Graphiques

Ewen Gallic
Université de Rennes 1, 2014 - 2015

Structure

Plusieurs packages

- Créer des graphiques en R peut se faire à l'aide de plusieurs packages :
  - `graphics`,
  - `lattice`,
  - `rgl`,
  - `ggplot2` ;
- Nous allons aborder les graphiques réalisés avec `ggplot2`.

Superposition de couches

- Les graphiques avec `ggplot2` sont créés par **couches** ("layers") ;
- La première contient les données brutes ;
- Les suivantes servent à mettre en forme ces données ;
- Le tout s'appuie sur une **grammaire graphique**.

Grammaire

- La grammaire créé une "carte" qui fait un **plan** ("mapping") pour passer :
  - des données,
  - aux **attributs esthétiques** (couleur, forme, taille, etc.) des **objets géométriques** (points, lignes, polygones, etc.) ;
- Elle permet également d'appliquer des **transformations** aux données ;
- Ou encore de faire du **facettage** ("faceting").

Grammaire

- Il est nécessaire de charger le package `ggplot2` (et de l'installer lors de la première utilisation) :

```
install.packages("ggplot2")
library(ggplot2)
```

- L'aide en ligne est précieuse : http://docs.ggplot2.org/current/

Éléments de structure

- Les éléments de la grammaire graphique de `ggplot2` sont :
  - des données brutes (`data`),
  - une projection graphique (`mapping`),
  - des objets géométriques (`geom`),
  - des transformations statistiques (`stats`),
  - des échelles (`scale`),
  - un système de coordonnées (`coord`),
  - une indication de regroupement éventuels (`facet`).

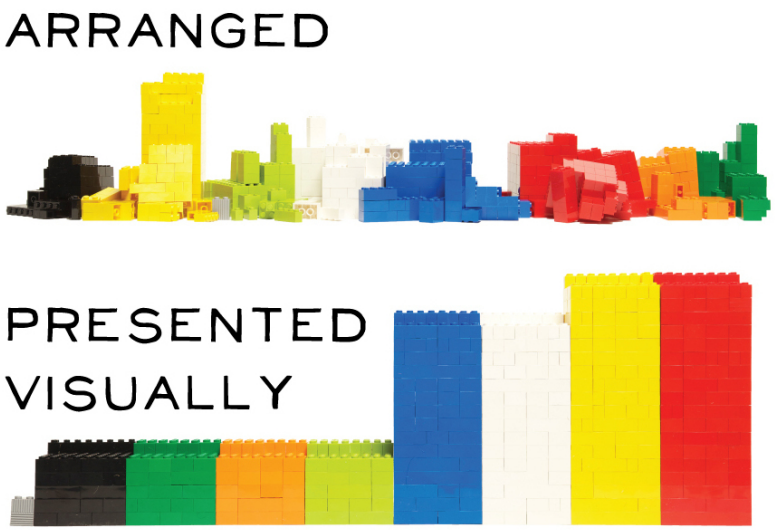
Syntaxe

- La création d'un graphique avec `ggplot2` commence par l'appel de la fonction `ggplot()` ;
- On peut lui fournir les données et la projection graphique ;
- Mais il faut lui ajouter (à l'aide de `+`) une couche indiquant la géométrie.

```
ggplot(data, aes(x, y)) + layers
```

- Les données doivent être dans un `data.frame`.

Des graphiques élaborés avec `ggplot()`



Source : http://www.hotbutterstudio.com/#/alps/

Données pour les exemples

- Échantillon de 135 films (Source : freebase) ;
- Informations sur :
    - nom,
    - date de sortie,
    - durée,
    - année,
    - budget estimé,
    - revenu brut estimé,
    - principal lieu de tournage,
    - abréviation du principal lieu de tournage.

```
load(url("http://editerna.free.fr/films.rda")
```

Données pour les exemples

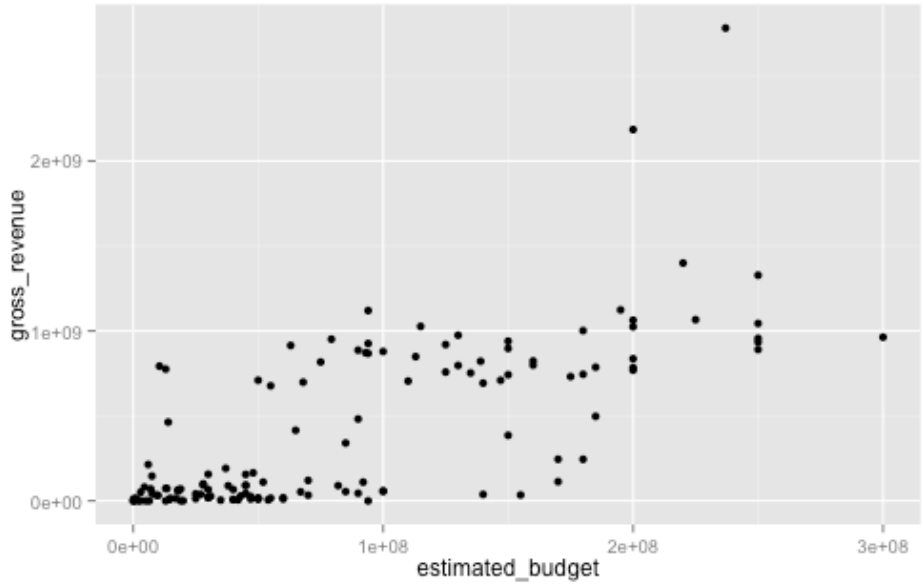
- Créons une sous-base, qui ne concerne que quelques pays :

```
pays_liste <- c("United States of America", "New Zealand",
                "United Kingdom", "Spain")
films_reduit <- films[which(films$country %in% pays_liste),]
```

Un premier graphique

- Voici un premier exemple de graphique : un nuage de points :

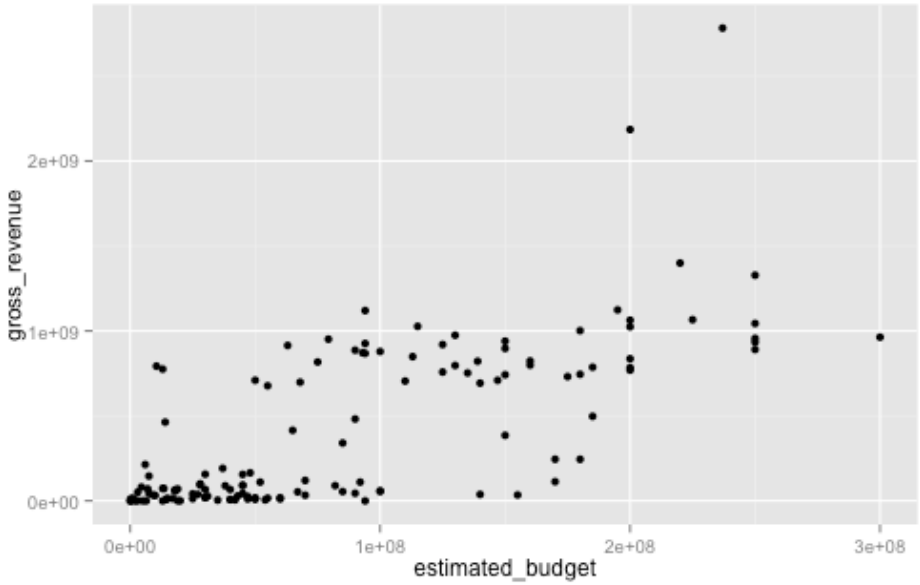
```
ggplot(data = films, aes(x = estimated_budget, y = gross_revenue)) + geom_point()
```



Un premier graphique

- Il est possible de stocker un graphique `ggplot2` dans une variable :

```
p <- ggplot(data = films, aes(x = estimated_budget, y = gross_revenue)) + geom_point()
print(p)
```

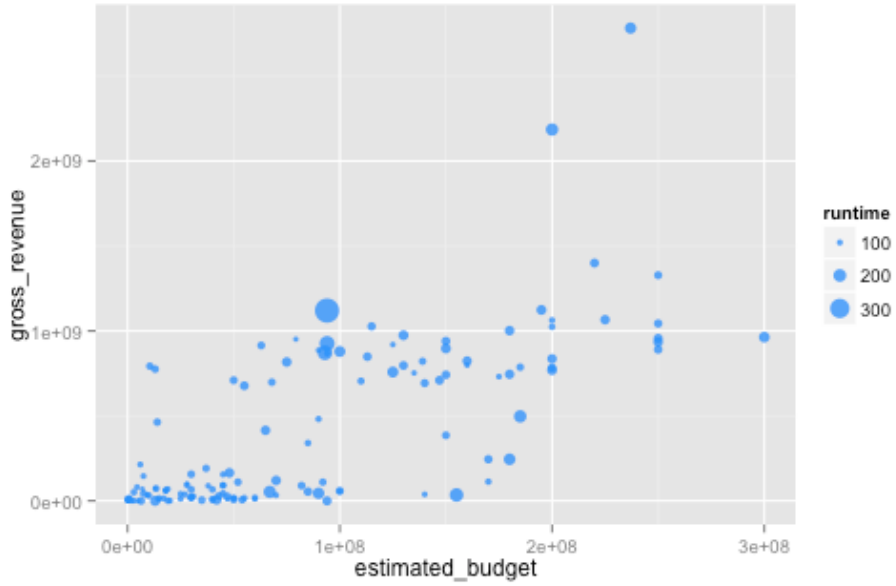


Paramètres esthétiques

- Parmi les paramètres esthétiques on retrouve :
  - `colour` : la couleur,
  - `shape` : la forme,
  - `size` : la taille,
  - `alpha` : la transparence,
  - `fill` : le remplissage ;
- La valeur prise par ces paramètres peut :
  - être identique pour toutes les observations,
  - dépendre d'une variable facteur.

Paramètres esthétiques

```
ggplot(data = films, aes(x = estimated_budget, y = gross_revenue)) +
  geom_point(colour = "dodger blue", alpha = .8, aes(size = runtime))
```



Exercice

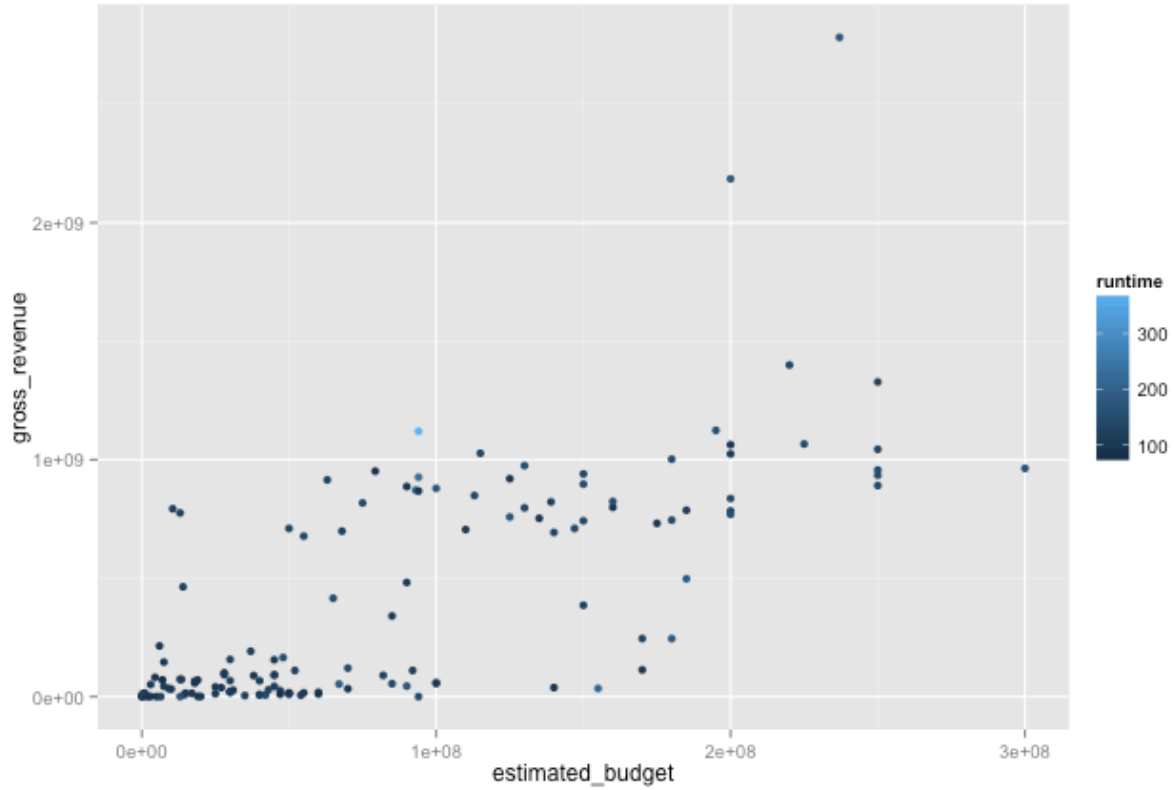
1. Créer un nuage de points représentant la durée des films en fonction de leur budget :
    - la couleur devra correspondre au budget estimé,
    - tester plusieurs valeurs de taille pour les points,
    - tester plusieurs valeurs de formes pour les points,
    - tester différentes valeurs de transparence.

Paramètres esthétiques

- Lorsqu'un paramètre esthétique dépend d'une variable du `data.frame`, une distinction est faite en fonction du mode de cette variable ;
- Par exemple, pour une variable **numérique** fournie à l'esthétique de couleurs, une **échelle de couleurs** sera utilisée ;
- Si en revanche la variable est un **facteur**, une **palette de couleurs** sera utilisée.

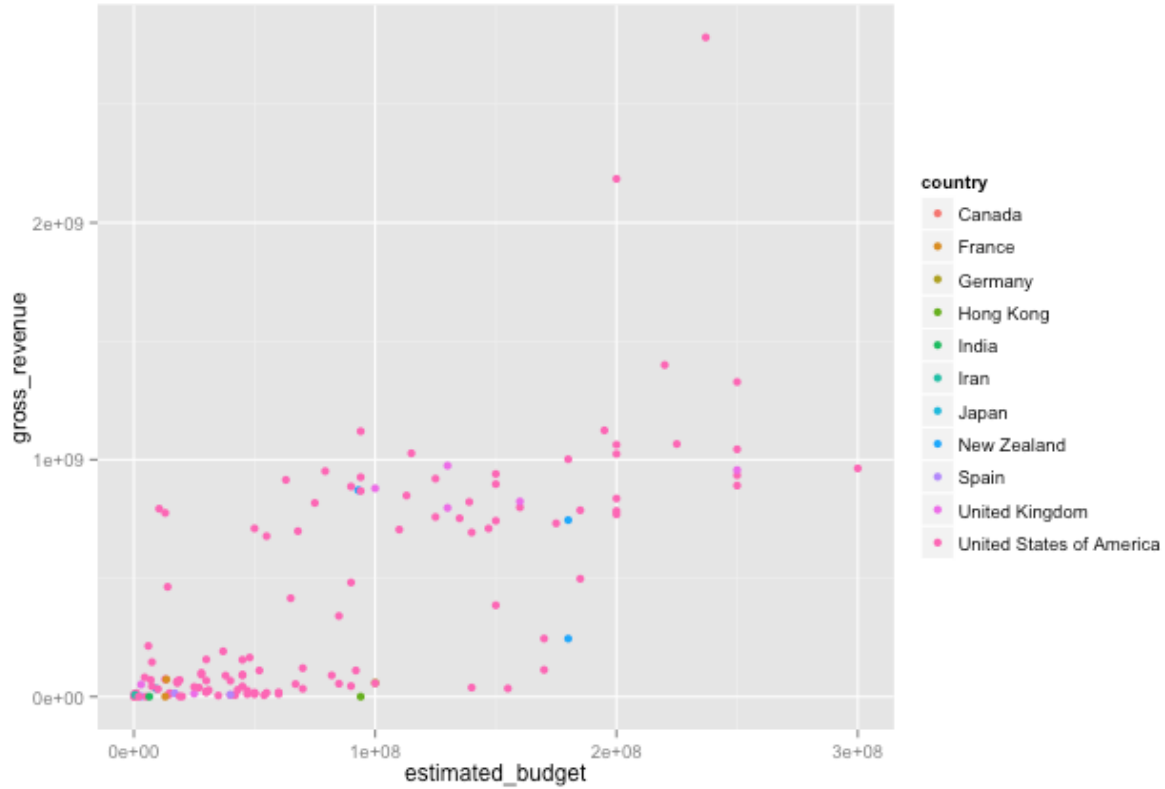
Paramètres esthétiques

```
ggplot() + geom_point(data = films,
                      aes(x = estimated_budget, y = gross_revenue, col = runtime))
```



Paramètres esthétiques

```
ggplot() + geom_point(data = films,
                      aes(x = estimated_budget, y = gross_revenue, col = country))
```



Paramètres géométriques

- Les paramètres géométriques indiquent quel type de représentation on souhaite afficher ;
- On le précise en ajoutant une couche au graphique :
  - `geom_point()` : tracer des points ;
  - `geom_line()` : tracer des lignes ;
  - `geom_polygon()` : tracer des lignes ;
  - `geom_path()` : tracer des points dans l'ordre du data frame ;
  - `geom_step()` : faire un graphique en escalier ;
  - `geom_boxplot()` : tracer une boîte à moustache ;
  - `geom_jitter()` : mettre des points côte à côte pour une variable catégorielle ;
  - `geom_smooth()` : ajouter une courbe de tendance ;
  - `geom_histogram()` : tracer un histogramme ;
  - `geom_bar()` : tracer un diagramme en bâton ;
  - `geom_density()` : tracer une estimation de densité.

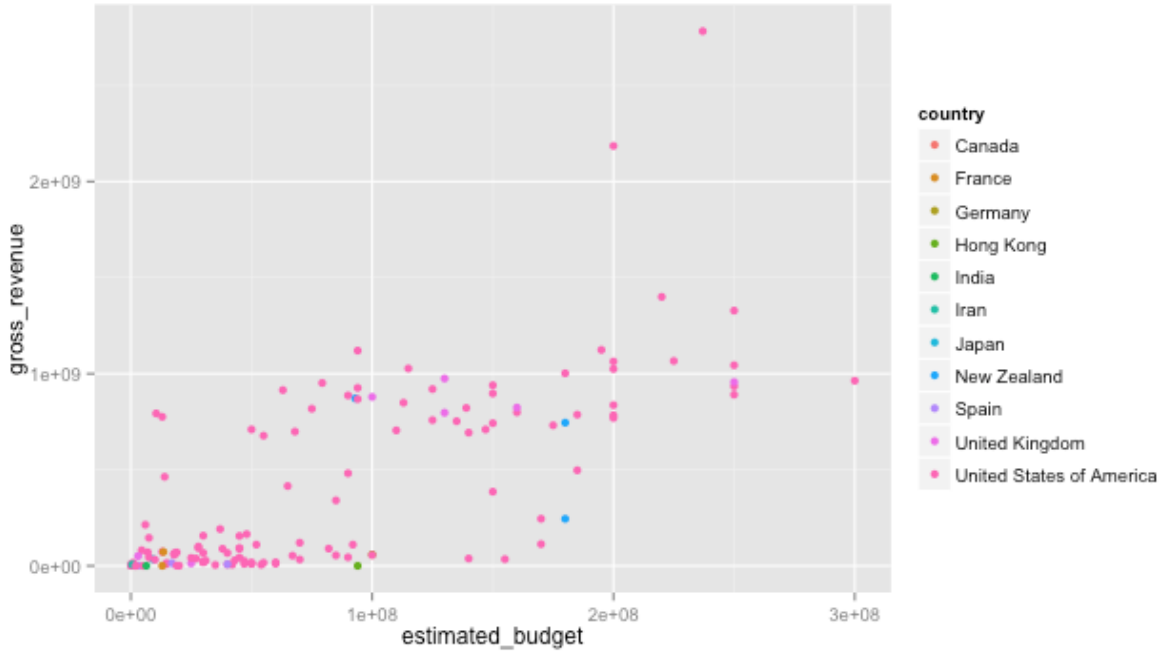
Paramètres géométriques

- Les fonctions `geom_*` possèdent des paramètres optionnels (voici les principaux) :
  - `data` : données brutes à représenter,
  - `mapping` : projection graphique à employer,
  - `stat` : transformation statistique,
  - `position` : des positions pour éviter le chevauchement ;
- Lorsque ces paramètres sont omis, les valeurs automatiquement attribuées sont héritées de celles de `ggplot()`.

Points

- Nous l'avons déjà employée, la `geom_point()` permet de faire des nuages de points.

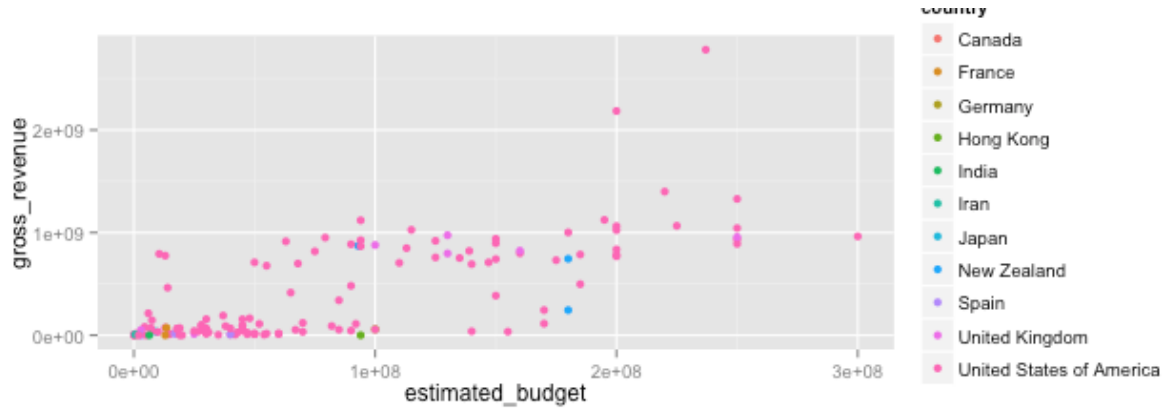
```
ggplot(data = films,
       aes(x = estimated_budget, y = gross_revenue, col = country)) +
  geom_point()
```



Points

- Pour tracer les points, il est nécessaire de préciser les données brutes et le mapping ;
- On l'a fait lors de l'appel de `ggplot()` ;
- Il est également possible de le faire lors de l'appel de `geom_point()` ;
- Attention à bien mettre `data =` !

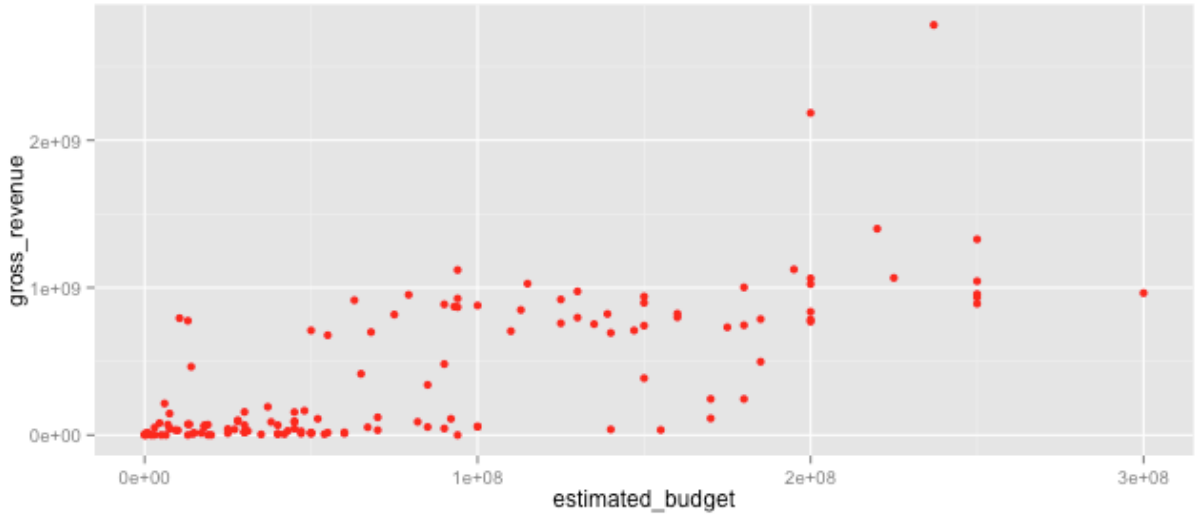
```
ggplot() +
  geom_point(data = films,
       aes(x = estimated_budget, y = gross_revenue, col = country))
```



Points

- Attention, pour que tous les points aient la même couleur, il faut que le paramètres esthétique `colour` soit fourni à `geom_point()` et non pas à `ggplot()` !

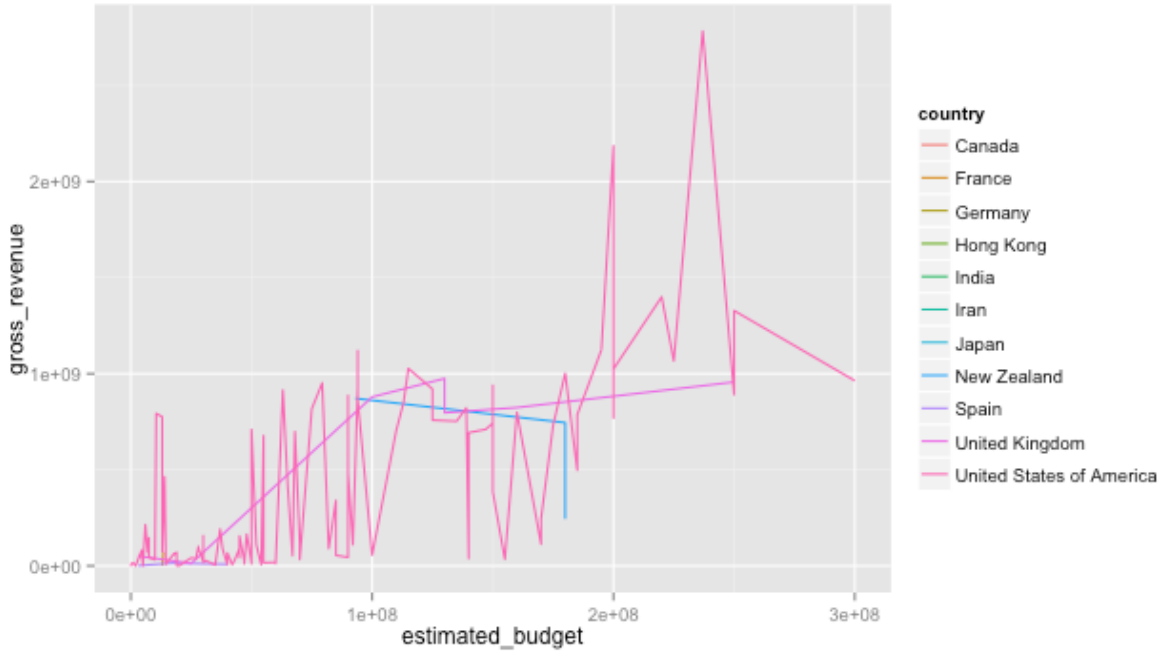
```
ggplot(data = films,
       aes(x = estimated_budget, y = gross_revenue)) +
  geom_point(colour = "red")
```



Lignes

- Si on souhaite joindre les points, on peut utiliser `geom_line()` ;

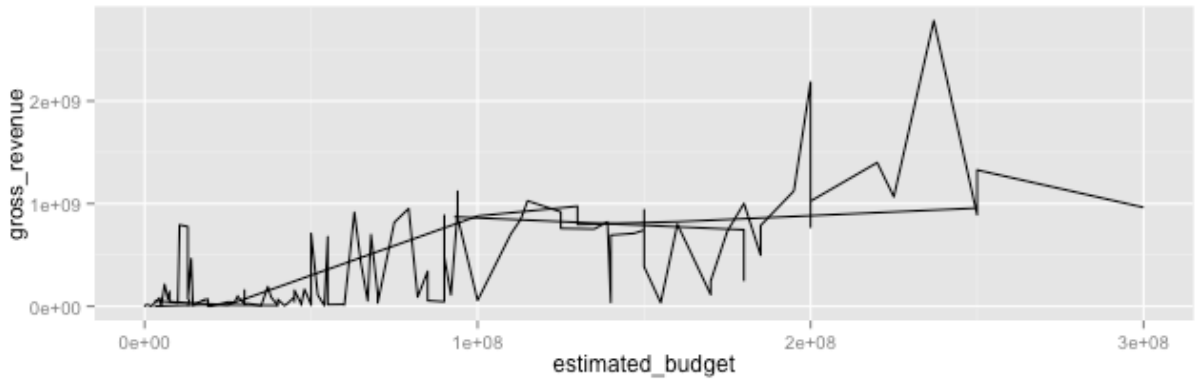
```
ggplot(data = films,
       aes(x = estimated_budget, y = gross_revenue, col = country)) +
  geom_line()
```



Lignes

- Le fait d'ajouter l'esthétique `colour` qui dépend d'une variable facteur du `data.frame` créé automatiquement des groupes ;
- Si on souhaite conserver les groupes, mais ne pas proposer différentes couleurs, on peut utiliser le paramètre `group` :

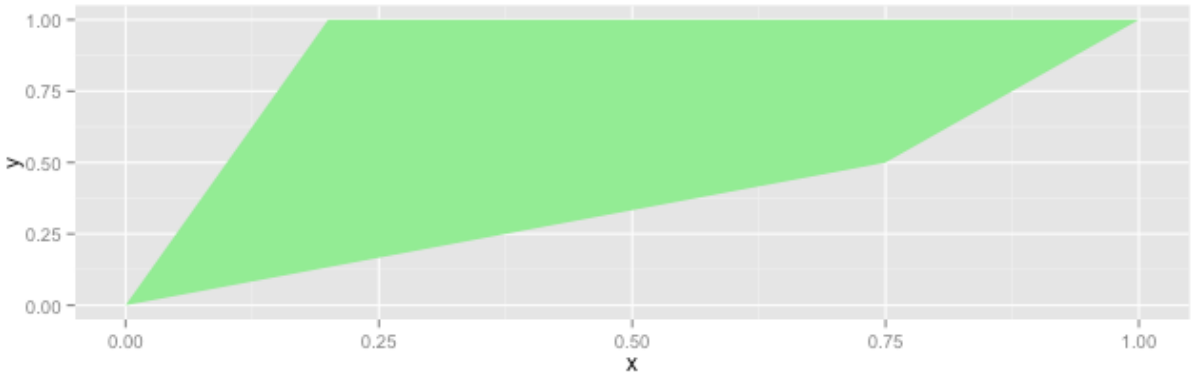
```
ggplot(data = films,
       aes(x = estimated_budget, y = gross_revenue, group = country)) +
  geom_line()
```



Polygones

- Les polygones se tracent à l'aide de `geom_polygon()` ;
- Les coordonnées doivent être rangées dans le sens indirect ;
- Dans la plupart des fonctions utilisant des polygones, la première et la dernières observations doivent être identiques, pour permettre de fermer le polygon ;
- Ce n'est pas obligatoire avec `ggplot2`.

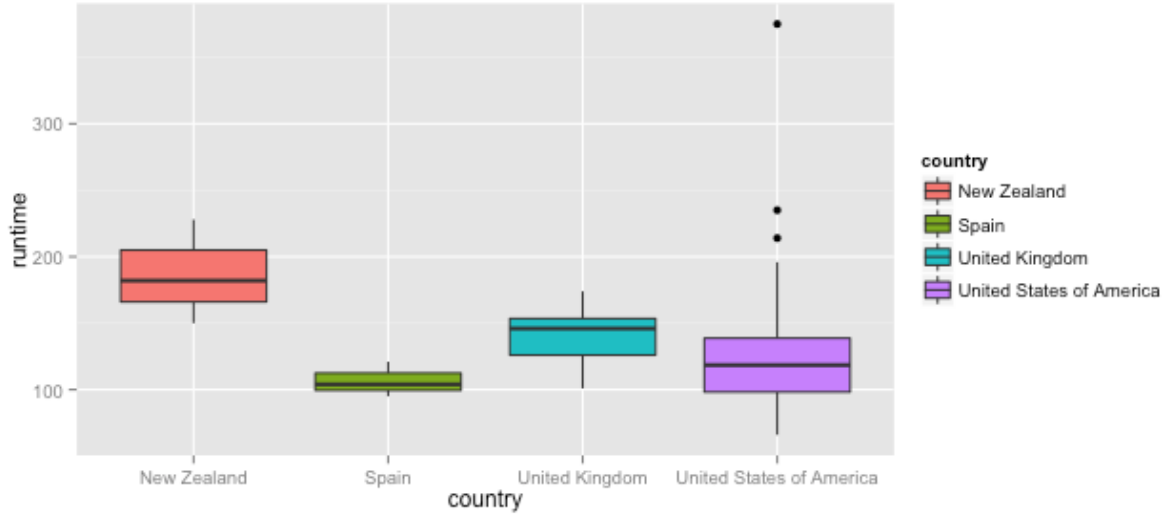
```
df <- data.frame(x = c(0, 0.2, 1, 0.75), y = c(0, 1, 1, 0.5))
ggplot() + geom_polygon(data = df, aes(x = x, y = y), fill = "light green")
```



Boxplot

- Les boîtes à moustaches sont réalisées avec `geom_boxplot()`

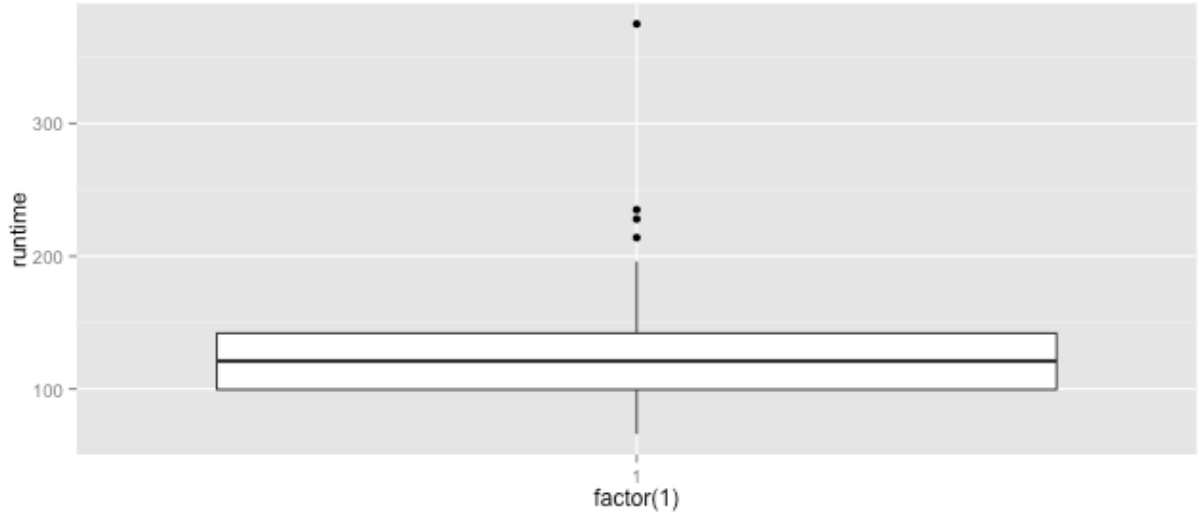
```
ggplot(data = films_reduit, aes(x = country, y = runtime, fill = country)) +
  geom_boxplot()
```



Boxplot

- Pour afficher le boxplot d'une seule variable, il faut "tricher" ;
- On fournit une variable facteur à une modalité à l'esthétique `x` :

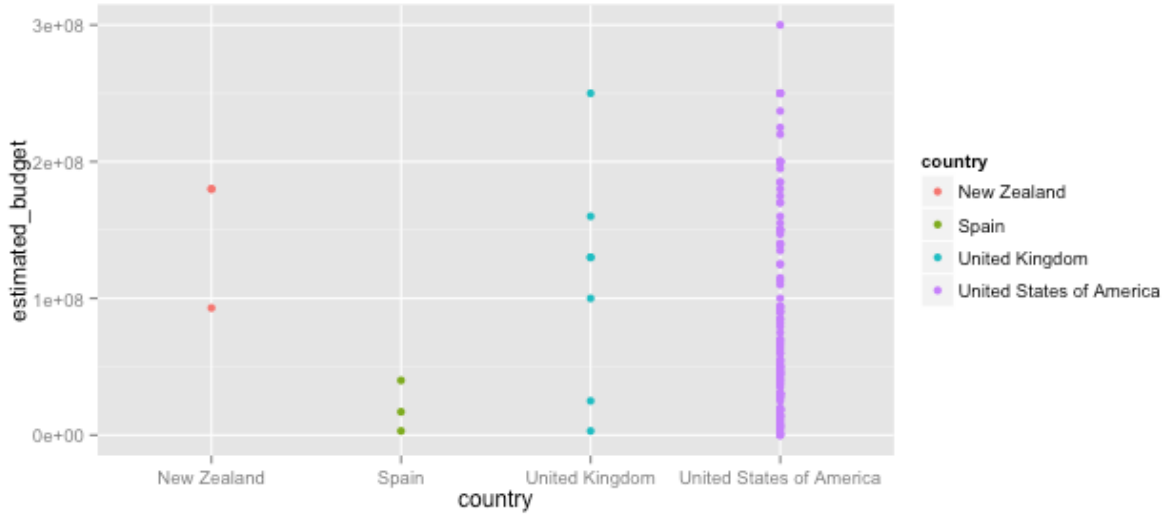
```
ggplot(data = films_reduit, aes(x = factor(1), y = runtime)) + geom_boxplot()
```



Gigue

- Quand on a des points entassés, il est possible de les placer côte-à-côte pour faciliter la visualisation ;

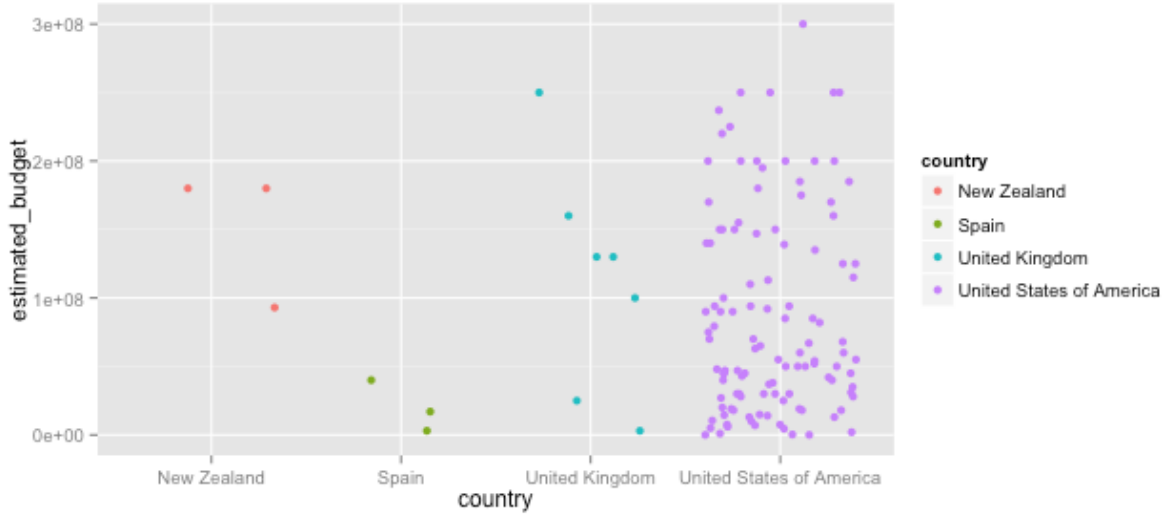
```
ggplot(data = films_reduit, aes(x = country, y = estimated_budget, col = country)) +
  geom_point()
```



Gigue

- Pour ce faire, on utilise `geom_jitter()`

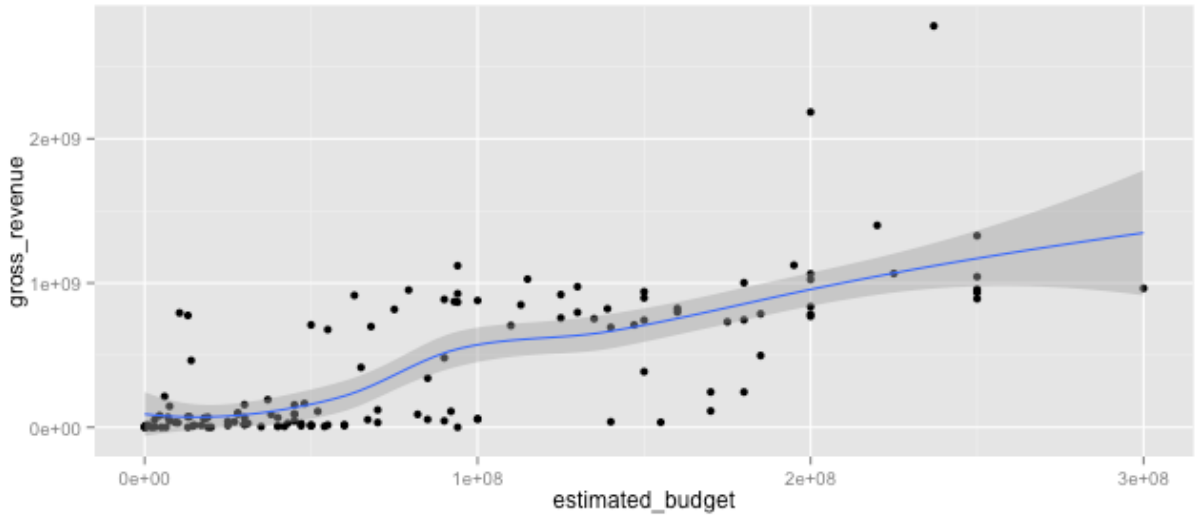
```
ggplot(data = films_reduit, aes(x = country, y = estimated_budget, col = country)) +
  geom_jitter()
```



Courbe de tendance

- L'ajout d'une courbe de tendance peut se faire de plusieurs façons ;
- Par exemple, on peut utiliser `geom_smooth()` :

```
ggplot(data = films, aes(x = estimated_budget, y = gross_revenue)) +
  geom_point() + geom_smooth()
```



Courbe de tendance

- On vient de voir qu'il est possible d'ajouter facilement une autre couche ;
- Il suffit d'appeler une autre fonction `geom_*()` !

```
ggplot() + layer_1 + layer_2 + ...
```

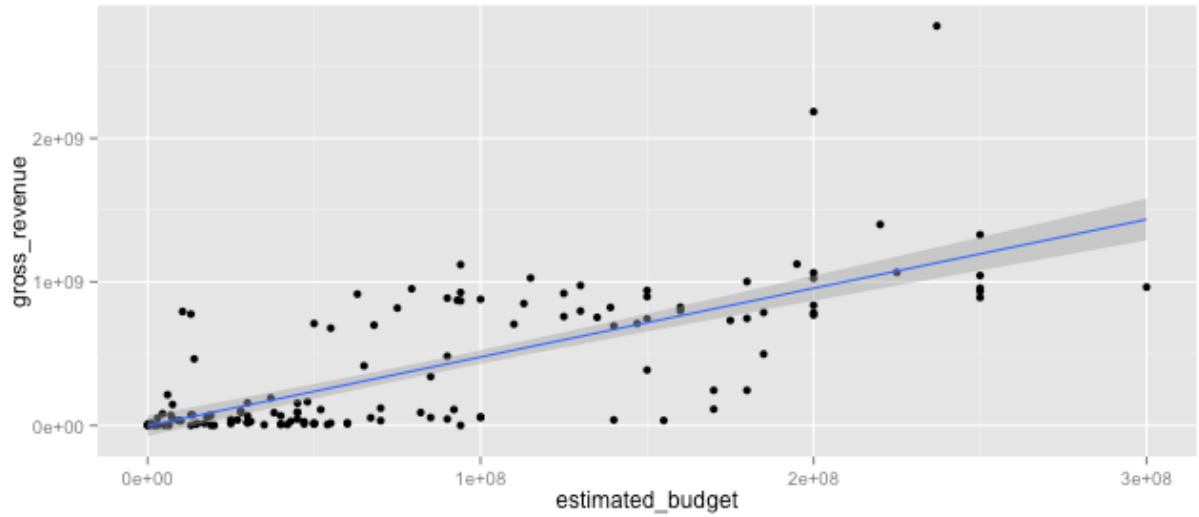
Courbe de tendance

- Le paramètre le plus important de `geom_smooth()` est `method` ;
- Il gère le type de lissage à utiliser (`lm`, `glm`, `gam`, `loess`, `rlm`) ;
- Par défaut, si les données sont moins de 1000, `loess` est utilisé, sinon, `gam` est employé ;
- Au besoin, la formule à utiliser pour le lissage se paramètre avec `formula` ;
- Il est possible de ne pas afficher les intervalles de confiance (`se=FALSE`) ;
- Le niveau des intervalles de confiance peut être changé via `level` ;
- Voir l'aide de la fonction `stat_smooth()` pour plus de détails.

Courbe de tendance

Par exemple, avec un lissage par régression linéaire, avec des intervalles de confiance de prévision à 90% :

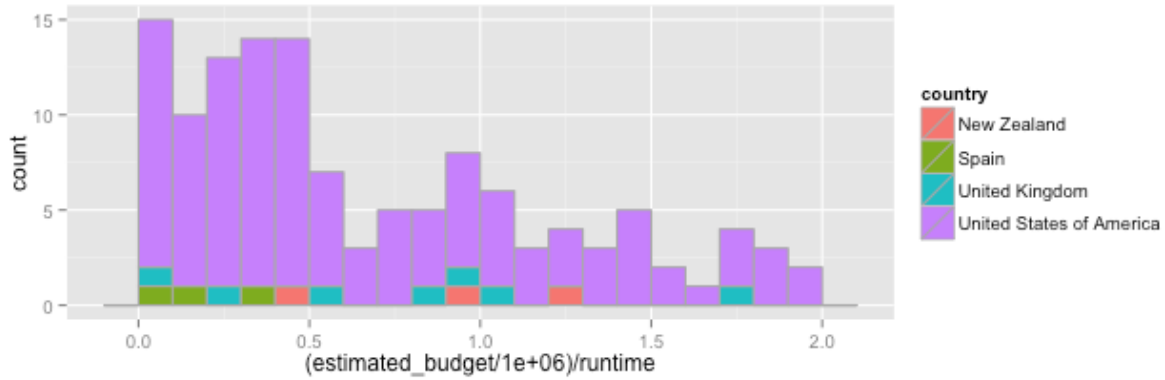
```
ggplot(data = films, aes(x = estimated_budget, y = gross_revenue)) +
  geom_point() + stat_smooth(method = "lm", level = 0.9)
```



Histogramme

- Les histogrammes se font avec `geom_histogram()` ;
- On peut choisir la fenêtre via le paramètre `binwidth` ;
- La fenêtre par défaut est 1/30 de l'étendue.

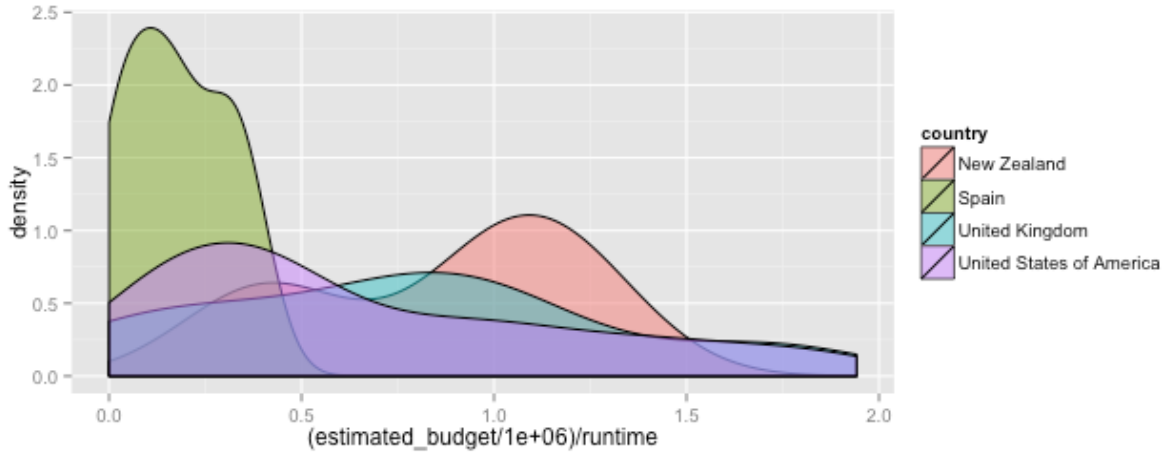
```
ggplot(data = films_reduit,
       aes(x = (estimated_budget/1000000)/runtime,
           fill = country)) +
  geom_histogram(binwidth = 0.1, colour = "dark grey")
```



Densité

- Une estimation de la densité, avec un noyau gaussien, s'effectue avec `geom_density()` ;
- Le noyau peut être changé via le paramètre `kernel` ;
- La page d'aide `?stat_density` fournit plus de détails.

```
ggplot(data = films_reduit,
       aes(x = (estimated_budget/1000000)/runtime,
           fill = country)) +
  geom_density(colour = "black", alpha = .5)
```



Exercices

À partir du jeu de données `diamonds` :

1. Créer le `data.frame` `diamonds_s`, un échantillon de `diamonds` de taille $n = 1000$
2. Représenter le nuage de points du prix (`price`) en fonction de la masse (`carat`), afficher les points en rouge ;
3. Reprendre le graph de la question 2. en colorant les points en fonction de la coupe (`cut`) ;
4. Afficher l'histogramme des masses des diamants (`carat`), avec une fenêtre de 0.05 ;
5. Afficher les boîtes à moustaches de la masse (`carat`) pour chaque coupe (`cut`).

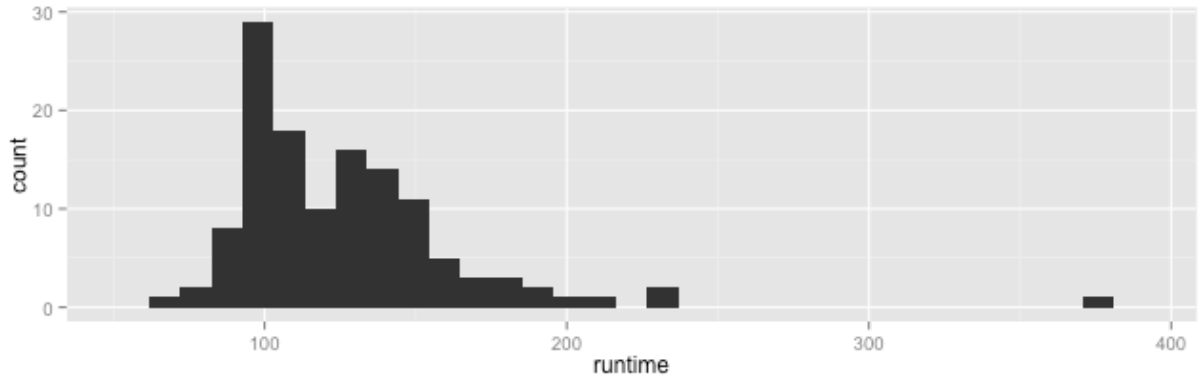
Fonctions statistiques

- Les fonctions `geom_*()` s'appuient, pour certaines, sur des fonctions `stat_*()` ;
- Ces fonctions permettent d'effectuer des opérations statistiques sur les données brutes ;
- Des variables sont créées lors de l'appel à ces fonctions ;
- On peut accéder à ces variables... à condition de connaître leur nom !

Fonctions statistiques

- Par exemple, pour produire un histogramme, la fonciton `geom_histogram()` s'appuie sur `stat_bin()` ;
- La fonction `stat_bin()` créé et retourne (entre autres) :
  - la variable `count`, qui indique le nombre d'observations pour chaque intervalle,
  - la variable `density`, qui donne une estimation de la densité de points dans chaque classe (avec mise à l'échelle pour que l'intégrale vaille 1 sur le support).

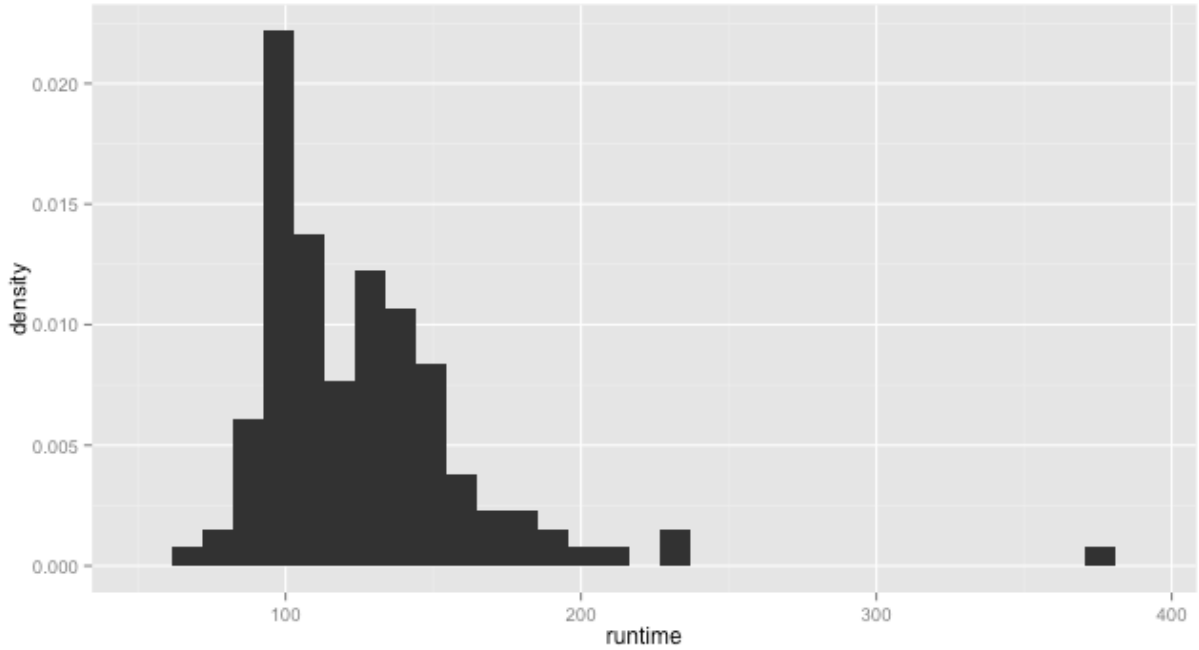
```
ggplot(data = films_reduit, aes(x = runtime)) + geom_histogram()
```



Fonctions statistiques

- Si on désire afficher une estimation de la densité, plutôt que le nombre d'observations :
  - on va chercher la variable `density` retournée par `stat_bin()`, en entourant son nom de "`..`" :

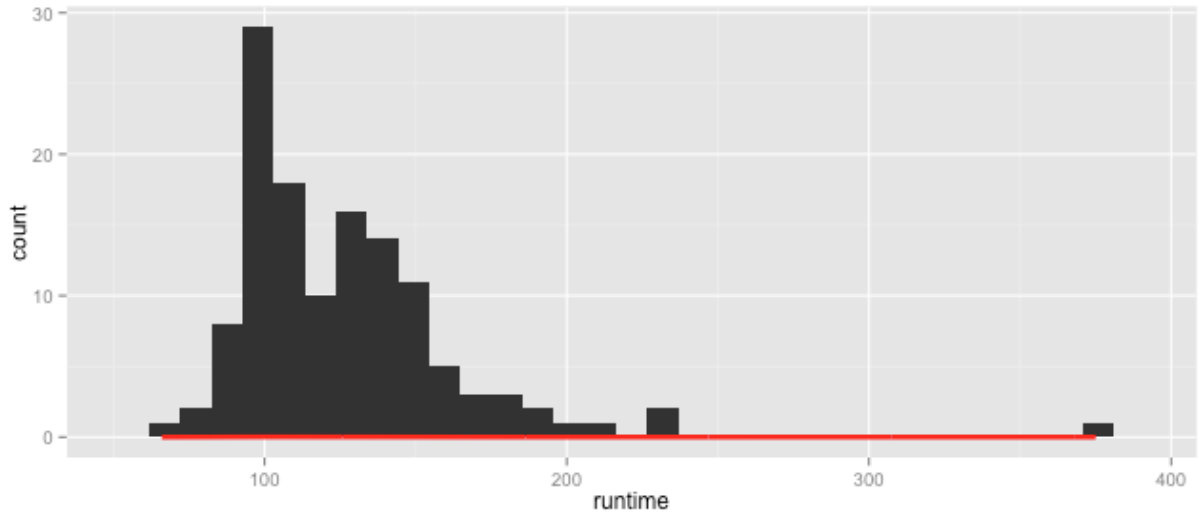
```
ggplot(data = films_reduit, aes(x = runtime)) + geom_histogram(aes(y = ..density..))
```



Fonctions statistiques

- L'ajout d'une courbe de densité sur un histogramme se fait en additionnant deux couches ;
- En faisant (silencieusement) appel à la fonction `stat_bin()` ;
- Et en appelande la fonction `stat_density()` :

```
p <- ggplot(data = films_reduit, aes(x = runtime))
p + geom_histogram() + geom_line(stat="density", col = "red", size = 1.2)
```

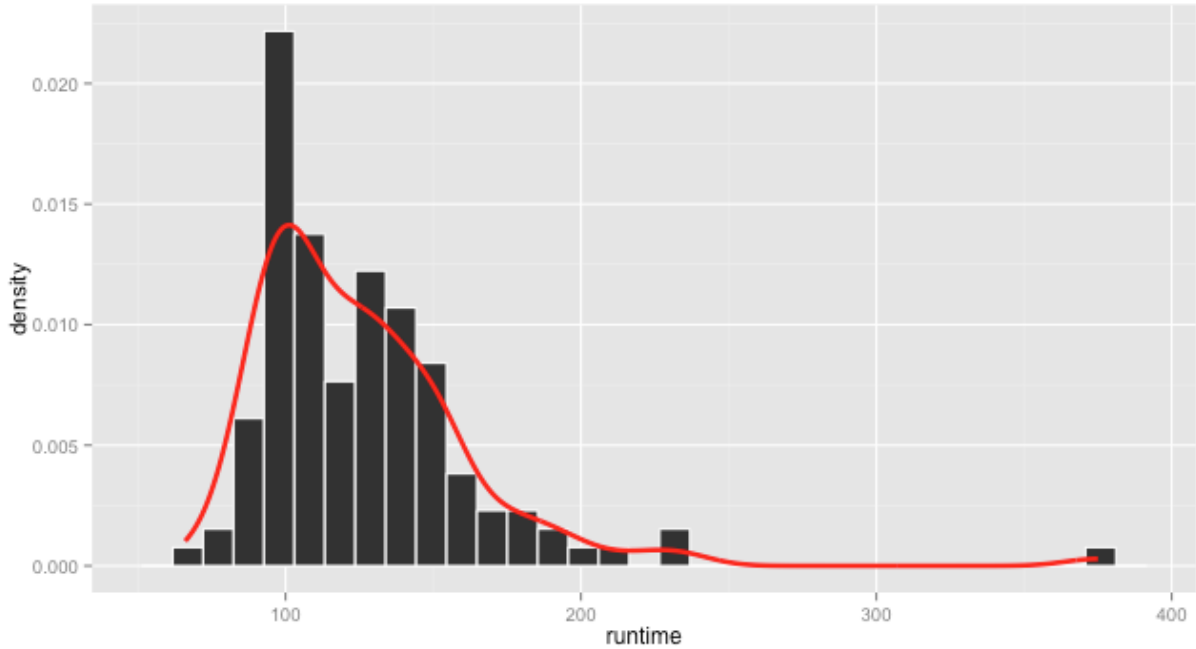


Fonctions statistiques

- Mais les valeurs de l'axe des ordonnées sont celles retournées par `stat_bin()` et `stat_density()` !
- On observe donc une densité toute plate.
- Il suffit alors de changer le mapping du paramètre esthétique `y` :
  - `y = ..density..` ;
- La valeur utilisée pour l'axe des `y` sera donc la densité plutôt que le comptage.

Fonctions statistiques

```
p <- ggplot(data = films_reduit, aes(x = runtime, y = ..density..))
p + geom_histogram(colour = "white") +
  geom_line(stat="density", col = "red", size = 1.2)
```



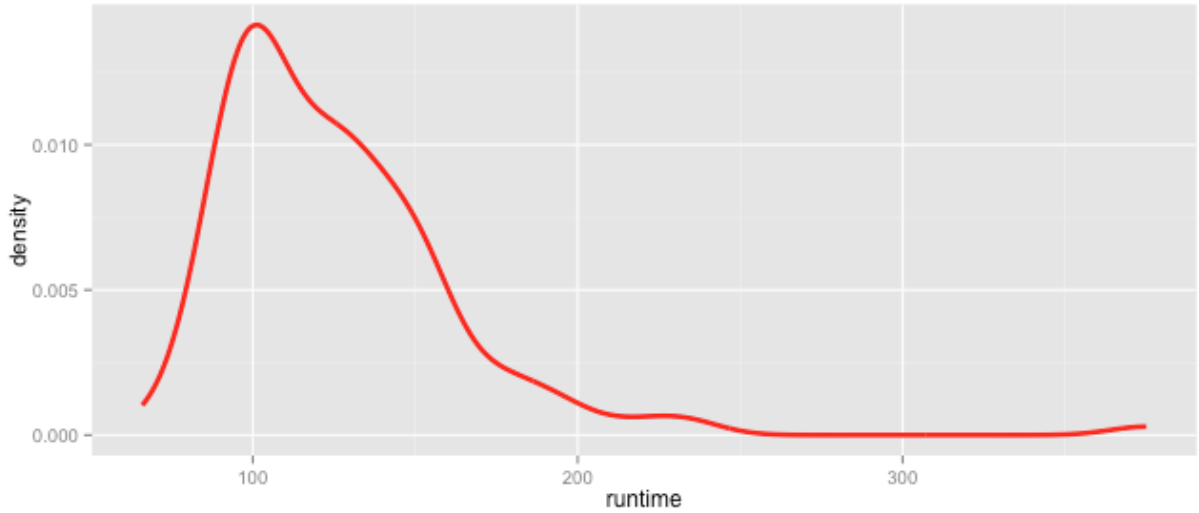
Fonctions statistiques

FONCTION	DESCRIPTION
`stat_bin()`	répartition des données en classes
`stat_contour()`	calculer les contours des données en 3d
`stat_density()`	estimation de densité 1d par la méthode du noyau
`stat_density2d()`	estimation de densité 2d
`stat_identity()`	ne transforme pas les données
`stat_qq()`	qqplot (droite de Henry)
`stat_quantile()`	quantiles continus
`stat_smooth()`	lissage
`stat_sum()`	somme les valeurs uniques
`stat_summary()`	appliquer une fonction pour faire des summaries sur les valeurs de y
`stat_unique()`	retire les valeurs dupliquées

Fonctions statistiques

- Les fonctions principales de type `stat_*()` peuvent être :
  - appelées directement,
  - appelées via le paramètre `stat` d'une fonction `geom_*()`, en fournissant le suffixe.

```
ggplot(data = films_reduit, aes(x = runtime, y = ..density..)) +
  geom_line(stat="density", col = "red", size = 1.2)
```



Exercices

1. Afficher l'histogramme des masses des diamants (`carat` dans le `data.frame` `diamonds`) ;
2. Afficher une estimation de la densité de la masse des diamants ;
3. Faire figurer sur un même graphique une visualisation de la répartitition des masses de diamants, ainsi qu'une estimation de la densité.

Les fonctions d'échelles

- Les fonctions `scale_*()` permettent de **définir** et **contrôler** le mapping entre les données et les attributs esthétiques ;
- Chaque paramètre esthétique possède son échelle et sa fonction `scale_*()` ;
- On peut décomposer les échelles en 4 catégories :
  - échelles de position,
  - échelles de couleur,
  - échelles manuelles discrètes,
  - identity (pas de mise à l'échelle).

Les fonctions d'échelles

Le lien est donc fait entre :

- Le **domaine** : l'espace de données. Si les données sont
  - discrètes (facteurs, logiques, chaînes de caractères) : le domaine correspond à une énumération des valeurs possibles,
  - continues (numerique) : le domaine est un intervalle ;
- La **gamme** : l'espace des esthétique. La gamme est :
  - discrète quand le domaine est discret, et consituée des valeurs correspondantes aux valeurs des données d'input,
  - continue quand le domaine est continu, et fournit alors un chemin indiquant comment passer d'une valeur à l'autre.

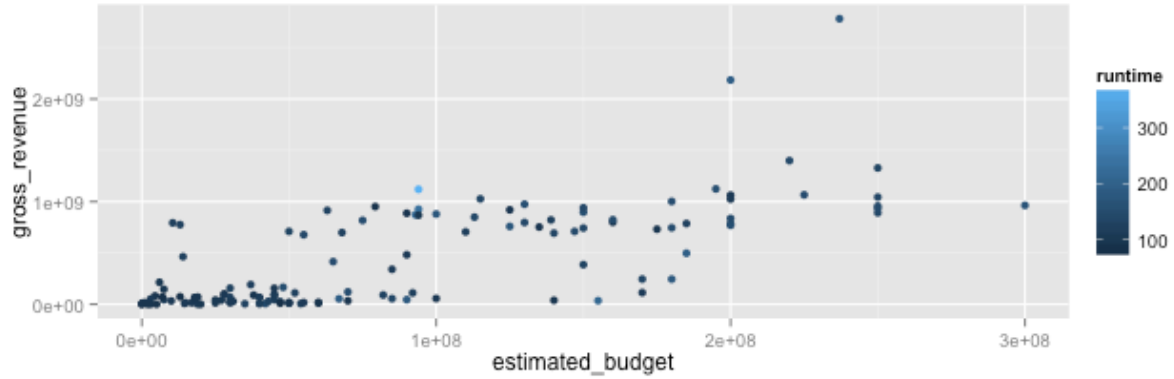
Les fonctions d'échelles

- Les échelles sont automatiquement créées lors de la création du graphique ;
- On peut les modifier avec les fonctions `scale_*()` ;
- Pour cela, il faut en ajouter une nouvelle, en couche supplémentaire ;
- Le nom des échelles est composé de deux ou trois parties :
  - le préfixe `scale_`,
  - le nom de l'esthétique,
  - le nom de l'échelle à utiliser.
- Exemple : pour changer l'échelle qui gère la couleur, en présence d'une variable continue dont dépend la couleur : `scale_colour_gradient()`

Les fonctions d'échelles

- Le graphique de base

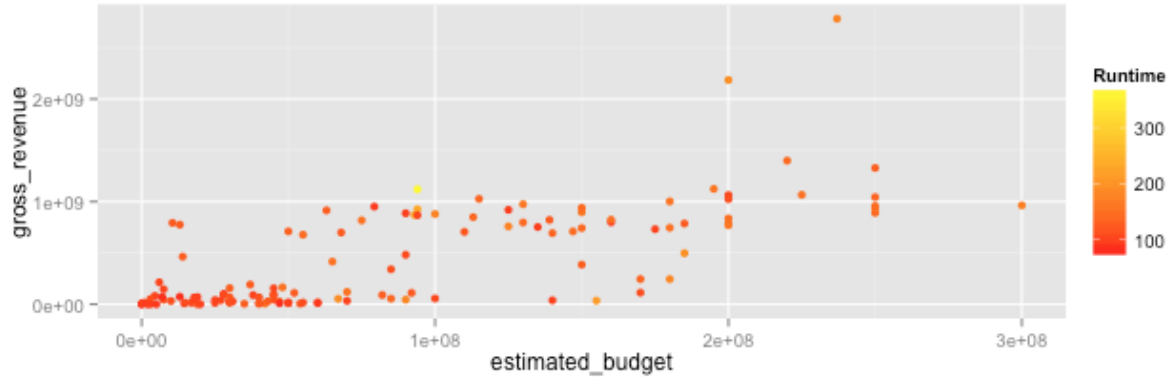
```
p <- ggplot(data = films_reduit, aes(x = estimated_budget,
                                     y = gross_revenue, colour = runtime)) +
  geom_point()
p
```



Les fonctions d'échelles

- Changer l'échelle de couleur pour que :
  - les films les plus courts soient en jaunes,
  - les films les plus longs en rouge,
  - le tout en dégradé,
  - tout en changeant le titre de la légende.

```
p + scale_colour_gradient(name = "Runtime", low = "#FF0000", high ="#FFFF00")
```



Les fonctions d'échelles

ESTHÉTIQUE	VARIABLE DISCRÈTE	VARIABLE CONTINUE
Transparence (alpha)	`scale_alpha_discrete()`	`scale_alpha_continuous()`
	`scale_alpha_manual()`	
	`scale_alpha_identity()`	`scale_alpha_identity()`
Couleur (colour)	`scale_colour_discrete()`	`scale_colour_continuous()`
	`scale_colour_brewer()`	`scale_colour_dilstiller()`
	`scale_colour_grey()`	`scale_colour_gradient()`
	`scale_colour_hue()`	`scale_colour_gradient2()`
	`scale_colour_manual()`	`scale_colour_gradientn()`
	`scale_colour_identity()`	`scale_colour_identity()`

Les fonctions d'échelles

ESTHÉTIQUE	VARIABLE DISCRÈTE	VARIABLE CONTINUE
Remplissage (fill)	`scale_fill_discrete()`	`scale_fill_continuous()`
	`scale_fill_brewer()`	`scale_fill_distiller()`
	`scale_fill_grey()`	`scale_fill_gradient()`
	`scale_fill_hue()`	`scale_fill_gradient2()`
	`scale_fill_manual()`	`scale_fill_gradientn()`
	`scale_fill_identity()`	`scale_fill_identity()`
Type de ligne (linetype)	`scale_linetype_discrete()`	`scale_linetype_continuous()`
	`scale_linetype_manual()`	
	`scale_linetype_identity()`	`scale_linetype_identity()`
Forme	`scale_shape_discrete()`	`scale_shape_continuous()`
	`scale_shape_manual()`	
	`scale_shape_identity()`	`scale_shape_identity()`

Les fonctions d'échelles

ESTHÉTIQUE	VARIABLE DISCRÈTE	VARIABLE CONTINUE
Taille (size)	`scale_size_discrete()`	`scale_size_continuous()`
	`scale_size_manual()`	`scale_size_area()`
	`scale_size_identity()`	`scale_size_identity()`
Position (x, y)	`scale_x_discrete()`	`scale_x_continuous()`
	`scale_y_discrete()`	`scale_y_continuous()`
		`scale_x_date()`
		`scale_y_date()`
		`scale_x_datetime()`
		`scale_y_datetime()`
		`scale_x_log10()`
		`scale_y_log10()`
		`scale_x_reverse()`
		`scale_y_reverse()`
		`scale_x_sqrt()`
		`scale_y_sqrt()`

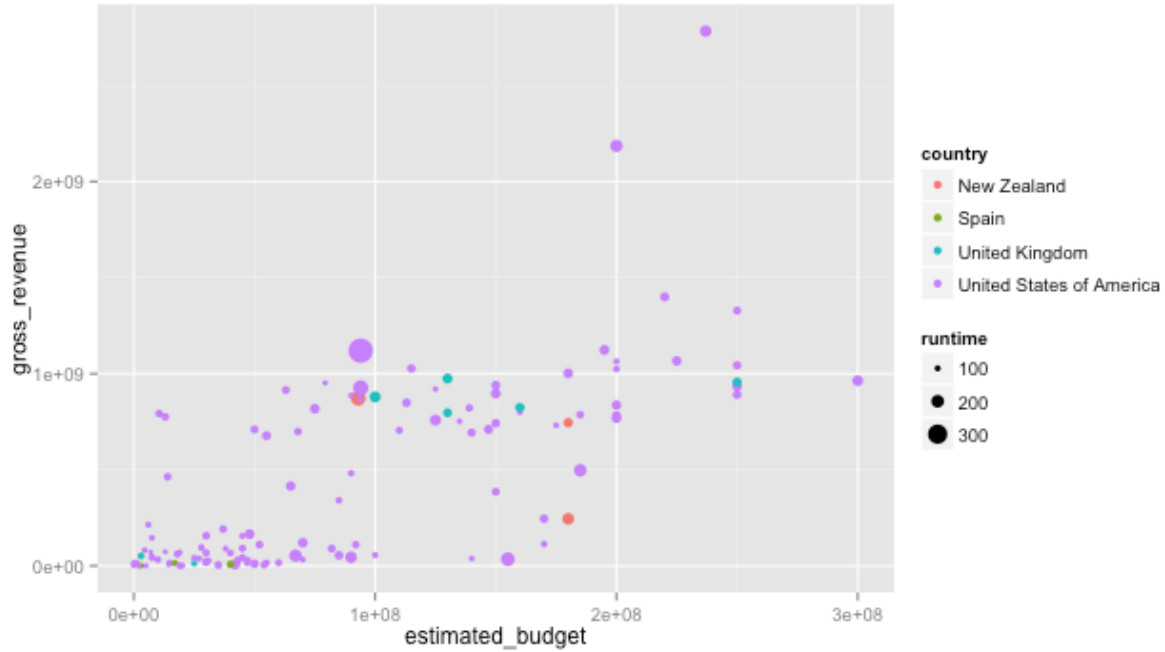
Les fonctions d'échelles

- Revenu en fonction du budget ;
- Taille et couleur des points : dépend du revenu et de la longueur, resp. ;
- Couleur : dépend d'une variable discrète ;
- Taille : dépend d'une variable continue.

```
p <- ggplot(data = films_reduit, aes(x = estimated_budget,
                                     y = gross_revenue,
                                     colour = country,
                                     size = runtime)) +
  geom_point()
```

Les fonctions d'échelles

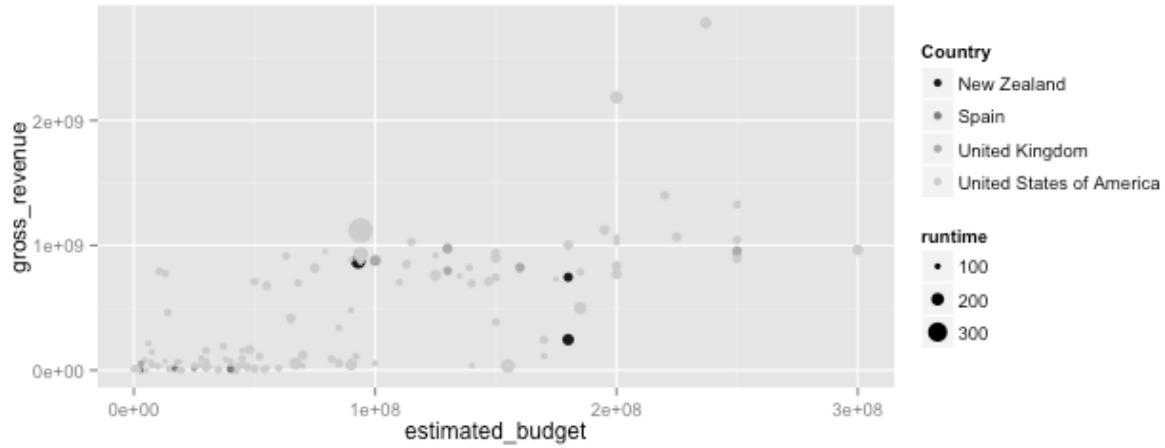
```
p
```



Les fonctions d'échelles

- Dégradé de gris pour la couleur ;
- Valeurs manquantes en orange ;
- Modification du titre de la légende.

```
p + scale_colour_grey(name = "Country", start = .1, end = .8,
                      na.value = "orange")
```



Les fonctions d'échelles

- Définissons nous-même la couleur pour chaque pays ;
- Si on souhaite en plus afficher un nom différent, il faut faire attention à bien effectuer le matching... ;
- La variable "country" est transformée en facteur ;
- L'ordre est alphabétique (mais peut être changé avec la fonction `order()`)

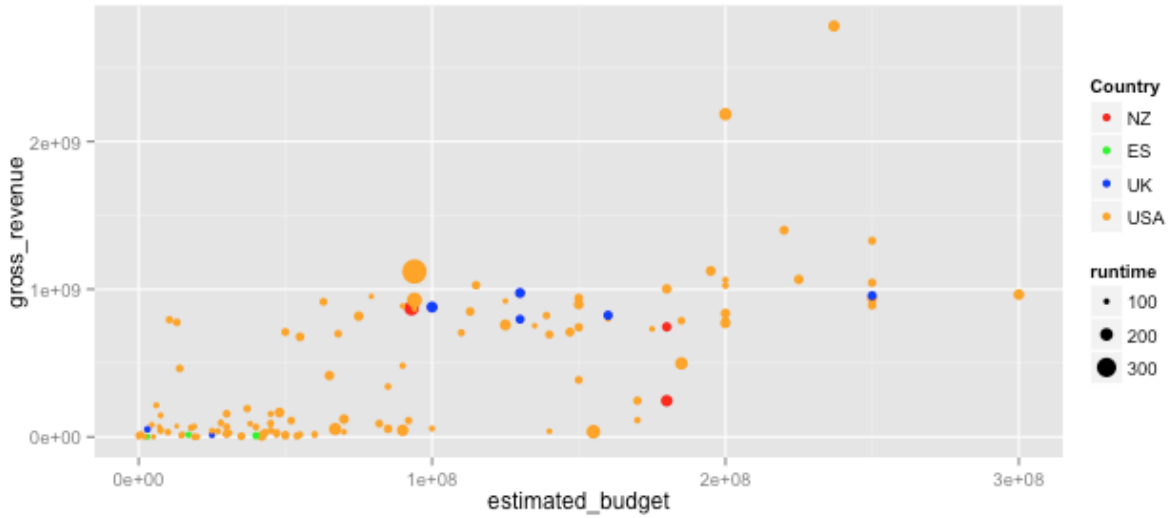
```
levels(factor(films_reduit$country))
```

```
## [1] "New Zealand"              "Spain"
## [3] "United Kingdom"           "United States of America"
```

Les fonctions d'échelles

- Une première solution :

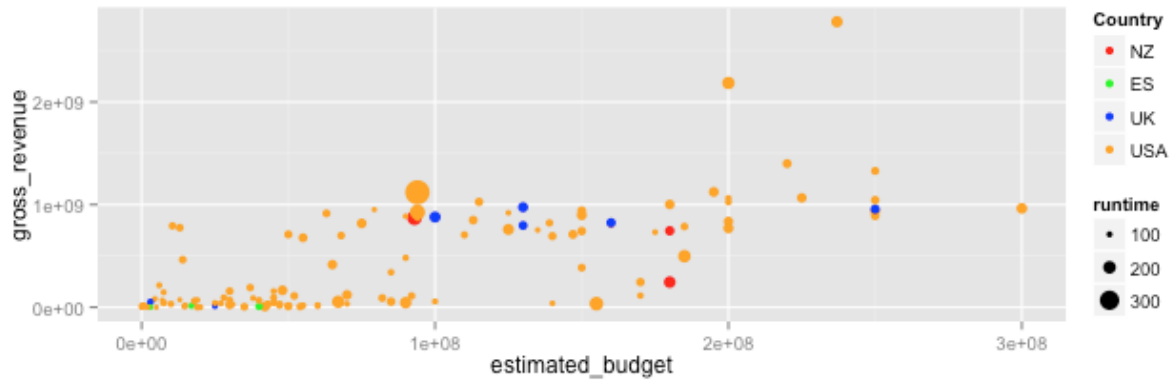
```
p + scale_colour_manual(name = "Country",
                        values = c("red", "green", "blue", "orange"),
                        labels = c("NZ", "ES", "UK", "USA"))
```



Les fonctions d'échelles

- Une seconde solution, plus sûre...

```
(p <- p + scale_colour_manual(name = "Country",
                              values = c("Spain" = "green", "New Zealand" = "red",
                                         "United States of America" = "orange",
                                         "United Kingdom" = "blue"),
                              labels = c("Spain" = "ES", "New Zealand" = "NZ",
                                         "United States of America" = "USA",
                                         "United Kingdom" = "UK")))
```



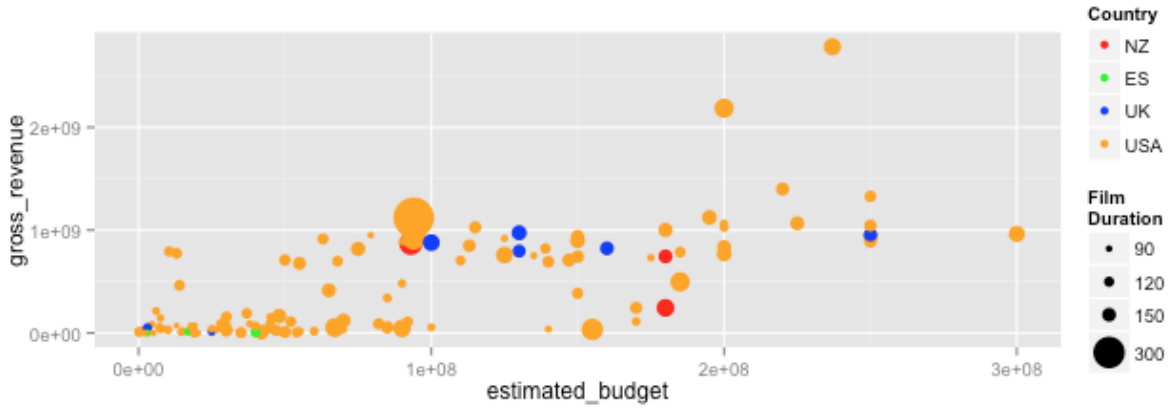
Les fonctions d'échelles

- Changeons également la taille des points.

```
range(films_reduit$runtime)
```

```
## [1]  66 375
```

```
p + scale_size_continuous(name = "Film\nDuration",
                          breaks = c(0, 60, 90, 120, 150, 300, Inf),
                          range = c(1,10))
```

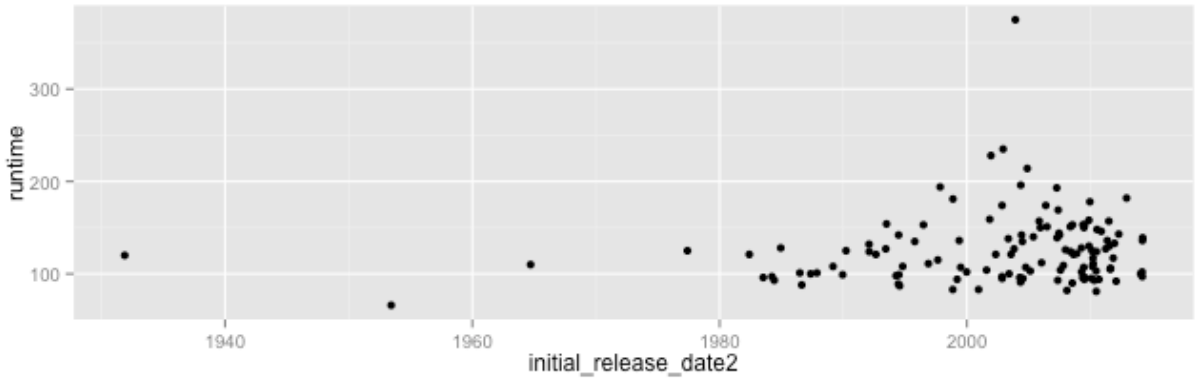


Les fonctions d'échelles

- Un autre graphique, avec des dates en abscisses

```
films_reduit$initial_release_date2 <- as.Date(films_reduit$initial_release_date)
(p_2 <- ggplot(data = films_reduit,
               aes(x = initial_release_date2, y = runtime)) +
  geom_point())
```

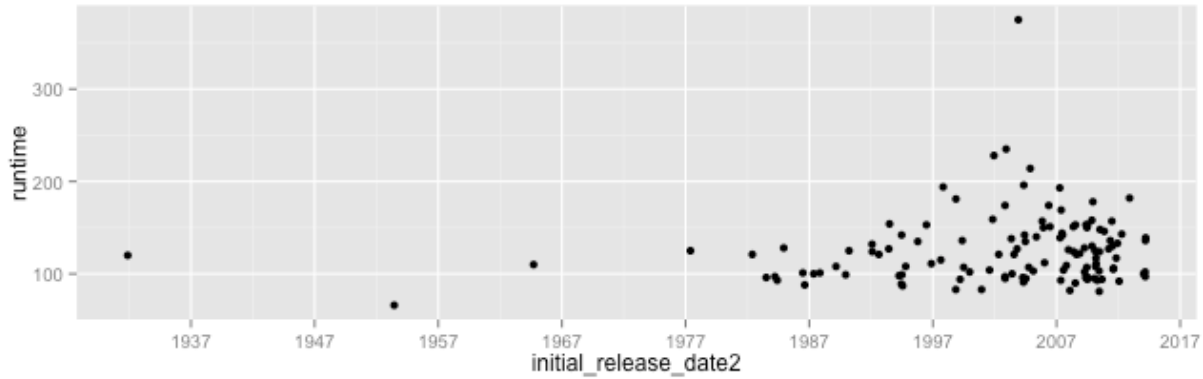
```
## Warning: Removed 3 rows containing missing values (geom_point).
```



Les fonctions d'échelles

- La fonction `scale_x_date()` est pratique pour gérer les affichages des étiquettes des marques.

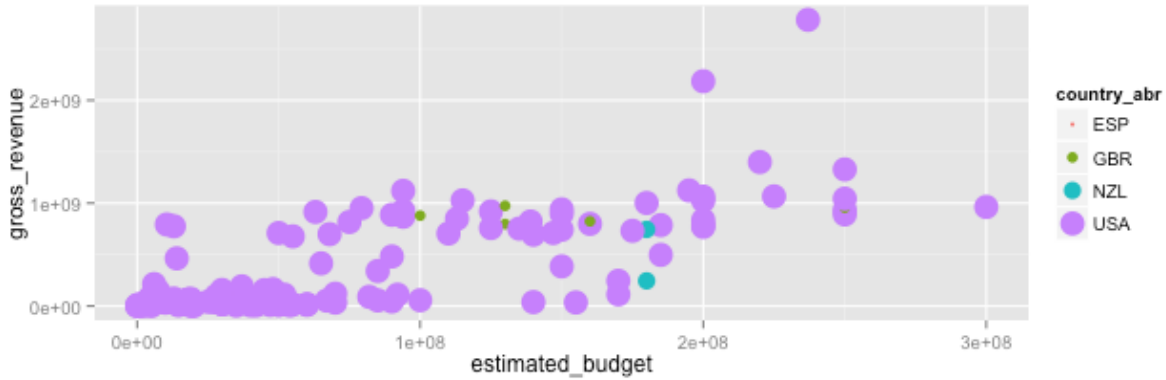
```
library(scales)
p_2 + scale_x_date(breaks = date_breaks("10 year"), labels = date_format("%Y"))
```



Les fonctions d'échelles

- `ggplot2` essaie de combiner les légendes autant que possible ;
- Par exemple, deux esthétiques de couleur et de forme concernant une même variable seront regroupés pour la légende, au lieu d'avoir deux légendes ;
- Si on modifie une légende, pour que l'autre le soit aussi, il faut impérativement conserver le même nom !

```
(p <- ggplot(data = films_reduit, aes(x = estimated_budget, y = gross_revenue,
                                      colour = country_abr, size = country_abr)) +
  geom_point())
```



Les fonctions d'échelles

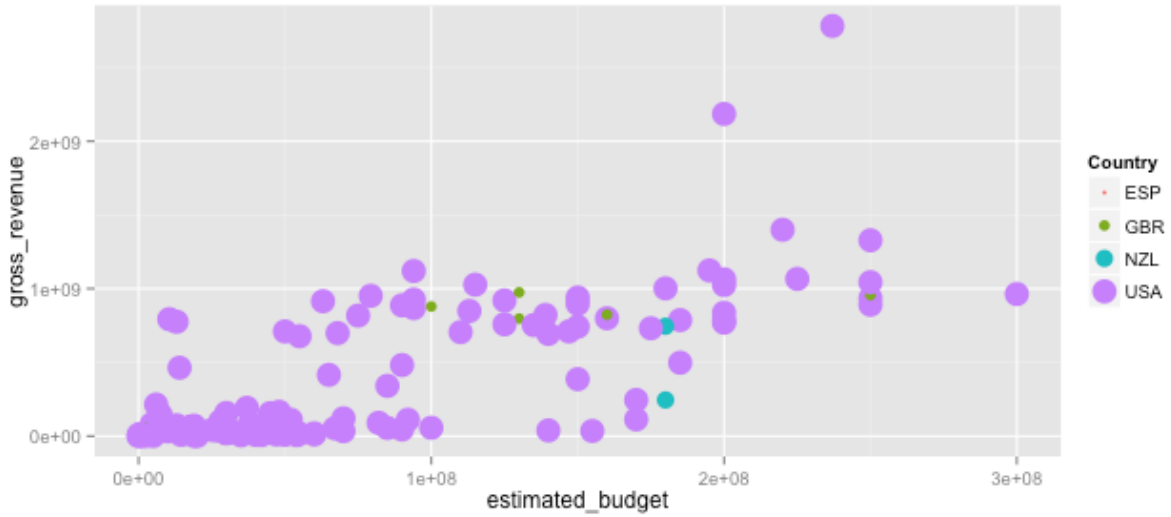
- Dans cet exemple, si on change le nom de l'échelle concernant la couleur, sans modifier celle de la taille : deux légendes aparaissent.

```
p + scale_colour_discrete(name = "Country")
```

Les fonctions d'échelles

- On peut rétablir une seule échelle en donnant le même nom aux échelles.

```
p + scale_colour_discrete(name = "Country") +
  scale_size_discrete(name = "Country")
```



Exercices

1. Afficher un nuage de points de la masse des diamants (`carat`) en fonction du prix associé (`price`) ;
2. Reprendre ce graphique et faire dépende la couleur des points de la qualité de la coupe (`cut`) ;
3. Changer le nom de la légende en "Qualité de la coupe" ;
4. Indiquer la traduction française dans la légende, plutôt que le terme anglais ;
5. Changer les couleurs par défaut des points pour 5 couleurs de votre choix.

Groupes

- `ggplot2` effectue des regroupements automatiquement dans de nombreux cas ;
- C'est le cas quand un paramètre esthétique de couleur ou de remplissage est basé sur une variable facteur ;
- Les groupes s'effectuent en fonction des interactions des variables qualitatives d'un graphique ;
- On peut vouloir définir soi-même les groupes, via le paramètre `group`.

```
library(reshape2)
df <- data.frame(year = rep(1949:1960, each = 12), month = rep(1:12, 12),
passengers = c(AirPassengers))
```

Groupes

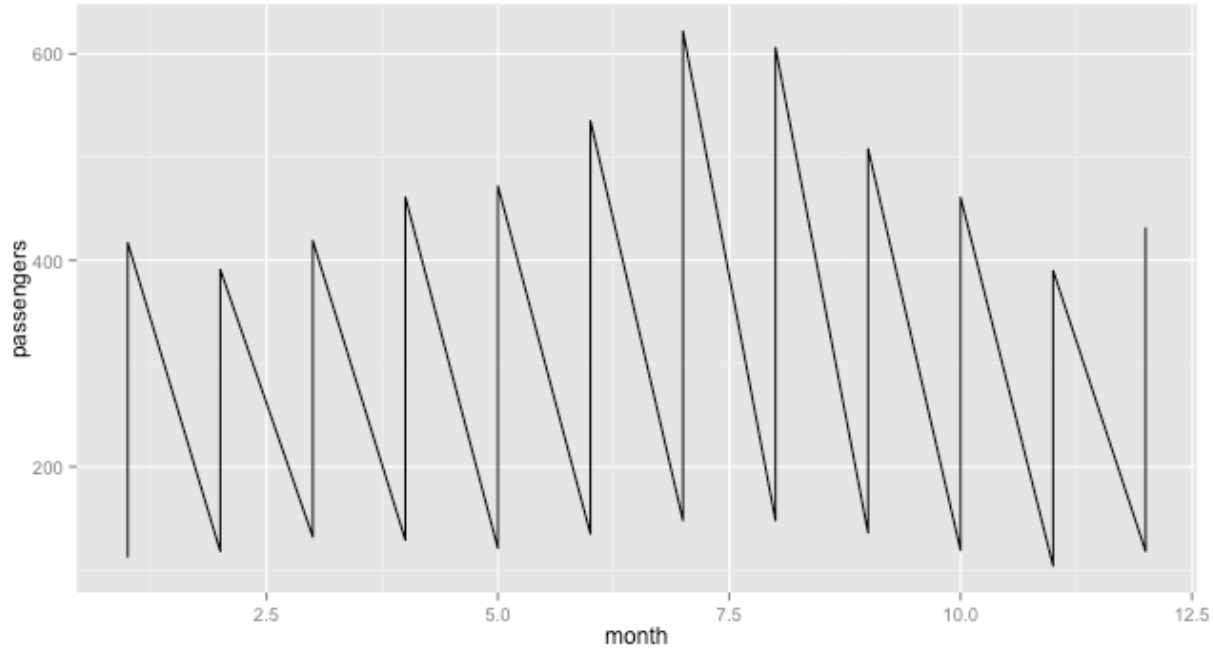
```
head(df)
```

```
##   year month passengers
## 1 1949     1        112
## 2 1949     2        118
## 3 1949     3        132
## 4 1949     4        129
## 5 1949     5        121
## 6 1949     6        135
```

Groupes

- Sans préciser de groupes :

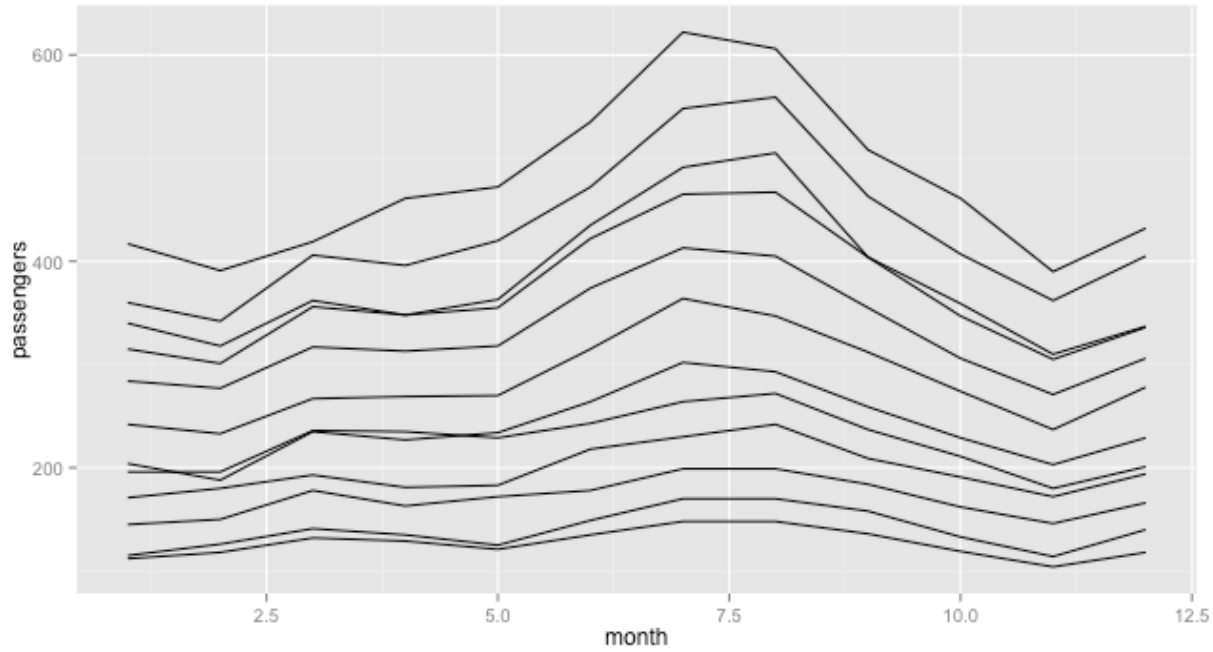
```
ggplot(data = df, aes(x = month, y = passengers)) + geom_line()
```



Groupes

En précisant que l'on souhaite regrouper par années

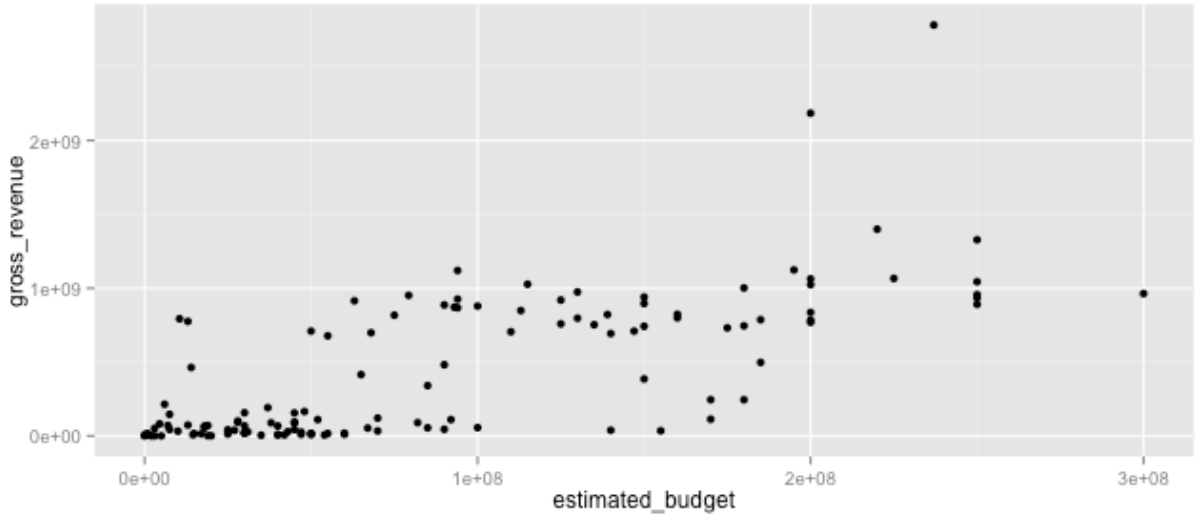
```
ggplot(data = df, aes(x = month, y = passengers, group = year)) +
  geom_line()
```



Annotations

- Il est possible d'ajouter du texte ou des objets géométrique sur un graphique ;
- Les exemples qui suivent s'appuient sur le graphique suivant :

```
p <- ggplot(data = films_reduit,
            aes(x = estimated_budget, y = gross_revenue)) +
  geom_point()
p
```



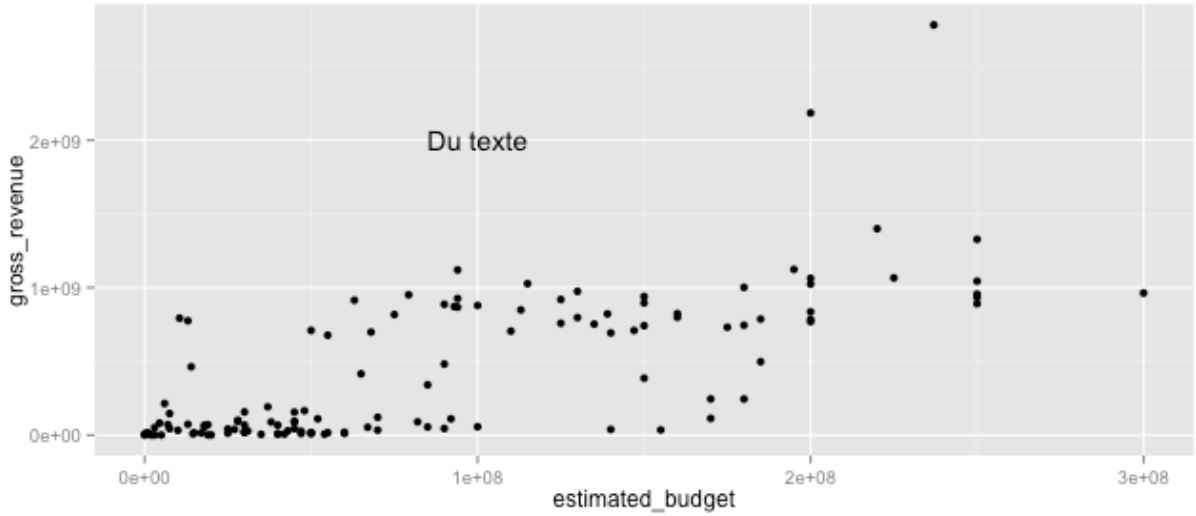
Annotations : ajout de texte

- Pour ajouter du texte : `geom_text()` ou `annotate()` ;
- Résultats plus performants avec `annotate()` ;
- Les données fournies à `annotate()` ne sont pas héritées ;
- Chaque annotation est une couche ;
- Le paramètre `geom` doit prendre la valeur `"texte"` ;
- Les paramètres `x` et `y` indiquent la (les) position(s) (x,y) du centre du texte ;
- Le paramètre `legend` doit recevoir un vecteur de caractères ;
- Le paramètre `colour` permet de changer la couleur, noire par défaut.

Annotations : ajout de texte

- Une annoation basique :

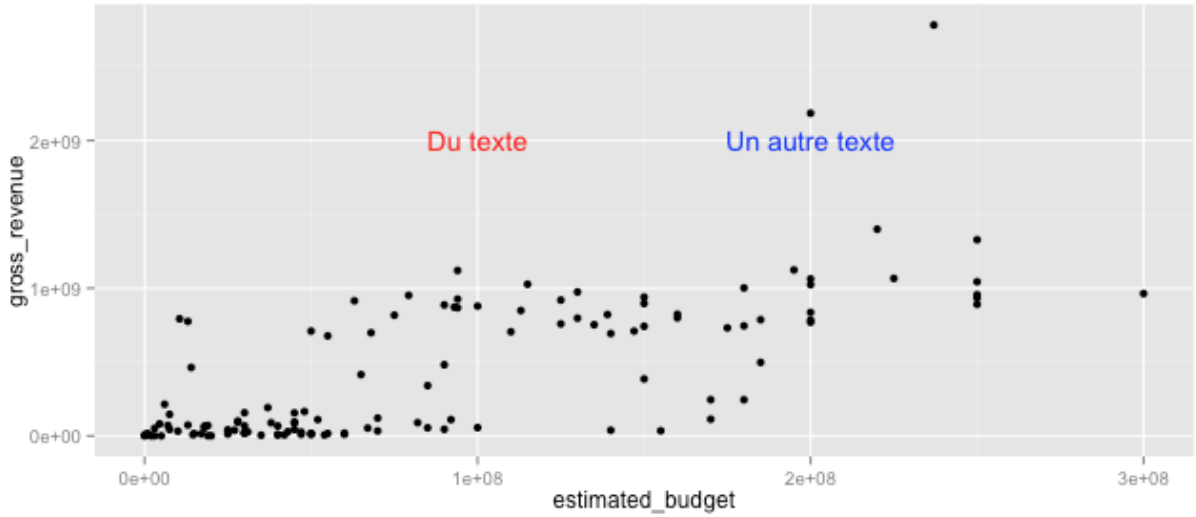
```
p + annotate("text", x = 1e8, y = 2e9, label = "Du texte")
```



Annotations : ajout de texte

- Avec deux textes, le premier en rouge, et l'autre en bleu :

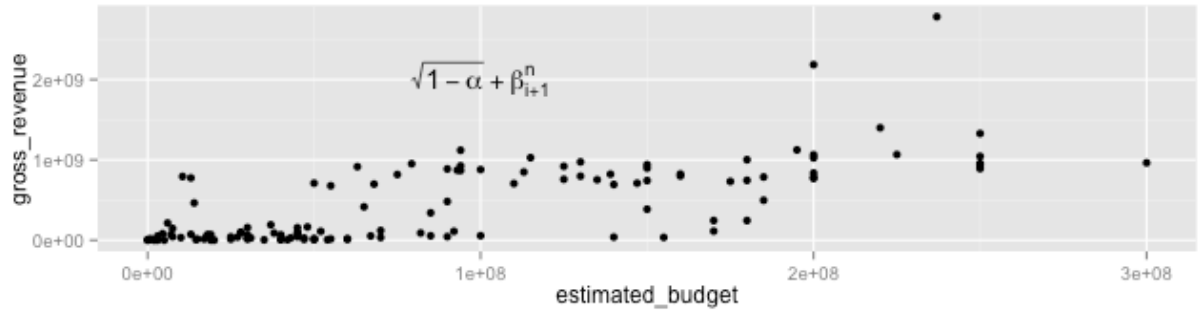
```
p + annotate("text", x = c(1e8, 2e8), y = 2e9,
             label = c("Du texte", "Un autre texte"),
             colour = c("red", "blue"))
```



Annotations : ajout de texte

- Pour ajouter des expression mathématiques : `parse = TRUE` :
  - lettres grecques : écrire leur nom en entier,
  - écrire en indice : `[texte]`,
  - écrire en exposant : `^texte`,
  - indice et exposant pour un terme : d'abord en indice, puis en exposant.

```
p + annotate("text", x = 1e8, y = 2e9,
             label = "sqrt(1-alpha) + beta[i+1]^n", parse = TRUE)
```



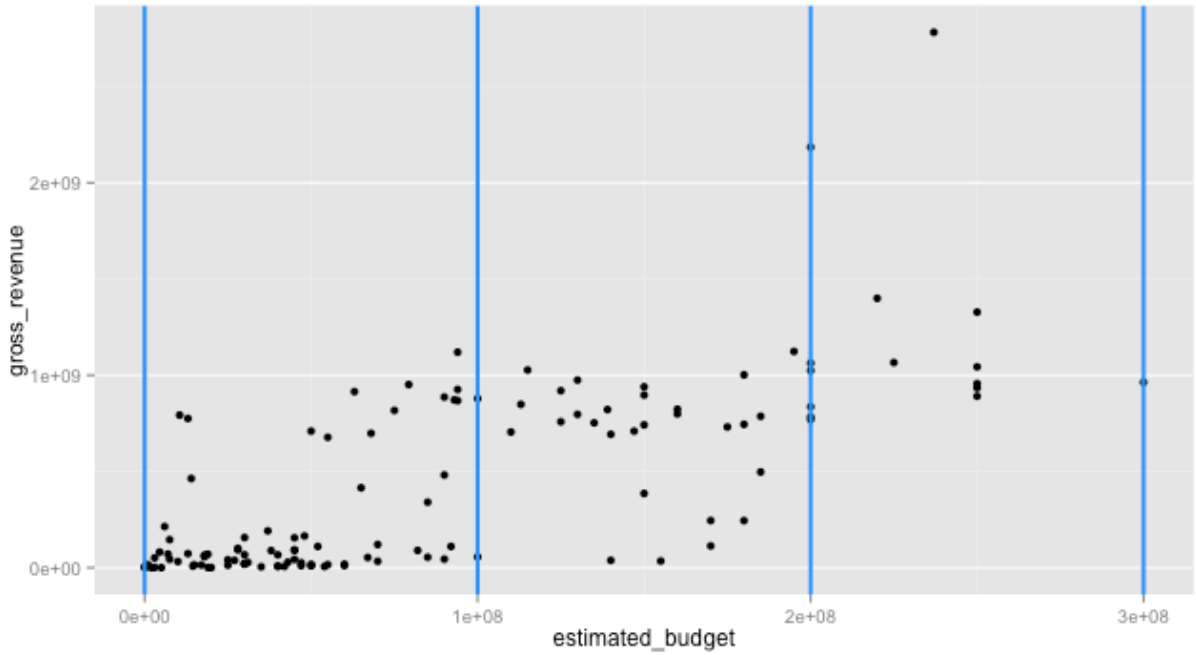
Annotations : ajout de lignes

- Quatre fonctions pour ajouter des lignes sur un graphique, en plus de `geom_line()` et `geom_path()` :
  - `geom_vline()` : ligne verticale ;
  - `geom_hline()` : ligne horizontale ;
  - `geom_abline()` : ligne spécifiée par sa pente et son ordonnée à l'origine ;
  - `geom_segment()` : segment (ou flèche en utilisant `arrow()`).

Annotations : ajout de lignes

- Ligne verticale :

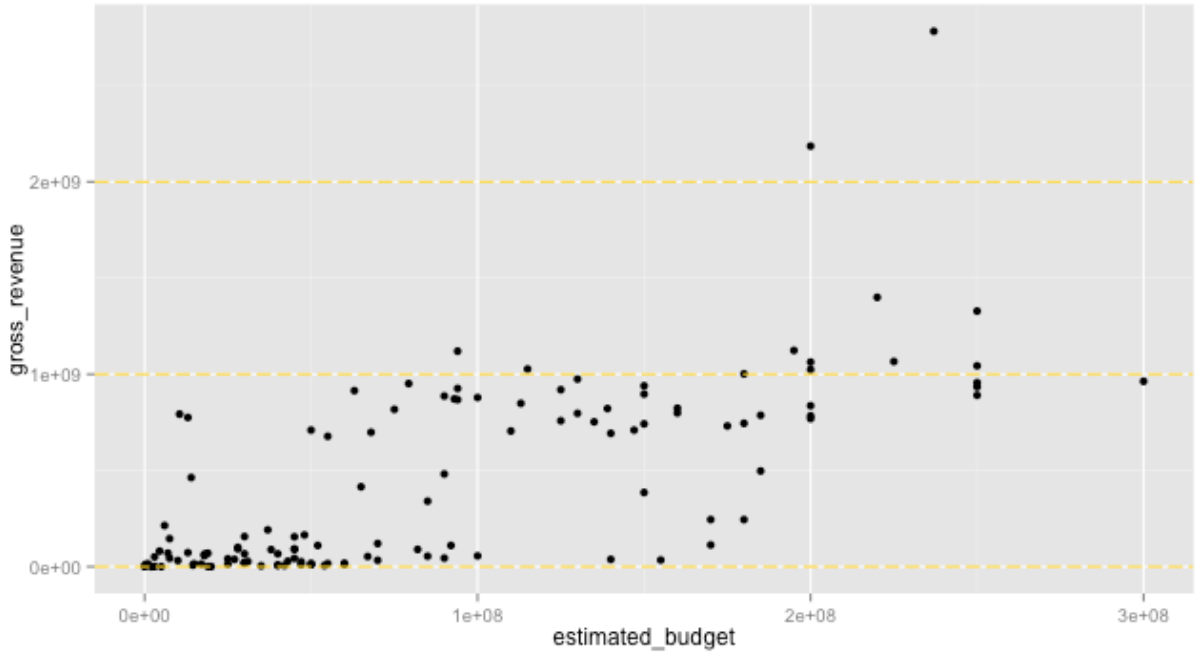
```
p + geom_vline(xintercept = seq(0, 3e8, by = 1e8),
               size = 1, col = "dodger blue")
```



Annotations : ajout de lignes

- Ligne horizontale :

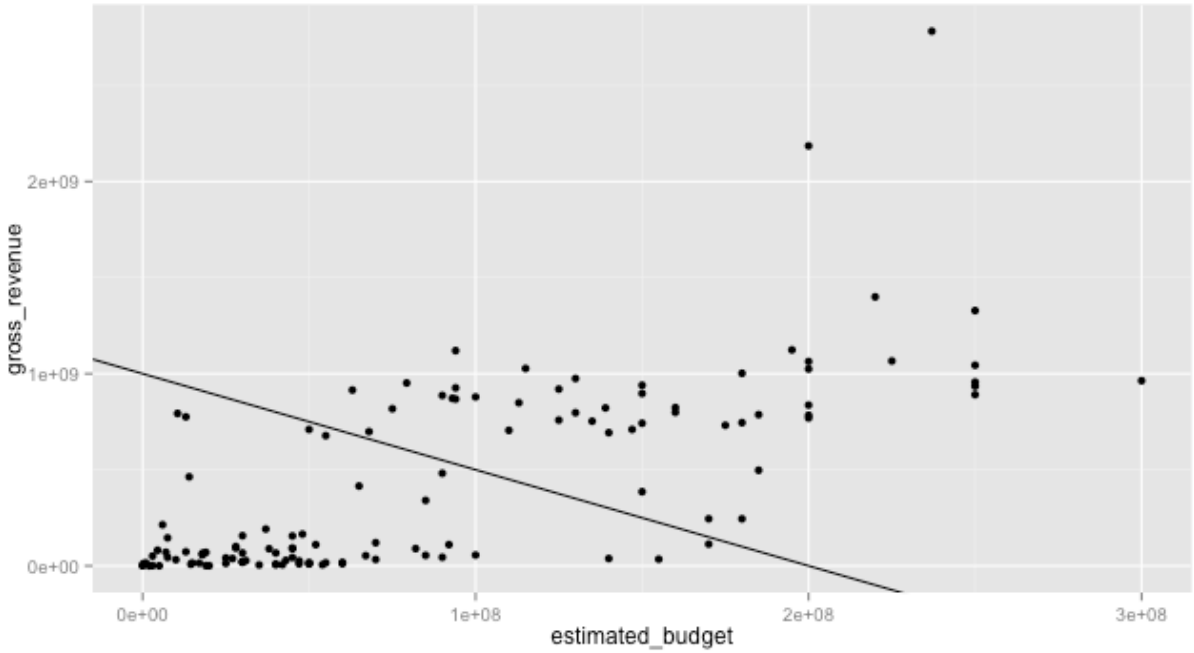
```
p + geom_hline(yintercept = seq(0, 2e9, by = 1e9),
               col = "gold", linetype = "longdash")
```



Annotations : ajout de lignes

- Droite :

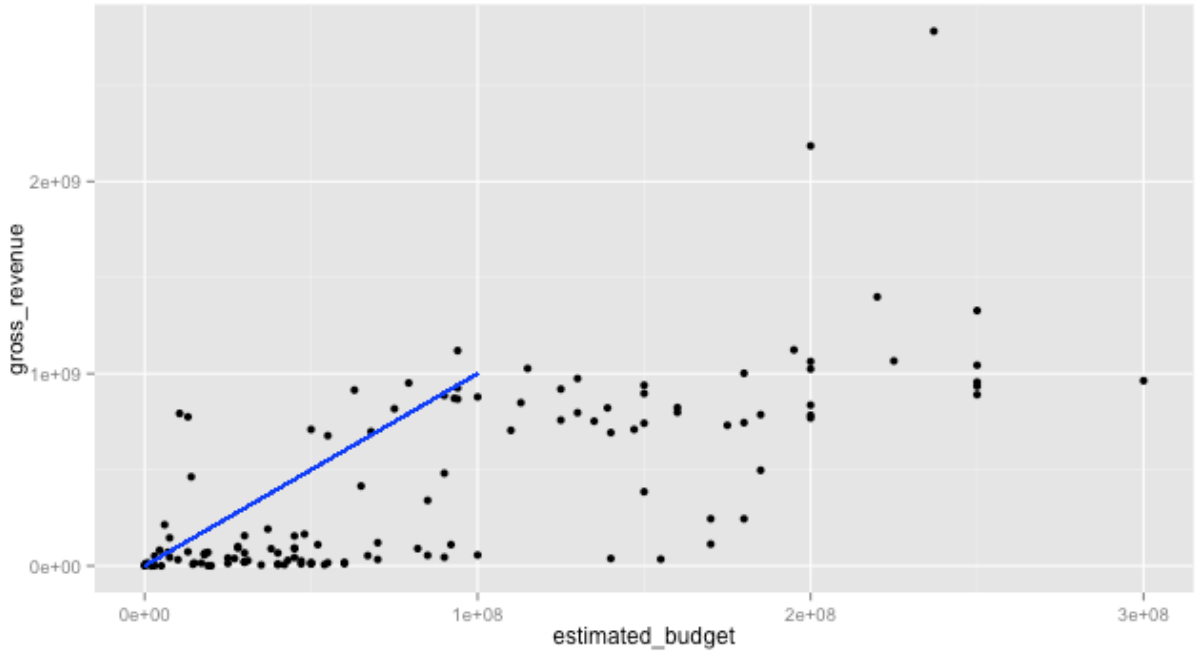
```
p + geom_abline(intercept = 1e9, slope = -5)
```



Annotations : ajout de lignes

- Segment :

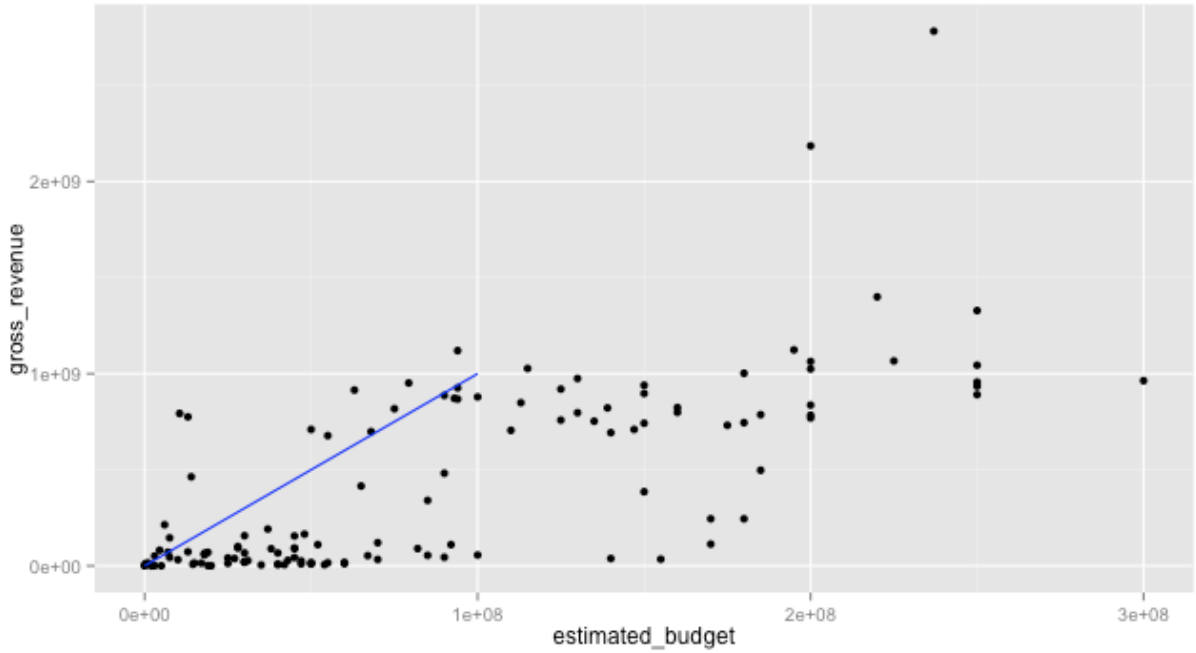
```
p + geom_segment(aes(x = 0, xend = 1e8, y = 0, yend = 1e9), col = "blue")
```



Annotations : ajout de lignes

- Pour les segments, on peut utiliser `annotate()` :

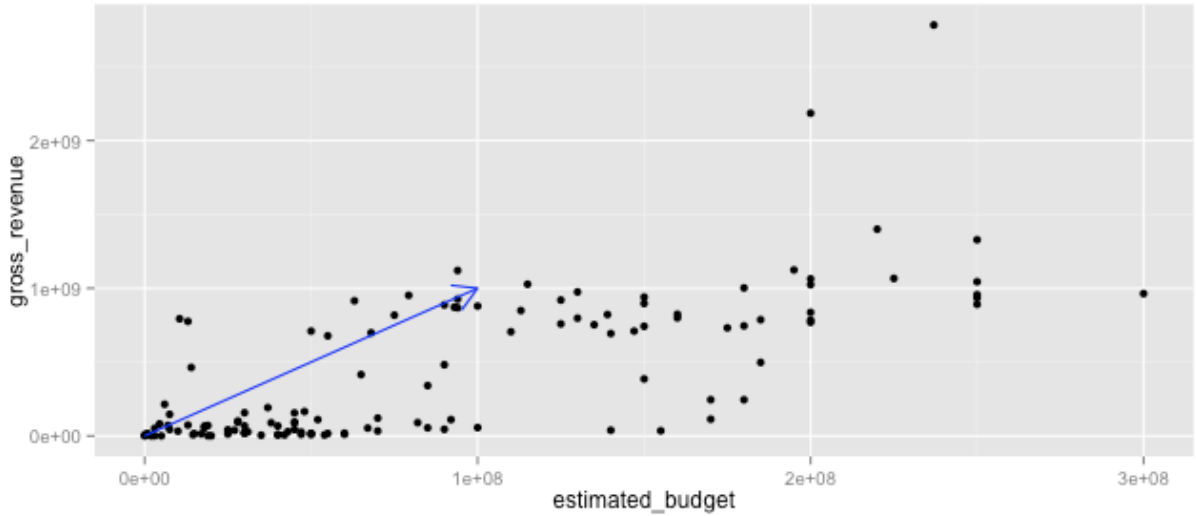
```
p + annotate(geom = "segment", x = 0, xend = 1e8,
             y = 0, yend = 1e9, col = "blue")
```



Annotations : ajout de lignes

- Pour tracer une flèche : `arrow()` (package `grid`) :

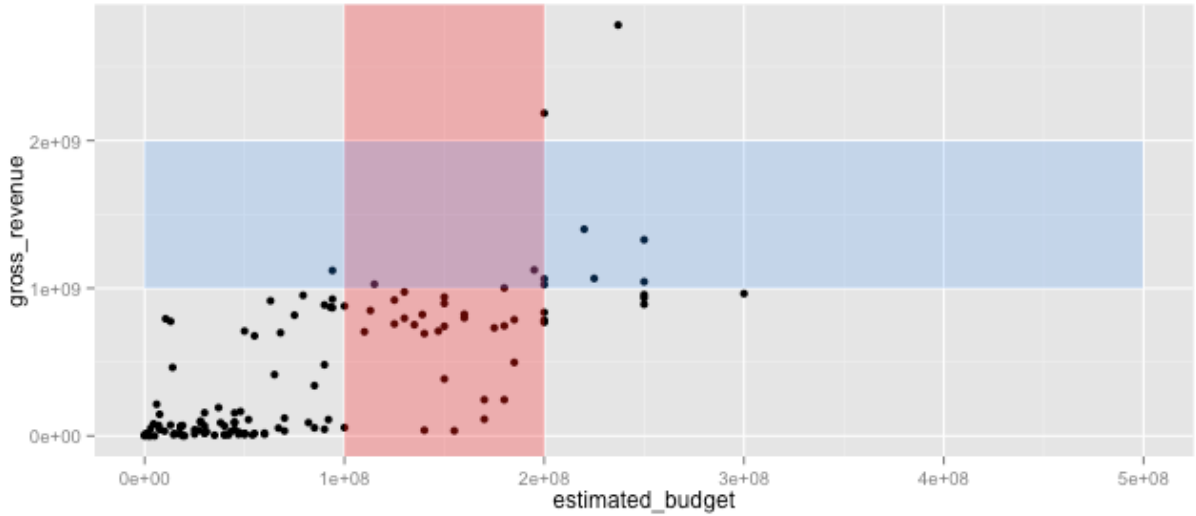
```
library(grid)
p + annotate(geom = "segment", x = 0, xend = 1e8,
             y = 0, yend = 1e9, col = "blue",
             arrow = arrow(length = unit(0.5, "cm")))
```



Annotations : ajout de rectangles

- L'ajout de rectangle peut se faire avec `geom_rect()` ou `annotate()` :

```
p + annotate(geom = "rect", xmin = 1e8, xmax = 2e8, ymin = -Inf, ymax = Inf,
             alpha = .3, fill = "red") +
  annotate(geom  ="rect", xmin = 0, xmax = 5e8, ymin = 1e9, ymax = 2e9,
           alpha = .2, fill = "dodger blue")
```



Exercices

À partir du graphique suivant :

```
p <- ggplot() + geom_point(data = diamonds[sample(1:nrow(diamonds), 1000),],
                           aes(x = carat, y = price, colour = cut))
```

1. Ajouter une ligne horizontale à $y = 10,000$, en pointillés gris ;
2. Ajouter le texte "Echantillon de taille 1000" à n'importe quel endroit du graphique.

Positions

- Pour modifier le positionnement de certains éléments des graphiques :

FONCTION	DESCRIPTION
`position_dodge`	évite les chevauchements, place les éléments côte à côte
`position_fill`	empile les éléments qui se chevauchent, en normalisant pour avoir une hauteur égale
`position_identity`	n'ajuste pas la position
`position_jitter`	place les éméments côte à côte en essyant d'optimiser l'espace
`position_stack`	empile les éléments qui se chevauchent

- On peut appeler ces fonctions via le paramètre `position` des fonctions `geom_*()`, en conservant uniquement le suffixe.

Positions : exemples pour un histogramme

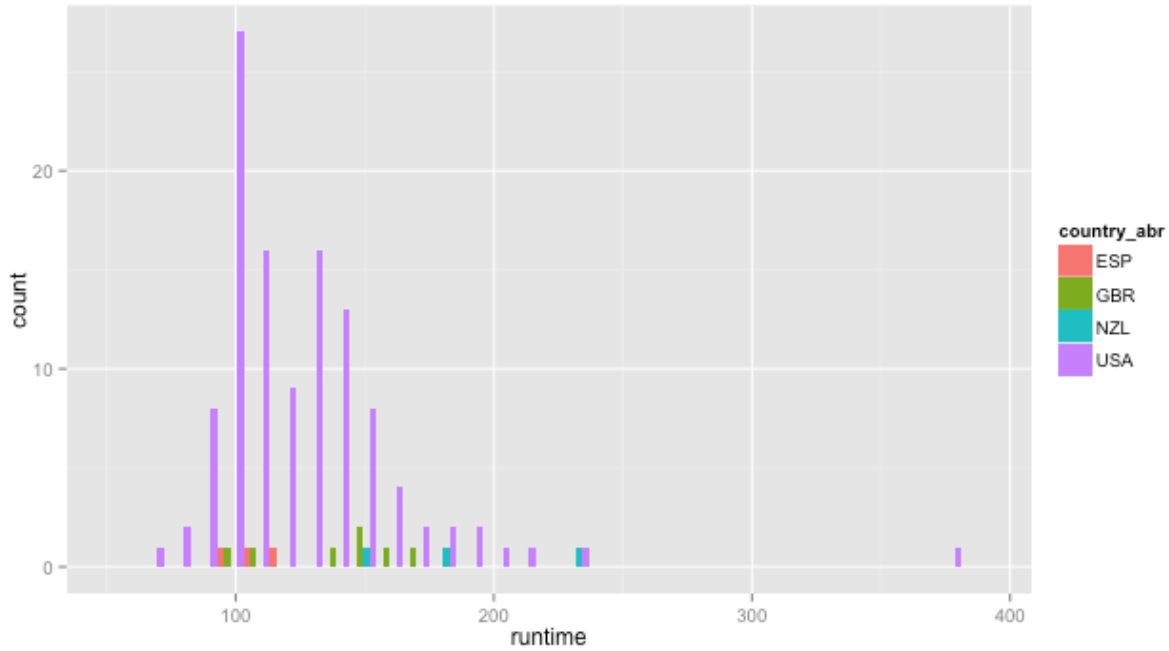
- La base du graphique :

```
p <- ggplot(data = films_reduit, aes(x = runtime, fill = country_abr))
```

Positions : exemples pour un histogramme

- Dodge :

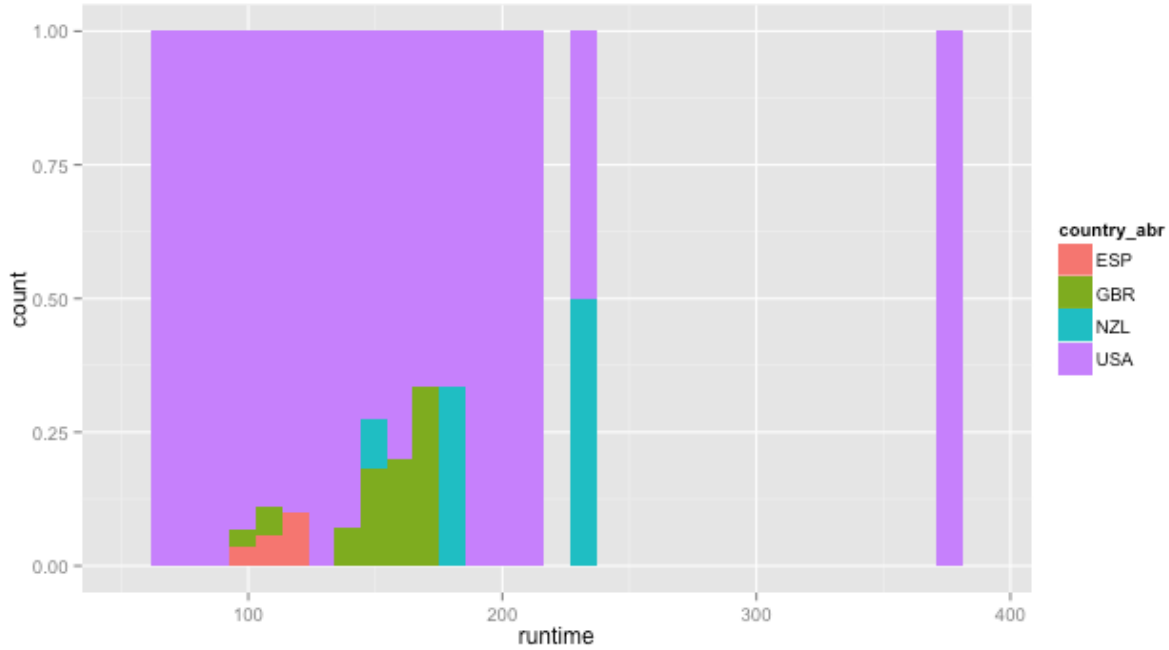
```
p + geom_bar(position = "dodge")
```



Positions : exemples pour un histogramme

- Fill :

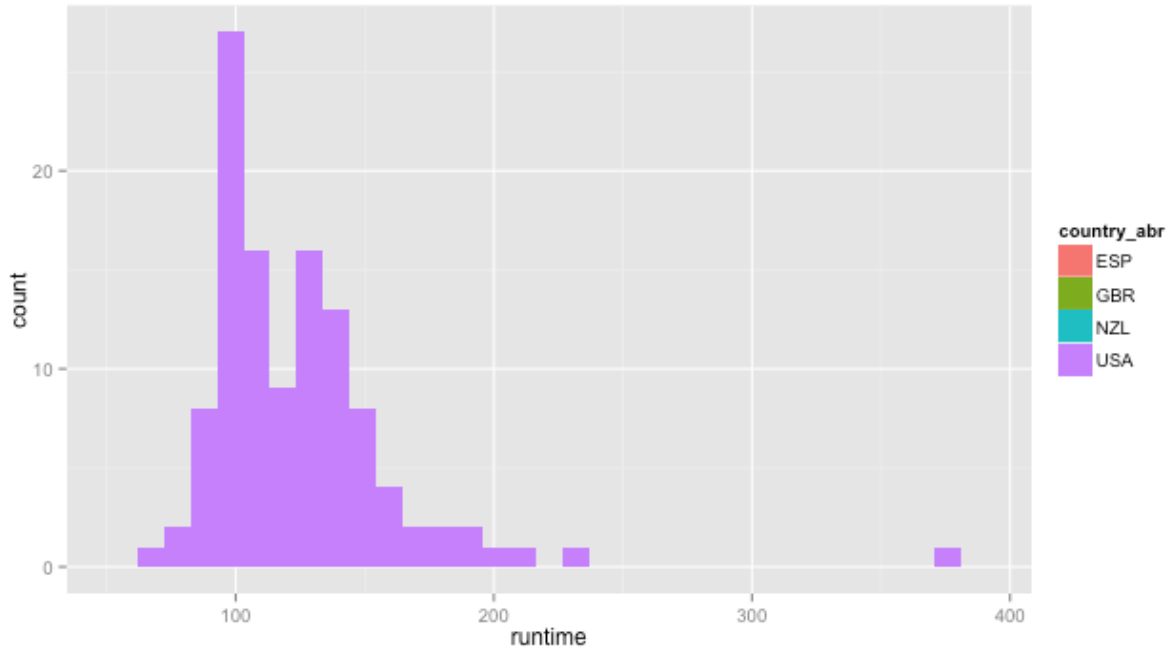
```
p + geom_bar(position = "fill")
```



Positions : exemples pour un histogramme

- Identity (pas pratique avec un barchart, cela peut cacher certaines barres)

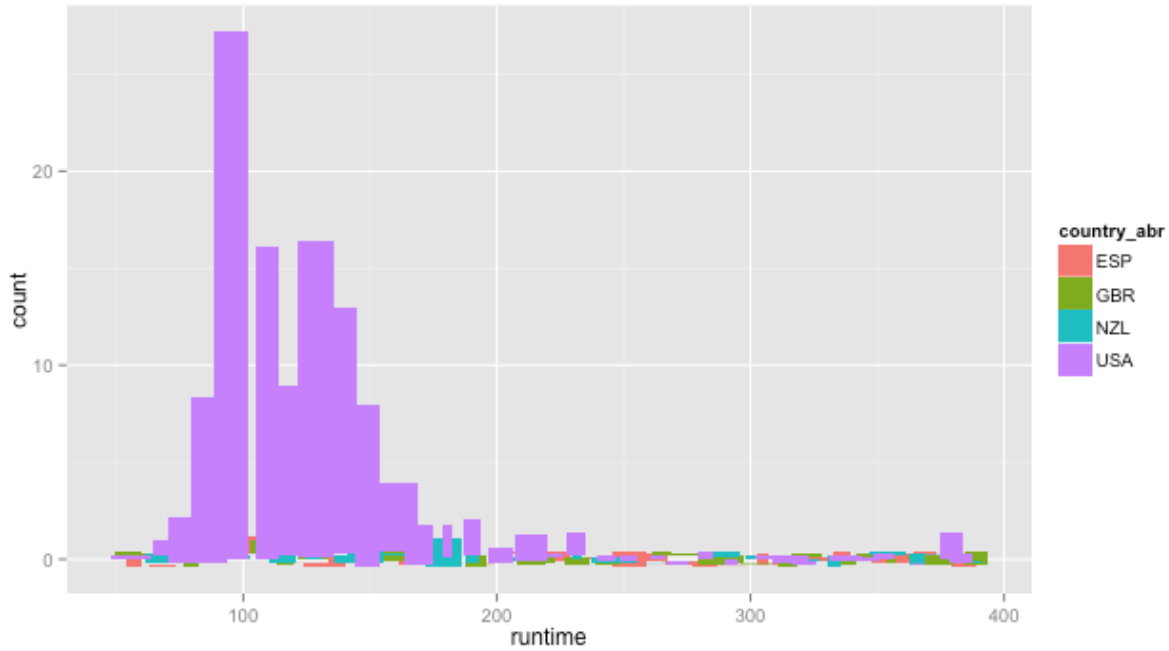
```
p + geom_bar(position = "identity")
```



Positions : exemples pour un histogramme

- Jitter :

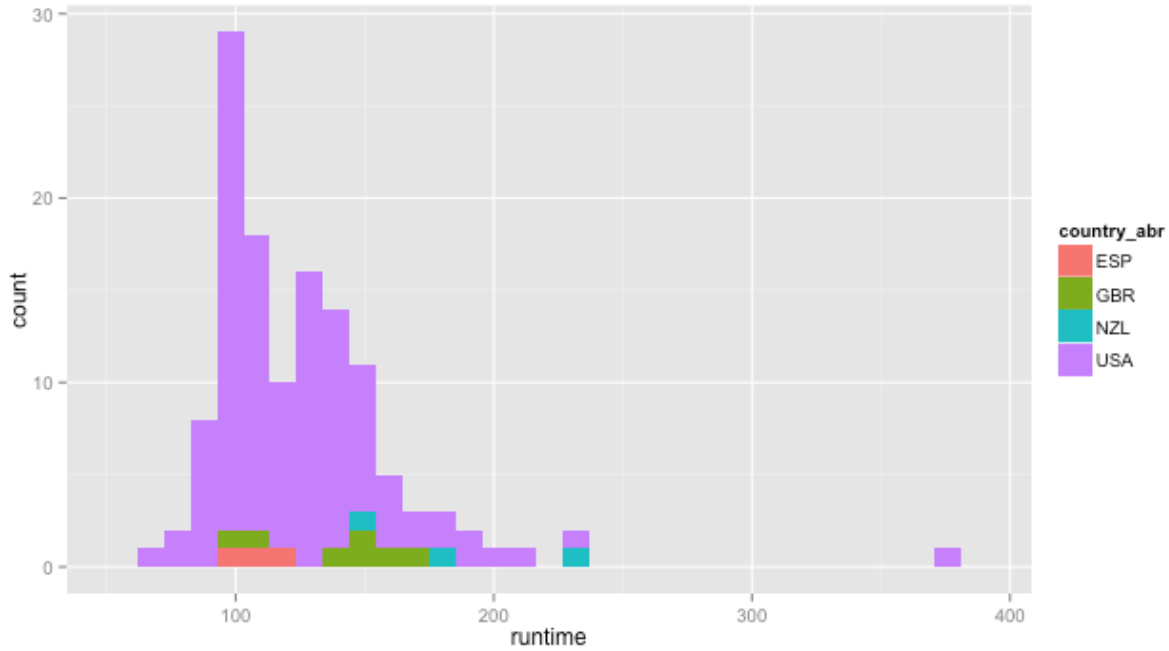
```
p + geom_bar(position = "jitter")
```



Positions : exemples pour un histogramme

- Stack :

```
p + geom_bar(position = "stack")
```



Facettes

- Produire des graphiques de même types pour différentes sous-divisions du `data.frame` ;
- Les aligner sur une grille ;
- Deux fonctions :
  - `facet_grid()` : grille 2d, variables définissant lignes et colonnes,
  - `facet_wrap()` : graphiques pour chaque sous-division, mis bout-à-bout dans une matrice 2x2.
- Deux paramètres principaux :
  - les variables servant au faceting,
  - un logique pour indiquer si les échelles doivent être locales ou globales.

Facettes

- Il nous faut une autre variable indicatrice (sans trop de modalités), pour les exemples.

```
films_reduit$old <- ifelse(films_reduit$year <= 2000, "ancien", "nouveau")
```

Facettes : `facet_grid()`

- Création d'une grille ;
- Colonnes pour les valeurs d'un facteur ;
- Lignes pour les valeurs d'un autre facteur ;
- Le paramètre principal est `facets` :

VALEUR	EFFET
`. ~ .` **(par défaut)**	pas de faceting
`. ~ variable_colonne`	une ligne, autant de colonnes que de valeurs pour `variable_colonne`
`variable_ligne ~ .`	autant de lignes que de valeurs possibles pour `variable_ligne`
`variable_ligne ~ variable_colonne`	autant de lignes que de valeurs possibles pour `variable_ligne`, autant de colonnes que de valeurs possibles pour `variable_colonne`

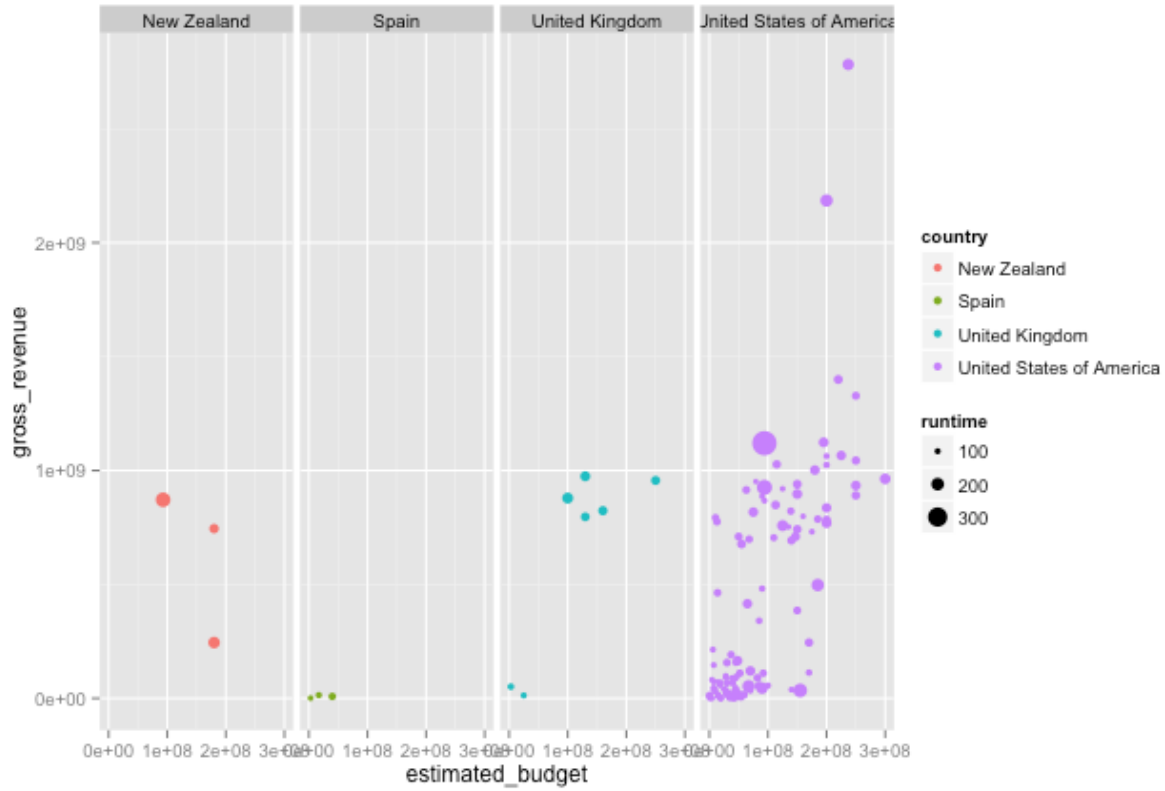
Facettes : `facet_grid()`

```
p <- ggplot(data = films_reduit, aes(x = estimated_budget,
                                     y = gross_revenue,
                                     colour = country,
                                     size = runtime)) +
  geom_point()
```

Facettes : `facet_grid()`

- Avec une colonne pour chaque pays :

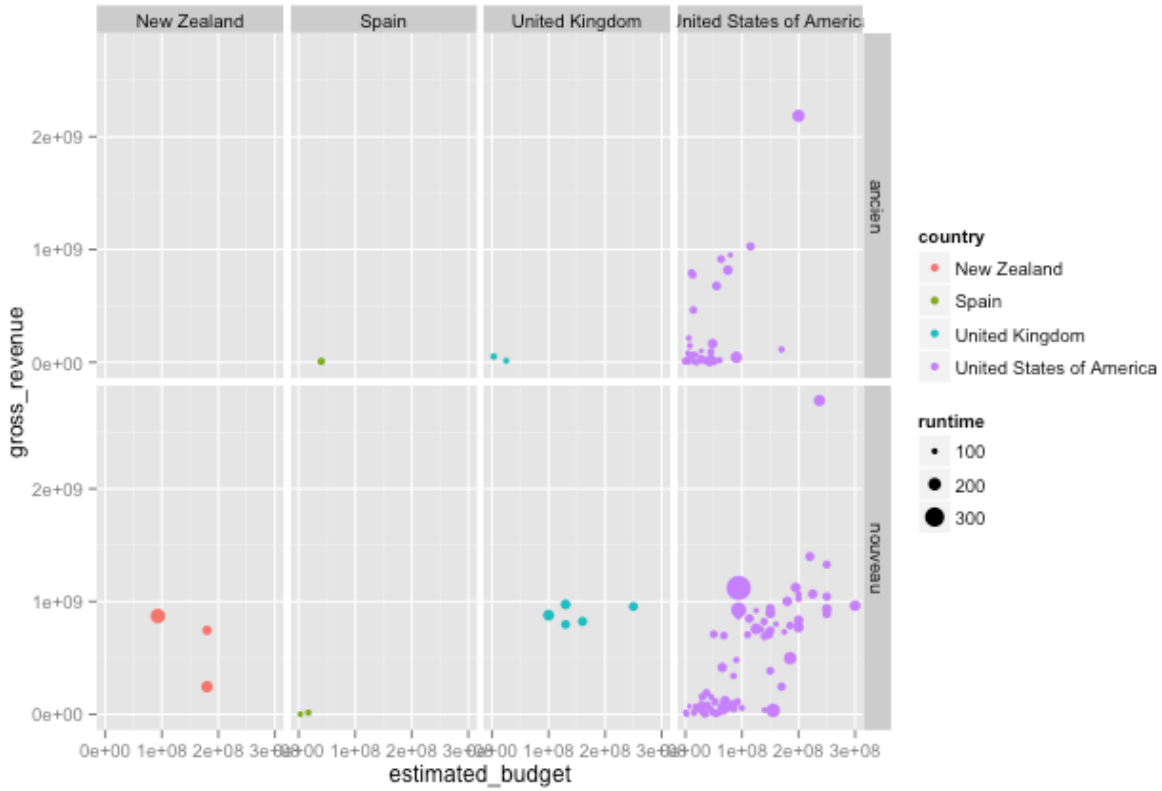
```
p + facet_grid(. ~ country)
```



Facettes : `facet_grid()`

- Avec en ligne la récence du film, et en colonne le pays :

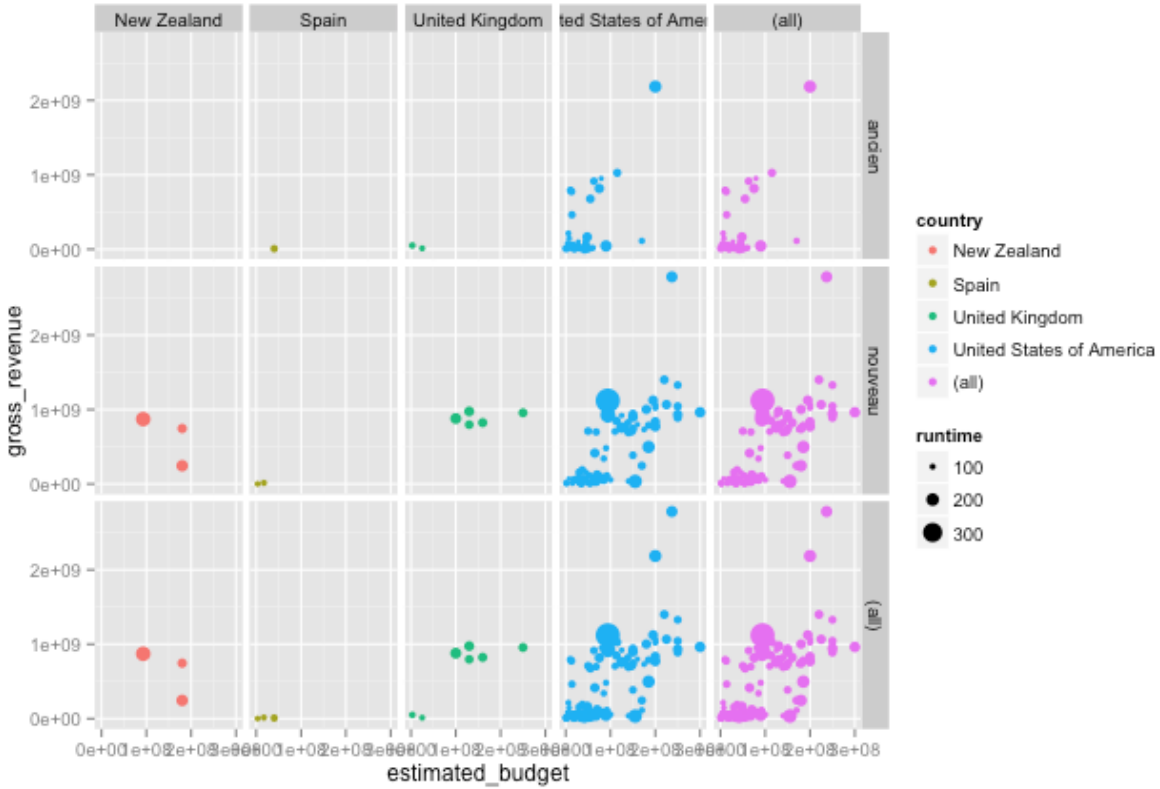
```
p + facet_grid(old ~ country)
```



Facettes : `facet_grid()`

- Si on désire également afficher les situations marginales : `margins = TRUE` ;

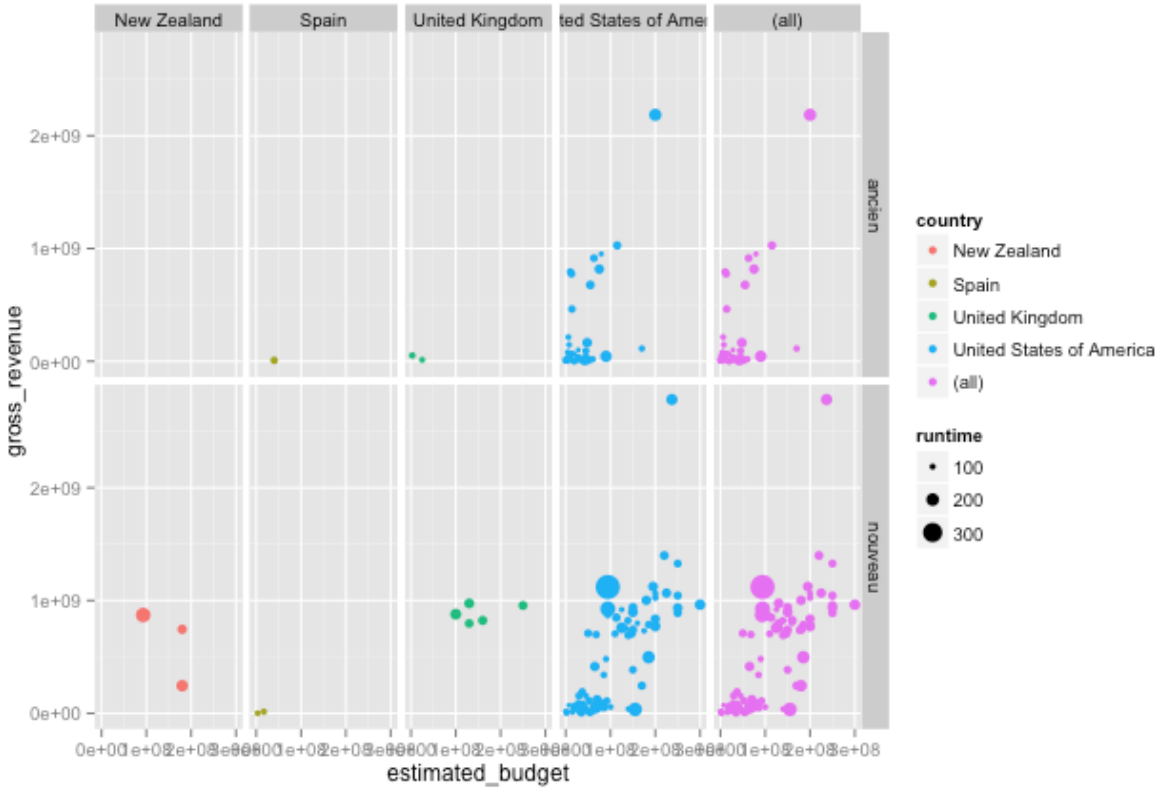
```
p + facet_grid(old ~ country, margins = TRUE)
```



Facettes : `facet_grid()`

- Les situations marginales pour une seule variable : donner son nom à `margins`.

```
p + facet_grid(old ~ country, margins = "country")
```



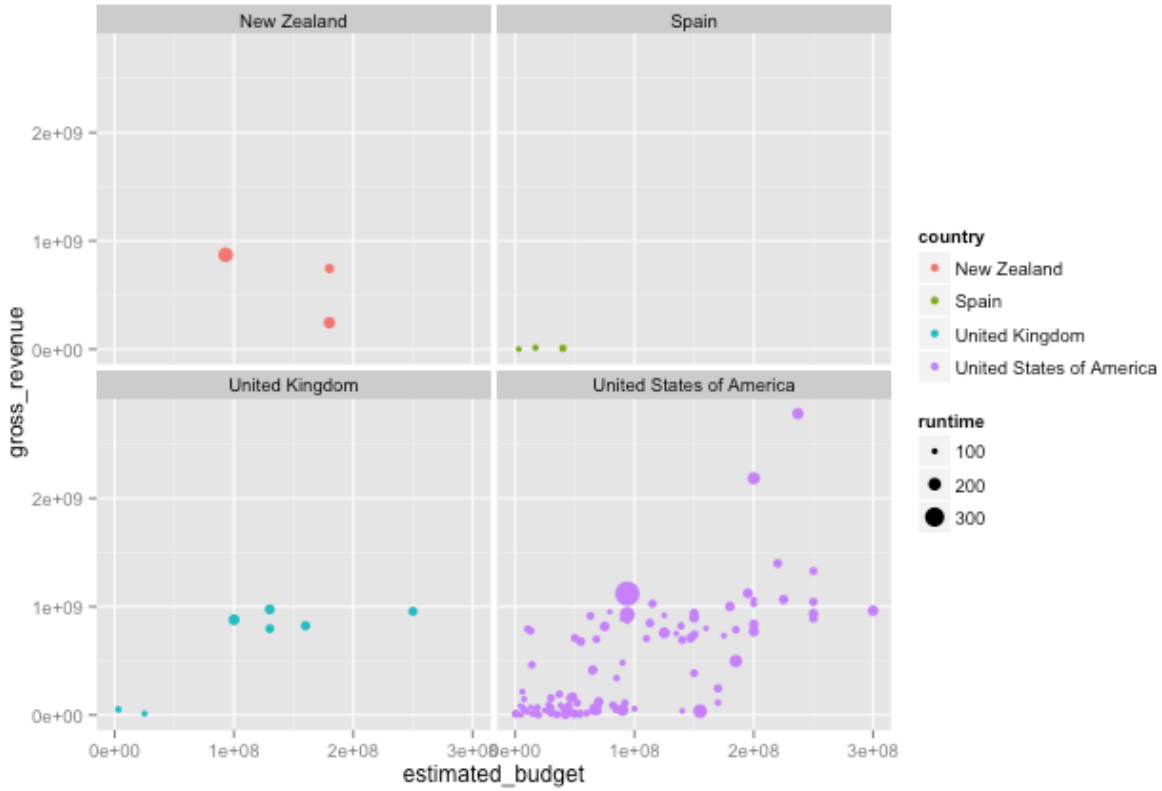
Facettes : `facet_wrap()`

- Même type de graphique appliqué à des sous-échantillons de la base, selon un ou des facteurs ;
- Graphiques ensuite placés sur une grille, les uns à la suite des autres.
- À nouveau, le paramètre principal est `facets` ;
- La syntaxe est la suivante : `~ variable_1 + variable_2 ... + variable_n` ;
- La grille finale sera ce qui se rapproche au mieux d'un carré ;
- Une préférence pour les grilles larges plutôt que longue transparaît.

Facettes : `facet_wrap()`

- Avec une seule variable :

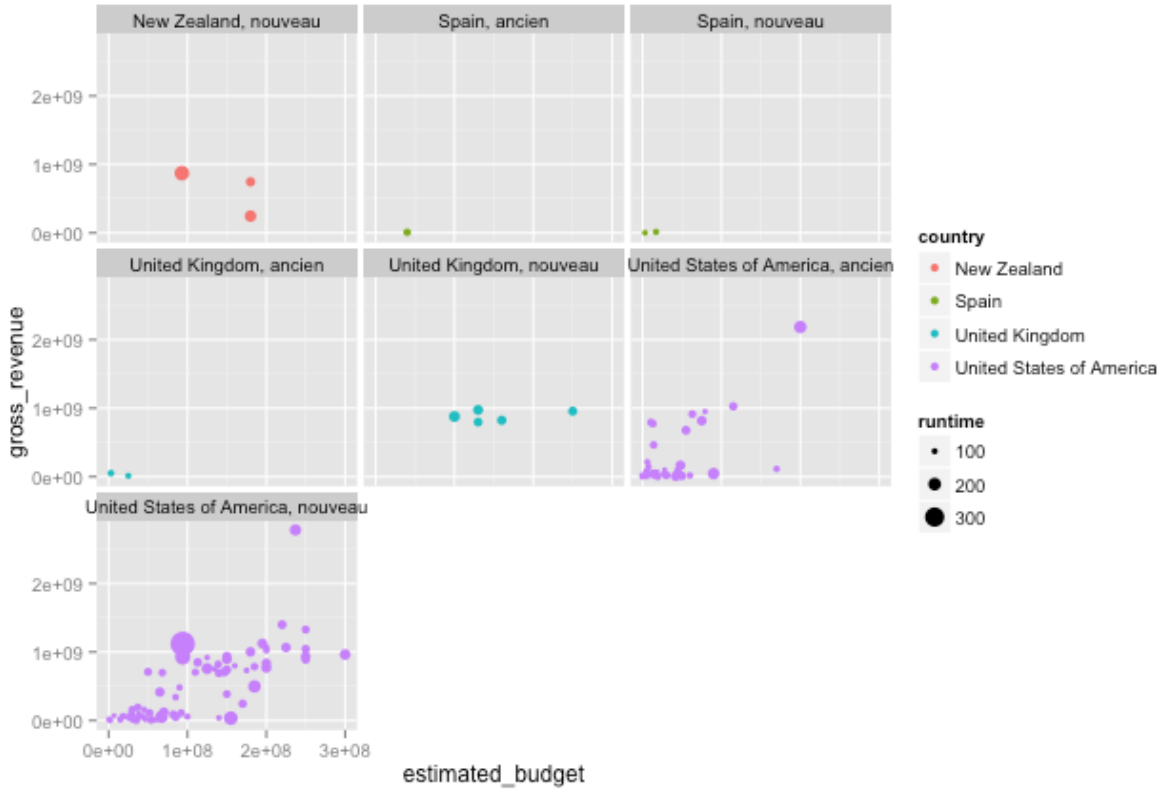
```
p + facet_wrap(facets = ~ country)
```



Facettes : `facet_wrap()`

- Avec une seule variable :

```
p + facet_wrap(facets = ~ country + old)
```



Facettes : échelles

- Les échelles des axes peuvent être identiques pour tous les graphiques de la grille
- Ou bien être propres à chaque graphique ;
- Il faut renseigner le paramètre `scales`.

VALEUR	EFFET
`fixed`	échelles fixes, identiques pour chaque graphique
`free`	échelles libres, pouvant varier en fonction de chaque graphique
`free_x`	seule l'échele pour les `x` peut varier, l'échelle pour les `y` est fixe
`free_y`	seule l'échelle pour les `y` peut varier, l'échelle pour les `x` est fixe

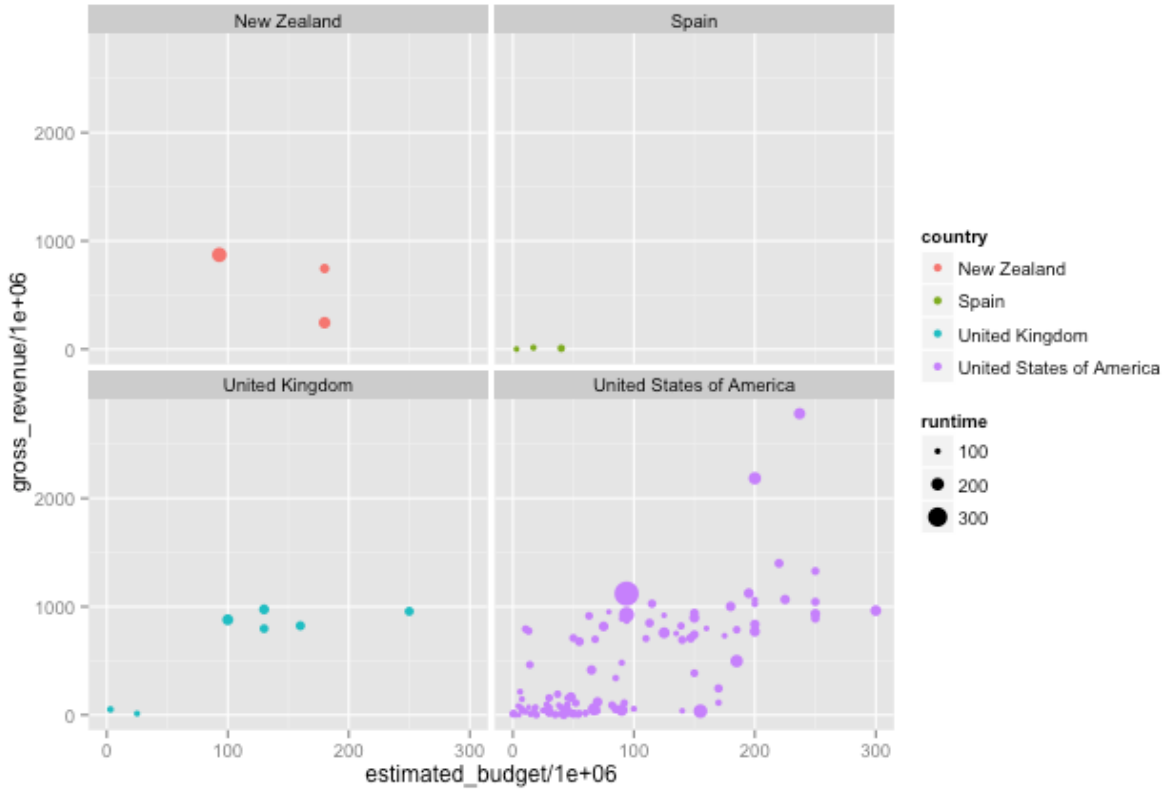
Facettes : échelles

```
p_m <- ggplot(data = films_reduit,
       aes(estimated_budget/1000000,
           gross_revenue/1000000,
           colour = country,
           size = runtime)) +
  geom_point()
```

Facettes : échelles

- Toutes les échelles identiques

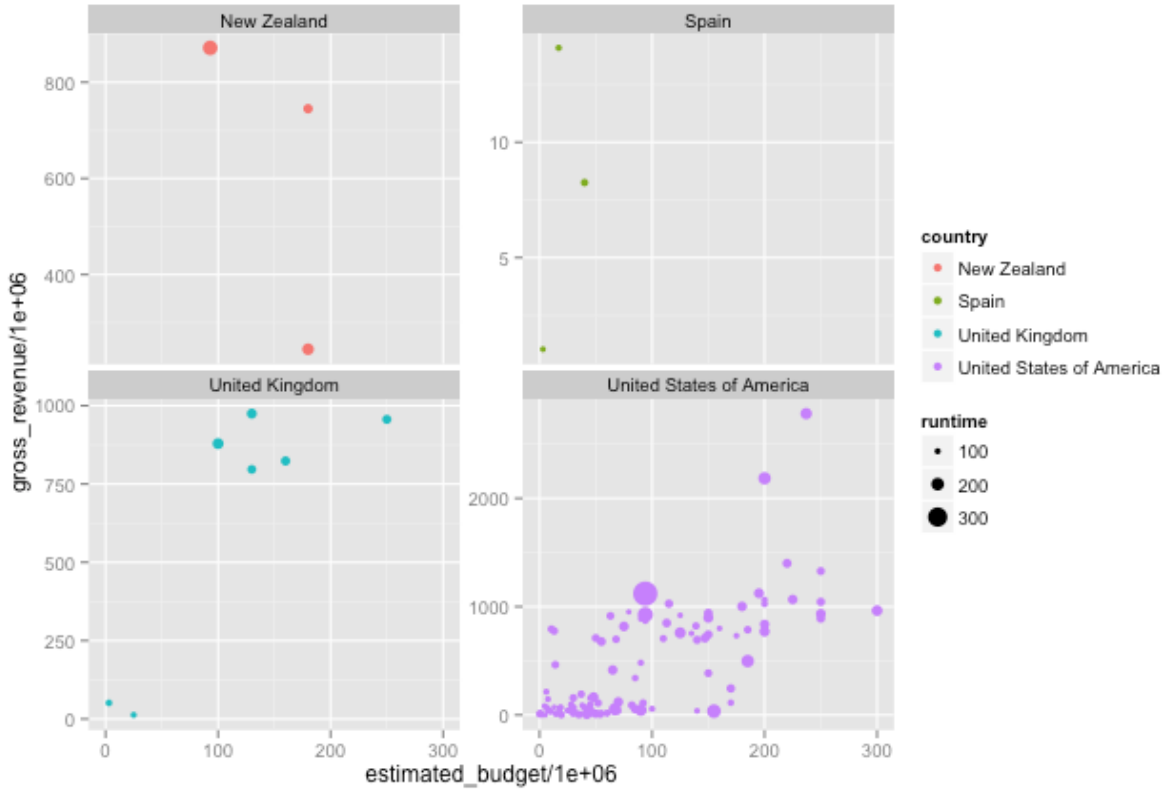
```
p_m + facet_wrap( ~ country, scales = "fixed")
```



Facettes : échelles

- Échelles variant pour chaque graphique de la grille

```
p_m + facet_wrap( ~ country, scales = "free_y")
```



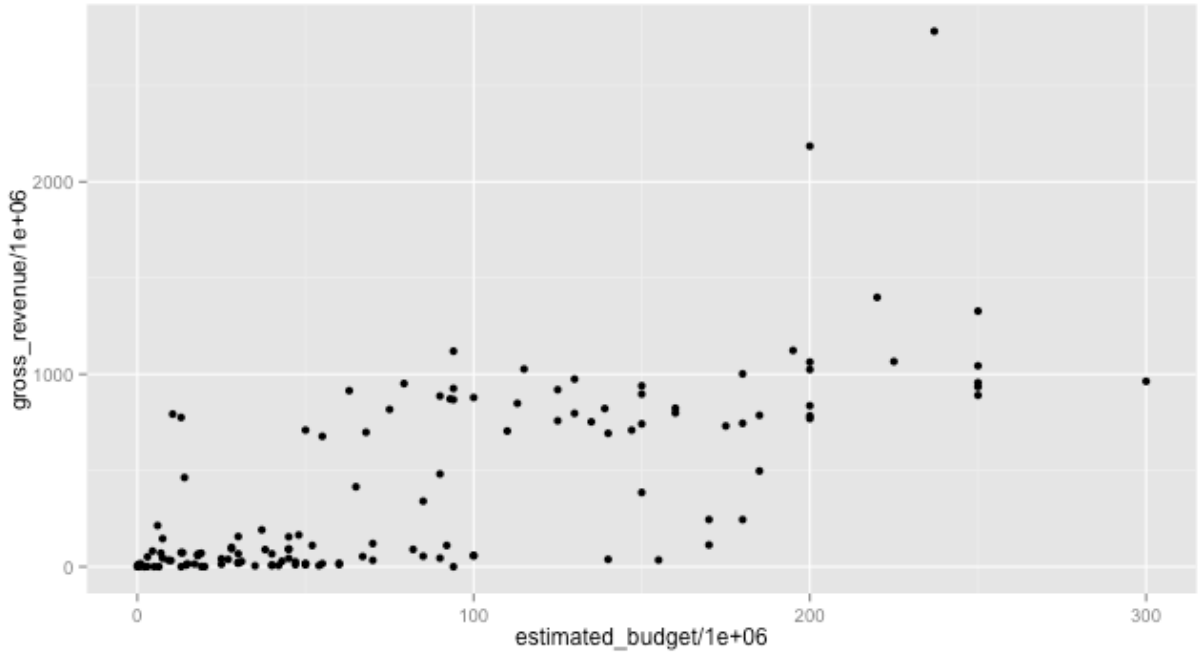
Coordonnées

- Le système de coordonnées par défaut utilisé par `ggplot2` est le système cartésien ;
- Pour en changer, on peut utiliser :
  - `coord_cartesian()` : coordonnées cartésiennes,
  - `coord_fixed()` : coordonnées cartésiennes avec la même échelle pour les deux axes,
  - `coord_flip()` : coordonnées cartésiennes avec les axes renversés,
  - `coord_map()` : projections pour les cartes,
  - `coord_polar()` : coordonnées polaires,
  - `coord_trans()` : coordonnées cartésiennes transformées.

Coordonnées

- Graphique de référence :

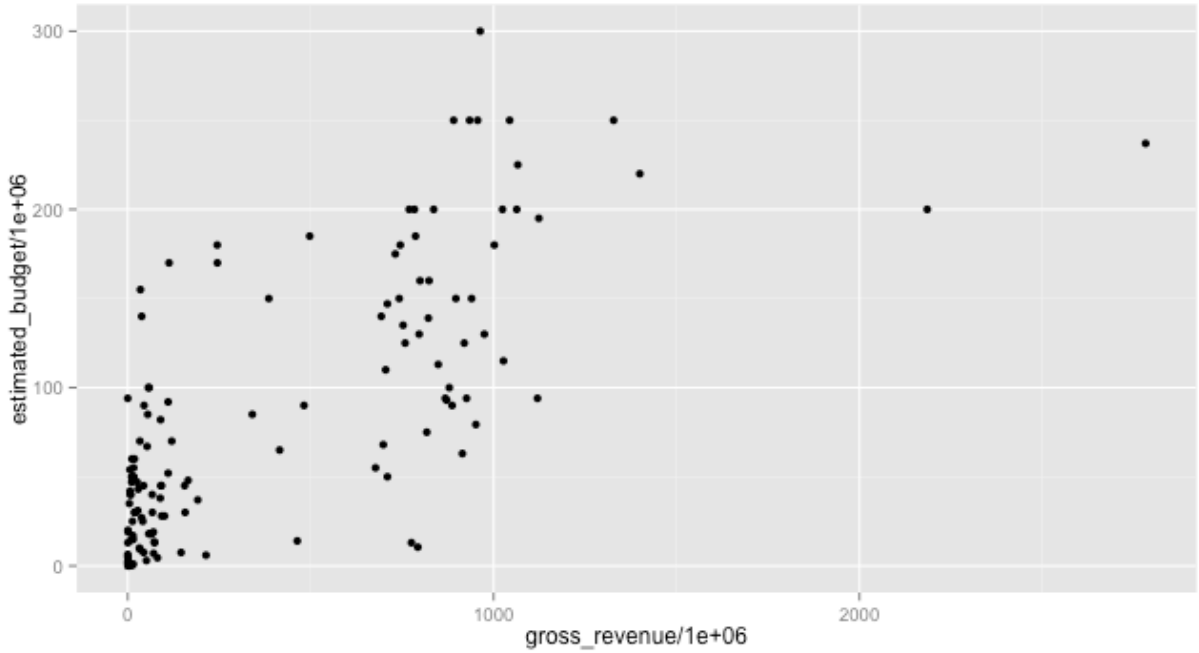
```
(p <- ggplot(data = films,
             aes(x = estimated_budget/1e6, y = gross_revenue/1e6)) +
  geom_point())
```



Coordonnées

- En renversant les axes :

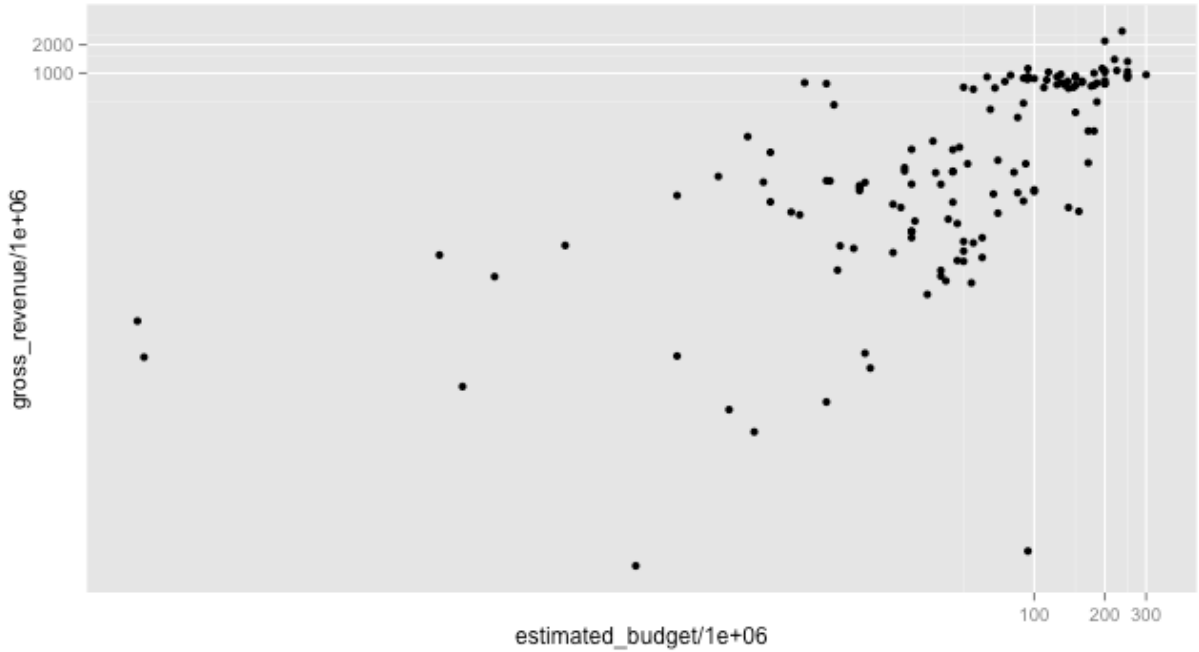
```
p + coord_flip()
```



Coordonnées

- En appliquant une transformation :

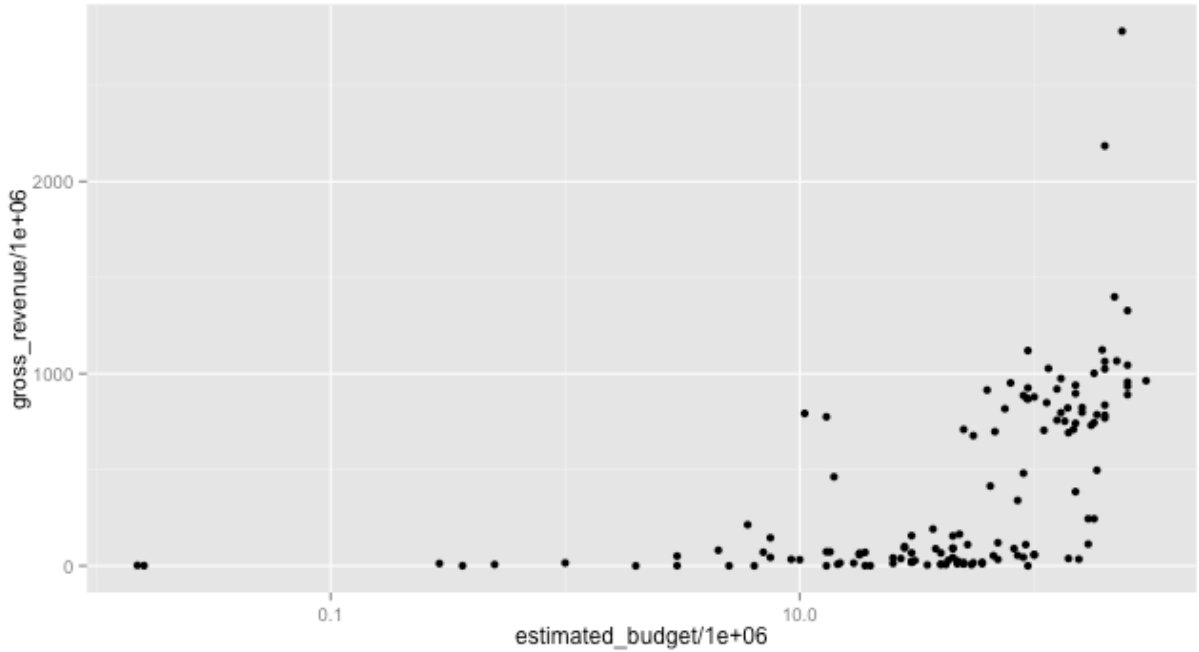
```
p + coord_trans(x = "log10", y = "log10")
```



Coordonnées

- Quand on veut appliquer la fonction log aux coordonnées, il est préférable d'utiliser la fonction `scale_x_log10()` ;
- En effet, la transformation a lieu **avant** de calculer les positions des grilles.

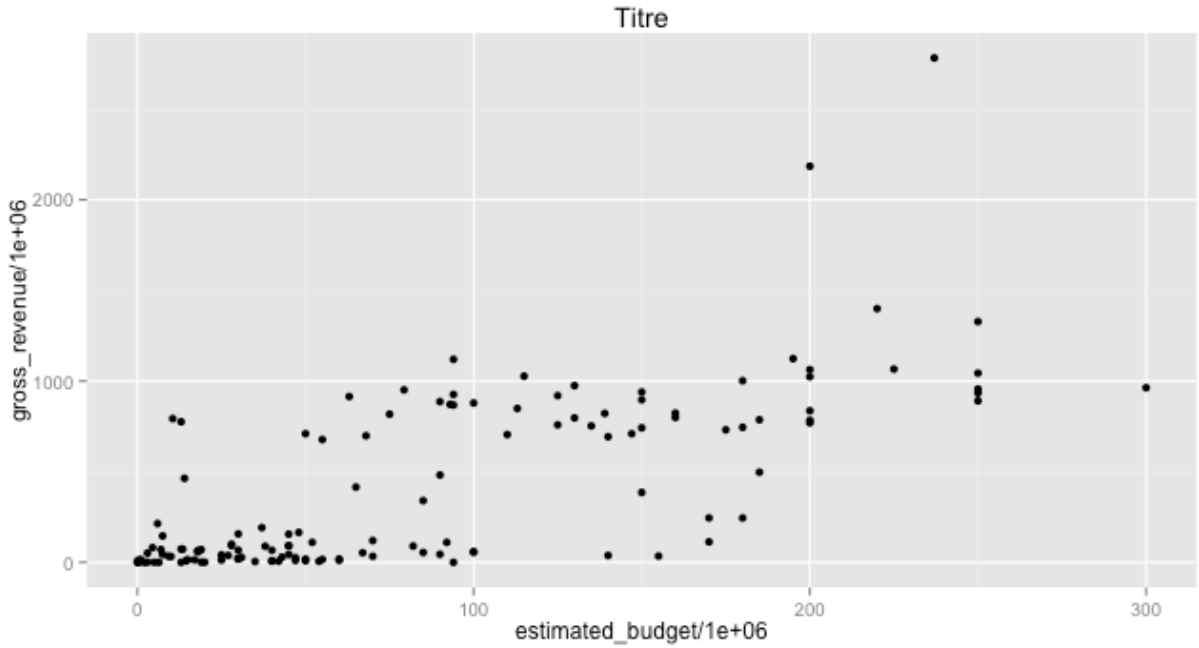
```
p + scale_x_log10()
```



Titre

- Le titre d'un graphique s'ajoute à l'aide de la fonction `ggtitle()` :

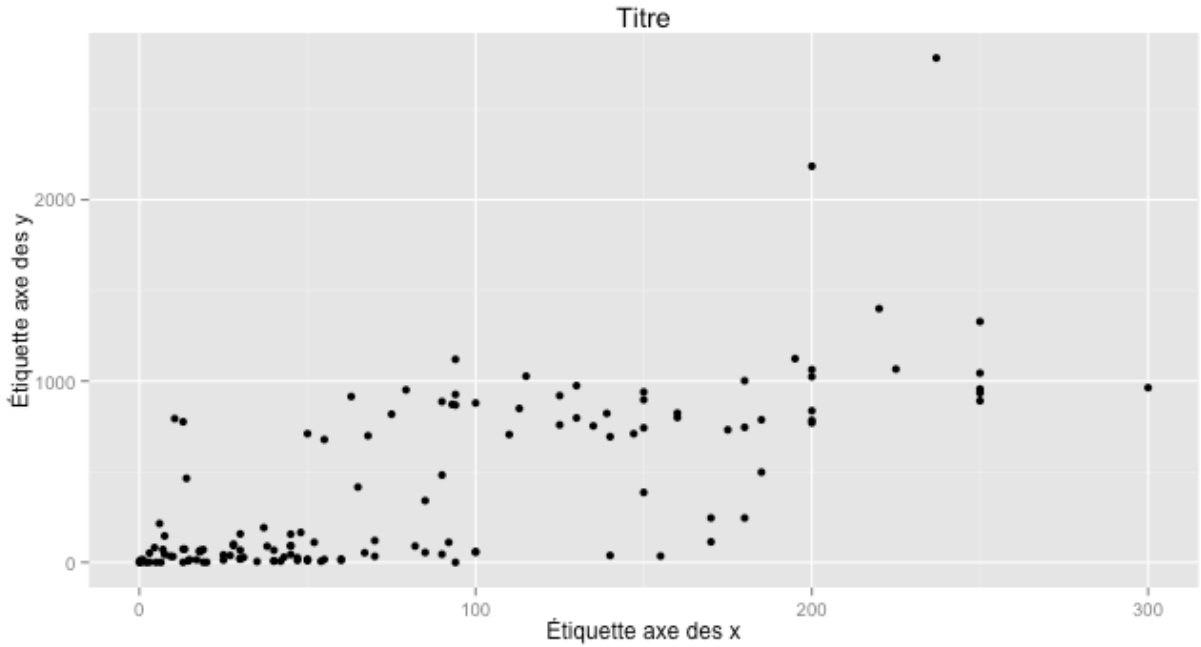
```
ggplot(data = films,
       aes(x = estimated_budget/1e6, y = gross_revenue/1e6)) +
  geom_point() + ggtitle("Titre")
```



Labels des axes

- Les étiquettes des axes se modifient à l'aide `xlab()` et `ylab()` :

```
ggplot(data = films,
       aes(x = estimated_budget/1e6, y = gross_revenue/1e6)) +
  geom_point() + ggtitle("Titre") +
  xlab("Étiquette axe des x") + ylab("Étiquette axe des y")
```



Axes

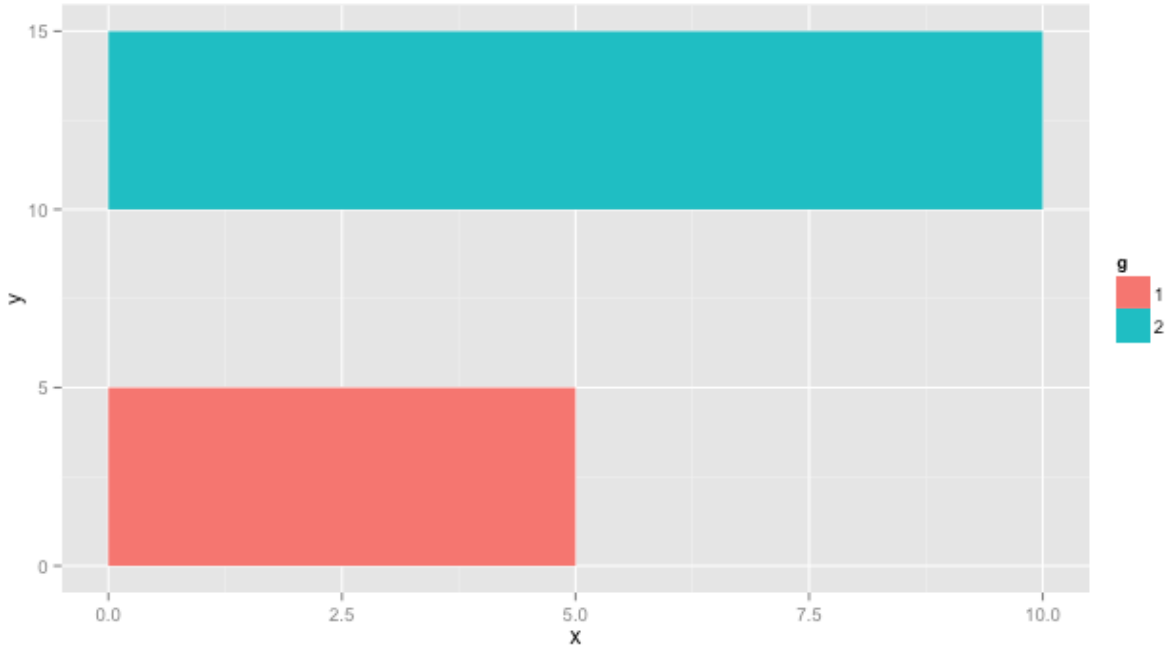
- Pour définir manuellement les limites des axes, on peut utiliser les fonctions `xlim()` et `ylim()` ;
- Mais les valeurs à afficher n'appartenant pas aux intervalles sont jetées ;
- Si on ne désire pas jeter les valeurs (dans le cas d'un "zoom" par exemple) :
  - utiliser la fonction `coord_cartesian()`,
  - renseigner les intervalles de définition aux paramètres `xlim` et `ylim`.

```
df <- data.frame(x = c(0, 0, 5, 5, 0, 0, 10, 10),
                 y = c(0,5, 5, 0, 10, 15, 15, 10),
                 g = factor(rep(1:2, each = 4)))
```

Axes

- Le graphique original :

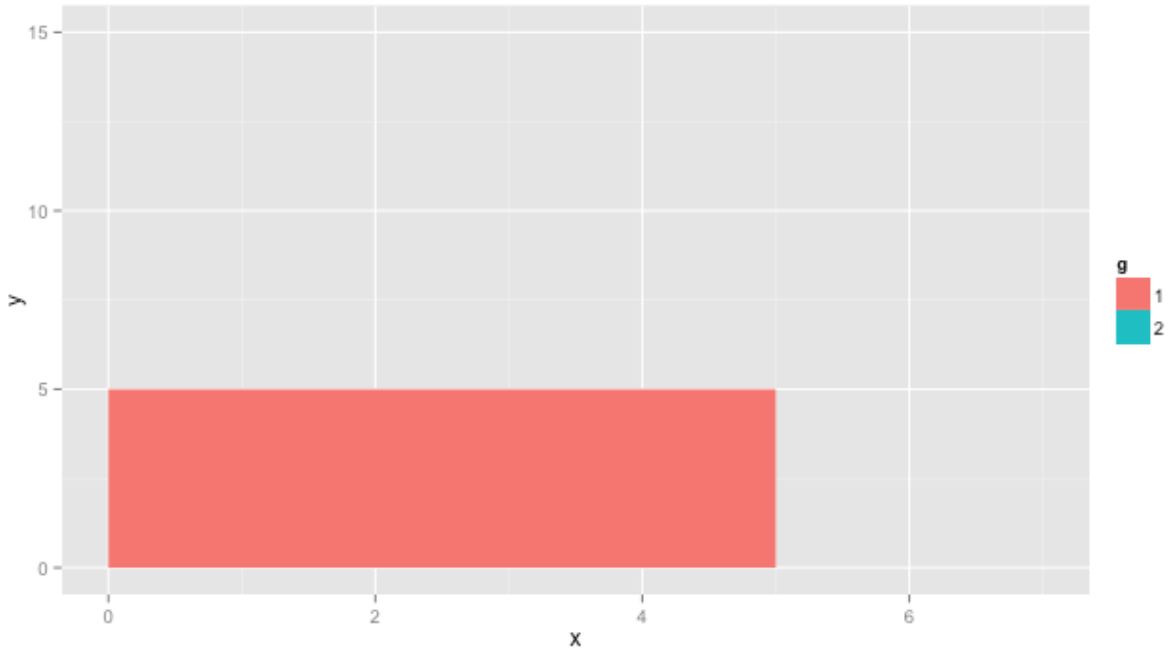
```
(p_2 <- ggplot(data = df, aes(x = x, y = y, group = g, fill = g)) +
  geom_polygon())
```



Axes

- En jouant avec la limite des x avec la fonction xlim()

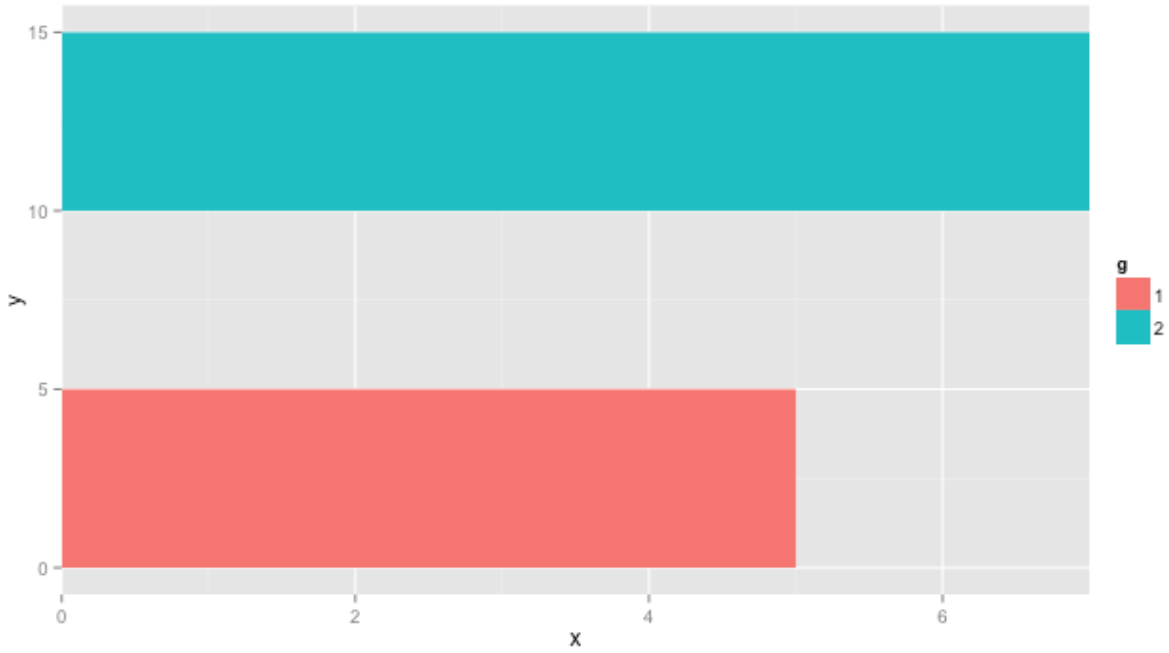
```
p_2 + xlim(0, 7)
```



Axes

- Avec la fonction coord_cartesian()

```
p_2 + coord_cartesian(xlim = c(0,7))
```



Exercices

À partir du graphique suivant :

```
p <- ggplot() + geom_point(data = diamonds[sample(1:nrow(diamonds), 1000),],
                           aes(x = carat, y = price, colour = cut))
```

1. Ajouter un titre cohérent ;
2. Changer les étiquettes des axes pour y mettre des étiquettes en français.

Aspect général

- Le changement des autres paramètres est moins évident ;
- Il faut passer par la fonction `theme()` ;
- http://docs.ggplot2.org/current/theme.html ;
- Quatre types de fonctions :
  - `element_text()` : pour toutes les étiquettes, ce qui est au format texte,
  - `element_line()` : pour toutes les lignes tracées,
  - `element_rect()` : pour les backgrounds et les cadres,
  - `element_blank()` permet de ne rien dessiner.

Aspect général : `element_text()`

PARAMÈTRE	VALEUR
`family`	la famille de la fonte
`face`	graisse ("plain", "italic", "bold", "bold.italic")
`colour`	couleur
`size`	taille en pts
`hjust`	justification horizontale, dans [0, 1]
`vjust`	justification verticale, dans [0, 1]
`angle`	angle, dans [0, 360]
`lineheight`	hauteur de ligne (pour l'espacement entre les lignes)

Aspect général : `element_line()`

PARAMÈTRE	VALEUR
`colour`	la couleur de ligne
`size`	la taille
`linetype`	le type de ligne ("blank", "solid", "dashed", "dotted", "dotdash",
"longdash", "twodash")	
`lineend`	le type de fin de ligne ("round", "butt" ou "square")

Aspect général : `element_rect()`

PARAMÈTRE	VALEUR
`fill`	la couleur de remplissage
`colour`	la couleur de la bordure
`size`	la taille de la bordure
`linetype`	le type de ligne ("blank", "solid", "dashed", "dotted", "dotdash", "longdash", "twodash)

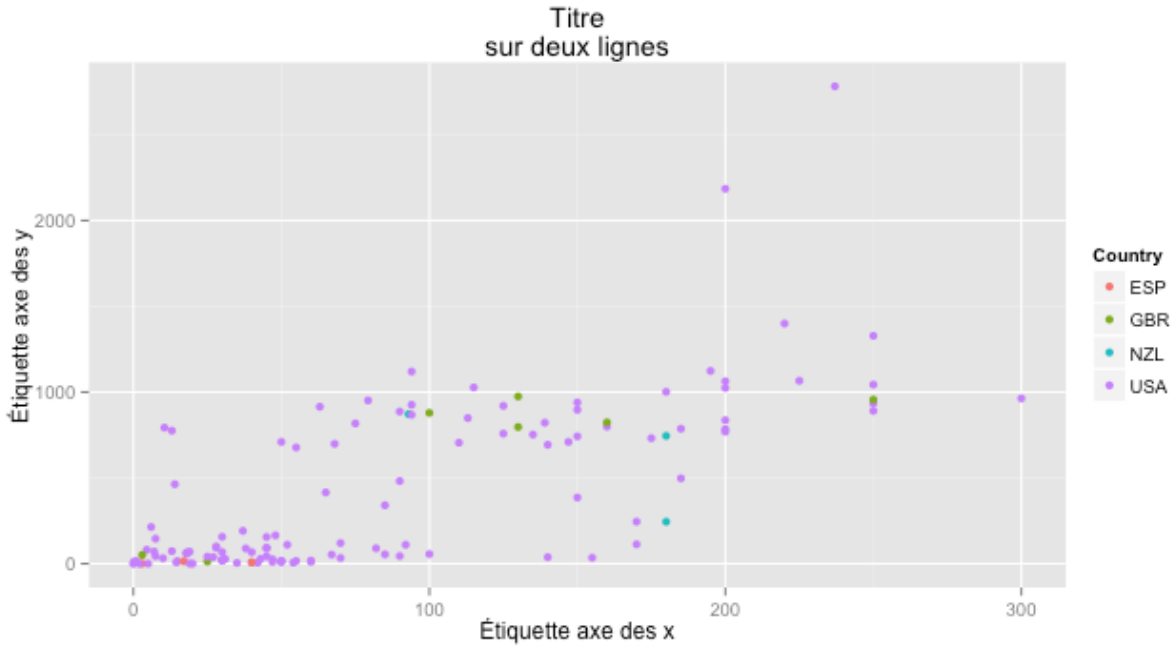
Aspect général

```
library(grid)
p <- ggplot(data = films_reduit,
            aes(x = estimated_budget/1e6, y = gross_revenue/1e6, colour = country_abr)) +
  # Tracer des points
  geom_point() +
  # Ajout d'un titre
  ggtitle("Titre\nsur deux lignes") +
  # Changement des étiquettes pour les axes
  xlab("Étiquette axe des x") +
  ylab("Étiquette axe des y") +
  # Changement du titre de la légende
  scale_colour_discrete(name = "Country")
```

Aspect général

- Le graphique utilisant le thème par défaut :

```
p
```



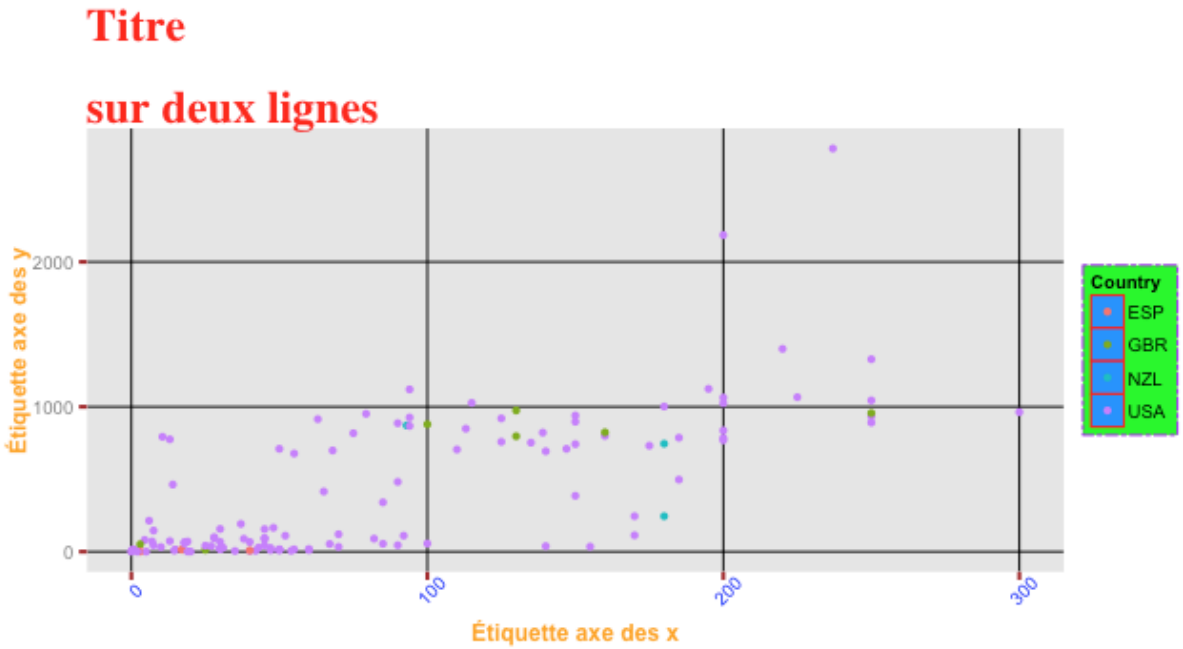
Aspect général

- En changeant certains éléments

```
p_2 <- p + theme(plot.title = element_text(family = "Times",
                                           face = "bold", colour = "red",
                                           size = rel(2), hjust = 0,
                                           lineheight = 1.5),
                 axis.title = element_text(face = "bold", colour = "orange"),
                 axis.text.x = element_text(colour = "blue", angle = 45),
                 axis.ticks = element_line(colour = "brown", size = rel(2)),
                 legend.key = element_rect(fill = "dodger blue", colour = "red"),
                 legend.background = element_rect(fill = "green",
                                                  colour = "purple",
                                                  linetype = "twodash"),
                 panel.grid.minor = element_blank(),
                 panel.grid.major = element_line(colour = "black") )
```

Aspect général

```
p_2
```

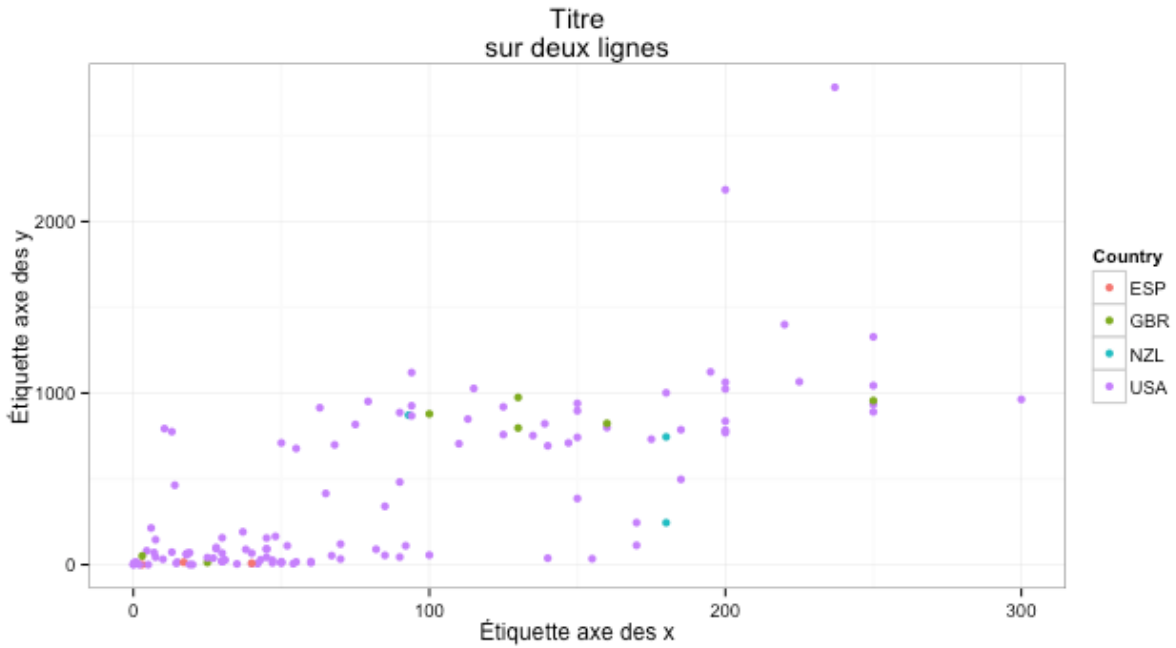


Aspect général

- Pour éviter ce genre d'horreurs : des thèmes prédéfinis existent :
  - `theme_grey()` (thème par défaut),
  - `theme_bw()`,
  - `theme_linedraw()`,
  - `theme_light()`,
  - `theme_minimal()`,
  - `theme_classic()`.

Aspect général

```
p + theme_bw()
```



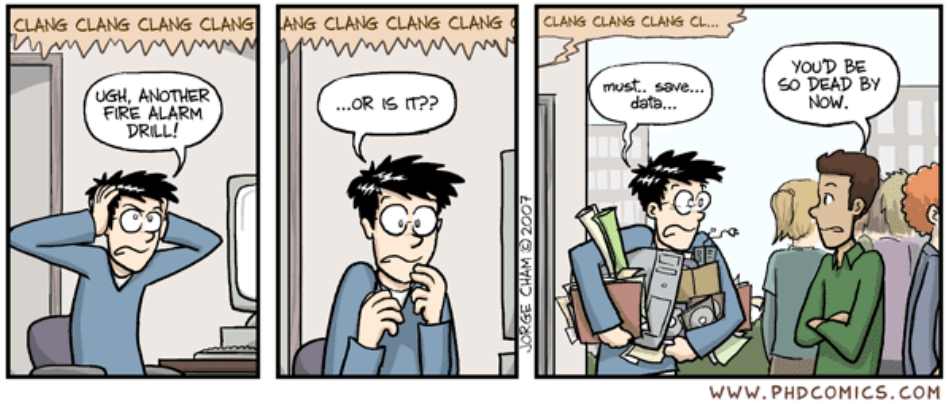
Exercices

À partir du graphique suivant :

```
p <- ggplot() + geom_point(data = diamonds[sample(1:nrow(diamonds), 1000),],
                           aes(x = carat, y = price, colour = cut))
```

1. Mettre le fond de la zone de graphique en blanc ;
2. Changer la taille des étiquettes des axes ;
3. Changer la couleur des lignes secondaires de la grille de la zone de graphique.

Enregistrement des graphiques



Source : PhD Comics : http://www.phdcomics.com/comics/archive.php?comicid=814

Enregistrement des graphiques

- Pour enregistrer un graphique réalisé avec `ggplot()` : `ggsave()`.

PARAMÈTRE	VALEUR
`filename`	nom du fichier, ou chemin et nom du fichier
`plot`	graphique à sauvegarder (par défaut, le dernier graphique, en faisant appel à la fonction last_plot())
`device`	dispositif à utiliser (automatiquement extrait de l'extension du fichier indiqué au paramètre filename)
`path`	chemin vers le fichier
`scale`	facteur d'échelle
`width`	largeur (par défaut, celle de la fenêtre de graphique actuelle)
`height`	hauteur (par défaut, celle de la fenêtre de graphique actuelle)
`units`	unité pour la largeur et la longueur ("in", "cm" ou "mm")
`dpi`	nombre de points par pouce, uniquement pour les images matricielles
`limitsize`	quand TRUE (la valeur par défaut), l'image sauvegardée ne dépassera pas les 50 × 50 in

Enregistrement des graphiques

- `ggsave()` reconnaît automatiquement les extensions suivantes :
  - eps/ps,
  - tex,
  - pdf,
  - jpeg,
  - tiff,
  - png,
  - bmp,
  - svg,
  - wmf (uniquement pour Windows).

Enregistrement des graphiques

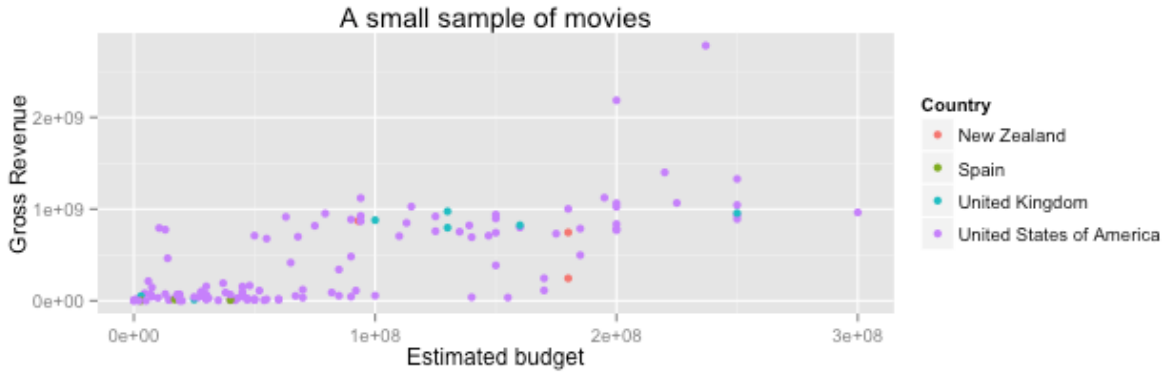
- Exemple :

```
p <- ggplot(data = films_reduit,
            aes(x = estimated_budget, y = gross_revenue,
                colour = country)) +
  geom_point() +
  xlab("Estimated budget") + ylab("Gross Revenue") +
  scale_colour_discrete(name = "Country") +
  ggtitle("A small sample of movies")
```

Enregistrement des graphiques

- En registrement du dernier graphique affiché, aux dimensions de la fenêtre de graphiques

```
p
```




```
ggsave("estim_bud.pdf")
```

Enregistrement des graphiques

- En précisant plus de paramètres :

```
ggsave(p, file = "estim_bud.pdf", width = 15, height = 8, unit = "cm", scale = 2)
```

Réaliser des cartes avec `ggplot2`

Source : Great Maps with ggplot2, http://spatial.ly

Récupérer des cartes toutes faites

- Pour réaliser des cartes en `R`, il est nécessaire d'avoir les données à disposition ;
- On peut obtenir ces données de deux manières :
  - en dehors de `R`,
  - en chargeant un package.

Package `rworldmap`

- Le package `rworldmap` propose une carte du monde ;
- On accède aux données avec la fonction `getMap()` ;
- Puis il faut effectuer quelques transformations pour donner un format lisible par `ggplot()` :
  - on utilise `fortify()` pour transformer le `SpatialPolygonsDataFrame` en `data.frame`.

```
library(ggplot2)
library(rworldmap)
```

Package rworldmap

```
# Carte du monde
worldMap <- getMap()
# Format lisible pour ggplot()
world_df <- fortify(worldMap)
```

```
## Regions defined for each Polygons
```

```
head(world_df)
```

```
##       long      lat order  hole piece         group          id
## 1 61.21082 35.65007     1 FALSE     1 Afghanistan.1 Afghanistan
## 2 62.23065 35.27066     2 FALSE     1 Afghanistan.1 Afghanistan
## 3 62.98466 35.40404     3 FALSE     1 Afghanistan.1 Afghanistan
## 4 63.19354 35.85717     4 FALSE     1 Afghanistan.1 Afghanistan
## 5 63.98290 36.00796     5 FALSE     1 Afghanistan.1 Afghanistan
## 6 64.54648 36.31207     6 FALSE     1 Afghanistan.1 Afghanistan
```

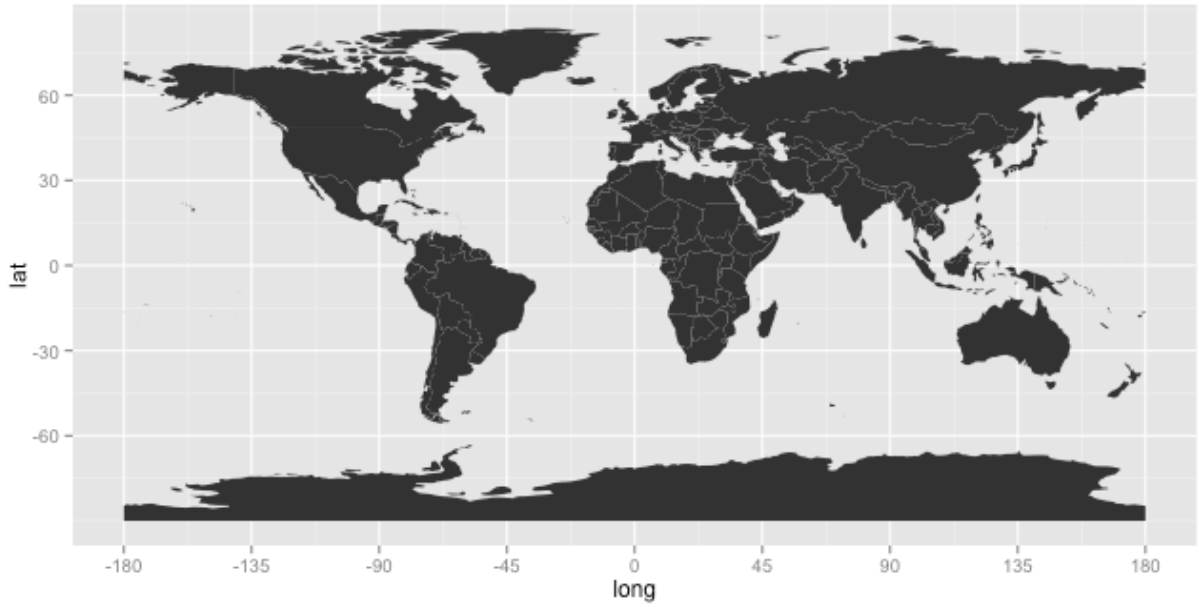
Package `rworldmap`

- Il reste à préciser le mapping ;
- Il ne faut pas oublier d'ajouter le paramètre `group`, pour bien définir chaque polygone ;
- On peut changer les positions des lignes d'arrière plan ;
- Enfin, `coord_equal()` permet de s'assurer de respecter un ratio de 1 entre les unités de `x` et `y`.

```
worldmap <- ggplot() +
  geom_polygon(data = world_df, aes(x = long, y = lat, group = group)) +
  scale_y_continuous(breaks = (-2:2) * 30) +
  scale_x_continuous(breaks = (-4:4) * 45) +
  coord_equal()
```

Package `rworldmap`

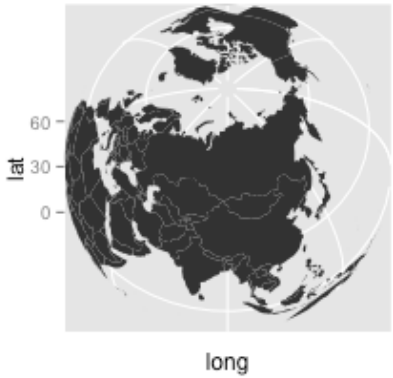
```
worldmap
```



Package `rworldmap`

- La fonction `cord_map()` permet de changer le système de coordonnées ;

```
(worldmap <- ggplot() +
  geom_polygon(data = world_df, aes(x = long, y = lat, group = group)) +
  scale_y_continuous(breaks = (-2:2) * 30) +
  scale_x_continuous(breaks = (-4:4) * 45) +
  coord_map("ortho", orientation=c(61, 90, 0)))
```



- Plusieurs exemples sont visibles sur le biller Moving the North Pole to the Equator, Freakonometrics.

Package `maps`

- Avec `rworldmap`, on obtient des frontières de pays ;
- Pour avoir des découpages plus fins, pour certains pays, on peut regarder dans le package `maps` ;
- La fonction `map_data()` (package `ggplot2`) s'appuie sur la fonction `map` du package du même nom ;
- Elle retourne un `data.frame`, prêt à l'emploi par `ggplot()` !

Package `maps`

- Il faut fournir à `map_data()` le nom d'un pays (qui correspond au nom d'une carte) :

NOM	DESCRIPTION
`county`	carte des counties américains
`france`	carte de la France
`italy`	carte de l'Italie
`nz`	carte de la Nouvelle-Zélande
`state`	carte des États-Unis avec chaque état
`usa`	carte des États-Unis avec uniquement les frontières
`world`	carte du monde
`world2`	carte du monde centrée sur le Pacifique

Package `maps`

- Si on souhaite une région spécifique pour le pays, il faut renseigner le paramètre `region`.

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

# Le nom des régions
head(unique(map_fr$region))
```

```
## [1] "Nord"           "Pas-de-Calais"  "Somme"          "Ardennes"
## [5] "Seine-Maritime" "Aisne"
```

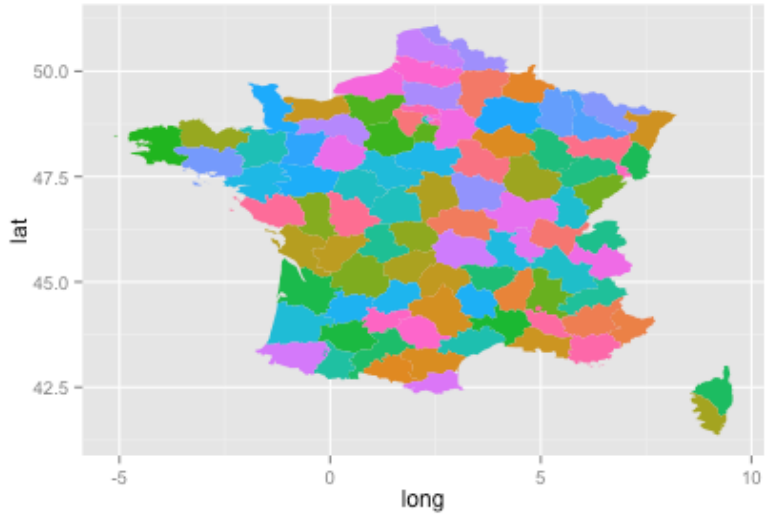
```
head(map_fr, 3)
```

```
##       long      lat group order region subregion
## 1 2.557093 51.09752     1     1   Nord      <NA>
## 2 2.579995 51.00298     1     2   Nord      <NA>
## 3 2.609101 50.98545     1     3   Nord      <NA>
```

Package `maps`

- Carte de la France :

```
(p_map_fr <- ggplot(data = map_fr,
                    aes(x = long, y = lat, group = group, fill = region)) +
  geom_polygon() + coord_equal() + scale_fill_discrete(guide = "none"))
```



Package `maps`

- Carte de la Bretagne :

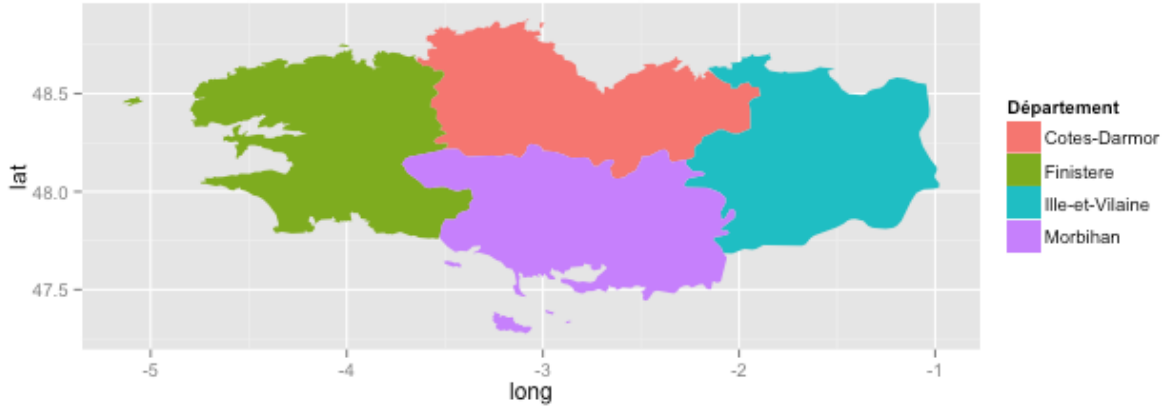
```
ind_bzh <- grep("armor|finis|vilaine|morb",
                unique(map_fr$region), ignore.case = TRUE)

(dep_bzh <- unique(map_fr$region)[ind_bzh])
```

```
## [1] "Cotes-Darmor"    "Finistere"       "Ille-et-Vilaine" "Morbihan"
```

```
map_fr_bzh <- map_data("france", region = dep_bzh)
```

Package `maps`

```
(p_map_fr_bzh <- ggplot(data = map_fr_bzh,
                        aes(x = long, y = lat, group = group, fill = region)) +
  geom_polygon() + coord_equal() + scale_fill_discrete(name = "Département"))
```



Fichier shapefile

- On peut importer des fichiers `shp` (shapefile) ;
- Par exemple, pour tracer les quartiers de Rennes, on peut télécharger le `shp` : http://www.data.rennes-metropole.fr/ ;
- L'importation se fait à l'aide de la fonction `readOGR()`, du package `rgdal`.

```
library("rgdal")
library("maptools")
library("ggplot2")
library("plyr")
```

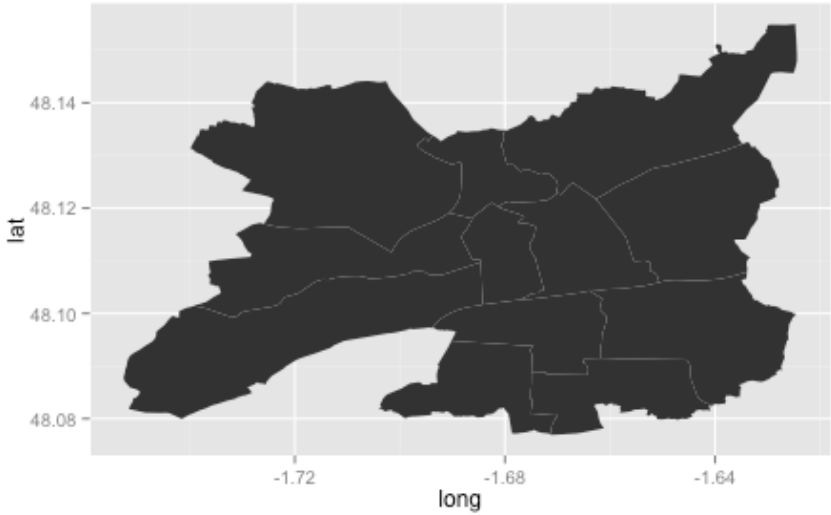
- Pour des informations plus détaillées : https://github.com/hadley/ggplot2/wiki/plotting-polygon-shapefiles

Fichier shapefile

```
# Importer les polygones
rennes <- readOGR(dsn="./quartiers_shp_lamb93", layer="quartiers")

# Étape pour changer la projection de la carte
rennes <- spTransform(rennes, CRS("+proj=longlat +ellps=GRS80"))

# Pour permettre la jointure des objets géométriques
rennes@data$id <- rownames(rennes@data)

# Transformer en data frame pour fournir à ggplot()
rennes_points <- fortify(rennes, region="id")

# Permet d'éviter des trous éventuels
rennes_df <- join(rennes_points, rennes@data, by="id")
```

Fichier shapefile

```
(p_map_rennes <- ggplot(data = rennes_df,
                        aes(x = long, y = lat, group = group)) +
  geom_polygon() +
  coord_equal())
```



Cartes choroplèthes

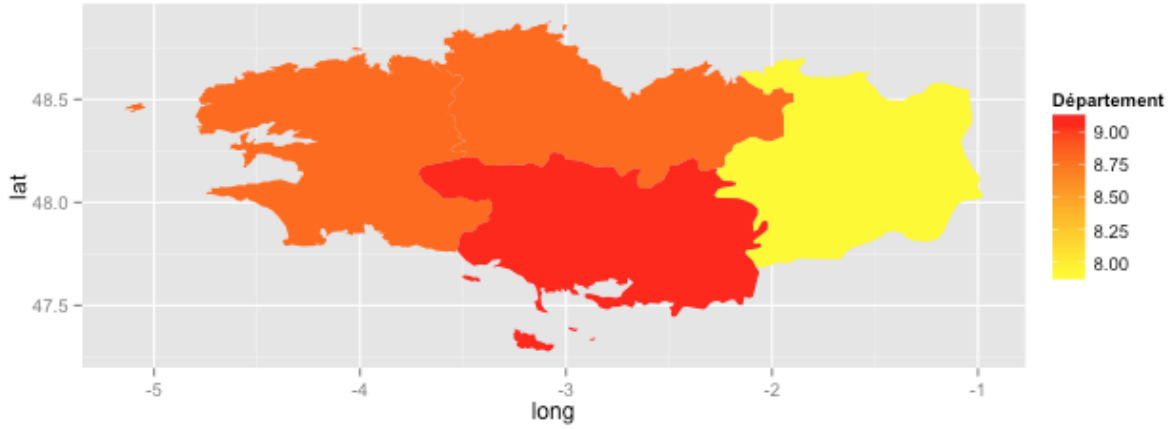
- Cartes où les régions sont remplies par une couleur ;
- En fonction d'une statistique ;
- Il suffit d'ajouter une variable at `data.frame`, avec la valeur de la statistique, pour chaque ligne.

```
tx_chomage_2014_T1 <- data.frame(
  region = c("Cotes-Darmor","Finistere",
             "Ille-et-Vilaine", "Morbihan"),
  tx_chomage_2014_T1 = c(8.8, 8.8,7.9, 9.1))

# Ajout des valeurs pour chaque région
ind_match <- match(map_fr_bzh$region, tx_chomage_2014_T1$region)
map_fr_bzh$tx_chomage_2014_T1 <- tx_chomage_2014_T1[ind_match,
                                                    "tx_chomage_2014_T1"]
```

Cartes choroplèthes

- Il suffit alors de préciser la variable `map_fr_bzh$tx_chomage_2014_T1` à l'esthétique de remplissage.

```
(p_map_fr_bzh <- ggplot(data = map_fr_bzh,
                       aes(x = long, y = lat, group = group,
                           fill = tx_chomage_2014_T1)) +
  geom_polygon() + coord_equal() +
  scale_fill_gradient(name = "Département", low ="#FFFF00", high = "#FF0000"))
```



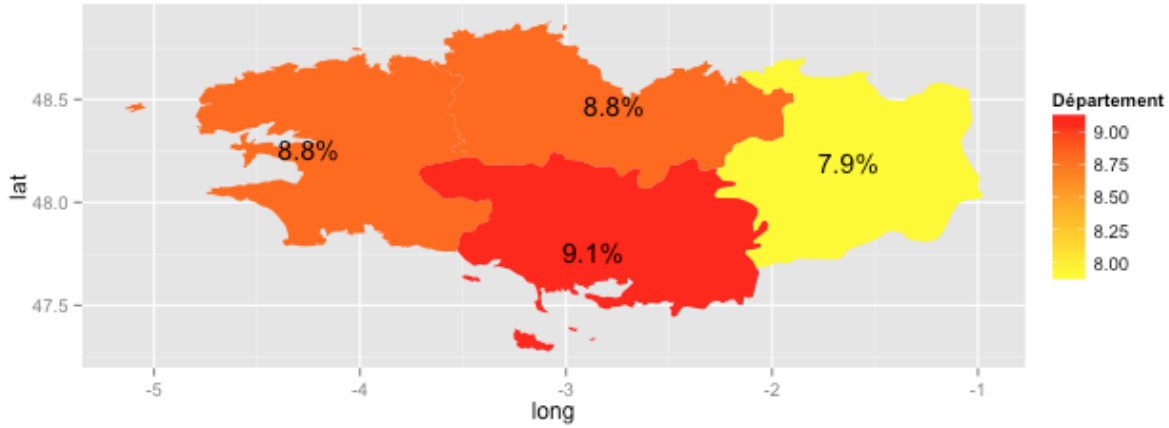
Cartes choroplèthes

- On peut ajouter des annotations, par exemple, au barycentre de chaque département :

```
# Fonction pour trouver le point central du polygone
mid_range <- function(x) mean(range(x, na.rm = TRUE))
centres <- ddply(map_fr_bzh, .(region), colwise(mid_range, .(lat, long)))

# Rajout des taux de chômage
ind_match <- match(centres$region, tx_chomage_2014_T1$region)
centres$tx_chomage_2014_T1 <- tx_chomage_2014_T1$tx_chomage_2014_T1[ind_match]

label_chomage <- paste0(centres$tx_chomage_2014_T1, "%")
```

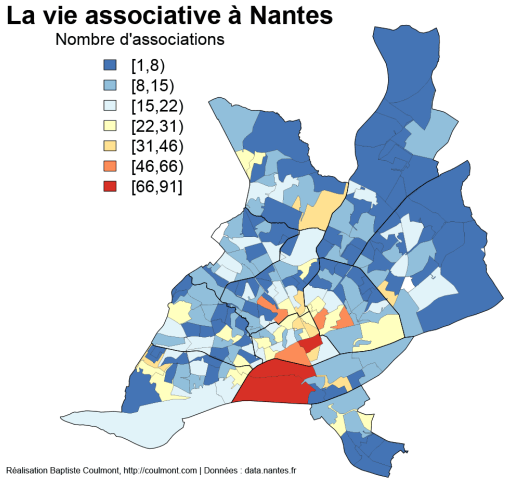
Cartes choroplèthes

```
p_map_fr_bzh + annotate("text", x = centres$long,
                        y = centres$lat, label = label_chomage)
```



Autres cartes, sans utiliser `ggplot2`

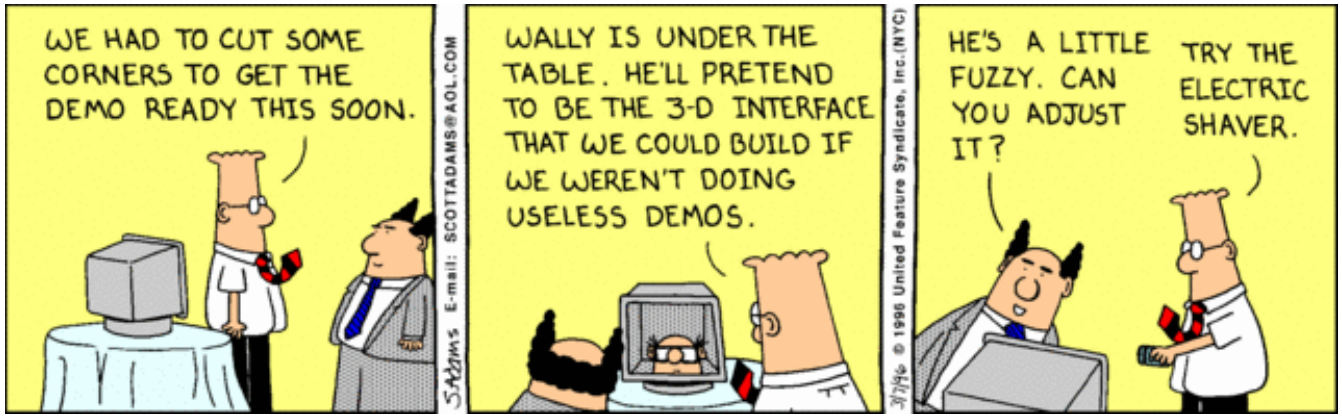
- Il est possible de réaliser de très jolies cartes sans passer par `ggplot2` :



Source : La vie associative à Nantes (1), Baptiste Coulmont

- Une bonne référence pour réussir à faire ces cartes : http://coulmont.com/cartes/rcarto.pdf ;
- Voir aussi http://freakonometrics.hypotheses.org/2319 ;
- Ou encore, plus récent : http://freakonometrics.hypotheses.org/17113.

Un mot sur les graphiques en 3D



Source : Dilbert http://dilbert.com/strips/comic/1996-03-07/

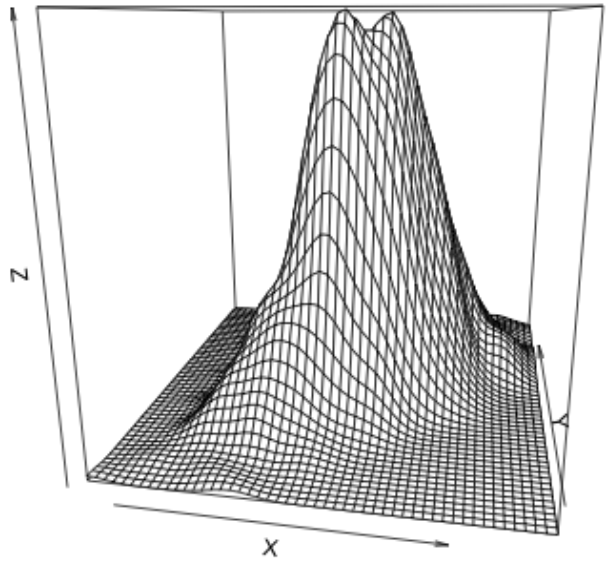
Un mot sur les graphiques en 3D

- La gestion de la 3D n'est pas encore intégrée dans `ggplot2` ;
- Nous ne nous étendrons pas sur le sujet.

```
library(MASS)
set.seed(1)
# Normale bivariée
Sigma <- matrix(c(10,3,3,2),2,2)
biv_n <- mvrnorm(n=1000, rep(0, 2), Sigma)

# Estimation de la densité par la méthode du noyau
biv_n_kde <- kde2d(biv_n[,1], biv_n[,2], n = 50)
```

Un mot sur les graphiques en 3D

```
persp(biv_n_kde, theta = 10, phi = 15, xlab = "X")
```



Un mot sur les graphiques en 3D

- Pour avoir une représentation interactive, on peut utiliser `plot3D()` du package `rgl`.

```
library(rgl)
set.seed(1)
n <- 10000
x <- rnorm(n, mean = 38)
y <- rnorm(n, mean = 42)

biv_kde <- kde2d(x, y, n = 50)
den_z <- biv_kde$z

surface3d(biv_kde$x,biv_kde$y,den_z*20,color="#FF2222",alpha=0.5)
```