

# Graphiques en R

*Sophie Baillargeon, Université Laval*

*2018-02-07*

## Table des matières

<b>Introduction</b>	<b>2</b>
La visualisation graphique, une étape importante . . . . .	2
Les graphiques, une force de R . . . . .	3
Présentation des données utilisées pour les exemples . . . . .	3
<b>Système graphique de base en R</b>	<b>4</b>
Procédure de création d'un graphique . . . . .	4
La fonction générique <b>plot</b> . . . . .	5
Exemples : Graphique produit par <b>plot</b> en fonction des classes des arguments <b>x</b> et <b>y</b> . . . . .	5
Résumé des graphiques produits par <b>plot</b> . . . . .	18
Autres fonctions de création d'un graphique . . . . .	18
Exemple : Diagramme de dispersion (ou en lignes) - fonctions <b>plot</b> , <b>pairs</b> et <b>matplot</b> . . . . .	19
Exemple : Diagramme en secteurs - fonction <b>pie</b> . . . . .	21
Exemple : Diagramme en bâtons - fonction <b>barplot</b> . . . . .	21
Exemple : Diagramme en mosaïque - fonction <b>mosaicplot</b> . . . . .	23
Exemple : Histogramme - fonction <b>hist</b> . . . . .	23
Exemple : Diagramme en boîte - fonction <b>boxplot</b> . . . . .	24
Exemple : Courbe d'estimation de densité à noyau - méthode <b>plot.density</b> . . . . .	25
Exemple : Diagramme quantile-quantile - fonctions <b>qqnorm</b> et <b>qqplot</b> . . . . .	25
Exemple : Représentation graphique d'une fonction - fonction <b>curve</b> . . . . .	26
Arguments et paramètres graphiques . . . . .	27
Exemple : Ajout ou modification des annotations . . . . .	28
Exemple : Modification de la mise en forme . . . . .	29
Types de représentation - argument <b>type</b> . . . . .	30
Types de lignes - paramètre <b>lty</b> . . . . .	31
Symboles pour les points - paramètre <b>pch</b> . . . . .	31
Couleurs . . . . .	31
Ajout d'éléments à un graphique . . . . .	33
Exemple : Ajout de lignes . . . . .	34
Exemple : Ajout de courbes à un histogramme . . . . .	35
Possibilités graphiques spécifiques . . . . .	36
Annotations mathématiques . . . . .	36
Plusieurs graphiques dans une même fenêtre . . . . .	38
Aspects techniques . . . . .	42
Fenêtres graphiques . . . . .	42
Enregistrer un graphique . . . . .	43
Autres fonctions et trucs utiles . . . . .	43
Résumé et point de vue . . . . .	46
Faiblesse du système graphique de base . . . . .	46
<b>Autres systèmes graphiques en R</b>	<b>48</b>
Package <b>lattice</b> . . . . .	49
Exemples : Diagrammes de dispersion - fonction <b>xyplot</b> . . . . .	49
Package <b>ggplot2</b> . . . . .	51
La grammaire graphique . . . . .	52

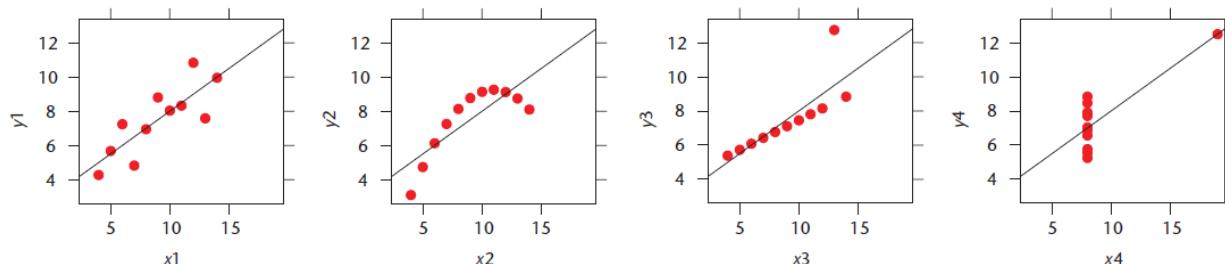
Survol des principales fonctions . . . . .	52
Code gabarit minimaliste . . . . .	52
Objet de classe <code>ggplot</code> . . . . .	53
Quelques fonctions de type <code>geom_</code> . . . . .	53
Quelques autres fonctions . . . . .	54
Exemples simples . . . . .	54
Exemples plus poussés . . . . .	62
Enregistrement d'un graphique - fonction <code>ggsave</code> . . . . .	69
Comparaison entre <code>ggplot2</code> et le système graphique R de base . . . . .	70
<b>Autres possibilités graphiques en R</b>	<b>70</b>
Graphiques 3D . . . . .	70
Fonction <code>persp</code> du système de base . . . . .	70
Fonction <code>cloud</code> du package <code>lattice</code> . . . . .	71
Graphiques 3D orientables - package <code>rgl</code> . . . . .	72
Cartes géographiques . . . . .	72
Packages <code>maps</code> et <code>mapdata</code> . . . . .	72
Packages <code>ggmap</code> . . . . .	73
Graphiques interactifs . . . . .	73
Graphiques interactifs Plotly - package <code>plotly</code> . . . . .	74
Google Charts - package <code>googleVis</code> . . . . .	74
<b>Conclusion</b>	<b>74</b>
R = un excellent outil pour les graphiques . . . . .	75
<b>Références</b>	<b>75</b>

---

## Introduction

### La visualisation graphique, une étape importante

La communauté statistique insiste depuis longtemps sur l'importance de visualiser graphiquement des données avant de les analyser. En 1973, Anscombe<sup>1</sup> publiait les [quatre jeux de données suivants](#), ayant les mêmes paramètres estimés de régression linéaire simple ( $y = 3 + 0.5x$ ), mais présentant des relations bien différentes !

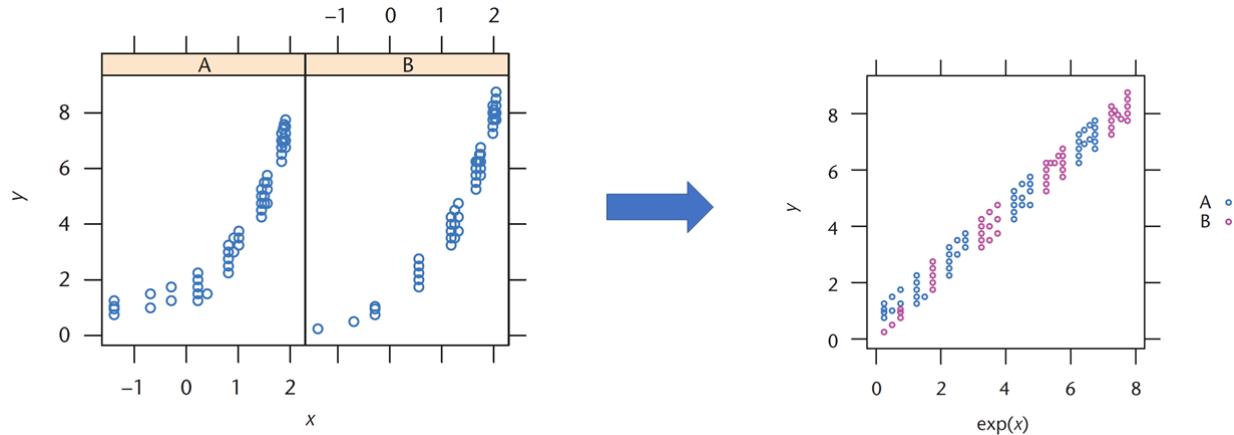


Plusieurs autres exemples de jeux de données possédant les mêmes statistiques descriptives, mais des représentations graphiques très différentes ont été publiés depuis<sup>2</sup>.

<sup>1</sup>Anscombe, F.J. (1973). Graphs in Statistical Analysis. *The American Statistician*, **27**(1). pp. 17-21.

<sup>2</sup>Matejka, J. et Fitzmaurice, G. (2017, Mai). Same Stats, Different Graphs : Generating Datasets with Varied Appearance and Identical Statistics through Simulated Annealing. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM. pp. 1290-1294. URL : <https://www.autodeskresearch.com/publications/samestats>

Certains ont même mis en évidence l'importance de représenter graphiquement des données de plusieurs manières différentes afin de découvrir les surprises qu'elles peuvent cacher. Par exemple, les données suivantes, créées pour enseigner à des étudiants l'importance des graphiques en analyse de données<sup>3</sup>, forment le mot SUPERMAN si elles sont représentées sur une échelle adéquatement transformée et si les observations des deux groupes sont superposées.



## Les graphiques, une force de R

En plus d'être un environnement de calculs statistiques, R permet de produire des graphiques. Il s'agit d'une utilité importante de R. La toute première phrase du [site web de R](#) mentionne cette utilité.

« R is a free software environment for statistical computing and graphics. »

Lors de comparaison entre R et d'autres logiciels permettant de faire de l'analyse de données, les capacités graphiques de R sont souvent considérées comme un de ses meilleurs atouts<sup>45</sup>.

L'installation de base de R comporte beaucoup de fonctions graphiques. La première partie de ces notes présente ces fonctions. Elles permettent de produire des graphiques pour lesquels l'utilisateur garde le plein contrôle sur l'apparence du graphique.

De plus, deux systèmes graphiques alternatifs ont également été développés en R et sont abordés ici :

- **lattice** : package graphique qui met l'emphase sur les représentations multivariables,
- **ggplot2** : package graphique conçu en ayant comme objectif la simplicité d'utilisation et la qualité des graphiques produits.

Pour finir, quelques autres possibilités graphiques en R sont mentionnées rapidement :

- graphiques 3D,
- cartes géographiques,
- graphiques interactifs.

C'est la grande quantité de possibilités graphiques qui en fait une force (à reformuler).

## Présentation des données utilisées pour les exemples

Plusieurs des exemples de graphiques de ces notes représentent les mêmes données : celles stockées dans le [data frame quakes](#) du package [datasets](#) (chargé par défaut lors de l'ouverture de session).

<sup>3</sup>Kozak, M. (2002). Watch out for superman : first visualize, then analyze. *IEEE Computer Graphics and Applications*. **32**(3), pp. 6–9.

<sup>4</sup><https://www.datacamp.com/community/tutorials/r-or-python-for-data-analysis>

<sup>5</sup><http://www.theanalyticslab.nl/2017/03/18/python-r-vs-spss-sas/>

```

str(quakes)

## 'data.frame':   1000 obs. of  5 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 ...

```

Il s'agit d'observations pour 1000 événements sismiques survenus près des îles Fidji depuis 1964. Pour chaque événement, nous avons :

- sa localisation géographique en latitude-longitude,
  - sa **profondeur**,
  - sa **magnitude** et
  - le nombre de stations sismiques ayant rapporté l'événement.
- 

## Système graphique de base en R

Les graphiques de base en R sont créés grâce aux fonctions provenant des packages **graphics** et **grDevices** développés par le « R core team ». Ces packages sont intégrés à l'installation de base de R et ils sont chargés par défaut lors de l'ouverture d'une session R. Dans cette section, nous couvrirons les possibilités de ces packages.

### Procédure de création d'un graphique

Un programme pour créer un graphique avec le système graphique de base en R se décompose typiquement en 3 étapes, les suivantes :

1. La configuration des paramètres graphiques (au besoin) :
  - énoncé par ou layout.
2. L'initialisation d'un graphique :
  - fonction de base : **plot**,
  - ou fonction pour un type spécifique graphiques : **pairs**, **matplot**, **pie**, **barplot**, **mosaicplot**, **hist**, **boxplot**, **qqnorm**, **qqplot**, **curve**, etc.
3. L'ajout séquentielle d'éléments au graphique (au besoin) :
  - fonctions d'ajouts à un graphique déjà initialisé :
    - **points**, **matpoints**, **lines**, **matlines**, **abline**, **segments**, **arrows**, **rect**, **polygon**, **legend**, **text**, **mtext**, **title**, **axis**, **box**, **qqline**, etc.;
    - **matplot**, **barplot**, **hist**, **boxplot**, **curve** avec l'argument **add = TRUE**.

Lorsque les paramètres graphiques sont modifiés avec un appel à la fonction **par** au début du programme, une bonne pratique est d'ajouter aussi à la fin du programme une commande pour redonner aux paramètres graphiques leurs valeurs par défaut.

La mise en forme et les annotations du graphique (p. ex. titre et noms d'axe) sont déterminées par les paramètres graphiques ainsi que par des arguments des fonctions graphiques. Avec un peu de patience et quelques lignes de code, il est possible d'arriver à contrôler parfaitement l'apparence du graphique.

Les prochaines sections présentent les concepts à connaître pour suivre cette procédure de création de graphique et contiennent beaucoup d'exemples. Cependant, la matière n'est pas couverte dans le même ordre que les étapes de la procédure. Même si dans le code de création d'un graphique la configuration des paramètres est l'étape 1, il faut d'abord savoir créer un graphique pour que cette étape ait du sens. De

plus, la configuration des paramètres graphique est une étape facultative. Parfois, les valeurs par défaut des paramètres font l'affaire.

Ainsi, nous allons d'abord étudier la principale fonction du système graphique de base en R, soit la fonction `plot`. Dans un deuxième temps, nous ferons un survol de plusieurs autres fonctions de création de graphiques. Ensuite, les arguments et paramètres graphiques qui permettent de contrôler les annotations et la mise en forme seront présentés. Pour terminer la couverture des étapes de la procédure de création, il ne restera plus qu'à parler des fonctions d'ajouts d'éléments à un graphique.

Les exemples présentés seront simple au début, mais se complexifieront vers la fin de la section.

## La fonction générique `plot`

La principale fonction graphique en R est la fonction `plot`. En fait, `plot` est une fonction générique. Il existe plusieurs méthodes associées à cette fonction générique. Pour afficher la liste des méthodes `plot` chargées dans notre session R, il faut soumettre la commande suivante (sortie non affichée ici).

```
methods(plot)
```

```
## [1] plot.acf*      plot.data.frame*   plot.decomposed.ts* plot.default
## [5] plot.dendrogram* plot.density*     plot.ecdf          plot.factor*
## [9] plot.formula*    plot.function     plot.hclust*       plot.histogram*
## [13] plot.HoltWinters* plot.isoreg*     plot.lm*           plot.medpolish*
## [17] plot.mlm*        plot.ppr*        plot.prcomp*      plot.princomp*
## [21] plot.profile.nls* plot.raster*     plot.spec*        plot.stepfun
## [25] plot.stl*        plot.table*     plot.ts            plot.tskernel*
## [29] plot.TukeyHSD*
## see '?methods' for accessing help and source code
```

Si nous donnons en entrée à la fonction `plot` des vecteurs en arguments `x` et `y`, comme dans l'exemple précédent, c'est la méthode `plot.default` qui est utilisée. Cette méthode trace un diagramme de dispersion.

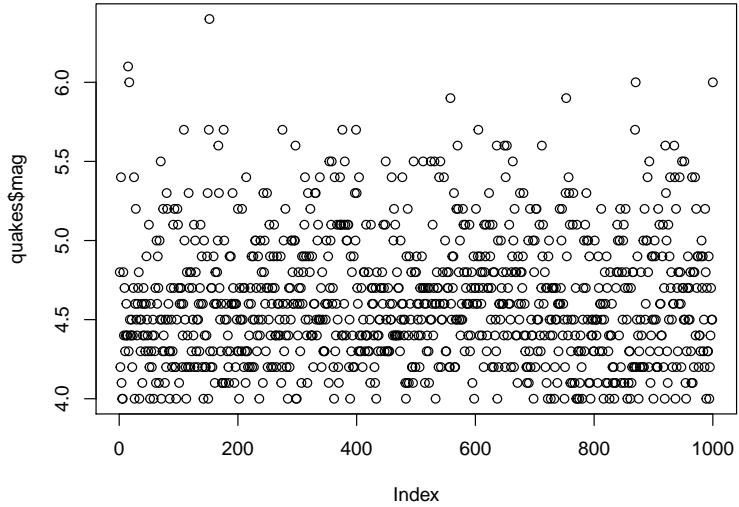
La fonction `plot` choisit la méthode à utiliser en fonction des classes des arguments `x` et `y` fournis en entrée. C'est une caractéristique « orientée objet » du langage R. Nous reviendrons sur ce concept plus tard. Pour l'instant, il suffit de comprendre que les différentes méthodes produisent différents résultats et acceptent différents arguments. En fait, ces méthodes font souvent appels à d'autres fonctions de création de graphique.

### Exemples : Graphique produit par `plot` en fonction des classes des arguments `x` et `y`

Faisons quelques essais pour mieux comprendre la fonction `plot`. Appelons la fonction en lui fournissant en entrée des arguments `x` et `y` de différentes classes et observons le résultat. Les données `quakes` sont utilisées ici.

#### Un seul vecteur numérique

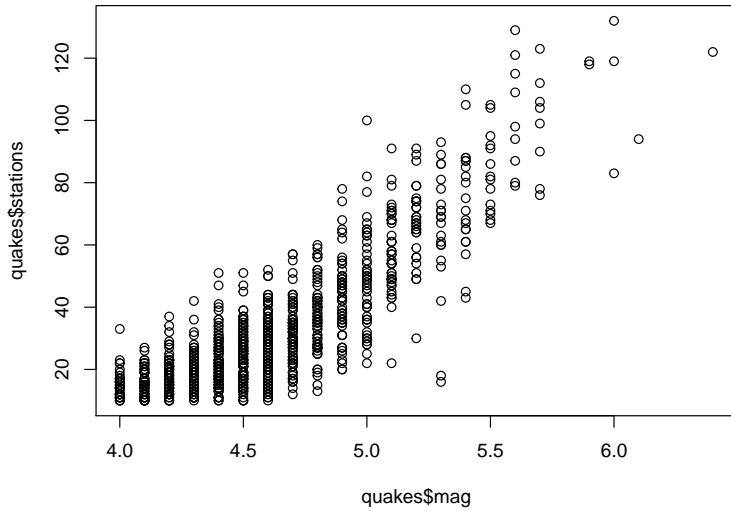
```
plot(x = quakes$mag)
```



Si seulement un vecteur est fourni en entrée, en argument `x`, et qu'aucun argument `y` n'est fourni, alors les observations contenues dans le vecteur sont placées sur l'axe des `y` (ordonnées), et les entiers 1 à `length(x)` sont placés sur l'axe des `x` (abscisses). C'est la méthode `plot.default` qui a tracé ce graphique.

## Deux vecteurs numériques

```
plot(x = quakes$mag, y = quakes$stations)
```

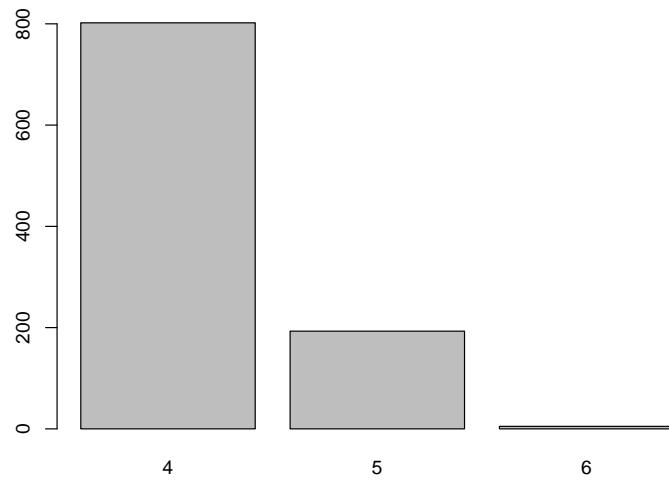


Un diagramme de dispersion est produit, encore par la méthode `plot.default`.

## Un seul facteur

```
# Création d'un facteur à partir de la variable mag
quakes$magFactor <- factor(floor(quakes$mag))

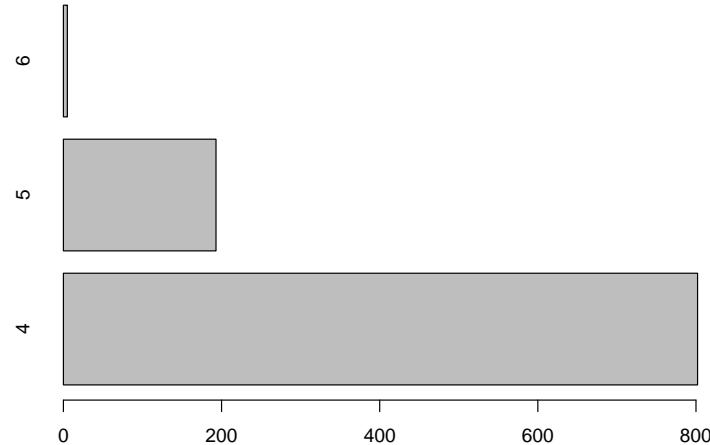
# Exemple d'appel à la fonction plot en lui donnant un facteur en entrée
plot(x = quakes$magFactor)
```



Si seulement un facteur est fourni en entrée, un diagramme en bâtons est produit. La fonction générique `plot` a commencé par envoyer les arguments qu'elle a reçus à la méthode `plot.factor` parce que la classe de l'argument `x` fourni en entrée était `factor`. Ensuite, la méthode `plot.factor` a choisi d'appeler la fonction `barplot`, parce qu'aucun argument `y` n'a été fourni. En fait, ce sont les fréquences des niveaux du facteur (`table(x)`) qui sont données comme premier argument à `barplot`.

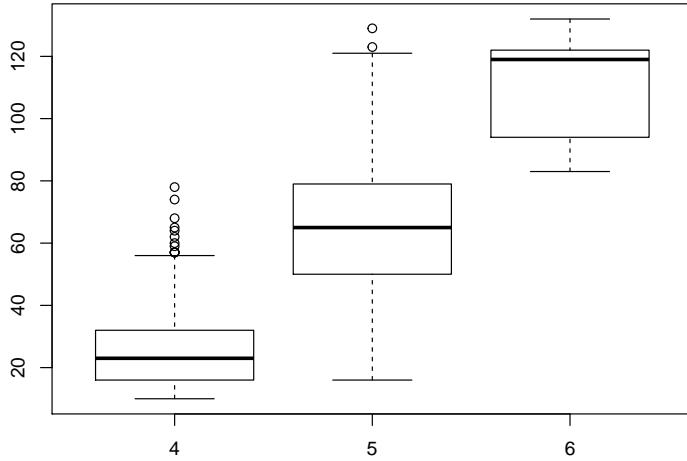
Ainsi, les arguments acceptés par la fonction générique `plot` lorsqu'elle reçoit en entrée comme premier argument (`x`) un facteur sans recevoir d'argument `y` sont les arguments acceptés par la fonction `barplot`. Par exemple, nous pourrions produire un diagramme à bâtons horizontaux comme suit.

```
plot(x = quakes$magFactor, horiz = TRUE)
```



### Un facteur et un vecteur numérique

```
plot(x = quakes$magFactor, y = quakes$stations)
```

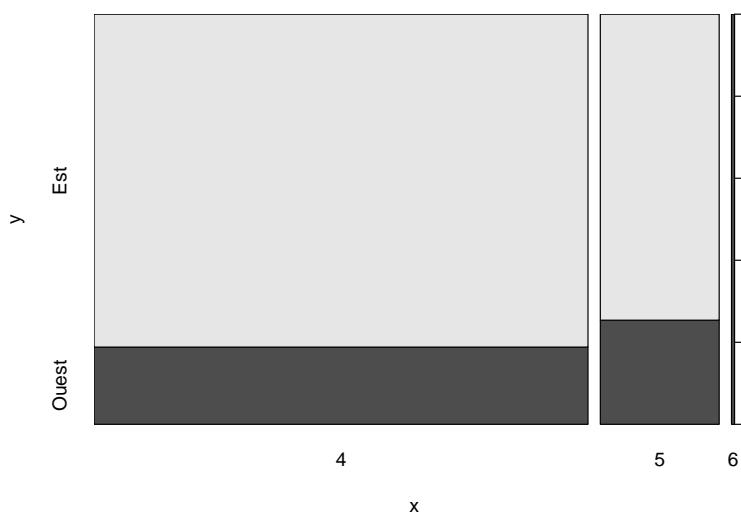


Si un vecteur numérique est fourni en argument `y`, avec le facteur fourni en `x`, la méthode `plot.factor` choisit d'appeler la fonction `boxplot` pour produire des boxplots de la variable `y` pour chacun des niveaux de la variable `x`.

## Deux facteurs

```
# Création d'un deuxième facteur par clustering (classification
# non supervisée), par la méthode des k-moyennes, à partir
# des variables lat et long (coordonnées géographiques)
outkmeans <- kmeans(quakes[, c("lat", "long")], centers = 2)
if (mean(quakes$long[outkmeans$cluster == 1]) > mean(quakes$long[outkmeans$cluster == 2])) {
  regionkmeans <- ifelse(outkmeans$cluster == 1, 2, 1)
} else {
  regionkmeans <- outkmeans$cluster
}
quakes$region <- factor(regionkmeans, labels = c("Ouest", "Est"))

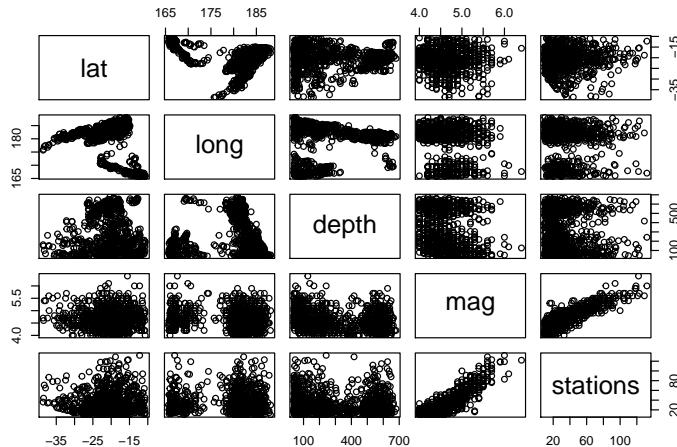
# Exemple d'appel à la fonction plot en lui donnant deux facteurs en entrée
plot(x = quakes$magFactor, y = quakes$region)
```



Si deux facteurs sont fournis en entrée, la méthode `plot.factor` choisit d'appeler la fonction `spineplot`. Elle produit un type particulier de diagramme en mosaïque, nommé en anglais *spineplot*.

### Data frame à plus de 2 variables contenant des valeurs numériques ou transformables en valeurs numériques

```
plot(datasets::quakes)
```



Si un data frame est fourni en entrée à la fonction générique `plot`, elle délègue son travail à la méthode `plot.data.frame`. Si ce data frame possède plus de 2 variables contenant des valeurs numériques (ou transformables en valeurs numériques), alors la méthode `plot.data.frame` choisit d'appeler la fonction `pairs`. Celle-ci produit une matrice de diagramme de dispersion.

**Remarque :** Nous avons maintenant deux objets nommés `quakes` dans notre chemin de recherche, parce que nous avons ajouté des variables au data frame dans des exemples précédents. Le data frame modifié, soit celui avec les variables ajoutées, est situé dans notre environnement de travail. Alors que le data frame `quakes` original est toujours dans l'environnement du package `datasets`. Celui-là n'a pas été modifié.

Le chemin de recherche de notre session R, pour une commande soumise dans la console, est le suivant.

```
search()
```

```
## [1] ".GlobalEnv"           "package:stats"      "package:graphics"
## [4] "package:grDevices"    "package:utils"       "package:datasets"
## [7] "package:methods"      "Autoloads"          "package:base"
```

Lorsque nous demandons à R d'accéder à l'objet nommé `quakes`, il cherche ce nom dans les environnements de son chemin de recherche, un environnement à la fois, en respectant l'ordre des environnements dans la liste précédente. L'environnement "`.GlobalEnv`", soit notre environnement de travail, est le premier de la liste. Lorsque R trouve le nom, il cesse sa recherche. Nous reviendrons sur le fonctionnement des évaluations en R dans un autre cours.

Ainsi, toute commande dans la console contenant le nom `quakes`, accédera à l'objet nommé `quakes` de notre environnement de travail.

```
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 ...
## $ magFactor: Factor w/ 3 levels "4","5","6": 1 1 2 1 1 1 1 1 1 ...
## $ region   : Factor w/ 2 levels "Ouest","Est": 2 2 2 2 2 2 1 2 2 2 ...

```

Mais il est possible de forcer R à aller chercher l'objet `quakes` dans le package `datasets` grâce à l'opérateur `::`:

```
str(datasets::quakes)
```

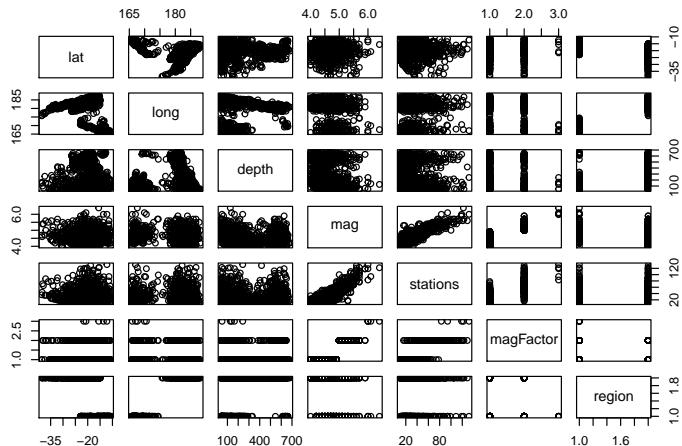
```

## 'data.frame':    1000 obs. of  5 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 ...

```

Dans le graphique précédent, nous avons forcé R à utiliser le jeu de données `quakes` original. Avec le nouveau jeu de données, nous aurions obtenu ceci.

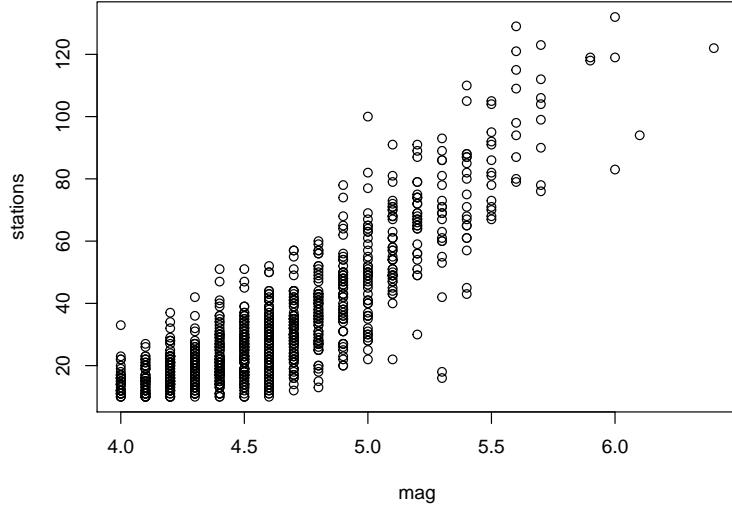
```
plot(quakes)
```



Les facteurs ont été transformés en vecteurs numériques.

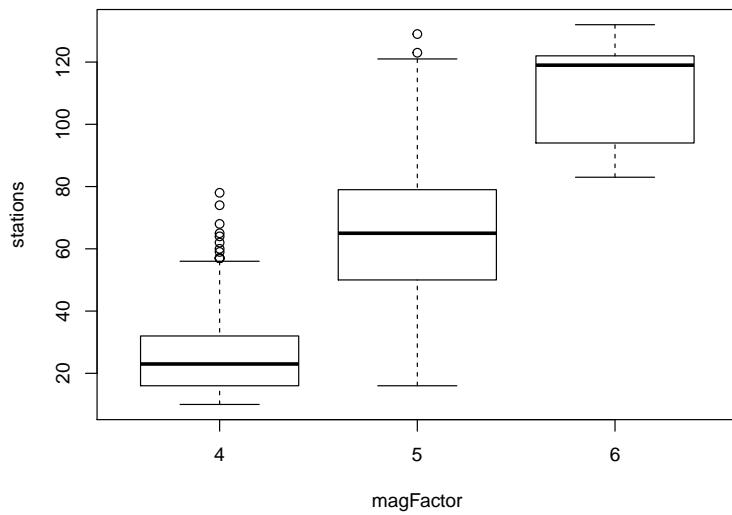
### Data frame à 2 variables

```
plot(quakes[, c("mag", "stations")])
```



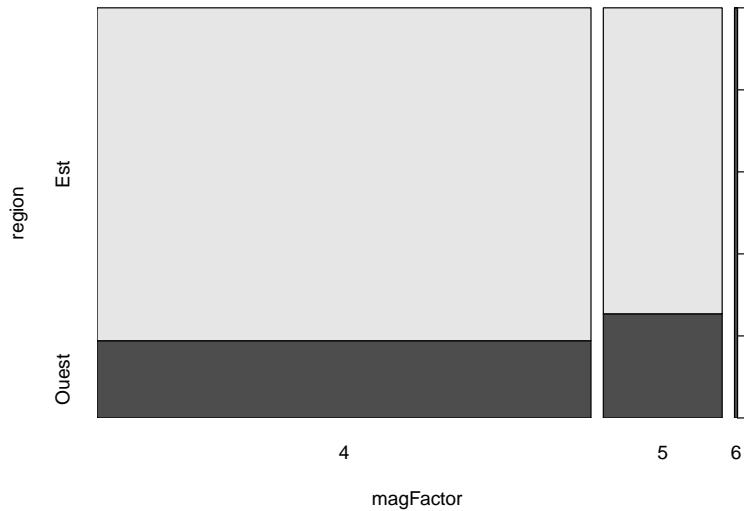
Si un data frame à seulement 2 variables est fourni en entré à `plot`, la méthode `plot.data.frame` évalue le type des données dans ces variables. Si les deux variables contiennent des valeurs numériques, la méthode `plot.default` est appelée pour produire un seul diagramme de dispersion. Les observations dans la première colonne sont placées sur l'axe des x et celle de la deuxième colonne sur l'axe des y.

```
plot(quakes[, c("magFactor", "stations")])
```



Si la première colonne est un facteur, des boxplots des observations dans la deuxième colonne, par niveaux du facteur dans la première colonne, sont produits avec la fonction `boxplot`.

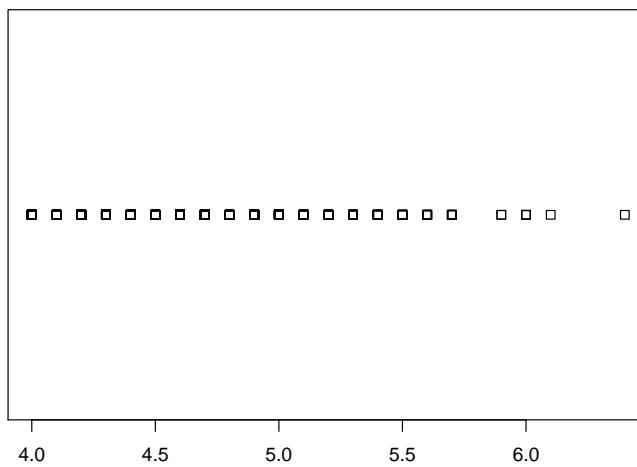
```
plot(quakes[, c("magFactor", "region")])
```



Si les deux variables sont des facteurs, `plot.data.frame` se comporte alors comme `plot.factor` lorsqu'il reçoit deux facteurs.

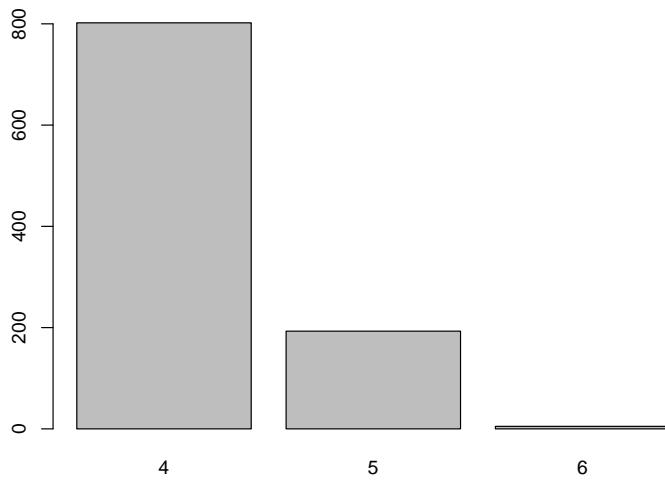
#### Data frame à une colonne

```
plot(quakes[, "mag", drop = FALSE])
```



Si un data frame à seulement une colonne est fourni à `plot` et que cette colonne contient des observations numériques, un diagramme de dispersion à une dimension est produit (*dot plot* à une variable), en appelant la fonction `stripchart`.

```
plot(quakes[, "magFactor", drop = FALSE])
```



Si la colonne contient plutôt un facteur, c'est un diagramme en bâtons qui est produit par la fonction `barplot`.

**Remarque :** Sans l'argument `drop = FALSE` dans l'extraction d'une colonne de `quakes`, le résultat de l'extraction aurait été un vecteur ou un facteur (un objet à une dimension) plutôt qu'un data frame à une colonne.

```
str(quakes[, "magFactor"])

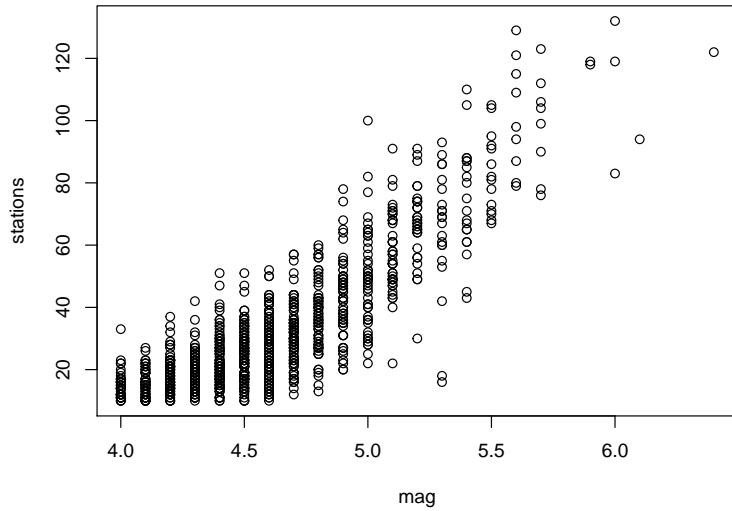
## Factor w/ 3 levels "4","5","6": 1 1 2 1 1 1 1 1 1 1 ...
str(quakes[, "magFactor", drop = FALSE])

## 'data.frame':    1000 obs. of  1 variable:
## $ magFactor: Factor w/ 3 levels "4","5","6": 1 1 2 1 1 1 1 1 1 1 ...
```

### Formule en entrée

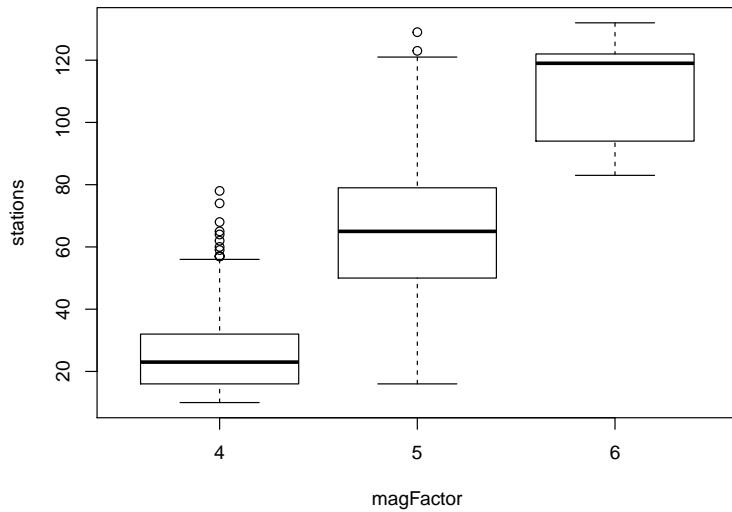
La méthode `plot` accepte aussi une formule en entrée, auquel cas la méthode `plot.formula` est appelée. La méthode `plot.formula` est très versatile. La variable que nous lui fournissons dans la partie de gauche de la formule (à gauche de `~`) est celle positionnée sur l'axe des y. La variable que nous lui fournissons dans la partie de droite de la formule est celle positionnée sur l'axe des x.

```
plot(stations ~ mag, data = quakes)
```



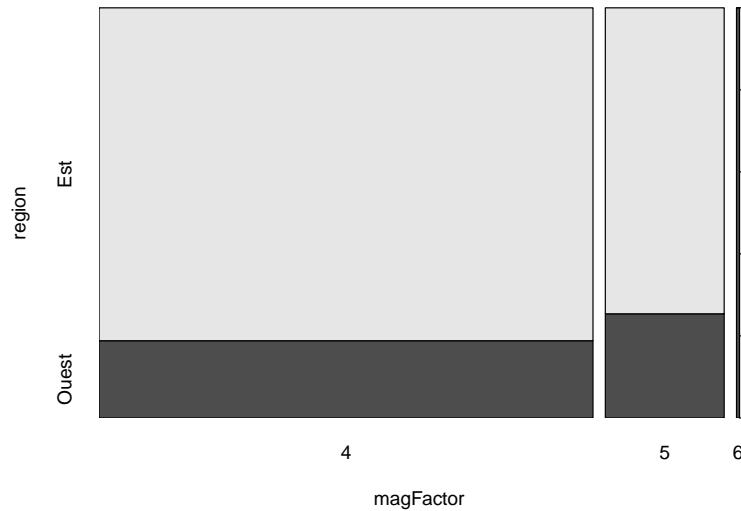
Si les deux variables sont numériques, un diagramme de dispersion est produit. Mais si la variable à droite est un facteur, des boxplots de la variable dans la partie de gauche de la formule, par niveau de ce facteur, sont produits.

```
plot(stations ~ magFactor, data = quakes)
```



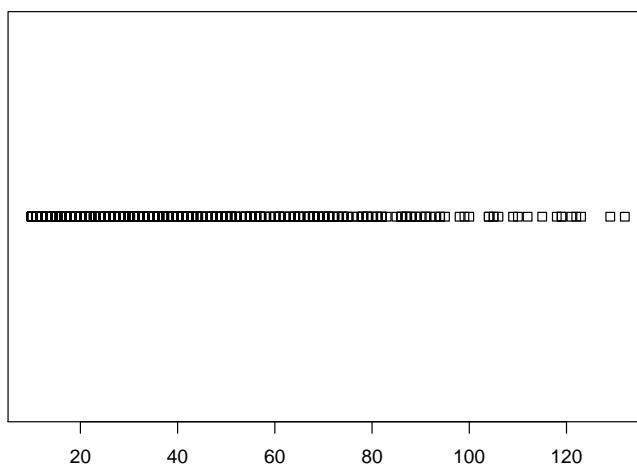
Si les deux variables sont des facteurs, un diagramme en mosaïque est produit.

```
plot(region ~ magFactor, data = quakes)
```



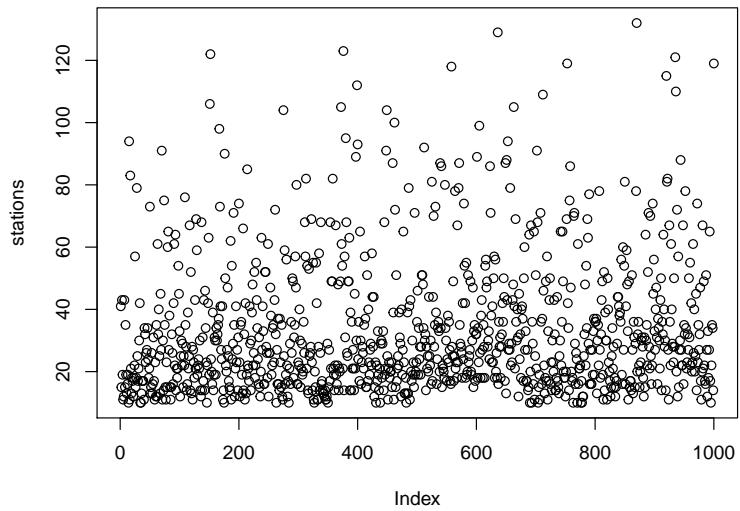
Nous pouvons aussi spécifier seulement une variable dans la formule. Par exemple, avec une seule variable dans la partie de droite de la formule (rien dans la partie de gauche de la formule), nous obtenons le même résultat qu'en fournissant un data frame à une colonne à la fonction générique `plot`.

```
plot(~ stations, data = quakes)
```



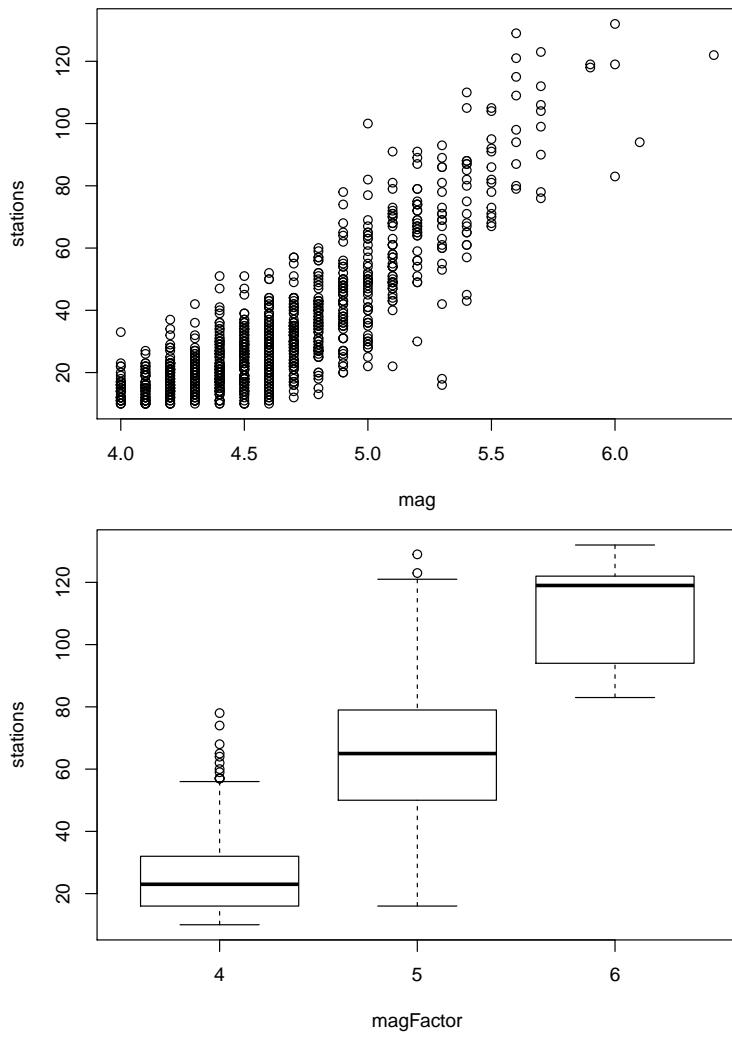
En plaçant plutôt l'unique variable dans la partie de gauche de la formule, avec la valeur 1 dans la partie de droite, nous obtenons le même résultat qu'en fournissant un seul vecteur ou facteur à la fonction générique `plot`.

```
plot(stations ~ 1, data = quakes)
```



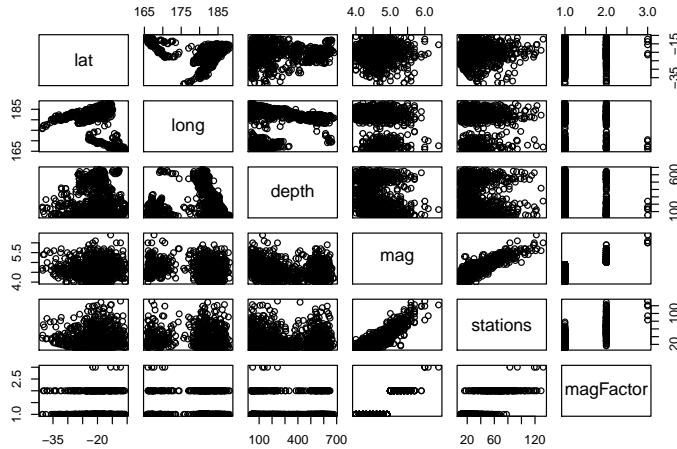
Si la formule contient une variable dans la partie de gauche et plus d'une variable dans la partie de droite, plus d'un graphique est produit.

```
plot(stations ~ mag + magFactor, data = quakes)
```



Mais si la formule ne contient pas de variable dans la partie de gauche et plus de deux variables dans la partie de droite, une matrice de diagrammes de dispersion est produite.

```
plot(~ lat + long + depth + mag + stations + magFactor, data = quakes)
```

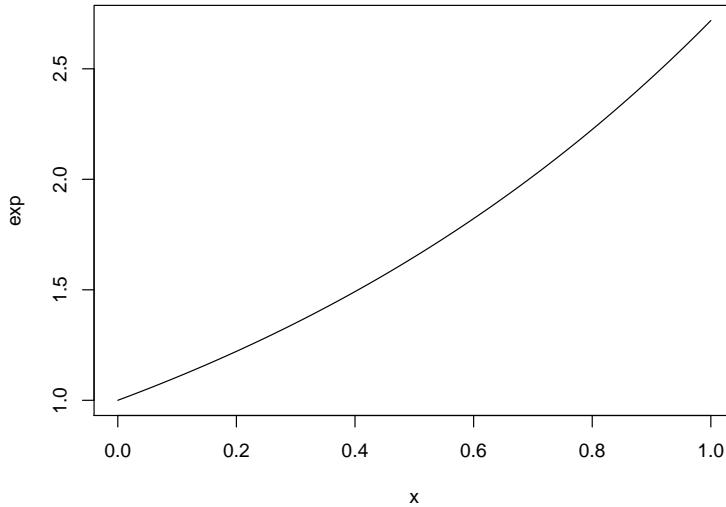


Bref, la méthode `plot.formula` fait appel à une des autres méthodes vues jusqu'à maintenant, en fonction de la position et de la nature (vecteur numérique ou facteur) des variables dans la formule.

### Fonction en entrée

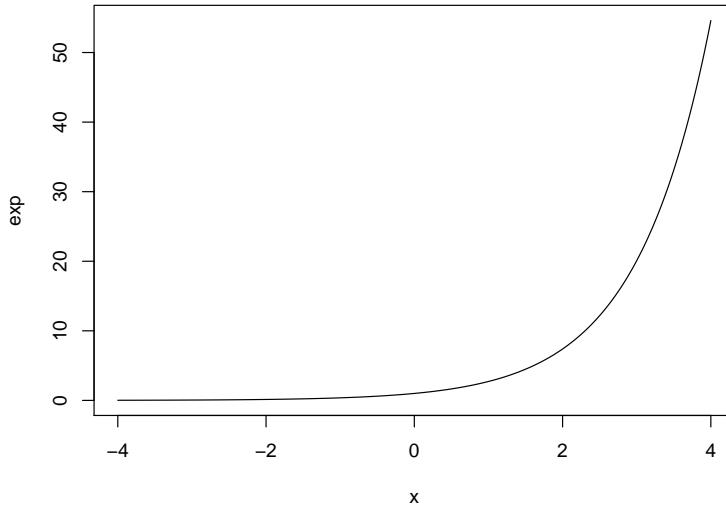
Il y a en R plusieurs fonctions prenant comme premier argument un vecteur numérique et retournant un vecteur de valeurs numériques, par exemple les fonctions R de type mathématique `abs`, `srqt`, `exp`, `log`, `sin`, `cos`, etc. Ce sont en quelque sorte des représentations de fonctions mathématiques d'une variable. La fonction générique `plot` peut tracer ces fonctions, grâce à sa méthode `plot.function`, après avoir évalué leurs valeurs en certains points.

```
plot(exp)
```



Par défaut, elle fait une évaluation de la valeur de la fonction en 101 points également répartis entre 0 et 1. Il est possible de modifier ces points grâce aux arguments `from`, `to` et `n` (une séquence de points est créée avec la fonction `seq`).

```
plot(exp, from = -4, to = 4, n = 501)
```



En fait, la méthode `plot.function` appelle la fonction `curve`.

## Résumé des graphiques produits par `plot`

Type x	Type y	Graphique produit (méthode utilisée → fonction ou autre méthode appelée)
vecteur	-	diagramme de dispersion en fonction de l'index des observations ( <code>plot.default</code> )
	vecteur	diagramme de dispersion ( <code>plot.default</code> )
facteur	-	diagramme en bâtons ( <code>plot.factor</code> → <code>barplot</code> )
	vecteur	diagrammes en boîtes juxtaposés ( <code>plot.factor</code> → <code>boxplot</code> )
	facteur	diagramme en mosaïque ( <code>plot.factor</code> → <code>spineplot</code> )
data frame	-	dépend du nombre de variables, de leur ordre et de leurs natures ( <code>plot.data.frame</code> → <code>pairs</code> ou <code>plot.default</code> ou <code>plot.factor</code> ou <code>stripchart</code> )
	formule	dépend de la position des variables dans la formule et de leurs natures ( <code>plot.formula</code> → <code>plot.default</code> ou <code>plot.factor</code> ou <code>plot.data.frame</code> )
fonction	-	courbe ( <code>plot.function</code> → <code>curve</code> )
:	:	:

Ce tableau résume seulement ce qui a été vu dans les exemples précédents. La fonction `plot` peut en faire beaucoup plus que ça grâce à ses nombreuses méthodes. Par exemple, si elle reçoit en entrée la sortie d'un appel à la fonction `lm` qui ajuste un modèle linéaire à des données, elle produit des graphiques de résidus de ce modèle via sa méthode `plot.lm`.

## Autres fonctions de création d'un graphique

En explorant les capacités de la fonction générique `plot`, nous avons découvert plusieurs autres fonctions de création de graphique. Ces fonctions permettent de créer toutes sortes de graphiques. Voici un résumé des principaux types de graphiques pouvant être créés dans le système graphique de base en R, suivi d'exemples.

### Représentations d'une variable (observations stockées dans x)

Type de graphique	Appel de fonction	Type de x
diagramme en secteurs ( <i>pie chart</i> )	<code>pie(table(x), ...)</code>	facteur
diagramme en bâtons ( <i>bar plot</i> )	<code>barplot(table(x), ...)</code>	facteur
histogramme	<code>hist(x, ...)</code>	vecteur numérique
courbe de densité à noyau ( <i>kernel density plot</i> )	<code>plot(density(x), ...) → méthode plot.density</code>	vecteur numérique
diagramme en boîte ( <i>boxplot</i> )	<code>boxplot(x, ...)</code>	vecteur numérique
diagramme quantile-quantile théorique normal	<code>qqnorm(x, ...)</code>	vecteur numérique

### Représentations de deux variables (observations stockées dans x et y)

Type de graphique	Appel de fonction	Type de x	Type de y
diagramme en bâtons empilés ou groupés	<code>barplot(table(x, y), ...)</code> avec <code>beside = TRUE</code> pour bâtons groupés	facteur	facteur
diagramme en mosaïque	<code>mosaicplot(table(x, y), ...)</code>	facteur	facteur
diagrammes en boîte juxtaposés	<code>boxplot(y ~ x, ...)</code>	facteur	vecteur
diagramme de dispersion ( <i>scatterplot</i> ) ou en lignes ( <i>line chart</i> )	<code>plot(x, y, ...)</code> → méthode <code>plot.default</code>	vecteur	vecteur
diagramme quantile-quantile empirique	<code>qplot(x, y, ...)</code>	vecteur	vecteur
		numérique	numérique

### Représentation de plus de trois variables numériques :

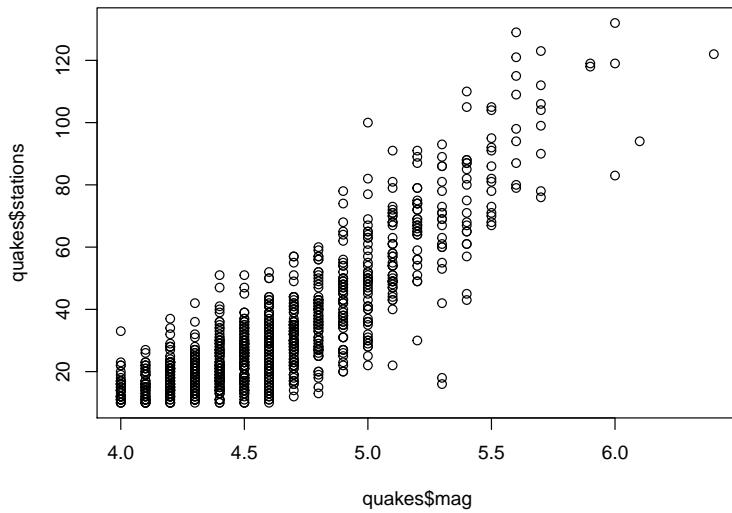
- matrice de diagrammes de dispersion : `pairs(~ x + y + z, ...)`
- diagrammes de dispersion superposés : `matplot(matriceX, matriceY, ...)`

### Représentation d'une expression mathématique : `curve(expr, ...)`

#### Exemple : Diagramme de dispersion (ou en lignes) - fonctions `plot`, `pairs` et `matplot`

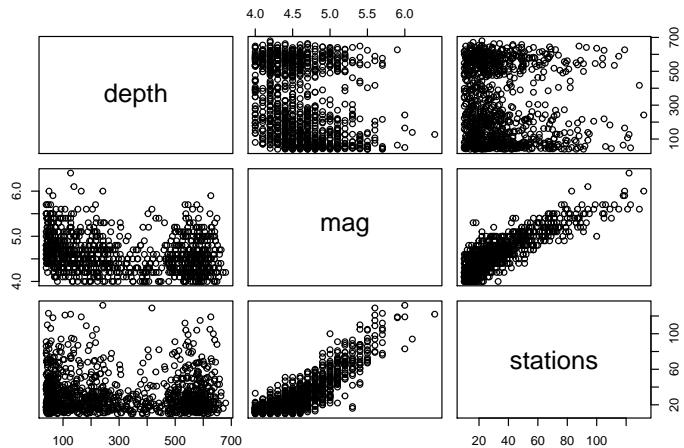
Méthode `plot.default` (via la fonction générique `plot` ou directement) :

```
plot.default(x = quakes$mag, y = quakes$stations)
```



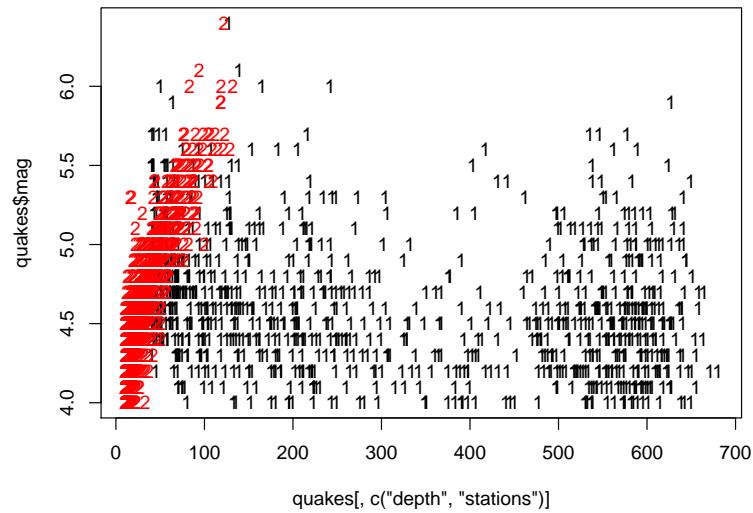
Fonction `pairs` :

```
pairs(~ depth + mag + stations, data = quakes)
```



Fonction `matplot` :

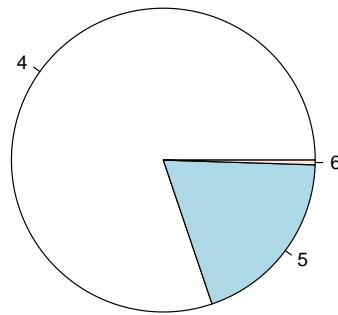
```
matplot(x = quakes[, c("depth", "stations")], y = quakes$mag)
```



Exemple : Diagramme en secteurs - fonction pie

Fonction pie :

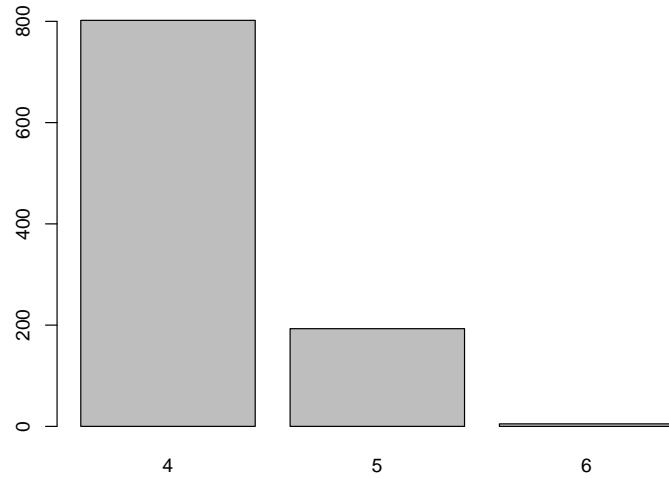
```
pie(table(quakes$magFactor))
```



Exemple : Diagramme en bâtons - fonction barplot

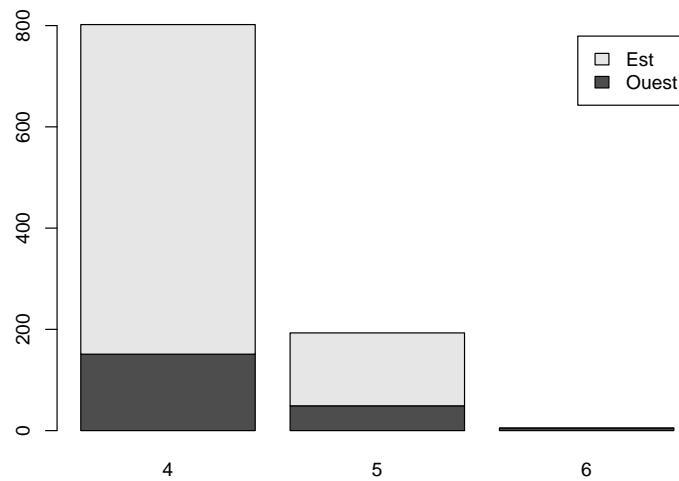
Fonction barplot :

```
barplot(table(quakes$magFactor))
```



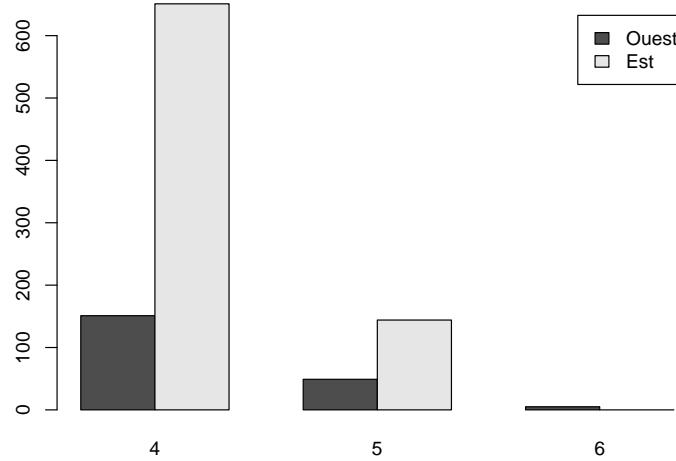
#### Deux variables - bâtons empilés

```
barplot(table(quakes$region, quakes$magFactor), legend.text = TRUE)
```



#### Deux variables - bâtons groupés

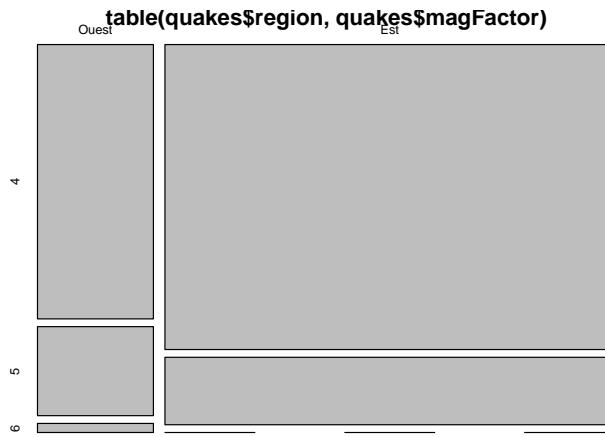
```
barplot(table(quakes$region, quakes$magFactor), legend.text = TRUE, beside = TRUE)
```



Exemple : Diagramme en mosaïque - fonction `mosaicplot`

Fonction `mosaicplot` :

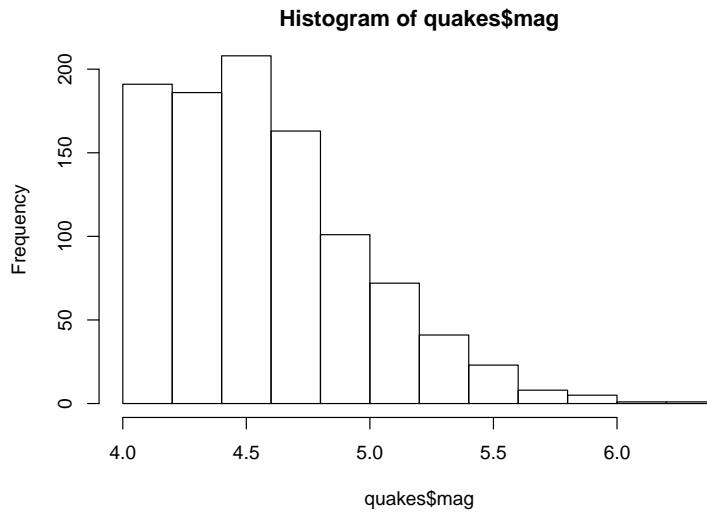
```
mosaicplot(table(quakes$region, quakes$magFactor))
```



Exemple : Histogramme - fonction `hist`

Fonction `hist` :

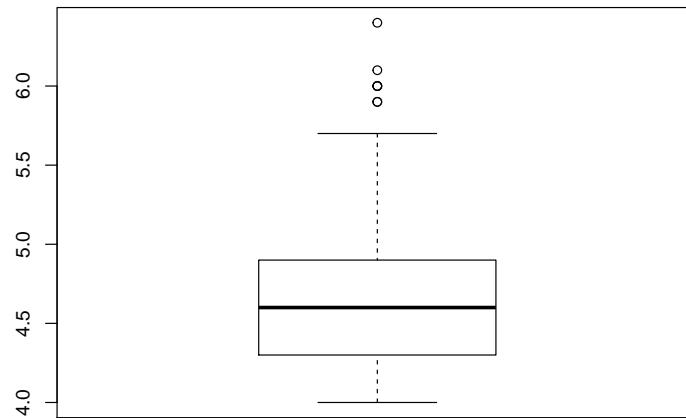
```
hist(quakes$mag)
```



Exemple : Diagramme en boîte - fonction `boxplot`

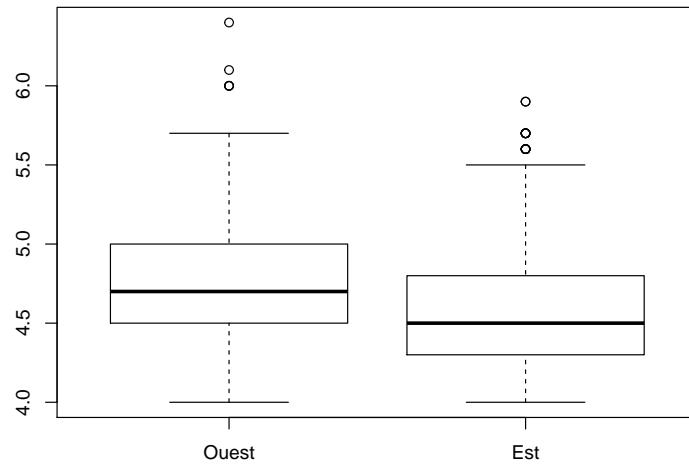
Fonction `boxplot` :

```
boxplot(quakes$mag)
```



Deux variables - diagrammes juxtaposés

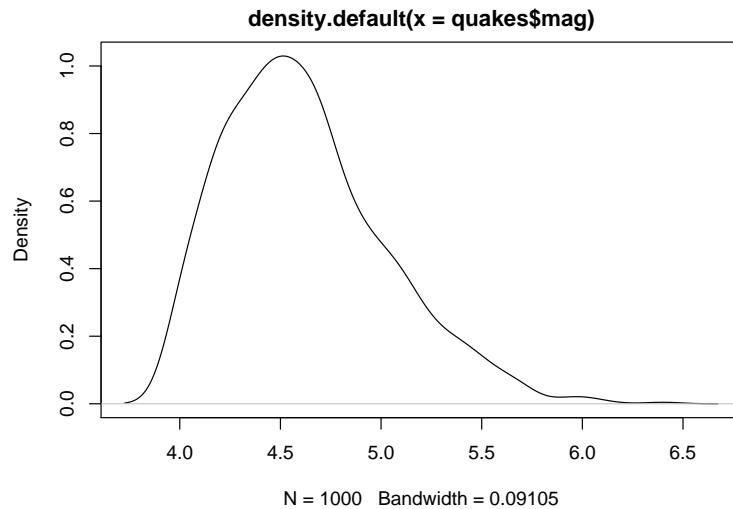
```
boxplot(mag ~ region, data = quakes)
```



Exemple : Courbe d'estimation de densité à noyau - méthode `plot.density`

Méthode `plot.density` (via la fonction générique `plot` ou directement) :

```
plot(density(quakes$mag))
```

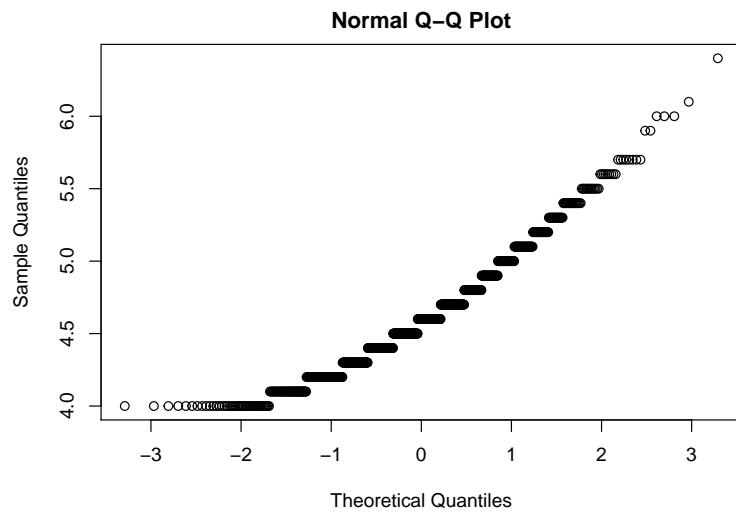


Exemple : Diagramme quantile-quantile - fonctions `qqnorm` et `qqplot`

Une variable - diagramme quantile-quantile théorique normal

Fonction `qqnorm` :

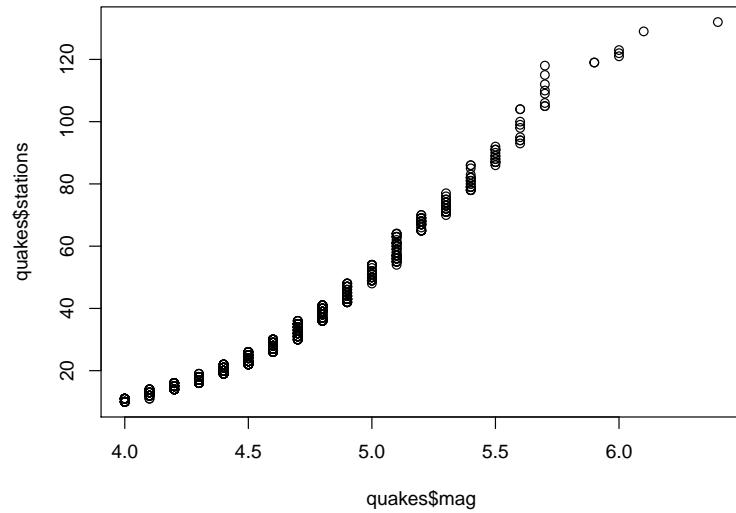
```
qqnorm(quakes$mag)
```



**Deux variables - diagramme quantile-quantile empirique**

Fonction `qqplot` :

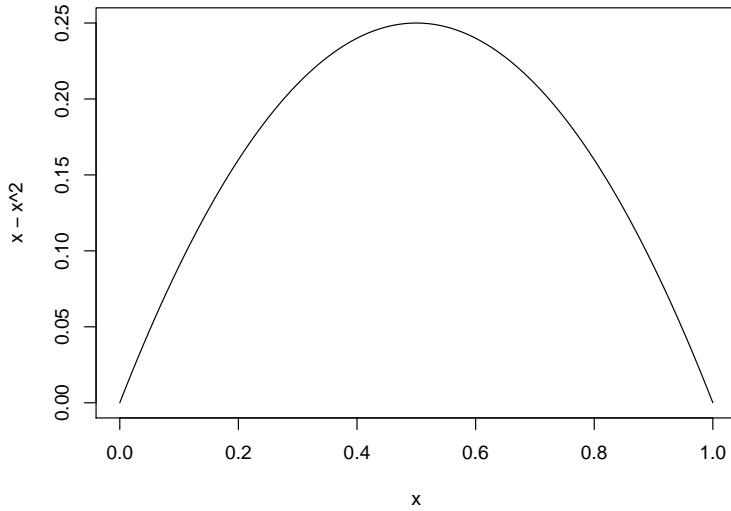
```
qqplot(quakes$mag, quakes$stations)
```



**Exemple : Représentation graphique d'une fonction - fonction `curve`**

Fonction `curve` :

```
curve(x ~ x^2)
```



Le premier argument n'est pas contraint à être une fonction, comme avec la méthode `plot.function`. Il peut s'agir de n'importe quelle expression écrite comme une fonction de `x`.

## Arguments et paramètres graphiques

Les fonctions vues jusqu'à présent possèdent toutes des arguments pour contrôler la mise en forme et les annotations des graphiques. Les listes complètes de ces arguments varient d'une fonction à l'autre, mais certains arguments sont communs à presque toutes les fonctions graphiques. Nous allons voir ici les arguments les plus utiles.

Les arguments permettant de contrôler la mise en forme sont appelés paramètres graphiques. La plupart peuvent être spécifiés soit dans l'appel à la fonction graphique, soit dans un appel à la fonction `par`. Par contre, certains paramètres graphiques peuvent uniquement être fixés avec `par`.

Un paramètre graphique fourni dans un appel à une fonction graphique est effectif seulement pour le graphique produit, alors qu'un paramètre graphique fourni dans un appel à `par` reste effectif jusqu'à ce que nous le modifions de nouveau. Alors avant de modifier des paramètres graphiques avec `par`, il est bon d'enregistrer les valeurs par défaut des paramètres, comme suit,

```
par.default <- par(no.readonly = TRUE)
```

afin de pouvoir facilement réattribuer ces valeurs par défaut aux paramètres, comme suit,

```
par(par.default)
```

à la fin des nos commandes pour produire un graphique.

Notons que les changements apportés aux paramètres graphiques ne sont pas conservés lors de la fermeture de la session R. À l'ouverture d'une session R, les paramètres graphiques prennent donc leurs valeurs par défaut.

Voici deux tableaux résumant les arguments et paramètres graphiques les plus communs (il y en a beaucoup d'autres !) :

Arguments	Éléments graphiques contrôlé
main	titre
sub	note en bas de page
xlab, ylab	noms des axes
xlim, ylim	étendue des axes
type	type de représentation ("p" = points, "l" = lignes, "b" = les deux, etc.)

Paramètres	Éléments graphiques contrôlé
ann	si FALSE, retire le titre et les noms d'axes
bty	si "n", retire le cadre autour de la zone graphique (autres valeurs acceptées : "o", "l", "7", "c", "u" et "]")
lwd	épaisseur des lignes
lty	type des lignes
pch	symbole pour les points
font	type de <b>police de caractères</b> dans la zone graphique, (1 = normal, 2 = gras, 3 = italique, etc.)
font.main, font.sub, font.axis, font.lab	dans le titre, dans la note en bas de page, dans les marques des axes, dans les noms des axes
family	famille de police de caractères ("serif", "sans", "mono", "symbol", etc.)
cex	<b>taille des caractères</b> dans la zone graphique,
cex.main, cex.sub,	dans le titre, dans la note en bas de page,
cex.axis, cex.lab	dans les marques des axes, dans les noms des axes
col	<b>couleur</b> des éléments de la zone graphique, du titre, de la note en bas de page, des marques des axes, des noms des axes
col.main, col.sub, col.axis, col.lab	<b>couleur</b> de l'intérieur du symbole pour les points lorsque pch prend une valeur entre 21 et 25 (pas la même signification dans par)
bg	
par(mfrow = c( , ))	division de la fenêtre graphique
par(new = TRUE)	superposition de graphiques
par(mar = c( , , , ))	tailles des marges
par(oma = c( , , , ))	tailles des marges externes (par défaut nulles)

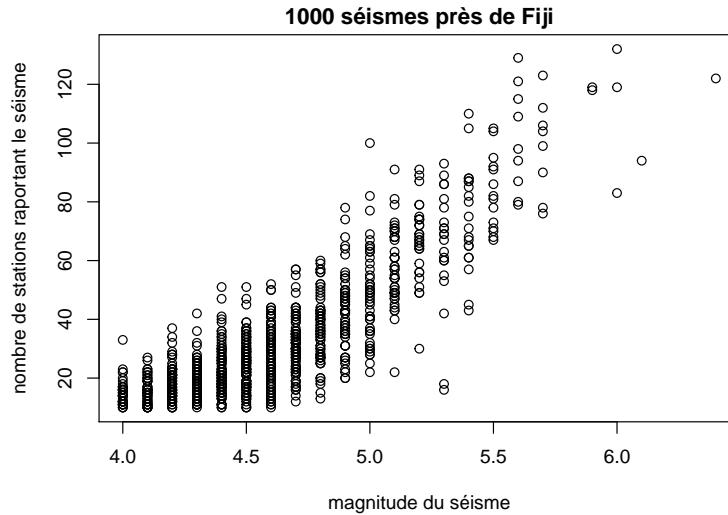
Voici quelques exemples suivis d'un peu plus d'information pour certains arguments et paramètres.

### Exemple : Ajout ou modification des annotations

Ajoutons un titre et modifions les noms des axes du diagramme de dispersion de la variable `stations` de `quakes` en fonction de la variable `mag`.

```
plot(x = quakes$mag, y = quakes$stations,
      xlab = "magnitude du séisme",
```

```
ylab = "nombre de stations rapportant le séisme",
main = "1000 séismes près de Fiji")
```

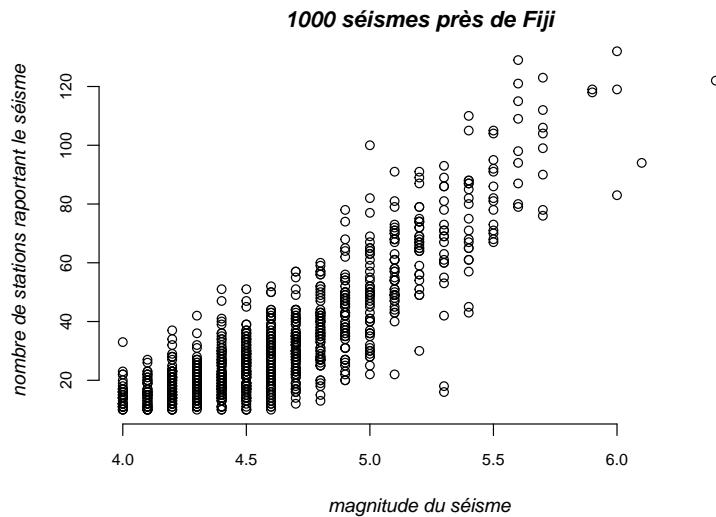


Remarque : Les accents ne causent pas problèmes.

### Exemple : Modification de la mise en forme

Modifions maintenant quelques éléments de la mise en forme du même graphique.

```
plot(x = quakes$mag, y = quakes$stations,
      xlab = "magnitude du séisme",
      ylab = "nombre de stations rapportant le séisme",
      main = "1000 séismes près de Fiji",
      font.lab = 3,      # nom d'axes en italique
      font.main = 4,      # titre en gras italique
      cex.axis = 0.8,    # marques des axes plus petites
      bty="n")           # pas de cadre autour de la zone graphique
```

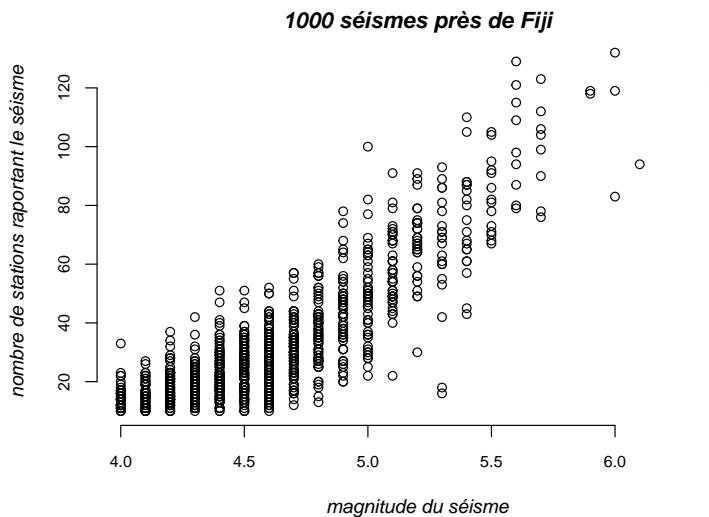


Nous aurions aussi pu modifier ces paramètres graphiques par un appel à la fonction `par` comme suit.

```

par(font.lab = 3,
   font.main = 4,
   cex.axis = 0.8,
   bty="n")
plot(x = quakes$mag, y = quakes$stations,
      xlab = "magnitude du séisme",
      ylab = "nombre de stations rapportant le séisme",
      main = "1000 séismes près de Fiji")
par(par.default)
# en supposant que par.default a été créé avant de modifier des paramètres avec par

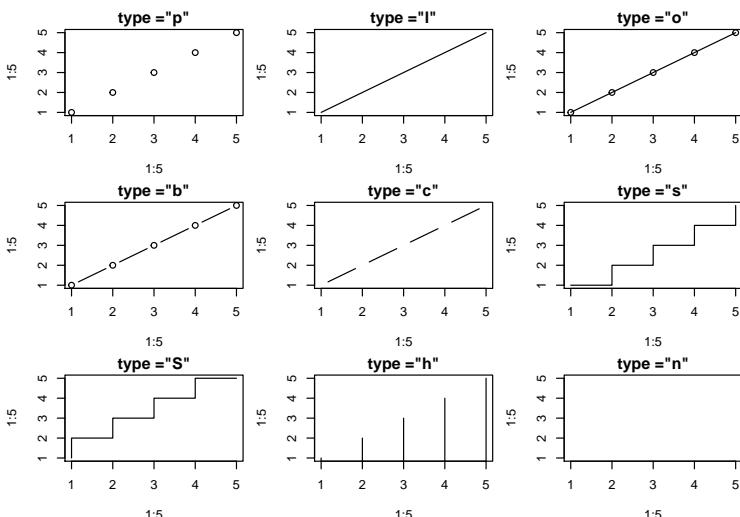
```



Utiliser la fonction `par` est pratique si nous avons plusieurs graphiques à produire avec les mêmes paramètres.

### Types de représentation - argument `type`

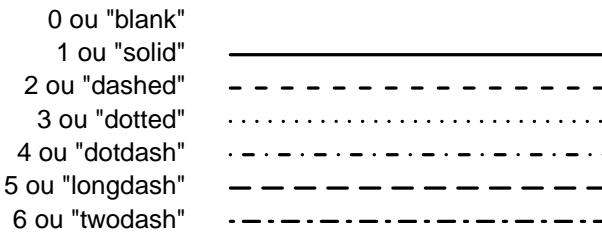
Voici un graphique qui représente toutes les valeurs que peut prendre l'argument `type`.



Ce graphique est une adaptation d'un graphique sur la page web suivante :  
<http://www.statmethods.net/graphs/line.html>.

## Types de lignes - paramètre lty

Voici un graphique qui représente toutes les valeurs que peut prendre le paramètre `lty`.

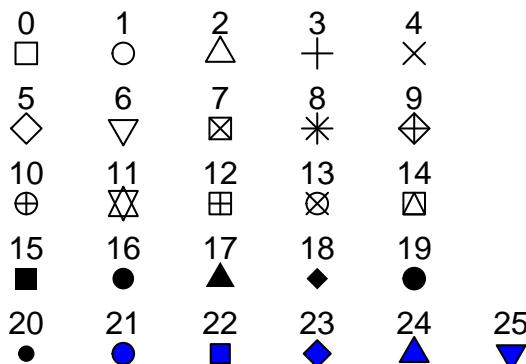


Ce graphique est une adaptation d'un graphique sur la page web suivante :

<http://www.sthda.com/french/wiki/les-different-types-de-traits-dans-r-lty>

## Symboles pour les points - paramètre pch

Voici un graphique qui représente toutes les valeurs numériques que peut prendre le paramètre `pch`. Ce paramètre accepte aussi comme valeur un caractère quelconque.



Ce graphique est une adaptation d'un graphique sur la page web suivante :

<http://www.sthda.com/french/wiki/les-different-types-de-points-dans-r-comment-utiliser-pch>

## Couleurs

Pour connaître toutes les possibilités par rapport aux couleurs dans les graphiques en R, une source d'information très complète est la page web suivante : <http://research.stowers.org/mcm/efg/R/Color/Chart/>

Une couleur peut être spécifiée de différentes façons :

- par une chaîne de caractère contenant un nom de couleur,
- par une chaîne de caractères hexadécimaux de la forme "#rrggbb" ou "#rrggbbaa",
- par un nombre entier référant à une palette de couleur.

### Nom de couleur :

La commande suivante affiche tous les noms de couleurs compris par R (sortie non affichée ici). Il y en a 657 !

`colors()`

Il est possible de visualiser ces couleurs dans le PDF suivant :

<http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf>

### Chaîne de caractères hexadécimaux de la forme "#rrggb" ou "#rrggbbaa" :

Dans une chaîne de caractères hexadécimaux de la forme "#rrggb" ou "#rrggbbaa", les paires des caractères rr, gg, bb et aa sont des digits hexadécimaux spécifiant une valeur entre 00 (minimum) et FF (maximum). Ces digits indiquent respectivement un niveau de rouge, de vert, de bleu et d'opacité. Le niveau d'opacité est facultatif. Par défaut les couleurs sont complètement opaques. Une valeur d'opacité minimale 00 représente une transparence complète et une valeur d'opacité maximale FF représente une opacité complète.

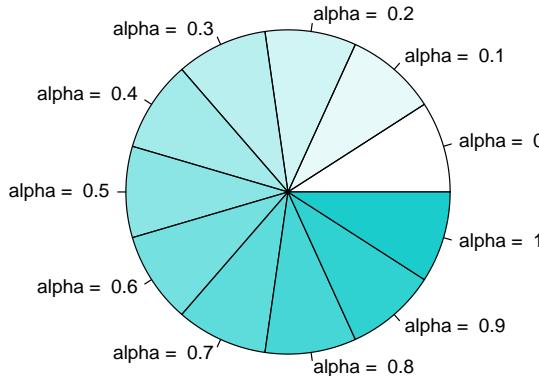
Pour déterminer un code hexadécimal de couleur, les fonctions suivantes sont utiles :

- `rgb` : prend en entrée des niveaux de rouge (`red`), de vert (`green`), de bleu (`blue`) et optionnellement d'opacité (`alpha`), retourne en sortie le code hexadécimal associé ;
- `hsv` : prend en entrée des niveaux de teinte (`hue`), de saturation (`saturation`), de valeur de luminosité (`value`) et optionnellement d'opacité, retourne en sortie le code hexadécimal associé ;
- `hcl` : prend en entrée des niveaux de teinte (`hue`), de chroma (`chroma`), de luminance (`luminance`) et optionnellement d'opacité, retourne en sortie le code hexadécimal associé ;
- `gray` ou `grey` : prend en entrée un niveau de gris entre 0 (noir) et 1 (blanc) et optionnellement d'opacité, retourne en sortie le code hexadécimal associé.

Exemple d'un turquoise avec différents niveau d'opacité :

```
opacite <- rgb(red = 0.1, green = 0.8, blue = 0.8,
                 alpha = seq(from = 0, to = 1, by = 0.1))
opacite

## [1] "#1ACCCC00" "#1ACCCC1A" "#1ACCCC33" "#1ACCCC4D" "#1ACCCC66" "#1ACCCC80"
## [7] "#1ACCCC99" "#1ACCCCB3" "#1ACCCCCC" "#1ACCCCE6" "#1ACCCCF9"
```



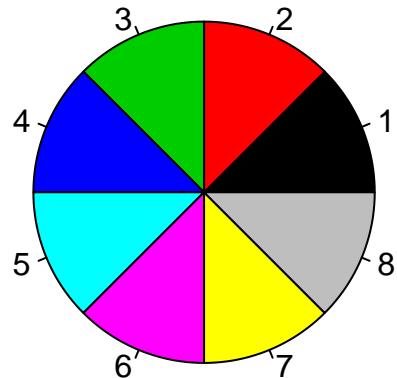
### Nombre entier référant à une palette de couleur :

Une palette de couleur est définie en R et il est possible de spécifier une couleur par un entier référant à une position dans cette palette. Par défaut, la palette de couleur est la suivante :

```
palette()

## [1] "black"     "red"       "green3"    "blue"      "cyan"      "magenta"   "yellow"    "gray"
```

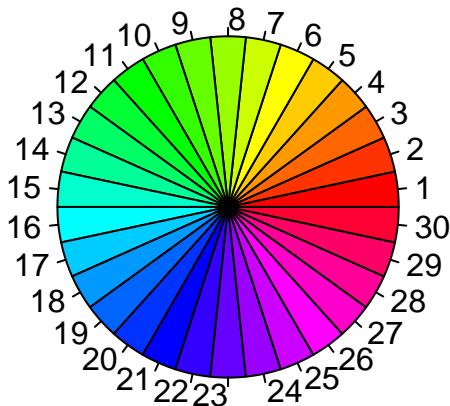
Voici un diagramme en secteur représentant les couleurs de cette palette et les entiers associés.



Il est possible de modifier la palette en lui assignant un nouveau vecteur de noms de couleurs ou de chaînes de caractères hexadécimaux de la forme "#rrggbb" ou "#rrggbbaa" pour représenter des couleurs. Certaines fonction R sont pratiques pour créer de tels vecteurs contenant des dégradés de couleurs. Il s'agit des fonctions : `rainbow`, `heat.colors`, `terrain.colors`, `topo.colors` et `cm.colors`. Par exemple, utilisons la fonction `rainbow` pour modifier la palette de couleurs en un dégradé de 30 couleurs comme suit.

```
palette(rainbow(30))
```

Maintenant, le diagramme en secteur représentant les couleurs de la palette et les entiers associés devient :



Pour ramener la palette de couleurs à ses valeurs par défaut, il faut soumettre la commande suivante.

```
palette("default")
```

Certains packages R offrent des palettes de couleurs supplémentaires, notamment :

- le package `RColorBrewer` : <https://CRAN.R-project.org/package=RColorBrewer>,
- le très populaire package `viridisLite` : <https://github.com/sjmgarnier/viridisLite>

## Ajout d'éléments à un graphique

Nous pouvons ajouter des éléments étape par étape à un graphique en R, en appelant les unes après les autres des fonctions graphiques. Le tableau suivant présente les principales fonctions graphiques qui ajoutent des éléments à un graphique initialisé.

Fonction(s) R	Élément(s) ajouté(s)
<code>points</code> et <code>matpoints</code>	points selon des coordonnées
<code>lines</code> et <code>matlines</code>	segments de droites reliant des points
<code>abline</code>	droites traversant toute la zone graphique
<code>segments</code>	segments de droites entre des paires de coordonnées
<code>arrows</code>	flèches entre des paires de coordonnées
<code>rect</code>	rectangles
<code>polygon</code>	polygones quelconques
<code>legend</code>	légende
<code>text</code>	texte dans la zone graphique
<code>mtext</code>	texte dans la marge
<code>title</code>	titre
<code>axis</code>	axe
<code>box</code>	boîte autour de la zone graphique
<code>qqline</code>	ligne dans un graphique quantile-quantile théorique

Aussi, certaines fonctions de création de graphique peuvent devenir des fonctions d'ajout d'éléments à un graphique grâce à l'argument `add`. C'est le cas notamment des fonctions suivantes :

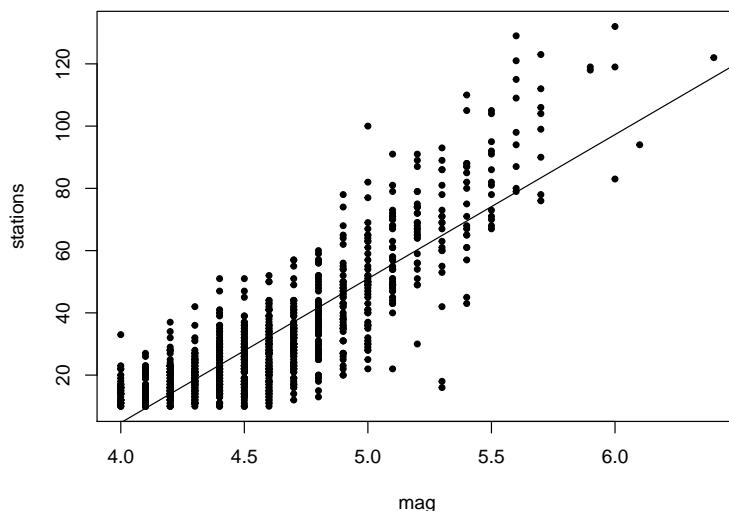
`matplotlib`, `barplot`, `hist`, `boxplot` et `curve`.

En donnant la valeur `TRUE` à l'argument `add` lors de l'appel de ces fonctions, elle ajoute des éléments au graphique de la fenêtre graphique active au lieu de créer un nouveau graphique.

### Exemple : Ajout de lignes

La fonction `abline` permet d'ajouter une droite à un graphique qui traverse toute la zone graphique.

```
lmout <- lm(stations ~ mag, data = quakes) # ajustement d'une droite de régression
plot(stations ~ mag, data = quakes, pch = 20)
abline(lmout)
```



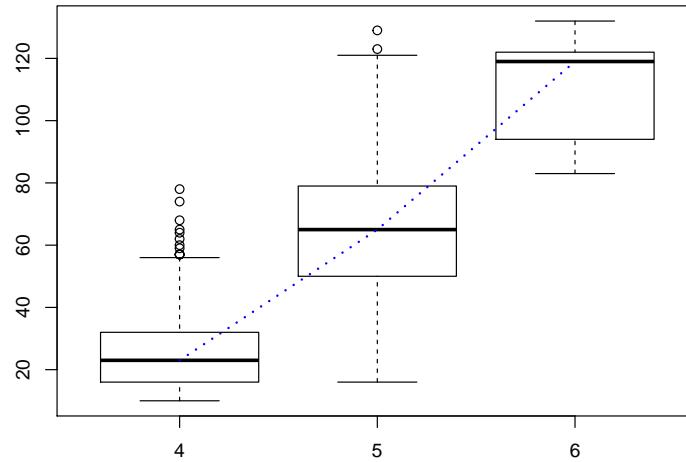
Avec la fonction `lines`, nous pouvons ajouter des segments de droites reliant des points pour lesquels les coordonnées sont fournies.

```

aggout <- aggregate(stations ~ magFactor, data = quakes, FUN = median)
aggout$magFactor <- as.numeric(aggout$magFactor)

boxplot(stations ~ magFactor, data = quakes)
lines(aggout, lty = 3, lwd = 2, col = 4)

```



### Exemple : Ajout de courbes à un histogramme

```

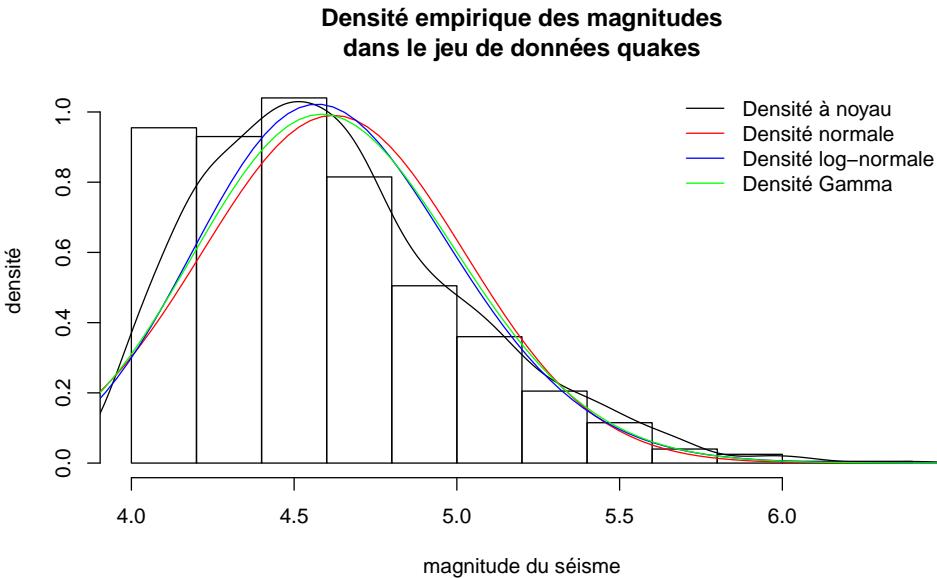
# Initialisation de l'histogramme
hist(quakes$mag, xlab = "magnitude du séisme", ylab = "densité", freq = FALSE,
      main = "Densité empirique des magnitudes\ndans le jeu de données quakes")

# Ajout de la courbe de densité à noyau
lines(density(quakes$mag), xlim = range(quakes$mag))

# Ajout de courbes de densité théoriques
# avec paramètres estimés à partir des données
moy <- mean(quakes$mag)
et <- sd(quakes$mag)
curve(dnorm(x, mean = moy, sd = et),
      add = TRUE, col = "red", xlim = c(3.5, 7))
curve(dlnorm(x, meanlog = mean(log(quakes$mag)), sdlog = sd(log(quakes$mag))),
      add = TRUE, col = "blue", xlim = c(3.5, 7))
curve(dgamma(x, shape = moy^2/et^2, rate = moy/et^2),
      add = TRUE, col = "green", xlim = c(3.5, 7))

# Ajout d'une légende
legend(x = "topright", lty = 1, bty = "n",
       col = c("black", "red", "blue", "green"),
       legend = c("Densité à noyau", "Densité normale",
                 "Densité log-normale", "Densité Gamma"))

```



## Possibilités graphiques spécifiques

### Annotations mathématiques

Des annotations mathématiques sont des caractères spéciaux communs en science, telle que des exposants, des indices, des fractions, des lettres grecques, etc. Tout élément textuel d'un graphique peut en contenir (p. ex. les valeurs données à un argument `main`, `xlab`, `ylab`, `labels`, `legend`, `text`, etc.). La fiche d'aide intitulée `plotmat` énumère toutes les annotations possibles. Ces annotations impliquent une syntaxe particulière, qui ressemble un peu à du LaTeX. Elles doivent être exprimée sous la forme d'une expression R. La fonction `expression` permet de créer une telle expression.

### Exemple :

Dans le graphique précédent, utilisons des notations mathématiques dans la légende.

```
# Initialisation de l'histogramme
hist(quakes$mag, xlab = "magnitude du séisme", ylab = "densité", freq = FALSE,
      main = "Densité empirique des magnitudes\nndans le jeu de données quakes")

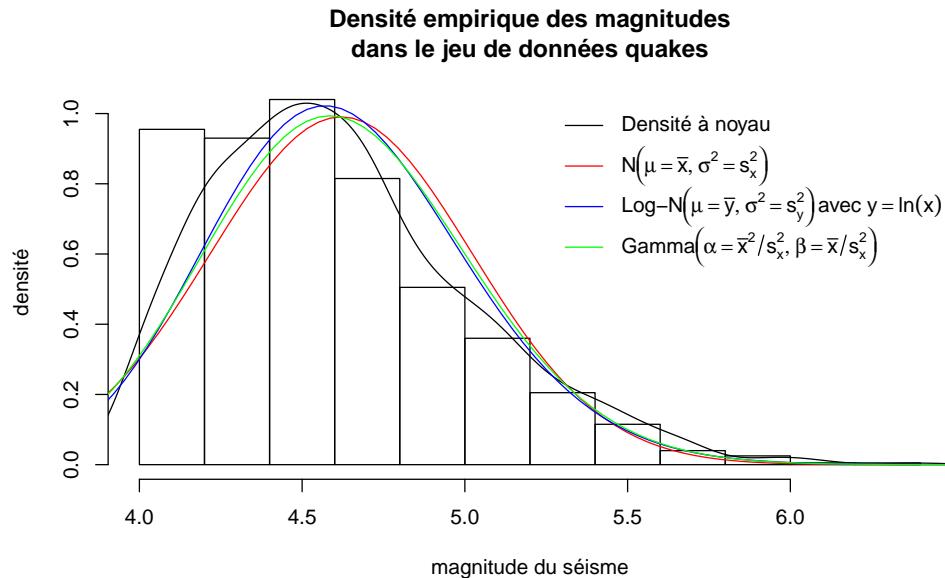
# Ajout de la courbe de densité à noyau
lines(density(quakes$mag), xlim = range(quakes$mag))

# Ajout de courbes de densité théoriques
# avec paramètres estimés à partir des données
moy <- mean(quakes$mag)
et <- sd(quakes$mag)
curve(dnorm(x, mean = moy, sd = et),
      add = TRUE, col = "red", xlim = c(3.5, 7))
curve(dlnorm(x, meanlog = mean(log(quakes$mag)), sdlog = sd(log(quakes$mag))),
      add = TRUE, col = "blue", xlim = c(3.5, 7))
curve(dgamma(x, shape = moy^2/et^2, rate = moy/et^2),
      add = TRUE, col = "green", xlim = c(3.5, 7))
```

```

# Ajout d'une légende
legend(
  x = "topright", lty = 1, bty = "n",
  col = c("black", "red", "blue", "green"),
  legend = c(
    "Densité à noyau",
    expression(
      paste(
        "N",
        bgroup("(", list(mu == bar(x), sigma^2 == s[x]^2), ")")
      )
    ),
    expression(
      paste(
        "Log-N",
        bgroup("(", list(mu == bar(y), sigma^2 == s[y]^2), ")"),
        " avec ",
        y == ln(x)
      )
    ),
    expression(
      paste(
        "Gamma",
        bgroup("(", list(alpha == bar(x)^2/s[x]^2, beta == bar(x)/s[x]^2), ")")
      )
    )
  )
)

```



Dans ce code, les trois derniers éléments du vecteur fourni à l'argument `legend` de la fonction du même nom sont des expressions. Ces éléments sont créés par des appels à la fonction `expression`. Penchons-nous sur un de ces appels pour tenter de mieux le comprendre.

```

expression(
  paste(
    "Log-N",
    bgroup("(,list(mu == bar(y), sigma^2 == s[y]^2),")",
    " avec ",
    y == ln(x)
  )
)

```

L'appel à la fonction `expression` contient un appel à la fonction `paste`. Nous sommes contraints à cette syntaxe parce que le libellé doit contenir à la fois des chaînes de caractères ordinaires (ici "Log-N" et " avec ") et des expressions mathématiques. Mais la fonction `paste` n'agit pas ici tout à fait comme d'habitude. Dans des expressions mathématiques, `paste` ne travaille pas de façon vectorielle et ne possède pas d'arguments. La fonction ne fait que juxtaposer des expressions.

La première expression mathématique dans le `paste` précédent est `bgroup("(,list(mu == bar(y), sigma^2 == s[y]^2),")")`. Cette expression permet de créer l'annotation ressemblant à  $(\mu = \bar{y}, \sigma^2 = s_y^2)$ . Certains caractères de cet élément ont une signification particulière, par exemple :

- `bgroup("(", x, ")")` permet d'encadrer des éléments de parenthèses dont la hauteur s'adapte à la hauteur des éléments ;
- `list(x, y)` sépare des éléments par une virgule (pas la même signification que la fonction `list`)
- `mu` devient la lettre grecque  $\mu$ ,
- `==` devient un signe d'égalité,
- `^2` devient un exposant,
- `[y]` devient un indice,
- etc.

#### Autres exemples :

- <http://vis.supstat.com/2013/04/mathematical-annotation-in-r/>

#### Autres options :

Certains packages permettent de carrément mettre des équations LaTeX dans les annotations de graphiques R, notamment les deux packages suivants.

- <https://CRAN.R-project.org/package=latex2exp>
- <https://CRAN.R-project.org/package=tikzDevice>

### Plusieurs graphiques dans une même fenêtre

Il est possible de diviser la fenêtre graphique en sous-fenêtres. Deux outils sont offerts dans le système de base pour effectuer cette division :

- arguments `mfrow` ou `mfcol` de la fonction `par` : produit une grille de sous-fenêtres de tailles égales ;
- fonction `layout` : permet de contrôler les dimensions des sous-fenêtres.

#### Exemple : Sous-fenêtres de tailles égales - paramètre `mfrow` ou `mfcol` de `par`.

Réunissons plusieurs représentations de la variable `mag` du jeu de données `quakes` dans un seul graphique.

```

# Enregistrement d'une copie des valeurs par défaut des paramètres
# graphiques contrôlés par la fonction par
par.default <- par(no.readonly = TRUE)

```

```

# Modification du paramètre graphique mfrow de façon à diviser la

```

```

# fenêtre graphique en 3 sous-fenêtres, une en dessous de l'autre
par(mfrow = c(3,1))

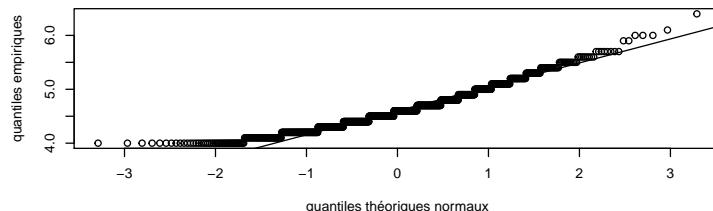
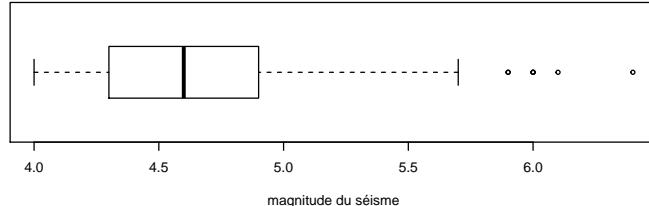
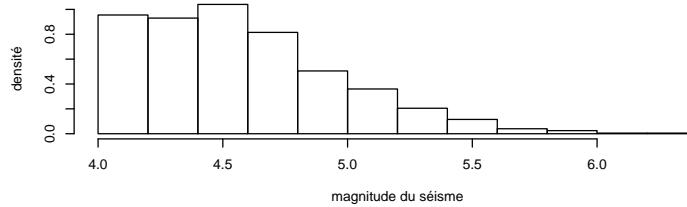
# Graphique dans la première sous-fenêtre
hist(quakes$mag, xlab = "magnitude du séisme", ylab = "densité", freq = FALSE,
     main = "")

# Graphique dans la deuxième sous-fenêtre
boxplot(quakes$mag, horizontal = TRUE, xlab = "magnitude du séisme")

# Graphique dans la troisième sous-fenêtre
qqnorm(quakes$mag, xlab = "quantiles théoriques normaux",
       ylab = "quantiles empiriques", main = "")
qqline(quakes$mag)

# Réattribution des valeurs par défaut aux paramètres graphiques
par(par.default)

```



### Exemple : Sous-fenêtres de tailles inégales - fonction layout

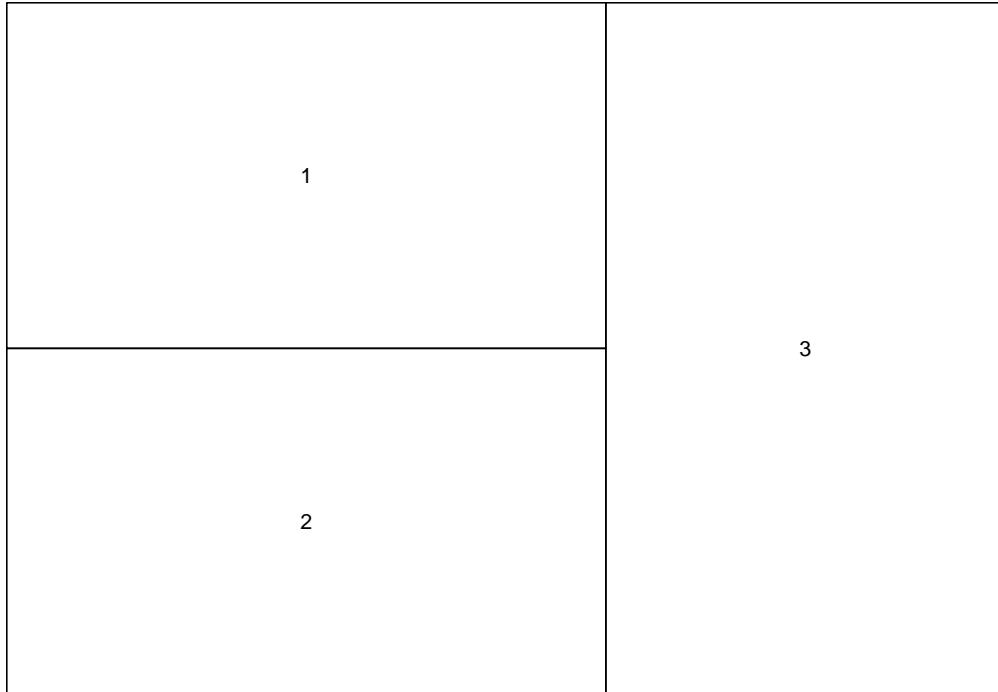
La fonction `layout` découpe la fenêtre graphique en quadrillage. Les hauteurs et largeurs des bandes sont déterminées par les arguments `widths` et `heights`. La matrice à donner en argument `mat` contient un entier non négatif pour chaque sous-fenêtre créé par le quadrillage. Dans les commandes qui suivent l'appel à la fonction `layout`, le premier graphique produit est affiché dans la sous-fenêtre associée au chiffre 1, le deuxième dans la sous-fenêtre associée au chiffre 2 et ainsi de suite. L'attribution du chiffre zéro à une sous-fenêtre signifie qu'aucun graphique ne doit être affiché dans cette sous-fenêtre. L'attribution d'un même entier positif

à plusieurs sous-fenêtres adjacentes permet de les fusionner en une seule sous-fenêtre.

La fonction `layout.show` permet de visualiser les sous-fenêtres créées.

Voici un exemple de découpage en sous-fenêtre avec `layout`.

```
layout(matrix(c(1,2,3,3), nrow = 2), widths = c(3,2))
layout.show(n = 3)
```



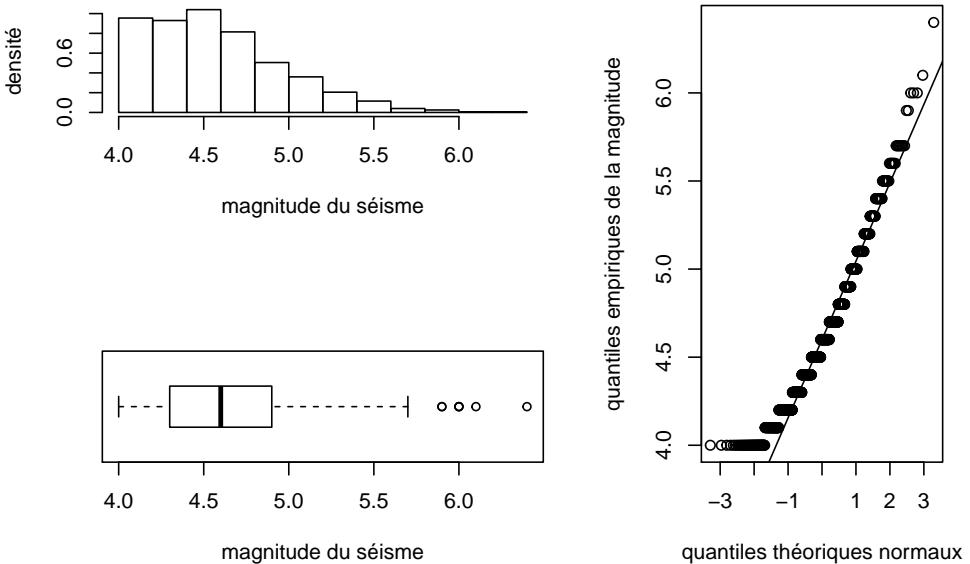
Maintenant, reprenons l'exemple précédent et répartissons les graphiques dans les sous-fenêtre de la disposition que nous venons de créer avec `layout`.

```
# Configuration des sous-fenêtres
layout(matrix(c(1,2,3,3), nrow = 2), widths = c(3,2))

# Graphique dans la première sous-fenêtre
hist(quakes$mag, xlab = "magnitude du séisme", ylab = "densité", freq = FALSE,
     main = "")

# Graphique dans la deuxième sous-fenêtre
boxplot(quakes$mag, horizontal = TRUE, xlab = "magnitude du séisme")

# Graphique dans la troisième sous-fenêtre
qqnorm(quakes$mag, xlab = "quantiles théoriques normaux",
       ylab = "quantiles empiriques de la magnitude", main = "")
qqline(quakes$mag)
```



Nous pourrions améliorer ce graphique en ajustant les marges. Nous pourrions également mettre des sous-titres et un titre global.

```
# Configuration des sous-fenêtres
layout(matrix(c(1,2,3,3), nrow = 2), widths = c(3,2))

# Pour créer un espace pour le titre global
par(oma = c(0, 0, 3, 0))

# Graphique dans la première sous-fenêtre
par(mar = c(3.1, 4.1, 2.1, 2.1), cex.main = 1)
hist(quakes$mag, ylab = "densité", freq = FALSE, main = "Histogramme")

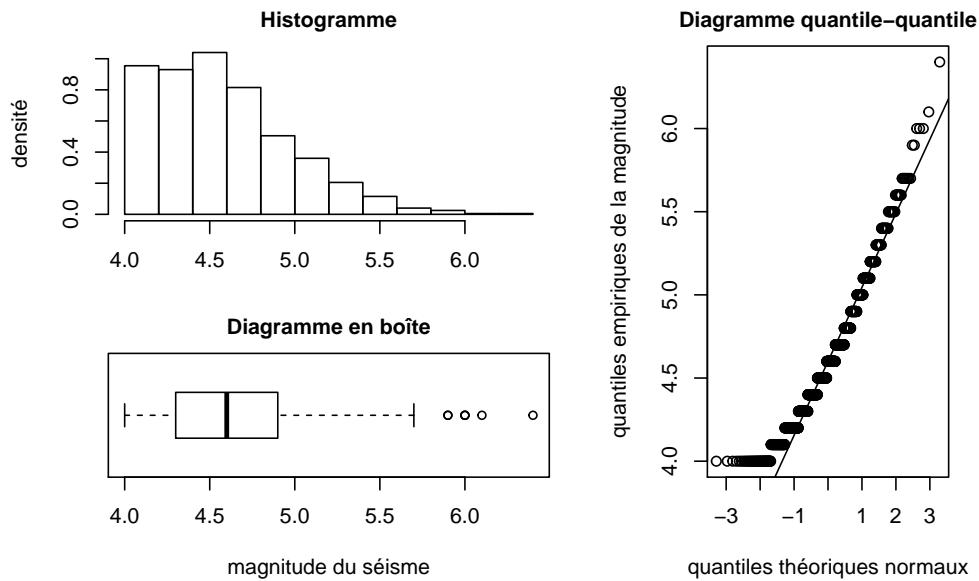
# Graphique dans la deuxième sous-fenêtre
par(mar = c(5.1, 4.1, 2.1, 2.1))
boxplot(quakes$mag, horizontal = TRUE, xlab = "magnitude du séisme",
        main = "Diagramme en boîte")

# Graphique dans la troisième sous-fenêtre
qqnorm(quakes$mag, xlab = "quantiles théoriques normaux",
       ylab = "quantiles empiriques de la magnitude",
       main = "Diagramme quantile-quantile")
qqline(quakes$mag)

# Ajout du titre global
mtext("Densité empirique des magnitudes dans le jeu de données quakes",
      outer = TRUE, cex = 1.2, line = 1.5)

# Réattribution des valeurs par défaut aux paramètres graphiques
par(par.default)
```

## Densité empirique des magnitudes dans le jeu de données quakes



### Autres exemples d'utilisation de layout :

- <http://www.statmethods.net/advgraphs/layout.html>
- [http://sas-and-r.blogspot.ca/2012/09/example-103-enhanced-scatterplot-with.html?utm\\_source=feedburner&utm\\_medium=feed&utm\\_campaign=Feed%3A+SASandR+\(SAS+and+R\)&utm\\_content=Google+Reader](http://sas-and-r.blogspot.ca/2012/09/example-103-enhanced-scatterplot-with.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+SASandR+(SAS+and+R)&utm_content=Google+Reader)

## Aspects techniques

Créer un graphique en R signifie soumettre un programme R qui génère le graphique. Pour produire de nouveau le même graphique, il suffit de conserver le programme permettant de générer le graphique et de le soumettre de nouveau dans la console. Le fait de produire le graphique par un programme plutôt que par un menu dans lequel nous sélectionnons des options permet d'automatiser le travail et de sauver du temps lorsque nous avons à produire plusieurs graphiques similaires.

### Fenêtres graphiques

En R, les graphiques sont par défaut tous créés dans la même fenêtre. RStudio conserve un historique des graphiques, ce qui permet de réafficher un graphique produit antérieurement, mais au cours d'une même session R.

Nous pouvons aussi choisir d'ouvrir une nouvelle fenêtre graphique avec la commande :

- `windows()` sur Windows,
- `quartz()` sur Mac OS X / OS X / macOS,
- `X11()` sur Unix / Linux.

Les dimensions de la fenêtre graphique courante sont les suivantes

```
par("din")
```

```
# ou  
dev.size()
```

```
## [1] 6.5 4.5
```

Note : Ce paramètre graphique n'est pas modifiable (R.O. = *Read Only* dans la documentation de la fonction `par`).

Comment modifier les dimensions de la fenêtre graphique ?

1. Redimensionner la fenêtre graphique avec la souris. En Windows, dans une fenêtre ouverte avec `windows()`, les modes de redimensionnement sont les suivants :
  - mode R : dimensions fenêtres = dimensions du graphique ;
  - ajuster à la fenêtre : dimensions du graphique ajustées aux dimensions de la fenêtre, mais ratio largeur/hauteur conservé ;
  - taille fixe : dimensions du graphique fixes, donc non affectées par les dimensions de la fenêtre.
2. Ouvrir une nouvelle fenêtre avec X11 (Unix / Linux), `windows` (Windows) ou `quartz` (Mac) et spécifier ses dimensions avec les arguments `width` et `height`. Exemple :

```
windows(width = 10, height = 7.5)
```

Commandes utiles :

- `dev.list()` : pour voir la liste de toutes les fenêtres graphiques ouvertes,
- `dev.cur()` : pour connaître la fenêtre graphique courante,
- `dev.off()` : pour fermer la fenêtre graphique courante (peut aussi être fermée avec la souris).

Pour d'informations sont offertes dans la fiche d'aide R nommée `dev`.

## Enregistrer un graphique

En RStudio, nous pouvons enregistrer un graphique dans la fenêtre graphique par le menu Export. Nous pouvons aussi le faire, comme dans l'exemple suivant, avec les fonctions `bmp`, `postscript`, `pdf`, `png`, `tiff`, `svg` ou `jpeg`, selon le format désiré.

```
# Ouverture de la connexion avec un fichier
png("test.png")

# Code pour créer le graphique
plot(x = quakes$mag, y = quakes$stations)
# potentiellement plusieurs lignes de code ici

# Fermeture de la connexion avec le fichier
dev.off()
```

En fait, ces fonctions redirigent l'affichage de graphiques vers un fichier. La commande `dev.off()` est nécessaire pour mettre un terme à la communication entre R et le fichier ouvert.

## Autres fonctions et trucs utiles

À titre de référence, voici une liste de quelques autres fonctions utiles avec des graphiques produits à l'aide du système de base en R.

- `locator` : pour identifier avec la souris une position ;
- `identify` : pour identifier avec la souris des observations ;
- `clip` : pour restreindre la zone d'ajouts dans un graphique ;
- `jitter` : pour ajouter un peu de bruits à des valeurs numériques, ce qui peut permettre de visualiser plus facilement des observations superposées (exemple : <https://blog.jumpingrivers.com/posts/2018/2018-01-24-base-r-graphics/>).

Aussi, il est bon de savoir que plusieurs fonctions graphiques de base peuvent aussi retourner des valeurs en plus de créer un graphique. Les valeurs renvoyées contiennent, par exemple :

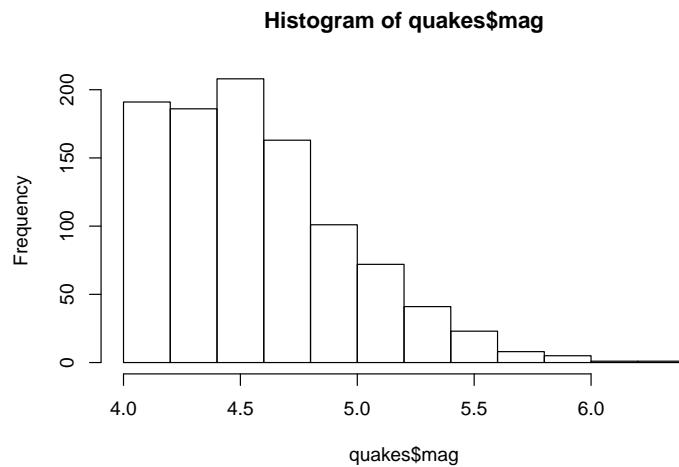
- les coordonnées de certains éléments dans le graphique, par exemple :
  - dans un histogramme produit avec `barplot` : les coordonnées des limites et des centres des bâtons (en d'autres mots des intervalles),
  - dans un diagramme en bâtons produit avec `barplot` : les coordonnées centrales des bâtons sur l'axe de la variable catégorique ;
  - etc. ;
- des statistiques calculées pour produire le graphique, par exemple :
  - dans un histogramme produit avec `hist` : les fréquences des observations dans les intervalles,
  - dans des diagrammes en boîte produits par la fonction `boxplot` : les statistiques représentées dans les diagrammes,
  - etc.

### Exemples :

Exemple de valeurs renvoyées par la fonction `hist` (toutes décrites dans la [fiche d'aide](#)).

```
out_hist <- hist(quakes$mag)
out_hist

## $breaks
## [1] 4.0 4.2 4.4 4.6 4.8 5.0 5.2 5.4 5.6 5.8 6.0 6.2 6.4
##
## $counts
## [1] 191 186 208 163 101  72  41  23   8   5   1   1
##
## $density
## [1] 0.955 0.930 1.040 0.815 0.505 0.360 0.205 0.115 0.040 0.025 0.005
## [12] 0.005
##
## $mids
## [1] 4.1 4.3 4.5 4.7 4.9 5.1 5.3 5.5 5.7 5.9 6.1 6.3
##
## $xname
## [1] "quakes$mag"
##
## $equidist
## [1] TRUE
##
## attr(),"class")
## [1] "histogram"
```

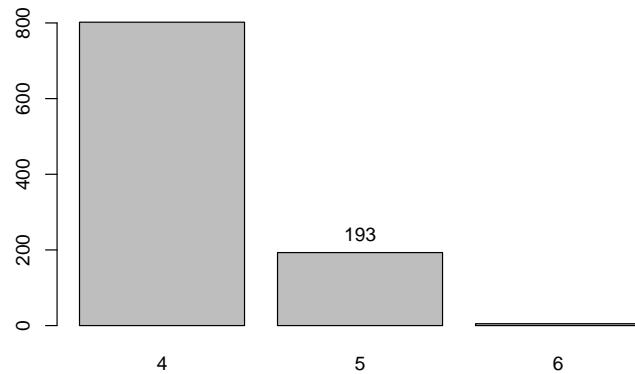


Exemple d'utilisation des valeurs renvoyées par la fonction `barplot`.

```
# Calcul des fréquences
freq <- table(quakes$magFactor)
# Production du diagramme en bâtons
out_barplot <- barplot(freq)
out_barplot

##      [,1]
## [1,]  0.7
## [2,]  1.9
## [3,]  3.1

# Ajout de texte au-dessus du deuxième bâton
text(x = out_barplot[2, 1], # coordonnée en x du centre du bâton tirée de out_barplot
      y = freq[2],
      labels = freq[2],
      pos = 3)
```



## Résumé et point de vue

La fonction générique `plot` choisit un bon type de graphique à produire selon le **nombre** et la **nature des variables**.

Sinon, les fonctions graphiques de base en R font très peu de choix pour l'utilisateur.

Les graphiques initialisés sont typiquement minimalistes. Nous y ajoutons ce que nous voulons. Et les possibilités sont presque infinies !

- Avantage : Nous avons le plein contrôle sur l'apparence du graphique.
- Désavantage : Il faut parfois travailler fort pour arriver à nos fins.

## Faiblesse du système graphique de base

Les représentations multivariées ne sont pas simples à produire. Il n'est pas toujours facile, avec les fonctions graphiques de base, de représenter trois variables et plus sur le même graphique.

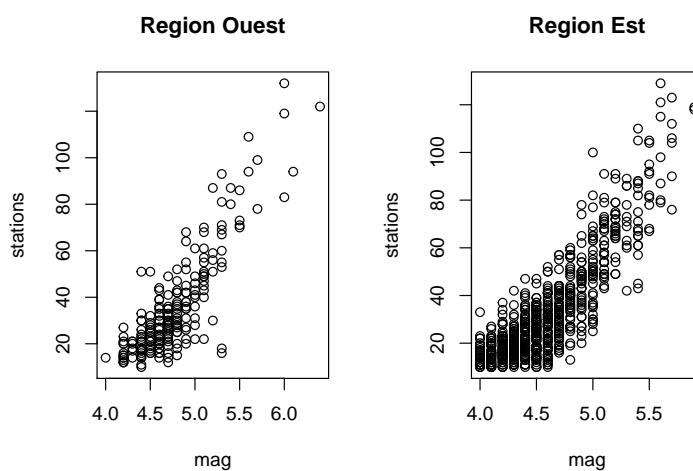
### Exemple - Représenter deux variables numériques et une variable catégorique

Par exemple, tentons de représenter la relation entre les variables `stations` et `mag` de `quakes` en fonction du facteur `region`.

#### Option 1 : Sous-fenêtres selon la variable catégorique

Graphiques par niveau du facteur placés côte-à-côte.

```
par(mfrow = c(1,2))
plot(stations ~ mag, data = quakes, subset = region == "Ouest", main = "Region Ouest")
plot(stations ~ mag, data = quakes, subset = region == "Est", main = "Region Est")
par(par.default)
```



Il faudrait idéalement :

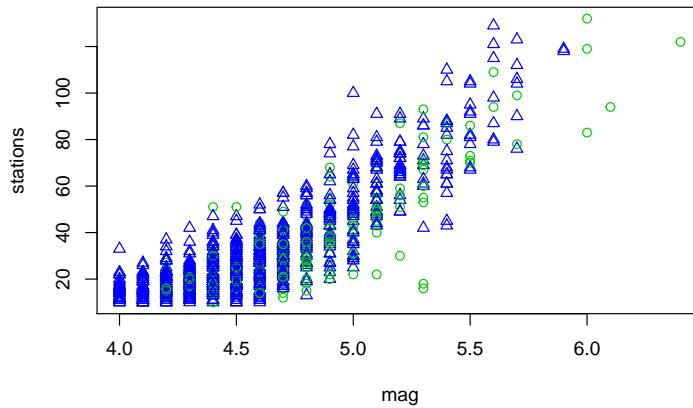
- ajuster les axes afin que tous les graphiques couvrent la même région,
- éviter la redondance dans les noms des axes.

## Option 2 : Superposition avec aspect distinctif selon la variable catégorique

Faisons varier le symbole utilisé pour les points et sa couleur selon le niveau du facteur `region`.

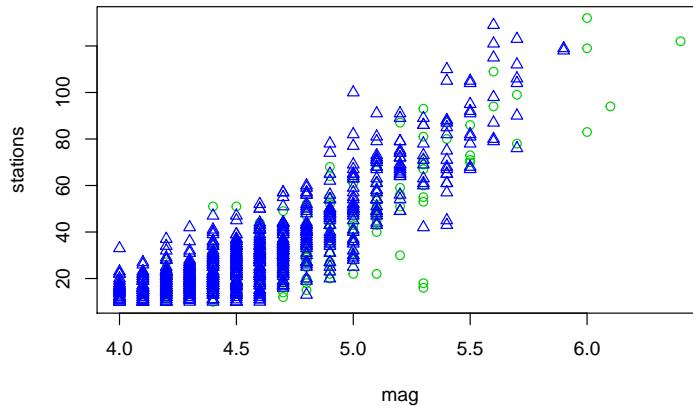
Nous pouvons donner aux arguments graphiques un vecteur de valeurs, plutôt qu'une seule valeur. Ce vecteur doit être de même longueur que le nombre d'observations à représenter (sinon il sera recyclé ou tronqué).

```
plot(stations ~ mag, data = quakes,  
      pch = as.numeric(region), col = as.numeric(region) + 2)
```



Nous pouvons aussi ajouter les points pour chaque groupe dans des étapes séparées.

```
# Initialisation d'un graphique vide, mais avec les bonnes étendues  
plot(stations ~ mag, data = quakes, type = "n")  
# Ajout des points pour la région Ouest  
points(stations ~ mag, data = quakes, subset = region == "Ouest", pch = 1, col = 3)  
# Ajout des points pour la région Est  
points(stations ~ mag, data = quakes, subset = region == "Est", pch = 2, col = 4)
```

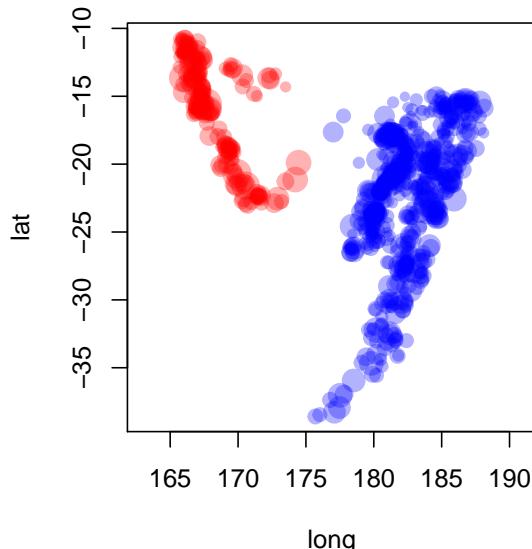


Il faudrait ajouter une légende pour identifier à quoi réfère les couleurs (et les symboles).

## Exemple - Représenter trois variables numériques et une variable catégorique

Tentons maintenant de représenter dans un même graphique les variables suivantes de `quakes` : `lat`, `long`, `mag` et `region`.

```
par(pty = "s")
# pty = "s" permet d'avoir une région graphique carrée
plot(lat ~ long, data = quakes, asp = 1,
     # asp = 1 permet d'avoir des axes sur la même échelle,
     # ce qui est préférable ici, car les variables sont des coordonnées géographiques
     cex = 3*(mag - min(mag))/(max(mag) - min(mag)) + 1, pch = 20,
     # la taille du symbole dépend de la valeur de la variable mag
     col = ifelse(region == "Ouest", rgb(1, 0, 0, 0.3), rgb(0, 0, 1, 0.3)))
     # la couleur du symbole dépend du facteur région
     # des couleurs transparentes sont utilisées
par(par.default)
```



Il faudrait ajouter une légende pour identifier à quoi réfère les couleurs différents (`region`) et les tailles différentes (`mag`).

Nous avons réussis à créer les deux dernières représentations multivariables, mais les autres systèmes graphiques en R permettent de créer ce genre de graphiques plus facilement que le système de base.

## Autres systèmes graphiques en R

Deux packages offrent une alternative aux fonctions graphiques de base. Il s'agit des packages `lattice` et `ggplot2`. Ils sont présentés dans cette section.

Tout d'abord, voici une comparaison rapide des différents systèmes graphiques en R.

1. Système graphique de base (packages `graphics` et `grDevices`)
  - fait partie de l'installation de base de R depuis le début  
(mais a été amélioré au fil du temps).

## 2. Package `lattice`

- publié pour la première fois en 2001 ;
- développé par Deepayan Sarkar ;
- basé sur le « trellis » système de S-PLUS, qui implémente le système graphique présenté dans : Cleveland, W. S. (1993). *Visualizing data*. Hobart Press.

## 3. Package `ggplot2`

- publié pour la première fois en 2006 sous le nom de `ggplot` ;
- amélioré de façon importante et renommé `ggplot2` en 2007 ;
- développé par Hadley Wickham (créateur) et Winston Chang ;
- basé sur le système graphique présenté dans : Wilkinson, L. (2005). *The grammar of graphics*, Second Edition. Springer.

**Note :** À partir d'ici, le texte est plutôt minimaliste. Le matériel est tiré d'une présentation que j'ai donné. Je n'ai pas eu le temps d'enrober les informations de phrases complètes.

## Package `lattice`

Force = graphiques conditionnels à la valeur d'un ou de plusieurs facteurs

Caractéristiques :

- Les fonctions prennent idéalement une formule en entrée.
  - L'opérateur `|` sert à créer des panneaux (*panel*) de graphiques côte-à-côte,
    - \* chacun des sous-graphique est conditionnel à la valeur de facteur(s), il représente donc seulement le sous-ensemble des observations ayant une modalité particulière pour ce(s) facteur(s).
- L'argument `groups` sert à superposer des éléments avec un aspect (couleur, forme, taille, etc.) qui varie selon les niveaux d'un facteur.
- Un graphique est créé par un seul appel à une fonction du package.
  - Les fonctions ont beaucoup de paramètres.
  - Il n'est pas possible d'ajouter des éléments à un graphique comme dans le système de base.

Semble avoir été quelque peu éclipsé par `ggplot2`.

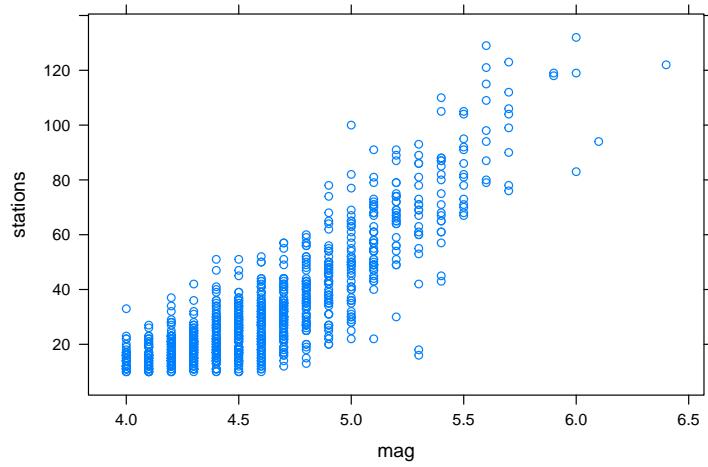
Je ne présente que quelques exemples, avec la fonction `xyplot` servant à créer des diagrammes de dispersion.

### Exemples : Diagrammes de dispersion - fonction `xyplot`

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

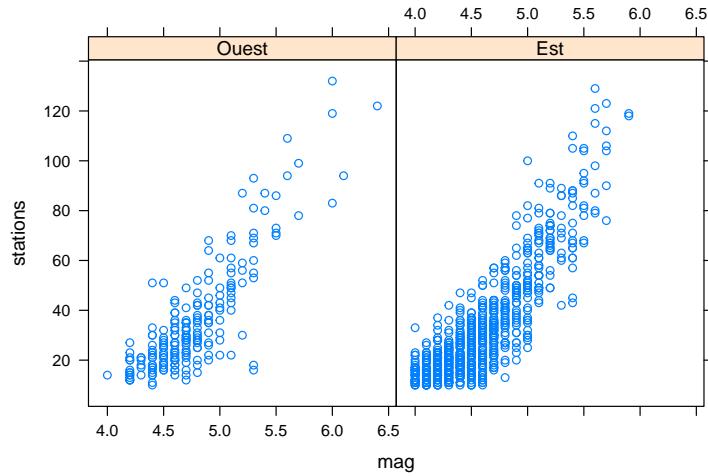
#### Diagramme de dispersion

```
xyplot(stations ~ mag, data = quakes)
```



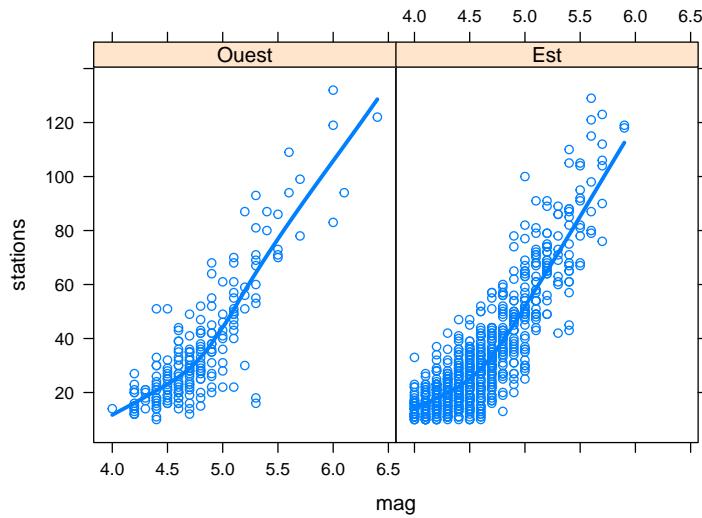
### Diagrammes de dispersion juxtaposés

```
xyplot(stations ~ mag | region, data = quakes)
```



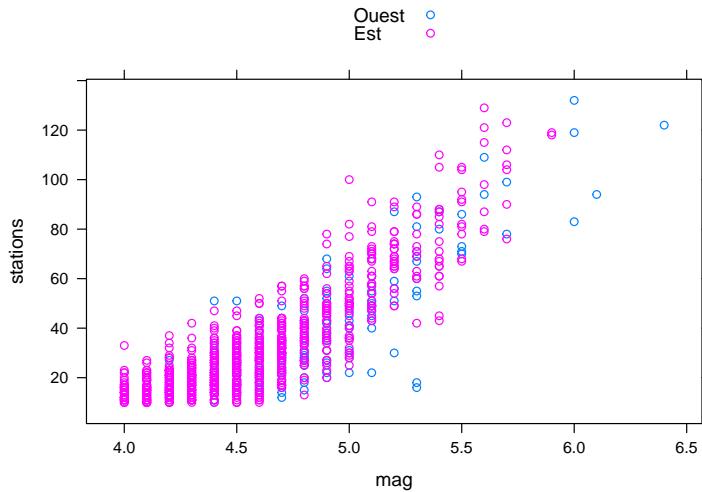
### Ajout d'une courbe de lissage :

```
xyplot(stations ~ mag | region, data = quakes,
       type = c("p", "smooth"), lwd = 3)
```



### Diagrammes de dispersion superposés

```
xypplot(stations ~ mag, data = quakes, groups = region, auto.key = TRUE)
```



L'argument `auto.key` permet d'ajouter automatiquement une légende.

## Package `ggplot2`

Objectifs du créateur :

- Reprendre les forces de chacun des systèmes graphiques R précédents :
  - système de base : création de graphiques par couches (ajouts séquentiels d'éléments) ;
  - package `lattice` : représentations multivariables simples.
- Améliorer les systèmes :
  - automatiser certaines configurations graphiques (p. ex. les légendes),
  - faciliter l'ajout de transformations statistiques communes (p. ex. courbes de lissage, barres d'erreur),
  - fournir une esthétique par défaut pensée de façon à transmettre plus efficacement les informations contenues dans le graphique (p. ex. choix de couleur).

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.

## La grammaire graphique

Principes de base de la grammaire graphique :

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

- **ggplot** : initialisation d'un objet de classe **ggplot**
- **qplot** : initialisation rapide (**q** pour *quick*) d'un objet **ggplot**
  - Utilisation plus intuitive que **ggplot** pour des gens familiers avec **plot**.
  - N'offre cependant pas toutes les possibilités de **ggplot**.
  - *Ne sera pas couvert ici.*
- **+** : 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 des propriétés visuelles et des variables
- **ggsave** : enregistrement d'un graphique

Avec **ggplot2**,

**produire un graphique = afficher un objet ggplot.**

## Code gabarit minimalisté

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

- source : <http://r4ds.had.co.nz/data-visualisation.html#a-graphing-template>
- éléments entre < et > à remplacer :
  - <DATA> = jeu de données, obligatoirement stocké dans un **data frame**, 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>) ;

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

- à la fonction `ggplot` : leur valeur est alors commune à toutes les 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.

### Syntaxes équivalentes au gabarit minimaliste

```
# Version « tout sur la même ligne » du gabarit :
ggplot(data = <DATA>) + <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

Étant donné que le graphique produit par ce gabarit ne comprend qu'une seule couche, les syntaxes suivantes sont 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>()
```

### Objet de classe `ggplot`

Un graphique produit en `ggplot2` est en fait un objet de classe `ggplot`.

Il faut commencer par l'initialiser :

```
monGraph <- ggplot(data = <DATA>)
```

puis y ajouter des couches :

```
monGraph <- monGraph + <GEOM_FUNCTION>(mapping = aes(<MAPPINGS>))
```

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

```
str(monGraph)
summary(monGraph)
```

et enfin de produire un graphique à partir de l'objet, simplement en l'affichant :

```
monGraph # ou print(monGraph)
```

### Quelques fonctions de type `geom_`

Fonction	Type de graphique	Élément(s) ajouté(s)
<code>geom_point</code>	diagramme de dispersion	points selon des coordonnées
<code>geom_line</code>	diagramme en lignes	segments de droites reliant des points
<code>geom_bar</code>	diagramme en bâtons	barres disjointes, de hauteurs spécifiées ou calculées = fréquences des niveaux d'un facteur
<code>geom_histogram</code>	histogramme	barres collées, de hauteurs calculées = fréquences d'observations d'une variable numérique tombant dans des intervalles joints ( <i>bin</i> )
<code>geom_boxplot</code>	diagramme en boîte	<i>boxplots</i>
<code>geom_density</code>	courbe de densité à noyau	courbe de la densité estimée par noyau ( <i>kernel density</i> )
<code>geom_qq</code>	diagramme quantile-quantile théorique	points pour les couples de quantiles empiriques et théoriques

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

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

- fonctions `ggttitle`, `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 `theme_` : modifier des configurations reliées à l'apparence du graphique ;
- fonctions de type `scale_` : modifier les échelles de certaines propriétés visuelles (p. ex. couleurs, formes, tailles, etc.)
- fonctions de type `facet_` : créer des panneaux (*panel*) de graphiques côte-à-côte
  - chacun des sous-graphique est conditionnel à la valeur de facteur(s), il représente donc seulement le sous-ensemble des observations ayant une modalité particulière pour ce(s) facteur(s) ;
- fonctions de type `stat_` : ajouts d'éléments tirés d'un calcul mathématique ou statistique, p. ex. :
  - fonction `stat_function` pour tracer une fonction mathématique.

Il y a beaucoup d'autres fonctions ! <http://ggplot2.tidyverse.org/reference/>

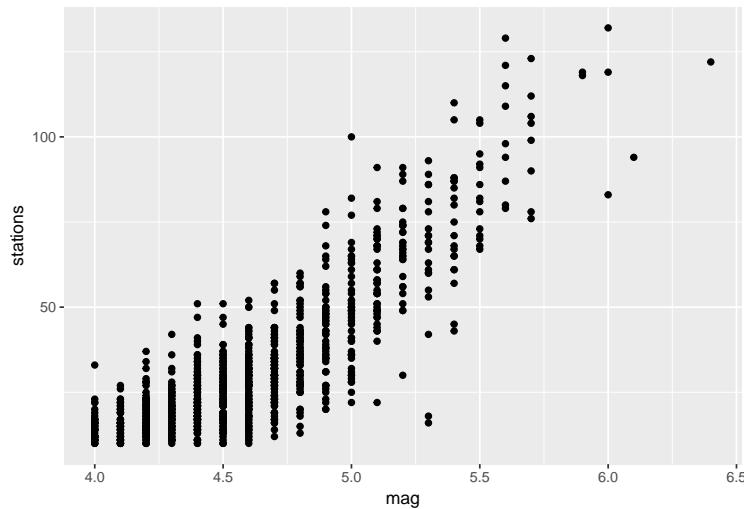
## Exemples simples

Production de graphiques équivalent à certains de ceux produits avec le R de base

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

### Diagramme de dispersion - fonction `geom_point`

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



### Observation d'un objet de type `ggplot`

Création de l'objet

```

diag <- ggplot(data = quakes) + geom_point(mapping = aes(x = mag, y = stations))
# ou
diag <- ggplot(data = quakes)
diag <- diag + geom_point(mapping = aes(x = mag, y = stations))

```

Contenu de l'objet

```

str(diag)
# sortie non affichée, car trop longue

summary(diag)

```

```

## data: lat, long, depth, mag, stations, magFactor, region [1000x7]
## facetting: <ggproto object: Class FacetNull, Facet>
##   compute_layout: function
##   draw_back: function
##   draw_front: function
##   draw_labels: function
##   draw_panels: function
##   finish_data: function
##   init_scales: function
##   map: function
##   map_data: function
##   params: list
##   render_back: function
##   render_front: function
##   render_panels: function
##   setup_data: function
##   setup_params: function
##   shrink: TRUE
##   train: function
##   train_positions: function
##   train_scales: function
##   vars: function
##   super:  <ggproto object: Class FacetNull, Facet>
## -----
## mapping: x = mag, y = stations
## geom_point: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity

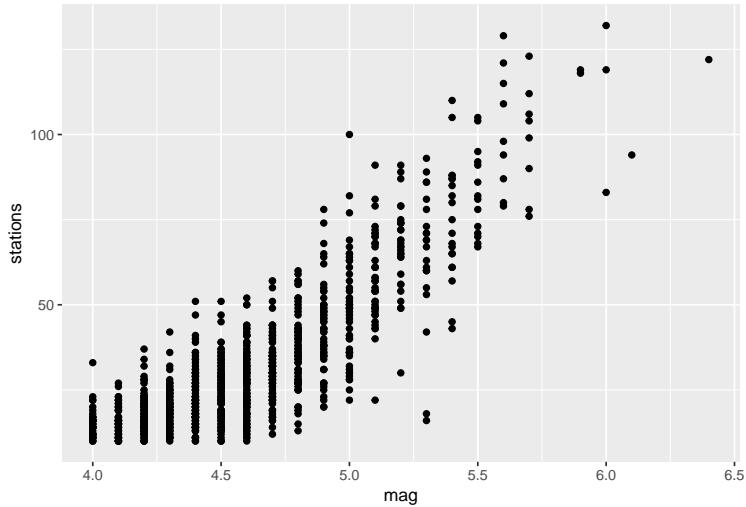
```

Production d'un graphique à partir de l'objet = affichage de l'objet

```

diag
# ou
# print(diag)

```



Avec la commande suivante, on produit le même graphique, mais on ne l'enregistre pas dans un objet R.

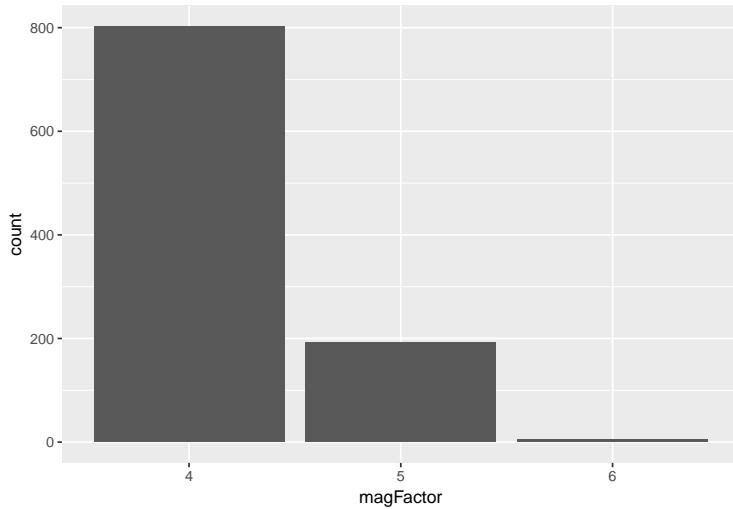
```
ggplot(data = quakes) + geom_point(mapping = aes(x = mag, y = stations))
```

### Diagramme en secteurs

Nous allons y revenir plus loin, pas si simple.

### Diagramme en bâtons - fonction geom\_bar

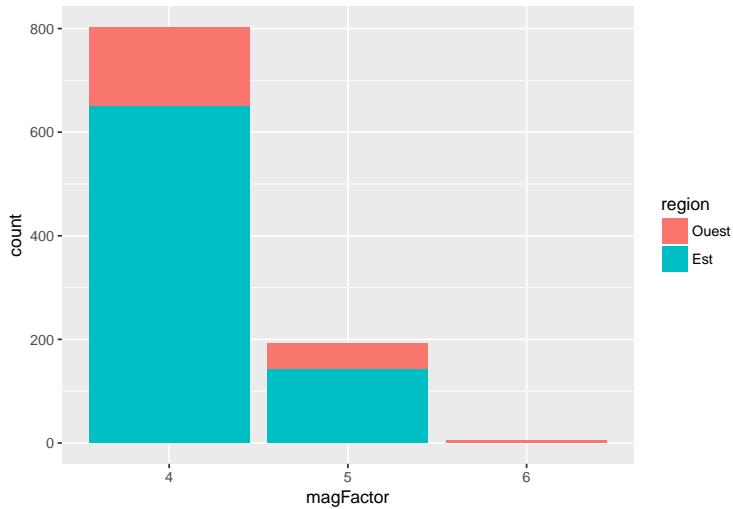
```
ggplot(data = quakes) + geom_bar(mapping = aes(x = magFactor))
```



**Remarque :** Pas besoin d'utiliser `table` pour calculer les fréquences.

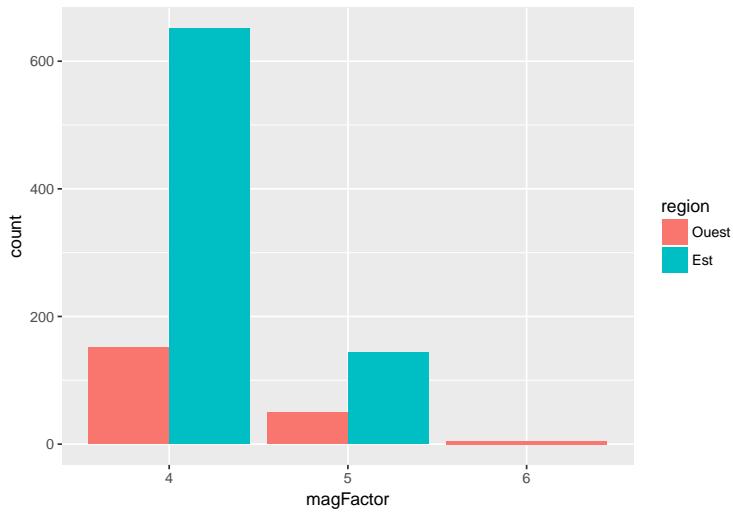
### Deux variables - bâtons empilés

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



### Deux variables - bâtons groupés

```
ggplot(data = quakes) +
  geom_bar(mapping = aes(x = magFactor, fill = region), position = "dodge")
```



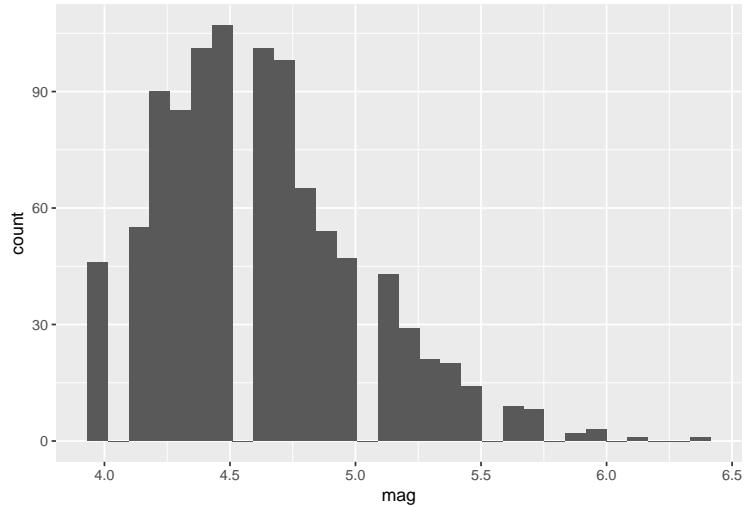
### Diagramme en mosaïque

Pas de fonction conçue pour produire ce type de graphique (mais on trouve des solutions sur le web).

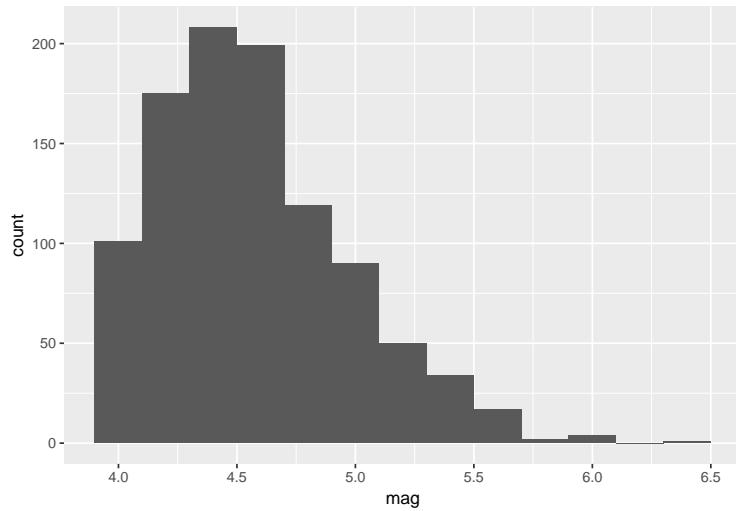
### Histogramme - fonction `geom_histogram`

```
ggplot(data = quakes) + geom_histogram(mapping = aes(x = mag))
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



```
ggplot(data = quakes) +  
  geom_histogram(mapping = aes(x = mag), binwidth = 0.2)
```



### Deux variables - diagrammes superposés

```
ggplot(data = quakes) +  
  geom_histogram(mapping = aes(x = mag, fill = region), binwidth = 0.2, alpha = 0.5)
```

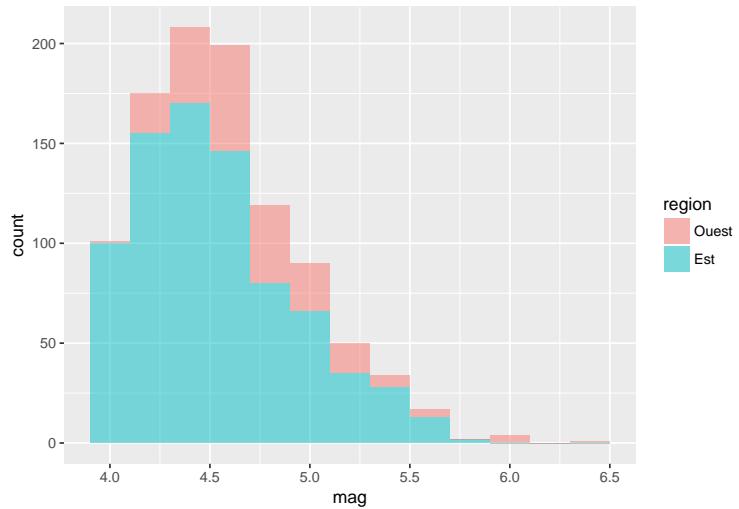
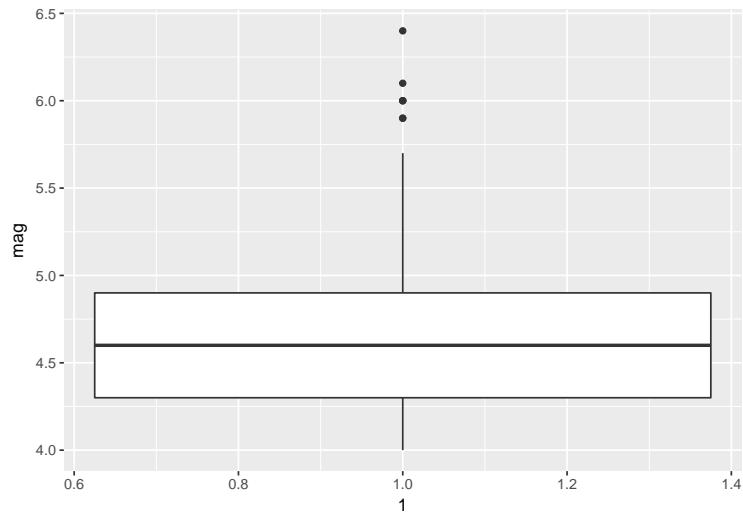


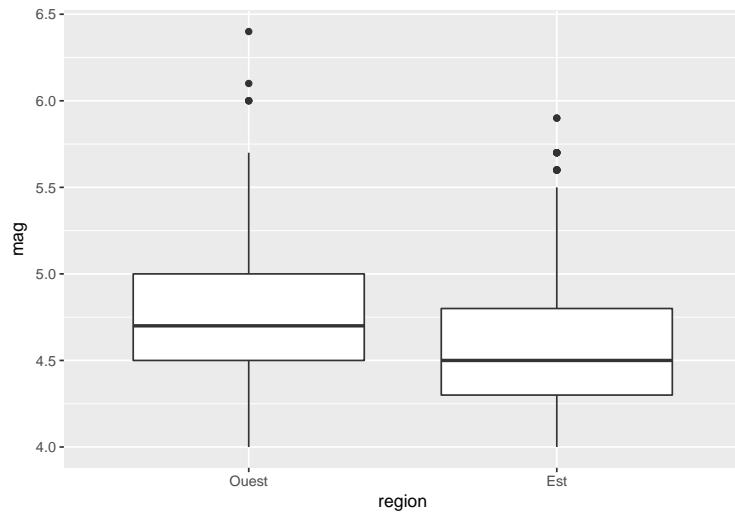
Diagramme en boîte - fonction `geom_boxplot`

```
ggplot(data = quakes) +
  geom_boxplot(mapping = aes(x = 1, y = mag))
```



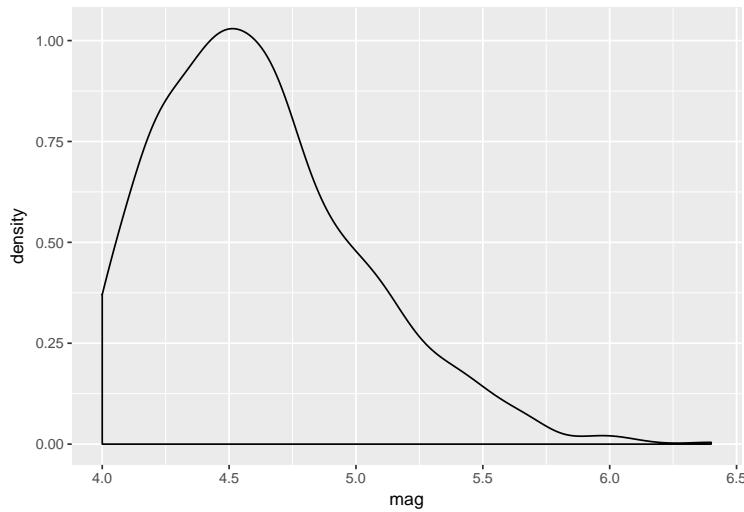
Deux variables - diagrammes juxtaposés

```
ggplot(data = quakes) +
  geom_boxplot(mapping = aes(x = region, y = mag))
```



Courbe d'estimation de densité à noyau - fonction `geom_density`

```
ggplot(data = quakes) +
  geom_density(mapping = aes(x = mag))
```



Deux variables - courbes de densité superposés

```
ggplot(data = quakes) +
  geom_density(mapping = aes(x = mag, fill = region), alpha = 0.5)
```

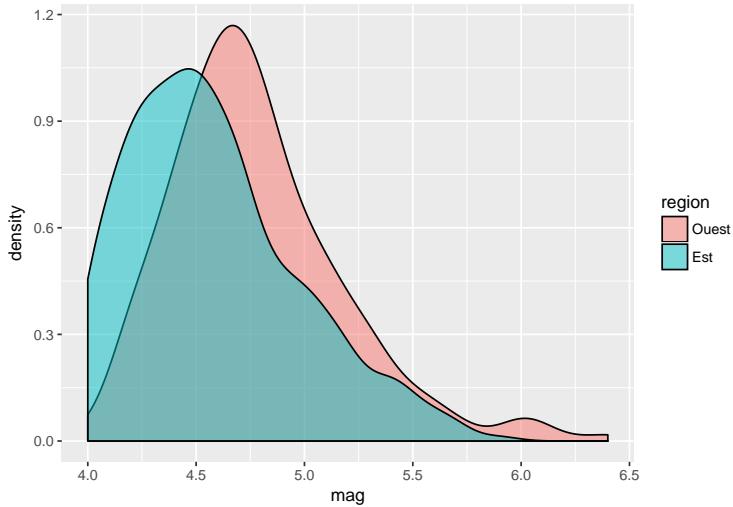
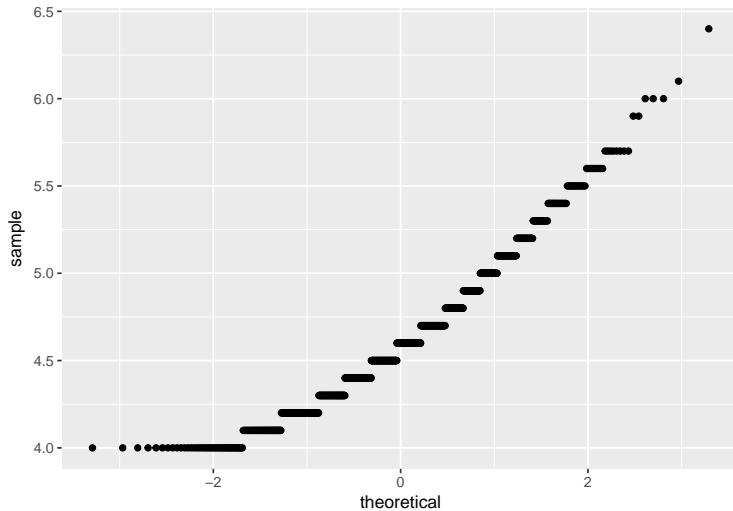


Diagramme quantile-quantile - fonction `geom_qq`

Une variable - diagramme quantile-quantile théorique normal

```
ggplot(data = quakes) +
  geom_qq(mapping = aes(sample = mag))
```



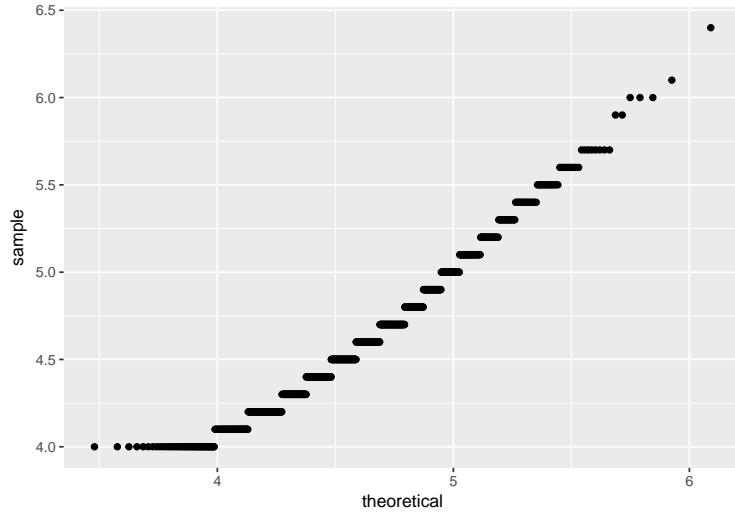
Code équivalent : utiliser l'argument `stat` de `geom_point` plutôt que `geom_qq`.

```
ggplot(data = quakes) +
  geom_point(mapping = aes(sample = mag), stat = "qq")
```

Une variable - diagramme quantile-quantile théorique non normal

Pour une distribution autre que normale

```
ggplot(data = quakes) +
  geom_qq(mapping = aes(sample = mag), distribution = qlnorm,
         dparams = list(meanlog = mean(log(quakes$mag)), sdlog = sd(log(quakes$mag))))
```



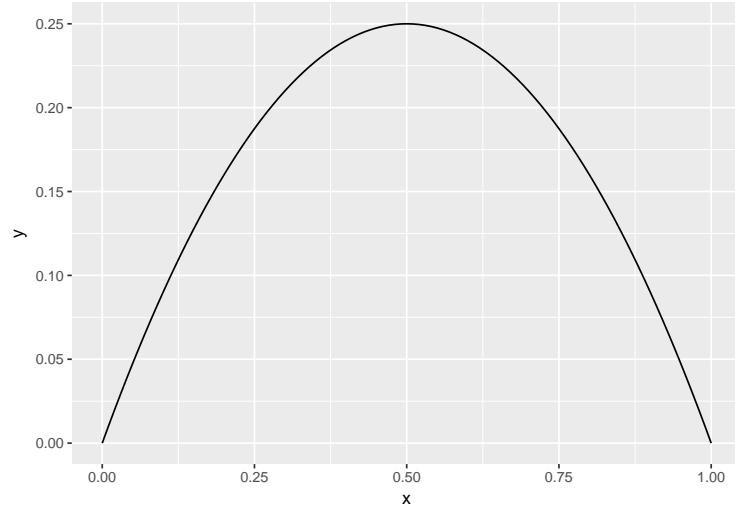
Avec le système graphique R de base c'est aussi faisable, mais c'est plus compliqué.

### Deux variables - diagramme quantile-quantile empirique

Pas de fonction conçue pour produire ce type de graphique (mais on trouve des solutions sur le web).

### Représentation graphique d'une fonction `stat_function`

```
ggplot(data = data.frame(x = c(0, 1)), mapping = aes(x)) +
  stat_function(fun = function(x) { x - x^2 })
```



Ce type de graphique est plus simple à produire avec le système graphique de base, parce que :

- `ggplot2` est conçu pour faire des graphiques à partir de données,
- la fonction `curve` du système de base accepte une expression en entrée.

### Exemples plus poussés

#### Diagramme en secteurs

J'ai l'impression que les auteurs de `ggplot2` n'offrent pas de fonction conviviale pour la création de diagramme en secteurs parce qu'ils ne recommandent pas leur utilisation. Il y a d'ailleurs une mise en garde concernant l'utilisation de ce type de graphique dans la [fiche d'aide de la fonction `pie`](#).

Ce qui est reproché au diagramme en secteurs est qu'il difficile pour l'oeil humain de rapidement comparer les aires des secteurs (qui représentent les fréquences). Il est plus facile de comparer des hauteurs de bâtons. Donc le diagramme en bâtons serait un meilleur outil pour représenter des fréquences.

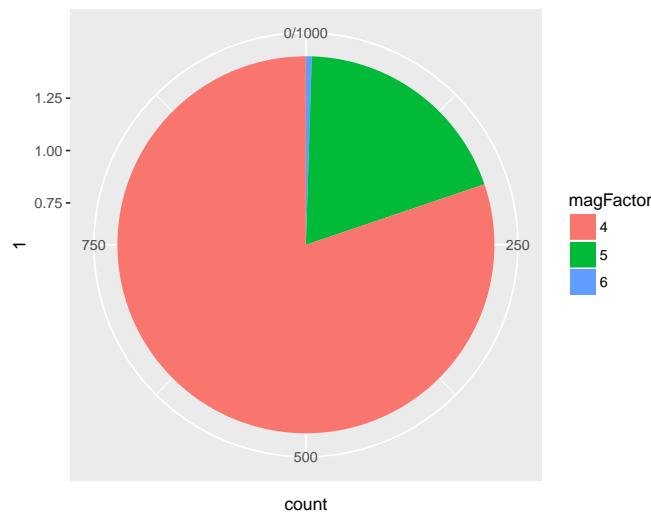
Malgré tout, je crois que le diagramme en secteur est utile pour représenter les fréquences d'une **variable à peu de modalités**, dans le cas où il n'y a pas de fréquences presque égales.

Au-delà de ces considérations, tenter de tracer un diagramme en secteur avec `ggplot2` aide à comprendre davantage les possibilités du package.

### Coordonnées polaires - fonction `coord_polar`

Pour produire un diagramme en secteurs avec `ggplot2`, il faut d'abord produire un diagramme à bâtons empilés, puis demander l'utilisation d'un système de coordonnées polaires par un appel à la fonction `coord_polar`. C'est d'ailleurs pour illustrer l'utilisation de ce type de système de coordonnées que j'ai choisi d'inclure l'exemple suivant.

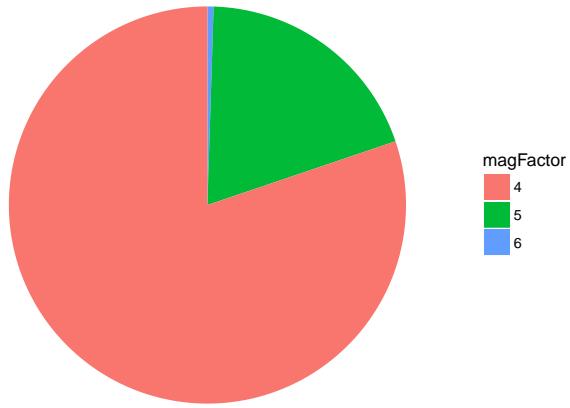
```
ggplot(data = quakes) +
  geom_bar(mapping = aes(x = 1, fill = magFactor)) +
  coord_polar("y")
```



### Configurations graphiques - fonction `theme`

Pour améliorer le résultat, il faut aller modifier des configurations graphiques avec la fonction `theme`.

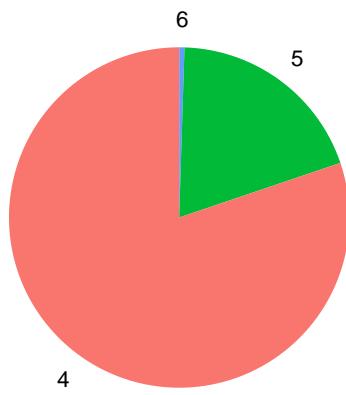
```
ggplot(data = quakes) +
  geom_bar(mapping = aes(x = 1, fill = magFactor)) +
  coord_polar("y") +
  theme(axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.grid = element_blank(),
        panel.background = element_blank())
```



### Annotations textuelles - fonction geom\_text

Nous pourrions même remplacer la légende par des annotations textuelles vis-à-vis les secteurs.

```
ggplot() +
  geom_bar(data = quakes, mapping = aes(x = 1, fill = magFactor)) +
  geom_text(data = as.data.frame(table(magFactor = quakes$magFactor)),
            mapping = aes(x = 1.6,
                           y = Freq/2 + c(rev(cumsum(rev(Freq))))[-1], 0),
            label = magFactor,
            size = 5) +
  coord_polar("y") +
  theme(legend.position="none", # pour retirer la légende
        axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.grid = element_blank(),
        panel.background = element_blank())
```



## Échelle de couleurs - fonctions `scale_colour_xxx` et `scale_fill_xxx`

Le deuxième terme dans le nom de la fonction fait référence aux arguments de `aes` :

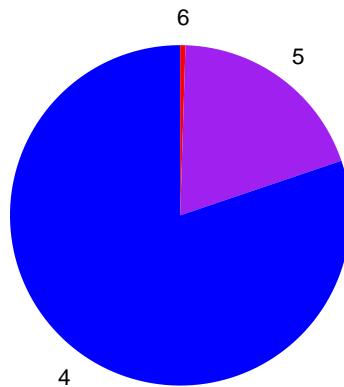
- `fill` = couleur de remplissage,
- `colour` = couleur de points et de lignes.

Le troisième terme, ici désigné par `xxx`, peut être :

- `manual` : couleurs spécifiques individuellement,
- `brewer` : utiliser une palette de couleur offerte dans le package R `RColorBrewer`,
- `grey` : utiliser une palette (dégradé) de couleurs grises,
- etc.

Nous allons changer la palette de couleur utilisée dans le diagramme en secteur.

```
ggplot() +
  geom_bar(data = quakes, mapping = aes(x = 1, fill = magFactor)) +
  geom_text(data = as.data.frame(table(magFactor = quakes$magFactor)),
            mapping = aes(x = 1.6,
                           y = Freq/2 + c(rev(cumsum(rev(Freq))))[-1], 0),
            label = magFactor,
            size = 5) +
  coord_polar("y") +
  theme(legend.position="none", # pour retirer la légende
        axis.title = element_blank(),
        axis.text = element_blank(),
        axis.ticks = element_blank(),
        panel.grid = element_blank(),
        panel.background = element_blank()) +
  scale_fill_manual(values=c("blue", "purple", "red"))
```

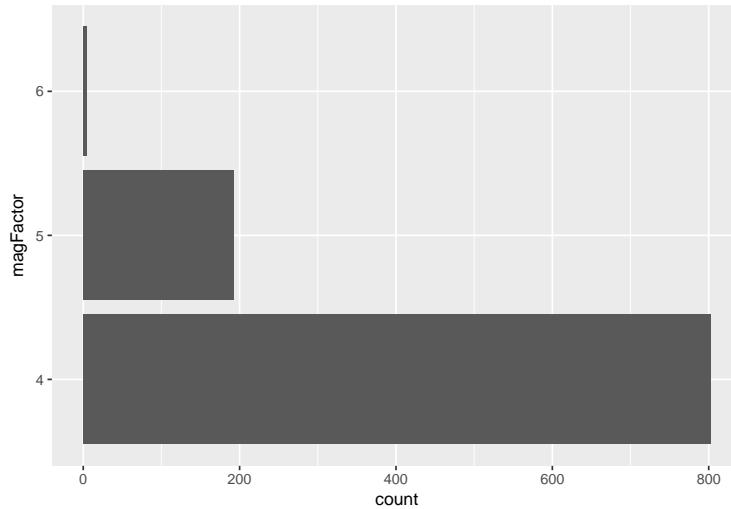


## Diagramme en bâtons horizontaux

### Intervertir des axes - fonction `coord_flip`

Une autre configuration possible en lien avec le système de coordonnées : intervertir les axes.

```
ggplot(data = quakes) +
  geom_bar(mapping = aes(x = magFactor)) +
  coord_flip()
```



### Ajout de courbes à un histogramme

```
moy <- mean(quakes$mag)
et <- sd(quakes$mag)

ggplot(data = quakes, mapping = aes(x = mag)) +

  # Ajout de l'histogramme :
  geom_histogram(mapping = aes(y = ..density..), binwidth = 0.2,
                 colour = "black", fill = "white") +
  xlab("magnitude du séisme") + ylab("densité") +
  ggtitle("Densité empirique des magnitudes dans le jeu de données quakes") +
  theme_minimal()

  # Ajout de la courbe de densité à noyau
  geom_density(aes(colour = "c1")) +
  theme_minimal()

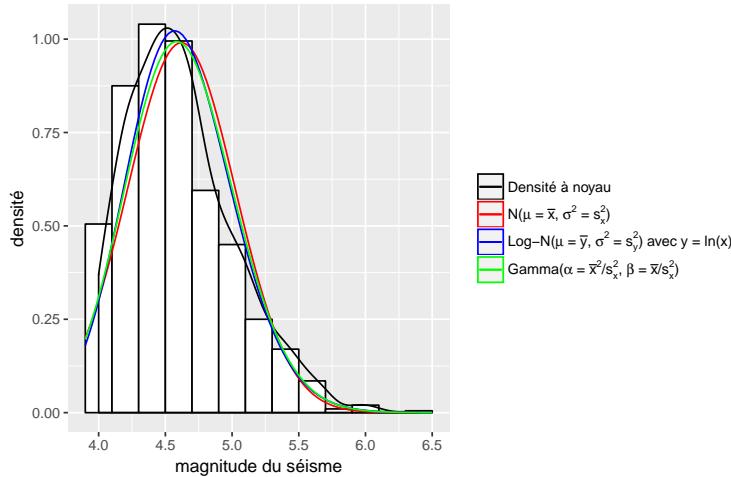
  # Ajout de courbes de densité théoriques
  # avec paramètres estimés à partir des données
  stat_function(aes(colour = "c2"),
                fun = dnorm,
                args = list(mean = moy, sd = et),
                xlim = c(3.9, 6.5)) +
  stat_function(aes(colour = "c3"),
                fun = dlnorm,
                args = list(meanlog = mean(log(quakes$mag)),
                            sdlog = sd(log(quakes$mag))),
                xlim = c(3.9, 6.5)) +
  stat_function(aes(colour = "c4"),
                fun = dgamma,
                args = list(shape = moy^2/et^2, rate = moy/et^2),
                xlim = c(3.9, 6.5)) +
```

```

# Ajout manuel d'une légende
scale_colour_manual(name = "",
  values = c(c1 = "black", c2 = "red", c3 = "blue", c4 = "green"),
  labels = c(c1 = "Densité à noyau",
    c2 = expression(paste("N(", mu, " = ", bar(x), ", ",
      sigma^2, " = ", s[x]^2, ")")),
    c3 = expression(paste("Log-N(", mu, " = ", bar(y), ", ",
      sigma^2, " = ", s[y]^2, ") avec y = ln(x)")),
    c4 = expression(paste("Gamma(", alpha, " = ", bar(x)^2, "/",
      beta, " = ", bar(x), "/", s[x]^2, ")")))) +
  theme(legend.text.align = 0)

```

Densité empirique des magnitudes dans le jeu de données quakes



### Remarque :

- Il est compliqué avec `ggplot2` d'ajouter une légende manuellement comme nous avons dû le faire dans ce graphique.
- Les annotations mathématiques sont toujours possibles avec `ggplot2`, comme dans le système graphique de base.

### Division de la fenêtre en sous-fenêtres

Cette fonctionnalité n'est pas offerte en `ggplot2`, sauf pour des graphiques conditionnels à la valeur d'un facteur.

Par contre, certains packages offrent des fonctions pour le faire, notamment

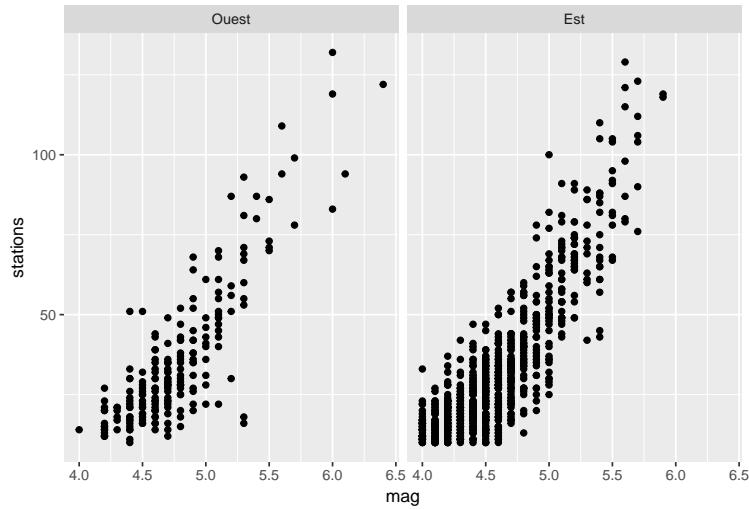
- la fonction `grid.arrange` du package `gridExtra`,
- des opérateurs du package `patchwork`.

Je ne vais pas illustrer ici l'utilisation de cette fonction.

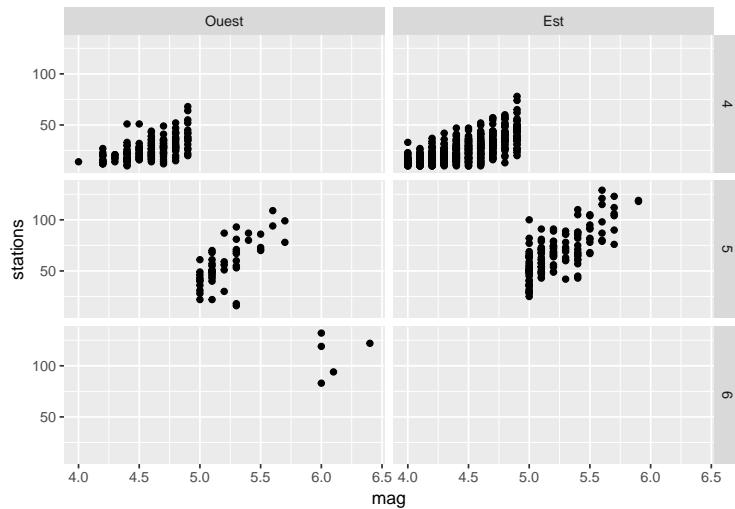
### Représentations multivariables

#### Sous-fenêtres selon une ou des variable(s) catégorique(s) - fonctions `facet_wrap` et `facet_grid`

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

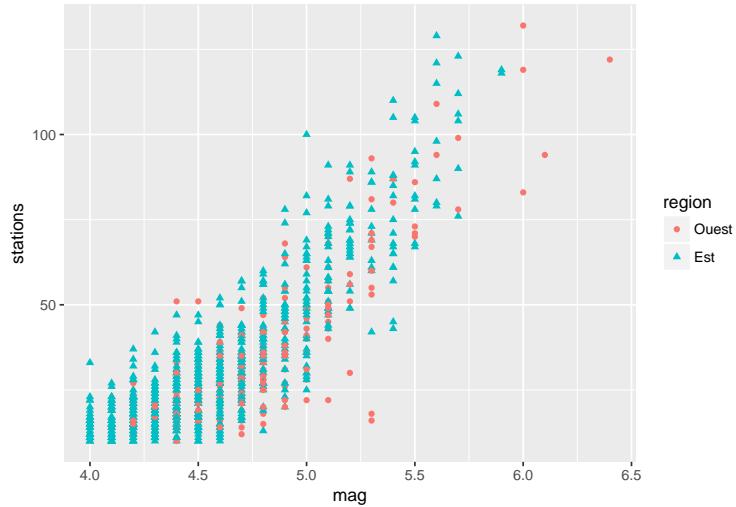


```
ggplot(data = quakes, mapping = aes(x = mag, y = stations)) +
  geom_point() +
  facet_grid(facets = magFactor ~ region)
```



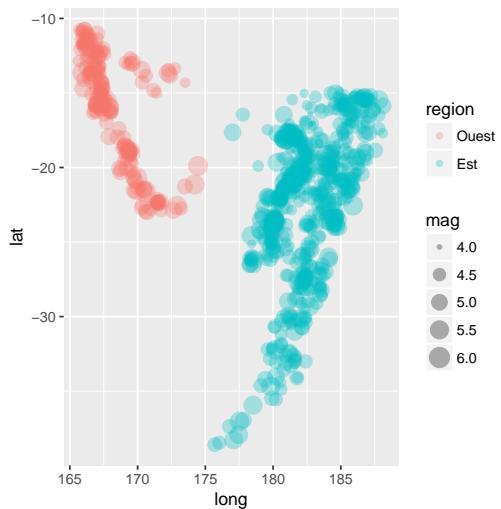
Superposition avec aspect distinctif selon la variable catégorique

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



### Représentation de trois variables numériques et d'une variable catégorique

```
ggplot(data = quakes) +
  geom_point(mapping = aes(x = long, y = lat, colour = region, size = mag),
             alpha = 0.3) +
  coord_quickmap()
# coord_quickmap() permet d'avoir des axes sur la même échelle,
# ce qui est préférable ici, car les variables sont des coordonnées géographiques
```



### Enregistrement d'un graphique - fonction ggsave

La fonction `ggsave` enregistre le dernier graphique `ggplot2` produit, en conservant les dimensions de la fenêtre graphique.

```
ggsave(file = "ExempleGraphique_ggplot2.pdf")
```

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

Aspects plus faciles avec `ggplot2` :

- intégration des légendes (souvent automatisée),
- représentations multivariables,
- ajouts d'éléments qui requièrent un calcul statistique (p. ex. barres erreurs).

Avantages du système graphiques R de base :

- l'utilisateur garde le contrôle sur tout, ce qui peut être essentiel pour produire un graphique pour une publication.

Références :

- <http://seananderson.ca/ggplot2-FISH554/>
  - <http://flowingdata.com/2016/03/22/comparing-ggplot2-and-r-base-graphics/>
- 

## Autres possibilités graphiques en R

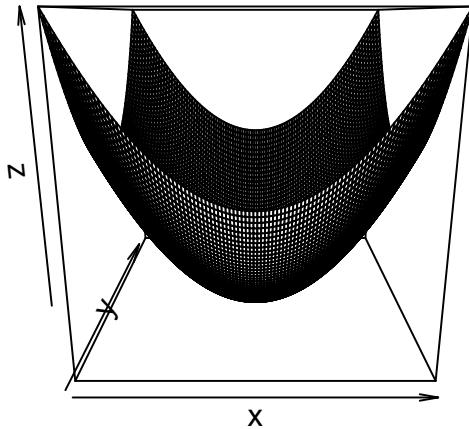
- Graphiques 3D orientables :
  - fonction `persp` du système de base,
  - fonctions `cloud` et `wireframe` du package `lattice`,
  - package `rgl` : fonction `plot3d` et autres.
- Cartes géographiques :
  - packages `maps` et `mapdata`,
  - package `ggmap`.
- Graphiques interactifs :
  - package `plotly` pour créer des graphiques `Plotly`,
  - package `googleVis` pour créer des `Google charts`.
- [Shiny App](#)
- et bien d'autres : <https://cran.r-project.org/web/views/Graphics.html>

## Graphiques 3D

### Fonction `persp` du système de base

Graphique 3D produit avec la fonction `persp` graphique de base :

```
x <- y <- seq(from = -1, to = 1, length = 100)
grille <- expand.grid(x = x, y = y)
z <- grille$x^2 + grille$y^2
persp(x = x, y = y, z = matrix(z, ncol = length(x)), zlab = "z")
```

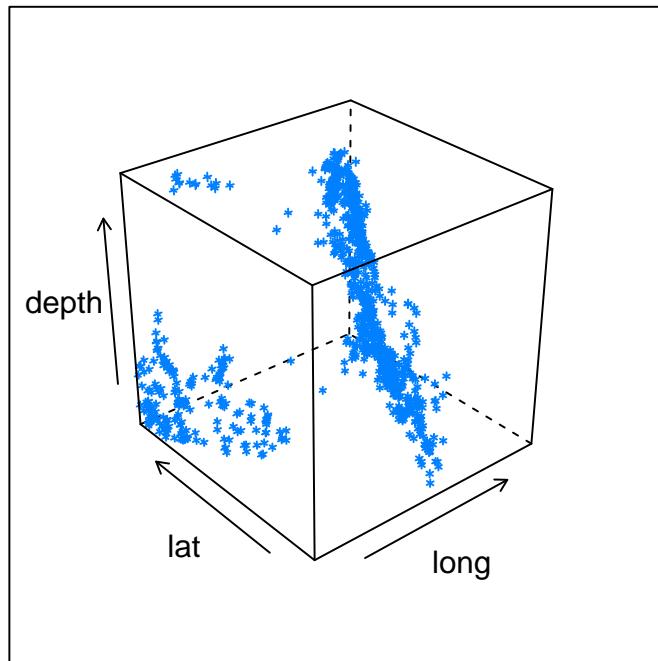


Cette fonction ne permet pas de représenter un nuage de points en 3 dimensions. Elle permet plutôt de tracer une fonction mathématique de 2 variables. Il faut fournir à l'argument `z` la valeur de la fonction mathématique pour toutes les combinaisons des valeurs en `x` et `y`.

### Fonction `cloud` du package `lattice`

Graphique 3D produit avec la fonction `cloud` du package `lattice` :

```
cloud(depth ~ long + lat, data = quakes)
```



### Graphiques 3D orientables - package rgl

Le package `rgl` permet de produire des graphiques 3D. Ces graphiques sont affichés dans une fenêtre interactive dans laquelle il est possible de tourner le graphique. Il n'est cependant pas possible de visualiser le résultat dans un document HTML ou PDF. Voici un exemple de commande pour créer un graphique 3D avec la fonction `plot3d` du package `rgl`.

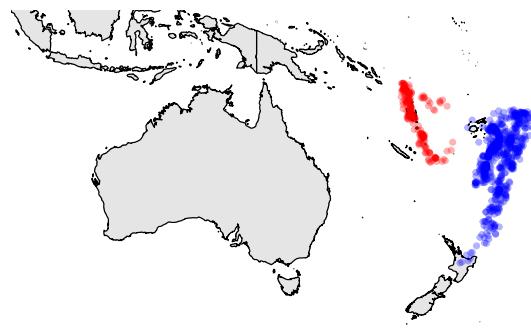
```
library(rgl)
plot3d(x = quakes$long, y = quakes$lat, z = quakes$depth,
       col = ifelse(quakes$region == "Ouest", rgb(1, 0, 0, 0.3), rgb(0, 0, 1, 0.3)),
       aspect = TRUE, type = "s", size = 1)
# Le graphique produit n'apparaît pas dans le fichier HTML, ni dans le fichier PDF,
# car il est affiché dans une fenêtre graphique interactive. Soumettez ces instructions
# en R pour voir le résultat (après avoir ajouté la variable region à quakes).
```

## Cartes géographiques

### Packages maps et mapdata

Carte tirée de banques de données accessibles par l'intermédiaire des packages `maps` et `mapdata`.

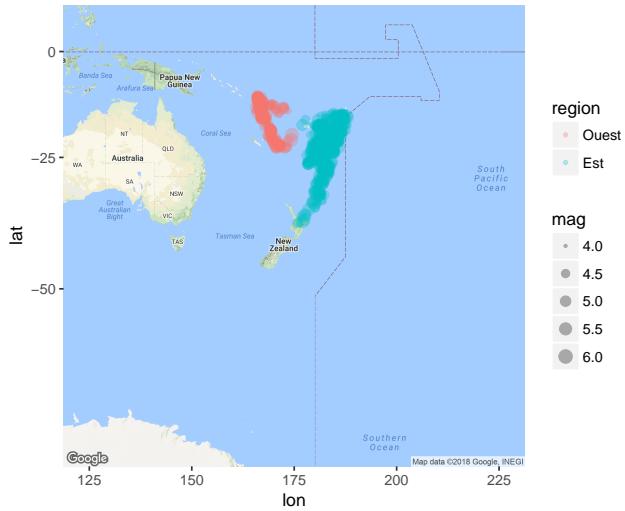
```
library(maps)
library(mapdata)
map(database = "worldHires", xlim = c(100, 190), ylim = c(-50, 0),
     col = "gray90", fill = TRUE)
points(x = quakes$long, y = quakes$lat, pch = 20,
       col = ifelse(quakes$region == "Ouest", rgb(1, 0, 0, 0.3), rgb(0, 0, 1, 0.3)))
```



## Packages ggmap

Carte tirée de Google Maps.

```
library(ggmap)
carte <- get_map(location = 'new zealand', zoom = 3)
ggmap(carte) +
  geom_point(data = quakes,
             aes(x = long, y = lat, colour = region, size = mag),
             shape = 20, alpha = .3)
```



Il est possible d'ajouter des éléments à ces cartes.

Il existe plusieurs autres packages pour la création de cartes en R.

## Graphiques interactifs

Un graphique interactif est un graphique qui n'est pas statique (une image fixe), mais qui est plutôt capable d'interagir avec la personne qui l'observe. La forme la plus simple d'interaction est de fournir de l'information

supplémentaire lorsque la souris est placée sur certains éléments du graphique. Une forme plus avancée d'interaction est de modifier certains éléments du graphique en fonction de choix fait par l'observateur, par exemple via des boutons ou un menu cliquable.

Ce genre de graphique se visualise typiquement dans un navigateur internet. Il s'intègre facilement à une page web, mais pas à un document PDF.

Je me limite ici à la présentation rapide de 2 outils pour produire des graphiques interactifs en R. La page web suivante en présente d'autres : <http://ouzor.github.io/blog/2014/11/21/interactive-visualizations.html>

### Graphiques interactifs Plotly - package plotly

L'application web Plotly (<https://plot.ly/>) est devenue très populaire pour produire des graphiques interactifs. Le package R `plotly` permet de créer facilement des graphiques Plotly en R. Voici un exemple affiché seulement dans la version HTML des notes, pas dans le PDF.

```
library(plotly)

plot_ly(quakes, x = ~mag, y = ~stations, color = ~depth,
        size = ~depth, text = ~paste("Region: ", region))
```

### Google Charts - package googleVis

Google Charts est un outil de création de graphiques développé par Google (<https://developers.google.com/chart/>). Il permet d'intégrer des graphiques à des pages web. Il propose plusieurs types de graphiques.

Exemple : (affiché seulement dans la version HTML des notes, pas dans le PDF)

```
library(googleVis)

# Il faut travailler un peu pour mettre les données sous le bon format
freq <- table(quakes$magFactor, quakes$region)
df <- data.frame(rownames(freq), as.matrix.data.frame(freq))
colnames(df) <- c("mag", colnames(freq))

# Production du graphique
Bar <- gvisBarChart(df)
plot(Bar)

## starting httpd help server ... done
```

Autres Exemple : *Geo Chart*

Les *Geo Chart* sont des cartes géographiques intégrant des couleurs dans des régions selon le niveau d'une variable. Ce type de carte est parfois nommé carte choroplète. En voici un exemple :

<https://developers.google.com/chart/interactive/docs/gallery/geochart>

## Conclusion

### Quelques conseils :

- Trouver de l'information :
  - Système graphique de base :
    - lire les fiches d'aide, nous y apprenons toutes les possibilités des fonctions ;

- Autres systèmes (en particulier `ggplot2`) et packages graphiques : faire des recherches sur internet.
- Stratégie 1 - Procéder par essai/erreur jusqu'à obtenir le graphique souhaité.
- Stratégie 2 - Partir d'un modèle :
  - un graphique que nous avons déjà fait ou
  - un graphique trouvé dans un livre ou sur internet (voir les références).

## R = un excellent outil pour les graphiques

- R produit des graphiques d'assez bonne qualité pour des publications et facilement reproductibles à partir de leur code source.
- La variété des graphiques possibles de créer en R est impressionnante.
- De nouvelles fonctions graphiques sont fréquemment ajoutées.

## Références

Livres (les trois derniers traitent surtout ou exclusivement de `ggplot2`) :

- Murrell, P. (2011). R Graphics, Second Edition. CRC Press. URL pour le code R : <https://www.stats.auckland.ac.nz/~paul/RG2e/>
- 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/>
- Wickham, H. (2016). ggplot2 : Elegant Graphics for Data Analysis, Second Edition. Springer. URL pour le code R : <http://ggplot2.org/book>
- Wickham, H. et Grolemund, G. (2016). R for Data Science. O'Reilly Media, Inc. Chapitre 3. URL <http://r4ds.had.co.nz/>

Quelques sites web (il y en a bien plus !) :

**Système de base :**

- <http://www.statmethods.net/graphs/index.html>
- <http://www.statmethods.net/advgraphs/index.html>

**Package `lattice` :**

- <https://CRAN.R-project.org/package=lattice>
- <http://lattice.r-forge.r-project.org/>

**Package `ggplot2` :**

- <https://CRAN.R-project.org/package=ggplot2>
- <http://ggplot2.org/> and <http://ggplot2.tidyverse.org/>
- Tutoriels, rédigés par d'anciens étudiants du cours :
  - Boxplot : [https://stt4230.rbind.io/tutoriels\\_etudiants/hiver\\_2015/boxplot\\_ggplot2/](https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/boxplot_ggplot2/)
  - Graphique de série temporelle : [https://stt4230.rbind.io/tutoriels\\_etudiants/hiver\\_2015/graphique\\_temporel\\_ggplot2/](https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/graphique_temporel_ggplot2/)
  - Histogramme et courbe de densité de Kernel : [https://stt4230.rbind.io/tutoriels\\_etudiants/hiver\\_2015/histogramme\\_ggplot2/](https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/histogramme_ggplot2/)
  - Diagramme en bâtons : [https://stt4230.rbind.io/tutoriels\\_etudiants/hiver\\_2015/diagramme\\_batons\\_ggplot2/](https://stt4230.rbind.io/tutoriels_etudiants/hiver_2015/diagramme_batons_ggplot2/)

## Exemples de graphiques

- <http://www.r-graph-gallery.com/> : exemples de graphiques avec leur code source permettant de les reproduire (tous systèmes, mais beaucoup de `ggplot2`)

## Autres

- Référence concernant les couleurs en R :
  - <http://research.stowers.org/mcm/efg/R/Color/Chart/>
- Feuilles de triche (cheat sheets) :
  - Système de base
    - \* <http://www.joyce-robbins.com/wp-content/uploads/2016/04/BaseGraphicsCheatsheet.pdf>
    - \* Paramètres graphiques : <http://gastonsanchez.com/visually-enforced/resources/2015/09/22/R-cheat-sheet-graphical-parameters/>
  - `ggplot2` :
    - \* <https://www.rstudio.com/wp-content/uploads/2016/11/ggplot2-cheatsheet-2.1.pdf>
    - \* traduction française de ce document :  
<http://thinkr.fr/pdf/ggplot2-french-cheatsheet.pdf>
- Vidéos :
  - vidéos de Roger Peng concernant `ggplot2` :
    - \* <https://www.youtube.com/watch?v=HeqHMM4ziXA>
    - \* <https://www.youtube.com/watch?v=n8kYa9vu1l8>
- Revue des packages R ajoutant de nouvelles possibilités graphiques à R :
  - <https://cran.r-project.org/web/views/Graphics.html>