Visualisation

Vincent Guillemot
Amaury Vaysse

Avant toutes choses

Nous aurons besoin du package `RColorBrewer` :

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

```
library(RColorBrewer)
```

Nous allons également avoir besoin des données fruits :

```
data("fruits", package = "minidebuter")
```

Au programme

- Les couleurs, avec ou sans `RColorBrewer`
- La personnalisation avancée de ses `ggplot` favoris
- Les diagrammes de Venn avec `ggplot2`
- Les cartes de chaleur avec `pheatmap`

Trois façons de colorier des objets

Trois façons de voir les couleurs

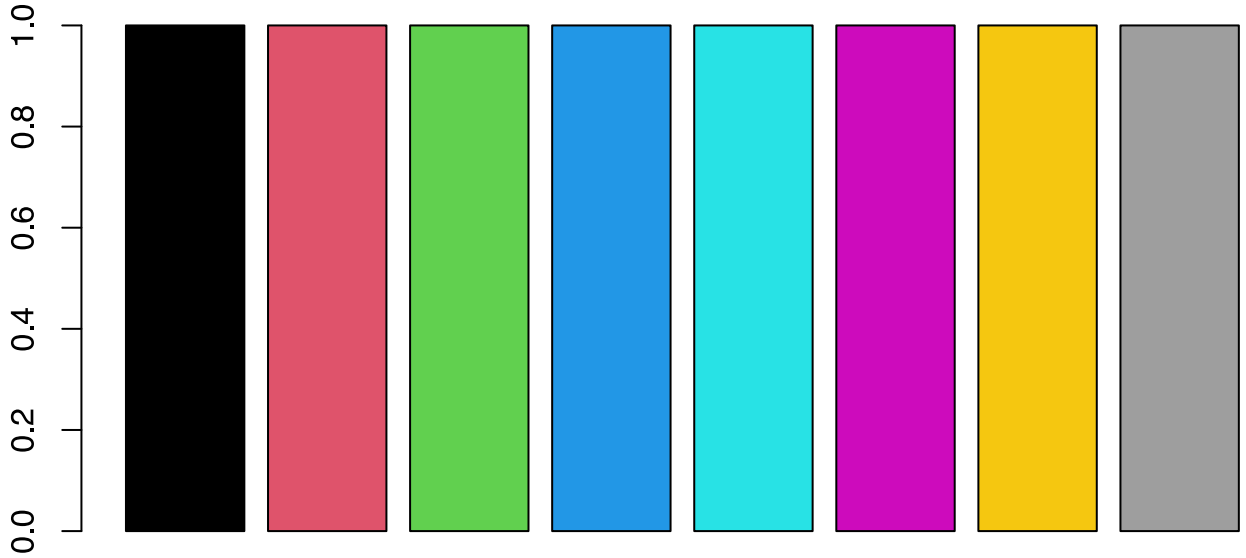
Voici trois manières de définir une couleur, qui permettent de colorier un graphe soit très rapidement (entiers), soit avec plus de possibilités (couleurs nommées) ou encore de manière très raffinée (HEX).

1	2	3	4	5	6	7	8
black	indianred2	palegreen3	dodgerblue2	turquoise	magenta3	darkgoldenrod1	gray62
#000000	#DF536B	#61D04F	#2297E6	#28E2E5	#CD0BBC	#F5C710	#9E9E9E

Les chiffres (rapide !)

La palette de 8 couleurs par défaut de R est codée par les entiers de 1 à 8.

```
barplot(rep(1,8), col = 1:8)
```



Les noms de couleurs (plus de couleurs)

On peut également colorier avec des “noms” de couleur (e.g. `"black"`, `"tomato"`, `"steelblue"`, `"darkorchid"` etc.)

On peut accéder à tous ces noms de couleur avec la commande `colors()` :

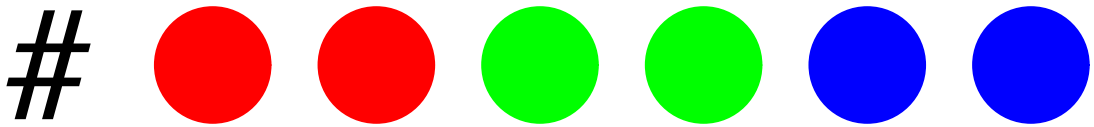
```
sample(colors(), 7)
#> [1] "cadetblue3"      "hotpink3"        "ivory2"
#> [4] "gray73"          "brown2"          "gray55"
#> [7] "mediumslateblue"
```

Ces couleurs “nommées” s’utilisent de la même façon que les couleurs “numériques”.

HEX (encore plus de couleurs !)

On peut aussi utiliser des codes hexadécimaux pour coder une couleur dans le système de référence “Rouge - Vert - Bleu” :

- deux symboles hexadécimaux par couleur ($16 \times 16 = 256$ valeurs possible)
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Trois couleurs de base : rouge, vert et bleu
- 00 = pas de cette couleur
- FF = max de cette couleur



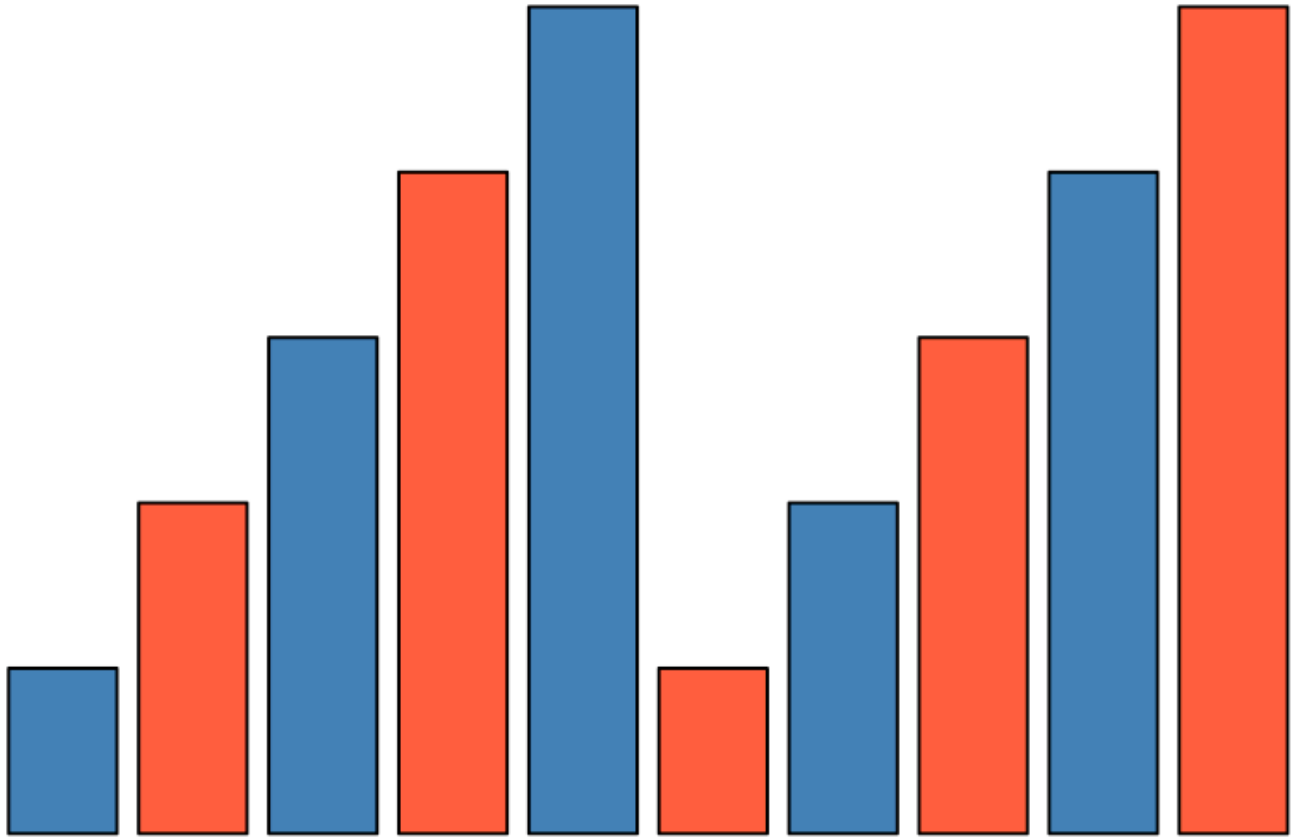
Exemple

Voici trois commandes équivalentes pour réaliser le même graphe ci-dessous :

```
barplot(rep(1, 3), col = 2:4)
barplot(rep(1, 3), col = c("indianred2", "palegreen3", "dodgerblue2"))
barplot(rep(1, 3), col = c("#DF536B", "#61D04F", "#2297E6"))
```

A vous !

Reproduisez le graphe ci-dessous avec le système de couleurs de votre choix :



ATTENTION !

Attention au recyclage !

S'il y a plus d'objets à colorier que de couleurs, les couleurs sont recyclées !

```
par(mar = c(0, 0, 0, 0))
barplot(rep(1,80), col = 1:8, border = NA, space = 0, axes = FALSE)
```

Les palettes

Utiliser des palettes

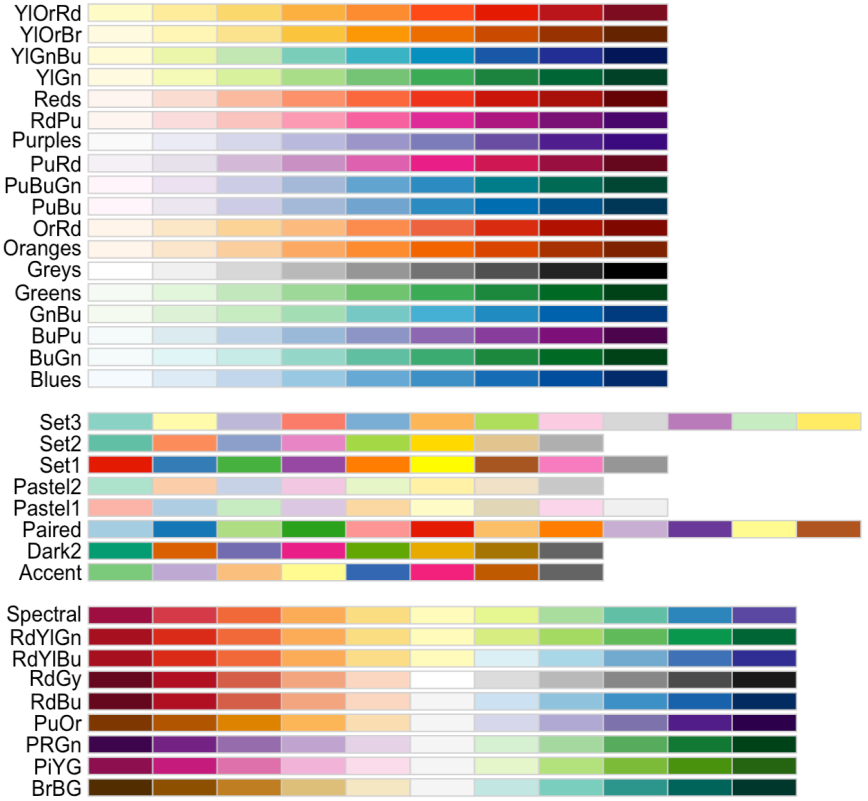
Il existe de nombreux packages en R permettant de générer des palettes de couleur. Nous allons en utiliser un seul pendant ce cours : `RColorBrewer`.

La commande suivante permet de visualiser toutes les palettes que ce package permet d'utiliser :

```
display.brewer.all()
```

Pour extraire des couleurs de ces palettes :

```
brewer.pal(n = 3, name = "Set3")
#> [1] "#8DD3C7" "#FFFFB3" "#BEBADA"
```



En quelques mots

Il y a trois types de palettes : séquentielles, divergentes et qualitatives.

1. Les palettes séquentielles permettent de distinguer des valeurs petites (en clair) de valeurs grande (en foncé)
2. Les palettes divergentes permettent de distinguer les valeurs petites des valeurs grandes, mais ces deux extrêmes sont toutes les deux de ton foncé dans des couleurs très différentes. Les valeurs intermédiaires sont représentées en ton clair.
3. Les palettes qualitatives sont faites pour distinguer toutes les couleurs les unes des autres. Elle sont adaptées à la représentation de données qualitatives.

A vous

Complétez le code suivant pour obtenir le graphe ci-contre :

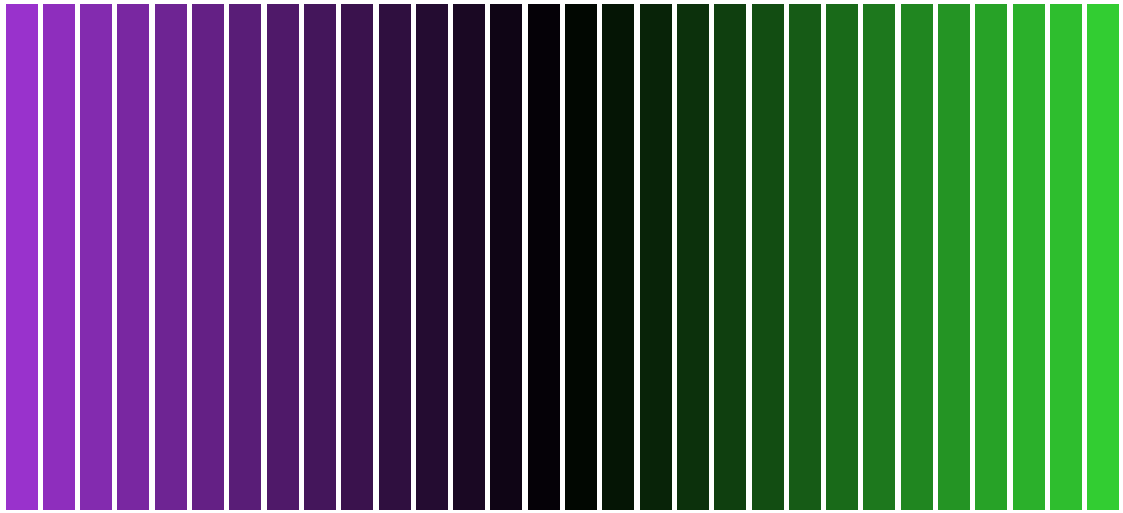
```
pal <- brewer.pal(***, ***)
barplot(rep(1, 7),
        col = pal,
        axes = ***,
        border = ***)
```

Extrapoler des couleurs

Pour extrapoler des couleurs…

On utilise la fonction de base `grDevices::colorRampPalette` :

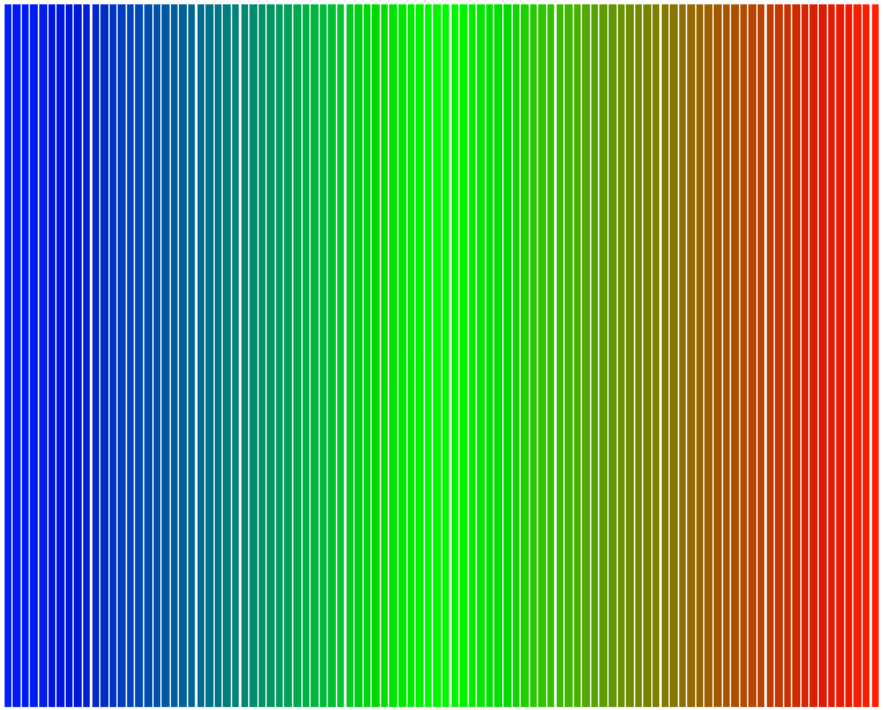
```
colfun <- colorRampPalette(c("darkorchid", "black", "limegreen"))
barplot(rep(1, 30), col = colfun(30), axes = F, border = NA)
```



Exercice

Reproduisez le graphe ci-contre en modifiant la commande ci-dessous.

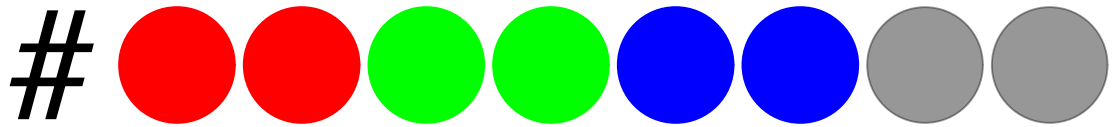
```
colfun <- colorRampPalette(
  c(***, ***, ***))
barplot(rep(1, 100),
        col = colfun(100),
        axes = F, border = NA)
```



Paramètre d'opacité

Dans `ggplot2`, l'opacité se gère avec `alpha` : 0 = invisible, 1 = opaque.

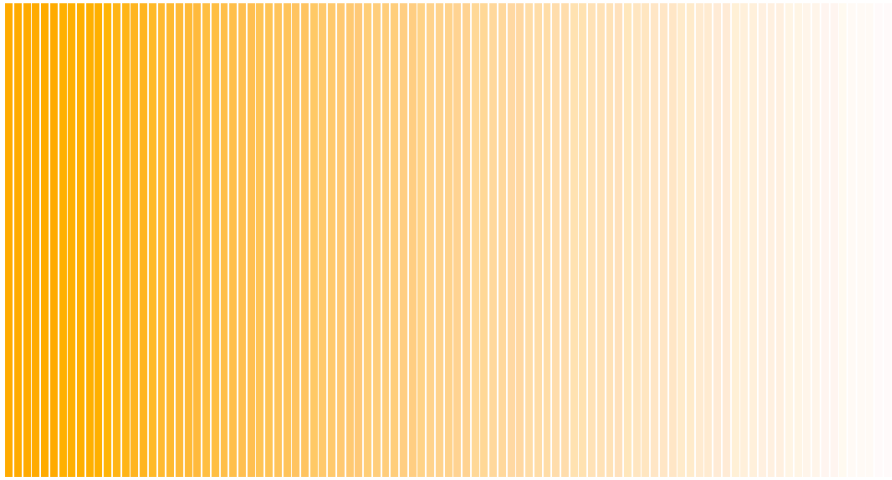
De manière générale : on peut opacifier sa couleur préférée avec le code HEX : on ajoute deux chiffres héxadécimaux **OPTIONNELS** pour régler l'opacité à la fin d'un code couleur.



Exercice

Reproduisez le graphe ci-contre en modifiant la commande ci-dessous.

```
colfun <- colorRampPalette(
  c("#FFAA00FF", ***),
  alpha = TRUE)
barplot(***,
        col = colfun(100),
        axes = F, border = NA)
```



Personnalisation de graphes `ggplot2`

Avant toutes choses

Nous aurons besoin du package `ggpubr` :

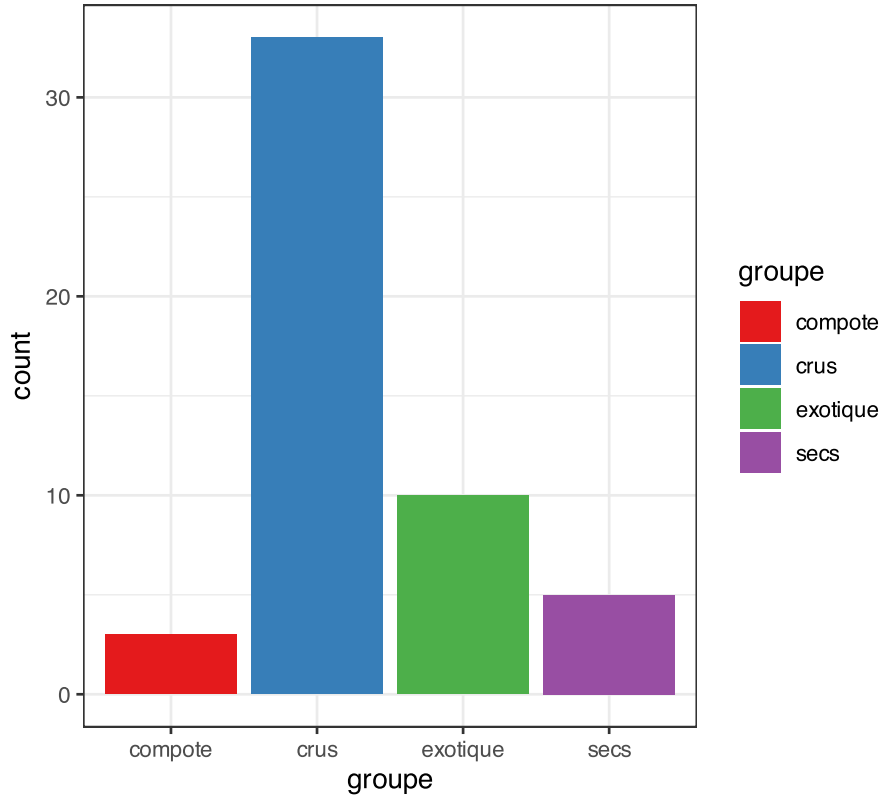
- Vérifier que le package `ggpubr` est bien installé
- Si non, l'installer, puis le charger

```
library(ggpubr)
```

Changer de palette pour un diagramme en bâtons

Avec la commande `scale_fill_brewer`

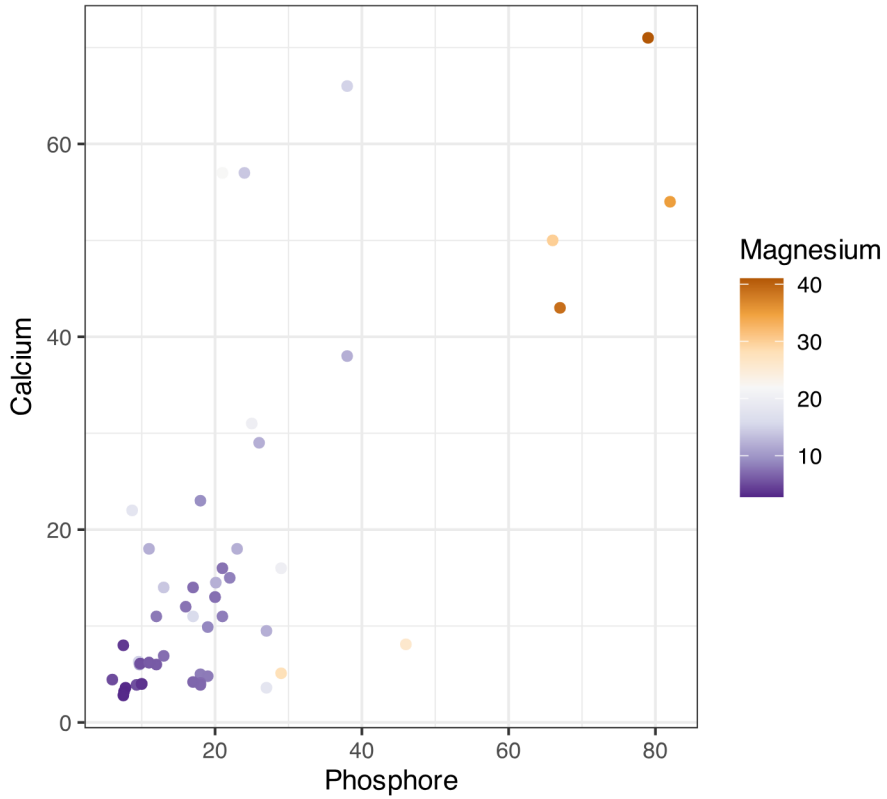
```
ggplot(fruits, aes(x = groupe,
                   fill = groupe)) +
  geom_bar() +
  scale_fill_brewer(palette = "Set1") +
  theme_bw()
```



Changer de palette pour un nuage de points

Avec la commande `scale_color_distiller`

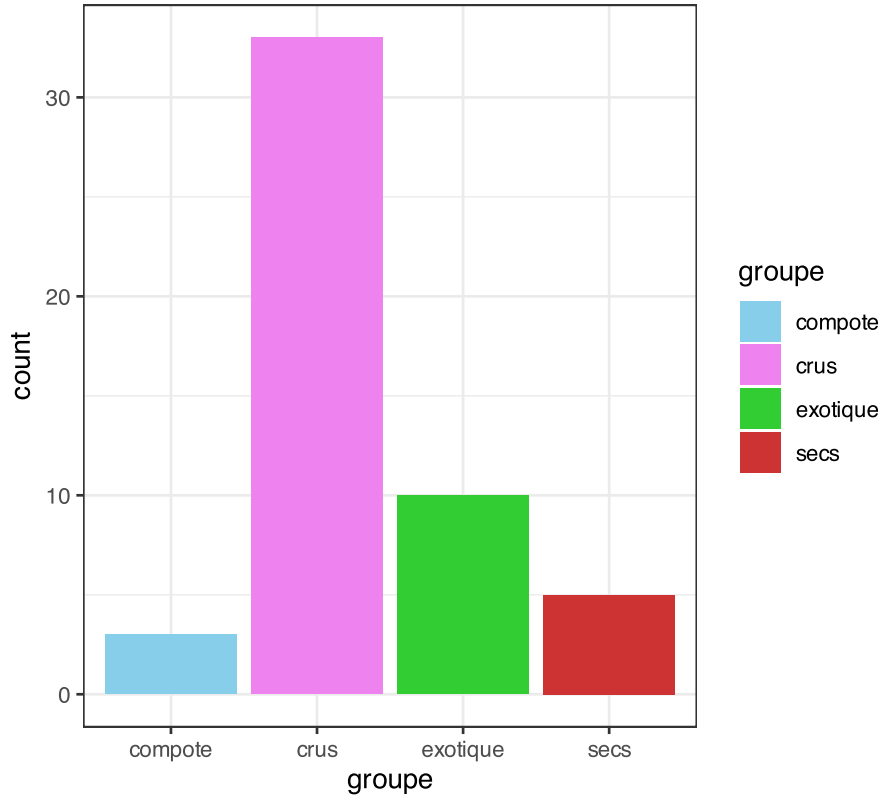
```
ggplot(fruits, aes(x = Phosphore,
                   y = Calcium,
                   color = Magnesium)) +
  geom_point() +
  scale_color_distiller(palette = "PuOr") +
  theme_bw()
```



Personnaliser les couleurs d'un diagramme en bâtons

Avec la commande `scale_fill_manual`

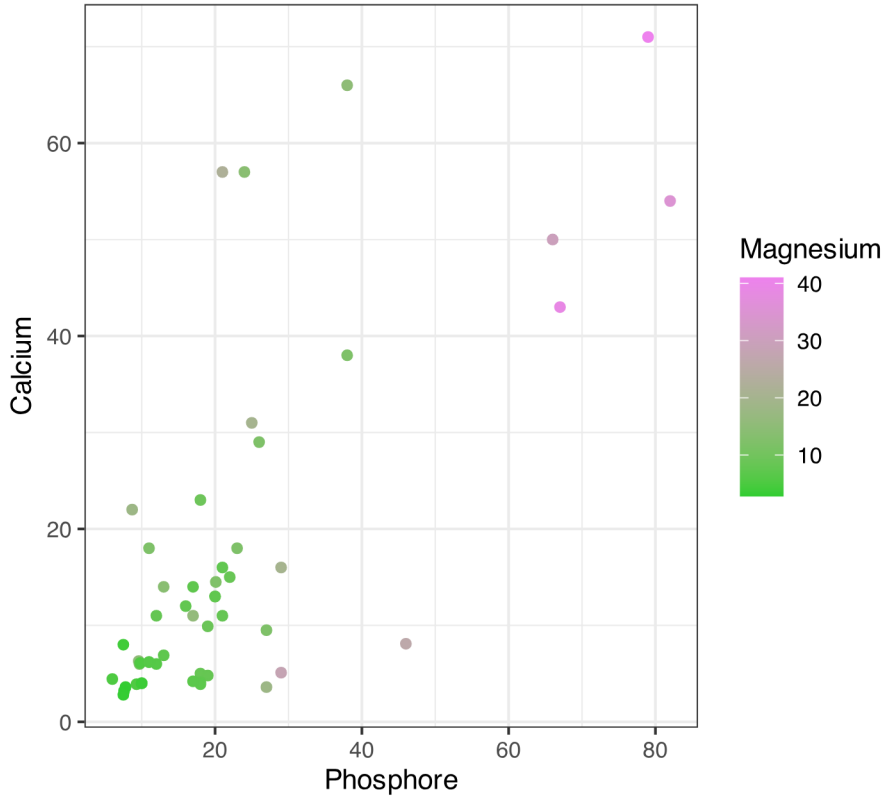
```
ggplot(fruits, aes(x = groupe,
                   fill = groupe)) +
  geom_bar() +
  scale_fill_manual(
    values = c(crus = "violet",
               secs = "brown3",
               exotique = "limegreen",
               compote = "skyblue")) +
  theme_bw()
```



Personnaliser les couleurs d'un nuage de points

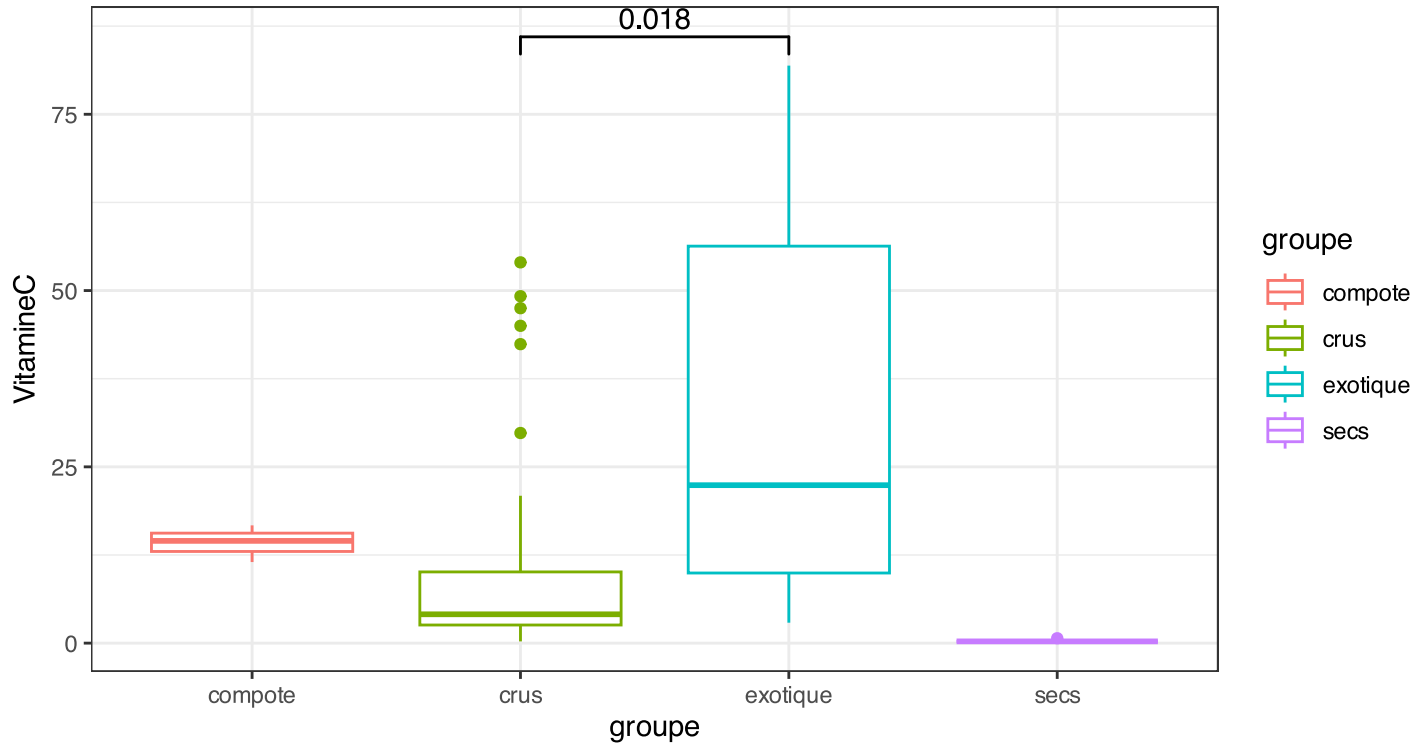
Avec la commande `scale_color_gradient`

```
ggplot(fruits, aes(x = Phosphore,
                   y = Calcium,
                   color = Magnesium)) +
  geom_point() +
  scale_color_gradient(
    low = "limegreen",
    high = "violet") +
  theme_bw()
```



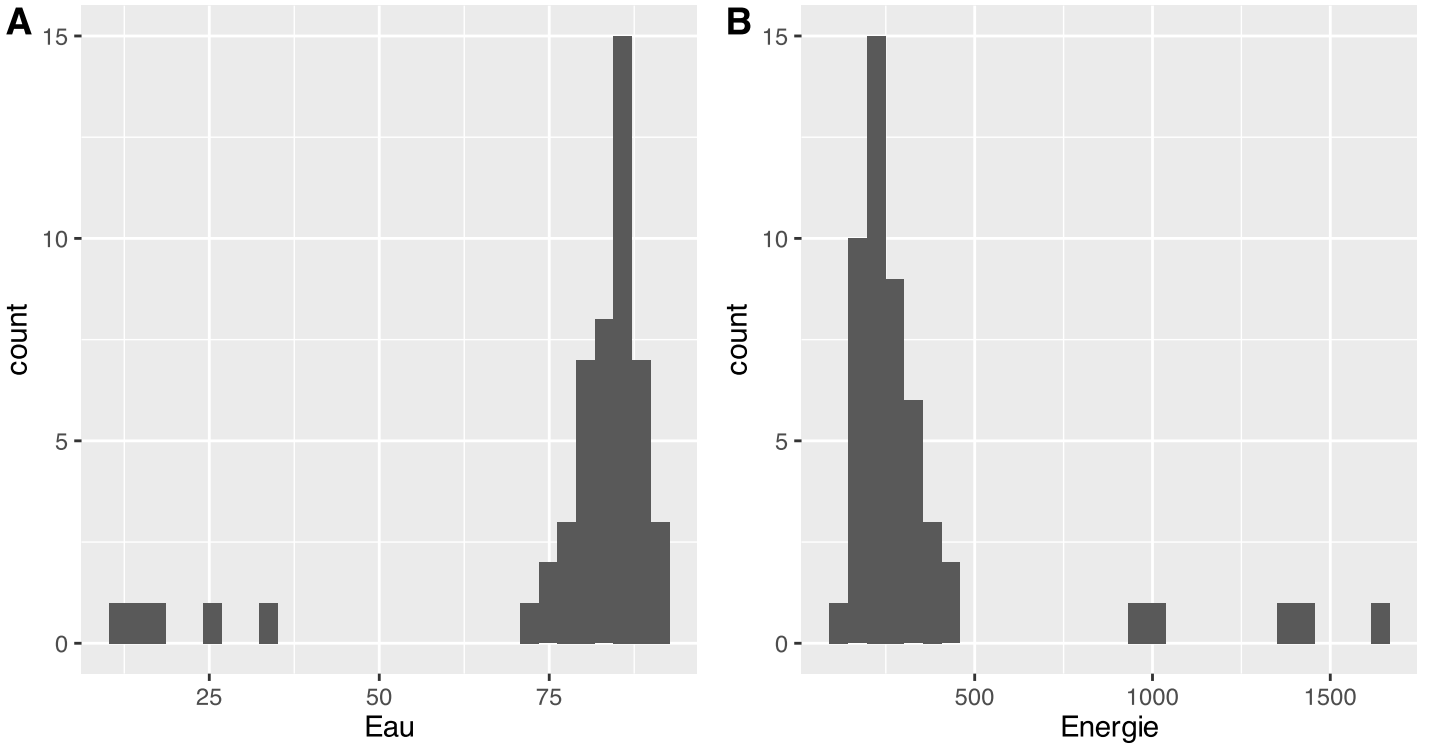
Ajouter des p-valeurs

```
ggplot(fruits, aes(groupe, VitamineC)) +
  geom_boxplot(aes(color = groupe)) +
  geom_signif(comparisons = list(c("crus", "exotique"))) +
  theme_bw()
```



Mosaïque de graphes

```
g1 <- ggplot(fruits, aes(Eau)) + geom_histogram()
g2 <- ggplot(fruits, aes(Energie)) + geom_histogram()
ggarrange(g1, g2, labels = "AUTO")
```



Diagrammes de Venn avec **ggvenn**

Avant toutes choses

Nous aurons besoin du package `ggvenn` :

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

```
library(ggvenn)
```

Visualiser des relations entre listes

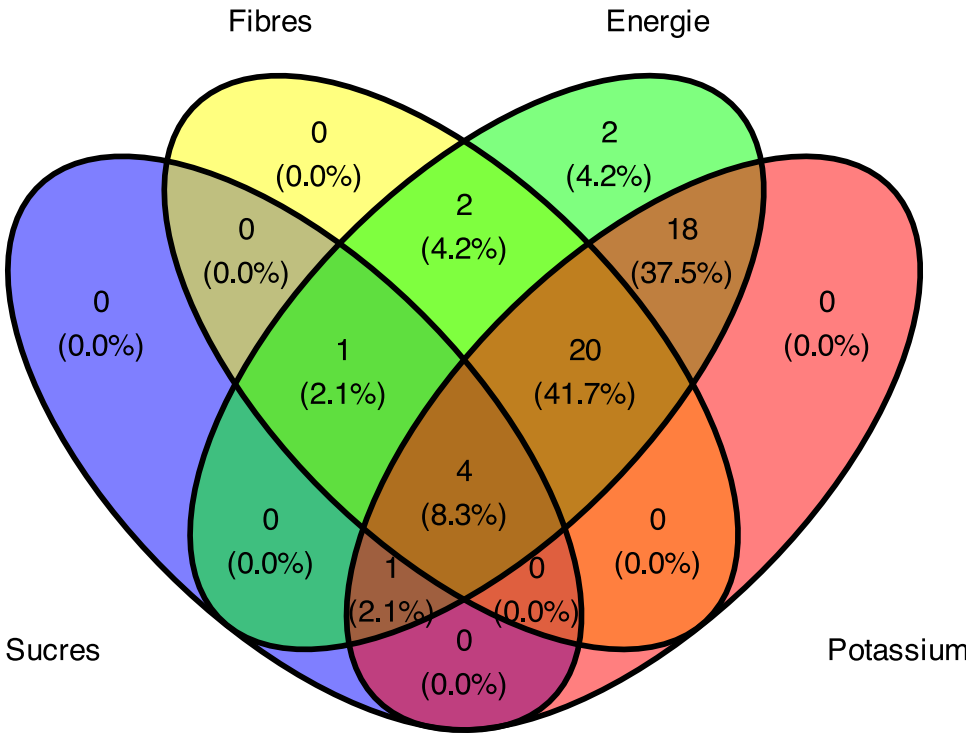
Créons une liste d'objets :

```
flist <- list(
    Sucres = fruits$nom[fruits$Sucres > 20],
    Fibres = fruits$nom[fruits$Fibres > 2],
    Energie = fruits$nom[fruits$Energie > 50],
    Potassium = fruits$nom[fruits$Potassium > 100]
)
```

Diagramme de Venn

Compliqués à lire à partir de 4 ensembles :

```
ggvenn(flist, set_name_size = 4)
```



Sauvegarder un diagramme de Venn

En utilisant la (très pratique) fonction `ggsave` :

```
g <- ggvenn(flist, set_name_size = 4)
ggsave(filename = "fruit_venn.pdf", plot = g)
```

Pour le personnaliser…

Voir l'aide de la fonction `?ggvenn`

- `fill_color`, `fill_alpha` pour les couleurs à l'intérieur des cercles,
- `stroke_color`, `stroke_alpha`, `stroke_size`, `stroke_linetype` pour le contour des cercles,
- `set_name_color`, `set_name_size` pour les noms des ensembles,
- `text_color`, `text_size` pour le texte à l'intérieur des intersections

Upset plots avec **UpSetR**

Avant toutes choses

Nous aurons besoin du package `UpSetR` :

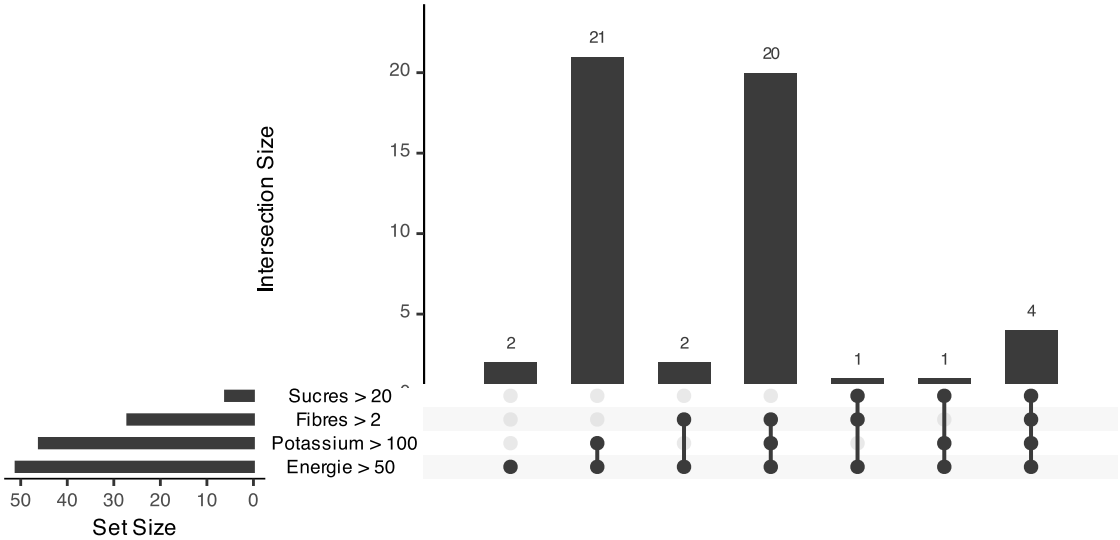
- Vérifier que le package `UpSetR` est bien installé
- Si non, l'installer, puis le charger

```
library(UpSetR)
```

On reprend les 4 ensembles

On reprend l'exemple avec la comparaison des fruits selon 4 critères.

```
mat01 <- data.frame(
    "Sucres > 20" = fruits$Sucres > 20,
    "Fibres > 2" = fruits$Fibres > 2,
    "Energie > 50" = fruits$Energie > 50,
    "Potassium > 100" = fruits$Potassium > 100,
    check.names = FALSE) + 0
upset(mat01)
```



Cartes de chaleur avec `pheatmap`

Avant toutes choses

Nous aurons besoin du package `pheatmap` :

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

```
library(pheatmap)
```

Premier essai

```
pheatmap(fruits)
```

```
Error in hclust(d, method = method) :
  NA/NaN/Inf dans un appel à une fonction externe (argument 10)
De plus : Warning messages:
1: In dist(mat, method = distance) :
  NAs introduits lors de la conversion automatique
2: In dist(mat, method = distance) :
  NAs introduits lors de la conversion automatique
```

Pourquoi ça ne fonctionne pas ?

Deuxième essai : C'est déjà mieux ?

```
pheatmap(fruits[, -(1:2)])
```

Les arguments

- `cluster_rows = FALSE` : enlever le dendrogramme sur les lignes
- `scale = "column"` : pour standardiser les variables
- `show_rownames = FALSE` : pour cacher les noms des lignes
- `cellwidth = 10` : pour avoir des plus petites cellules

Pour avoir une liste complète des arguments : `?pheatmap`

Troisième essai

```
pheatmap(
  fruits[, -(1:2)],
  cluster_rows = FALSE,
  scale = "column",
  show_rownames = FALSE,
  cellwidth = 10
)
```

Quatrième essai : changer les couleurs

```
colfun <- colorRampPalette(
  c("darkorchid",
    "white",
    "limegreen"))

pheatmap(
  fruits[, -(1:2)],
  cluster_rows = FALSE,
  scale = "column",
  show_rownames = FALSE,
  cellwidth = 10,
  color = colfun(20)
)
```

Cinquième essai : ajouter des informations “qualitatives”

```
colfun <- colorRampPalette(
  c("darkorchid",
    "white",
    "limegreen"))
fruitsDF <- data.frame(
  fruits[,-1],
  row.names = make.unique(fruits$nom))
annotLignes <- fruitsDF[, "groupe",
                        drop = FALSE]
```

```
pheatmap(
  fruitsDF[, -1],
  cluster_rows = FALSE,
  scale = "column",
  show_rownames = FALSE,
  cellwidth = 10,
  color = colfun(20),
  annotation_row = annotLignes
)
```

A vous !

Changez la commande suivante pour obtenir un joli graphe.

```
pheatmap(
  t(fruits),
  scale = "row",
  color = c("black", "black"),
  legend_breaks = c(-6, 0,+6),
  border_color = "pink",
  cellheight = 100,
  cellwidth = 0.1,
  show_colnames = "FALSE"
)
```