HAI718 Probabilité et statistiques

Exploration de quelques fonctions de R

1 Quelques fonctions et structures de contrôles utiles

Exercice 1 *Tester les fonctions* `c`*, l'opérateur* `:`*, les fonctions* `seq` *et* `rep` *:*

```
> c(1,4,5)
> 1:6
> seq(1,5,by=0.5)
> rep(3,5)
```

Produire les séquences suivantes à la main et en utilisant les opérations précédentes :

```
1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
10   3   4   5   6  10 100 100  10  20  30  40

> c(1,2,3,1,2,3,1,2,3,1,2,3,1,2,3)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
> rep(1:3,5)
 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
> c(10,3,4,5,6,10,100,100,10,20,30,40)
 [1]  10   3   4   5   6  10 100 100  10  20  30  40
> c(10,3:6,10,rep(100,2),seq(10,40,by=10))
 [1]  10   3   4   5   6  10 100 100  10  20  30  40
```

Exercice 2 *Consulter l'aide de la fonction* `vector` *(rappel, on demande l'aide avec help ou en mettant un ? devant la fonction surlaquelle on demande l'aide). Tester :*

```
a <- vector(length=10, mode="numeric")
```

Comment produit-on un vecteur de caractères de taille 50 ?

```
> vector(length=50,mode="character")
 [1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
[26] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
```

Exercice 3 *Sélection d'éléments dans un vecteur : tester les commandes suivantes*

```
> x<-10:15
> x[2]
> x[c(2,4)]
> x[-4]

> x <- 10:15
# Créé le vecteur x des entiers de 10 à 15
> x
[1] 10 11 12 13 14 15
> x[2]
```

```
[1] 11
# Affiche l'élément d'indice 2 de x
> x[c(2,4)]
[1] 11 13
# Affiche les éléments d'indice 2 et 4 de x
> x[-4]
[1] 10 11 12 14 15
# Affiche tous les éléments de x sauf l'élément d'indice 4
```

1. *Créer un vecteur x contenant 1, 4 et 5*

   ```
   > x <- c(1,4,5)
   ```

2. *Créer un vecteur y contenant les chiffres impairs de 1 à 9*

   ```
   > y <- seq(1,9,by=2)
   > y
   [1] 1 3 5 7 9
   ```

   (a) *afficher le deuxième élément de y*

   ```
   > y[2]
   [1] 3
   ```

   (b) *afficher tous les éléments de y sauf le deuxième*

   ```
   > y[-2]
   [1] 1 5 7 9
   ```

3. *Créer un vecteur xy contenant le 1er, le 4ème et le 5ème élément de y (utiliser le vecteur x pour cela)*

   ```
   > xy <- y[x]
   > xy
   [1] 1 7 9
   ```

4. *Afficher les éléments 2 à 4 de y*

   ```
   > y[2:4]
   [1] 3 5 7
   ```

Exercice 4 *Tester :*

```
> x <- 1:3
> x**2
> x/c(2,4,6)
> y <- 1:5
> x*y
```

Dans la dernière instruction, que fait R et à quoi correspondent les chiffres donnés ?

```
> x <- 1:3
# Créé le vecteur x des entiers de 1 à 3
> x**2
[1] 1 4 9
# Met tous les éléments de x au carré
> x/c(2,4,6)
```

```
[1] 0.5 0.5 0.5
# Divise les éléments de x successivement par 2,4, et 6
> y <- 1:5
# Créé le vecteur y des entiers de 1 à 5
> x*y
[1]  1  4  9  4 10
Warning message:
longer object length
        is not a multiple of shorter object length in: x * y
```

La dernière instruction multiplie les éléments de x par les éléments de y, mais comme y est plus long que x, cela affiche un avertissement, et colle après x à nouveau les éléments de x pour avoir la même longueur que y.

Exercice 5 *Les fonctions* `sort, rev, order, rank`*.*

1. *Lire les aide correspondant à ces fonctions.*
2. *Comment afficher en ordre décroissant les éléments du vecteur* `1:10`*, avec chacune des trois premières fonctions ?*

```
> x <- 1:10
> sort(x,decreasing=T)
 [1] 10  9  8  7  6  5  4  3  2  1
> rev(x)
 [1] 10  9  8  7  6  5  4  3  2  1
> order(x,decreasing=T)
 [1] 10  9  8  7  6  5  4  3  2  1
```

Exercice 6 *Aggrégation :* `cbind, rbind`*. Tester :*

```
> a <- 1:2
> b <- 3:4
> rbind(a,b)
> cbind(a,b)
```

```
> a <- 1:2
# Créé le vecteur a composé des entiers 1 et 2
> b <- 3:4
# Créé le vecteur b composé des entiers 3 et 4
> rbind(a,b)
  [,1] [,2]
a    1    2
b    3    4
# Créé un tableau dont a et b sont les lignes (row -> le r de rbind)
> cbind(a,b)
     a b
[1,] 1 3
[2,] 2 4
# Créé un tableau dont a et b sont les colonnes (column -> le c de cbind)
```

Il est possible de filtrer des données dans un vecteur en utilisant la syntaxe suivante : `nom_vecteur[condition_a_verifier]`. Exemple :

```
> x <- 1:10
> y <- x[x>7]
> y
[1]  8  9 10
```

Ainsi, si on veut compter le nombre d'éléments positifs d'un vecteur, on utilise un filtre et la fonction de comptage `length`.

Exercice 7

1. *Engendrer un vecteur x de 100 éléments suivant la loi normale centrée réduite*
   La correction est faite à partir d'un vecteur de taille 10, pour plus de lisibilité...

   ```
   > vec <- rnorm(10)
   > vec
    [1] -0.4165318 -1.2238177 -0.3082954  0.3981509 -0.6784446 -1.4088679
    [7]  0.8814005  1.4249969 -1.0718534 -1.3720872
   ```

2. *Compter son nombre d'éléments strictement positifs*

   ```
   > vec[vec>0]
   [1] 0.3981509 0.8814005 1.4249969
   > length(vec[vec>0])
   [1] 3
   ```

3. *Prendre le logarithme de ce vecteur, dans un autre vecteur y (fonction* `log`*). Que constatez-vous ?*

   ```
   > y <- log(vec)
   Warning message:
   NaNs produced in: log(x)
   > y
    [1]        NaN        NaN        NaN -0.9209243        NaN        NaN
    [7] -0.1262432  0.3541696        NaN        NaN
   ```

   On ne peut pas prendre le logarithme d'une valeur négative. Lorsque c'est le cas, R remplace alors les données par "NaN" qui signifie "Not a Number". Autrement dit, une donnée manquante. Les données réelles comportent souvent des données manquantes, dont il faut soit se débarasser, soit tenir compte d'une façon ou d'une autre.

4. *Les données manquantes sont détectées par la fonction* `is.na`*. Compter le nombre de données manquantes de y*

   ```
   > y[!is.na(y)]
   [1] -0.9209243 -0.1262432  0.3541696
   > length(y[is.na(y)])
   [1] 7
   ```

5. *Prendre la moyenne de y avec la fonction* `mean`*. Quelle est l'option de cette fonction permettant de ne pas tenir compte des données manquantes ?*

   ```
   > mean(y)
   [1] NaN
   > mean(y,na.rm=T)
   [1] -0.2309993
   ```

2 Bibliothèques de données de R

R possède des bibliothèques prédéfinies de données statistiques. Elles peuvent être pratique lorsque l'on écrit un programme et que l'on veut disposer de jeux de données pour tester ce programme. On va s'en servir ici pour découvrir quelques fonctions sur les jeux de données (dataframes) en R. On travaillera sur le jeu de données `airquality`, disponible de base dans R. Stockez ce jeu de données dans une variable `air1`.

```
> air1 <- airquality
> air1
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
3      12     149 12.6   74     5   3
4      18     313 11.5   62     5   4
5      NA      NA 14.3   56     5   5
...
```

Remarque : Pour prendre connaissance de tous les jeux de données existants sous R, il suffit de taper `data()`. Une aide plus précise est disponible pour décrire chacun de ces jeux de données en tapant `help(nomdujeudedonnes)`.

2.1 Manipulations de base sur les dataframes

Pour récupérer une colonne, on utilise le caractère `$` : `air1$Ozone` produit le vecteur des données de la colonne Ozone. On peut aussi référencer une colonne par son numéro ou son nom : `air1[,1]`, `air1[,"Ozone"]`, idem pour une ligne : `air1[3,]`.

L'extraction d'une sous-base se fait avec la fonction `subset` qui prend en paramètre le jeu de donnée et le critère de sélection. Exemple :Le jeu de donnée extrait dont la colonne Temp a une valeur > 92 est produit grâce à :

```
> subset(air1,Temp>92)
    Ozone Solar.R Wind Temp Month Day
42     NA     259 10.9   93     6  11
120    76     203  9.7   97     8  28
121   118     225  2.3   94     8  29
122    84     237  6.3   96     8  30
123    85     188  6.3   94     8  31
126    73     183  2.8   93     9   3
127    91     189  4.6   93     9   4
```

ou, de la même manière que pour les vecteurs (notez que la sélection se fait sur les lignes ici, le critère sur la colonne étant laissé vide) :

```
> air1[air1$Temp>92,]
    Ozone Solar.R Wind Temp Month Day
42     NA     259 10.9   93     6  11
120    76     203  9.7   97     8  28
121   118     225  2.3   94     8  29
122    84     237  6.3   96     8  30
123    85     188  6.3   94     8  31
126    73     183  2.8   93     9   3
127    91     189  4.6   93     9   4
```

Pour transformer les données d'un jeu de données, on utilise la fonction `transform`, qui permet par exemple d'ajouter une colonne. Voici trois façons équivalentes de rajouter une colonne `logtemp` contenant le log de la température :

```
> air2 <- transform(air1,logTemp=log(Temp))
> air3 <- air1
> air3$logTemp <- log(air3$Temp)
> air4 <- cbind(air1,logTemp=log(air1$Temp))
```

Au passage, apprenons à nous servir d'une structure de contrôle fort utile en R, à savoir l'alternative `ifelse` :

```
> x <- 6:-4
> x
 [1]  6  5  4  3  2  1  0 -1 -2 -3 -4
> sqrt(x)
 [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000      NaN
 [9]      NaN      NaN      NaN
Warning message:
NaNs produced in: sqrt(x)
> sqrt(ifelse(x>=0,x,NA))
 [1] 2.449490 2.236068 2.000000 1.732051 1.414214 1.000000 0.000000       NA
 [9]       NA       NA       NA
```

Exercice 8 *Manipulation du dataframe* `airquality`

1. *Créer à partir de* `air1` *un jeu de données* `air2` *dans lequel :*
   — *Il n'y a plus de données manquante dans la colonne Ozone,*
   — *La température est* ≤ 94 *degrés Fahrenheit.*

   ```
   > air2 <- air1[(!is.na(air1$Ozone) & air1$Temp <= 94),]
   > air2
       Ozone Solar.R Wind Temp Month Day
   1      41     190  7.4   67     5   1
   2      36     118  8.0   72     5   2
   3      12     149 12.6   74     5   3
   4      18     313 11.5   62     5   4
   6      28      NA 14.9   66     5   6
   7      23     299  8.6   65     5   7
   8      19      99 13.8   59     5   8
   9       8      19 20.1   61     5   9
   11      7      NA  6.9   74     5  11
   12     16     256  9.7   69     5  12
   13     11     290  9.2   66     5  13
   ...
   ```

   La colonne 5 et la colonne 10 ont été filtrées à cause des données manquantes dans la colonne Ozone. Petit exercice supplémentaire : combien y avait-il de lignes pour lesquelles la température est > 94 ? [1].
2. *Créer à partir de* `air1` *un jeu de données* `air3` *dans lequel la donnée Ozone n'est pas manquante.*

1. Utiliser `length(air1[air1$temp >94, ]$Ozone)` par exemple

```
> air3 <- air1[!is.na(air1$Ozone),]
> air3
    Ozone Solar.R Wind Temp Month Day
1      41     190  7.4   67     5   1
2      36     118  8.0   72     5   2
3      12     149 12.6   74     5   3
4      18     313 11.5   62     5   4
6      28      NA 14.9   66     5   6
7      23     299  8.6   65     5   7
8      19      99 13.8   59     5   8
9       8      19 20.1   61     5   9
11      7      NA  6.9   74     5  11
...
```

Rajouter à `air3` *une colonne dont la valeur est 1 si*
— on est au mois de juin (6),
— et la température est > 78*,*
et 0 sinon. (L'égalité se teste avec `==` *et l'opérateur logique "et" se traduit par le symbole* `&`*)*

```
> air3 <- transform(air3,Macolonne=ifelse((air3$Month==6 & air3$Temp>78),1,0))
> air3
    Ozone Solar.R Wind Temp Month Day Macolonne
1      41     190  7.4   67     5   1         0
2      36     118  8.0   72     5   2         0
...
29     45     252 14.9   81     5  29         0
30    115     223  5.7   79     5  30         0
31     37     279  7.4   76     5  31         0
38     29     127  9.7   82     6   7         1
40     71     291 13.8   90     6   9         1
41     39     323 11.5   87     6  10         1
44     23     148  8.0   82     6  13         1
47     21     191 14.9   77     6  16         0
...
```

3. *Combien de lignes donnent la valeur 1 (obtenir l'information avec la fonction* `length` *et une sélection) ?*

```
> length(air3[air3$Macolonne == 1 ,]$Ozone)
[1] 4
```

Pour ordonner selon une ou plusieurs variables, on utilise la fonction `order` :

```
> air1[order(air1$Month,air1$Day),]
```

Exercice 9 *Sur le jeu de données* `iris`*, ordonner les données selon la longueur des sépales, puis la longueur des pétales.*

```
> help(iris)
...
Format:

     'iris' is a data frame with 150 cases (rows) and 5 variables
     (columns) named 'Sepal.Length', 'Sepal.Width', 'Petal.Length',
```

```
     'Petal.Width', and 'Species'.
...
> z <- iris[order(iris$Sepal.Length, iris$Petal.Length),]
> z
    Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
14           4.3         3.0          1.1         0.1     setosa
39           4.4         3.0          1.3         0.2     setosa
43           4.4         3.2          1.3         0.2     setosa
9            4.4         2.9          1.4         0.2     setosa
42           4.5         2.3          1.3         0.3     setosa
23           4.6         3.6          1.0         0.2     setosa
7            4.6         3.4          1.4         0.3     setosa
48           4.6         3.2          1.4         0.2     setosa
4            4.6         3.1          1.5         0.2     setosa
3            4.7         3.2          1.3         0.2     setosa
30           4.7         3.2          1.6         0.2     setosa
13           4.8         3.0          1.4         0.1     setosa
...
```

La fonction `match` permet de tester si une donnée est présente dans le jeu de données.

```
> month.name
 [1] "January"   "February"  "March"     "April"     "May"       "June"
 [7] "July"      "August"    "September" "October"   "November"  "December"
> match(c("Mai","May"),month.name,nomatch=0)
[1] 0 5
```

Exercice 10

1. *Que produit la commande suivante :*

   ```
   > month.name[match(air1$Month,1:12)]
   ```

   Cette commande renvoie le vecteur correspondant aux noms de mois des lignes de `air1`. Très précisément, `match(air1$Month,1:12)` est un vecteur qui renvoie le numéro correspondant au mois de chaque ligne de `air1`, et la commande globale est une sélection du nom de mois selon ce numéro. Le vecteur renvoyé a autant de lignes que `air1`.

   ```
   > match(air1$Month,1:12)
     [1] 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 6 6 6 6 6
    [38] 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 7 7 7 7 7 7 7 7 7 7 7
    [75] 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
   [112] 8 8 8 8 8 8 8 8 8 8 8 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9 9
   [149] 9 9 9 9 9
   > month.name[match(air1$Month,1:12)]
     [1] "May"       "May"       "May"       "May"       "May"       "May"
     [7] "May"       "May"       "May"       "May"       "May"       "May"
    [13] "May"       "May"       "May"       "May"       "May"       "May"
    [19] "May"       "May"       "May"       "May"       "May"       "May"
    [25] "May"       "May"       "May"       "May"       "May"       "May"
    [31] "May"       "June"      "June"      "June"      "June"      "June"
    [37] "June"      "June"      "June"      "June"      "June"      "June"
    [43] "June"      "June"      "June"      "June"      "June"      "June"
    [49] "June"      "June"      "June"      "June"      "June"      "June"
    [55] "June"      "June"      "June"      "June"      "June"      "June"
   ...
   ```

2. *Rajouter à* `air1` *une colonne avec le nom du mois.*

```
> air1 <- transform(air1,MonthName = month.name[match(air1$Month,1:12)])
> air1
    Ozone Solar.R Wind Temp Month Day MonthName
1      41     190  7.4   67     5   1       May
2      36     118  8.0   72     5   2       May
3      12     149 12.6   74     5   3       May
4      18     313 11.5   62     5   4       May
5      NA      NA 14.3   56     5   5       May
6      28      NA 14.9   66     5   6       May
7      23     299  8.6   65     5   7       May
8      19      99 13.8   59     5   8       May
9       8      19 20.1   61     5   9       May
10     NA     194  8.6   69     5  10       May
...
```

2.2 Fonctions statistiques

2.2.1 Un peu de graphiques

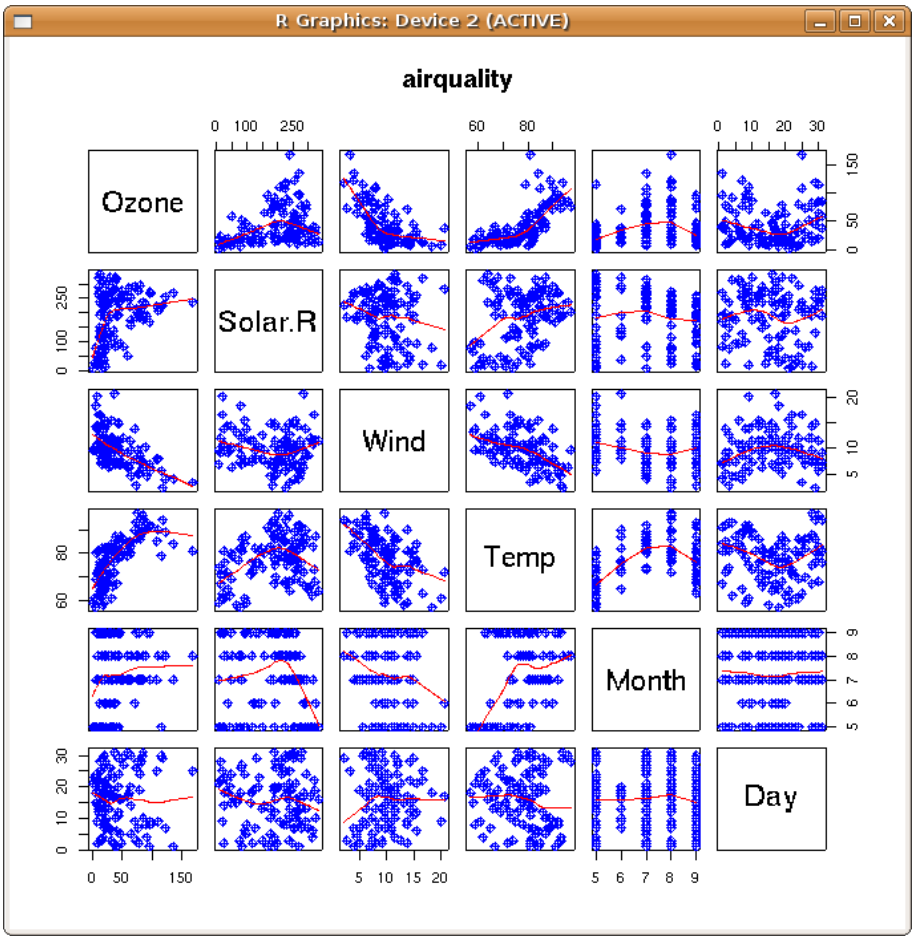
La fonction `pairs` permet de représenter les variables 2 à 2.

```
> data(airquality)
> is(airquality)
> air <- airquality
> names(air)
> help(air)
> air <- na.omit(air)
> pairs(air, panel = panel.smooth, main="airquality",col="blue",pch=3*3)
```

À noter que `is` donne le type d'objet, ici un data.frame. La commande `names` donne les noms des colonnes du jeu de données. L'aide est inactive sur `air`, en revanche elle sera active sur `airquality`.

```
airquality              package:datasets              R Documentation

New York Air Quality Measurements

Description:
     Daily air quality measurements in New York, May to September 1973.

Usage:
     airquality

Format:
     A data frame with 154 observations on 6 variables.

       '[,1]'  'Ozone'    numeric  Ozone (ppb)
       '[,2]'  'Solar.R'  numeric  Solar R (lang)
       '[,3]'  'Wind'     numeric  Wind (mph)
       '[,4]'  'Temp'     numeric  Temperature (degrees F)
       '[,5]'  'Month'    numeric  Month (1-12)
       '[,6]'  'Day'      numeric  Day of month (1-31)
```

La commande `na.omit` permet de nettoyer les données manquantes, car celles-ci sont assez incompatibles avec une bonne représentation graphique. Voici ce que donne la commande `pairs` sur cet exemple :



On pourrait représenter quatre variables sur un même graphique. On utilisera l'axe des abscisses et des ordonnées pour représenter les variables Ozone et Wind, la taille des points pour représenter la variable Temperature et la couleur pour la variable Month.

La taille des points d'un graphique est généralement comprise entre 0.1 et 2. Pour se mettre dans cette gamme et mieux visualiser l'effet des températures on effectue l'opération suivante. Observer le raccourci `air$T` pour `air$Temp`...

```
> air$T <- (air$T - min(air$T) +1)/10
```

On peut alors lancer le graphique après avoir changé la palette des couleurs.

```
> palette()
> palette(rainbow(5))
> palette()
```

```
> plot(air$O,air$W,cex=air$T,col=air$M-4, pch=16)
> text(air$O,air$W,lab=air$M,cex=0.5)
> title("Parametres atmospheriques (New-York, 1973)")
> legend(130,20,legend=c("Mai","Juin","Juillet","Aout","Sept."),fill=palette())
```

La commande `palette()` permet de manipuler les nuances de couleurs qui sont urilisées pour le graphique. La palette par défaut comprend tous les couleurs usuelles. La palette "rainbow" propose des couleurs pastels un peu moine agressives. Le codage des couleurs qui ne sont pas "nommées" se fait de la façon suivante : un caractère # suivi de 6 caractères entre 0 et F (donc dans $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\}$). C'est ce qu'on appelle un codage en héxadécimal. Les deux premiers caractères représentent la composante rouge de la couleur, les deux suivants la composante verte, et les deux derniers la composante bleue. Chaque composante est donc codée par deux "chiffres" entre 0 et F, ce qui donne 256 valeurs possibles. On peut donc utiliser en R $256 \times 256 \times 256 = 16777216$ couleurs sur un graphique... ce qui laisse une certaine marge !

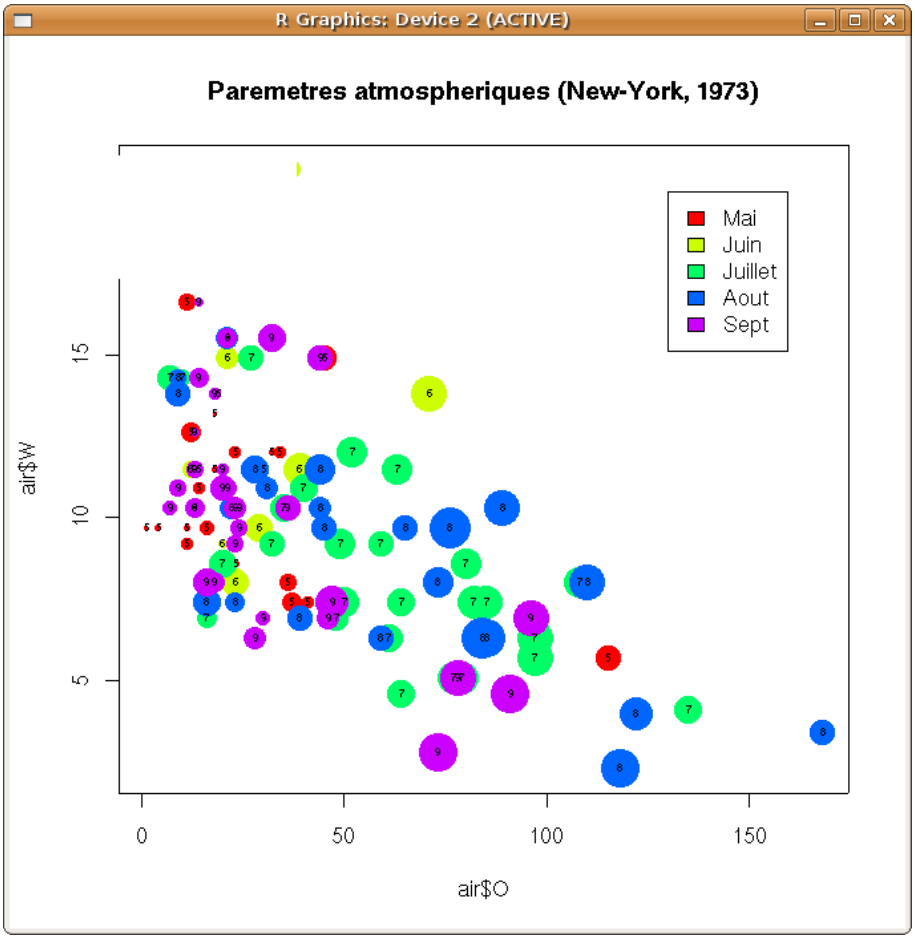
```
> palette()
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
> palette(rainbow(5))
> palette()
[1] "red"     "#CCFF00" "#00FF66" "#0066FF" "#CC00FF"
```

La commande `text` permet de rajouter les informations sur les données qui sont présentes dans ce graphique, à savoir les données en abscisse, en ordonnée, et la taille des points.

La commande `title` permet de rajouter, comme son nom l'indique, un titre au graphique.

La commande `legend` rajoute la petite légende avec les noms des mois et les couleurs qui leur correspondent. Il est à noter que rajouter d'autres types de données nuirait à la lisibilité du graphique. Il vaut mieux faire plusieurs "projections" pour mieux cerner les phénomènes intéressants.

Voici ce que donne le graphique produit :



2.2.2 Moyenne

La moyenne d'un vecteur s'obtient avec la fonction `mean`. La fonction `tapply` permet de calculer la moyenne selon certains paramètres.

Exercice 11 *Lire l'aide de cette fonction, et reprendre le jeu de données* `iris`. *Calculer la moyenne de la longueur des sépales pour chaque catégorie d'espèce.*

```
tapply                  package:base                  R Documentation

Apply a Function Over a "Ragged" Array

Description:
     Apply a function to each cell of a ragged array, that is to each
     (non-empty) group of values given by a unique combination of the
     levels of certain factors.

Usage:
     tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)

Arguments:
```

```
       X: an atomic object, typically a vector.

   INDEX: list of factors, each of same length as 'X'.  The elements
          are coerced to factors by 'as.factor'.

     FUN: the function to be applied.  In the case of functions like
...
```

L'aide de `tapply` nous indique que cette fonction applique une même fonction (troisième argument FUN), sur chaque groupe défini par le deuxième argument (INDEX) sur la donnée X. Dans notre cas, X sera la colonne `iris$Sepal.Length`, les groupes sont les lignes de même espèce, donc le facteur de regroupement est la colonne `iris$Species`, et la fonction est la fonction `mean` :

```
> tapply(iris$Sepal.Length,iris$Species, mean)
    setosa versicolor  virginica
     5.006      5.936      6.588
```

La fonction `mean` permet de gérer les données manquantes d'une façon différente de l'élimination pure et simple des lignes qui en contiennent : on peut remplacer ces données par la moyenne des autres données.

Exercice 12 *Réaliser cette opération sur la colonne Ozone de* `airquality` *(on utilisera la fonction* `apply` *simple).*

```
> apply(airquality, 2, function(x) ifelse(is.na(x), mean(x, na.rm = TRUE), x))
           Ozone  Solar.R Wind Temp Month Day
  [1,]  41.00000 190.0000  7.4   67     5   1
  [2,]  36.00000 118.0000  8.0   72     5   2
  [3,]  12.00000 149.0000 12.6   74     5   3
  [4,]  18.00000 313.0000 11.5   62     5   4
  [5,]  42.12931 185.9315 14.3   56     5   5
  [6,]  28.00000 185.9315 14.9   66     5   6
  [7,]  23.00000 299.0000  8.6   65     5   7
  [8,]  19.00000  99.0000 13.8   59     5   8
  [9,]   8.00000  19.0000 20.1   61     5   9
 [10,]  42.12931 194.0000  8.6   69     5  10
 [11,]   7.00000 185.9315  6.9   74     5  11
 [12,]  16.00000 256.0000  9.7   69     5  12
 [13,]  11.00000 290.0000  9.2   66     5  13
 [14,]  14.00000 274.0000 10.9   68     5  14
 [15,]  18.00000  65.0000 13.2   58     5  15
 [16,]  14.00000 334.0000 11.5   64     5  16
```

On remarque que les données 5 et 10 qui étaient manquantes, ont été remplacées par la moyenne du taux d'ozone sur toutes les observations.

Remarque : cette façon de faire applique la fonction sur toutes les colonnes de la donnée. Si on veut juste l'appliquer sur la colonne `Ozone`, on passe par autre chose que `apply`, par exemple une transformation :

```
> x <- ifelse(is.na(air1$Ozone), mean(air1$Ozone,na.rm=T), air1$Ozone)
> transform(air1, Ozone=x)
        Ozone Solar.R Wind Temp Month Day
1    41.00000     190  7.4   67     5   1
```

```
2    36.00000     118  8.0   72     5   2
3    12.00000     149 12.6   74     5   3
4    18.00000     313 11.5   62     5   4
5    42.12931      NA 14.3   56     5   5
6    28.00000      NA 14.9   66     5   6
7    23.00000     299  8.6   65     5   7
8    19.00000      99 13.8   59     5   8
9     8.00000      19 20.1   61     5   9
10   42.12931     194  8.6   69     5  10
11    7.00000      NA  6.9   74     5  11
12   16.00000     256  9.7   69     5  12
...
```

2.2.3 Ecart-type

La fonction `sd` permet d'obtenir l'écart-type d'un échantillon.

Exercice 13

1. *Produire un vecteur de 100 données suivant la loi normale de moyenne 4 et d'écart-type 0,5. Calculer la moyenne et l'écart-type sur cet échantillon.*

   ```
   > y <- rnorm(100,4,0.5)
   > mean(y)
   [1] 3.998185
   > sd(y)
   [1] 0.5036079
   ```

2. *Calculer l'écart relatif entre la moyenne et la moyenne empirique, ainsi que le rapport entre l'écart-type et l'écart-type empirique.*

   ```
   > abs(mean(y)-4)/4
   [1] 0.0004537526
   > sd(y)/0.5
   [1] 1.007216
   ```

2.2.4 Boucles

On veut généraliser l'expérience précédente et stocker les données dans un dataframe. Celui-ci devra comporter 6 colonnes : Moyenne, EcartType, MoyenneEmp, EcartTypeEmp, EcartRelMoyenne, RapportEcartType. Pour engendrer le vecteur Moyenne et le vecteur EcartType, on utilisera `runif`, en mutlipliant les valeurs par 10 pour la moyenne. On veut obtenir 50 lignes. Pour créer les 4 autres vecteurs, on utilise la fonction `vector`, cf. tout au début du TP.

Enfin, pour calculer les valeurs de chaque ligne, on utilise une boucle `for` :

```
for(i in 1:50){
   u<- rnorm(100);
   MoyenneEmp[i]<-mean(u);
   EcartTypeEmp[i] <- sd(u)
}
```

Exercice 14

1. *Créer tous les vecteurs demandés.*

```
> # Création du vecteur Moyenne
> Moyenne <- 10*runif(50)
> Moyenne
 [1] 8.2442804 4.2348969 2.5289714 9.2988966 3.1537276 7.7851286 1.4408052
 [8] 6.7880548 6.7093245 5.4291605 7.8862863 3.7353642 1.2981620 8.0086573
[15] 2.9913616 3.0866796 1.0063866 5.5635717 7.9404475 9.8719082 7.5963723
[22] 6.1157570 7.0715515 3.6504279 0.5042990 1.4331775 5.2967836 8.9044669
[29] 5.3097095 2.6032302 1.9607748 9.8602845 9.5723196 1.7780522 8.7115354
[36] 4.9565160 4.6282648 5.0651319 9.5295089 9.7372866 0.3906031 6.1733378
[43] 0.1184487 7.4734756 2.1748491 7.1850850 3.2771914 8.6379482 9.2587086
[50] 1.4332294
> # Création du vecteur EcartType
> EcartType <- runif(50)
> EcartType
 [1] 0.63581656 0.82601156 0.42052600 0.91729431 0.72713852 0.71595809
 [7] 0.18550466 0.30139110 0.06445308 0.77049144 0.07619294 0.79545703
[13] 0.32380628 0.22192903 0.39914539 0.13249283 0.31333704 0.87518837
[19] 0.87184087 0.35913189 0.95396694 0.61354677 0.67999697 0.70565332
[25] 0.28219749 0.89791563 0.68022567 0.40503556 0.79551933 0.51738639
[31] 0.50645828 0.69351789 0.85097510 0.09778479 0.56776518 0.81003192
[37] 0.38357913 0.40268050 0.87160954 0.01022521 0.09615604 0.38799214
[43] 0.73357969 0.34505798 0.89561279 0.79065320 0.22957124 0.93554720
[49] 0.87790275 0.47844460
> #Création des 4 autres vecteurs
> MoyenneEmp <- vector(50, mode="numeric")
> EcartTypeEmp <- vector(50, mode="numeric")
> EcartRelMoyenne <- vector(50, mode="numeric")
> RapportEcartType <- vector(50, mode="numeric")
```

2. *Réaliser le dataframe demandé (utiliser la fonction* `data.frame`*).*
   On remplit les 4 vecteurs dans une même boucle, dans laquelle on commence par se donner un échantillon de 100 données. Attention, la moyenne et l'écart-type sont ceux donnés par les vecteurs `Moyenne` et `EcartType`.

```
> # Remplissage des 4 vecteurs
> for(i in 1:50){
    u <- rnorm(100,mean=Moyenne[i],sd=EcartType[i]);
    MoyenneEmp[i] <- mean(u);
    EcartTypeEmp[i] <- sd(u);
    EcartRelMoyenne[i] <- abs(mean(u) - Moyenne[i])/Moyenne[i];
    RapportEcartType[i] <- sd(u)/EcartType[i];
 }
```

   On vérifie que les données ont bien été créées comme il faut :

```
> MoyenneEmp
 [1] 8.2879587 4.2128201 2.4813443 9.3351595 3.2318627 7.7988155 1.4325256
 [8] 6.8046489 6.7015230 5.3669749 7.8868624 3.6934454 1.3481049 8.0178892
[15] 3.0853325 3.0648236 1.0085582 5.3816303 7.9022461 9.8125739 7.5542711
[22] 6.0944338 7.0236671 3.6482773 0.4875401 1.3225840 5.3338975 8.9430153
[29] 5.1794658 2.6158853 2.0504310 9.8271664 9.5523811 1.7864487 8.8352091
[36] 4.9040682 4.5428069 5.0349606 9.5055718 9.7363483 0.3775100 6.2161371
[43] 0.1445930 7.4419211 2.2514177 7.2058381 3.2524914 8.7432514 9.3599029
```

```
[50] 1.4322929
> EcartTypeEmp
 [1] 0.54571919 0.89926974 0.47444851 0.84819548 0.71403486 0.73594517
 [7] 0.16802173 0.31307781 0.06338227 0.73354057 0.06800007 0.75265769
[13] 0.32484251 0.21474116 0.44339865 0.13071337 0.28360740 0.92089697
[19] 0.80370671 0.35894821 1.03584875 0.58047724 0.63144510 0.73038955
[25] 0.25030648 0.84872361 0.65966770 0.42803045 0.78257941 0.47606023
[31] 0.54108554 0.68793566 0.83083444 0.10100266 0.52589716 0.86970417
[37] 0.40626471 0.39998064 0.82627663 0.01023083 0.09932095 0.40532754
[43] 0.70202484 0.32242636 1.01651083 0.74254941 0.19371614 0.83441681
[49] 0.74774666 0.40985797
> EcartRelMoyenne
 [1] 5.103742e-03 2.449733e-03 1.391716e-02 3.010279e-03 1.568706e-03
 [6] 1.137554e-02 3.287462e-02 3.568617e-03 1.519425e-02 6.210131e-05
[11] 4.498144e-04 5.814619e-04 1.717676e-02 5.284777e-03 1.208788e-02
[16] 7.206683e-03 6.068019e-03 2.123916e-03 7.825986e-03 2.174964e-03
[21] 1.955221e-03 2.346363e-03 1.282762e-02 6.942735e-03 1.504265e-02
[26] 1.725691e-02 2.612632e-03 3.238759e-05 1.380421e-03 9.956774e-04
[31] 4.021095e-04 2.097953e-03 4.200793e-03 5.238365e-02 1.828674e-03
[36] 5.269163e-04 2.027624e-03 9.427980e-03 8.133514e-04 9.094620e-03
[41] 3.386138e-02 2.116745e-02 3.751939e-01 2.692022e-03 6.720612e-03
[46] 1.500925e-03 4.972029e-03 6.578903e-05 6.060780e-03 2.582929e-02
> RapportEcartType
 [1] 0.8582966 1.0886890 1.1282263 0.9246710 0.9819791 1.0279166 0.9057547
 [8] 1.0387759 0.9833862 0.9520425 0.8924721 0.9461953 1.0032002 0.9676119
[15] 1.1108700 0.9865694 0.9051193 1.0522272 0.9218502 0.9994885 1.0858330
[22] 0.9461010 0.9285999 1.0350544 0.8869904 0.9452153 0.9697777 1.0567725
[29] 0.9837340 0.9201252 1.0683714 0.9919509 0.9763323 1.0329076 0.9262582
[36] 1.0736665 1.0591419 0.9932953 0.9479894 1.0005491 1.0329144 1.0446798
[43] 0.9569851 0.9344121 1.1349892 0.9391594 0.8438171 0.8919024 0.8517420
[50] 0.8566467
```

Création du dataframe :

```
> blob <- data.frame(Moyenne,EcartType,MoyenneEmp,EcartTypeEmp,
EcartRelMoyenne,RapportEcartType)
> names(blob)
[1] "Moyenne"          "EcartType"        "MoyenneEmp"       "EcartTypeEmp"
[5] "EcartRelMoyenne"  "RapportEcartType"
> length(blob$Moyenne)
[1] 50
> head(blob,15)
    Moyenne  EcartType MoyenneEmp EcartTypeEmp EcartRelMoyenne RapportEcartType
1  8.244280 0.63581656   8.287959   0.54571919    0.0052980202        0.8582966
2  4.234897 0.82601156   4.212820   0.89926974    0.0052130618        1.0886890
3  2.528971 0.42052600   2.481344   0.47444851    0.0188325901        1.1282263
4  9.298897 0.91729431   9.335160   0.84819548    0.0038997055        0.9246710
5  3.153728 0.72713852   3.231863   0.71403486    0.0247754667        0.9819791
6  7.785129 0.71595809   7.798815   0.73594517    0.0017580769        1.0279166
7  1.440805 0.18550466   1.432526   0.16802173    0.0057465057        0.9057547
8  6.788055 0.30139110   6.804649   0.31307781    0.0024446021        1.0387759
9  6.709324 0.06445308   6.701523   0.06338227    0.0011627851        0.9833862
10 5.429161 0.77049144   5.366975   0.73354057    0.0114540081        0.9520425
11 7.886286 0.07619294   7.886862   0.06800007    0.0000730551        0.8924721
12 3.735364 0.79545703   3.693445   0.75265769    0.0112221530        0.9461953
```

```
13 1.298162 0.32380628   1.348105   0.32484251    0.0384719878        1.0032002
14 8.008657 0.22192903   8.017889   0.21474116    0.0011527479        0.9676119
15 2.991362 0.39914539   3.085333   0.44339865    0.0314141078        1.1108700
```

3 *Sauver le dataframe dans un fichier (exemple :* `write.table(blob,"blob.dat")` *).*

```
> write.table(blob,"blob.dat")
```