

TP 5 : Visualisation de données, graphiques

Les lignes qui commencent par > dans ce fichier Rmd sont des lignes qui correspondent à des questions qui vous sont posées.

Les chunks que vous devez modifier ont un nom qui commence par ANS. Les questions qui nécessitent une réponse rédigée sont en italique, donc entourées d'étoiles dans le fichier Rmd. Vous devez y répondre après le mot "Réponse :." dans le fichier Rmd.

1 Le package ggplot2

Nous présentons rapidement le nouveau système de graphique `ggplot2` du package du même nom. Il existe d'autres façons de faire des graphiques en R (en particulier avec les fonctions de bases `plot`, `points`, `lines`, `abline`... ou le package `lattice`). Mais `ggplot2` est le plus récent, et celui qui produit les graphiques les plus aboutis.

Chargeons le package. Il fait partie de `tidyverse`, on peut donc se contenter de charger cet ensemble de package.

```
library(tidyverse)
# Si tidyverse est mal installé chez vous, chargez :
# library(ggplot2)
# library(tibble)
# library(tidyr)
# library(readr)
# library(dplyr)
```

Le package `ggplot2` repose sur une idée de grammaire des graphiques, que décompose toute représentation graphique ainsi.

- Des **données** (data), que l'on veut visualiser, et un ensemble de **correspondances** graphiques (aesthetic mappings) qui décrivent comment les variables sont reliées à des attributs graphiques (abscisses, ordonnées, couleurs, formes, transparence, etc.).
- Des **couches** (layers), faites d'éléments graphiques et de transformations statistiques. Les objets **géo-métriques** (ou `geom` en abrégé) représentent ce que l'on voit : des points, des courbes, des polygones, des lignes, etc. Les **transformations statistiques** (`stat` en abrégé) résument les données de façon utile, en créant des comptages par intervalle (pour les histogrammes), un modèle de régression,...
- Des **échelles** (scales) qui relient les valeurs dans l'espace des données à des valeurs graphiques : échelles sur les axes, échelles de couleurs, de tailles ou de formes.
- Un système de **coordonnées** (`coord` en abrégé), qui sont le plus souvent le système cartésien, mais on peut aussi utiliser les coordonnées polaires, etc.
- Une spécification de **facetage** (`facet` en abrégé), qui décrit comment séparer les données en sous-ensemble et afficher ses sous-ensembles dans différents sous-graphiques
- Un **thème** (theme en anglais), qui contrôle les paramètres fins de réglage du graphique, comme la police de caractère, la couleur du fond, etc.

2 Consommation d'essence

Le jeu de données `mpg` du package `ggplot2` contient des informations sur la consommation des voitures les plus populaires aux USA entre 1999 et 2008. La page d'aide de ce jeu de données permet de comprendre les différentes variables.

?mpg

Les variables `cty` et `hwy` contiennent le nombre de miles parcourus avec 1 gallon d'essence, respectivement en ville et sur autoroute. On rappelle qu'un gallon d'essence fait 3.78541 L d'essence, et qu'un mile fait 1.60934 km. Convertir ces deux variables pour créer deux nouvelles variables `ctvol` et `hwvol` qui contiennent le nombre de litres d'essence nécessaire pour parcourir 100 km. On pourra utiliser la fonction `mutate`.

```
mpg <- mpg %>%
  mutate(ctvol = 3.78541 / 1.60934 / cty * 100,
         hwvol = 3.78541 / 1.60934 / hwy * 100)
```

2.1 Graphiques simples

Un graphique ggplot2 a au minimum ces trois ingrédients principaux :

- des données,
- des correspondances graphiques,
- une couche, qui décrit comment rendre les observations.

Exemple :

```
ggplot(mpg, aes(x = displ, y = hwvol)) +
  geom_point()
```

Ici, les données sont dans la table `mpg`, on a mis en correspondance l'axe des x (ou des abscisses) avec la cylindrée du moteur (`displ`), en litre, et l'axe des y avec la consommation sur autoroute (`hwvol`) en litre pour 100 km. La seule couche graphique propose de représenter les données sous forme de points.

1. Représenter graphiquement la consommation sur autoroute (en L / 100km) en fonction de la consommation en ville.
2. Que se passe-t-il si on exécute uniquement `ggplot(mpg, aes(x = displ, y = hwvol))` sans ajouter la couche qui affiche les points ?

Réponse :

1.

2.

Pour ajouter d'autres variables à ce graphique, on peut utiliser d'autres correspondances graphiques comme la couleur, la forme et la taille. Les correspondances graphiques sont ajoutées dans l'appel à la fonction `aes()` :

- `aes(x = displ, y = hwvol, colour = class)` ou `aes(x = displ, y = hwvol, color = class)`
- `aes(x = displ, y = hwvol, shape = drv)`
- `aes(x = displ, y = hwvol, size = cyl)`

Tester ces trois cas dans le chunk ci-dessous, en remplaçant ce qu'il faut dans le code du chunk exemple1.

Ggplot2 se charge de convertir les données brutes (par exemple `f r` ou `4` pour `drv`) sur le graphique avec une échelle. On peut définir les correspondances dans l'appel à la fonction `ggplot` (dans ce cas, ce sont des correspondances par défaut pour toutes les couches que l'on ajoute au graphique), ou à l'intérieur de l'appel à la

fonction `geom_point()`. On peut même aller jusqu'à utiliser `ggplot()` sans aucun argument. Dans ce cas, il faut aussi passer le jeu de données à la couche géométrique.

Exemples :

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

Les paramètres graphiques qui sont données en dehors de l'appel à la fonction `aes()` sont des éléments communs à toute la couche.

Si l'on veut dessiner tous les points en bleu, laquelle des deux commandes ci-dessous faut-il utiliser ?

Réponse :

```
# Commande 1  
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(aes(colour = "blue"))  
# Commande 2  
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point(colour = "blue")
```

Combien y a-t-il d'échelles dans le graphique produit par la commande 1 ? Lesquelles ?

Réponse :

Que se passe-t-il si l'on met en correspondance une variable continue avec la couleur des points ? Tester sur un exemple. Même question pour la taille des points (qui se règle avec `size` =).

Enfin, le facettage permet de découper le jeu de données en sous-jeux et de produire plusieurs graphiques. Pour cela, on découpe en fonction des valeurs prises par une variable discrète. Ici, on peut s'intéresser à `drv` ou `class`. On utilise la fonction `facet_wrap` et on passe en argument le nom de la variable, précédé d'un tilde `~`. Par exemple :

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

1. Que se passe-t-il si on essaie le facettage sur une variable continue comme `hwy` ?
2. Refaire le graphique du chunk couleur ci-dessus, mais en facetant suivant les roues qui sont motrices.

```
# 1.
```

```
# 2.
```

Si on veut faire une grille de graphique, où la variable `cy1` varie en abscisse, et la variable `drv` en ordonnées, il faut utiliser la fonction `facet_grid` comme ceci.

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

2.2 Ajouter plusieurs couches

On peut ajouter un “lisseur” au dessus du nuage de points, pour faire de la régression. On utilise pour cela la fonction `geom_smooth`.

Par défaut, cette fonction utilise une méthode de régression locale (non paramétrique, appelée “loess”). Pour obtenir le résultat d’une régression linéaire, il faut spécifier `method = "lm"` :

```
ggplot(mpg, aes(x = displ, y = hwvol)) +  
  geom_point() +  
  geom_smooth()  
ggplot(mpg, aes(x = displ, y = hwvol)) +  
  geom_point() +  
  geom_smooth(method = "lm", colour = "red", fill = "DarkSalmon")
```

2.3 D’autres géométries

Vous n’avez sans doute pas remarqué que, dans les graphiques dessinés jusqu’à maintenant, plusieurs points sont les uns au dessus des autres. Pour éviter ce chevauchement, on peut utiliser une couche de type `geom_jitter` qui commence par ajouter un petit bruit aléatoire sur la valeur avant de l’utiliser pour l’affichage. Cela permet de départager les exaequos.

Exemple :

```
ggplot(mpg, aes(x = displ, y = hwvol, colour = class)) +  
  geom_jitter()
```

Pour représenter la distribution conditionnelle d’une variable continue en fonction d’une variable discrète, on peut utiliser des boxplots ou des diagrammes en violon, qui sont plus parlant qu’un nuage de points, même obtenu avec la fonction `jitter`.

Exemples :

```
ggplot(mpg, aes(x = drv, y = hwvol)) +  
  geom_point()  
ggplot(mpg, aes(x = drv, y = hwvol)) +  
  geom_jitter()  
ggplot(mpg, aes(x = drv, y = hwvol)) +  
  geom_boxplot()  
ggplot(mpg, aes(x = drv, y = hwvol)) +  
  geom_violin()
```

En utilisant le paramètre `fill`, refaire le dernier diagramme en violon, en utilisant la couleur “blue4”.

3 Transformations statistiques

Le jeu de données `diamonds` contient des informations concernant un peu moins de cinquante quatre mille diamants.

```
?diamonds
```

On peut utiliser un diagramme en barre pour représenter la distribution de la variable `cut` dans ce jeu de données. Dans ce cas, `ggplot` doit d'abord faire des comptages pour chacune des valeurs possibles avant de faire le graphique. Il s'agit d'une transformation du jeu de données. Ajouter la transformation statistique `stat_count` plutôt que `geom_bar` produit le même graphique. Ainsi

```
ggplot(diamonds) +  
  geom_bar(aes(x = cut))  
ggplot(diamonds) +  
  stat_count(aes(x = cut))
```

À l'aide des fonction `group_by`, `summarise` et de la fonction `n` du package `dplyr`, retrouver ces comptages numériquement, en créant un tableau à deux colonnes, la première nommée `cut`, et la seconde nommée `eff`. Enregistrer le résultat dans la table `effectifs_cut`

On peut utiliser directement cette table d'effectifs dans une couche `geom_bar`, à condition de lui dire de ne pas faire de transformation statistique, en passant `stat = "identity"`. On peut aussi lui dire de calculer des fréquences (proportions) plutôt que des effectifs, mais dans ce cas, il faut définir un regroupement. Exemple :

```
ggplot(effectifs_cut, aes(x = cut, y = eff)) +  
  geom_bar(stat = "identity")  
ggplot(diamonds) +  
  geom_bar(aes(x = cut, y = ..prop.., group = 1))
```

Enfin, la transformation `stat_summary` permet de représenter plusieurs statistiques sur la distribution d'une variable continue en fonction d'une variable discrète.

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

1. Que fait une couche de type `geom_linerange` ? (Voir la page d'aide)

Réponse :

2. Reproduire le graphique du chunk `stat_summary` en utilisant une couche `geom_linerange` et une couche `geom_point` plutôt que `stat_summary`. Il faudra commencer par faire les calculs dans une nouvelle table, à l'aide des fonctions `group_by` et `summarise`.
3. Que se passe-t-il si l'on remplace `geom_linerange` par `geom_errorbar` ? Et si on ajoute `width = 0.2` dans l'appel à `geom_errorbar` ?
4. Dans le code du chunk `stat_summary`, remplacer les fonctions `min` et `max` par des fonctions qui calculent les quantiles d'ordre 0.25 et 0.75 (appelés respectivement 1er et 3ème quartiles).

```
# 2.
```

```
# 3.
```

4.

On peut essayer de représenter deux variables discrètes simultanément. Par exemple :

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

Recopier la commande du chunk deux_discretes dans le chunk ci-dessous en modifiant le paramètre position : Que se passe-t-il si on remplace "identity" par "fill" ou par "dodge" dans la commande du chunk deux_discretes ?

Que fait la fonction geom_col ? (Regarder l'exemple de la page d'aide.) En quoi est-elle différente de geom_bar ?

Réponse :

4 Manipuler des graphiques et changer le fonctionnement de base

La fonction labs permet de modifier les légendes des différents axes (x, y, couleur, forme), et la fonction ggtitle permet de mettre un titre et un sous-titre. Voici un exemple

```
ggplot(data = diamonds) +
  stat_summary(
    mapping = aes(x = cut, y = depth),
    fun.ymin = min,
    fun.ymax = max,
    fun.y = median
  ) +
  labs(
    x = "Qualité de la taille",
    y = "Profondeur"
  ) +
  ggtitle(
    "Distribution de la profondeur",
    subtitle = "en fonction de la qualité de la taille du diamant"
  )
```

```
ggplot(mpg, aes(displ, hwvol, colour = factor(cyl))) +
  geom_jitter()
```

Corriger les légendes du graphique ci-dessus avec labs, en mettant les unités entre parenthèses.

On peut aussi enregistrer le graphique, ou une partie du graphique dans un objet. Dans ce cas, l'affichage ne se fait pas. Il faut le demander explicitement. La fonction ggsave permettent d'exporter le dernier graphique (ou un graphique enregistré dans un objet) dans un fichier pdf, jpeg, png,...

Que font les 4 lignes commentées ci-dessous ? Lequel des deux fichiers pdf créé contient le mot lapin ? Pourquoi ?

```
p <- ggplot(mpg, aes(displ, hwvol, colour = factor(cyl)))
p <- p + geom_jitter()
p # ligne 1
```

```
p + labs(x = "Lapin") # ligne 2
ggsave("graphique1.pdf") # ligne 3
ggsave("graphique2.pdf", plot = p) # ligne 4
```

Enfin, on peut modifier le thème du graphique. Il existe plusieurs thèmes achevés dans ggplot2 : bw, classic, dark, gray ou grey, light, linedraw, minimal ou void. Le package ggtheme en fournit encore d'autres. On change le thème en utilisant des fonctions dont le nom commence par `theme_`. Par exemple :

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

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

On peut également échanger x et y avec `coord_flip` et passer en coordonnées polaires avec `coord_polar`. Par exemple :

```
bar <- ggplot(data = diamonds) +
  geom_bar(
    mapping = aes(x = cut, fill = cut),
    show.legend = FALSE,
    width = 1
  ) +
  labs(x = NULL, y = NULL)
bar
bar + coord_flip()
bar + coord_polar()
```

5 La carte de la France

On peut même dessiner des cartes de pays ou de régions, pour faire de la statistique spatiale ensuite.

```
fr <- map_data("france")
ggplot(fr, aes(long, lat, group = group)) +
  geom_polygon(fill = "white", colour = "black") +
  coord_quickmap()
```

L'objet `fr` ci-dessus contient les coordonnées de points sur les frontières des départements. On utilise donc la géométrie polygone pour tracer ces frontières. Si on ne trace que les points, on obtient le graphique ci-dessous. Notez que l'utilisation de la fonction `coord_quickmap` permet d'avoir la même échelle sur les deux axes du graphique, ce qui est assez utile ici.

```
ggplot(fr, aes(long, lat)) + geom_point(size = .2)
```

La région PACA est constituée de six départements, que l'on peut récupérer et afficher tous seuls.

```
PACA <- c("Alpes-de-Haute-Provence", "Hautes-Alpes", "Alpes-Maritimes",
          "Bouches-du-Rhone", "Var", "Vaucluse")
ggplot(fr %>% filter(region %in% PACA), aes(long, lat, group = group)) +
  geom_polygon(aes(fill = region), colour = "black") +
  coord_quickmap() +
  theme_void()
```