

A photograph of a modern building with a glass facade, featuring a prominent blue diagonal structural element. The sky is blue with a single white cloud. In the foreground, there is a courtyard with greenery and a building with a sign that partially reads 'Institut Pasteur'.

Graphiques avec ggplot2

Vincent Guillemot

Avant toutes choses

Nous aurons besoin du package `ggplot2` :

- Vérifier que le package `ggplot2` est bien installé
- Si non, l'installer, puis le charger

```
library(ggplot2)
```

Nous allons également avoir besoin des données `fruits` :

```
fruits <- readxl::read_excel("fruits.xlsx")
```

Bien choisir sa représentation graphique



from Data to Viz

From Data to Viz : <https://www.data-to-viz.com/>

Un échantillon de ce qu'il est possible de faire avec ggplot2

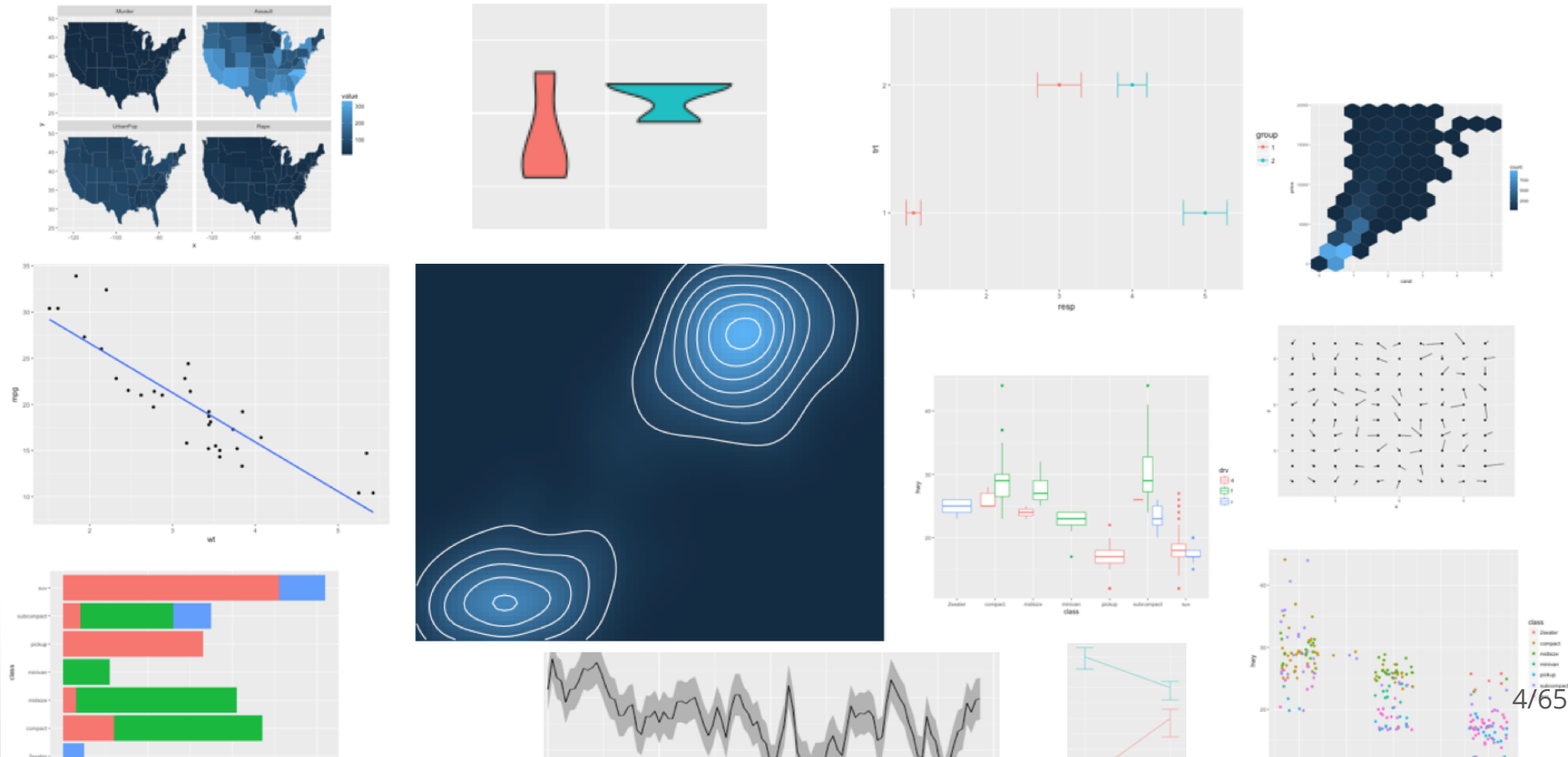
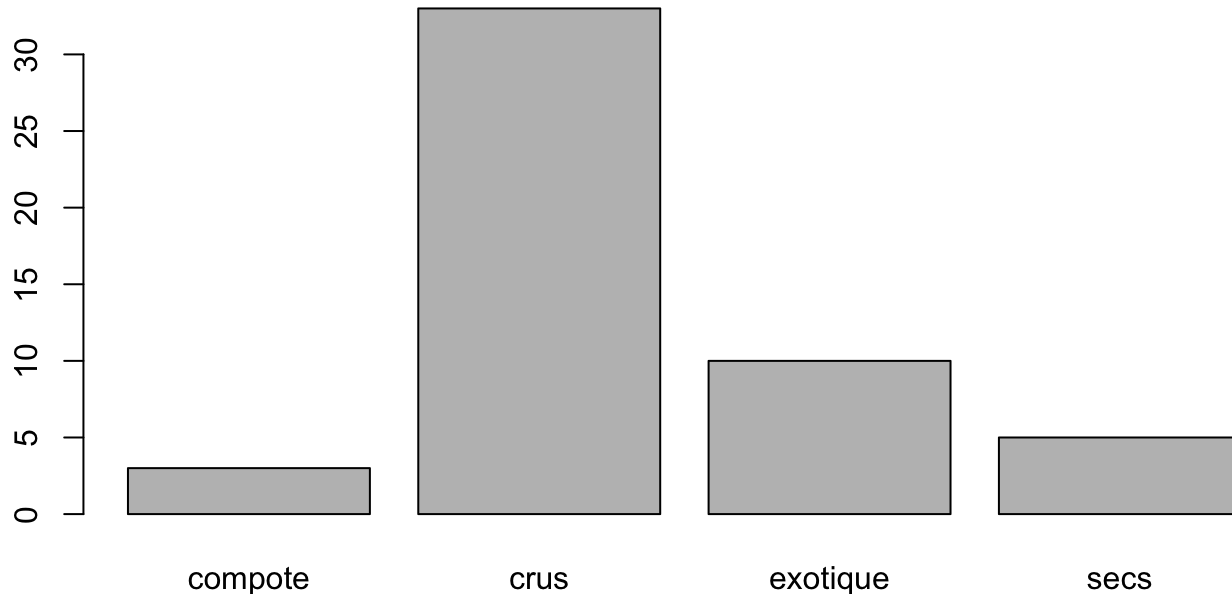


Diagramme en bâtons

Rappel : la fonction barplot

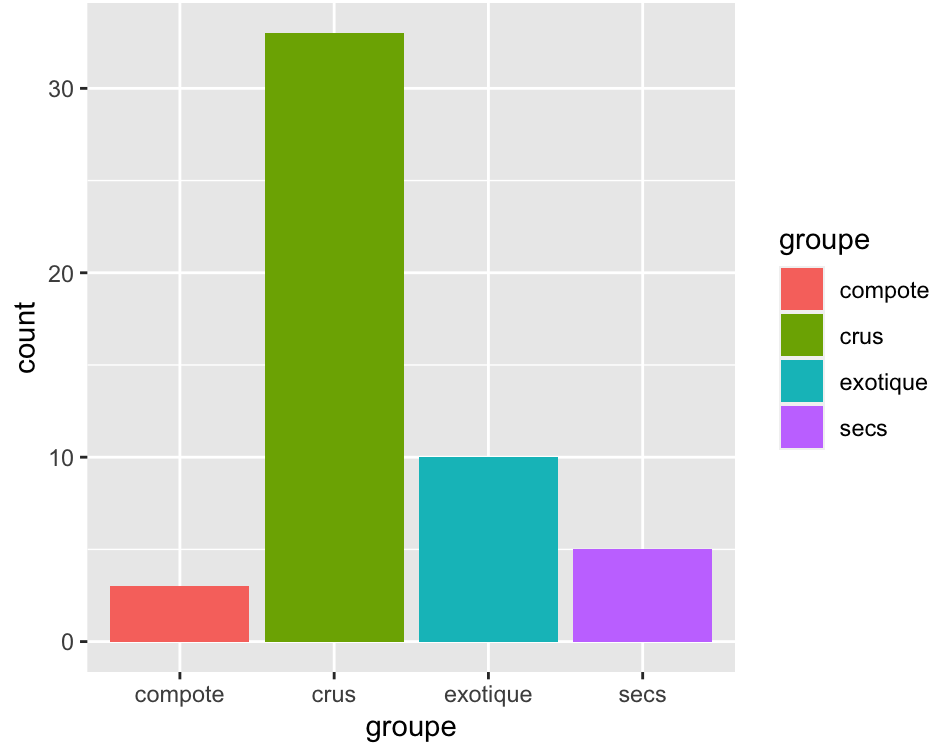
La fonction de base pour les diagrammes en bâtons est `barplot` :

```
barplot(table(fruits$groupe))
```



La fonction geom_bar

```
ggplot(data = fruits, aes(x = groupe, fill = groupe)) +  
  geom_bar()
```



Décomposition de la commande

- `ggplot` : création d'un canevas vide dans on va dessiner le(s) graphe(s)
- `aes` : déclaration des paramètres esthétiques du graphes (position horizontale, verticale, couleur épaisseur, forme, transparence etc...)
- `geom_bar` : utilisation d'une géométrie

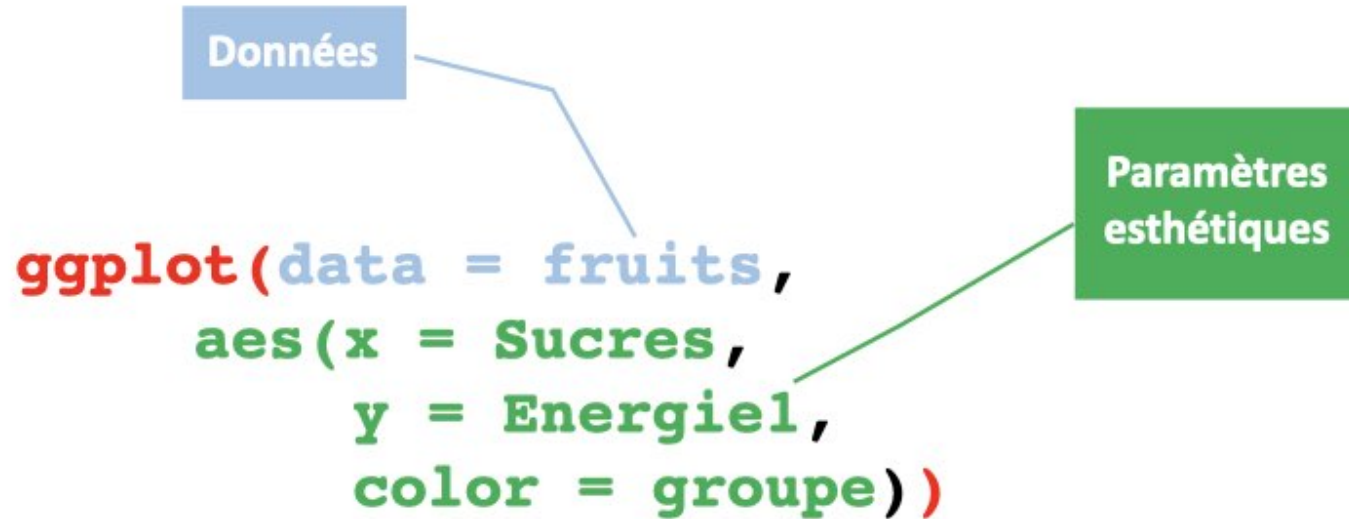
Les données

Données

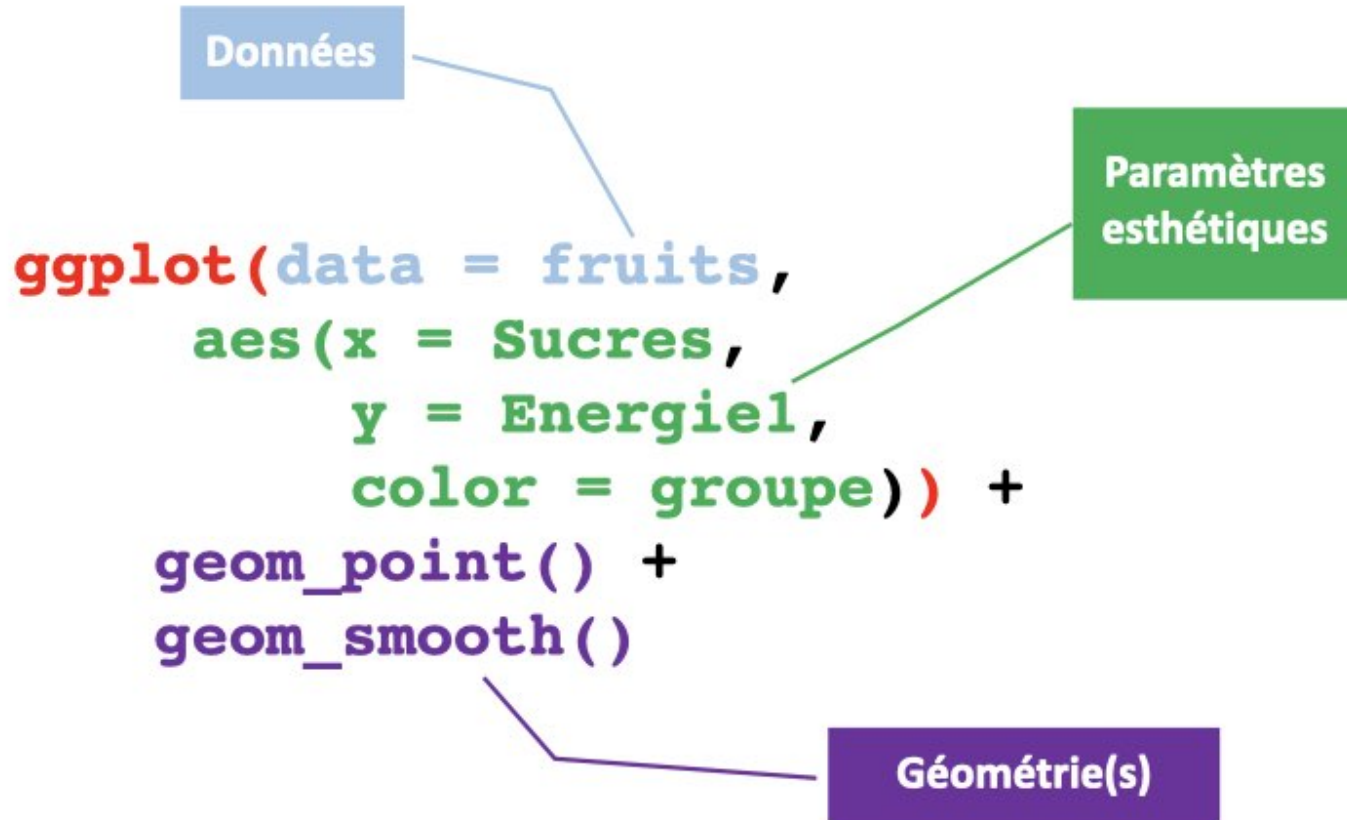
```
ggplot(data = fruits,
```

	V1	V2	Groups	
—	1	1	Group 1	→ ●
—	2	2	Group 2	→ ●
—	3	5	Group 1	→ ●
—	4	10	Group 1	→ ●
—	5	17	Group 2	→ ●
—	6	9	Group 1	→ ●
—	7	11	Group 1	→ ●
—	8	13	Group 2	→ ●

Les paramètres esthétiques



Les géométries



L'image à retenir



[G]rammar of [G]raphics

- 1) DATA: a set of data operations that create variables from datasets,
- 2) TRANS: variable transformations (*e.g.*, *rank*),
- 3) SCALE: scale transformations (*e.g.*, *log*),
- 4) COORD: a coordinate system (*e.g.*, *polar*),
- 5) ELEMENT: graphs (*e.g.*, *points*) and their aesthetic attributes (*e.g.*, *color*),
- 6) GUIDE: one or more guides (*axes*, *legends*, etc.).

The grammar of Graphics, Leland Wilkinson

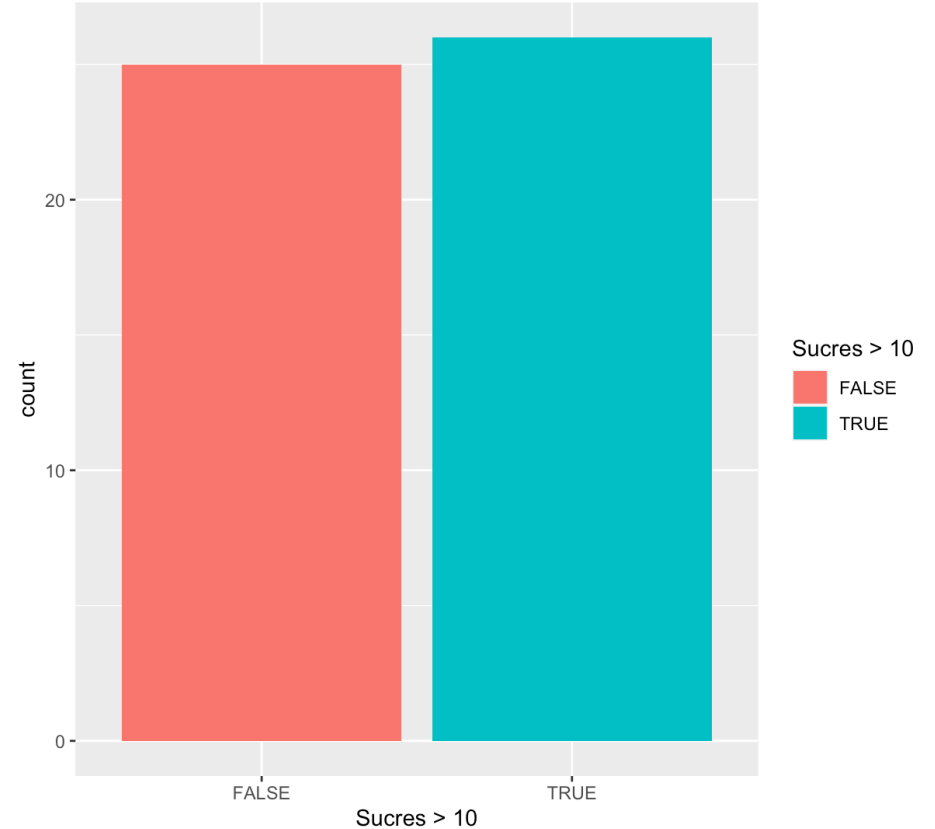
Implémentation dans ggplot2

Données	<code>data</code>	Les données à représenter. Chaque ligne représente un élément à représenter graphiquement.
Géométries	<code>geom_</code>	Les formes à créer pour représenter les données. Cela peut être des points, des lignes, des surfaces etc.
Esthétiques	<code>aes()</code>	Les paramètres esthétiques de ces formes. Par exemple la position, la couleur, la taille, la forme, la transparence etc.
Échelles	<code>scale_</code>	Des fonctions permettant de paramétrer la transformation de données en formes ou en objets graphiques. Par exemple la fonction <code>scale_color_manual</code> permet de choisir soi-même les couleurs à utiliser dans un graphique.

A vous !

Comment corriger la commande suivante pour obtenir le graphe à droite ?

```
ggplot(***,  
  aes(***,  
    fill = Sucres > 10)) +  
  geom_***()
```



Un peu d'histoire

- Il y a eu un `ggplot`"1" (voir [ici](#))
- Développement commencé en 2005
- Hadley Wickham (*Chief Scientist at RStudio + Adjunct Professor of Statistics*)
- D'excellents cours, parfois avec sa soeur Charlotte



Hadley Wickham

Quelques géométries

Nous allons voir ensemble quelques géométries particulières qui permettent de créer des graphes classiques.

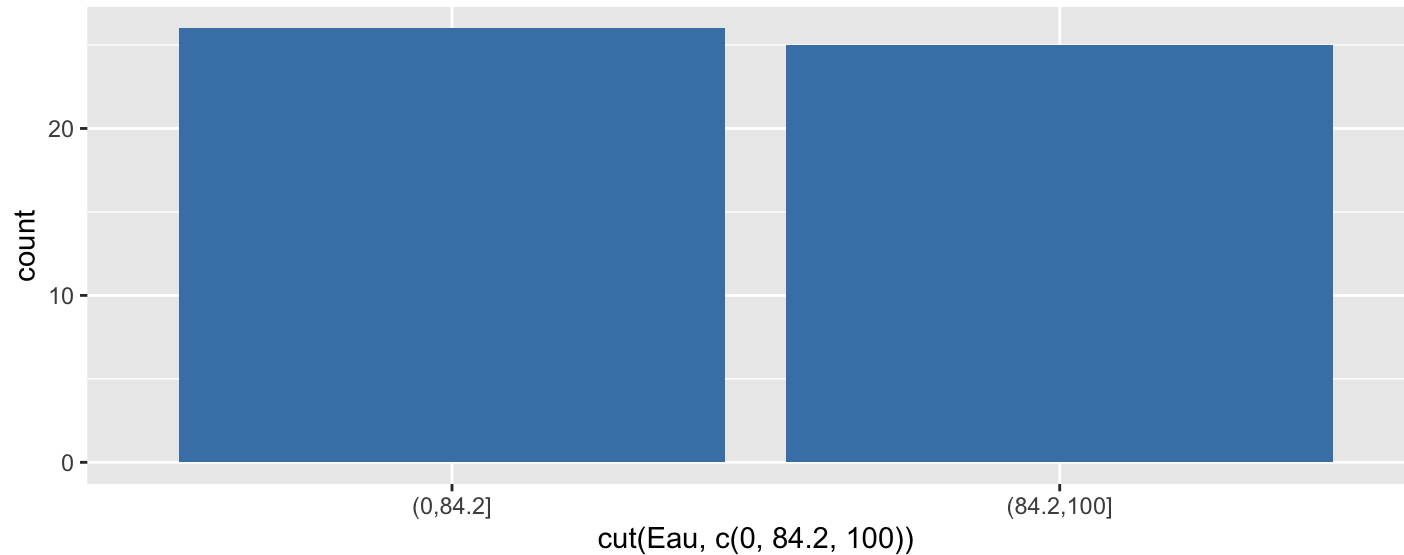
<code>geom_bar</code>	Diagramme en bâtons sur des données non-agrégées
<code>geom_col</code>	Diagramme en bâtons (encore) sur des comptages existants
<code>geom_histogram</code>	Histogramme d'une variable quantitative
<code>geom_boxplot</code>	Diagramme de Tukey aka "boîte à moustache" aka <i>boxplot</i>
<code>geom_violin</code>	Diagramme en "violons"
<code>geom_point</code>	Nuage de points créé à partir de deux variables quantitatives
<code>geom_line</code>	Ligne tracée à partir de deux variables quantitatives

Diagrammes en bâtons

Avec geom_bar

On a déjà vu comment faire :

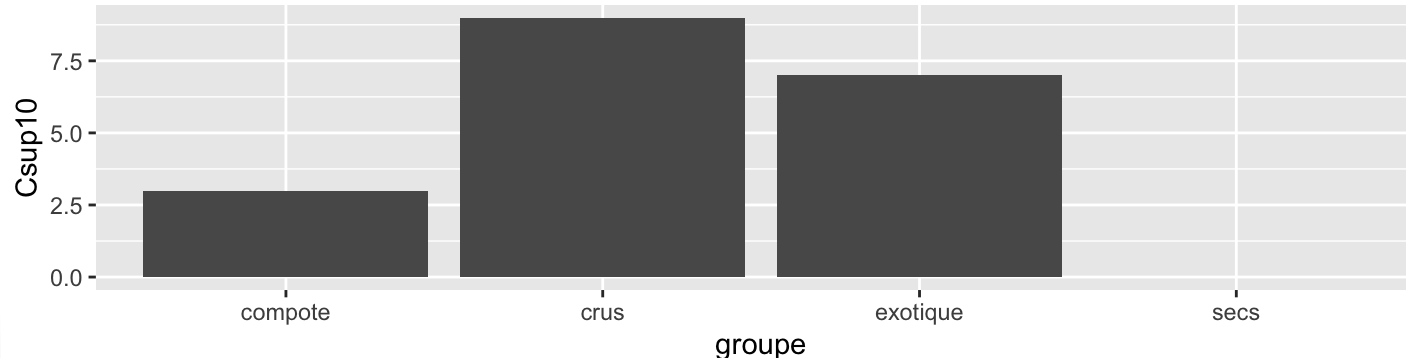
```
ggplot(fruits, aes(cut(Eau, c(0, 84.2, 100)))) +  
  geom_bar(fill = "steelblue")
```



Avec geom_col

S'utilise quand on dispose déjà de comptages.

```
dat.count <- fruits %>%  
  group_by(groupe) %>%  
  summarize(Csup10 = sum(VitamineC >= 10))  
  
ggplot(data = dat.count, aes(x = groupe, y = Csup10)) +  
  geom_col()
```



A vous !

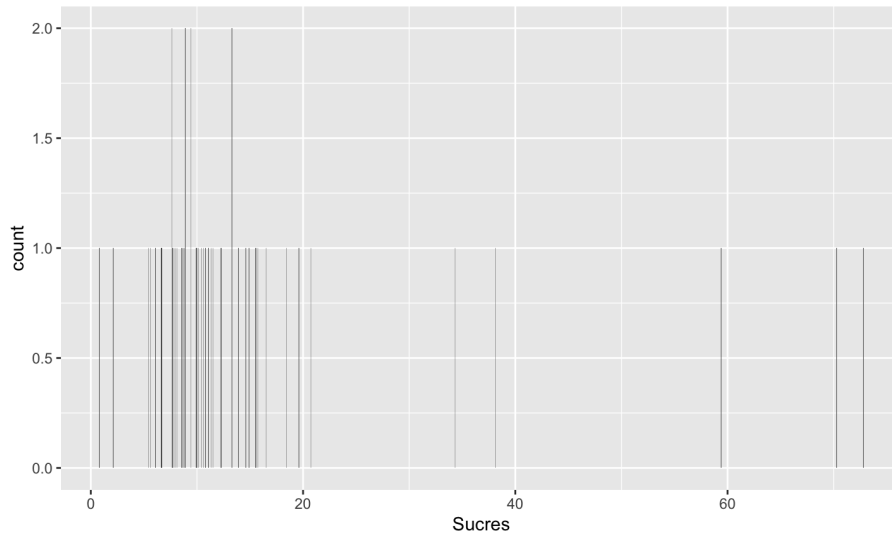
Comment modifier les deux commandes précédentes pour faire un diagramme en bâtons montrant des pourcentages plutôt que des comptages bruts ?

(PS : il y a plus d'une solution possible)

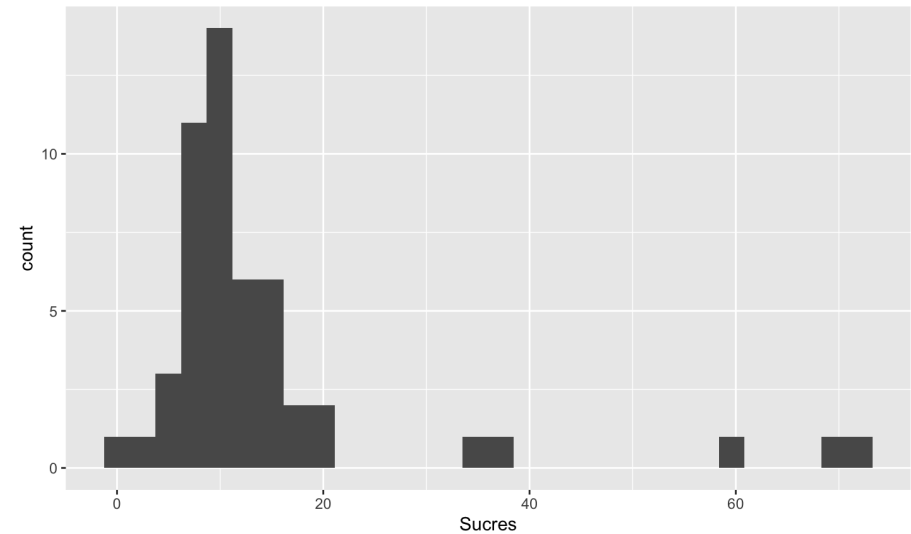
Histogrammes

Histogramme ou diagramme en bâtons ?

```
ggplot(fruits, aes(Sucres)) +  
  geom_bar()
```



```
ggplot(fruits, aes(Sucres)) +  
  geom_histogram()
```



Histogramme ou diagramme en bâtons ?

Diagramme en bâtons

Représente des comptages de :

- variables nominales
- variables ordinales
- variables discrètes

Histogramme

Représente des comptages ou des densités de :

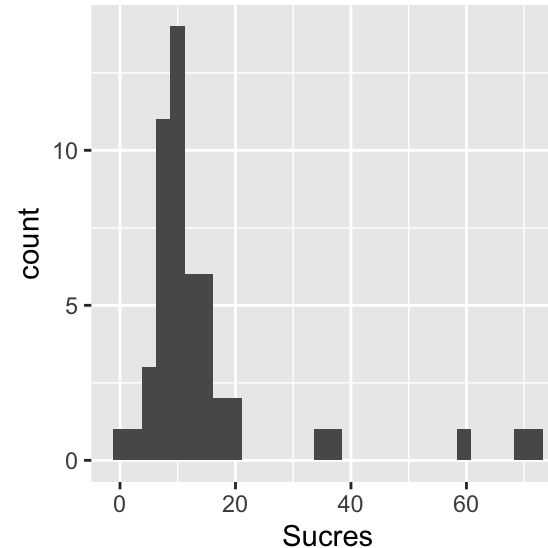
- variables continues
- variables discrètes

Pour ce genre de graphe, il est important de choisir les intervalles sur lesquels les données seront énumérées.

L'histogramme par défaut

- Axe des y : les comptages de l'intervalle donné
- Axe des x :
 - des intervalles de même amplitude,
 - trente intervalles,
 - pas de séparation visuelle entre les intervalles (sauf ceux qui sont vides)
 - des *rectangles* gris foncé
 - un *message*

```
ggplot(fruits, aes(Sucres)) +  
  geom_histogram()
```



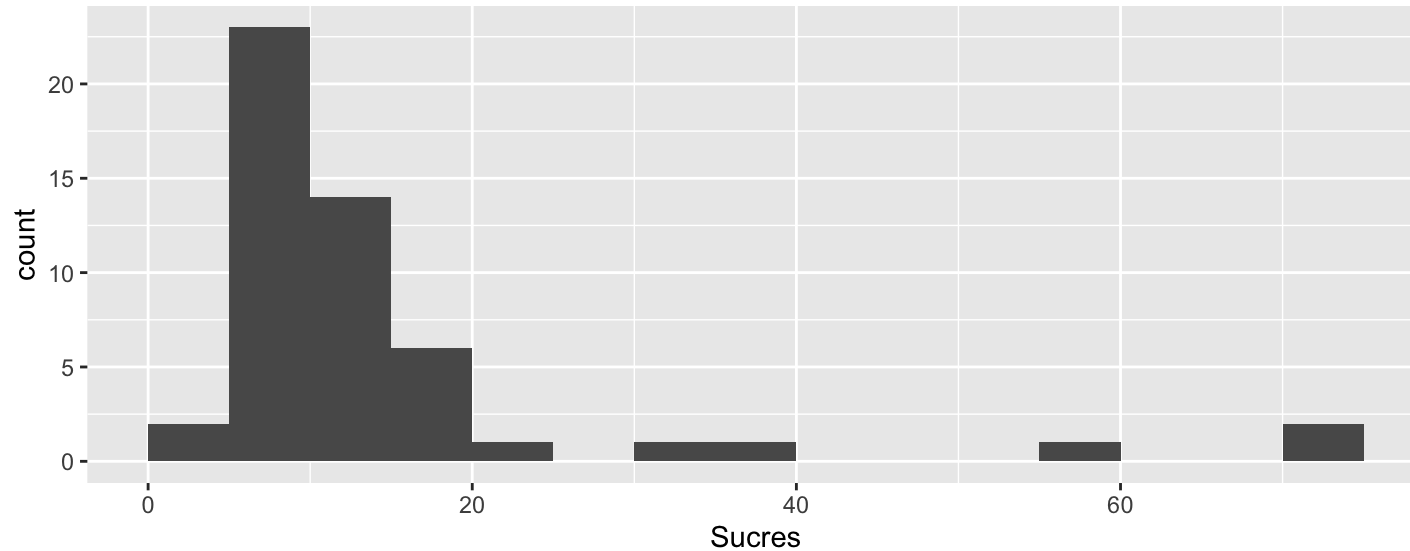
``stat_bin()` using `bins = 30`.`
Pick better value with
``binwidth``.

Exercice qui n'a (presque) rien à voir!

Dans un document R Markdown, comment faire pour ne pas afficher le message dans votre rapport ?

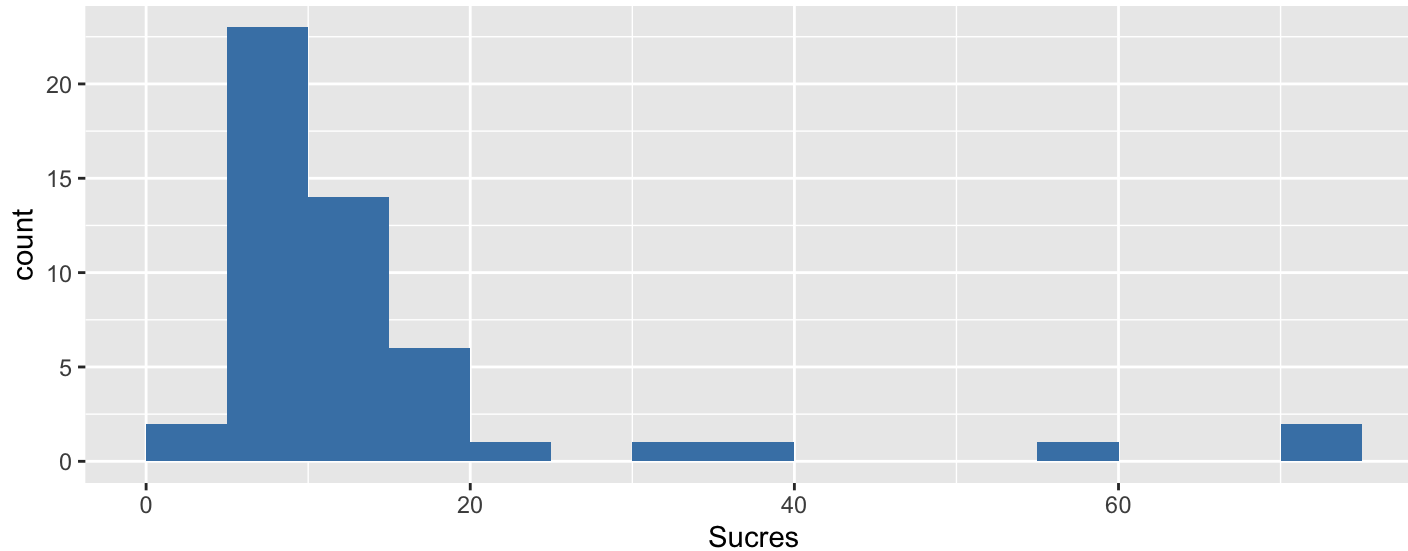
... il y a plusieurs solutions possibles...

Modification des intervalles



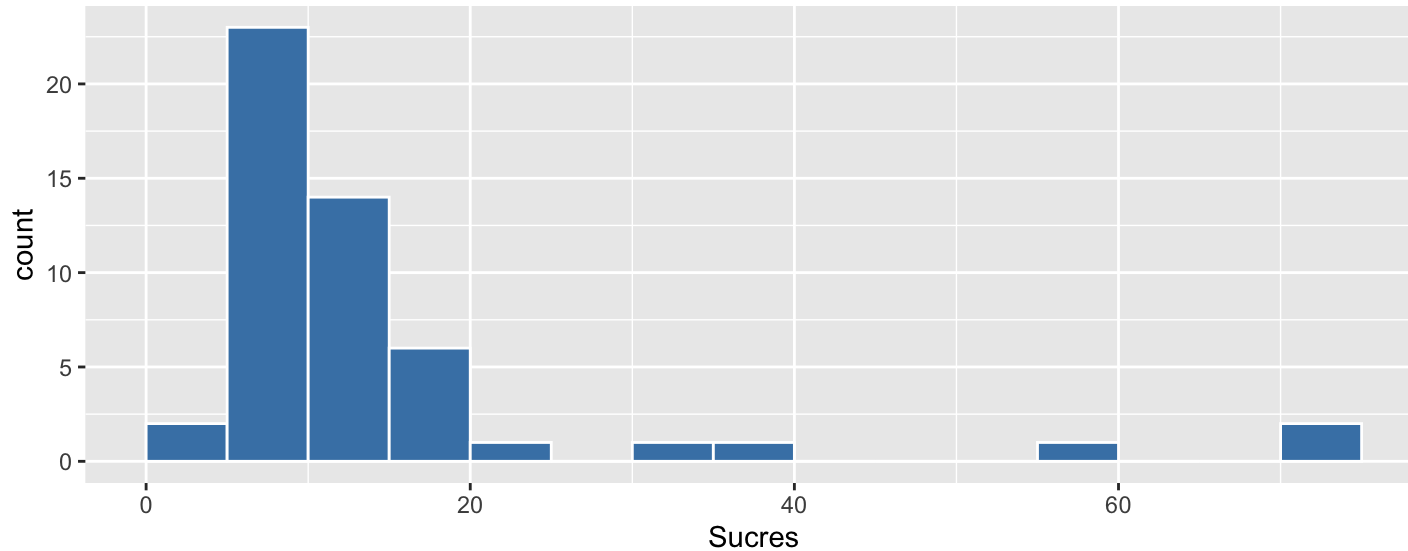
```
ggplot(fruits, aes(Sucres)) +  
  geom_histogram(breaks = seq(0, 75, 5))
```

Modification de la couleur



```
ggplot(fruits, aes(Sucres)) +  
  geom_histogram(breaks = seq(0, 75, 5),  
                 fill = "steelblue")
```

Modification de la couleur

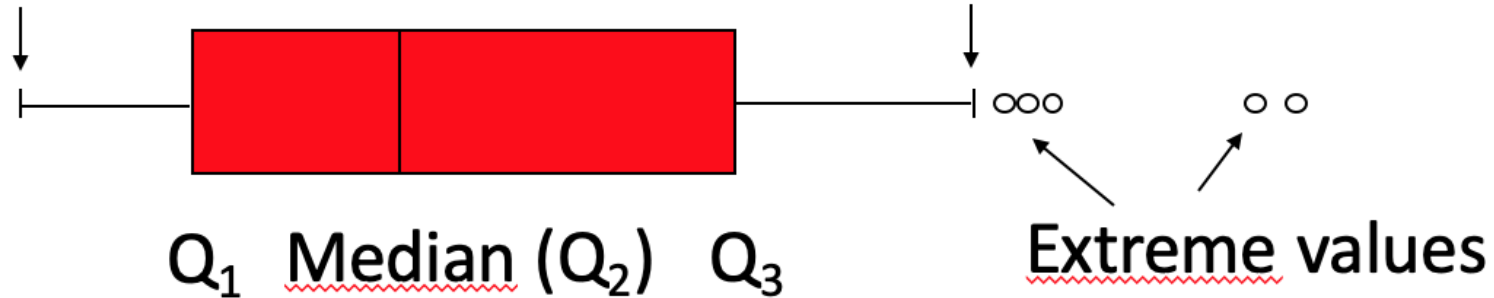


```
ggplot(fruits, aes(Sucres)) +  
  geom_histogram(breaks = seq(0, 75, 5),  
                 fill = "steelblue",  
                 color = "white")
```

Boxplot

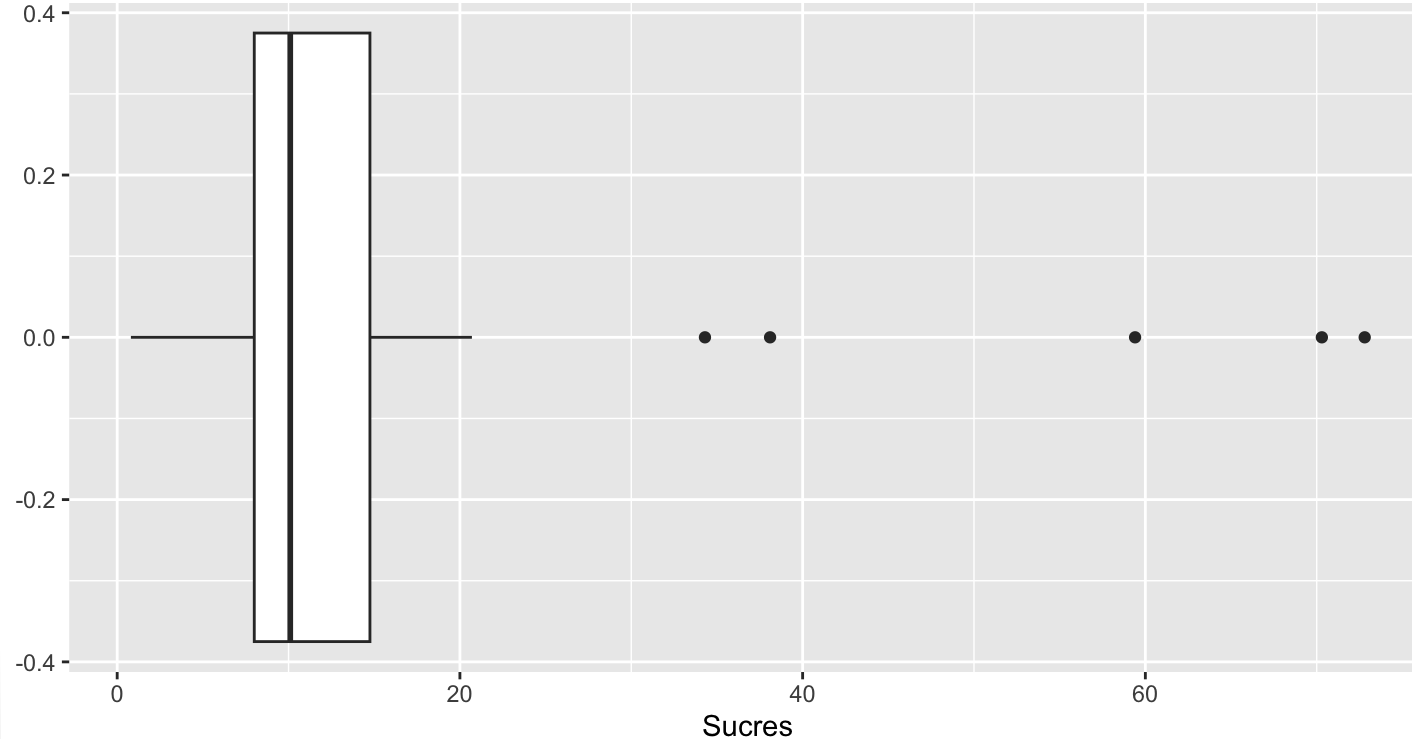
Smallest value above
 $Q_1 - 1,5 * (Q_3 - Q_1)$

Largest value under
 $Q_3 + 1,5 * (Q_3 - Q_1)$



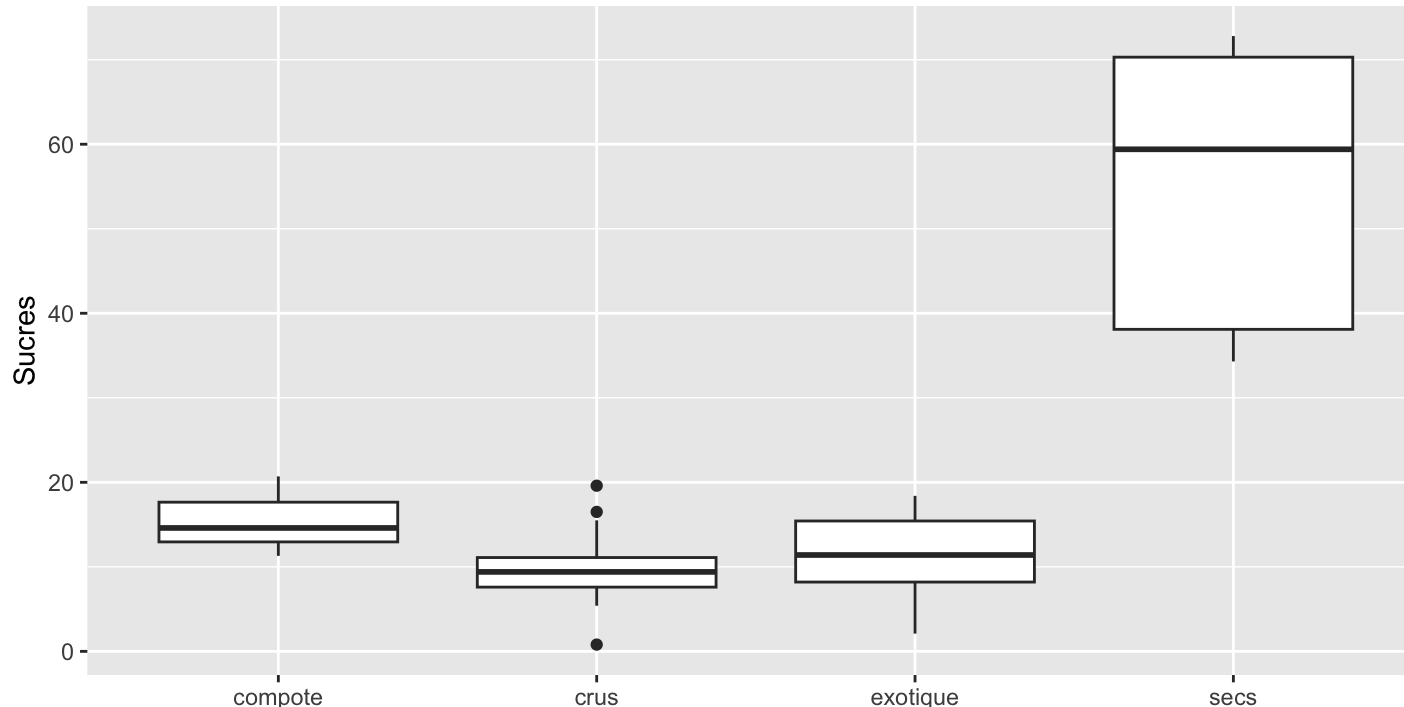
Boxplot d'une variable

```
ggplot(data=fruits, aes(x = Sucres)) +  
  geom_boxplot()
```



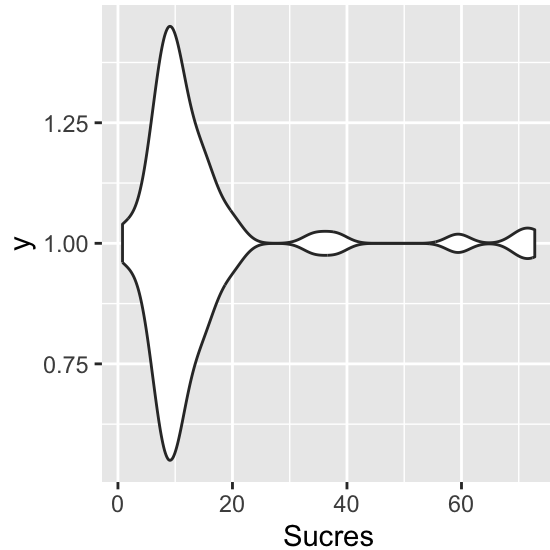
Boxplot : lien entre une variable catégorielle et une variable continue

```
ggplot(data=fruits, aes(x=groupe, y=Sucres)) +  
  geom_boxplot()
```

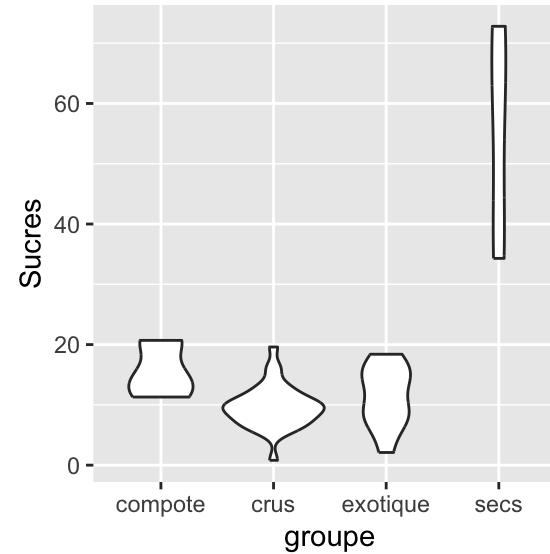


Avec des violons

```
ggplot(data=fruits,  
       aes(x = Sucres, y = 1)) +  
  geom_violin()
```



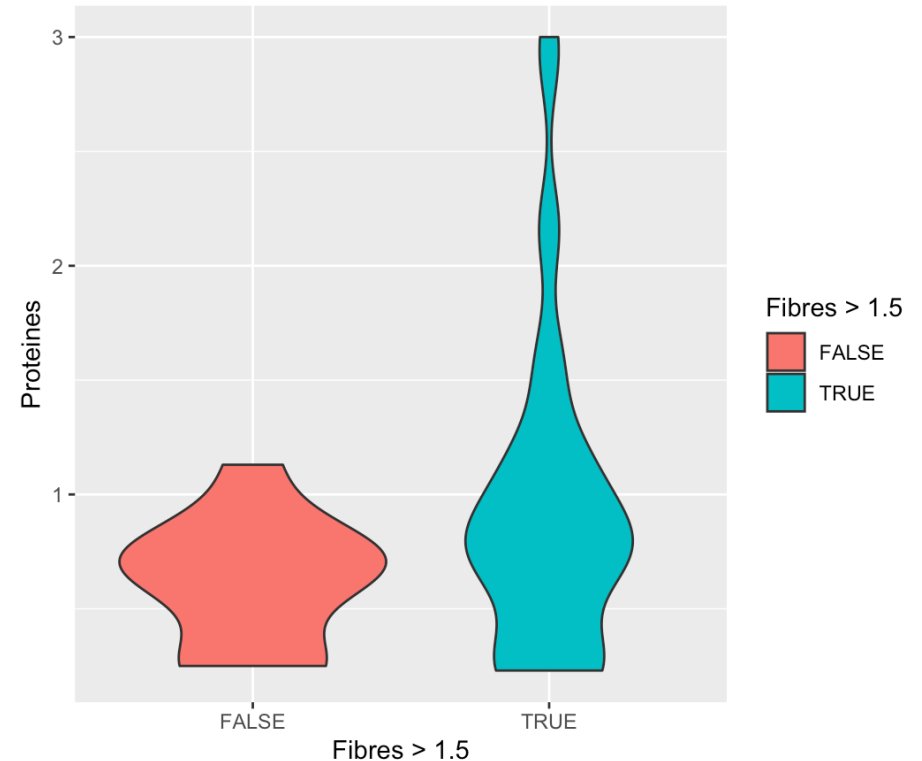
```
ggplot(data=fruits,  
       aes(x = groupe, y = Sucres)) +  
  geom_violin()
```



A vous !

Complétez le code suivant pour obtenir la figure de droite :

```
ggplot(fruits,  
      aes(x = Fibres > 1.5,  
          y = Proteines,  
          fill = ***) +  
  geom_***()
```



Personnalisation

Thèmes

Les *thèmes* sont des fonctions qui permettent de modifier certains paramètres graphiques comme :

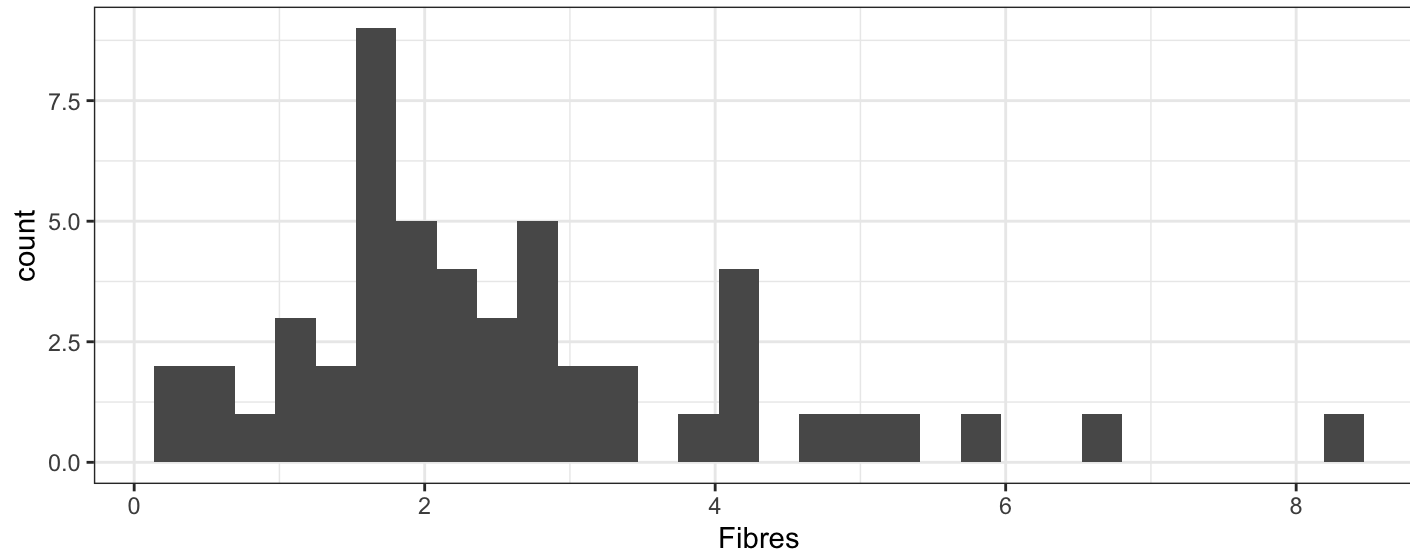
- la couleur du fond,
- la couleur des axes,
- la couleur des grilles majeures et mineures,
- etc.

Exemple de fonctions thèmes (`theme_***()`) :

- `theme_bw()` : pour un thème en noir et blanc,
- `theme_minimal()` : pour un thème minimaliste,
- `theme_void()` : pour un thème dépouillé

Exemple sur un histogramme : `theme_bw()`

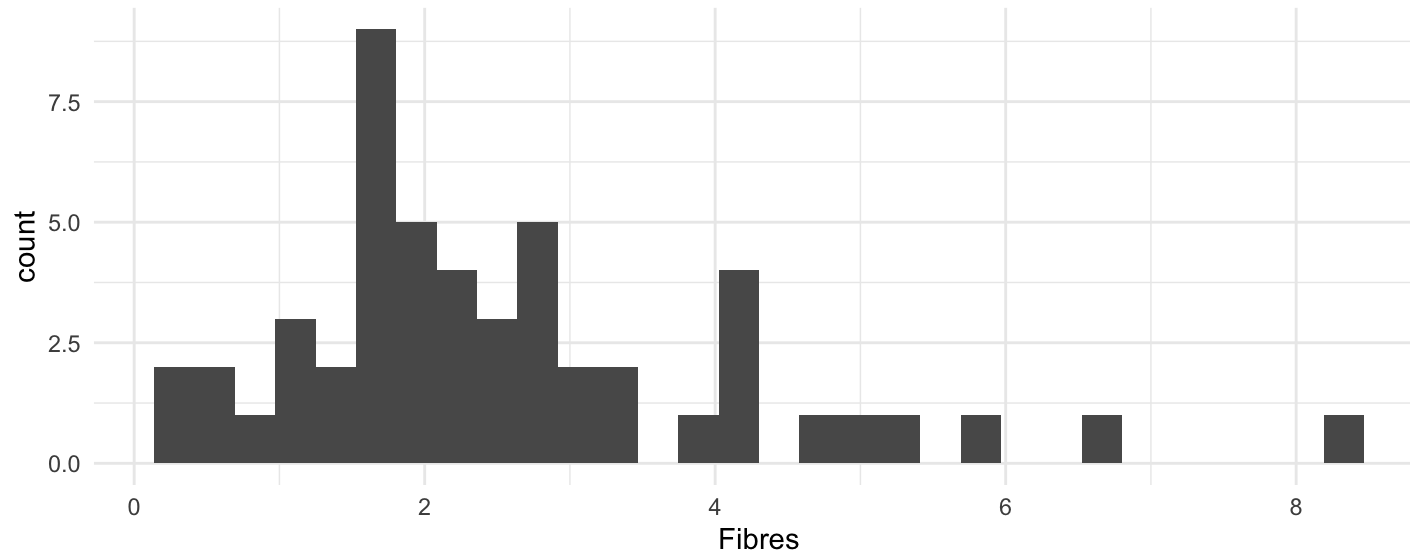
```
ggplot(fruits, aes(Fibres)) +  
  geom_histogram() +  
  theme_bw()
```



Exemple sur un histogramme :

theme_minimal()

```
ggplot(fruits, aes(Fibres)) +  
  geom_histogram() +  
  theme_minimal()
```



Exemple sur un histogramme : `theme_void()`

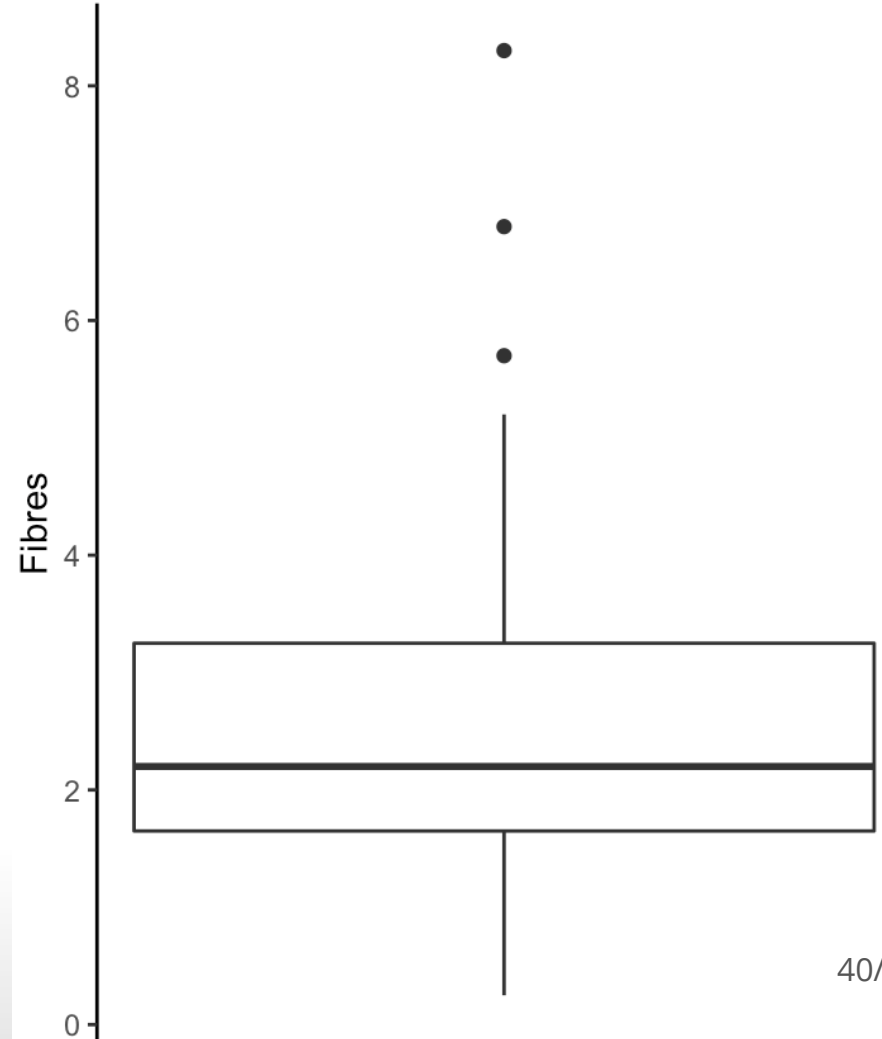
```
ggplot(fruits, aes(Fibres)) +  
  geom_histogram() +  
  theme_void()
```



A vous !

1. Consultez la page d'aide de la fonction `theme_bw` avec la commande `?theme_bw`
2. Choisissez la thème permettant de réaliser le graphe à droite en complétant la commande suivante.

```
ggplot(fruits, aes(y = Fibres)) +  
  geom_boxplot() +  
  theme_***()
```



Autres personnalisations “simples”

- Titre : avec la fonction `ggtitle`
- Titre de l'axe des x : avec la fonction `xlab`
- Titre de l'axe des y : avec la fonction `ylab`

Mais je vous conseille d'utiliser la fonction `labs` qui permet de faire tout cela, et plus !

```
labs(  
  title = "Titre du graphe",  
  subtitle = "Sous-titre du graphe",  
  x = "Titre de l'axe des x",  
  y = "Titre de l'axe des y",  
  color = "Titre de la légende des couleurs",  
  shape = "Titre de la légende des formes"  
)
```

Personnalisations avancées

Avec la fonction `theme()`, qui a une syntaxe bien particulière : chaque élément doit être spécifié selon sa nature.

- Pour changer la taille, la police, la couleur d'élément de type "texte", on utilise `element_text(size=, colour = "", family = "")`
- Pour changer la couleur et la taille d'une ligne, on utilise `element_line(colour="", size=)`
- Pour personnaliser la couleur du fond, on utilise `element_rect(fill = "")`

Ce que l'on peut changer avec `theme()`

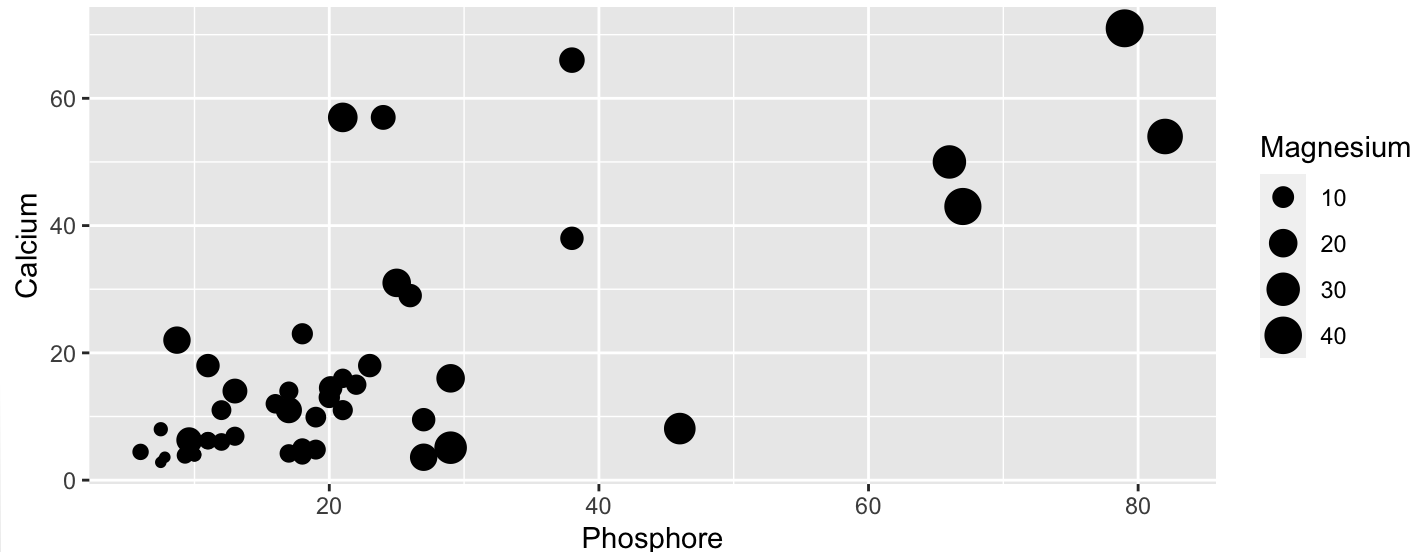
- `axis.title`, `axis.title.x`, `axis.title.y` : taille, police, couleur, ...
- `axis.text`, `axis.text.x`, `axis.text.y` : taille, police, couleur, ...
- `axis.ticks`, `axis.ticks.x`, `axis.ticks.y`
- `axis.line`, `axis.line.x`, `axis.line.y`
- `panel.background` : couleur
- `panel.grid.major`, `panel.grid.minor` : couleur, taille
- `legend.text` : taille, police, couleur
- `legend.position`
- `plot.title` : taille, police, couleur

Nuages de points

Avec `geom_point`

Cette géométrie nécessite des paramètres esthétiques de position (en x et y), et accepte optionnellement des paramètres esthétiques de taille, couleur et forme.

```
ggplot(fruits, aes(x = Phosphore, y = Calcium, size = Magnesium)) +  
  geom_point()
```



Paramètres esthétiques

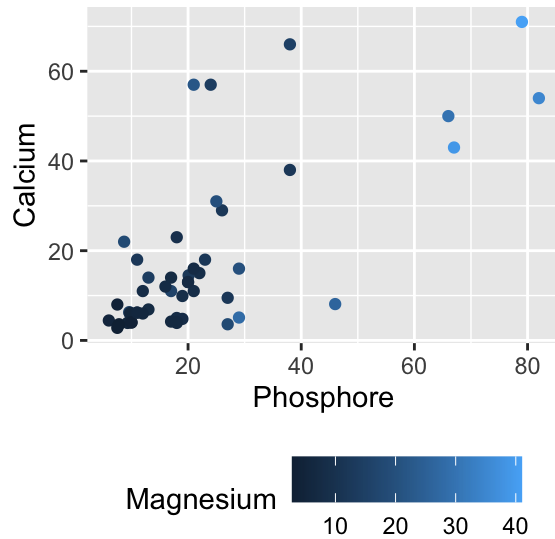
Lorsqu'ils sont spécifiés dans la fonction `aes()`, ces paramètres appliquent les valeurs d'une variable à une caractéristique des objets graphiques tracés par les géométries.

- `color` ou `colour` : couleur du point
- `fill` : couleur de remplissage
- `size` : taille
- `shape` : forme
- `alpha` : transparence
- `linetype` : type de ligne
- `label` : étiquettes

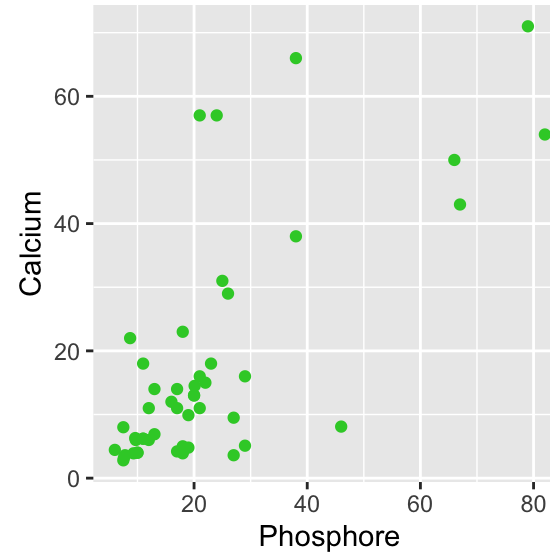
Lorsqu'ils sont appliqués en dehors de la fonction `aes()`, leur comportement est plus général !

Example

```
ggplot(fruits,  
  aes(x = Phosphore, y = Calcium,  
    color = Magnesium)) +  
  geom_point() +  
  theme(legend.position = "bottom")
```



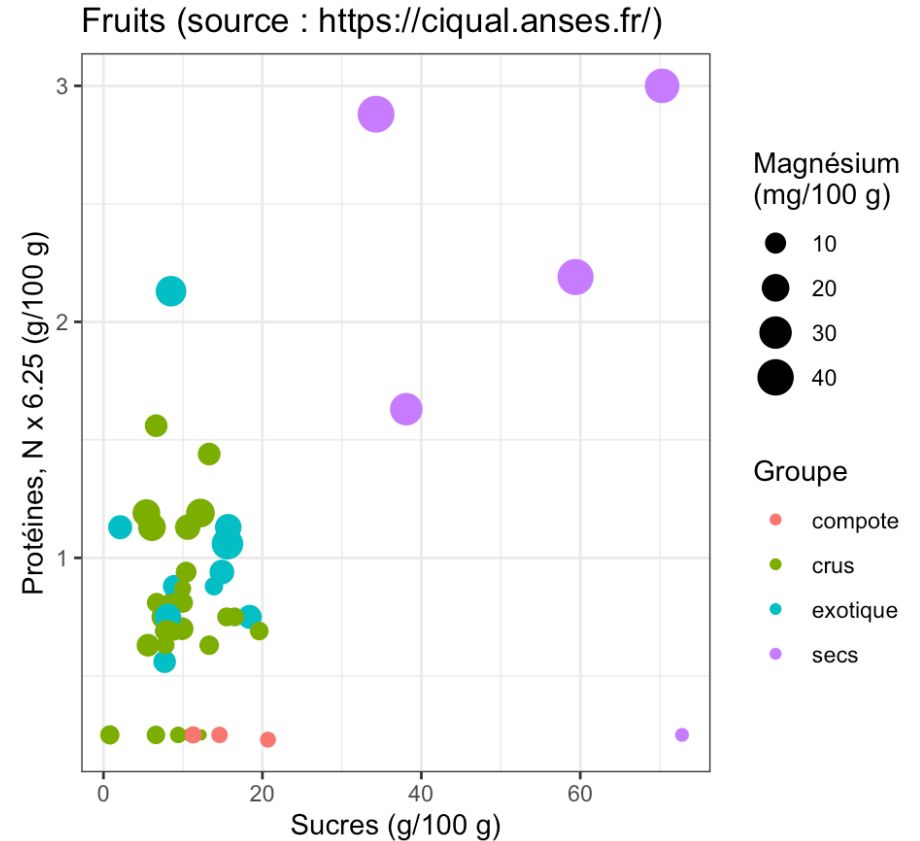
```
ggplot(fruits,  
  aes(x = Phosphore, y = Calcium)) +  
  geom_point(color = "limegreen")
```



A vous !

Complétez la commande suivante pour obtenir le graphe ci-contre.

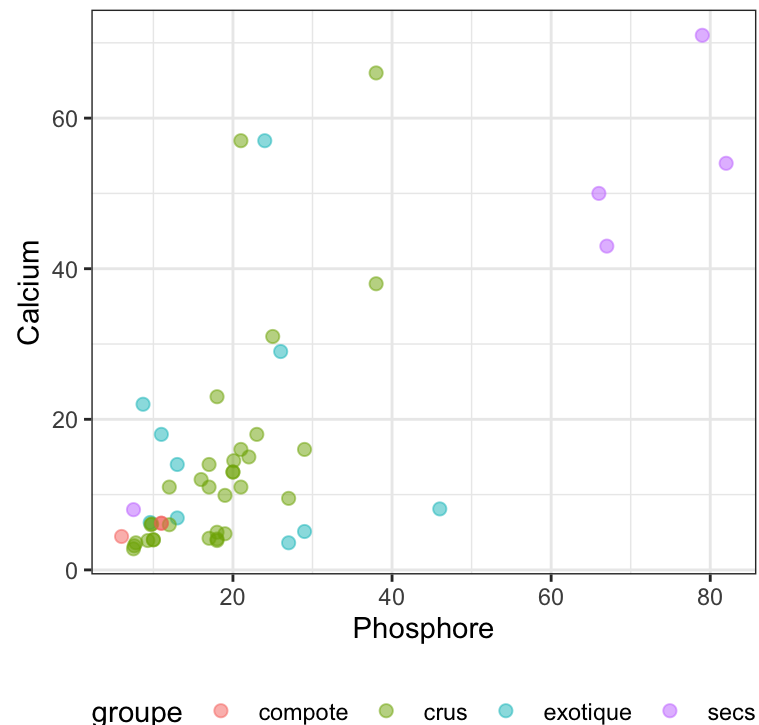
```
ggplot(fruits,  
  aes(x = Sucres,  
      y = Proteines,  
      *** = Magnesium,  
      *** = ***) +  
geom_***() +  
  ***(title = "Fruits",  
    x = "Sucres (g/100 g)",  
    y = "Protéines, N x 6.25 (g/100 g)",  
    size = "Magnésium\n(mg/100 g)",  
    ***= "Groupe") +  
theme_***()
```



Au secours, mes points se superposent!

Pas de panique, on peut utiliser la transparence (aka alpha) :

```
ggplot(fruits,  
       aes(x = Phosphore,  
           y = Calcium,  
           color = groupe)) +  
  geom_point(alpha = 0.5,  
             size = 2) +  
  theme_bw() +  
  theme(legend.position =  
        "bottom")
```



Changer les échelles

Avec les fonctions `scale_***`

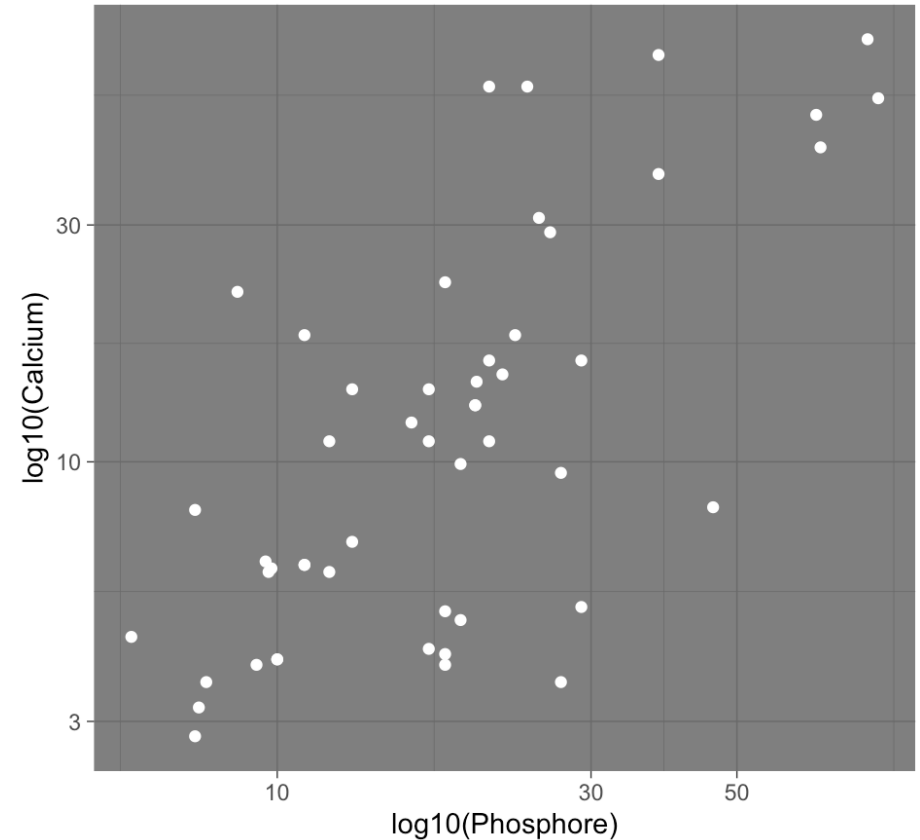
Ces fonctions vont vous permettre de personnaliser l'échelle, en x, en y, mais pas seulement ! Ce concept est généraliser dans `ggplot2` à de nombreux paramètres esthétiques. Par exemple :

- `scale_x_log10()` permet de transformer l'échelle des x en échelle logarithmique,
- `scale_y_log10()` permet de transformer l'échelle des y en échelle logarithmique,
- `scale_color_manual()` permet de personnaliser les couleurs,
- `scale_fill_manual()` permet de personnaliser les couleurs de remplissage,
- `scale_x_continuous()` permet de personnaliser l'axe des x lorsque x est une variable "continue",
- `scale_y_discrete()` permet de personnaliser l'axe des y lorsque y est une variable "discrète",
- `scale_x_discrete()` permet de personnaliser l'axe des x lorsque x est une variable "discrète",
- `scale_y_continuous()` permet de personnaliser l'axe des y lorsque y est une variable "discrète".

A vous !

Complétez la commande suivante pour obtenir le graphe ci-contre.

```
ggplot(fruits,  
      aes(Phosphore,  
          Calcium)) +  
  geom_point(*** = "white") +  
  scale_***() +  
  scale_***() +  
  labs(x = "log10(Phosphore)",  
       y = "log10(Calcium)") +  
  theme_dark()
```



Avec les fonctions `coord_***`

Pour modifier le système de coordonnées après avoir appliqué toutes les transformations spécifiées auparavant (par une fonction `scale_***` par exemple). Par exemple :

- `coord_fixed` pour fixer le ratio des unités de l'axe des y sur les unités de l'axe des x,
- `coord_equal` quand ce ratio vaut 1,
- `coord_flip` pour échanger les axes,
- `coord_polar` pour passer d'un système de coordonnées cartésien à un système de coordonnées polaires

Pour s'amuser !

Transformez les coordonnées du graphe suivant en coordonnées polaires (`coord_polar(theta = "y")`). Quel est le résultat ?

```
ggplot(fruits, aes(x = 1, fill = groupe)) +  
  geom_bar(width = 1) +  
  theme_void()
```

Avec les fonctions `*lim*`

Change le minimum et le maximum d'un axe. Attention, toutes les valeurs en dehors des nouveaux axes sont **éliminées** !

- `xlim`, `ylim` ou `lims` pour spécifier l'étendue,
- `expand_limits` pour *étendre* l'étendue à certaines valeurs.

Pour faire un “zoom” sans perdre de points, il faut utiliser la fonction `coord_cartesian` ou une fonction du type `scale_***`

Créer des “facettes”

La fonction `facet_wrap`

S'utilise pour diviser le graphe en panneaux selon les modalités d'une variable catégorielle.

Attention à la syntaxe : elle est basée sur l'utilisation du terme `vars`, qui permet d'accéder aux variables du jeu de données spécifié.

Par exemple, pour diviser le graphe `g` en plusieurs panneaux selon les modalités d'un facteur `fac`, on écrira

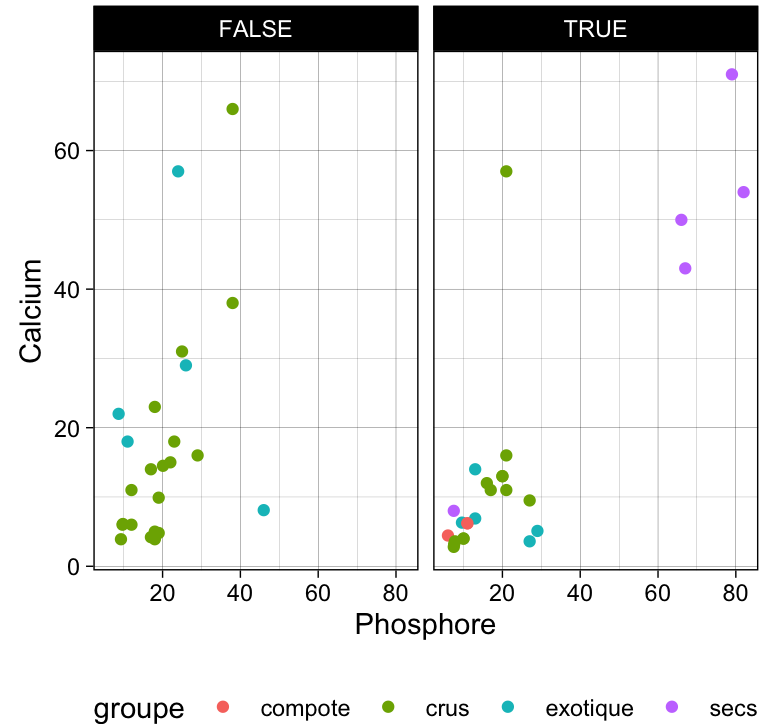
```
g + facet_wrap(facets = vars(fac))
```

On peut également utiliser une "formule" :

```
g + facet_wrap(~ fac)
```

Exemple

```
ggplot(fruits,  
      aes(x = Phosphore,  
          y = Calcium,  
          color = groupe)) +  
  geom_point() +  
  facet_wrap(vars(Sucres > 10)) +  
  theme_bw() +  
  theme(legend.position =  
        "bottom")
```



Avec la fonction `facet_grid`

S'utilise de la même façon que `facet_wrap`.

Par exemple, pour diviser le graphe `g` en plusieurs panneaux selon les modalités d'un facteur `factorow` pour les lignes et `factocol` pour les colonnes, on écrira

```
g + facet_grid(rows = vars(factorow), cols = vars(factocol))
```

On peut aussi utiliser une formule :

```
g + facet_grid(factorow ~ factocol)
```

CONSEIL : pour l'utilisation de facettes, faites attention à bien nommer les modalités de vos facteurs pour rendre le graphe plus clair.

Sauvegarder un graphe

Méthode conseillée : avec ggsave

Fonctionnement et exemple :

```
g <- ggplot(fruits, aes(groupe)) + geom_bar()  
ggsave(filename = "mongraphe.png", plot = g)
```

L'extension donnée dans `filename` sera magiquement détectée pour sauvegarder le fichier au bon format !

Conclusion

En quelques mots

Nous avons vu un package de représentations graphiques très puissant ! `ggplot2` fonctionne sur la base d'un canevas, de paramètres esthétiques comme la position en x, en y, les couleurs, la forme etc. A partir de ces paramètres, on va tracer les graphes à l'aides de *géométries*, qui peuvent se superposer !

Quelques remarques

1. `ggplot2` est très complet... et très complexe
2. Il faut prendre son temps, et savoir travailler avec ses petits jeux de données préférés bien connus et **très très très très propres** pour travailler sur son graphe,
3. Personnaliser un graphe prend du temps, il faut s'armer de patience...

Data Visualization

with ggplot2

Cheat Sheet



N'oubliez pas la feuille de triche !
