Capsule 7

Objectifs de la capsule

À la fin de cette capsule, vous serez en mesure de:

1. Utiliser les options les plus courantes pour chaque type de graphique.
2. Ajouter des couleurs.
3. Ajouter des points et des lignes (abline).
4. Ajouter du texte.
5. Ajouter une légende.
6. Combiner plusieurs graphiques dans un seul.

Capsule vidéo

https://youtu.be/Jdf5KX4jn-E

Exercices

> **Note**
>
> Veuillez noter qu'il est possible d'avoir plus d'une bonne réponse par question. Vous pouvez reprendre chaque exercice grâce aux boutons "Start Over". Le bouton "Indice" est là pour être utilisé!

> **Note**

> **Note**

> **Note**

> **Note**

> **Note**

> **Note**

> **Note**

> **Note**

> **Note**

> **Note**

Matériel accompagnateur

Modifier le format par défaut des graphiques: pourquoi?

Une fois que nous avons un graphique, il est souvent avantageux de modifier le format par défaut, et ce pour plusieurs raisons. Premièrement, nous voulons présenter les données de la façon la plus claire possible, ce qui n'est pas toujours le cas avec les valeurs par défaut. Par exemple, des points peuvent se chevaucher sur un graphique de nuages de points, ou le nombre de classes par défaut d'un histogramme peuvent ne pas présenter adéquatement la distribution de fréquence des données. Deuxièmement, les standards requis pour inclure un graphique dans un travail ou une publication ne sont pas toujours atteints avec les valeurs par défaut. En effet, nous voulons avoir le titre et les axes les plus clairs possibles, sans avoir à faire des changements manuellement plus tard dans un autre logiciel.

Ces paramètres qui précisent les options de visualisation doivent être ajoutées à la fonction de base que nous tapons pour obtenir un graphique.

Si on veut modifier le format des symboles d'un graphique à nuage de points, on invoque la fonction de base plot() et on y ajoute les options, ici pour la forme des symboles pch.

```
plot(x, y, pch = 16)
```

Nous verrons des exemples avec les graphiques de base que vous avez déjà explorés, soit l'histogramme, le nuage de point, le graphique à moustache et le barplot.

La puissance de la fonction d'aide

Il n'y a pas d'avantage à retenir par coeur toutes les options qui s'offrent à vous pour la modification d'un graphique dans R. Faites comme les pros et servez-vous de la fonction d'aide pour aller chercher les paramètres optionnels:

```
# Pour connaître les paramètres pour la fonction histogramme
?hist
```

Files Plots Packages Help Viewer

R: Histograms - Find in Topic

hist {graphics} R Documentation

Histograms

Description

The generic function `hist` computes a histogram of the given data values. If `plot = TRUE`, the resulting object of class `"histogram"` is plotted by `plot.histogram`, before it is returned.

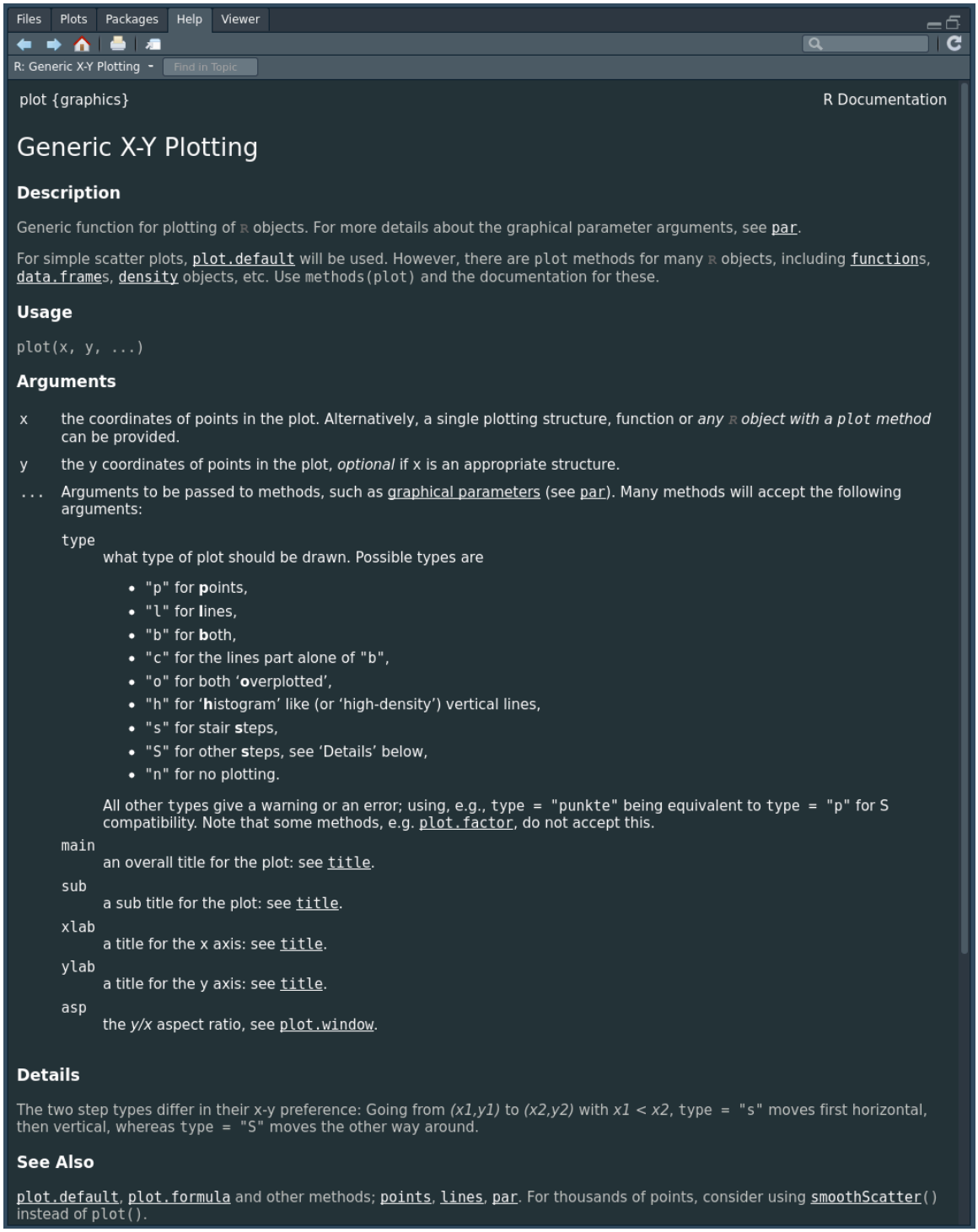
Usage

```
hist(x, ...)

## Default S3 method:
hist(x, breaks = "Sturges",
     freq = NULL, probability = !freq,
     include.lowest = TRUE, right = TRUE,
     density = NULL, angle = 45, col = NULL, border = NULL,
     main = paste("Histogram of" , xname),
     xlim = range(breaks), ylim = NULL,
     xlab = xname, ylab,
     axes = TRUE, plot = TRUE, labels = FALSE,
     nclass = NULL, warn.unused = TRUE, ...)
```

Arguments

`x`	a vector of values for which the histogram is desired.
`breaks`	one of: • a vector giving the breakpoints between histogram cells, • a function to compute the vector of breakpoints, • a single number giving the number of cells for the histogram, • a character string naming an algorithm to compute the number of cells (see 'Details'), • a function to compute the number of cells. In the last three cases the number is a suggestion only; as the breakpoints will be set to `pretty` values, the number is limited to `1e6` (with a warning if it was larger). If `breaks` is a function, the `x` vector is supplied to it as the only argument (and the number of breaks is only limited by the amount of available memory).
`freq`	logical; if TRUE, the histogram graphic is a representation of frequencies, the `counts` component of the result; if FALSE, probability densities, component `density`, are plotted (so that the histogram has a total area of one). Defaults to TRUE *if and only if* `breaks` are equidistant (and `probability` is not specified).
`probability`	an *alias* for `!freq`, for S compatibility.
`include.lowest`	logical; if TRUE, an `x[i]` equal to the `breaks` value will be included in the first (or last, for `right = FALSE`) bar. This will be ignored (with a warning) unless `breaks` is a vector.
`right`	logical; if TRUE, the histogram cells are right-closed (left open) intervals.
`density`	the density of shading lines, in lines per inch. The default value of `NULL` means that no shading lines are drawn. Non-positive values of `density` also inhibit the drawing of shading lines.
`angle`	the slope of shading lines, given as an angle in degrees (counter-clockwise).
`col`	a colour to be used to fill the bars. The default of `NULL` yields unfilled bars.
`border`	the color of the border around the bars. The default is to use the standard foreground color.
`main, xlab, ylab`	these arguments to `title` have useful defaults here.
`xlim, ylim`	the range of `x` and `y` values with sensible defaults. Note that `xlim` is *not* used to define the histogram (breaks), but only for plotting (when `plot = TRUE`).
`axes`	logical. If TRUE (default), axes are draw if the plot is drawn.
`plot`	logical. If TRUE (default), a histogram is plotted, otherwise a list of breaks and counts is returned. In the latter

```
# Pour connaître les paramètres des graphiques en nuage de points
?plot
```



Files Plots Packages Help Viewer

R: Generic X-Y Plotting — Find in Topic

plot {graphics} R Documentation

Generic X-Y Plotting

Description

Generic function for plotting of R objects. For more details about the graphical parameter arguments, see par.

For simple scatter plots, plot.default will be used. However, there are plot methods for many R objects, including functions, data.frames, density objects, etc. Use methods(plot) and the documentation for these.

Usage

```
plot(x, y, ...)
```

Arguments

x	the coordinates of points in the plot. Alternatively, a single plotting structure, function or *any* R *object with a* plot *method* can be provided.
y	the y coordinates of points in the plot, *optional* if x is an appropriate structure.
...	Arguments to be passed to methods, such as graphical parameters (see par). Many methods will accept the following arguments:

type
: what type of plot should be drawn. Possible types are

- "p" for **p**oints,
- "l" for **l**ines,
- "b" for **b**oth,
- "c" for the lines part alone of "b",
- "o" for both '**o**verplotted',
- "h" for '**h**istogram' like (or 'high-density') vertical lines,
- "s" for stair **s**teps,
- "S" for other **s**teps, see 'Details' below,
- "n" for no plotting.

All other types give a warning or an error; using, e.g., type = "punkte" being equivalent to type = "p" for S compatibility. Note that some methods, e.g. plot.factor, do not accept this.

main
: an overall title for the plot: see title.

sub
: a sub title for the plot: see title.

xlab
: a title for the x axis: see title.

ylab
: a title for the y axis: see title.

asp
: the *y/x* aspect ratio, see plot.window.

Details

The two step types differ in their x-y preference: Going from *(x1,y1)* to *(x2,y2)* with *x1 < x2*, type = "s" moves first horizontal, then vertical, whereas type = "S" moves the other way around.

See Also

plot.default, plot.formula and other methods; points, lines, par. For thousands of points, consider using smoothScatter() instead of plot().

Le cas spécial des couleurs

R nous donne la possibilité de choisir parmi un grand nombre de couleurs pour nos graphiques. On utilise l'argument `col = "nom_de_la_couleur"` pour préciser notre choix (en anglais: "blue",

“red”, etc.). La palette disponible est assez large. Les noms des couleurs possibles sont présentés dans le document *Colors* in R à l’adresse suivante:

- http://www.stat.columbia.edu/~tzheng/files/Rcolor.pdf

On peut donc utiliser `col = "darkturquoise"` ou `col = "darkorange"` et sortir des sentiers battus!

Une façon plus avancée de choisir une palette de couleurs est d’utiliser la librairie `RColorBrewer`. Si cette option vous intéresse, il faut tout d’abord installer la librairie dans R.

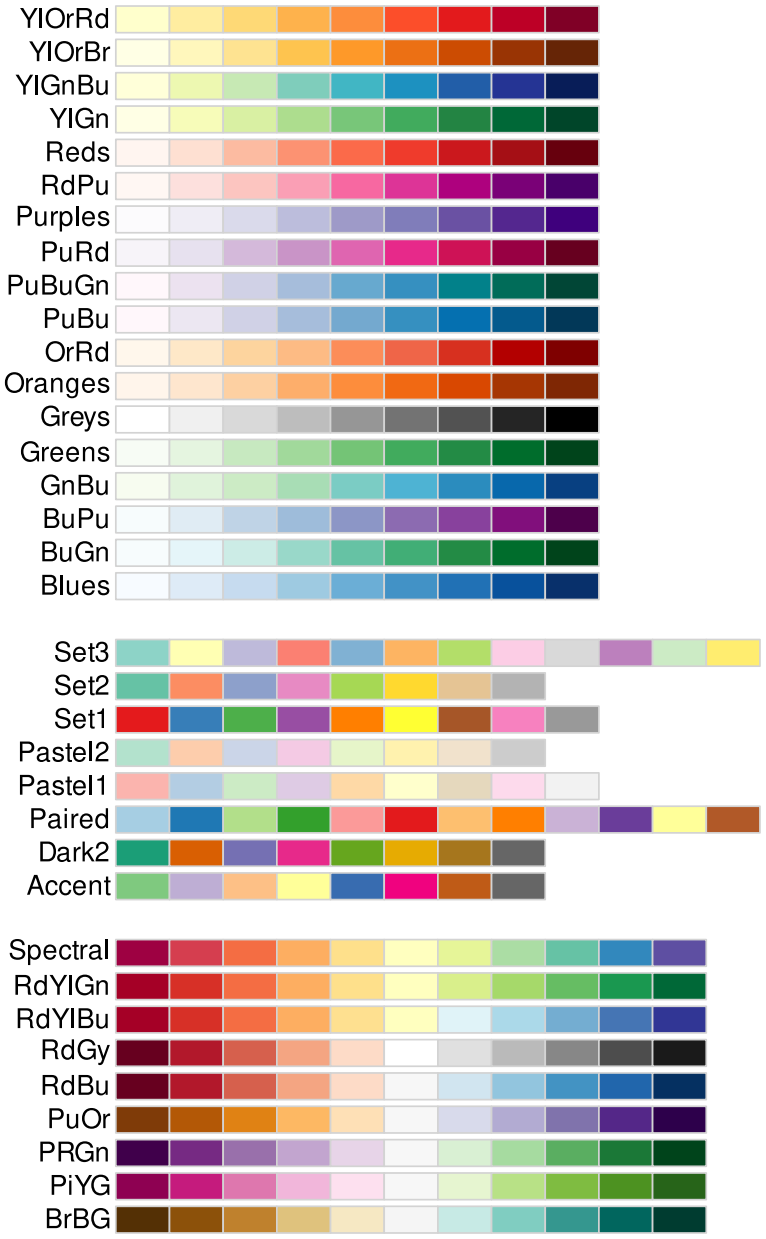
```
# Installer la librairie
install.packages("RColorBrewer")
```

Il faut ensuite la charger dans notre session de R lorsqu’on veut s’en servir.

```
library(RColorBrewer)
```

On peut ensuite voir les choix de palette de couleurs et choisir le nom de la palette à utiliser

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


YlOrRd
YlOrBr
YlGnBu
YlGn
Reds
RdPu
Purples
PuRd
PuBuGn
PuBu
OrRd
Oranges
Greys
Greens
GnBu
BuPu
BuGn
Blues
Set3
Set2
Set1
Pastel2
Pastel1
Paired
Dark2
Accent
Spectral
RdYlGn
RdYlBu
RdGy
RdBu
PuOr
PRGn
PiYG
BrBG

Utilisez help(RColorBrewer) pour en apprendre plus sur son utilisation.

Les histogrammes

Nous allons tester différentes modifications aux graphiques de base en utilisant le jeu de données Loblolly, qui est déjà pré-téléchargé dans la version de base de R. Celui-ci contient les données de taille (height, en pieds, une variable numérique) et d'âge (age, en années, une autre variable numérique) pour des pins (*Pinus taeda*) provenant de différents parents (Seed, une variable de type "facteur" où le chiffre représente la paire d'arbres parentaux).

On importe le jeu de données avec la fonction data().

```
data("Loblolly")
```

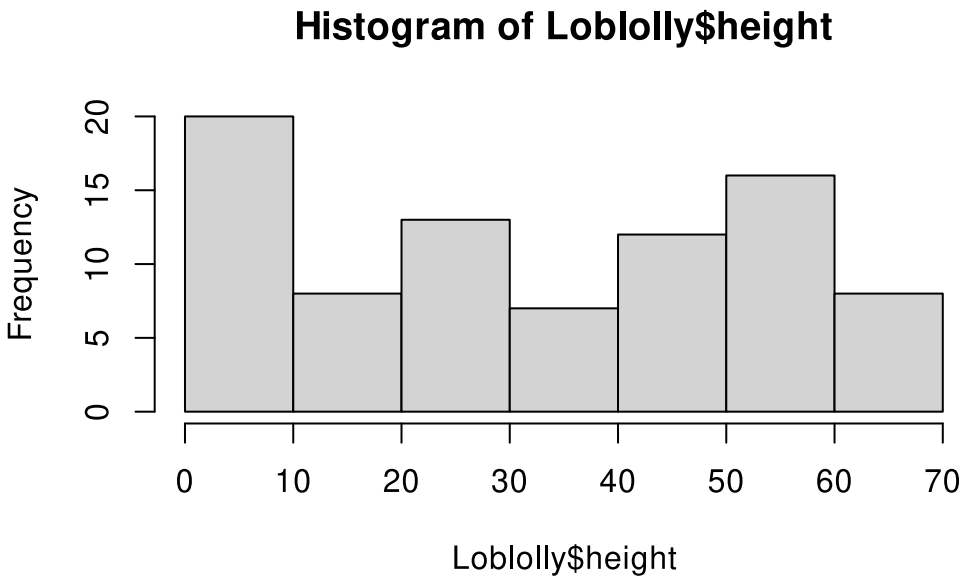
On peut utiliser la fonction head() pour vérifier le contenu de notre jeu de données.

```
head(Loblolly)
```

```
   height age Seed
1    4.51   3  301
15  10.89   5  301
29  28.72  10  301
43  41.74  15  301
57  52.70  20  301
71  60.92  25  301
```

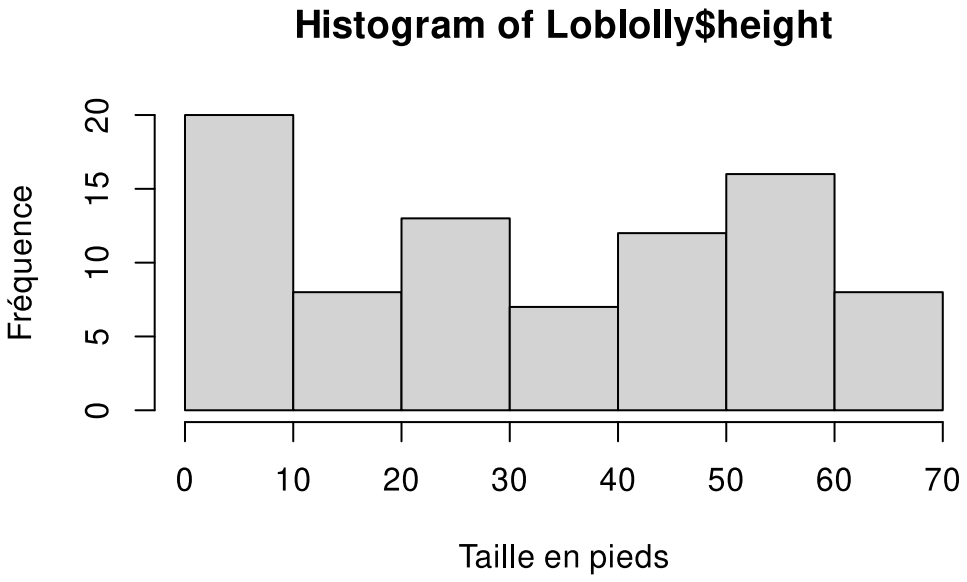
Nous allons créer un histogramme pour ensuite le modifier et illustrer les différentes options qui s'offrent à nous.

```
# Construire un histogramme de base avec la taille des arbres
hist(Loblolly$height)
```



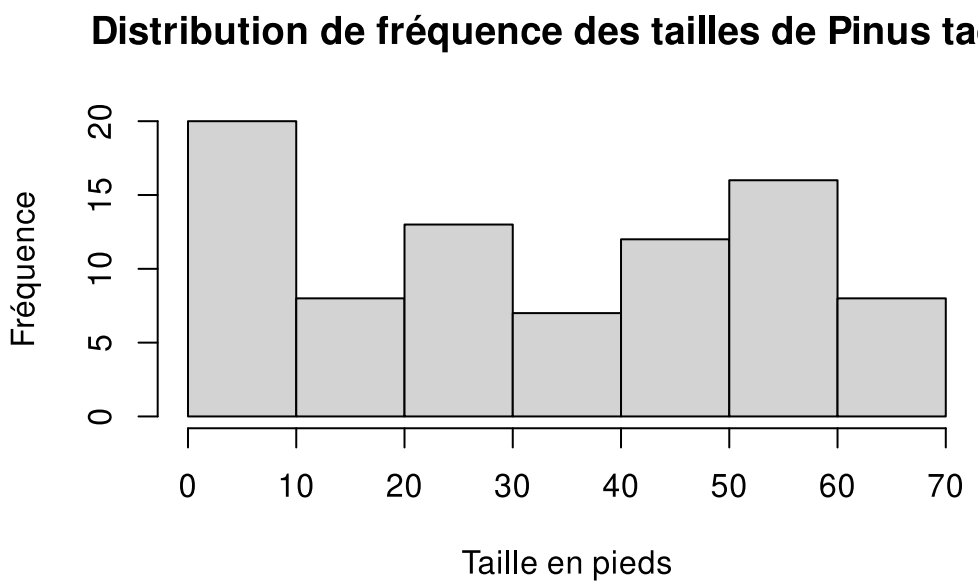
On peut modifier cet histogramme présentant la distribution de fréquence des tailles en améliorant les étiquettes des axes avec les paramètres xlab pour l'étiquette des abscisses et ylab pour l'étiquette des ordonnées.

```
hist(Loblolly$height, xlab = "Taille en pieds", ylab = "Fréquence")
```



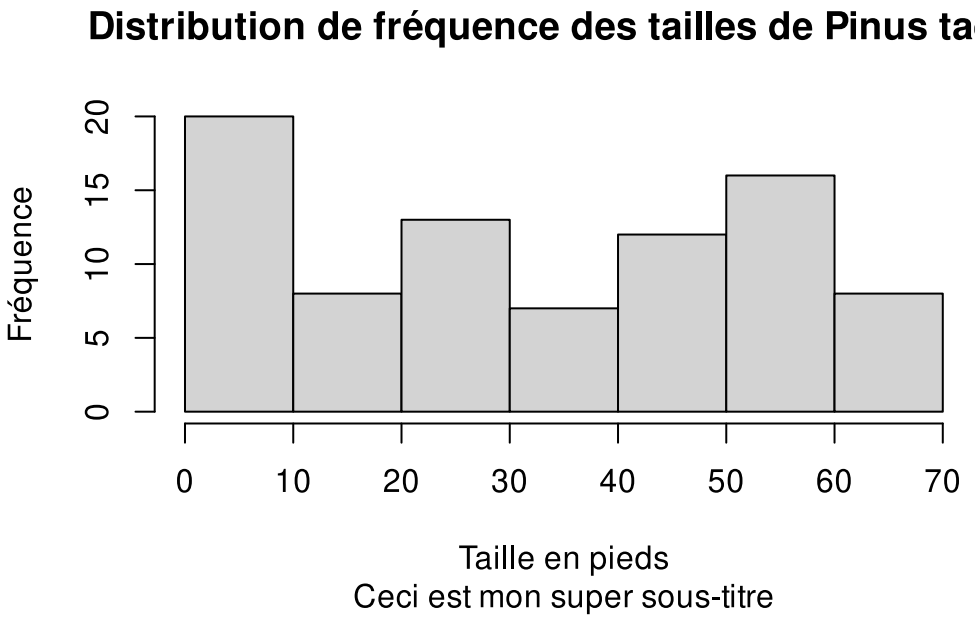
Le titre de notre histogramme peut aussi être modifié avec le paramètre main.

```
hist(
  Loblolly$height,
  xlab = "Taille en pieds",
  ylab = "Fréquence",
  main = "Distribution de fréquence des tailles de Pinus taeda"
)
```



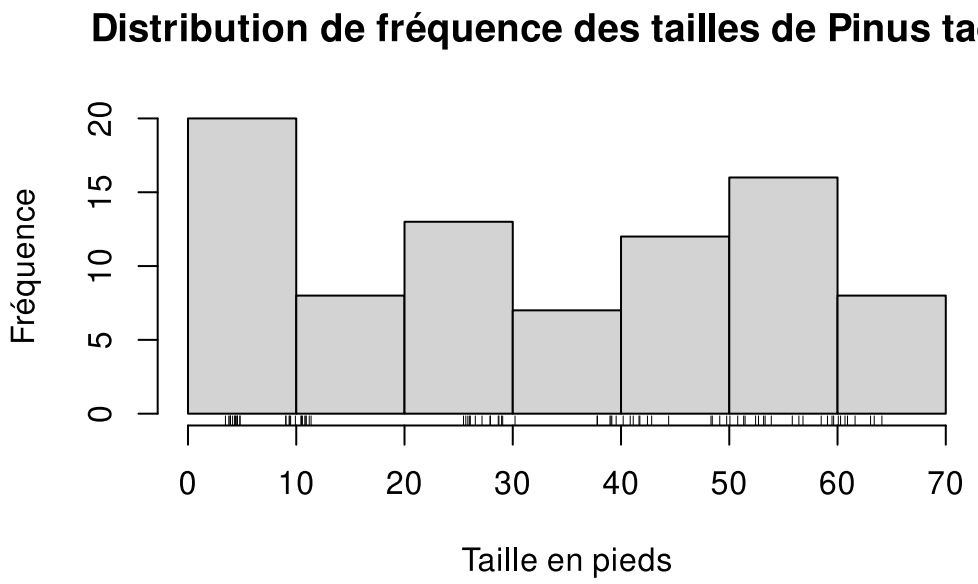
On peut également ajouter un sous-titre avec le paramètre sub.

```
hist(
  Loblolly$height,
  xlab = "Taille en pieds",
  ylab = "Fréquence",
  main = "Distribution de fréquence des tailles de Pinus taeda",
  sub = "Ceci est mon super sous-titre"
)
```



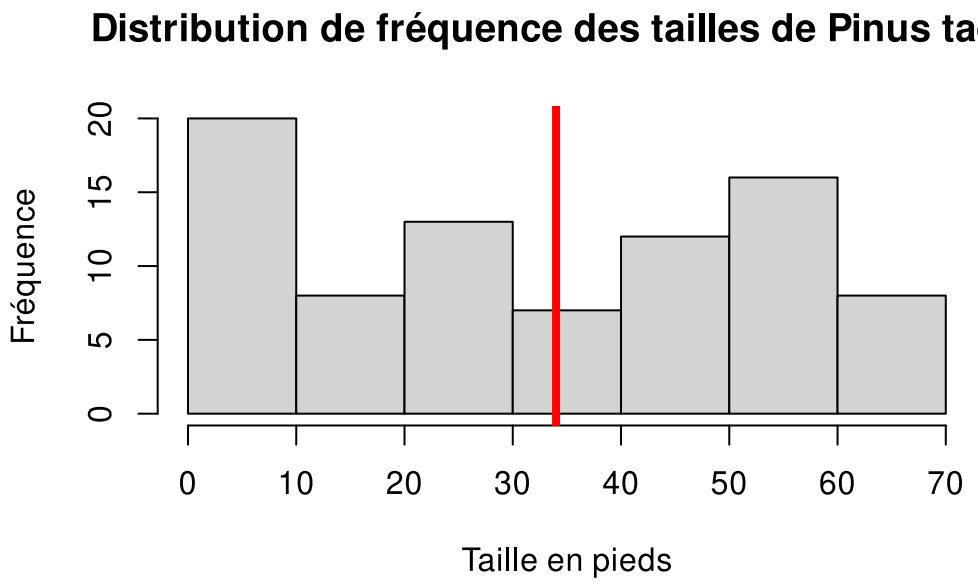
On peut ajouter la distribution de chacune des valeurs du jeu de données pour cette variable avec la fonction rug() invoquée après avoir déjà créé l'histogramme.

```
hist(
  Loblolly$height,
  xlab = "Taille en pieds",
  ylab = "Fréquence",
  main = "Distribution de fréquence des tailles de Pinus taeda"
)

rug(Loblolly$height)
```



On peut finalement ajouter la valeur de la médiane à l'aide de la fonction abline(). On peut spécifier la valeur utilisée pour tracer la ligne verticale avec le paramètre v, la couleur de la ligne verticale avec col et sa largeur avec lwd.

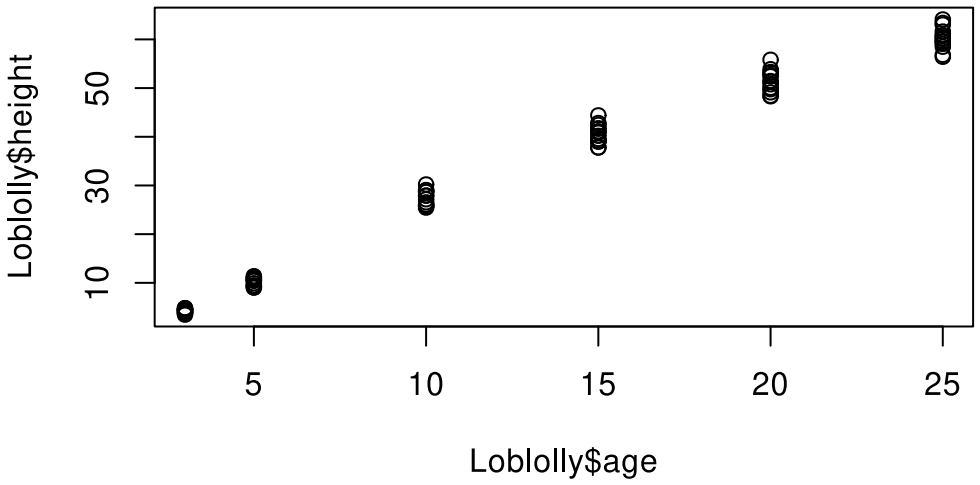
```
hist(
  Loblolly$height,
  xlab = "Taille en pieds",
  ylab = "Fréquence",
  main = "Distribution de fréquence des tailles de Pinus taeda"
)

abline(v = median(Loblolly$height), col = "red", lwd = 4)
```



Les graphiques en nuage de points

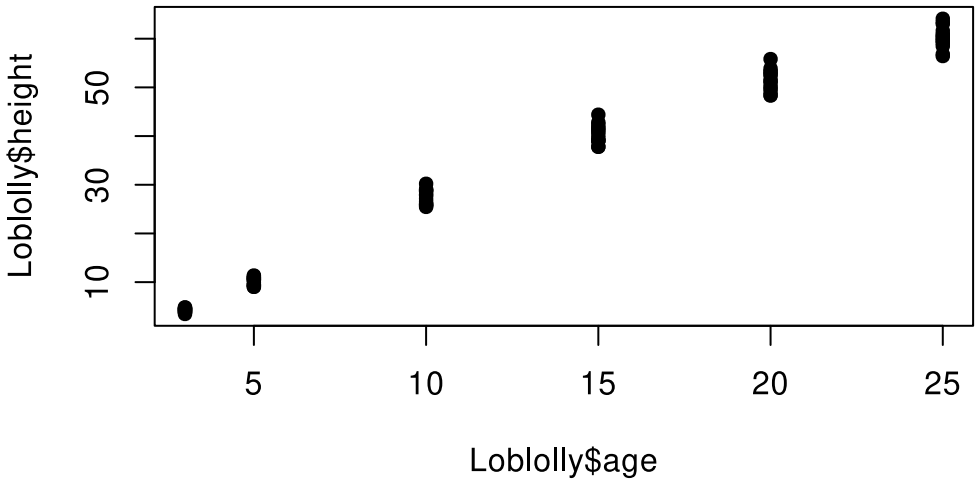
Comme nous l'avons vu précédemment, les graphiques en nuage de points peuvent servir à voir la relation entre deux variables. Commençons par voir s'il y a une relation entre la taille des arbres et leur âge.

```
# Graphique de la taille (axe des y) en fonction de l'âge (axe des x)
plot(Loblolly$age, Loblolly$height)
```



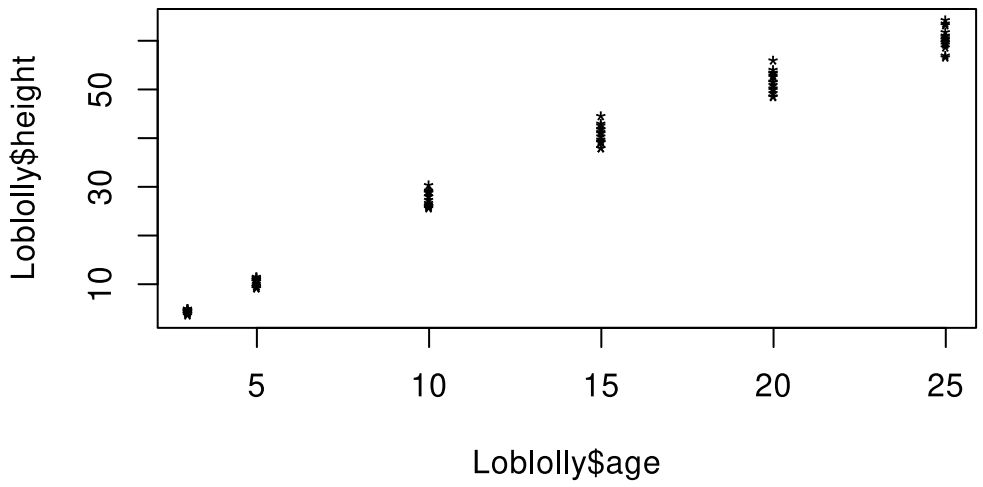
C'est bien, mais pourrait-on modifier l'apparence du symbole des points? C'est possible en utilisant le paramètre pch. On peut avoir des symboles pleins ou vides, ou modifier la forme.

```
# changer les symboles pour des symboles de cercle pleins noirs
plot(Loblolly$age, Loblolly$height, pch = 16)
```



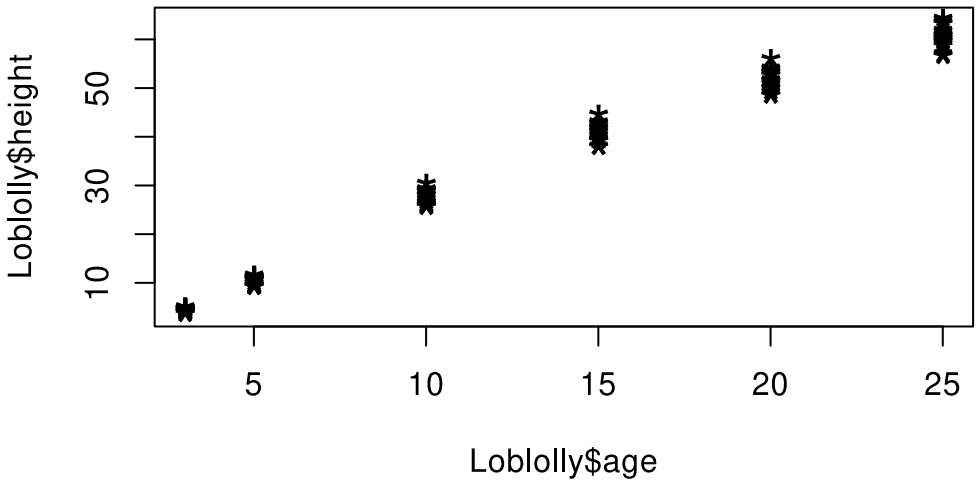
Il y a de nombreux symboles possibles, explorez!

```
# Faire le même graphique mais avec des étoiles comme symbole
plot(Loblolly$age, Loblolly$height, pch = 42)
```



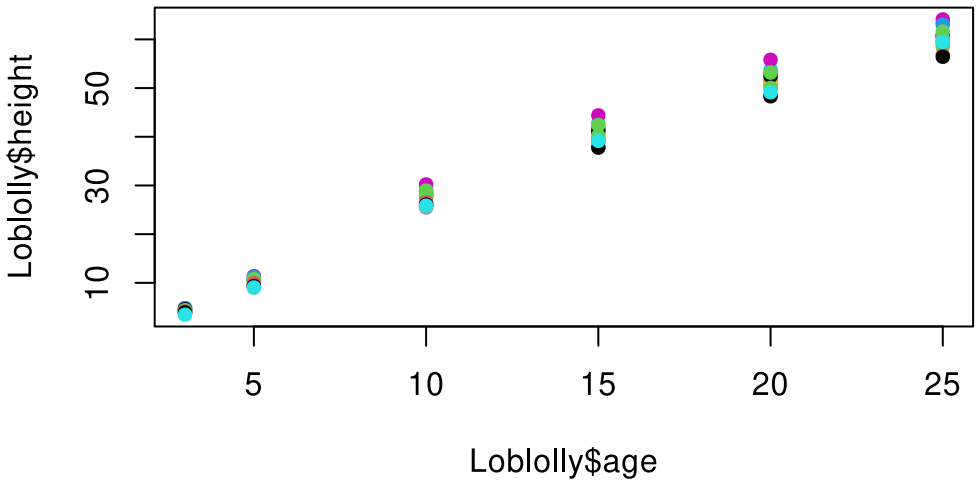
Il est aussi possible de modifier la taille des symboles, avec le paramètre cex.

```
# On augment la taille des symboles
plot(Loblolly$age, Loblolly$height, pch = 42, cex = 2)
```



Ces arbres proviennent de différentes familles (la variable `Loblolly$Seed` qui est un facteur). Il est possible que leur famille d'origine affecte leur croissance. On pourrait utiliser les couleurs pour visualiser séparément les différentes familles. Pour y arriver, on spécifie que l'option de couleur `col` doit représenter la variable `Loblolly$Seed`.

```
# Spécifier la couleur avec une variable qui est un facteur
plot(Loblolly$age, Loblolly$height, pch = 16, col = Loblolly$Seed)
```

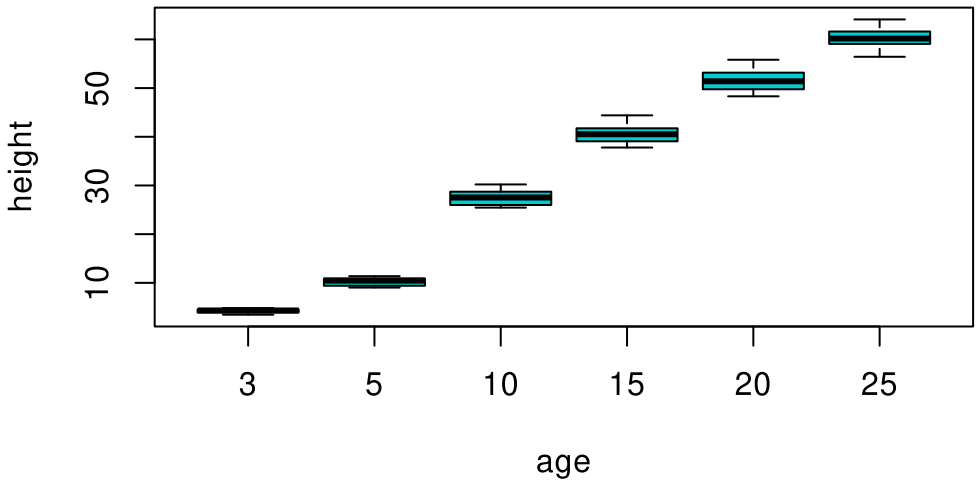


Vous pouvez donc utiliser une variable en format facteur car elle permet de grouper les données pour déterminer la couleur à utiliser, comme par exemple selon la population d'origine, ou un groupe contrôle et un groupe traité, l'année de récolte des données, ou bien le sexe des individus.

Les graphiques à moustache

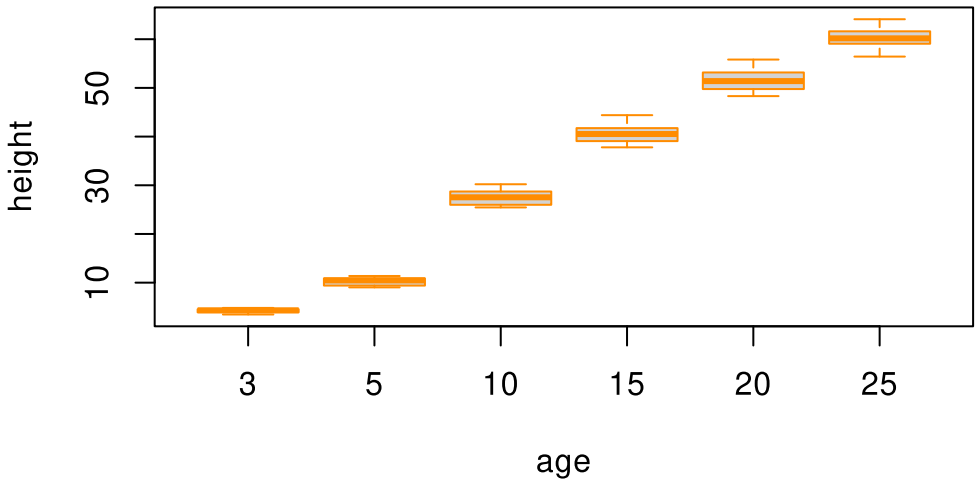
Il est possible de modifier la couleur des graphiques à moustache. Par exemple, on pourrait vouloir que celui-ci soit dessiné en turquoise, ce que l'on précise avec `col = "nom de la couleur"`

```
boxplot(height ~ age, data = Loblolly, col = "darkturquoise")
```

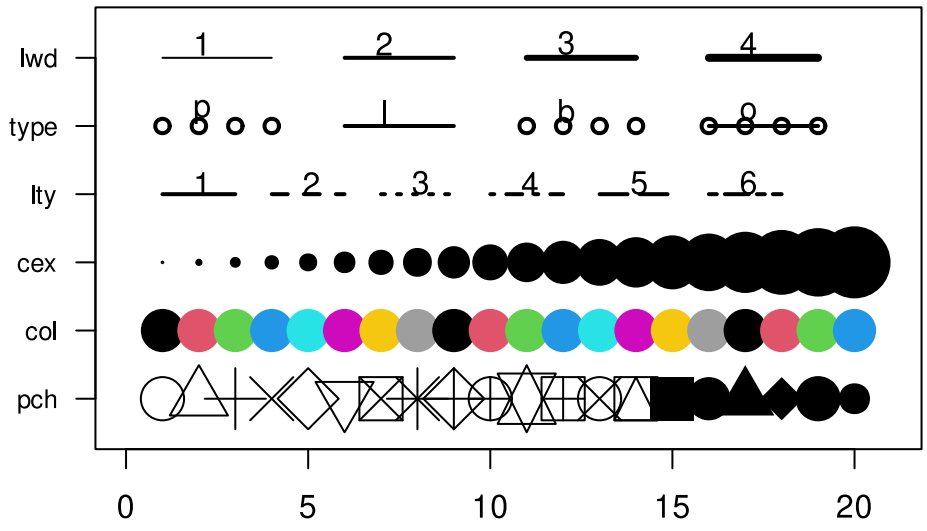


On pourrait aussi plutôt modifier la couleur du tracé

```
boxplot(height ~ age, data = Loblolly, border = "darkorange")
```



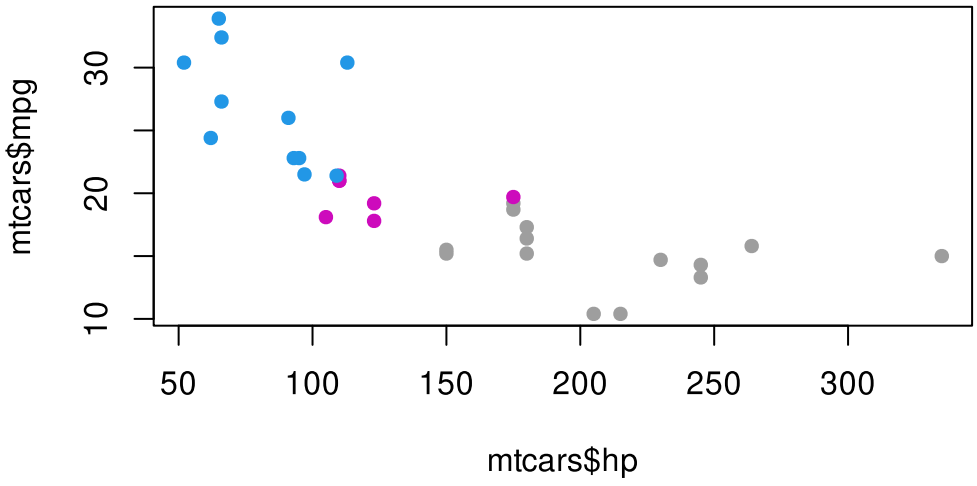
Voici un graphique aide-mémoire pour vous aider avec les différents paramètres de base pour modifier vos graphiques.



Légendes

Supposons ce graphique fait avec les données du *data frame* mtcars. La couleur des points est associée au nombre de cylindres du moteur (cyl).

```
plot(mtcars$hp, mtcars$mpg, pch = 16, col = mtcars$cyl)
```

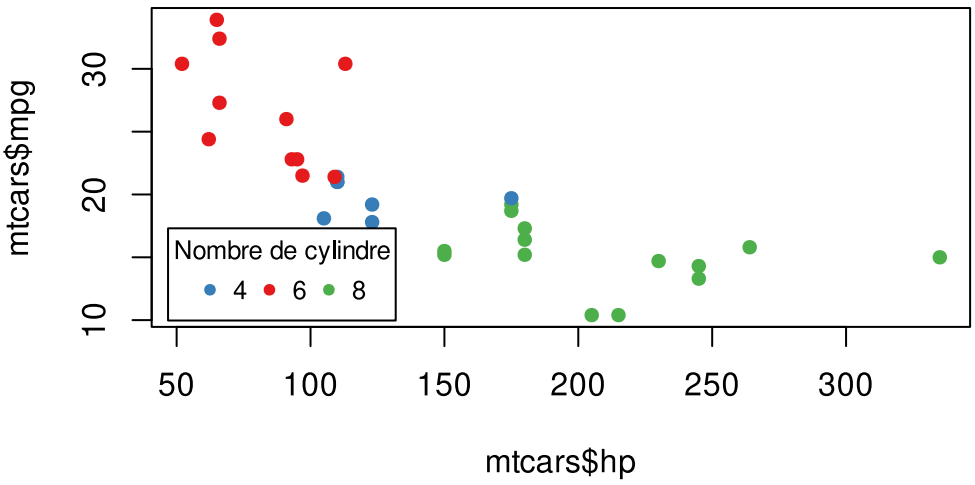


Il serait intéressant de pouvoir ajouter une légende à ce graphique pour identifier à quoi correspondent les couleurs. Pour y arriver, il faut utiliser la fonction legend() après la création du graphique. Cette fonction utilise plusieurs paramètres. Prenez le temps d'explorer par vous-même pour bien comprendre.

```
mes_couleurs <- brewer.pal(n = 3, "Set1") # Choisir une palette de couleur
mes_couleurs
```

```
[1] "#E41A1C" "#377EB8" "#4DAF4A"
```

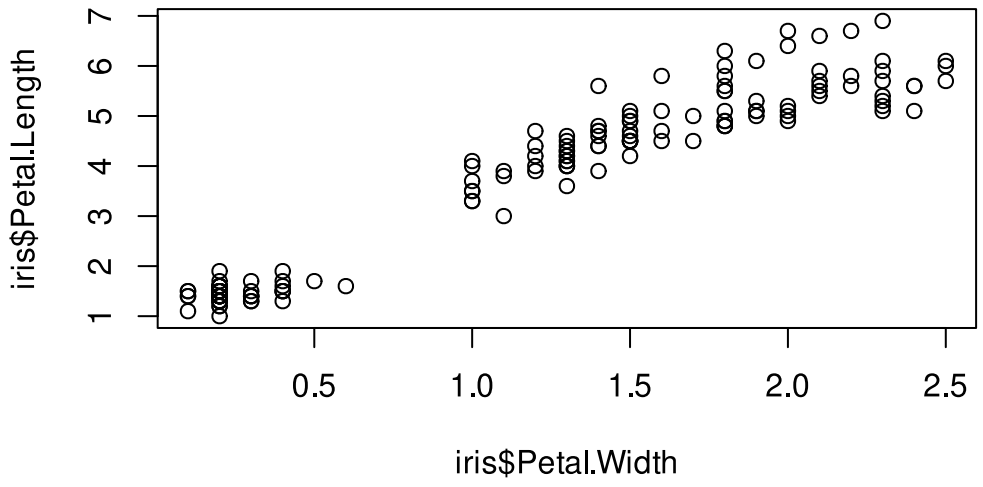
```
# On fait le graphique de base. Ne pas oublier d'utiliser la fonction facteur
pour changer la classe de la variable cyl
plot(mtcars$hp, mtcars$mpg, pch = 16, col = mes_couleurs[factor(mtcars$cyl)])

legend(
  x = "bottomleft", # position de la légende
  inset = .02, # Ajustement de la position (pour déplacer légèrement la légende)
  title = "Nombre de cylindre", # Titre de la légende
  c("4", "6", "8"), # Éléments de la légende
   col = unique(mes_couleurs[factor(mtcars$cyl)]), # Couleur des point de la
légende
  horiz = TRUE, # On veut une légende horizontale
  pch = 16, # Symboles à utiliser dans la légende
  cex = 0.8 # Grosseur des symboles
)
```



Droite de régression

Nous avons déjà utilisé la fonction abline pour tracer des lignes verticales. On peut également utiliser cette fonction pour tracer des droites de régression. Nous utiliserons les données iris pour démontrer comment ajouter une telle droite. Dans le graphique suivant, il semble y avoir une relation linéaire entre les variables `Petal.Width` et `Petal.Length`.

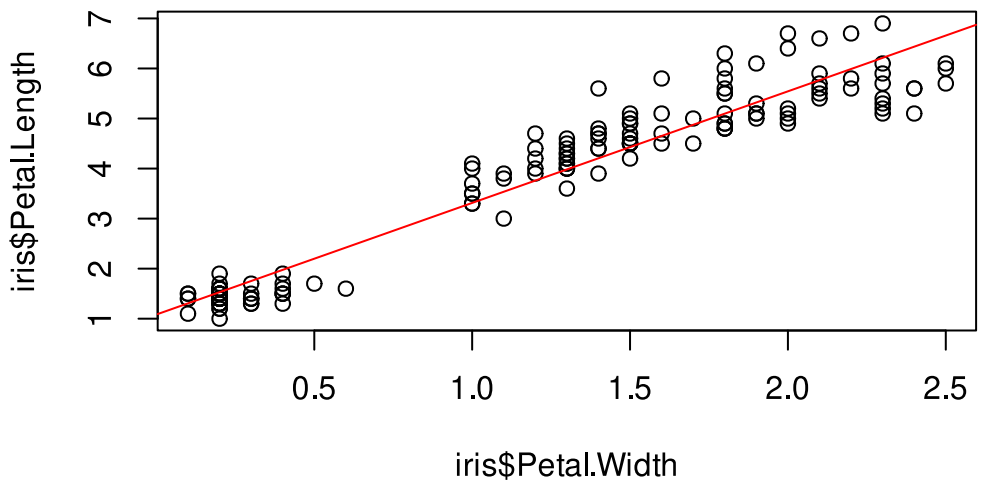
```
data(iris)
plot(iris$Petal.Width, iris$Petal.Length)
```



Pour ajouter une droite de régression, il faut d'abord faire un modèle linéaire à l'aide de la fonction lm (*linear model* en anglais). Lorsque le modèle est fait, on peut utiliser la fonction abline(modele) pour ajouter la droite de régression[1].

```
mod <- lm(Petal.Length ~ Petal.Width, data = iris)

plot(iris$Petal.Width, iris$Petal.Length)
abline(mod, col = "red")
```

[1] Voir la capsule statistique portant sur les modèles linéaire.

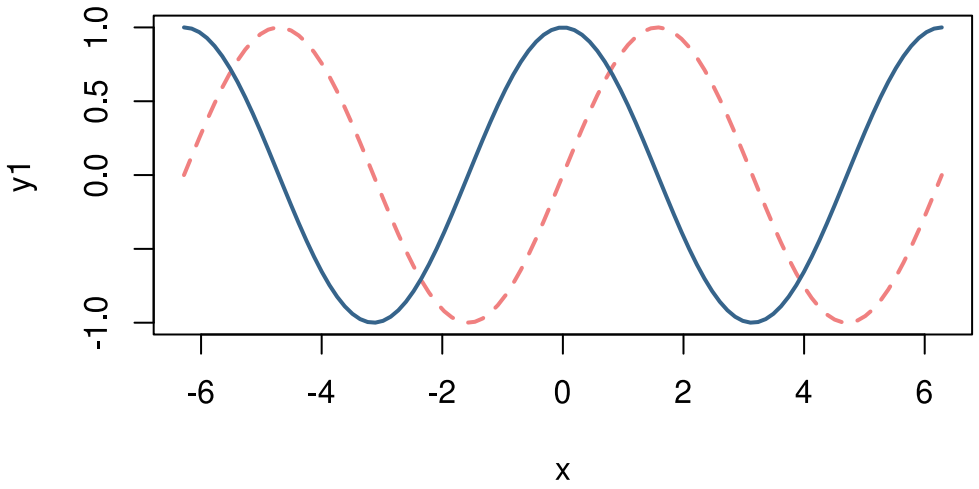


Ajout de courbes

Jusqu'à maintenant, nous avons seulement affiché une série de données dans nos graphiques. Il est souvent intéressant de pouvoir visualiser plusieurs courbes dans un seul graphique. Pour y arriver, il suffit d'utiliser la fonction lines() après l'appel de la fonction plot(). La fonction lines() fonctionne de la même manière que la fonction plot(), à la différence qu'elle doit être utilisée une fois le graphique principale créée.

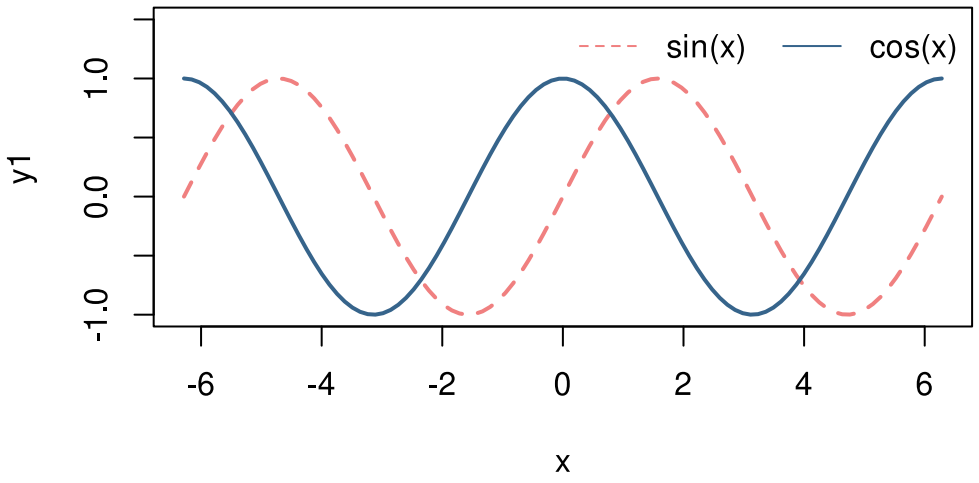
Nous allons créer un jeu de données pour montrer comment utiliser la fonction lines().

```
x <- seq(from = -2 * pi, to = 2 * pi, length.out = 100) # Variable x
y1 <- sin(x) # Première variable y
y2 <- cos(x) # Deuxième variable y

# Afficher la première série
plot(x, y1, col = "lightcoral", type = "l", lty = 2, lwd = 2)

# Ajouter la deuxième série
lines(x, y2, col = "steelblue4", lty = 1, lwd = 2)
```



Évidemment, il est nécessaire d'ajouter une légende pour identifier les deux courbes.

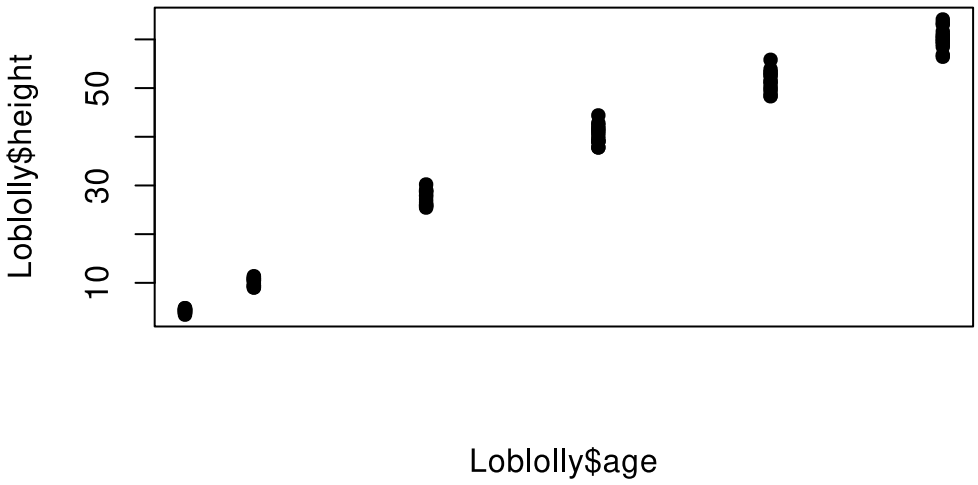
```
plot(x, y1, col = "lightcoral", type = "l", lty = 2, lwd = 2, ylim = c(-1, 1.5))
lines(x, y2, col = "steelblue4", lty = 1, lwd = 2)
legend(
  "topright",
  legend = c("sin(x)", "cos(x)"),
  col = c("lightcoral", "steelblue4"),
  lty = c(2, 1),
  horiz = TRUE,
  bty = "n"
)
```



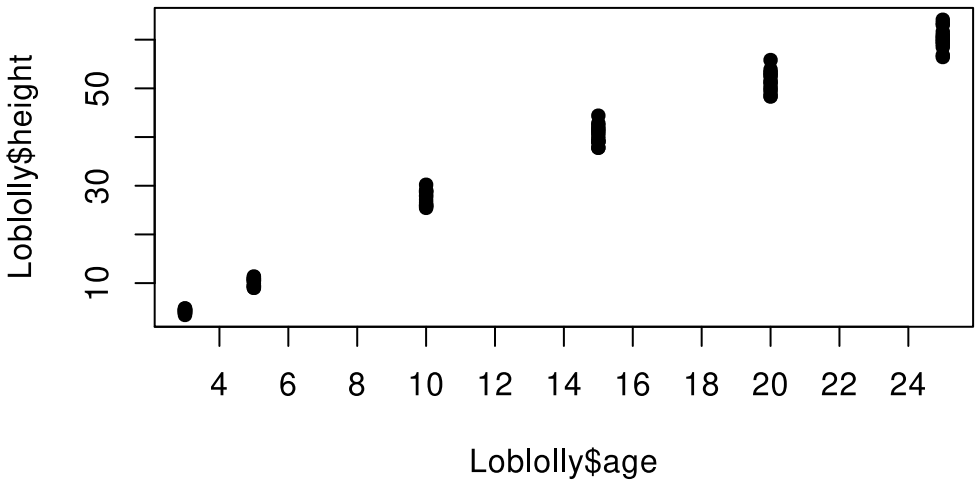
Modification des axes

Par défaut, R tentera de placer un nombre de marqueurs sur les axes de manière optimale. Cependant, nous pouvons contrôler comment ils apparaissent. La première étape est de dire à R de ne pas afficher les marqueurs sur l'axe des x avec le paramètre `xaxt = "n"`. La seconde étape est d'utiliser la fonction axis() pour ajouter manuellement l'axe des x. Pour l'axe des x, il faut spécifier `side = 1`. Par la suite, on utilise le paramètre `at = v` où `v` est un vecteur numérique indiquant la position des marqueurs. Dans le cas suivant, on place un marqueur à chaque valeur de l'âge entre 0 et 25.

```
plot(Loblolly$age, Loblolly$height, pch = 16, xaxt = "n")
```

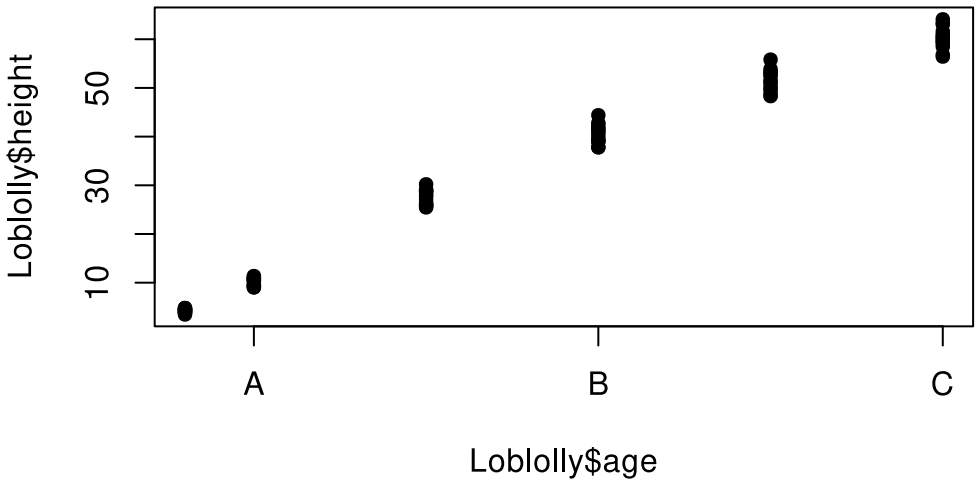



```
axis(side = 1, at = seq(0, 25, by = 2))
```



Finalement, on peut facilement remplacer la valeur des étiquettes en utilisant le paramètre labels dans la fonction axis().

```r
plot(Loblolly$age, Loblolly$height, pch = 16, xaxt = "n")
axis(side = 1, at = c(5, 15, 25), labels = c("A", "B", "C"))
```



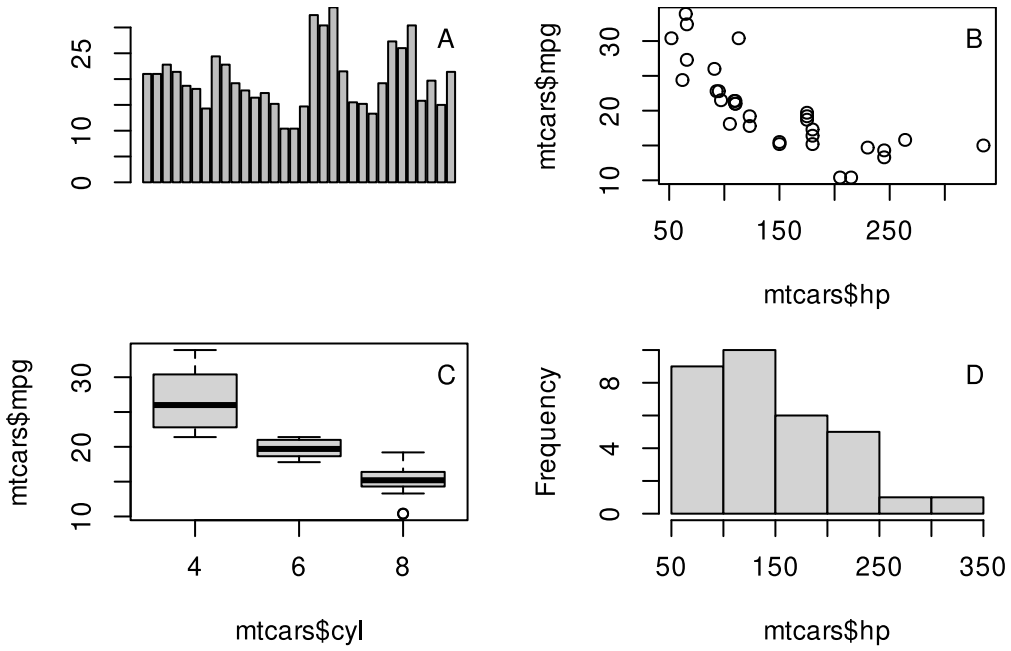
Mise en page de plusieurs graphiques

Dans certains cas, il est peut-être utile de combiner plusieurs graphiques dans un seul. Pour y arriver, il faut utiliser la fonction par() avant de faire les graphiques. C'est avec cette fonction qu'il faut définir comment les graphiques seront présentés. Il existe plusieurs paramètres que l'on peut modifier avec cette fonction, mais les plus fréquentes sont mfrow et mar. Le paramètre mfrow (un vecteur numérique de 2 éléments) permet de spécifier combien de ligne et de colonnes le graphique contiendra. Par exemple, mfrow = c(1, 2) indique un graphique avec 1 ligne et 2 colonnes. Le paramètre mar (un vecteur numérique de 4 éléments) permet de spécifier les marges à utiliser autours de chaque sous-graphiques. Par exemple, mar = c(1, 1, 1, 1) indique d'utiliser une valeur de 1 ligne sur chaque coté des graphiques.

Dans l'exemple suivant, on créer un graphique avec 4 sous-graphiques et on ajouter une légende pour identifier les panneaux A, B, C et D.

```r
# Un graphique avec 4 panneaux (2 lignes * 2 colonnes)
par(mfrow = c(2, 2), mar = c(5, 5, 0, 1))

barplot(mtcars$mpg) # plot 1
legend("topright", "A", bty = "n")

plot(mtcars$hp, mtcars$mpg) # plot 2
legend("topright", "B", bty = "n")
```

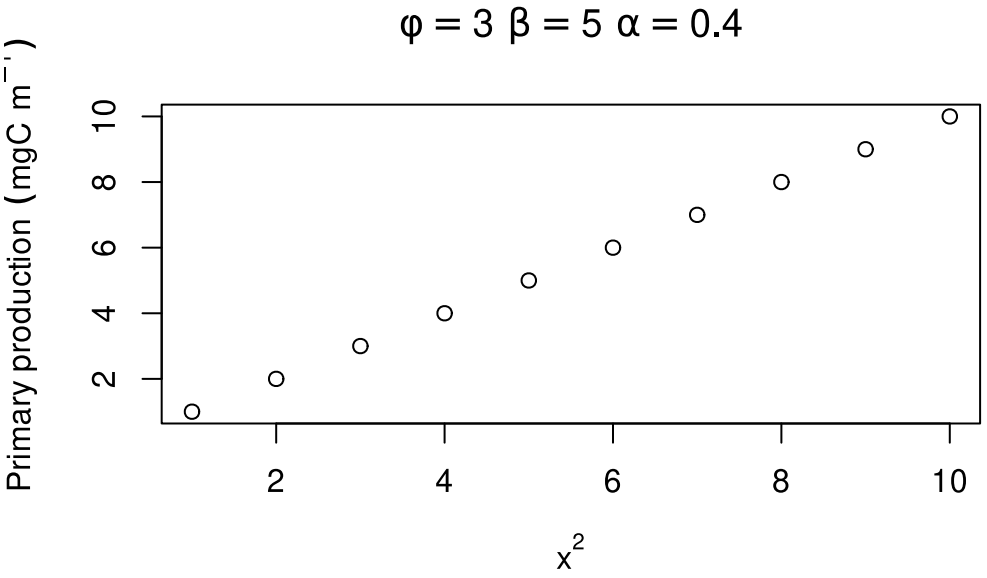
```r
boxplot(mtcars$mpg ~ mtcars$cyl) # plot 3
legend("topright", "C", bty = "n")

hist(mtcars$hp, main = "") # plot 4
legend("topright", "D", bty = "n")
```



Notation mathématique

En utilisant la fonction bquote() il est possible d'ajouter des symboles mathématiques dans les graphiques. Cette fonction est relativement complexe, on vous invite à consulter l'aide pour bien la comprendre. Voici cependant quelques exemples.

```r
plot(
  1:10,
  1:10,
  xlab = bquote(x^2),
  ylab = bquote("Primary production" ~ (mgC ~ m^{
    -1
  })),
  main = bquote(phi == 3 ~ beta == 5 ~ alpha == 0.4)
)
```


φ = 3 β = 5 α = 0.4
Primary production (mgC m⁻¹)
2
4
6
8
10
2
4
6
8
10
x²