Formation ggplot2

Sébastien Déjean

www.math.univ-toulouse.fr/~sdejean

Contents


```
library(ggplot2)
```

Premier graphique

Au début, il existe mais il ne montre rien...

```
MonPremierGraphique <- ggplot(data=iris)
MonPremierGraphique
```


Je ne vois rien ! Pourtant l'objet `MonPremierGraphique` existe et il contient plein de choses.

```
summary(MonPremierGraphique)
```

```
## data: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species
##   [150x5]
```

```
## faceting: <ggproto object: Class FacetNull, Facet, gg>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map_data: function
##     params: list
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train_scales: function
##     vars: function
##     super:  <ggproto object: Class FacetNull, Facet, gg>
```

```
typeof(MonPremierGraphique)
```

```
## [1] "list"
```

```
names(MonPremierGraphique)
```

```
## [1] "data"        "layers"      "scales"      "mapping"     "theme"
## [6] "coordinates" "facet"       "plot_env"    "labels"
```

```
class(MonPremierGraphique)
```

```
## [1] "gg"     "ggplot"
```

```
head(MonPremierGraphique$data)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
MonPremierGraphique$layers
```

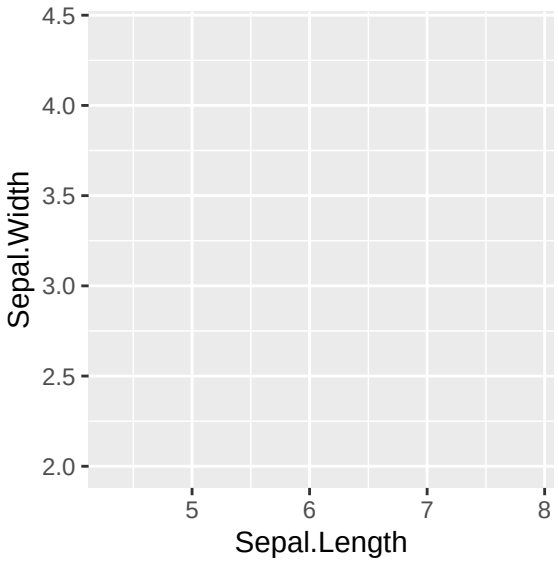
```
## list()
```

HW a dit : *The plot is not ready to be displayed until one layer is added.*

OK, donc c'est normal si je ne vois rien, car le composant `layers` de l'objet `MonPremierGraphique` ne contient rien.

... alors j'essaie d'y mettre des choses...

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



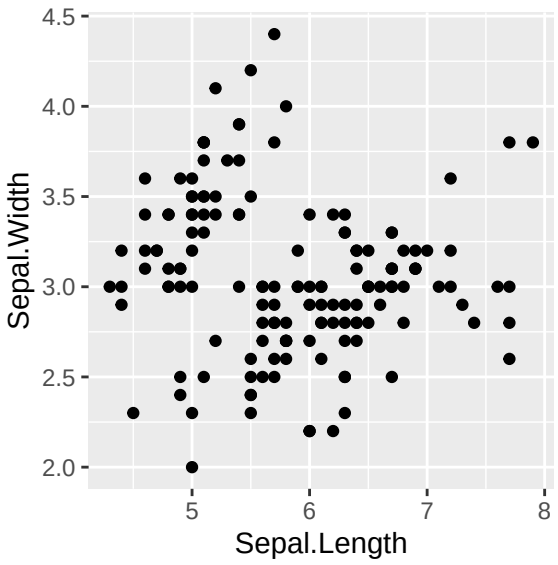
C'est mieux, je vois des axes avec les variables que je veux représenter en x et y, mais rien de plus.

> HW a dit : *A minimal layer may do nothing more than specify a* ***geom****, a way of visually representing the data.*

... et finalement, je vois !

Alors, je vais ajouter un `layer`. Pour cela, plein de fonctions existent pour dire quels objets géométriques, je veux utiliser pour la représentation. Soyons modestes, commençons par des points avec la fonction `geom_point()`.

```
MonPremierGraphique + geom_point()
```



Youpi, trompettes et confettis ! Voilà, mon premier graphique réalisé avec `ggplot2`.

Sauvons-le et auscultons-le.

```
MonPremierGraphiqueLeVrai <- MonPremierGraphique + geom_point()
summary(MonPremierGraphiqueLeVrai)
```

```
## data: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width, Species
```

```
##   [150x5]
## mapping:  x = ~Sepal.Length, y = ~Sepal.Width
## faceting: <ggproto object: Class FacetNull, Facet, gg>
##     compute_layout: function
##     draw_back: function
##     draw_front: function
##     draw_labels: function
##     draw_panels: function
##     finish_data: function
##     init_scales: function
##     map_data: function
##     params: list
##     setup_data: function
##     setup_params: function
##     shrink: TRUE
##     train_scales: function
##     vars: function
##     super:  <ggproto object: Class FacetNull, Facet, gg>
## -----------------------------------
## geom_point: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
```

```
MonPremierGraphiqueLeVrai$layers
```

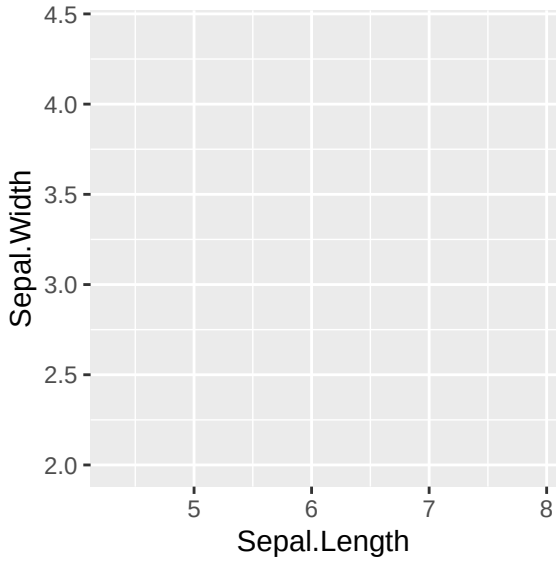
```
## [[1]]
## geom_point: na.rm = FALSE
## stat_identity: na.rm = FALSE
## position_identity
```

Il a bien un `layer` et c'est pourquoi je peux voir quelque chose.

Pour mieux (?) comprendre ce qui se passe

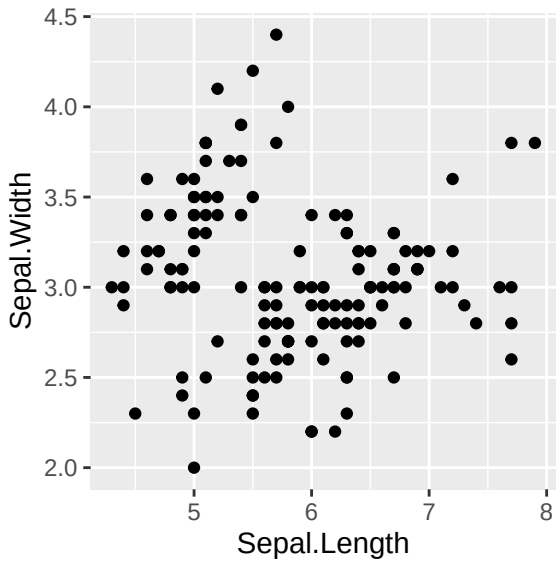
Reprenons l'objet `MonPremierGraphique`

```
MonPremierGraphique
```



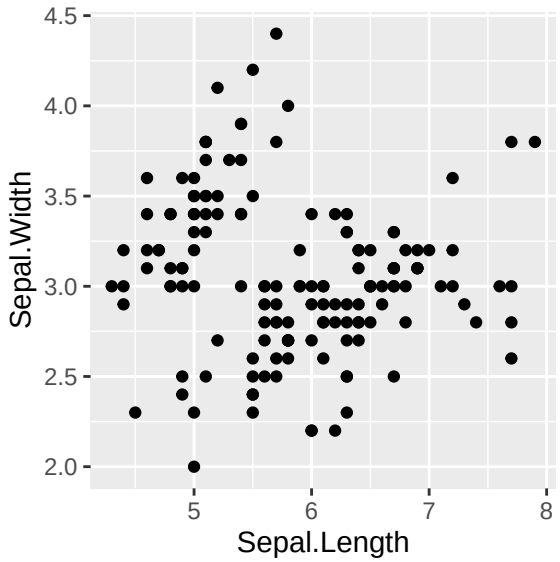
On sait qu'il n'affiche rien car il n'a pas de `layer`. On a ajouté un `layer` avec la fonction `geom_point()`.

```
MonPremierGraphique + geom_point()
```



Mais on aurait aussi bien pu le faire avec la fonction `layer()`, fonction dans laquelle on spécifie tous les attributs d'un `layer` : `mapping`, `data` (ici c'est inutile car on les a déjà spécifié dans l'objet `MonPremierGraphique`, `geom`, `stat` et `position`.

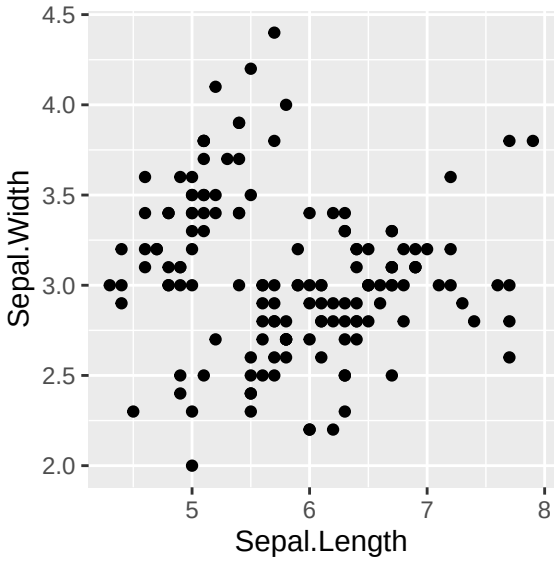
```
MonPremierGraphique +
  layer(
    mapping = NULL,
    data = NULL,
    geom = "point",
    stat = "identity",
    position = "identity"
)
```



En fait, la fonction `geom_point()` nous simplifie la vie en appelant elle-même la fonction `layer()` avec les paramètres qu'il faut.

A noter qu'il existe une façon de faire la même chose en cachant encore plus les choses : la fonction `qplot()`.

```
qplot(Sepal.Length, Sepal.Width, data=iris)
```



Mais, dans la préface à la 2eme édition de *`ggplot:` Elegant Graphics for Data Analysis*

> HW a dit : *Switched from `qplot()` to `ggplot()` in the introduction, Chapter 2. Feedback indicated that `qplot()` was a crutch: it makes simple plots a little easier, but it doesn???t help with mastering the grammar.*

Et effectivement, en utilisant `qplot()`, on garde à peu près nos habitudes liées à `plot()` mais la *layered grammar* est complètement cachée.

Bilan

Pour faire un graphique :

- les données sont stockées dans un `data.frame` et pas autrement, ce n'est pas négociable !
- les `aesthetics mappings` lient des variables du jeu de données à des propriétés visuelles (utiliser la fonction `aes()`).
- seul l'ajout d'un `layer` permet de visualiser quelque chose (utiliser les fonctions `geom_*()`).
- Ce qui donne une syntaxe comme celle-ci : `ggplot(iris, aes(x=Sepal.Length, y=Sepal.Width)) + geom_point()`

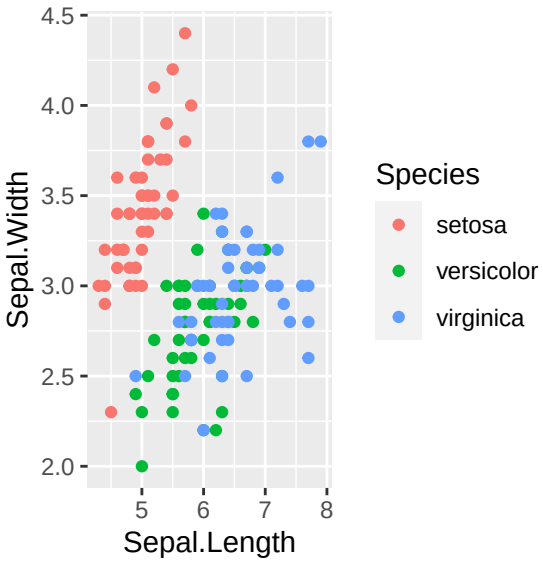
Améliorons un peu ce premier graphique

Un peu de couleur

J'aimerais bien que les points soient colorés en fonction de la variable `Species` du jeu de données `iris`.

Il suffit pour cela de préciser un lien entre cette variable du jeu de données et la propriété visuelle `colour` au moyen de la fonction `aes()`.

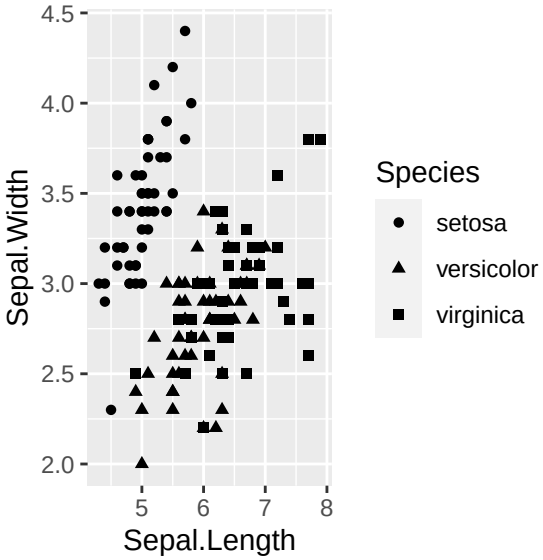
```
MonPremierGraphiqueLeVraiEnCouleur <-
  MonPremierGraphiqueLeVrai + aes(colour = Species)
MonPremierGraphiqueLeVraiEnCouleur
```



En changeant la forme

En fait, les couleurs c'est bien, mais si mes lecteurs les distinguent mal, je préfère changer la forme (`shape`) plutôt que la couleur.

```
MonPremierGraphiqueLeVraiEnFormes <-
  MonPremierGraphiqueLeVrai + aes(shape = Species)
MonPremierGraphiqueLeVraiEnFormes
```



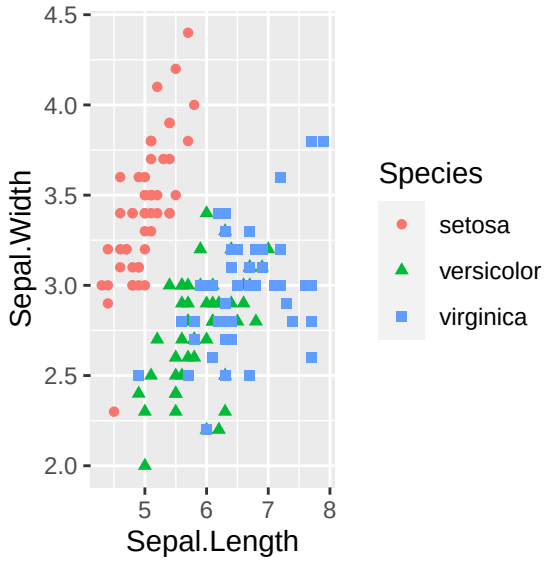
Mouais, pas très lisible. Et si je combinais les deux...

Un peu de couleur... en changeant la forme

Les noms deviennnent un peu long, je vais me calmer un peu et je reprends depuis le début pour voir si j'ai compris.

```
PointCouleurForme <- ggplot(data = iris,
                            aes(x=Sepal.Length, y = Sepal.Width,
                                colour = Species,
```

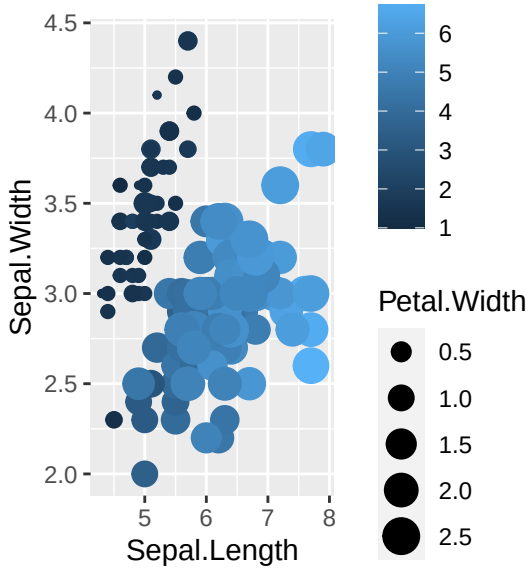
```
                                shape = Species)) + geom_point()
PointCouleurForme
```



Et bien voilà !

Et si je m'amusais avec les autres variables

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



Ouh là, là… qu'ai-je donc fait ? Reprenons calmement :

- je suis toujours dans un nuage de points (`geom_point()`) avec `Sepal.Length` en abscisses (`aes(x= )`) et `Sepal.Width` en ordonnées (`aes(y= )`).
- dans la fonction `aes()`, j'ai lié la couleur à la variable `Petal.Length` obtenant ansi une dégradé de bleu (plus la valeur de `Petal.Length` est élevée, plus le symbole est clair).

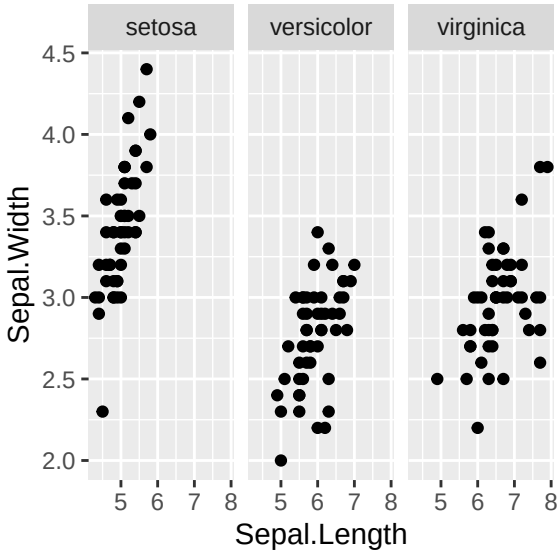
- toujours dans la fonction `aes()`, j'ai lié la taille à la variable `Petal.Width` obtenant ansi des points d'autant plus grands que la valeur de `Petal.Width` est élevée.
- pour le même prix, j'ai obtenu les deux légendes associées.
- tous ces éléments illustrent la notion de `scaling` qui contrôle le lien entre les données et un attribut `aesthetic`. Tout attribut `aesthetic` a besoin d'une échelle (parfois - souvent - on ne s'en rend pas compte parce qu'une valeur par défaut est prévue).

En vue éclatée

Ok, je peux m'amuser à personnaliser mon graphique avec des couleurs, des formes, des tailles différentes. Et si je restais sobre en présentant un graphique par modalité de la variable `Species`.

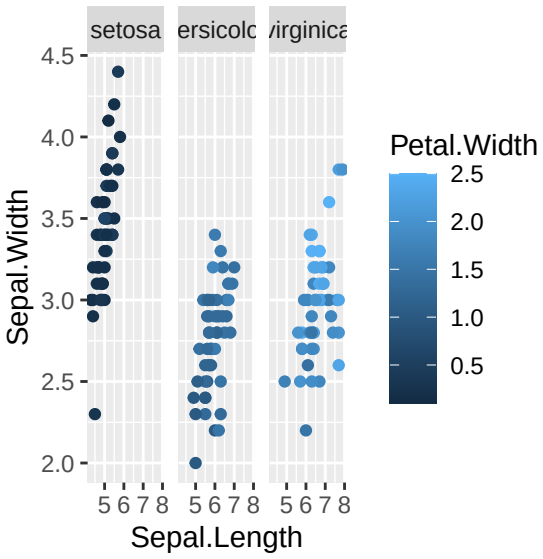
Très facile (comme le reste !), il suffit d'utiliser le `facetting`.

```
MonPremierGraphiqueLeVrai + facet_wrap(~Species)
```



Il va de soi que l'on peut combiner tout ça.

```
MonPremierGraphiqueLeVrai +
  aes(colour = Petal.Width) +
  facet_wrap(~Species)
```



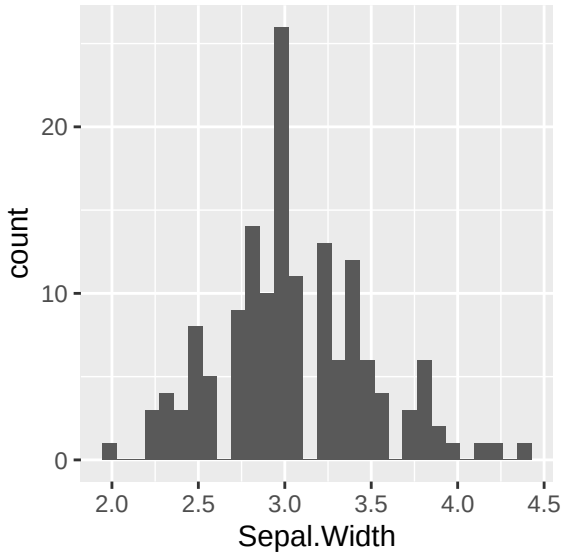
C'est bien mais il n'y a pas que les nuages de points dans la vie

Histogramme

Pour réaliser un histogramme, il suffit... de le demander. On utilise toujours le data.frame `iris` avec cette fois-ci seulement une variable (ici `Sepal.Width` en `x`).

```
HistogrammeSepalW <- ggplot(iris, aes(x=Sepal.Width)) +
  geom_histogram()
HistogrammeSepalW
```

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

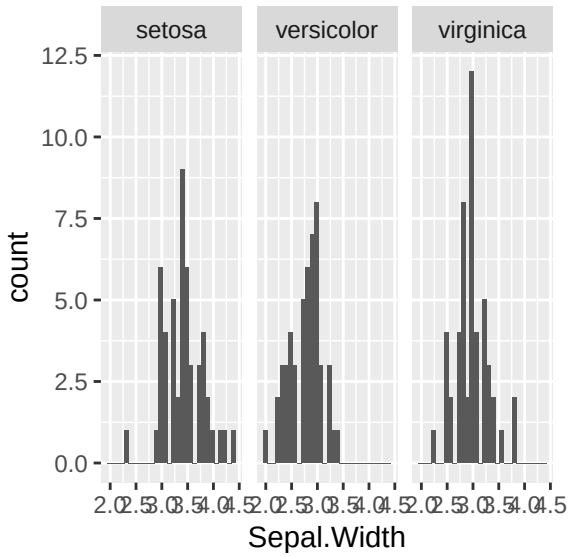


OK, pas génial, mais on a bien un histogramme. Histogramme sur lequel, on peut appliquer les recettes précédentes.

- Le `facetting` pour avoir un histogramme par modalité du facteur `Species`

```
HistogrammeSepalW + facet_wrap(~Species)
```

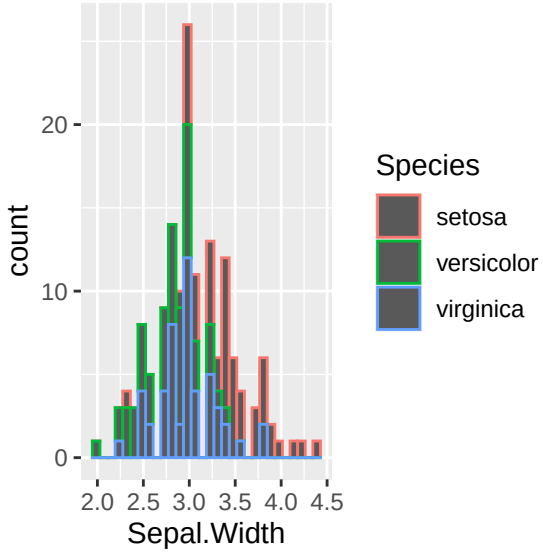
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



- et aussi des couleurs (même si l'interprétation n'en est pas forcément améliorée)

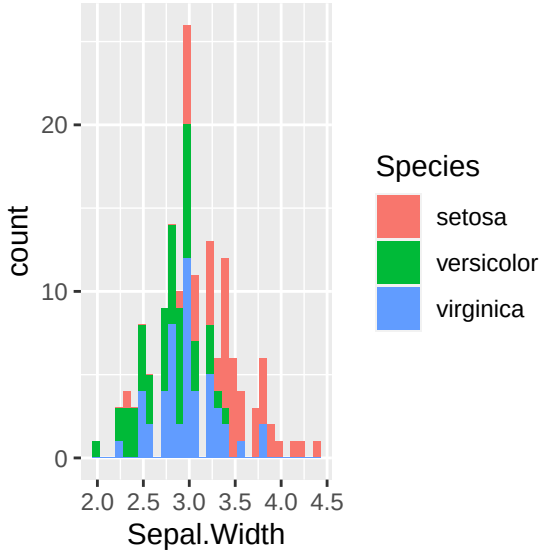
```
HistogrammeSepalW + aes(colour=Species)
```

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




```
HistogrammeSepalW + aes(fill=Species)
```

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



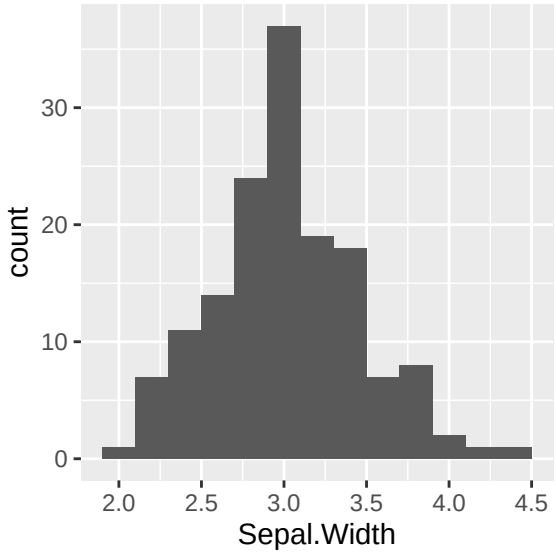
Au fait, vous avez vu qu'il nous embête avec un warning qui parle de `stat_bin()`.

> HW a dit : *The choice of bins is very important for histograms [...]. In* ***ggplot2****, a very simple heuristic is used for the default number of bins: it uses 30, regardless of the data. This is perverse, and ignores all of the research on selecting good bin sizes automatically, but sends a clear message to the user that they need to think about and experiment with the bin width.*

Et bim !

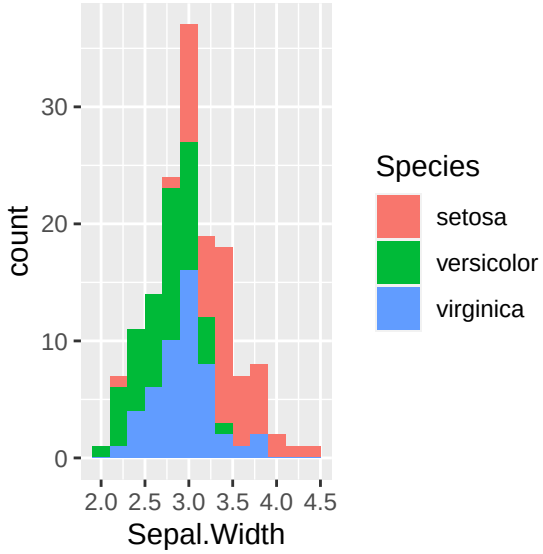
Alors, on va essayer de penser un peu et augmenter la largeur des intervalles (`binwidth`).

```
ggplot(iris, aes(x=Sepal.Width)) +
  geom_histogram(binwidth = 0.2)
```



OK, ça a l'air d'aller mieux. Et, juste manière, avec des couleurs.

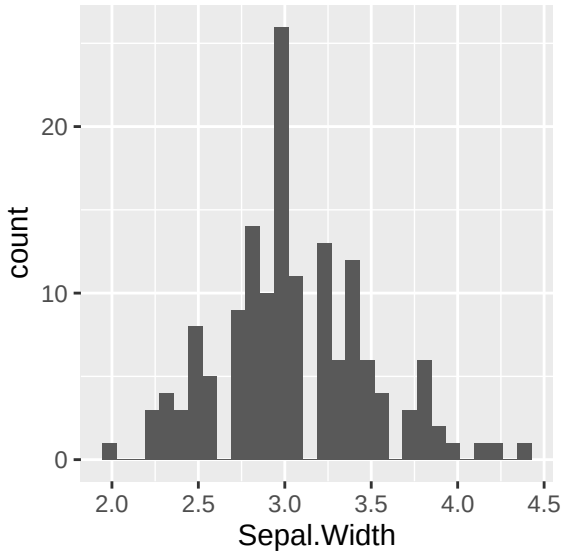
```
ggplot(iris, aes(x=Sepal.Width, fill=Species)) +
  geom_histogram(binwidth = 0.2)
```



Au fait, juste pour voir si on a compris le principe, comparons

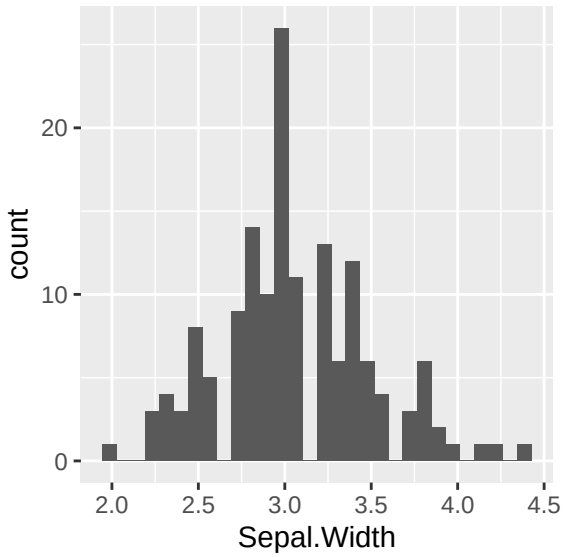
```
ggplot(iris, aes(x=Sepal.Width)) + geom_histogram()
```

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




```
ggplot(iris, aes(x=Sepal.Width)) + geom_bar(stat = "bin")
```

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

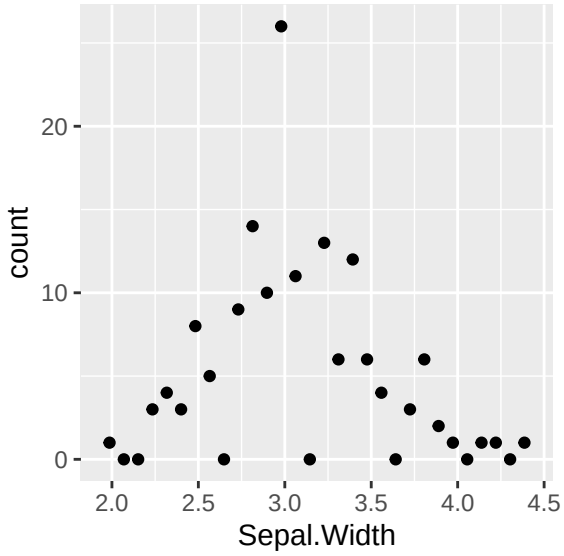


C'est la même chose ! Logique non ? Un histogramme n'est rien d'autre qu'un diagramme en bâtons prenant comme données le nombre de valeurs dans chaque *bin*.

Du coup, on peut s'amuser avec d'autres représentations graphiques de ces *counts*.

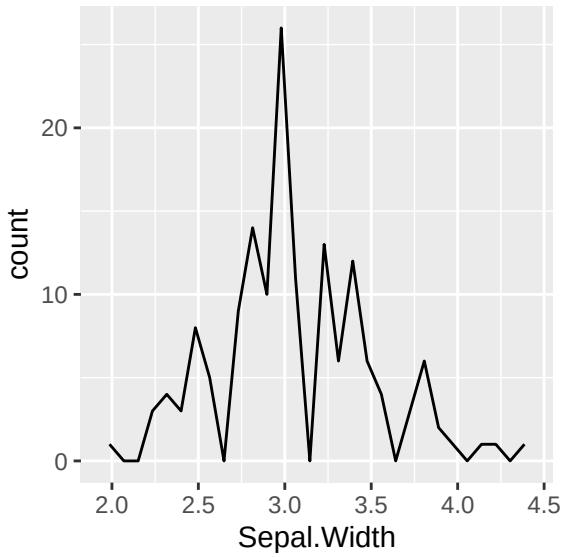
```
ggplot(iris, aes(x=Sepal.Width)) + geom_point(stat = "bin")
```

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




```
ggplot(iris, aes(x=Sepal.Width)) + geom_line(stat = "bin")
```

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

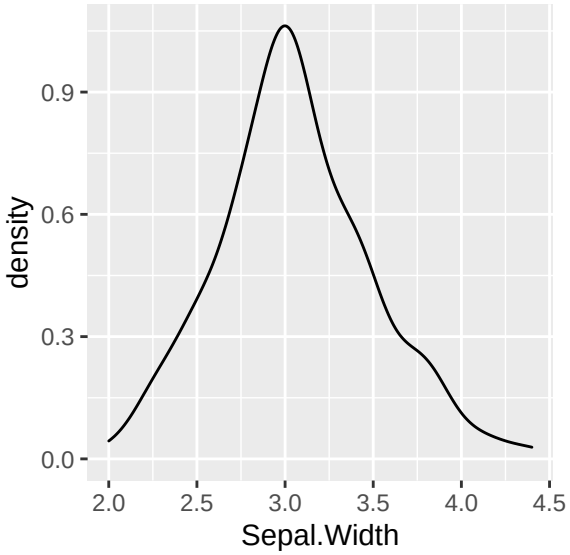


Densité

On poursuit autour de la distribution des données avec la représentation de densité grâce à la fonction... devinez... `geom_density()`.

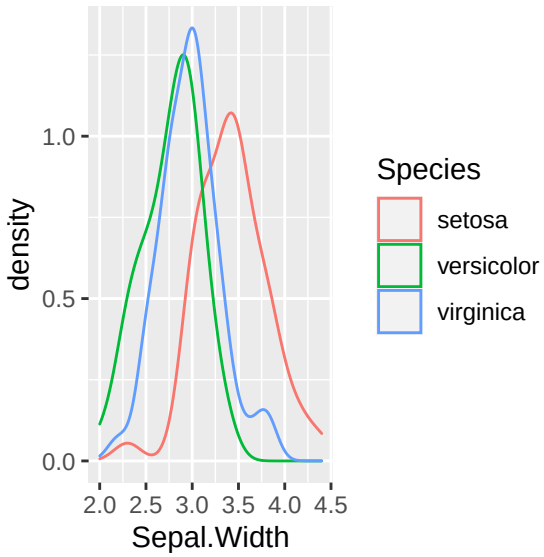
Pour démarrer, c'est facile, maintenant on gère ça tranquillement.

```
ggplot(iris, aes(x=Sepal.Width)) + geom_density()
```



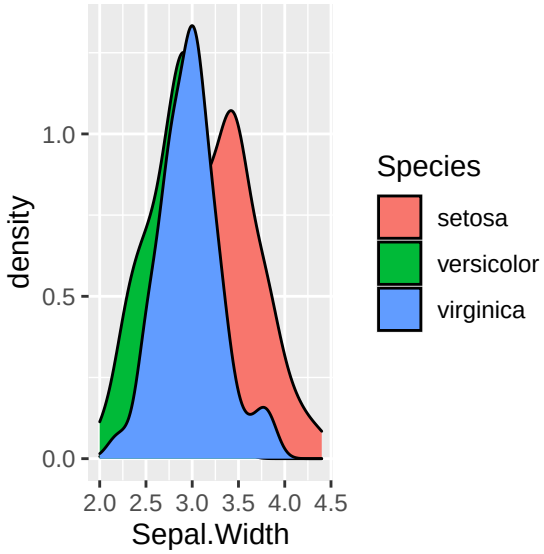
On va le faire aussi en couleur par modalité de `Species`, maintenant qu'on est à l'aise.

```
ggplot(iris, aes(x=Sepal.Width, colour = Species)) +
  geom_density()
```



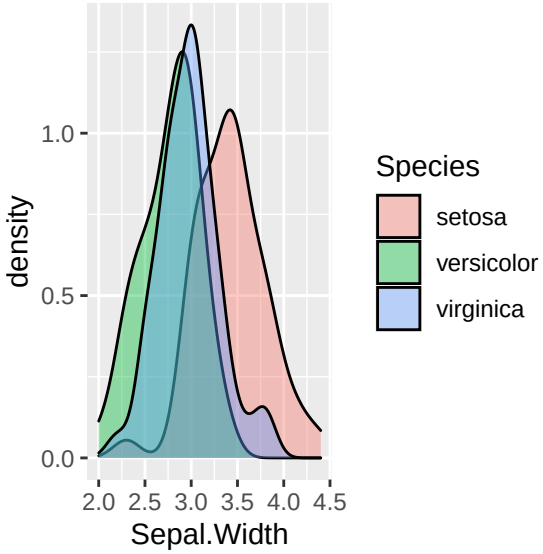
Au fait, on a vu `fill` tout à l'heure, on va regarder ce que ça donne ici

```
ggplot(iris, aes(x=Sepal.Width, fill = Species)) +
  geom_density()
```



Superbe ! Euh oui, sauf que je m'attendais à mieux en fait, mais je ne sais pas trop quoi ? Que les courbes ne se cachent les unes derrière les autres par exemple ? Oui c'est exactement ça ! Et alors, on fait comment ? On ajoute de la transparence avec le paramètre `alpha`.

```
ggplot(iris, aes(x=Sepal.Width, fill = Species)) +
  geom_density(alpha=0.4)
```

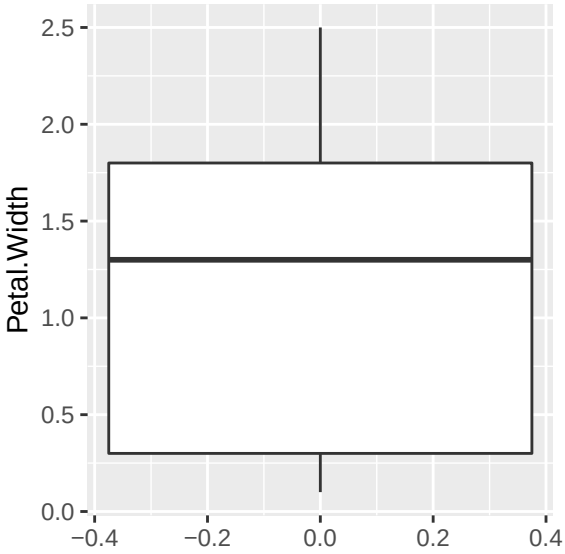


Ah ouais, là chapeau. Je prends.

Boxplot

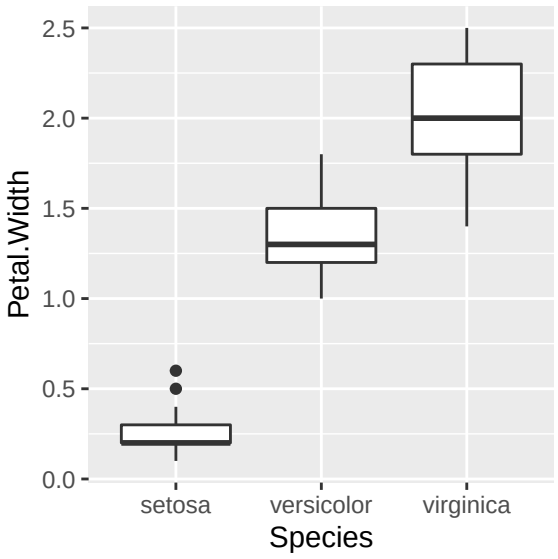
Poursuivons avec des boxplots. D'abord un seul

```
MonGrapheIris <- ggplot(iris)
MonGrapheIris + geom_boxplot(aes(y=Petal.Width))
```



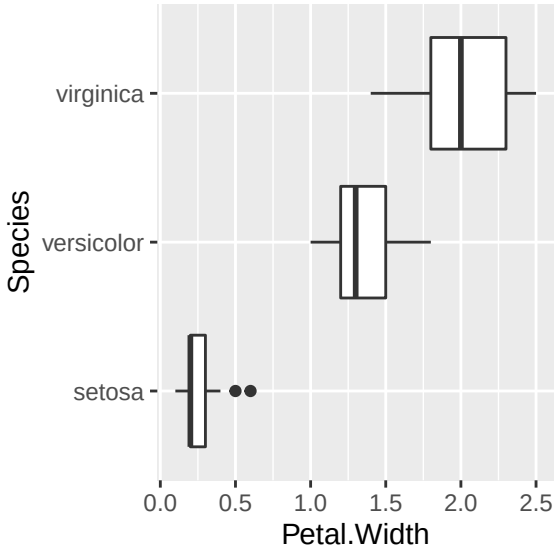
puis par modalité de `Species` encore une fois. Pour cela, il suffit de préciser que l'axe des x du graphique est lié à la variable `Species` (oui, elle est qualitative, et alors ?).

```
MonGrapheIris + geom_boxplot(aes(x = Species, y=Petal.Width))
```



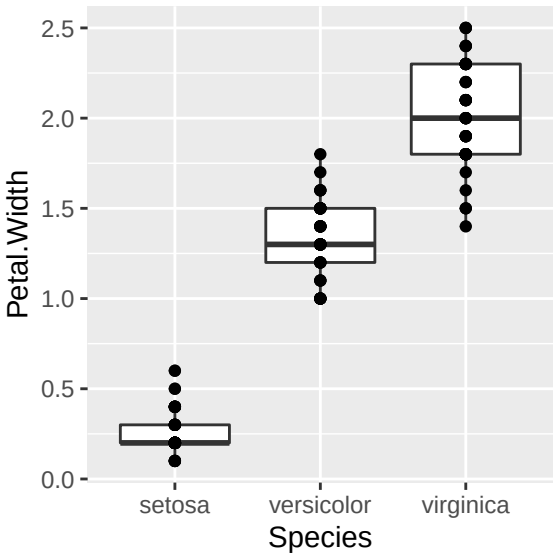
Et si je prefère des boxplots à l'horizontale, il suffit d'échanger les coordonnées.

```
MonGrapheIris + geom_boxplot(aes(x = Species, y=Petal.Width)) +
  coord_flip()
```



Ce qui est bien avec les boxplots c'est quand on peut aussi voir les points. Simple, il suffit d'ajouter un `layer` au graphique des boxplots.

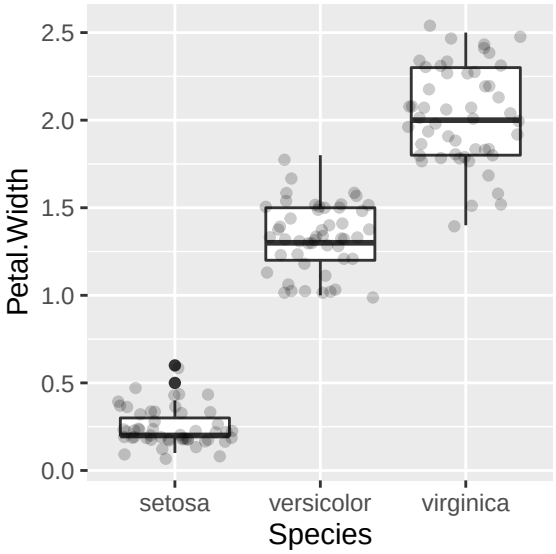
```
MonGrapheIris +
  geom_boxplot(aes(x=Species, y=Petal.Width)) +
  geom_point(aes(x=Species, y=Petal.Width))
```



Euh oui d'accord, mais c'est pas terrible, on peut pas faire mieux ? Je reformule : comment peut-on faire mieux ?

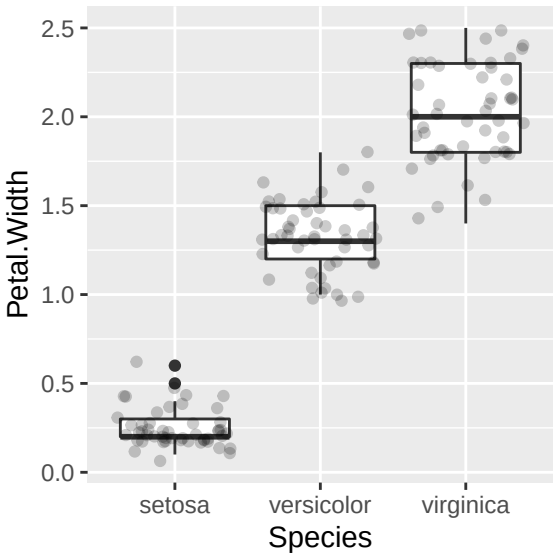
En évitant la superposition des points par exemple et en jouant aussi sur la tranparence.

```
MonGrapheIris +
  geom_boxplot(aes(x = Species, y=Petal.Width)) +
  geom_point(aes(x=Species, y=Petal.Width),
             alpha=0.2, position = "jitter")
```



J'ai écrit deux fois la même chose dans ma commande (`aes(x = Species, y=Petal.Width)`), ne pouvais-je pas m'en passer ? Si bien sûr, comme pour les calculs, on factorise. Reprenons depuis le début .

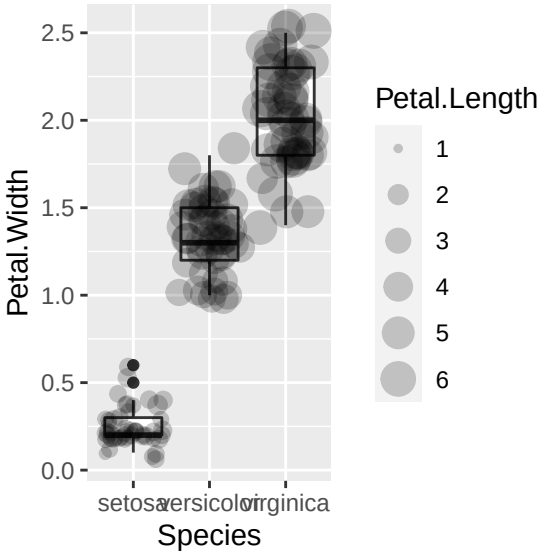
```
ggplot(iris, aes(x = Species, y=Petal.Width)) +
  geom_boxplot() +
  geom_point(alpha=0.2, position = "jitter")
```



Le fait d'utiliser `Species` en x et `Petal.width` en y étant commun aux deux graphiques que je veux réaliser, je positionne ces informations à un niveau supérieur.

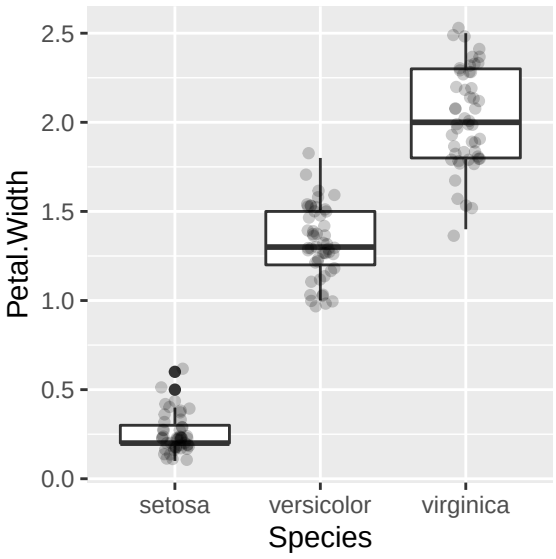
Mais je ne suis pas obligé de tout remonter, il peut y avoir des `aesthetics` propres à un `layer` comme ici où on utilise la variable `Petal.Length` pour la taille des points (et cela n'a rien à voir avec les boxplots).

```
ggplot(iris, aes(x = Species, y=Petal.Width)) +
  geom_boxplot() +
  geom_point(aes(size=Petal.Length),
             alpha=0.2, position = "jitter")
```



Sinon, il y aussi la fonction `geom_jitter()` qui fait le boulot aussi bien (mieux ?) et on peut même resserrer un peu les points.

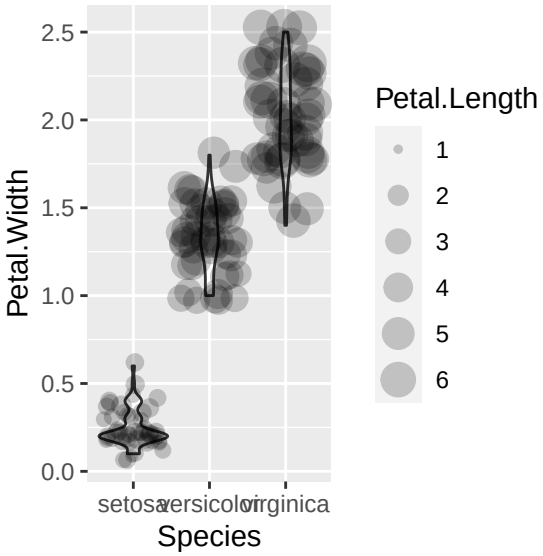
```
ggplot(iris, aes(x = Species, y=Petal.Width)) +
  geom_boxplot() +
  geom_jitter(alpha=0.2, width = 0.1)
```



Jouons du violon

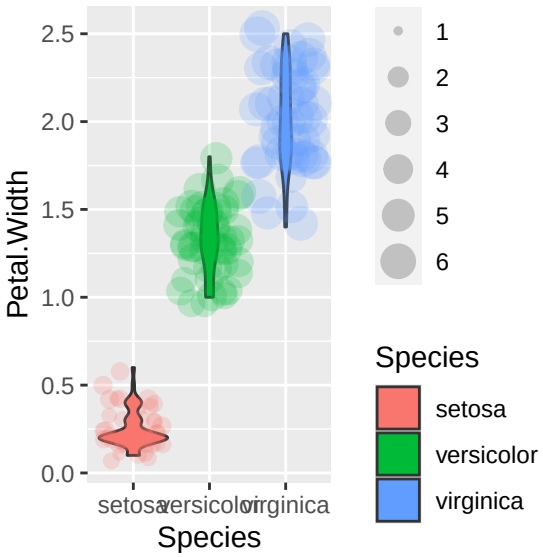
Si vous préférez les violin plot au boxplot, il suffit de remplacer `geom_boxplot` par `geom_violin`.

```
ggplot(iris, aes(x = Species, y=Petal.Width)) +
  geom_violin() +
  geom_point(aes(size=Petal.Length),
             alpha=0.2, position = "jitter")
```



Si vous préférez des violons (et des points) en couleur :

```
ggplot(iris, aes(x = Species, y=Petal.Width)) +
  geom_violin(aes(fill=Species)) +
  geom_point(aes(size=Petal.Length, colour = Species),
             alpha=0.2, position = "jitter")
```

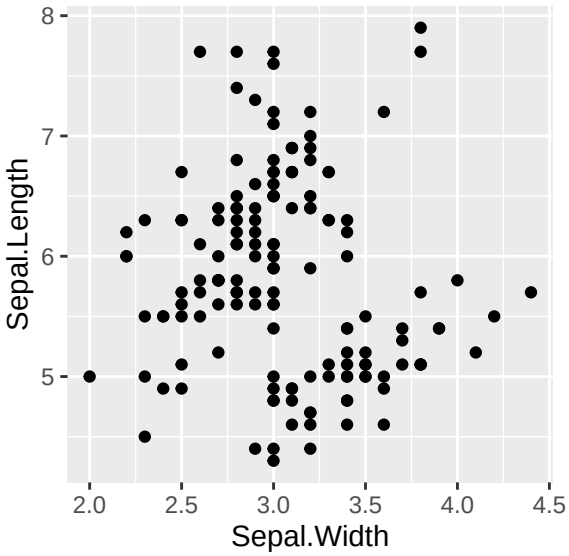


Les nuages de points : le retour

Au cas où vous ne seriez pas encore convaincus par `ggplot2`, on va revenir un moment sur les nuages de points. Et là, normalement, c'est l'*effet waouh* garanti.

On a vu quelque part qu'un `layer` a 5 parties : 1/ les données 2/ les `aesthetics`, 3/ les `geoms`, 4/ les transformations statistiques et 5/ les ajustements de position. Jusqu'à présent, on a vu un peu de tout ça, sauf des statistiques, non ? Alors, allons-y !

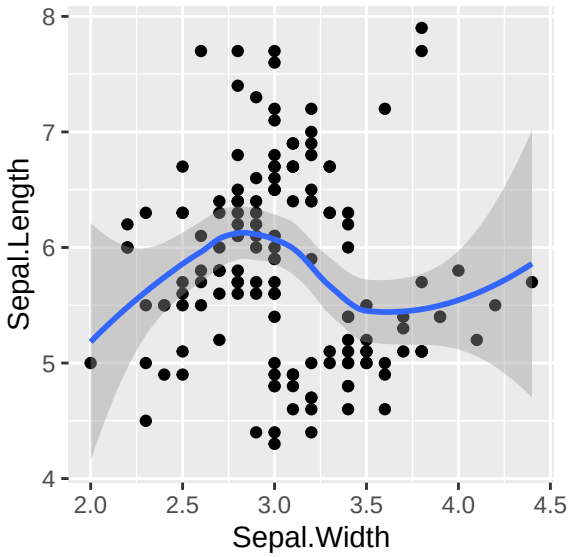
```
MonNuage <- ggplot(iris, aes(x=Sepal.Width, y = Sepal.Length)) +
  geom_point()
MonNuage
```



OK, jusqu'ici, on connaît, pas de quoi s'enflammer. Attention, le spectacle va commencer.

```
MonNuage + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Euh oui d'accord, c'est bien mais il sort d'où ce truc ? Et bien, il nous l'a dit `geom_smooth() using method = 'loess' and formula 'y ~ x'`.

Un peu trop lisse ?

```
MonNuage + geom_smooth(span=0.3)
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 3
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 0.1
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 0
```

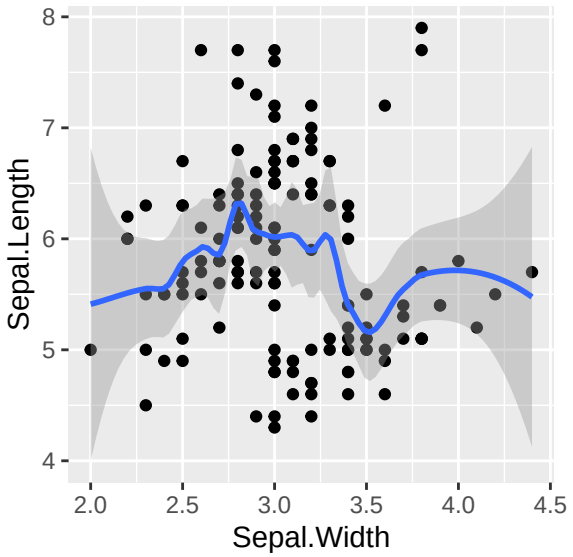
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 0.01
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at 3
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius 0.1
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition
## number 0
```

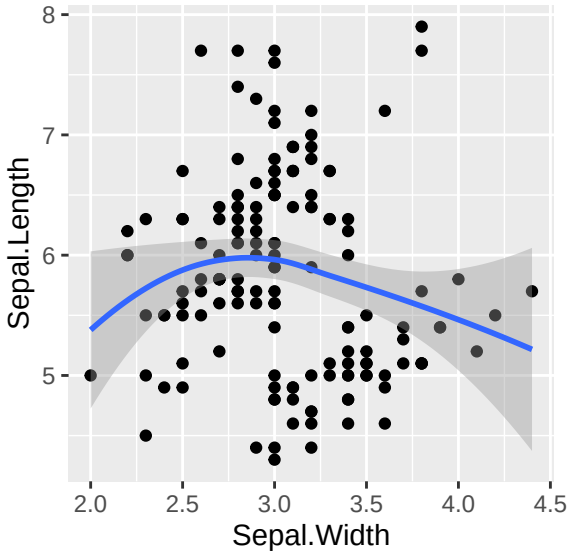
```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : There are other near
## singularities as well. 0.01
```



Pas assez ?

```
MonNuage + geom_smooth(span=1.5)
```

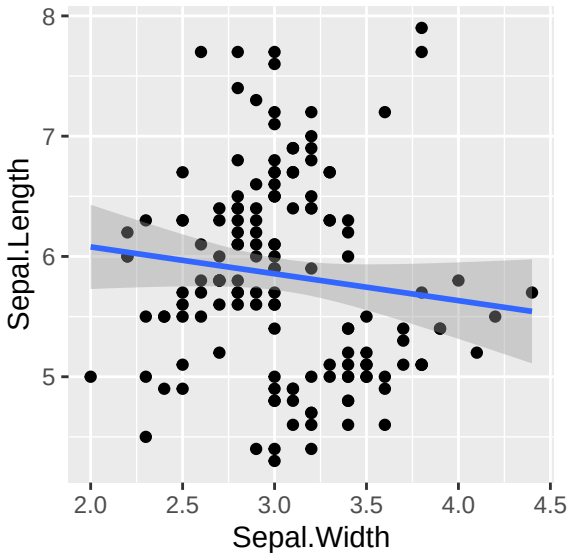
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



OK, si vous préférez une régression linéaire plutôt qu'un loess, il suffit de demander.

```
MonNuage + geom_smooth(method = "lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



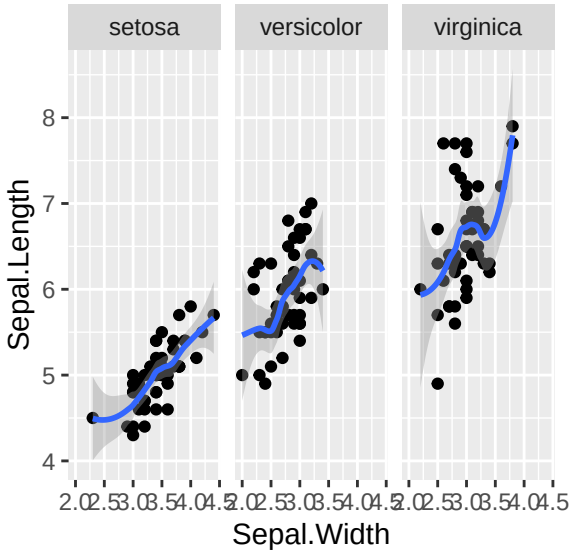
Et d'autres, voir `?geom_smooth`.

Déclinons ce que l'on sait faire.

- En facettes :

```
MonNuage + geom_smooth() + facet_wrap(~Species)
```

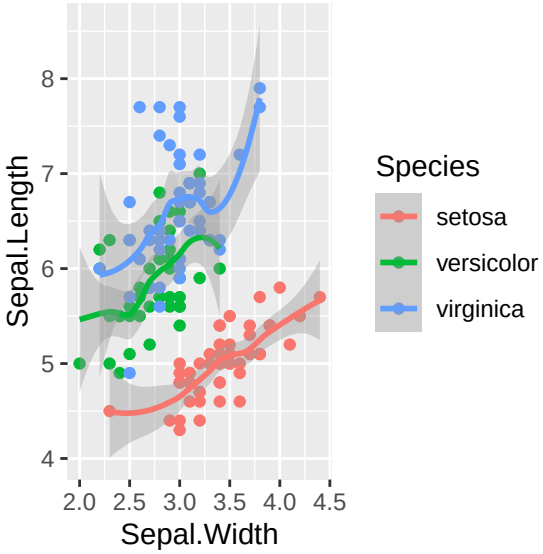
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



- En couleurs

```
MonNuage + aes(colour=Species) + geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



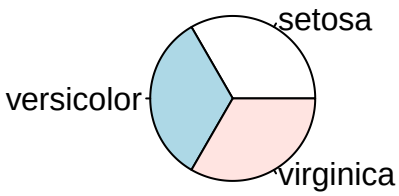
Ca commence à être sympa non ? Et noter l'aspect synthétique et concis du code.

Un petit truc pour parler des systèmes de coordonnées ou comment réaliser un camembert ?

On peut apprécier ou pas les diagrammes circulaires ou camembert ou *pie chart*, mais ce serait bien de savoir en faire avec `ggplot2`. Cherchez bien dans l'aide, vous ne trouverez pas de fonction `geom_pie()`.

Alors que R de base, fait ça très simplement.

```
pie(table(iris$Species))
```



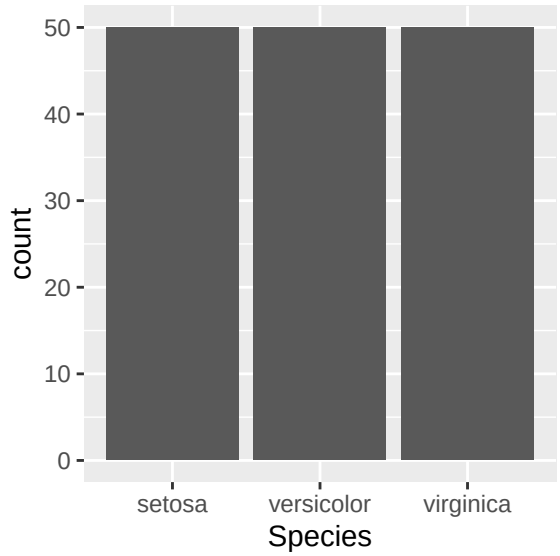
Et là, on ne serait pas un peu déçu de ne pas avoir de légende automatiquement ? C'est qu'on y prend goût à `ggplot2`.

Au fait, si ces graphiques ne sont pas trop appréciés, c'est parce qu'on leur reproche le fait de devoir interpréter des angles ce que l'oeil humain ne parvient pas forcément très bien à faire. Et c'est pourquoi, on peut lui préférer un simple diagramme en bâtons, que R de base fait aussi très bien.

```
barplot(table(iris$Species))
```

Ouais, c'est quand même un peu tristounet. Mais les diagrammes en bâtons, je sais faire avec `ggplot2`

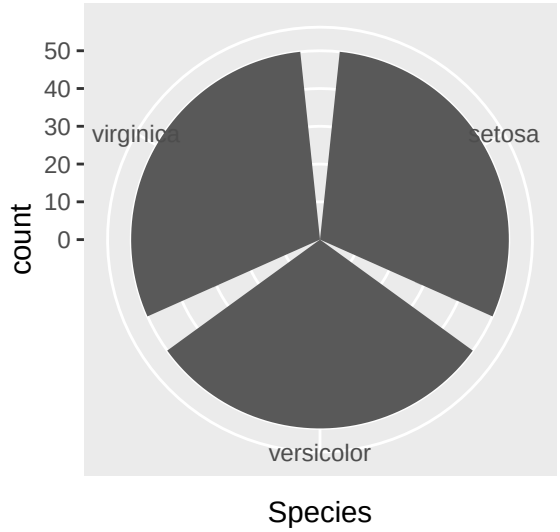
```
ggplot(iris, aes(x=Species)) + geom_bar()
```



Et là, révélation ! Un diagramme circulaire, c'est un diagramme en bâtons ! euh, non... Mais si, mais dans un repère de coordonnées polaires ! Ah, oui, euh, peut-être... [Sceptique].

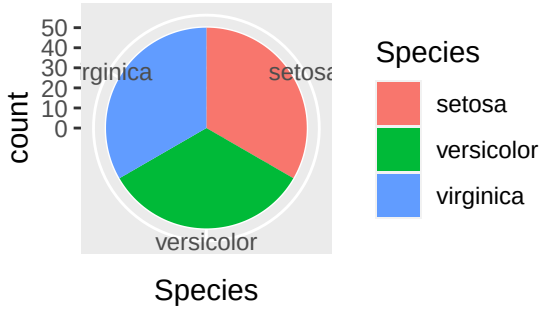
HW a dit : *In the grammar, a pie chart is a stacked bar geom drawn in a polar coordinate system.*

```
ggplot(iris, aes(x=Species)) + geom_bar() + coord_polar()
```



Ah oui ! Bizarre non ? OK, on arrange un peu tout ça.

```
ggplot(iris, aes(x=Species, fill=Species)) +
  geom_bar(width=1) +
  coord_polar()
```



Tada !

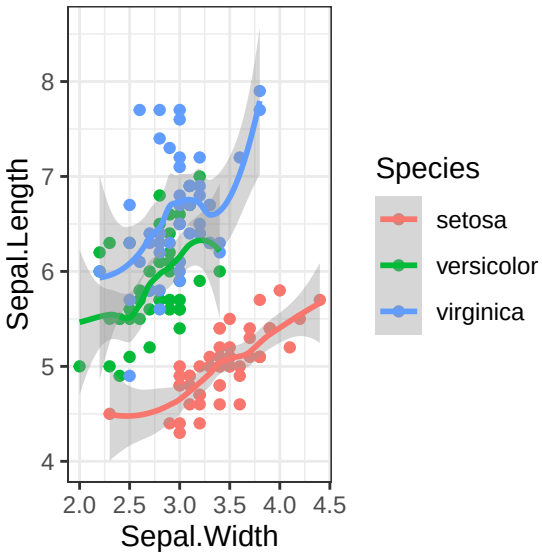
En fait, c'était juste pour parler de systèmes de coordonnées...

Et si je veux changer de style

Choisis ton thème ! Voir `help(theme_bw)`.

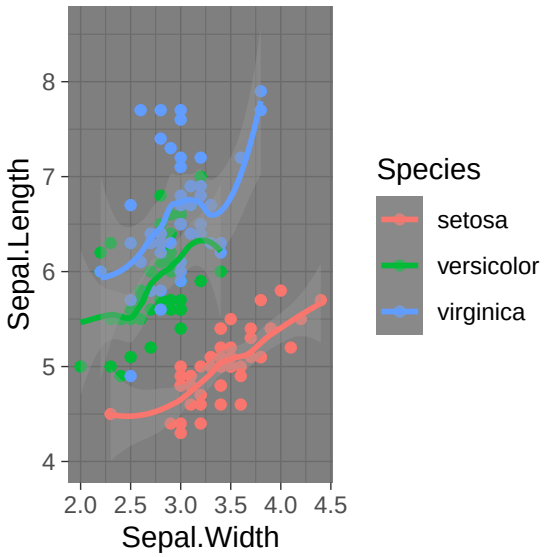
```
MonNuageTest <- MonNuage + aes(colour=Species) + geom_smooth()
MonNuageTest+ theme_bw()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



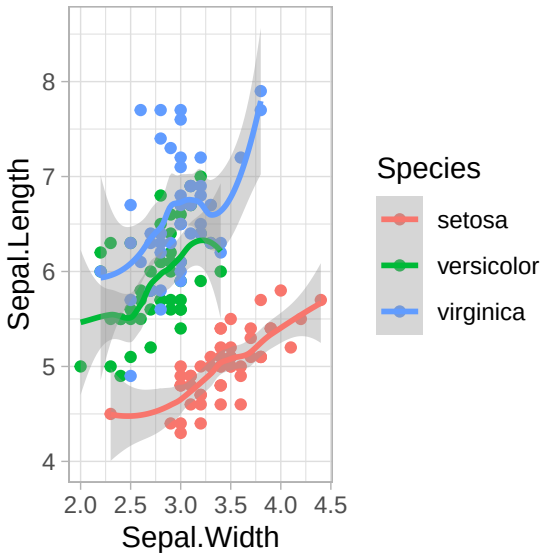

```
MonNuageTest + theme_dark()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



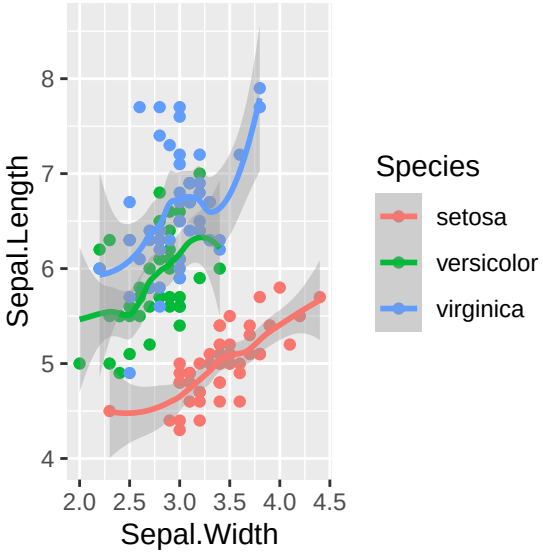

```
MonNuageTest + theme_light()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```




```
MonNuageTest + theme_gray()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



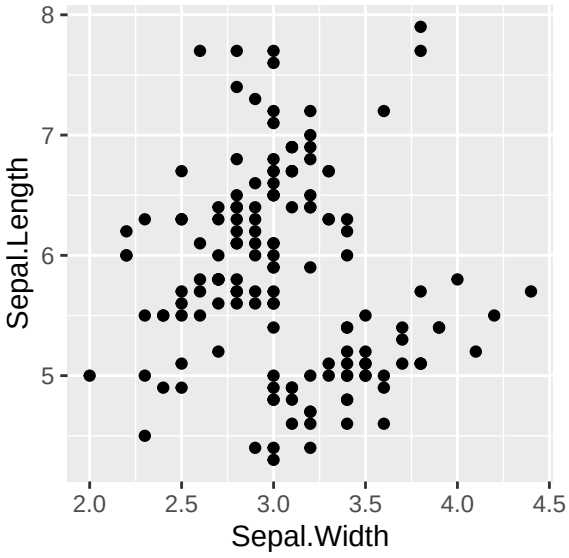
Jouons maintenant avec les échelles

Pour jouer avec les échelles, il y a de la matière ! Il suffit de regarder le nombre de fonctions `scale_*()`. Et comme on l'a vu plus tôt, chaque `aesthetic` a une échelle.

On pense d'abord aux échelles des axes

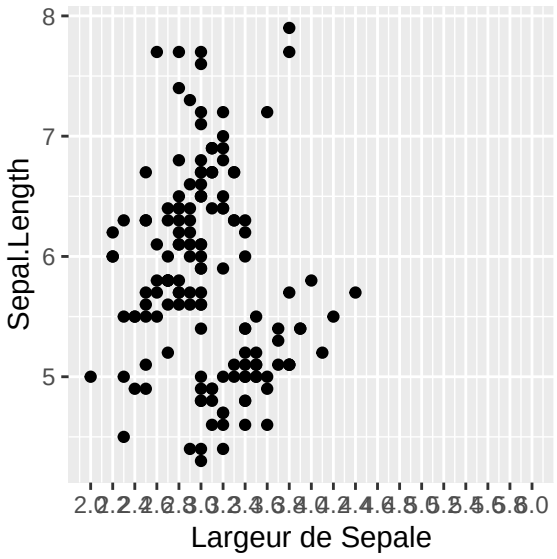
Les échelles linéaires Reprenons le graphique `MonNuage`. Et (re-)constatons qu'il suffit de saisir son nom pour qu'il apparaisse.

```
MonNuage
```



Allons bricoler l'axe des abscisses. En parcourant la liste des fonctions `scale_*()`, on peut deviner (non ?) que la fonction à manipuler est `scale_x_continuous()` et en consultant sa fiche d'aide, on peut aisément voir comment modifier le titre (`name`), le découpage (`breaks`), les limites (`limits`) et d'autres choses...

```
MonNuage + scale_x_continuous(name = "Largeur de Sepale",
                              breaks = seq(2,6,by=0.2),
                              limits = c(2,6))
```

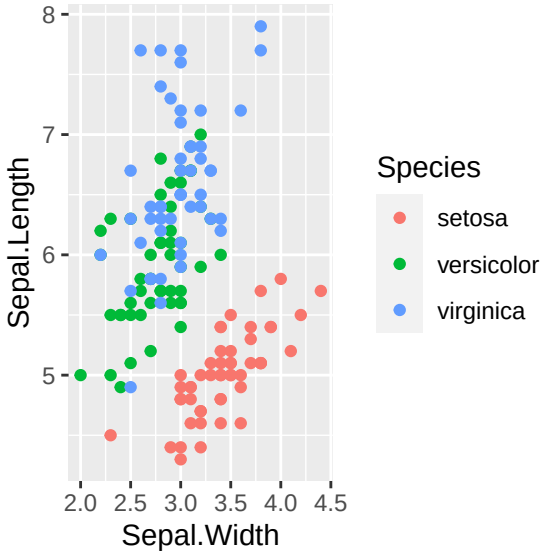


Question : comment modifier l'axe des ordonnées ? Il serait quasiment insultant de fournir ici une réponse à cette question.

Mais il y a aussi les légendes (?!)

Pour des couleurs liées à une variable qualitative On va ajouter un `aesthetics` à notre graphique pour continuer à explorer les échelles.

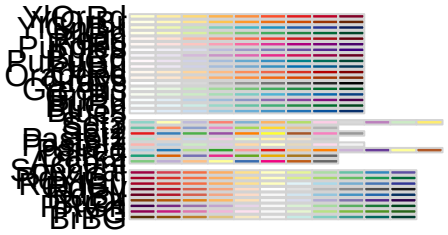
```
MonNuageCouleur <- MonNuage + aes(colour = Species)
MonNuageCouleur
```



Comme dit et re-dit précédemment, à chaque `aesthetics` est associée une échelle. On a ajouté l'aesthetics colour, donc il y a une échelle associée.

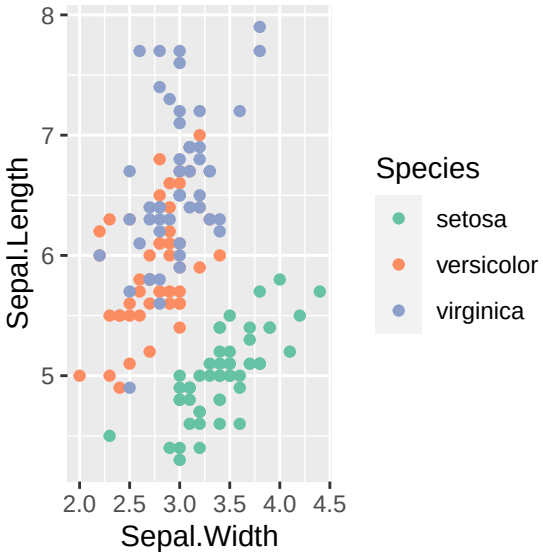
Pour la modifier et utiliser par exemple des palettes (qualitatives dans le cas présent) de couleur du package `RcolorBrewer`

```
library(RColorBrewer)
display.brewer.all()
```

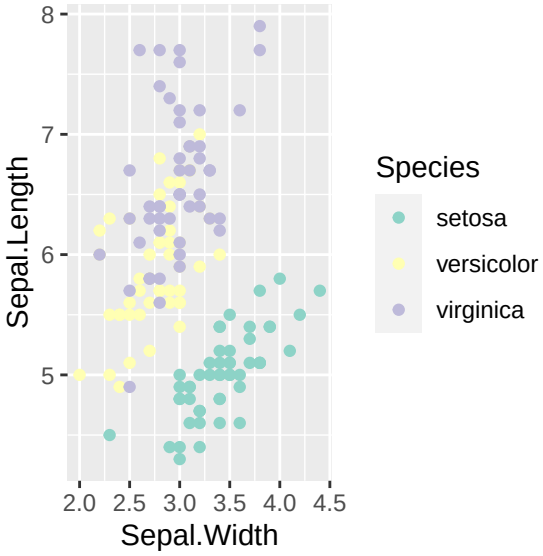


utilisons la fonction `scale_colour_brewer()`

```
MonNuageCouleur + scale_colour_brewer(palette="Set2")
```

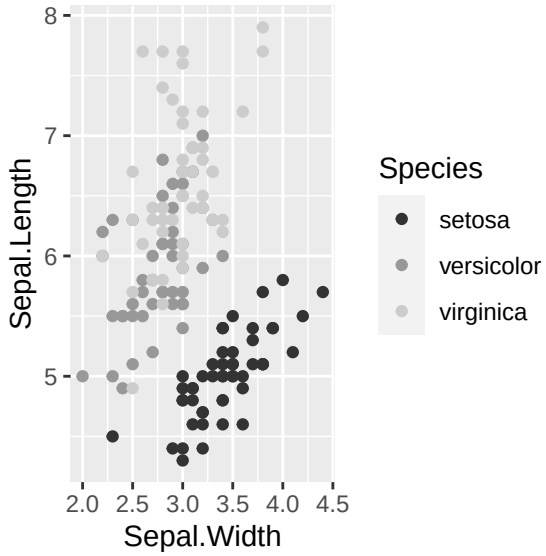



```
MonNuageCouleur + scale_colour_brewer(palette="Set3")
```

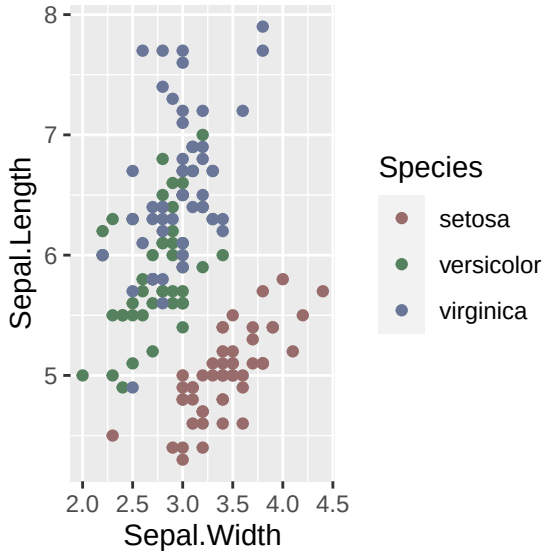


ou d'autres fonctions

```
MonNuageCouleur + scale_colour_grey()
```

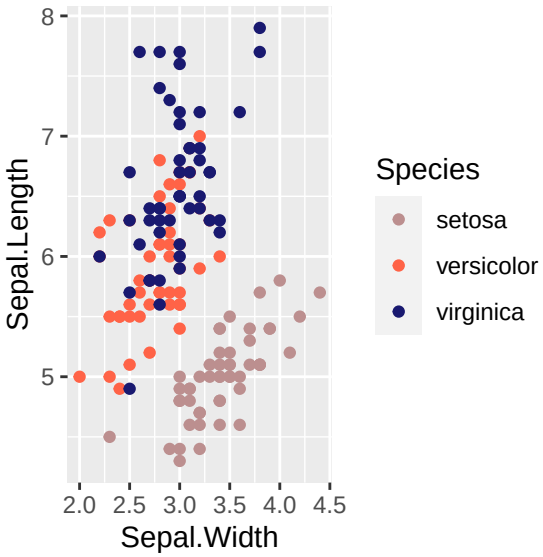



```
MonNuageCouleur + scale_colour_hue(l = 50, c = 30)
```



Et pour mettre ses couleurs favorites, on peut même le faire à la main.

```
MonNuageCouleur +
  scale_colour_manual(values = c("rosybrown","tomato","midnightblue"))
```

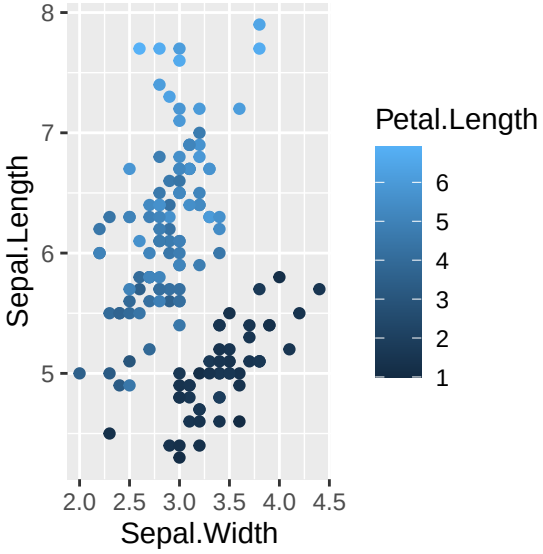


En cas de manque d'inspiration, la fonction `colours()` est là.

Pour des couleurs liées à une variable quantitative Et pour une légende en couleur liée à une variable quantitative, comment ça marche ? Pareil !

Modifions l'aesthetic `colour`

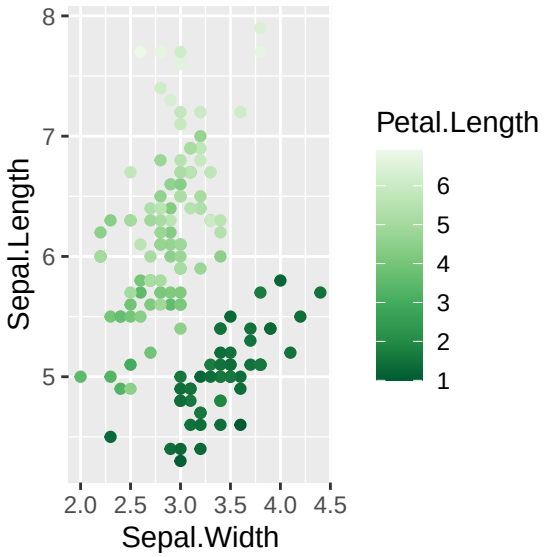
```
MonNuageCouleurQuanti <- MonNuage + aes(colour = Petal.Length)
MonNuageCouleurQuanti
```



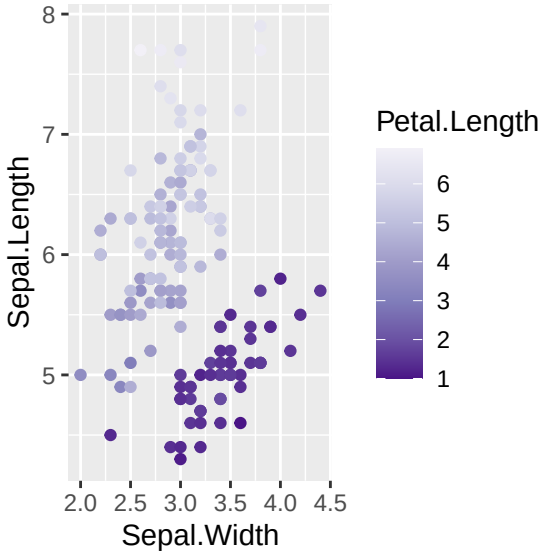
On a, comme on l'a déjà vu, une teinte de bleu qui s'éclaircit quand la valeur de `Petal.Length` augmente. Et si je veux changer du bleu ?

On peut encore aller piocher dans les palettes (séquentielles cette fois-ci) du package `RColorBrewer`, mais, là dans `ggplot2`, on ne brasse plus, on distille.

```
MonNuageCouleurQuanti + scale_colour_distiller(palette = "Greens")
```

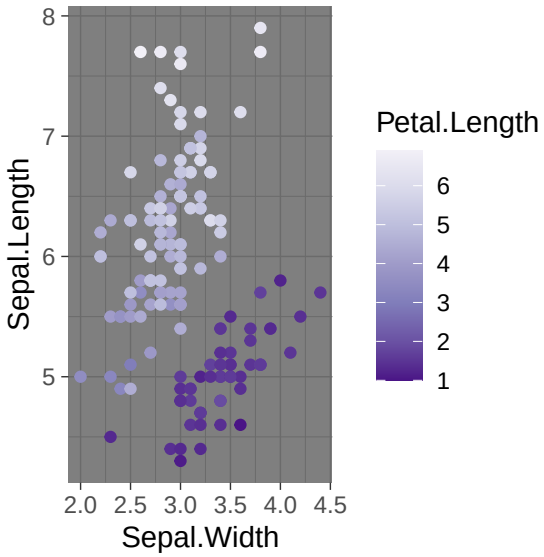



```
MonNuageCouleurQuanti + scale_colour_distiller(palette = "Purples")
```



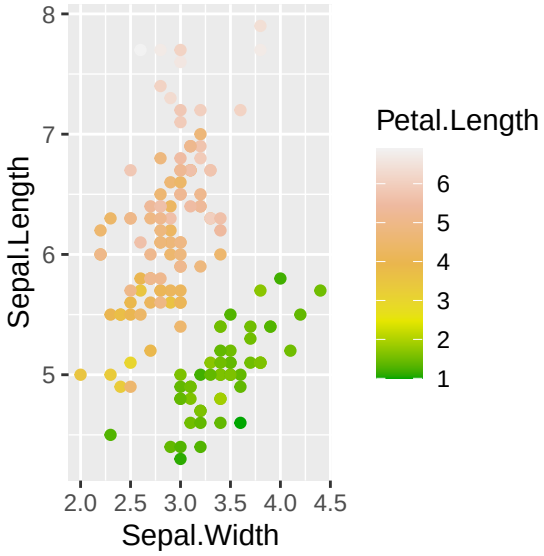
Pas mal, mais je ne vois pas bien les points les plus clairs. Pas de problème, je vais passer en `theme_dark`.

```
MonNuageCouleurQuanti +
  scale_colour_distiller(palette = "Purples") +
  theme_dark()
```



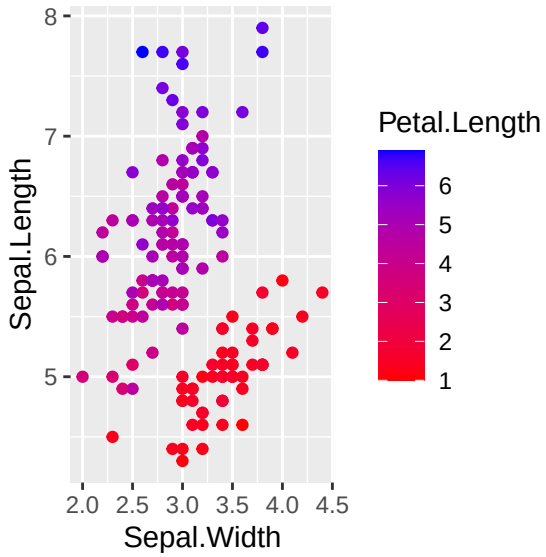
On peut aussi récupérer d'autres palettes de R

```
MonNuageCouleurQuanti + scale_colour_gradientn(colors=terrain.colors(5))
```

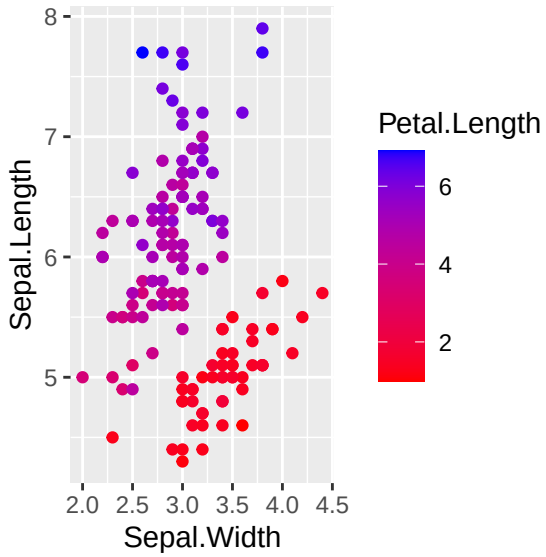


et faire en fait tout ce que l'on veut ! (mais ça, on le savait déjà).

```
MonNuageCouleurQuanti + scale_colour_gradient(low="red", high="blue")
```

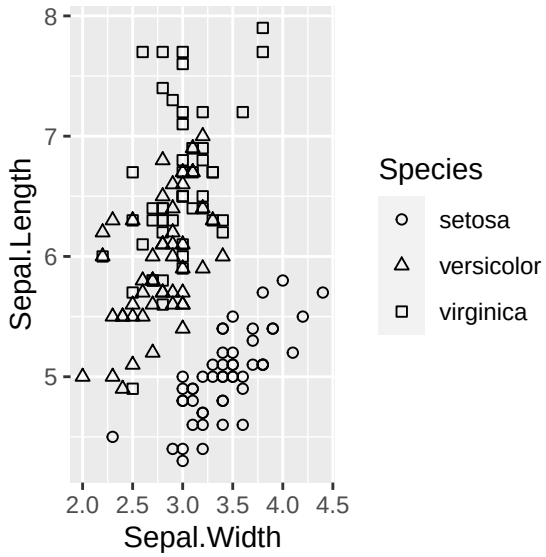



```
MonNuageCouleurQuanti +
  scale_colour_gradient(low="red", high="blue", breaks=seq(0,6,by=2))
```

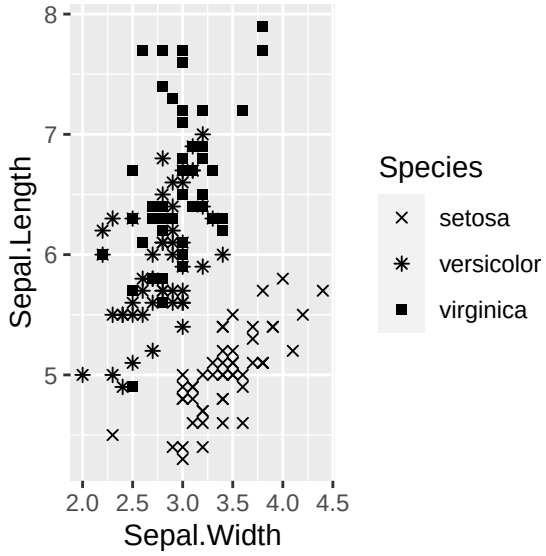


Pour des formes aussi ! A un `aesthetic` correspond une échelle, donc si mon `aesthetic` est une forme, alors une échelle lui est associée (pas de discrimination !) et on peut donc la modifier avec les fonctions `scale_shape_*()`.

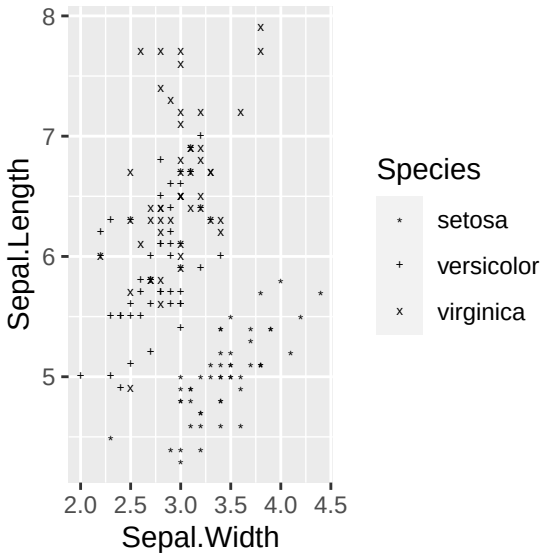
```
MonNuageForme <- MonNuage + aes(shape = Species)
MonNuageForme + scale_shape(solid=FALSE)
```




```
MonNuageForme + scale_shape_manual(values = c(4,8,15))
```

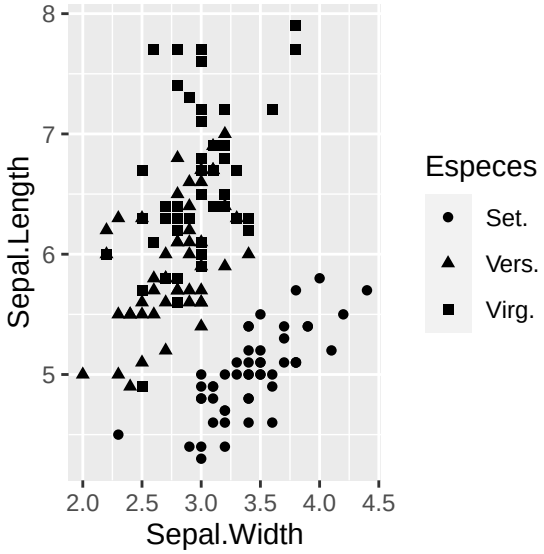



```
MonNuageForme + scale_shape_manual(values = c("*","+","x"))
```



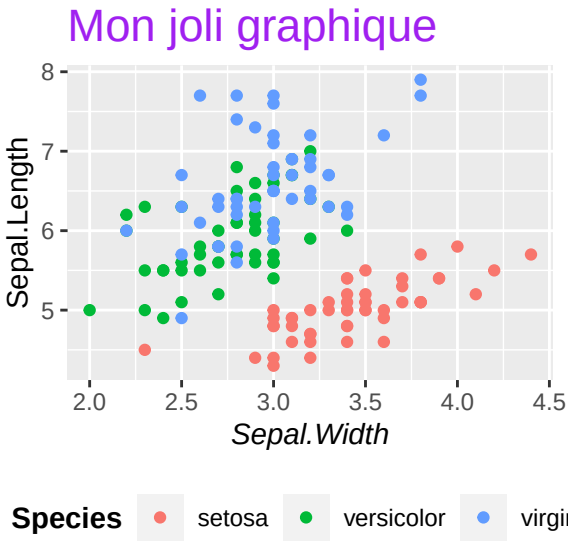
Et si je veux modifier quelques bricoles dans ma légende, aucun problème, je peux aussi faire ce genre de choses :

```
MonNuageForme + scale_shape(name = "Especes",
                            labels = c("Set.","Vers.","Virg."))
```



Pour aller plus loin dans les modifications d'apparence de la légende (et d'autres choses, en fait de tout ce que l'on voit), il faut aller bricoler le thème... Ca devient très touchy et pas forcément d'une grande utilité, mais voyons quand même 2 ou 3 trucs. Un petit coup d'oeil à l'aide de la fonction `theme()` peut suffire à se convaincre de ne pas aller plus loin.

```
MonNuageCouleur + ggtitle("Mon joli graphique") +
  theme(legend.position = "bottom",
        legend.title = element_text(face="bold"),
        axis.title.x = element_text(face="italic"),
        plot.title = element_text(colour = "purple",size = rel(1.5)))
```



Mon graphique est parfait, je veux le sauvegarder !

Deux solutions possibles :

- Je sauve l'objet dans mon environnement de travail : fonction `save()`, en fait comme n'importe quel objet `R`. En fait, ça, on l'a déjà pratiqué.
- Je sauvegarde la figure dans un fichier graphique : fonction `ggsave()`. Consulter l'aide de `ggsave()` pour en savoir plus, mais ne vous attendez pas à de grandes surprises. Seule précaution à signaler : il faut réaliser le graphique, puis le sauvegarder (pas comme avec les fonctions `pdf()`, `jpeg()`, `png()` qu'il faut appeler avant de réaliser le graphique et qu'il faut ensuite fermer avec `dev.off()`).