Manipulation de données avec le langage R
(suite)

G. San Martin

gilles.sanmartin@gmail.com
Centre Wallon de Recherche Agronomique

cra-w

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de dates et caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

R pour faire des Graphiques

Différents systèmes graphiques :

Base
Lattice, ggplot2 (Grid)
RGL : 3D, manipulables

Tout se règle en ligne de commande !
On ne peut pas modifier les éléments déjà tracés.
Mais on peut facilement retracer tous les éléments en modifiant les paramètres

Possibilité d'exporter en vectoriel et d'éditer dans un logiciel adapté (pex Inkscape)

R pour faire des Graphiques

Synthèse des fonctions disponibles et principes de base:
http://cran.r-project.org/doc/contrib/Paradis-rdebuts_fr.pdf

Exploration systématique des paramètres graphiques
(c'est ce qui est le plus long à maîtriser):
http://pbil.univ-lyon1.fr/R/pdf/tdr75.pdf

Couleurs :
http://research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf

Gallerie de graphiques (avec code)
gallery.r-enthusiasts.com

5 types de "fonctions" graphiques

fonctions de haut niveau
fonctions de bas niveau
paramètres graphiques par()
fonctions pour interagir avec le graphique
dispositifs graphiques ("devices")

+

Possibilités de diviser la fenêtre graphique en sous-parties

Fonctions graphiques de haut niveau

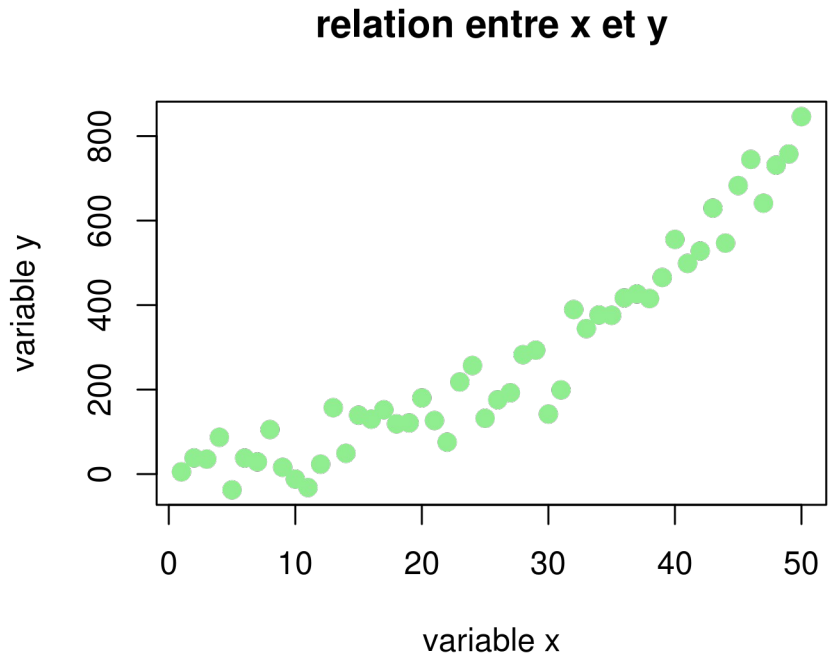
Servent à créer le graphique
Fonctions différentes pour chaque type de graphique (boxplot, barplot,...)
Seul élément indispensable

```
x <- 1:50
mu <- x + 0.3 * x^2
y <- rnorm(50, mu,50)
```

Fction graphique de haut niveau

```
plot(y ~ x,  pch=19, cex = 1.2, col = "lightgreen",
        xlab = "variable x", ylab = "variable y",
        main = "relation entre x et y")
```

arguments
voir ?plot.default



Fonctions graphiques de bas niveau

Ajoutent des éléments à un graphique existant

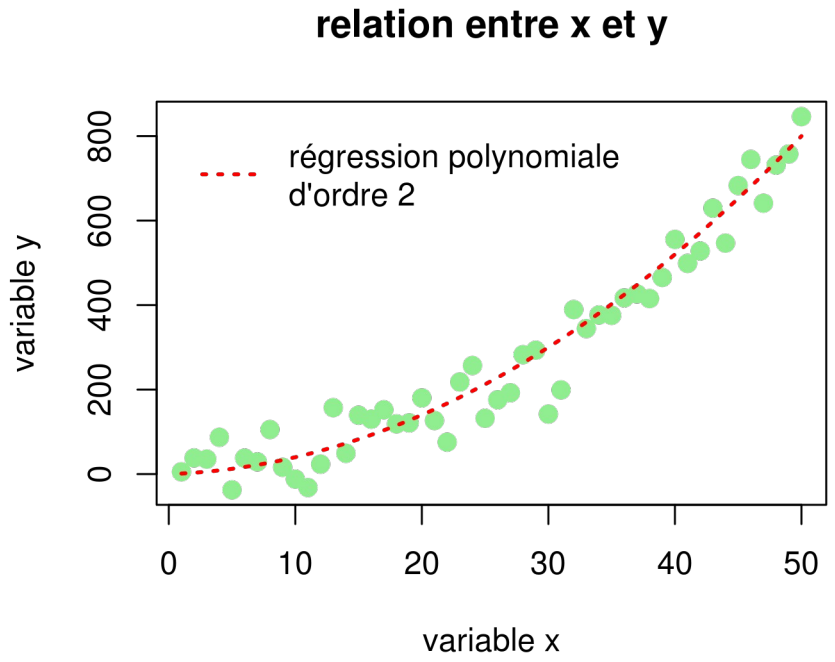
Fction graphique de haut niveau

```
plot(y ~ x,  pch=19, cex = 1.2, col = "lightgreen",
        xlab = "variable x", ylab = "variable y",
        main = "relation entre x et y")
```

arguments

Fctions graphiques de bas niveau

```
lines(mu ~ x, lty = 3, col = "red", lwd = 2)
legend("topleft", lty=3, col = "red", lwd = 2,
        legend="régression polynomiale \nd'ordre 2",
        bty="n", inset = 0.025)
```



Paramètres graphiques par()

Modifient certains aspects des graphiques.

Certains paramètres se modifient uniquement dans par() certains uniquement dans les arguments des fonctions graphiques, beaucoup dans les deux.

paramètres graphiques

```
par(bg = "lightyellow", mar = c(4,4,3,3), las=1, mgp = c(2.5,0.5,0))
```

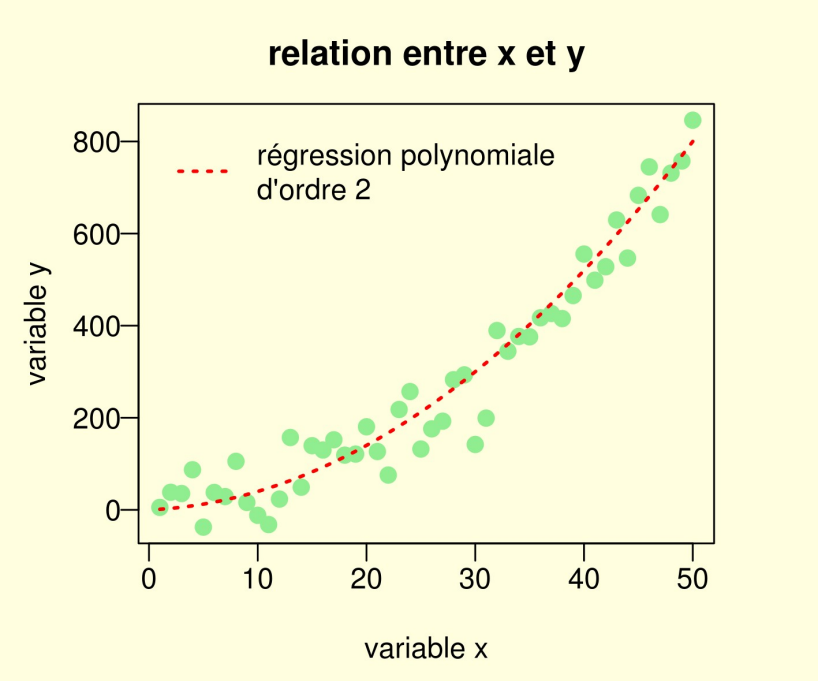
Fction graphique de haut niveau

```
plot(y ~ x,  pch=19, cex = 1.2, col = "lightgreen",
          xlab = "variable x", ylab = "variable y",
          main = "relation entre x et y")
```

Fctions graphiques de bas niveau

```
lines(mu ~ x, lty = 3, col = "red", lwd = 2)
legend("topleft", lty=3, col = "red", lwd = 2,
          legend="régression polynomiale \nd'ordre 2",
          bty="n", inset = 0.025)
```

bg : couleur de fond
mar : taille des marges
las : orientation des étiquettes
mgp : positions des étiquettes, titres d'axes, ...



Fonctions pour Interagir avec le graphique

Peu nombreuses

Interaction via des clics avec la souris sur le graphique

paramètres graphiques

```
par(bg = "lightyellow", mar = c(4,4,3,3), las=1, mgp = c(2.5,0.5,0))
```

Fction graphique de haut niveau

```
plot(y ~ x,  pch=19, cex = 1.2, col = "lightgreen",
        xlab = "variable x", ylab = "variable y",
        main = "relation entre x et y")
```

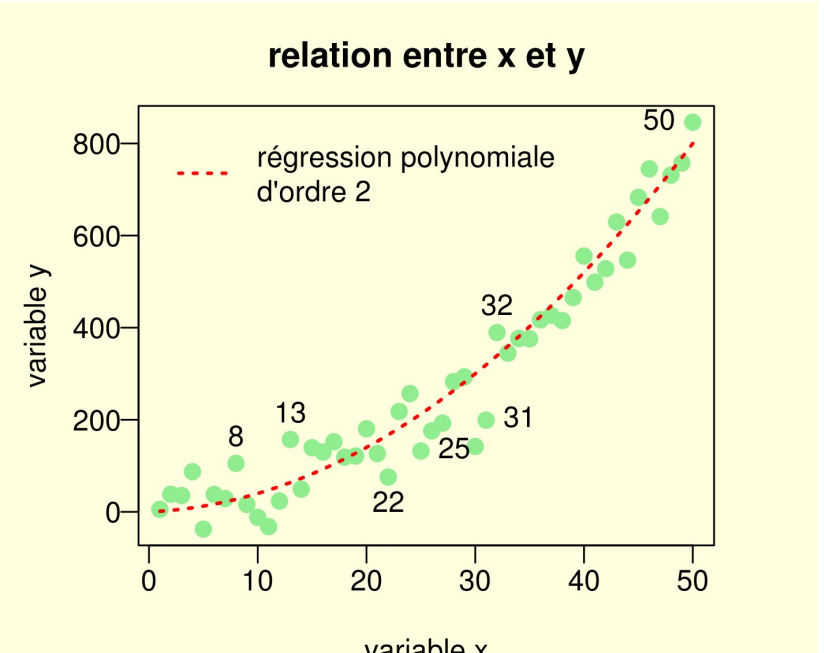
Fctions graphiques de bas niveau

```
lines(mu ~ x, lty = 3, col = "red", lwd = 2)
legend("topleft", lty=3, col = "red", lwd = 2,
        legend="régression polynomiale \nd'ordre 2",
        bty="n", inset = 0.025)
```

Interaction avec le graphique

```
identify(x,y)
```

Cliquer sur un point pour faire apparaître son étiquette
Clic droit pour stopper



Dispositifs graphiques

"Devices" en anglais

Fenêtre ou fichier dans lesquels sont affichés/sauvés les graphiques.

On les utilise quand on veut spécifier les dimensions du graphique (ea à l'écran) et surtout pour l'exportation dans un fichier

Ouvre une nouvelle fenêtre graphique de taille définie à l'écran

```
dev.new(width = 12/2.54, height = 10/2.54)

par(bg = "lightyellow", mar = c(4,4,3,3), las=1, mgp = c(2.5,0.5,0))

plot(y ~ x,  pch=19, cex = 1.2, col = "lightgreen",
        xlab = "variable x", ylab = "variable y",
        main = "relation entre x et y")

lines(mu ~ x, lty = 3, col = "red", lwd = 2)
legend("topleft", lty=3, col = "red", lwd = 2,
        legend="régression polynomiale \nd'ordre 2",
        bty="n", inset = 0.025)

dev.print(png, "myplot.png", width = 12/2.54,
            height = 10/2.54, units = "in", res=300)
dev.off()
```

Ferme la fenêtre graphique

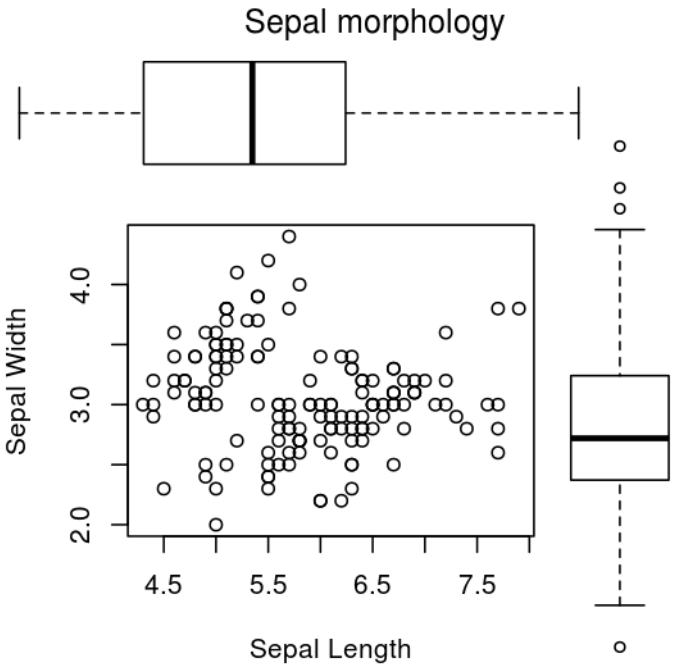
Exporte le graphique courant dans un fichier .png

Diviser le device graphique

Permet d'afficher plusieurs graphiques sur la même figure

Division en 3 zones de tailles différentes

```
data(iris)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE),
       widths=c(3,1), heights=c(1,2))

par(fig=c(0,0.8,0,0.8), bg="white")
plot(iris$Sepal.Length, iris$Sepal.Width, xlab="Sepal Length", ylab="Sepal
Width")
par(mar = c(1,0,1,0), fig=c(0,0.8,0.55,1), new=TRUE)
boxplot(iris$Sepal.Length, horizontal=TRUE, axes=FALSE)
par(fig=c(0.65,1,0,0.8),new=TRUE)
boxplot(iris$Sepal.Width, axes=FALSE)
mtext("Sepal morphology", side=3, outer=TRUE, line=-3)
```



Fonctions graphiques de haut niveau

`plot(numeric,numeric)`	Nuage de points
`boxplot(factor, numeric)`	Boîte à moustaches
`barplot(matrix, beside=FALSE)`	Bâtons
`pie(numeric)`	Tarte/camembert
`dotchart(matrix)`	Cleveland dot plot
`hist(numeric, breaks = "sturges")`	Histogramme
`plot(density(numeric))`	Densité
`mosaicplot(table)`	Mosaic plot
`mathplot()`	Séries de données
`coplot(y ~ x | a * b)`	Graphique conditionnel
`pairs(data.frame)`	Matrice de nuage de points

Fonctions graphiques de haut niveau

Pseudo 3D et 3D

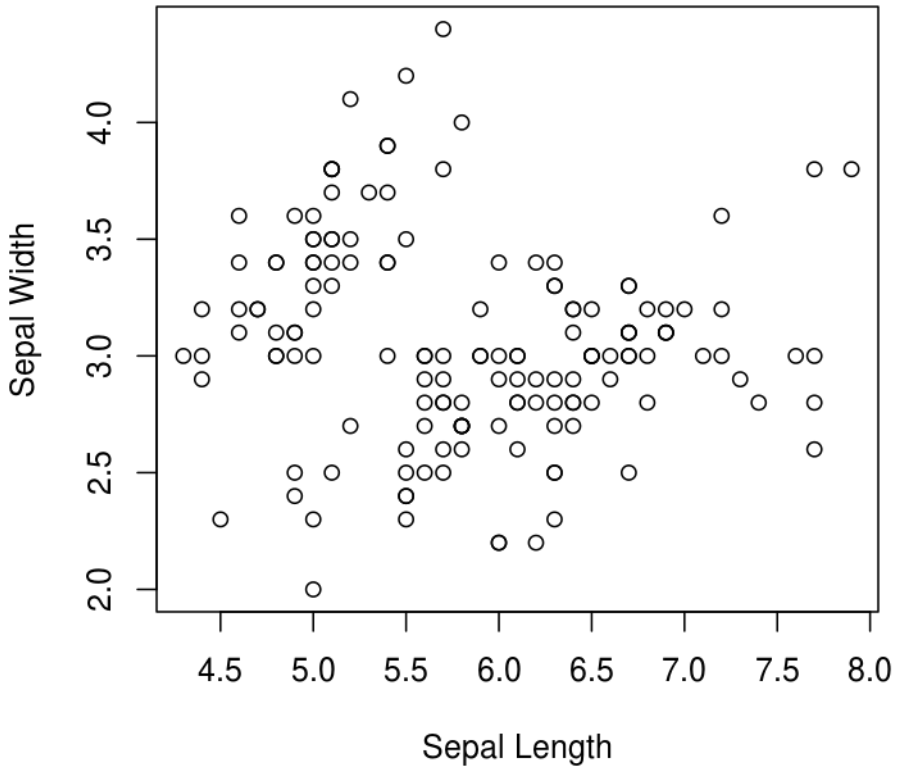
`image(numeric,numeric, numeric)`	pixels colorés
`contour(numeric,numeric, numeric)`	contours de valeurs identiques
`filled.contour(numeric,numeric, numeric)`	contours avec pixels de couleur
`persp(numeric,numeric, numeric)`	surface en 3D
`scatterplot3D (numeric,numeric, numeric)`	nuage de points en 3D

+ Package rgl pour vraie 3D interactive

Fonctions graphiques de haut niveau

Scatter plot (nuage de points) :
`plot(numeric, numeric)`

```
data(iris)
plot(iris$Sepal.Length, iris$Sepal.Width, xlab="Sepal Length", ylab="Sepal
Width")
```



Problème fréquent :
superposition de points
("over ploting")

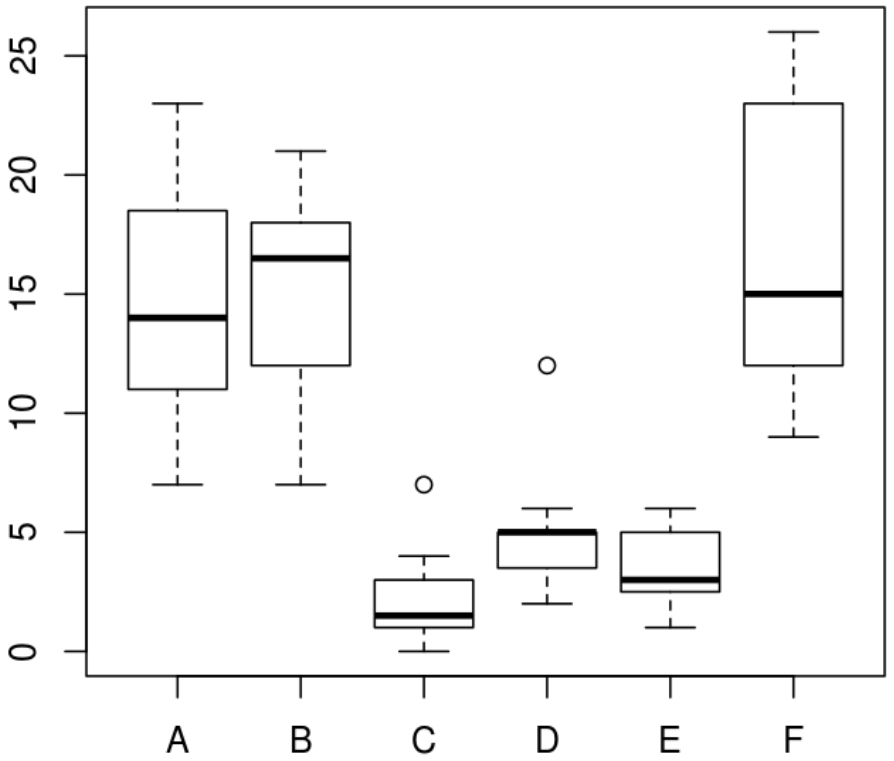
Fonctions graphiques de haut niveau

Boxplot (boîte à moustaches)

`plot(factor, numeric)`

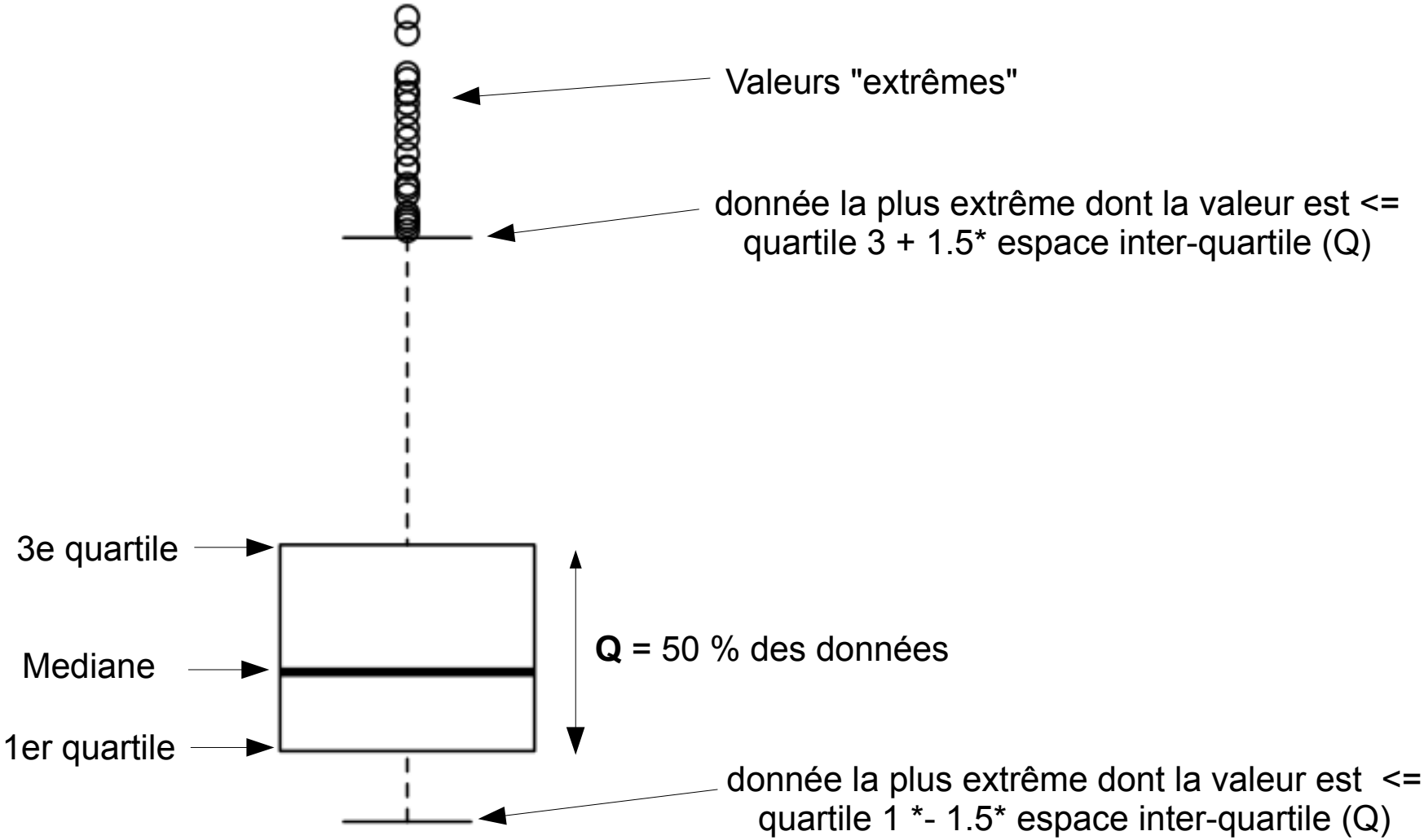
`boxplot(factor, numeric)`

```
plot(count ~ spray, data = InsectSprays)
boxplot(count ~ spray, data = InsectSprays)
```



Fonctions graphiques de haut niveau

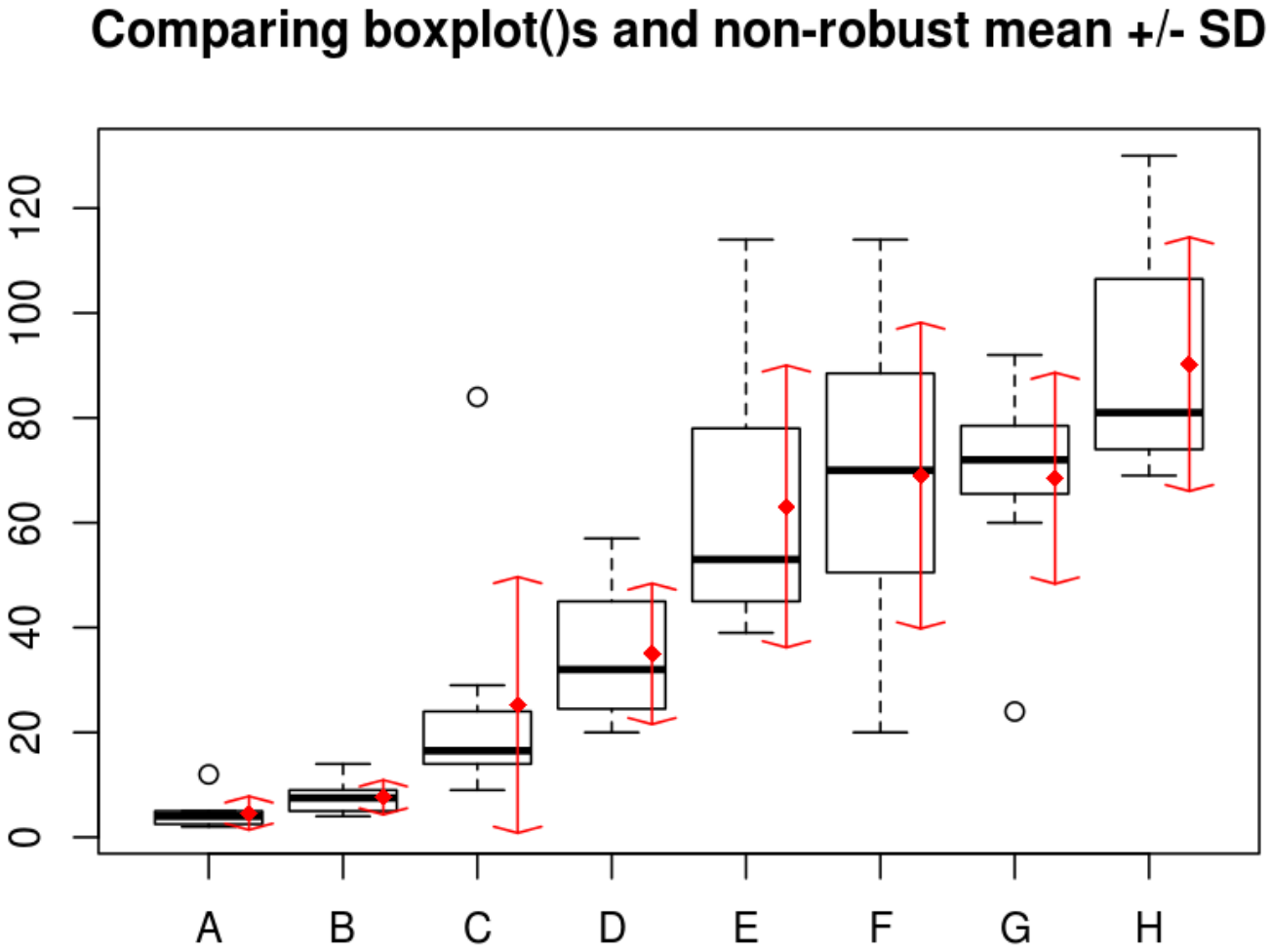
Boxplot : interprétation



Dans ce cas précis il s'agit
donc du minimum

Fonctions graphiques de haut niveau

Boxplot : intérêt



Fonctions graphiques de haut niveau

Barplot (à ne pas confondre avec l'histogramme !!)

`barplot(matrix)`

```
> VADeaths
      Rural Male Rural Female Urban Male Urban Female
50-54       11.7          8.7       15.4          8.4
55-59       18.1         11.7       24.3         13.6
60-64       26.9         20.3       37.0         19.3
65-69       41.0         30.9       54.6         35.1
70-74       66.0         54.3       71.1         50.0
```

Mortalité pour 1000 individus

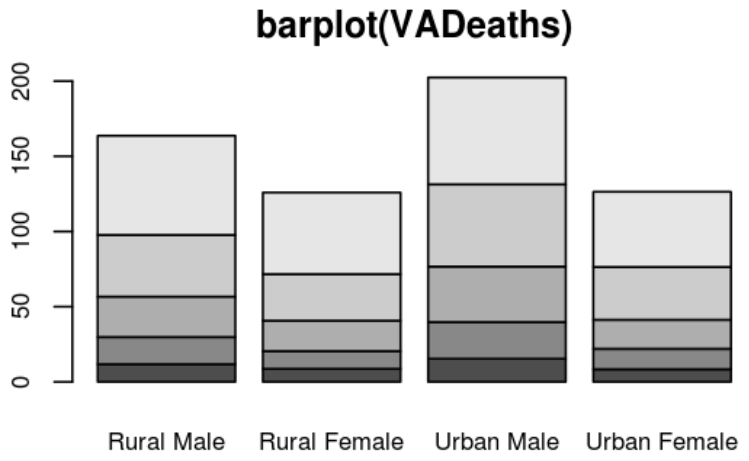
```
> par(mfrow=c(2,2), mar=c(3,2,3,1), cex.axis = 0.8)

> barplot(VADeaths, main = 'barplot(VADeaths)')
> barplot(t(VADeaths), main = 'barplot(t(VADeaths))')
> barplot(VADeaths, beside = TRUE, main = 'barplot(VADeaths, beside = TRUE)')
> barplot(t(VADeaths), beside = TRUE, main = 'barplot(t(VADeaths), beside =
TRUE)')
```

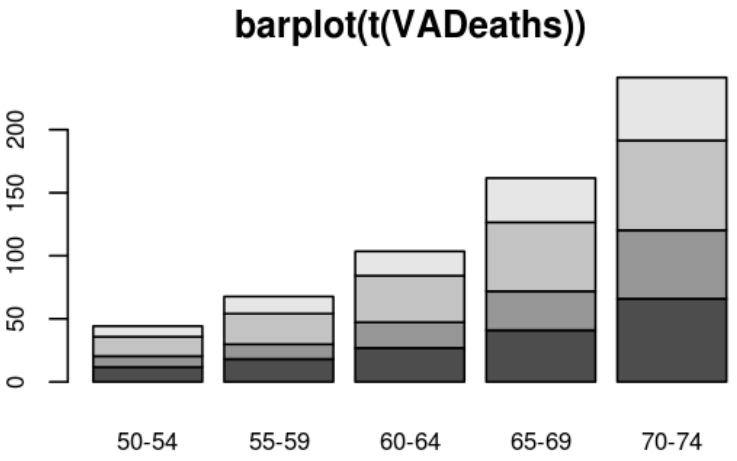
Fonctions graphiques de haut niveau

Barplot (à ne pas confondre avec l'histogramme !!)

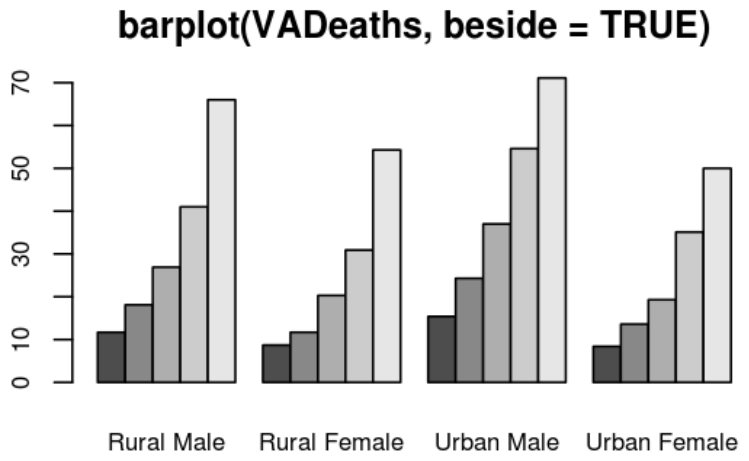
`barplot(matrix)`

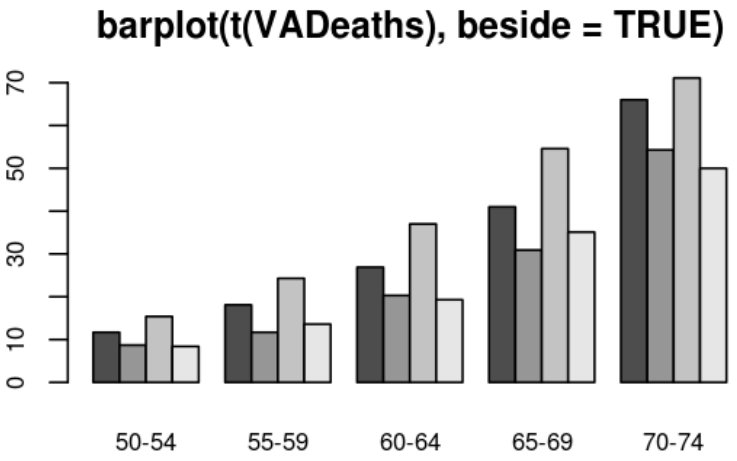
A éviter en général





Souvent mieux





NB : il faudrait une légende et d'autres couleurs

Fonctions graphiques de haut niveau

Pie chart (camenbert / tarte)
`pie(vector)`

Extrait de l'aide de `pie()` :
"Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data."

```
par(mfrow=c(2,2), mar=c(0.5, 0.5, 3, 0.5), cex = 0.8, bg="white")
pie(VADeaths[,1], main = colnames(VADeaths)[1])
pie(VADeaths[,2], main = colnames(VADeaths)[2])
pie(VADeaths[,3], main = colnames(VADeaths)[3])
pie(VADeaths[,4], main = colnames(VADeaths)[4])

VADeaths <- t(VADeaths)
pie(VADeaths[,1], main = colnames(VADeaths)[1])
pie(VADeaths[,2], main = colnames(VADeaths)[2])
pie(VADeaths[,3], main = colnames(VADeaths)[3])
pie(VADeaths[,4], main = colnames(VADeaths)[4])
```

Fonctions graphiques de haut niveau

Pie chart (camembert / tarte)

`pie(numeric)`

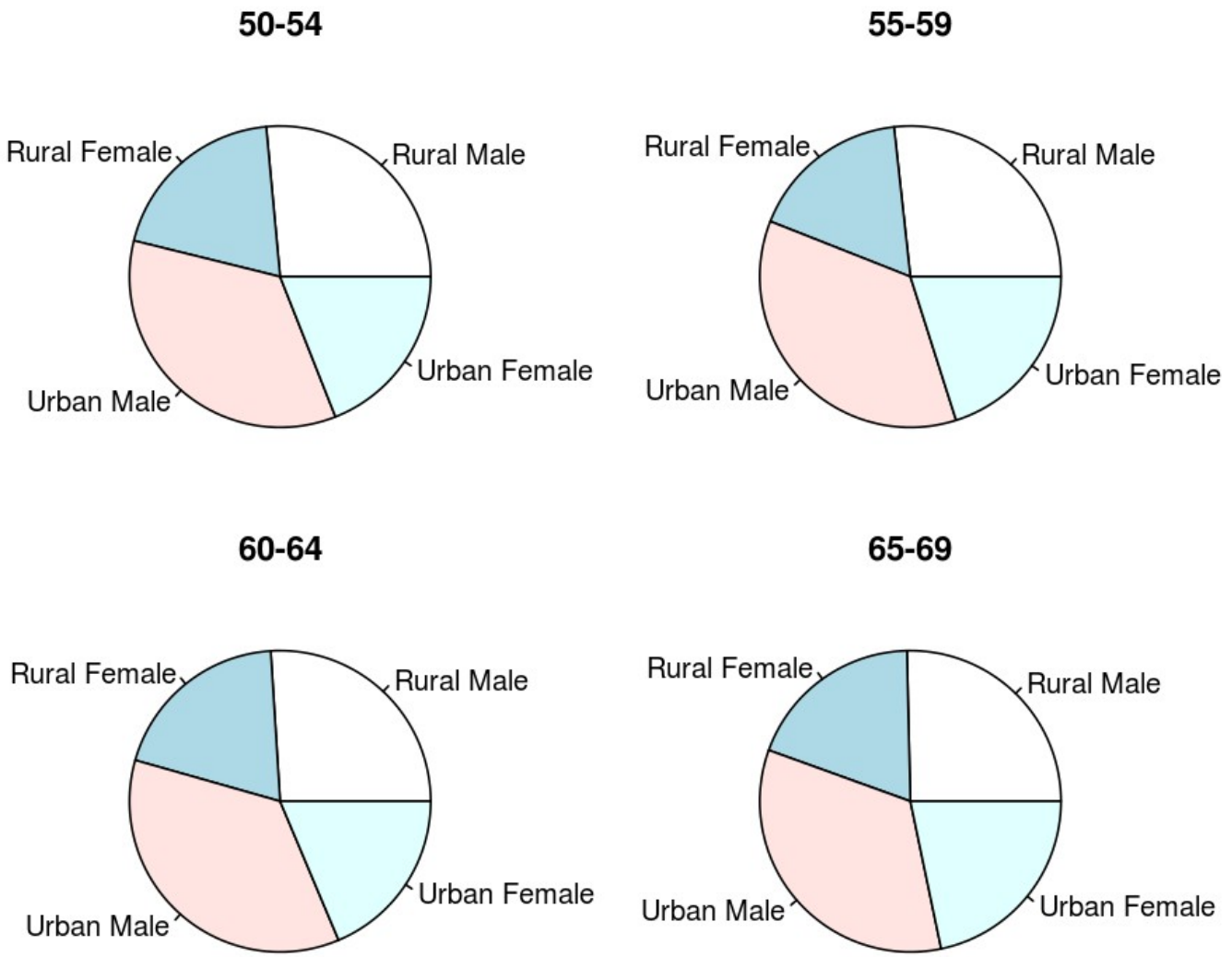
A éviter vraiment ...

Fonctions graphiques de haut niveau

Pie chart (camembert / tartes)

`pie(numeric)`

A éviter vraiment, vraiment !

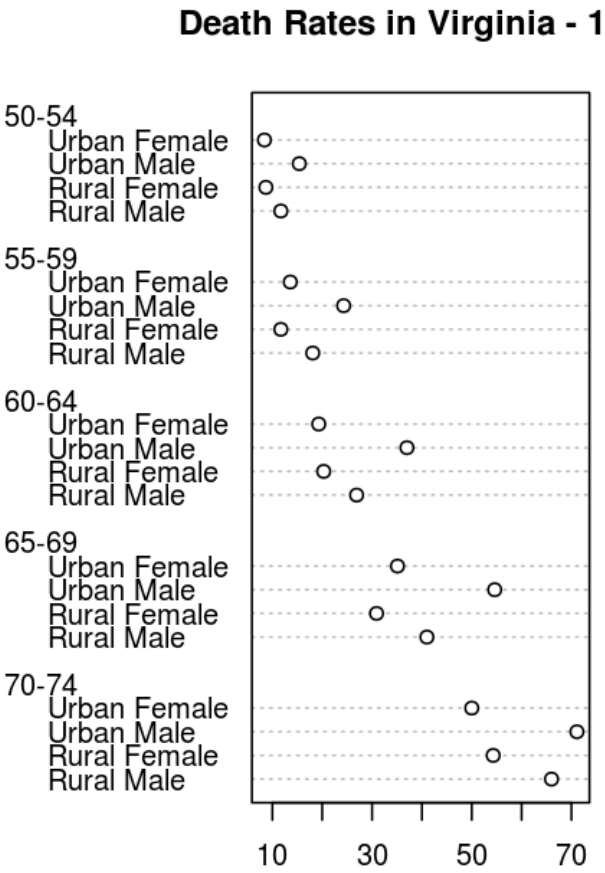


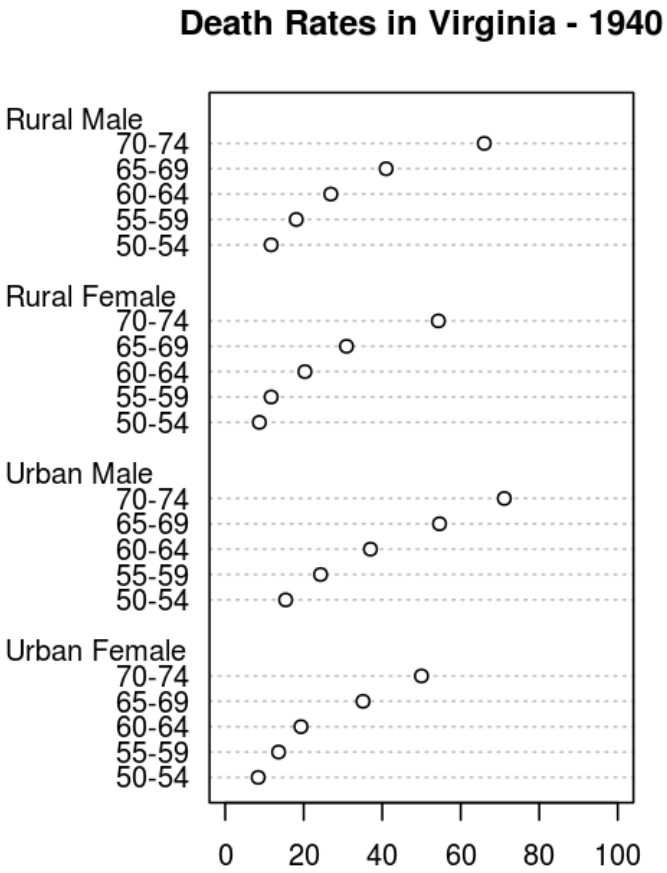
Fonctions graphiques de haut niveau

Cleveland dot plot
`dotchart(matrix)`

En général : très bonne alternative aux graphiques précédents

```
par(mfrow=c(1,2), cex = 0.8, bg="white")
dotchart(VADeaths, main = "Death Rates in Virginia - 1940")
dotchart(t(VADeaths), xlim = c(0,100),  main = "Death Rates in Virginia -
1940")
```





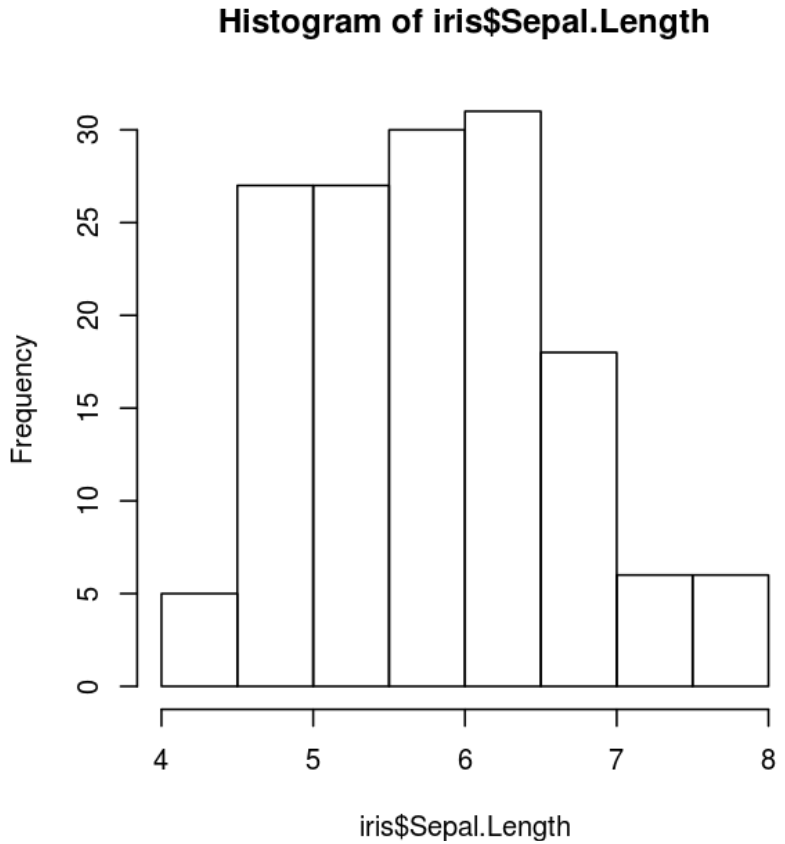
Fonctions graphiques de haut niveau

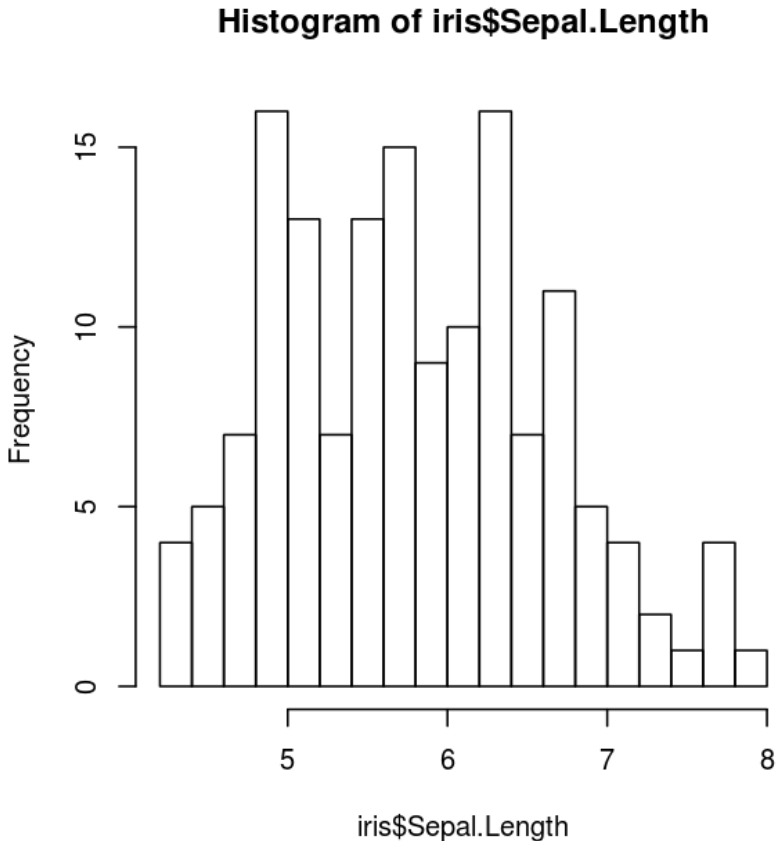
Histogramme :
Distribution de fréquence d'une variable

`hist(vector)`

L'allure du graphique dépend du niveau de lissage (nombre de catégories) et de où on coupe

```
hist(iris$Sepal.Length)
hist(iris$Sepal.Length, breaks = 25)
```





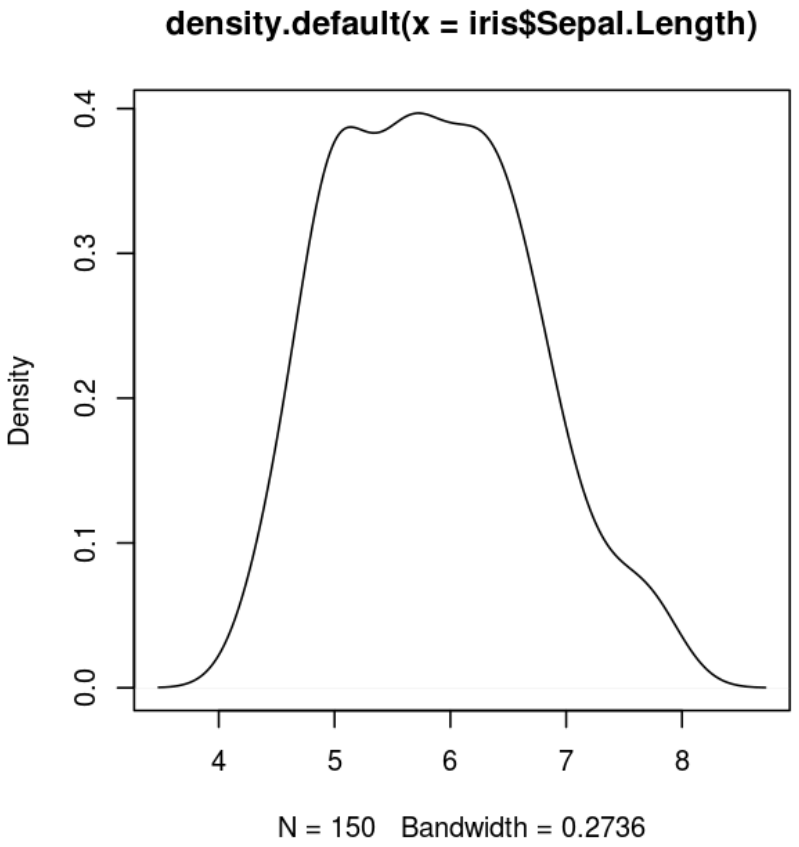
Fonctions graphiques de haut niveau

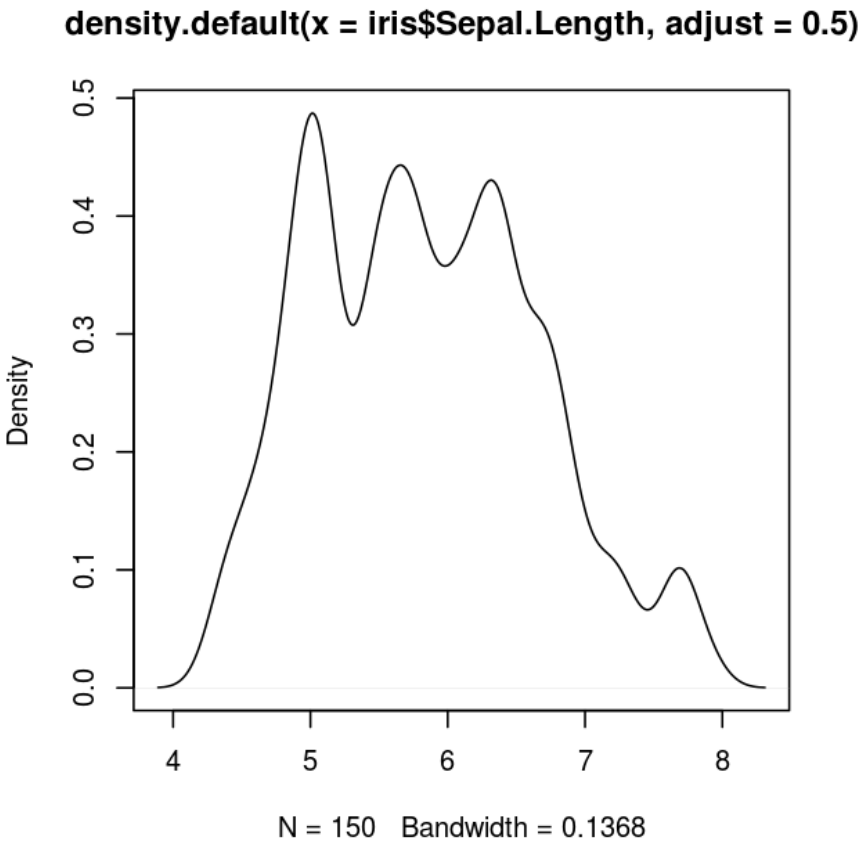
Graphique de densité :
Alternative avantageuse aux histogrammes

`plot(density(vector))`

Dépend uniquement du niveau de lissage,
permet de mettre plusieurs distributions sur le même graphique

```
plot(density(iris$Sepal.Length))
plot(density(iris$Sepal.Length, adjust = 0.5))
```

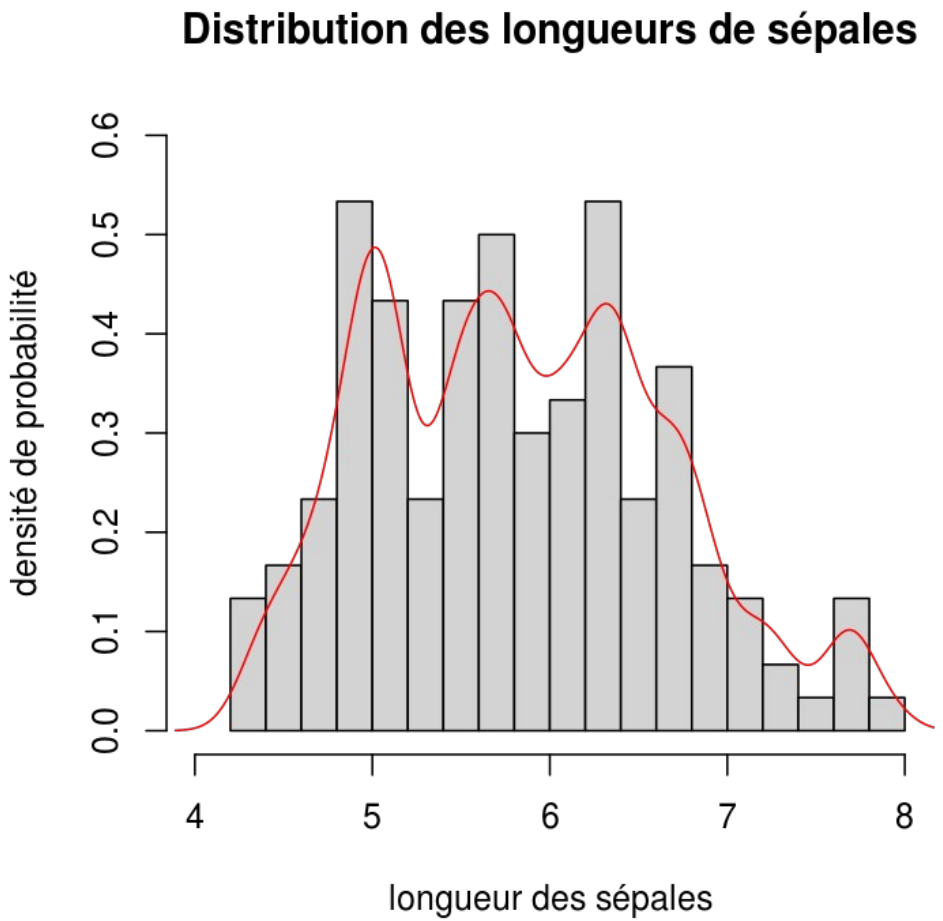




Fonctions graphiques de haut niveau

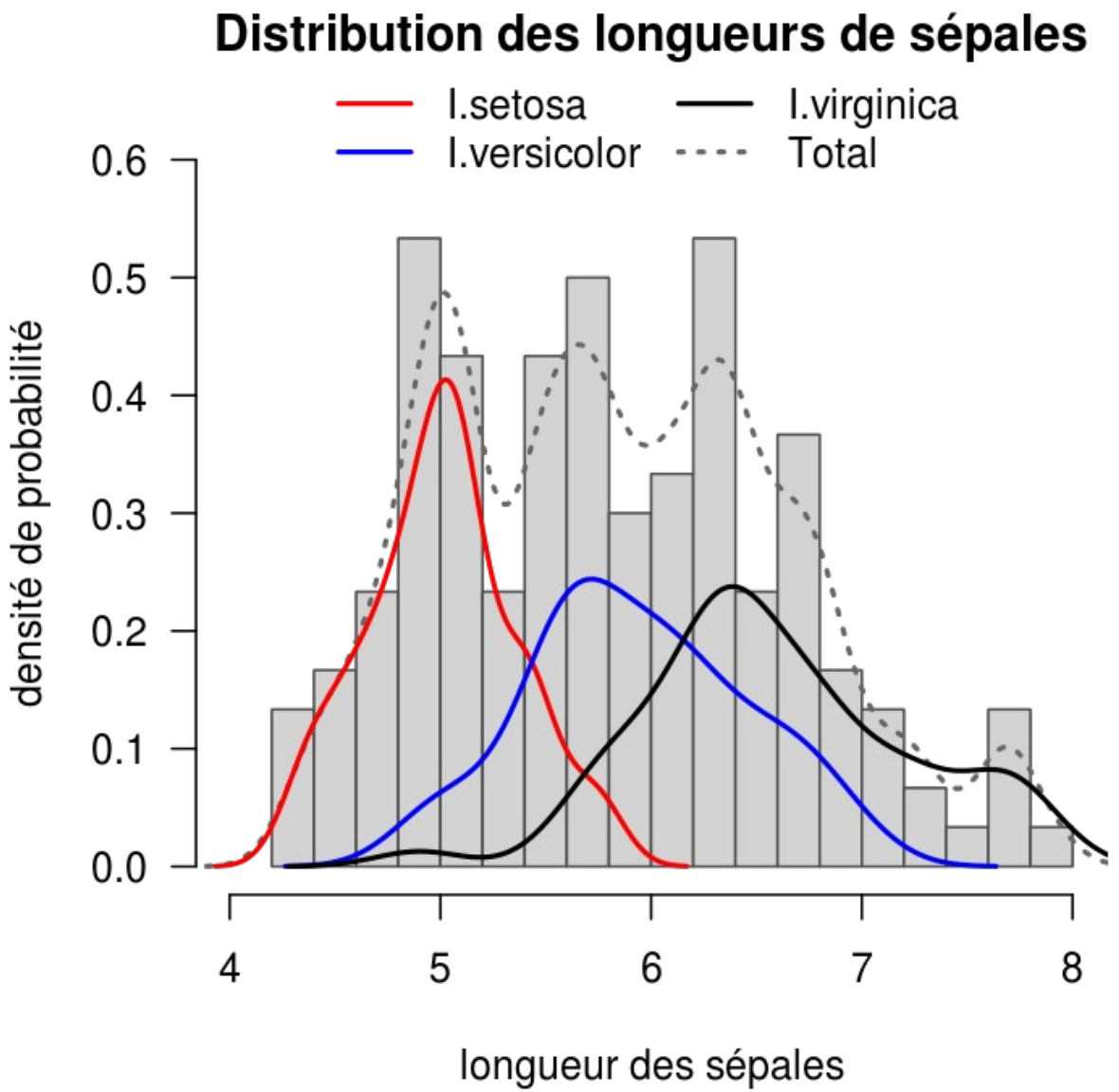
Graphique de densité :

```
hist(iris$Sepal.Length, breaks = 25, col = "lightgray", freq = FALSE,
     xlim = c(4,8), ylim = c(0,0.6), main = "Distribution des longueurs de
     sépales", xlab = "longueur des sépales", ylab = "densité de probabilité")

lines(density(iris$Sepal.Length, adjust = 0.5), col = "red",xlim = c(4,8),
      ylim = c(0,0.6),  main = "", xlab = "", ylab = "")
```



Fonctions graphiques de haut niveau

Graphique de densité :



Fonctions graphiques de haut niveau

Graphique de densité :

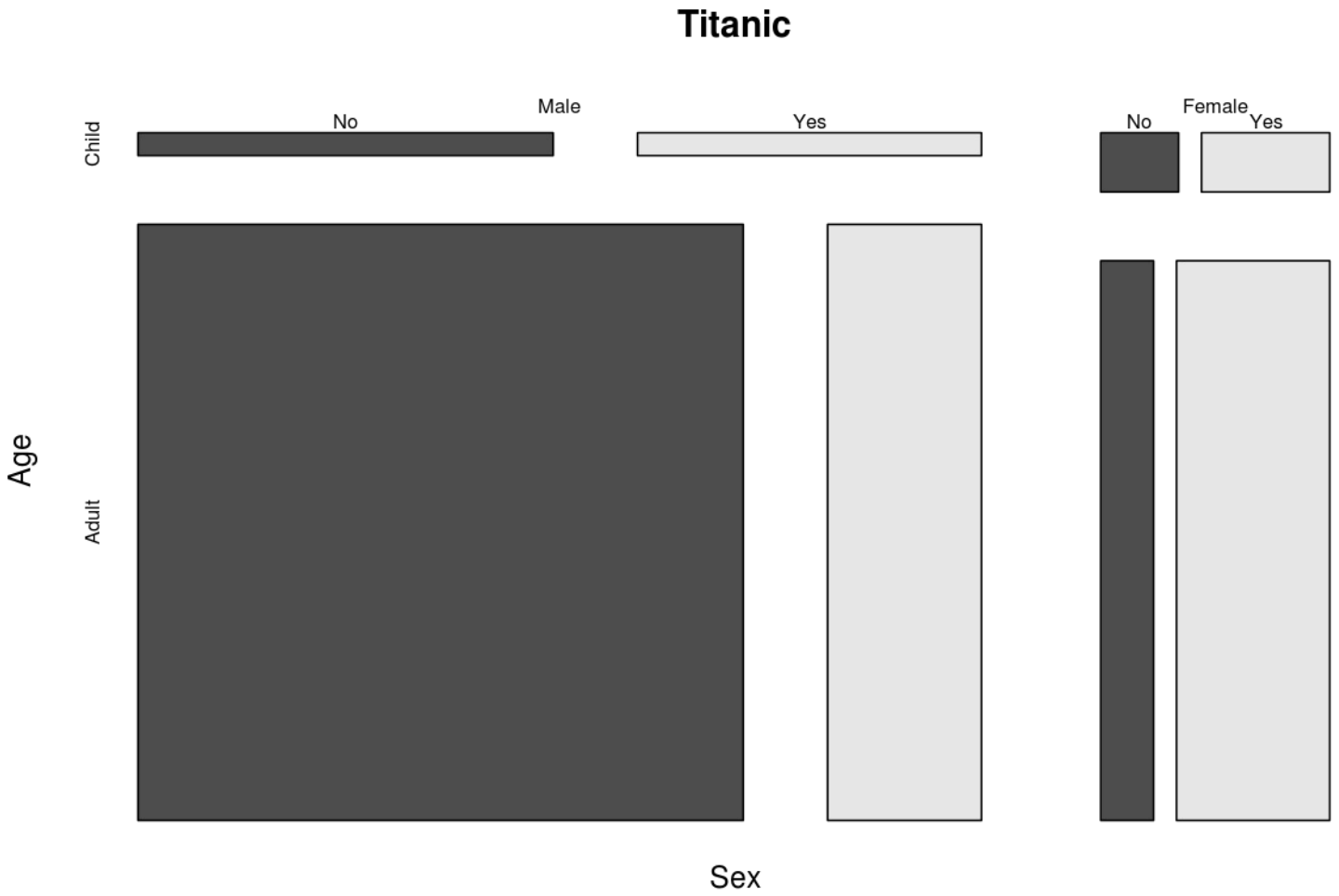
```
# superposition de plusieurs densités + histogramme
hist(iris$Sepal.Length, breaks = 25, col = "lightgray", border = 'grey30',
     freq = FALSE, las=1, xlim = c(4,8), ylim = c(0,0.6),
     main = "Distribution des longueurs de sépales",
    xlab = "longueur des sépales", ylab = "densité de probabilité")

lines(density(iris$Sepal.Length, adjust = 0.5), col = "grey40",
    lwd = 2, lty = 3)

dens1 <- density(iris[iris$Species == "setosa", "Sepal.Length"],
    adjust = 1)
dens2 <- density(iris[iris$Species == "versicolor", "Sepal.Length"],
    adjust = 1)
dens3 <- density(iris[iris$Species == "virginica", "Sepal.Length"],
    adjust = 1)

lines(dens1$x, dens1$y/3, col = "red", lwd = 2)
lines(dens2$x, dens2$y/3, col = "blue", lwd = 2)
lines(dens3$x, dens3$y/3, col = "black", lwd = 2)

legend("top", inset = -0.1, xpd = TRUE, ncol = 2, bty = "n",
       lwd = 2, lty= c(1,1,1,3), col = c("red","blue", "black", "grey40"),
       legend = c("I.setosa", "I.versicolor", "I.virginica", "Total") )
```

Fonctions graphiques de haut niveau

Mosaic plot :
Pour représenter des tables de contingence

`plot(table)`
`mosaicplot(table)`

```
mosaicplot(~ Sex + Age + Survived, data = Titanic, color = TRUE)
```

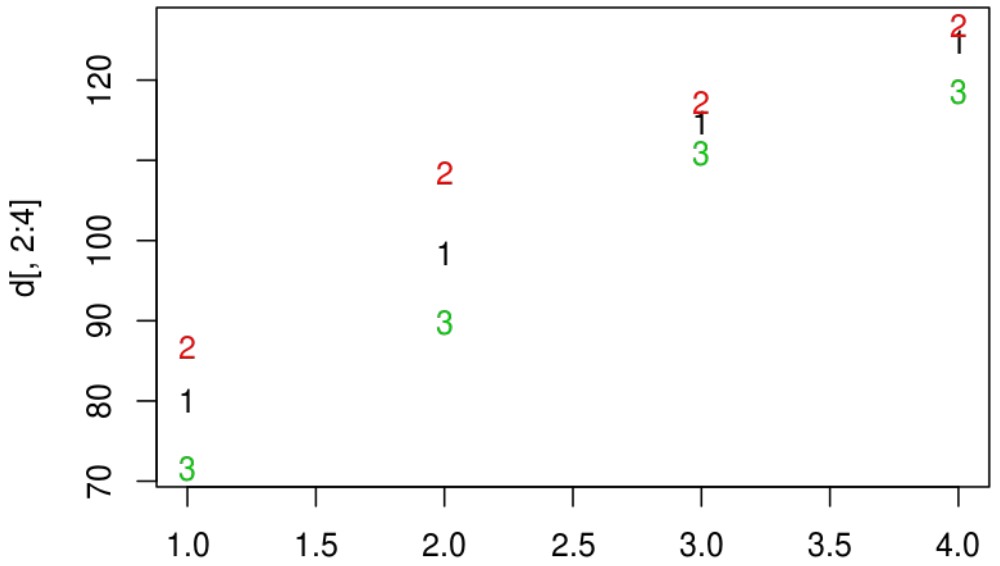


Fonctions graphiques de haut niveau

Séries de données avec `matplot()`

NB : souvent on trace plutôt les séries à la main avec des fonctions graphique de bas niveau
--> matplot pas très utilisé

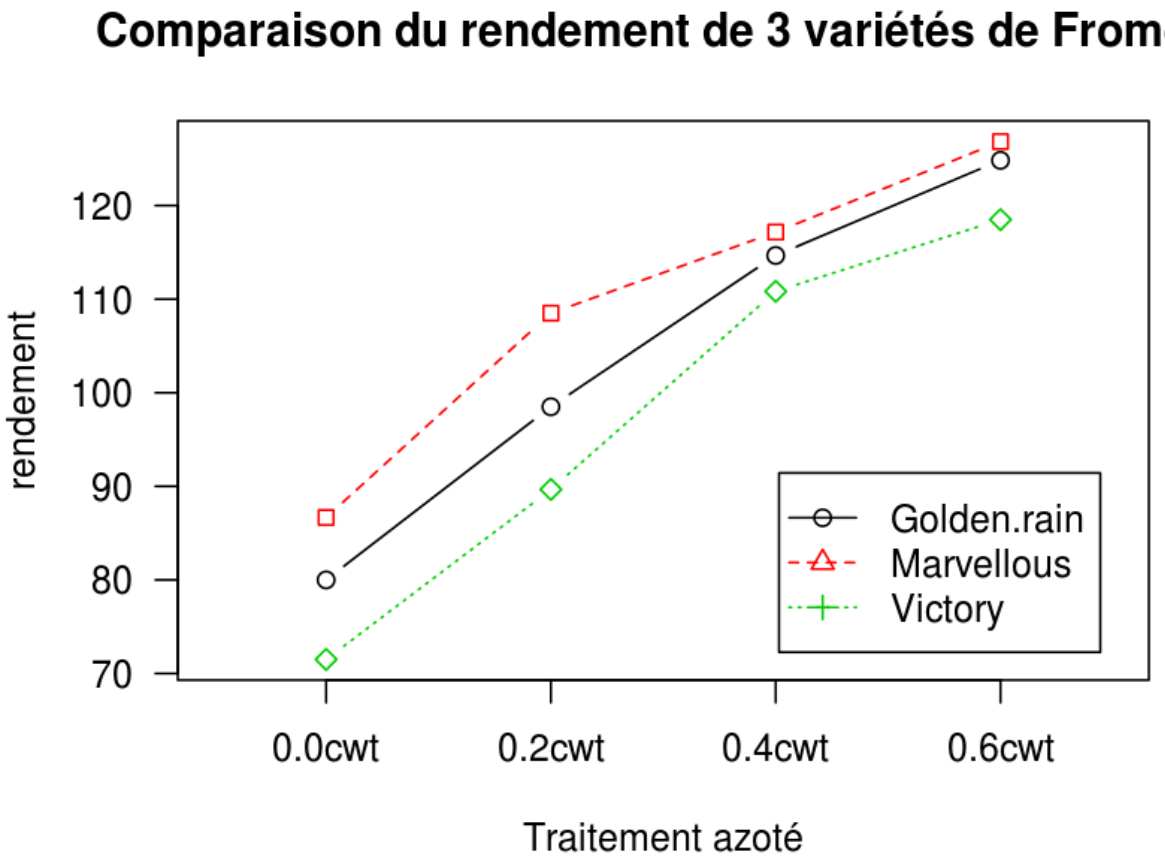
```
> d
       N Golden.rain Marvellous   Victory
1 0.0cwt     80.0000   86.66667  71.50000
2 0.2cwt     98.5000  108.50000  89.66667
3 0.4cwt    114.6667  117.16667 110.83333
4 0.6cwt    124.8333  126.83333 118.50000

> matplot(d[,2:4])
```



Fonctions graphiques de haut niveau

Séries de données avec `matplot()` :

```
matplot(d[,2:4], type = "b", pch = 21:23, xaxt = "n", , las = 2,
     xlim = c(0.5, 4.5),  xlab = "Traitement azoté",  ylab = "rendement",
     main = "Comparaison du rendement de 3 variétés de Froment")
axis(1, at = 1:4, labels = d$N)
legend("bottomright", inset = 0.05, lty = 1:3, col = 1:3, pch = 1:3,
     legend = colnames(d)[-1])
```



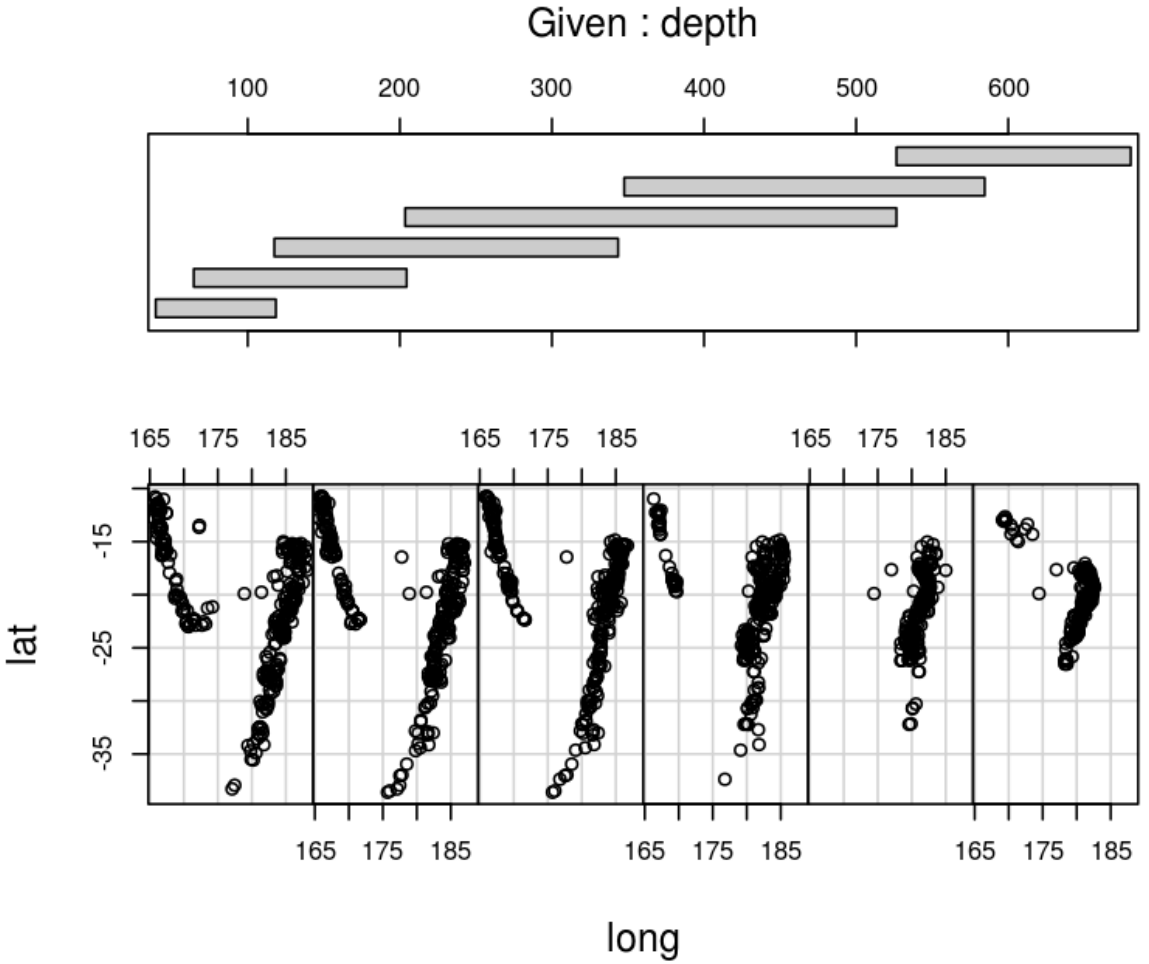
Fonctions graphiques de haut niveau

Plot de y~x conditionnellement à a

`coplot(y ~x | a) :`

Principe poussé beaucoup plus loin dans ggplot2 et lattice

```
coplot(lat ~ long | depth, data = quakes,  rows = 1)
```

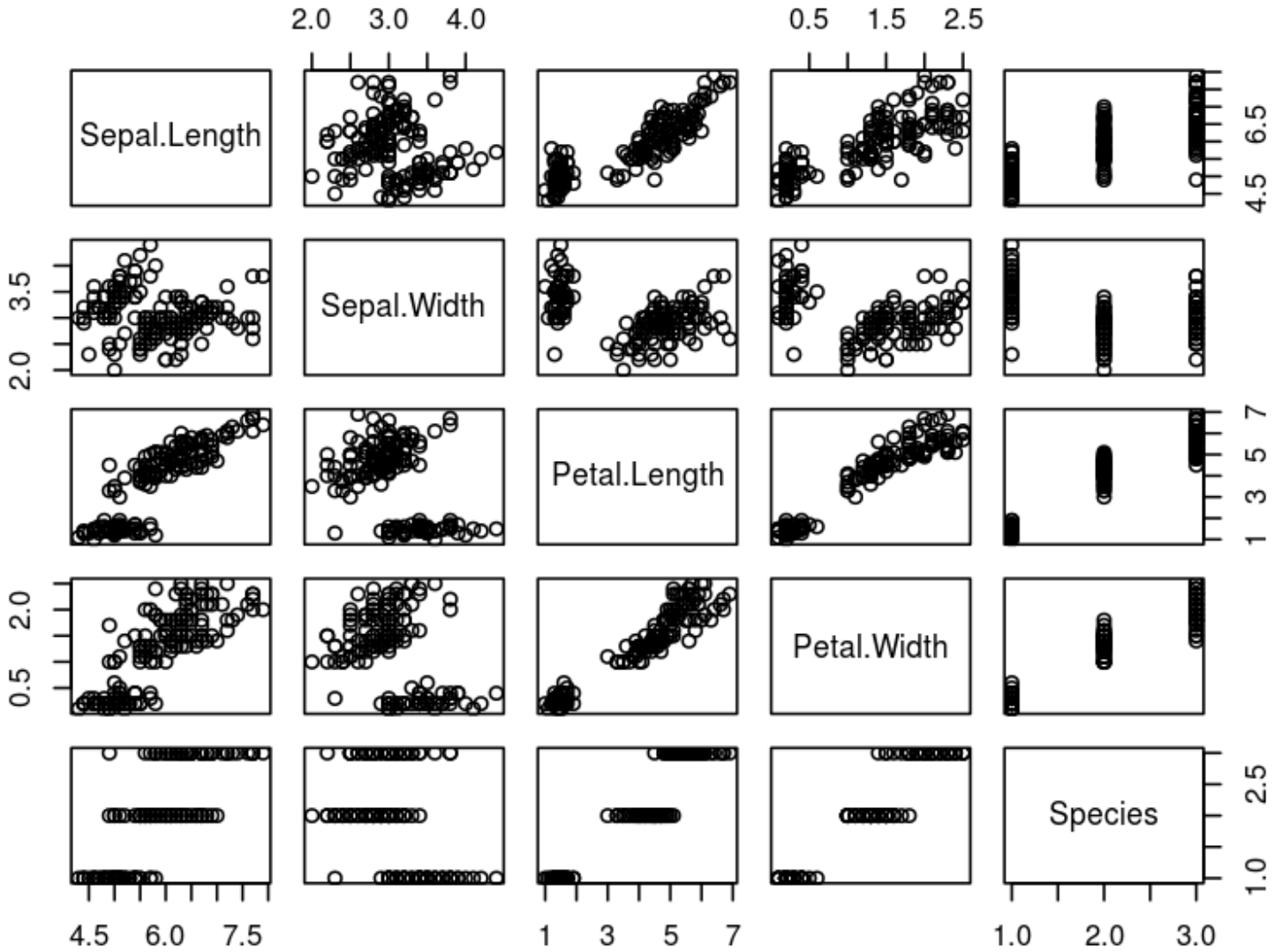


Fonctions graphiques de haut niveau

Scatterplot matrix (matrice de nuages de points)

`pairs(data.frame)`

```
pairs(iris)
```



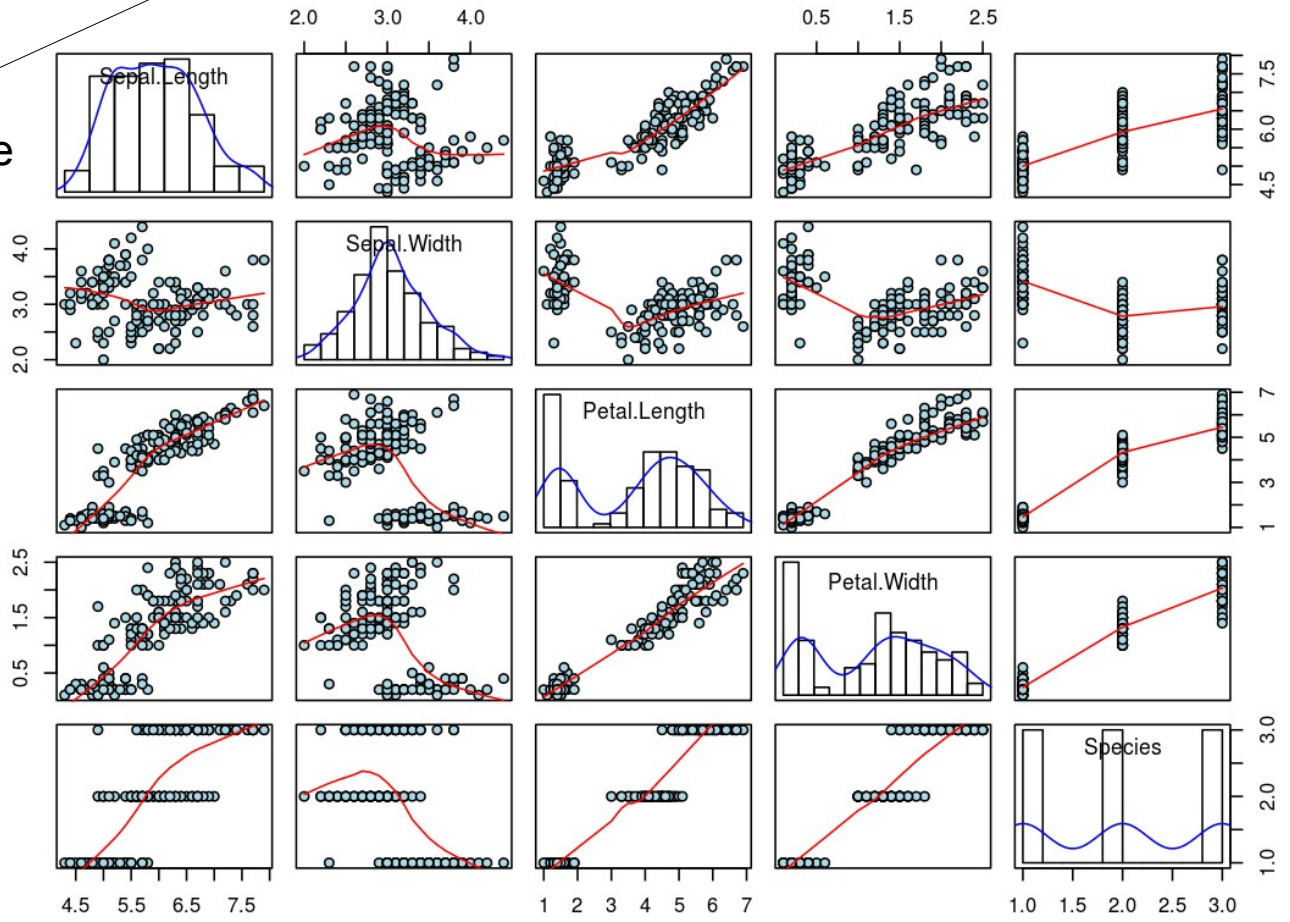
Fonctions graphiques de haut niveau

Scatterplot matrix (matrice de nuages de points)
On peut personnaliser fortement le contenu

```
panel.hist <- function(x, ...) {
    par(new=T)
    hist(x,main="",axes=FALSE,nclass=12, proba = TRUE,...)
    lines(density(x, na.rm = TRUE), lwd = 1, col = "blue",...)
}
pairs(iris, panel = panel.smooth, diag.panel = panel.hist,
      pch = 21, bg = "light blue")
```

Fonction personnalisée

Fonction disponible par défaut



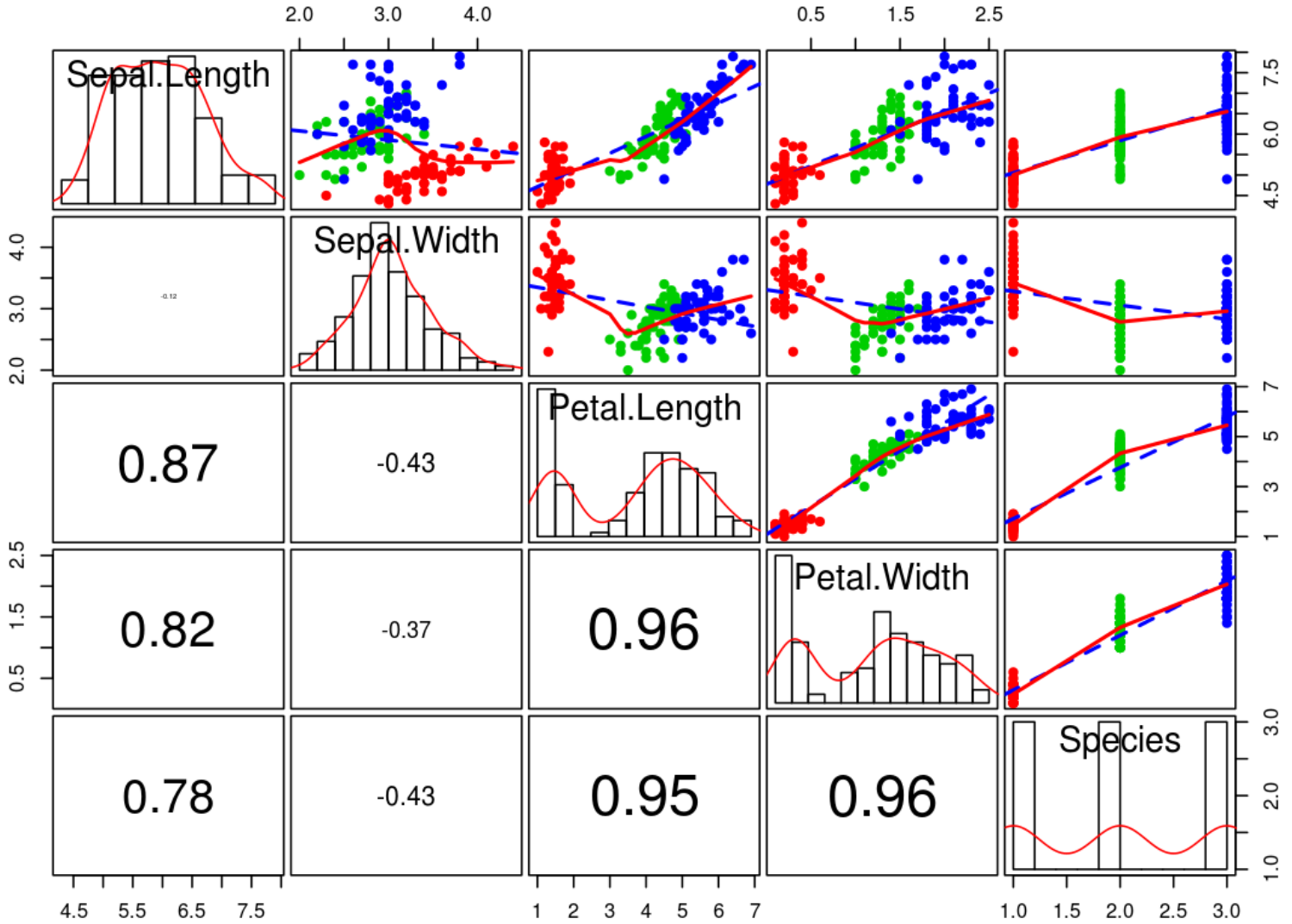
Fonctions graphiques de haut niveau

Scatterplot matrix (matrice de nuages de points)
On peut personnaliser fortement le contenu

```
pairs2 <- function (var, scaleR=TRUE, Rmethod="pearson",gap=0.3,
    col= "black",pch = 1, pt.cex = 0.9,...){
    pairs(var, gap=gap,
          upper.panel=function(x,y, ...) {
              d <- na.omit(data.frame(x = x, y = y, col = col, pch=pch))
              points(d$x,d$y,cex=pt.cex, col = as.character(d$col),
                 pch = pch,...)
              abline(lm(y~x, data=d),lty=2,lwd=2,col="blue")
              lines(lowess(d$x,d$y),col="red", lwd=2)
          },
          lower.panel=function(x,y,method=Rmethod){
              usr <- par("usr"); on.exit(par(usr))
              par(usr = c(0, 1, 0, 1))
              r <-cor(x,y,use="pairwise.complete.obs", method=Rmethod)
              txt <- round(r,2)
              if (scaleR) {text(0.5,0.5,txt,cex= 3*(abs(r)+0.001)) }
              else {text(0.5,0.5,txt,cex= 2.5) }
          },
          diag.panel=function(x){
              par(new=T)
              hist(x,main="",axes=FALSE,nclass=12, proba = TRUE)
              lines(density(x, na.rm = TRUE), lwd = 1, col = "red")

          })
}

pairs2(iris, pch = 19, col = c("red", "green3", "blue")[unclass(iris$Species)])
```

Fonctions graphiques de haut niveau

Scatterplot matrix (matrice de nuages de points)
On peut personnaliser fortement le contenu



Fonctions graphiques de haut niveau

Pixels colorés sur base d'une matrice

`image(numeric, numeric, numeric)` ou
`image(matrix)`

Volcano : coordonnées x, y et altitude :

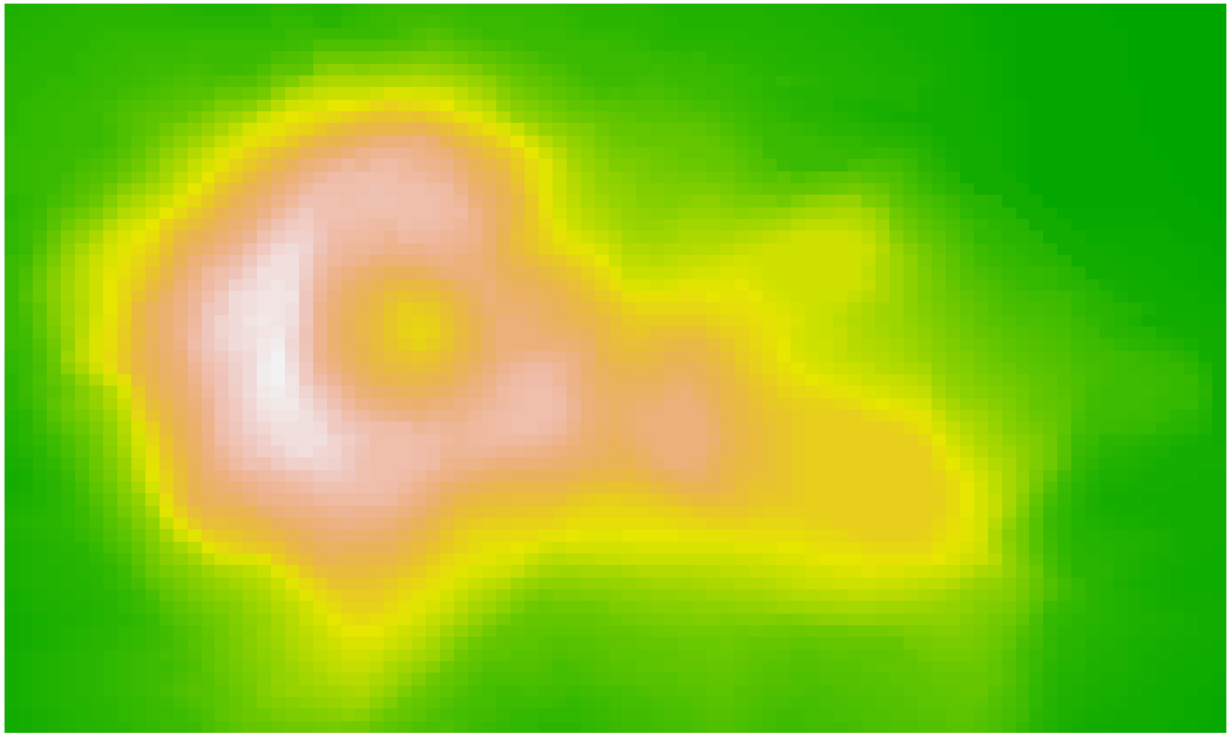
```
> volcano[1:10, 1:10]
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]  100  100  101  101  101  101  101  100  100   100
 [2,]  101  101  102  102  102  102  102  101  101   101
 [3,]  102  102  103  103  103  103  103  102  102   102
 [4,]  103  103  104  104  104  104  104  103  103   103
 [5,]  104  104  105  105  105  105  105  104  104   103
 [6,]  105  105  105  106  106  106  106  105  105   104
 [7,]  105  106  106  107  107  107  107  106  106   105
 [8,]  106  107  107  108  108  108  108  107  107   106
 [9,]  107  108  108  109  109  109  109  108  108   107
[10,]  108  109  109  110  110  110  110  109  109   108
```

Fonctions graphiques de haut niveau

Pixels colorés sur base d'une matrice

`image(numeric, numeric, numeric)` ou
`image(matrix)`

```
image(volcano, col = terrain.colors(100), axes = FALSE)
```



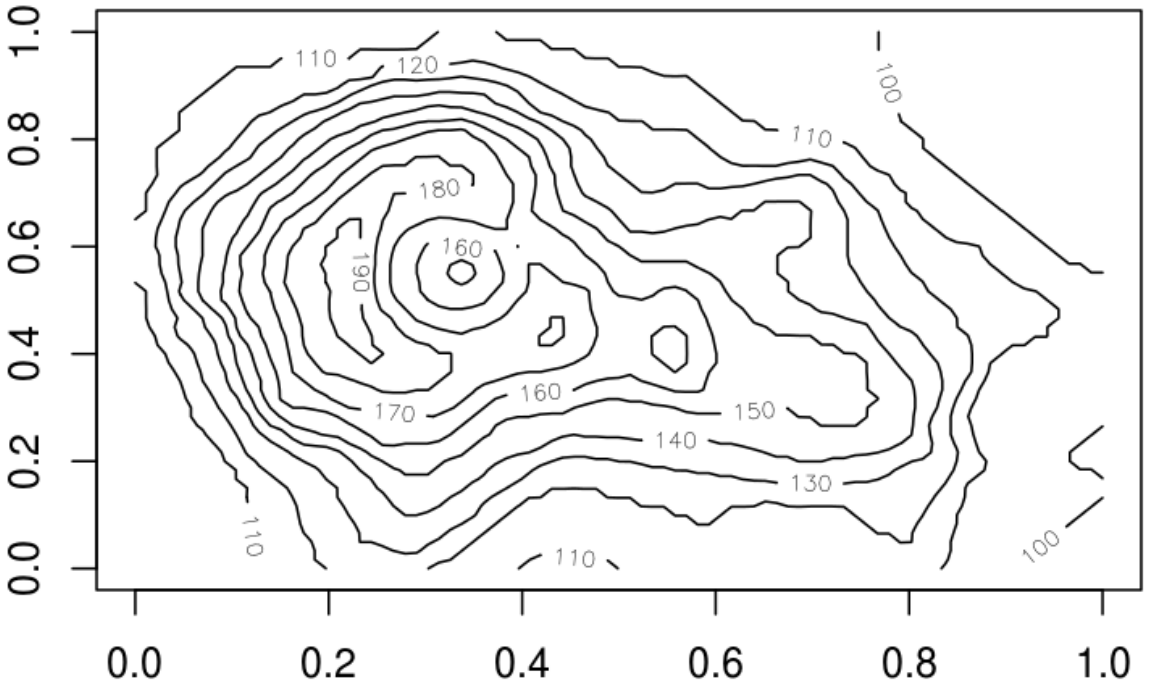
Fonctions graphiques de haut niveau

contours : lignes délimitant des gammes de valeurs identiques

`contour(numeric, numeric, numeric)` ou

`contour(matrix)`

```
contour(volcano)
```



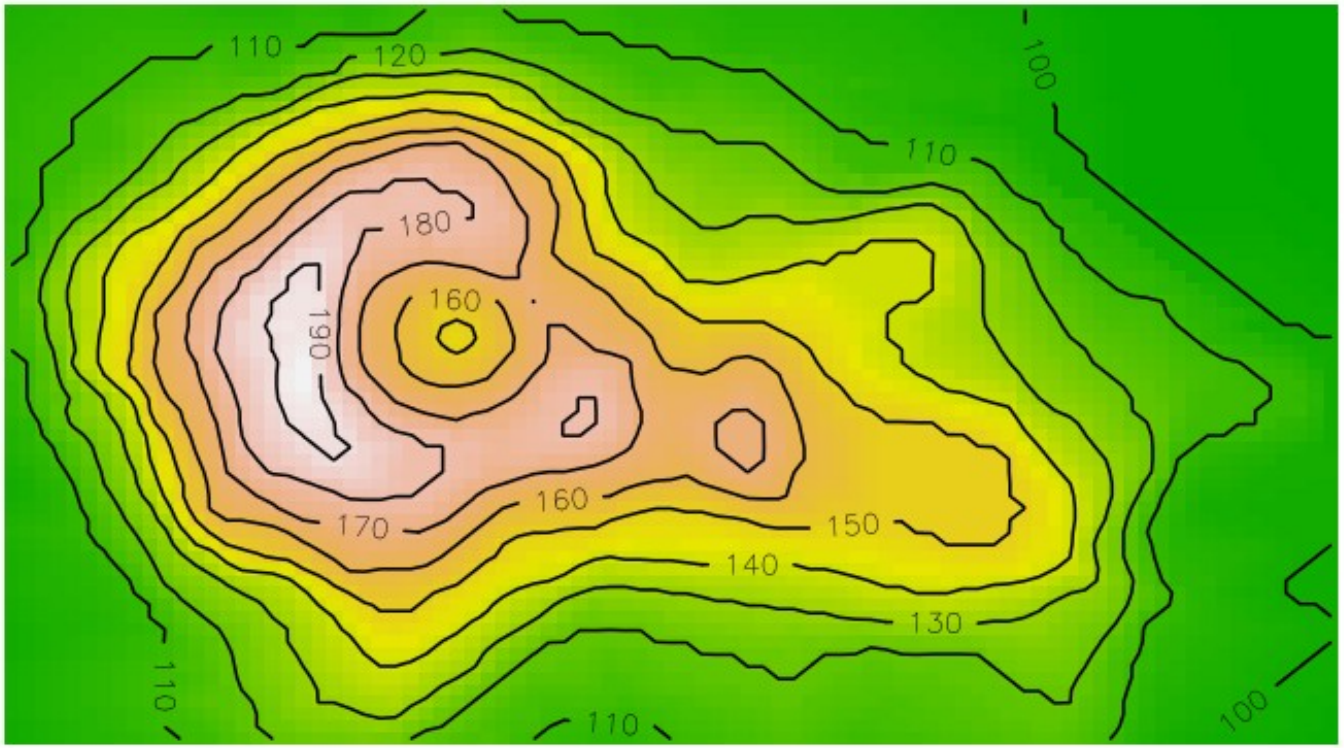
Fonctions graphiques de haut niveau

contours : lignes délimitant des gammes de valeurs identiques

`contour(numeric, numeric, numeric)` ou

`contour(matrix)`

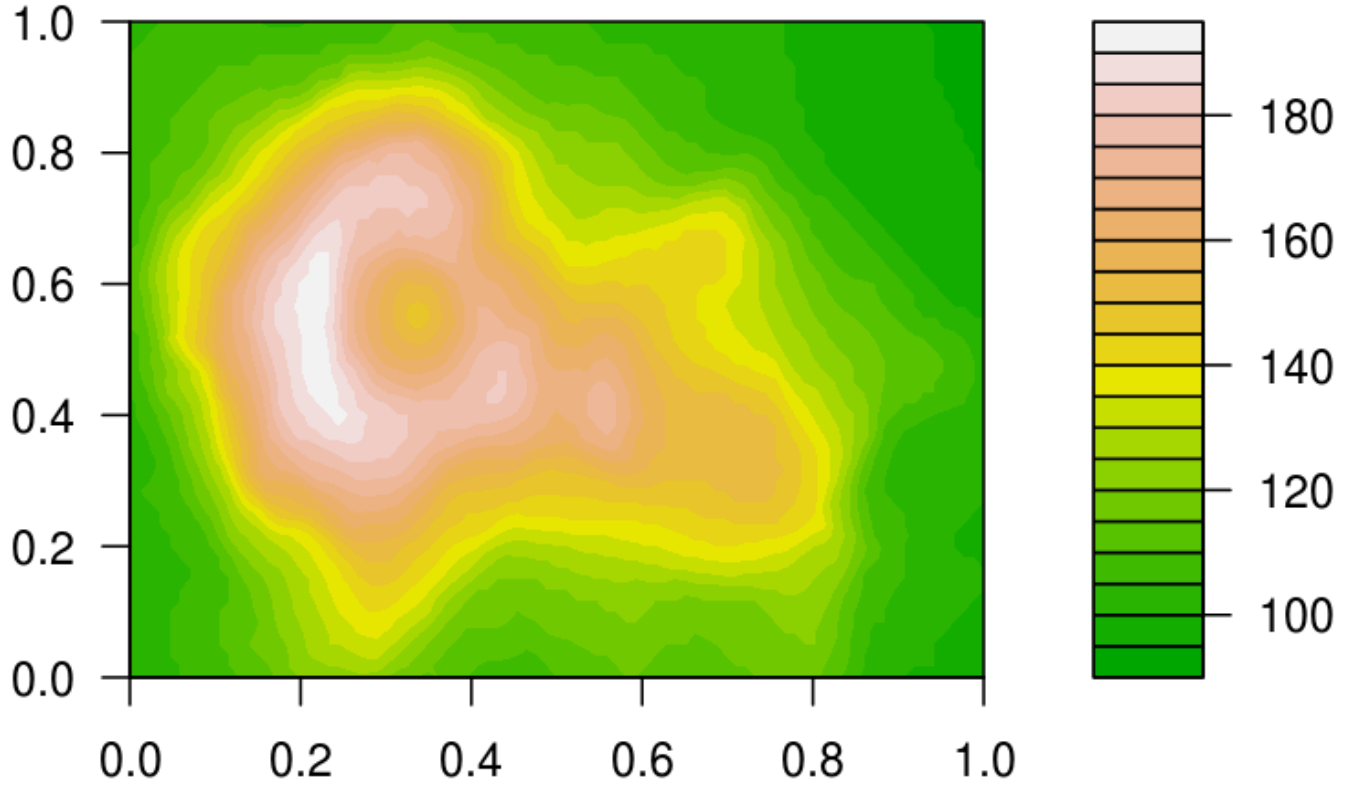
```
image(volcano, col = terrain.colors(100), axes = FALSE)
contour(volcano, add=TRUE)
```



Fonctions graphiques de haut niveau

filled.contour : comme contour mais zones entre deux courbes remplie d'une couleur + échelle
Plus difficile d'ajouter des choses sur ce type de graphiques que sur des graphiques image()
`filled.contour(numeric, numeric, numeric)` ou `filled.contour(matrix)`

```
filled.contour(volcano, color = terrain.colors)
```



Fonctions graphiques de haut niveau

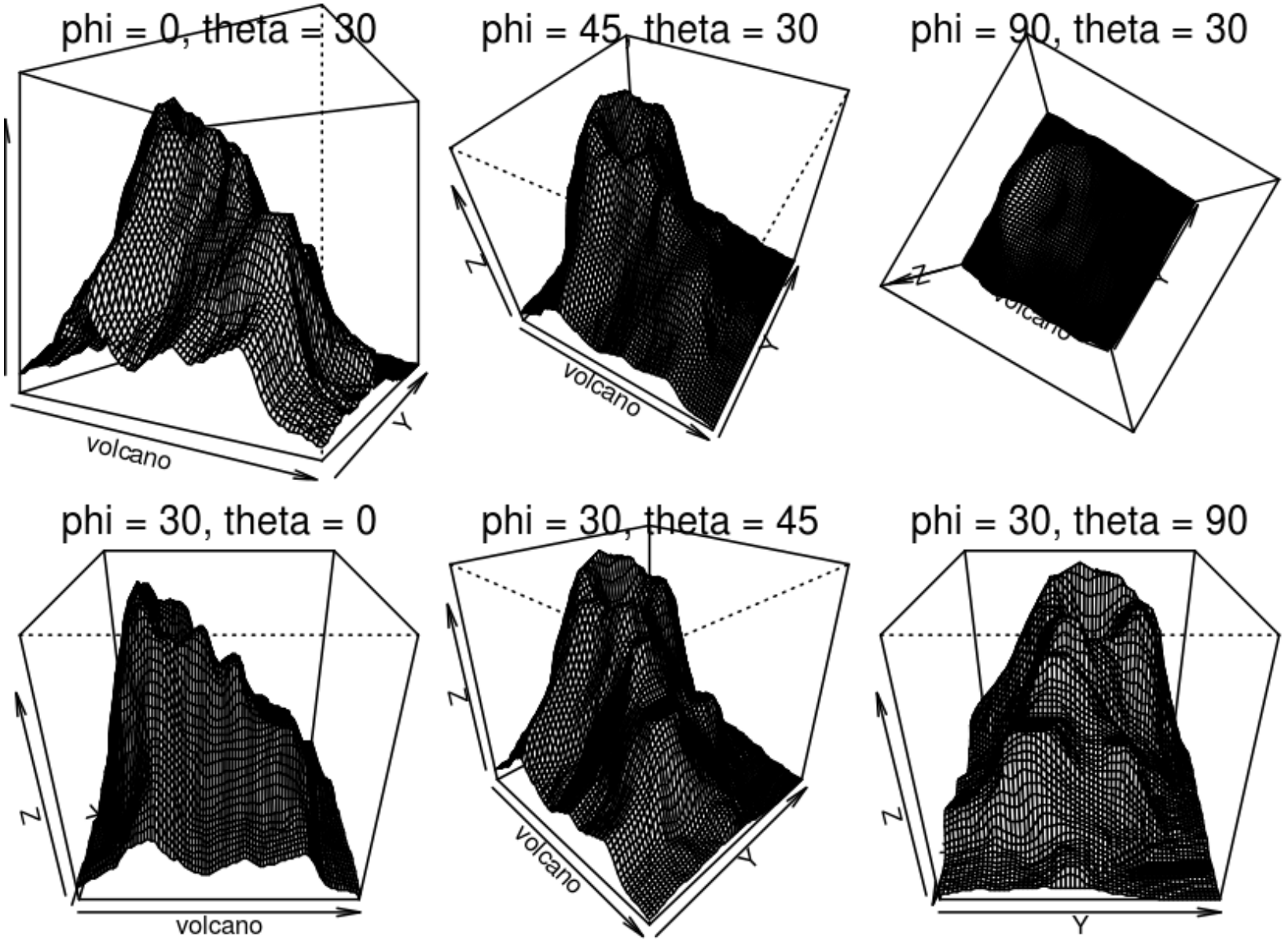
persp : surface 3D
argument theta : rotation gauche-droite
argument phi : rotation haut-bas

`persp(numeric, numeric, numeric)` ou
`persp(matrix)`

```
par(mfrow = c(2,3), mar = c(0,0,0,0))
persp(volcano, phi = 0, theta = 30) ; mtext('phi = 0, theta = 30', line = -2)
persp(volcano, phi = 45, theta = 30) ; mtext('phi = 45, theta = 30', line = -2)
persp(volcano, phi = 90, theta = 30) ; mtext('phi = 90, theta = 30', line = -2)
persp(volcano, phi = 30, theta = 0) ; mtext('phi = 30, theta = 0', line = -2)
persp(volcano, phi = 30, theta = 45) ; mtext('phi = 30, theta = 45', line = -2)
persp(volcano, phi = 30, theta = 90) ; mtext('phi = 30, theta = 90', line = -2)
```

Fonctions graphiques de haut niveau

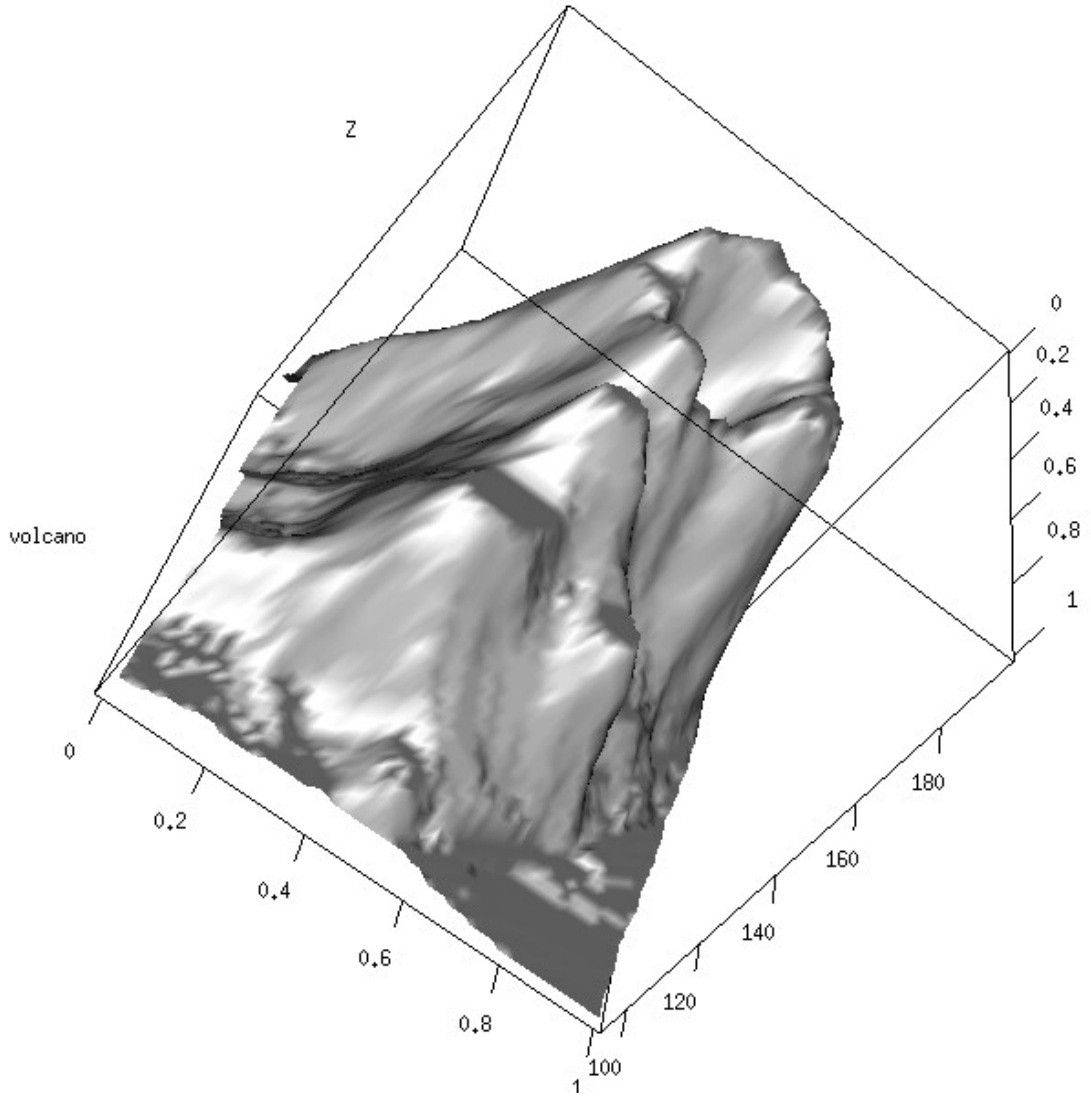
persp : surface 3D.



Fonctions graphiques de haut niveau

persp3D (package rgl) : surface 3D interactive

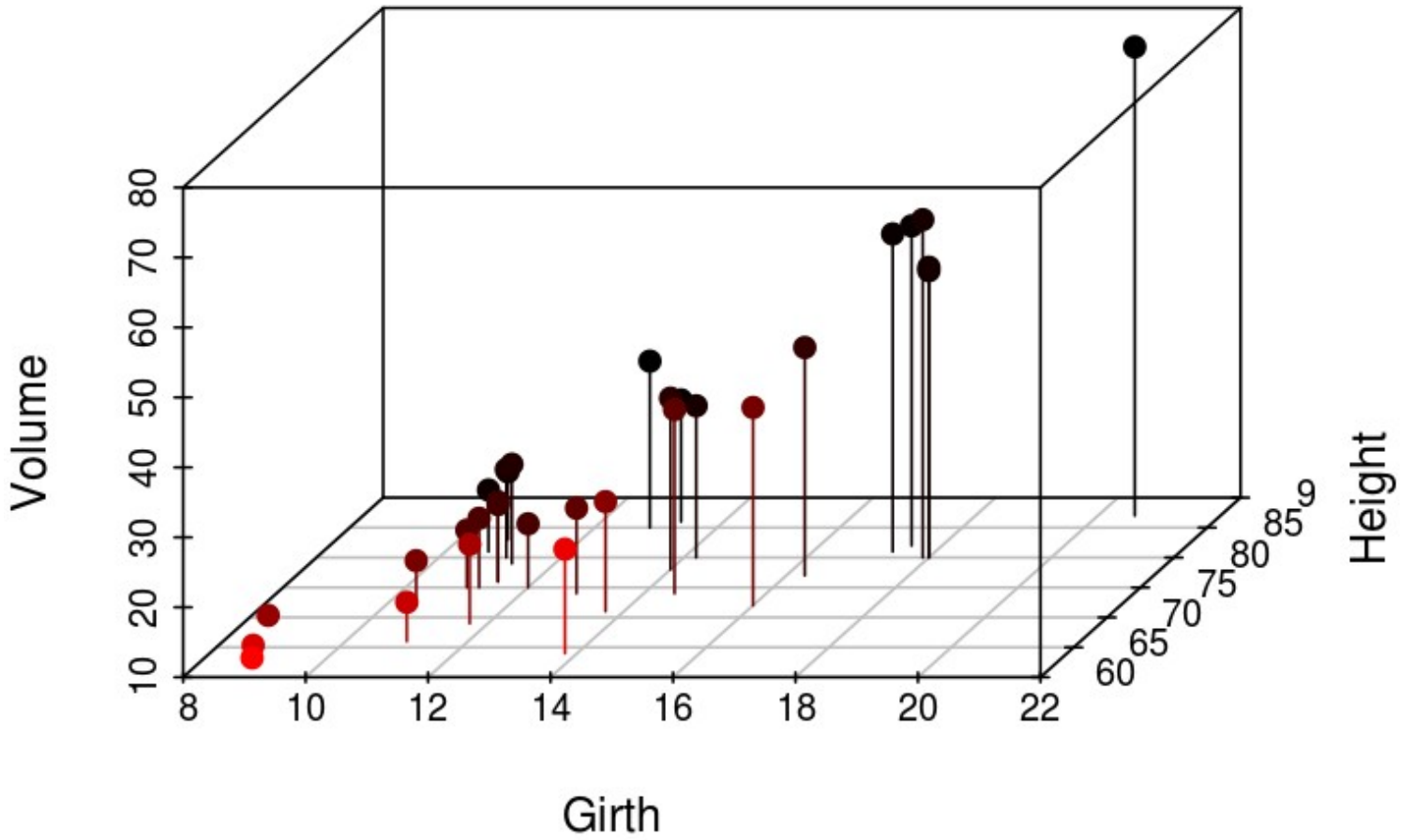
```
library(rgl)
persp3d(volcano, col = "gray", aspect = TRUE)
```



Fonctions graphiques de haut niveau

scatterplot3D (package scatterplot3D) : scatterplot 3D

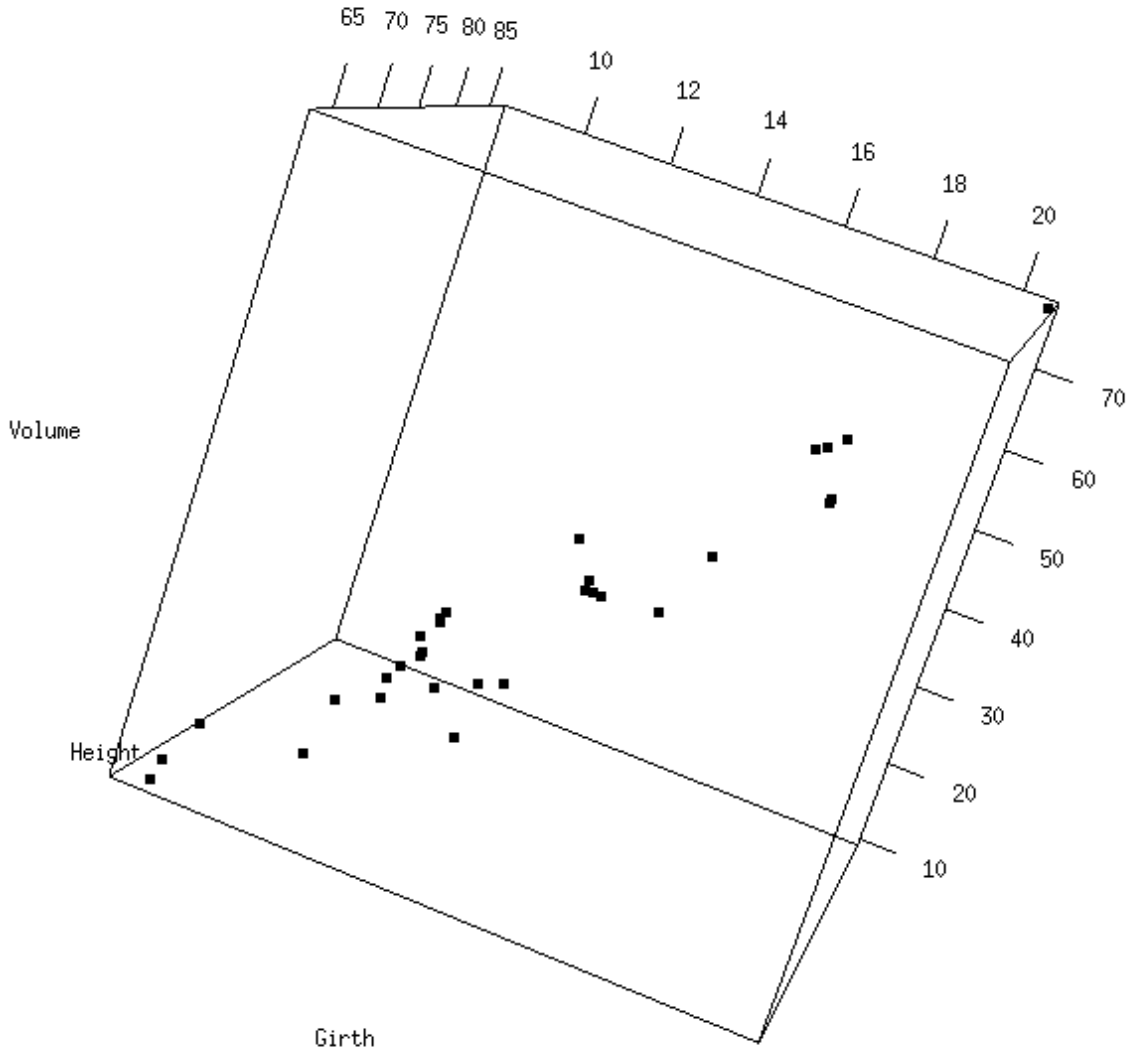
```
library(scatterplot3d)
data(trees)
s3d <- scatterplot3d(trees, type="h", highlight.3d=TRUE,
                     angle=55, scale.y=0.7, pch=19)
```



Fonctions graphiques de haut niveau

plot3D (package rgl) : scatterplot 3D interactif

```
library(rgl)
plot3d(trees, size = 5, pch = 19, aspect = TRUE)
```



Fonctions graphiques de bas niveau

`points(x, y)`	points (symboles) en position xy
`lines(x,y)`	lignes (courbes) reliant des points xy

`abline(v=0) ; abline(h=0)`
`abline(a,b) ;`
`abline(lm.object)`
	ligne droite soit verticale soit horizontale soit définie par une pente et un intercept soit droite de régression (lm.object)

`segments(x0,y0,x1,y1)`	segment de droite entre (x0,y0) et (x1,y1)
`arrows(x0,y0,x1,y1,angle=30)`	segment avec une pointe de flèche
`rect(xl,yb,xr,yt)`	rectangle avec coins en (xl, yb) et (xr, yt)
`polygon (x,y)`	polygone reliant les points xy
`text(x,y,labels)`	texte aux points xy
`maptools::pointLabel` `plotrix::thigmophobe.labels`	étiquettes limitant les chevauchements
`legend(x,y,legend)`	légende (nombreuses options)
`title`	ajoute titres et légende des axes
`mtext(text, side=3, line=0)`	texte dans les marges
`axis`	ajoute un axe

Fonctions graphiques de bas niveau

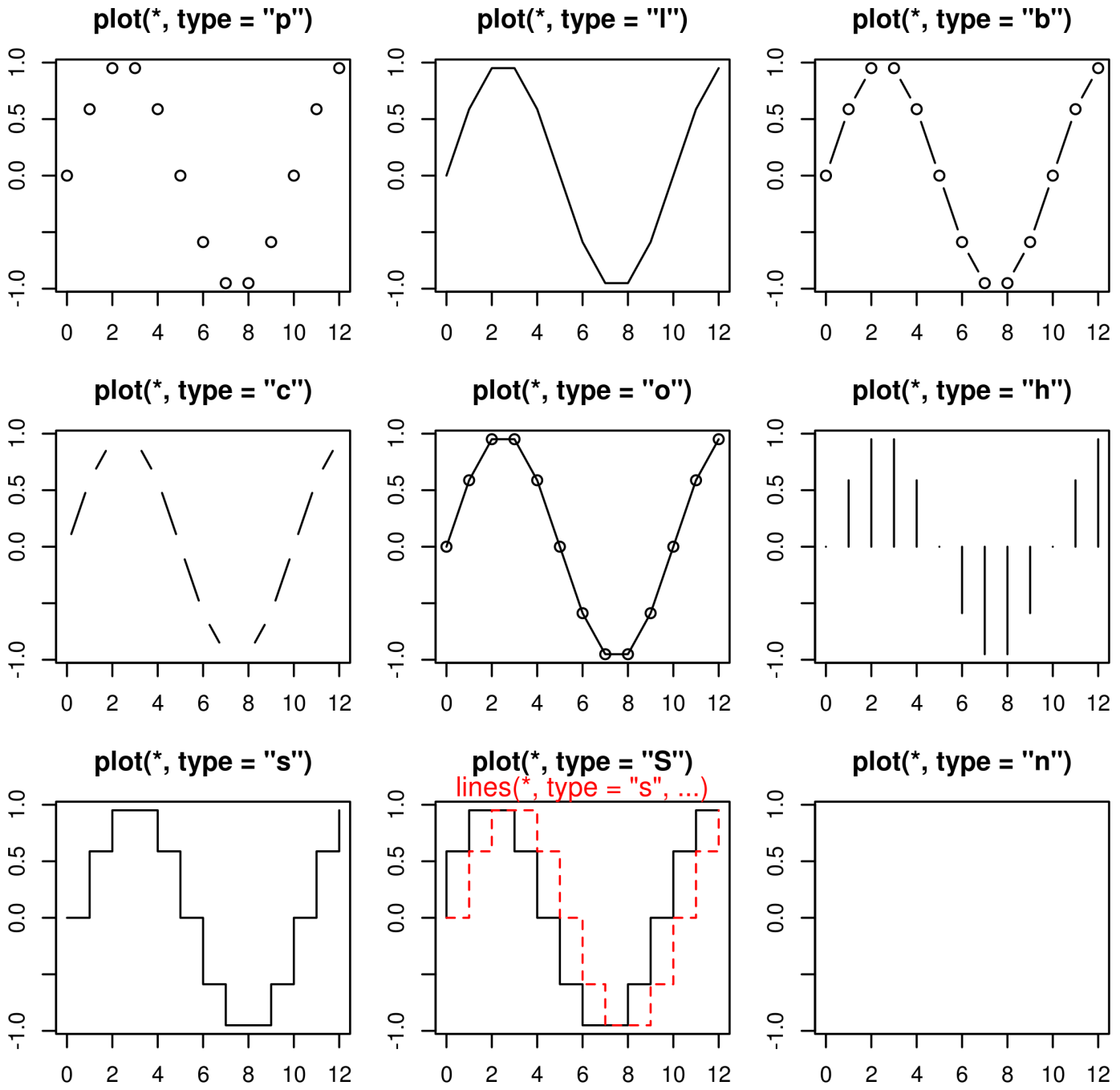
```
g1 <- rnorm(10, 20, 5)
g2 <- rnorm(10, 60, 5)

par(mar = c(3,3,3,3), mgp = c(2, 0.5, 0))
plot(1, type="n", axes = FALSE, xlim = c(0,100), ylim = c(0,100), ylab = "Y",
xlab = "X")
title("mon graphique")
```

Graphique vide
(cfr `type = "n", axes = FALSE`)

mon graphique

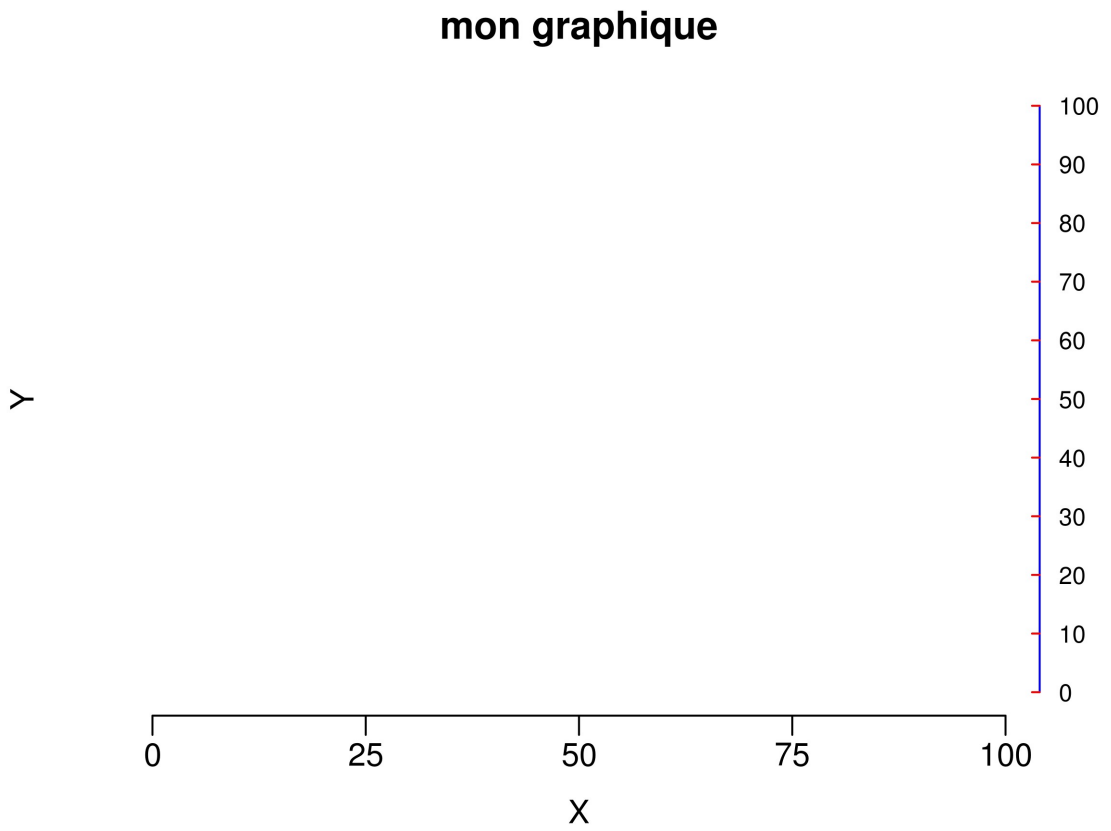
X

Différents types de plots



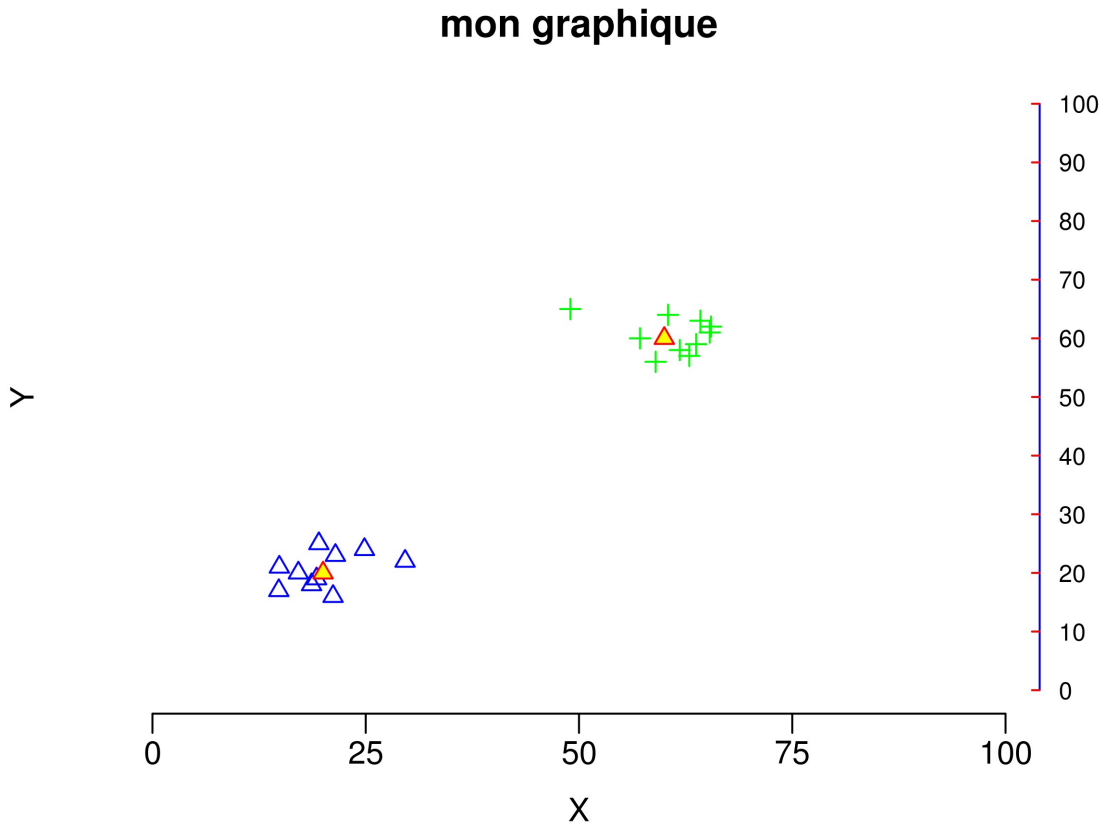
Fonctions graphiques de bas niveau

```
axis(side = 1, at = c(0,25, 50, 75, 100), labels = c(0,25, 50, 75, 100))
axis(side = 4, at = seq(0,100, 10), labels = seq(0,100, 10), col = "blue",
col.ticks = "red", las = 2,  tcl = 0.2, cex.axis = 0.75)
```



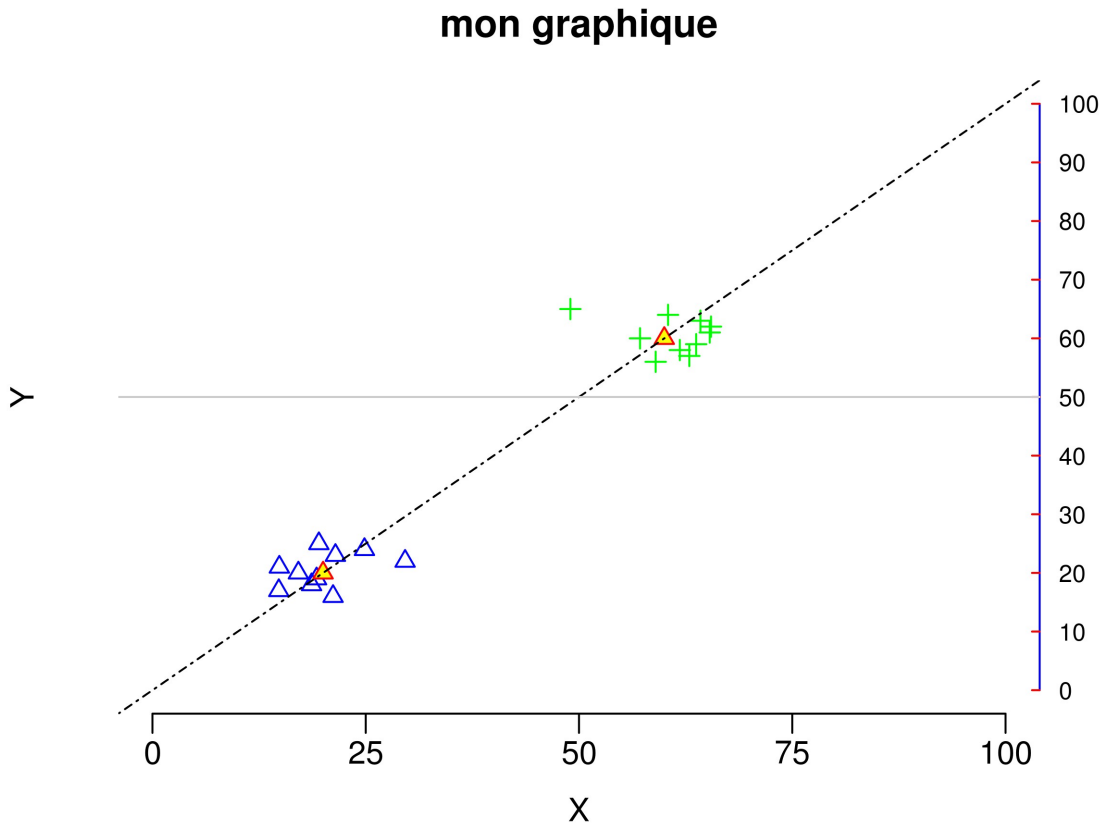
Fonctions graphiques de bas niveau

```
points(g1, 16:25, pch = 2, col = "blue")
points(g2, 56:65 , pch = 3, col = "green")
points(c(20, 60),c(20,60), pch = 24, bg = "yellow", col = "red")
```



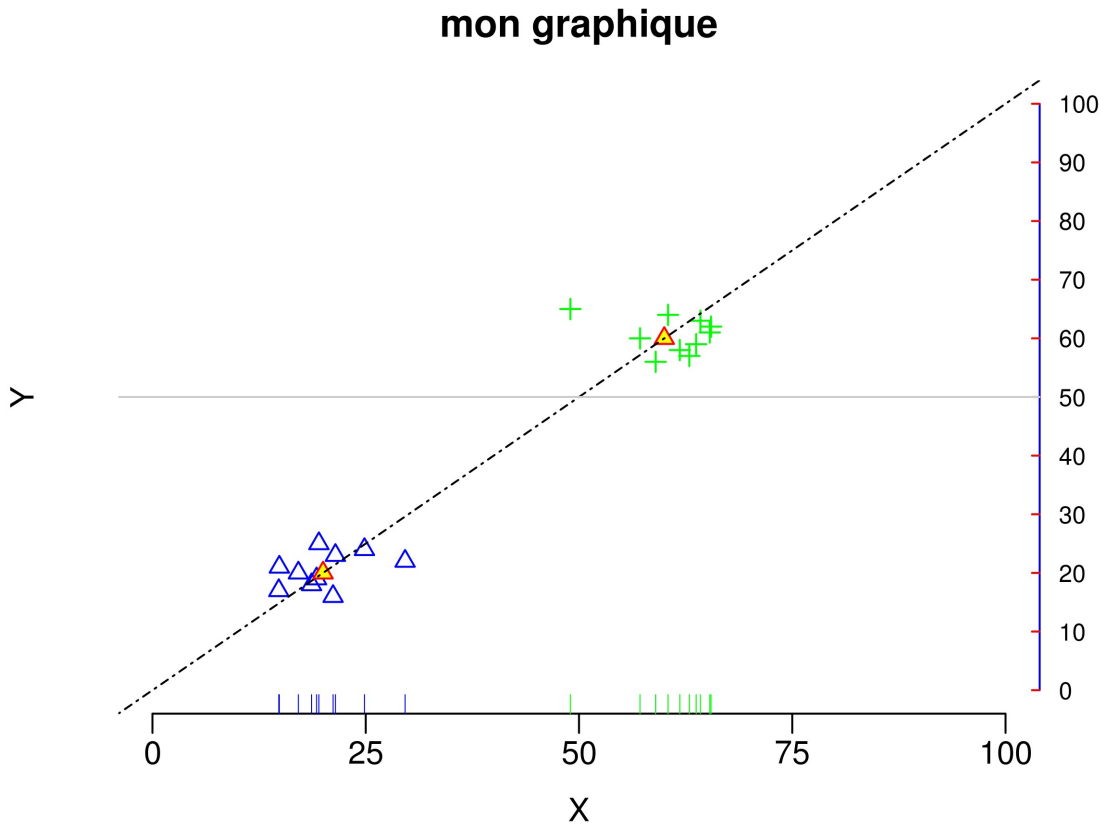
Fonctions graphiques de bas niveau

```
abline(a = 0, b=1, lty = 4)
abline(h=50, col = "grey80")
```



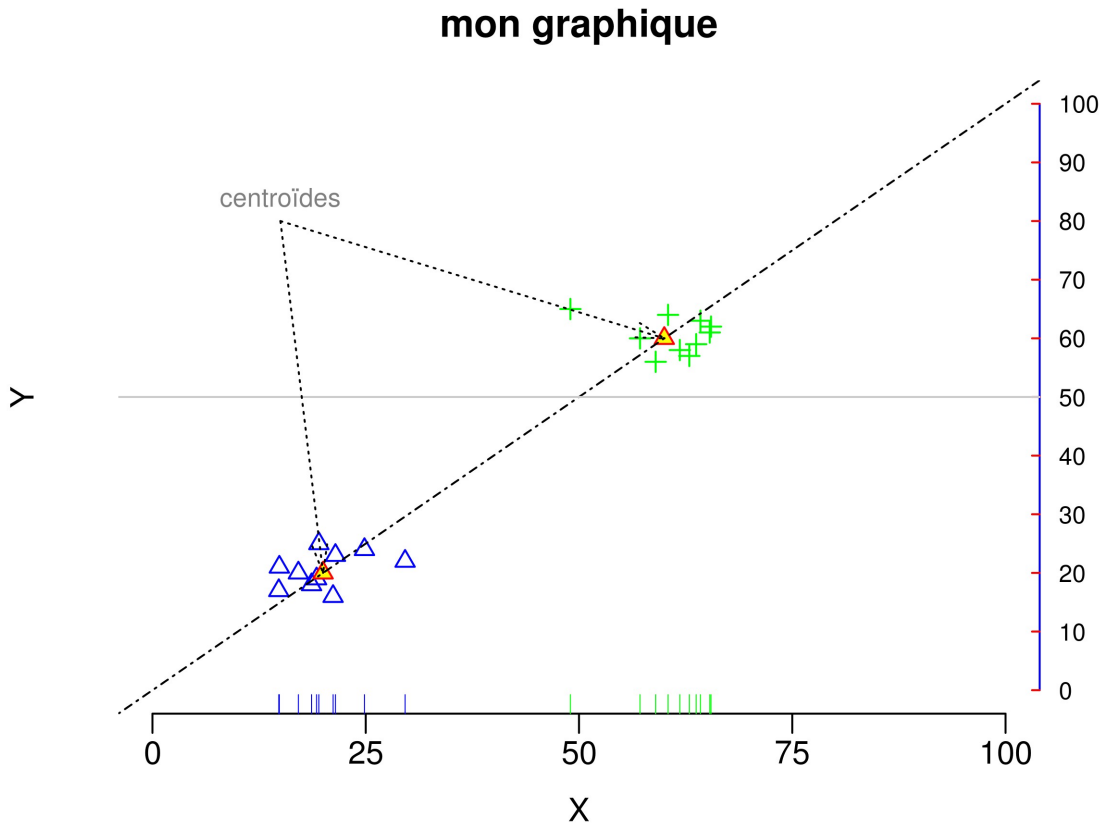
Fonctions graphiques de bas niveau

```
rug(g1, col = "blue")
rug(g2, col = "green")
```



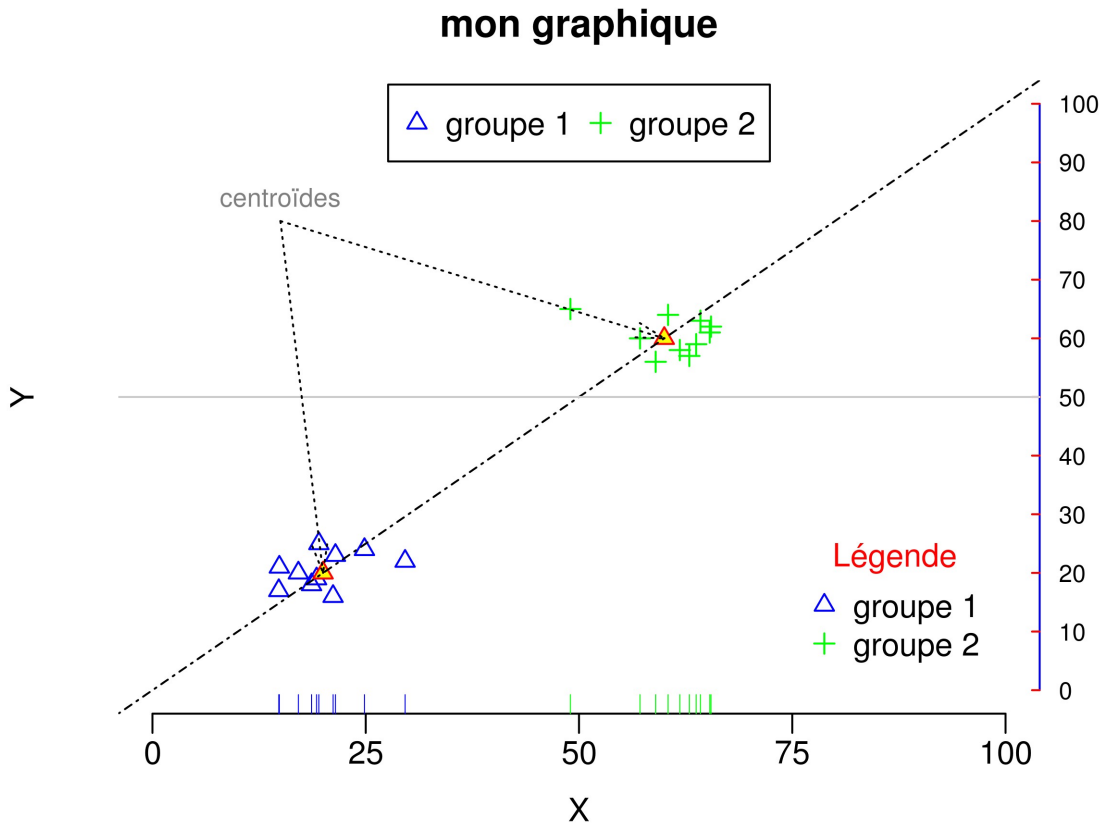
Fonctions graphiques de bas niveau

```
arrows(x0= c(15,15), y0 = c(80, 80), x1 = c(20,60), y1 = c(20,60), length =
0.15, angle = 15, lty=3)
text(15, 84, labels = "centroïdes", cex = 0.8, col = "grey50")
```



Fonctions graphiques de bas niveau

```
legend(x=50, y=100, legend = c("groupe 1", "groupe 2"),
    col = c("blue", "green"), pch = c(2,3),
    xpd = TRUE, xjust = 0.5, yjust = 0.75, ncol = 2 )

legend(x = "bottomright", xpd = TRUE, legend = c("groupe 1", "groupe 2"),
    col = c("blue", "green"), pch = c(2,3),
    inset = 0.05, bty="n", title = "Légende", title.col = "red")
```



Dispositifs graphiques

"Graphical devices"
Ouvrir une nouvelle fenêtre graphique de taille prédéfinie à l'écran

Paramètres width et height en pouces
Diviser les dimensions en pouce par 2.54 pour pouvoir les exprimer en cm

Définir la taille est **TRES** important si vous voulez utiliser vos graphiques en dehors de R
(pour éviter des problèmes avec la taille des marges, des polices, etc)

Fonctions : `windows()`, `quartz()` et `x11()`
pour Windows, Mac OS, et GNU/Linux respectivement

Fonction `dev.new()` : fonctionne pour tous les systèmes
Mais incompatible avec R studio <= 0.99

```
dev.new(width = 12/2.54, height = 10/2.54)
plot(1)
dev.off()
```

Ferme la fenêtre

Dispositifs graphiques

Pour utiliser `dev.new()` dans Rstudio, utiliser par exemple la fonction suivante :

```
dev.new <- function(width = 10/2.54, height = 10/2.54) {

    platform <- sessionInfo()[[1]]$platform
    if (grepl("linux",platform)) {
        x11(width=width, height=height)
    } else if (grepl("pc",platform)) {
        windows(width=width, height=height)
    } else if (grepl("apple", platform)) {
        quartz(width=width, height=height)
    }
    par(bg="white")
}
```


Pas testé sous Mac...

Dispositifs graphiques

Sauver les graphiques dans des fichiers :

Format bitmap (png : le plus recommandé)

Bien spécifier la résolution (au moins 300 dpi)

`png()`, `jpg()`, `bmp()`, etc...

Format vectoriel :

`pdf()`, `cairo_pdf()` `postscript()`, `svg()`

Vectoriel : idéal car peut être redimensionné et modifié par la suite dans un programme de dessin vectoriel (pex Inkscape)

Mais parfois certains symboles / lettres sont modifiés accidentellement
Utiliser la fonction `embedFonts` une fois le graphique créé pour inclure les polices et limiter ce genre de problèmes ou bien utiliser `cairo_pdf()` à la place de `pdf()`

```
png(filename = "Mongraphique.png", width = 12, height = 10,
    units = "cm", pointsize = 12, res = 300)
plot(1)
dev.off()
```

Dispositifs graphiques

Sauver les graphiques dans des fichiers :

On peut aussi utiliser `dev.print ()` pour sauver le dernier graphique affiché à l'écran dans le format de son choix

```
dev.new(width = 12/cm(1), height = 10/cm(1))

plot(1)

dev.print(png, "myplot.png", width = 12/cm(1),
            height = 10/cm(1), units = "in", res=300)
dev.off()
```

Dispositifs graphiques

Sauver les graphiques dans des fichiers :

Vous pouvez écrire vos fonctions utilisant `dev.print ()` pour sauver le dernier graphique affiché à l'écran à la taille à laquelle il est afiché

```
savepng <- function(filename="Rplot%03d.png", res=400, units="in", width =
            par("din")[1], height = par("din")[2]) {
                    dev.print(filename=filename, device= png, res=res, width =
                    width, height=height, units= units)
            }

savepdf <- function(file= "Rplot%03d.pdf", width = par("din")[1], height =
                    par("din")[2]) {
             dev.print(file=file, device= cairo_pdf, width = width, height=height)
            }


plot(rnorm(20))
savepng()
savepdf("Mongraphique.pdf")
```

Extrait les dimensions de la fenêtre graphique affichée à l'écran

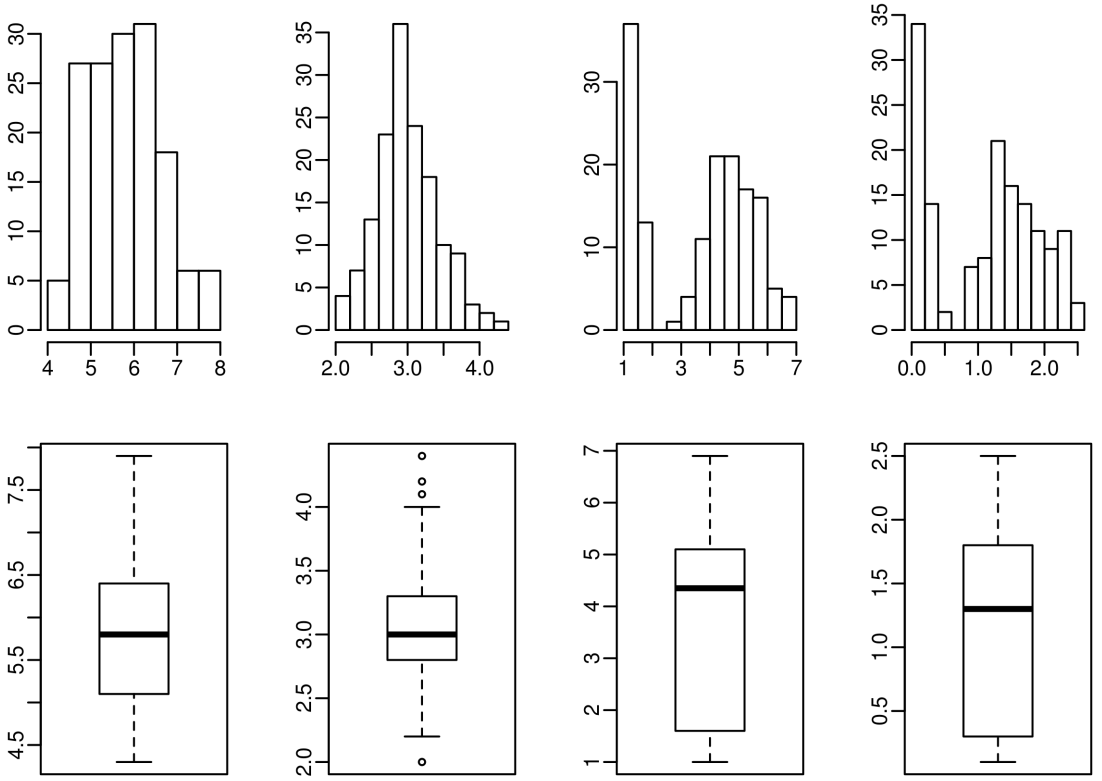
Diviser le device graphique

Permet d'afficher plusieurs graphiques sur la même figure

Le plus simple : paramètre `mfrow` qui divise le graphique en zones de dimensions identique en lignes et colonnes :

division en 2 lignes et 4 colonnes

```
data(iris)
par(mar = c(2,2,2,2) mfrow = c(2,4))
lapply(iris[,-5], hist, main = "")
lapply(iris[,-5], boxplot)
```

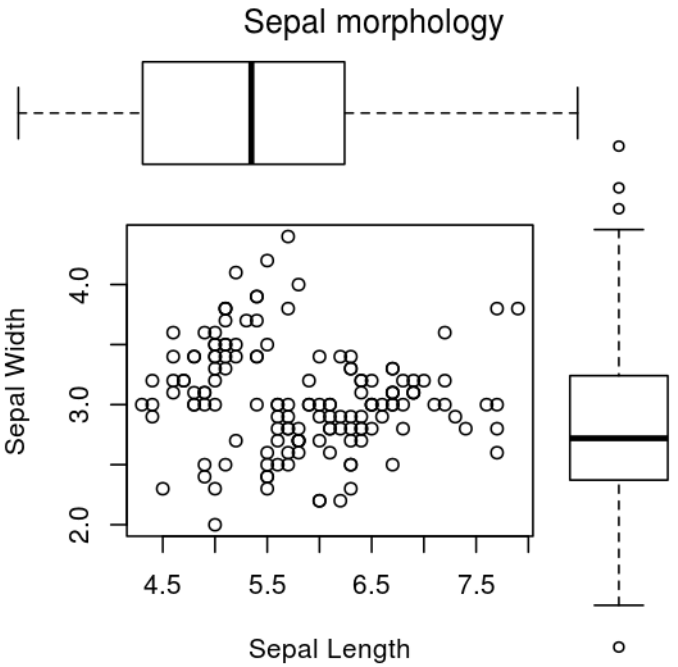


Diviser le device graphique

`layout()` permet de faire des subdivisions beaucoup plus complexes :

Division en 3 zones de tailles différentes

```
data(iris)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE),
       widths=c(3,1), heights=c(1,2))

par(fig=c(0,0.8,0,0.8), bg="white")
plot(iris$Sepal.Length, iris$Sepal.Width, xlab="Sepal Length", ylab="Sepal
Width")
par(mar = c(1,0,1,0), fig=c(0,0.8,0.55,1), new=TRUE)
boxplot(iris$Sepal.Length, horizontal=TRUE, axes=FALSE)
par(fig=c(0.65,1,0,0.8),new=TRUE)
boxplot(iris$Sepal.Width, axes=FALSE)
mtext("Sepal morphology", side=3, outer=TRUE, line=-3)
```



Paramètres graphiques

Très nombreux, à apprendre "sur le tas"
--> le plus long à acquérir...

Certains paramètres se définissent dans `par()`
--> valable pour tous les graphiques suivants tant qu'on ne ferme pas le device
Certains dans les arguments de la fonction
--> valable uniquement pour ce graphique
Certains dans les deux...

Voir `?par` pour la liste complète

+ aide des fonctions graphiques

+ exploration systématique de `par()` dans cet excellent document :
http://pbil.univ-lyon1.fr/R/pdf/tdr75.pdf

+ regarder des exemples existants (avec le code) pex :
gallery.r-enthusiasts.com

Paramètres graphiques

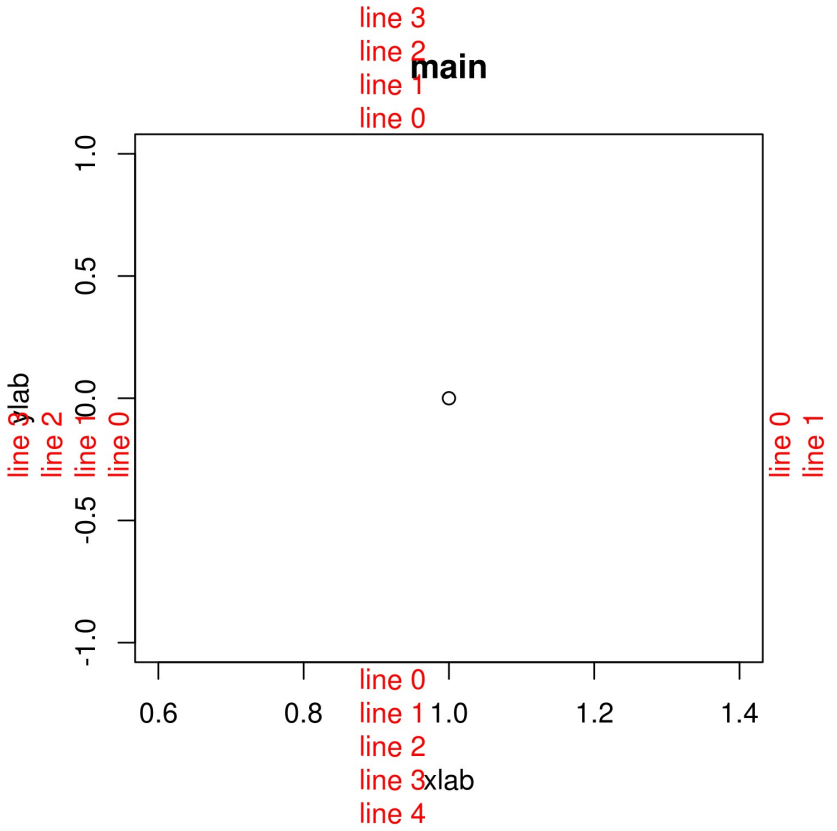
Dimensions des marges

```
par(mar = c(5.1, 4.1, 4.1, 2.1)
```

Nombre de lignes dans chaque marge

Bas, gauche, haut, droite

```
dev.new(width = 5, height = 5)
plot(0, xlab = "xlab", ylab = "ylab", main = "main")
for(i in 0:4) mtext(text = paste("line",i), side = 1, line = i, col = "red", adj = 0.4)
for(i in 0:3) mtext(text = paste("line",i), side = 2, line = i, col = "red", adj = 0.4)
for(i in 0:3) mtext(text = paste("line",i), side = 3, line = i, col = "red", adj = 0.4)
for(i in 0:2) mtext(text = paste("line",i), side = 4, line = i, col = "red", adj = 0.4)
```

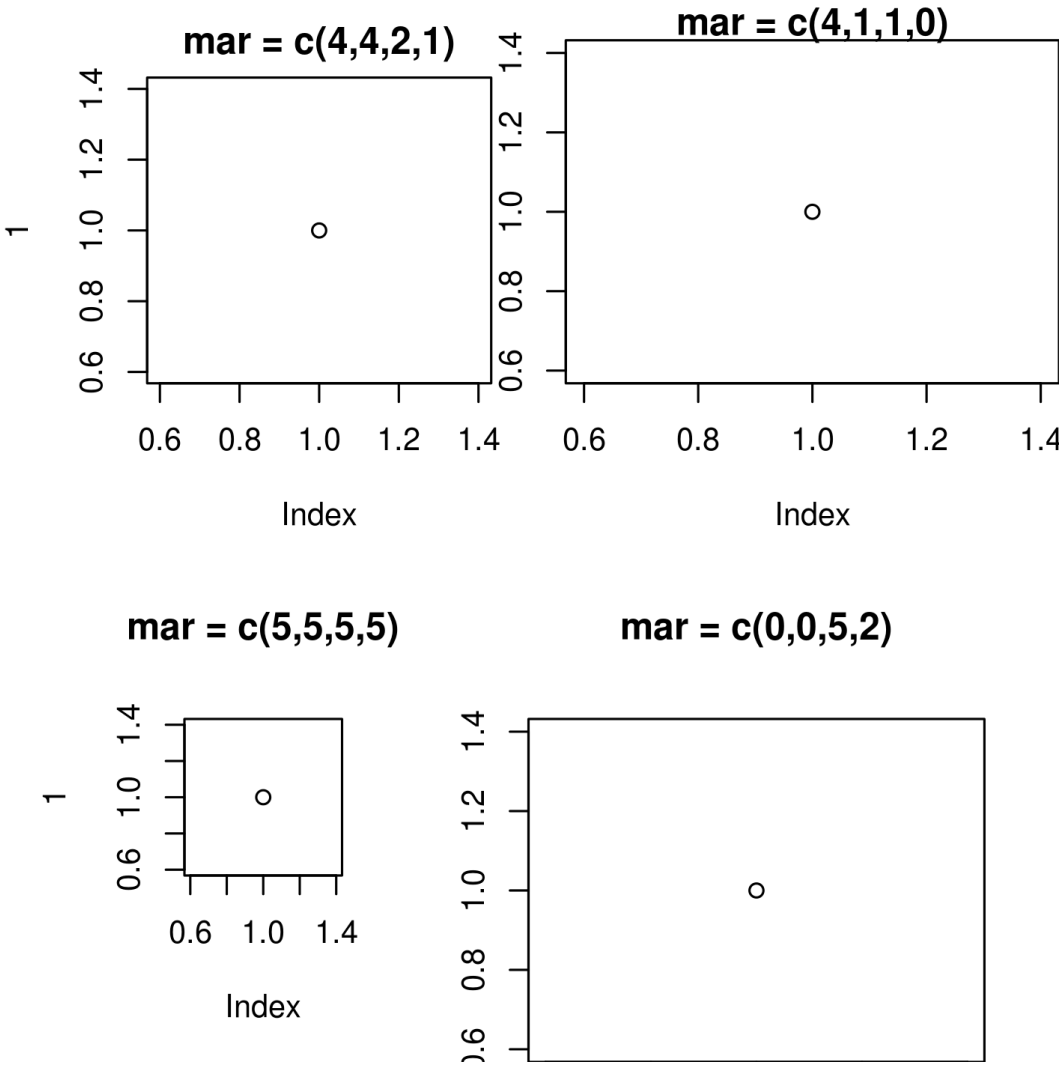


Paramètres graphiques

Dimensions des marges

`par(mar = c(5.1, 4.1, 4.1, 2.1)`

```
dev.new(width = 12/2.54, height = 12/2.54)

par(mfrow=c(2,2))
par(mar = c(4,4,2,1))
plot(1, main = "mar = c(4,4,2,1)")
par(mar = c(4,1,1,0))
plot(1, main = "mar = c(4,1,1,0)")
par(mar = c(5,5,5,5))
plot(1, main = "mar = c(5,5,5,5)")
par(mar = c(0,0,5,2))
plot(1, main = "mar = c(0,0,5,2)")
```



Paramètres graphiques

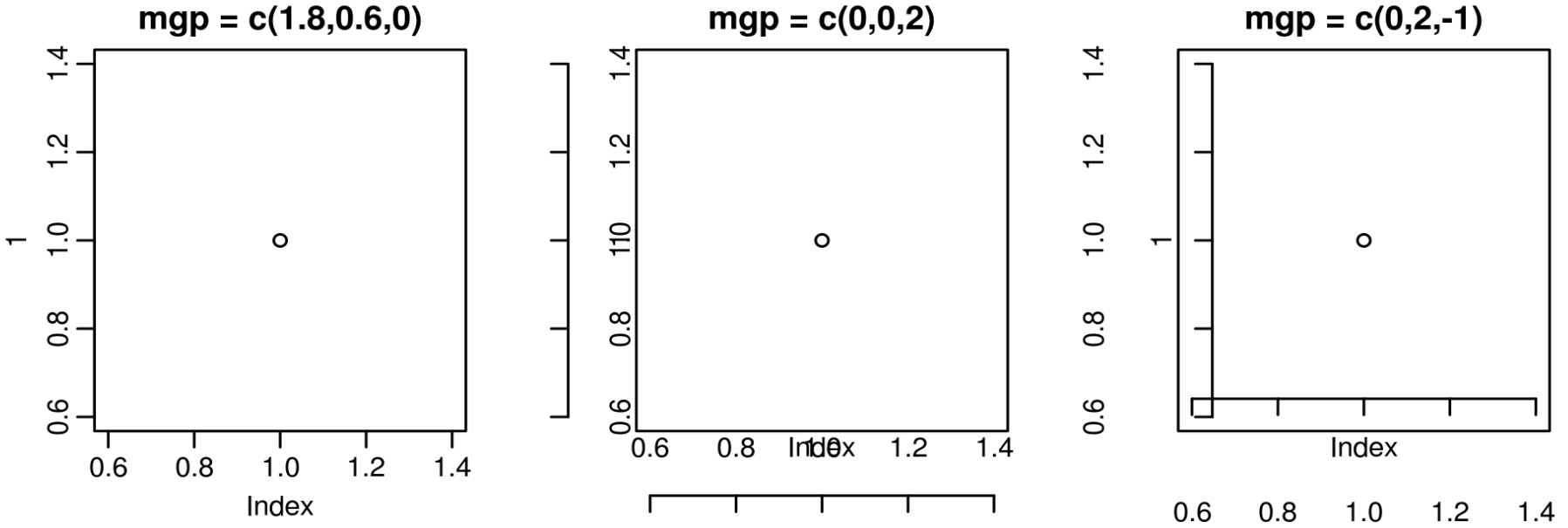
Position des axes et leurs titres et étiquettes

```
par(mgp = c(3, 1, 0)
```

Titre sur la ligne 3
Étiquettes sur la ligne 1
Axe sur la ligne 0

Titre, étiquettes, axe

```
dev.new(width = 16/2.54, height = 6/2.54)
par(mfrow=c(1,3), mar = c(4,4,2,1))
par(mgp = c(1.8,0.6,0))
plot(1, main = "mgp = c(1.8,0.6,0)")
par(mgp = c(0,0,2))
plot(1, main = "mgp = c(0,0,2)")
par(mgp = c(0,2,-1))
plot(1, main = "mgp = c(0,2,-1)")
```

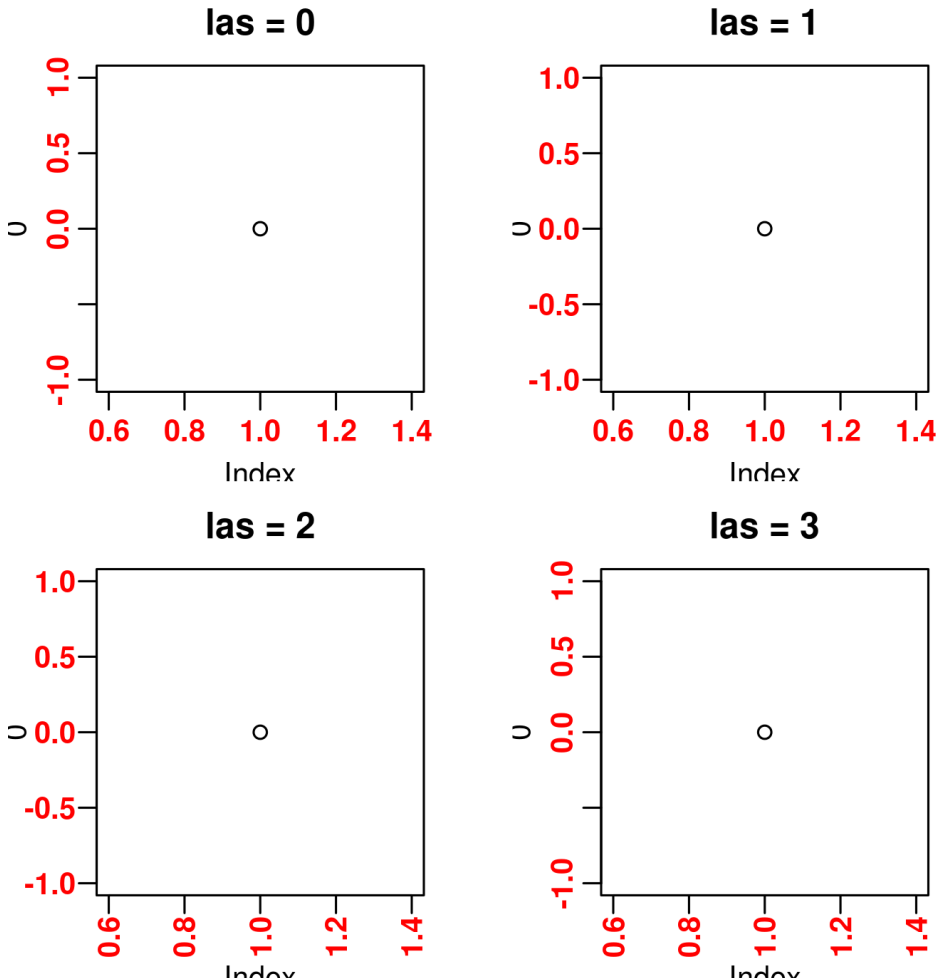


Paramètres graphiques

Orientation des étiquettes des axes

`par(las = 0)`

```
dev.new(width = 12/2.54, height = 12/2.54)
par(mfrow = c(2, 2), mar = c(2.5, 2.5, 2.5, 2.5), mgp = c(1.8,0.6,0))
for(i in 0:3) plot(0, las = i, main = paste("las =", i),
                   font.axis = 2, col.axis = "red")
```

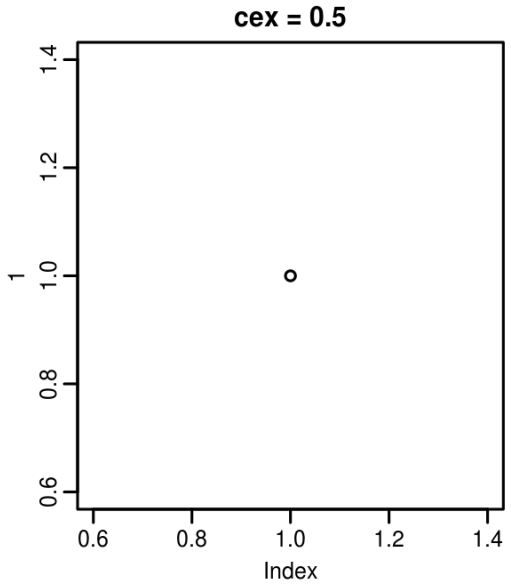


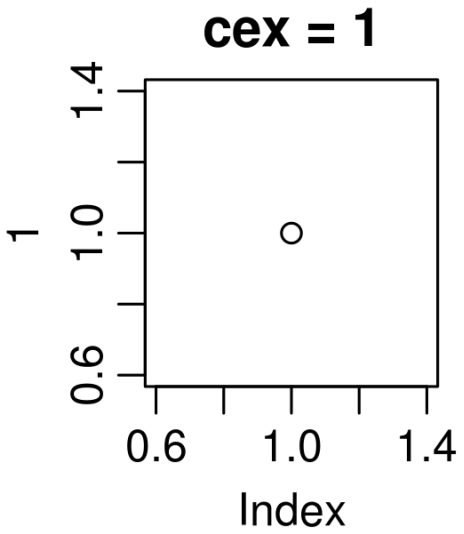
Paramètres graphiques

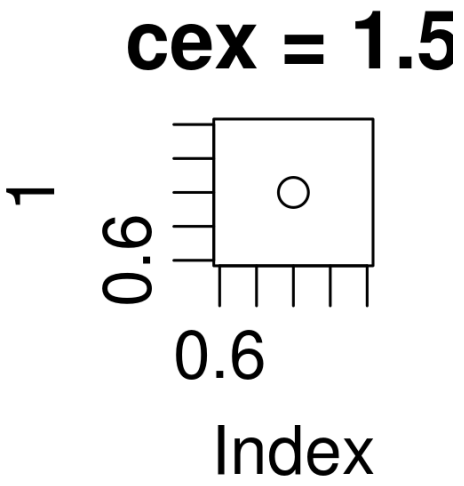
Taille relative du texte et des symboles

`par(cex = 1)`

```
dev.new(width = 16/2.54, height = 6/2.54)
par(mfrow=c(1,3), mar = c(4,4,2,1) , mgp = c(1.8,0.6,0))
par(cex = 0.5)
plot(1, main = "cex = 0.5")
par(cex = 1)
plot(1, main = "cex = 1")
par(cex = 1.5)
plot(1, main = "cex = 1.5")
```







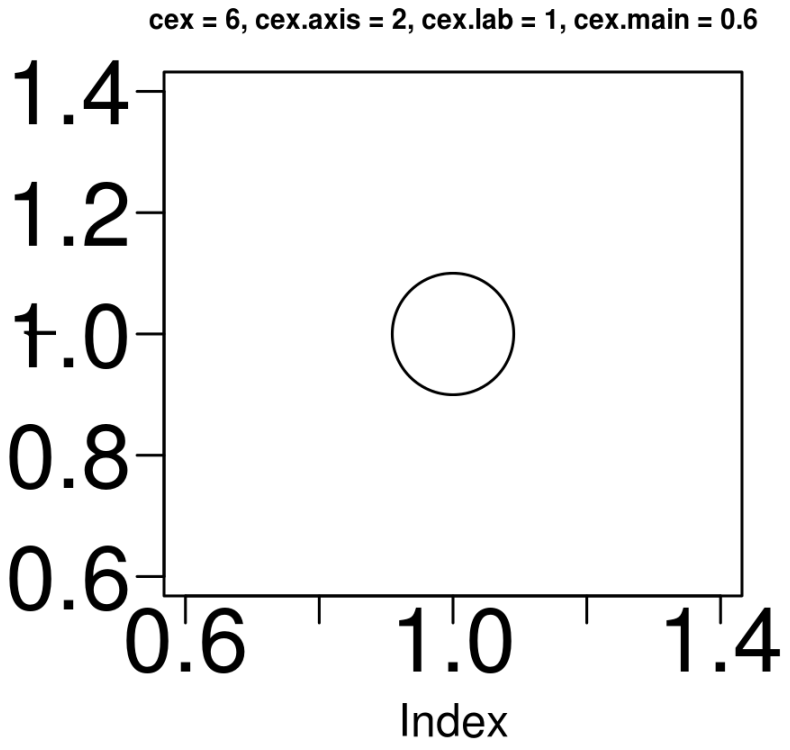
Paramètres graphiques

Taille relative du texte et des symboles

```
par(cex=1, cex.axis=cex, cex.lab=cex, cex.main=cex)
```

symbôles — titre des axes — étiquettes des axes — titre

```
dev.new(width = 8/2.54, height = 8/2.54)
par(mar = c(4,4,2,1) , mgp = c(1.8,0.6,0), las = 1)
plot(1, cex = 6, cex.axis = 2, cex.lab = 1, cex.main = 0.6,
     main = "cex = 6, cex.axis = 2, cex.lab = 1, cex.main = 0.6")
```

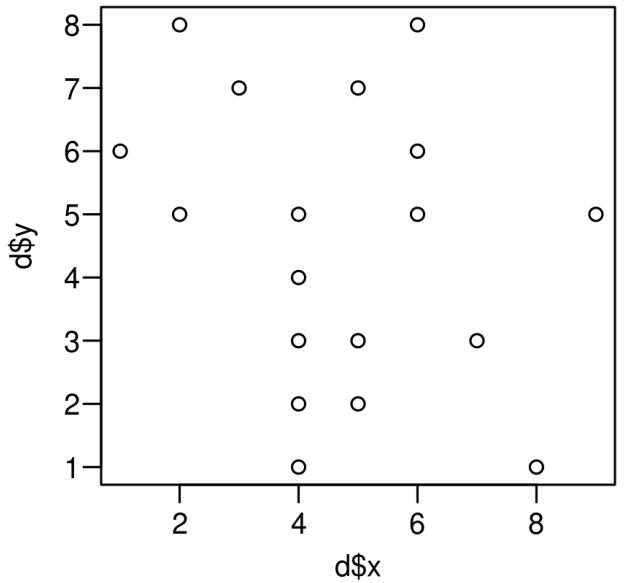


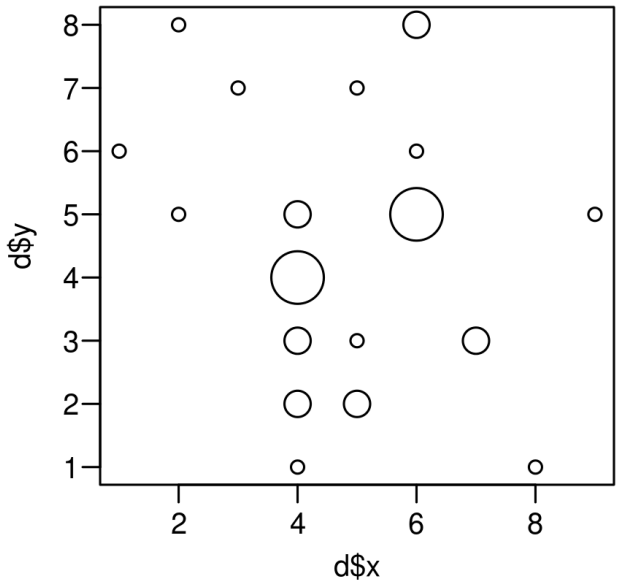
Paramètres graphiques

Taille relative des symboles

Quand `cex` est utilisé comme argument d'une fonction graphique :
peut recevoir un vecteur de valeurs
qui multiplient la valeur de `par("cex")`

```
set.seed(123)
y <- floor(rnorm(30, mean = 10, sd = 2))
x <- floor(rnorm(30, mean = 10, sd = 2))
# compte le nombre d'observations pour chaque combinaison de x et y
d <- as.data.frame(table(y,x))
d <- d[d$Freq>0,]
d[,1:2] <- lapply(d[,1:2], as.numeric)

dev.new(width = 16/2.54, height = 8/2.54)
par(mfrow=c(1,2), mar = c(4,4,2,1) , mgp = c(1.8,0.6,0), las = 1, cex = 0.8)
plot(d$y ~ d$x)
plot(d$y ~ d$x, cex = d$Freq)
```



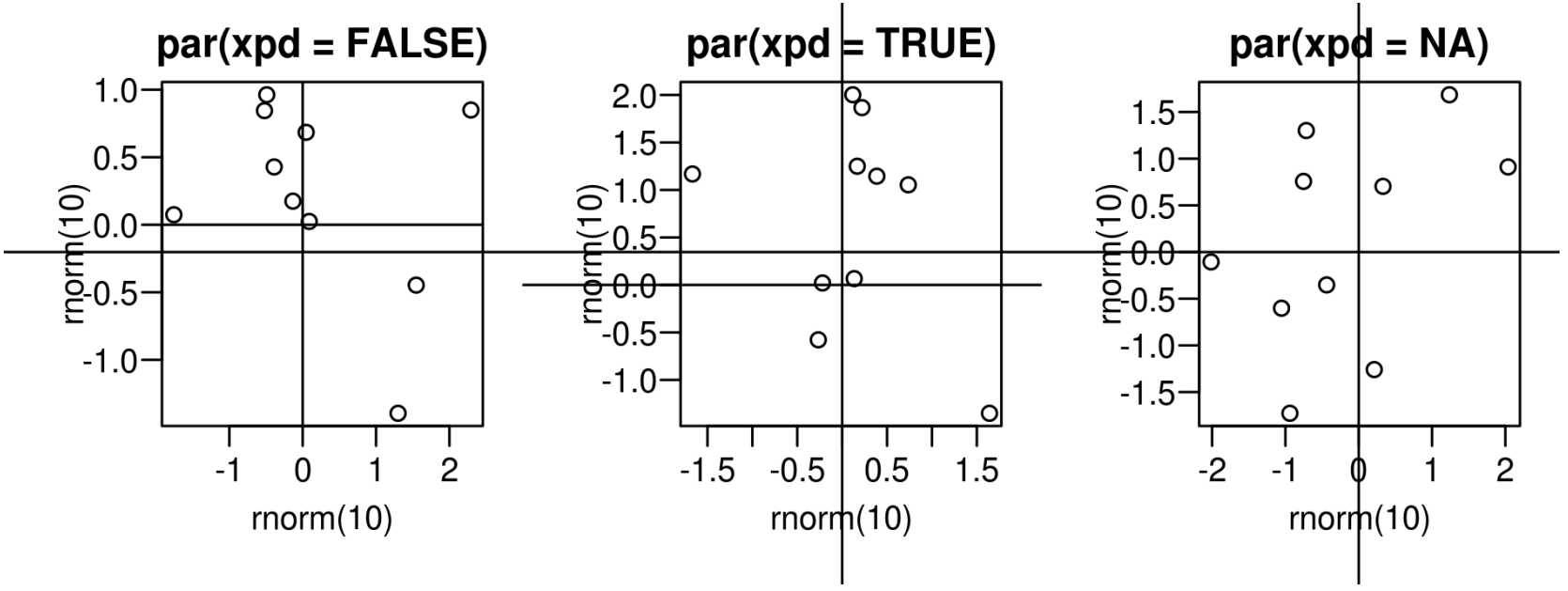


Paramètres graphiques

Dessiner en dehors de la zone de graphique

`par(xpd = FALSE)`

```
dev.new(width = 16/2.54, height = 6/2.54)
par(mfrow=c(1,3), mar = c(4,4,2,1) , mgp = c(1.8,0.6,0), las = 1, cex = 0.8)
par(xpd = FALSE)
plot(y = rnorm(10), x = rnorm(10), main = "par(xpd = FALSE)")
abline(v=0)
abline(h=0)
par(xpd = TRUE)
plot(y = rnorm(10), x = rnorm(10), main = "par(xpd = TRUE)")
abline(v=0)
abline(h=0)
par(xpd = NA)
plot(y = rnorm(10), x = rnorm(10), main = "par(xpd = NA)")
abline(v=0)
abline(h=0)
```

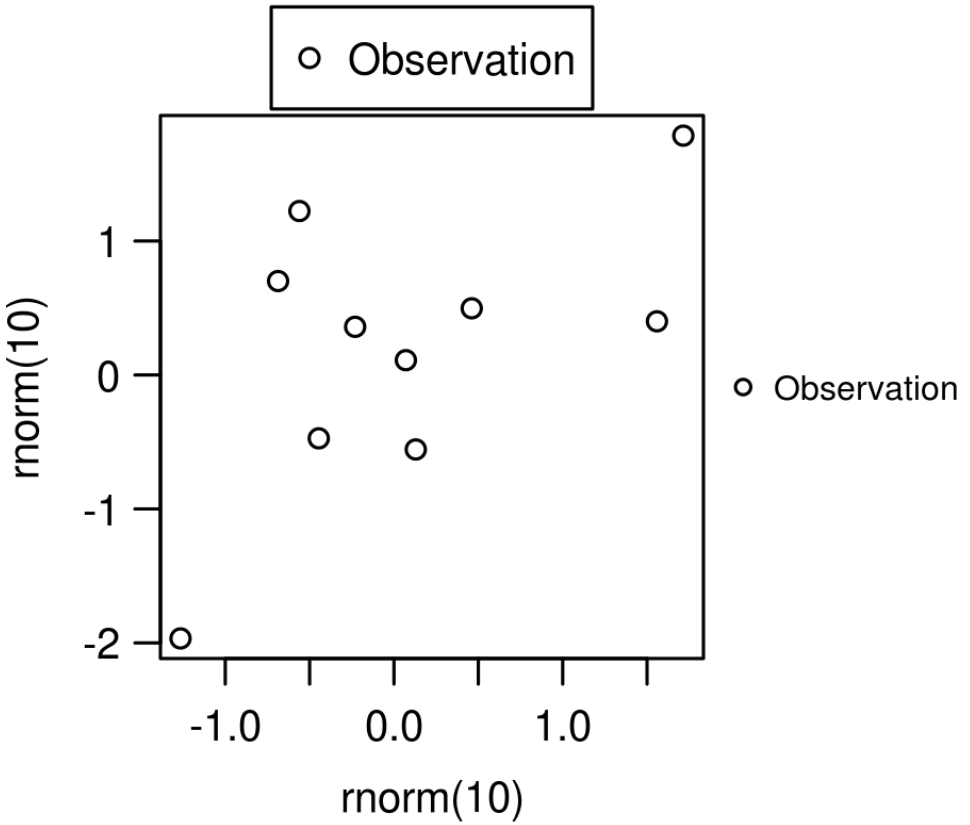


Paramètres graphiques

Dessiner en dehors de la zone de graphique

Utile notamment pour positioner une légende dans les marges

```
dev.new(width = 8/2.54, height = 8/2.54)
par(mar = c(4,4,5,5) , mgp = c(2.2,0.8,0), las = 1, cex = 0.8)
set.seed(123)
plot(y = rnorm(10), x = rnorm(10))
legend(x = "top", legend = c("Observation"), pch = 1, xpd = NA, inset = -0.2)
legend(x = "right", legend = c("Observation"), pch = 1, xpd = NA, inset = -0.5, bty="n",
       cex = 0.8)
```



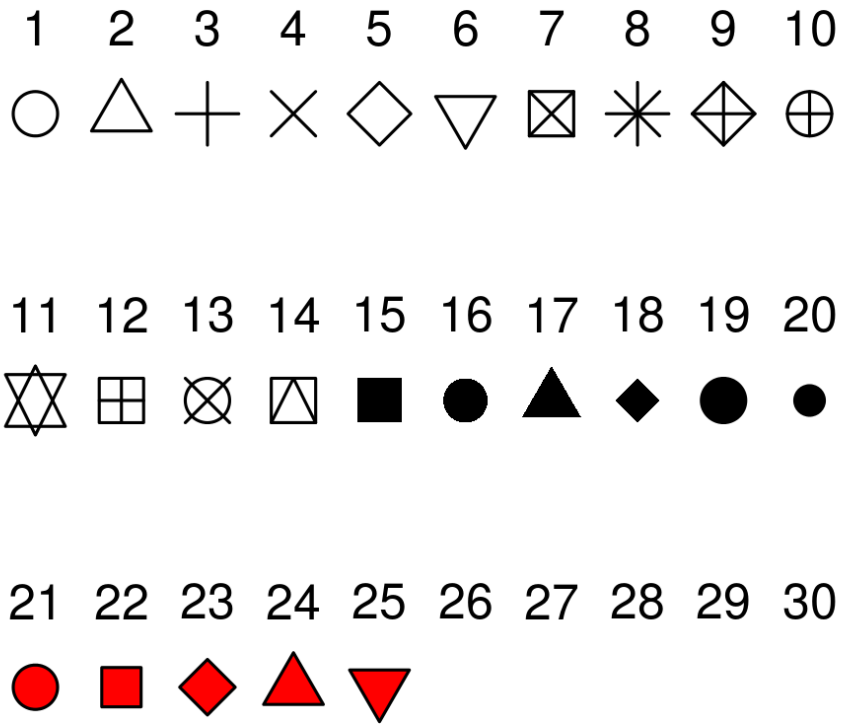
Paramètres graphiques

Type de symboles

`par(pch = 1)`

pch = "Point Character"
?points pour voir la liste

```
dev.new(width = 8/2.54, height = 8/2.54)
par(mar = c(0,0,0,0))
plot(y = rep(3:1, each = 10) , x = rep(1:10,3), pch = 1:30 , cex = 2, axes = FALSE,
     ylim = c(0.5,3.5), xlim = c(0.5, 10.5), xlab = "", ylab = "", bg = "red")
text(y = rep(3:1, each = 10)+0.3 , x = rep(1:10,3), labels = 1:30)
```

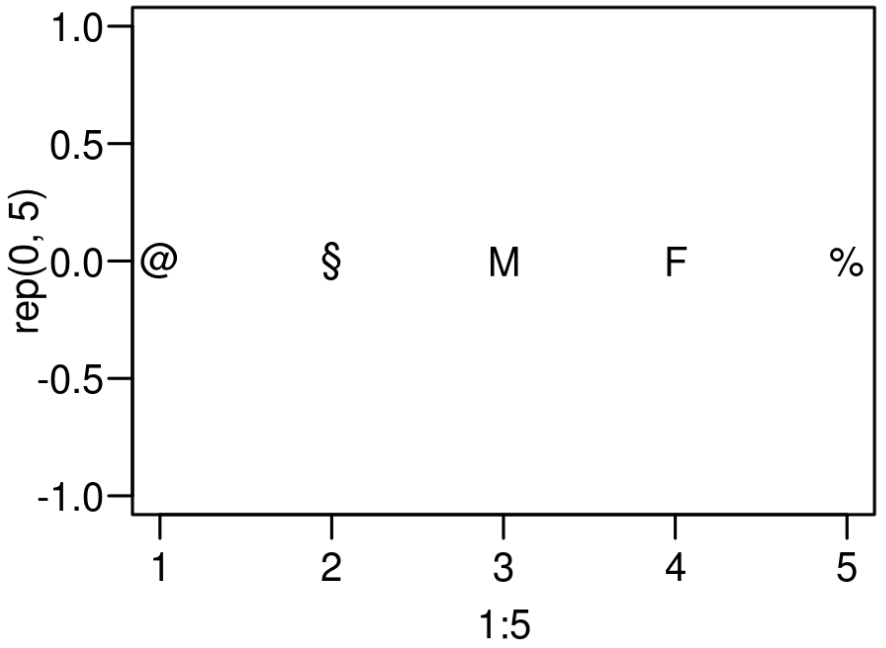


Paramètres graphiques

Type de symboles

On peut aussi donner n'importe quel caractère à pch

```
dev.new(width = 8/2.54, height = 6/2.54)
par(mar = c(3,3,1,1) , mgp = c(1.8,0.6,0), las = 1, cex = 0.8)
plot(x = 1:5, y = rep(0,5), pch = c("@", "§", "M", "F", "%"))
```

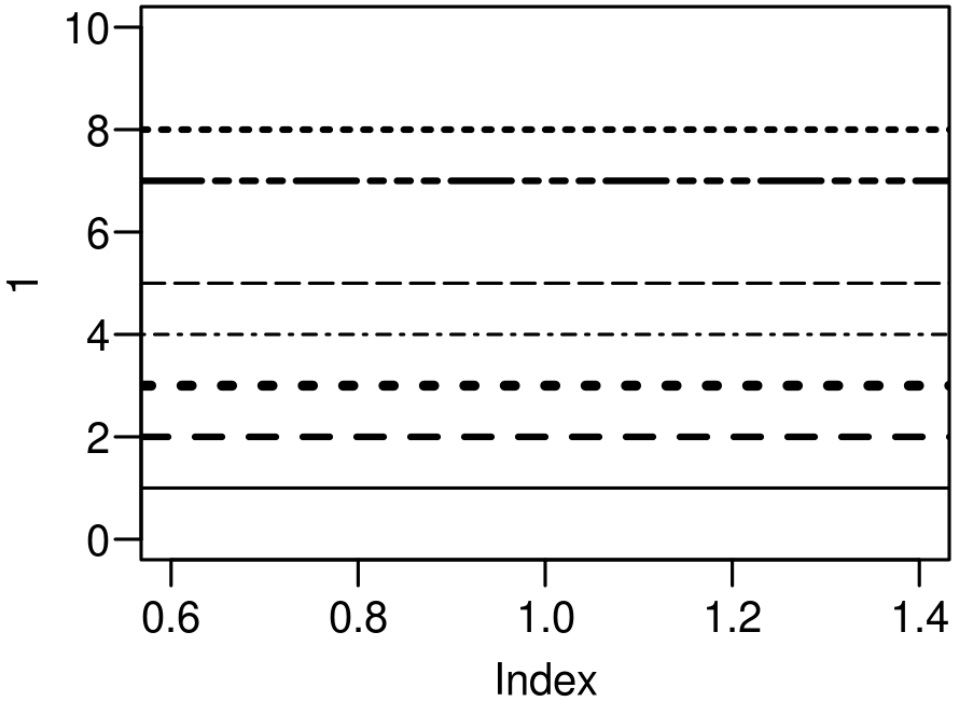


Paramètres graphiques

Type de lignes et largeur de lignes

`par(lty = 1, lwd = 1)`

```
dev.new(width = 8/2.54, height = 6/2.54)
par(mar = c(3,3,1,1) , mgp = c(1.8,0.6,0), las = 1, cex = 0.8)
plot(1, type = "n", ylim = c(0,10))
abline(h=1, lty = 1, lwd = 1)
abline(h=2, lty = 2, lwd = 2)
abline(h=3, lty = 3, lwd = 3)
abline(h=4, lty = 4, lwd = 1)
abline(h=5, lty = 5, lwd = 1)
abline(h=7, lty = "92222222", lwd = 2)
abline(h=8, lty = "12", lwd = 2)
```



?par :
Line types can either be specified by giving an index into a small built-in table of line types (1 = solid, 2 = dashed, etc, see lty above) or directly as the lengths of on/off stretches of line. This is done with a string of an even number (up to eight) of characters, namely non-zero (hexadecimal) digits which give the lengths in consecutive positions in the string. For example, the string "33" specifies three units on followed by three off and "3313" specifies three units on followed by three off followed by one on and finally three off. The 'units' here are (on most devices) proportional to lwd, and with lwd = 1 are in pixels or points or 1/96 inch.

The five standard dash-dot line types (lty = 2:6) correspond to c("44", "13", "1343", "73", "2262").

Paramètres graphiques

Couleurs

```
> par("col")
[1] "black"
```

Généralement utilisé comme argument d'une fonction
(accepte alors un vecteur qui peut être recyclé)

Les couleurs peuvent être décrites par :

des nombres entre 1 et 8
(= couleurs extraites de palette())

une chaine de texte
pex : "green", "darkgoldenrod1", "gray34"
`colors()` donne la liste
research.stowers-institute.org/efg/R/Color/Chart/ColorChart.pdf

le code Hexadécimal de la couleur
"#00FF80", "#00FFFF"
souvent généré par d'autres fonctions comme `rgb()`

Voir par exemple : https://color.adobe.com pour choisir visuellement la couleur

Paramètres graphiques

Couleurs

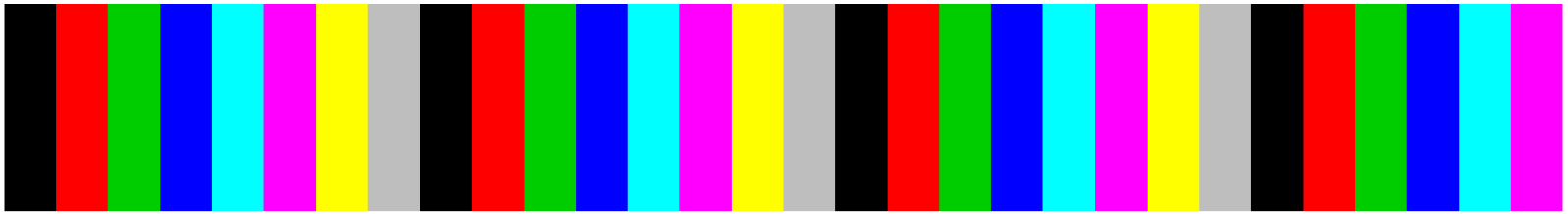
```
> colors()[30:35]
[1] "blue4"      "blueviolet" "brown"      "brown1"     "brown2"     "brown3"
> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"  "gray"
> col2rgb("green")
red      0
green  255
blue     0
> rgb(1,0.5, 0.2, alpha = 0.6)
[1] "#FF803399"
> rgb(c(1, 0, 0.5),c(0.5, 1, 0) , c(0.2, 0.8, 0.5), alpha = 0.6)
[1] "#FF803399" "#00FFCC99" "#80008099"
> m <- matrix(runif(9, 0, 1), ncol = 3)
> m
          [,1]      [,2]      [,3]
[1,] 0.1479872 0.2041886 0.8130490
[2,] 0.4922076 0.2706627 0.3918320
[3,] 0.7769890 0.2154635 0.6248218
> rgb(m)
[1] "#2634CF" "#7E4564" "#C6379F"
> hsv(0.5,0.5,0.5, alpha = 0.2)
[1] "#40808033"
> rainbow(4)
[1] "#FF0000FF" "#80FF00FF" "#00FFFFFF" "#8000FFFF"
> heat.colors(4)
[1] "#FF0000FF" "#FF8000FF" "#FFFF00FF" "#FFFF80FF"
> gray(1:4/10)
[1] "#1A1A1A" "#333333" "#4D4D4D" "#666666"
> colorRampPalette(c("blue", "red"))(4)
[1] "#0000FF" "#5500AA" "#AA0055" "#FF0000"
```

Paramètres graphiques

Couleurs

```
dev.new(width = 16/2.54, height = 2/2.54)
par(mar = c(0,0,0,0), fg = NA)
barplot(rep(10,30), col = 1:30, axes = TRUE, space = 0)
```

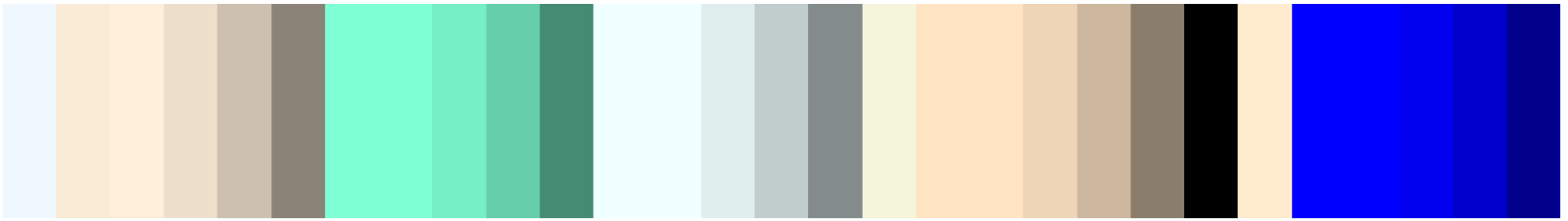
```
> palette()[1:30]
 [1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"  "gray"
 [9] NA        NA        NA        NA        NA        NA        NA        NA
[17] NA        NA        NA        NA        NA        NA        NA        NA
[25] NA        NA        NA        NA        NA        NA
```



Paramètres graphiques

Couleurs

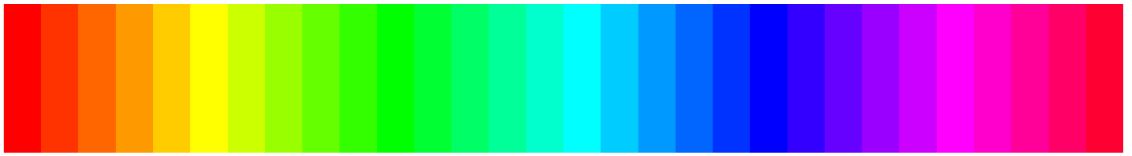
```
dev.new(width = 16/2.54, height = 2/2.54)
par(mar = c(0,0,0,0), fg = NA)
barplot(rep(10,30), col = colors()[1:30], axes = TRUE, space = 0)
> colors()[1:30]
 [1] "white"           "aliceblue"      "antiquewhite"   "antiquewhite1"
 [5] "antiquewhite2"   "antiquewhite3"  "antiquewhite4"  "aquamarine"
 [9] "aquamarine1"     "aquamarine2"    "aquamarine3"    "aquamarine4"
[13] "azure"           "azure1"         "azure2"         "azure3"
[17] "azure4"          "beige"          "bisque"         "bisque1"
[21] "bisque2"         "bisque3"        "bisque4"        "black"
[25] "blanchedalmond" "blue"           "blue1"          "blue2"
[29] "blue3"           "blue4"
```



Paramètres graphiques

Couleurs

```
dev.new(width = 16/2.54, height = 2/2.54)
par(mar = c(0,0,0,0), fg = NA)
barplot(rep(10,30), col = rainbow(30), axes = TRUE, space = 0)
```



```
dev.new(width = 16/2.54, height = 2/2.54)
par(mar = c(0,0,0,0), fg = NA)
barplot(rep(10,30), col = heat.colors(30), axes = TRUE, space = 0)
```

```
dev.new(width = 16/2.54, height = 2/2.54)
par(mar = c(0,0,0,0), fg = NA)
barplot(rep(10,30), col = gray(0:30/30), axes = TRUE, space = 0)
```

Paramètres graphiques

Couleurs

`colorRampPalette()` : crée une fonction qui peut générer n'importe quelle palette à partir des couleurs fournie**s**

```
dev.new(width = 16/2.54, height = 2/2.54)
par(mar = c(0,0,0,0), fg = NA)
barplot(rep(10,30), col = colorRampPalette(c("blue", "yellow", "red"))(30)
        , axes = TRUE, space = 0)
```

Paramètres graphiques

Couleurs

On peut créer une fonction qui génère une séquence de couleurs pastel les plus différentes possibles
(teintes équidistantes sur une "roue de couleurs")

--> couleurs similaires à ggplot

```
mypalette <- function(n, alpha = 1, l=65, c=100) {
    hues = seq(15, 375, length=n+1)
    hcl(h=hues, l=l, c=c, alpha = alpha)[1:n]
}

dev.new(width = 16/2.54, height = 2/2.54)
par(mar = c(0,0,0,0), fg = NA)
barplot(rep(10,10), col = mypalette(10)
        , axes = TRUE, space = 0)
```

Paramètres graphiques

Couleurs

Attribuer des couleurs selon les niveaux d'un facteur

Exemple : une couleur pour chaque espèce d'iris

```
> data(iris)
> levels(iris$Species)
[1] "setosa"     "versicolor" "virginica"
> as.numeric(iris$Species)
  [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 [42] 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 [83] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
[124] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
> c("red", "green", "blue")[as.numeric(iris$Species)]
  [1] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
 [11] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
 [21] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
 [31] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
 [41] "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"   "red"
 [51] "green" "green" "green" "green" "green" "green" "green" "green" "green" "green"
 [61] "green" "green" "green" "green" "green" "green" "green" "green" "green" "green"
 [71] "green" "green" "green" "green" "green" "green" "green" "green" "green" "green"
 [81] "green" "green" "green" "green" "green" "green" "green" "green" "green" "green"
 [91] "green" "green" "green" "green" "green" "green" "green" "green" "green" "green"
[101] "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"
[111] "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"
[121] "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"
[131] "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"
[141] "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"  "blue"
```

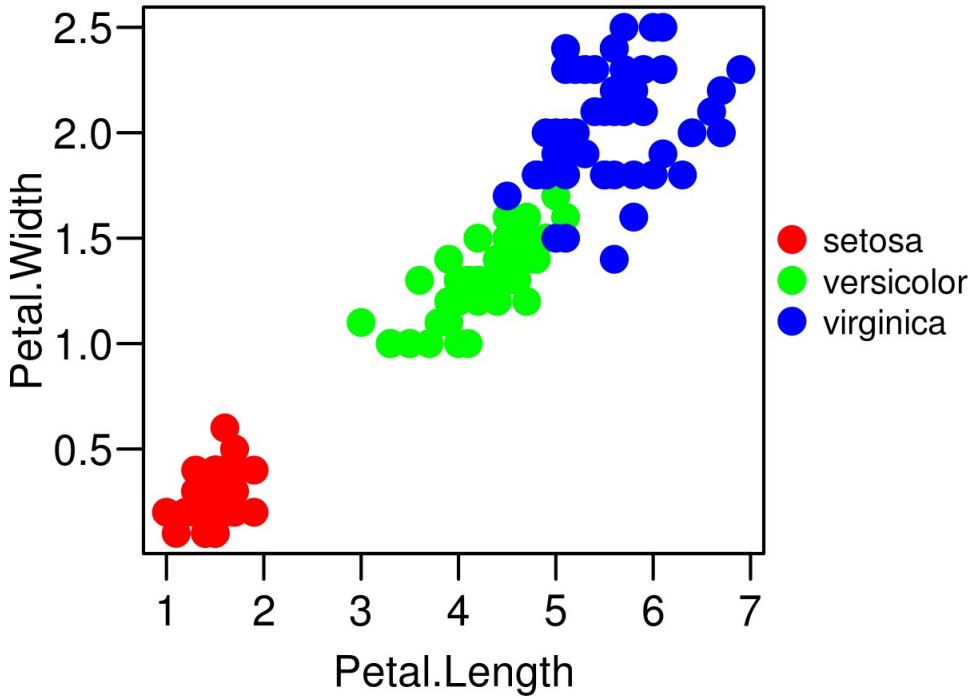
Paramètres graphiques

Couleurs

Attribuer des couleurs selon les niveaux d'un facteur

Exemple : une couleur pour chaque espèce d'iris

```
dev.new(width = 8/2.54, height = 6/2.54)
par(mar = c(3,3,1,4.5) , mgp = c(1.8,0.6,0), las = 1, cex = 0.8)
plot(Petal.Width ~ Petal.Length, data = iris, cex = 2, pch = 20,
     col = c("red", "green", "blue")[as.numeric(iris$Species)])
legend("right", pch = 20, pt.cex = 2, cex = 0.8,
       legend = levels(iris$Species),
       col = c("red", "green", "blue"),
       xpd = NA, inset = -0.35, bty = "n" )
```



Paramètres graphiques

Couleurs - Transparence

Très utile en cas d' "overplotting"

Plusieurs fonctions ont un argument alpha pour spécifier la transparence de la couleur

```
rgb(red, green, blue, alpha, names = NULL, maxColorValue = 1)
hcl(h = 0, c = 35, l = 85, alpha, fixup = TRUE)
```

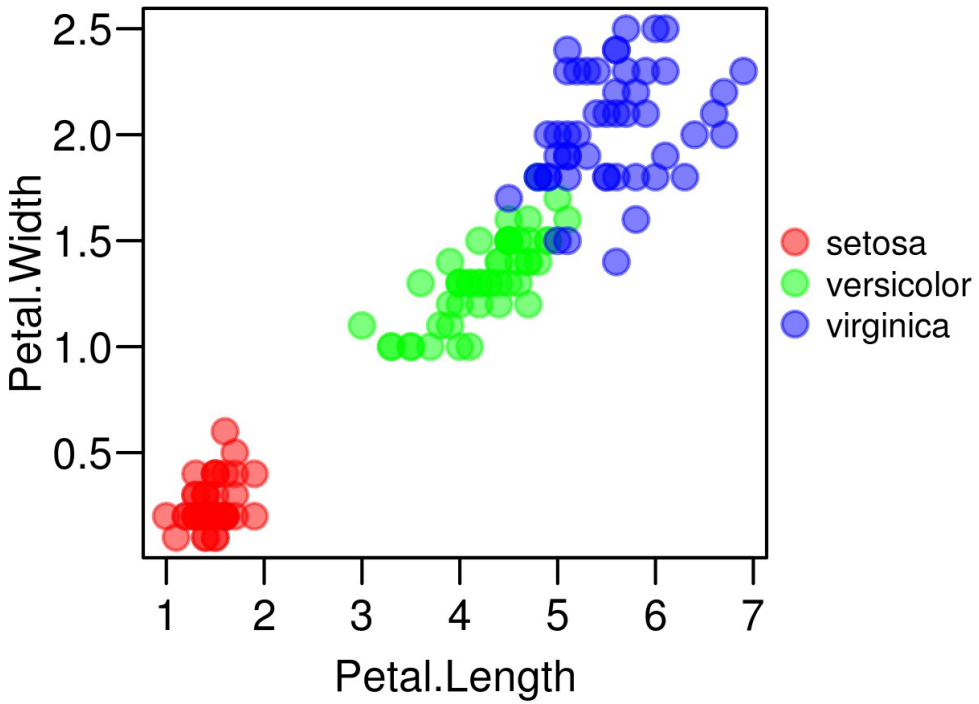
Il est souvent plus pratique de modifier une palette existante avec la fonction `adjustcolor`

```
> adjustcolor( c("red", "green", "blue"), alpha.f = 0.2)
[1] "#FF000033" "#00FF0033" "#0000FF33"
```

Paramètres graphiques

Couleurs - Transparence

```
mycols <- adjustcolor( c("red", "green", "blue"), 0.5)

dev.new(width = 8/2.54, height = 6/2.54)
par(mar = c(3,3,1,4.5) , mgp = c(1.8,0.6,0), las = 1, cex = 0.8)
plot(Petal.Width ~ Petal.Length, data = iris, cex = 2, pch = 20,
     col = mycols[as.numeric(iris$Species)])
legend("right", pch = 20, pt.cex = 2,, cex = 0.8,
       legend = levels(iris$Species),
       col = mycols,
       xpd = NA, inset = -0.35, bty = "n" )
```



Paramètres graphiques

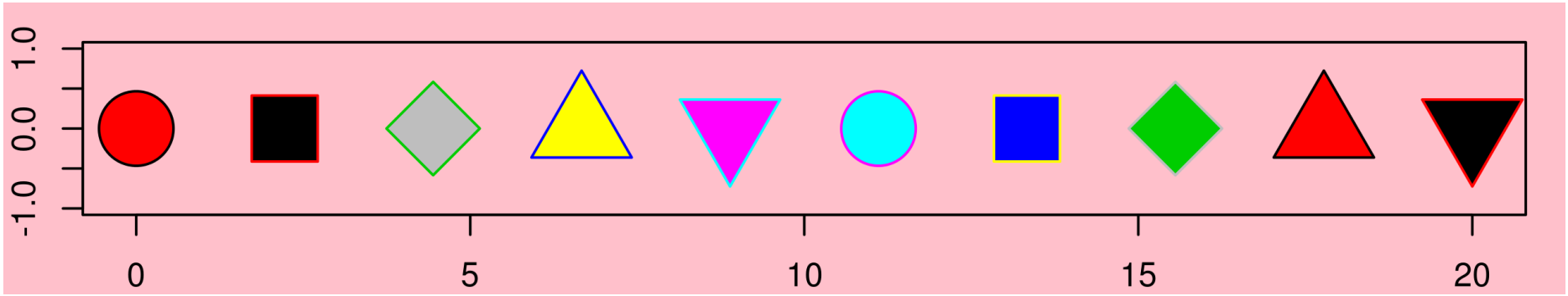
Couleurs - background

`par(bg = "white")`

Permet de définir la couleur de fond du graphique
Quand il est utilisé dans une fonction graphique peut recevoir un vecteur de couleurs

pex pour définir la couleur de la bordure des symboles 21 à 25

```
dev.new(width = 16/2.54, height = 3/2.54)
par(mar = c(2,2,1,1), bg = "pink", cex = 0.8)
plot(y = rep(0,10), x = seq(0, 20, length.out = 10),
     pch = 21:25, cex = 5,
     col = 1:10, bg = 10:1)
```



Paramètres graphiques

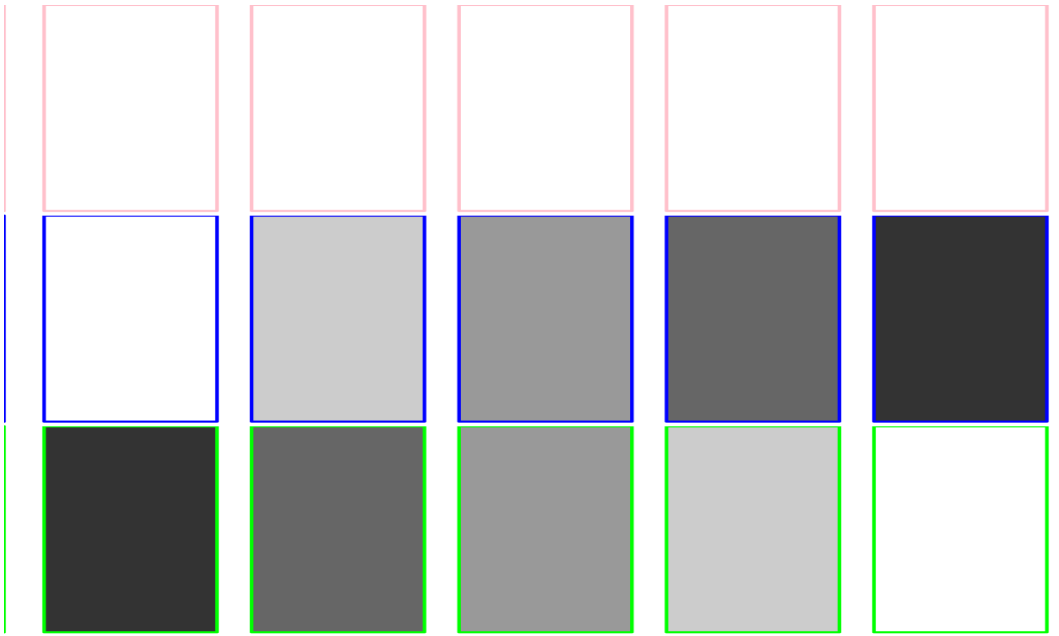
Couleurs - foreground

`par(fg = "black")`

Défini la couleur de certaines parties des graphiques

pex la bordure des barplots (`fg = NA` : pas de bordure)

```
dev.new(width = 8/2.54, height = 8/2.54)
par(mfrow = c(3, 1), mar = c(0.1,0,0,0), fg = "pink")
barplot(rep(10,5), col = "white", axes = TRUE, space = 0.2)
par(fg = "blue")
barplot(rep(10,5), col =  paste("gray", seq(100, 20, -20)), axes = TRUE, space = 0.2)
par(fg = "green")
barplot(rep(10,5), col = paste("gray", seq(20, 100, 20)), axes = TRUE, space = 0.2)
```



ggplot

Un autres système graphique : ggplot2

Avantages :

Syntaxe très structurée

Faceting

(= subdivision du graphique en fonction d'une ou plusieurs variables)

Gère automatiquement marges, légendes, etc...

Rendu par défaut plus soigné/plus joli

Possibilité d'utiliser des thèmes

On peut donc faire facilement et rapidement des graphiques jolis et/ou complexes

Désavantages :

Syntaxe très structurée : plus difficile à apprendre

Lent (problématique avec gros jeux de données)

Ne travaille que sur des data.frame

(les objets cartes doivent être convertis en data.frame pex)

NB : lattice permet à peu près la même chose.
Il est plus rapide mais la syntaxe est moins structurée, le rendu par défaut parfois moins joli et permet de faire moins de choses automatiquement (ajustement pex).

ggplot

ggplot = layered "grammar of graphics plot"

Selon cette version de la grammaire des graphiques tout graphique est composés des mêmes éléments

coord :
1 seul système de coordonnées (souvent cartésien)
permet de positionner les données dans l'espace

scale :
1 ou plusieurs échelles continues ou discrètes
permettent de transformer les données numériques en caractéristiques esthétiques (aesthetics) des objets : position horizontale, position verticale, couleur, taille, forme,...

facet :
facultativement on peut subdiviser les données et les graphiques selon les niveaux de 1 à 2 variables qualitatives

layer :
1 ou plusieurs "couches"

ggplot

Toute "couche", "layer" est composée des mêmes éléments :

`layer(data, mapping, geom, stat, position)`

data

1 seul data.frame

On peut avoir des data.frame différents d'un layer à l'autre pour un même graphique

geom

1 seul type d'objet géométrique dessiné sur le graphique pour représenter les données

points, lignes, barres, boxplots, ...

mapping

"mapping variables to aesthetics"

Indique quelles variables du dataset correspondent à quelles caractéristiques esthétiques ("aesthetics") des objets dessinés ("geoms") sur le graphique.

pex : la forme des points correspond à la variable "sexe", leur couleur correspond à la variable "site" et leur taille correspond au nombre d'échantillons par site

"set value to aesthetics"

On peut aussi fixer une valeur pour les propriétés esthétiques des geoms sans les lier à une variable

pex : les points seront rouges, de taille 2 et de forme triangulaire

ggplot

Toute "couche", "layer" est composée des mêmes éléments :

`layer(data, mapping, geom, stat, position)`

stat

Une transformation statistique des données

Pour un geom = "bar", on aura :
un barplot avec stat = "identity" : aucune modification des données
un histogramme avec stat = "bin" : variable continue transformée en catégories

stat = "smooth", method = "lm" : courbe de tendance (ici régression linéaire)
stat = "summary", fun.y = mean : représente la moyenne de la variable

position

Facultativement, on peut ajuster la position des objets pour qu'ils ne se recouvrent pas

Pour un nuage de points où les points se recouvrent :
position = position_jitter() : on ajoute un peu de bruit aléatoire

Pour des barplots :
position = position_stack() : on empile les barres
position = position_dodge() :on place les barres les unes à côté des autres

ggplot

ggplot a aussi un système de "theme"

qui permet de définir les caractéristiques de ce qu'on appelle les "décorations" du graphique :
taille, position, couleur du texte, du fond, des "guides" ("ticks" des axes, grille,...), des bordures, des légendes,...

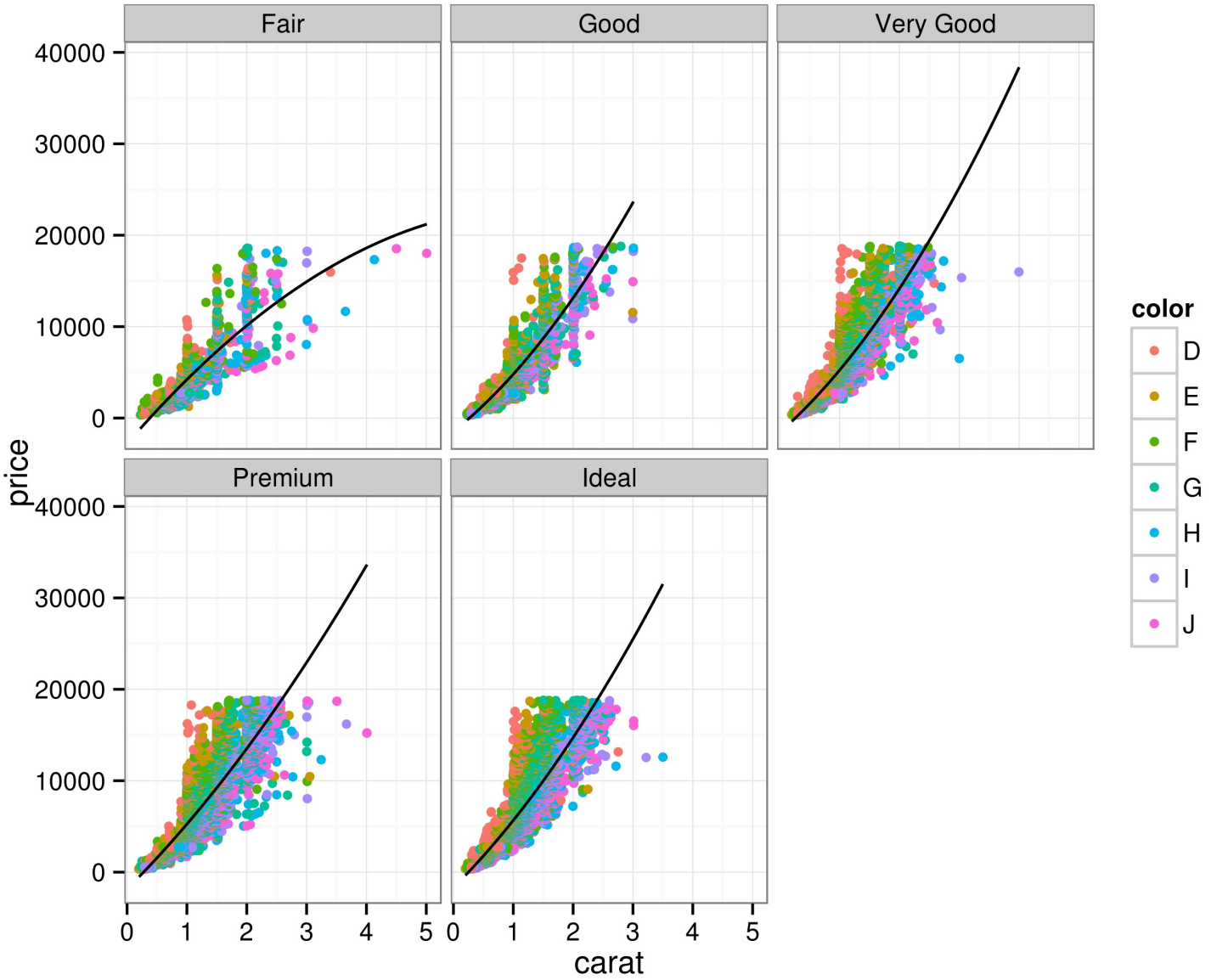
ggplot

Jeu de données fourni avec ggplot : diamonds :

```
> library(ggplot2)
> data(diamonds)
> d <- diamonds
>
> summary(d)
     carat               cut        color        clarity          depth           table
 Min.   :0.2000   Fair      : 1610   D: 6775   SI1     :13065   Min.   :43.00   Min.   :43.00
 1st Qu.:0.4000   Good      : 4906   E: 9797   VS2     :12258   1st Qu.:61.00   1st Qu.:56.00
 Median :0.7000   Very Good:12082   F: 9542   SI2     : 9194   Median :61.80   Median :57.00
 Mean   :0.7979   Premium   :13791   G:11292   VS1     : 8171   Mean   :61.75   Mean   :57.46
 3rd Qu.:1.0400   Ideal     :21551   H: 8304   VVS2    : 5066   3rd Qu.:62.50   3rd Qu.:59.00
 Max.   :5.0100                     I: 5422   VVS1    : 3655   Max.   :79.00   Max.   :95.00
                                    J: 2808   (Other): 2531
     price             x                y                z
 Min.   :  326   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
 1st Qu.:  950   1st Qu.: 4.710   1st Qu.: 4.720   1st Qu.: 2.910
 Median : 2401   Median : 5.700   Median : 5.710   Median : 3.530
 Mean   : 3933   Mean   : 5.731   Mean   : 5.735   Mean   : 3.539
 3rd Qu.: 5324   3rd Qu.: 6.540   3rd Qu.: 6.540   3rd Qu.: 4.040
              Max.   :18823   Max.   :10.740   Max.   :58.900   Max.   :31.800
```

ggplot

Un graphique avec tous ces éléments :



ggplot

Un graphique avec tous ces éléments :

coordonées cartésiennes

3 échelles

facettes définies selon la variable "cut"

layer "nuage de points"

layer "courbe de tendance"

theme "noir et blanc"

```
ggplot() +
    coord_cartesian() +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_color_hue() +
    facet_wrap(~cut) +
    layer(
        data=diamonds,
        mapping=aes(x=carat, y=price, color=color),
        stat="identity",
        stat_params=list(),
        geom="point",
        geom_params=list(shape = 20),
        position=position_jitter()
    ) +
    layer(
        data=diamonds,
        mapping=aes(x=carat,y=price),
        stat="smooth",
        stat_params=list(method="glm", formula=y~poly(x,2)),
        geom="line",
        geom_params=list(color="black"),
        position=position_identity()
    ) +
    theme_bw()
```

"aesthetic mappings" :
la position des points va varier selon les variables carat et price et la couleur des points va varier selon la variable "color"

"aesthetic settings" :
la forme des points est fixée à un petit cercle (cfr pch =20)

on ajoute du bruit en x et y pour limiter le recouvrement des points

on transforme les variable carat et price avec un modèle linéaire (glm) polynomial d'ordre 2

on représente le modèle estimé par stat par une ligne

layer "nuage de points" :

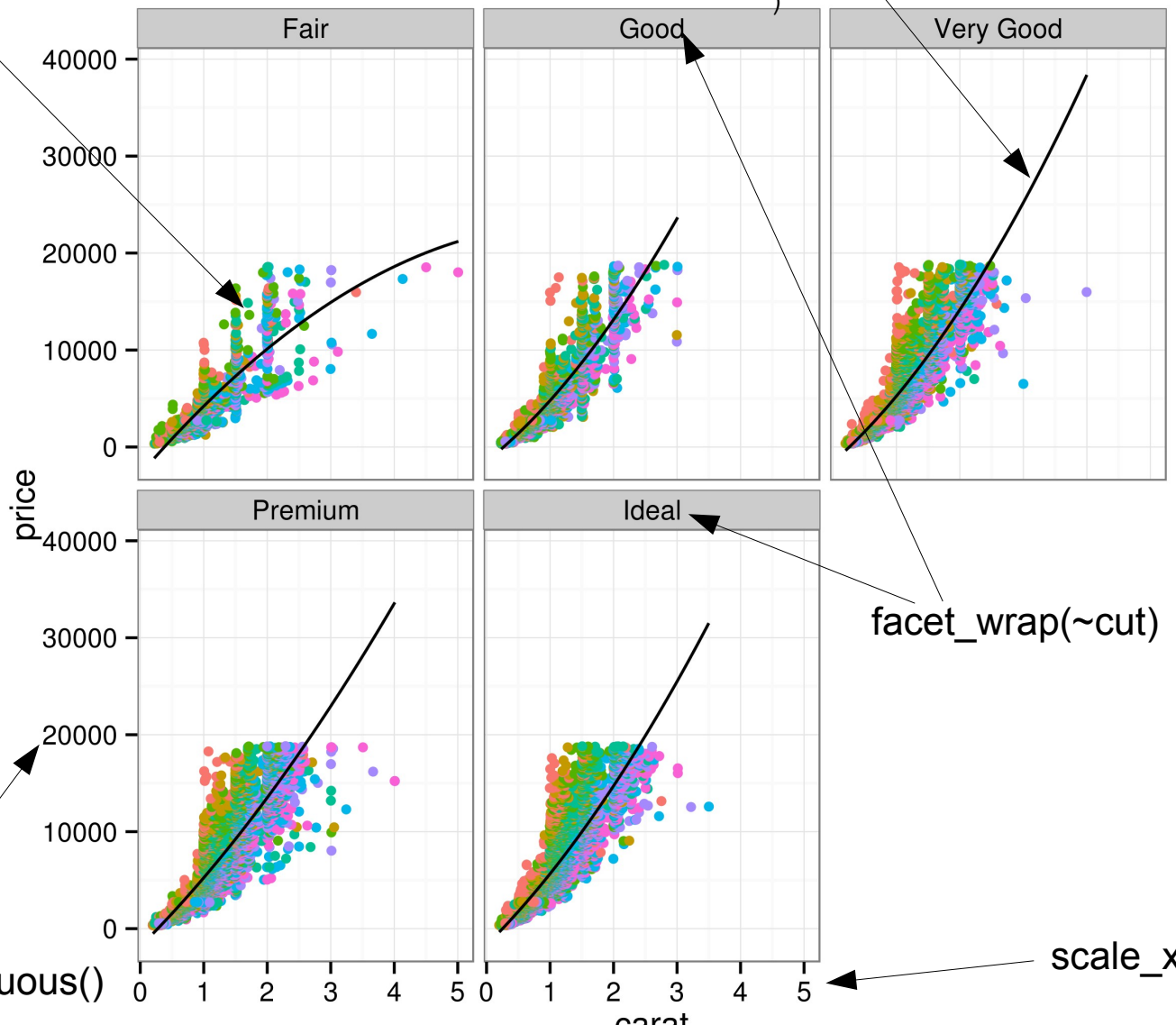
```
layer(
    data=diamonds,
    mapping=aes(x=carat,
        y=price, color=color),
    stat="identity",
    stat_params=list(),
    geom="point",
    geom_params=list(shape = 20),
    position=position_jitter()
    )
```

"aesthetic mappings"

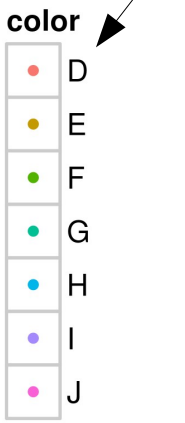
"aesthetic settings"

layer "courbe de tendance" :

```
layer(
    data=diamonds,
    mapping=aes(x=carat,y=price),
    stat="smooth",
    stat_params=list(method="glm",
                     formula=y~poly(x,2)),
    geom="line",
    geom_params=list(color="black"),
    position=position_identity()
    )
```



scale_color_hue()



facet_wrap(~cut)

scale_y_continuous()

scale_x_continuous()

ggplot

Syntaxe de base très verbeuse : On peut la simplifier considérablement

Les valeurs par défaut de nombreux éléments conviennent
--> on ne doit pas les spécifier (coord, scales,...)

On utilise rarement `layer()`, on utilise en général des fonctions avec des arguments typiques définis par défaut :
`geom_XXX()` ou `stat_XXX()`
où XXX correspond à une géométrie ou une transformation statistique

On spécifier un dataset et des mappings par défaut dans la fonction `ggplot()` qui seront utilisés par tous les layers si ils ne sont pas spécifiés

ggplot

Syntaxe complète

```
ggplot() +
    coord_cartesian() +
    scale_x_continuous() +
    scale_y_continuous() +
    scale_color_hue() +
    facet_wrap(~cut) +
    layer(
        data=diamonds,
        mapping=aes(x=carat, y=price,
                            color=color),
        stat="identity",
        stat_params=list(),
        geom="point",
        geom_params=list(shape = 20),
        position=position_jitter()
    ) +
    layer(
        data=diamonds,
        mapping=aes(x=carat,y=price),
        stat="smooth",
        stat_params=list(method="glm",
                    formula=y~poly(x,2)),
        geom="line",
        geom_params=list(color="black"),
        position=position_identity()
    ) +
    theme_bw()
```

Syntaxe simplifiée

"aesthetic mappings" par défaut pour tous les layers

```
ggplot(d, aes(x=carat, y=price)) +
    geom_point(aes(color = color),
            shape = 20,
            position = position_jitter()) +
    stat_smooth(method = "glm",
            formula=y~poly(x,2),
            geom = "line", color = "black") +
    facet_wrap(~cut) +
    theme_bw()
```

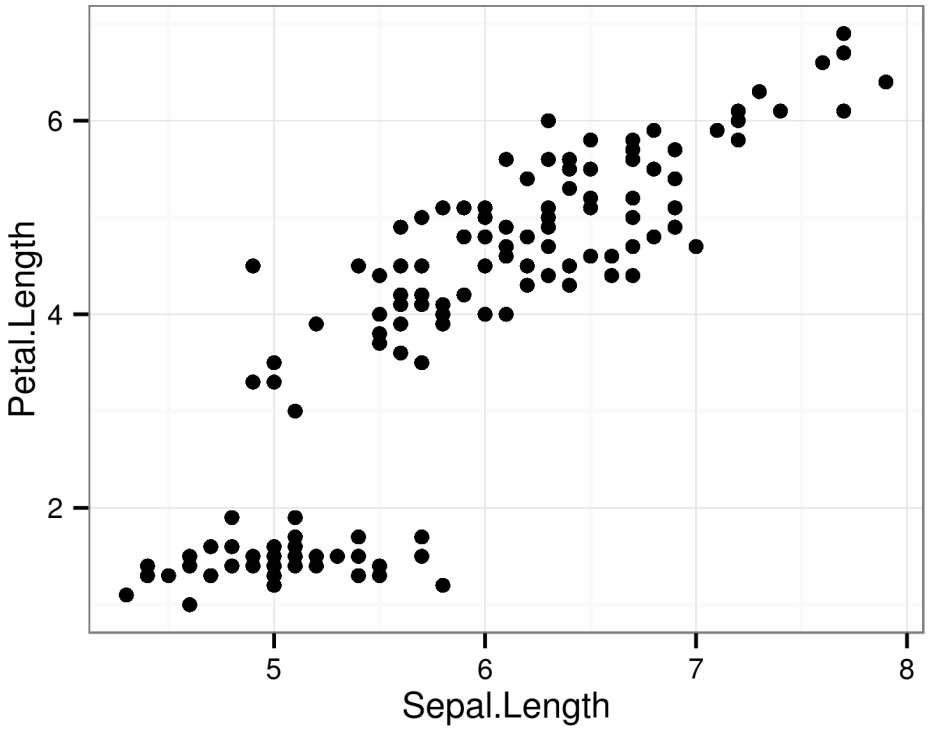
ggplot - exemples

```
data(iris)
d <- iris

mytheme <- theme_bw(10) + theme(axis.text.x=element_text(size=8),
                                legend.key = element_rect(color = NA))
theme_set(mytheme)

dev.new(10/2.54, 8/2.54)

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

On défini un nouveau thème (police de taille 10, taille 8 pour l'axe des x etc...

ce thème sera utilisé pendant la session

deux syntaxes strictement équivalentes

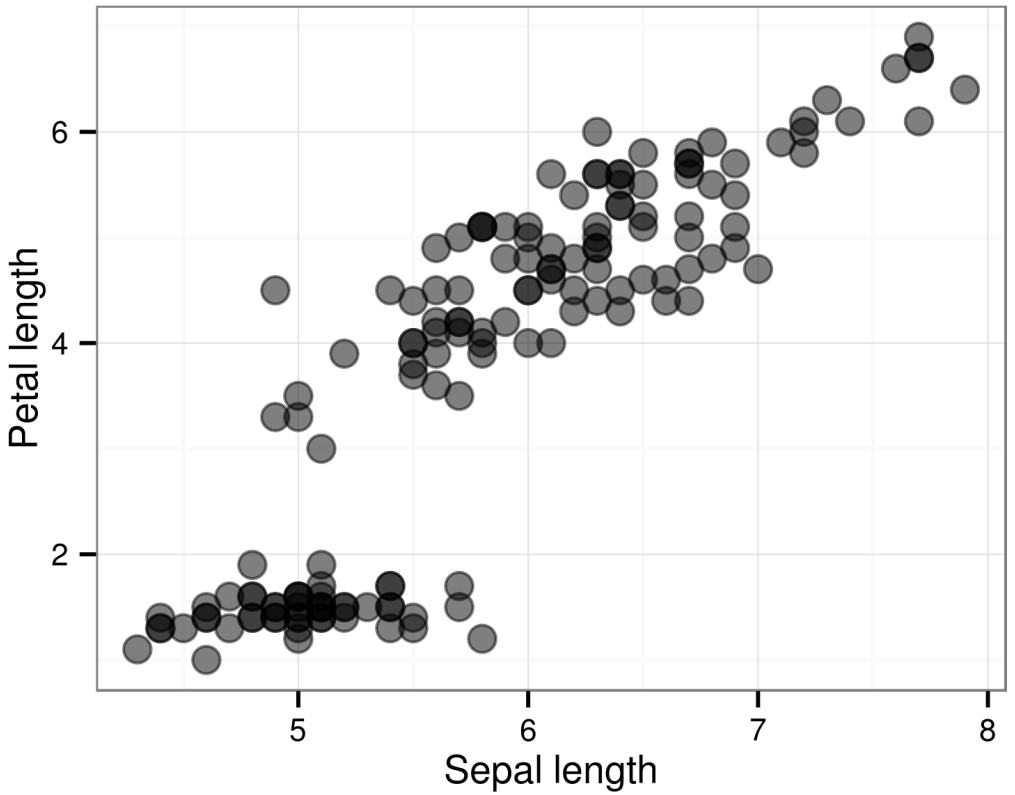


ggplot - exemples

```
# on modifie quelques valeurs
ggplot(d, aes(x = Sepal.Length, y = Petal.Length)) +
    geom_point(shape = 20, size = 5, alpha = 0.5) +
    ylab("Petal length") + xlab("Sepal length")

# équivalent
ggplot(d, aes(x = Sepal.Length, y = Petal.Length)) +
    geom_point(shape = 20, size = 5, alpha = 0.5) +
    scale_y_continuous(name = "Petal length") +
    scale_x_continuous(name = "Sepal length")
```

On fixe (set) la taille, la forme et la transparence..

On peut aussi spécifier les étiquettes des axes via les échelles



ggplot - exemples

"aesthetic setting"
On fixe la couleur à la valeur "darkgreen"

A

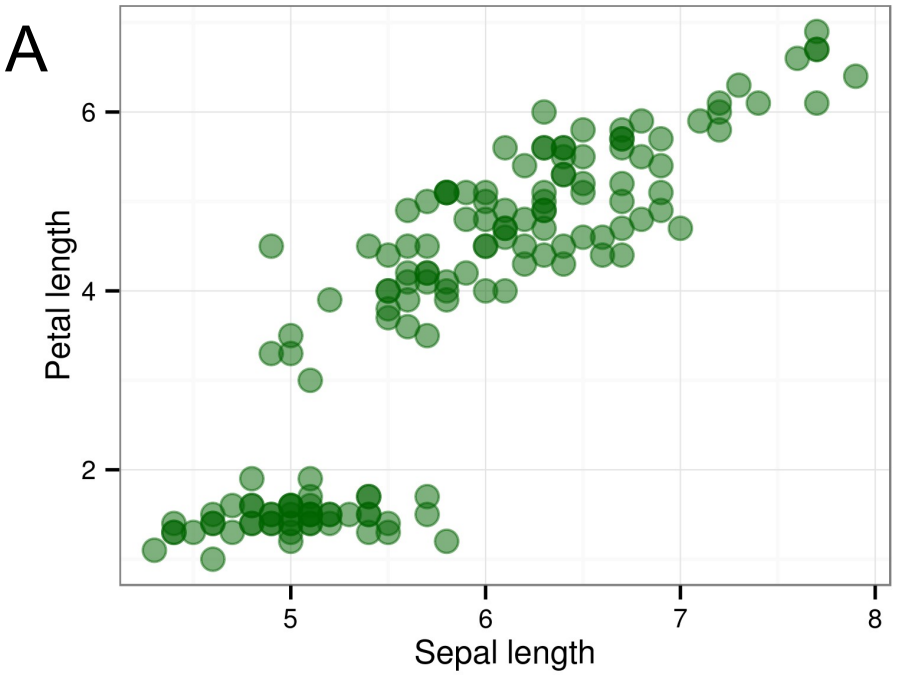
```
ggplot(d, aes(x = Sepal.Length, y = Petal.Length)) +
    geom_point(shape = 20, size = 5, alpha = 0.5, color = "darkgreen") +
    ylab("Petal length") + xlab("Sepal length")
```

B

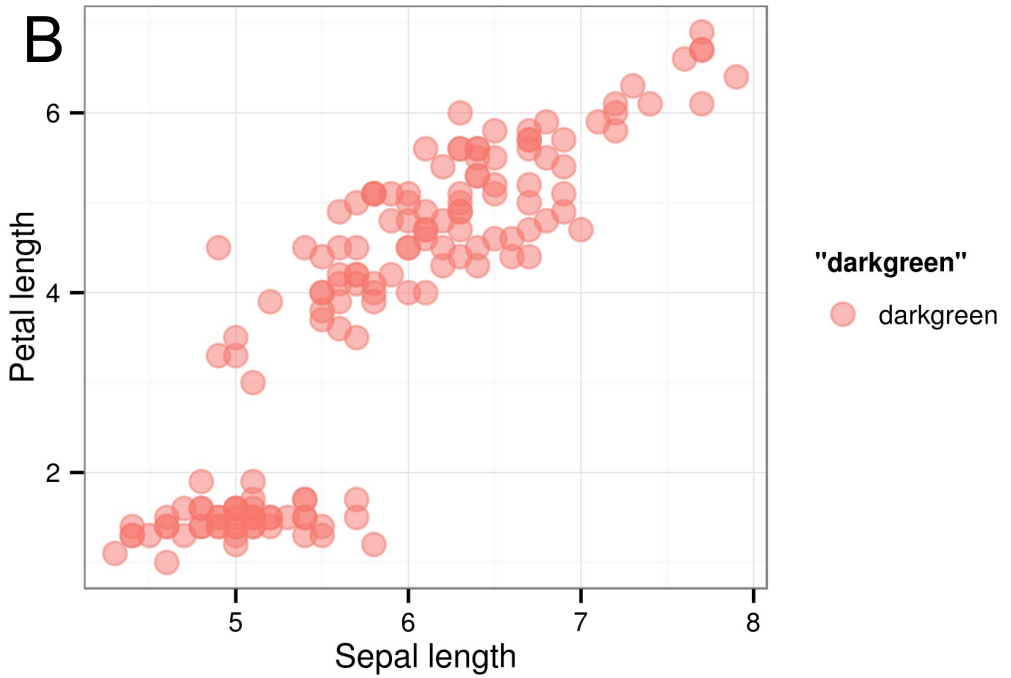
```
dev.new(12/2.54, 8/2.54)
ggplot(d, aes(x = Sepal.Length, y = Petal.Length, color = "darkgreen")) +
    geom_point(shape = 20, size = 5, alpha = 0.5) +
    ylab("Petal length") + xlab("Sepal length")
```

"aesthetic mapping"
On crée une variable appelée "darkgreen" et on lui attribue une couleur par défaut
--> ce n'est pas ce qu'on veut !!

A



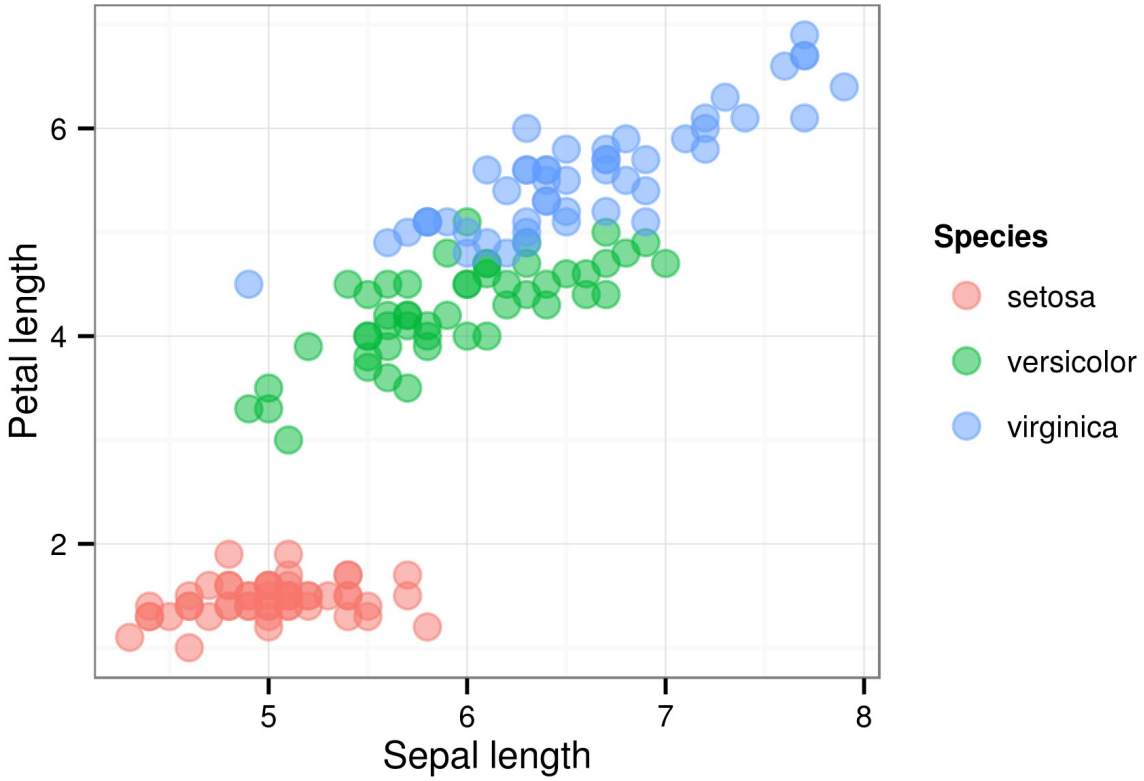
B



ggplot - exemples

```
ggplot(d, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
    geom_point(shape = 20, size = 5, alpha = 0.5) +
    ylab("Petal length") + xlab("Sepal length")
```

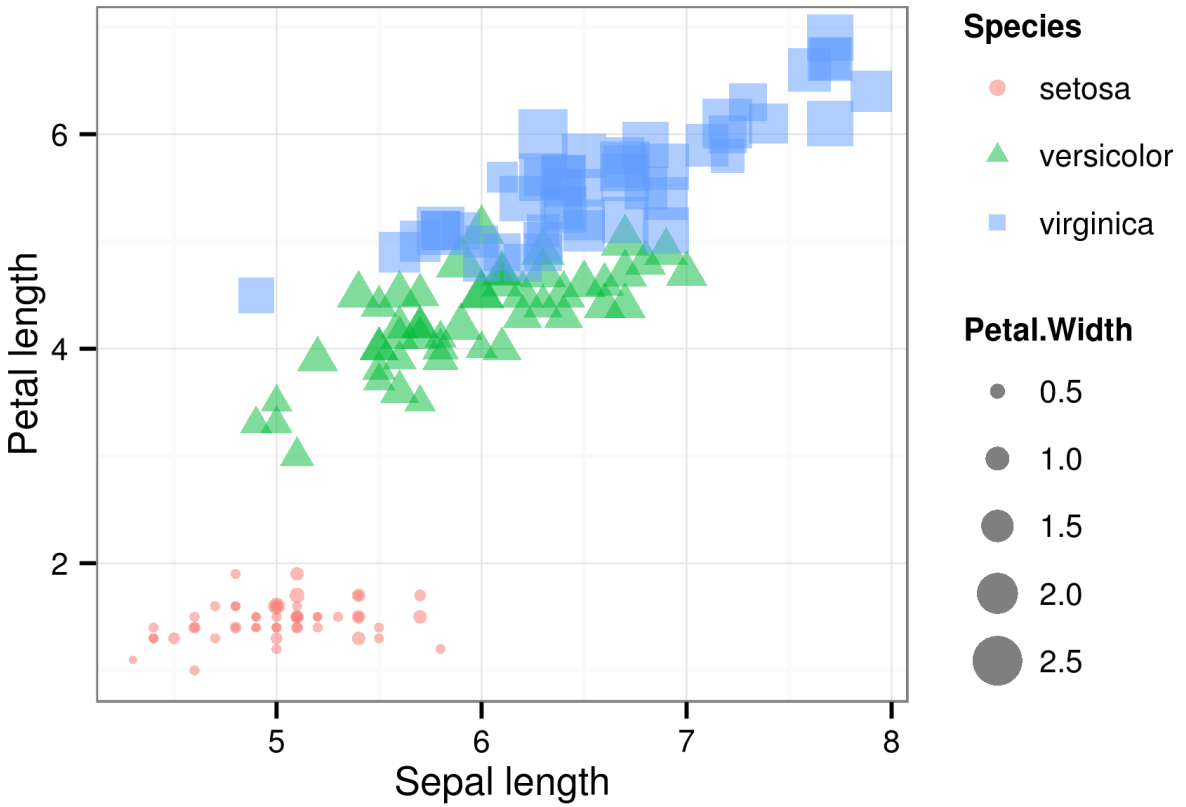
"aesthetic mapping"
On attribue la couleur en fonction de la variable "Species"



ggplot - exemples

```
ggplot(d, aes(x = Sepal.Length, y = Petal.Length,
              color = Species, shape = Species, size = Petal.Width)) +
    geom_point(alpha = 0.5) +
    ylab("Petal length") + xlab("Sepal length")
```

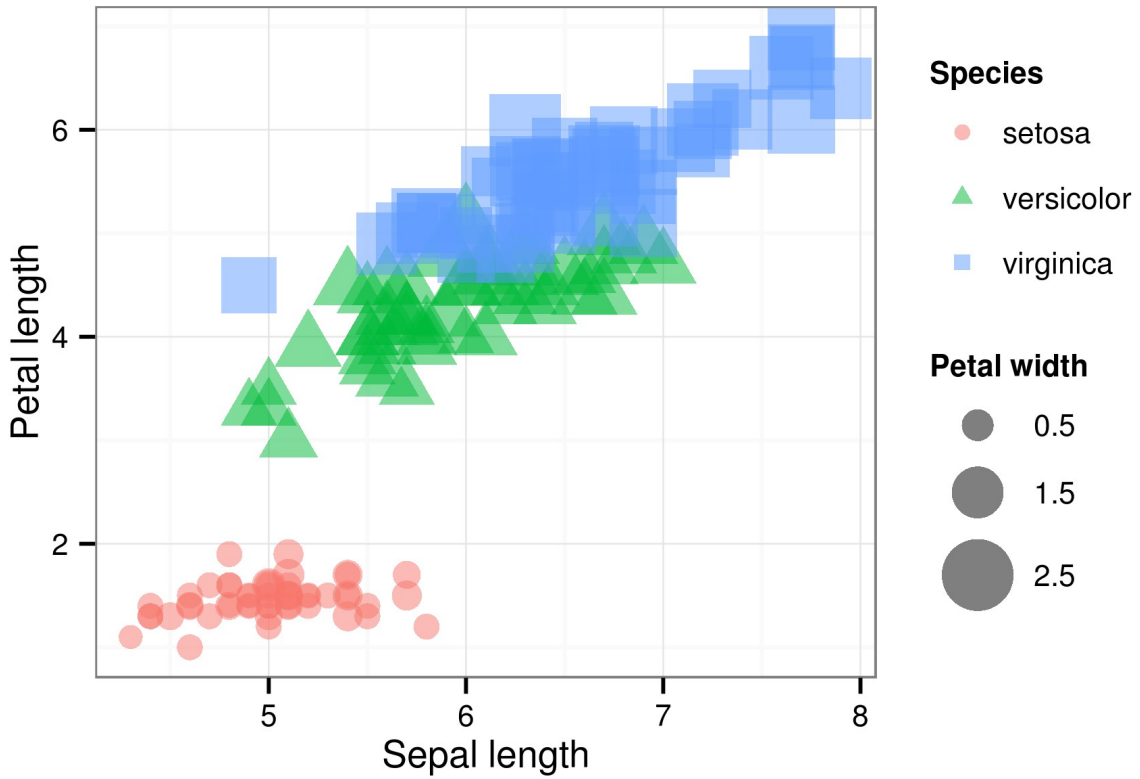
"aesthetic mapping"
On attribue la couleur et la forme en fonction de la variable "Species" et la taille en fonction de la variable "Sepal.length"



ggplot - exemples

```
ggplot(d, aes(x = Sepal.Length, y = Petal.Length,
              color = Species, shape = Species, size = Petal.Width)) +
    geom_point(alpha = 0.5) +
    ylab("Petal length") + xlab("Sepal length") +
    scale_size_continuous(range = c(3,9), breaks = c(0.5, 1.5, 2.5),
                          name = "Petal width")
```

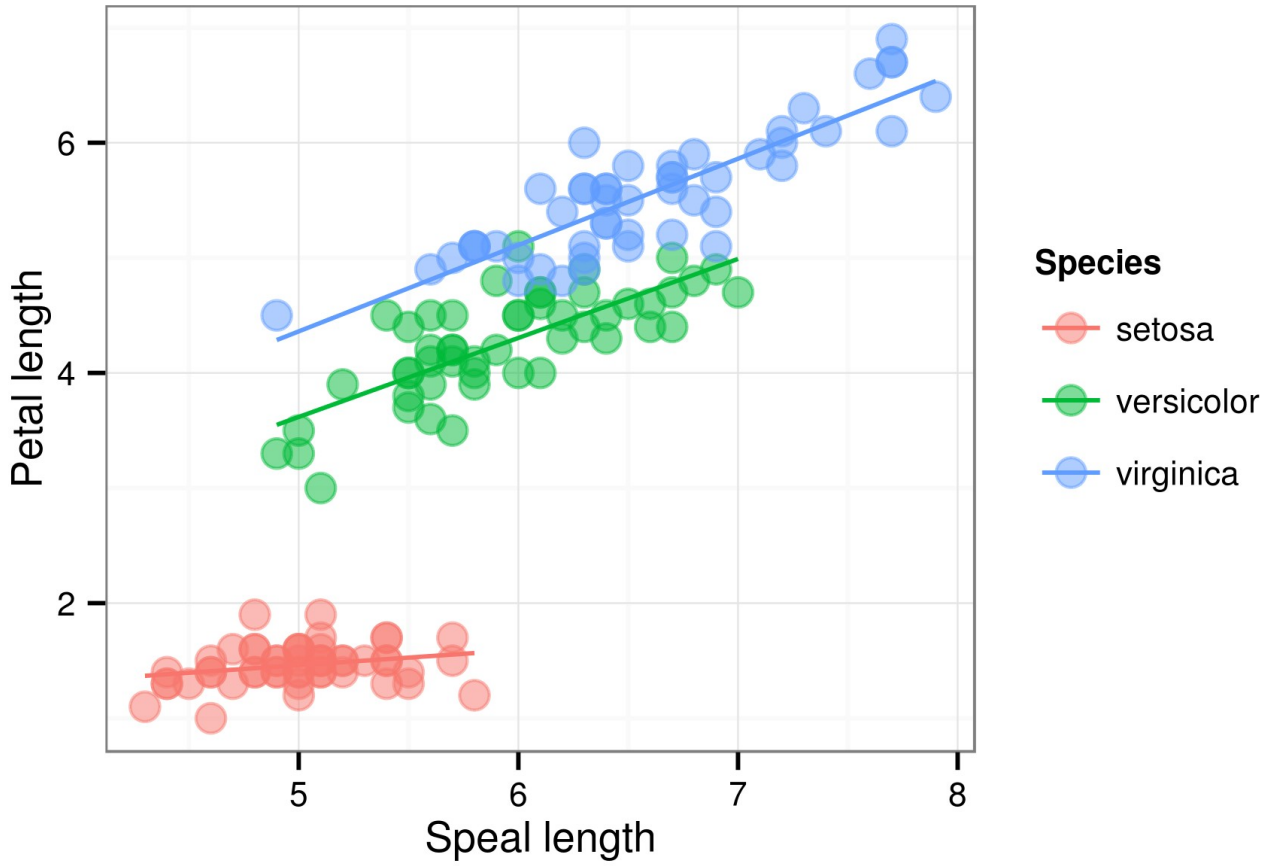
On peut modifier la manière dont l'échelle de taille converti les données en taille



ggplot - exemples

```
ggplot(d, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
    geom_point(shape = 20, size = 5, alpha = 0.5) +
    stat_smooth(method = "lm", se = FALSE) +
    ylab("Petal length") + xlab("Speal length")
```

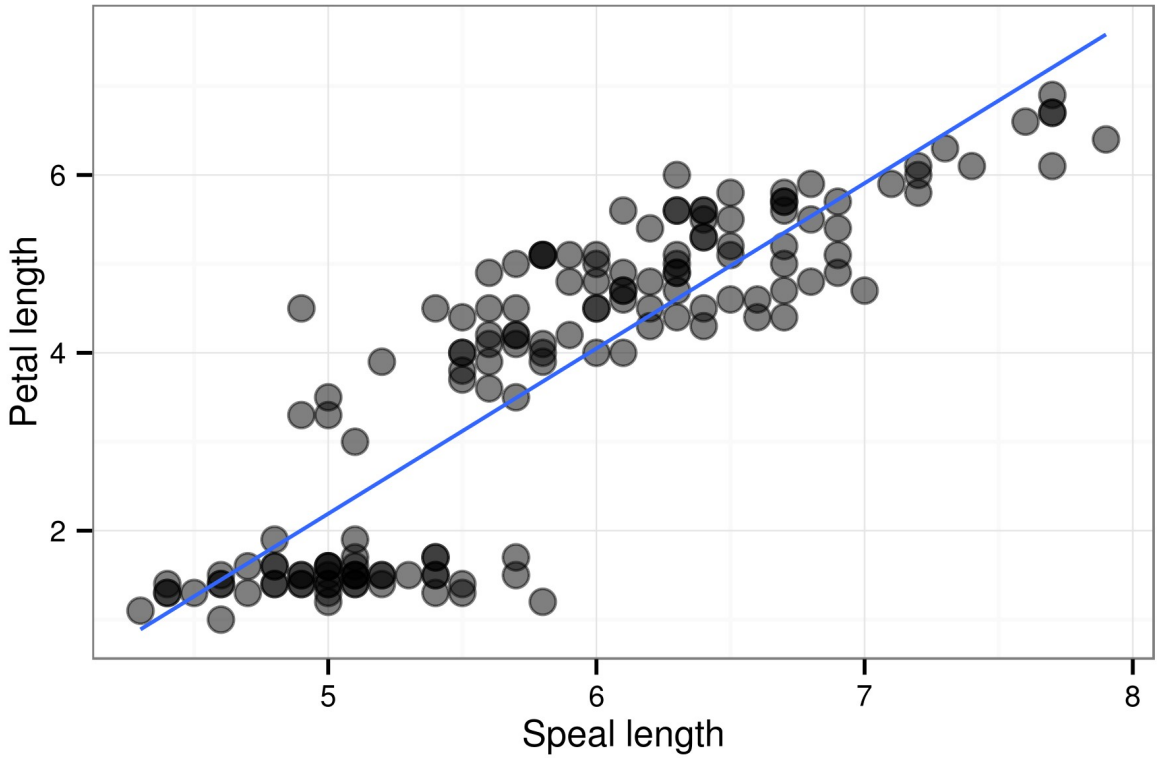
On ajoute une courbe de tendance sous la forme d'une régression linéaire



ggplot - exemples

```
ggplot(d, aes(x = Sepal.Length, y = Petal.Length)) +
    geom_point(shape = 20, size = 5, alpha = 0.5)+
    stat_smooth(method = "lm", se = FALSE) +
    ylab("Petal length") + xlab("Speal length")
```

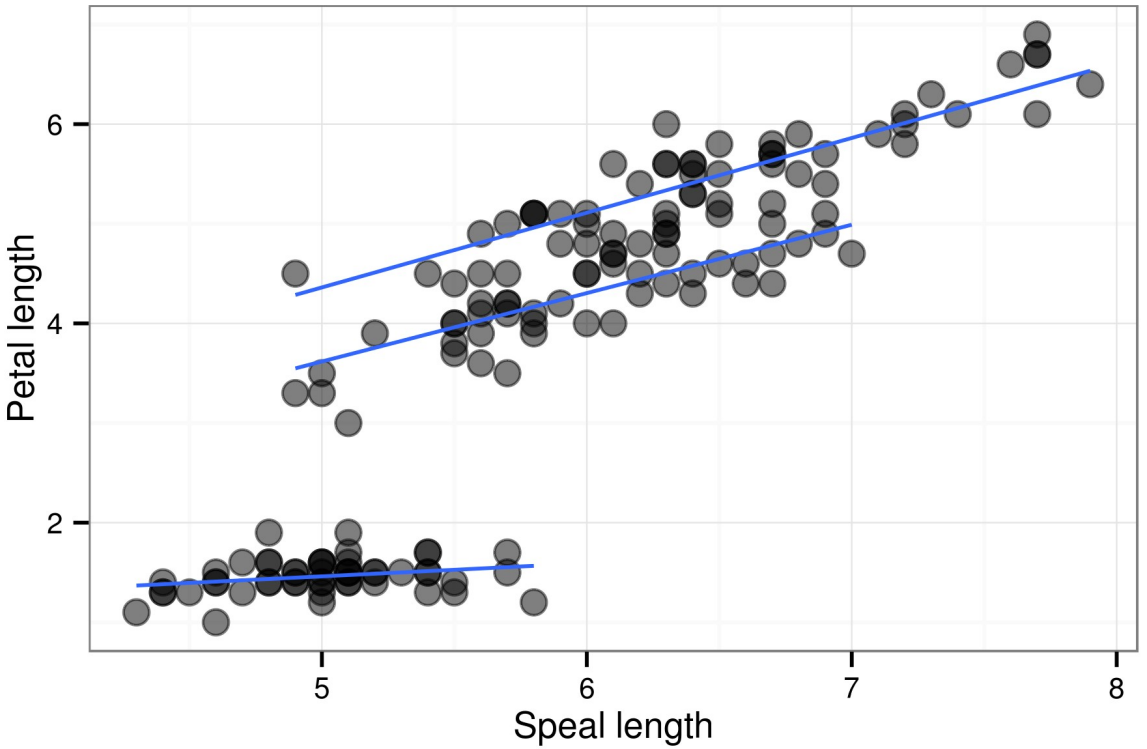
Si on ne spécifie pas de couleur par espèce dans aes() il trace une seule droite de régression à travers tous les points



ggplot - exemples

```
ggplot(d, aes(x = Sepal.Length, y = Petal.Length, group = Species)) +
    geom_point(shape = 20, size = 5, alpha = 0.5) +
    stat_smooth(method = "lm", se = FALSE) +
    ylab("Petal length") + xlab("Speal length")
```

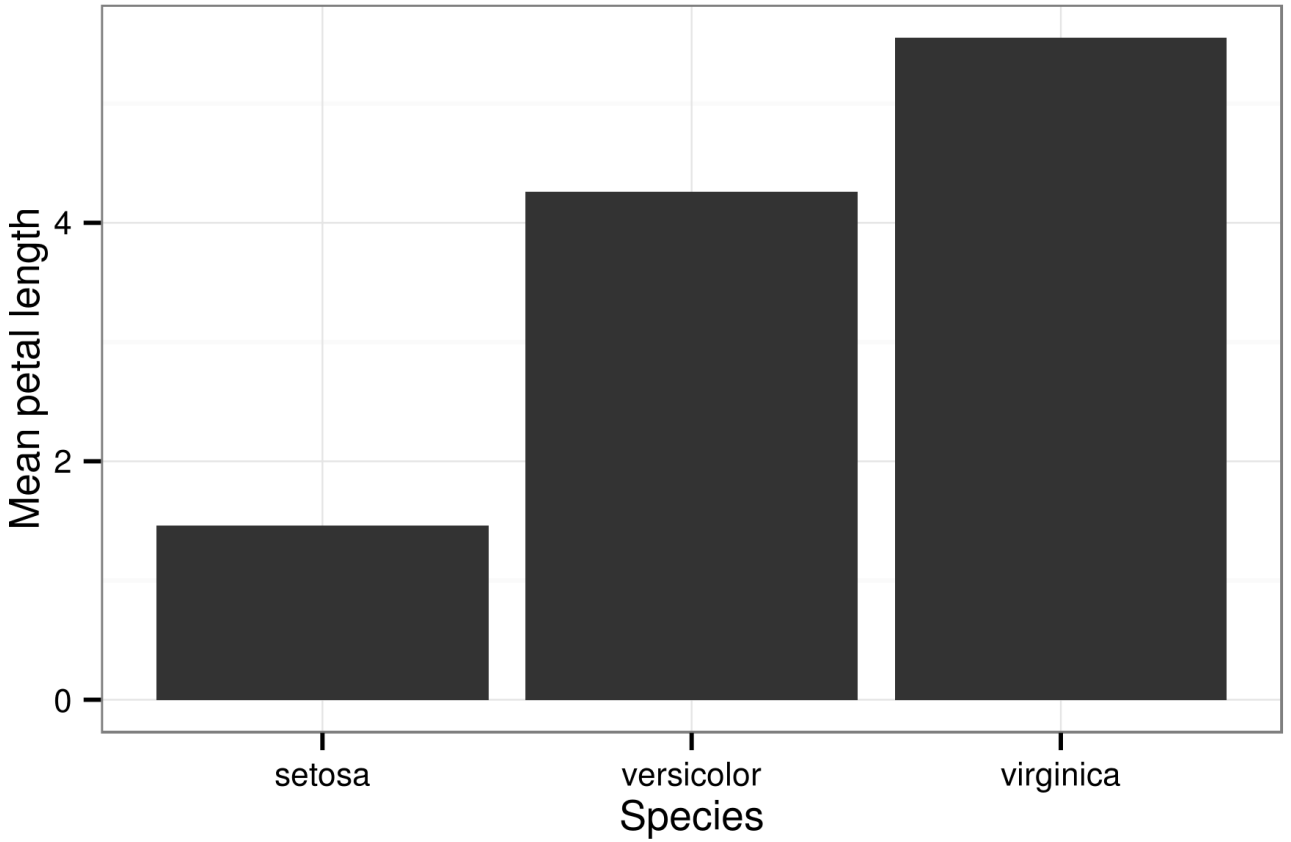
on peut spécifier une aestethic "group" pour estimer une droite pour chaque espèce même si on ne spécifie pas la couleur



ggplot - exemples

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    stat_summary(fun.y = mean, geom = "bar") +
    ylab("") + ylab("Mean petal length")
```

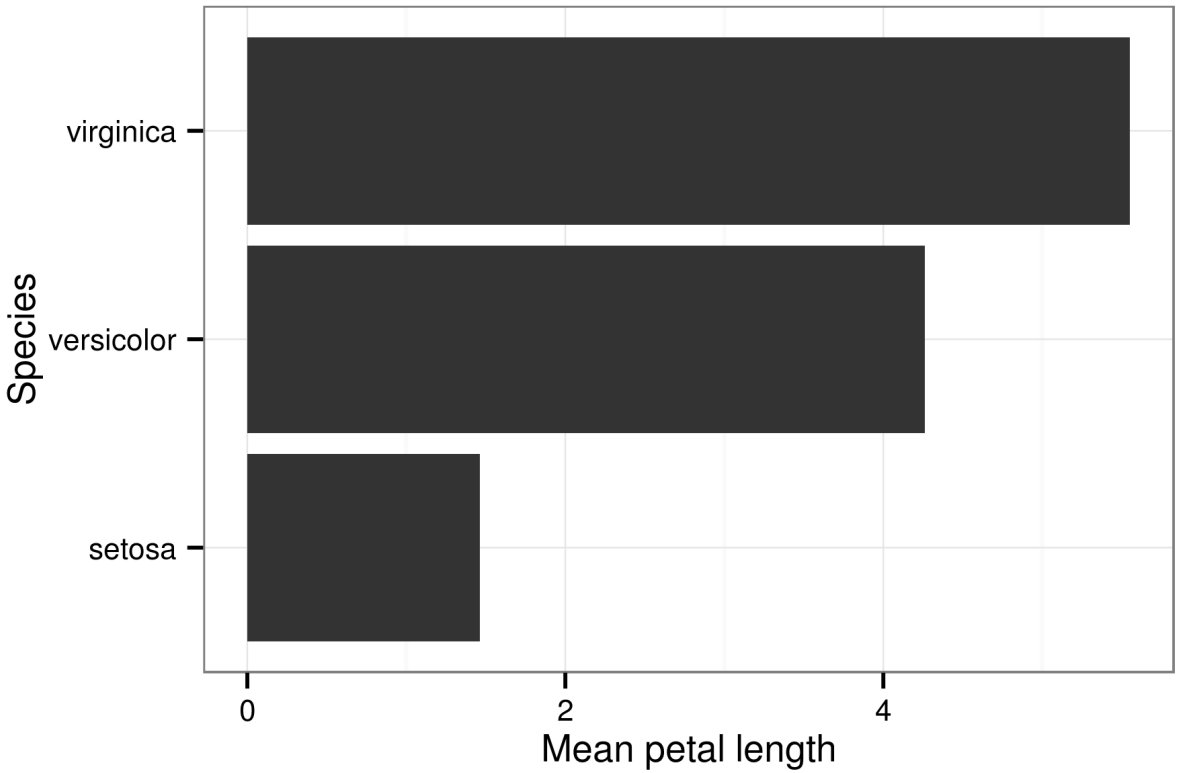
pas besoin de calculer des valeurs moyennes, on peut le faire avec stat_summary directement dans le graphe



ggplot - exemples

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    stat_summary(fun.y = mean, geom = "bar") +
    ylab("") + ylab("Mean petal length") +
    coord_flip()
```

Souvent les barplot horizontaux sont plus adaptés (lecture des étiquettes plus facile

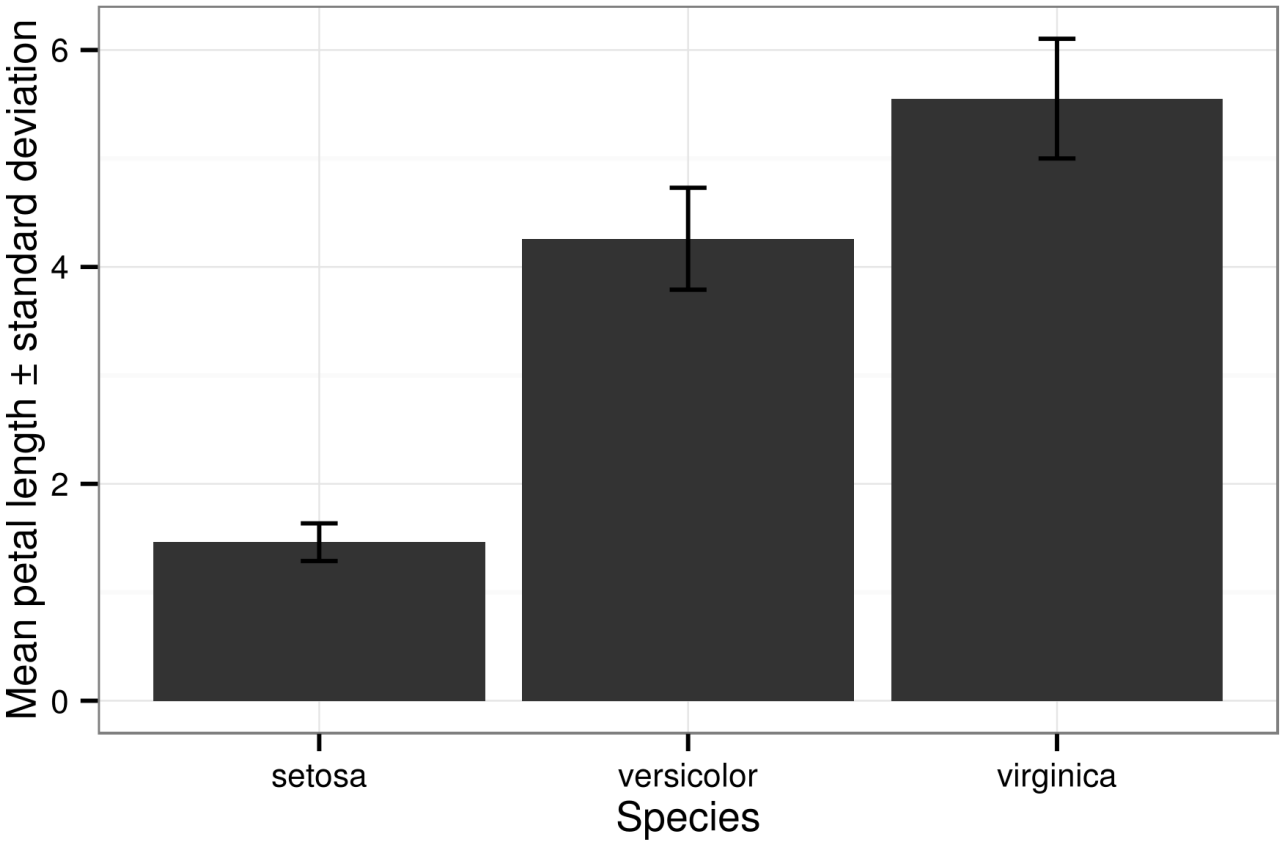


ggplot - exemples

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    stat_summary(fun.y = mean, geom = "bar") +
    stat_summary(fun.ymin = function(x) mean(x) - sd(x),
                 fun.ymax = function(x) mean(x) + sd(x),
                 geom = "errorbar", width = 0.1) +
    ylab("") + ylab("Mean petal length \u00B1 standard deviation")
```

Ajout de barres d'erreur

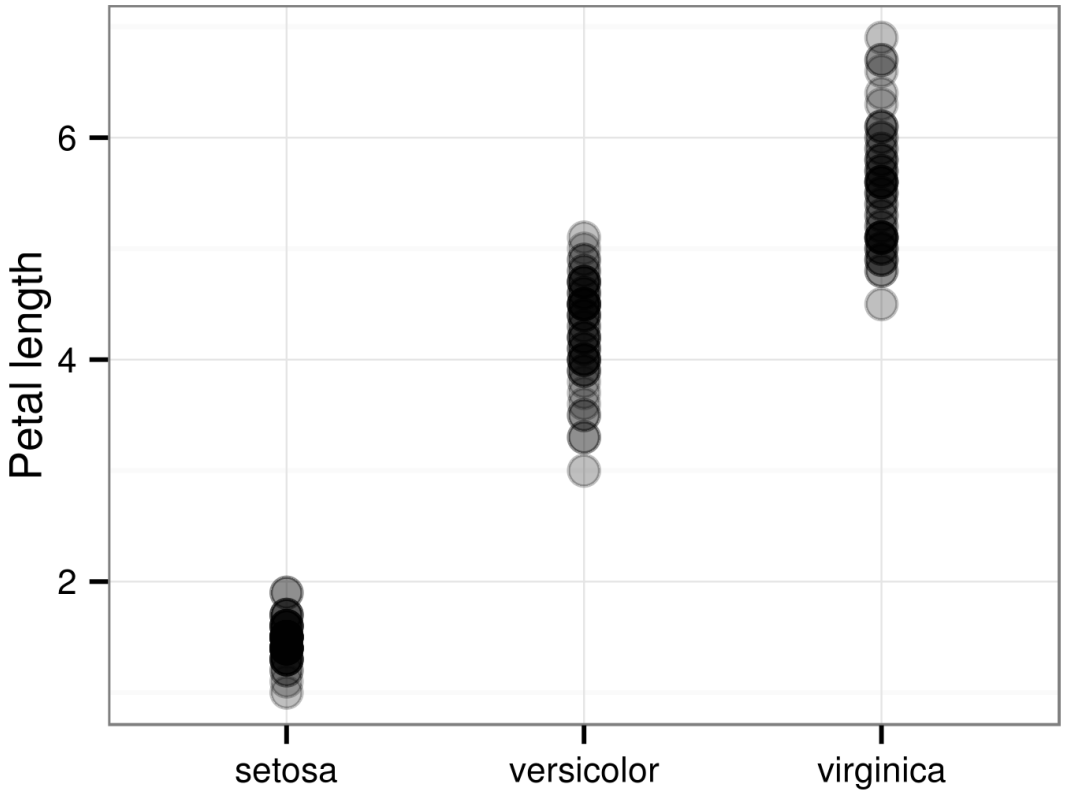
notation Unicode du symbole ±
http://unicode-table.com/fr/



ggplot - exemples

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    geom_point(shape = 20, size = 5, alpha = 0.25) +
    xlab("") + ylab("Petal length")
```

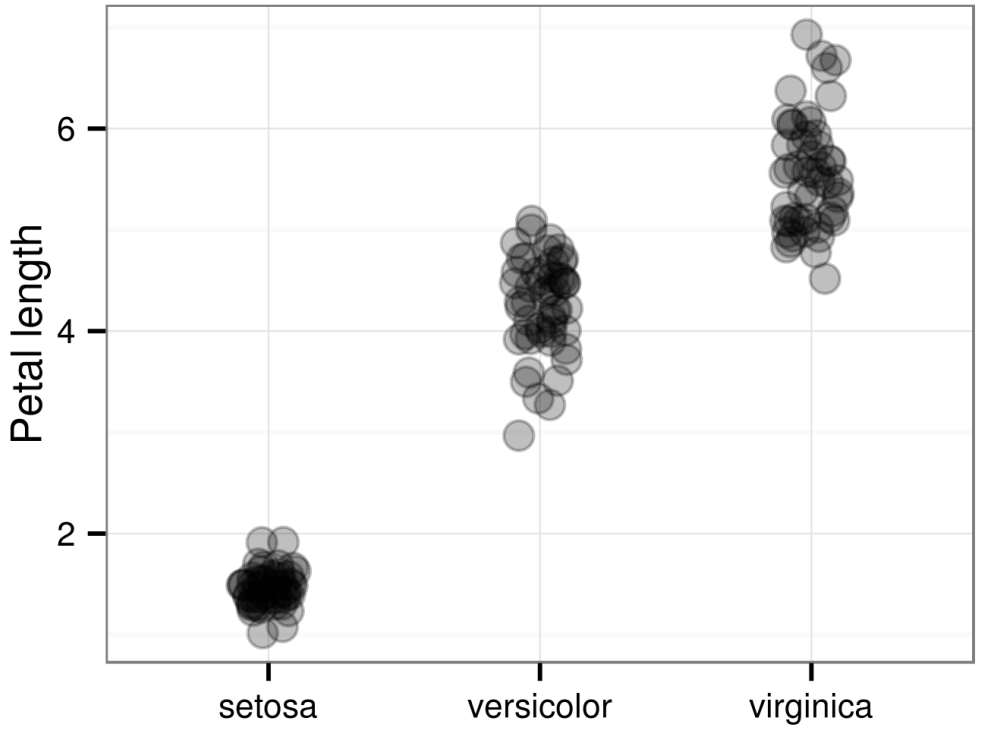
Autre réprésentation des données **Overplotting !**



ggplot - exemples

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    geom_point(shape = 20, size = 5, alpha = 0.25,
               position = position_jitter(width = 0.1)) +
    xlab("") + ylab("Petal length")
```

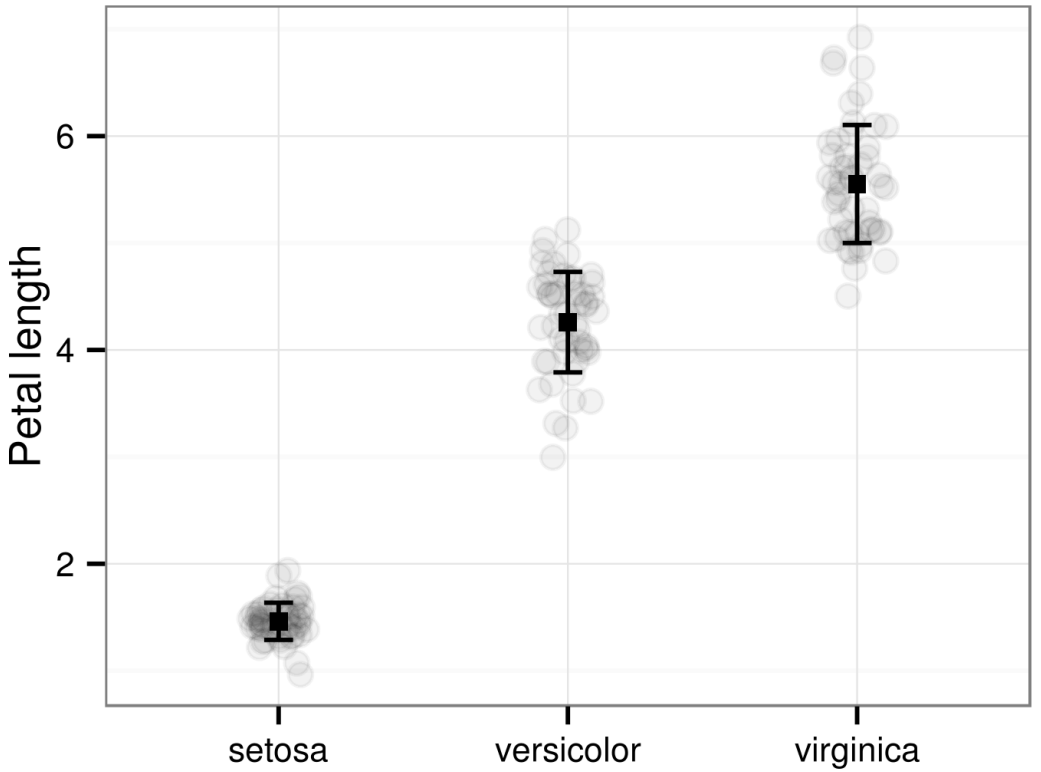
"jittering"
On ajoute un peu de bruit sur l'axe des x pour limiter les effets de l'overplotting



ggplot - exemples

On ajoute la moyenne + barres d'erreur représentant l'écart type

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    geom_point(shape = 20, size = 4, alpha = 0.05,
               position = position_jitter(width = 0.1)) +
    stat_summary(fun.y = mean, geom = "point", shape = 15, size = 2,
                 color = "black") +
    stat_summary(fun.ymin = function(x) mean(x) - sd(x),
                 fun.ymax = function(x) mean(x) + sd(x),
                 geom = "errorbar", width = 0.1, color = "black") +
    xlab("") + ylab("Petal length")
```



Edward Tufte graphical principles

(partim)

For non-data-ink, less is more.
For data-ink, less is a bore.

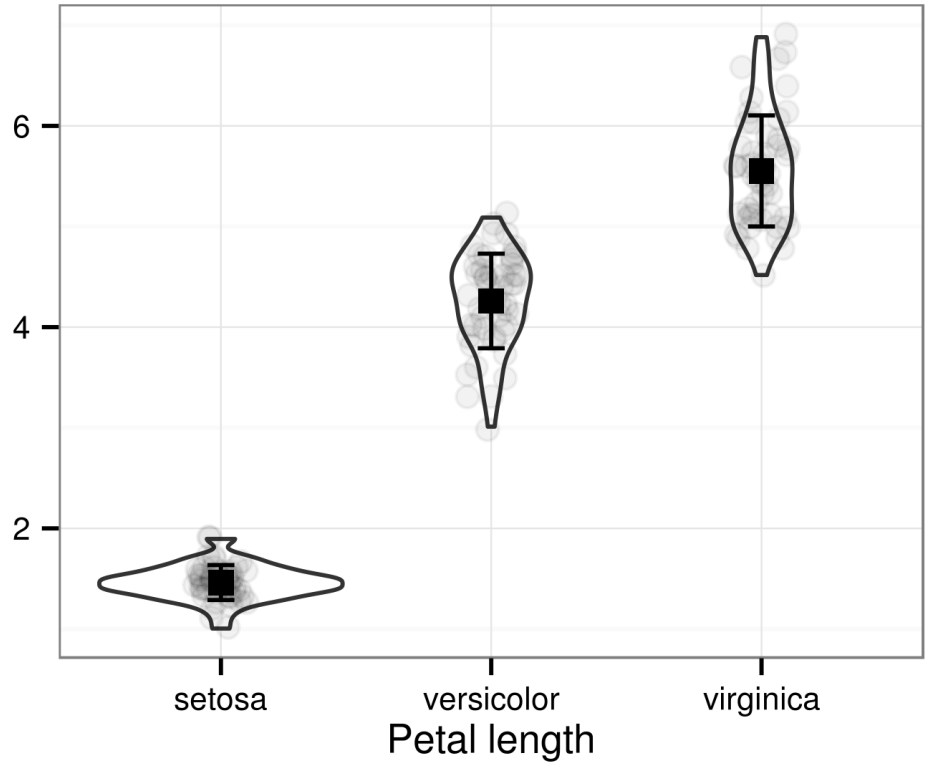
Above all else show the data.
Maximize the data-ink ratio. ← data-ink/total ink
Erase non-data-ink.
Erase redundant data-ink.
Revise and edit.

Graphics must not quote data out of context

ggplot - exemples

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    geom_violin() +
    geom_point(shape = 20, size = 4, alpha = 0.05,
               position = position_jitter(width = 0.1)) +
    stat_summary(fun.y = mean, geom = "point", shape = 15, size = 3,
                 color = "black") +
    stat_summary(fun.ymin = function(x) mean(x) - sd(x),
                 fun.ymax = function(x) mean(x) + sd(x),
                 geom = "errorbar", width = 0.1, color = "black") +
    ylab("") + xlab("Petal length")
```

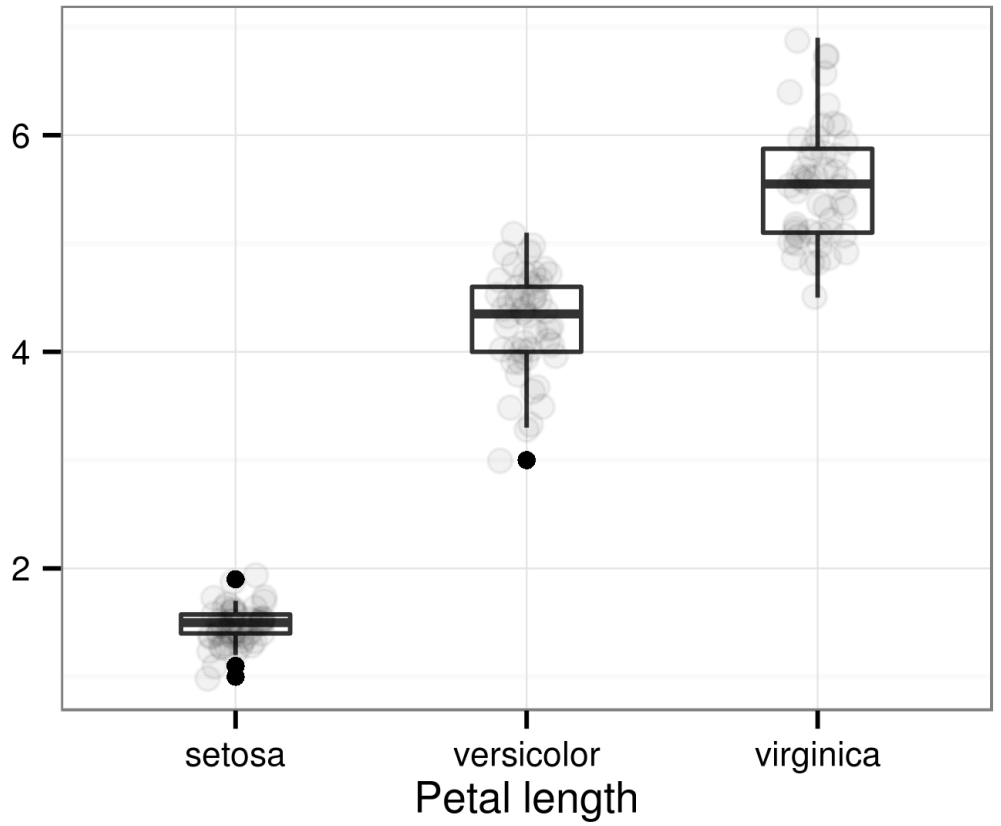
Violin plot représentant la densité des données. plus informatif qu'un boxplot !



ggplot - exemples

Boxplot + points

```
ggplot(d, aes(x = Species, y = Petal.Length)) +
    geom_boxplot(, width = 0.5) +
    geom_point(shape = 20, size = 4, alpha = 0.05,
               position = position_jitter(width = 0.1)) +
    ylab("") + xlab("Petal length")
```



ggplot - exemples

On va travailler sur le même jeu de données en format long
On sépare les information Pétale/Sépale de la largeur/longueur

```
library(reshape)
data(iris)
d <- melt(iris, id = "Species")
d$Flowerpart <- gsub("(.*)\\.(.*)", "\\1", d$variable)
d$Dimension <- gsub("(.*)\\.(.*)", "\\2", d$variable)
d$Size <- d$value
```

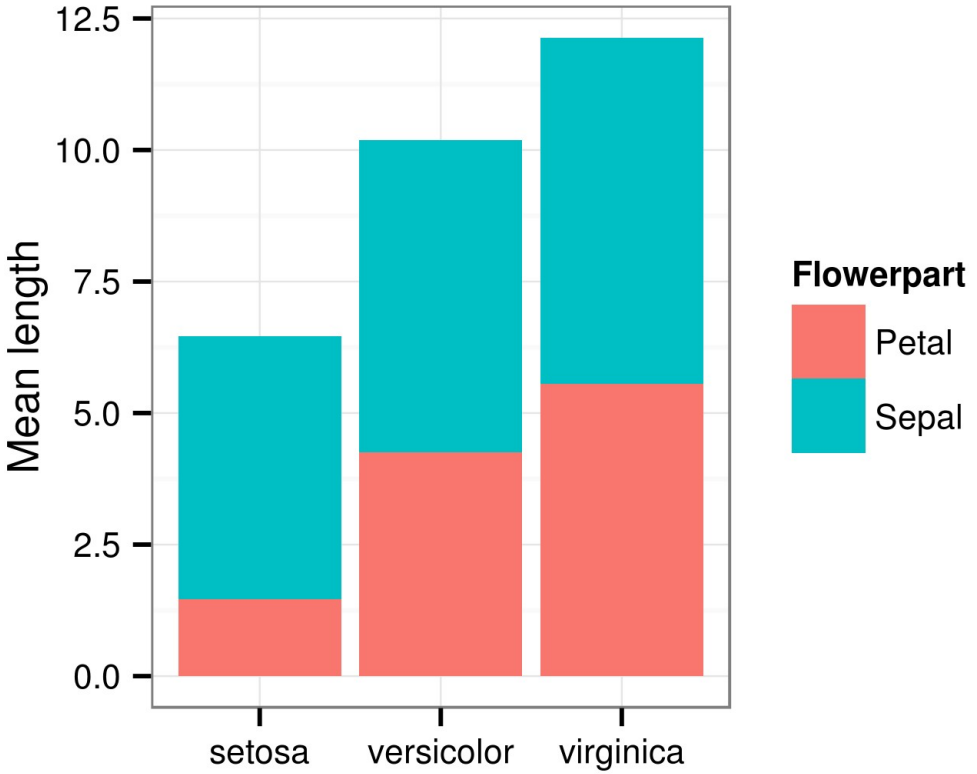
```
> d[sample(1:nrow(d), 10),]

       Species     variable value Flowerpart Dimension Size
396 versicolor Petal.Length   4.2      Petal    Length  4.2
199     setosa  Sepal.Width   3.7      Sepal     Width  3.7
314     setosa Petal.Length   1.1      Petal    Length  1.1
219 versicolor  Sepal.Width   2.2      Sepal     Width  2.2
388 versicolor Petal.Length   4.4      Petal    Length  4.4
519 versicolor  Petal.Width   1.5      Petal     Width  1.5
441  virginica Petal.Length   5.6      Petal    Length  5.6
156     setosa  Sepal.Width   3.9      Sepal     Width  3.9
427  virginica Petal.Length   4.8      Petal    Length  4.8
162     setosa  Sepal.Width   3.4      Sepal     Width  3.4
```

On ne travaille que avec les longueurs

ggplot - exemples

```
ggplot(d[d$Dimension == "Length",], aes(x = Species, y = Size,
                                        fill = Flowerpart)) +
    geom_bar(stat= "summary", fun.y = mean) +
    xlab("") + ylab("Mean length")
```

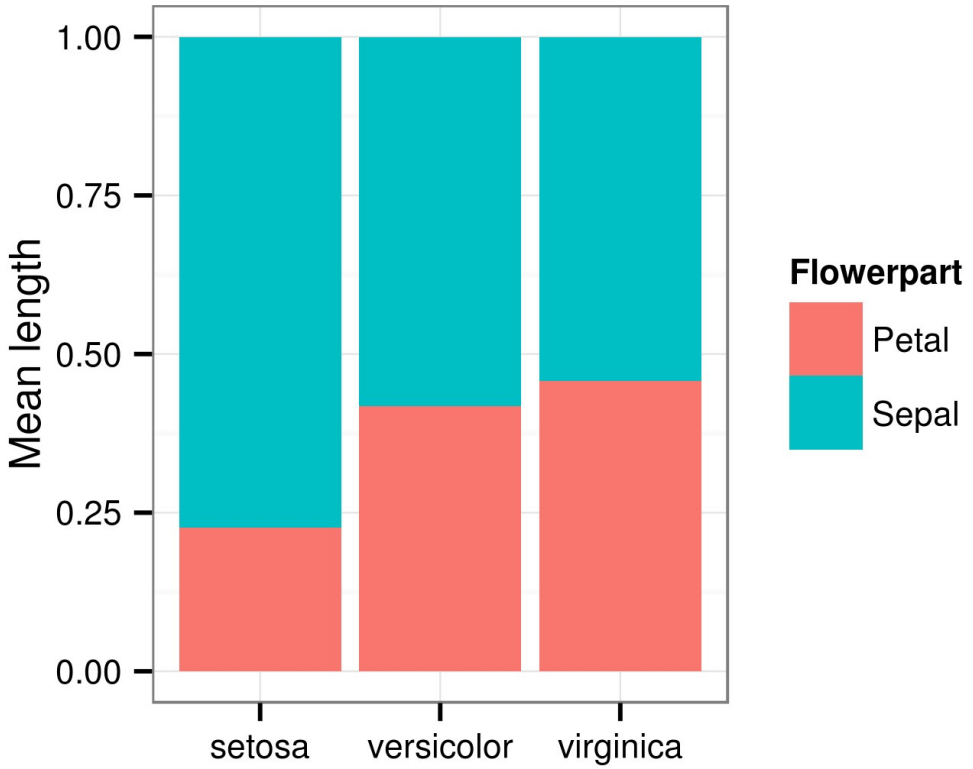
Barplot avec couleurs en fonction de la partie de la fleur mesurée (pétale ou sépale



ggplot - exemples

```
ggplot(d[d$Dimension == "Length",], aes(x = Species, y = Size,
                                        fill = Flowerpart)) +
    geom_bar(stat= "summary", fun.y = mean,
             position = position_fill()) +
    xlab("") + ylab("Mean length")
```

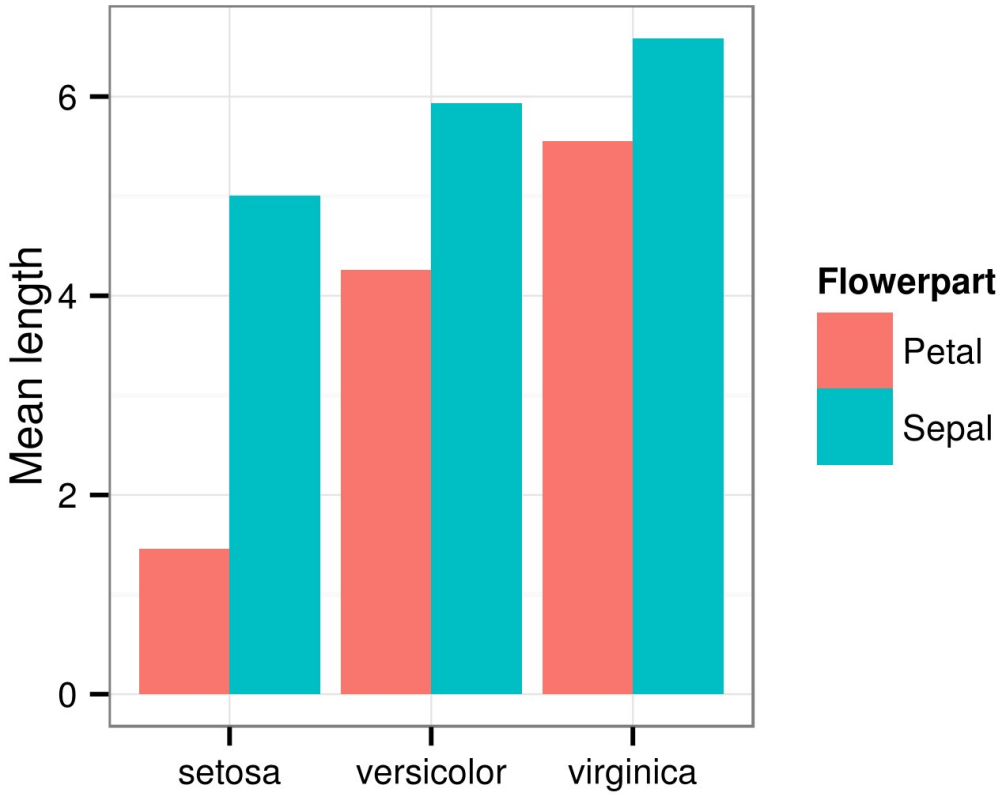
On change juste l'ajustement de la position. Représentation sans intérêt !!



ggplot - exemples

```
ggplot(d[d$Dimension == "Length",], aes(x = Species, y = Size,
                                        fill = Flowerpart)) +
    geom_bar(stat= "summary", fun.y = mean,
             position = position_dodge()) +
    xlab("") + ylab("Mean length")
```

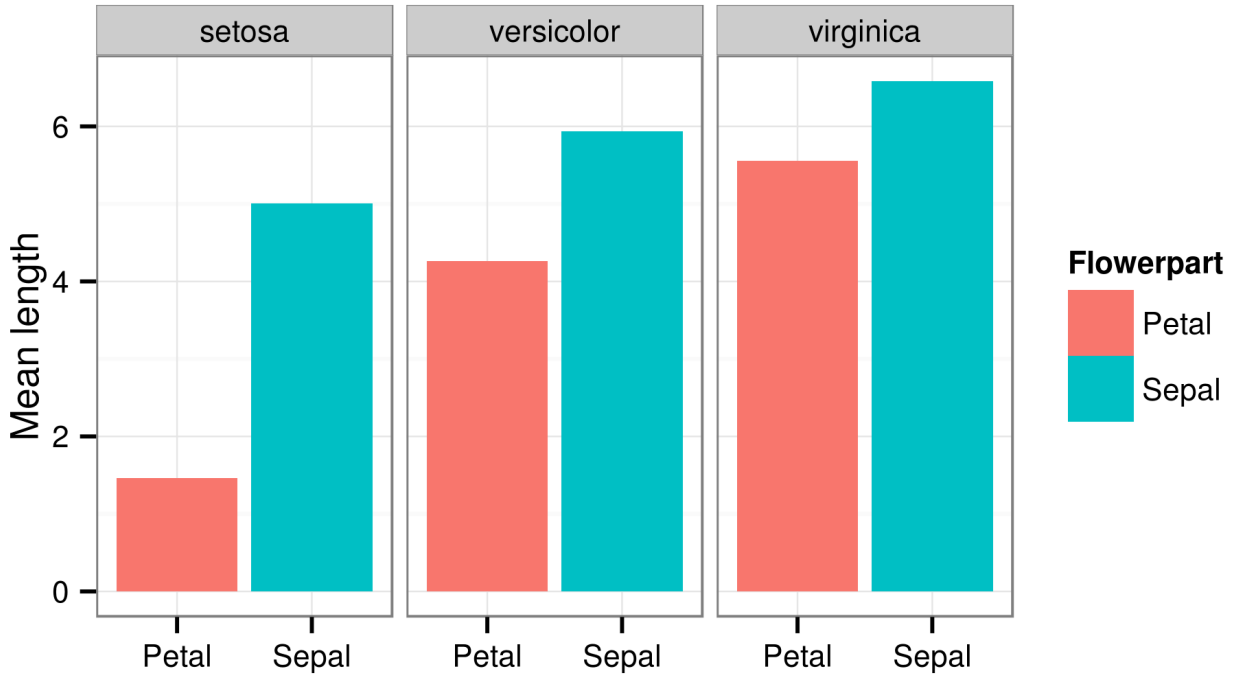
On change encore l'ajustement de la position. Représentation meilleure que les deux précédentes !!



ggplot - exemples

```
ggplot(d[d$Dimension == "Length",], aes(x = Flowerpart, y = Size,
                                        fill = Flowerpart)) +
    geom_bar(stat= "summary", fun.y = mean,
             position = position_dodge()) +
    facet_grid(.~Species) +
    xlab("") + ylab("Mean length")
```

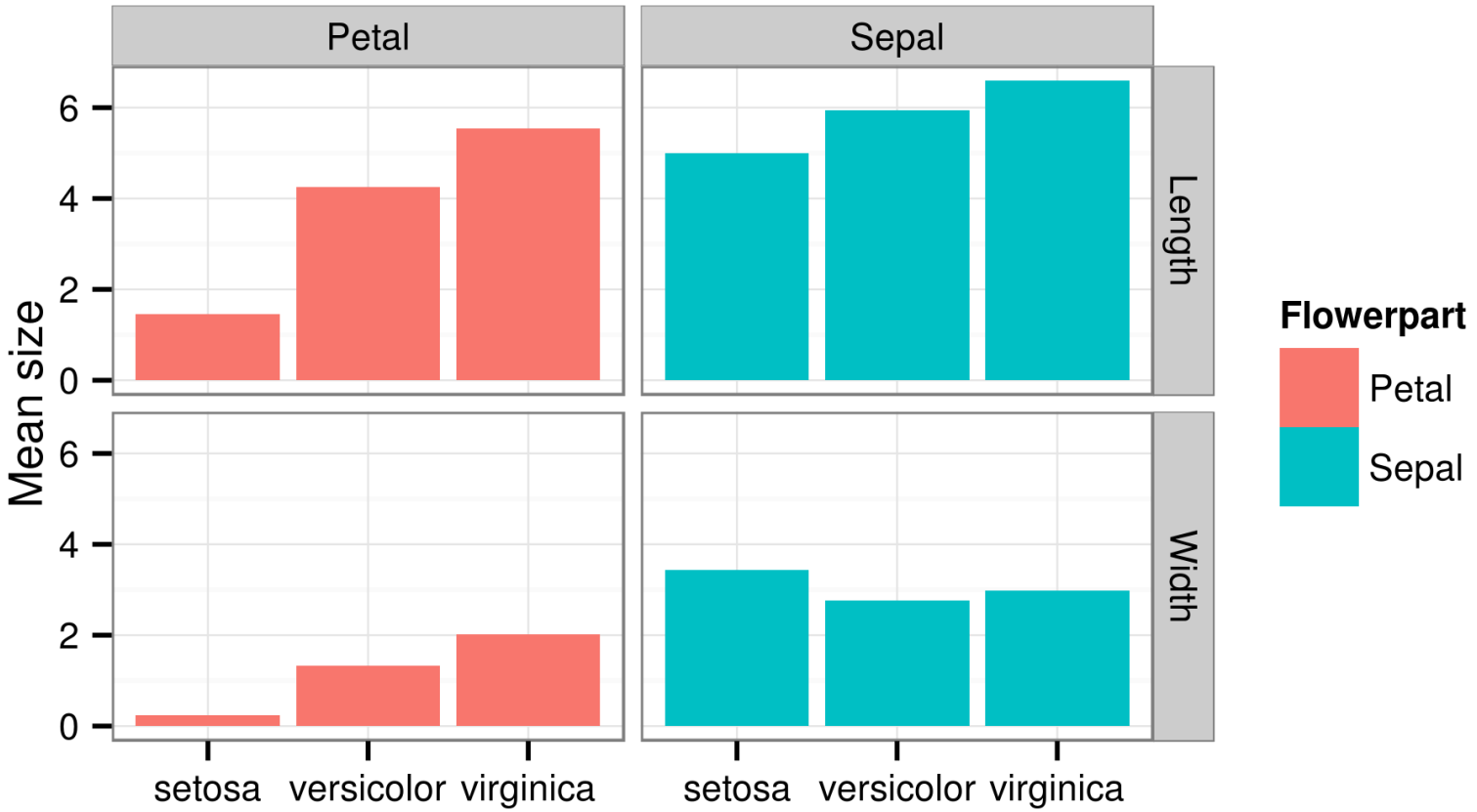
Facetting : résultat très similaire au précédent dans ce cas (barplot avec position = dodge)



ggplot - exemples

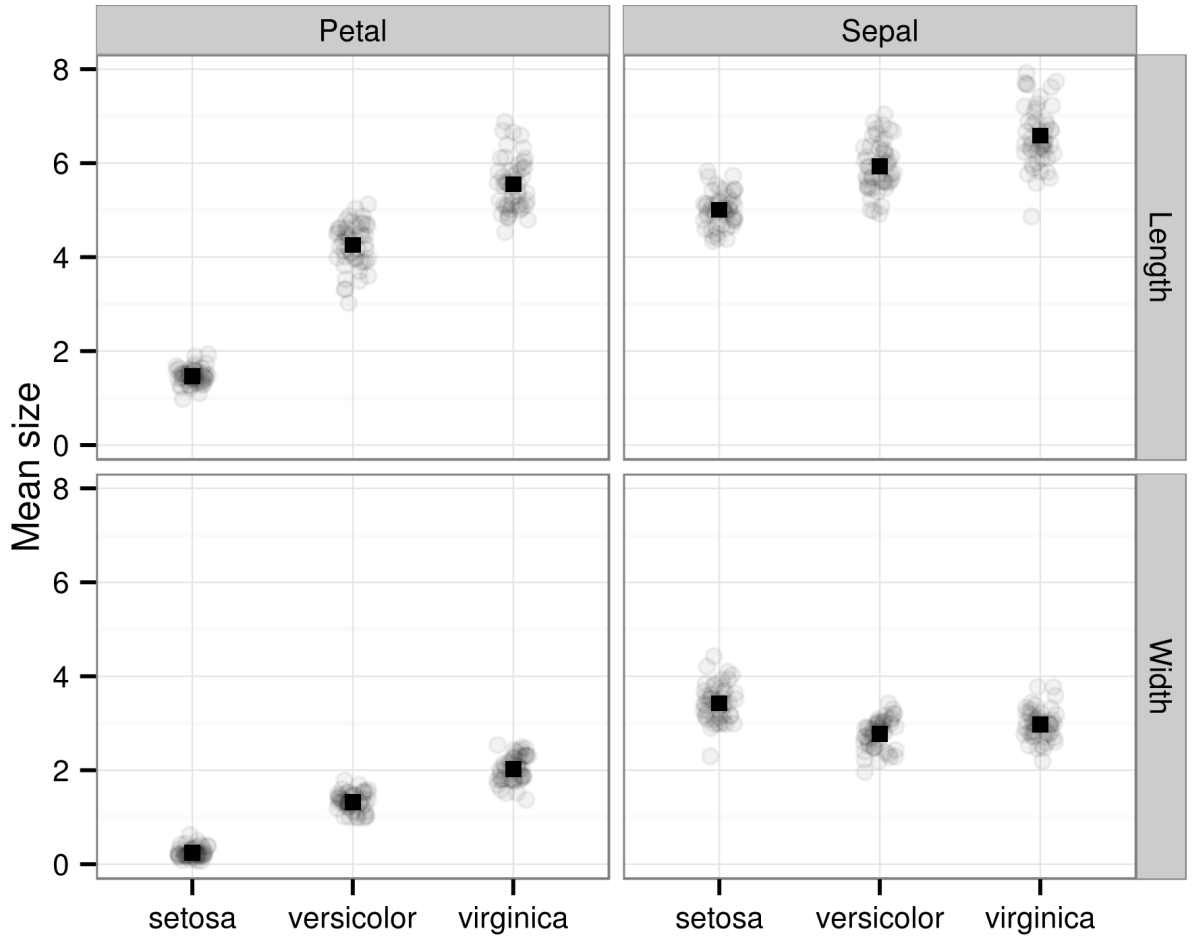
```
ggplot(d, aes(x = Species, y = Size, fill = Flowerpart)) +
    geom_bar(stat= "summary", fun.y = mean,
             position = position_dodge()) +
    facet_grid(Dimension ~ Flowerpart) +
    xlab("") + ylab("Mean size")
```

Facetting avec deux variables



ggplot - exemples

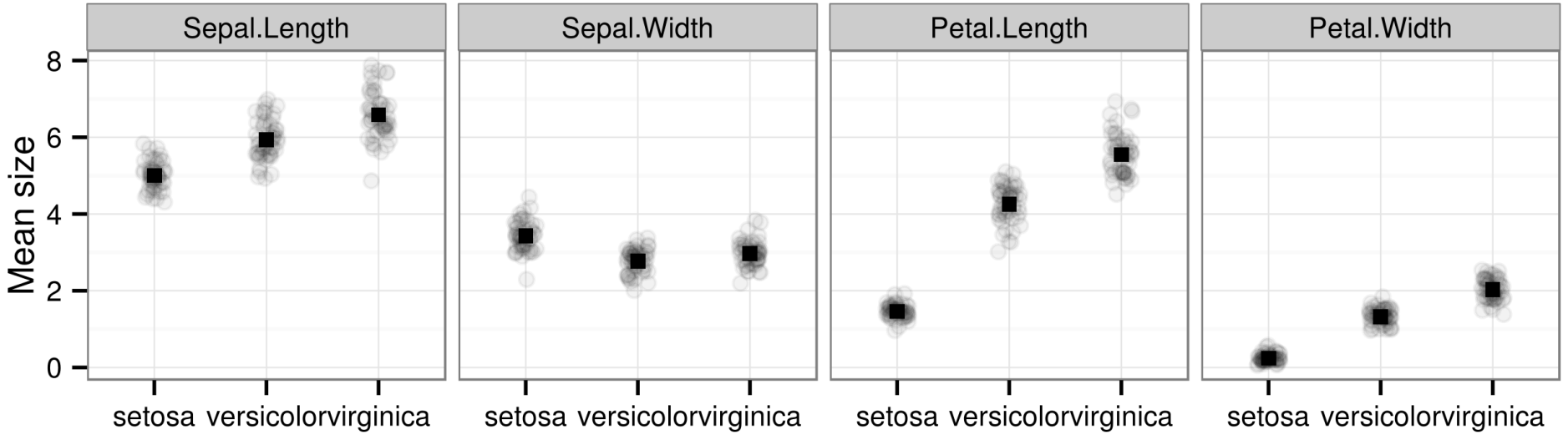
```
ggplot(d, aes(x = Species, y = Size)) +
    geom_point(shape = 20, size = 3, alpha = 0.05,
               position = position_jitter(width = 0.1)) +
    stat_summary(fun.y = mean, geom = "point", shape = 15, size = 2, color = "black") +
    facet_grid(Dimension ~ Flowerpart) +
    xlab("") + ylab("Mean size")
```



ggplot - exemples

```
ggplot(d, aes(x = Species, y = Size)) +
    geom_point(shape = 20, size = 3, alpha = 0.05,
               position = position_jitter(width = 0.1)) +
    stat_summary(fun.y = mean, geom = "point", shape = 15, size = 2, color = "black") +
    facet_wrap(~ variable, nrow = 1) +
    xlab("") + ylab("Mean size")
```

facet_wrap au lieu de
facet_grid- une seule variable

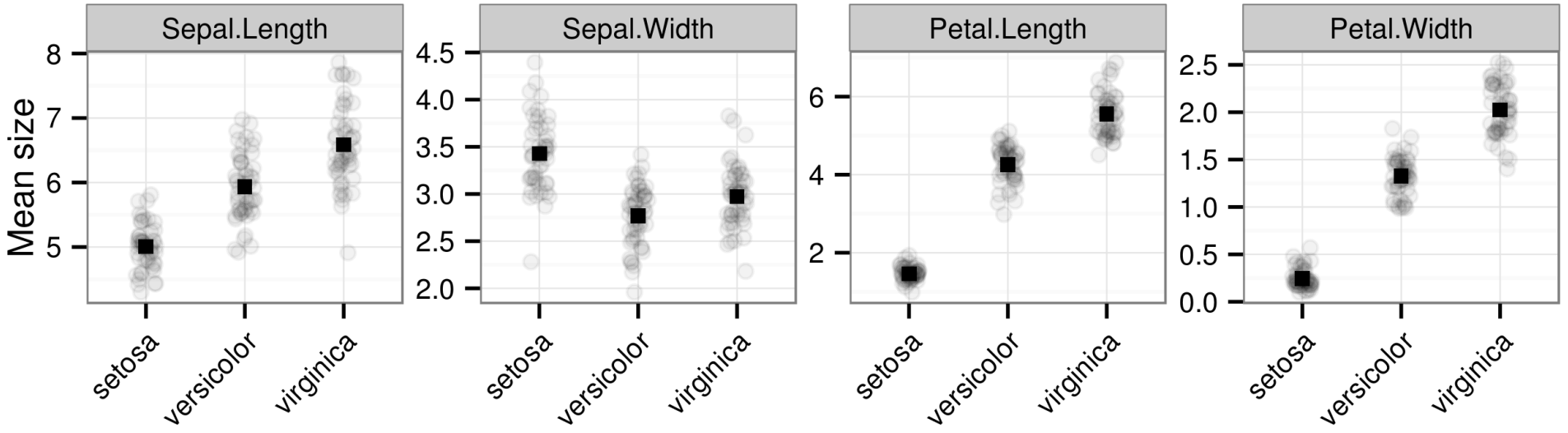


ggplot - exemples

```
ggplot(d, aes(x = Species, y = Size)) +
    geom_point(shape = 20, size = 3, alpha = 0.05,
               position = position_jitter(width = 0.1)) +
    stat_summary(fun.y = mean, geom = "point", shape = 15, size = 2, color = "black") +
    facet_wrap(~ variable, scales = "free", nrow = 1) +
    xlab("") + ylab("Mean size") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```

Echelle variable d'une facette à l'autre

étiquette des axes x avec une rotation de 45°



ggplot

Ressources pour aller plus loin :

Excellente synthèse en 1 page A3
"Cheatsheet" de Rstudio :
http://www.rstudio.com/wp-content/uploads/2015/11/ggplot2-cheatsheet.pdf

Aide en ligne de ggplot avec les graphiques !
http://docs.ggplot2.org/current/

Très bonne introduction bien structurée
(Syntaxe un peu dépassée)
http://sape.inf.usi.ch/quick-reference/ggplot2

Le livre de l'auteur du package
(Syntaxe un peu dépassée)
Wickham, H., 2009. ggplot2. Springer.

Le langage R

Contenu

Les objets
Importer des données

Extraire des données (subscripting)
Manipulation de dates et caractères

Fusionner des tableaux de données
R comme langage de script
(structures de contrôle : boucles, fonctions, etc...)
Agrégation et « reshaping »

Les graphiques
La création de rapports

R pour faire des rapports automatiques

Documents textes contenant du code R

Documents odt (Libre Office cfr .doc),
documents Latex, HTML, Markdown, etc...
Packages R : odfWeave, Sweave, knitr, rmarkdown

Le code est remplacé par le résultat de son exécution (y compris graphiques)
avec de nombreuses options.

Rapports générés directement par du code R

R2HTML : crée des rapports HTML, R2wd : vers MS Word
Généralement peu de texte, surtout des tableaux et des graphiques

Situation intermédiaire

Un script R comprend du texte et des commandes
--> transformé en document Latex, HTML, Markdown,...
Packages Knitr : fonctions stitch, spin

Litterate programming

Litterate programming paradigm

Mélanger le code source et la documentation sur ce code

"Let us change our traditional attitude to the construction of programs. Instead of imagining that our main task is to instruct a computer what to do, let us concentrate rather on explaining to human beings what we want a computer to do."

Donald Knuth - "Literate Programming" (1984)

On mélange dans le même document du texte ("*narratives*") décrivant le contexte, ce que fait le code, les résultats qu'il donne,... et du code qui fait des calculs/graphiques etc...

--> Pour obtenir le document final, on doit :

1) séparer le code du texte
2) exécuter le code et enregistrer les résultats
3) mélanger (*weave*) les résultats avec le texte

markdown + knitr + pandoc

Historiquement :
documents écrits en **LaTeX** + Package **Sweave**

Aujourd'hui : Très nombreuses possibilités

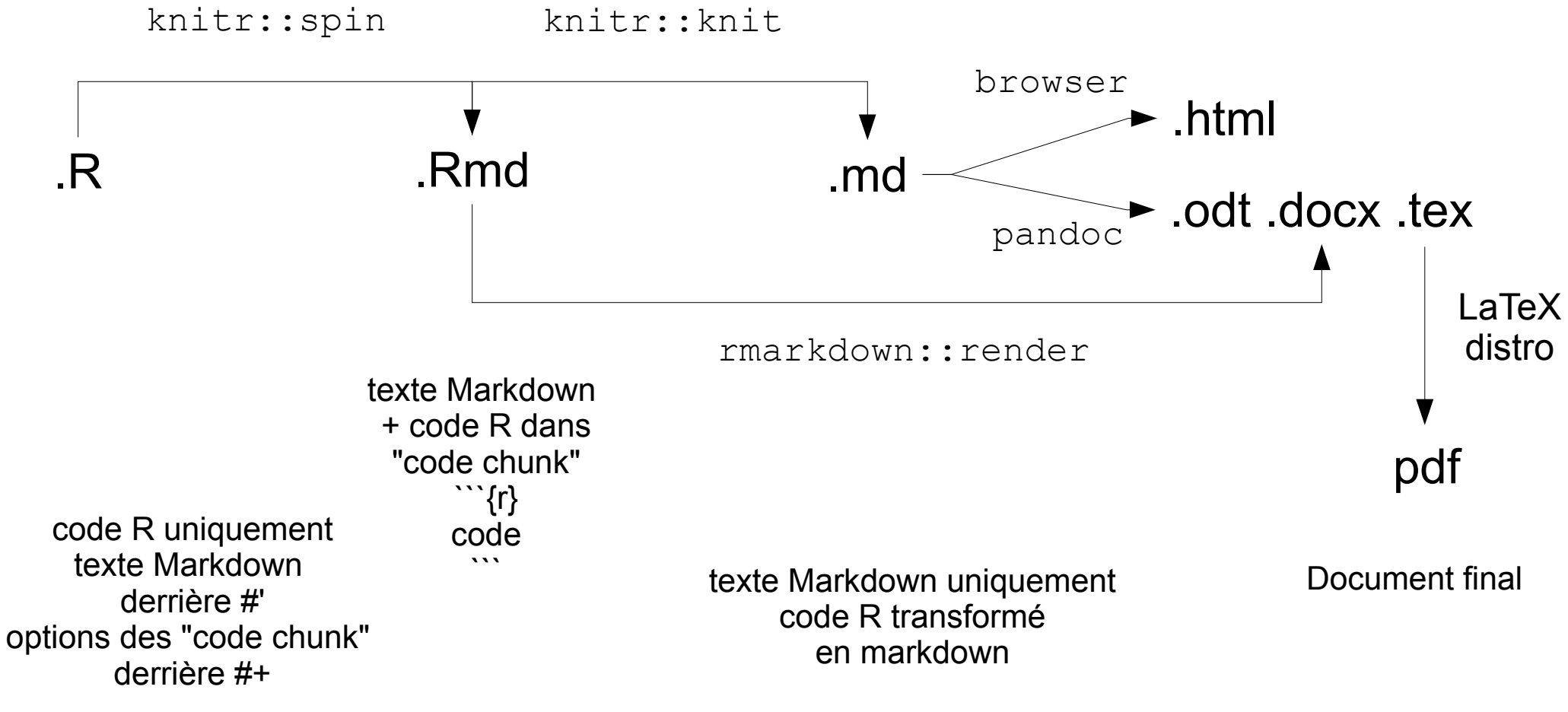
Un des meilleurs rapport
facilité d'usage et d'apprentissage / flexibilité :

Documents écrits en **Markdown** + Package **knitr**
--> transformés par le programme **pandoc** en de nombreux formats (.docx ; .odt, .html, pdf Latex,...)

Le package rmarkdown permet aujourd'hui de passer directement du markdown + code au document final sans passer directement par pandoc ou knitr

markdown + knitr + pandoc

On va détailler ce processus morceau par morceau...



markdown : fichiers .md

Langage de balise très simplifié

Les balises (symboles de texte) délimitent des zones de texte qui peuvent ensuite être mises en forme
(pex : titres, gras, italique, liste à puce, ...)

Documents de type texte simple (.txt) avec une extension .md

Très facile à écrire
Très facile à lire par un humain
Très facile à transformer dans d'autres formats
Très facile à apprendre

Couvre 90 % des besoins de mise en page
On peut intégrer des balises html ou LaTeX dans le markdown si on veut faire des choses plus compliquées
(mais on ne pourra alors transformer le .md que dans ces formats)

Pour écrire du Markdown dans Rstudio : File / New File / R Markdown

Markdown --> HTML

```
# Ceci est un titre de niveau 1

#### Ceci est un titre de niveau 4

**texte en gras** ou __texte en gras__
*texte en italique* ou _texte en italique_

Deux espaces à la fin d'une ligne
pour démarrer un nouveau paragraphe

Ligne horizontale :

*****************************

* liste à puce
    + sous-liste
    + sous-liste
* liste à puce

1. liste numérotée
    1. sous-liste
    1. sous liste
1. liste numérotée

Équation dans le texte :
$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$
Équation sur une ligne :
$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n}x_i$$

[Lien](https://www.codecogs.com/latex/eqneditor.php)
vers un éduteur d'équations en ligne

Code dans le texte : `read.table()`
Citation :

> Que la force soit avec toi !

image : ![](/home/gilles/divers/images/Rlogo.jpg)
```

Ceci est un titre de niveau 1

Ceci est un titre de niveau 4

texte en gras ou **texte en gras** *texte en italique* ou *texte en italique*

Deux espaces à la fin d'une ligne
pour démarrer un nouveau paragraphe

Ligne horizontale :

- liste à puce
  - sous-liste
  - sous-liste
- liste à puce

1. liste numérotée
    1. sous-liste
    2. sous liste
2. liste numérotée

Équation dans le texte : $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2$
Équation sur une ligne :

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n}x_i$$

Lien vers un éduteur d'équations en ligne

Code dans le texte : `read.table()`
Citation :

> Que la force soit avec toi !

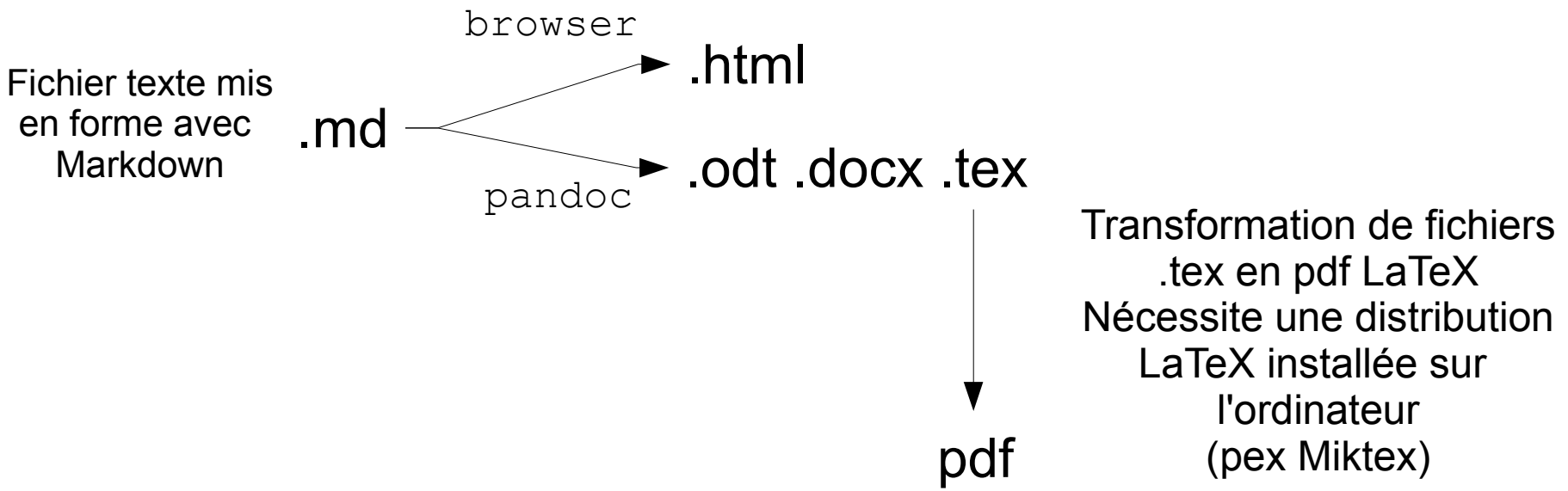
image : 

markdown - pandoc

Au départ prévu pour être transformé en HTML (pages web)

La plupart des navigateurs internet peuvent mettre en forme les fichiers .md (avec des modules complémentaires)

Le logiciel pandoc permet de les transformer en HTML et de nombreux autres formats
(voir plus loin comment)



Fichiers .Rmd : Markdown + Code

Dans un fichier markdown les blocs de code ("code chunks") commencent avec 3 "back ticks" ``` suivi d'un petit r entre acolades ({r}) et se terminent par 3 back ticks ```
On peut ajouter dans l'accolade un nom pour le bloc de code et des options (tous sont facultatifs)

````
```{r nom_du_chunk,  option1 = A, option2 = B}

mean(a)

```
````

Dans R studio, dans un fichier makdown, taper Ctrl+Alt+i pour insérer un code chunk vide

On peut aussi ajouter du code R au milieu du texte ("inline code").
Blablabla `` `r mean(a)` ``

Fichiers .Rmd : Markdown + Code

````
# Exemple de rapport Markdown

On utilise le jeu de données iris :
```{r}
data(iris)
summary(iris)
```

Il est composé de 4 variables :

* petal width
* petal length
* sepal width
* sepal lenght

La moyenne de la longueur des pétales est de `r round(mean(iris$Petal.Length), 2)` cm.

Voici la distribution des longueurs  de pétales :

```{r Petal_length_density, fig.width = 8/2.54, fig.height = 8/2.54}
par(mar = c(3,3, 2, 1))
plot(density(iris$Petal.Length), main = "Petal Length density")
```

# Environnement utilisé

```{r echo = FALSE}
sessionInfo()
```
````

Fichiers .Rmd : Markdown + Code

````
# Exemple de rapport Markdown

On utilise le jeu de données iris :
```{r}
data(iris)
summary(iris)
```

Il est composé de 4 variables :

* petal width
* petal length
* sepal width
* sepal lenght

La moyenne de la longueur des pétales est de `r round(mean(iris$Petal.Length), 2)` cm.

Voici la distribution des longueurs  de pétales :

```{r Petal_length_density, fig.width = 8/2.54, fig.height = 8/2.54, fig.align='center'}
par(mar = c(3,3, 2, 1))
plot(density(iris$Petal.Length), main = "Petal Length density")
```

# Environnement utilisé

```{r echo = FALSE}
sessionInfo()
```
````

"Code Chunk"

"Inline code"

Code Chunk name

Code Chunk options

Exemple de rapport Markdown

On utilise le jeu de données iris :

```
data(iris)
summary(iris)
```

```
##   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
##  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100
##  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300
##  Median :5.800   Median :3.000   Median :4.350   Median :1.300
##  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199
##  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
##  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
##        Species
##  setosa    :50
##  versicolor:50
##  virginica :50
##
##
##
```

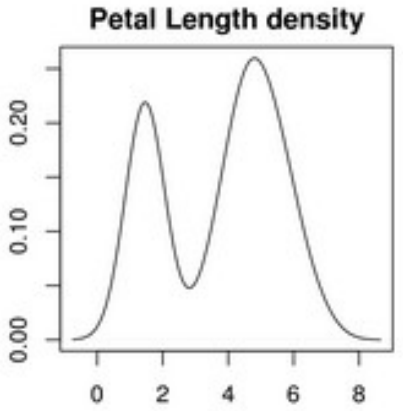
Il est composé de 4 variables :

- petal width
- petal length
- sepal width
- sepal lenght

La moyenne de la longueur des pétales est de 3.76 cm.

Voici la distribution des longueurs de pétales :

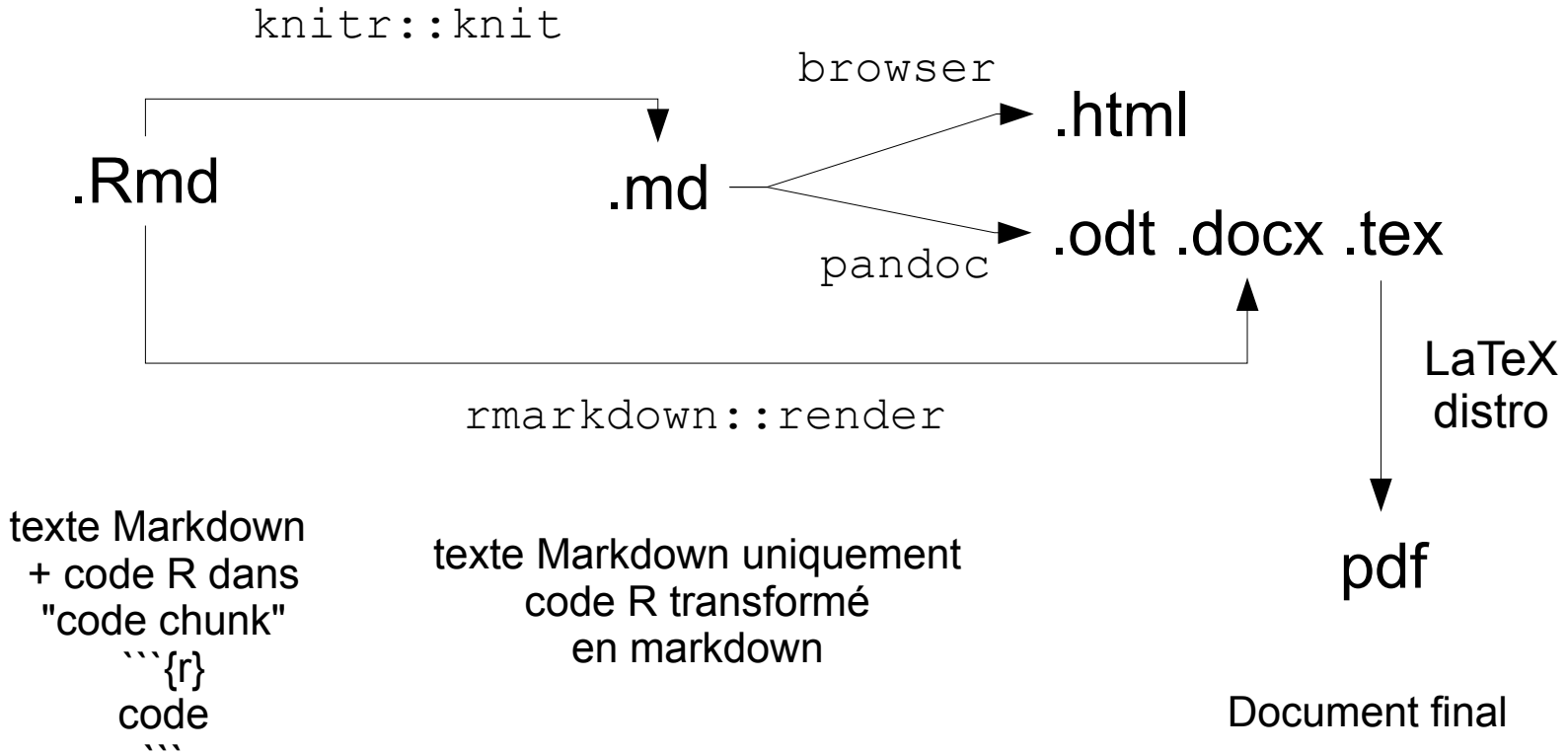
```
par(mar = c(3,3, 2, 1))
plot(density(iris$Petal.Length), main = "Petal Length density")
```



Environnement utilisé

compiler un .Rmd

Transformer le .Rmd en un autre format

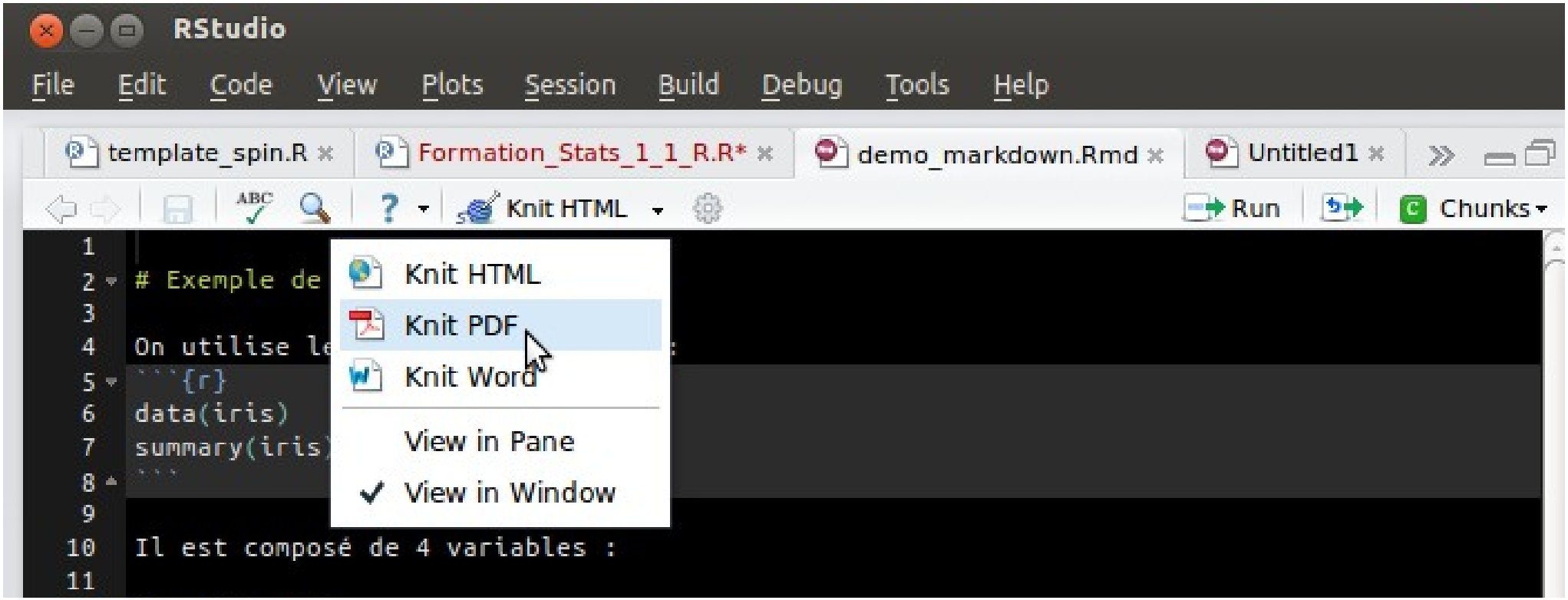


compiler un .Rmd

Transformer le .Rmd en un autre format - via R Studio

Dans R studio, utilisez le bouton Knit HTML
(ou ctrl + shift + K)

uniquement disponible si vous êtes dans un document de type markdown
File/New File / Rmarkdown pour créer un document markdown dans R studio



compiler un .Rmd

Transformer le .Rmd en .md avec knitr

A la base, c'est la fonction `knit` du package `knitr` qui va exécuter le code du fichier .Rmd puis incorporer les résultats dans le texte (weave) pour fournir un document .md complet

Il faut sauver son document .Rmd sur le disque puis :

```
library(knitr)
knit("/home/gilles/mondocument.Rmd")
```

knit2html peut directement convertir le .Rmd en html mais pas en d'autres formats :

```
knit2html("/home/gilles/mondocument.Rmd")
```

compiler un .Rmd

Transformer le .md en un autre format avec pandoc

Une fois qu'on a un fichier .md, on peut utiliser pandoc (à installer séparément) pour le transformer en d'autres formats

NB : pour créer des pdfs LaTeX, il faut installer aussi une distribution LaTeX (pex MikTeX sous windows)

On peut commander pandoc depuis R par exemple comme ceci :

```
system("pandoc mondocument.md -o mondocument.odt")
system("pandoc mondocument.md -o mondocument.pdf")
system("pandoc mondocument.md -o mondocument.html")
system("pandoc mondocument.md -o mondocument.docx")
```

compiler un .Rmd

Transformer le .md en un autre format avec pandoc

Il existe de nombreuses options (voir http://pandoc.org/README.html)
Voici celles que j'utilise :

```
system("pandoc mondocument.md -o mondocument.odt
--reference-odt=template.odt")

   system("pandoc mondocument.md -o mondocument.html -s --self-contained --toc
--toc-depth=2 --mathjax --highlight-style kate  -c template.css")


system("pandoc mondocument.md -o mondocument.tex -s -S --toc -V
geometry:margin=0.7in --highlight-style=tango --latex-engine=xelatex")

system("pdflatex mondocument.tex")
system("pdflatex mondocument.tex")
```

NB : on Compile 2 fois le fichier .tex pour avoir une table des matières

NB : pour les .html et .odt, on peut se baser sur une feuille de style .css ou un document .odt avec des styles pour mettre en forme les documents

compiler un .Rmd

Transformer le .Rmd directement en un autre format
rmarkdown ::render

Aujourd'hui, une solution tout en un est disponible :

La fonction `render` du package `rmarkdown`
utilise `knit` et `pandoc` pour convertir
un .Rmd en .html, .docx ou .pdf (si on a installé LaTeX)

Il faut sauver son document .Rmd sur le disque puis :

```
library(rmarkdown)
render("/home/gilles/demo_markdown.Rmd",
    output_format = c("html_document", "pdf_document" ))
```

Markdown + Code

Options des blocs de code

très nombreuses possibilités : http://yihui.name/knitr/options/

`echo = TRUE` :

affiche ou pas (FALSE) le code

`dev = c("cairo_pdf", "png")` :

sauve les graphiques en pdf et png

`dpi = 300` :

résolution des graphiques (NB : pour des sorties HTML garder `dpi = 96`)

`fig.width = 8/2.54, fig.height = 8/2.54` :

largeur et hauteur par défaut des figures de 8 cm

`warning = FALSE, error = FALSE, message = FALSE` :

ne pas afficher les messages d'errueur, d'avertissement, etc...

Markdown + Code

opts_chunk$set

On peut changer les valeurs par défaut des options pour la suite d'un document en mettant dans un code chunk pex :

```
library(knitr)
opts_chunk$set(dev = c("png", "cairo_pdf"), dpi = 300,
fig.width = 10/2.54, fig.height= 10/2.54, echo=TRUE, cache = TRUE)
```

Markdown + Code

YAML

Entête structurée que l'on place en début de document .Rmd et qui permet de définir certaines options mais aussi le titre du document, l'auteur, etc...

Une entête YAML basique comme la suivante entraînera la création d'un fichier .pdf lors d'une compilation avec R Studio ou avec `rmarkdown::render`

```
---
title: 'Example report with iris data'
author: 'Gilles San Martin'
date: '01 January 2015'
output: pdf_document
---
```

Markdown + Code

YAML

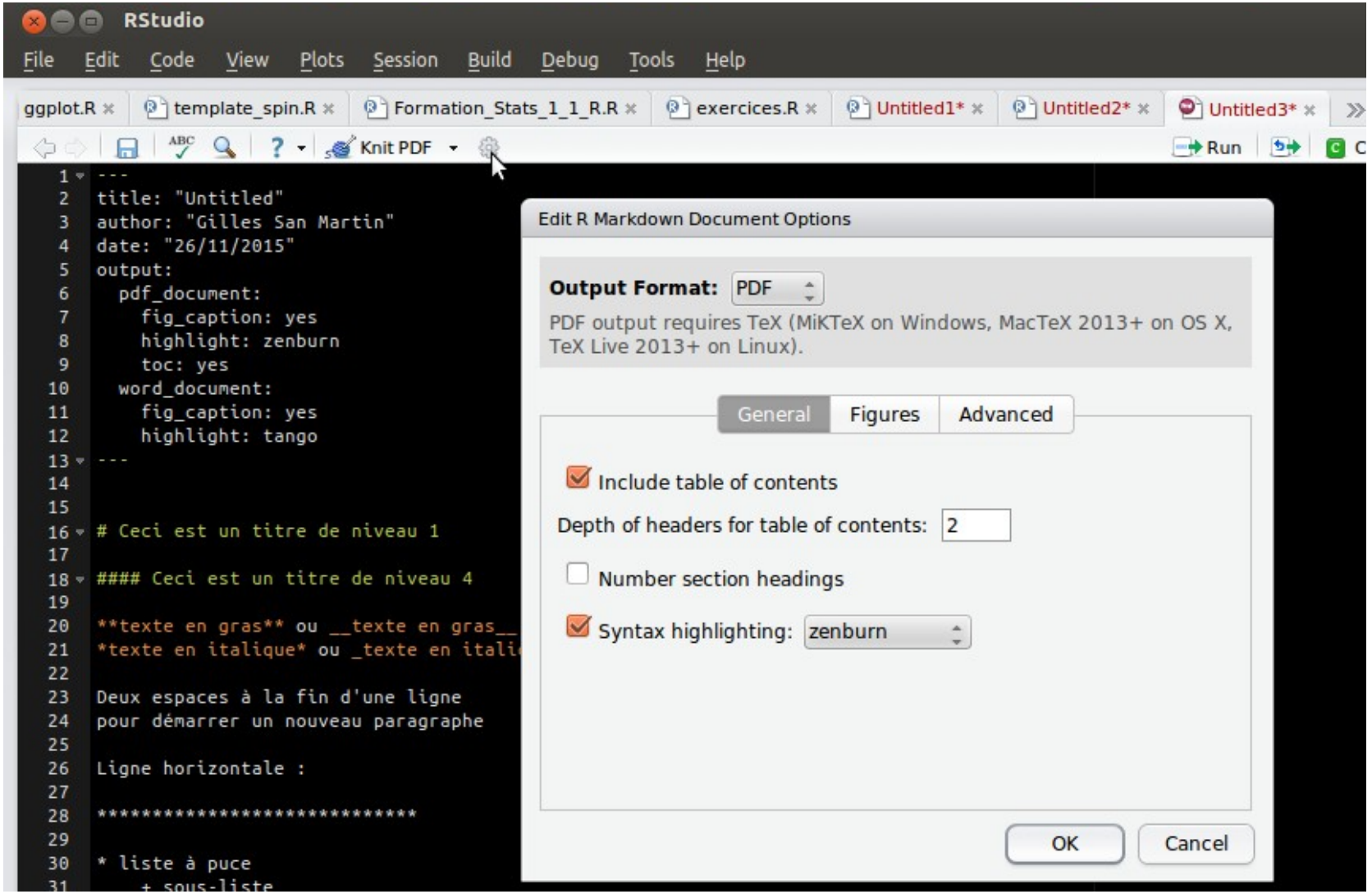
Entête YAML et options knitr utilisées pour générer des pdf LaTeX :

```
---
title: 'Example report with iris data'
author: 'Gilles San Martin'
date: '`r Sys.setlocale("LC_ALL", "en_GB.UTF-8"); format(Sys.time(), "%d %B %Y")`'
output:
     pdf_document:
         toc: true
         fig_caption: true
         highlight: tango
geometry: margin=0.7in
header-includes:
     - \usepackage{float}
     - \usepackage{fancyvrb}
     - \usepackage{graphicx} # needed for fig.align = 'center' option
---

library(knitr)
opts_chunk$set(dev = c("png", "cairo_pdf"), fig.width = 10/2.54, fig.height= 10/2.54,
               fig.align='center', dpi = 300,
               echo=TRUE, results= "markup", warning = FALSE,
               error = FALSE, message = FALSE, cache = TRUE, tidy = FALSE,
               tidy.opts=list(width.cutoff=95))
```

Markdown + Code

YAML

On peut définir une partie des options de l'entête YAML via l'interface graphique de R Studio



Markdown + Code

`pander` - tables

Il est possible de créer des tables en markdown
La fonction `pander` du package `pander` :

```
> library(pander)
> data(iris)
> pander(iris[1:5,])
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width
5.1	3.5	1.4	0.2
4.9	3	1.4	0.2
4.7	3.2	1.3	0.2
4.6	3.1	1.5	0.2
5	3.6	1.4	0.2

Table: Table continues below

Species
setosa

Markdown + Code

`pander` - tables

On peut changer les options de pander :

```
> library(pander)
> panderOptions("table.style" , "rmarkdown")
> panderOptions("table.split.table" , Inf)
> panderOptions("table.alignment.rownames" , "right")

> pander(iris[1:5,])
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa

Markdown + Code

`pander` - tables

Résultat dans le document final :

```
pander(iris[1:5,])
```

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5	3.6	1.4	0.2	setosa

```
iris[1:5,]
```

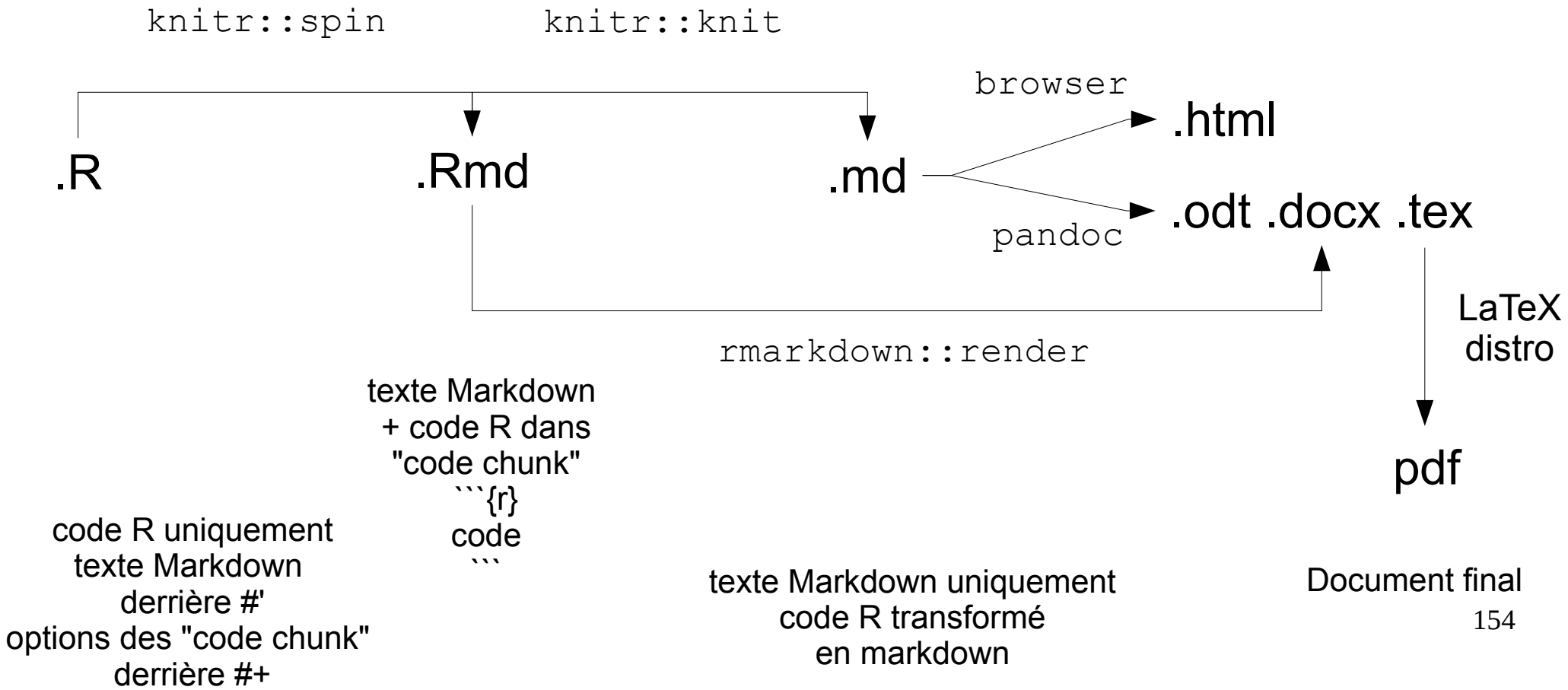
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5  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

Markdown + Code

Spin

Il est aussi possible partir d'un simple fichier R puis de le transformer en fichier .Rmd avec la fonction `spin` du package `knitr`

Idéal quand on a relativement peu de texte (mon approche de prédilection)



Markdown + Code

Spin

Le texte markdown se trouve derrière des "roxygen comment" : #'
Les options de code chunk sont derrière : #+
Les code chunks ne sont pas explicitement définis

```
# /*
# ---------------- Graphs -------------------
# */

#' # Graphical representation of the data
#'
#' ## Scatterplot matrix

#+ fig.width = 12/2.54, fig.height = 8/2.54
pairs(d, gap = 0)

#' ## Boxplot
#'
#+ fig.width = 7.5/2.54, fig.height = 7/2.54
# this is a simple R comment
par(mar = c(3,3,2,1), mgp = c(1.8, 0.5, 0), las = 1, cex = 0.8)
plot(Sepal.Length ~ Species, data=d)
```

Les commentaires compris entre /* et */ ne se rtrouveront pas dans le document final

Texte markdown derrière #'

Options du code chunk derrière #+

Commentaire R classique

Code R

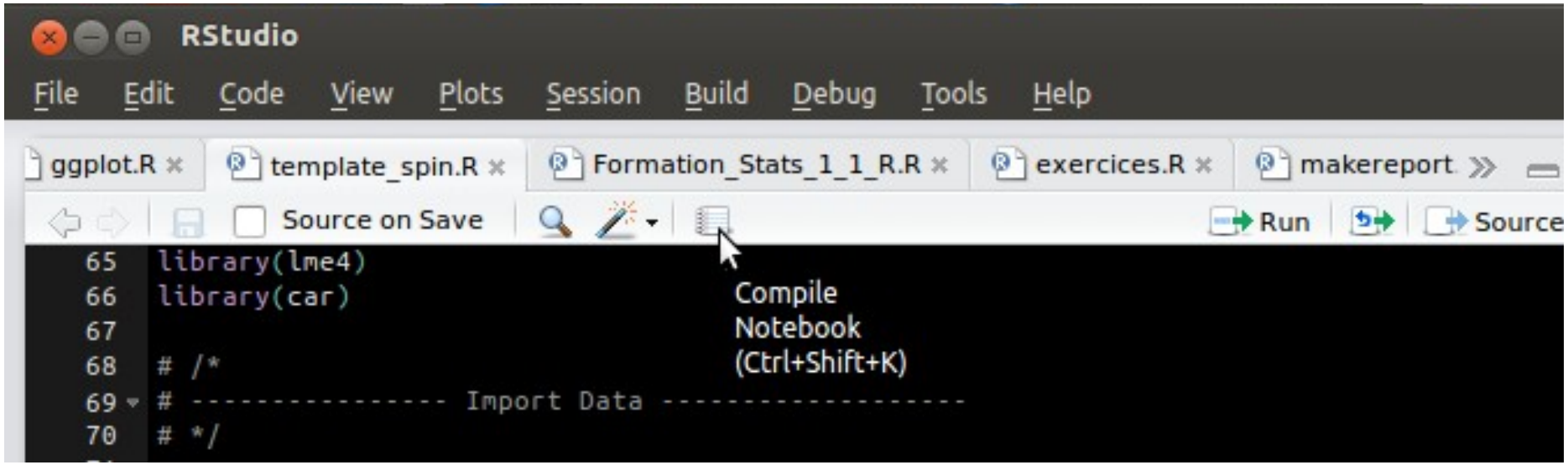
Markdown + Code

Spin

On peut transformer le script .R en .Rmd en ligne de commande :

```
library(knitr)
spin("/home/gilles/mondocument.R", knit = FALSE)
knit("/home/gilles/mondocument.Rmd")
```

Ou via le bouton "compile notebook" de R studio (ou Ctrl+Shift+K) :



Markdown + Code

knit_child

Permet d'insérer dans un document parent le contenu de documents "enfants".
Permet du coup de répéter le même rapport sur des sous-ensembles de données quand on l'utilise dans une boucle.

Document parent (type spin) :

```
#' # Description of each variable
#'
#+ echo = FALSE, results = "hide"
# Here we compile each subreport template with knit_child that returns the results as
# markdown. Then the result is displayed in the next chunk with result = "asis" option
d <- data(iris)
res <- NULL
for (i in 1:4) {
    var <- d[,i]
    varname <- colnames(d)[i]
    res <- c(res, knit_child("subreport_child.Rmd"))
}
```

`echo = FALSE` : le code ne sera pas affiché
`results = "hide"` : le code sera exécuté pas pas affiché

```
#+ results = "asis"
cat (res)
```

`results = "asis"` : le résultat de `cat` sera considéré comme du texte markdown (tel quel) et pas comme une sortie R

Markdown + Code

knit_child

Document enfant (type .Rmd) qui sera répété pour chacune des 4 variables dans le document parent.

````
## `r varname`

The mean `r varname` is `r mean(var)` cm.

### Density

```{r fig.width = 7.5/2.54, fig.height = 7/2.54}
par(mar = c(3,3,2,1), mgp = c(1.8, 0.6, 0), cex = 0.8)
plot(density(var), xlab = varname, main = "")
```


### Boxplot

```{r fig.width = 7.5/2.54, fig.height = 7/2.54}
par(mar = c(3,3,2,1), mgp = c(1.8, 0.6, 0), cex = 0.8)
boxplot(var, xlab = varname, main = "", horizontal = TRUE)
```
````

Markdown + Code

`sessionInfo()`

A la fin d'un rapport il est utile d'inclure les sorties da la fonction de Base `sessionInfo()` qui décrit les packages utilisés, leur version, la plate-forme utilisée

```
> sessionInfo()

R version 3.2.2 (2015-08-14)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu precise (12.04.5 LTS)

locale:
 [1] LC_CTYPE=fr_BE.UTF-8       LC_NUMERIC=C                   LC_TIME=fr_BE.UTF-8
 [4] LC_COLLATE=fr_BE.UTF-8     LC_MONETARY=fr_BE.UTF-8        LC_MESSAGES=fr_BE.UTF-8
 [7] LC_PAPER=fr_BE.UTF-8       LC_NAME=C                      LC_ADDRESS=C
[10] LC_TELEPHONE=C             LC_MEASUREMENT=fr_BE.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

loaded via a namespace (and not attached):
[1] htmltools_0.2.6 tools_3.2.2     yaml_2.1.13     rmarkdown_0.6.1 knitr_1.10.5
digest_0.6.3
```

Markdown + Code

citation()

A la fin d'un rapport il est utile aussi de donner les références de R et des principaux packages utilisés grâce à la fonction citation()

```
> #+ echo = FALSE
> opts_chunk$set(results= "asis", echo = FALSE)
>
> #'
> print(citation("lme4"), style = "text")
Bates D, Mächler M, Bolker B and Walker S (2015). "Fitting Linear Mixed-Effects Models
Using lme4." _Journal of Statistical Software_, *67*(1), pp. 1-48. <URL:
http://doi.org/10.18637/jss.v067.i01>.
> #'
> print(citation("car"), style = "text")
Fox J and Weisberg S (2011). _An R Companion to Applied Regression_, Second edition. Sage,
Thousand Oaks CA. <URL: http://socserv.socsci.mcmaster.ca/jfox/Books/Companion>.
> #'
> print(citation("multcomp"), style = "text")
Hothorn T, Bretz F and Westfall P (2008). "Simultaneous Inference in General Parametric
Models." _Biometrical Journal_, *50*(3), pp. 346-363.
> #'
> print(citation(), style = "text")
R Core Team (2015). _R: A Language and Environment for Statistical Computing_. R
Foundation for Statistical Computing, Vienna, Austria. <URL: https://www.R-project.org/>.
>
> opts_chunk$set(results= "markup", echo = TRUE)
```

Markdown + Code

Ressources

Le site de knitr :
http://yihui.name/knitr/

Le site de R Markdown
http://rmarkdown.rstudio.com/

Les pense-bête de R Studio (options, syntaxe, etc...) :
https://www.rstudio.com/resources/cheatsheets/