



# Atelier 3: Introduction à ggplot2

Série d'ateliers R du CSBQ

Centre de la Science de la Biodiversité du Québec



# À propos de cet atelier

 REPO

 DEV

 WIKI

03



DIAPOS

03



DIAPOS

03

 SCRIPT

03

# Packages requis

- grid
- gridExtra
- ggplot2
- ggsignif
- ggdendro
- maps
- mapproj
- RColorBrewer
- GGally
- patchwork
- plotly

```
install.packages(c('grid', 'gridExtra', 'ggplot2', 'ggsignif', 'ggdendro', 'maps', 'mapproj', 'RColorBrewer', 'GGally', 'patchwork', 'plotly'))
```

# Objectifs d'apprentissage

1. Utilisez R pour créer différents graphiques

# Introduction

# Introduction

## Pour suivre cet atelier:

Code et .HTML sont disponibles à l'adresse suivante:

<http://qcbs.ca/wiki/r/workshop3>

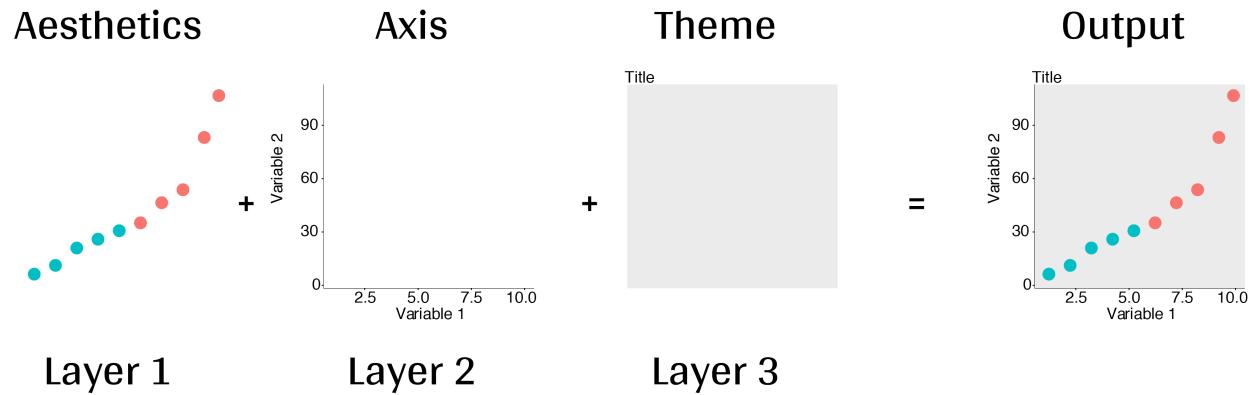
## Recommandations:

1. créez votre propre script R ;
2. référez-vous au script R fourni uniquement si nécessaire ;
3. évitez de copier-coller ou d'exécuter le code directement à partir du script.

`ggplot2` est aussi sur GitHub: <https://github.com/tidyverse/ggplot2>

# Plan de l'atelier

## 1. La mécanique `ggplot2`



## 2. Visualisation avancée

## 3. Ajustement et peaufinage

## 4. Enregistrer un graphique

## 5. Conclusion

# Objectifs d'apprentissage

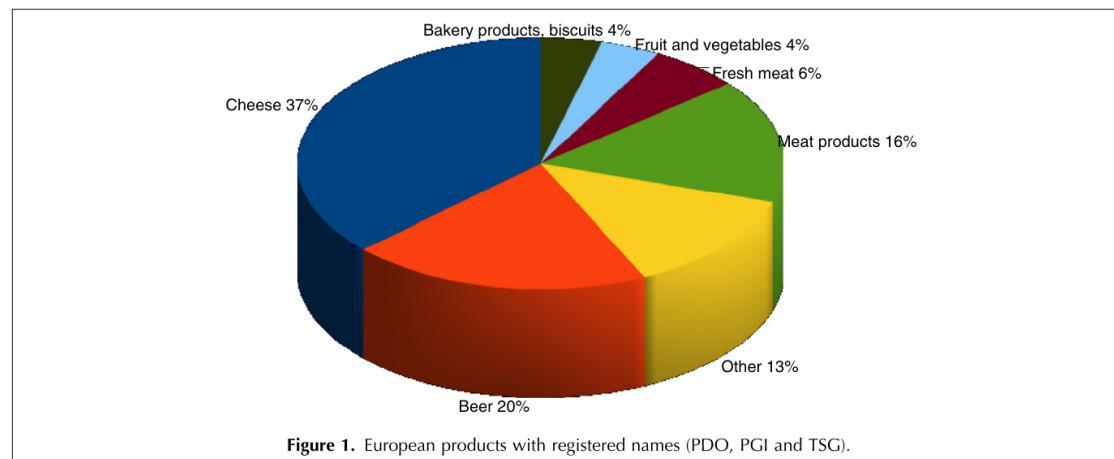
- Apprendre les bases de la visualisation des données à l'aide de R.
- Trouver des librairies et des ressources pour répondre à vos besoins.
- Inspirer plus de créativité dans les domaines scientifiques !
- Développer votre compréhension de la conception pour une communication graphique efficace.

# Visualisation en science

# Visualisation en science

Pourquoi utilisons-nous la visualisation de données ?

Qu'est-ce qui rend une visualisation efficace ?



Que pensez-vous de celle-ci ?

# Visualisation en science

1. Représenter les résultats des analyses statistiques
2. Formuler des hypothèses et comprendre/résumer des théories
3. Explorer vos propres données (analyse exploratoire, détection des valeurs aberrantes)
4. Communiquer et rendre compte
  - Clairement (en utilisant de bons principes de conception)
  - De manière précise et exacte (un bon graphique vaut 1000 mots)
  - De manière effective et efficace

# Visualisation en science

## Questions importantes :

- Que voulez-vous communiquer ?
- Qui est votre public ?
- Quelle est la meilleure façon de le faire ?

**Une règle générale : restez simple - utilisez moins d'encre !**

## Ressources utiles

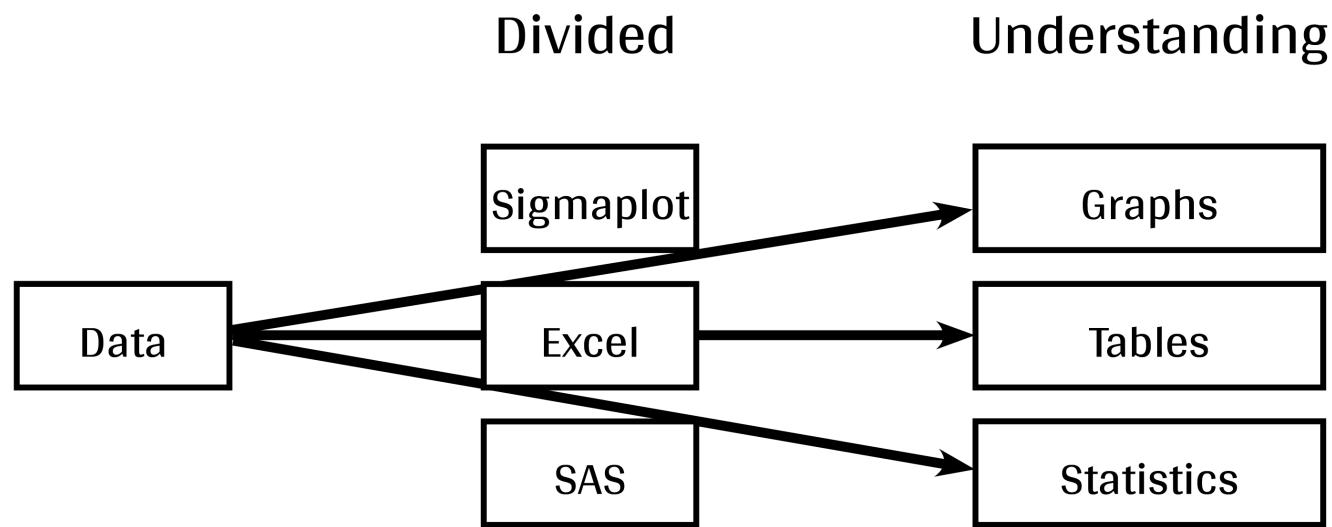
- [Fundamentals of Data Visualization](#) (ggplot)
- [A Compendium of Clean Graphs in R](#) (base plot)
- [Graphics Principles](#) (trucs et astuces de conception)

# AVERTISSEMENT!

 **n'est pas fait pour le dessin.**

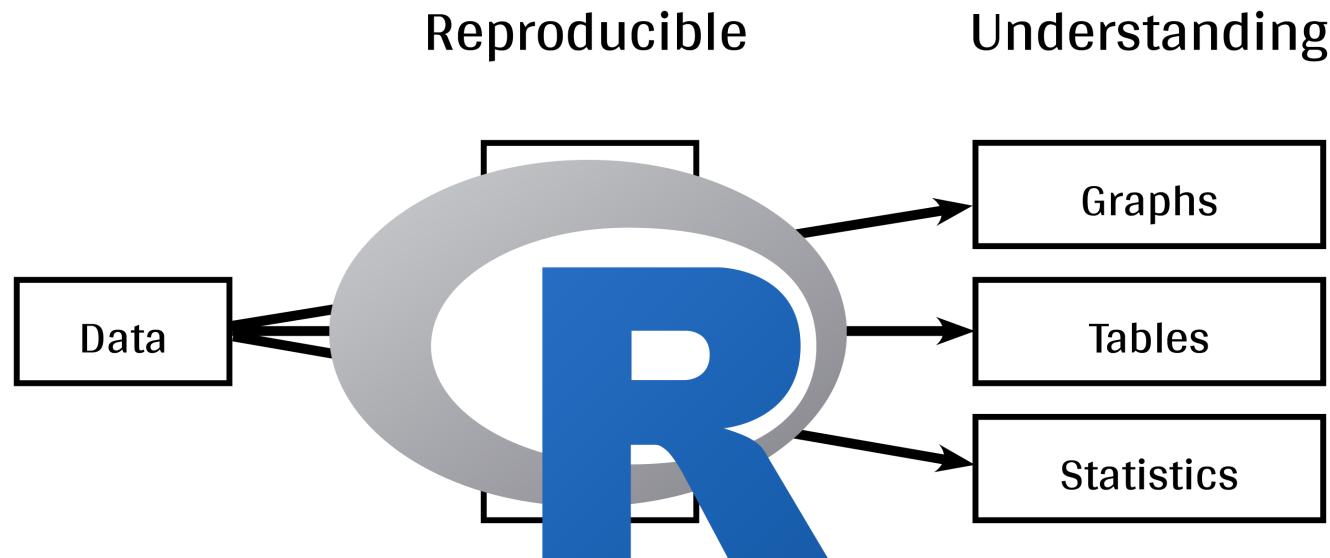
D'autres logiciels de dessin sont probablement de meilleures options comme [GIMP](#) ou [Inkscape](#). Il est important d'utiliser le bon outil pour la bonne tâche !

# Pourquoi utiliser R pour les graphiques ?



# Pourquoi utiliser R pour les graphiques ?

## Reproductibilité



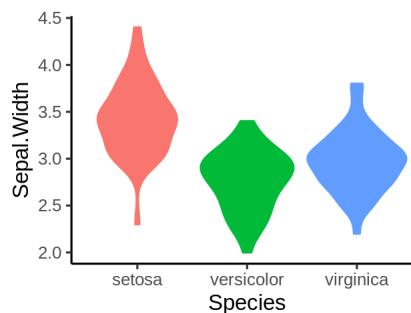
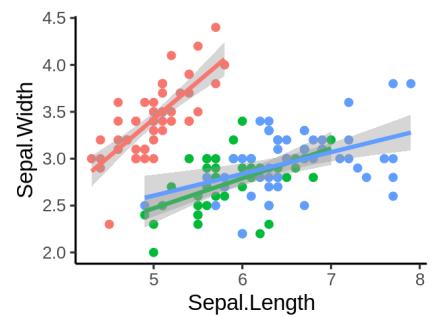
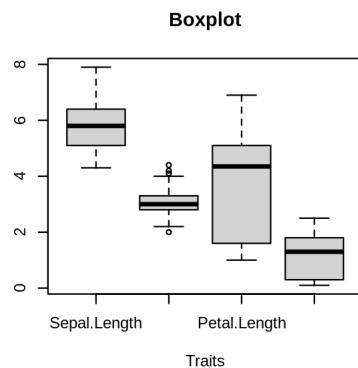
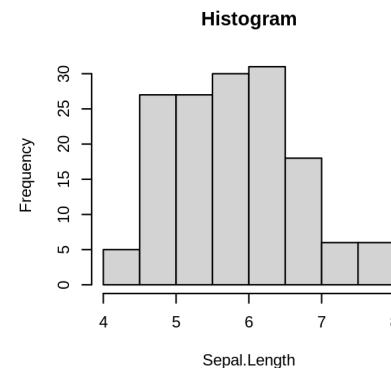
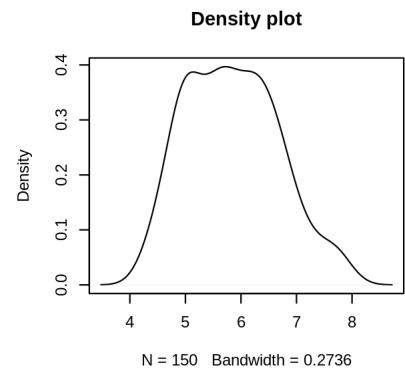
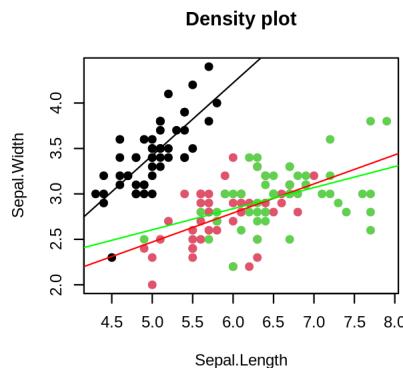
### La science reproduicible requiert un effort:

- commentez votre script
- ajoutez des informations dans vos figures (titres, étiquettes, légendes, annotations)

# Pourquoi utiliser R pour les graphiques ?

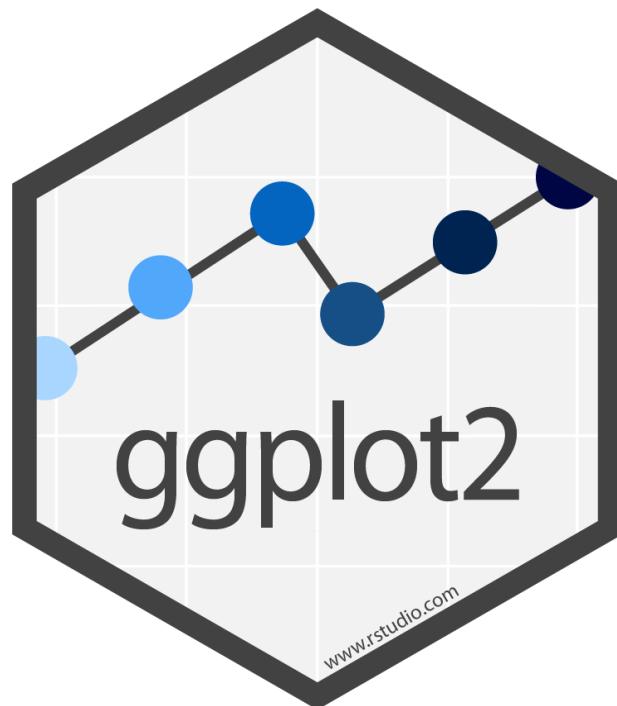
## En raison de ses puissantes fonctionnalités !

Dans cet atelier, nous nous concentrons uniquement sur `ggplot2`, mais plusieurs librairies et fonctions peuvent être utilisées pour une excellente visualisation (par exemple, "base R", plotly, sjPlot, mapview, igraph).



# `ggplot2` est polyvalent

1. La librairie `ggplot2` vous permet de réaliser de *beaux graphiques* personnalisables;
2. `ggplot2` met en œuvre la grammaire des graphiques, un système fiable pour construire des graphiques;
3. `ggplot2` a de [nombreuses extensions](#).



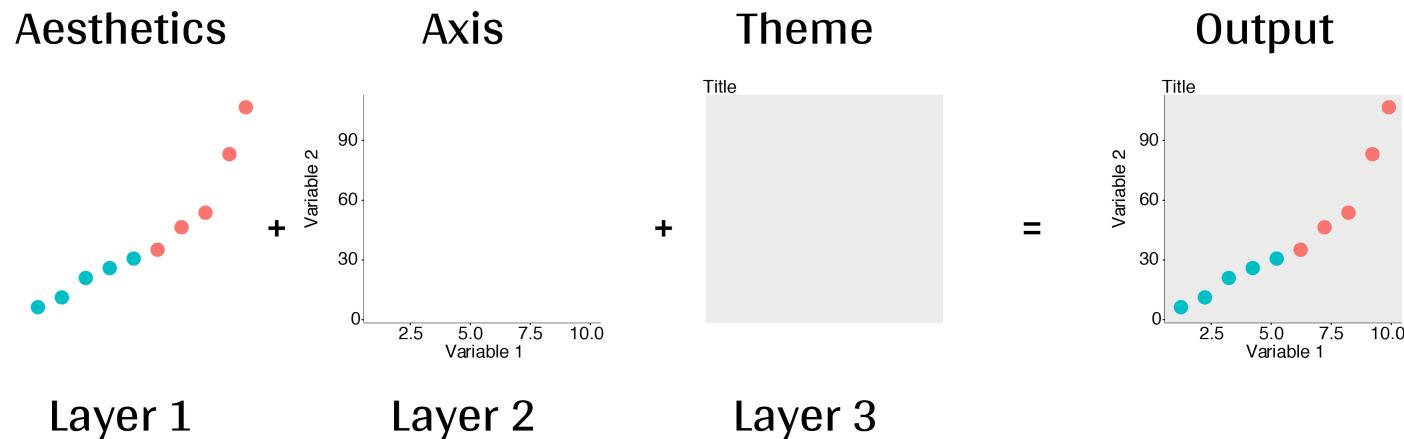
[www.rstudio.com](http://www.rstudio.com)

# La mécanique `ggplot2`: les bases

# Les bases de la grammaire des graphiques (GG)

```
install.packages("ggplot2") # if not already installed  
library(ggplot2)
```

Un graphique est construit à partir de différentes couches:



En utilisant le système GG, nous pouvons construire des graphiques étape par étape pour des résultats personnalisables.

# Les bases de la grammaire des graphiques (GG)

Ces couches ont des noms spécifiques que vous verrez tout au long de la présentation :

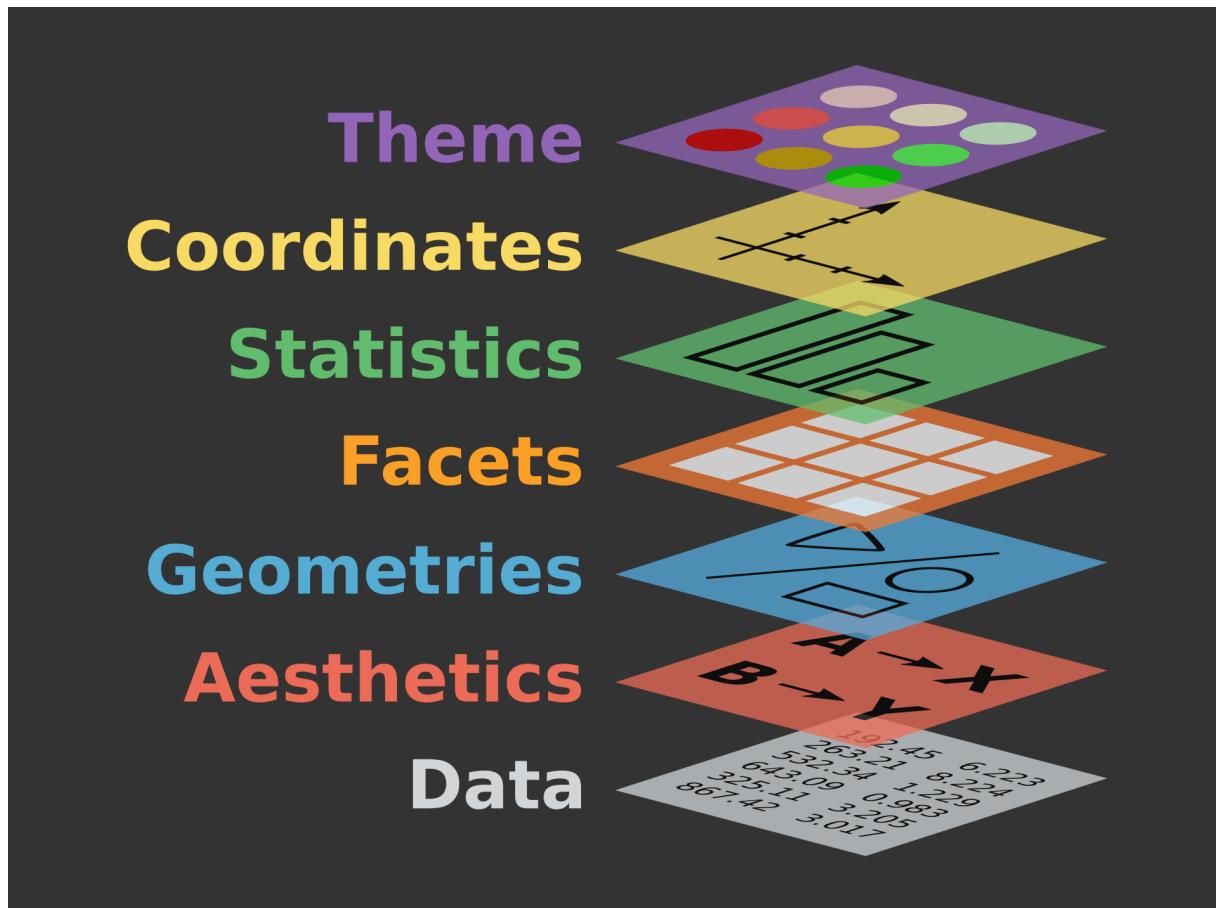


image adaptée de [The Grammar of Graphics](#)

# Les bases de la grammaire des graphiques (GG)

Voici les éléments de base pour dessiner le plus simple des `ggplot`:

```
ggplot(data = <DATA>) +  
  <GEOM function>(mapping=aes(<mappings>),  
    stat = <STAT>, position = <POSITION> ) +  
  <COORDINATE function> +  
  <SCALE function> +  
  <THEME function> +  
  <FACET function> +...
```

Required

Not required

# La grammaire des graphiques (GG)

Un graphique est constitué d'éléments (couches):

- Données
  - vos données, dans un format bien ordonné, fourniront les ingrédients de votre graphique
  - utiliser les techniques `dplyr` pour préparer les données en vue d'un format de tracé optimal
  - en général, une ligne pour chaque observation que vous voulez représenter

# La grammaire des graphiques (GG)

Un graphique est constitué d'éléments (couches):

- Données
- Esthétique (aes), pour rendre les données visibles
  - `x`, `y`: position le long des axes x et y
  - `colour`: la couleur des géométries selon les données
  - `fill`: la couleur intérieure des géométries
  - `group`: à quel groupe appartient une géométrie
  - `shape`: la forme des points
  - `linetype`: le type de ligne utilisé (pleine, pointillée, etc.)
  - `size`: la taille des points ou des lignes
  - `alpha`: la transparence des géométries

**note: l'esthétique est basée sur les quantités dans vos données, mais les graphiques peuvent également avoir certaines de ces qualités qui ne sont pas fonction des données (p. ex. la coloration basée sur un groupe de données est une esthétique, mais les points peuvent avoir une couleur qui n'est pas fonction des données).**

# La grammaire des graphiques (GG)

Un graphique est constitué d'éléments (couches):

- Données
- Esthétique (aes)
- Objets géométriques (geoms)
  - `geom_point()` : diagramme de dispersion
  - `geom_line()` : lignes reliant des points par une valeur croissante de x
  - `geom_path()` : lignes reliant des points dans l'ordre d'apparition
  - `geom_boxplot()` : diagramme en boîte (boxplot) pour les variables catégoriques
  - `geom_bar()` : diagrammes à barres pour un axe des x catégorique
  - `geom_histogram()` : histogramme pour l'axe des x continu
  - `geom_violin()` : kernel de distribution de la dispersion des données
  - `geom_smooth()` : ligne de lissage en fonction des données

# La grammaire des graphiques (GG)

Un graphique est constitué d'éléments (couches):

- Données
- Esthétique (aes)
- Objets géométriques (geoms)
- Facettes
  - `facet_wrap()` ou `facet_grid()` pour de petits multiples

# La grammaire des graphiques (GG)

Un graphique est constitué d'éléments (couches):

- Données
- Esthétique (aes)
- Objets géométriques (geoms)
- Facettes
- Statistiques
  - similaire aux géométries, mais issue de calcul sur les données
  - indique les moyennes, les comptages et autres résumés statistiques des données

# La grammaire des graphiques (GG)

Un graphique est constitué d'éléments (couches):

- Données
- Esthétique (aes)
- Objets géométriques (geoms)
- Facettes
- Statistiques
- Coordonnées - ajustement des données sur une page
  - `coord_cartesian` pour fixer des limites
  - `coord_polar` pour les graphiques circulaires
  - `coord_map` pour différentes projections cartographiques

# La grammaire des graphiques (GG)

Un graphique est constitué d'éléments (couches):

- Données
- Esthétique (aes)
- Objets géométriques (geoms)
- Facettes
- Statistiques
- Coordonnées
- Thèmes
  - Effets visuels globaux par défaut
  - polices, couleurs, formes, contours

# Comment fonctionnent les couches dans ggplot

1. Créer un objet graphique simple :
  - `plot.object <- ggplot()`
2. Ajouter des couches géométriques :
  - `plot.object <- plot.object + geom_*( )`
3. Ajouter des couches d'apparence :
  - `plot.object <- plot.object + coord_*( ) + theme()`
4. Répétez l'étape 2-3 jusqu'à ce que vous soyez satisfait, puis imprimez :
  - `plot.object` ou `print(plot.object)`.

# Préparer les données pour `ggplot2`

La librairie `ggplot2` requiert de préparer les données comme un objet de la classe "data.frame" ou "tibble" (commun dans l'univers `tidyverse`).

```
library(tibble)
class(iris) # Tout est prêt !
# [1] "data.frame"

ir <- as_tibble(iris) # acceptable
class(ir)
# [1] "tbl_df"     "tbl"        "data.frame"
```

## ♻ Rappel de l'atelier [Charger et manipuler des données](#):

Les graphiques plus complexes dans les `ggplot2` nécessitent que les données soient en format long

# Le jeu de données iris

```
## ?iris  
str(iris)  
# 'data.frame': 150 obs. of 5 variables:  
# $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...  
# $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...  
# $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...  
# $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...  
# $ Species     : Factor w/ 3 levels "setosa", "versicolor", ... 1 1 1 1 1 1 1 1 1 1
```

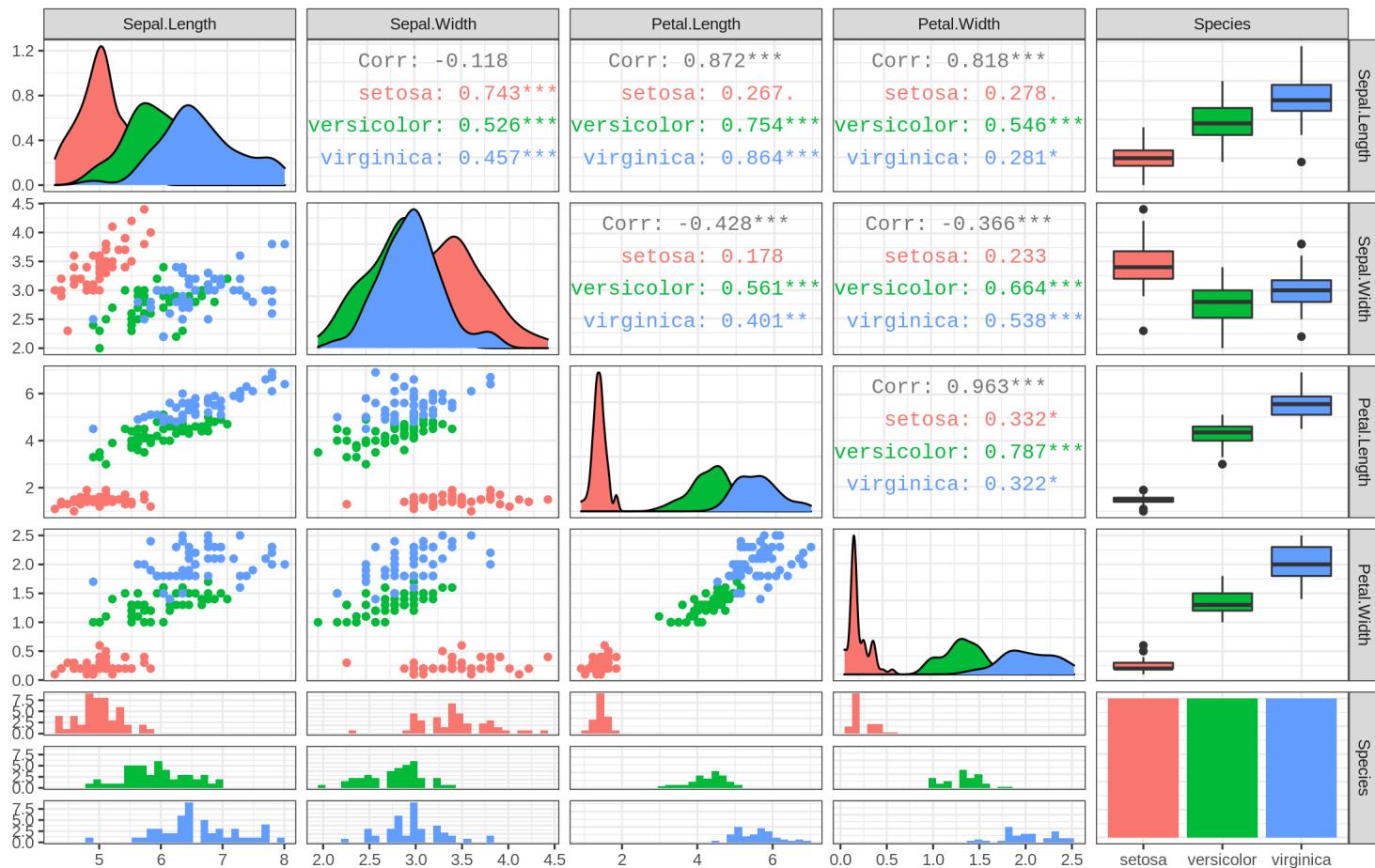
## Questions scientifiques

- Y a-t-il une relation entre la **longueur** et la **largeur** des **sépales** des iris ?
  - Est-ce que la taille des **pétales** et des **sépales** varie ensemble ?
  - Comment ces mesures sont-elles réparties entre les 3 espèces d'iris ?

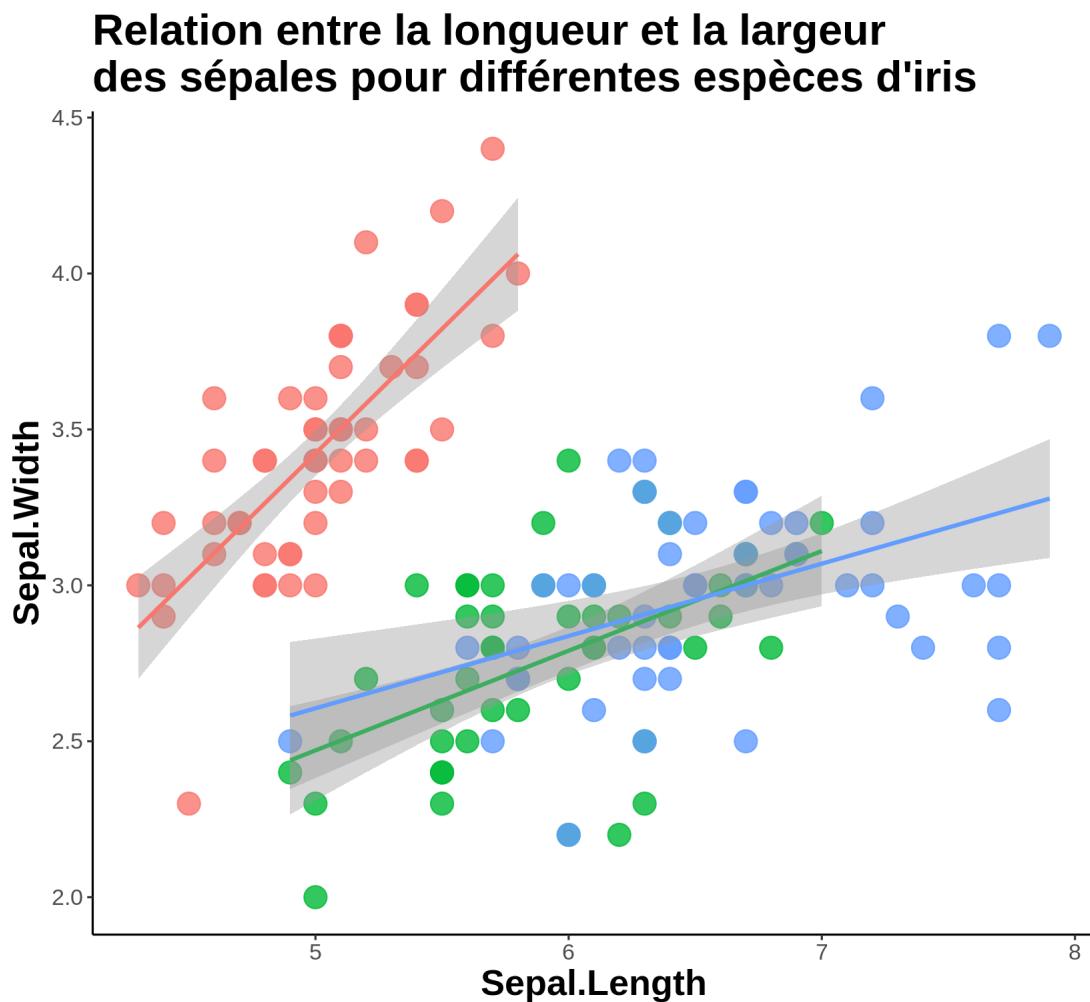
## Comment pouvons-nous répondre graphiquement à ces questions avec ggplot?

# Explorer la structure des données

```
install.packages("GGally")
library(GGally)
ggpairs(iris, aes(colour = Species)) + theme_bw()
```



# Explorer la structure des données

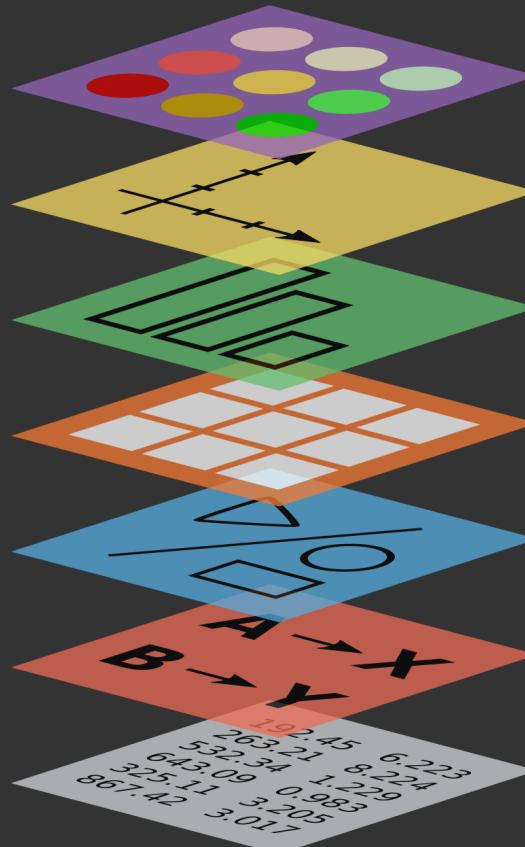


♻️ Voir l'atelier [Charger et manipuler des données](#) pour apprendre comment nettoyer vos données.

# La grammaire des graphiques: rappel

Un graphique est constitué de différentes couches :

Theme  
Coordinates  
Statistics  
Facets  
Geometries  
Aesthetics  
Data



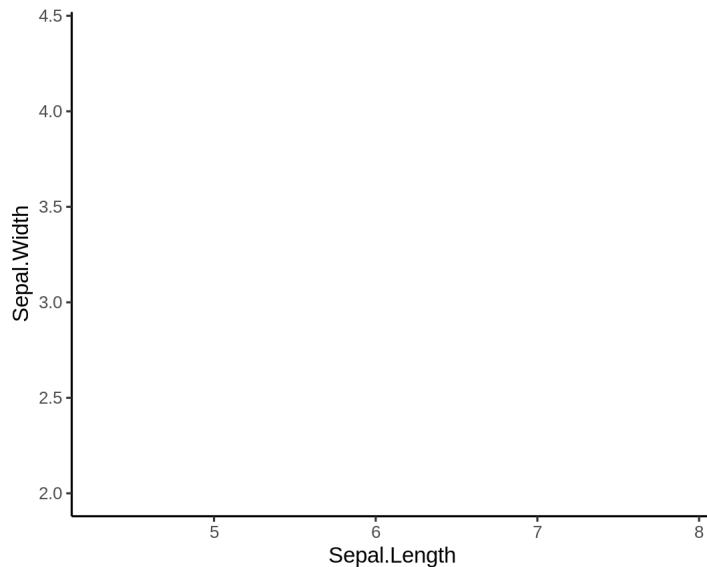
# La dynamique `ggplot()` : la couche de données

```
ggplot(data = iris)
```



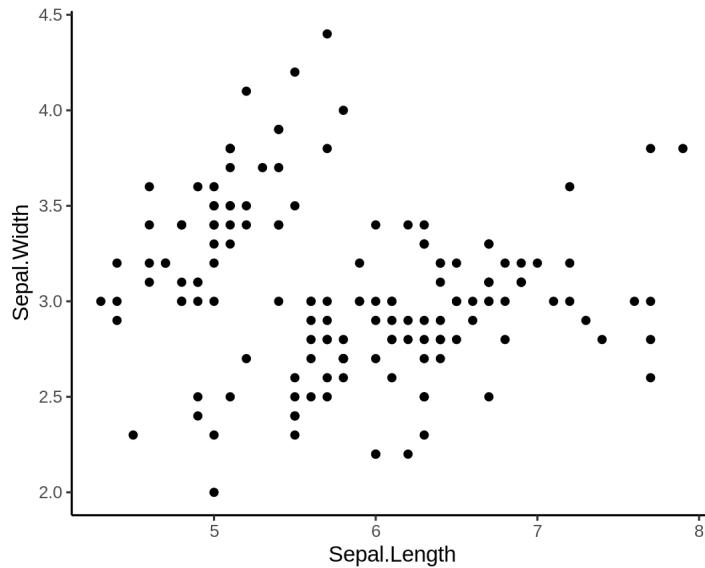
# La dynamique `ggplot()` : la couche d'esthétiques

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width))
```



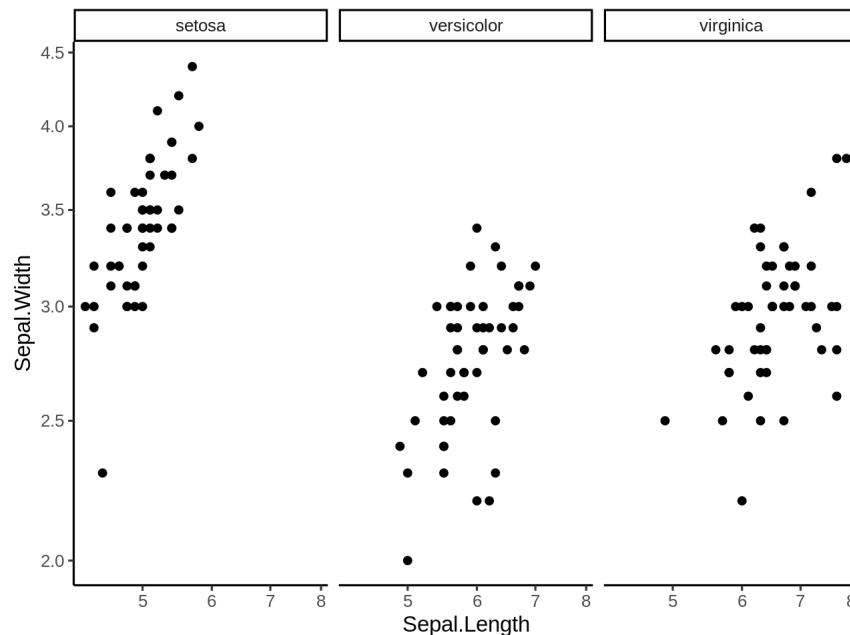
# La dynamique `ggplot()` : la couche de géométries

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point()
```



# # La dynamique `ggplot()` : les couches extras - facette, stats, coordonnées

```
ggplot(data = iris, aes(x = Sepal.Length, y = Sepal.Width)) +  
  geom_point() +  
  facet_wrap(~Species) +  
  coord_trans(x = "log10",  
              y = "log10")
```



# Défi #1 (5min)



## Créer votre 1er graphique ggplot!

### Question

Y a-t-il une relation entre la **longueur** et la **largeur** des **pétales** des iris?

La **largeur** des pétales augmente-t-elle avec leur **longueur**?

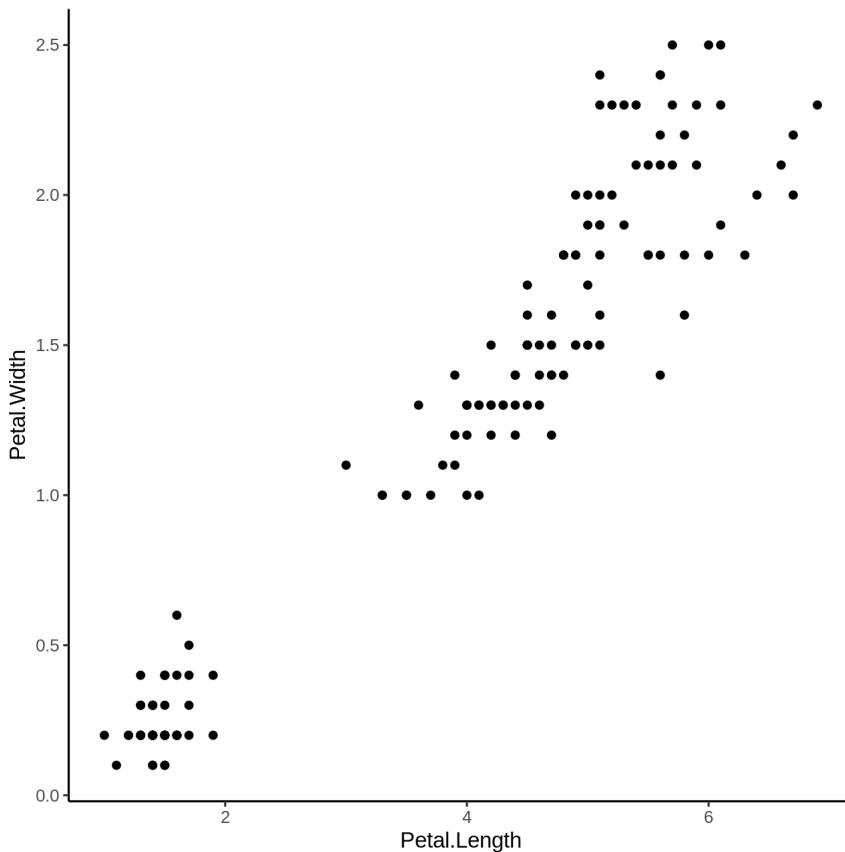
Paramètres à prendre en compte pour répondre à cette question :

données	géométrie	valeurs x	valeurs y
iris	geom_point	longueur des pétales	largeur des pétales



# Défi 1#: Solution

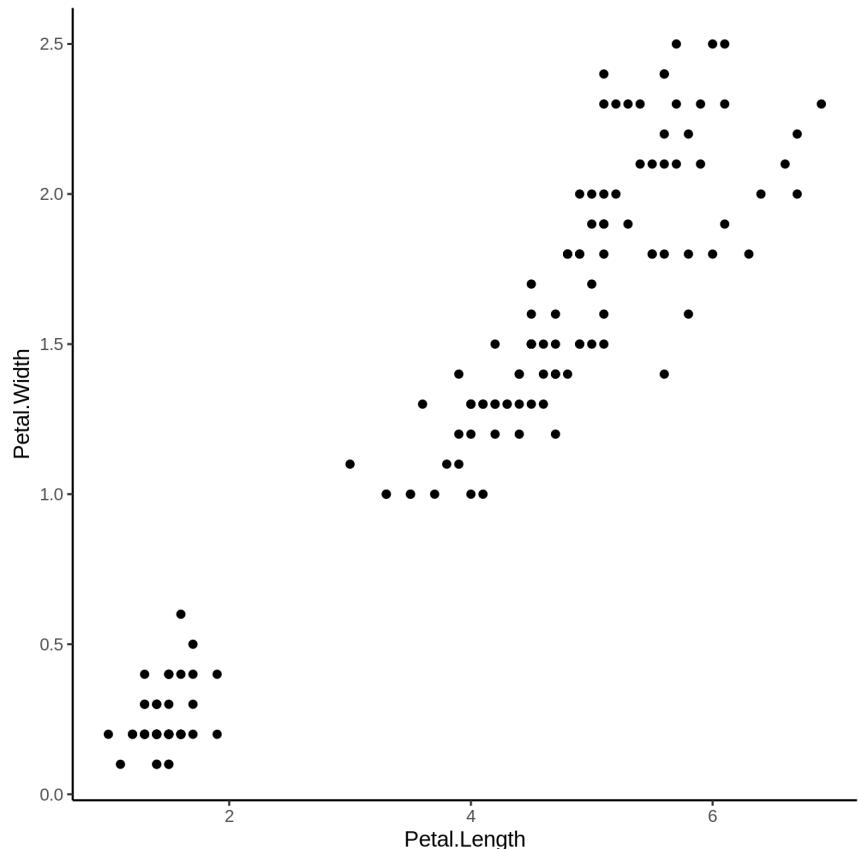
```
ggplot(data = iris, aes(x = Petal.Length,  
                      y = Petal.Width)) +  
  geom_point()
```





# Défi 1#: Solution

```
ggplot(data = iris, aes(x = Petal.Length,  
                        y = Petal.Width)  
       geom_point())
```

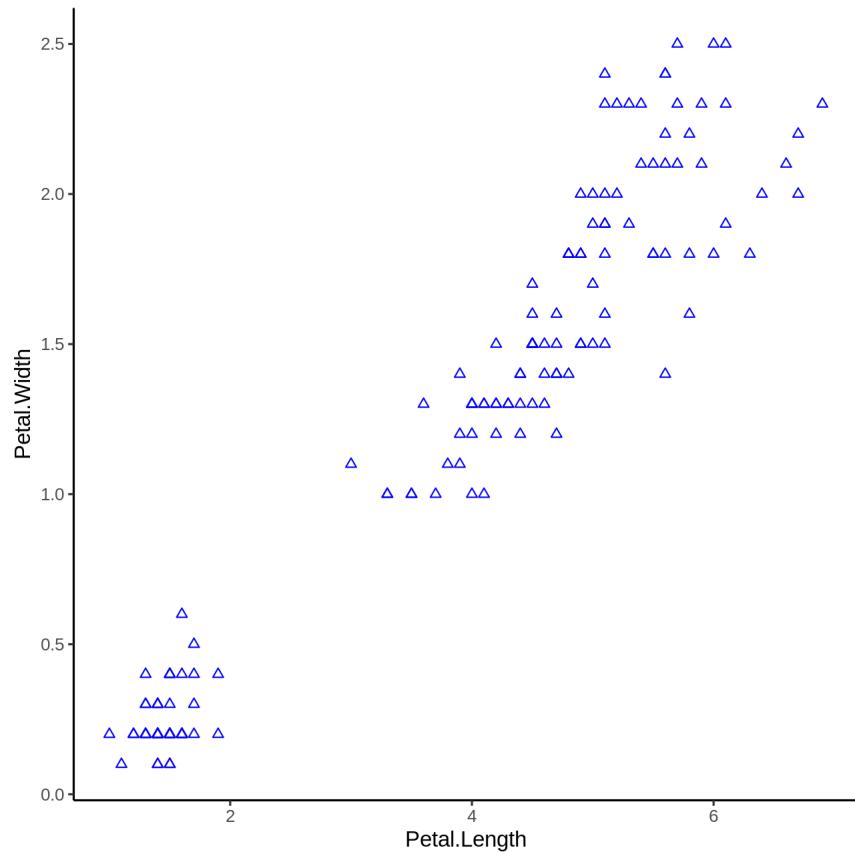


**note:** L'esthétique peut être soit dans la ligne `ggplot()`, et sera héritée par chaque géométrie, soit dans la ligne `geom_*`() à appliquer à cette géométrie seulement !



# Défi 1#: Solution

```
ggplot(data = iris, aes(x = Petal.Length,  
                        y = Petal.Width,  
                        geom_point(shape = 2, color = "blue"))
```



**note:** L'esthétique peut être soit dans la ligne `ggplot()`, et sera héritée par chaque géométrie, soit dans la ligne `geom_*`() à appliquer à cette géométrie seulement !

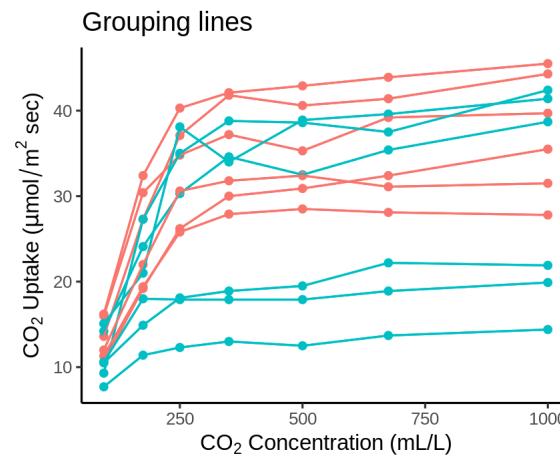
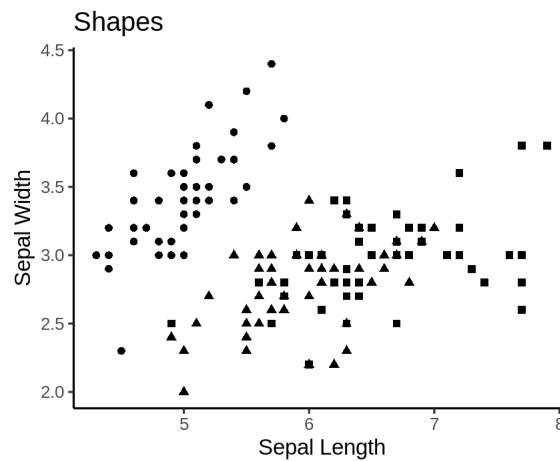
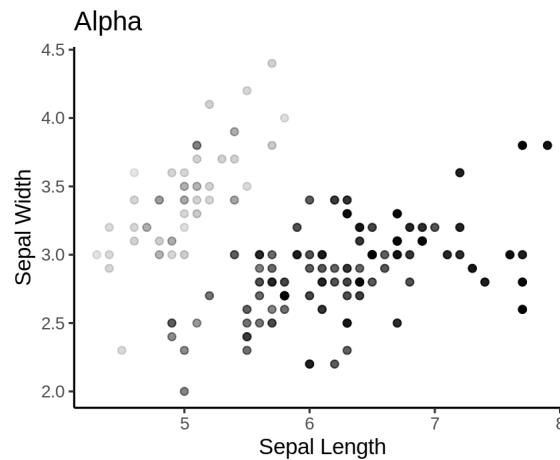
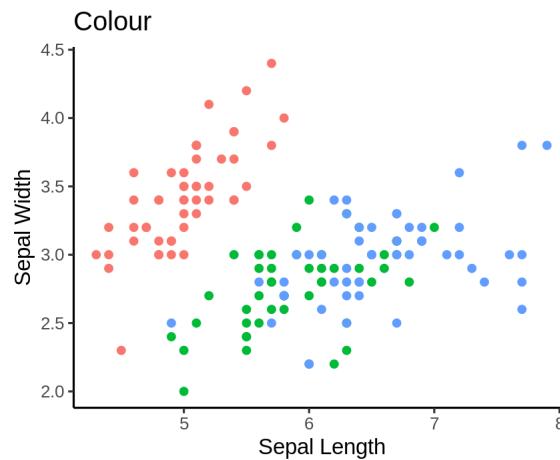
**note 2:** les commandes `colour`, `alpha`, `shape`, et `size` peuvent être définies en dehors des valeurs de `aes()`, et seront alors statiques, non dépendantes des données.

**Esthétique ou *Aesthetic mappings***

**couleur, forme, taille, étiquettes et transparence**

# Esthétique

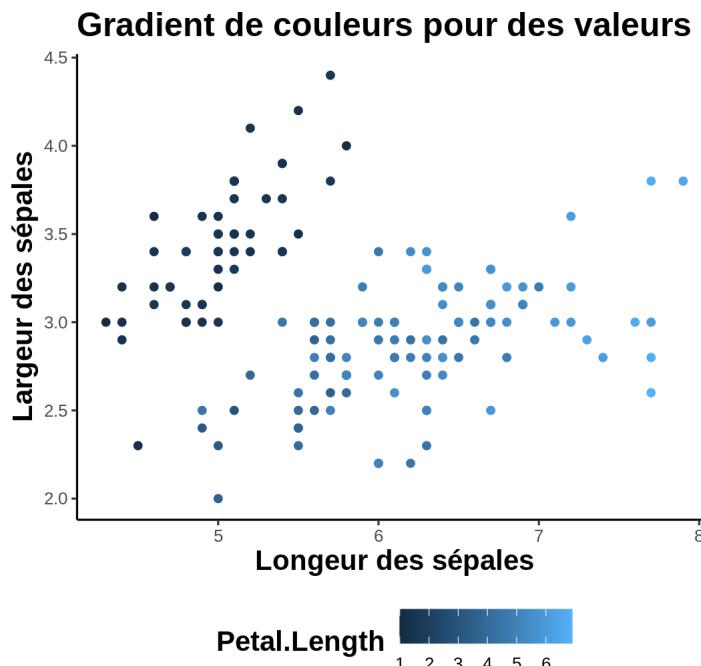
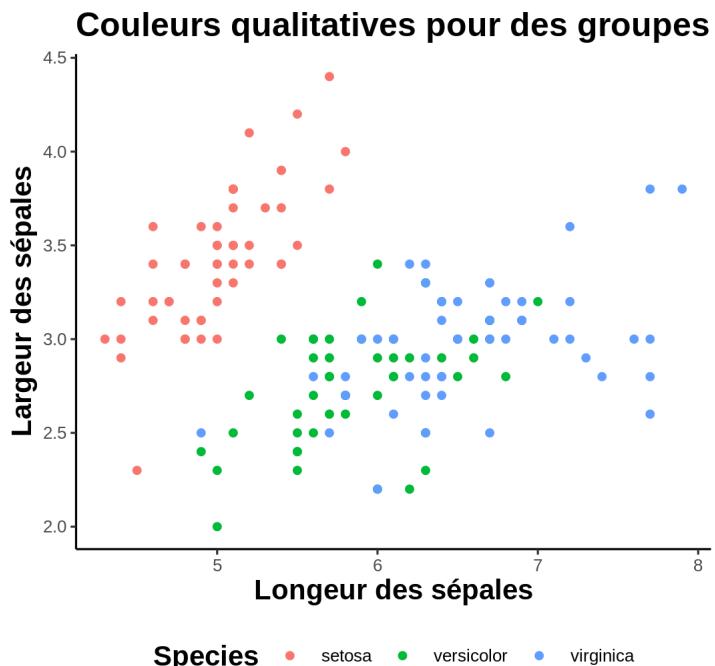
Utiliser l'esthétique (`aes()`) pour distinguer les classes, les groupes et la structure



# Les couleurs: faire parler vos données

Changez la **couleur** pour

- ➔ différencier les groupes
- ➔ représenter les valeurs des données
- ➔ mettre en évidence des éléments spécifiques

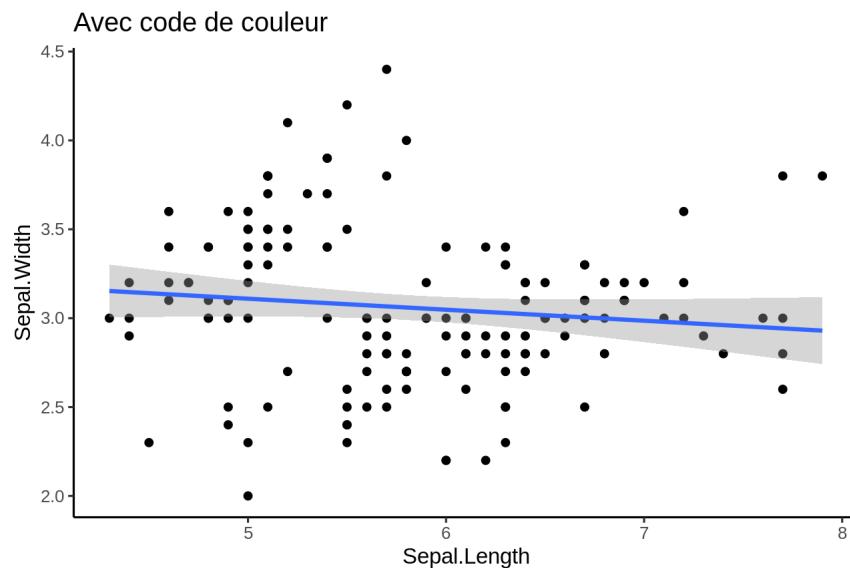


Voir [Fundamentals of Data Visualization](#)

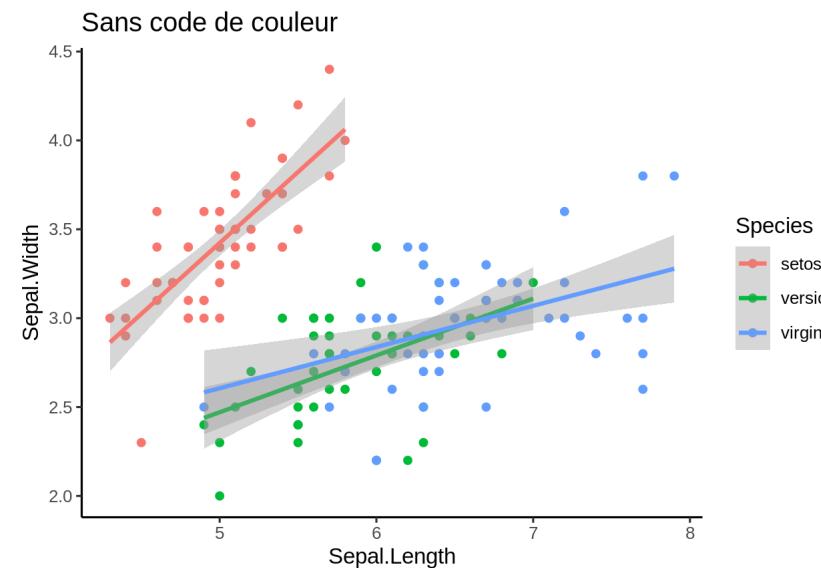
# Utiliser `aes()` pour changer les couleurs

La longueur et la largeur des sépales varient-elles selon les espèces ?

```
ggplot(data = iris,  
       aes(x = Sepal.Length, y = Sepal  
geom_point() +  
geom_smooth(method = lm)+  
labs(title = "Avec code de couleur")
```

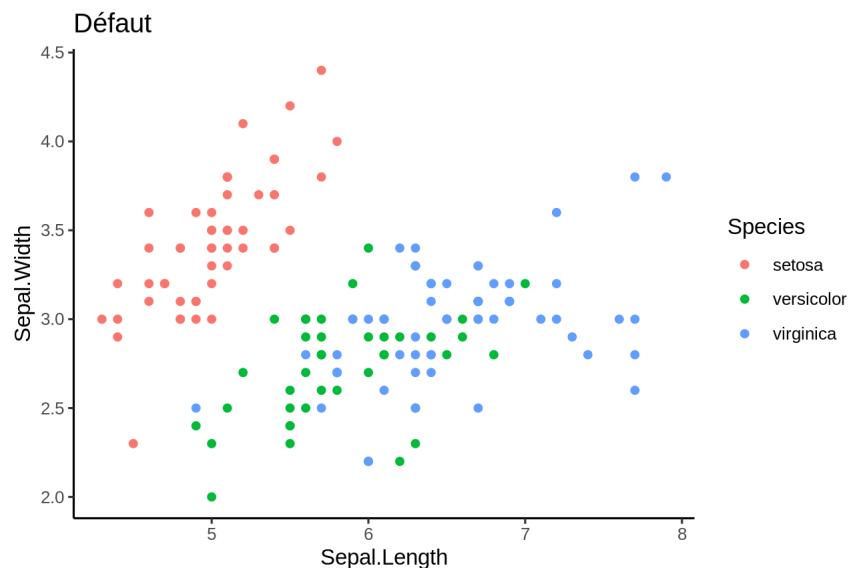


```
ggplot(data = iris,  
       aes(x = Sepal.Length, y = Sepal  
geom_point() +  
geom_smooth(method = lm) +  
labs(title = "Sans code de couleur")
```

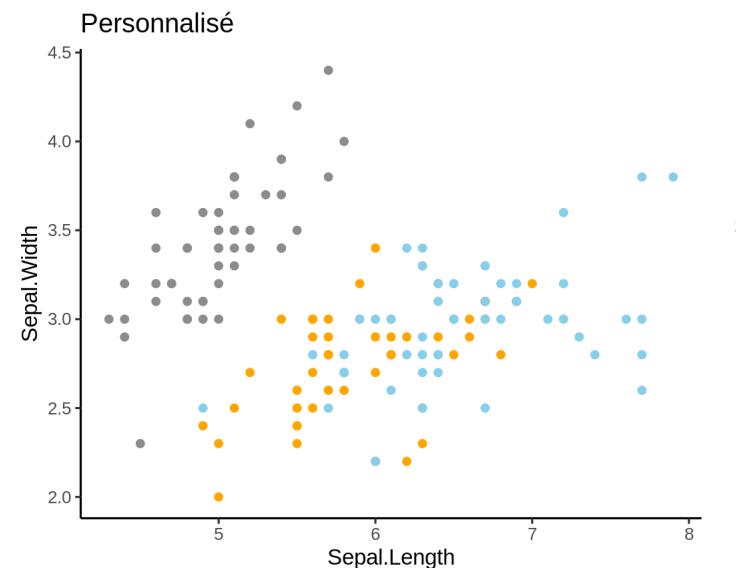


# Changer les couleurs manuellement

```
pp <- ggplot(data = iris) +  
  geom_point(aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 colour = Species))  
pp + labs(title = "Défaut")
```

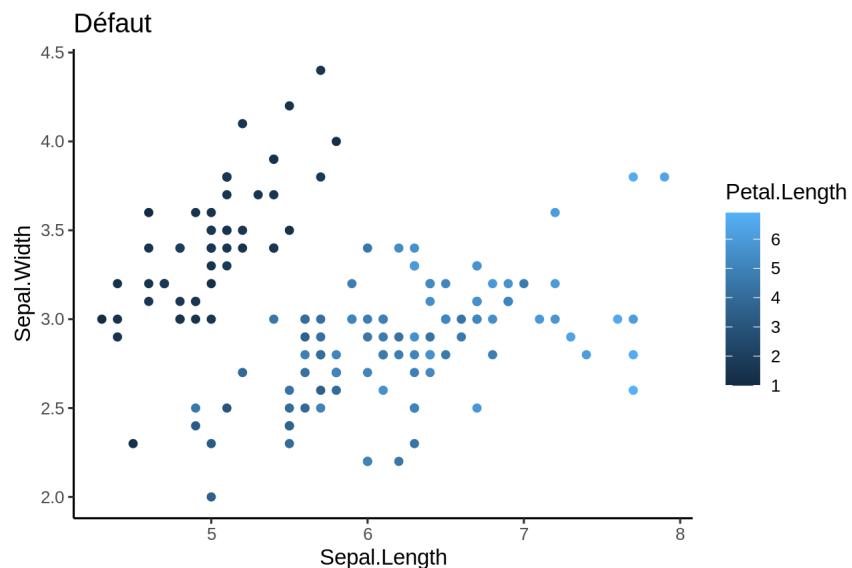


```
pp +  
  scale_colour_manual(values = c("grey",  
                                 "orange",  
                                 "skyblue"))  
pp + labs(title = "Personnalisé")
```

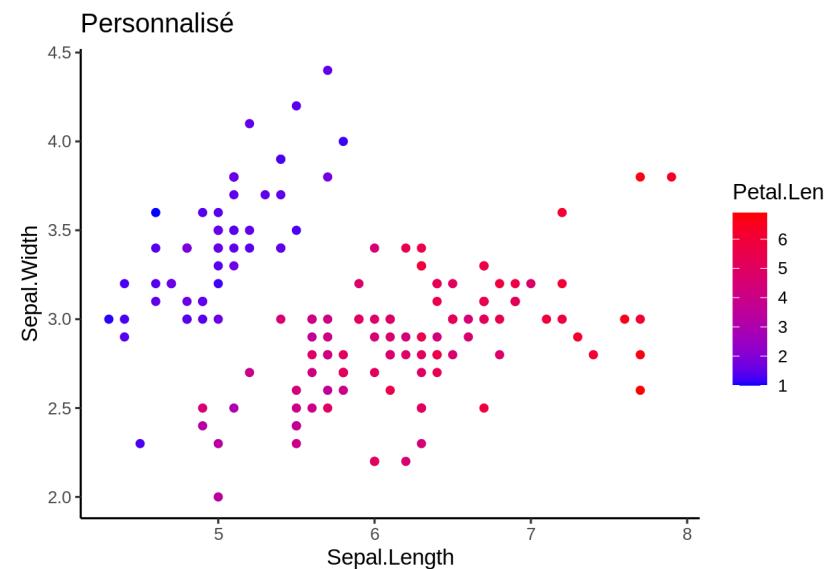


# Gradient de couleurs

```
pp2 <- ggplot(data = iris) +  
  geom_point(aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 colour = Petal.Length))  
pp2 + labs(title = "Défaut")
```



```
pp2 +  
  scale_colour_gradient(low = "blue",  
                        high = "red")  
  labs(title = "Personnalisé")
```



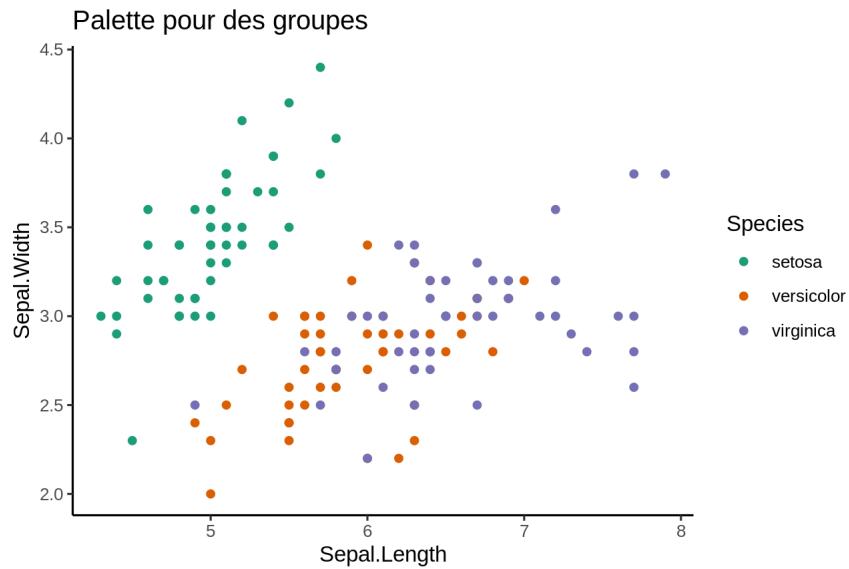
# Utiliser une palette de couleurs prédéfinie

```
install.packages("RColorBrewer")
library(RColorBrewer)
display.brewer.all()
```



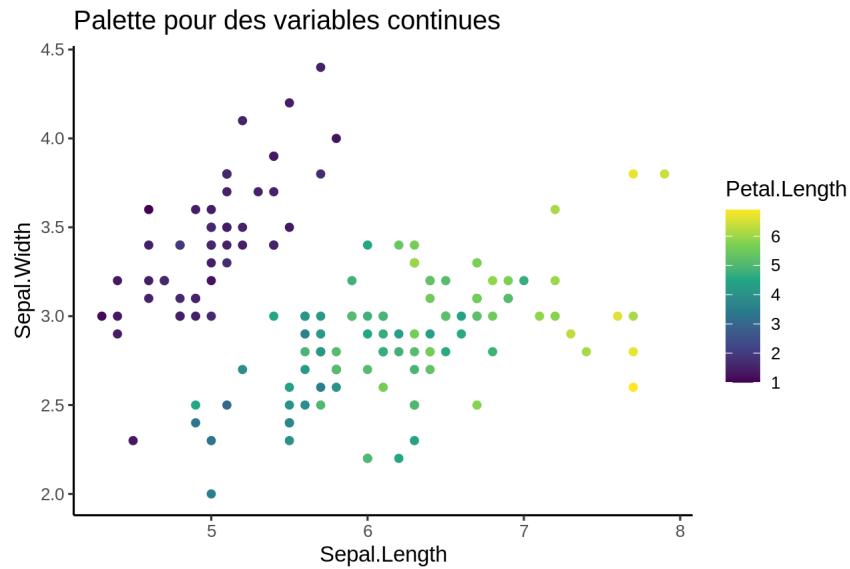
# Utiliser une palette de couleurs prédéfinie

```
pp + scale_colour_brewer(palette = "Dark2") +  
  labs(title = "Palette pour des groupes")
```



# Utiliser une palette de couleurs prédéfinie

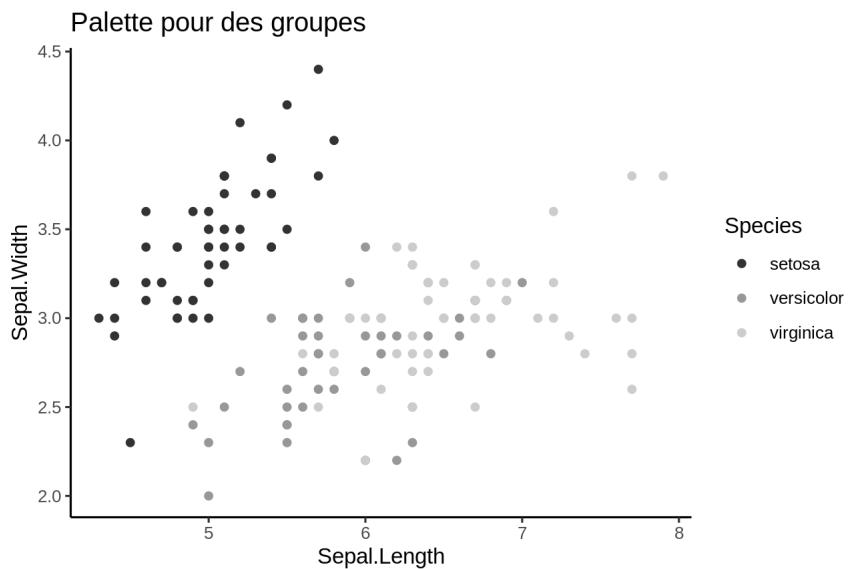
```
pp2 + scale_color_viridis_c()+
  labs(title = "Palette pour des variables continues")
```



# Utiliser une palette de couleurs prédéfinie

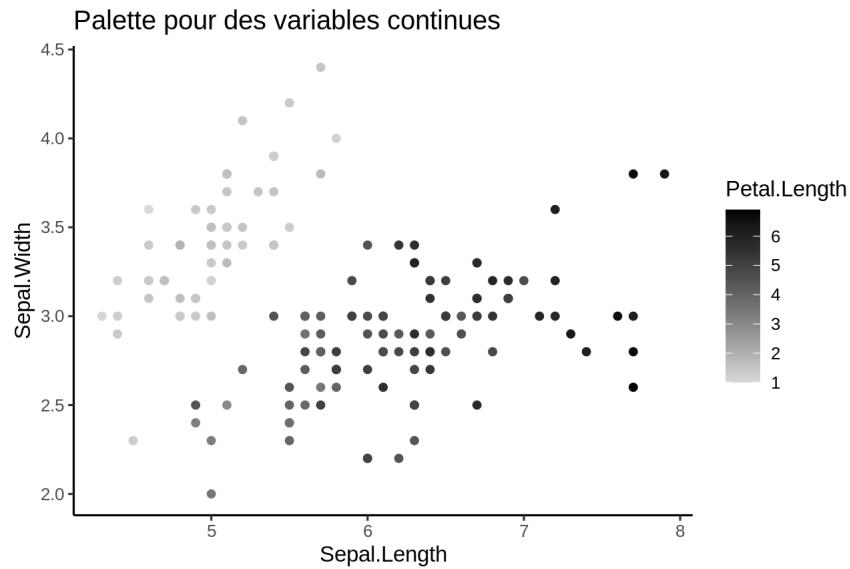
Palette de gris pour la publication

```
pp + scale_colour_grey() +  
  labs(title = "Palette pour des groupes")
```



# Utiliser une palette de couleurs prédéfinie

```
pp2 + scale_colour_gradient(low = "grey85", high = "black") +  
  labs(title = "Palette pour des variables continues")
```



# Utiliser des palettes de couleurs visibles pour les daltoniens

Comment votre figure peut-elle apparaître sous différentes formes de daltonisme ? Nous pouvons utiliser [colorblindr](#) qui n'est pas actuellement sur CRAN, donc nous l'installons avec la librairie [remotes](#).

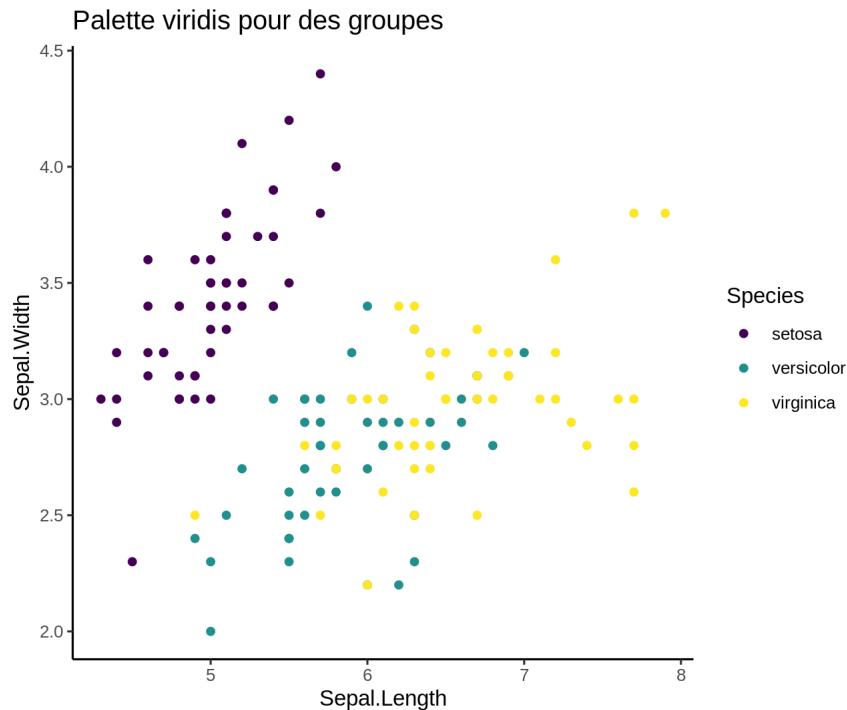
```
# install.packages("remotes")
remotes::install_github("clauswilke/colorblindr", quiet = TRUE)
library(colorblindr)
```

# Utiliser des palettes de couleurs visibles pour les daltoniens

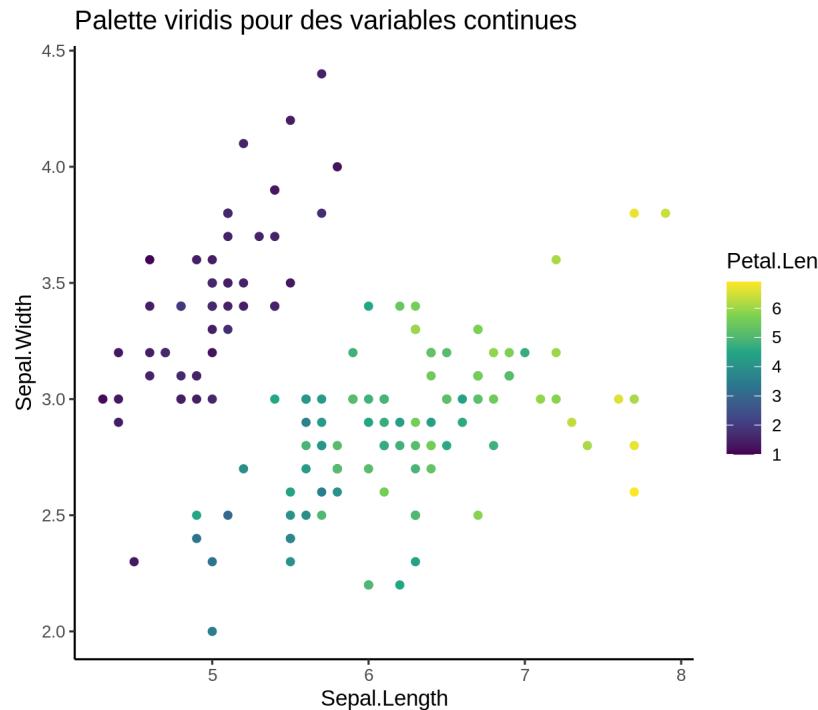
```
cvd_grid(pp)
```

# Utiliser des palettes de couleurs visibles pour les daltoniens

```
pp + scale_colour_viridis_d() +  
  labs(title = "Palette viridis pour des groupes")
```

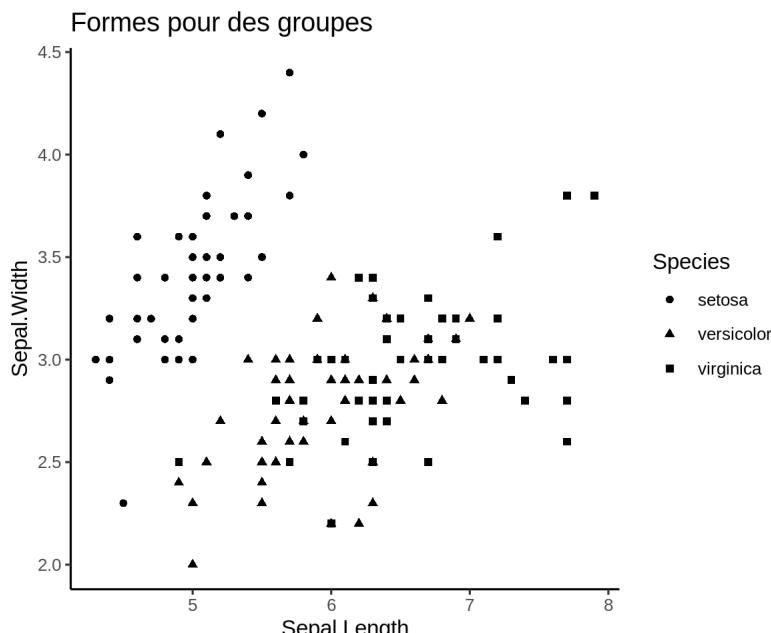


```
pp2 + scale_colour_viridis_c() +  
  labs(title = "Palette viridis pour des variables continues")
```

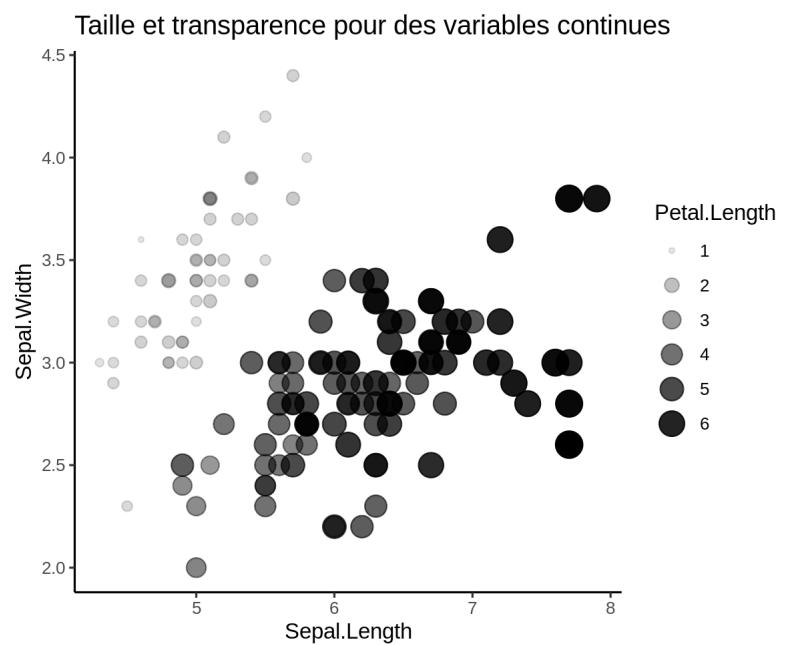


# Changer la forme, la taille et la transparence

```
ggplot(data = iris) +  
  geom_point(aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 shape = Species)) +  
  labs(title = "Formes pour des groupes")
```



```
ggplot(data = iris) +  
  geom_point(aes(x = Sepal.Length,  
                 y = Sepal.Width,  
                 size = Petal.Length,  
                 alpha = Petal.Length)) +  
  labs(title = "Taille et transparence")
```





# Défi #2

- Créer un graphique informatif à partir de jeu de données disponible de R, comme `mtcars`, `CO2` ou `msleep`.
- Utiliser les esthétiques appropriés pour différents types de données

Données	x	y	Esthétiques
mtcars	<i>wt</i>	<i>mpg</i>	<i>disp</i> et <i>hp</i>
CO2	<i>conc</i>	<i>uptake</i>	<i>Treatment</i> et <i>Type</i>
msleep	$\log_{10}(\text{bodywt})$	<i>awake</i>	<i>vore</i> et <i>conservation</i>
ToothGrowth	<i>dose</i>	<i>len</i>	<i>supp</i>

**⚠️ Faites attention aux types de données !**



# Défi #2

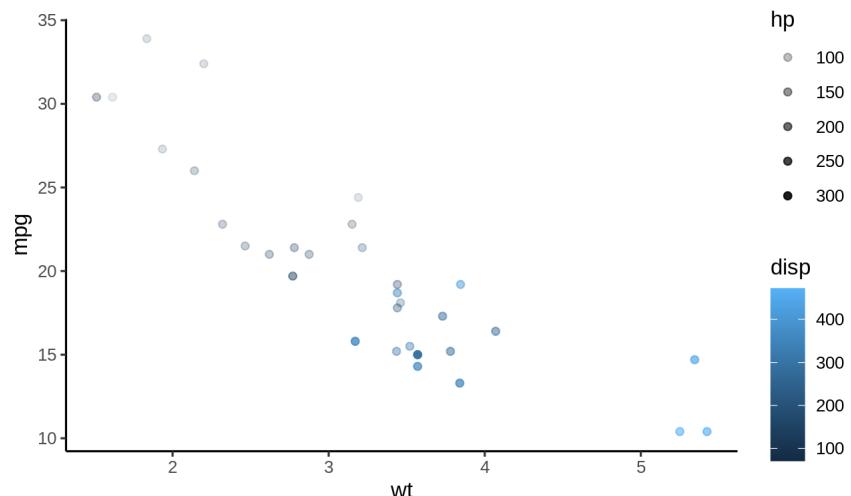
## Salles de réunion ! - 15 min

Créez un graphique à partager avec le groupe

# Défi #2 - Solution exemple #1

Données	x	y	Esthétiques
mtcars	wt	mpg	disp et hp

```
data(mtcars)
ggplot(data = mtcars) +
  geom_point(mapping = aes(x = wt, y = mpg,
                           colour = disp,
                           alpha = hp))
```

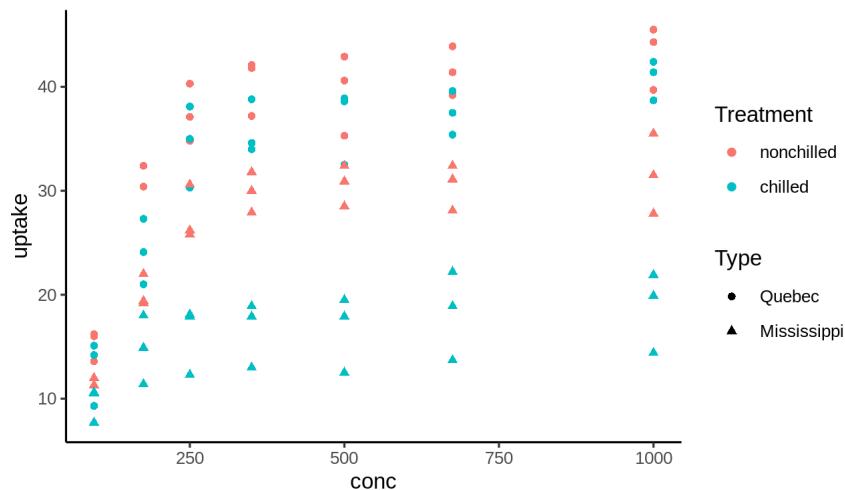


Pourriez-vous utiliser `size` au lieu de `alpha`? Et `shape`?

# Défi #2 - Solution exemple #2

Données	x	y	Esthétiques
CO2	conc	uptake	<i>Treatment et Type</i>

```
data(CO2)
ggplot(data = CO2) +
  geom_point(mapping = aes(x = conc, y = uptake,
                           colour = Treatment, shape = Type))
```

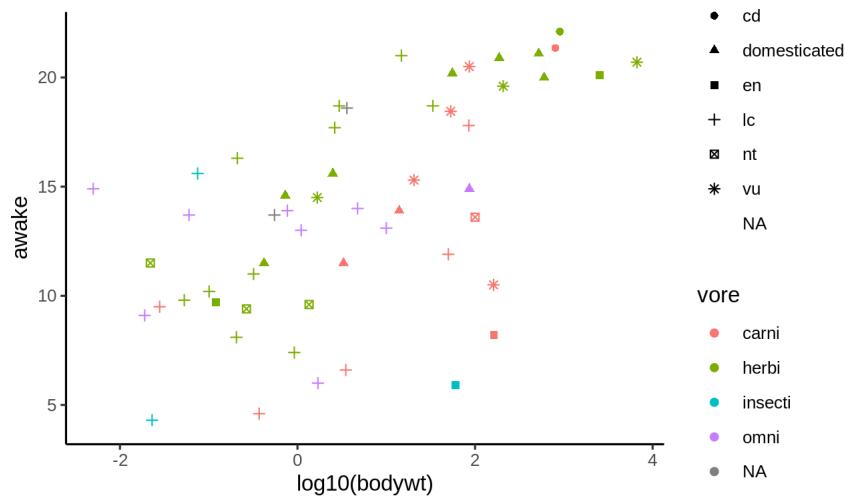


Pourquoi ne pas utiliser `size = Type`?

# Défi #2 - Solution exemple #3

Données	x	y	Esthétiques
msleep	$\log_{10}(\text{bodywt})$	awake	vore et conservation

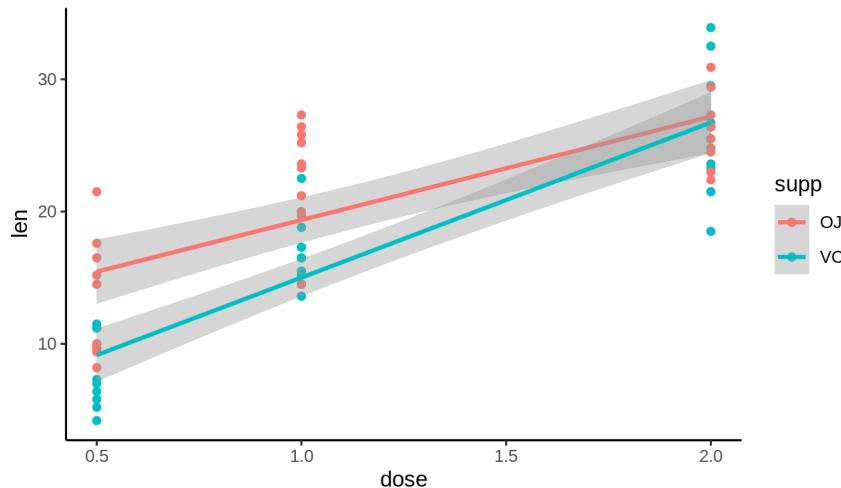
```
data(msleep)
ggplot(data = msleep) +
  geom_point(mapping = aes(x = log10(bodywt), y = awake,
                           colour = vore, shape = conservation))
```



# Défi #2 - Solution exemple #4

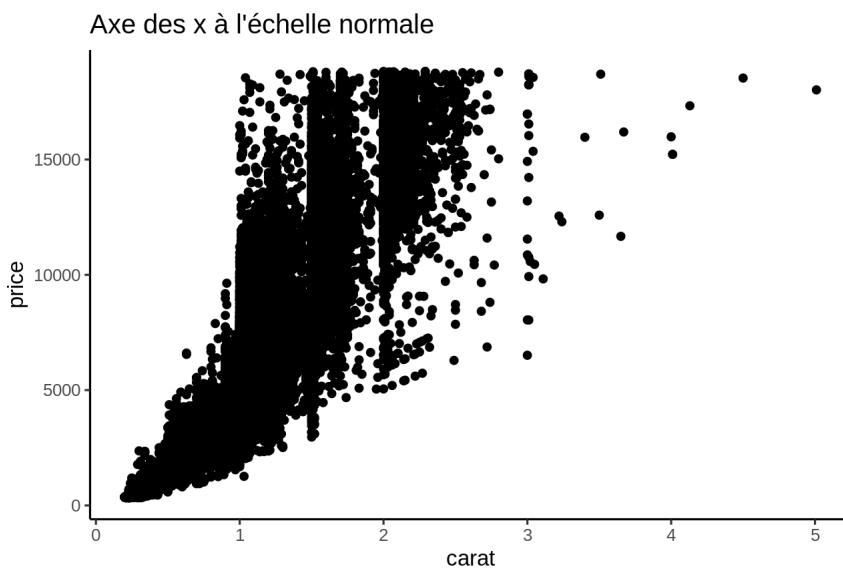
Données	x	y	Esthétiques
ToothGrowth	dose	len	supp

```
data(ToothGrowth)
ggplot(ToothGrowth, aes(x = dose, y = len, color = supp)) +
  geom_point() +
  geom_smooth(method = lm, formula = 'y~x')
```

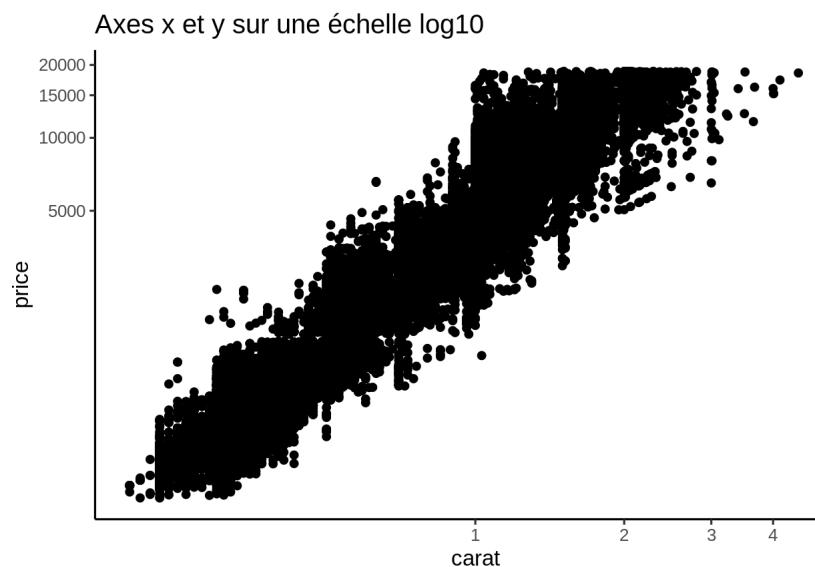


# Changer l'échelle des axes

```
ggplot(diamonds) +  
  geom_point(aes(x = carat, y = price))  
  labs(title = "Axe des x à l'échelle normale")
```



```
ggplot(diamonds) +  
  geom_point(aes(x = carat, y = price))  
  coord_trans(x = "log10",  
              y = "log10") +  
  labs(title = "Axes x et y sur une échelle log10")
```



Il est également possible de transformer le système de coordonnées en utilisant

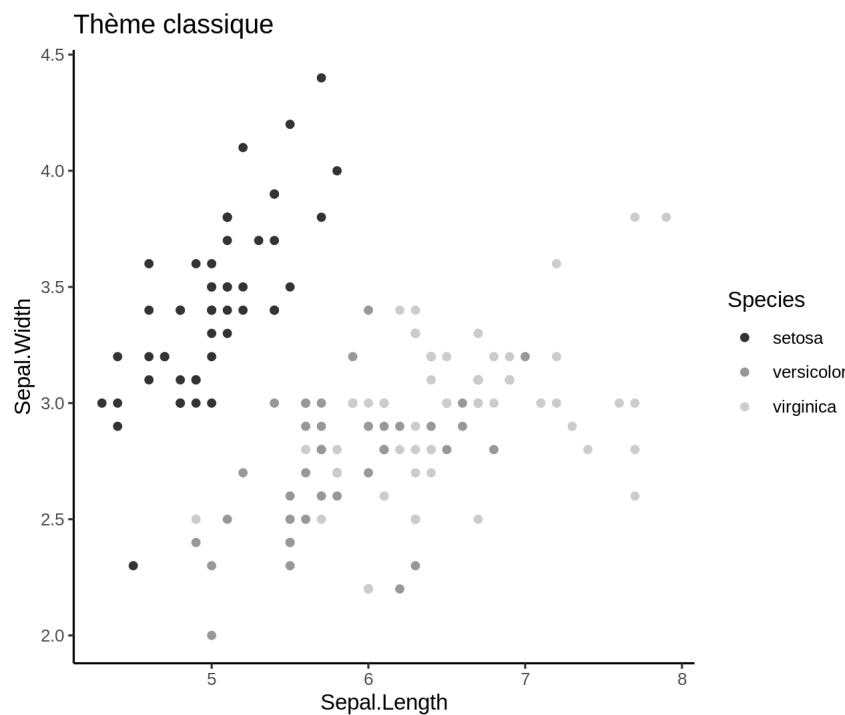
`scale_x_log10()` et `scale_y_log10()`

# Perfectionner vos graphiques

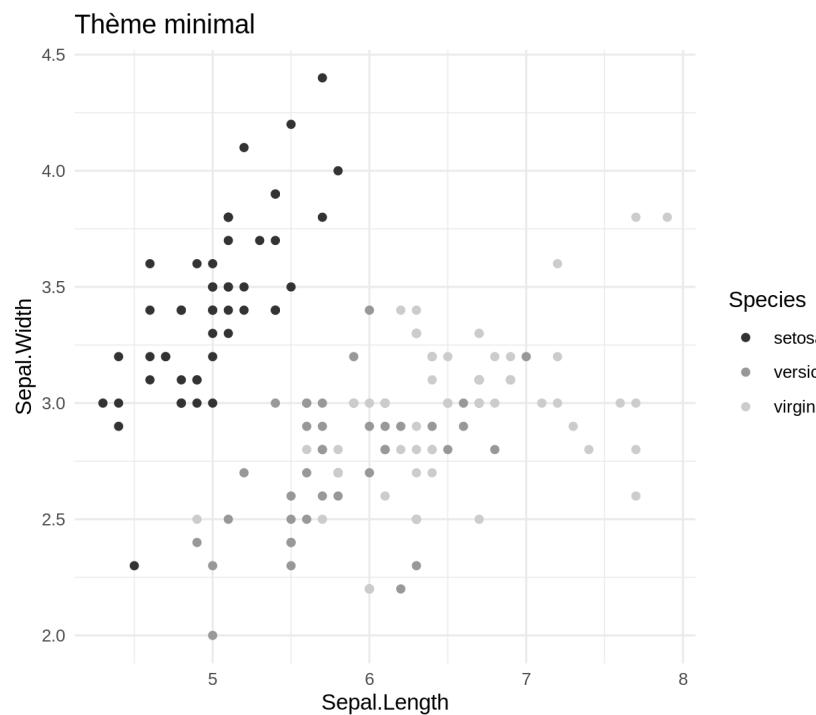
Utiliser le `theme()` pour faire bonne figure!

# theme()

```
pp + scale_colour_grey() +  
  theme_classic() +  
  labs(title = "Thème classique")
```



```
pp + scale_colour_grey() +  
  theme_minimal() +  
  labs(title = "Thème minimal")
```



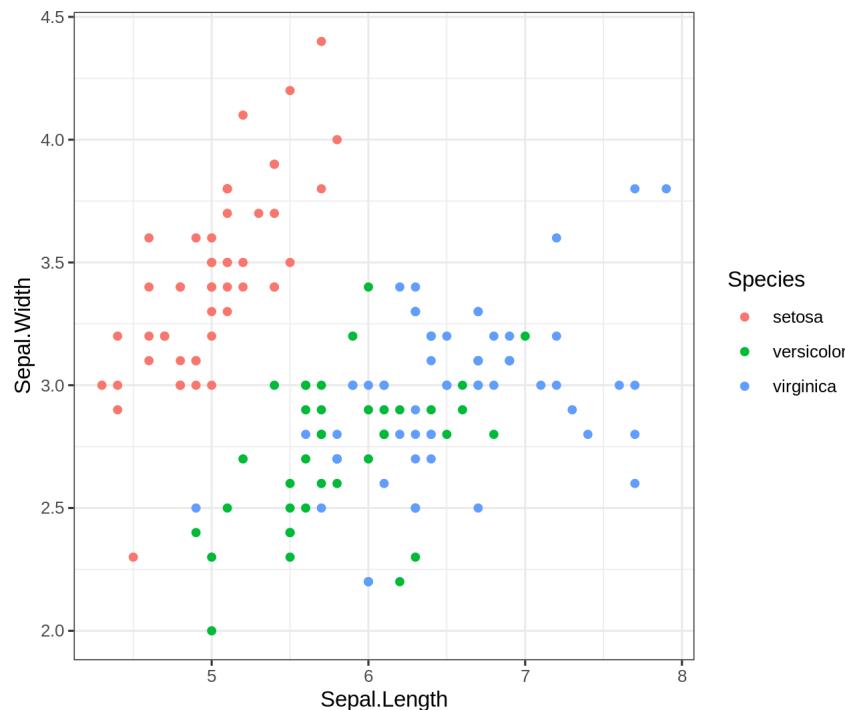
NB: De bons choix pour la publication !

# theme()

Utiliser `theme_set()` pour changer le thème pour tous vos prochains graphiques, ou `theme_update()` pour modifier certains aspects d'un thème.

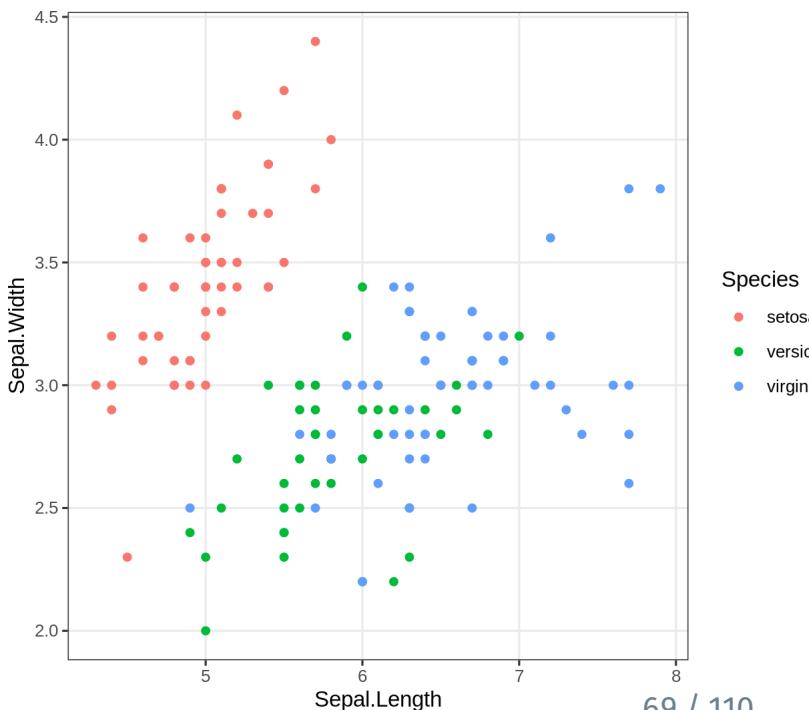
## Choisir le thème

```
# Choisir le thème classic par défaut  
theme_set(theme_bw())  
pp
```



## Mettre à jour le thème

```
# supprimer la grille mineure  
theme_update(panel.grid.minor = element  
pp
```



# theme()

## Les éléments d'un thème

### ggplot2 theme elements reference

Set minimal as the baseline theme:

```
theme_minimal() +  
  theme(element = element_type())
```

Use element\_blank() to remove an element

Axis titles, text, ticks, and lines can be specified per axis using theme inheritance by putting .x/.y at the end of the theme element.

```
axis.line.y = element_line()
```

```
axis.title.y = element_text()
```

```
panel.grid.major = element_line()
```

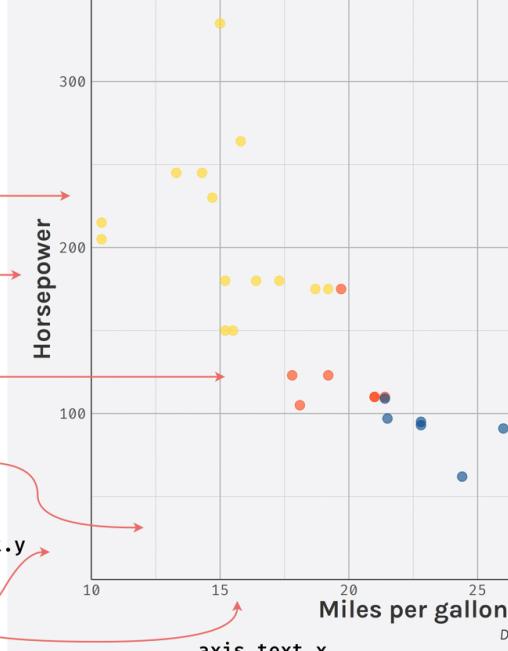
```
panel.grid.minor = element_line()
```

```
axis.text.y  
axis.text = element_text()
```

plot.title.position = "plot"  
plot.caption.position = "plot"

plot.title = element\_text()  
plot.subtitle = element\_text()

Miles per Gallon & Horsepower  
of 32 Automobiles(1973-74 models)



plot.margin = margin(25, 25, 25, 25)

legend.title = element\_text()

legend.background = element\_rect()

legend.text = element\_text()

legend.position = c(.85,.85) / "none" /  
"left" / "right" /  
"bottom" / "top"

plot.background = element\_rect()

plot.caption = element\_text()

text = element\_text() ← modifications will be applied to all text elements

Full list of elements at [ggplot2.tidyverse.org/reference/theme](http://ggplot2.tidyverse.org/reference/theme)

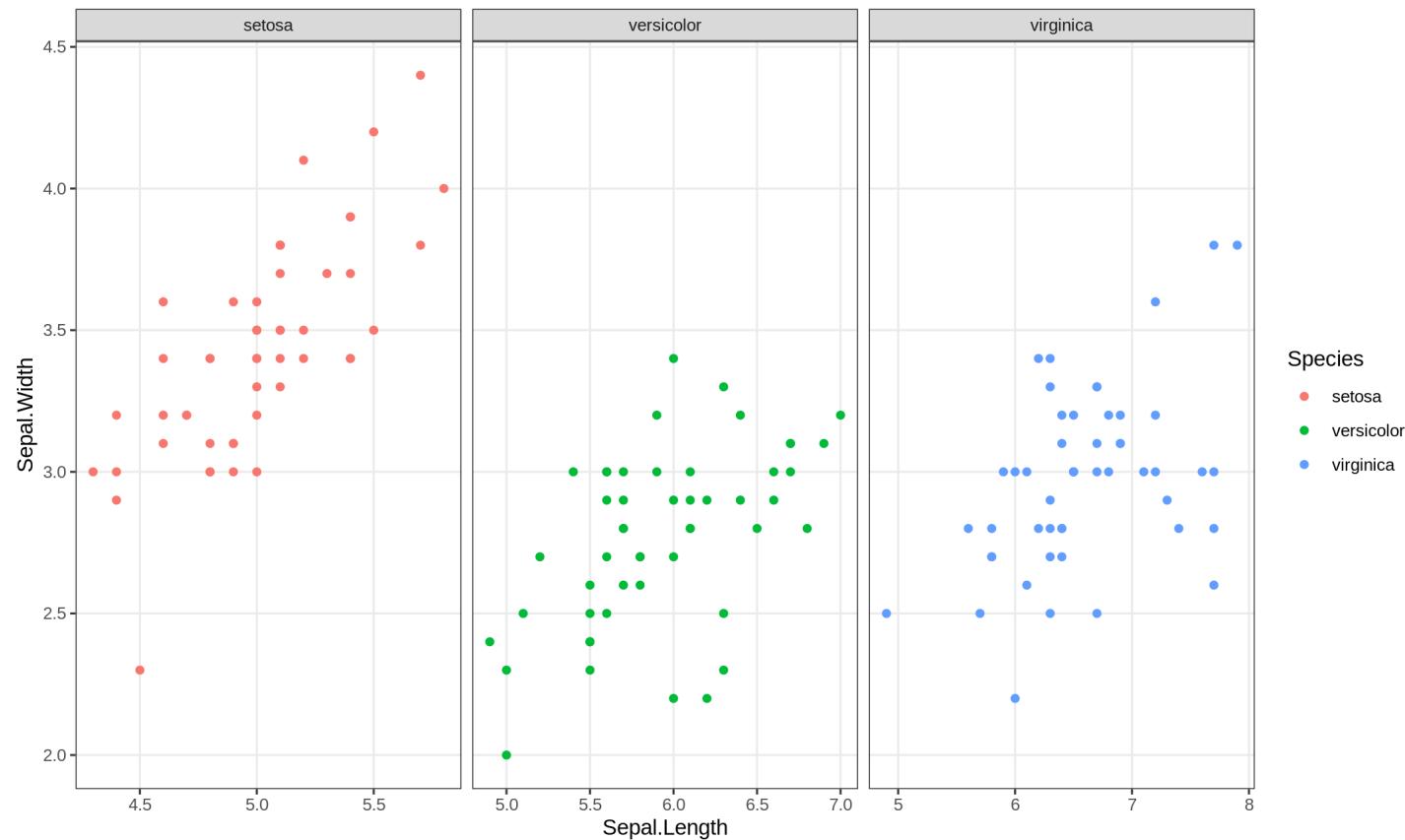
isabella-b

# Perfectionner vos graphiques

Utiliser `facet_grid()` pour changer la disposition des graphiques

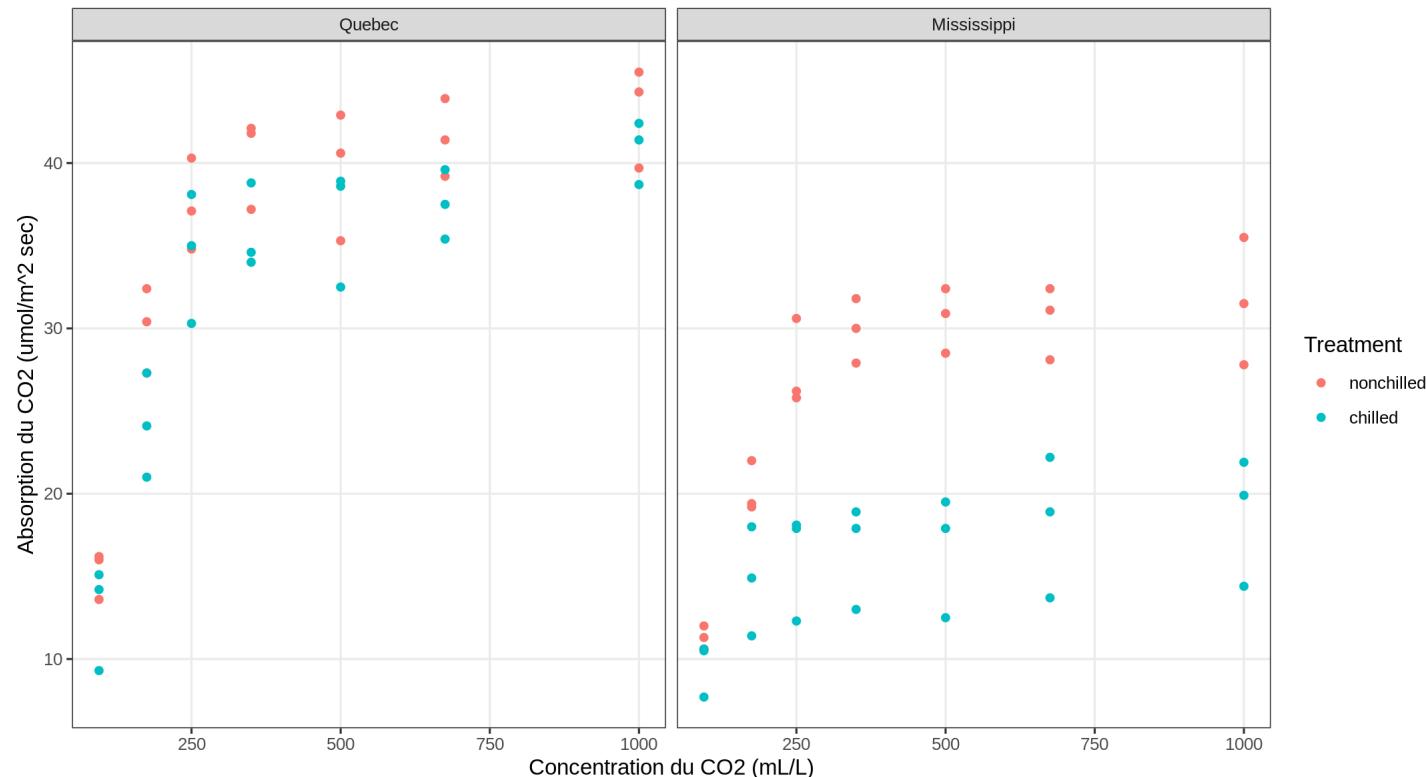
# iris: facettes par espèce

```
ggplot(data = iris) +  
  geom_point(mapping = aes(x = Sepal.Length, y = Sepal.Width, colour = Species)) +  
  facet_grid(~ Species, scales = "free")
```



# CO<sub>2</sub>: facettes par type

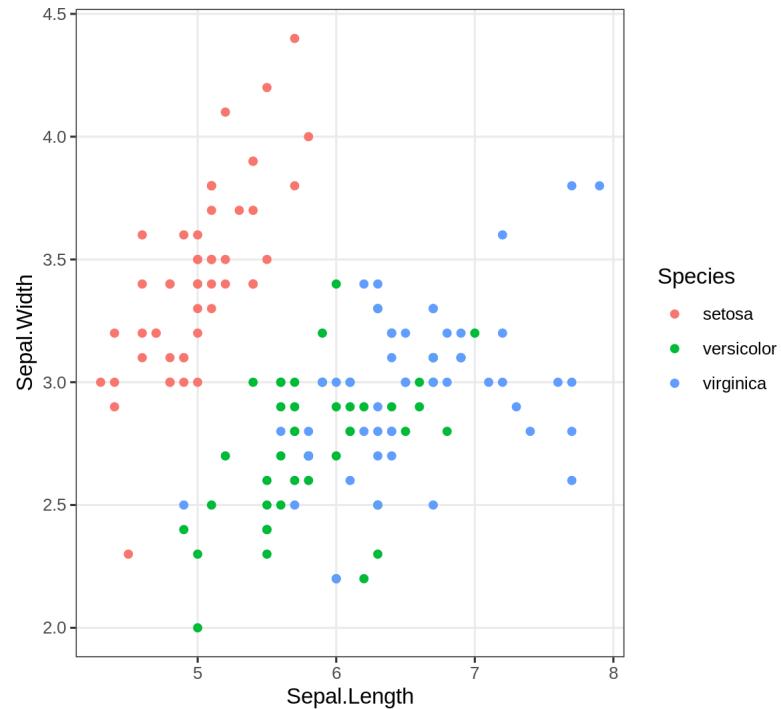
```
ggplot(data = CO2) +  
  geom_point(mapping = aes(x = conc, y = uptake, colour = Treatment)) +  
  xlab("Concentration du CO2 (mL/L)") + ylab("Absorption du CO2 (umol/m2 sec)") +  
  facet_grid(~ Type)
```



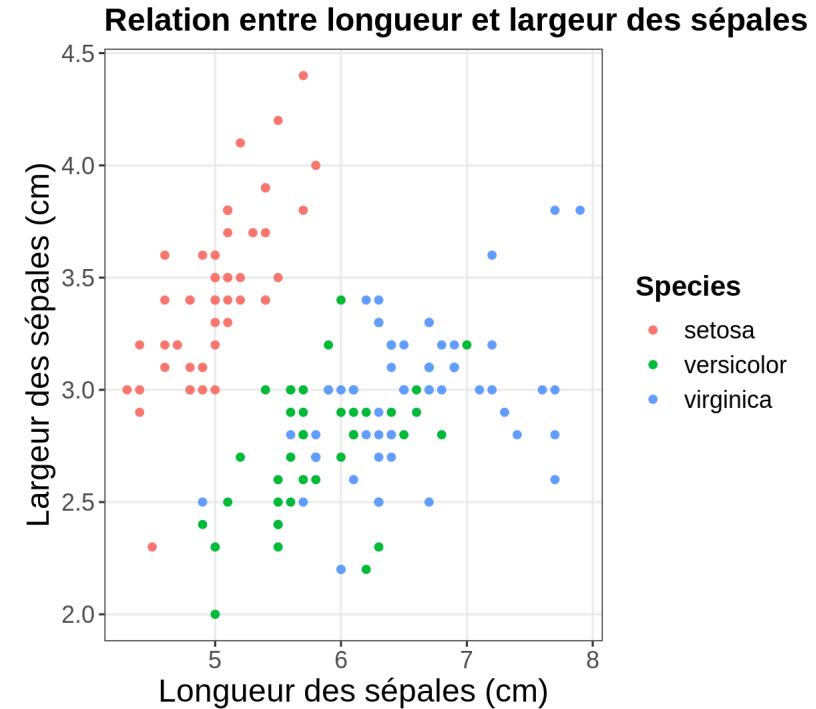
# Titre et axes: changer la taille, la couleur et l'apparence

# Titre et axes: changer la taille, la couleur et l'apparence

## Défaut



## Ajustement des axes et du titre

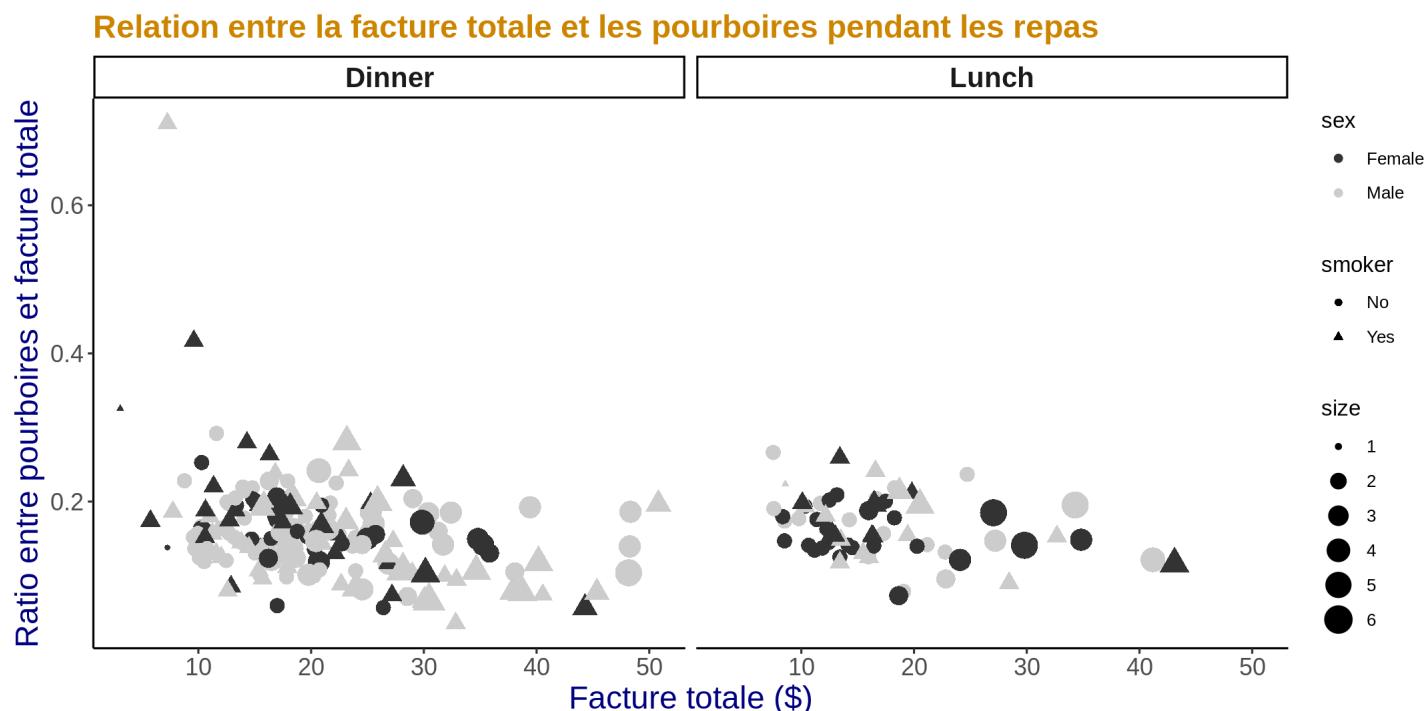




# Défi #3

Utilisez les données `tips` qui se trouvent dans `reshape2` pour reproduire le graphique ci-dessous.

Notre conseil: commencez par `theme_classic()` et ajoutez `theme()` pour faire vos changements supplémentaires.





# Défi #3

**Salles de réunion! - 15 min**



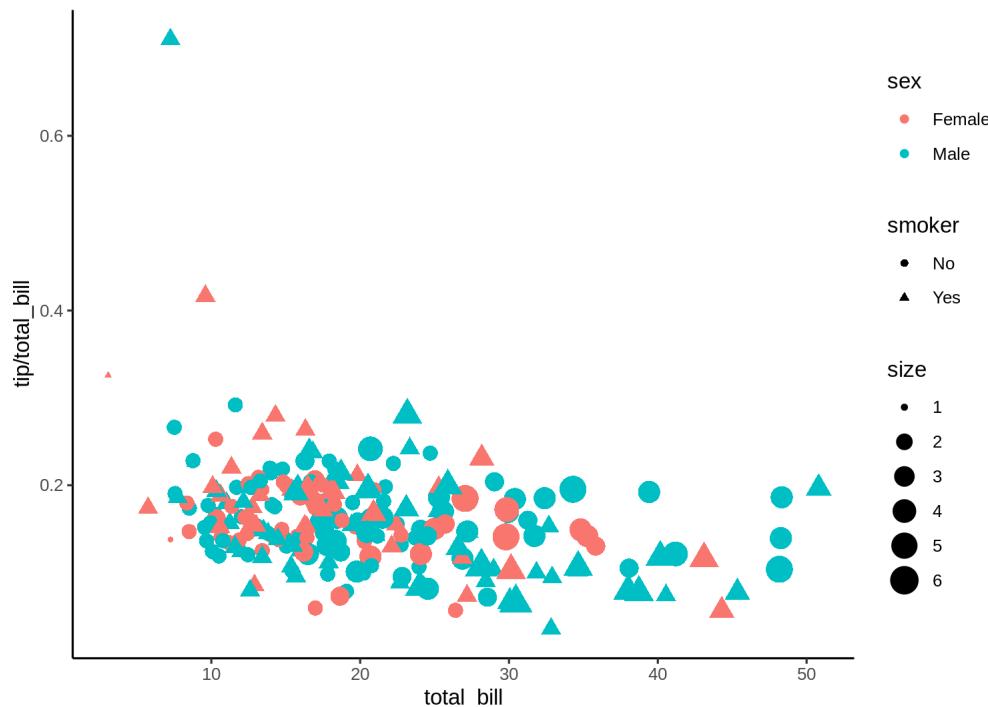
# Défi #3: Solution

```
library(reshape2)
tips.gg <- ggplot(tips, aes(x = total_bill,
                               y = tip/total_bill,
                               shape = smoker,
                               colour = sex,
                               size = size)) +
  geom_point() +
  facet_grid( ~ time) +
  scale_colour_grey() +
  labs(title = "Relation entre la facture totale et les pourboires pendant les repas",
       x = "Facture totale ($)", y = "Ratio entre pourboires et facture totale") +
  theme_classic() +
  theme(axis.title = element_text(size = 16,
                                   colour = "navy"),
        axis.text = element_text(size = 12),
        plot.title = element_text(size = 16,
                                  colour = "orange3",
                                  face = "bold"),
        strip.text.x = element_text(size = 14, face ="bold"))
tips.gg
```

# Défi #3: solution par étapes

aes()

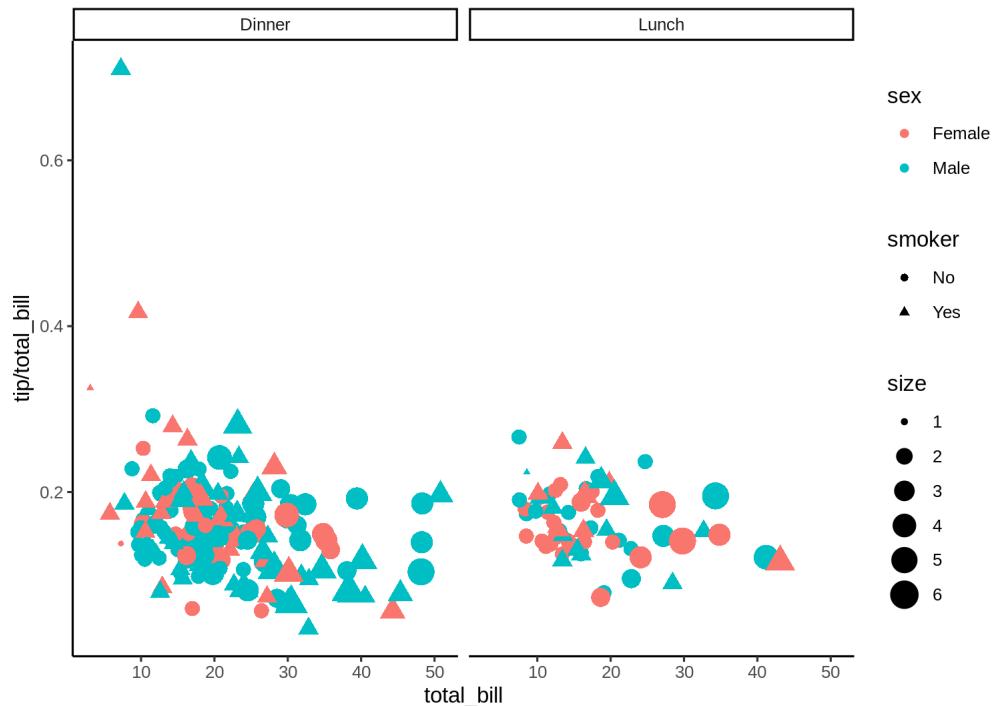
```
theme_set(theme_classic())
tips.gg <- ggplot(tips) +
  geom_point(mapping = aes(x = total_bill, y = tip/total_bill,
                           shape = smoker, colour = sex, size = size))
tips.gg
```



# Défi #3: solution par étapes

facet\_grid()

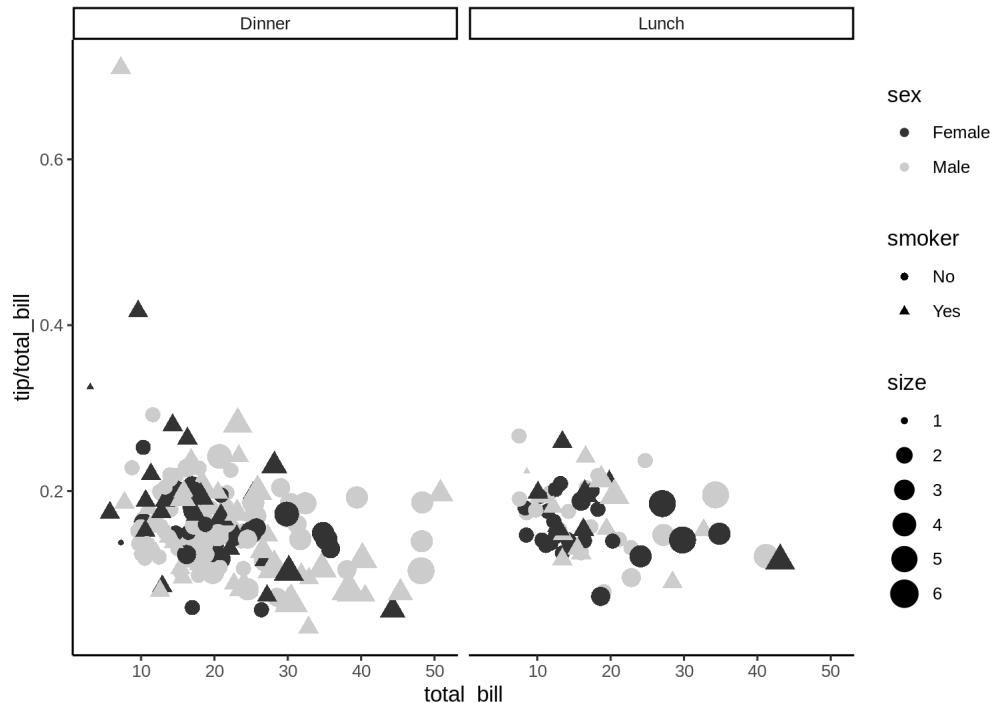
```
tips.gg <- tips.gg +  
  facet_grid( ~ time)  
tips.gg
```



# Défi #3: solution par étapes

## Palette de gris

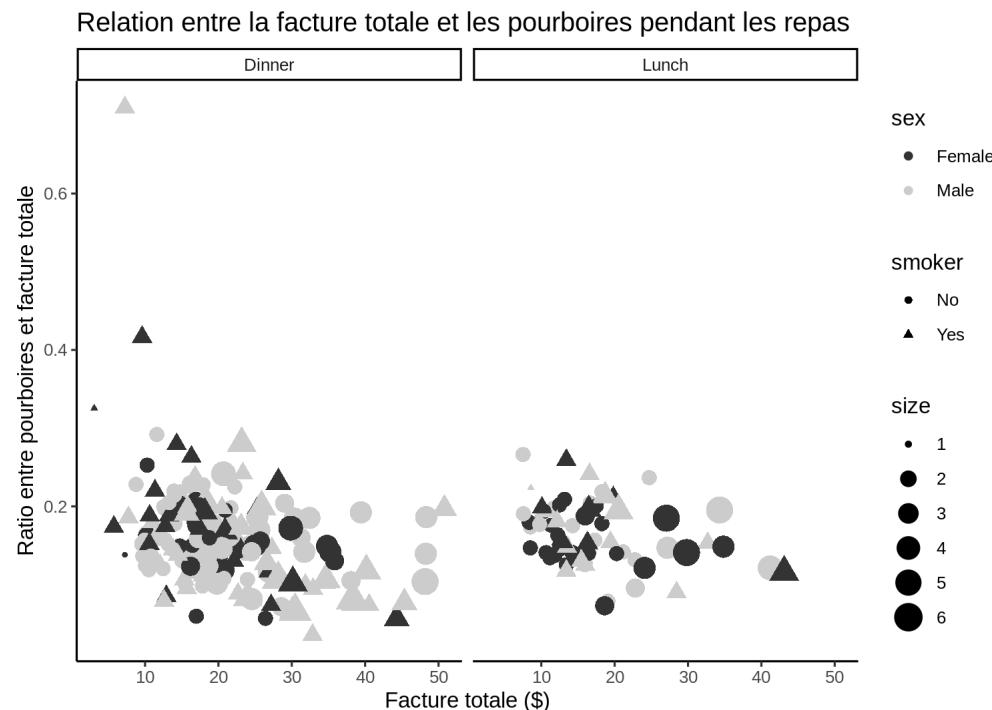
```
tips.gg <- tips.gg +  
  scale_colour_grey()  
tips.gg
```



# Défi #3: solution par étapes

## Ajout du titre et des étiquettes du graphique

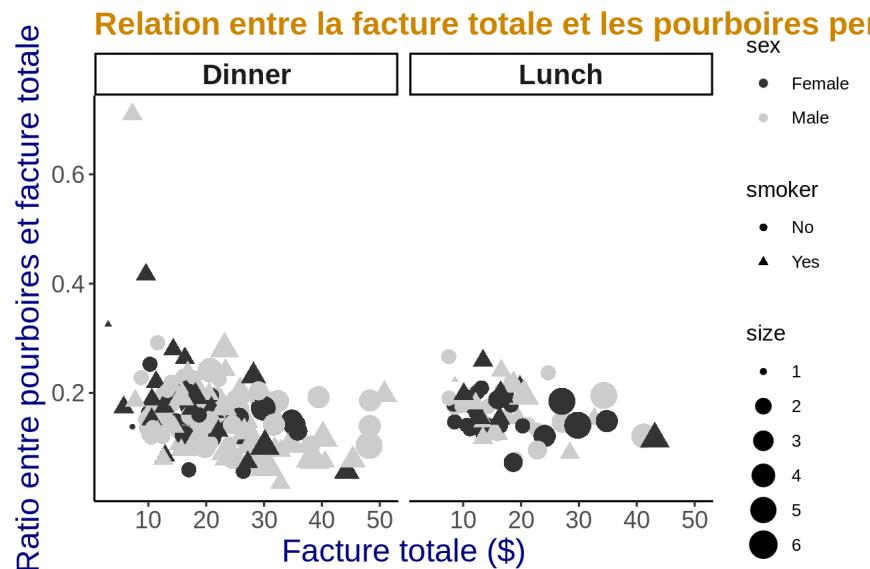
```
tips.gg <- tips.gg +  
  labs(title = "Relation entre la facture totale et les pourboires pendant les repas",  
        x = "Facture totale ($)", y = "Ratio entre pourboires et facture totale")  
tips.gg
```



# Défi #3: solution par étapes

## Ajout et ajustement du thème

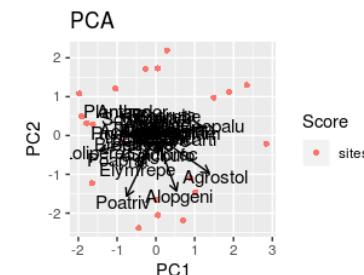
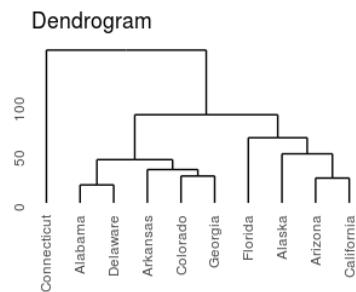
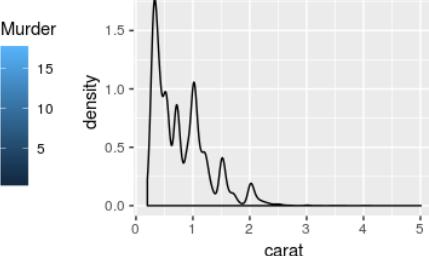
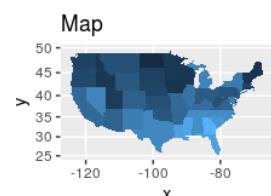
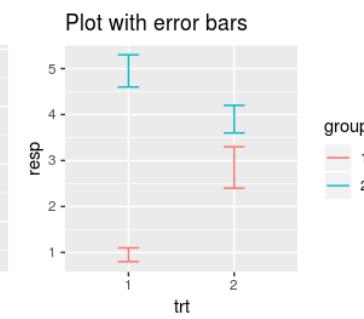
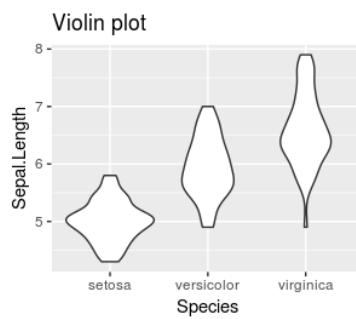
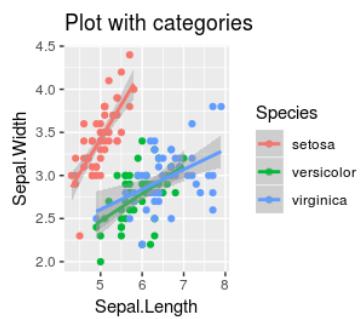
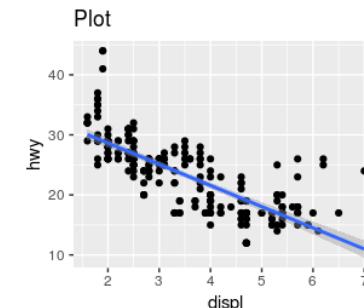
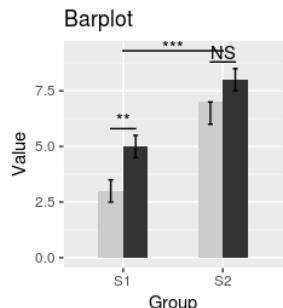
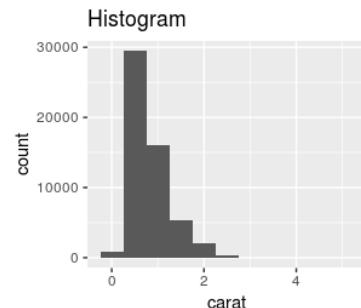
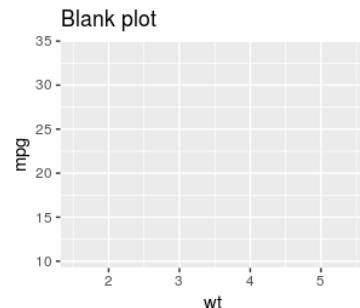
```
tips.gg <- tips.gg +  
  theme_classic() +  
  theme(axis.title = element_text(size = 16, colour = "navy"),  
        axis.text = element_text(size = 12),  
        plot.title = element_text(size = 16, colour = "orange3", face = "bold"),  
        strip.text.x = element_text(size = 14, face = "bold"))  
tips.gg
```



# Perfectionner vos graphiques

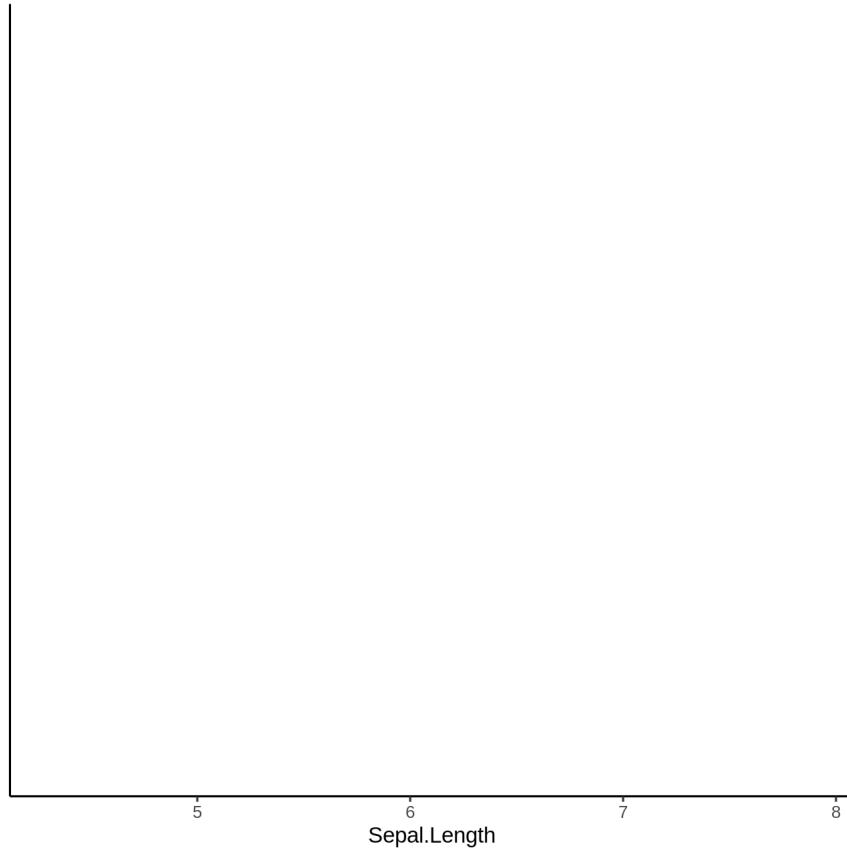
Utiliser `geom_*( )` pour créer différents types de graphique

# ggplot2::geom\_\*( )



## ggplot2::ggplot()

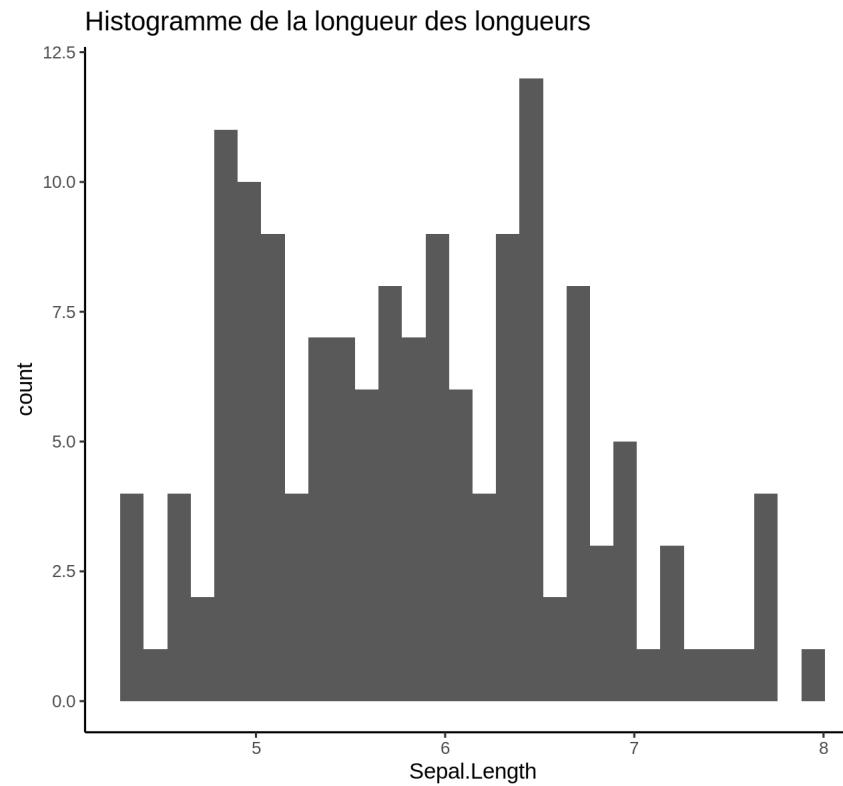
```
ggplot(iris, aes(Sepal.Length))
```



# Histogrammes: `geom_histogram()`

Un **histogramme** est une représentation graphique précise de la répartition des données numériques - une seule esthétique requise

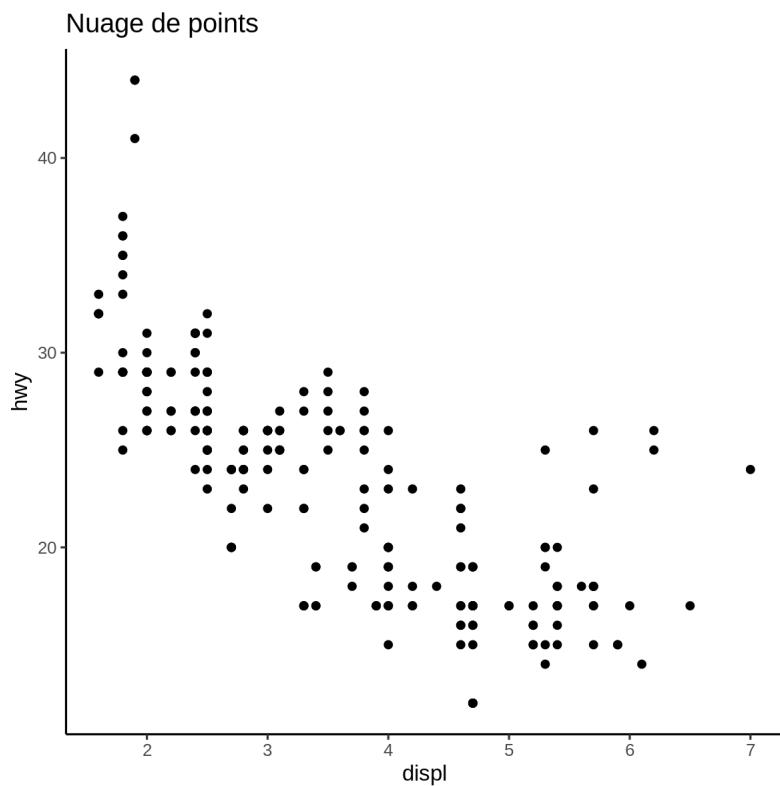
```
ggplot(iris, aes(Sepal.Length)) +  
  geom_histogram() +  
  ggtitle("Histogramme de la longueur des longueurs")
```



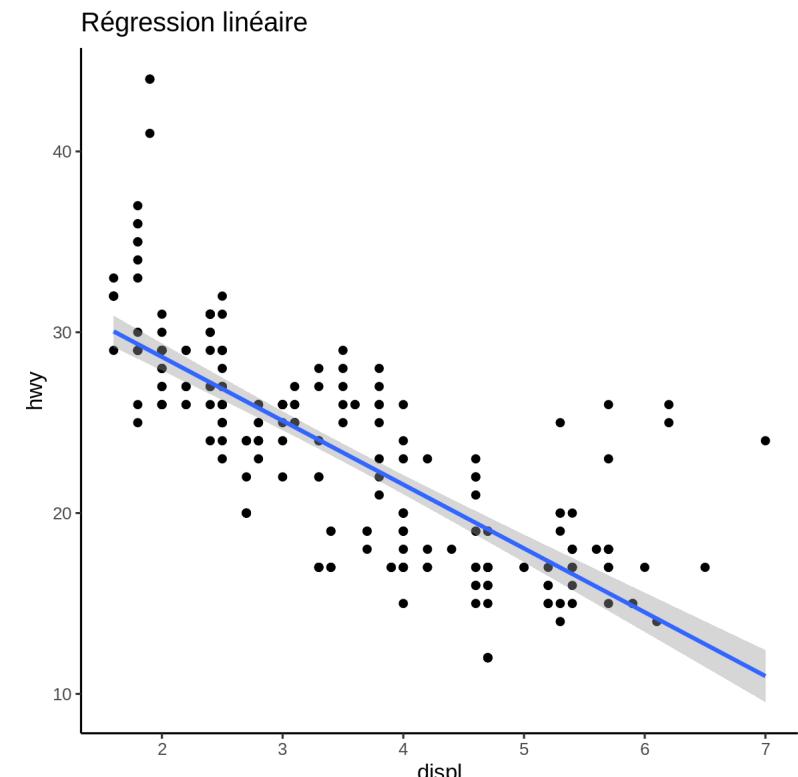
# Nuage de points et régression linéaire:

`geom_point()` et `geom_smooth()`

```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point() +  
  labs(title = "Nuage de points")
```

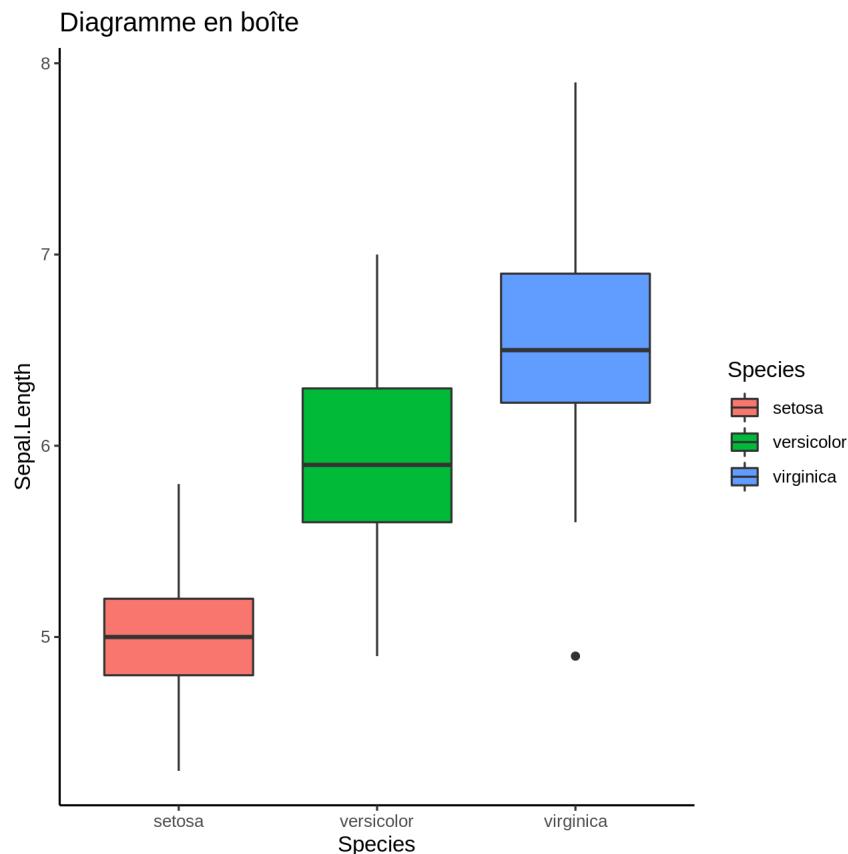


```
ggplot(mpg, aes(displ, hwy)) +  
  geom_point() +  
  labs(title = "Régression linéaire") +  
  geom_smooth(method = lm)
```

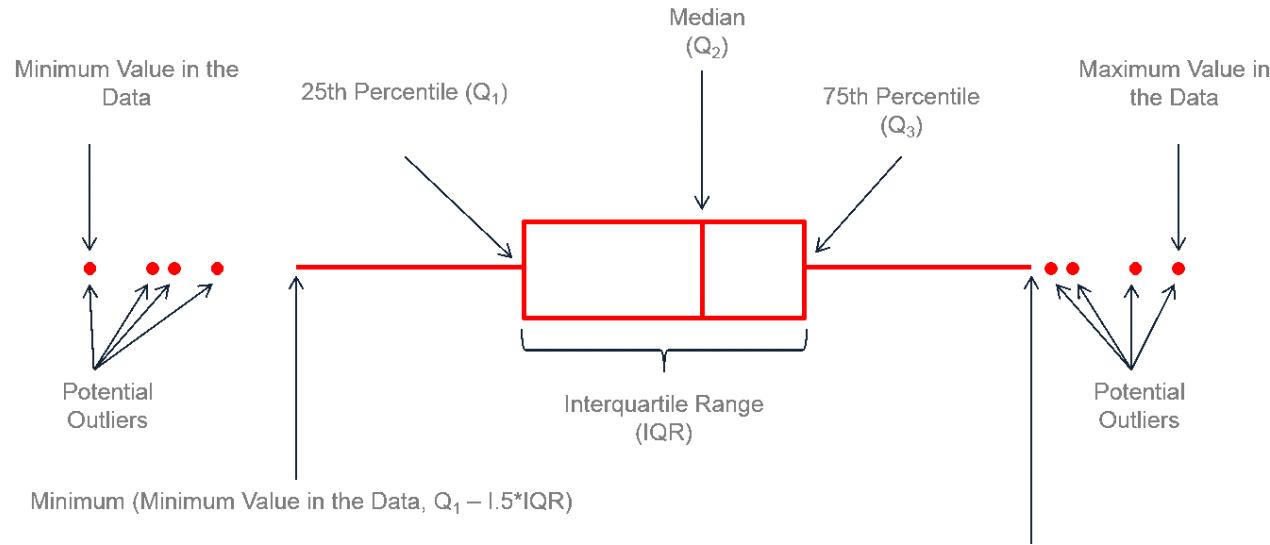
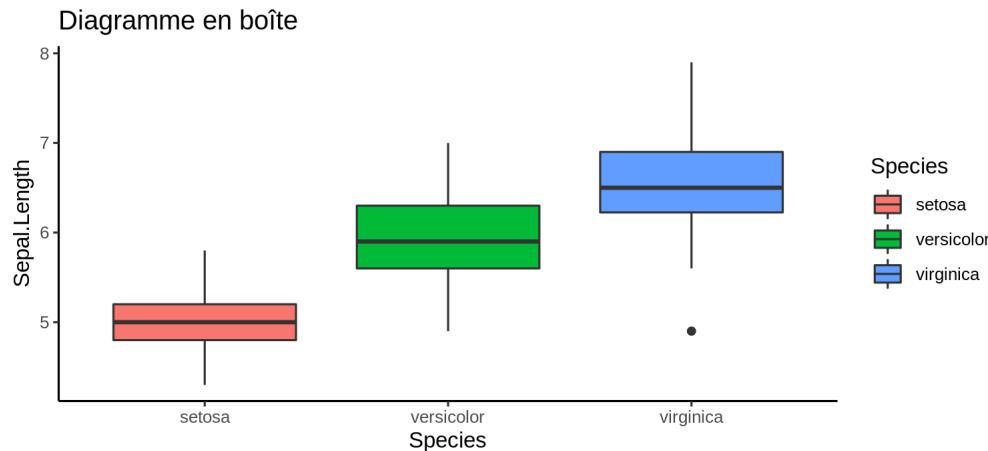


# Diagramme en boîte: `geom_boxplot()`

```
ggplot(data = iris, aes(Species, Sepal.Length,  
fill = Species)) +  
  geom_boxplot() +  
  labs(title = "Diagramme en boîte")
```



# Boxplot: `geom_boxplot()`



credit to [sc.studysixsigma.com](http://sc.studysixsigma.com)

# Diagramme en boîte avec annotations:

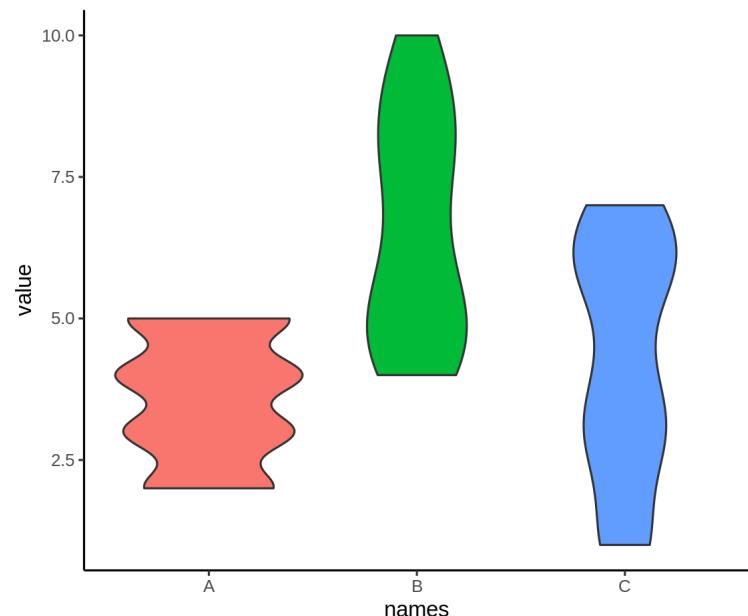
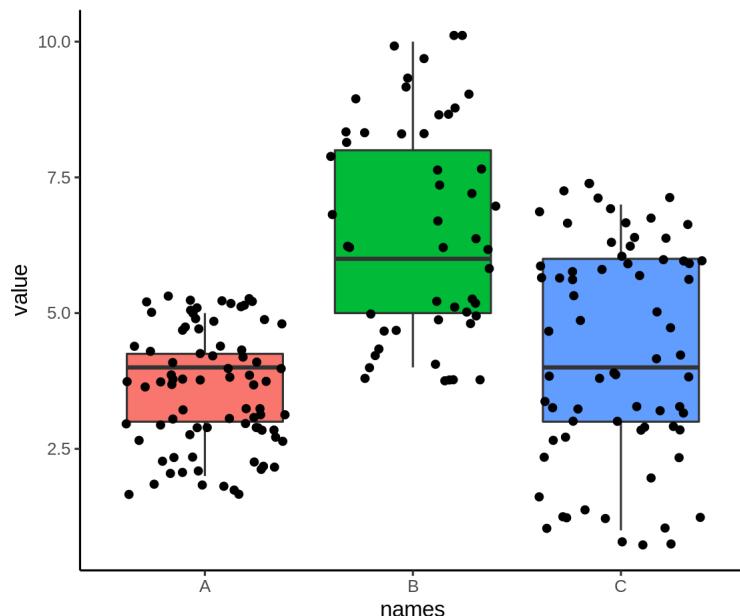
`geom_boxplot()` et `geom_signif()`

```
library(ggsignif)
ggplot(data = iris, aes(Species, Sepal.Length)) +
  geom_boxplot() +
  geom_signif(comparisons = list(c("versicolor", "virginica")),
              map_signif_level = TRUE)
```

# Graphique en violon: `geom_violin()`

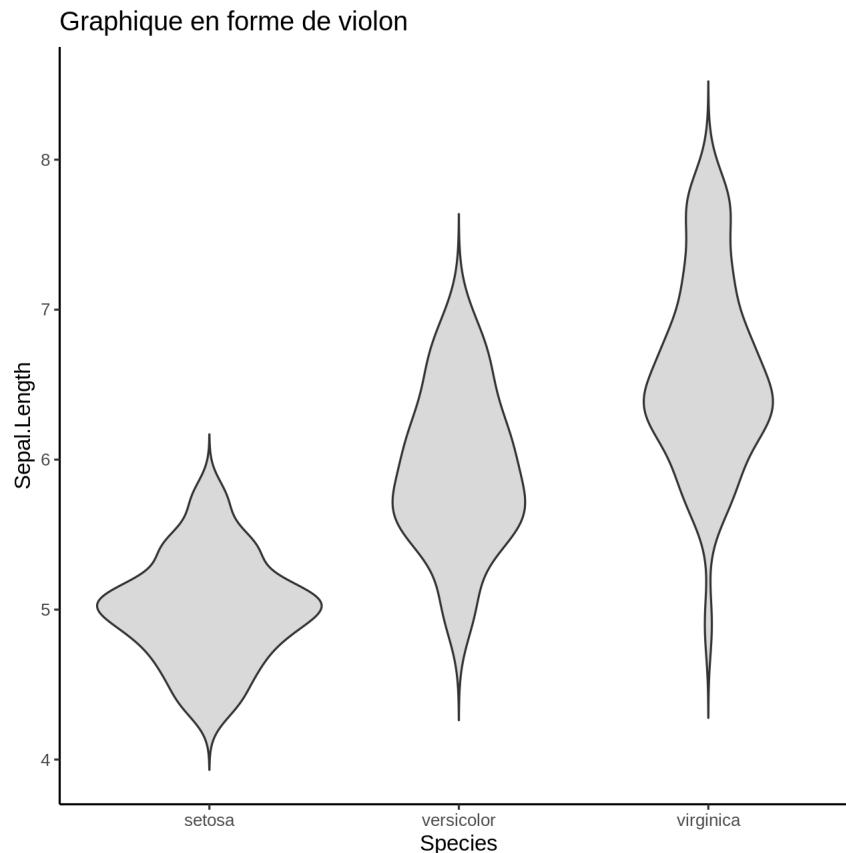
Comme expliqué sur le site [from Data to Viz](#)

Le graphique en forme de violon permet de visualiser la distribution d'une variable numérique pour un ou plusieurs groupes. [...] Il est très proche du diagramme en boîte, mais permet une compréhension plus approfondie de la distribution.



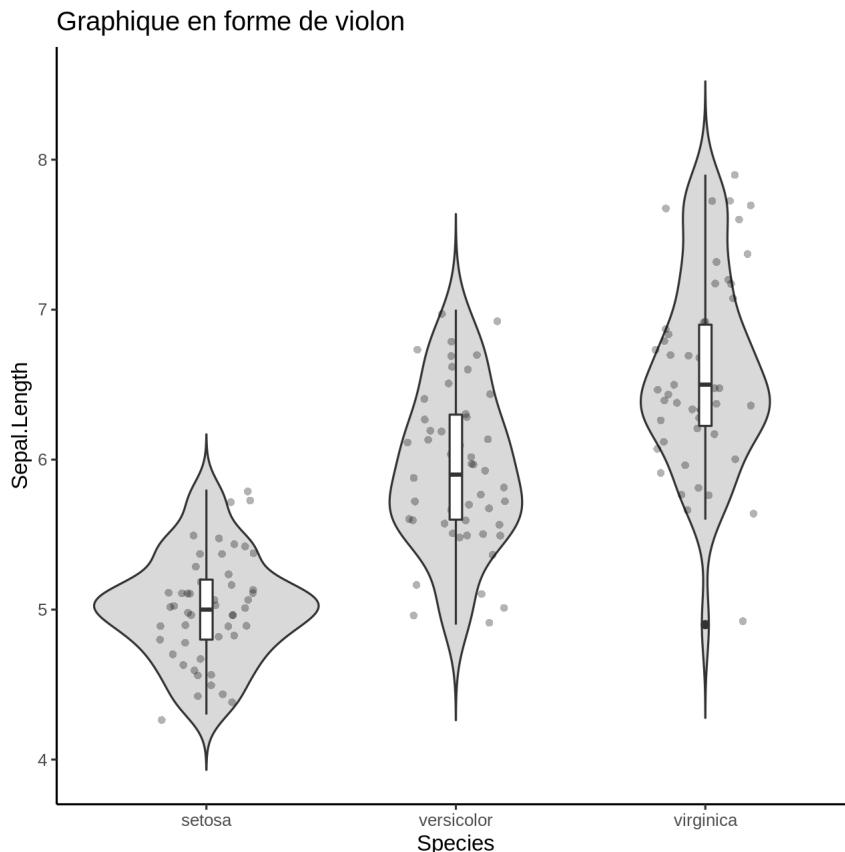
# Graphique en violon: `geom_violin()`

```
violin <- ggplot(data = iris, aes(x = Species, y = Sepal.Length)) +  
  geom_violin(trim = FALSE, fill = "grey70", alpha = .5) +  
  labs(title = "Graphique en forme de violon")  
violin
```



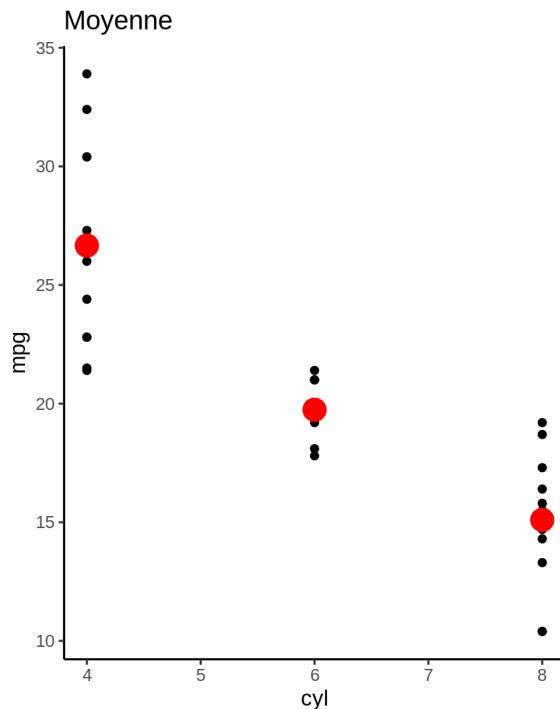
# Graphique en violon: `geom_violin()` + `_boxplot()` + `_jitter()`

```
violin +
  geom_jitter(shape = 16, position = position_jitter(0.2), alpha = .3) +
  geom_boxplot(width = .05)
```

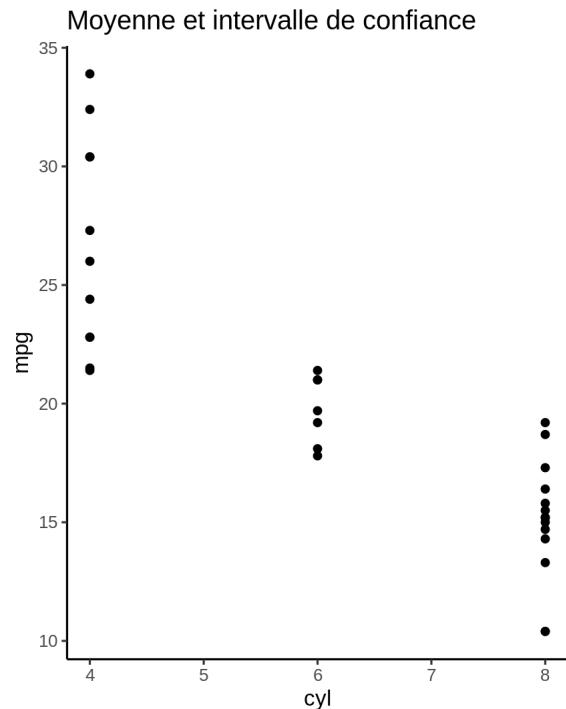


# Résumer des valeurs: `stat_summary()`

```
ggplot(mtcars, aes(cyl, mpg)) +  
  geom_point() +  
  stat_summary(fun = "mean", geom = "point",  
               colour = "red", size = 5)  
  labs(title = "Moyenne")
```



```
ggplot(mtcars, aes(cyl, mpg)) +  
  geom_point() +  
  stat_summary(fun.data = "mean_cl_boot",  
               colour = "red", size = 5)  
  labs(title = "Moyenne et intervalle de confiance")
```



# Résumer des valeurs

Voir aussi `geom_errorbar()`, `geom_pointrange()`, `geom_linerange()`,  
`geom_crossbar()` pour que les géométries puissent afficher des données  
résumées.

# Créer des cartes: `geom_map()`

```
crimes <- data.frame(state = tolower(rownames(USArrests)), USArrests)
crimesm <- reshape2::melt(crimes, id = 1)

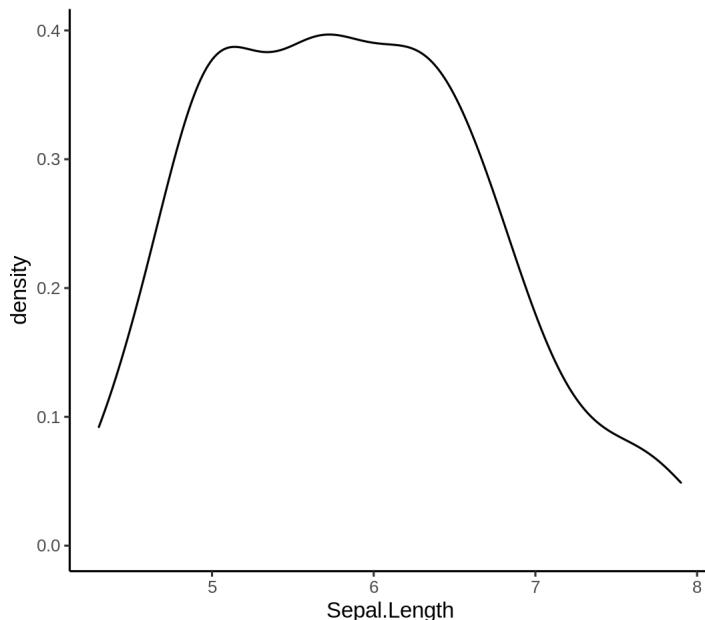
library(maps)
states_map <- map_data("state")

ggplot(crimes, aes(map_id = state)) +
  geom_map(aes(fill = Murder), map = states_map) +
  expand_limits(x = states_map$long, y = states_map$lat) +
  coord_map()
```

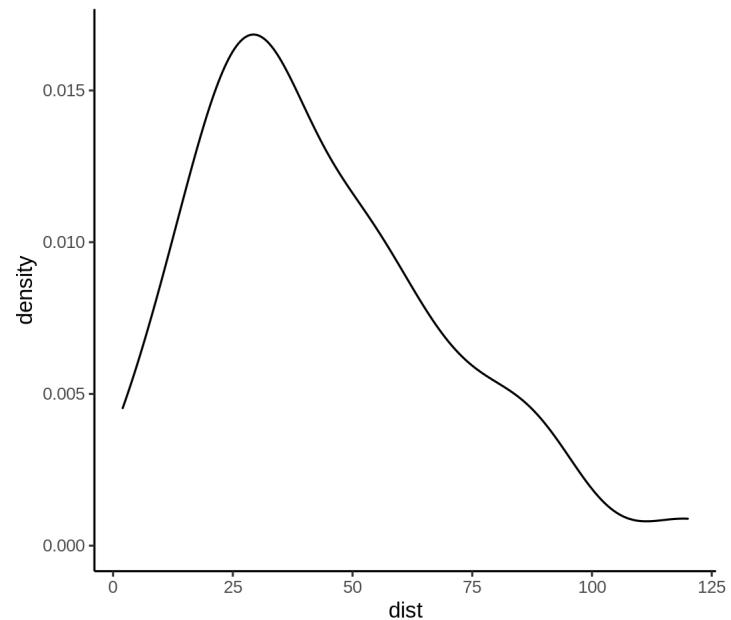
# Graphique de densité: `geom_density()`

Un graphique de densité montre la distribution d'une variable numérique et il ne prend qu'un ensemble de valeurs numériques comme entrée.

```
iris.dens <- ggplot(iris, aes(Sepal.Length)) +  
  geom_density()  
iris.dens
```



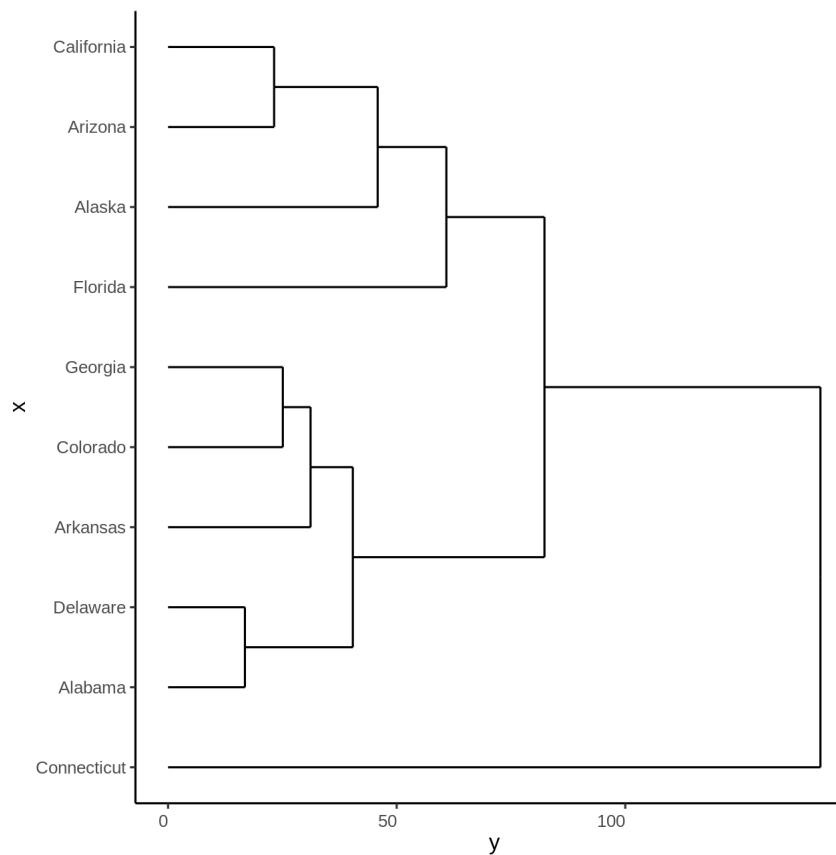
```
cars.dens <- ggplot(cars, aes(dist)) +  
  geom_density()  
cars.dens
```



# Dendrogramme: `ggdendrogram( )`

```
library(ggdendro)
USArrests.short = USArrests[1:10, ]
hc <- hclust(dist(USArrests.short), "ave")

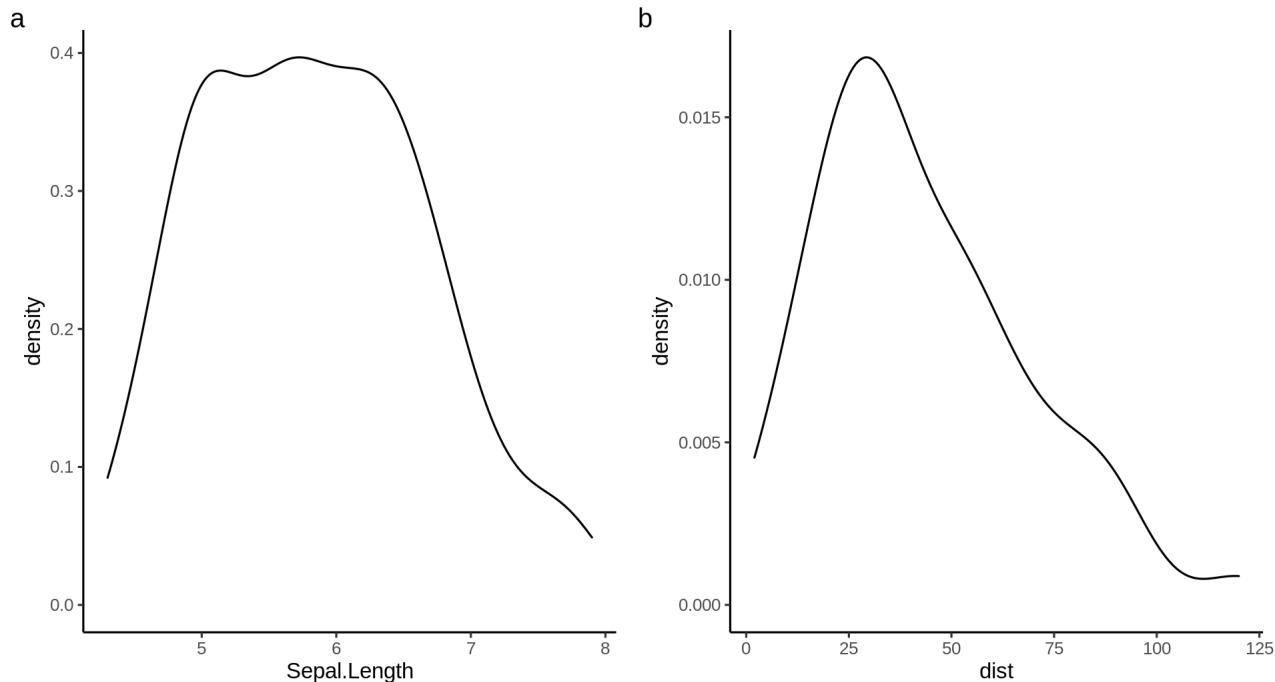
ggdendrogram(hc, rotate = TRUE, theme_dendro = FALSE)
```



# Disposition des graphiques: patchwork

## Ajouter plusieurs graphiques sur une seule figure

```
# install.packages("patchwork")
library(patchwork)
iris.dens + cars.dens +
  plot_annotation(tag_levels = 'a')
```





# Défi final

Créez votre propre graphique et suivez ces recommandations :

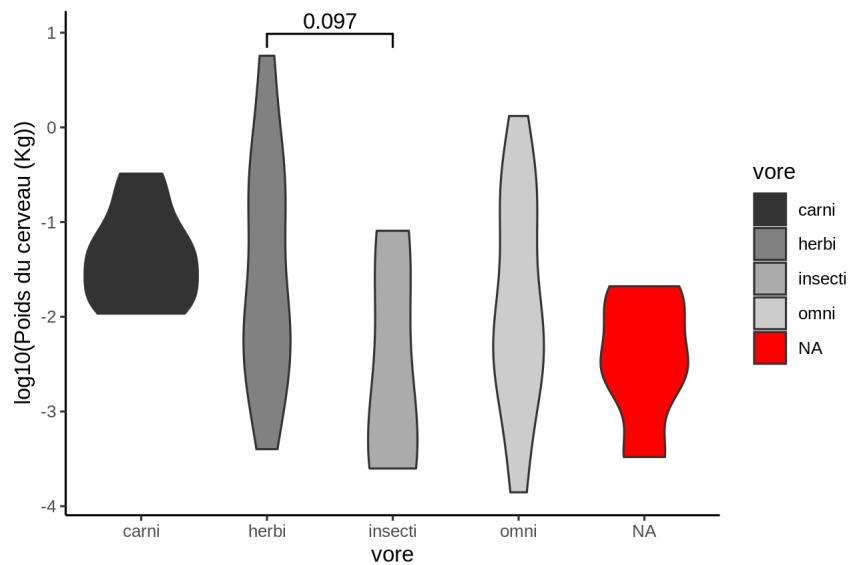
- Jeu de données : n'importe lequel (recommandation : utilisez votre jeu de données)
- Explorez un nouveau `geom_*` et d'autres couches de graphiques

Utilisez les liens suivants pour obtenir des conseils :

- <https://ggplot2.tidyverse.org/reference/index.html>
- <http://shinyapps.stat.ubc.ca/r-graph-catalog/>

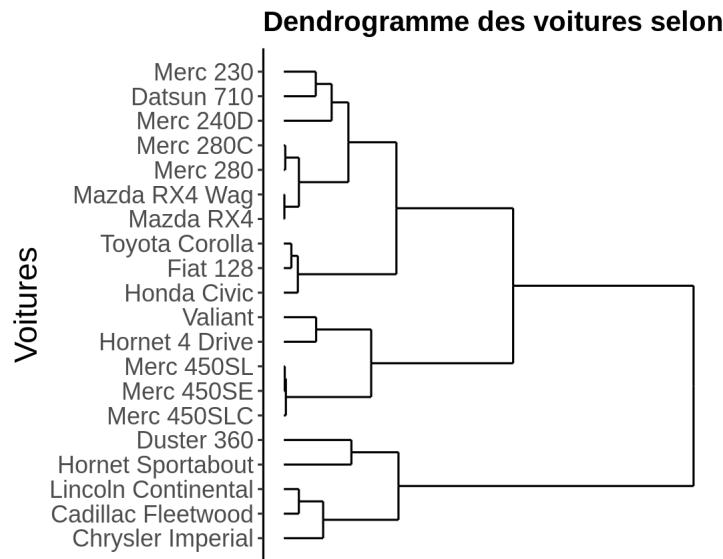
# Défi final: Solution exemple #1

```
data(msleep)
msleep.defi4 <- ggplot(msleep, aes(vore, log10(brainwt), fill = vore))
msleep.defi4 + geom_violin() +
  geom_signif(comparisons = list(c("herbi", "insecti")))) +
  labs(main = "Poids du cerveau pour les différents -vores", y = "log10(Poids du cer-
scale_fill_grey() +
theme_classic()
```



# Défi final: Solution exemple #2

```
data(mtcars)
mtcars.short <- mtcars[1:20,]
mtcars.short.hc <- hclust(dist(mtcars.short), "complete")
dendro.defi4 <- ggdendrogram(mtcars.short.hc, rotate = TRUE, theme_dendro = FALSE)
dendro.defi4 + ggtitle("Dendrogramme des voitures selon les spécifications du moteur")
xlab("Voitures") +
theme(axis.title.y = element_text(size = 16), axis.title.x = element_blank(),
      axis.text.x = element_blank(), axis.text.y = element_text(size = 12),
      plot.title = element_text(size = 14, face = "bold"))
```

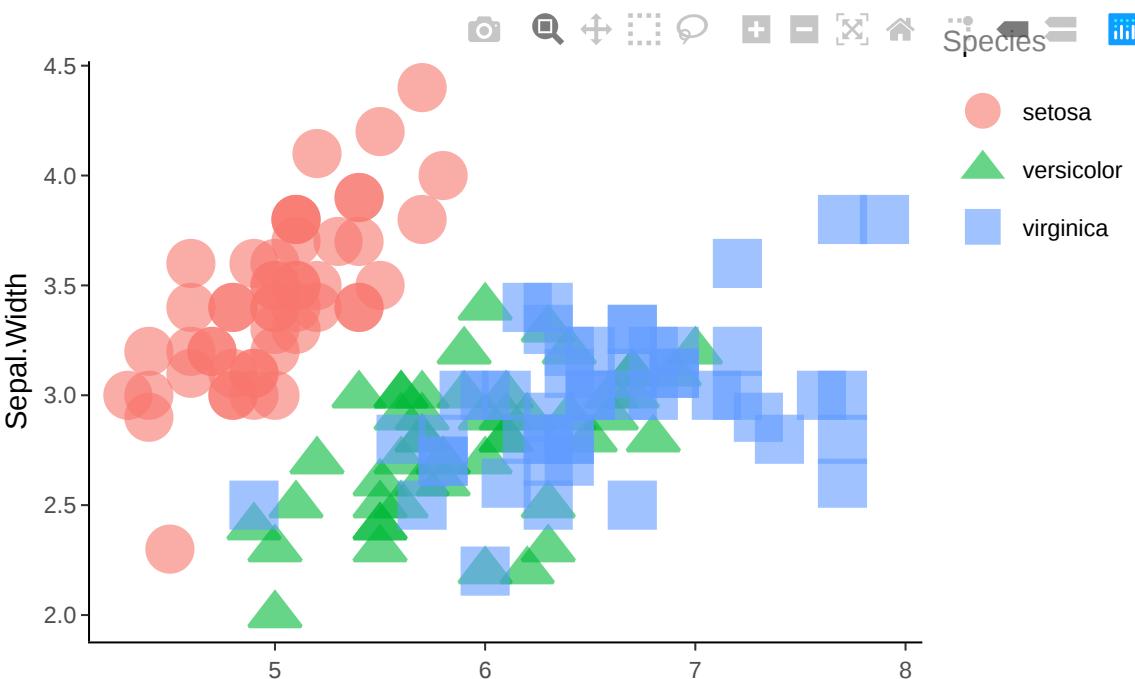


# Divers : graphiques interactifs utilisant `plotly()`

Sélectionnez les données sur les espèces à afficher directement dans la légende

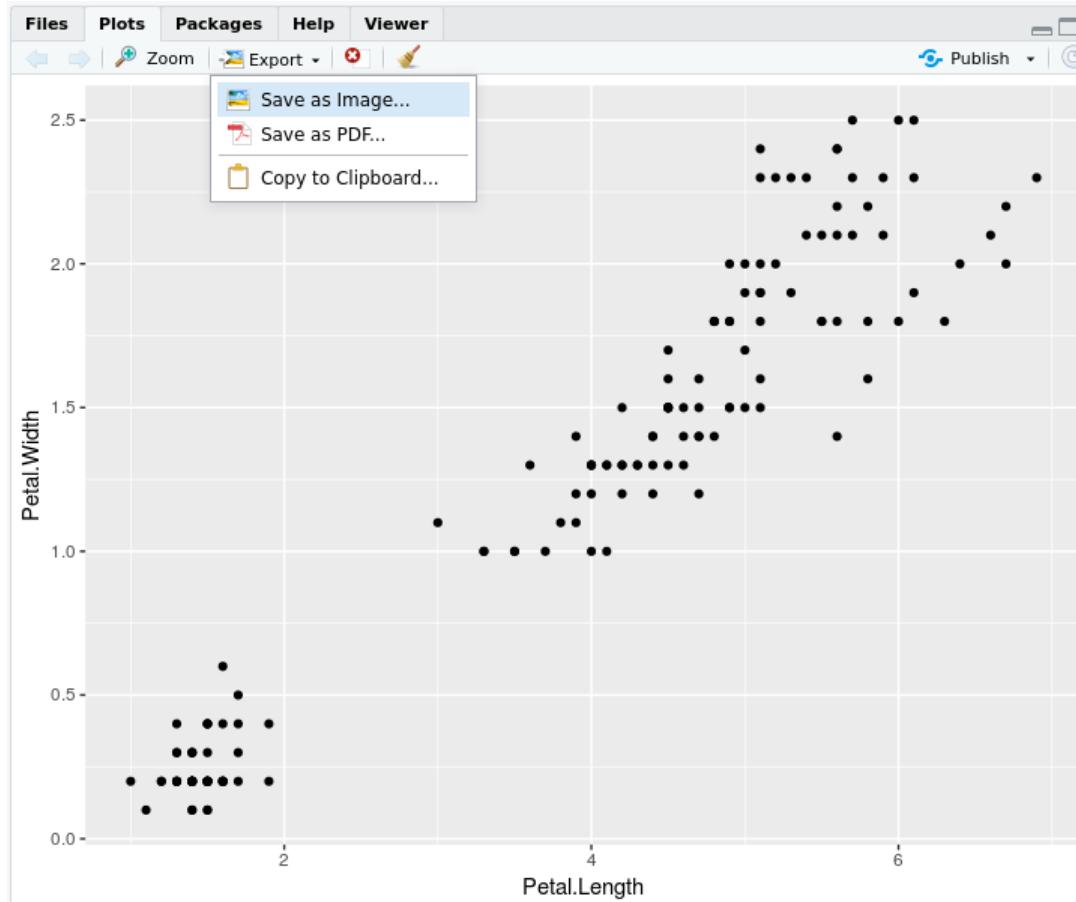
```
library(plotly)
p <- ggplot(iris,
            aes(x = Sepal.Length, y = Sepal.Width, colour = Species, shape = Species))
            geom_point(size = 6, alpha = 0.6)

ggplotly(p)
```



# Sauvegarder vos graphiques avec ggplot2

# Sauvegarder des graphiques en RStudio



Pensez à la marge du document que vous utilisez. Si vous redimensionnez l'image après l'avoir enregistrée, les étiquettes et le texte changeront également de taille, ce qui pourrait être difficile à lire.

# Sauvegarder des graphiques à l'aide de fonction R

`ggsave()` écrit directement dans votre répertoire de travail et vous permet de spécifier le nom du fichier, les dimensions du graphique, la résolution, etc.

```
mon_1er_graph <- ggplot(iris, aes(Petal.Length, Petal.Width)) +  
  geom_point()  
  
ggsave("mon_1er_graph.pdf",  
       mon_1er_graph,  
       height = 8.5, width = 11, units = "in", res = 300)
```

NB: Le format vectoriel (par exemple, pdf, svg) est plus flexible que le format raster (jpeg, png, ...) si l'image doit être modifiée par la suite.

# Sauvegarder des graphiques à l'aide de fonction R

Autres méthodes de sauvegarde des figures: voir [?pdf](#) [?jpeg](#)

```
pdf("./graph_du_jour.pdf")
print(mon_1er_graph) # la fonction print est nécessaire
graphics.off()
```

# Sauvegarder des graphiques à l'aide de fonction R

**Tip:** Les fonctions `quartz()` (sur mac) ou `window()` (sur pc) facilitent la mise en forme avant de sauvegarder avec `ggsave()`!

Merci d'avoir participé !

