Travaux Dirigés ISV51 - Matrices, Listes, Tableaux de données - corrigé

Julien Chiquet

9 octobre 2015

Exercice 1: opérations algébriques élémentaires

1. Créer deux vecteurs u et v de taille 6. Calculer $u^\intercal v$, $uv^\intercal$ avec les commandes `%*%` puis les commandes `crossprod` et `tcrossprod`.

Tout d'abord, on définit les vecteurs u et v comme deux objets sans dimensions.

```
u <- 1:6; v <- 6:1
# les commandes (t)crossprod forcent les dimensions
crossprod(u,v)
```

```
##      [,1]
## [1,]   56
```

```
tcrossprod(u,v)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    6    5    4    3    2    1
## [2,]   12   10    8    6    4    2
## [3,]   18   15   12    9    6    3
## [4,]   24   20   16   12    8    4
## [5,]   30   25   20   15   10    5
## [6,]   36   30   24   18   12    6
```

Ensuite, on force les vecteurs u et v à être des objets matrices pour qu'ils s'apparentent à des vecteurs colonnes.

```
u <- as.matrix(u, ncol=1)
v <- as.matrix(v, ncol=1)
## dans ce cas, les opérateurs matricielles font sens
t(u) %*%v
```

```
##      [,1]
## [1,]   56
```

```
u %*% t(v)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    6    5    4    3    2    1
## [2,]   12   10    8    6    4    2
## [3,]   18   15   12    9    6    3
## [4,]   24   20   16   12    8    4
## [5,]   30   25   20   15   10    5
## [6,]   36   30   24   18   12    6
```

On remarque que la fonction `crossprod` peut également s'appliquer un seul objet

```
crossprod(u)
```

```
##      [,1]
## [1,]   91
```

```
crossprod(u,u)
```

```
##      [,1]
## [1,]   91
```

2. Créer une matrice A à 3 lignes et 2 colonnes à partie du vecteur u puis une matrice B à 2 lignes et 3 colonnes à partir du vecteur v. Effectuer un produit matriciel entre A et B puis entre B et A.

```
A <- matrix(u, 3,2)
B <- matrix(v, 2,3)
A %*% B
```

```
##      [,1] [,2] [,3]
## [1,]   26   16    6
## [2,]   37   23    9
## [3,]   48   30   12
```

```
B %*% A
```

```
##      [,1] [,2]
## [1,]   20   56
## [2,]   14   41
```

3. À partir des matrice A et B, calculer les produits scalaires entre les deux dernières lignes de A et la 1ère colonne de B.

```
A[2:3, ] %*% B[, 1]
```

```
##      [,1]
## [1,]   37
## [2,]   48
```

```
(A %*% B)[2:3,1, drop=FALSE]
```

```
##      [,1]
## [1,]   37
## [2,]   48
```

4. Concaténer les matrices A et $B^\intercal$ en colonne pour former une matrice 3×4. De même, concaténer $A^\intercal$ et B en ligne pour obtenir une matrice 4×3

```r
cbind(A, t(B))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    6    5
## [2,]    2    5    4    3
## [3,]    3    6    2    1
```

```r
rbind(t(A), B)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    6    4    2
## [4,]    5    3    1
```

Exercice 2: création et manipulation de matrices

1. **Données Iris**

a) Charger les valeurs numériques des données iris à l'aide de la commande

```r
data(iris)
## pour l'instant, on ne concerve que la partie numérique pour manipuler un objet matrice
iris <- as.matrix(iris[, -5])
```

b) Donner la dimension de la matrice ainsi construite. Trouver la plus grande valeur observée. Donner le numéro de ligne et de colonne correspondant.

```r
max(iris)
```

```
## [1] 7.9
```

```r
arrayInd(c(which.max(iris), which.min(iris)), dim(iris))
```

```
##      [,1] [,2]
## [1,]  132    1
## [2,]   10    4
```

Si on veux toutes les valeurs minimales

```r
arrayInd(which(iris == min(iris)), dim(iris))
```

```
##      [,1] [,2]
## [1,]   10    4
## [2,]   13    4
## [3,]   14    4
## [4,]   33    4
## [5,]   38    4
```

c) Calculer la moyenne en ligne et en colonne, d'abord avec les commandes `rowSums`, `colSums` et `nrow`, `ncol`, puis à l'aide de la commande `apply`. Quel individu à la plus grande longueur de Sépale ? Largeur de Pétale

```
colSums(iris)/nrow(iris)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##     5.843333     3.057333     3.758000     1.199333
```

```
head(rowSums(iris)/ncol(iris))
```

```
## [1] 2.550 2.375 2.350 2.350 2.550 2.850
```

Si on cherche l'individu minimal par attribut

```
apply(iris, 2, which.min)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width
##           14           61           23           10
```

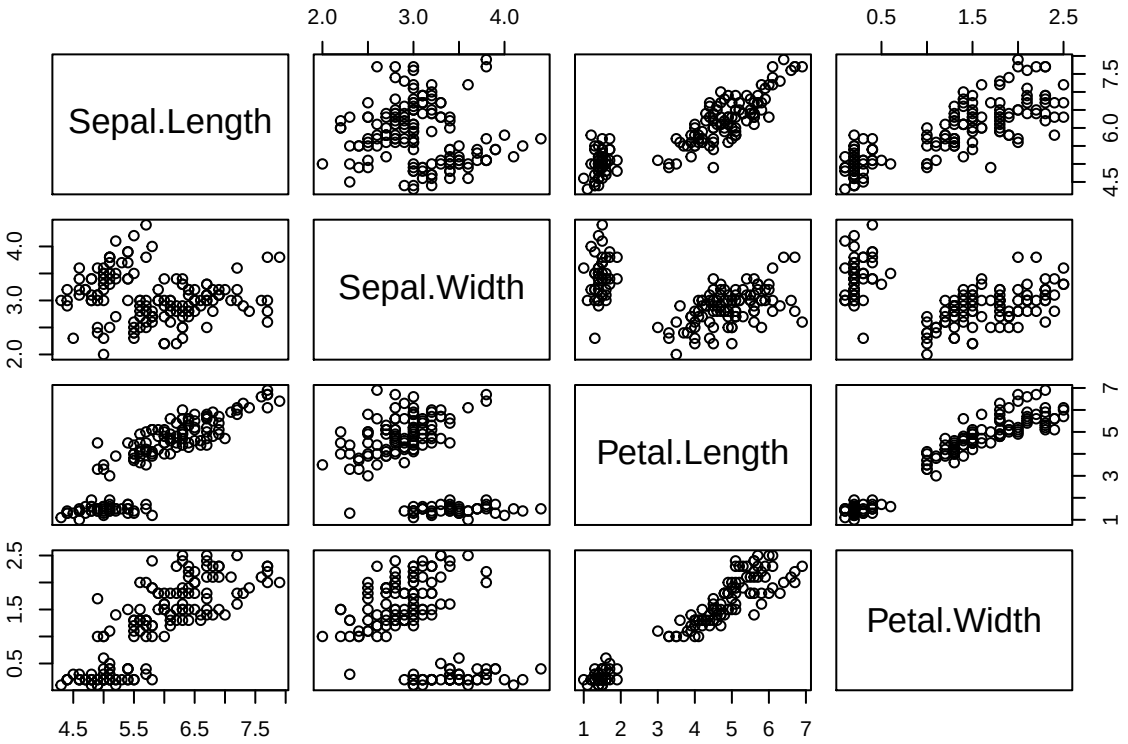
Et toutes les valeurs minimales par attribut

```
apply(iris, 2, function(x) which(x == min(x)))
```

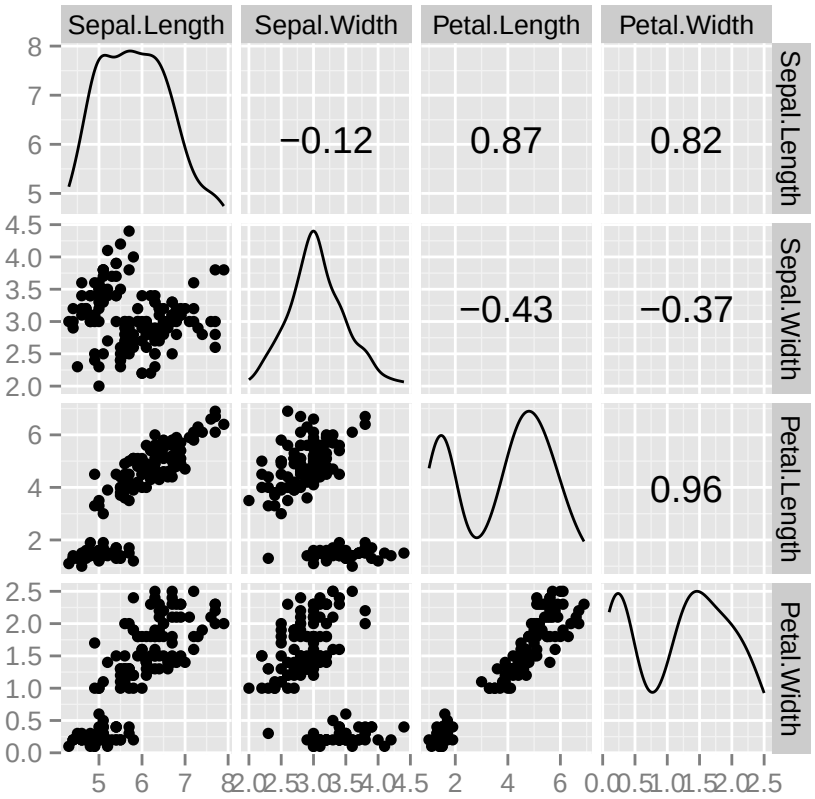
```
## $Sepal.Length
## [1] 14
##
## $Sepal.Width
## [1] 61
##
## $Petal.Length
## [1] 23
##
## $Petal.Width
## [1] 10 13 14 33 38
```

d) Représenter le graphe des paires de variable à l'aide de la commande `pairs`.

```
pairs(iris)
## plus joli
library(GGally)
```




```
ggscatmat(data.frame(iris))
```



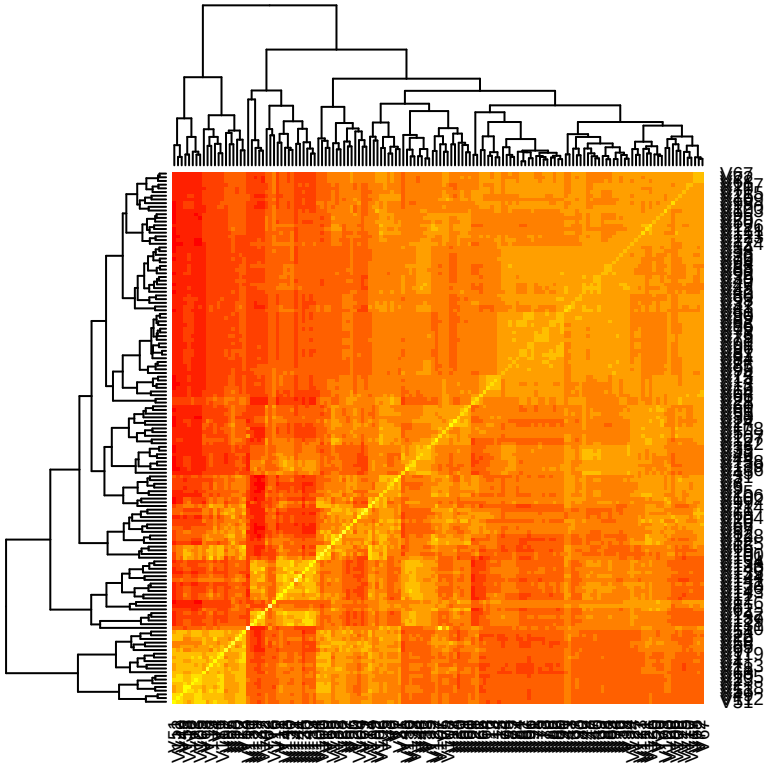
2. **Données microarray**

a) Charger les valeurs numériques des données de puces pour le cancer à l'aide de la commande

```
microarray <- as.matrix(read.table("http://statweb.stanford.edu/~tibs/ElemStatLearn/datasets/14cancer.x
```

b) Calculer la covariance entre les échantillons. Représenter le résultat sous forme d'image. Transformer la covariance en corrélation et représenter à nouveau cette image. Représenter ensuite le résultat de la fonction `heatmap`.

```
heatmap(cor(microarray))
```



Exercice 3: création et manipulation de listes

On utilise un programme permettant de calculer le nombre d'occurence des 4 nucléotides "a", "c", "g" et "t" dans une séquence d'ADN. Celui-ci renvoie une liste comportant 4 éléments, chacun étant un vecteurs décrivant les indices des occurrences des lettres correspondantes.

1. a) Considérons la séquence "AATTCCTCCCGTGACGAAATATA". Créer l'objet `R` correspondant à l'exécution du programme ci-dessus.

```
## déclaration de la chaîne de caractère
sequence <- strsplit("AATTCCTCCCGTGACGAAATATA", "")[[1]]
```

```
## solution 1
res <- list(where.A=which(sequence == "A"),
            where.C=which(sequence == "C"),
            where.G=which(sequence == "G"),
            where.T=which(sequence == "T"))
```

```
to.find <- c(A="A",C="C",G="G",T="T")
res <- lapply(to.find, grep, sequence)
```

b) Déterminer le nombre d'occurence de chaque lettre dans la séquence à partir de cette liste.

```
sapply(res, length)
```

```
## A C G T
## 8 6 3 6
```

2. a) On dispose maintenant de 3 chaînes "ATTCG", "CCGT" et "GCGAGG". Créer une liste comprenant 3 entrées, chacune étant une liste comme celle décrite aux deux questions précédentes.

```
chaines <- list(ch1 = strsplit("ATTCG" , "")[[1]],
                ch2 = strsplit("CCGT"  , "")[[1]],
                ch3 = strsplit("GCGAGG", "")[[1]])
```

b) Déterminer la longueur de chaque séquence à partir de cette liste

```
sapply(chaines, length)
```

```
## ch1 ch2 ch3
##   5   4   6
```

c) Déterminer le nombre d'occurence de chaque nucléotide dans chacune des listes. Renvoyer le résultat

```
sapply(chaines, function(ch) {
  sapply(lapply(to.find, grep, ch), length)
})
```

```
##   ch1 ch2 ch3
## A   1   0   1
## C   1   2   1
## G   1   1   4
## T   2   1   0
```

Une autre manière de faire est d'utiliser les facteurs:

```
## conversion de chaque chaînes en facteurs en imposant les niveaux
chaines <- lapply(chaines, function(x) factor(x, levels=to.find))
sapply(chaines, table)
```

```
##   ch1 ch2 ch3
## A   1   0   1
## C   1   2   1
## G   1   1   4
## T   2   1   0
```

Exercice 4: création et manipulation de data.frame

1. Charger le tableau de données `diamonds` (commande `data`). Vérifier qu'il s'agit bien d'un `data.frame`. Déterminer les noms des variables considérées et leur nature. Faites un résumé numérique.

```
library(ggplot2)
data("diamonds")
is.data.frame(diamonds)
```

```
## [1] TRUE
```

```
names(diamonds)
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

```
sapply(diamonds, is.factor)
```

```
##   carat     cut   color clarity   depth   table   price       x       y
##   FALSE    TRUE    TRUE    TRUE   FALSE   FALSE   FALSE   FALSE   FALSE
##       z
##   FALSE
```

```
summary(diamonds)
```

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

2. À l'aide de la commande `subset`, extraire les entrée du tableau telles que

- les diamands soit de qualité `Premium`

```
head(subset(diamonds, cut == "Premium"))
```

```
##    carat     cut color clarity depth table price    x    y    z
## 2   0.21 Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 4   0.29 Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 13  0.22 Premium     F     SI1  60.4    61   342 3.88 3.84 2.33
## 15  0.20 Premium     E     SI2  60.2    62   345 3.79 3.75 2.27
## 16  0.32 Premium     E      I1  60.9    58   345 4.38 4.42 2.68
## 27  0.24 Premium     I     VS1  62.5    57   355 3.97 3.94 2.47
```

- le carat soit supérieur à 3

```
head(subset(diamonds, carat > 3))
```

```
##       carat     cut color clarity depth table price    x    y    z
## 19340  3.01 Premium     I      I1  62.7    58  8040 9.10 8.97 5.67
## 21759  3.11    Fair     J      I1  65.9    57  9823 9.15 9.02 5.98
## 21863  3.01 Premium     F      I1  62.2    56  9925 9.24 9.13 5.73
## 22429  3.05 Premium     E      I1  60.9    58 10453 9.26 9.25 5.66
## 22541  3.02    Fair     I      I1  65.2    56 10577 9.11 9.02 5.91
## 22742  3.01    Fair     H      I1  56.1    62 10761 9.54 9.38 5.31
```

- le volume (approximatif) soit supérieur à $500 mm^3$

```
diamonds$volume <- diamonds$x*diamonds$y*diamonds$z
head(subset(diamonds, volume > 500))
```

```
##       carat     cut color clarity depth table price     x     y    z
## 23645  3.65    Fair     H      I1  67.1    53 11668  9.53  9.48 6.38
## 24068  2.00 Premium     H     SI2  58.9    57 12210  8.09 58.90 8.06
## 24132  3.24 Premium     H      I1  62.1    58 12300  9.44  9.40 5.85
## 24298  3.22   Ideal     I      I1  62.6    55 12545  9.49  9.42 5.92
## 24329  3.50   Ideal     H      I1  62.8    57 12587  9.65  9.59 6.03
## 25999  4.01 Premium     I      I1  61.0    61 15223 10.14 10.10 6.17
##          volume
## 23645  576.3973
## 24068 3840.5981
## 24132  519.1056
## 24298  529.2231
## 24329  558.0373
## 25999  631.8944
```

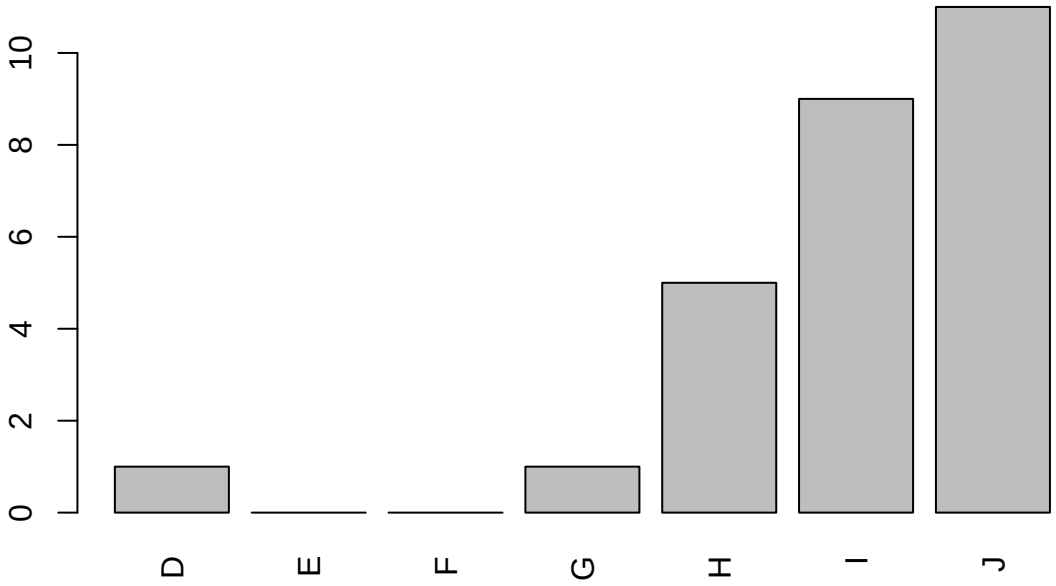
```
head(subset(diamonds, x*y*z > 500))
```

```
##       carat     cut color clarity depth table price     x     y    z
## 23645  3.65    Fair     H      I1  67.1    53 11668  9.53  9.48 6.38
## 24068  2.00 Premium     H     SI2  58.9    57 12210  8.09 58.90 8.06
## 24132  3.24 Premium     H      I1  62.1    58 12300  9.44  9.40 5.85
## 24298  3.22   Ideal     I      I1  62.6    55 12545  9.49  9.42 5.92
```

```
## 24329  3.50   Ideal     H      I1  62.8    57 12587  9.65  9.59 6.03
## 25999  4.01 Premium     I      I1  61.0    61 15223 10.14 10.10 6.17
##          volume
## 23645  576.3973
## 24068 3840.5981
## 24132  519.1056
## 24298  529.2231
## 24329  558.0373
## 25999  631.8944
```

- la qualité soit idéal, le prix inférieur à 1000 et le carat supérieur à .5. Déterminer la répartition

```
set <- subset(diamonds, cut == "Ideal" & price < 1000 & carat > 0.5 )
barplot(table(set$color), las=3)
```



3. Déterminer le prix moyen par classe de qualité. Même question par intervalle de carat (vous créerez une variable factorielle composée de 6 intervalled à l'aide de la fonction `cut`).

```
with(diamonds, tapply(price, cut, mean))
```

```
##      Fair      Good Very Good   Premium     Ideal
##  4358.758  3928.864  3981.760  4584.258  3457.542
```

```
aggregate(price~cut, diamonds, mean)
```

```
##         cut    price
## 1      Fair 4358.758
## 2      Good 3928.864
## 3 Very Good 3981.760
## 4   Premium 4584.258
## 5     Ideal 3457.542
```

```r
aggregate(price~cut+color, diamonds, mean)
```

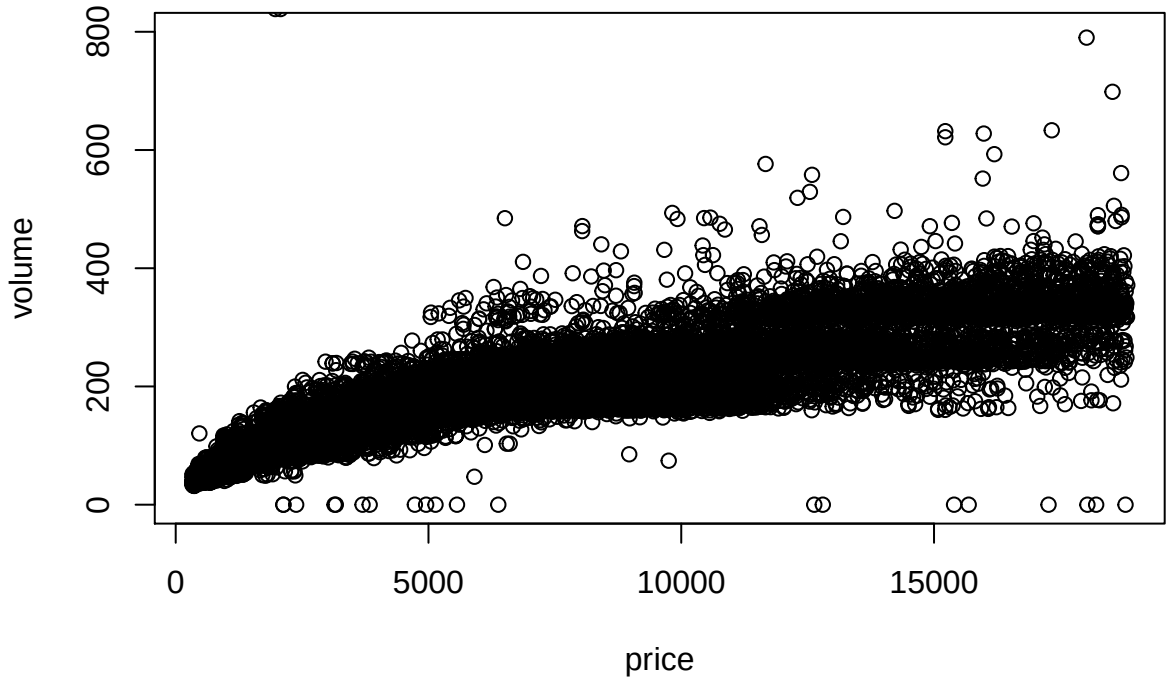
```
##          cut color    price
## 1       Fair     D 4291.061
## 2       Good     D 3405.382
## 3  Very Good     D 3470.467
## 4    Premium     D 3631.293
## 5      Ideal     D 2629.095
## 6       Fair     E 3682.312
## 7       Good     E 3423.644
## 8  Very Good     E 3214.652
## 9    Premium     E 3538.914
## 10     Ideal     E 2597.550
## 11      Fair     F 3827.003
## 12      Good     F 3495.750
## 13 Very Good     F 3778.820
## 14   Premium     F 4324.890
## 15     Ideal     F 3374.939
## 16      Fair     G 4239.255
## 17      Good     G 4123.482
## 18 Very Good     G 3872.754
## 19   Premium     G 4500.742
## 20     Ideal     G 3720.706
## 21      Fair     H 5135.683
## 22      Good     H 4276.255
## 23 Very Good     H 4535.390
## 24   Premium     H 5216.707
## 25     Ideal     H 3889.335
## 26      Fair     I 4685.446
## 27      Good     I 5078.533
## 28 Very Good     I 5255.880
## 29   Premium     I 5946.181
## 30     Ideal     I 4451.970
## 31      Fair     J 4975.655
## 32      Good     J 4574.173
## 33 Very Good     J 5103.513
## 34   Premium     J 6294.592
## 35     Ideal     J 4918.186
```

```r
bornes <- seq(min(diamonds$carat), max(diamonds$carat), len=7)
diamonds$carat.inter <- cut(diamonds$carat, bornes)
aggregate(price~carat.inter, diamonds, mean)
```

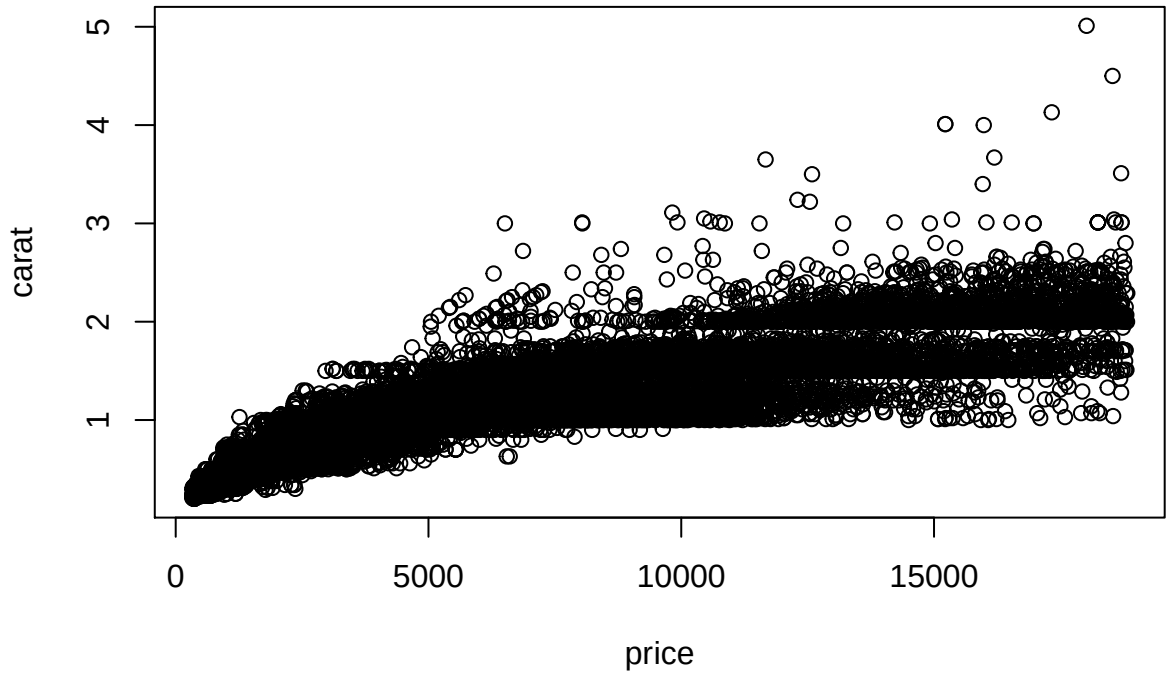
```
##   carat.inter     price
## 1     (0.2,1]  1787.419
## 2     (1,1.8]  7455.363
## 3   (1.8,2.6] 14731.454
## 4  (2.6,3.41] 14390.509
## 5 (3.41,4.21] 15363.500
## 6 (4.21,5.01] 18274.500
```

4. Tracer le volume en fonction du prix, le carat en fonction du prix. Représenter les boxplot de carat, prix et profondeur par classe de qualité et par couleur.

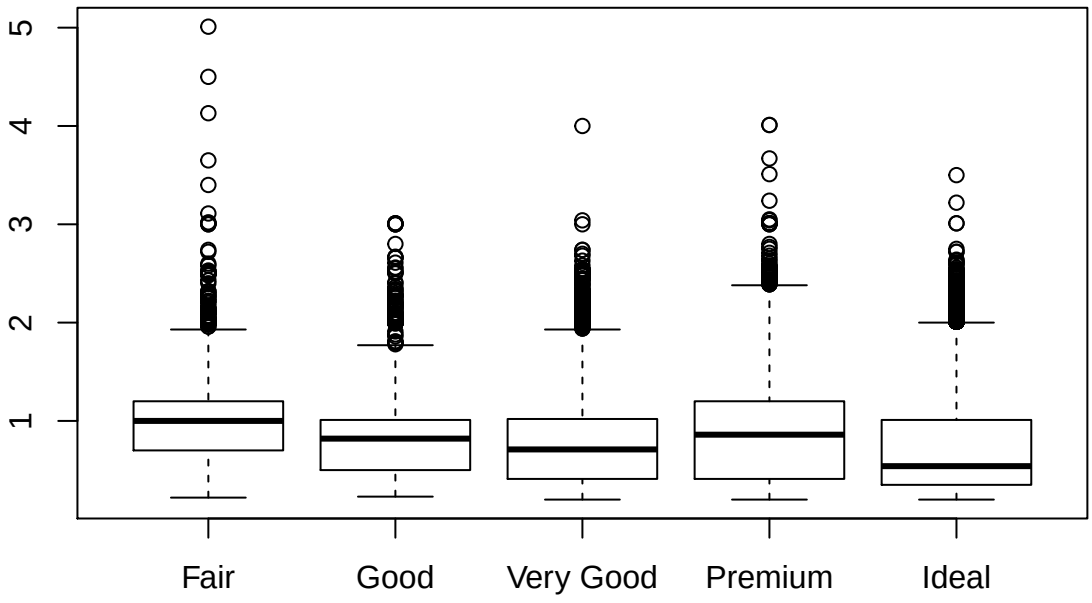
```
plot(volume~price, diamonds, ylim=c(0,800))
```



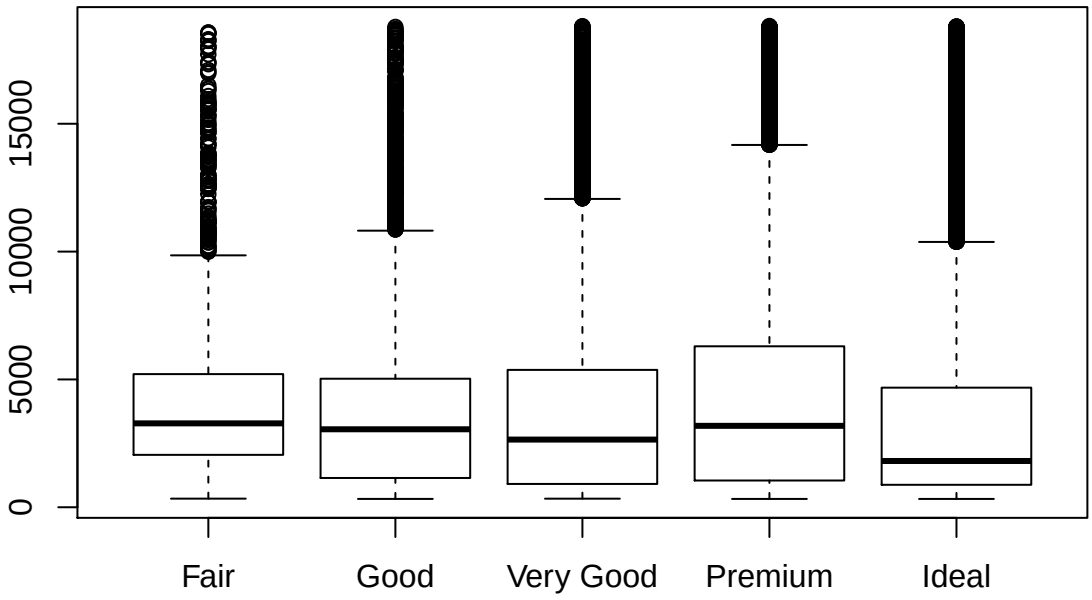

```
plot(carat~price, diamonds)
```



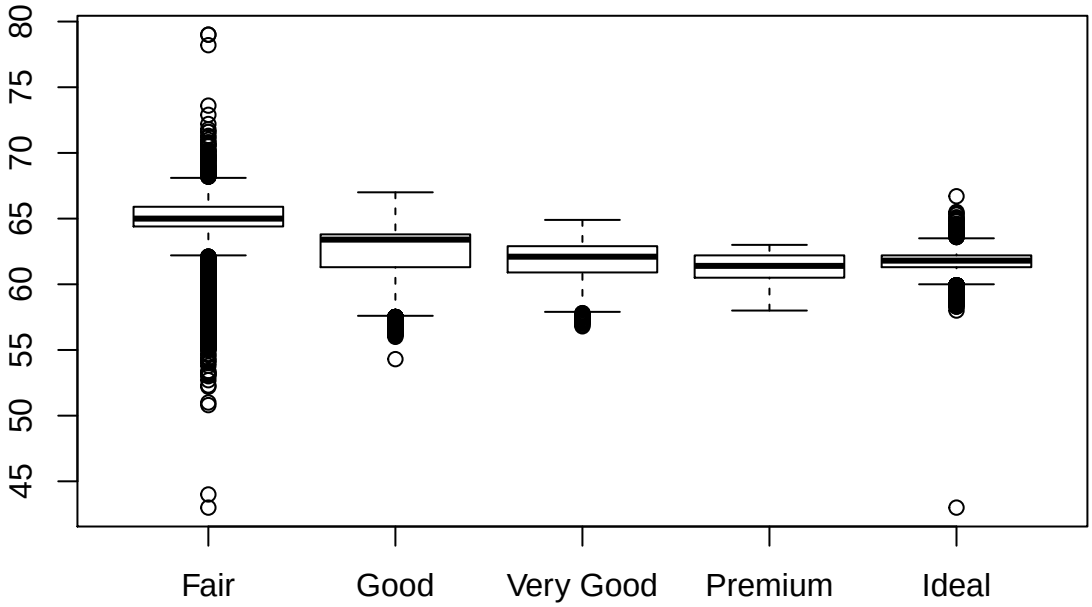

```
boxplot(carat~cut, diamonds)
```




```
boxplot(price~cut, diamonds)
```




```
boxplot(depth~cut, diamonds)
```



5. Pour chaque triplet (cut,color,clarity), renvoyer le prix moyen.

```
head(aggregate(price~cut+color+clarity,diamonds, mean))
```

```
##         cut color clarity    price
## 1      Fair     D      I1 7383.000
## 2      Good     D      I1 3490.750
## 3 Very Good     D      I1 2622.800
## 4   Premium     D      I1 3818.750
## 5     Ideal     D      I1 3526.923
## 6      Fair     E      I1 2095.222
```