Les graphiques (2ème partie)

Thibault LAURENT

19 Novembre 2014

Ce document a été généré directement depuis RStudio en utilisant l'outil Markdown. La version .pdf se trouve ici.

Résumé

Dans cette deuxième partie, on présentera les graphiques du point de vue du statisticien. C'est pourquoi on fera la distinction entre variable quantitative, variable qualitative, croisement entre 2 variables (quantitative/quantitative, quantitative/qualitative, qualitative/qualitative), qui selon le cas, fait appel à un graphique particulier.

Rappel

Avant de commencer, vous devez effectuer les opérations suivantes afin de disposer de tous les éléments nécessaires à l'apprentissage de cet E-thème.

1. Créer un dossier propre à cet E-thème et l'indiquer comme répertoire dans lequel vous allez travailler à l'aide de la fonction *setwd()*.

```
setwd("Z:/Thibault Pro/cours 14-15/R/cours4")
```

2. Dans un sous répertoire nommé "Ressource", placer le fichier donnees.txt. Il contient la définition des jeux de données utilisés au cours de cet E-thème. Le charger sous R avec la fonction *source()*.

```
source("Ressource/donnees.txt")
```

Pour vérifier que le **data.frame** nommé **df** a bien été chargé, utiliser la fonction *ls()* :

```
ls()
```

```
## [1] "df"  "vec"
```

2. Graphiques pour la statistique descriptive

On a vu dans la section précédente la "logique" de programmation utilisée pour représenter des graphiques sous R. On va s'intéresser dans cette section plus particulièrement aux fonctions graphiques disponibles sous R utiles pour l'analyse statistique descriptive.

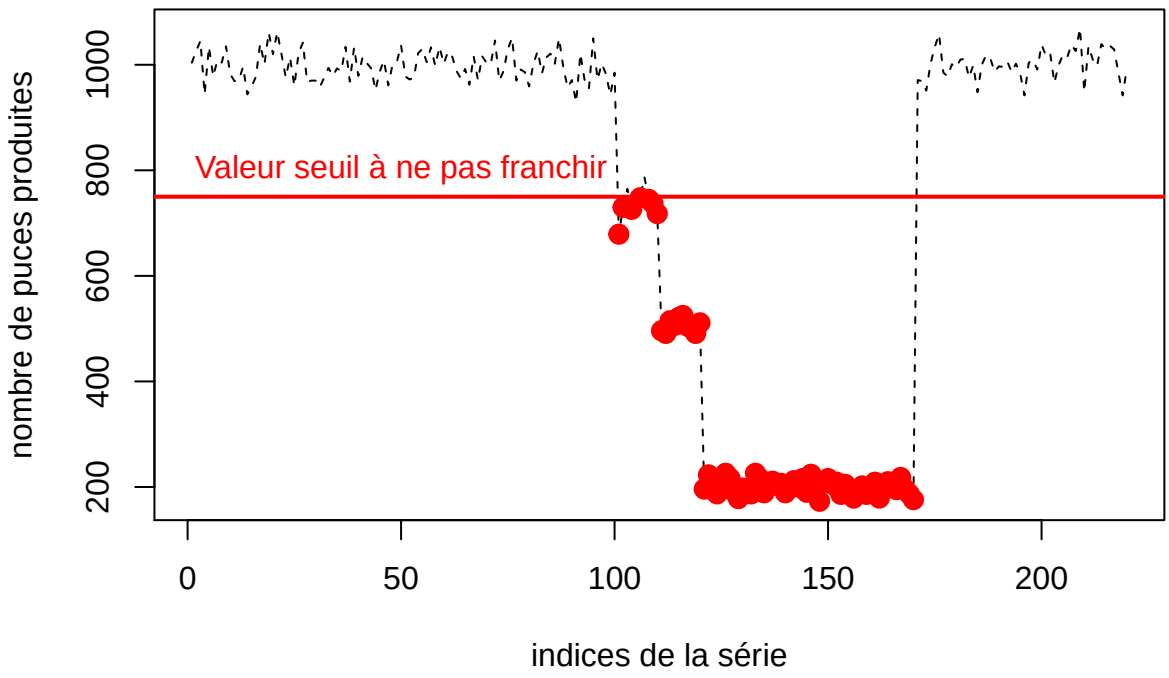
2.1. Analyse unidimensionnelle

a. Représentation d'un processus discret

La fonction *plot()* appliquée à une seule variable quantitative **x** de taille n (un objet de type **numeric** ou **integer**) renvoie le graphique des valeurs de **x** par rapport à leurs indices dans le vecteur (de 1 à n). Ce graphique n'a pas trop d'intérêt si les observations sont indépendantes et identiquement distribuées (i.i.d). En revanche, lorsqu'il s'agit de données issues d'un processus discret ou bien d'une série temporelle, ce type de représentation peut être intéressant, car il permet de visualiser l'évolution de la série; par exemple, cela peut permettre d'observer un changement de comportement.

Exemple : dans une usine qui produit des puces électroniques, on suit par heure le nombre de puces produites. Lorsque tout se passe bien, les machines ont été calibrées pour produire environ 1000 puces par heure. Lorsqu'il y a une machine défectueuse, cela entraîne une baisse de la production. Pour repérer s'il y a un problème dans la production, l'usine s'est fixée comme valeur seuil à ne pas franchir, la valeur 750. Nous avons simulé une série statistique supposant reproduire un tel phénomène. Les 100 premières valeurs sont simulées selon une loi de Poisson $\mathcal{P}(1000)$, puis nous avons simulé un incident tel que la production diminue progressivement jusqu'à atteindre une loi de Poisson $\mathcal{P}(200)$. Une fois le problème réparé, la série repart sur une loi de Poisson $\mathcal{P}(1000)$.

```
# simulation de la série
x=c(rpois(100,1000), rpois(10,750), rpois(10,500), rpois(50,200),
  rpois(50,1000))
# représentation de la série (fonction haut-niveau)
plot(x,xlab="indices de la série", ylab="nombre de puces produites",
  type="l", lty=2)
# valeur seuil (fonction bas-niveau)
abline(h=750, lwd=2, col="red")
# étiquette (fonction bas-niveau)
text(50,750, "Valeur seuil à ne pas franchir", pos=3, col="red")
# valeurs à problèmes
ind<-1:length(x)
points(ind[x<750],x[x<750], pch=20, cex=2, col="red")
```



b. Représentation d'une série temporelle

Si les indices du processus discret sont remplacés par des dates, alors on parlera de série temporelle. Il existe plusieurs façons de représenter une telle série dont nous allons présenter ici seulement le chronogramme. Pour plus de détails sur l'étude des séries temporelles, on réfèrera le lecteur à l'ouvrage d'Yves Aragon paru en 2011, "Séries temporelles avec R" (site web du livre).

Exemple : on a repris ici le tout premier exemple d'Aragon [2011] qui représente l'évolution de la population française et aux Etats-Unis. On commence par charger le package associé au livre ainsi que les données :

```
# chargement du package associé au livre
require("caschrono")
# chargement des données de population
data("popfr")
```

L'objet **popfr** est de classe **ts** qui contient la série des observations ainsi que les dates correspondantes. Pour obtenir ces dates, on utilise la fonction *time()*.

```
class(popfr)
```

```
## [1] "ts"
```

```
time(popfr)
```

```
## Time Series:
## Start = 1846
## End = 1951
## Frequency = 0.2
##  [1] 1846 1851 1856 1861 1866 1871 1876 1881 1886 1891 1896 1901 1906 1911
## [15] 1916 1921 1926 1931 1936 1941 1946 1951
```

Finalement, pour représenter le graphique d'un tel objet, l'idée est bien entendu de représenter les valeurs de la série en ordonnées et les dates correspondantes en abscisses. Pour cela, on va utiliser une fonction générique de la fonction *plot()*. Une **fonction générique** est en quelque sorte une extension d'une fonction connue afin d'utiliser le même nom de fonction, mais qui s'applique sur de nouvelles classes d'objets. Aussi, la fonction *plot()* vu précédémment peut s'appliquer sur différents type d'objet (dont la classe **ts**). Pour connaître les fonctions génériques de la fonction *plot()*, on utilise la fonction *methods()* :

```
methods(plot)
```

```
##  [1] plot.aareg*                  plot.acf*
##  [3] plot.agnes*                  plot.areg
##  [5] plot.areg.boot               plot.aregImpute
##  [7] plot.arma*                   plot.bats
##  [9] plot.biVar                   plot.clusGap*
## [11] plot.cox.zph*                plot.curveRep
## [13] plot.data.frame*             plot.decomposed.ts*
## [15] plot.default                 plot.dendrogram*
## [17] plot.density*                plot.diana*
## [19] plot.drawPlot                plot.ecdf
## [21] plot.ets                     plot.factor*
## [23] plot.forecast                plot.formula*
## [25] plot.function                plot.garch*
## [27] plot.gbayes                  plot.hclust*
## [29] plot.histogram*              plot.HoltWinters*
## [31] plot.irts*                   plot.isoreg*
## [33] plot.lm*                     plot.medpolish*
## [35] plot.mforecast*              plot.mlm*
## [37] plot.mona*                   plot.partition*
## [39] plot.ppr*                    plot.prcomp*
## [41] plot.princomp*               plot.profile.nls*
## [43] plot.Quantile2               plot.rm.boot
## [45] plot.shingle*                plot.silhouette*
## [47] plot.spec*                   plot.spline*
## [49] plot.splineforecast          plot.stepfun
## [51] plot.stl*                    plot.summary.formula.response
## [53] plot.summary.formula.reverse plot.summaryM
## [55] plot.summaryP                plot.summaryS
## [57] plot.survfit*                plot.table*
## [59] plot.tbats                   plot.timeSeries*
## [61] plot.transcan                plot.trellis*
## [63] plot.ts                      plot.tskernel*
## [65] plot.TukeyHSD*               plot.varclus
## [67] plot.xyVector*               plot.zoo
##
##    Non-visible functions are asterisked
```

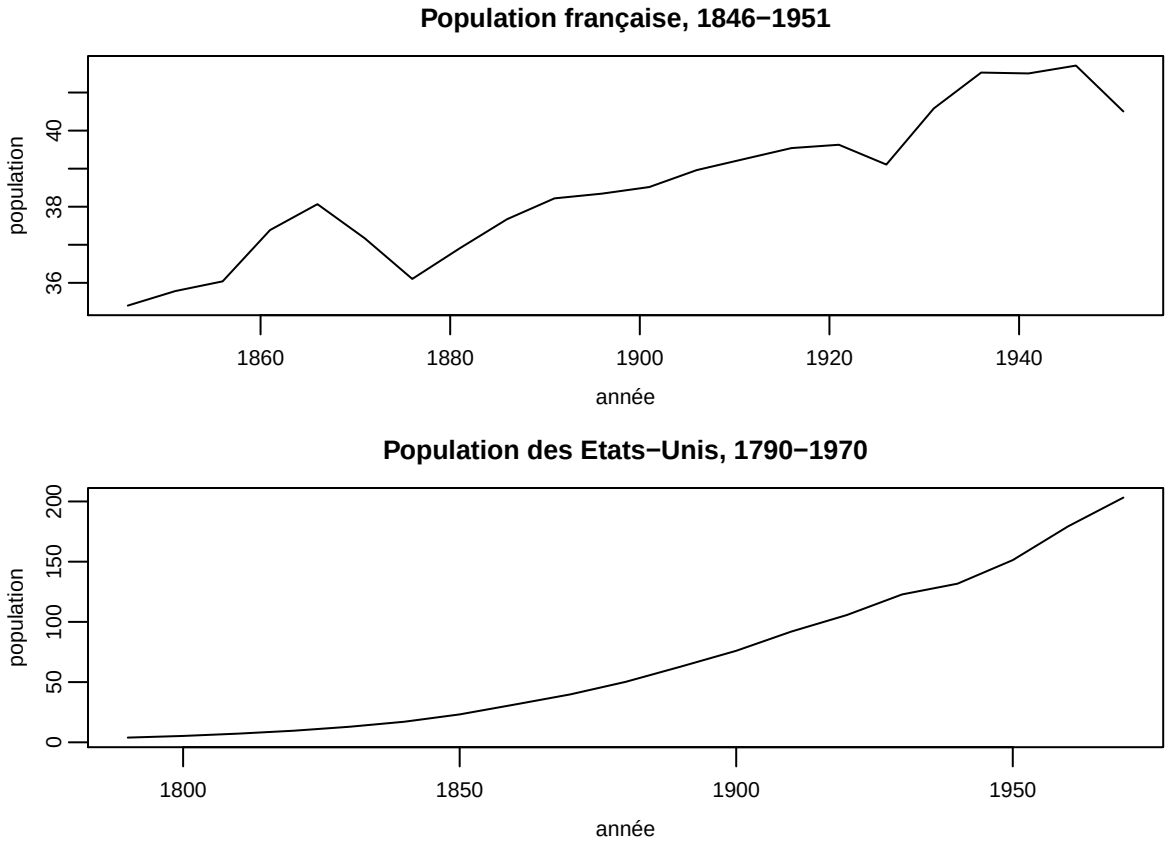
Remarque 1 : les fonctions suivies d'un astérisque sont des fonctions non visibles, c'est-à-dire que leurs codes ne s'affichent pas directement lorsqu'on tape leur nom dans la console. Pour afficher le code de ces fonctions, il faut utiliser la fonction *getAnywhere*. Par exemple :

```
getAnywhere(plot.acf)
```

Remarque 2 : on constate que le nombre de fonctions génériques de la fonction *plot()* est très important. Pour les appeler, on peut soit appeler la fonction par son nom complet (*plot.ts()* par exemple) ou simplement par la commande *plot()* (ce nombre varie selon les packages qui ont été installés sur les machines...).

Ici, on représente les deux séries temporelles dans la même fenêtre graphique en utilisant les commandes vues dans la partie précédente. Pour utiliser la fonction *plot.ts()*, il n'est pas nécessaire de mettre en abscisses les dates et en ordonnées les valeurs car la fonction *plot.ts()* s'en occupera elle-même.

```
# paramètres graphiques
op=par(mfrow=c(2,1),mar=c(2.5,2.5,2,2),mgp=c(1.5,.5,0),
  oma=c(0,0,0,0),cex.main=.8,cex.lab=.7,cex.axis=.7)
# appel de la fonction générique plot.ts
plot.ts(popfr, xlab='année', ylab='population',
   main ="Population française, 1846-1951")
plot(uspop, xlab='année', ylab='population',
  main ="Population des Etats-Unis, 1790-1970")
```




```
par(op)
```

c. Représentation de lois de distribution "théoriques"

Dans ce paragraphe, on montre comment représenter des lois de distribution théoriques. En général, on scinde les lois de distributions en deux familles selon la nature de la variable X :

- lorsque X est discrète, c'est-à-dire qu'elle prend ses valeurs parmi un nombre fini ou dénombrable de valeurs, les distributions les plus utilisés sont la loi binomiale (de paramètres n et p) et la loi de Poisson (de paramètre λ).
- lorsque X est continue, c'est-à-dire que les valeurs possibles pour x sont dans $\mathbb{R}$, les distributions les plus connues sont : la loi de Laplace/Gauss (de paramètres μ et σ), la loi de Fisher (de paramètres ν_1 et ν_2) et la loi de Student (de paramètre k).

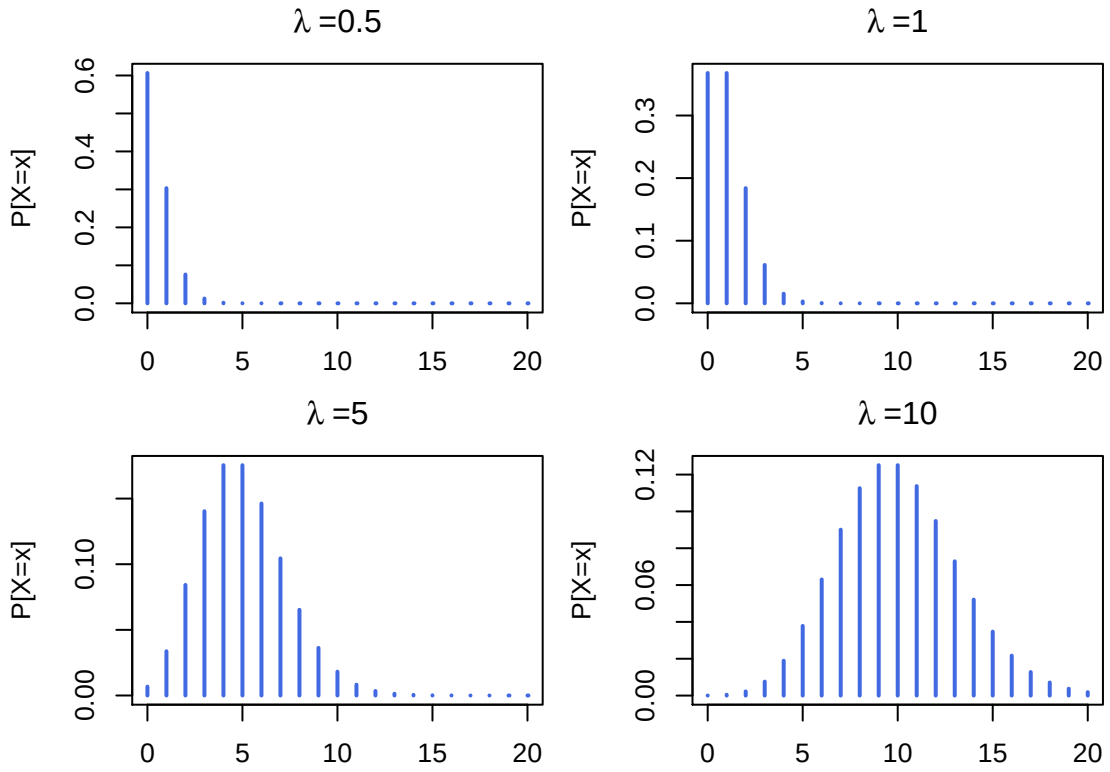
Pour caractériser une loi de distribution, on utilise plus particulièrement deux outils qui peuvent être représentés graphiquement :

- la densité de probabilité (usuellement notée f). Dans R, les fonctions qui permettent de calculer ces fonctions commencent par la lettre **d** suivi de l'abbréviation de la loi. Par exemple, **dpois**, **dbinom**, **dnorm**, etc.
- la fonction de répartition (usuellement notée F). Dans R, les fonctions qui permettent de calculer ces fonctions commencent par la lettre **p** suivi de l'abbréviation de la loi. Par exemple, **ppois**, **pbinom**, **pnorm**, etc.

Selon la nature de la variable X, ces outils ne seront pas représentés de la même façon (voir exemples ci-dessous). Pous plus d'informations, on se reportera à l'ouvrage de Saporta (1990) "Probabilités, Analyse des données et Statistique".

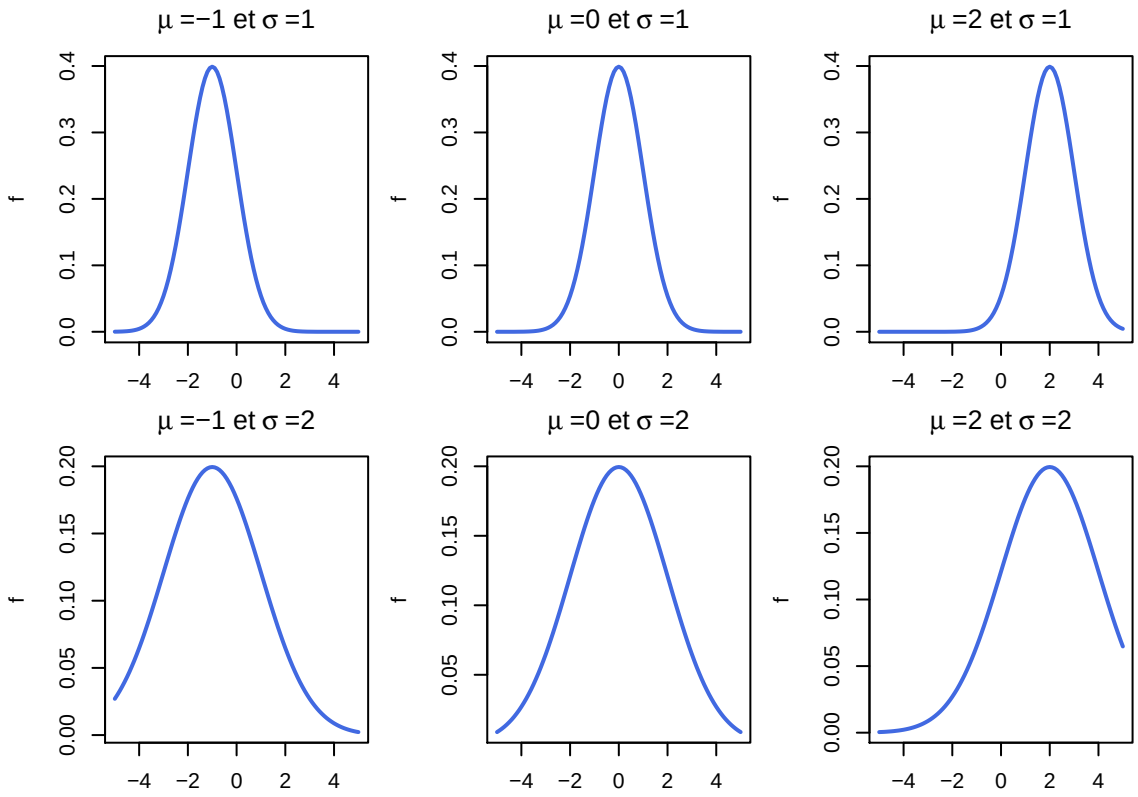
Exemple 1 : représentation de la densité de probabilités de la loi de Poisson pour différentes valeurs de λ. La fonction **dpois(x, lambda)** renvoie la probabilité d'obtenir la valeur entière **x** lorsque $X \sim \mathcal{P}(\lambda)$. Comme X ne prend que des valeurs discrètes, on représente les probabilités d'obtenir une valeur x par un trait vertical (option **type="h"** de la fonction *plot()*).

```
# valeurs de x entières
x<-seq(0,20,1)
# paramètres de la fenêtre graphique
op<-par(mfrow=c(2,2), oma=c(0.5,2,2,2),
        mai=c(0.4,0.6,0.6,0.15),mar=c(1.8,4,2.7,0.7))
# on boucle pour faire varier le paramètre
for(lambda in c(0.5,1,5,10))
 {# à chaque valeur du paramètre, on représente la fonction densité théorique
  plot(x,dpois(x,lambda),type="h", main=bquote(lambda~paste("=",.(lambda))),
       lwd=2, ylab="P[X=x]", col="royalblue")}
par(op)
title("Densité de probabilité : loi de Poisson", line = -1, outer=TRUE)
```



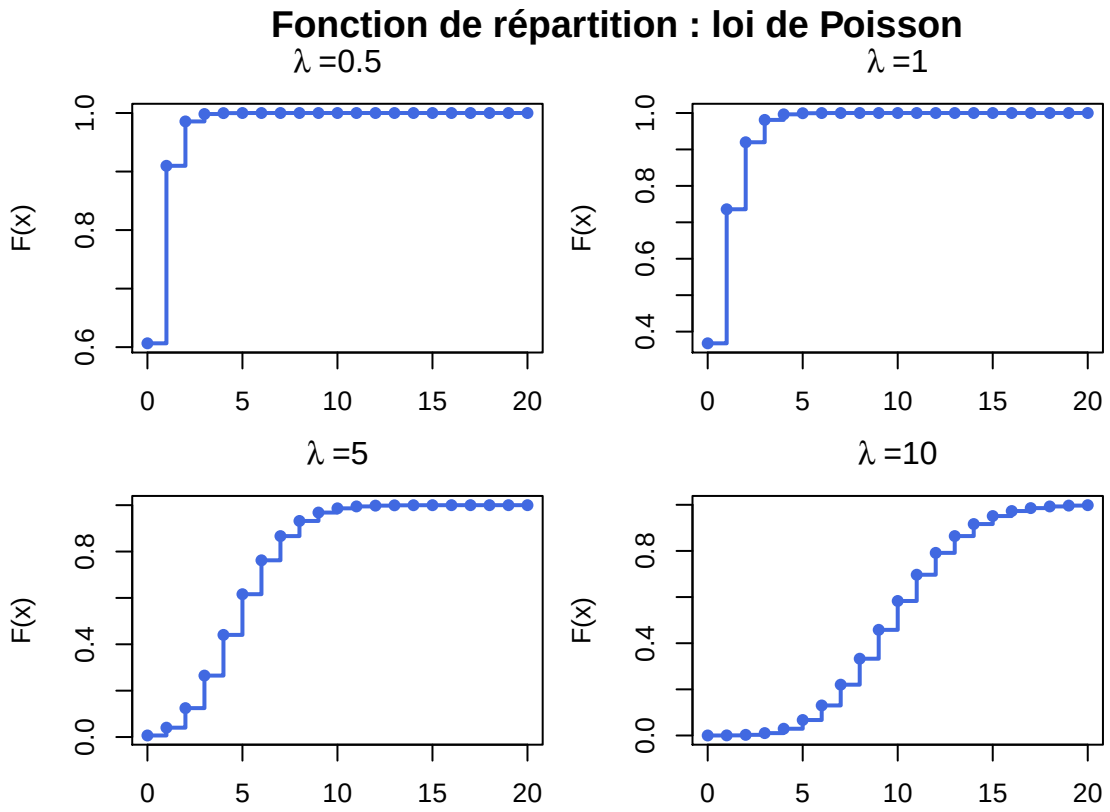
Exemple 2 : représentation de la densité de probabilités de la loi de Gauss/Laplace pour différentes valeurs de (μ, σ). La fonction **dnorm(x, mean = , sd =)** renvoie la valeur de la fonction de densité f_X au niveau de la valeur **x** lorsque $X \sim \mathcal{N}(\mu, \sigma^2)$. Comme X est continue, f_X existe pour tout $x \in \mathbb{R}$. On représente ici cette fonction pour différentes valeurs de μ et σ avec un trait continu (option **type="l"** de la fonction *plot()*)

```
# discrétisation de x
x<-seq(-5,5,0.1)
# paramètres de la fenêtre graphique
op<-par(mfrow=c(2,3), oma=c(0.5,2,2,2),
        mai=c(0.4,0.6,0.6,0.15),mar=c(1.8,4,2.7,0.7))
# on boucle pour faire varier les paramètres
for(sigma in c(1,2))
  {for(mu in c(-1,0,2))
   {# à chaque couple de paramètres, on représente la fonction densité théorique
    plot(x,dnorm(x,mu,sigma),type="l",lwd=2, ylab="f", col="royalblue",
         main=bquote(mu~paste("=",.(mu)," et")~sigma~paste("=",.(sigma))))
    }
  }
par(op)
title("Densité de probabilité : loi de Gauss/Laplace", line = -1, outer=TRUE)
```



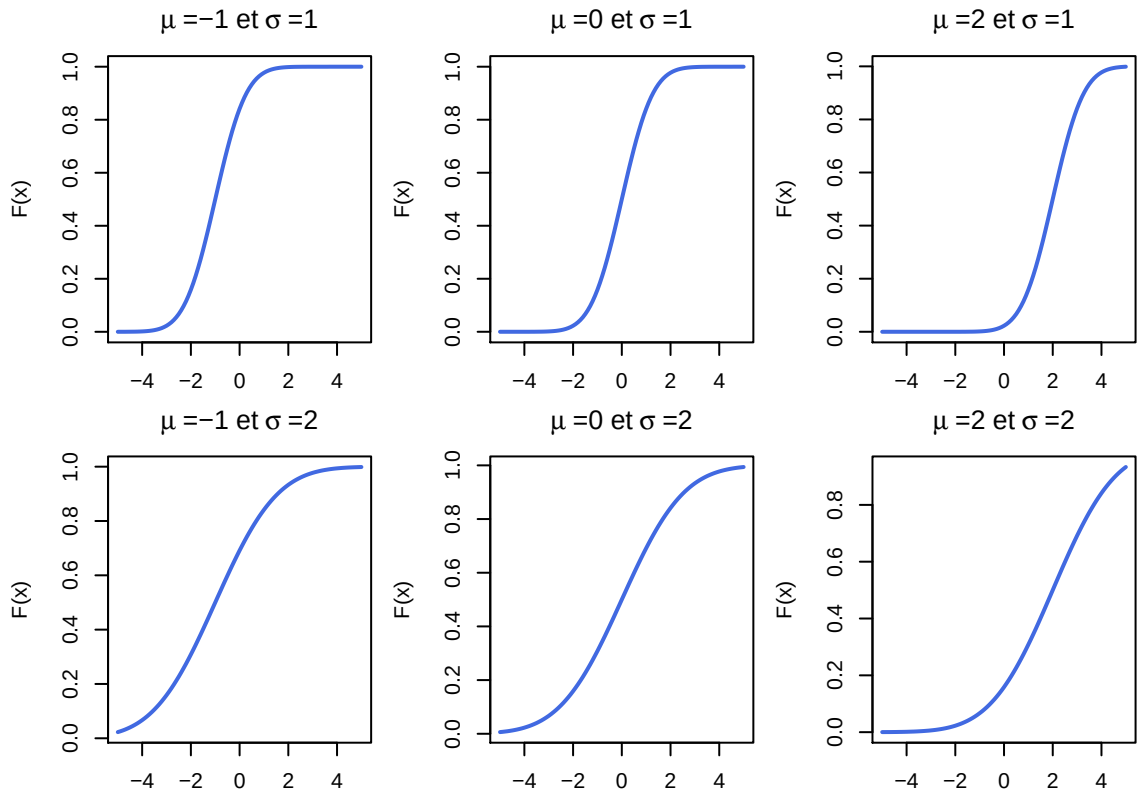
Exemple 3 : représentation de la fonction de répartition de la loi de Poisson pour différentes valeurs de λ. La fonction **ppois(q, lambda)** renvoie la probabilité d'obtenir une valeur inférieure ou égale à la valeur entière **x** lorsque $X \sim \mathcal{P}(\lambda)$. Comme X est discrète, la forme de la fonction de répartition est en escalier (option **type="s"** de la fonction *plot()*) :

```
x<-seq(0,20,1)
op<-par(mfrow=c(2,2), oma=c(0.5,2,2,2),
        mai=c(0.4,0.6,0.6,0.15),mar=c(1.8,4,2.7,0.7))
for(lambda in c(0.5,1,5,10))
 {
  plot(x,ppois(x,lambda),type="s", main=bquote(lambda~paste("=",.(lambda))),
     pch=16, lwd=2, ylab="F(x)", col="royalblue")
  points(x,ppois(x,lambda), col = "royalblue", pch=16)
 }
par(op)
title("Fonction de répartition : loi de Poisson")
```



Exemple 4 : représentation de la fonction de répartition de la loi de Laplace/Gauss pour différentes valeurs de (μ, σ). La fonction **pnorm(x, mean = , sd =)** renvoie la probabilité d'obtenir une valeur inférieure ou égale à la valeur **x** lorsque $X \sim \mathcal{N}(\mu, \sigma^2)$. Comme X est continue, la courbe est également continue :

```
x<-seq(-5,5,0.1)
op<-par(mfrow=c(2,3), oma=c(0.5,2,2,2),
        mai=c(0.4,0.6,0.6,0.15),mar=c(1.8,4,2.7,0.7))
for(sigma in c(1,2))
  {for(mu in c(-1,0,2))
   {plot(x,pnorm(x,mu,sigma),type="l",lwd=2, ylab="F(x)", col="royalblue",
         main=bquote(mu~paste("=",.(mu)," et")~sigma~paste("=",.(sigma))))
    }
  }
par(op)
title("Fonction de répartition : loi de Gauss/Laplace", line = -1, outer=TRUE)
```



d. Représentation d'une variable quantitative discrète

Pour faire un résumé d'une variable quantitative discrète, on peut réaliser un tableau de fréquences. Reprenons le jeu de données **df** sur lequel on enlève les observations dont la variable **TrH** vaut **0h-24h**.

```
df.new<-df[df$TrH!="0h-24h",]
```

On rappelle que les observations sont des nombres qui comptabilisent combien de voitures sont passées au feu vert, au feu orange, rouge après 1, 2, 3 secondes, etc. sur différents feux, à différents horaires, etc. Le nombre d'observations est :

```
nrow(df.new)
```

```
## [1] 72
```

Remarque: lorsqu'on demande d'afficher le nombre de modalités que contient la variable **TrH**, on en trouve toujours 5 alors qu'on vient d'enlever la modalité **0h-24h**. On décide donc de redéfinir la variable **TrH** pour qu'elle ne contienne que les 4 modalités qui nous intéressent :

```
levels(df.new$TrH)
```

```
## [1] "7h30-8h30" "0h-24h"    "16h-18h"   "12h-14h"   "0h-6h"
```

```
df.new$TrH<-factor(df.new$TrH)
levels(df.new$TrH)
```

```
## [1] "7h30-8h30" "16h-18h"   "12h-14h"   "0h-6h"
```

La variable **RO3** est le nombre de voitures qui sont passées au feu rouge après 3 secondes. Une façon de savoir combien il y a de valeurs distintes est d'utiliser la fonction *unique()*, couplée avec la fonction *length()* :

```
length(unique(df.new$RO3))
```

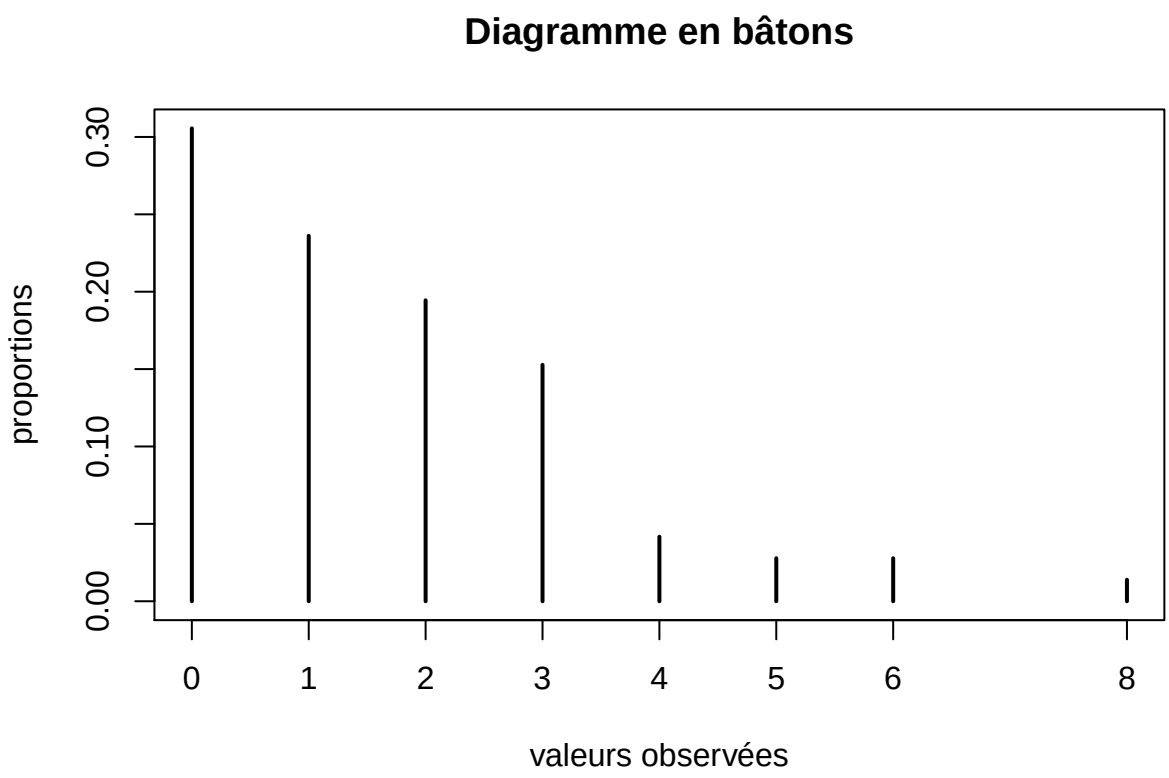
```
## [1] 8
```

On constate donc que ce sont souvent les mêmes valeurs qui reviennent : il existe un nombre fini de valeurs possibles pour cette variable qu'on peut donc considérer comme discrète. On résume cette variable par un tableau de fréquences absolues et/ou relatives :

valeurs	effectifs	proportion
0	22	0.3055556
1	17	0.2361111
2	14	0.1944444
3	11	0.1527778
4	3	0.0416667
5	2	0.0277778
6	2	0.0277778
8	1	0.0138889

L'outil graphique qui permet de représenter un tel tableau est le diagramme en bâtons :

```
# on construit le tableau des effectifs
tab<-table(df.new$RO3)
# pour représenter les proportions, on utilise prop.table()
plot(prop.table(tab), xlab="valeurs observées", ylab="proportions")
title("Diagramme en bâtons")
```

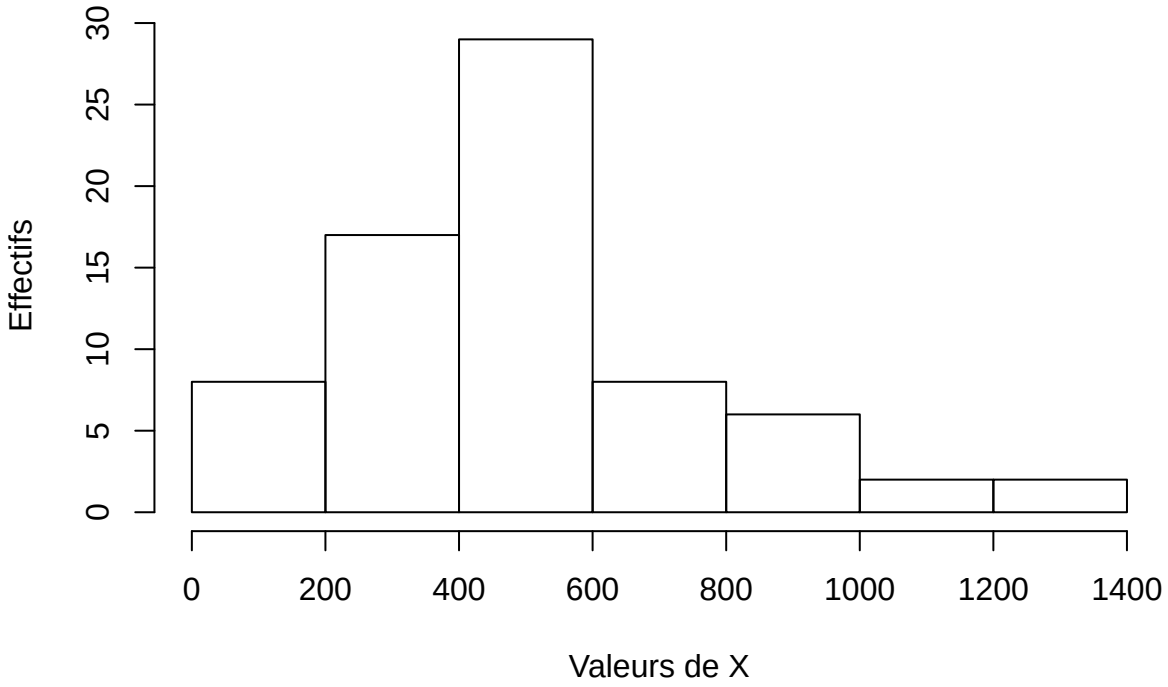


Remarque : ici, on a encore utilisé une fonction générique de *plot()*. Il s'agit de la fonction *plot.table()*.

e. Représentation d'une variable quantitative

L'histogramme En général, on utilise un histogramme pour représenter une variable quantitative X. Pour plus de détails sur l'histogramme, on renvoie le lecteur à ces quelques pages extraites du livre de Saporta [1990]. On reprendra les notations du livre de Saporta : on note $e_0, e_1, ..., e_k$ les bornes et l'on note pour chaque classe $[e_{i-1}, e_i[$ l'effectif n_i (*frequency* en anglais) et la fréquence f_i (à ne pas confondre avec *frequency* !). Par défaut, la fonction *hist()* représente les n_i en ordonnées car les classes sont d'amplitudes égales (c'est-à-dire que $(e_i - e_{i-1}) = cste$ quelque soit i). On notera que $e_0, e_1, ..., e_k$ sont calculées par défaut selon un algorithme particulier (voir *help(nclass)*). On peut aussi donner les valeurs de $e_0, e_1, ..., e_k$; pour cela, il faudrait utiliser l'option **breaks=** en précisant le vecteur des $e_0, e_1, ..., e_k$. On applique ici la fonction *hist()* sur la variable "nombre de personnes passés au feu vert" en utilisant les paramètres par défaut.

```
res.hist=hist(df.new$VER,
              xlab="Valeurs de X",ylab="Effectifs", main="")
```



On constate que la fonction *hist()* retourne un objet de type **list** qui contient un certain nombre d'informations dont :

- La valeur des $e_0, e_1, ..., e_k$:

```
res.hist$breaks
```

```
## [1]    0  200  400  600  800 1000 1200 1400
```

- La valeur des n_i :

```
res.hist$counts
```

```
## [1]  8 17 29  8  6  2  2
```

- La valeur des densités de probabiltés (c'est-à-dire $\frac{f_i}{e_i - e_{i-1}}$) :

```
res.hist$density
```

```
## [1] 0.0005555556 0.0011805556 0.0020138889 0.0005555556 0.0004166667
## [6] 0.0001388889 0.0001388889
```

- La valeur des barycentres des classes $\frac{1}{2}(e_i + e_{i-1})$:

```
res.hist$mids
```

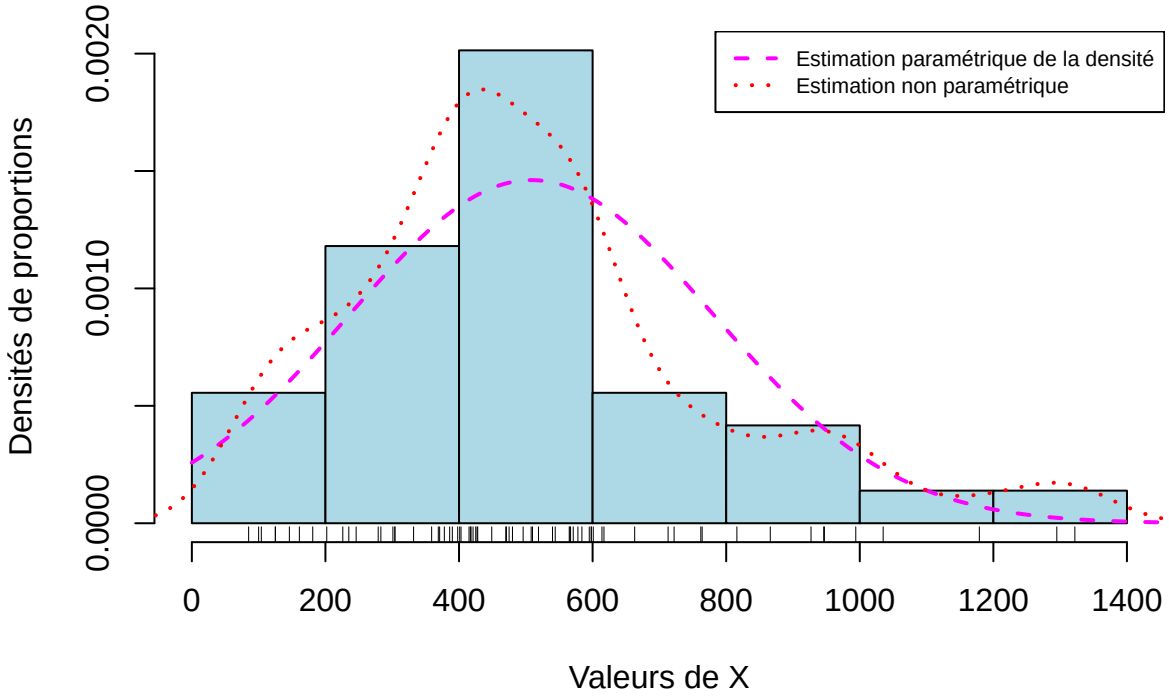
```
## [1]  100  300  500  700  900 1100 1300
```

L'inconvénient de représenter les effectifs en ordonnées est qu'on ne peut pas représenter l'estimation d'une densité par-dessus car les échelles ne correspondent pas. L'option **freq=FALSE** permet de représenter les densités de probabiltés en ordonnées.

La fonction de densité Dans certains cas, on a une connaissance *a priori* sur la distribution d'une variable quantitative. En d'autres termes, on sait que la variable X est issue d'une distribution connue et qu'on peut utiliser des outils de la "Statistique mathématique" pour estimer de façon adéquate le(s) paramètre(s) associé(s). Par exemple, si on suppose que la variable **VER** est issue d'une loi normale, alors les estimateurs les plus vrais semblants pour μ et σ^2 sont respectivement $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ et $\hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$.

Exemple : on va représenter par-dessus l'histogramme de la variable **VER**, la fonction de densité estimée en utilisant la fonction *dnorm()* vue précédemment.

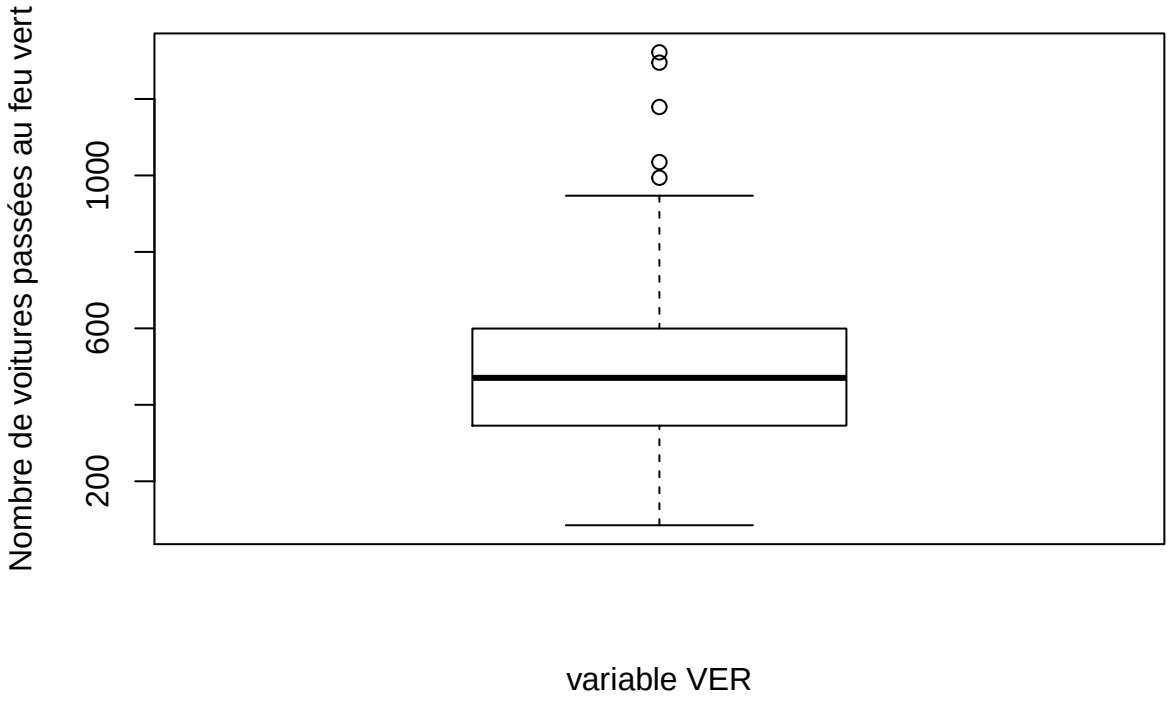
```
# représentation de l'histogramme
hist(df.new$VER, freq=FALSE, col="lightblue",
      xlab="Valeurs de X",ylab="Densités de proportions", main="")
# valeurs de x sur lesquelles on va calculer les fonctions de densité ...
x=seq(0,1500,1)
# ... théoriques d'une gaussienne
lines(x, dnorm(x,mean(df.new$VER),sd(df.new$VER)),
      lty=2, lwd=2, col="magenta")
# ... non paramétrique
lines(density(df.new$VER), col="red", lty=3, lwd=2)
# représentation de la légende
legend("topright",c("Estimation paramétrique de la densité","Estimation non paramétrique"),
  lty=2:3,lwd=2,col=c("magenta", "red"),cex=0.7)
# représentation des valeurs de x sur l'axe des abscisses
rug(df.new$VER)
```



Remarque : lorsqu'on ne sait rien de la loi de distribution théorique, on fait une estimation non paramétrique de la densité avec la fonction *density()* (ceci sera vu en détails dans un cours du M2). Par ailleurs, dans le code-ci-dessus,la fonction *rug()* permet de représenter les valeurs des observations sur l'axe des absicisses par un trait vertical.

La boîte à moustaches L'autre outil généralement utilisé pour représenter une variable quantitative est la boîte à moustache (fonction *boxplot()*) qui résume quelques caractéristiques de position (médiane, quartiles, minimum, maximum). Cet outil permet notamment de repérer plus facilement les valeurs extrêmes. Les options **xlab=** et **ylab=** permettent de donner une légende respectivement aux axes des abscisses et des ordonnées. Voici un premier exemple simple :

```
b<-boxplot(df.new$VER, xlab="variable VER", ylab="Nombre de voitures passées au feu vert")
```



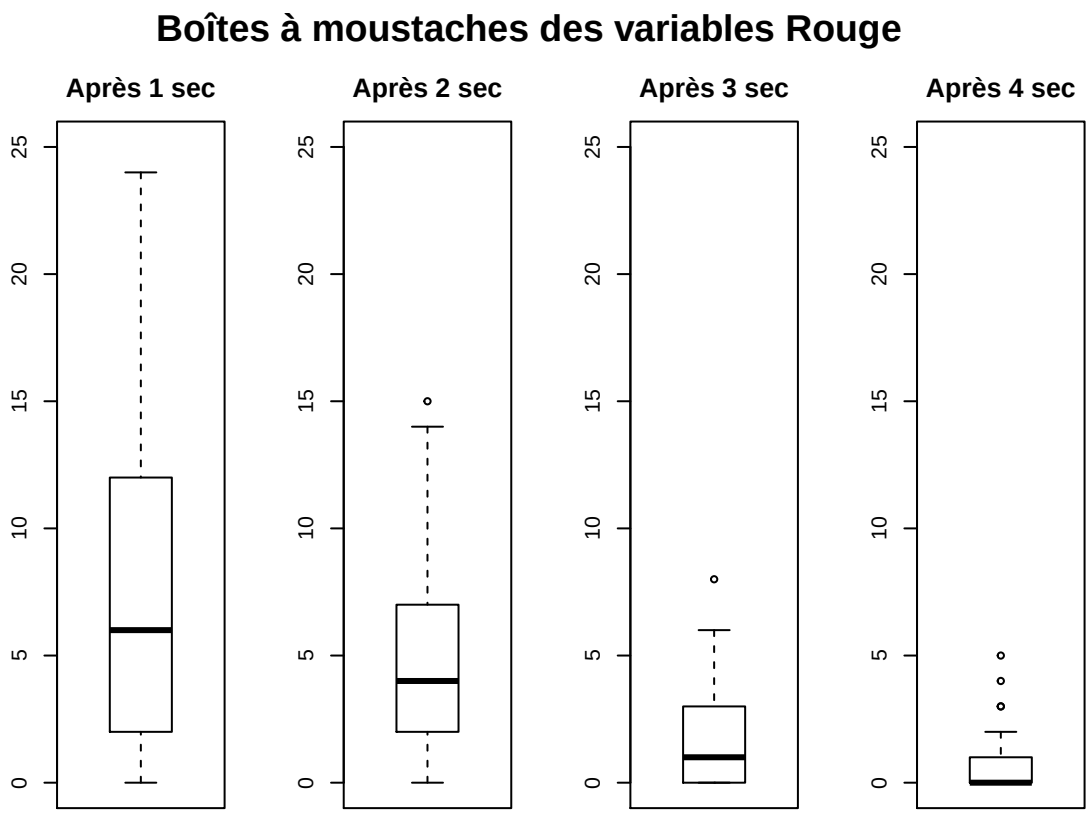

```
print(b)
```

```
## $stats
##       [,1]
## [1,]  85.0
## [2,] 345.5
## [3,] 470.5
## [4,] 599.5
## [5,] 947.0
##
## $n
## [1] 72
##
## $conf
##         [,1]
## [1,] 423.204
## [2,] 517.796
##
## $out
## [1] 1295 1322 1179 1035  994
##
## $group
## [1] 1 1 1 1 1
##
## $names
## [1] ""
```

L'objet **b** créé ci-dessus contient en autres les informations suivantes : **stats**=c($moustache_{inf}$, Q_1, Q_2, Q_3, $moustache_{sup}$) où Q_1, Q_2 et Q_3 sont les 1er, 2nd et 3ème quartiles. $moustache_{inf} \approx \max(Q_1 - 1.5(Q_3 - Q_1), \min_i x_i)$ et $moustache_{sup} \approx \min(Q_3 + 1.5(Q_3 - Q_1), \max_i x_i)$. **out** contient les valeurs extrêmes, c'est-à-dire les valeurs qui sont au-dessus (respectivement en-dessous) de $moustache_{sup}$ (resp. $moustache_{inf}$).

Exemple : lorsqu'on a un nombre important de variables quantitatives dont l'étendue est plus ou moins la même (c'est la cas notamment des données biologiques issues des puces ADN), on peut représenter les boîtes à moustaches côte à côte pour comparer les distributions entre elles. Ici, on représente les variables correspondant au nombre de voitures passées au feu Rouge après 1, 2, 3 et 4 secondes.

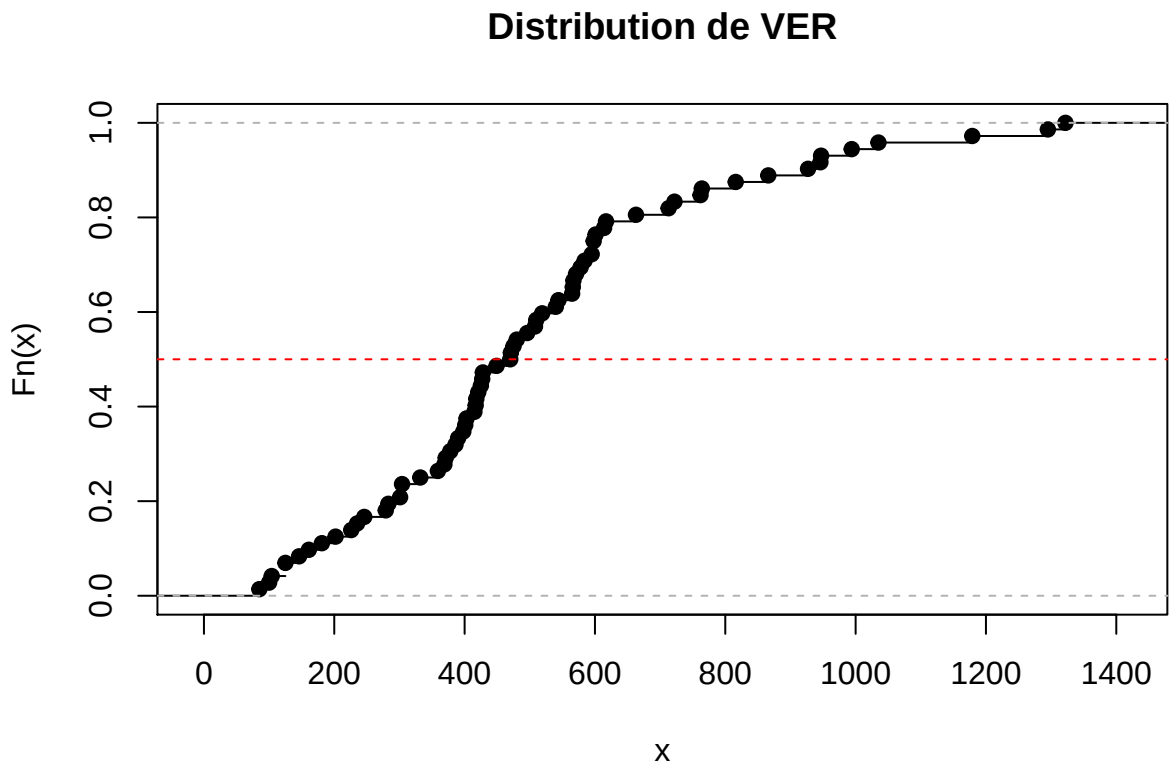
```
# églages des paramètres graphiques
op<-par(mfrow=c(1,4), oma=c(0.5,2,2,2),
        mai=c(0.4,0.6,0.6,0.15),mar=c(1.8,4,2.7,0.7))
# représentation de boîtes à moustaches de plusieurs variables
with(df.new,{
  boxplot(RO1,ylim=c(0,25), main="Après 1 sec")
  boxplot(RO2,ylim=c(0,25), main="Après 2 sec")
  boxplot(RO3,ylim=c(0,25), main="Après 3 sec")
  boxplot(RO4,ylim=c(0,25), main="Après 4 sec")
})
par(op)
title("Boîtes à moustaches des variables Rouge", line = -1, outer=TRUE)
```



Remarque : cette représentation n'est pas à confondre avec les boîtes à moustaches parallèles que nous allons voir dans une prochaine section et dont le principe est de croiser une variable quantitative avec une variable qualitative.

La fonction de répartition empirique Enfin, pour étudier la distribution d'une variable quantitative, on peut utiliser la fonction de répartition empirique qui s'obtient de la façon suivante :

```
plot(ecdf(df.new$VER), main="Distribution de VER")
abline(h=0.5, lty=2, col="red")
```



Interprétation : en abscisses, on représente les valeurs de X et en ordonnées, on lit les fréquences cummulées. Si on trace la droite d'équation $y = 0.5$, cela implique qu'il y a autant de valeurs de X situées en-dessous qu'au-dessus de la droite. La valeur de x qui vérifie $F_n(x) = 0.5$ n'est rien d'autre que la médiane.

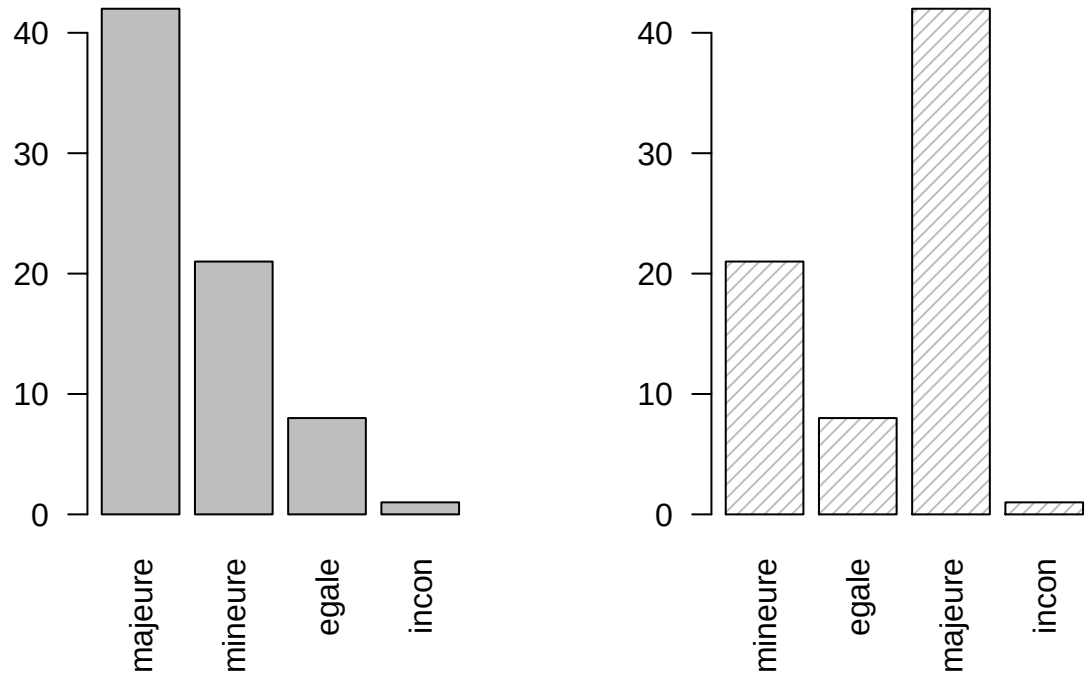
f. Représentation d'une variable qualitative

Les deux outils graphiques utilisés sont le diagramme en tuyaux d'orgues (*barplot* en anglais) et le camembert (*pie* en anglais).

Le diagramme en tuyaux d'orgues La fonction *plot()* appliquée à une variable qualitative (un objet de type **character** ou **factor**) renvoie automatiquement un diagramme en tuyaux d'orgues. En réalité, la fonction *plot()* appliquée à un objet **factor** fait appel à la fonction générique *plot.factor()* qui elle-même fait appel à la fonction *barplot()*. C'est donc dans l'aide de cette dernière qu'on trouvera les paramètres spécifiques à ce graphique (et aussi dans la fonction *par()* qui on le rappelle contient tous les paramètres graphiques communs à toutes les fonctions graphiques). La fonction *barplot()* s'utilise différemment de *plot.factor()* dans la mesure où *barplot* prend comme argument d'entrée un tableau de fréquences. Par exemple :

```
# paramètres graphiques
op<-par(mfrow=c(1,2))
```

```r
# plot.factor() s'applique sur un factor
plot(df.new$IMP,cex.lab=0.6, las=2)
# barplot() s'applique sur un objet table
barplot(table(factor(df.new$IMP,levels=c("mineure","egale","majeure","incon"))),
        cex.lab=0.6, las=2, density=20)
```




```r
par(op)
```

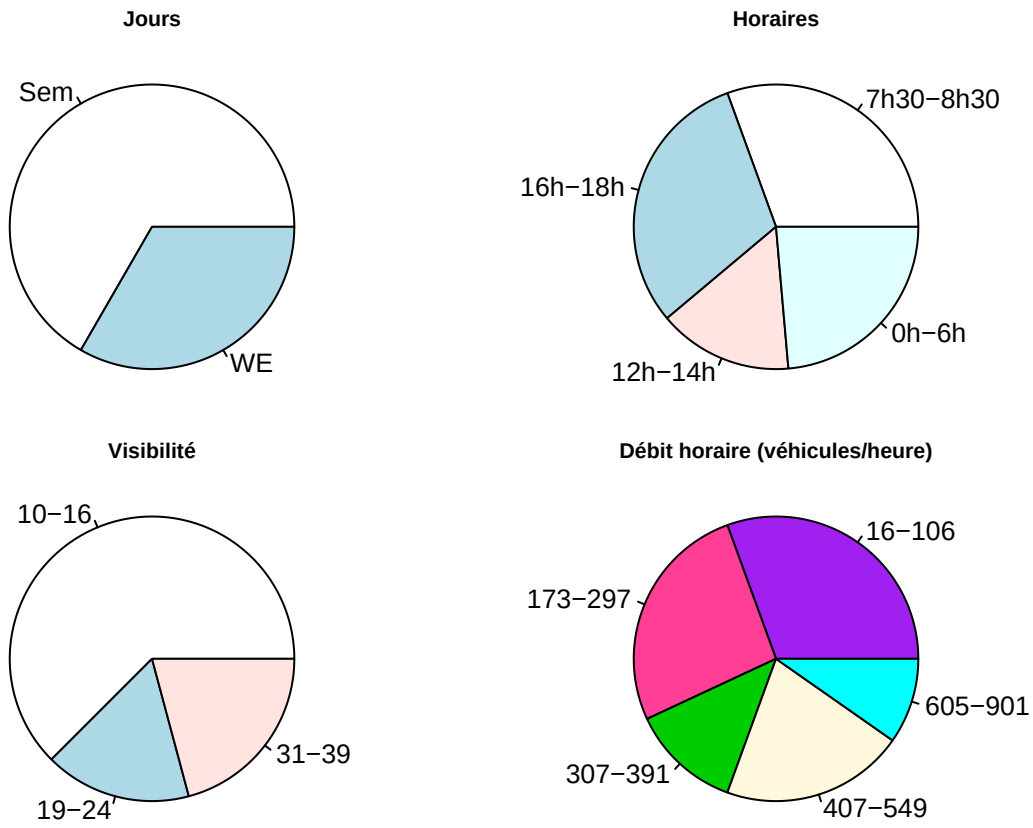
Remarque 1: l'option **las=2** nous a permis de représenter la légende de l'axe des abscisses en vertical. L'option **density=** permet d'hachurer les barres au lieu de les remplir.

Remarque 2: dans le cas où la variable est qualitative ordinale (c'est-à-dire qu'il y a une notion d'ordre entre les modalités), il peut être intéressant de trier les modalités pour représenter les modalités dans le bon ordre. C'est ce que nous avons fait dans le graphique de droite.

Le diagramme circulaire La fonction *pie(x)* permet de représenter un diagramme circulaire où **x** contient le tableau de fréquences (effectifs ou proportions) des modalités associées à la variable X. Voici un exemple :

```r
op<-par(mfrow=c(2,2),mar=c(0.5,0.5,1,1),mgp=c(1.5,.5,0),
  oma=c(0,0,0,0),cex.main=.8,cex.lab=.7,cex.axis=.7)
with(df.new,{
  pie(table(TYJ),main="Jours")
  pie(table(TrH),main="Horaires")
  pie(table(VIS),main="Visibilité")
```

```
  pie(table(DEB),main="Débit horaire (véhicules/heure)",
      col = c("purple", "violetred1", "green3", "cornsilk", "cyan"))
})
```




```
par(op)
```

2.2. Analyse bidimensionnelle

a. Croisement de deux variables qualitatives

La table de contingence permet de faire le résumé du lien entre deux variables qualitatives. Voici un exemple de table de contingence (variable X =**DEB**, croisée avec Y =**TrH**).

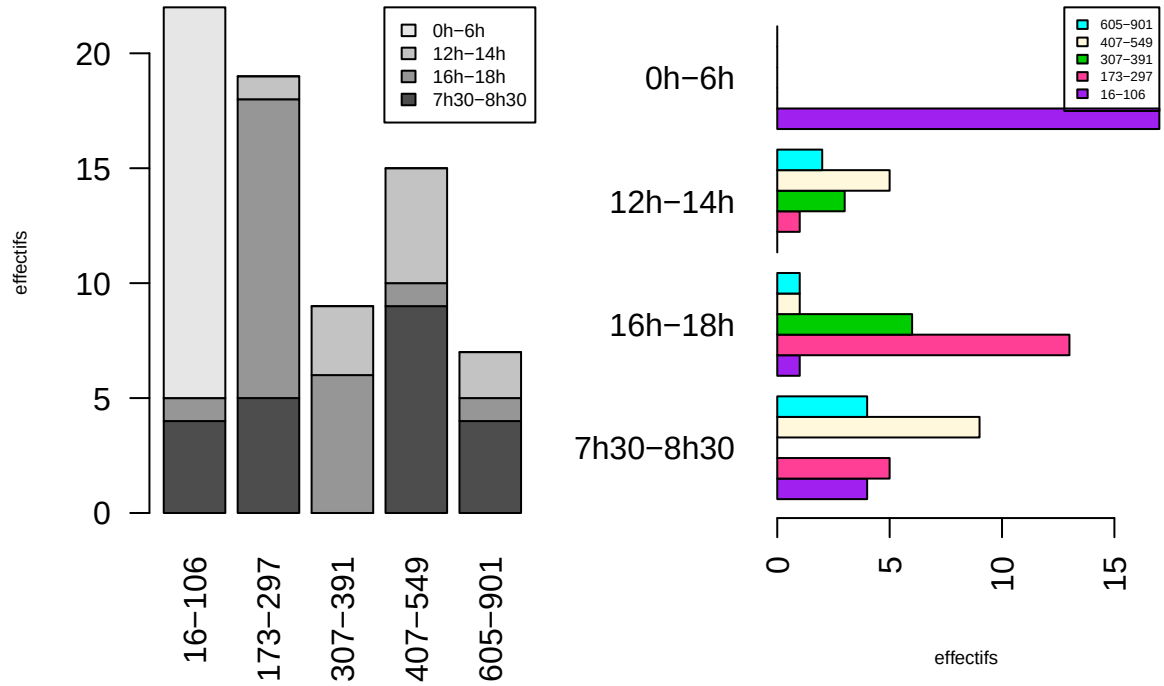
	7h30-8h30	16h-18h	12h-14h	0h-6h
16-106	4	1	0	17
173-297	5	13	1	0
307-391	0	6	3	0
407-549	9	1	5	0
605-901	4	1	2	0

Pour représenter cette table, on fera de nouveau appel à la fonction *barplot()* qui propose deux types de représentation selon l'option **beside=**.

```
# on construit la table de contingence
(tab<-table(df.new$DEB, factor(df.new$TrH)))
```

```
##
##           7h30-8h30 16h-18h 12h-14h 0h-6h
##   16-106          4       1       0    17
##   173-297         5      13       1     0
##   307-391         0       6       3     0
##   407-549         9       1       5     0
##   605-901         4       1       2     0
```

```
# paramètres graphiques
op<-par(mfrow=c(1,2))
# effectifs de Y en fonction de X
barplot(t(tab),cex.lab=0.6, las=2, legend.text=TRUE, ylab="effectifs",
        args.legend=list(x="topright",cex=0.6))
# effectifs de X en fonction de Y
barplot(tab, beside=TRUE, cex.lab=0.6, las=2, horiz=TRUE, legend.text=TRUE,
        col = c("purple", "violetred1", "green3", "cornsilk", "cyan"), xlab="effectifs",
        args.legend=list(x="topright", cex=0.45))
```

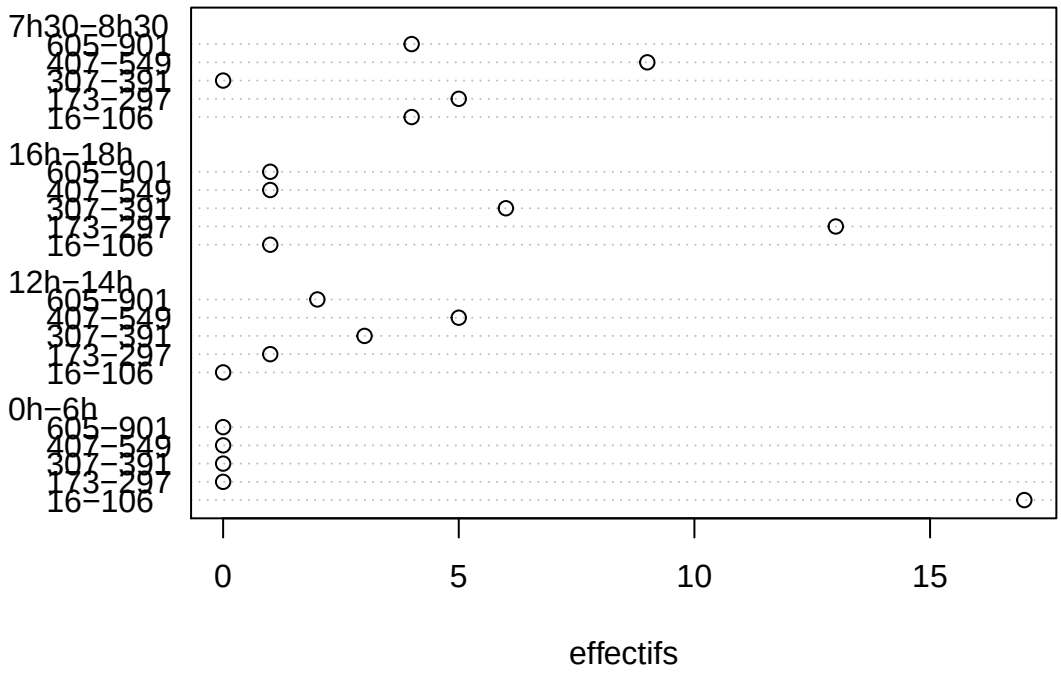


```
par(op)
```

Remarque : selon qu'on utilise **tab** ou **t(tab)** comme argument d'entrée, on permute l'emplacement des variables sur le graphique.

Il exite une alternative à la fonction *barplot()*. Il s'agit de la fonction *dotchart()* :

```
dotchart(tab, xlab="effectifs")
```

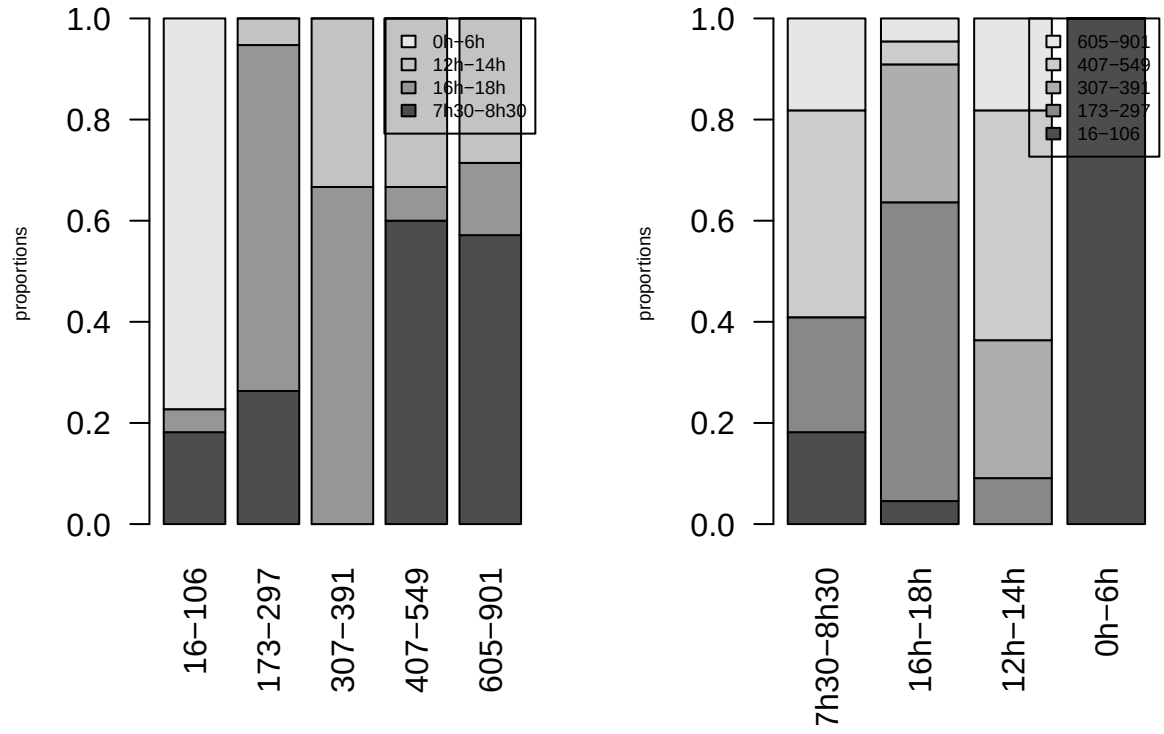


Interprétation statistique : pour étudier graphiquement le lien entre deux variables X et Y qualitatives, on a intérêt à représenter les profils-lignes (resp. profils-colonnes). Ceci permet de comparer la répartition (ici les fréquences) des modalités de Y (resp. X) en fonction des valeurs prises par X (resp. Y). Si la répartition des modalités de Y est la même quelque soit la modalité de X, alors les variables X et Y sont indépendantes. Si la répartition varie, cela signifie que X et Y sont liées. Une telle affirmation sera ensuite vérifiée (dans le chapitre suivant) par le test d'indépendance du χ^2.

Exemple : ici on représente les profils-lignes (resp. profils-colonnes) en divisant chaque ligne (resp. colonne) par la marge (i.e. la somme de la ligne (resp. colonne)). Cette opération se fait avec la fonction *prop.table()* en précisant l'option **margin=1** pour obtenir les profils-lignes et **margin=2** pour les obtenir les profils-colonnes. Ici, on voit clairement sur les deux graphiques que les bâtons de mêmes couleurs ne sont pas tous sur le même niveau, ce qui indique que les variables X et Y semblent liées.

```
op<-par(mfrow=c(1,2))
# représentation des profils-lignes
barplot(t(prop.table(tab,1)),cex.lab=0.6, las=2, legend.text=TRUE, ylab="proportions",
        args.legend=list(x="topright",cex=0.6))
```

```r
# représentation des profils-colonnes
barplot(prop.table(tab,2),cex.lab=0.6, las=2, legend.text=TRUE, ylab="proportions",
        args.legend=list(x="topright",cex=0.6))
```




```r
par(op)
```

b. Croisement de deux variables quantitatives

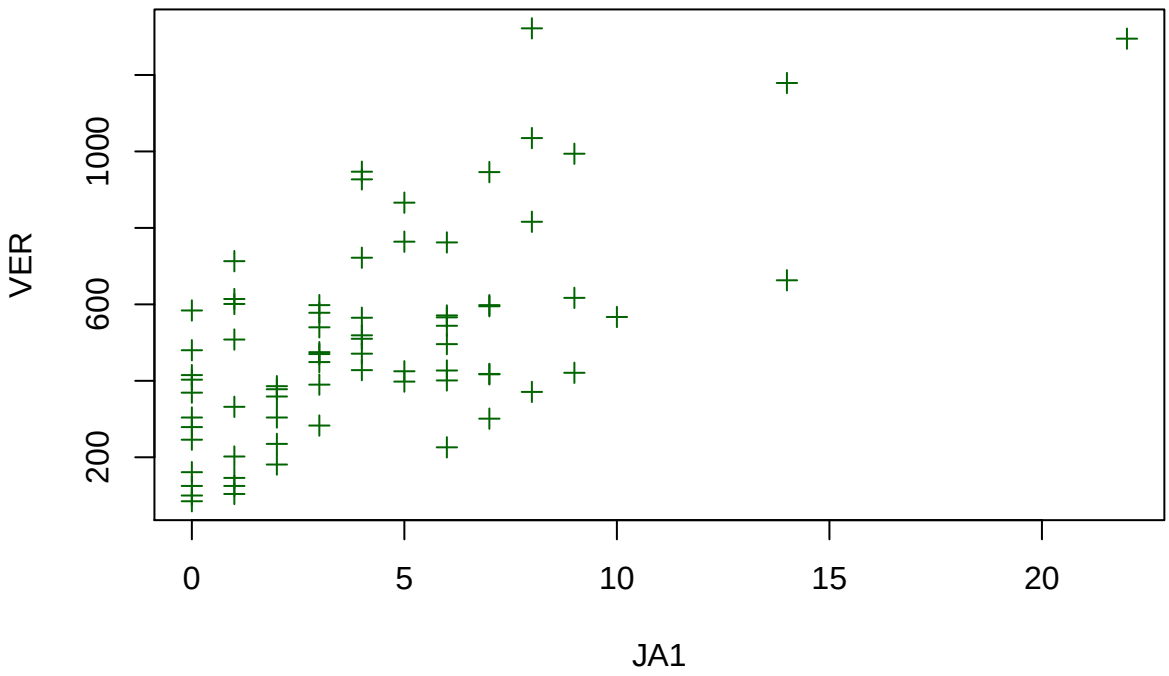
La fonction *plot(x,y, ...)* appliquée à deux vecteurs **x** et **y** de type **numeric** retourne le nuage de points de la variable Y en fonction de X. On a vu déjà vu dans la 1ère partie du cours certaines options disponibles :

- **col** permet de modifier la couleur des points.
- **pch** (entier compris entre 1 et 25) permet de modifier le symbole des points.

Lorsque les variables que l'on souhaite représenter sont incluses dans un **data.frame**, il peut être intéressant d'utiliser la syntaxe suivante *plot(y~x, data=nom.du.data.frame, ...)*. En effet **y~x** est une syntaxe reconnue et utilisée dans R pour définir un modèle du type "Y est expliquée par X". On utilise cette forme (**formula**) dans la plupart des modèles de régression.

Exemple 1 On va représenter le nuage de points de la variable **VER** en fonction de **JA1** :

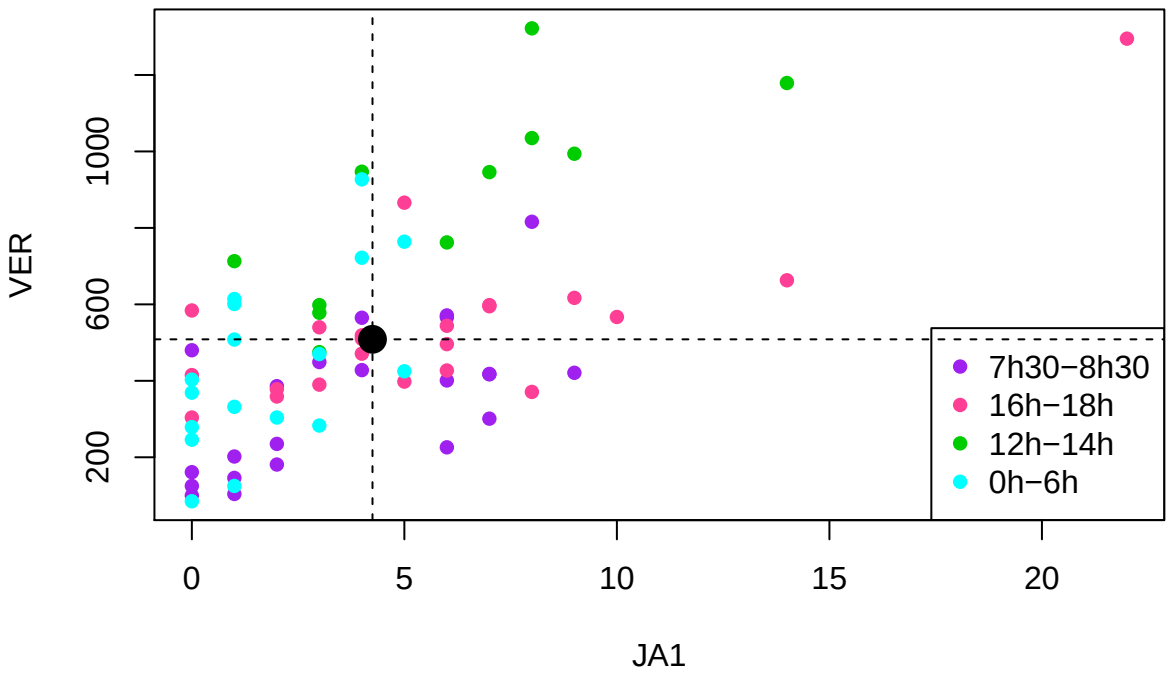
```r
plot(VER~JA1, data=df.new, col="darkgreen", pch=3)
```



Exemple 2
Au graphique précédent, on va ajouter deux informations supplémentaires :

- on va représenter les points avec des couleurs différentes selon la tranche horaire de l'observation (on représentera ainsi l'information sur 3 variables en même temps : 2 variables quantitatives et une variable qualitative).
- on va ajouter 4 quadrants centrés autour du barycentre du nuage de point.

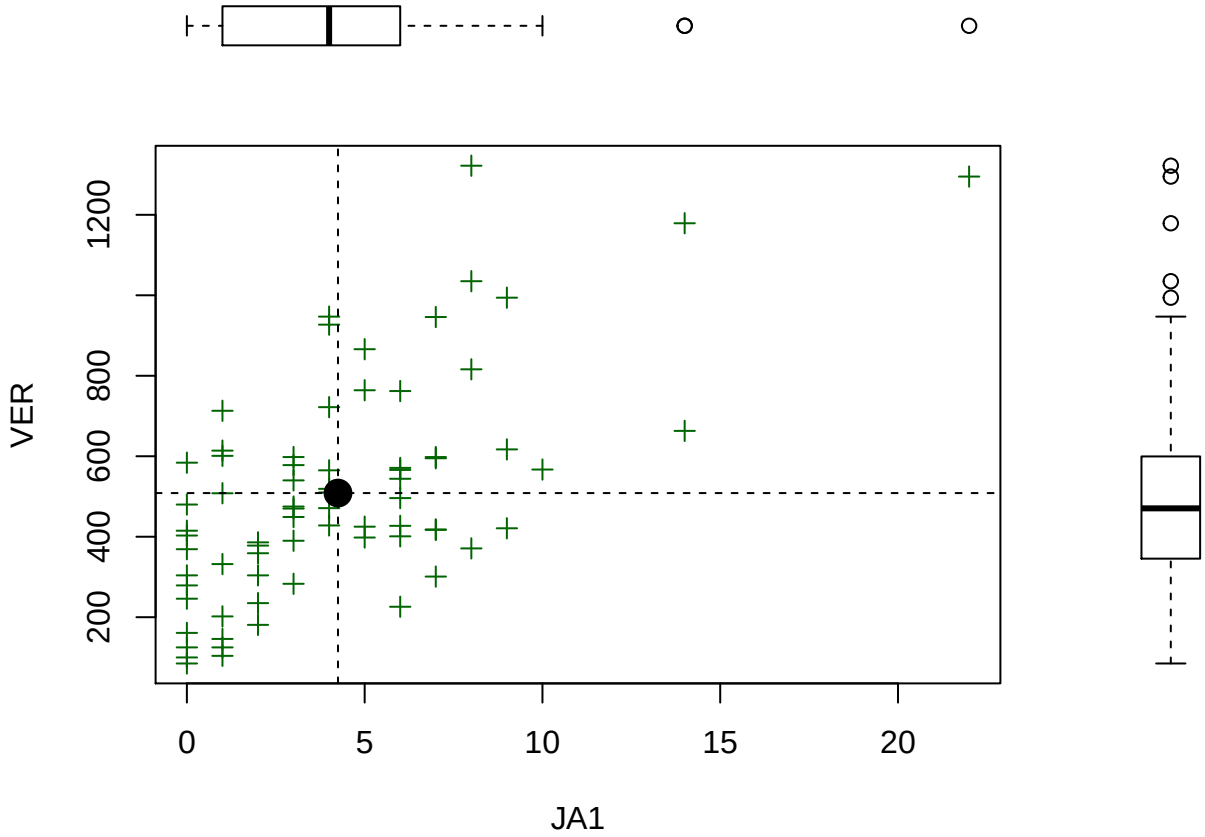
```
# on définit un vecteur de couleurs
vec.col=c("purple", "violetred1", "green3", "cyan")
# affichage du nuage de points
plot(VER~JA1, data=df.new, col=vec.col[as.numeric(df.new$TrH)], pch=16)
with(df.new, {
# représentation du centre de gravité
points(mean(JA1), mean(VER),pch=16,cex=2)
# représentation des quadrants
abline(h=mean(VER),lty=2)
abline(v=mean(JA1),lty=2)
# affichage de la légende
legend("bottomright", legend=levels(TrH),col=vec.col, pch=16)
})
```



Remarque : lorsqu'on fait **as.numeric()** sur une variable **factor**, on transforme les modalités de **factor** en une suite d'entier allant de 1 jusqu'à K où K est le nombre de modalités. En associant ce vecteur d'indices au vecteur des couleurs, ceci nous a permis d'attribuer à chaque modalité de **TrH** une couleur.

Exemple 3 : on va voir comment on peut ajouter au-dessus (respectivement à droite) du graphique les boîtes à moustaches de X (resp. Y). Pour cela, on utilise une autre technique que celle vue dans la première partie du cours pour séparer en 3 parties distinctes la fenêtre graphique.

```
# ouverture de la fenêtre graphique
par(fig=c(0,0.8,0,0.8),  mar=c(4,4,0,0))
# affichage du nuage de points
plot(VER~JA1, data=df.new, col="darkgreen", pch=3)
with(df.new, {
# représentation du centre de gravité
points(mean(JA1), mean(VER),pch=16,cex=2)
# représentation des quadrants
abline(h=mean(VER),lty=2)
abline(v=mean(JA1),lty=2)
# on reste dans la même fenêtre graphique
# mais dans une zone différente
par(fig=c(0,0.8,0.7,1), new=TRUE)
boxplot(JA1, horizontal=TRUE, axes=FALSE)
# on reste dans la même fenêtre graphique
# mais dans une zone différente
par(fig=c(0.75,1,0,0.8),new=TRUE)
boxplot(VER, axes=FALSE)
})
```



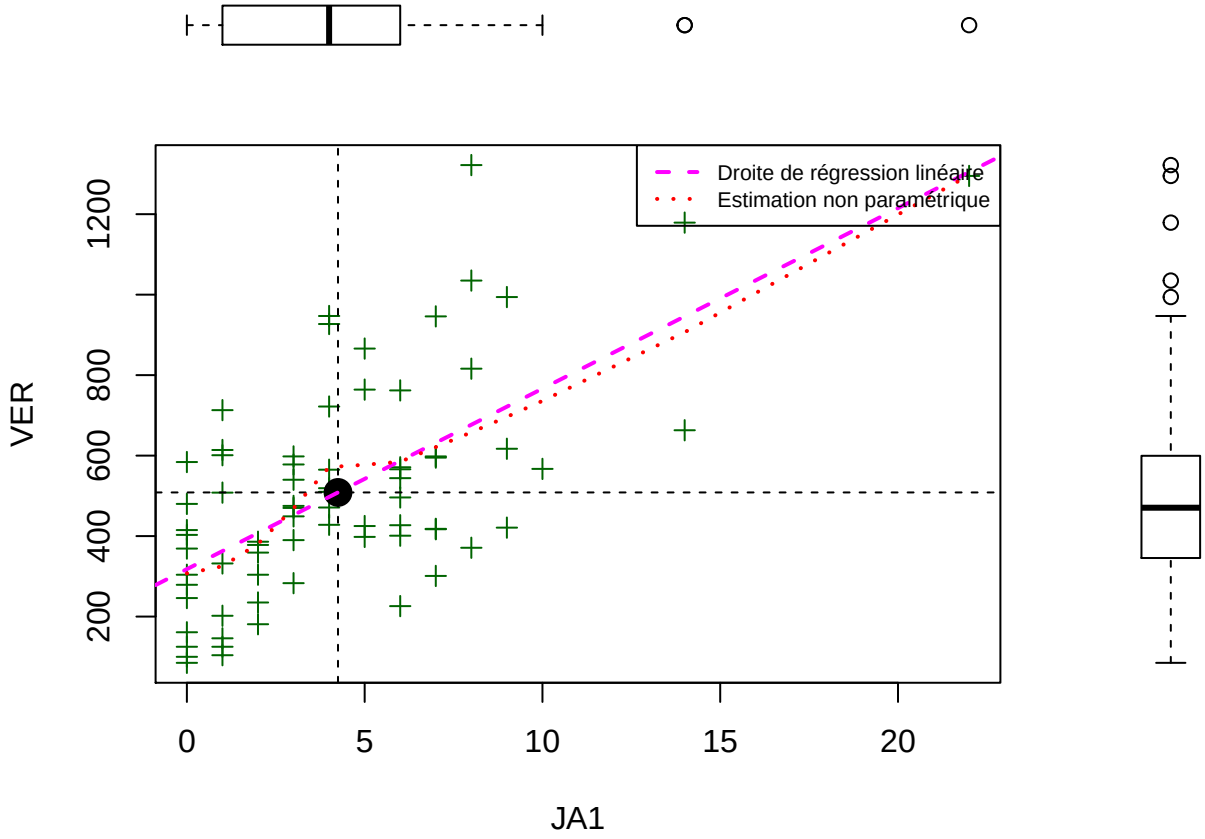
Remarque : cette méthode consiste donc à appeler plusieurs fois la fonction *par()* en modifiant les paramètres **fig=c(x1, x2, y1, y2)** (donne les dimensions de la zone de la fenêtre sur laquelle on va travailler, par défaut : **fig=c(0, 1, 0, 1)** donne la fenêtre graphique entière) et **new=TRUE** (permet d'indiquer qu'on continue à travailler dans la fenêtre graphique en cours d'utilisation).

Exemple 4 : enfin, on va ajouter au nuage de points la droite de régression linéaire de Y en fonction de X (fonction *lm()*) ainsi qu'une droite de régression non paramétrique (fonction *loess()*).

```
# régression linéaire
lm.res <- lm(VER~JA1, data=df.new)
# régression non paramétrique
lw1 <- loess(VER~JA1, data=df.new)

# ouverture de la fenêtre graphique
par(fig=c(0,0.8,0,0.8), mar=c(4,4,0,0))
plot(VER~JA1, data=df.new, col="darkgreen", pch=3)
with(df.new, {
# centre de gravité et quadrants
points(mean(JA1), mean(VER),pch=16,cex=2)
abline(h=mean(VER),lty=2)
abline(v=mean(JA1),lty=2)
# représentation de la droite de régression linéaire
abline(lm.res, col="magenta", lty=2, lwd=2)
# représentation de la courbe non paramétrique
lines(JA1[order(JA1)],lw1$fitted[order(JA1)],col="red",lty=3, lwd=2)
# affichage de la légende
legend("topright",c("Droite de régression linéaire","Estimation non paramétrique"),
```

```
  lty=2:3,lwd=2,col=c("magenta", "red"),cex=0.7)
# représentation des box plot
par(fig=c(0,0.8,0.7,1), new=TRUE)
boxplot(JA1, horizontal=TRUE, axes=FALSE)
par(fig=c(0.75,1,0,0.8),new=TRUE)
boxplot(VER, axes=FALSE)
})
```

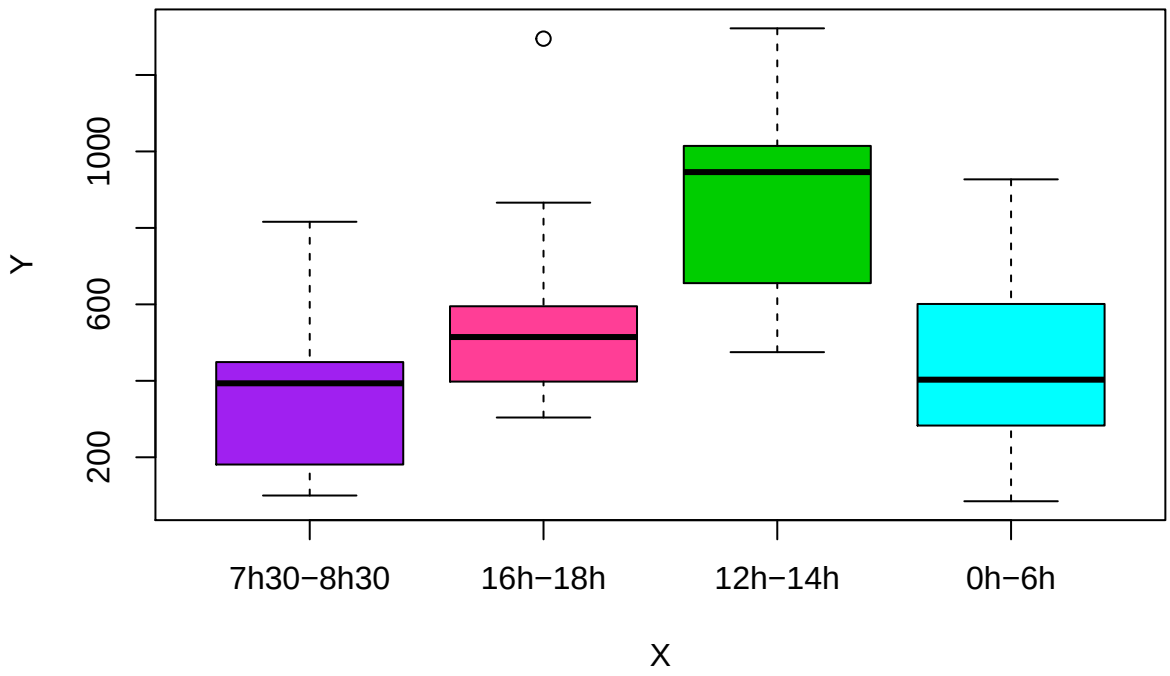


Interprétation statistique : on apprécie quel(s) type(s) de lien il existe entre X et Y. Ce lien est-il croissant, décroissant, ni l'un ni l'autre ? Ce lien peut-il s'apparenter à une fonction linéaire, parabolique, exponentielle, autres ? Est-ce que tous les points du nuage de points suivent le même lien ? Y-a-t-il des valeurs extrêmes ? C'est à cet ensemble de questions auxquelles il faut essayer de répondre lorsqu'on étudie ce type de graphique.

c. Croisement entre une variable quantitative et une variable qualitative

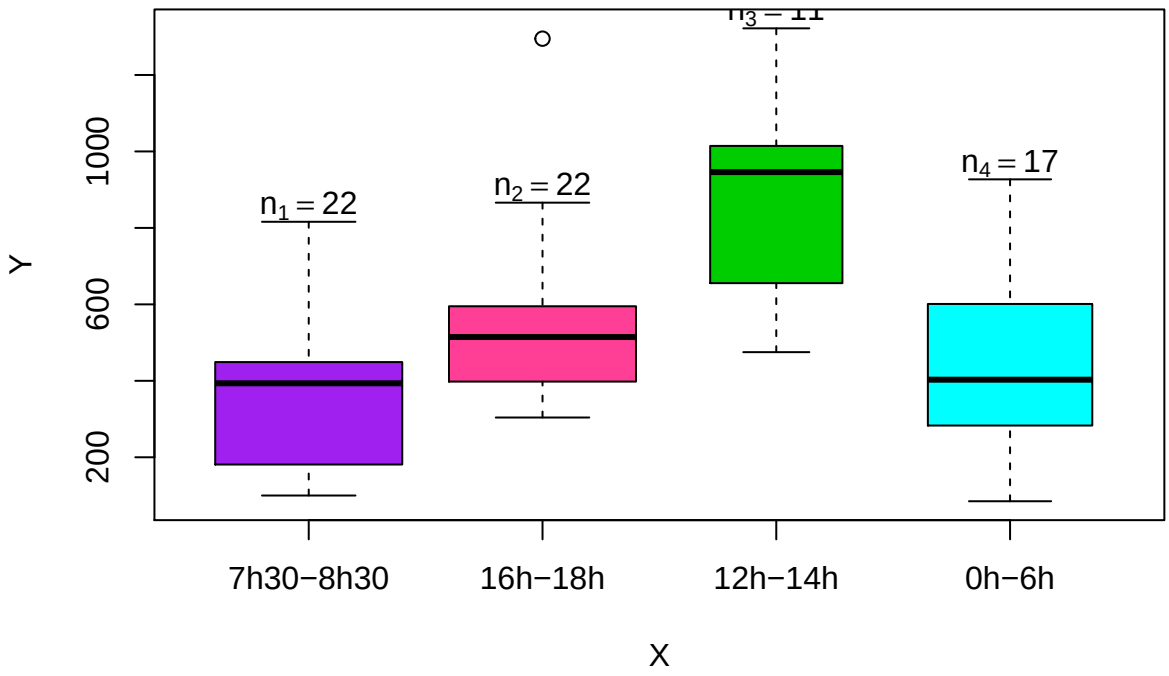
L'outil intéressant à utiliser est les boîtes à moustaches parallèles. Pour chaque modalité de la variable qualitative X, on représente la boîte à moustache de la variable quantitative Y. Pour réaliser ce graphique, on utilise la fonction *boxplot()* en précisant comme argument d'entrée une "**formula**".

```
boxplot(VER~TrH, data=df.new, col=vec.col, ylab="Y",xlab="X")
```



Remarque : contrairement au graphique dans lequel on avait représenter plusieurs variables Y côte à côte, ici il faut bien voir qu'on représente une seule variable quantitative qu'on divise en groupes. Chaque boîte contient donc un sous-ensemble de la population totale. Il peut donc être intéressant de représenter les boîtes par une largeur différente et proportionnelle à la taille du sous-échantillon correspondant. Par exemple :

```
# boxp plot et stockage des statistiques
res.box=boxplot(VER~TrH, data=df.new, col=vec.col, varwidth=TRUE,
                ylab="Y",xlab="X")
# représentation d'une légende à partir des statistiques
text(1:4,res.box$stats[5,],
     as.expression(c(bquote(n[1]==.(res.box$n[1])),
                     bquote(n[2]==.(res.box$n[2])),
                     bquote(n[3]==.(res.box$n[3])),
                     bquote(n[4]==.(res.box$n[4])))),
                     pos=3,offset=0.05)
```



Remarque : la fonction *boxplot()* appliquée à une **formula** retourne également un objet résumant les statistiques utilisées pour la construction du graphique.

Interprétation statistique : ce graphique permet d'apprécier visuellement si les variables Y et X sont liées entre elles. Autrement-dit, peut-on dire que le nombre de passage au feu vert dépend de l'horaire ? Le fait que les boîtes ne soient clairement pas situées au même niveau laisse suggérer qu'effectivement, selon les horaires, le nombre de passage au feu vert varie. Dans l'exemple ci-dessus, il semble que pendant le créneau 12h-14h, il y ait plus de passages que pendant les autres créneaux. On verra qu'on ne pourra affirmer cette hypothèse qu'en utilisant un test statistique.