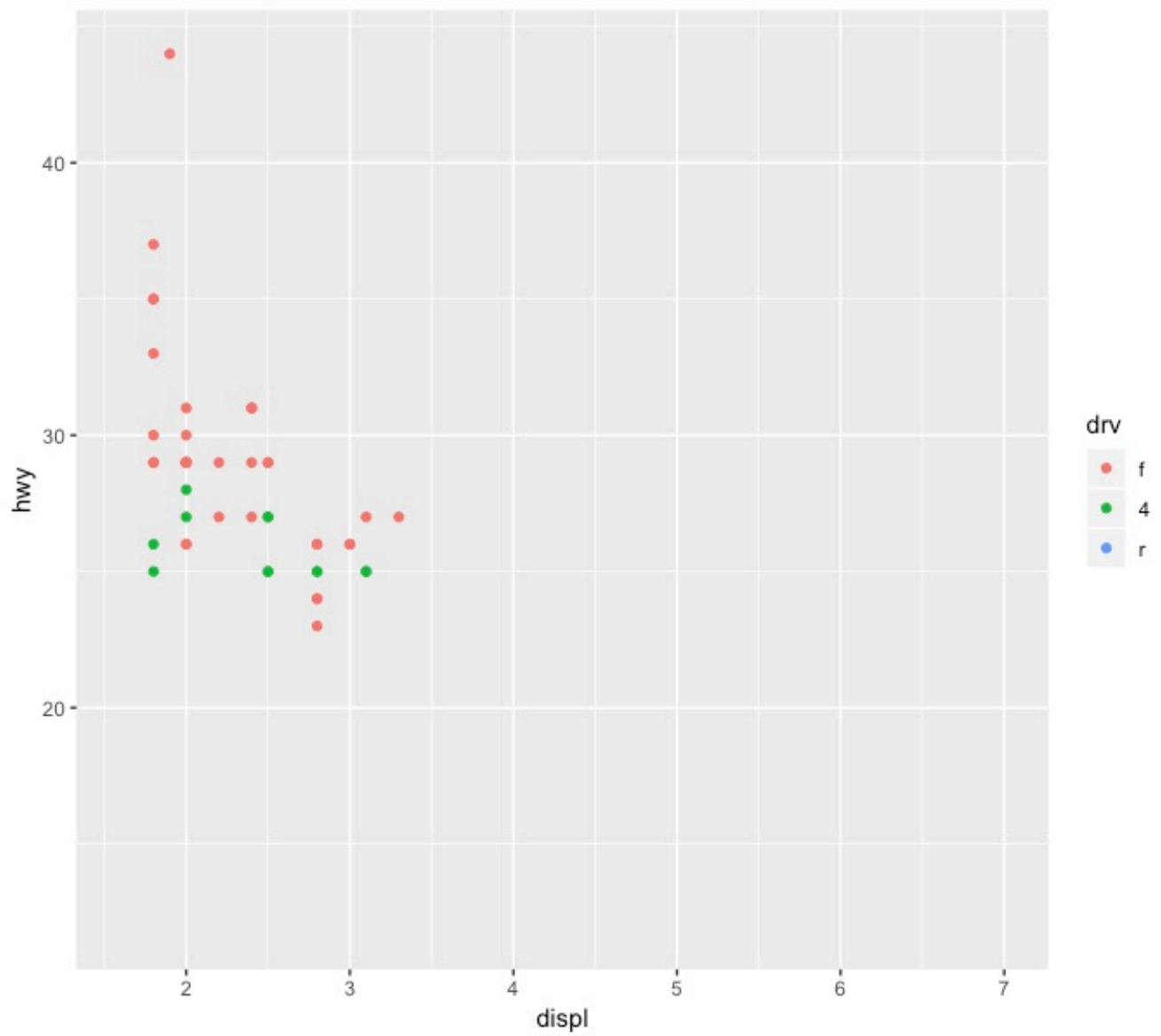
Zoom

```
ggplot(compact, aes(displ, hwy, color = drv)) +

  geom_point() +

  x_scale +

  y_scale +

  col_scale
```

Zoom



Zoom



Zoom

- Notons que, dans cet exemple précis, on aurait pu simplement utiliser un facettage.

- Dans d’autres cas, cette technique peut servir, notamment lorsque les graphiques sont répartis sur plusieurs pages d’un rapport.

Thèmes

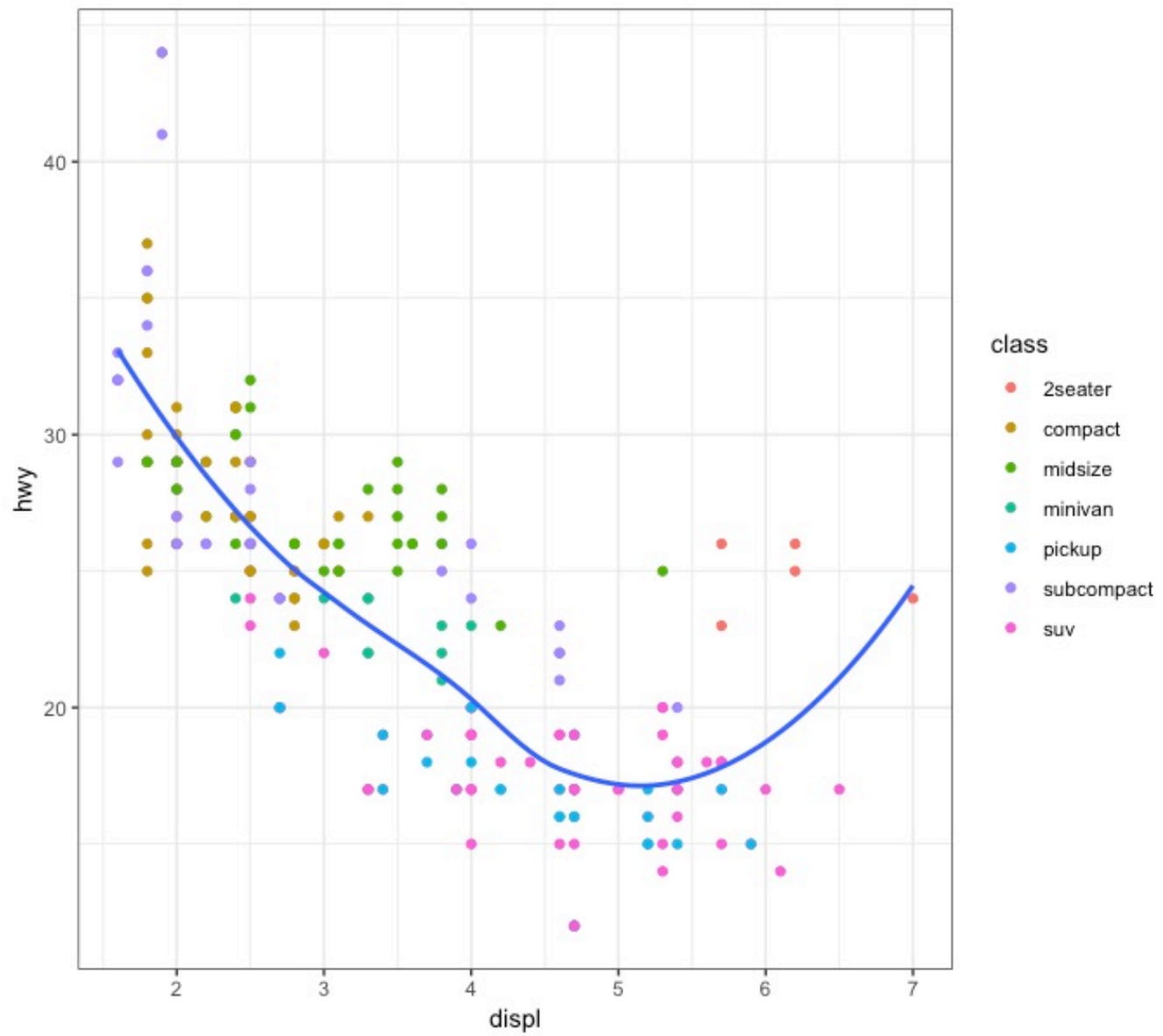
- Les éléments d’un graphique non liés aux données peuvent être personnalisés avec un thème.

- Exemple :

Thèmes

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

geom_point(aes(color = class)) +

geom_smooth(se = FALSE) +

theme_bw()
```

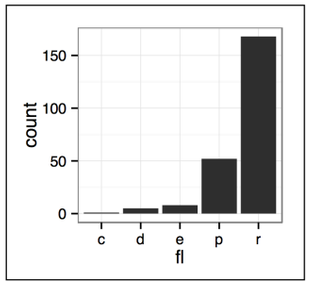
Thèmes


40
30
20
hwy
2
3
4
5
6
7
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv

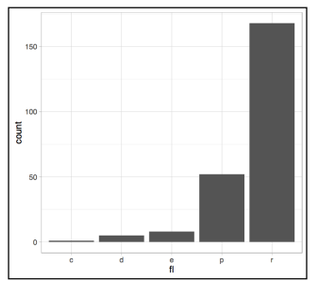
Thèmes

Themes

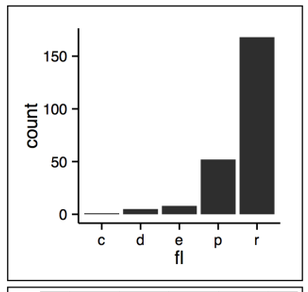
Theme functions change the appearance of your plot.



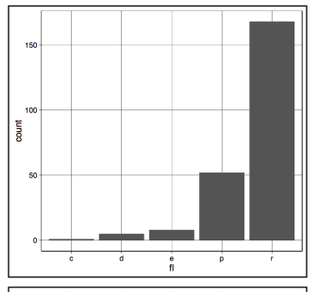
theme_bw()
White background with grid lines



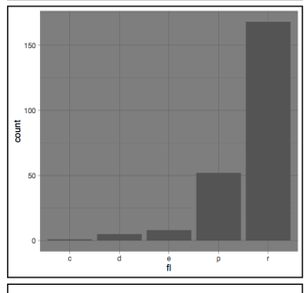
theme_light()
Light axes and grid lines



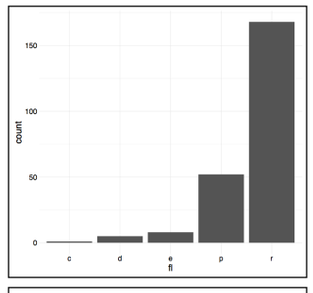
theme_classic()
Classic theme, axes but no grid lines



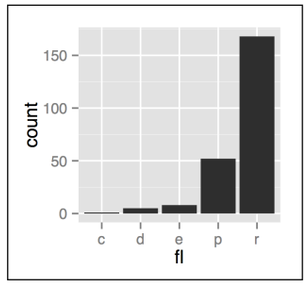
theme_linedraw()
Only black lines



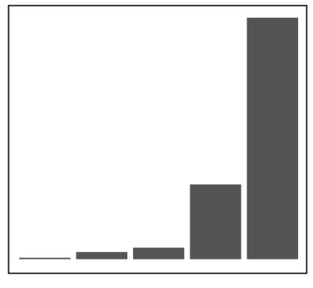
theme_dark()
Dark background for contrast



theme_minimal()
Minimal theme, no background



theme_gray()
Grey background (default theme)



theme_void()
Empty theme, only geoms are visible

Sauvegarde

- Pour sauvegarder un graphique, on peut soit utiliser le menu de la fenêtre graphique de R, soit utiliser la fonction `ggsave()`.

- Consulter l'aide pour pour plus d'informations.

Les tibbles

Introduction

- Dans les différents exemples que l'on a vu avec ggplot2, on a travaillé avec des « tibbles » et non avec les data frames traditionnels de R.

- Les tibbles sont des ***tableaux de données*** mais certains de leur comportements sont modifiés pour faciliter leur utilisation.

- R est un langage relativement ancien et certaines choses qui étaient considérées utiles il y a 10 ans sont maintenant obsolètes.

Introduction

- Pour maintenir la compatibilité avec le code existant, la plupart des innovations sont effectuées dans des packages et non dans le logiciel de base de R.

- La package qu'on va aborder ici est tibble qui fournit des tableaux de données organisés pour fonctionner plus efficacement avec le tidyverse.

Création de tibbles

- Les tibbles étant l'une des fonctionnalités essentielles du tidyverse, presque toutes les fonctions de celui-ci les utilisent automatiquement.

- La plupart des autres packages de R utilisent des data frames standard ; on doit donc parfois les convertir, ce qui peut être fait avec la fonction `as_tibble()`.

Création de tibbles

```
> as_tibble(iris)

# A tibble: 150 x 5

   Sepal.Length Sepal.Width Petal.Length Petal.Width Species

          <dbl>       <dbl>        <dbl>       <dbl> <fct>

 1          5.1         3.5          1.4         0.2 setosa

 2          4.9         3            1.4         0.2 setosa

 3          4.7         3.2          1.3         0.2 setosa

 4          4.6         3.1          1.5         0.2 setosa

 5          5           3.6          1.4         0.2 setosa

 6          5.4         3.9          1.7         0.4 setosa

 7          4.6         3.4          1.4         0.3 setosa

 8          5           3.4          1.5         0.2 setosa

 9          4.4         2.9          1.4         0.2 setosa

10          4.9         3.1          1.5         0.1 setosa

# ... with 140 more rows
```

Création de tibbles

- On peut aussi créer un nouveau tibble à partir de vecteurs individuels, avec la fonction `tibble()`.

- Celle-ci ***recycle*** automatiquement les entrées de taille 1 et autorise les références à des variables créées par calcul.

- Exemple :

Création de tibbles

```
tibble(
   x = 1:5,
   y = 1,
   z = x^2 + y
 )
```

```
# A tibble: 5 x 3
      x     y     z
  <int> <dbl> <dbl>
1     1     1     2
2     2     1     5
3     3     1    10
4     4     1    17
5     5     1    26
```

Création de tibbles

- Contrairement à `data.frame()`, `tibble()` ne change jamais le type des entrées (il ne convertit donc pas les chaînes en facteurs), ne change jamais le nom des variables et ne crée jamais de nom de ligne.

- Un tibble peut avoir des noms de colonnes qui ***ne sont pas*** des noms de variables valides pour R, ce qu'on appelle des noms ***non syntactiques***.

Création de tibbles

- Ils peuvent, par exemple, commencer par autre chose qu'une lettre, ou contenir des caractères particuliers, comme des espaces.

- Les références à ces variables doivent utiliser les guillemets arrières, c'est-à-dire des accent graves.

- Exemple :

Création de tibbles

```
tb <- tibble(
  `:)` = "smile",
  ` ` = "space",
  `2000` = "number"
)
> tb
# A tibble: 1 x 3
  `:)`  ` `   `2000`
  <chr> <chr> <chr>
1 smile space number
```

Création de tibbles

- On doit aussi utiliser ces guillemets lorsqu'on utilise ces variables dans d'autres packages, comme ggplot2 ou dplyr.

- Les tibbles peuvent aussi être créés avec `tribble()`, qui signifie « tibble transposé » (*transposed tibble*).

- `tribble()` est conçu pour saisir les données dans le code : les titres des colonnes sont définis par des formules (ils commencent par `~`) et les entrées sont séparées par des virgules.

- Exemple :

Création de tibbles

```
tribble(
  ~x, ~y, ~z,
  #--|--|----
  "a", 2, 3.6,
  "b", 1, 8.5
)
# A tibble: 2 x 3
  x         y     z
  <chr> <dbl> <dbl>
1 a         2   3.6
2 b         1   8.5
```

tibble vs. data.frame

- L'utilisation des ***tibbles*** présente principalement deux différences par rapport aux tableaux de données classiques (***data frames***) : l'**affichage** et l'**indiçage**.

- Les tibbles ont une méthode d'affichage spécifique, qui affiche uniquement les dix premières lignes et seulement autant de colonnes que l'écran peut contenir.

- Cela facilite la manipulation de données volumineuses.

tibble vs. data.frame

- En plus de son nom, chaque colonne indique son type, une fonctionnalité pratique venant de `str()`.

- Exemple :

tibble vs. data.frame

```
tibble(

  a = lubridate::now() + runif(1e3) * 86400,

  b = lubridate::today() + runif(1e3) * 30,

  c = 1:1e3,

  d = runif(1e3),

  e = sample(letters, 1e3, replace = TRUE)

)
```

tibble vs. data.frame

```
# A tibble: 1,000 x 5
   a                   b              c      d e
   <dttm>              <date>     <int>  <dbl> <chr>
 1 2018-12-11 11:56:23 2018-12-21     1 0.975  i
 2 2018-12-11 07:13:11 2018-12-29     2 0.651  x
 3 2018-12-11 03:31:34 2019-01-02     3 0.620  r
 4 2018-12-10 22:15:35 2018-12-14     4 0.574  j
 5 2018-12-11 08:22:42 2019-01-03     5 0.371  n
 6 2018-12-10 19:13:56 2019-01-02     6 0.0158 q
 7 2018-12-11 12:01:58 2018-12-15     7 0.419  r
 8 2018-12-11 10:56:41 2019-01-02     8 0.760  m
 9 2018-12-10 19:48:37 2019-01-01     9 0.183  f
10 2018-12-11 08:35:51 2018-12-16    10 0.888  k
# ... with 990 more rows
```

tibble vs. data.frame

- Les types de variables les plus fréquents sont :
- **int** (*integer*) correspond à des entiers
- **dbl** (*double*) correspond à des nombres réels
- **chr** (*character*) correspond à des chaînes de caractères
- **ddtm** (*date+time*) correspond à des données « date-heure »

tibble vs. data.frame

- Il en existe trois autres, qu'on rencontre moins fréquemment :

- **lgl** (*logical*) correspond à des données booléennes

- **fctr** (*factor*) correspond à un facteur donc à des données catégorielles dont les modalités sont fixées

- **date** (*date*) correspond à des dates.

tibble vs. data.frame

- L’affichage par défaut est conçu pour qu’on ne risque pas de submerger accidentellement la console en affichant un tableau de données, mais il n’est pas toujours suffisant.

- Certaines options peuvent alors s’avérer utiles.

- Tout d’abord, lorsqu’on demande explicitement un affichage avec la fonction `print()`, on peut contrôler le nombre de lignes (`n`) et le nombre de colonnes (`width`).

tibble vs. data.frame

- Spécifier `width = Inf` permet d'afficher toutes les colonnes.

- Exemple :

```
nycflights13::flights %>%

  print(n=3, width = Inf)
```

tibble vs. data.frame

```
# A tibble: 336,776 x 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
  sched_arr_time arr_delay carrier flight tailnum origin dest  air_time
           <int>     <dbl> <chr>    <int> <chr>   <chr>  <chr>    <dbl>
1            819        11 UA        1545 N14228  EWR    IAH        227
2            830        20 UA        1714 N24211  LGA    IAH        227
3            850        33 AA        1141 N619AA  JFK    MIA        160
  distance  hour minute time_hour
     <dbl> <dbl>  <dbl> <dttm>
1     1400     5     15 2013-01-01 05:00:00
2     1416     5     29 2013-01-01 05:00:00
3     1089     5     40 2013-01-01 05:00:00
# ... with 3.368e+05 more rows
```

tibble vs. data.frame

- On peut aussi contrôler le comportement d'affichage par défaut en définissant des options.

- `options(tibble.print_max = n, tibble.print_min =n)` : s'il y a plus de *m* lignes, n'afficher que *n* lignes.

- `options(dplyr.print_min = Inf)` permet d'afficher toujours toutes les lignes.

- `options(tibble.width = Inf)` permet d'afficher toujours toutes les colonnes.

tibble vs. data.frame

- La liste complète des options est disponible dans la documentation du package, accessible en faisant `package?tibble`.

- Enfin, on peut utiliser le visionneur intégré à Studio pour obtenir une fenêtre défilante contenant le jeu de données complet.

- Exemple :

```
nycflights13::flights %>%

  View()
```

Indiçage

- Lorsqu'on souhaite sélectionner une variable spécifique, on doit utiliser `$` ou `[[ ]]`.

- `[[ ]]` permet d'extraire par nom ou par position, `$` n'extrait que par nom, mais il est plus rapide à saisir.

- Exemple :

Indiçage

```
df <- tibble(
  x = runif(5),
  y = rnorm(5)
)
```

```
# Extraction par nom
df$x
df[["x"]]
```

```
# Extraction par position
df[[1]]
```

Indiçage

- Pour utiliser les tibbles dans un canal (on verra cette notion dans la suite), on doit utiliser le caractère de substitution :

```
df %>% .$x
```

```
df %>% .["x"]
```

Interaction avec du code plus ancien

- Certaines anciennes fonctions de R ne sont pas compatibles avec les tibbles.

- Si l'on rencontre une telle fonction, on peut utiliser `as.data.frame()` pour convertir le tibble en data.frame classique.

Questions

- Comment savoir si un objet est un tibble ?
- Si on a le nom d'une variable enregistré dans un objet (par exemple, `var <- « mpg »`), comment peut-on extraire depuis un tibble la variable référencée ?
- Que fait la fonction `tibble::enframe()` ? A quoi peut-elle servir ?

Transformation de données avec dplyr

Introduction

- La visualisation est un outil important pour l'exploration des données.

- Mais il est rare que ces dernières soient fournies sous une forme directement exploitable.

- On doit souvent créer de nouvelles variables et résumés ou renommer les variables et réorganiser les observations de façon à pouvoir les utiliser plus facilement.

- Nous allons voir comment faire cela avec le package dplyr.

Introduction

- On illustrera les idées principales avec les données du package `nycflights13`, qu'on pourra explorer à l'aide de ggplot2 pour mieux les comprendre :

```
library(nycflights13)

library(tidyverse)
```

Introduction

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1 ──
✔ ggplot2 3.1.0     ✔ purrr   0.2.5
✔ tibble  1.4.2     ✔ dplyr   0.7.8
✔ tidyr   0.8.2     ✔ stringr 1.3.1
✔ readr   1.2.1     ✔ forcats 0.3.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
✖ .GlobalEnv::alpha() masks ggplot2::alpha()
✖ dplyr::filter()     masks stats::filter()
✖ dplyr::lag()        masks stats::lag()
```

Introduction

- Lorsqu'on charge le tidyverse, des conflits avec des fonctions de base de R apparaissent.

- Si on souhaite utiliser quand même celles-ci après avoir chargé dplyr, on doit utiliser leur nom complet : par exemple `stats::filter()` et `stats::lag()`.

nycflights13

- Afin d'explorer les principales fonctionnalités de manipulation de données de dplyr, on va utiliser `nycflights13::flights`.

- Ce tableau de données contient des informations sur les 336 776 vols ayant décollé de New York en 2013.

- Ces données sont documentées dans `?flights`.

nycflights13

```
> print(flights, n=3, width=Inf)

# A tibble: 336,776 x 19

   year month   day dep_time sched_dep_time dep_delay arr_time

  <int> <int> <int>    <int>          <int>     <dbl>    <int>

1  2013     1     1      517            515         2      830

2  2013     1     1      533            529         4      850

3  2013     1     1      542            540         2      923

  sched_arr_time arr_delay carrier flight tailnum origin dest  air_time

           <int>     <dbl> <chr>    <int> <chr>   <chr>  <chr>    <dbl>

1            819        11 UA        1545 N14228  EWR    IAH        227

2            830        20 UA        1714 N24211  LGA    IAH        227

3            850        33 AA        1141 N619AA  JFK    MIA        160

  distance  hour minute time_hour

     <dbl> <dbl>  <dbl> <dttm>

1     1400     5     15 2013-01-01 05:00:00

2     1416     5     29 2013-01-01 05:00:00

3     1089     5     40 2013-01-01 05:00:00

# ... with 3.368e+05 more rows
```

Bases de dplyr

- Nous allons voir dans la suite les ***cinq fonctions élémentaires*** de dplyr, qui permettent de répondre à la majorité des besoins en manipulation de données :

- filter() : pour sélectionner des observations selon leurs valeurs

- arrange() : pour réordonner les lignes

- select() : pour sélectionner des variables par leur nom

Bases de dplyr

- mutate() : pour créer de nouvelles variables à partir de variables existantes

- summarize() : pour regrouper plusieurs valeurs en un résumé unique

- Toute ces fonctions peuvent être utilisées en conjonction avec group_by(), qui modifie la portée de chaque fonction de façon à ce qu'elle n'opère que sur un groupe et pas sur la totalité des données.

Bases de dplyr

- Ces six fonctions constituent les ***verbes*** d'un langage de manipulation des données.

- Chaque verbe fonctionne selon le même schéma :

1. le premier argument est un tableau de données

2. les arguments suivants décrivent les actions à effectuer sur le tableau, au moyen de noms de variables (sans guillemets)

3. le résultat est un nouveau tableau de données

Bases de dplyr

- Plusieurs étapes peuvent être combinées facilement pour atteindre un résultat complexe.

- On va voir chacun des verbes à l'oeuvre dans ce qui suit.

filter()

- filter() permet de créer des sous-ensembles d'observations en fonction de leurs valeurs.

- Son premier argument est le nom du tableau de données, les suivants sont les expressions qui vont filtrer le tableau.

- On peut par exemple sélectionner les vols du premier janvier en faisant :

filter()

```
> filter(flights, month == 1, day == 1)
```

```
# A tibble: 842 x 19
```

	year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
	<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
1	2013	1	1	517	515	2	830
2	2013	1	1	533	529	4	850
3	2013	1	1	542	540	2	923
4	2013	1	1	544	545	-1	1004
5	2013	1	1	554	600	-6	812

```
# ... with 832 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

filter()

- Lorsqu'on exécute la commande, dplyr réalise l'opération de filtrage et renvoie un nouveau tableau de données.

- ***Les fonctions de dplyr ne modifient jamais leur entrées.***

- Par conséquent, si on veut sauvegarder le résultat, on doit utiliser l'opérateur d'affectation :

```
jan1 <- filter(flights, month == 1, day == 1)
```

filter()

- Si on veut à la fois afficher les résultats et les sauvegarder dans une variable, il suffit de mettre la commande d'affectation entre parenthèses :

```
(jan1 <- filter(flights, month == 1, day == 1))
```

Comparaisons

- Pour utiliser efficacement le filtrage, on utilise les opérateurs de comparaison standards : >, >=, <, <=, != et ==.

- ***L'utilisation de == peut s'avérer problématique pour les nombres flottants.***

- Exemple :

Comparaisons

```
> sqrt(2)^2 == 2

[1] FALSE

> 1/49 * 49 == 1

[1] FALSE
```

Comparaisons

- Les ordinateurs utilisent une arithmétique à précision finie. Par conséquent, tous les nombres qu'ils manipulent sont des approximations.

- A la place de ==, on peut utiliser `near()` :

Comparaisons

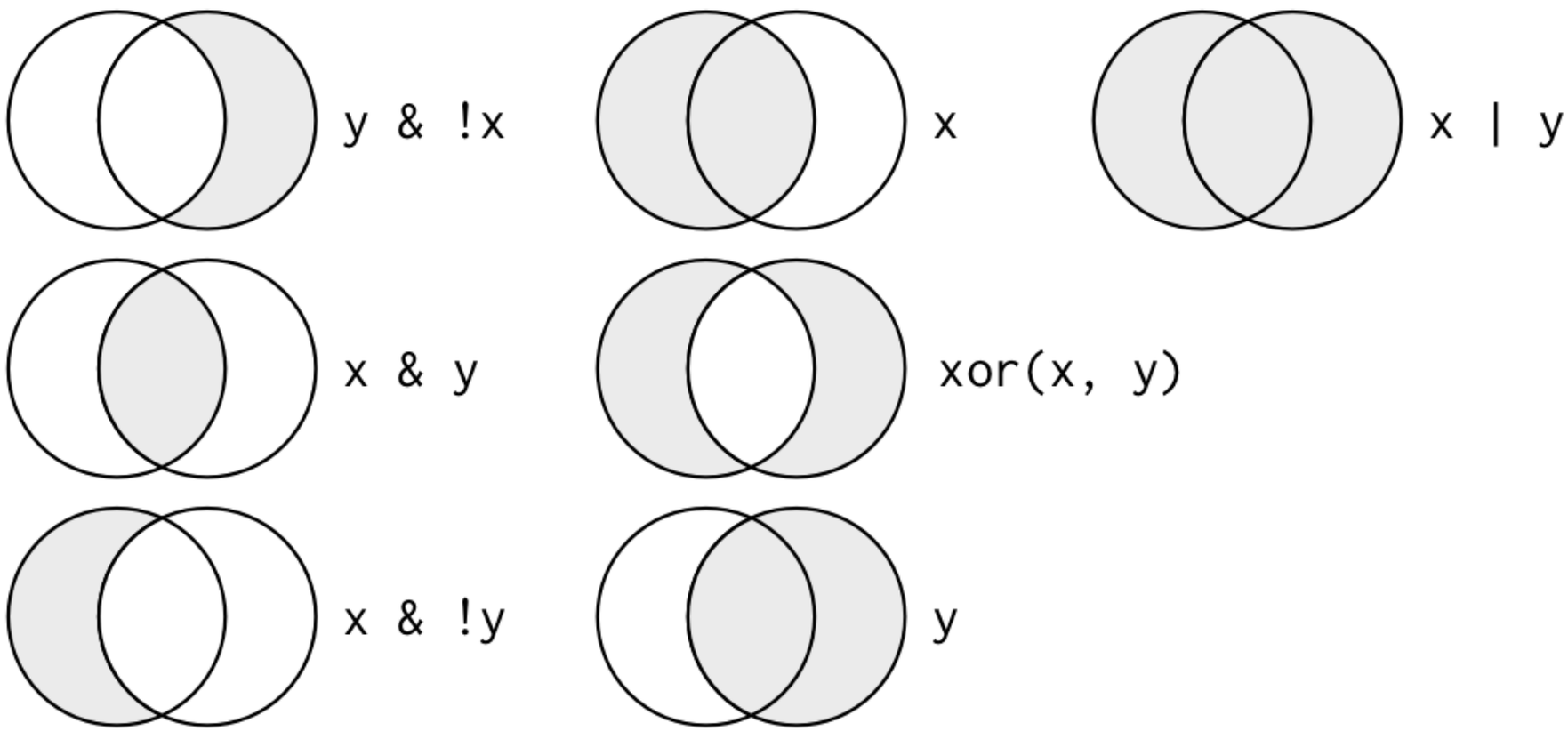
```
> near(sqrt(2)^2,2)

 [1] TRUE

> near(1/49 * 49, 1)

 [1] TRUE
```

Opérateurs logiques

- Il est possible de combiner plusieurs arguments de filter() avec « and ». Pour les autres types de comparaisons, on utilise directement les opérateurs booléens : &, | et !.

- La figure suivante affiche l'ensemble complet des opérations booléennes :

Opérateurs logiques



Opérateurs logiques

- Par exemple, le code suivant trouve les vols ayant décollé en novembre ou en décembre :

```
filter(flights, month == 11 | month == 12)
```

Opérateurs logiques

- Une autre façon de formuler l'instruction précédente est la suivante, faisant appel à l'opérateur `%in%` :

```
nov_dec <- filter(flights, month %in% c(11,12))
```

Valeurs manquantes

- Un aspect important de R, qui peut rendre les comparaisons délicates, est le problème des valeurs manquantes NA (*not available*).

- filter() ***n'inclut que les rangées pour lesquelles la condition est vraie : il exclut non seulement les valeurs fausses mais aussi les valeurs NA.***

- Si on souhaite les préserver, il faut le spécifier explicitement :

Valeurs manquantes

```
df <- tibble(x = c(1,NA,3))

filter(df, x > 1)

# A tibble: 1 x 1
      x
  <dbl>
1     3
```

Valeurs manquantes

```
filter(df, is.na(x) | x > 1)

# A tibble: 2 x 1
      x
  <dbl>
1    NA
2     3
```

Questions

- Trouvez tous les vols qui :

A. ont eu un retard de deux heures ou plus

B. on atterri à Houston (IAH ou HOU)

C. ont décollé en été (juillet, août, septembre)

D. sont arrivées avec plus de deux heures de retard, alors qu’ils étaient partis à l’heure

E. ont décollé entre minuit et 6h du matin inclus.

Questions

- `between()` est un autre outil de filtrage de dplyr. Que fait-il ? Peut-on l'utiliser pour simplifier le code des questions précédentes ?

- Quel est le nombre de vols présentant une heure de décollage (`dep_time`) manquante ? Quelles sont les autres variables manquantes ? Que peuvent représenter ces lignes ?

- Pourquoi NA | TRUE n'est-il pas une valeur manquante ? Pourquoi FALSE & NA n'est-il pas une valeur manquante ?

arrange()

- arrange() fonctionne comme filter() mais au lieu de sélectionner des lignes, elle modifie leur ordre.

- Elle prend en argument un tableau de données et un ensemble de noms de variables (ou d'expressions plus complexes) selon lesquelles on veut le réordonner.

- Si on fournit plus d'un nom de colonne, les colonnes suivantes sont utilisées en cas d'égalité :

arrange()

arrange(flights, month, day)

 # A tibble: 336,776 x 19

year	month	day	dep_time	sched_dep_time	dep_delay	arr_time
<int>	<int>	<int>	<int>	<int>	<dbl>	<int>
2013	1	1	517	515	2	830
2013	1	1	533	529	4	850
2013	1	1	542	540	2	923
2013	1	1	544	545	-1	1004
2013	1	1	554	600	-6	812
2013	1	1	554	558	-4	740
2013	1	1	555	600	-5	913
2013	1	1	557	600	-3	709
2013	1	1	557	600	-3	838
2013	1	1	558	600	-2	753

 # ... with 336,766 more rows, and 12 more variables:

 # sched_arr_time <int>, arr_delay <dbl>, carrier <chr>, flight <int>,

 # tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>,

 # distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>

arrange()

- On utilise `desc()` pour réordonner selon une colonne en ordre descendant :

```
arrange(flights, desc(arr_delay))
```

- Les valeurs manquantes sont toujours placées à la fin.

Questions

- Comment pourrait-on utiliser arrange() pour placer toutes les valeurs manquantes au début ? Indication : faire appel à `is.na()`.

- Trier les vols pour trouver ceux ayant eu les retards les plus importants.

- Trier les vols selon leur vitesse.

- Quels sont les vols qui ont parcouru la distance la plus grande ? Ceux qui ont parcouru la distance la plus petite ?

select()

- Il n’est pas rare de recevoir des jeux de données comprenant des centaines, voire des milliers de variables.

- Dans ce cas, la première tâche est souvent de cibler les variables auxquelles on s’intéresse réellement.

- select() permet de « zoomer » facilement sur un sous-ensemble pertinent, à l’aide d’opérations basées sur le nom des variables.

- Exemple sur le jeu de données `flights` :

select()

```
> select(flights, year, month, day)

# A tibble: 336,776 x 3

    year month   day

   <int> <int> <int>

 1  2013     1     1

 2  2013     1     1

 3  2013     1     1

 4  2013     1     1

 5  2013     1     1

# ... with 336,766 more rows
```

select()

- Sélectionner toutes les colonnes entre `year` et `day` (incluses) :

```
select(flights, year:day)
```

select()

- Sélectionner toutes les colonnes ***sauf celles*** entre **`year`** et **`day`** (incluses) :

```
select(flights, -(year:day))
```

select()

- On peut utiliser un certain nombre de fonctions auxiliaires au sein de select() :

- `starts with(« abc »)` : sélectionne les colonnes dont le nom commence par « abc ».

- `ends_with(« xyz »)` : sélectionne les colonnes dont le nom se terminer par « xyz ».

- `contains(« ijk »)` : sélectionne les colonnes dont le nom contient « ijk ».

select()

- `matches(« (.)\\1 »)` : sélectionne les colonnes dont le nom est compatible avec une expression régulière.

- `num_range(« x »,1:3)` : sélectionne x1, x2 et x3.

- On peut consulter `?select` pour plus d'informations.

select()

- select() peut également être utilisée pour renommer les variables mais c'est rarement le cas car elle supprime toutes les colonnes non mentionnées explicitement.

- On utilise plutôt **`rename()`**, qui fonctionne comme select() mais conserve les colonnes non mentionnées :

```
rename(flights, tail_num = tailnum)
```

select()

- Une autre option consiste à utiliser select() avec la fonction `everything()`.

- Cela permet notamment de déplacer certaines variables au début du tableau de données :

```
select(flights, time_hour, air_time, everything())
```

Questions

- Que se passe-t-il si on inclut plusieurs fois le nom d'une variable dans un appel à select() ?

- Quel est le rôle de la fonction `one_of()` ? Quel est son intérêt en combinaison avec select() ?

mutate()

- mutate() sert à ajouter de nouvelles colonnes à un tableau de données. Celles-ci doivent être définies à partir des colonnes existantes :

mutate()

```
flighst_sml <- select(flights,
                      year:day,
                      ends_with("delay"),
                      distance,
                      air_time
                      )
```

mutate()

```
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       speed = distance / air_time * 60
       )
```

mutate()

```
# A tibble: 336,776 x 9
    year month   day dep_delay arr_delay distance air_time  gain speed
   <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
 1  2013     1     1         2        11     1400      227     9  370.
 2  2013     1     1         4        20     1416      227    16  374.
 3  2013     1     1         2        33     1089      160    31  408.
 4  2013     1     1        -1       -18     1576      183   -17  517.
 5  2013     1     1        -6       -25      762      116   -19  394.
 6  2013     1     1        -4        12      719      150    16  288.
 7  2013     1     1        -5        19     1065      158    24  404.
 8  2013     1     1        -3       -14      229       53   -11  259.
 9  2013     1     1        -3        -8      944      140    -5  405.
10  2013     1     1        -2         8      733      138    10  319.
# ... with 336,766 more rows
```

mutate()

- Notons qu'on peut faire référence à des colonnes qu'on vient de créer :

```
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       hours = air_time * 60,
       gain_per_hour = gain / hours
       )
```

mutate()

- Si on ne veut conserver que les nouvelles variables, on utilise `transmute()` :

```
transmute(flights_sml,
  gain = arr_delay - dep_delay,
  hours = air_time * 60,
  gain_per_hour = gain / hours
  )
```

mutate()

```
# A tibble: 336,776 x 3
    gain hours gain_per_hour
   <dbl> <dbl>         <dbl>
 1     9 13620       0.000661
 2    16 13620       0.00117
 3    31  9600       0.00323
 4   -17 10980      -0.00155
 5   -19  6960      -0.00273
 6    16  9000       0.00178
 7    24  9480       0.00253
 8   -11  3180      -0.00346
 9    -5  8400      -0.000595
10    10  8280       0.00121
# ... with 336,766 more rows
```

mutate()

- De nombreuses fonctions peuvent être utiliser avec mutate() pour faciliter la création de nouvelles variables.

- Leur propriété essentielle est qu'elles doivent être ***vectorisées*** : elles doivent prendre un vecteur en entrée et renvoyer un vecteur de même longueur en sortie.

- Voici une sélection de fonctions souvent utiles :

mutate()

- + , -, *, /, ^
- Tous ces opérateurs sont vectoriels et utilisent des « règles de recyclage » : si un paramètre est plus court que l’autre, il est automatiquement prolongé.
- %/% (division entière), %% (reste)
- log(), log2(), log10()

mutate()

- Les décalages `lead()` et `lag()` sont utilisés pour accéder aux valeurs qui suivent ou qui précèdent immédiatement.

- Cela permet de calculer des différences glissantes `(x - lag(x))` ou de détecter lorsqu'une valeur change `(x != lag(x))`.

- Les décalages sont particulièrement utiles en conjonction avec group_by().

mutate()

- R fournit des fonctions pour effectuer des sommes et produits, ou des calculs de minimum et de maximum, cumulés : `cumsum()`, `cumprod()`, `cummin()` et `cummax()`.

- dplyr fournit en outre `cummean()` pour les moyennes cumulées.

mutate()

- Il existe par ailleurs plusieurs fonctions de classement.
- La plus commune est `min_rank()` qui classe selon l'ordre ascendant.
- Pour classer selon l'ordre descendant, on utilise `desc()` :

```
y <- c(1, 2, 2, NA, 3, 4)

min_rank(desc(y))
```

mutate()

- D’autres fonctions de classement sont : `row_number()`, `dense_rank()`, `percent_rank()`, `cume_dist()` et `ntile()`.

- Consulter l’aide pour plus d’information.

Questions

- Trouver les 10 vols les plus retardés au moyen d'une fonction de classement. Comment gérer les égalités ? Lire soigneusement la documentation de `min_rank()`.

- Que renvoie 1:3 + 1:10 ? Pourquoi ?

summarize()

- Le dernier verbe élémentaire de dplyr est summarize().
- Il réduit un tableau de données à une seule ligne :

```
> summarize(flights, delay = mean(dep_delay, na.rm = TRUE))

# A tibble: 1 x 1

  delay

  <dbl>

1  12.6
```

summarize()

- summarize() est surtout utile combiné à group_by(), qui modifie l'unité d'analyse pour la faire porter sur des groupes d'individus plutôt que sur les individus initiaux.

- Lorsqu'on utilise les verbes de dplyr sur un tableau de données groupées, ils sont automatiquement appliqués par groupe.

- Par exemple, si on exécute le code précédent sur un tableau de données groupées par date, on obtient le retard moyen par date :

summarize()

```
by_day <- group_by(flights, year, month, day)

summarize(by_day, delay = mean(dep_delay, na.rm = TRUE))
```

summarize()

```
# A tibble: 365 x 4
# Groups:   year, month [?]
    year month   day delay
   <int> <int> <int> <dbl>
 1  2013     1     1 11.5
 2  2013     1     2 13.9
 3  2013     1     3 11.0
 4  2013     1     4  8.95
 5  2013     1     5  5.73
 6  2013     1     6  7.15
 7  2013     1     7  5.42
 8  2013     1     8  2.55
 9  2013     1     9  2.28
10  2013     1    10  2.84
# ... with 355 more rows
```

summarize()

- La combinaison de group_by() et summarize() fournit l'un des outils les plus fréquemment utilisés de dplyr : les résumés groupés.

Canaux

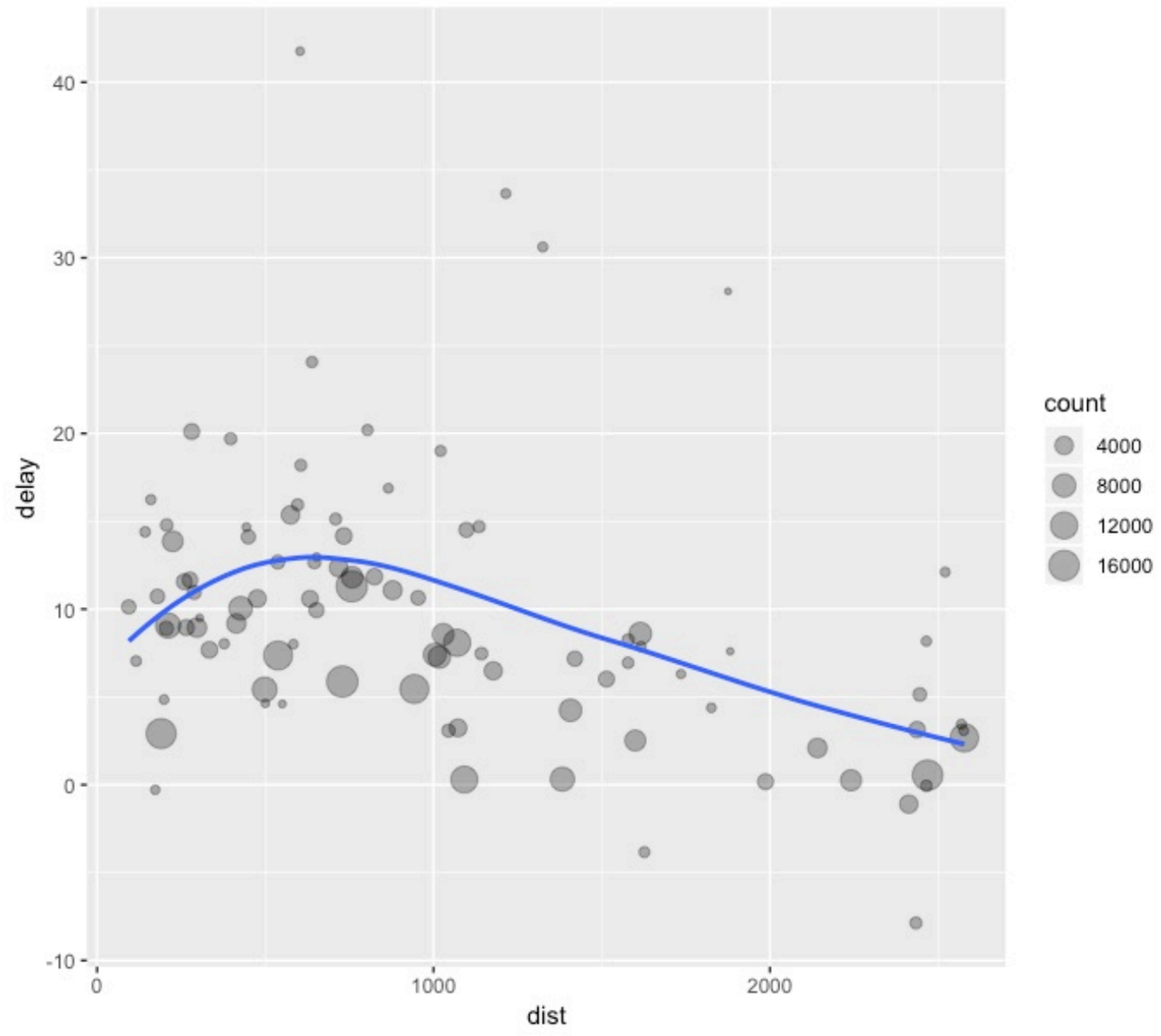
- Imaginons qu'on souhaite explorer la relation entre la distance et le retard moyen pour chaque aéroport. En particulier, on souhaite répondre à la question suivante : *les trajets plus longs permettent-ils de compenser le retard pris au décollage* ?

- En utilisant ce que l'on connait de dplyr, on peut écrire le code suivant :

Canaux

```
by_dest <- group_by(flights, dest)

delay <- summarize(by_dest,

                   count = n(),

                   dist = mean(distance, na.rm = TRUE),

                   delay = mean(arr_delay, na.rm = TRUE)

                   )

delay <- filter(delay, count > 20, dest != "HNL")



ggplot(data = delay, mapping = aes(x = dist, y = delay)) +

  geom_point(aes(size = count), alpha = 1/3) +

  geom_smooth(se = FALSE)
```

Canaux


40
30
20
10
0
-10
delay
0
1000
2000
dist
count
4000
8000
12000
16000

Canaux

- La préparation des données comporte trois étapes :

1. Grouper les vols par destination.

2. Résumer, pour calculer la distance, le retard moyen et le nombre de vols par destination.

3. Filtrer, pour supprimer les points perturbants (destinations à effectifs faibles) et l'aéroport d'Honolulu, presque deux fois plus loin que le second plus distant.

Canaux

- On est obligé de ***nommer*** chaque tableau de données intermédiaire, même s'il ne nous intéresse pas dans la pratique.

- Une façon alternative d'aborder le problème est d'utiliser les ***canaux*** (%>%) :

Canaux

```
delays <- flights %>%
  group_by(dest) %>%
  summarize(
    count = n(),
    dist = mean(distance, na.rm = TRUE),
    delay = mean(arr_delay, na.rm = TRUE)
  ) %>%
  filter(count > 20, dest!= "HNL")
```

Canaux

- Ce code se concentre sur les transformations et non sur les données transformées, ce qui le rend plus facile à lire… avec un peu d’habitude.

- On peut lire « puis » pour %>% lorsqu’on lit ce type de code.

Canaux

- Le fonctionnement interne réalise les opérations de la façon suivante :

- **`x %>% f(y)`** est transformé en **`f(x,y)`**, puis **`x %>% f(y) %>% g(z)`** est transformé en **`g(f(x,y),z)`**, etc.

- Le canal %>% provient du package magrittr. Les packages du tidyverse le chargent automatiquement.