Graphiques avec ggplot2 en R

Sophie Baillargeon, Université Laval

2021-02-25

Table des matières

Note préliminaire : Lors de leur dernière mise à jour, ces notes ont été révisées en utilisant R 4.0.3, le package `ggplot2` version 3.3.3, le package `dplyr` version 1.0.3, le package `forcats` version 0.5.1 et le package `maps` version 3.3.0. Pour d'autres versions, les informations peuvent différer.

---

1 Introduction

Le package `ggplot2` a été publié pour la première fois en 2006 sous le nom de `ggplot`. Il fut amélioré de façon importante et renommé `ggplot2` en 2007. Son créateur est Hadley Wickham, qui est aussi derrière plusieurs des packages du `tidyverse`, duquel `ggplot2` fait partie. Le package est maintenant développé par toute une équipe, dont des employés de RStudio. Il implémente la **grammaire graphique** présentée dans : « Wilkinson, L. (2005). *The grammar of graphics*, 2e édition. Springer ».

Le package `ggplot2` a été conçu en ayant comme objectif la **simplicité d'utilisation** et la **qualité des graphiques produits**. Il reprend les forces suivantes des systèmes graphiques R précédents :

- système de base : création de graphiques par **couches** (ajouts séquentiels d'éléments) ;
- package `lattice` : **représentations multivariées** simples.

tout en apportant les améliorations suivantes :

- **esthétique** par défaut pensée de façon à transmettre plus efficacement les informations contenues dans le graphique ;
- **automatisation de certaines configurations** graphiques, notamment les légendes ;
- **ajout de transformations statistiques** communes (p. ex. courbes de lissage, barres d'erreur) facilité.

Dans le package `ggplot2`, tout a été repensé pour être plus simple d'utilisation et surtout pour que le graphique produit transmette plus efficacement l'information qu'il contient.

Illustration de @allison_horst : https://github.com/allisonhorst/stats-illustrations

L'utilisation du package `ggplot2` est présentée ici de façon brève, mais avec tout de même assez de détails

pour démarrer un apprentissage de cet outil aux possibilités vastes.

```
# Chargement du package
library(ggplot2)
```

Présentation des données utilisées pour les exemples

Comme dans les autres notes sur les graphiques en R, les données `quakes` auxquelles deux facteurs sont ajoutés sont utilisées dans les exemples de cette fiche.

```
quakes$mag_catego <- factor(floor(quakes$mag))
quakes$region <- factor(
  ifelse(quakes$long < 175, yes = "Ouest", no = "Est"),
  levels = c("Ouest", "Est")
)
str(quakes)
```

```
## 'data.frame':    1000 obs. of  7 variables:
##  $ lat       : num  -20.4 -20.6 -26 -18 -20.4 ...
##  $ long      : num  182 181 184 182 182 ...
##  $ depth     : int  562 650 42 626 649 195 82 194 211 622 ...
##  $ mag       : num  4.8 4.2 5.4 4.1 4 4 4.8 4.4 4.7 4.3 ...
##  $ stations  : int  41 15 43 19 11 12 43 15 35 19 ...
##  $ mag_catego: Factor w/ 3 levels "4","5","6": 1 1 2 1 1 1 1 1 1 1 ...
##  $ region    : Factor w/ 2 levels "Ouest","Est": 2 2 2 2 2 2 1 2 2 2 ...
```

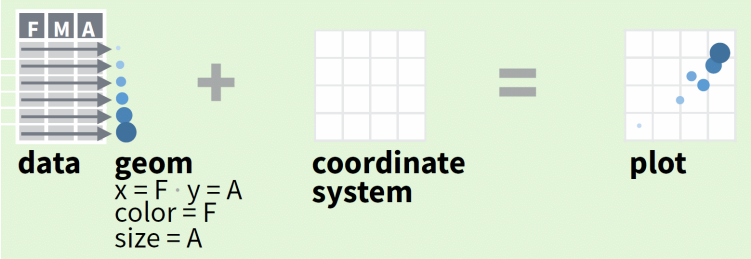
2 Principes de base de ggplot2

2.1 Grammaire graphique

Le principe de base derrière la **g**rammaire **g**raphique (d'où le gg dans `ggplot`) est qu'un graphique statistique est une représentation de **données**, dans un **système de coordonnées** spécifique, divisée en éléments de base :

- **éléments géométriques** (*geoms*) : points, lignes, barres, etc. ;
- **propriétés visuelles** (*aesthetics*) des éléments géométriques : axes, couleurs, formes, tailles, etc.
- **transformations statistiques**, si désiré : courbe de régression ou de lissage, région d'erreur, etc.

Un graphique est spécifié en **associant des variables**, provenant des données, **à des propriétés visuelles** des éléments géométriques du graphique. Ces principes sont représentés comme suit sur la feuille de triche officielle du package.



Source : https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf

2.2 Principales fonctions et gabarit de code

Les principales fonctions du package `ggplot2` sont les suivantes :

- `ggplot` : initialisation d'un objet de classe `ggplot` ;
- `qplot` : initialisation rapide (`q` pour *quick*) d'un objet `ggplot` ;
- `+` : opérateur pour l'ajout de couches ou la modification de configurations dans un objet `ggplot` ;
- fonctions de type `geom_*` (p. ex. `geom_point`, `geom_boxplot`, `geom_bar`, etc.) : spécification de couches à ajouter à un graphique ;
- `aes` : création d'un `mapping`, soit une association entre entre des propriétés visuelles et des variables ;
- `ggsave` : enregistrement d'un graphique.

Notons que l'utilisation de la fonction `qplot` est plus intuitive que celle de `ggplot` pour des gens familiers avec `plot`. La fonction `qplot` n'offre cependant pas toutes les possibilités de `ggplot`. Elle ne sera pas couverte dans ce document.

Voici un gabarit minimaliste de code de création d'un graphique `ggplot2` (*source* : http://r4ds.had.co.nz/data-visualisation.html#a-graphing-template) :

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

ou encore la version « tout sur la même ligne » :

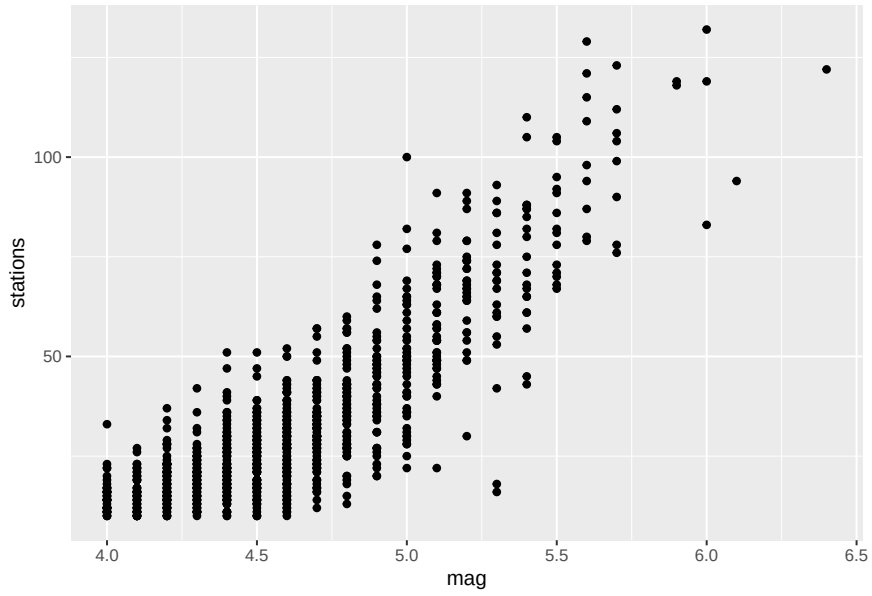
```
ggplot(data = <DATA>) + <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Dans ce gabarit, il faut remplacer les éléments entre `<` et `>` comme suit :

- `<DATA>` = jeu de données, stocké dans un **data frame** (ou un tibble), dans lequel les variables catégoriques doivent être des **facteurs** (dont **les libellés des niveaux ont avantage à être informatifs**, car ils apparaîtront dans le graphique) ;
- `<GEOM_FUNCTION>` = le nom d'une fonction de type `geom_*` pour ajouter une couche au graphique ;
- `<MAPPINGS>` = arguments à fournir à la fonction `aes` (les arguments acceptés varient un peu selon la `<GEOM_FUNCTION>`).

Voici un premier exemple simple.

```
ggplot(data = quakes) + geom_point(mapping = aes(x = mag, y = stations))
```



Les arguments `data` et `mapping` peuvent être fournis :

- à la fonction `ggplot` : leur valeur est alors commune à toutes le couches
  (il est tout de même possible de forcer l'utilisation d'autres données ou d'autres associations pour des

couches spécifiques) ;
- ou à une `<GEOM_FUNCTION>` : leur valeur est alors spécifique à la couche produite.

Étant donné que le graphique produit par le gabarit minimaliste ne comprend qu'une seule couche, les syntaxes suivantes sont donc équivalentes à celle du gabarit.

```
# Argument data spécifique (fourni dans l'appel à la <GEOM_FUNCTION>)
ggplot() + <GEOM_FUNCTION>(data = <DATA>, mapping = aes(<MAPPINGS>))
```

```
# Argument mapping global (fourni dans l'appel à ggplot)
ggplot(data = <DATA>, mapping = aes(<MAPPINGS>)) + <GEOM_FUNCTION>()
```

Quelques fonctions de type `geom_*` :

Fonction	Type de graphique	Élément(s) ajouté(s)
`geom_point`	diagramme de dispersion	points selon des coordonnées
`geom_line`	diagramme en lignes	segments de droites reliant des points
`geom_bar`	diagramme à barres	barres disjointes, de hauteurs spécifiées ou calculées = fréquences des niveaux d'un facteur
`geom_histogram`	histogramme	barres collées, de hauteurs calculées = fréquences d'observations d'une variable numérique tombant dans des intervalles joints (*bin*)
`geom_boxplot`	diagramme en boîte	*boxplots*
`geom_density`	courbe de densité à noyau	courbe de la densité estimée par noyau (*kernel density*)
`geom_qq`	diagramme quantile-quantile théorique	points pour les couples de quantiles empiriques et théoriques
⋮	il en existe plusieurs autres	voir http://ggplot2.tidyverse.org/reference/

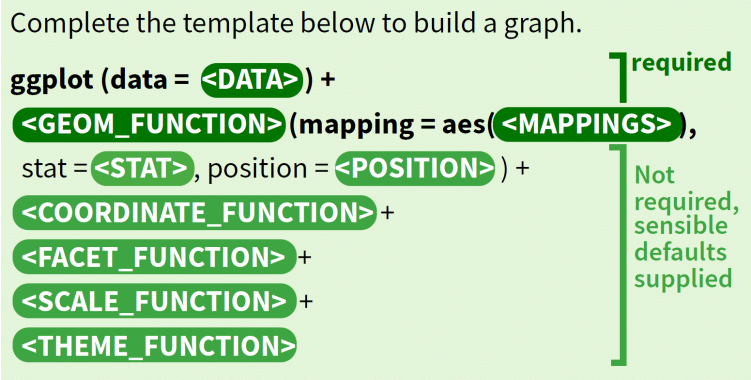
La plupart de ces fonctions cachent des transformations statistiques (p. ex. `geom_bar` et `geom_histogram` calculent des fréquences, `geom_boxplot` calcule des quantiles, `geom_density` estime une densité, etc.)

Quelques autres fonctions communes :

Voici quelques autres fonctions communes de `ggplot2` servant à ajouter des couches ou modifier des configurations dans un objet `ggplot` initialisé :

- fonctions `labs`, `ggtitle`, `xlab`, `ylab` : ajouter un titre, modifier les noms d'axes ;
- fonctions de type `coord_*` : modifier des configurations reliées au système de coordonnées ;
- fonctions de type `facet_*` : créer des grilles de sous-graphiques
  - chacun des sous-graphiques est conditionnel à la valeur de facteurs(s), il représente donc seulement le sous-ensemble des observations ayant une modalité particulière pour ce(s) facteur(s) ;
- fonctions de type `scale_*` : modifier les échelles de certaines propriétés visuelles (p. ex. couleurs, formes, tailles, etc.)
- fonctions de type `theme_*` : modifier des configurations reliées à l'apparence du graphique ;
- fonctions de type `stat_*` : ajouts d'éléments tirés d'un calcul mathématique ou statistique.

Le gabarit minimaliste présenté précédemment pourrait être rendu plus complet comme suit.



Source : https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf

2.3 Objet de classe `ggplot`

Un graphique produit en `ggplot2` est en fait un objet de classe `ggplot`. La fonction `ggplot` initialise un objet de cette classe.

```
mon_graph <- ggplot(data = quakes)
```

Nous pouvons ajouter des couches à un objet `ggplot` avec l'opérateur `+`.

```
mon_graph <- mon_graph + geom_point(mapping = aes(x = mag, y = stations))
```

Il est possible d'observer le contenu de l'objet.

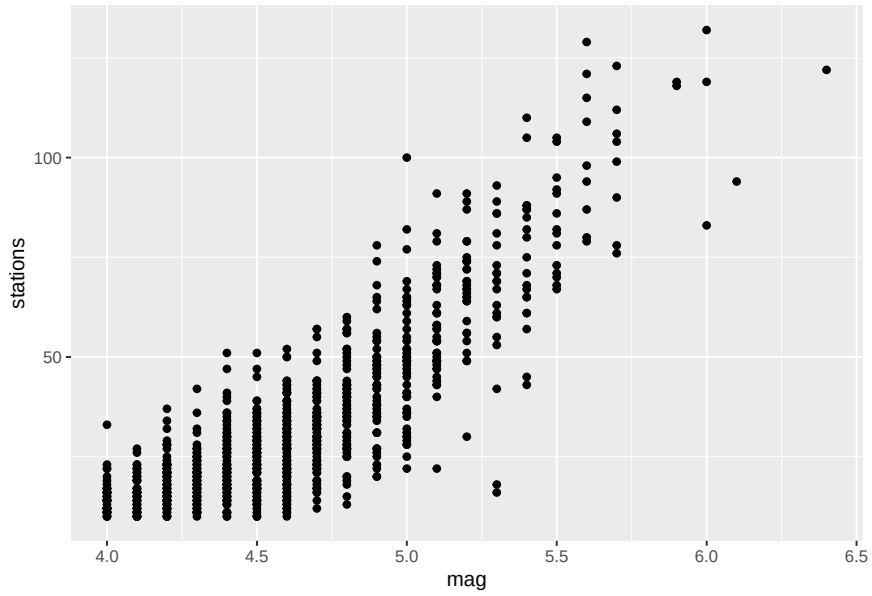
```
str(mon_graph)  # non évalué, car la sortie est longue
```

```
summary(mon_graph)
```

```
## data: lat, long, depth, mag, stations, mag_catego, region [1000x7]
## faceting: <ggproto object: Class FacetNull, Facet, gg>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map_data: function
##     params: list
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train_scales: function
##     vars: function
##     super:  <ggproto object: Class FacetNull, Facet, gg>
## -----------------------------------
## mapping: x = ~mag, y = ~stations
## geom_point: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
```

C'est en affichant une objet de classe `ggplot` qu'un graphique est produit.

```
mon_graph # ou print(mon_graph)
```



La fonction `ggsave` permet quant à elle d'enregistrer un graphique `ggpot2`.

```
ggsave(file = "ExempleGraphique_ggplot2.pdf", plot = mon_graph)
```

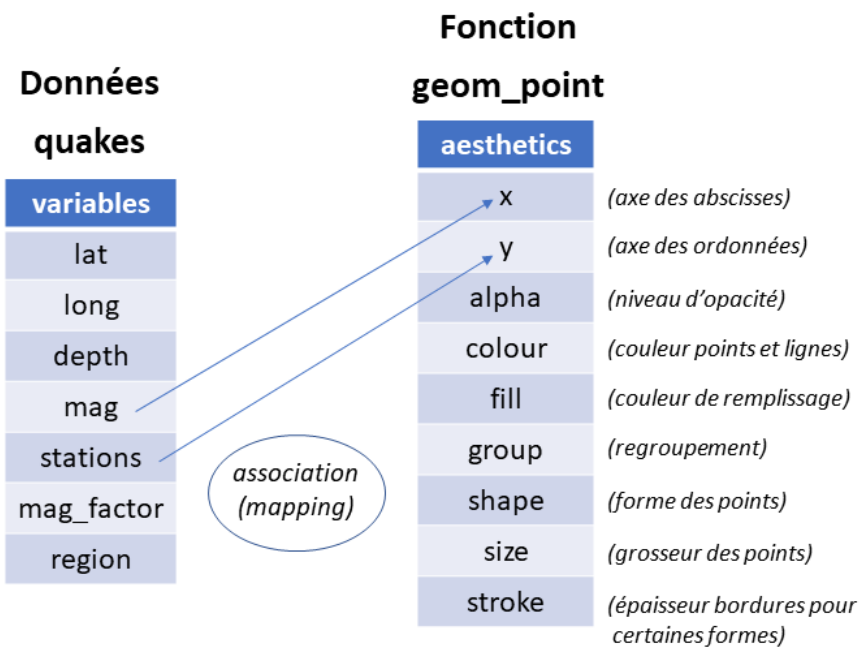
Le format du fichier à créer (p. ex. PDF, PNG, EPS, etc.) est déduit de l'extension du fichier si l'argument `device` n'est pas spécifié. Aussi, si aucune valeur n'est fourni à l'argument `plot`, c'est le dernier graphique `ggpot2` produit qui est enregistré. La dimension du graphique enregistré peut être contrôlée à l'aide des arguments `width`, `height` et `units`.

2.4 Démarche de création d'un graphique `ggplot2`

Voici une suggestion de démarche à suivre lors de la création d'une graphique `ggplot2`.

1. **Planifier** le travail - Répondre aux questions suivantes :
   a) Je veux représenter **quelles variables**, de **quel jeu de données** ?
   b) Je veux créer **quel type de graphique** ?
   c) **Quelle fonction de type `geom_*`** me permettra de produire les éléments géométriques de ce graphique ?
   d) Cette fonction accepte **quelles propriétés visuelles** ? (voir http://ggplot2.tidyverse.org/reference/)
   e) Je veux **associer** les variables concernées à quelles propriétés visuelles ?
2. Écrire et soumettre la **première version** du code de création du graphique.
3. Modifier le code de création du graphique pour **ajuster la mise en forme** (titre, nom d'axes, autres annotations, palette de couleur, etc.) selon mes besoins,
   - travail itératif : cycle « modification code » → « jugement du graphique créé » répété jusqu'à l'obtention d'un résultat satisfaisant.

La planification effectuée avant de produire le premier exemple de graphique `ggplot` pourrait être représentée comme suit.



Pour réaliser cette planification, il a fallu obtenir de la fiche d'aide de la fonction `geom_point` la liste des propriétés visuelles qu'elle accepte.

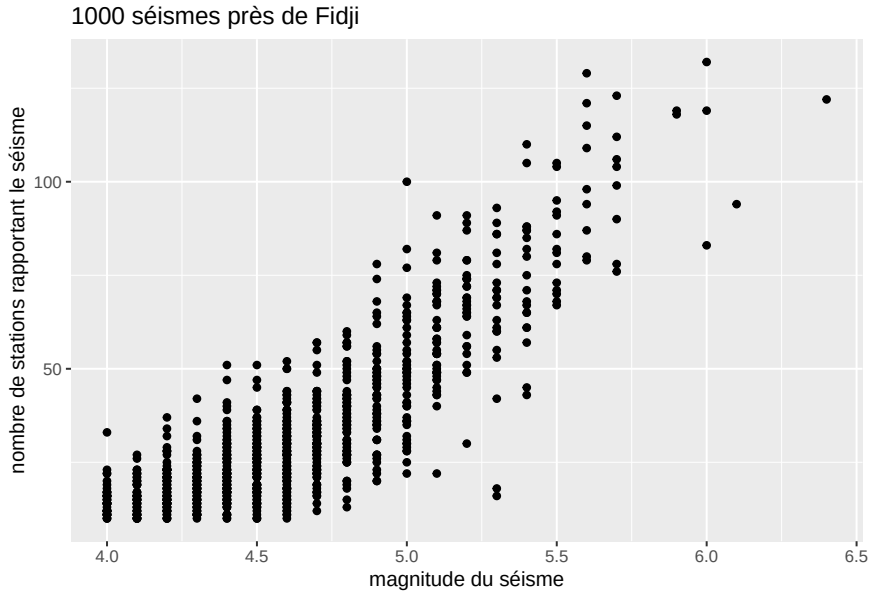
3 Quelques exemples

3.1 Exemples de configurations graphiques simples

3.1.1 Ajout d'un titre et de noms d'axes - fonction `labs`

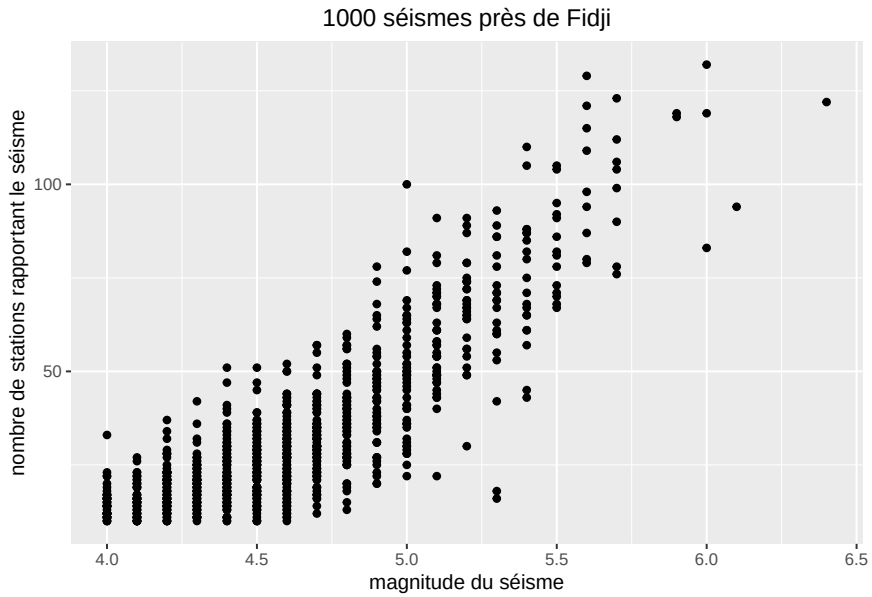
Ajoutons un titre et des noms d'axes à note premier graphique `ggplot`.

```
ggplot(data = quakes) +
  geom_point(mapping = aes(x = mag, y = stations)) +
  labs(
    title = "1000 séismes près de Fidji",          # ou ggtitle("...") +
    x = "magnitude du séisme",                     # ou xlab("...") +
    y = "nombre de stations rapportant le séisme"  # ou ylab("...")
  )
```



Vous avez envie de centrer le titre du graphique ? Voici comment faire.

```
ggplot(data = quakes) +
  geom_point(mapping = aes(x = mag, y = stations)) +
  labs(
    title = "1000 séismes près de Fidji",
    x = "magnitude du séisme",
    y = "nombre de stations rapportant le séisme"
  ) +
  theme(plot.title = element_text(hjust = 0.5))  # permet de centrer le titre
```



Références :

- Ajout d'un titre et de noms d'axes :
  - https://ggplot2.tidyverse.org/reference/labs.html
- Modification de l'alignement du titre :

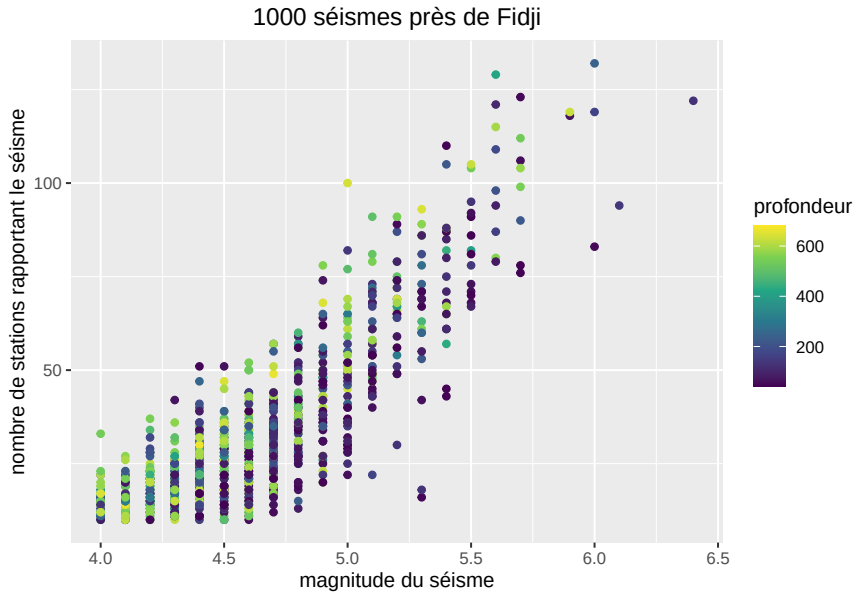
  - https://ggplot2.tidyverse.org/reference/theme.html
  - https://ggplot2.tidyverse.org/reference/element.html

3.1.2 Ajout d'une variable associée à une propriété visuelle autre qu'un axe

Le graphique précédent représente deux variables. Ajoutons une troisième variable au graphique, qui fera varier la couleur des points. Si la palette de couleur utilisée par défaut ne nous plaît pas, nous pouvons la changer.

```
scatterplot <- ggplot(data = quakes) +
  geom_point(mapping = aes(
    x = mag,
    y = stations,
    colour = depth  # permet de faire varier la couleur des points en fonction de depth
  )) +
  labs(
    title = "1000 séismes près de Fidji",
    x = "magnitude du séisme",
    y = "nombre de stations rapportant le séisme",
    colour = "profondeur"  # permet de modifier le titre de la légende
  ) +
  theme(plot.title = element_text(hjust = 0.5)) +
  scale_colour_viridis_c()       # permet d'utiliser la palette de couleur viridis

scatterplot
```



Référence :

- Modification de la palette de couleur :
  - https://ggplot2.tidyverse.org/reference/scale_colour_continuous.html

3.1.3 Échelles de couleurs - fonctions de type `scale_colour_*` et `scale_fill_*`

Deux propriétés visuelles (*aesthetics*) permettent de spécifier des couleurs :

- `colour` = couleur de points et de lignes,
- `fill` = couleur de remplissage.

Les fonctions permettant de contrôler les palettes de couleurs utilisées pour ses propriétés visuelles ont toutes un nom débutant par `scale_*` suivi du nom de la propriété visuelle concernée et de `_*`. Le nom de la fonction peut se terminer par :

- `viridis_d` ou `viridis_c` : utiliser une palette de couleur offerte dans le package R `viridisLite`, soit discrète (`_d`) ou continue (`_c`) ;
- `brewer` ou `distiller` : utiliser une palette de couleur de `ColorBrewer` ;
- `grey` : utiliser un dégradé de gris,
- `manual` : utiliser une palette discrète spécifiée manuellement ;
- `gradient`, `gradient2` ou `gradientn` : utiliser dégradé continu spécifié manuellement ;
- etc.

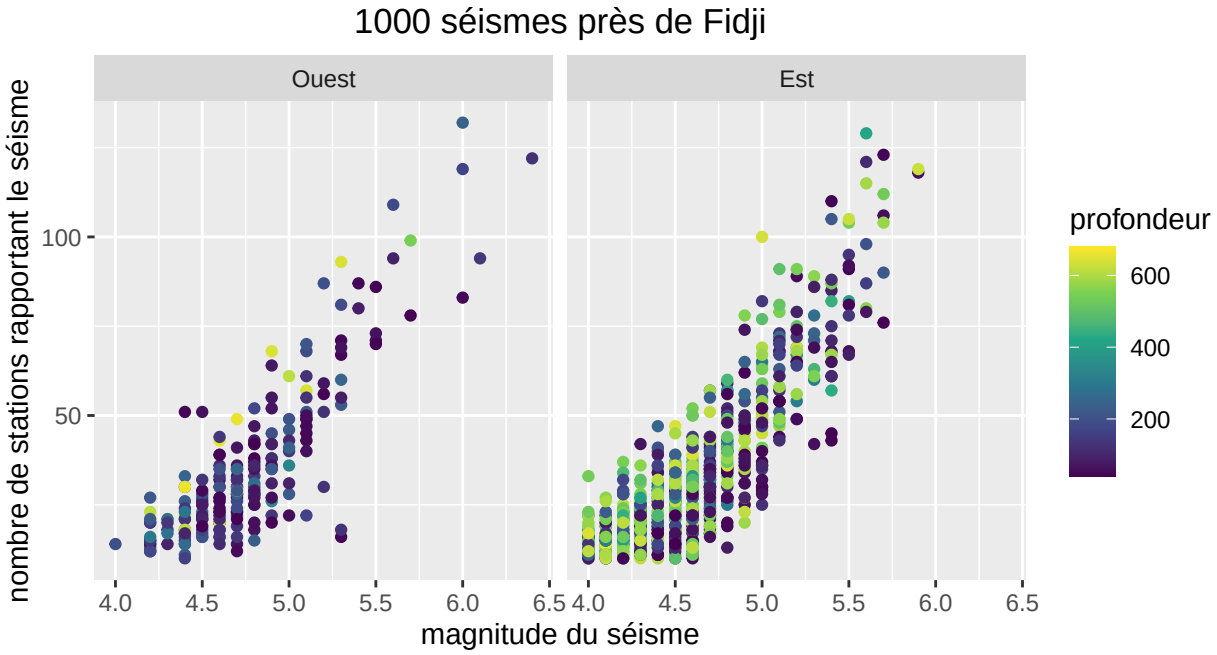
Une de ces fonctions a été utilisée dans l'exemple précédent : la fonction `scale_colour_viridis_c`.

3.1.4 Ajout de variables par l'intermédiaire d'une grille de sous-graphiques (*facets*)

Graphiques côte à côte selon les niveaux d'une variable catégorique

Ajoutons maintenant une quatrième variable au graphique, celle-ci catégorique, en créant des graphiques disjoints selon le niveau de la variable.

```
scatterplot +
  facet_wrap(facets = ~ region)   # permet de créer un sous-graphique par niveau de la variable region
```



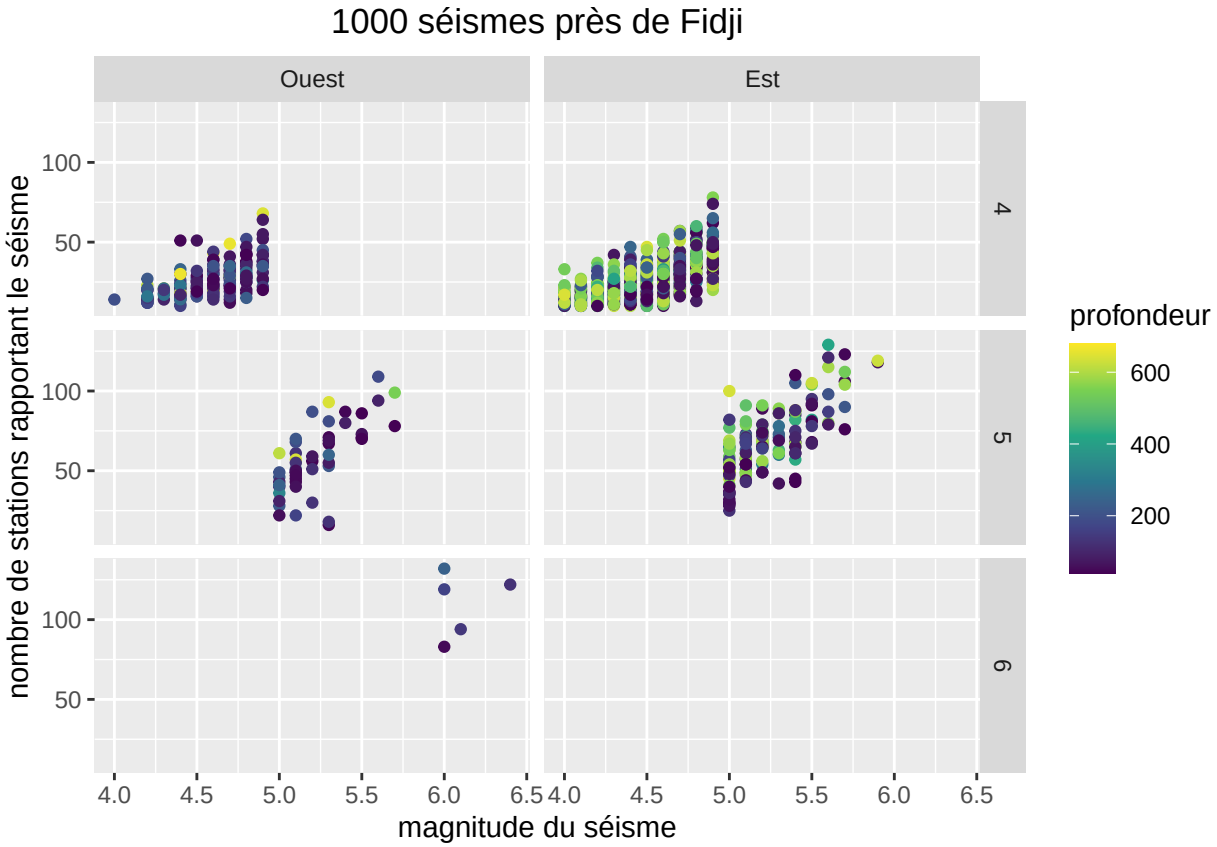
Référence :

- Utilisation de `facet_wrap` :
  - https://ggplot2.tidyverse.org/reference/facet_wrap.html

Grilles de graphiques selon les niveaux de deux variables catégoriques

Nous pourrions même facilement ajouter une cinquième variable en créant une grille à deux dimensions de sous-graphiques côte à côte.

```
scatterplot +
  facet_grid(mag_catego ~ region)  # ou facet_grid(rows = vars(mag_catego), cols = vars(region))
```



Référence :

- Utilisation de `facet_grid` :
  - https://ggplot2.tidyverse.org/reference/facet_grid.html

Remarque : Si une variable qui n'a pas été stockée sous forme de facteur doit être traitée comme une variable catégorique dans un graphique `ggplot`, il suffit d'encadrer son nom d'un appel à la fonction `factor` directement dans le code de création du graphique, comme dans l'exemple suivant.

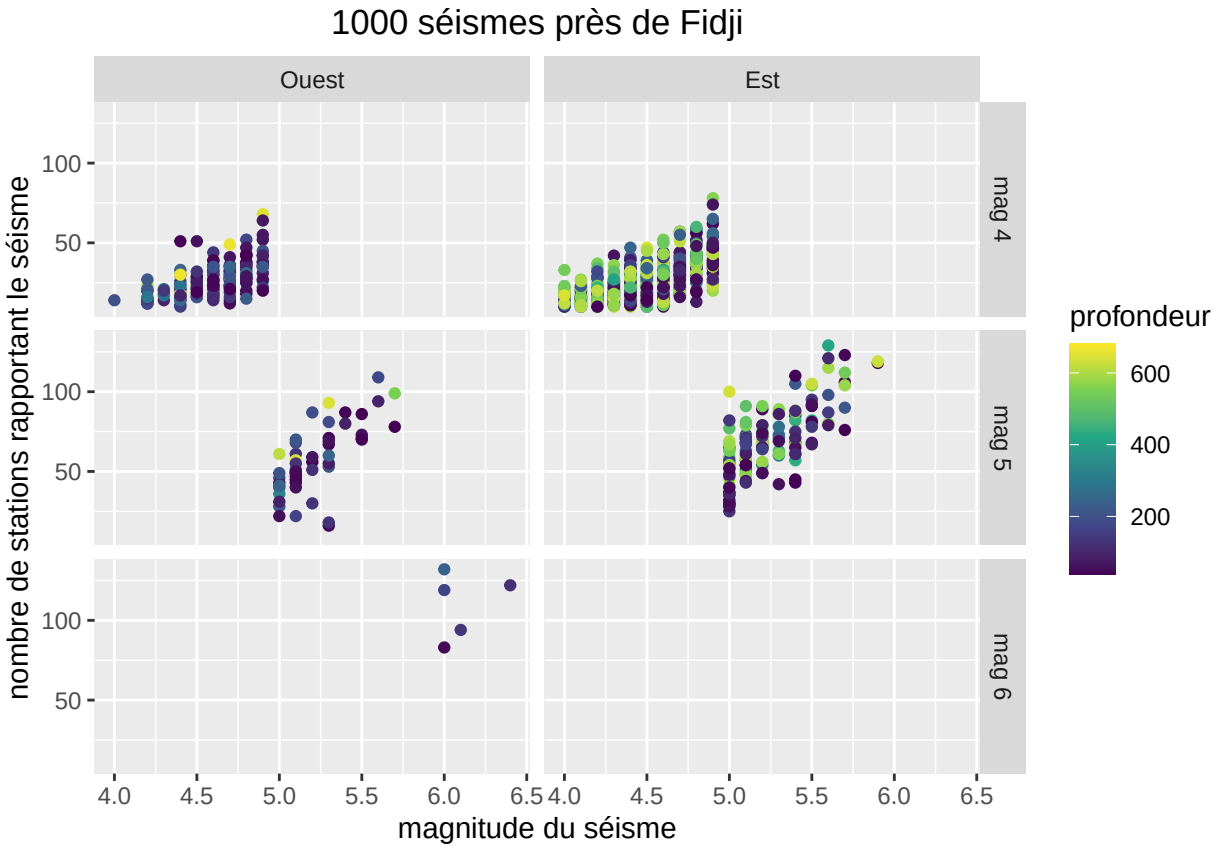
```
scatterplot +
  facet_grid(mag_catego ~ factor(region))
```

Modification des libellés des niveaux d'une variable catégorique

Les libellés des niveaux d'un facteur peuvent être modifiés ainsi.

```
# Création d'un vecteur faisant office de « lookup table », contenant l'association entre
# - les niveaux d'un facteur (nom des éléments dans le vecteur) et
# - leurs étiquettes à afficher dans le graphique (éléments du vecteur)
mag_catego_labels <- c(
  "4" = "mag 4",
  "5" = "mag 5",
  "6" = "mag 6"
)
# Production du graphique
scatterplot +
  facet_grid(
    mag_catego ~ region,
```

```
    labeller = labeller(mag_catego = mag_catego_labels)
  )
```



Références :

- Modification des libellés des niveaux d'un facteur :
  - https://ggplot2.tidyverse.org/reference/labeller.html
  - https://ggplot2.tidyverse.org/reference/labellers.html

3.2 Exemples de graphiques de différents types

Voici quelques exemples faisant intervenir divers types de graphiques, pour illustrer des possibilités de `ggplot2`. Si vous souhaitez voir encore plus d'exemples de code de création de graphique avec `ggplot2`, le web en regorge. Je recommande particulièrement la ressource suivante :

- https://evamaerey.github.io/ggplot_flipbook/ggplot_flipbook_xaringan.html#1

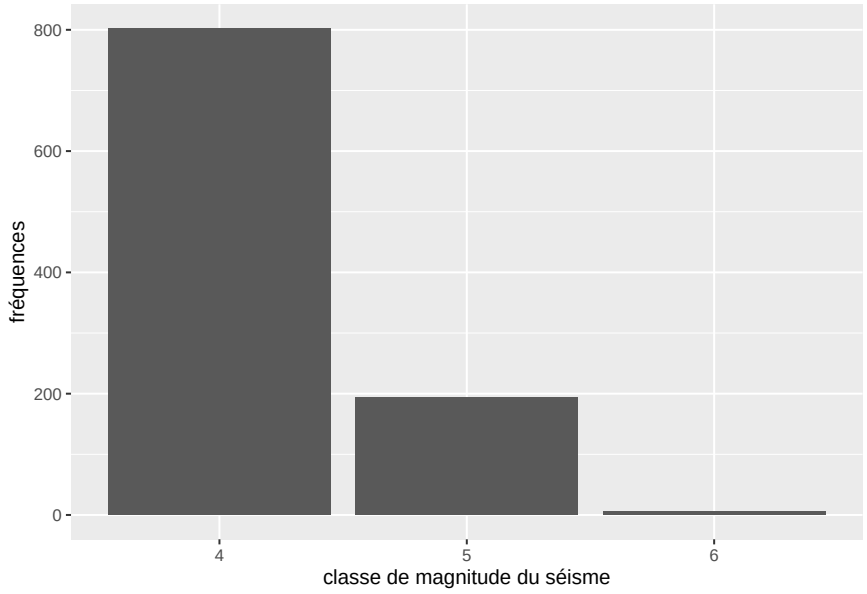
Il s'agit d'un *flipbook* très instructif, qui permet de visualiser l'impact de chaque petit ajout au code de création du graphique.

3.2.1 Diagramme à barres - fonctions `geom_bar` et `geom_col`

La fonction `geom_bar` calcule les fréquences des niveaux de facteurs et produit des diagrammes à barres. Avec cette fonction, pas besoin d'utiliser `table` ou une aute fonction similaire pour calculer les fréquences.

```
ggplot(data = quakes) +
  geom_bar(mapping = aes(x = mag_catego)) +
  labs(
```

```
    x = "classe de magnitude du séisme",
    y = "fréquences"
  )
```

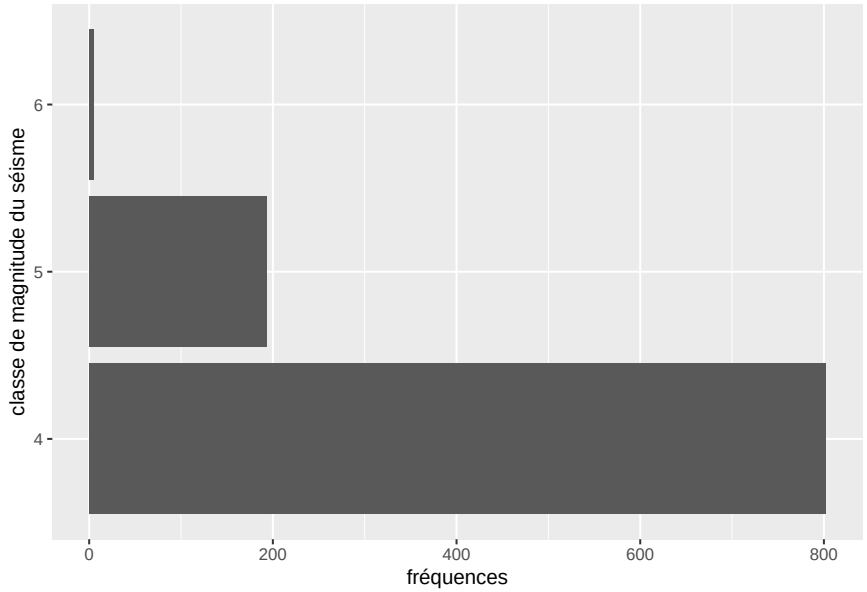


Si nous avons déjà en main les fréquences, il faut utiliser la fonction `geom_bar` avec l'argument `stat = "identity"`, ou encore la fonction `geom_col`, comme suit.

```
quakes_mag <- as.data.frame(xtabs(~ mag_catego, data = quakes))
quakes_mag
```

```
##   mag_catego Freq
## 1          4  802
## 2          5  193
## 3          6    5
```

```
ggplot(data = quakes_mag) +
  geom_col(mapping = aes(x = mag_catego, y = Freq)) +  # aesthetic y ajoutée
  coord_flip() +                                       # permet d'inverser les axes
  labs(
    x = "classe de magnitude du séisme",
    y = "fréquences"
  )
```



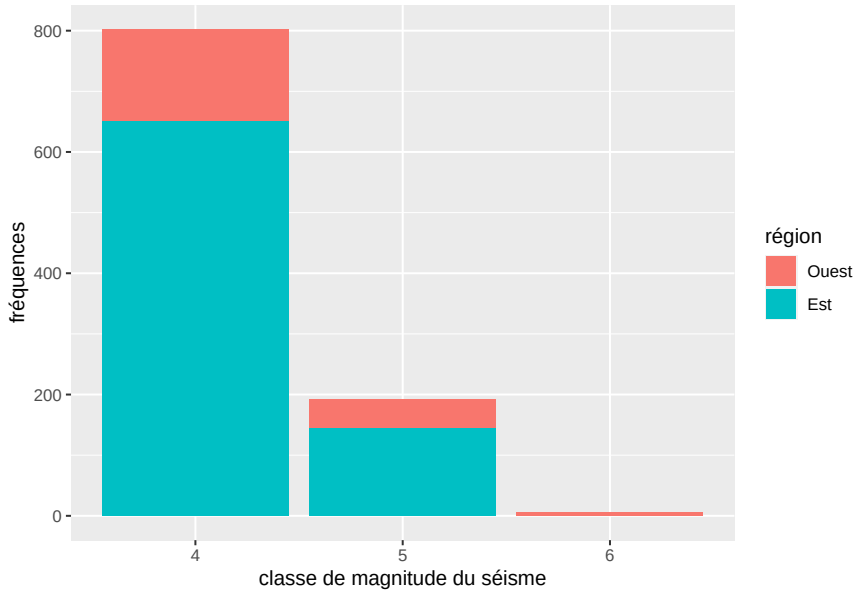
Un appel à la fonction `coord_flip` inverse les deux axes et permet d'obtenir des barres horizontales plutôt que verticales.

La fonction `geom_text` permet d'ajouter des annotations textuelles dans le graphique.

```
ggplot(data = quakes, mapping = aes(x = mag_catego)) + # aesthetics communs ici
  geom_bar() +
  geom_text(
    mapping = aes(label = after_stat(count)),          # texte ajouté = fréquences
    colour = "blue",
    stat = "count",                                    # calcul des fréquences demandé ici
    vjust = -0.2                                       # ajustement vertical
  ) +
  labs(
    x = "classe de magnitude du séisme",
    y = "fréquences"
  )
```

Pour représenter des fréquences croisées, on peut ajouter une variable catégorique au graphique via l'argument `fill`.

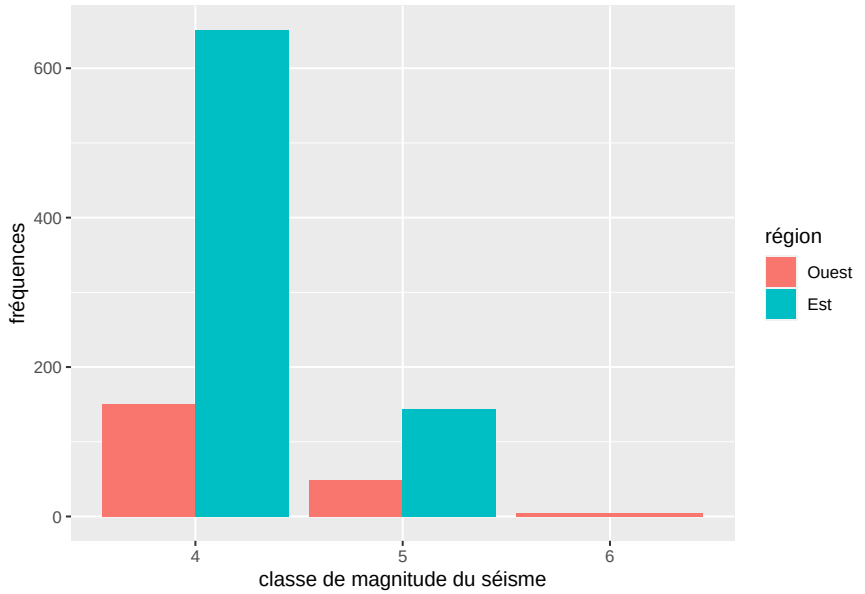
```
ggplot(data = quakes) +
  geom_bar(mapping = aes(x = mag_catego, fill = region)) +  # aesthetic fill ajoutée
  labs(
    x = "classe de magnitude du séisme",
    y = "fréquences",
    fill = "région"
  )
```



Pour avoir des bâtons groupés plutôt qu'empilés, il faut modifier la valeur de l'argument `position`.

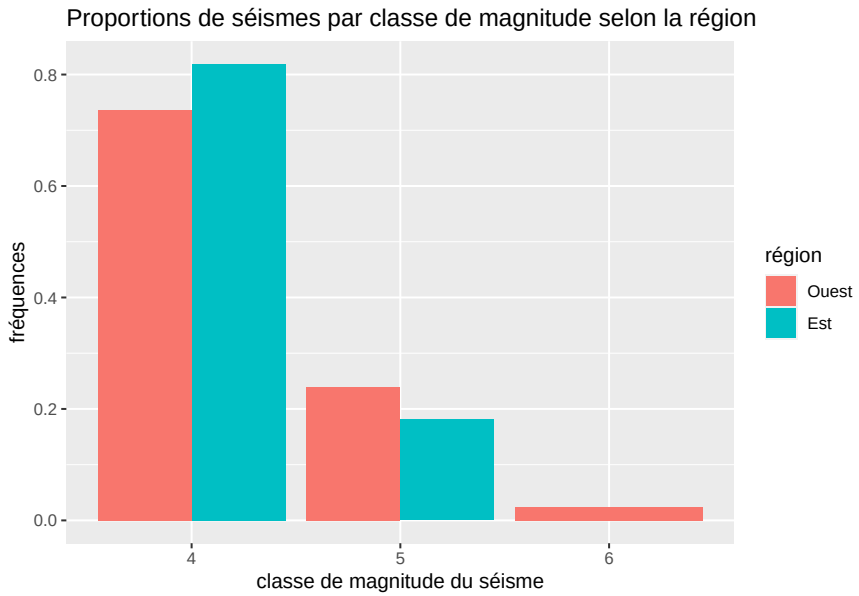
```
ggplot(data = quakes) +
  geom_bar(
    mapping = aes(x = mag_catego, fill = region),
```

```
    position = "dodge"                              # position des bâtons modifiée
  ) +
  labs(
    x = "classe de magnitude du séisme",
    y = "fréquences",
    fill = "région"
  )
```



Afin de présenter des fréquences relatives plutôt que brutes.

```
ggplot(data = quakes) +
  geom_bar(
    mapping = aes(
      x = mag_catego,
      y = after_stat(prop),  # permet le calcul de fréquences relatives (proportions)
      group = region,        # pour avoir les fréq. relatives de mag conditionnelles à region
      fill = region),
    position = "dodge"
  ) +
  labs(
    title = "Proportions de séismes par classe de magnitude selon la région",
    x = "classe de magnitude du séisme",
    y = "fréquences",
    fill = "région"
  )
```



Références :

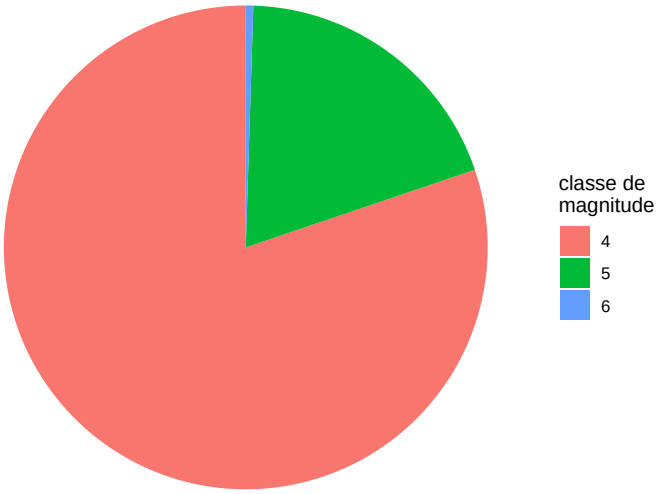
- https://ggplot2.tidyverse.org/reference/geom_bar.html
- https://ggplot2.tidyverse.org/reference/coord_flip.html
- https://ggplot2.tidyverse.org/reference/geom_text.html

3.2.2 Diagramme en secteurs - coordonnées polaires avec `coord_polar`

Les auteurs de `ggplot2` n'offrent pas de fonction conviviale pour la création de diagrammes en secteurs, probablement parce qu'ils ne recommandent pas leur utilisation. Malgré tout, tenter de tracer un diagramme en secteurs avec `ggplot2` aide à comprendre davantage les possibilités du package.

Pour produire un diagramme en secteurs avec `ggplot2`, il faut d'abord produire un diagramme à barres empilées, puis demander l'utilisation d'un système de coordonnées polaires par un appel à la fonction `coord_polar`.

```
ggplot(data = quakes) +
  geom_bar(mapping = aes(x = 1, fill = mag_catego)) +
  coord_polar(theta = "y") +                          # coordonnées polaires
  theme_void() +                                      # thème vide
  labs(fill = "classe de\nmagnitude")
```



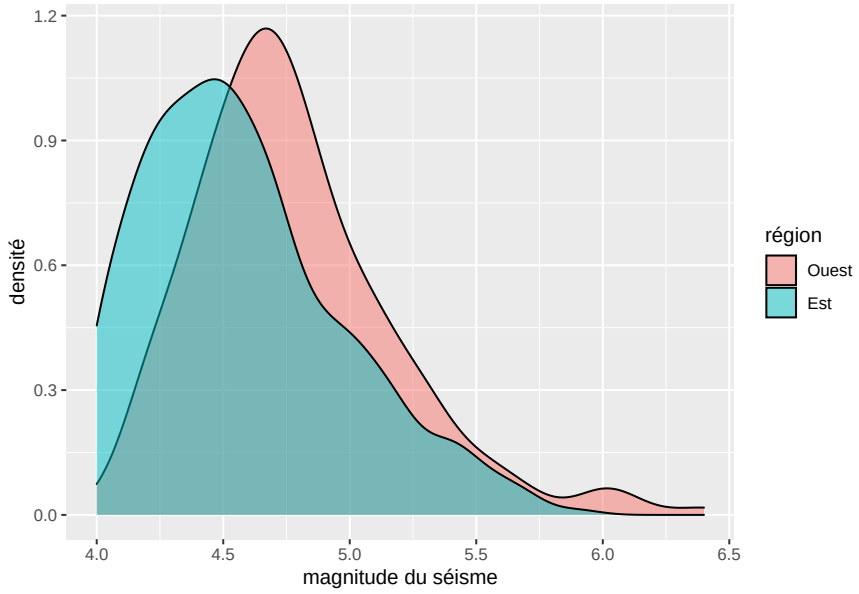
Références :

- https://ggplot2.tidyverse.org/reference/coord_polar.html

3.2.3 Courbes de densité à noyau superposées - fonction `geom_density`

Il est facile avec `ggplot2` de superposer des histogrammes ou des courbes de densités à noyau (comme ci-dessous).

```
ggplot(data = quakes) +
  geom_density(
    mapping = aes(x = mag, fill = region),
    alpha = 0.5  # niveau d'opacité (0 = transparent, 1 = complètement opaque)
  ) +
  labs(
    x = "magnitude du séisme",
    y = "densité",
    fill = "région"
  )
```



Référence :

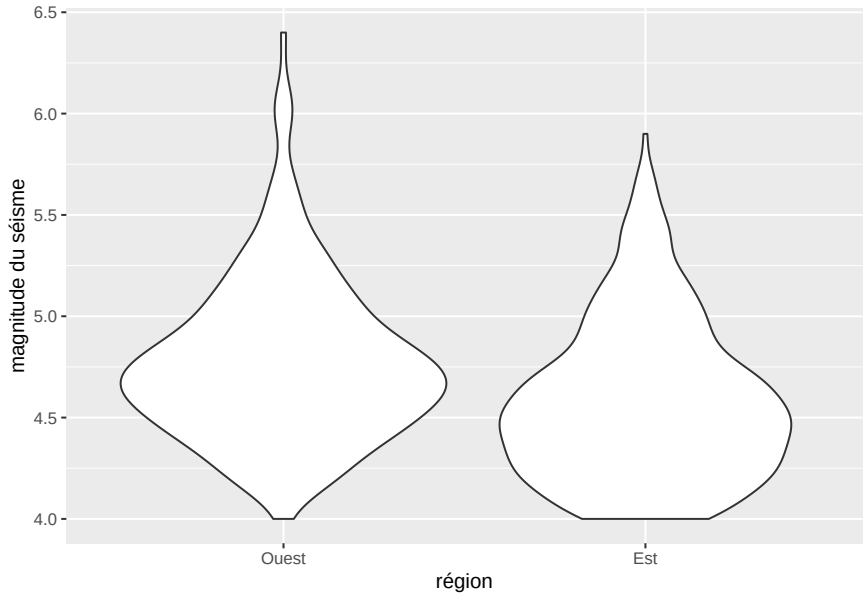
- https://ggplot2.tidyverse.org/reference/geom_density.html

3.2.4 Diagrammes en violons juxtaposés - fonction `geom_violin`

Un autre outil, dérivé des courbes de densités à noyau, permettant d'explorer l'association potentielle entre une variable numérique et d'une variable catégorique est le diagramme en violon (*violin plot*).

```
violinplots <- ggplot(data = quakes) +
  geom_violin(mapping = aes(x = region, y = mag)) +
  labs(
    x = "région",
    y = "magnitude du séisme"
  )

violinplots
```



Référence :

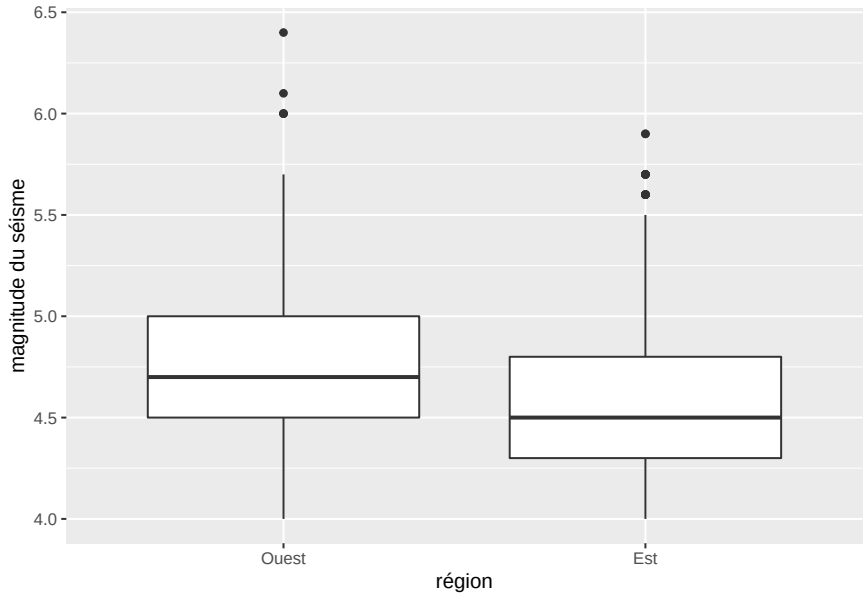
- https://ggplot2.tidyverse.org/reference/geom_violin.html

3.2.5 Diagrammes en boîtes juxtaposés - fonction `geom_boxplot`

Le diagramme en violon est en fait d'un mélange entre les courbes de densité à noyau et les diagrammes en boîtes (*boxplots*).

```
boxplots <- ggplot(data = quakes) +
  geom_boxplot(mapping = aes(x = region, y = mag)) +
  labs(
    x = "région",
    y = "magnitude du séisme"
  )

boxplots
```



Référence :

- https://ggplot2.tidyverse.org/reference/geom_boxplot.html

3.3 Exemples divers plus poussés

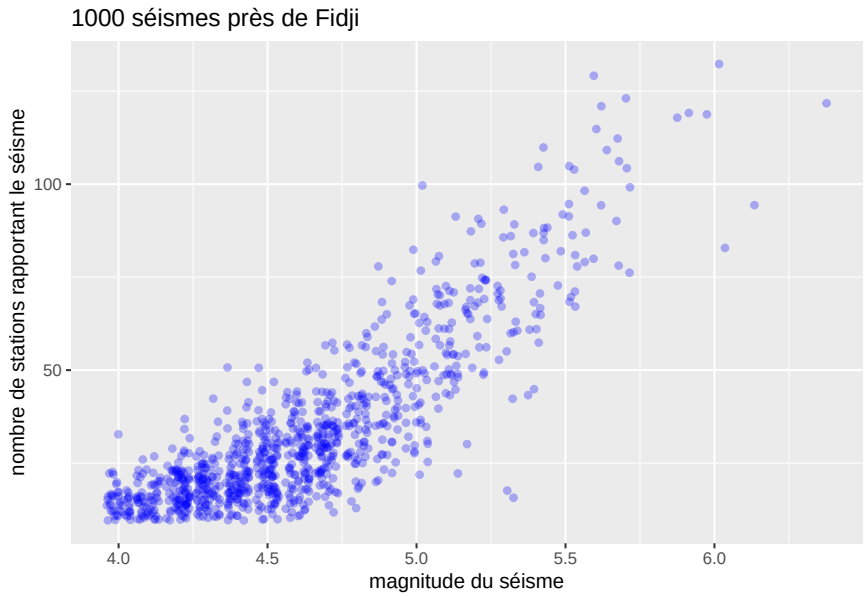
3.3.1 Solutions aux observations superposées (*overplotting*)

Dans le diagramme de dispersion du nombre de stations rapportant le séisme en fonction de la magnitude du séisme (premier exemple), plusieurs points représentent en fait plus d'une observation. Cet *overplotting* est dû au fait que la variable `stations` ne prend que des valeurs entières (il s'agit d'un dénombrement) et la précision de mesure de la variable `mag` est limitée au dixième.

3.3.1.1 Solution 1 : Ajout de *jitter* aux observations

Ajouter du *jitter* à des observations signifie de leur ajouter de petites quantités aléatoires, parfois appelées du bruit blanc.

```
ggplot(data = quakes) +
  geom_jitter(                                # geom_jitter est utilisé au lieu de geom_point
    mapping = aes(x = mag, y = stations),
    colour = "blue",
    alpha = 0.3
  ) +
  labs(
    title = "1000 séismes près de Fidji",
    x = "magnitude du séisme",
    y = "nombre de stations rapportant le séisme"
  )
```



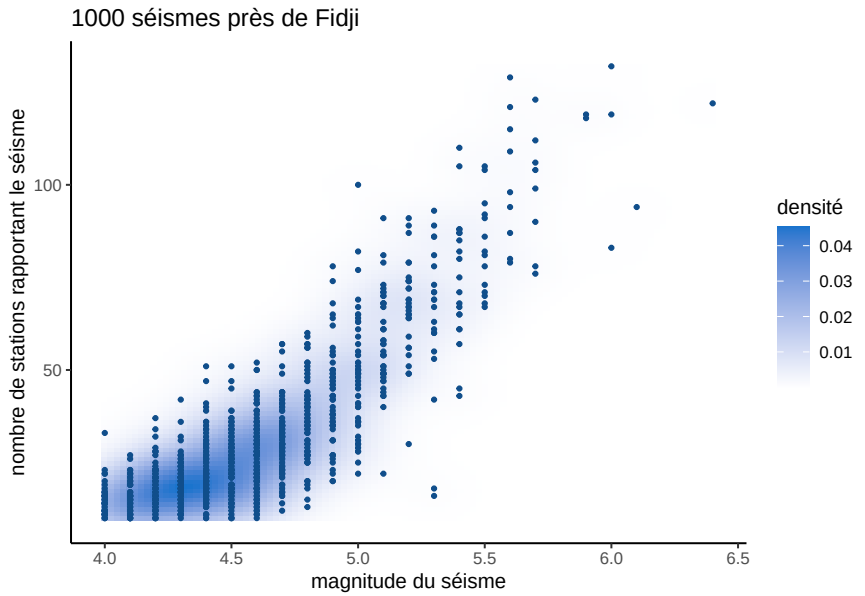
Référence :

- Utilisation de `geom_jitter` :
  - https://ggplot2.tidyverse.org/reference/geom_jitter.html

3.3.1.2 Solution 2 : Estimation à noyau de densité 2D

Nous pourrions ajouter en arrière-plan une estimation à noyau de la densité bivariée entre les deux représentées.

```
ggplot(
  data = quakes,
  mapping = aes(x = mag, y = stations)              # aesthetics communs placées ici
) +
  stat_density_2d(                                  # ajout de l'estimation de densité 2D
    aes(fill = after_stat(density)),
    geom = "tile",
    contour = FALSE
  ) +
  geom_point(
    shape = 20,                                     # pour modifier le symbole des points
    colour = "dodgerblue4"
  ) +
  labs(
    title = "1000 séismes près de Fidji",
    x = "magnitude du séisme",
    y = "nombre de stations rapportant le séisme",
    fill = "densité"
  ) +
  scale_fill_gradient(                              # pour modifier l'échelle de couleur
    low = "white",
    high = "dodgerblue3"
  ) +
  theme_classic()                                   # pour avoir rapidement un fond blanc
```



Plus la couleur de fond est foncée, plus la densité de points est forte dans la région.

Références :

- Densité 2D :
  - https://ggplot2.tidyverse.org/reference/geom_density_2d.html
  - https://www.inwt-statistics.com/read-blog/smoothscatter-with-ggplot2-513.html
- Type de points :
  - https://ggplot2.tidyverse.org/reference/aes_linetype_size_shape.html
- Thèmes complets :
  - https://ggplot2.tidyverse.org/reference/ggtheme.html

3.3.1.3 Autres solutions

D'autres solutions sont proposées ici : https://ggplot2.tidyverse.org/reference/geom_point.html#overplotting

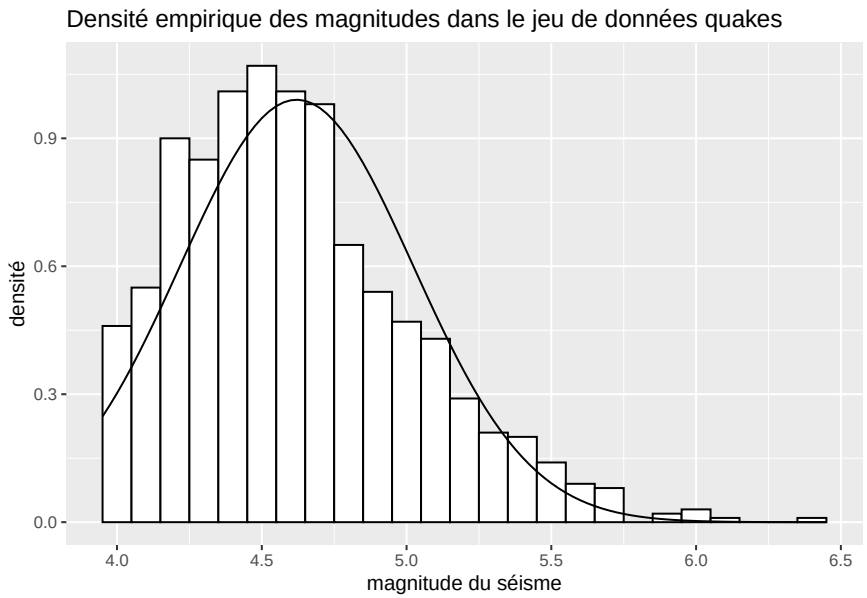
3.3.2 Représentation graphique d'une fonction

La fonction `stat_function` permet de représenter une fonction mathématique, un peu comme le fait la fonction `curve` du système graphique de base.

```
# Statistique dont nous aurons besoin
moy <- mean(quakes$mag)
et <- sd(quakes$mag)
```

```
histogramme <- ggplot(data = quakes, mapping = aes(x = mag)) +
  geom_histogram(
    mapping = aes(y = after_stat(density)),
    binwidth = 0.1,
    colour = "black",
    fill = "white"
  ) +
  labs(
    title = "Densité empirique des magnitudes dans le jeu de données quakes",
    x = "magnitude du séisme",
    y = "densité"
  )
```

```
histogramme + stat_function(           # ajout d'une courbe de densité normale
    fun = dnorm,
    args = list(mean = moy, sd = et),  # avec paramètres estimés à partir des données
    xlim = c(3.95, 6.45)
  )
```



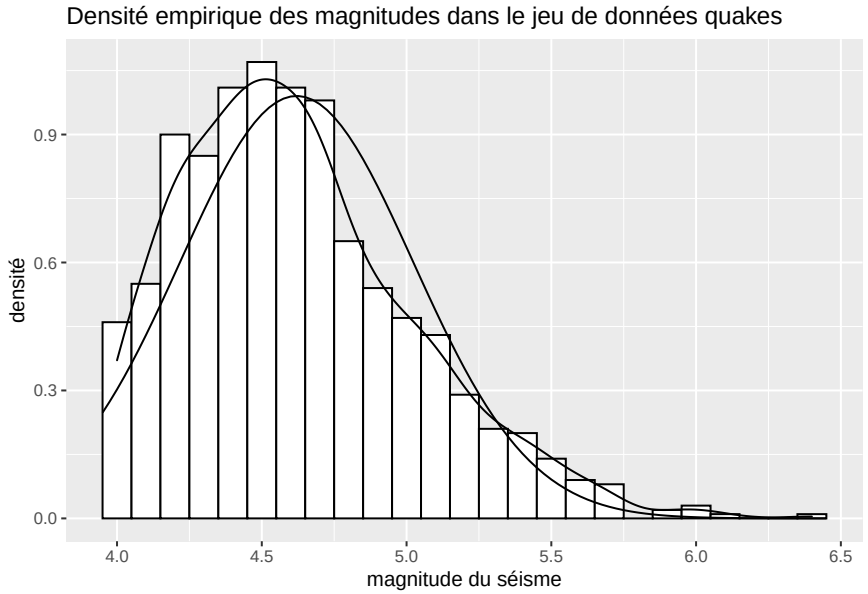
Référence :

- https://ggplot2.tidyverse.org/reference/geom_histogram.html
- https://ggplot2.tidyverse.org/reference/stat_function.html

3.3.3 Mise en forme d'une légende

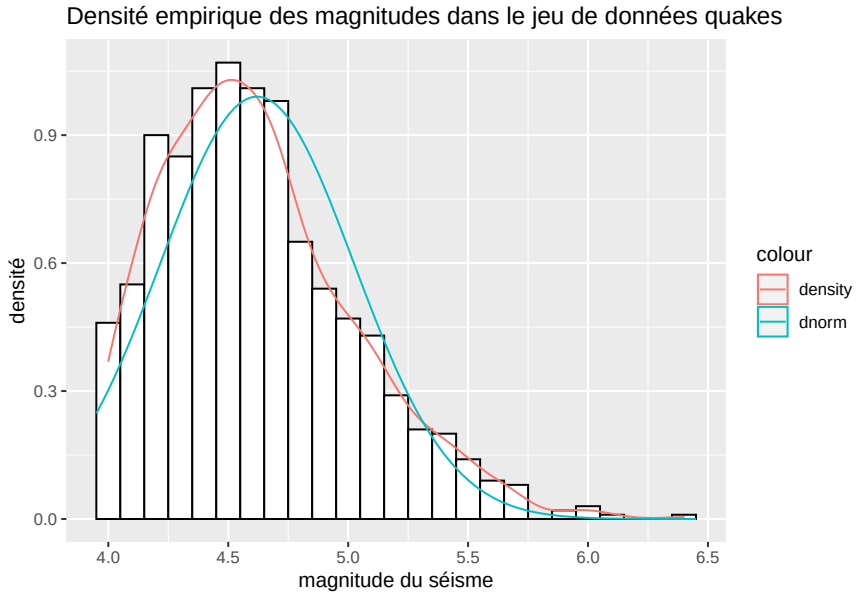
Ajoutons une courbe supplémentaire au graphique précédent.

```
histogramme +
  geom_density() +  # ajout d'une courbe d'estimation de densité à noyau
  stat_function(    # ajout d'une courbe de densité normale
    fun = dnorm,
    args = list(mean = moy, sd = et),
    xlim = c(3.95, 6.45)
  )
```



Afin de pouvoir identifier chacune des courbes, il faudrait choisir une propriété visuelle pour les distinguer. Choisissions la couleur.

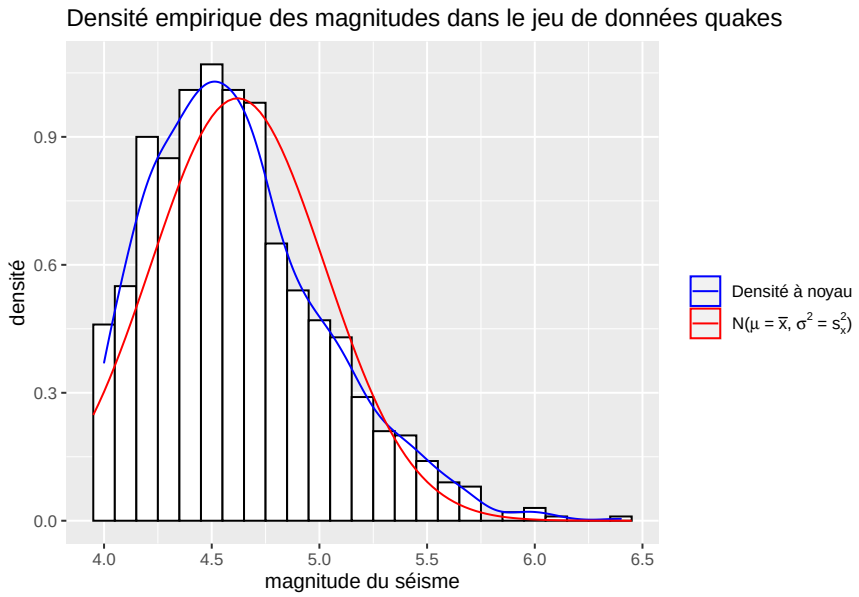
```
histogramme <- histogramme +
  geom_density(
    aes(colour = "density"),  # association de la propriété visuelle colour à une valeur
  ) +
  stat_function(
    aes(colour = "dnorm"),    # association de la propriété visuelle colour à une valeur
    fun = dnorm,
    args = list(mean = moy, sd = et),
    xlim = c(3.95, 6.45)
  )
histogramme
```



Dans les appels aux fonctions ajoutant les courbes, nous avons associé la propriété visuelle `colour` à une valeur fixe (et non à une variable comme nous avons l'habitude de le faire). Cet ajout a eu pour effet de provoquer un ajout automatique d'une légende. Les valeurs associées à `colour` se sont retrouvées par défaut dans la légende.

Modifions les couleurs ainsi que les étiquettes des courbes et retirons le titre de la légende.

```
histogramme + scale_colour_manual(
    name = "",                                          # pour retirer le titre de la légende
    values = c("density" = "blue", "dnorm" = "red"),    # pour modifier les couleurs
    labels = c(                                         # pour modifier les étiquettes
      "density" = "Densité à noyau",
      "dnorm" = expression(paste("N(", mu, " = ", bar(x), ", ", sigma^2, " = ",  s[x]^2, ")"))
    )
  )
```

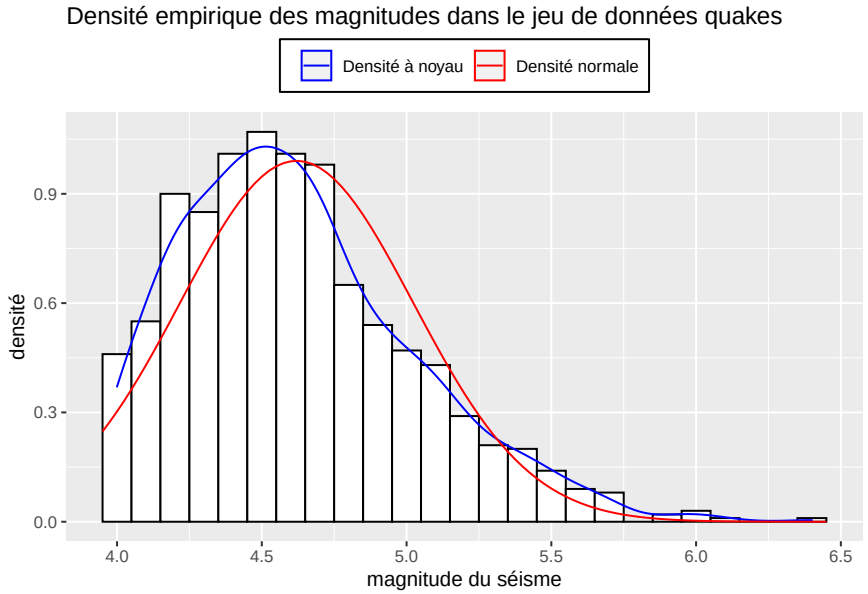


Remarque : Les annotations mathématiques sont toujours possibles avec `ggplot2`, comme dans le système graphique de base. Dans le graphique précédent, des annotations mathématiques ont été insérées dans la légende.

3.3.3.1 Configuration de l'apparence de la légende

Grâce à la fonction `theme`, nous pouvons aussi contrôler l'allure de la légende.

```
histogramme + scale_colour_manual(
    name = "",
    values = c("density" = "blue", "dnorm" = "red"),
    labels = c("density" = "Densité à noyau", "dnorm" = "Densité normale")
  ) +
  theme(
    legend.position = "top",            # modification de la position de la légende
    legend.background = element_rect(  # ajout d'une bordure rectangulaire autour de la légende
      size = 0.5, linetype = "solid", colour = "black"
    )
  )
```



Références :

- https://ggplot2.tidyverse.org/reference/scale_manual.html
- https://ggplot2.tidyverse.org/reference/theme.html
- https://ggplot2.tidyverse.org/reference/element.html
- http://www.sthda.com/french/wiki/ggplot2-legende-modifier-facilement-la-legende-d-un-graphique-logiciel-r-et-visualisation-de-donnees

4 Packages souvent utilisés avec `ggplot2`

4.1 Le `tidyverse`

Le package `ggplot2` est souvent utilisé conjointement à d'autres packages du `tidyverse`, notamment :

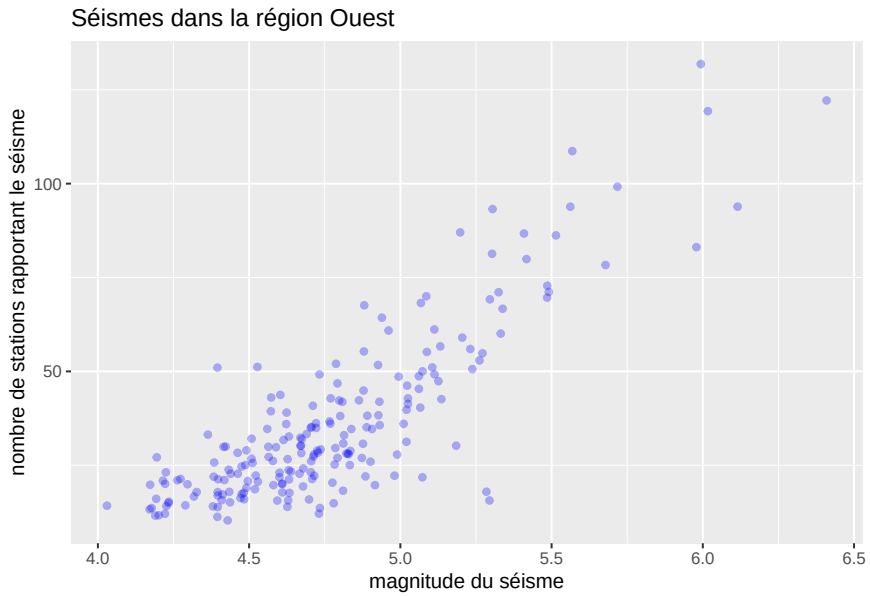
- `dplyr` pour la manipulation / le prétraitement de données ;
- `magrittr` qui offre l'opérateur « pipe » `%>%` ;
- `forcats` pour la manipulation de facteurs.

Exemple : production d'un graphique à partir d'un sous-ensemble de données

Supposons que nous voulions produire le diagramme de dispersion entre les variables `mag` et `stations` de `quakes` uniquement avec les observations dans la région Ouest. Nous pourrions nous servir de la fonction `filter` de `dplyr` et de l'opérateur `%>%` de `magrittr` de comme suit.

```
library(dplyr)

quakes %>%
  filter(region == "Ouest") %>%
  ggplot(mapping = aes(x = mag, y = stations)) +
  geom_jitter(colour = "blue", alpha = 0.3) +
  labs(
    title = "Séismes dans la région Ouest",
    x = "magnitude du séisme",
    y = "nombre de stations rapportant le séisme"
  )
```



Exemple : modification de l'ordre des niveaux d'un facteur

Nous avons vu dans les autres notes sur les graphiques en R que les niveaux d'un facteur sont présentés dans un graphique en respectant leur ordre dans les attributs du facteur. Cette affirmation est encore vraie avec `ggplot2`. Le `tidyverse` contient un package appelé `forcats` qui offre des fonctions facilitant la manipulation des facteurs. Ce package est particulièrement utile lors de la création de graphiques (avec `ggplot2` ou avec le système graphique de base).

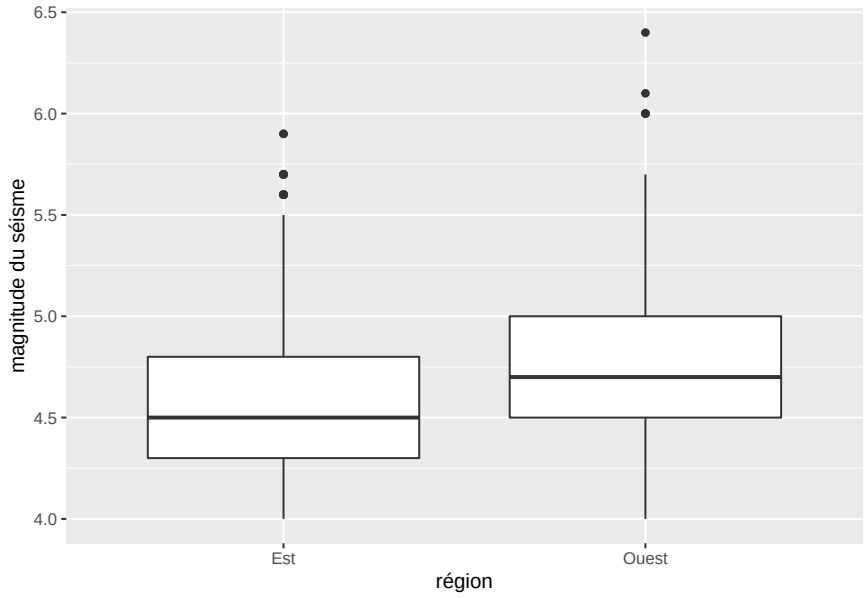
Par exemple, supposons que nous produisons des diagrammes en boîtes juxtaposés et que nous souhaitons que ceux-ci soient présentés dans l'ordre croissant des valeurs des médianes dans les diagrammes. La fonction `fct_reorder` du package `forcats` permet de réordonner les niveaux d'un facteur selon la valeur d'une statistique calculée sur une autre variable du jeu de données selon les niveaux du facteur. Par exemple, nous pourrions réordonner les niveaux du facteur `region` de `quakes` selon la médiane de la variable `mag` par région comme suit.

```
library(forcats)
region_reorder <- fct_reorder(
  .f = quakes$region,
  .x = quakes$mag,
  .fun = median
)
str(region_reorder)
```

```
##  Factor w/ 2 levels "Est","Ouest": 1 1 1 1 1 1 2 1 1 1 ...
```

Lorsqu'une fonction du package `forcats` est utilisée avec `ggplot`, elle est typiquement appelée directement dans l'appel à la fonction `aes`, comme dans cet exemple. Les noms des variables du jeu de données fourni à l'argument `data` sont alors directement accessibles, même dans l'appel à la fonction du package `forcats`.

```
ggplot(data = quakes) +
  geom_boxplot(mapping = aes(
    x = fct_reorder(.f = region, .x = mag, .fun = median),
    y = mag
  )) +
  labs(x = "région") +
  labs(y = "magnitude du séisme")
```



4.2 Extensions

Un nombre important d'extensions de `ggplot2` sont offertes. Plusieurs sont énumérées sur le site web suivant : https://exts.ggplot2.tidyverse.org/gallery/

Voici une liste d'extensions intéressantes qui ont été le sujet d'un tutoriel réalisé par d'anciens étudiants de ces cours. Des exemples d'utilisation des ces packages peuvent être visualisés dans les tutoriels.

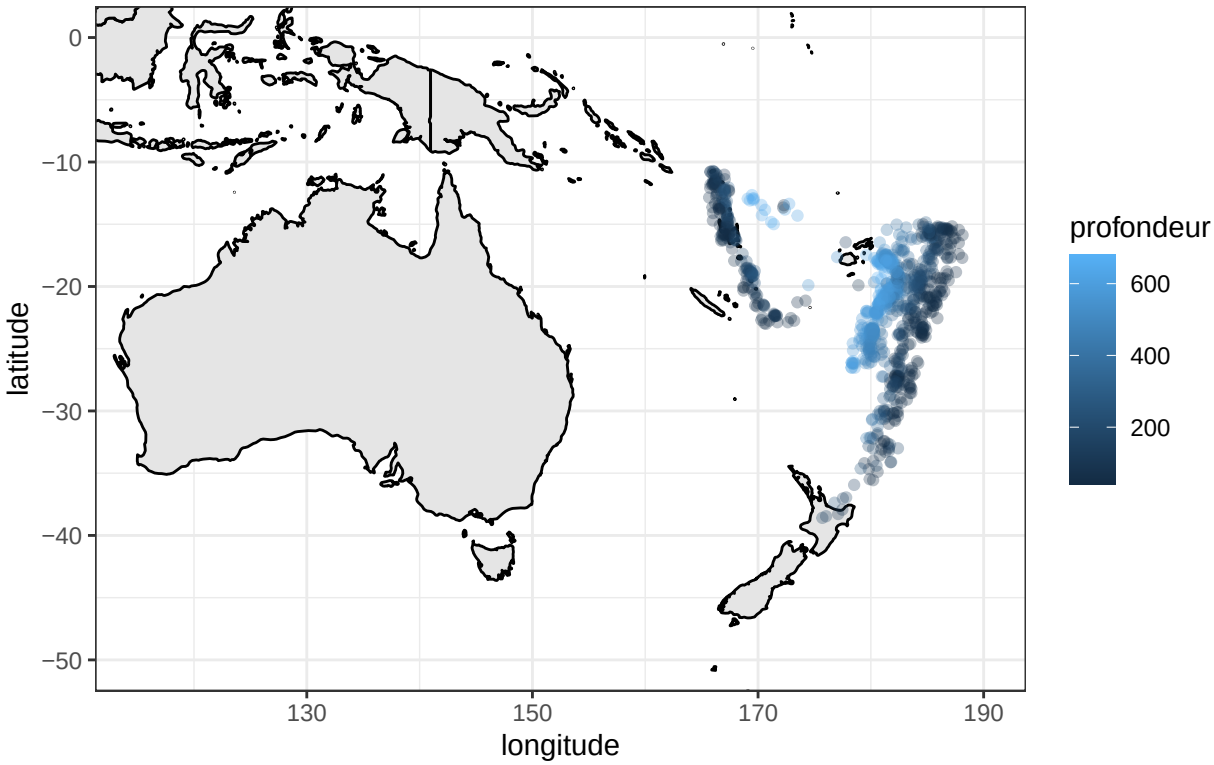
- `ggrepel` : fonction `geom_text_repel` pour éviter les textes superposés
  - tutoriel : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/ggrepel_ggthemes_paletteer/
- `ggthemes` : thèmes complets supplémentaires
  - tutoriel : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/ggrepel_ggthemes_paletteer/
- `cowplot` : thème adapté aux publications scientifiques et possibilité de combiner plusieurs graphiques en un
  - tutoriel : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2019/cowplot/
- `patchwork` : combinaison facile de plusieurs graphiques en un
  - tutoriel : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/patchwork_ggextra/
- `ggExtra` : fonction `ggMarginal` pour l'ajout facile d'histogrammes marginaux
  - tutoriel : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/patchwork_ggextra/
- `gganimate` : gif animés (ou autres formats d'animation)
  - tutoriel : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/gganimate/
- `ggiraph` : graphiques `ggplot` interactifs (format HTML)
  - tutoriel : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2018/ggiraph/

4.3 Cartes géographiques

Comme avec le système graphique de base, le package `maps` permet d'ajouter une carte en arrière-plan d'un graphique `ggplot`. Voici un exemple d'utilisation d'une carte tirée du package `maps` avec `ggplot2` :

```
library(maps)
monde <- map_data("world")  # va chercher des données provenant du package maps
ggplot() +
  geom_polygon(
    data = monde,
    aes(x = long, y = lat, group = group),
```

```
  fill = "gray90",
  col = "black"
) +
geom_point(
  data = quakes,
  aes(x = long, y = lat, colour = depth),
  alpha = .3
) +
coord_quickmap(           # ou coord_map() si le package mapproj est installé
  xlim = c(115, 190),
  ylim = c(-50, 0)
) +
labs(x = "longitude", y = "latitude", colour = "profondeur") +
theme_bw()
```



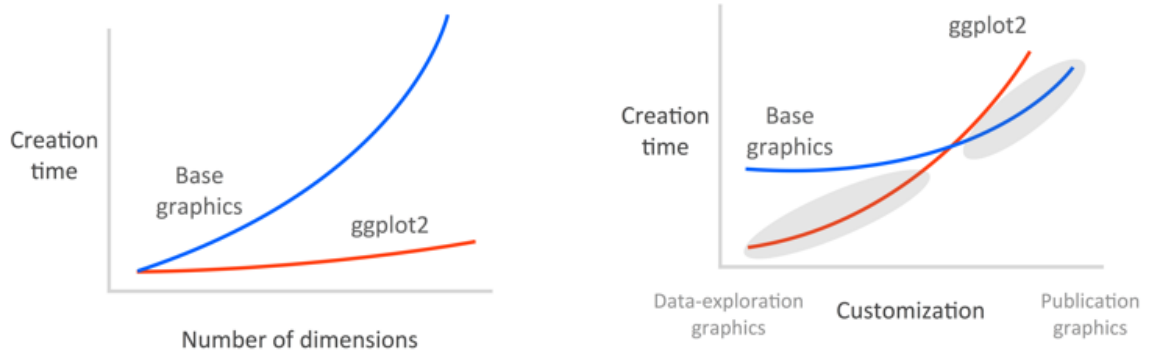
L'extension `ggmap` permet également d'intégrer une carte Google Maps (requière un compte sur https://cloud.google.com/maps-platform/ pour avoir accès aux cartes) ou Stamen Maps à un graphique produit avec `ggplot2`.

Références :

- https://ggplot2-book.org/maps.html
- http://eriqande.github.io/rep-res-web/lectures/making-maps-with-R.html

5 Comparaison entre `ggplot2` et le système graphique R de base

Entre `ggplot2` et le système graphique R de base, lequel devrions-nous utiliser ? Le graphique suivant résume bien ma réponse à cette question.



Source : http://seananderson.ca/ggplot2-FISH554/

Lorsque le nombre de variables à représenter dans un graphique est grand (*Number of dimensions* élevé), utiliser `ggplot2` peut potentiellement permettre de sauver beaucoup de temps. Les représentations multivariées sont plus faciles à produire avec `ggplot2` qu'avec le système graphique R de base. En grande partie pour cette raison, la production de graphiques pour de l'analyse exploratoire de données est souvent plus rapide à réaliser avec `ggplot2` qu'avec le système graphique de base. Malgré tout, il existe encore une situation dans laquelle le système de base surpasse `ggplot2` : la production de graphiques à mises en formes spécifiques pour des publications scientifiques. Les configurations graphiques sont souvent plus simples à réaliser avec le système de base qu'avec `ggplot2`.

---

6 Résumé

Principes de base de la **g**rammaire **g**raphique :

Graphique statistique = représentation de **données**, dans un **système de coordonnées** spécifique, divisée en éléments de base :

- **éléments géométriques** (*geoms*) : points, lignes, barres, etc. ;
- **propriétés visuelles** (*aesthetics*) des éléments géométriques : axes, couleurs, formes, tailles, etc.
- **transformations statistiques**, si désiré : courbe de régression ou de lissage, région d'erreur, etc.

Un graphique est spécifié en **associant des variables**, provenant des données, **à des propriétés visuelles** des éléments géométriques du graphique.

Survol des principales fonctions de `ggplot2`

- `ggplot` : initialisation d'un objet de classe `ggplot` ;
- `qplot` : initialisation rapide (`q` pour *quick*) d'un objet `ggplot` ;
- `+` : opérateur pour l'ajout de couches ou la modification de configurations dans un objet `ggplot` ;
- fonctions de type `geom_*` (p. ex. `geom_point`, `geom_boxplot`, `geom_bar`, etc.) : spécification de couches à ajouter à un graphique ;
- `aes` : création d'un `mapping`, soit une association entre entre des propriétés visuelles et des variables ;
- `ggsave` : enregistrement d'un graphique.

Produire un graphique = afficher un objet `ggplot`

Gabarit minimaliste de code de création d'un graphique `ggplot2` :

```
ggplot(data = <DATA>) +
  <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Quelques fonctions de type `geom_*`

Fonction	Type de graphique	Élément(s) ajouté(s)
`geom_point`	diagramme de dispersion	points selon des coordonnées
`geom_line`	diagramme en lignes	segments de droites reliant des points
`geom_bar`	diagramme à barres	barres disjointes, de hauteurs spécifiées ou calculées = fréquences des niveaux d'un facteur
`geom_histogram`	histogramme	barres collées, de hauteurs calculées = fréquences d'observations d'une variable numérique tombant dans des intervalles joints (*bin*)
`geom_boxplot`	diagramme en boîte	*boxplots*
`geom_density`	courbe de densité à noyau	courbe de la densité estimée par noyau (*kernel density*)
`geom_qq`	diagramme quantile-quantile théorique	points pour les couples de quantiles empiriques et théoriques
⋮	il en existe plusieurs autres	voir http://ggplot2.tidyverse.org/reference/

La plupart de ces fonctions cachent des transformations statistiques (p. ex. `geom_bar` et `geom_histogram` calculent des fréquences, `geom_boxplot` calcule des quantiles, `geom_density` estime une densité, etc.)

Quelques autres fonctions de `ggplot2`

Pour ajouter des couches ou modifier des configurations dans un objet `ggplot` initialisé :

- fonctions `labs`, `ggtitle`, `xlab`, `ylab` : ajouter un titre, modifier les noms d'axes ;
- fonctions de type `coord_*` : modifier des configurations reliées au système de coordonnées ;
- fonctions de type `facet_*` : créer des grilles de sous-graphiques
  - chacun des sous-graphiques est conditionnel à la valeur de facteurs(s), il représente donc seulement le sous-ensemble des observations ayant une modalité particulière pour ce(s) facteur(s) ;
- fonctions de type `scale_*` : modifier les échelles de certaines propriétés visuelles (p. ex. couleurs, formes, tailles, etc.)
- fonctions de type `theme_*` : modifier des configurations reliées à l'apparence du graphique ;
- fonctions de type `stat_*` : ajouts d'éléments tirés d'un calcul mathématique ou statistique.

Exemple de code avec `ggplot2`

```
# Chargement du package
library(ggplot2)
# Initialisation d'un objet ggplot
graph <- ggplot(
  data = quakes,
  # Définition de l'associant entre les variables
  # et les propriétés visuelles du graphique
  mapping = aes(x = mag, y = stations)
) +
  # Ajout de points
  geom_point() +
  # Ajout d'une droite de régression
```

```
  geom_smooth(method = 'lm')
  # Ajout d'un titre
  labs(title = "1000 séismes près de Fidji") +
# Affichage du graphique
graph
```

Références

- Page web du package `ggplot2` sur le CRAN : https://CRAN.R-project.org/package=ggplot2
- Documentation du package `ggplot2` : http://ggplot2.tidyverse.org/
- Livres :
  - Wickham, H. (2016). *ggplot2 : Elegant Graphics for Data Analysis*. 2e édition. Springer.
    - URL pour la 3e édition en développement : https://ggplot2-book.org
  - Wickham, H. et Grolemund, G. (2016). *R for Data Science*. O'Reilly Media, Inc. Chapitre 3. URL https://r4ds.had.co.nz/data-visualisation.html
  - Wilkinson, L. (2005). *The grammar of graphics*, 2e édition. Springer.
  - Chang, W. (2012). *R Graphics Cookbook : Practical Recipes for Visualizing Data*. O'Reilly Media, Inc.
    - URL pour le code R : http://www.cookbook-r.com/Graphs/
- Tutoriels web :
  - Tierney, L. (2019). **Data Visualization and Data Technologies**, Notes de cours, STAT :4580, University of Iowa. URL https://homepage.divms.uiowa.edu/~luke/classes/STAT4580/index.html
  - Cook, D. (2019). *Visualising high-dimensional data*. Tutoriel offert à la conférence useR ! 2019. URL https://github.com/dicook/useR2019_highd_vis
  - Scherer, C. (2020). *A ggplot2 Tutorial for Beautiful Plotting in R*. Billet de blogue. URL https://cedricscherer.netlify.app/2019/08/05/a-ggplot2-tutorial-for-beautiful-plotting-in-r
  - Extension et tutoriel de la BBC : https://bbc.github.io/rcookbook/
- Vidéos de Roger Peng concernant `ggplot2` :
  - https://www.youtube.com/watch?v=HeqHMM4ziXA
  - https://www.youtube.com/watch?v=n8kYa9vu1l8
- Feuilles de triche (cheat sheets) :
  - https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf
  - traduction française de ce document : http://thinkr.fr/pdf/ggplot2-french-cheatsheet.pdf
- Exemples de graphiques `ggplot` :
  - Flipbook : https://evamaerey.github.io/ggplot_flipbook/ggplot_flipbook_xaringan.html#1
  - Exemples (tous systèmes, mais beaucoup de `ggplot2`) avec leur code source permettant de les reproduire : http://www.r-graph-gallery.com/
- Extensions de `ggplot2` :
  - https://exts.ggplot2.tidyverse.org
  - https://github.com/erikgahner/awesome-ggplot2
- Conseils pour améliorer nos graphiques, illutrés avec `ggplot2` : http://www.thinkingondata.com/6-tips-to-make-your-visualizations-look-professional/
- Tutoriels rédigés par d'anciens étudiants du cours :

- Boxplot avec `ggplot2` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/boxplot_ggplot2/
- Graphique de série temporelle avec `ggplot2` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/graphique_temporel_ggplot2/
- Histogramme et courbe d'estimation de densité à noyau avec `ggplot2` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/histogramme_ggplot2/
- Diagramme à barres avec `ggplot2` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/diagramme_batons_ggplot2/
- Extensions `ggrepel` et `ggthemes` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/ggrepel_ggthemes_paletteer/
- Extension `cowplot` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2019/cowplot/
- Extension `patchwork` et `ggExtra` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/patchwork_ggextra/
- Extension `gganimate` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2020/gganimate/
- Extension `ggiraph` : https://stt4230.rbind.io/tutoriels_etudiants/hiver_2018/ggiraph/