Introduction à R, RStudio, Rmarkdown et tidyverse

Analyse des données

Master ISEFAR - M1

1. Présentation

À propos de R

R est un langage de programmation destiné aux statistiques et à la science des données. R a plusieurs avantages:

- il est multiplateforme: fonctionne aussi bien sur des sytèmes Linux, MacOS et Windows,
- il est gratuit,
- il est libre, développé par ses utilisateurs, diffusable et modifiable par tout un chacun.

Ce logiciel peut être téléchargé sur le site du CRAN (Comprehensive R Archive Network) à l'adresse suivante https://cran.r-project.org/. Il existe de nombreuses sources d'informations disponibles en ligne.

À propos de RStudio

RStudio est une sur-couche de R (également libre et gratuit) rendant son utilisation plus conviviale. Il peut-être téléchargée sur https://rstudio.com (après avoir téléchargé le logiciel R).

Lorsque vous ouvrez RStudio pour la première fois, l'interface est divisée par défaut en 3 fenêtres:

- la console interactive R qui sert à exécuter le code (à gauche)
- l'environnement / histoire qui contient l'espace de travail et l'historique des commandes (en haut à droite)
- Les Files / Plots / Packages / Help / Viewer (en bas à droite)

Une fois que vous ouvrez des fichiers, tels que des scripts R, l'éditeur s'ouvre au-dessus de la console. Il permet d'écrire du code et de le sauver.

À propos de Rmarkdown

Rmarkdown est un langage à balise qui permet des générer des rapports en mélangeant du texte, du code R et les résultats produits par ce code. En plus de produire des rapports agréables à lire (le code et ses résultats ne sont pas séparés des commentaires qui leur sont associés), son principal avantage est qu'il produit des rapports qui sont dynamiques et qui rendent les travaux reproductibles.

C'est un simple fichier texte dont l'extension est .Rmd. Voici un exemple très simple de document Rmarkdown:

```
---
  title: "Test R Markdown"
author: "Emilie Lebarbier"
date: "15 juin 2020"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```
```

Test R Markdown

Emilie Lebarbier

15 juin 2020

Codes R

Les blocs de code sont exécutés et leur résultat s'affiche juste après. Par exemple,

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

```
mean(cars$speed)
```

```
## [1] 15.4
```

Graphes

```
plot(cars$speed, cars$dist)
```

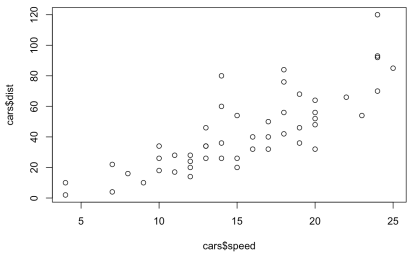


Figure 1:

```
## Codes R
Les blocs de code sont exécutés et leur résultat s'affiche juste après. Par exemple,
```{r}
summary(cars)
```

```{r}
mean(cars$speed)
```

Graphes
```{r, out.width = "60%",fig.align='center'}
plot(cars$speed, cars$dist)
```
```

Les documents peuvent être 'compilés' sous différents formats (HTML, PDF, Word). Voici le rendu du document précédent au format HTML :

À propos de `tidyverse`

Le terme `tidyverse` est une contraction de tidy (traduit comme "bien rangé") et de universe. Le `tidyverse` est un ensemble d'extensions pour R conçues pour fonctionner ensemble. Elle facilitent un très grand nombre d'opérations courantes dans R. Plusieurs packages constituent le coeur de `tidyverse`:

- ggplot2 (visualisation)
- dplyr (manipulation des données)
- tidyr (remise en forme des données)
- purrr (programmation)
- readr (importation de données)
- tibble (tableaux de données)

```
library(tidyverse)
```

Vous avez à disposition des documents synthétiques sur les grammaires des packages purrr, dplyr et tidyr, et ggplot2.

2. Prise en main de `R` et `Rstudio`

Console

Lorsque `RStudio` est lancé, l'utilisateur se trouve face à la console. C'est une fenêtre de texte qui commence par le signe >, signifiant que le logiciel attend les ordres. On peut commencer par taper quelques opérations simples:

```
2+3
```

```
## [1] 5
```

```
5-2
```

```
## [1] 3
```

```
3*7
```

```
## [1] 21
```

```
15/4
```

```
## [1] 3.75
```

```
5^2
```

```
## [1] 25
```

En utilisant les flèches ↑ et ↓, on peut rappeler les dernières commandes exécutées.

Objets

Objets simples

Il est également possible d'assigner une valeur à une variable. L'opérateur d'assignation est <-. Cette flèche stocke ce qu'il y a à sa droite dans un objet dont le nom est indiqué à sa gauche.

```
a <- 2
b <- 5
a
```

```
## [1] 2
```

```r
a-b
```

```
## [1] -3
```

```r
a+b
```

```
## [1] 7
```

Quand on assigne une nouvelle valeur à un objet déjà existant, la valeur précédente est perdue. Les objets n'ont pas de mémoire.

```r
x <- 2
x <- 5
x
```

```
## [1] 5
```

On peut aussi stocker du texte

```r
x <- "bonjour"
x
```

```
## [1] "bonjour"
```

Vecteurs

Un vecteur est une collection d'entités de même nature.

```r
# numérique
x0 <- 0
x1 <- c(-1,23,98.7)
x2 <- c(156,164,197,160,1182,145,177)

# caractère
y0 <- "bonjour"
y1 <- c("Pomme","Flore","Alexandre")

# logique
z0 <- TRUE
z1 <- c(FALSE,TRUE,FALSE,TRUE,TRUE)
```

L'opérateur : permet de générer rapidement un vecteur comprenant tous les nombres entre deux valeurs:

```r
numbers <- 1:10
```

On peut accéder au 2ème élément par l'instruction

```r
numbers[2]
```

```
## [1] 2
```

On peut concaténer des vecteurs de même type

```r
vect <- c(x1,x2)
vect <- c(x0,x1,x2)
```

Lorsqu'on applique une opération à un vecteur, celle-ci s'applique à toutes les valeurs qu'il contient.

```r
x1+10
```

```
## [1]   9.0  33.0 108.7
```

```r
x2/4
```

```
## [1]  39.00  41.00  49.25  40.00 295.50  36.25  44.25
```

```r
x2^2
```

```
## [1]   24336   26896   38809   25600 1397124   21025   31329
```

```r
x3 <- 6:8
x4 <- 2:4
x3/x4
```

```
## [1] 3.000000 2.333333 2.000000
```

Comparer les choses

On peut faire des comparaisons

```r
1 == 1 # égalité (lire comme "est égal à")
```

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

```r
## [1] TRUE
1 != 2 # inégalité (lire comme "n'est pas égal à")
```

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

```r
## [1] TRUE
1 <2 # moins que
```

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

```r
## [1] TRUE
1 <= 1 # inférieur ou égal à
```

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

```r
## [1] TRUE
1> 0 # plus grand que
```

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

Fonctions

```r
# retourne la nature des entités du vecteur
class(x1) # ou mode(x1)
```

```
## [1] "numeric"
```

```r
class(y1)
```

```
## [1] "character"
```

```r
# longueur du vecteur, i.e. nombre de composantes:
length(x2)
```

```
## [1] 7
```

```r
# moyenne
mean(x2)
```

```
## [1] 311.5714
```

```r
# écart-type
sd(x2)
```

```
## [1] 384.1835
```

```r
# logarithme
log(x2)
```

```
## [1] 5.049856 5.099866 5.283204 5.075174 7.074963 4.976734 5.176150
```

```r
# pour créer des vecteurs
x5 <- rep(1, times=5)
x6 <- rep(c("A","B","C"),each=3)
x6 <- rep(c("A","B","C"),c(3,2,4))
x7 <- seq(from=1,to=19, by=2)
```

Une fonction peut avoir plusieurs arguments. Considérons un vecteur qui contient une données manquantes, souvent codées par NA. La fonction `mean`retourne dans ce cas NA. On peut cependant indiquer à la fonction `mean` d'effectuer le calcul en ignorant les valeurs manquantes:

```r
x.NA <-c(156,164,197,160,1182,145,177,NA)
# même vecteur que x2 mais avec une données supplémentaire NA
mean(x.NA)
```

```
## [1] NA
```

```r
mean(x.NA,na.rm=TRUE)
```

```
## [1] 311.5714
```

Chercher de l'aide

Pour chercher de l'aide sur une fonction spécifique

```r
?nom_fonction
# ou
help(nom_fonction)
```

Cela chargera une page d'aide dans l'onglet `Help` (en bas à droite).

Installer et charger un package

`R` est un logiciel très vivant car de nouvelles fonctionnalités lui sont sans cesse ajoutées au travers de petits ensemble de fonction regoupées dans un package. Ainsi chaque personne est libre de contribuer au développement du logiciel. Ces package sont regroupés sur le CRAN et sont accessibles à tous.

Pour installer un package,

- aller dans l'ongle `Package`
- cliquer sur l'onglet `Install`
- une fenêtre s'ouvre, indiquer le nom du package à installer dans `Package`
- cliquer sur `Install` Le package sera installé une bonne fois pour toute (installé sur le disque dur de votre ordinateur).

Une fois le package installé, il faudra, à chaque fois que vous ouvrez `R`, le "charger" avant de pouvoir utiliser les fonctions qu'il propose. Ceci se fait avec la fonction `library`. Par exemple,

```r
library(rmarkdown)
```

Attention

Vous pouvez taper des commandes directement dans la console, mais elles seront oubliées lorsque vous fermerez la session. Deux solutions:

1. entrer les commandes dans un éditeur de script et le sauvegarder. Pour cela, cliquer sur `File -> New File -> R Script`. Vous pouvez:

- utilisez le signe # pour commenter (tout ce qui se trouve à droite d'un # est ignoré par `R`)
- cliquez sur le bouton Run situé au-dessus du panneau de l'éditeur pour exécuter la ligne en cours et des lignes sélectionnées. Un raccourci consiste à placer le curseur sur la ligne à exécuter ou sélectionner les lignes d'intérêt et faire Ctrl-Entrée.

2. créer un document `Rmarkdown` (cf paragraphe suivant).

3. Document Rmarkdown

Création

Pour créer un document Rmarkdown , cliquer sur `File -> New File -> R Markdown` et une boîte de dialogue s'ouvre:

- donner les informations nécessaires à la création de votre document: titre, auteur, format de par défaut. Après avoir appuyé sur "OK", un fichier .Rmd apparaît dans l'éditeur. Il contient les informations que vous lui avez donné ainsi que du code et du texte par défaut.
- enregistrez le.

Pour générer le fichier associé, cliquer sur `Knit` et choisissez le format que vous voulez. Un fichier avec l'extension du format donné portant le même nom que fichier.Rmd est alors créé dans le même directory et en même temps le rapport généré s'ouvre dans le viewer de `RStudio`.

La personnalisation du document généré se fait en modifiant des options dans le préambule du document. RStudio propose néanmoins une petite interface graphique permettant de changer ces options plus facilement. Pour cela, cliquez sur l'icône en forme d'engrenage à droite du bouton Knit et choisissez `Output Options`.

Texte

Pour écrire

```
- en gras: **gras**,
- en italique: _italics_.
```

Pour créer une liste numérotée, il suffit simplement d'utiliser des chiffres

```
1. gras
2. italique
```

Pour créer des en-têtes de section de différentes tailles, il suffit de mettre un certain nombre de symboles #:

```
# Titre
## Section principale
### Sous-section
#### Sous-sous-section
```

Code

Les morceaux de codes ressemblent à

```
```{r}
mean(x2)
```
```

Il y a une variété d'options pour affecter la façon dont les morceaux de code sont traités:

- echo = FALSE pour éviter que le code soit affiché
- results ="hide" pour éviter d'imprimer des résultats
- eval = FALSE pour afficher le code mais ne pas l'évaluer
- warning = FALSE et message = FALSE pour masquer les avertissements ou messages produits.
- fig.height et fig.width pour contrôler la taille des figures produites (en pouces).

4. Tidyverse et diamants

D'autres packages seront utilisés dans cette séance:

```r
library(corrplot)  # pour avoir un 'joli' grahe des corrélations
library(cowplot)   # pour mettre plusieurs graphes sur une même figure
library(questionr) # pour des tableaux
library(scales) # pour des plots
```

4.1. L'opérateur pipe %>%

Le but du pipe est de vous aider à écrire le code d'une manière plus facile à lire et à comprendre. Le principe:

```
- x %>% f() est équivalent à f(x)
- x %>% f(y) est équivalent à f(x,y)
- x %>% f(y,.) est équivalent à f(y,x)
```

Par exemple,

```r
x <- 3
# En R classique,
log(x)
# Avec le pipe
x %>% log()
```

4.2. Jeu de données sur des diamants

On considère le jeu de données `diamonds` qui contient le prix ainsi que 9 caractéristiques de 54000 diamants. Les 10 variables sont:

- `carat` : poids
- `cut` : qualité de la coupe (Fair, Good, Very Good, Premium, Ideal)
- `color` : couleur, de D (bonne) à J (mauvaise)
- `clarity` : clarté de I1 (moinds brillant), SI2, SI1, VS2, VS1, VVS2, VVS1 à IF (plus brillant)
- `depth` : pourcentage de profondeur total=z/mean(x, y)
- `table` : largeur du sommet du diamant par rapport au point le plus large
- `price` : prix en dollars
- `x` : longueur en mm

- `y` : largeur en mm
- `z` : profondeur en mm

Ce jeu de données existe dans le package `ggplot2`. Pour l'importer dans R, il suffit de faire:

```
data(diamonds)
```

Le jeu de données `diamonds`est au format `tibble`, un format de `tidyverse`. Il faut vérifier que l'importation s'est bien déroulée en demandant d'afficher le jeu de données

```
# Affichage
diamonds
```

Comme c'est une table `tibble`, seule les premières lignes s'affichent même si on demande d'afficher le jeu de données entier. Si on souhaite le visualiser, on peut faire

```
view(diamonds)
```

Si on peut afficher les 6 premières lignes

```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

ou les 10 dernières par exemple.

```
tail(diamonds, n=10)
```

```
## # A tibble: 10 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.71 Premium   E     SI1      60.5    55  2756  5.79  5.74  3.49
##  2  0.71 Premium   F     SI1      59.8    62  2756  5.74  5.73  3.43
##  3  0.7  Very Good E     VS2      60.5    59  2757  5.71  5.76  3.47
##  4  0.7  Very Good E     VS2      61.2    59  2757  5.69  5.72  3.49
##  5  0.72 Premium   D     SI1      62.7    59  2757  5.69  5.73  3.58
##  6  0.72 Ideal     D     SI1      60.8    57  2757  5.75  5.76  3.5
##  7  0.72 Good      D     SI1      63.1    55  2757  5.69  5.75  3.61
##  8  0.7  Very Good D     SI1      62.8    60  2757  5.66  5.68  3.56
##  9  0.86 Premium   H     SI2      61      58  2757  6.15  6.12  3.74
## 10  0.75 Ideal     D     SI2      62.2    55  2757  5.83  5.87  3.64
```

4.3. Structure du tableau de données

Nombre de lignes et de colonnes

Le nombre de lignes est donnée par l'instruction suivante:

```
nrow(diamonds)
```

```
## [1] 53940
```

Le nombre de colonnes est donnée par l'instruction suivante:

```r
ncol(diamonds)
```

```
## [1] 10
```

La fonction `dim` renvoit ses dimensions, donc les deux nombres précédents :

```r
dim(diamonds)
```

```
## [1] 53940    10
```

La fonction `names` retourne les noms des colonnes du tableau, c'est-à-dire la liste des variables:

```r
names(diamonds)
```

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

Nature des variables

Si on souhaite analyser les variables, les méthodes et fonctions utilisées seront différentes selon qu'il s'agit d'une variable quantitative (variable numérique pouvant prendre un grand nombre de valeurs comme `carat`, `price`, etc ...) ou d'une variable qualitative (variable pouvant prendre un nombre limité de valeurs appelées modalités comme `color` et `clarity`). Il faut donc bien vérifier le type des différentes variables.

```r
class(diamonds$color)
```

```
## [1] "ordered" "factor"
```

```r
class(diamonds$clarity)
```

```
## [1] "ordered" "factor"
```

```r
class(diamonds$price)
```

```
## [1] "integer"
```

La fonction `str` renvoit un descriptif plus détaillé de toutes les variables. Elle liste les différentes variables, indique leur type et liste les premières valeurs:

```r
str(diamonds)
```

```
## tibble [53,940 x 10] (S3: tbl_df/tbl/data.frame)
##  $ carat  : num [1:53940] 0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num [1:53940] 61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num [1:53940] 55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int [1:53940] 326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num [1:53940] 3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num [1:53940] 3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num [1:53940] 2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

Remarque: une variable qualitative peut être codée avec des chiffres 1, 2, 3, 4 mais n'a aucun sens numérique. Elle sera alors considérée par R comme 'numeric'. Il faut modifier son type, i.e. la transformer en facteur:

```r
table$Var <- as.factor(table$Var)
head(table)
```

Dans la table `table`, si vous l'affichez, rien n'aura changé ... mais elle sera bien maintenant considérée comme facteur

```
class(table$Var)
```

On serait tenté d'utiliser la fonction `as.numeric` pour passer du type 'factor' au type 'numeric'. En général, on ne retrouve pas les 'bonnes' valeurs. Si la variable est considérée par R comme 'factor' alors qu'elle aurait du être 'numeric', c'est qu'il y a eu un problème lors de l'importation. Typiquement, des valeurs manquantes codées avec des NA. Pour remédier à ce problème, on peut lors de l'importation le spécifier avec l'option suivante

```
donnees = read.table(file="table.csv", sep=";", na.strings = "NA")
```

Résumé statistique de toutes les variables du jeu de données

La fonction `summary` permet d'obtenir d'un coup plusieurs indicateurs classiques: si la variable est de type 'numeric', elle donne les quartiles, le min, le max et la moyenne; si la variable est de type 'factor', elle donne l'effectif de chaque modalité.

```
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
##
```

4.4. Manipulation avec dplyr

Grammaires

Les plus courantes sont

```
- select: sélectionne les variables en fonction de leur nom.
- slice: sélectionne des lignes.
- filter: filtre les observations en fonction de leurs valeurs.
```

```
- arrange: modifie l'ordre des lignes.
- mutate: ajoute de nouvelles variables qui sont des fonctions de variables existantes
- summarise: réduit plusieurs valeurs à un seul résumé.
- sample_n: sélectionne aléatoirement des lignes.
```

Les diamants

1. SELECT

```r
# sélection de la variable price
diamonds %>% select(price)
```

```
## # A tibble: 53,940 x 1
##    price
##    <int>
##  1   326
##  2   326
##  3   327
##  4   334
##  5   335
##  6   336
##  7   336
##  8   337
##  9   337
## 10   338
## # ... with 53,930 more rows
```

```r
## En R classique
## diamonds$price

# sélection des variable price et cut
diamonds %>% select(price,cut)
```

```
## # A tibble: 53,940 x 2
##    price cut
##    <int> <ord>
##  1   326 Ideal
##  2   326 Premium
##  3   327 Good
##  4   334 Premium
##  5   335 Good
##  6   336 Very Good
##  7   336 Very Good
##  8   337 Very Good
##  9   337 Fair
## 10   338 Very Good
## # ... with 53,930 more rows
```

```r
# sélection des variables quantitatives
diamonds %>% select_if(is.numeric)
```

```
## # A tibble: 53,940 x 7
##    carat depth table price     x     y     z
##    <dbl> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23   61.5    55   326  3.95  3.98  2.43
```

```
##  2 0.21   59.8    61   326  3.89  3.84  2.31
##  3 0.23   56.9    65   327  4.05  4.07  2.31
##  4 0.290  62.4    58   334  4.2   4.23  2.63
##  5 0.31   63.3    58   335  4.34  4.35  2.75
##  6 0.24   62.8    57   336  3.94  3.96  2.48
##  7 0.24   62.3    57   336  3.95  3.98  2.47
##  8 0.26   61.9    55   337  4.07  4.11  2.53
##  9 0.22   65.1    61   337  3.87  3.78  2.49
## 10 0.23   59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

```r
# On enlève la variable table
diamonds %>% select(-table)
```

```
## # A tibble: 53,940 x 9
##    carat cut       color clarity depth price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.23  Ideal     E     SI2      61.5   326  3.95  3.98  2.43
##  2 0.21  Premium   E     SI1      59.8   326  3.89  3.84  2.31
##  3 0.23  Good      E     VS1      56.9   327  4.05  4.07  2.31
##  4 0.290 Premium   I     VS2      62.4   334  4.2   4.23  2.63
##  5 0.31  Good      J     SI2      63.3   335  4.34  4.35  2.75
##  6 0.24  Very Good J     VVS2     62.8   336  3.94  3.96  2.48
##  7 0.24  Very Good I     VVS1     62.3   336  3.95  3.98  2.47
##  8 0.26  Very Good H     SI1      61.9   337  4.07  4.11  2.53
##  9 0.22  Fair      E     VS2      65.1   337  3.87  3.78  2.49
## 10 0.23  Very Good H     VS1      59.4   338  4     4.05  2.39
## # ... with 53,930 more rows
```

2. SLICE

```r
# sélection de la ligne 4 à 10
diamonds %>% slice(4:10)
```

```
## # A tibble: 7 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 2 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 3 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
## 4 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
## 5 0.26  Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
## 6 0.22  Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 7 0.23  Very Good H     VS1      59.4    61   338  4     4.05  2.39
```

```r
## En R classique
## diamonds[4:10,]
```

3. FILTER

```r
# Diamants dont le prix est supérieur à U$18000
diamonds %>% filter(price>18000)
## En R classique
## diamonds[diamonds$price>18000,]

# Diamants de couleur D dont le prix est supérieur à U$18000
```

```r
diamonds %>% filter(price>18000,color=="D")

# Nombre de diamants de couleur D dont le prix est supérieur à U$18000
diamonds %>% filter(price>18000,color=="D") %>% count()
## En R classique
## dim(diamonds[((diamonds$price>18000) & (diamonds$color=="D")),])[1]
```

4. ARRANGE

```r
# On ordonne le jeu de données selon les prix
diamonds %>% arrange(price)
```

5. MUTATE

```r
# Création dans la table diamonds de la variable priceR=price/carat
diamonds <- diamonds %>% mutate(priceR=price/carat)
## En R classique
## diamonds$priceR=diamonds$price/diamonds$carat
```

6. SUMMARISE

```r
# Moyenne de toutesles  variables quantitatives
diamonds %>% summarise_if(is.numeric,mean)
```

```
## # A tibble: 1 x 8
##   carat depth table price     x     y     z priceR
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 0.798  61.7  57.5 3933.  5.73  5.73  3.54  4008.
```

```r
# Ecart-type de toutes les variables quantitatives
diamonds %>% summarise_if(is.numeric,sd)
```

```
## # A tibble: 1 x 8
##   carat depth table price     x     y     z priceR
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 0.474  1.43  2.23 3989.  1.12  1.14 0.706  2013.
```

```r
# Moyenne et écart-type de la variable price
diamonds %>% select(price) %>% summarise(moyp=mean(price),sp=sd(price))
```

```
## # A tibble: 1 x 2
##    moyp    sp
##   <dbl> <dbl>
## 1 3933. 3989.
```

```r
# Moyenne et écart-type des prix par modalité de la variable cut
diamonds %>% group_by(cut) %>% summarise(moyp_cut=mean(price),sp_cut=sd(price))
```

```
## # A tibble: 5 x 3
##   cut       moyp_cut sp_cut
##   <ord>        <dbl>  <dbl>
## 1 Fair         4359.  3560.
## 2 Good         3929.  3682.
## 3 Very Good    3982.  3936.
## 4 Premium      4584.  4349.
## 5 Ideal        3458.  3808.
```

7. SAMPLE_N

```r
# On crée une nouvelle table diamonds_A qui contient 10 lignes prises au hasard dans diamonds
diamonds_A <- diamonds %>% sample_n(10)
diamonds_A
```

```
## # A tibble: 10 x 11
##    carat cut       color clarity depth table price     x     y     z priceR
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>  <dbl>
##  1  2.05 Premium   J     VS1      60.1    58 15067  8.25  8.19  4.94  7350.
##  2  1.22 Premium   G     SI1      61.2    60  6449  6.82  6.75  4.15  5286.
##  3  1.09 Ideal     F     SI1      62      56  5665  6.56  6.61  4.08  5197.
##  4  0.26 Very Good G     VVS1     61.8    57   545  4.06  4.1   2.52  2096.
##  5  0.5  Premium   D     SI1      62.2    59  1654  5.13  5.1   3.18  3308
##  6  0.36 Ideal     H     IF       61.6    56   912  4.56  4.59  2.82  2533.
##  7  0.31 Premium   G     SI2      61.3    58   523  4.4   4.37  2.69  1687.
##  8  1    Very Good H     SI1      63.4    56  4469  6.37  6.34  4.03  4469
##  9  0.41 Ideal     G     VS2      60.9    57  1061  4.84  4.79  2.93  2588.
## 10  0.31 Premium   F     VVS1     62.4    60   959  4.31  4.28  2.68  3094.
```

8. EFFECTIFS et EFFECTIFS CROISES

```r
### pour la variables cut
table(diamonds$cut)
```

```
##
##      Fair      Good Very Good   Premium     Ideal
##      1610      4906     12082     13791     21551
```

```r
diamonds %>% group_by(cut) %>% count()
```

```
## # A tibble: 5 x 2
## # Groups:   cut [5]
##   cut           n
##   <ord>     <int>
## 1 Fair       1610
## 2 Good       4906
## 3 Very Good 12082
## 4 Premium   13791
## 5 Ideal     21551
```

```r
### pour les variables cut et color
table(diamonds$cut,diamonds$color)
```

```
##
##                D    E    F    G    H    I    J
##   Fair       163  224  312  314  303  175  119
##   Good       662  933  909  871  702  522  307
##   Very Good 1513 2400 2164 2299 1824 1204  678
##   Premium   1603 2337 2331 2924 2360 1428  808
##   Ideal     2834 3903 3826 4884 3115 2093  896
```

4.5. Visualisation avec ggplot2

Le package ggplot2 reproduit la grammaire des graphiques (voir Grammar of Graphics, Leland Wilkinson), avec le même formalisme. Le principe de cette grammaire est qu'un graphique est composé de couches:

Graphique = données + Esthétique + Géométrie

```
- data: jeu de données

- esthétique (ou aesthetics en anglais): permet d'indiquer les variables x et y. Il peut également être

- géométrie: correspond au type de graphique (histogramme, boxplot, line plot, etc ...)
```

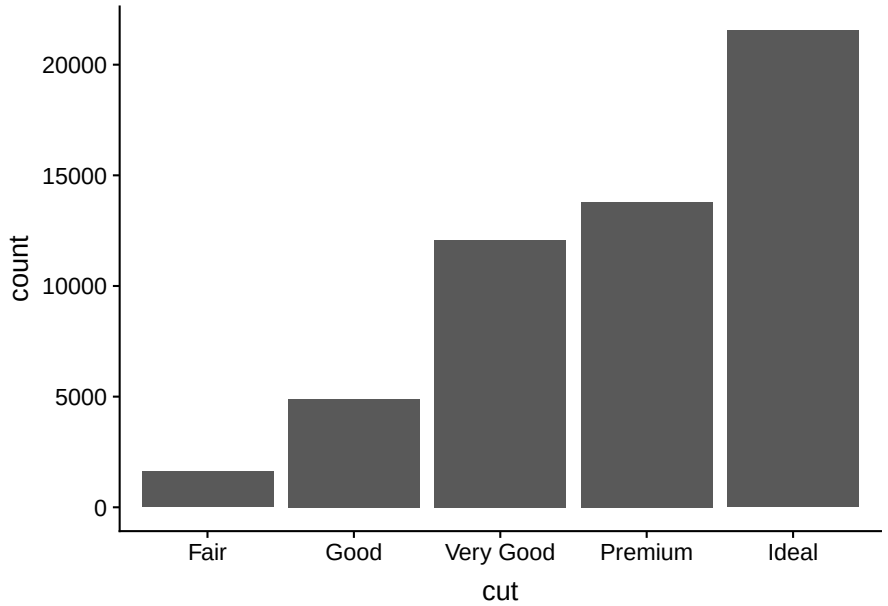
Syntaxe:

```
ggplot(data, aes(x = Variable en absisse, y = variable en ordonnée))+geom_point()
```

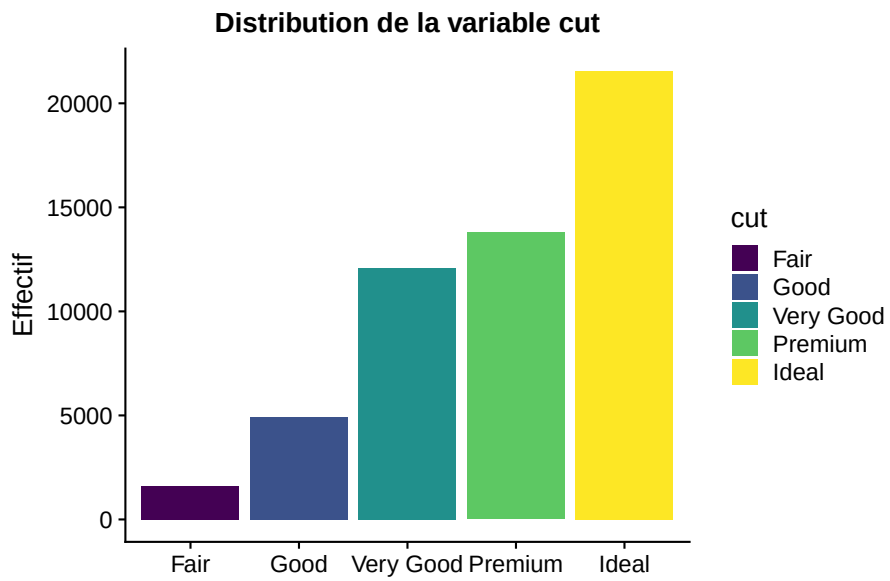
Une variable qualitative: `cut`

Diagramme en barres

```
ggplot(diamonds, aes(x = cut)) + geom_bar()
```

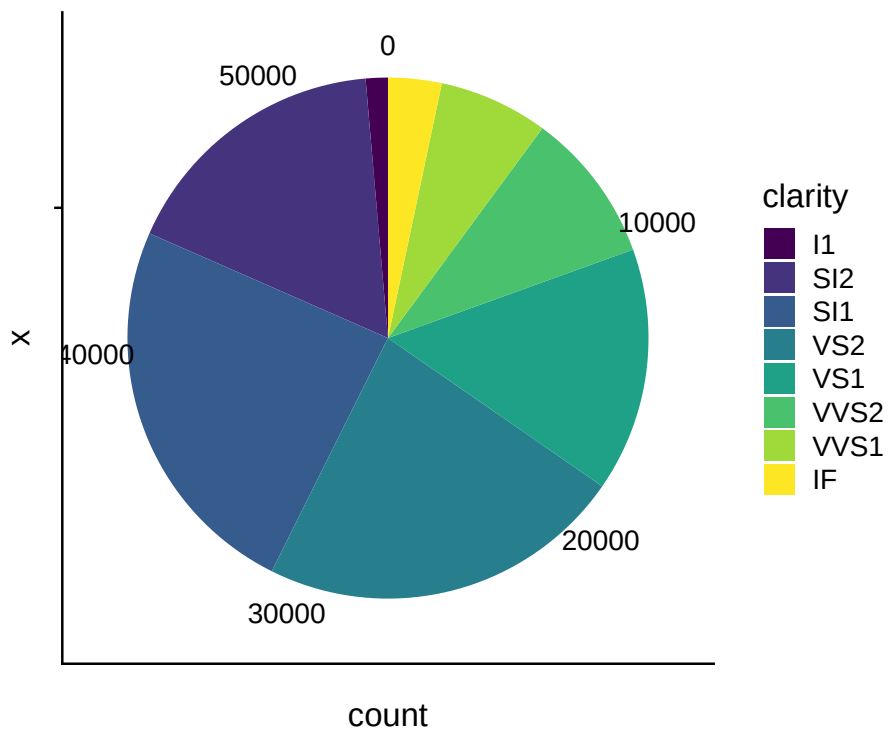



```
# avec des couleurs et des noms d'axes et titre
ggplot(diamonds, aes(x = cut, fill = cut)) +
  geom_bar() +
  ggtitle("Distribution de la variable cut") +
  xlab(" ") +
  ylab("Effectif")
```



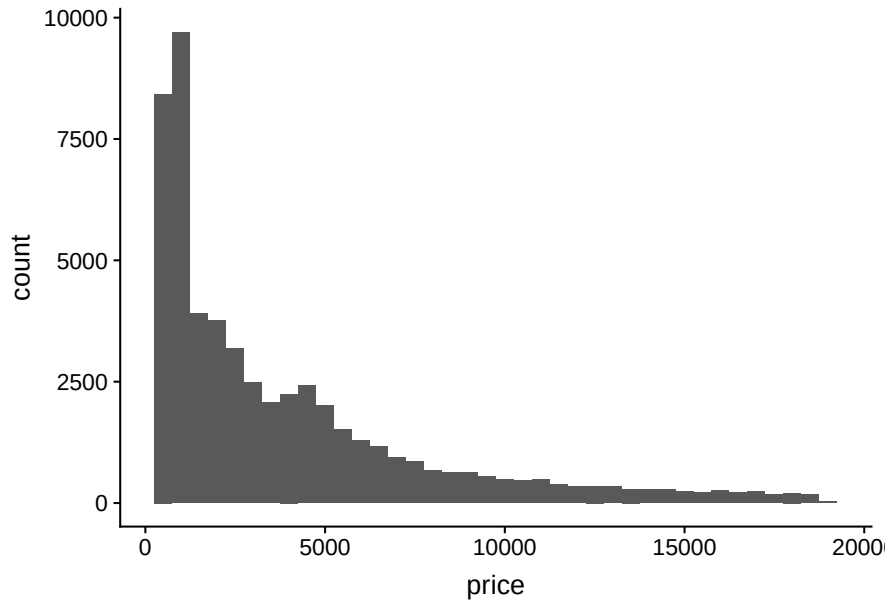
Camenbert

```
ggplot(diamonds, aes(x="", fill=clarity))+geom_bar(width = 2)+coord_polar("y")
```



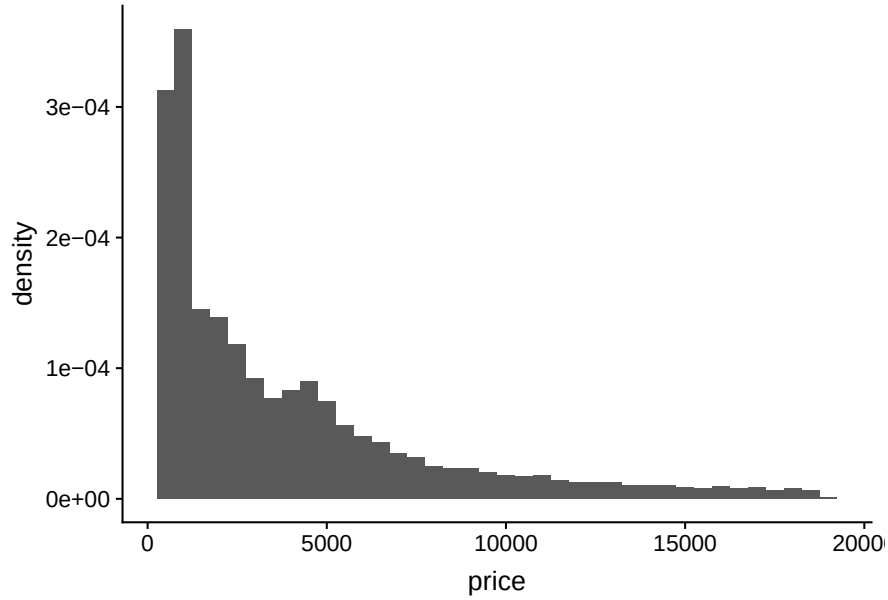
Une variable quantitative: `price`

```
# Histogramme de la variable price
ggplot(diamonds, aes(x=price)) + geom_histogram(binwidth=500)
```

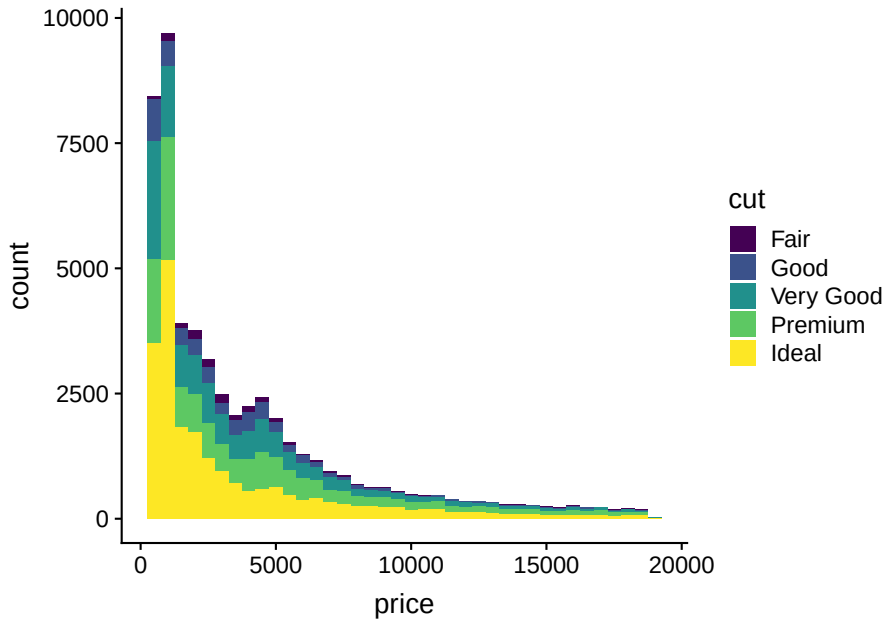



```r
# binwidth: largeur de la fenêtre

# si on veut la densité
ggplot(data=diamonds, aes(x=price)) + geom_histogram(aes(y = ..density..), binwidth=500)
```



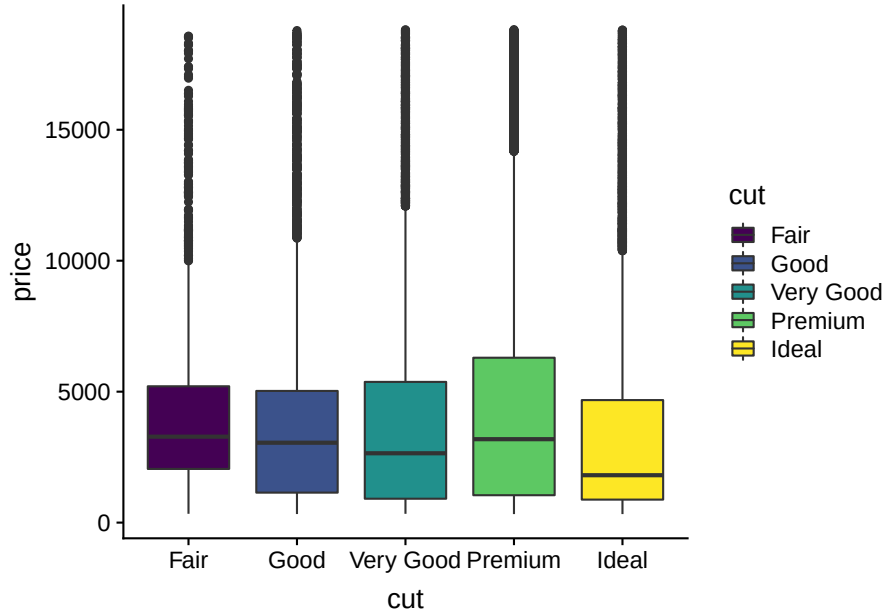

```r
# si on veut l'histogramme colorié selon la variable cut
ggplot(diamonds, aes(x=price,fill=cut)) + geom_histogram(binwidth=500)
```



Deux variables: une qualitative et une quantitative

1. Variable `price` en fonction de la variable `cut`

```
ggplot(diamonds, aes(x=cut, y= price, fill=cut)) + geom_boxplot()
```




```
diamonds %>% group_by(cut) %>% summarise(mean(price))
```

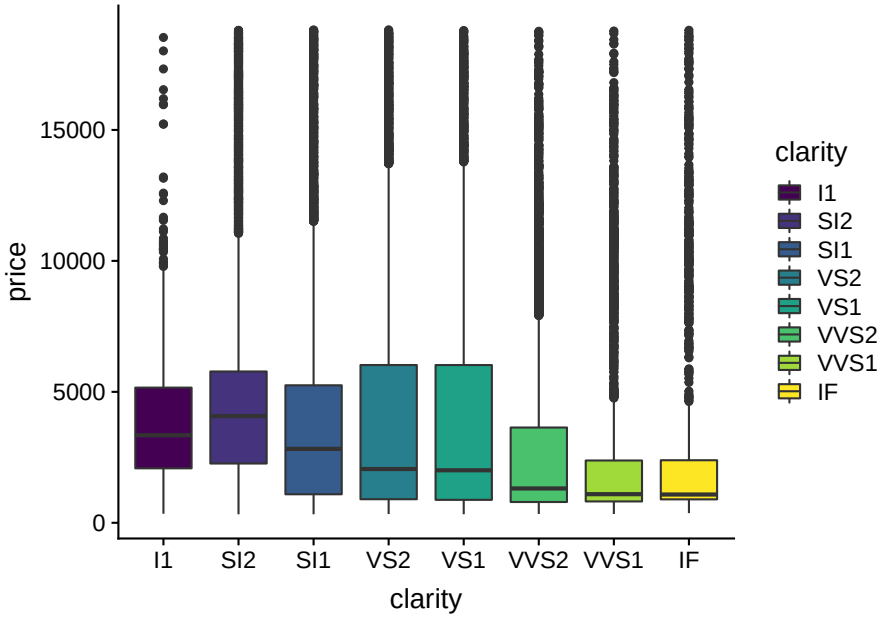
```
## # A tibble: 5 x 2
##   cut       `mean(price)`
##   <ord>             <dbl>
## 1 Fair              4359.
## 2 Good              3929.
## 3 Very Good         3982.
```

```
## 4 Premium           4584.
## 5 Ideal             3458.
```

Il semble que la coupe de tous les types comporte des prix de tous les types. Ce qui est surprenant est que les diamants idéaux sont moins chers que les diamants moyens.

2. Variable `price` en fonction de la variable `clarity`

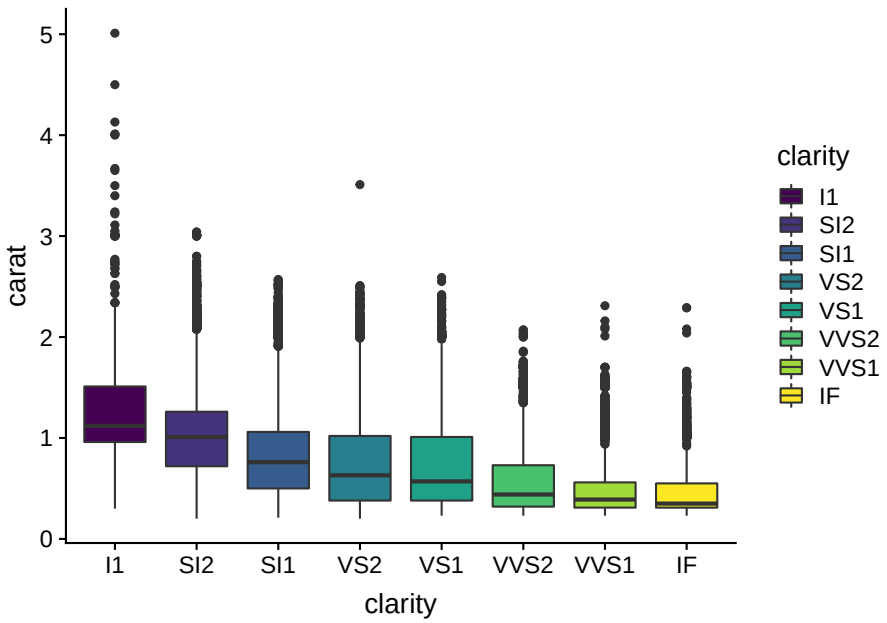
```
ggplot(diamonds, aes(x=clarity, y= price, fill=clarity)) + geom_boxplot()
```



Il semble y avoir des différences de prix selon la clarté. Et là encore, les personnes semblent plus apprécier les mauvais diamants (les moins brillants). Pourquoi ce phénomène?

3. Variable `carat` en fonction de la variable `clarity`

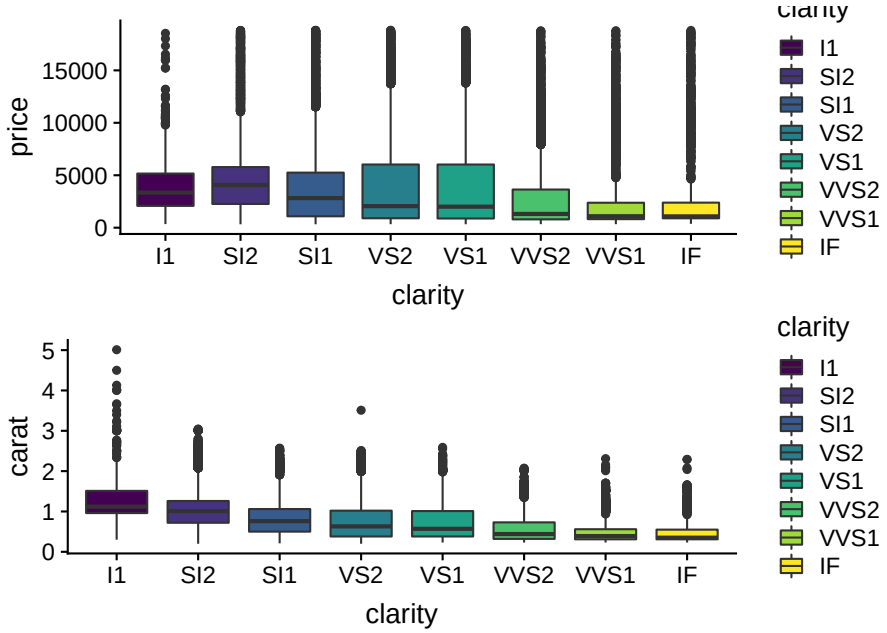
```
ggplot(diamonds, aes(x=clarity, y= carat, fill=clarity)) + geom_boxplot()
```



On observe la même tendance que sur le prix: moins le diamant est brillant, plus le poids est important. Pour rendre les prix comparables, il serait donc plus judicieux de comparer les prix rapportés au poids.

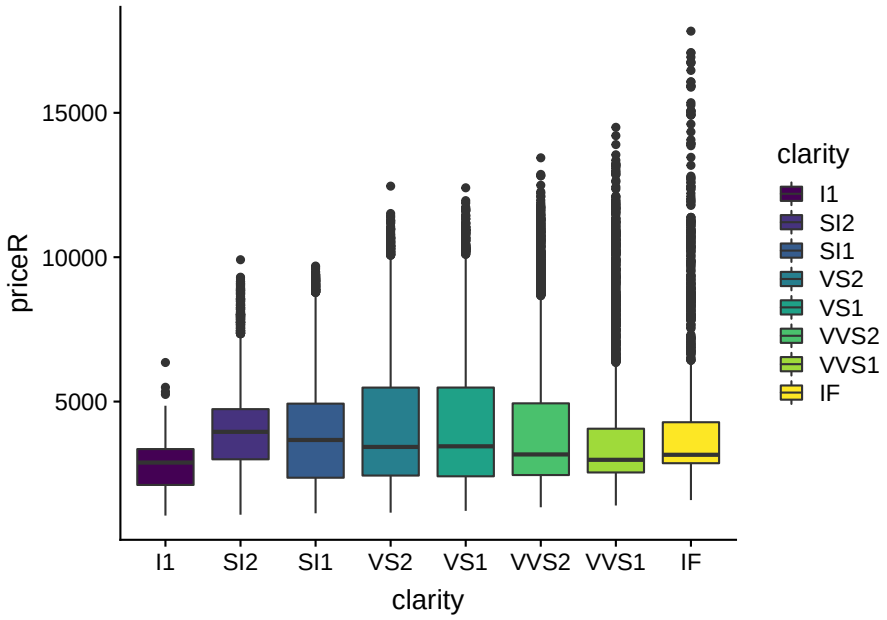
Si on souhaite mettre les deux graphes dans une même figure

```
fig1 <- ggplot(diamonds, aes(x=clarity, y= price, fill=clarity)) + geom_boxplot()
fig2 <- ggplot(diamonds, aes(x=clarity, y= carat, fill=clarity)) + geom_boxplot()
plot_grid(fig1, fig2, ncol = 1, nrow = 2)
```



4. Variable `priceR` en fonction de la variable `clarity`

```
ggplot(diamonds, aes(x=clarity, y= priceR, fill=clarity)) + geom_boxplot()
```



On n'observe plus la même chose: les diamants semblent être de prix proches sauf pour les moins brillants. Il y a cependant beaucoup de points aberrants pour les diamants brillants, beaucoup sont beaucoup plus chers.

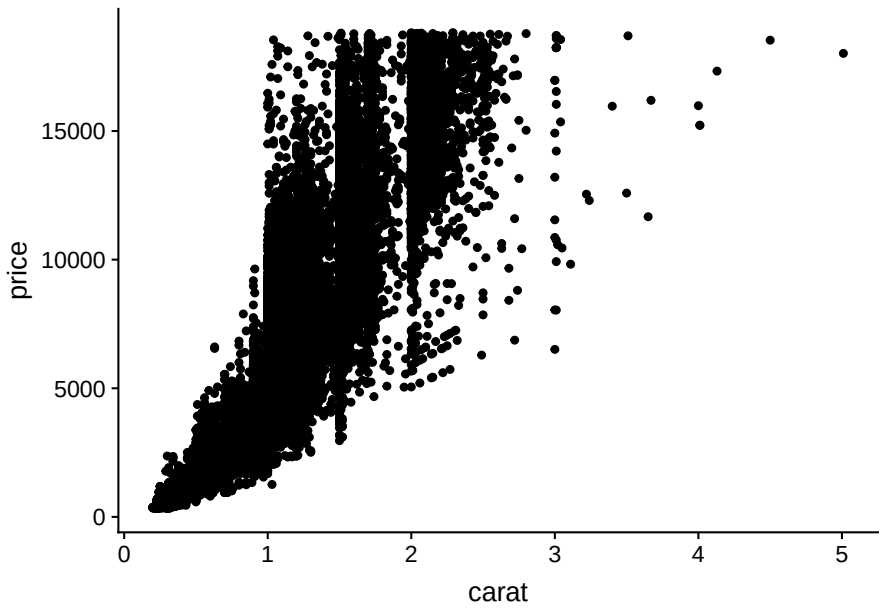
```
diamonds %>% group_by(clarity) %>% summarise(mean(priceR))
```

```
## # A tibble: 8 x 2
##   clarity `mean(priceR)`
##   <ord>            <dbl>
## 1 I1               2796.
## 2 SI2              4011.
## 3 SI1              3849.
## 4 VS2              4081.
## 5 VS1              4156.
## 6 VVS2             4204.
## 7 VVS1             3851.
## 8 IF               4260.
```

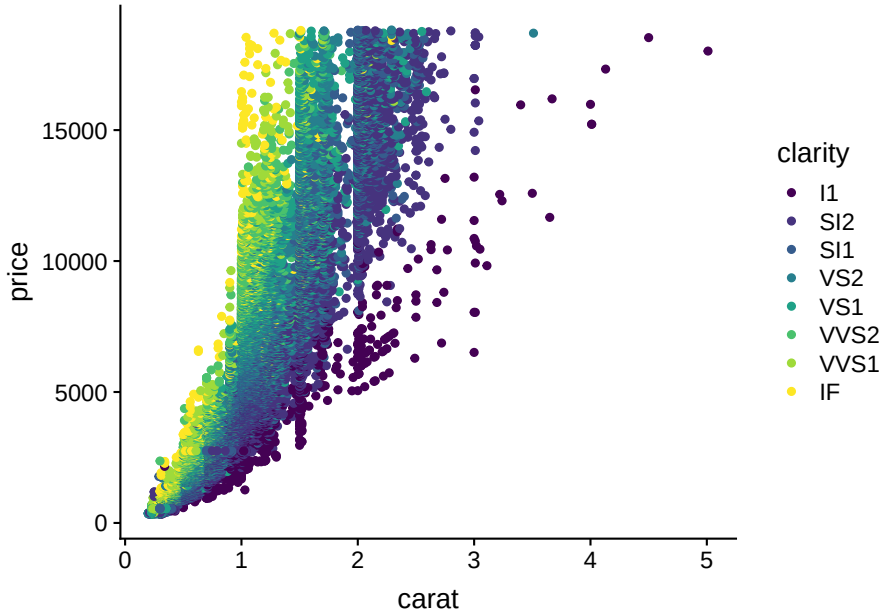
En moyenne, les diamants les plus brillants sont les plus chers.

Deux variables quantitatives: relation entre les variables `price` et `carat`

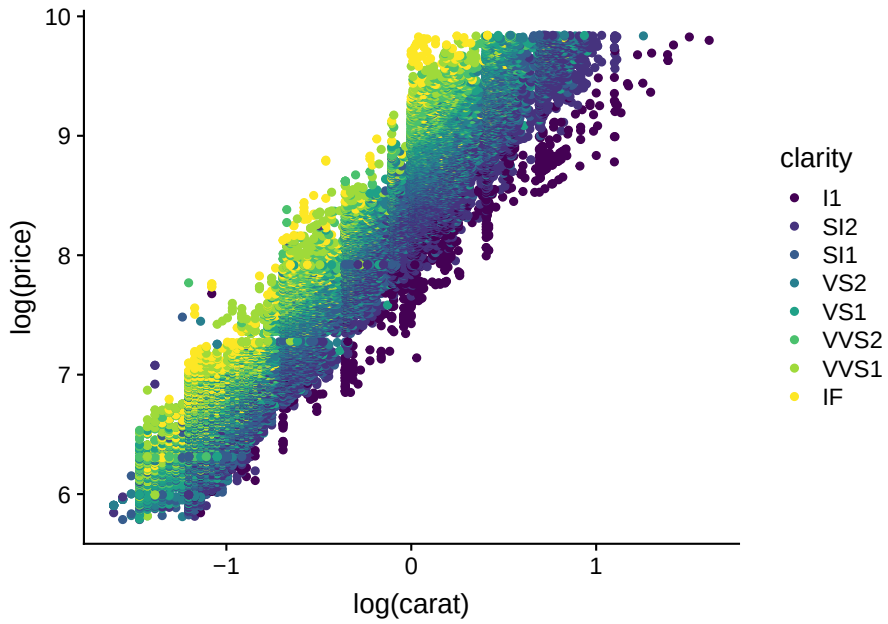
```
ggplot(diamonds, aes(x = carat, y = price)) + geom_point()
```




```r
# points colorié selon la clarity
ggplot(diamonds, aes(x = carat, y = price,color=clarity)) + geom_point()
```

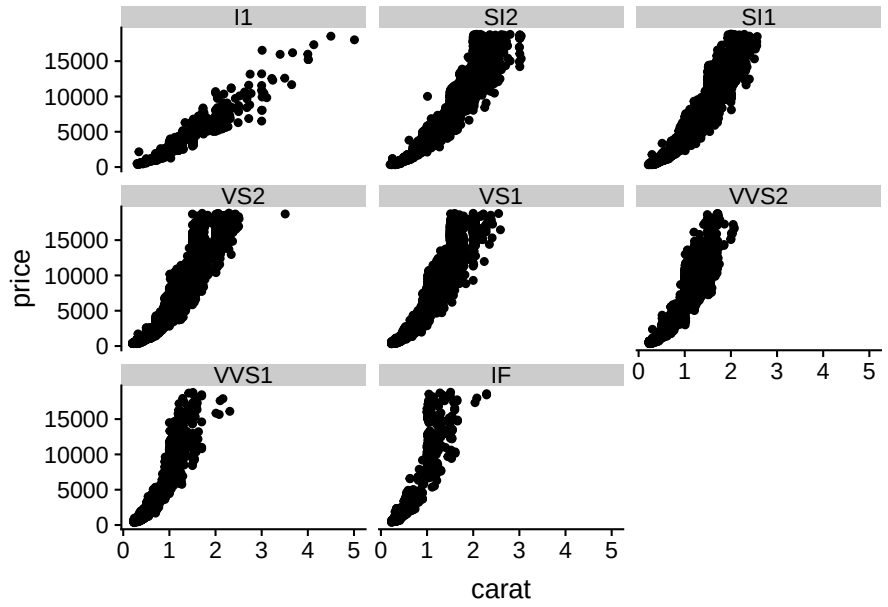



```r
# transformation des variables
ggplot(diamonds, aes(x = log(carat), y = log(price),color=clarity)) + geom_point()
```



On retrouve l'observation précédente, à savoir que pour un même poids de diamants, ceux sont les diamants les plus brillants qui semblent couter plus cher.

```
# pour chaque modalité de la variable clarity
ggplot(diamonds, aes(x = carat, y = price)) +
  geom_point() +
  facet_wrap(facets = ~ clarity)
```

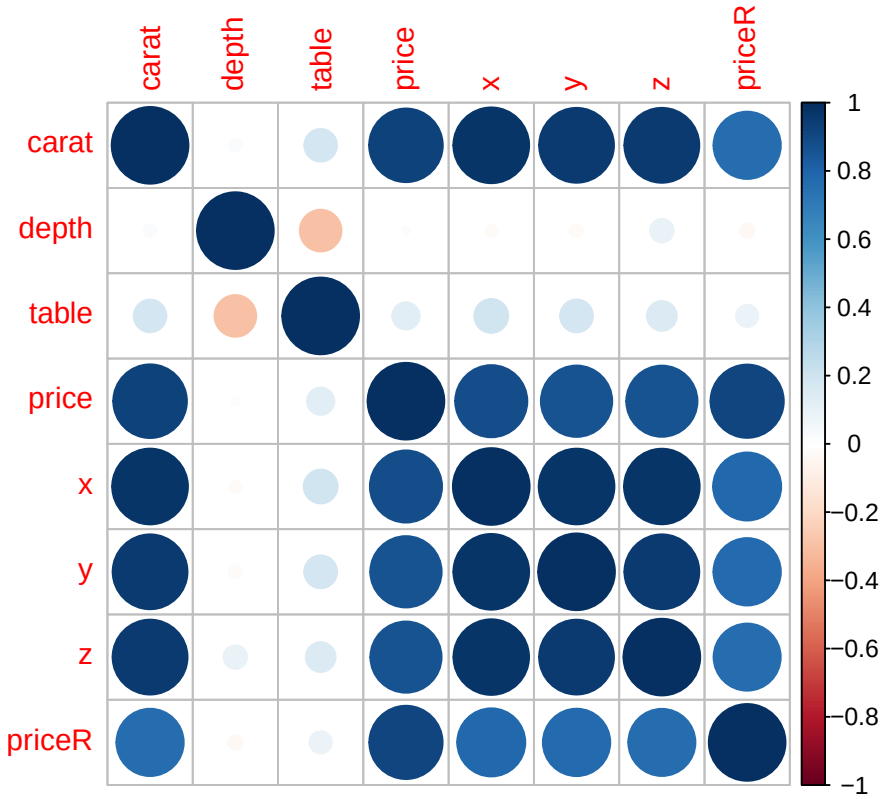


Corrélation entre toutes les variables quantitatives

```
correlation <- diamonds %>% select_if(is.numeric) %>% cor()
kable(correlation,digits=3)
```

	carat	depth	table	price	x	y	z	priceR
carat	1.000	0.028	0.182	0.922	0.975	0.952	0.953	0.770
depth	0.028	1.000	-0.296	-0.011	-0.025	-0.029	0.095	-0.036
table	0.182	-0.296	1.000	0.127	0.195	0.184	0.151	0.085
price	0.922	-0.011	0.127	1.000	0.884	0.865	0.861	0.912
x	0.975	-0.025	0.195	0.884	1.000	0.975	0.971	0.788
y	0.952	-0.029	0.184	0.865	0.975	1.000	0.952	0.772
z	0.953	0.095	0.151	0.861	0.971	0.952	1.000	0.766
priceR	0.770	-0.036	0.085	0.912	0.788	0.772	0.766	1.000

```
correlation %>% corrplot
```



Toutes les variables sont positivement corrélées, sauf `table` et `depth`: plus la profondeur est importante, moins le diamant est large au sommet.

Deux variables qualitatives: `clarity` et `cut`

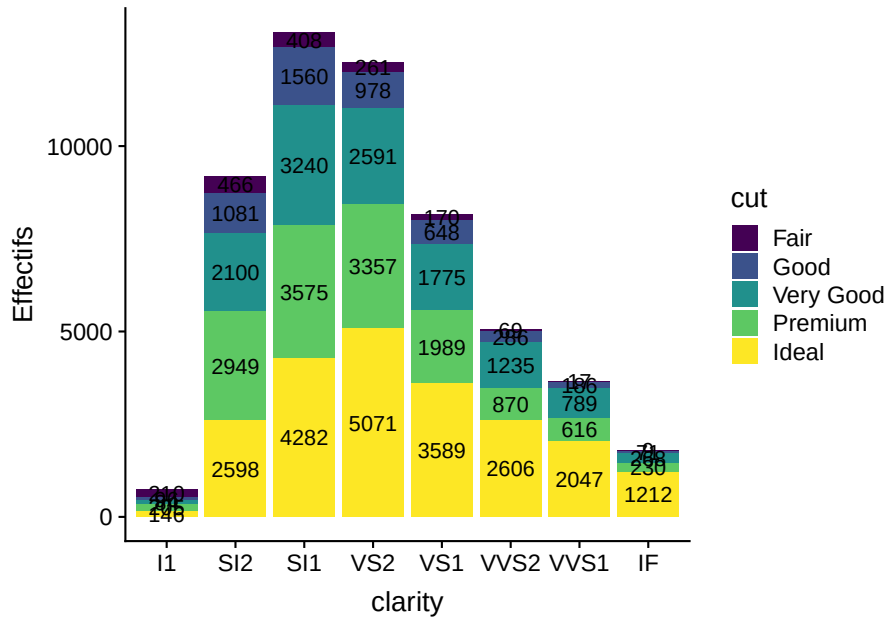
1. Effectifs croisés et représentation graphique

```
Eff  <- table(diamonds$clarity,diamonds$cut)
knitr::kable(Eff)
```

	Fair	Good	Very Good	Premium	Ideal
I1	210	96	84	205	146
SI2	466	1081	2100	2949	2598
SI1	408	1560	3240	3575	4282
VS2	261	978	2591	3357	5071

	Fair	Good	Very Good	Premium	Ideal
VS1	170	648	1775	1989	3589
VVS2	69	286	1235	870	2606
VVS1	17	186	789	616	2047
IF	9	71	268	230	1212

```
ggplot(diamonds) +
  aes(x = clarity, fill = cut) +
  geom_bar() +
  geom_text(aes(label = after_stat(count)), stat = "count", position = position_stack(.5)) +
  xlab("clarity") +
  ylab("Effectifs") +
  labs(fill = "cut")
```



2. Fréquences conjointes en pourcentage

```
F = Eff/sum(Eff)
knitr::kable(F,digits=3)
```

	Fair	Good	Very Good	Premium	Ideal
I1	0.004	0.002	0.002	0.004	0.003
SI2	0.009	0.020	0.039	0.055	0.048
SI1	0.008	0.029	0.060	0.066	0.079
VS2	0.005	0.018	0.048	0.062	0.094
VS1	0.003	0.012	0.033	0.037	0.067
VVS2	0.001	0.005	0.023	0.016	0.048
VVS1	0.000	0.003	0.015	0.011	0.038
IF	0.000	0.001	0.005	0.004	0.022

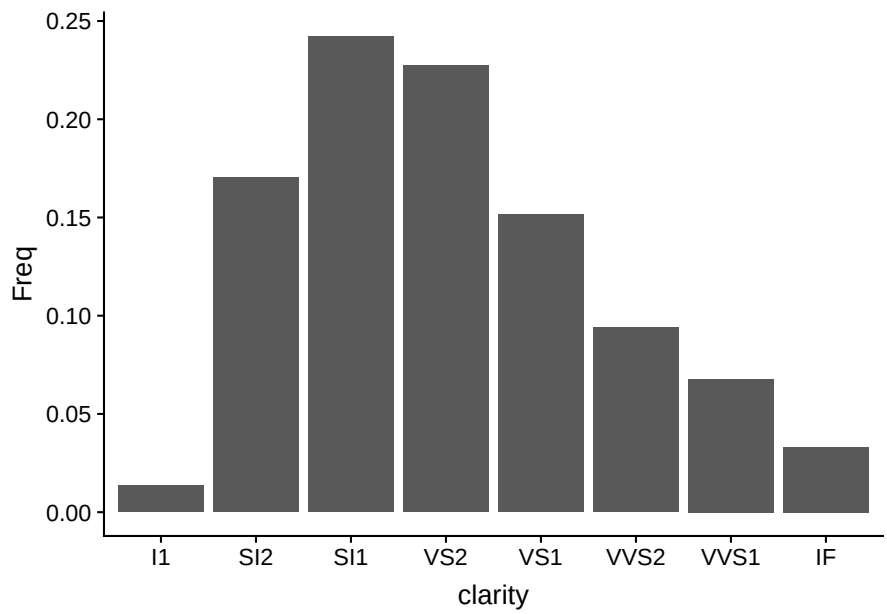
3. Fréquences marginales et représentation

En ligne

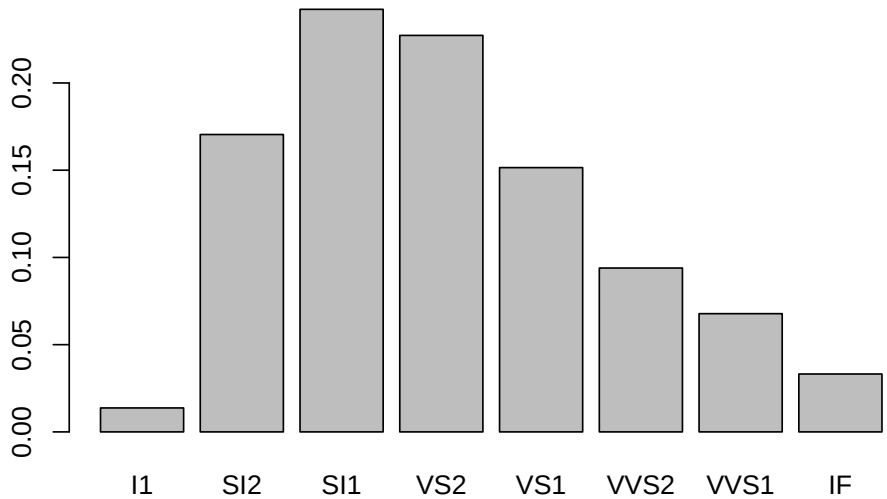
```r
FM_L = margin.table(F,1)
knitr::kable(FM_L,digits=2)
```

Var1	Freq
I1	0.01
SI2	0.17
SI1	0.24
VS2	0.23
VS1	0.15
VVS2	0.09
VVS1	0.07
IF	0.03

```r
# Si on dispose de la table d'origine
ggplot(diamonds) + aes(x = clarity,after_stat(count / sum(count))) +geom_bar() +
  xlab("clarity") +ylab("Freq")
```




```r
# Si on ne dispose que de FM_L
FM_L %>% barplot()
```

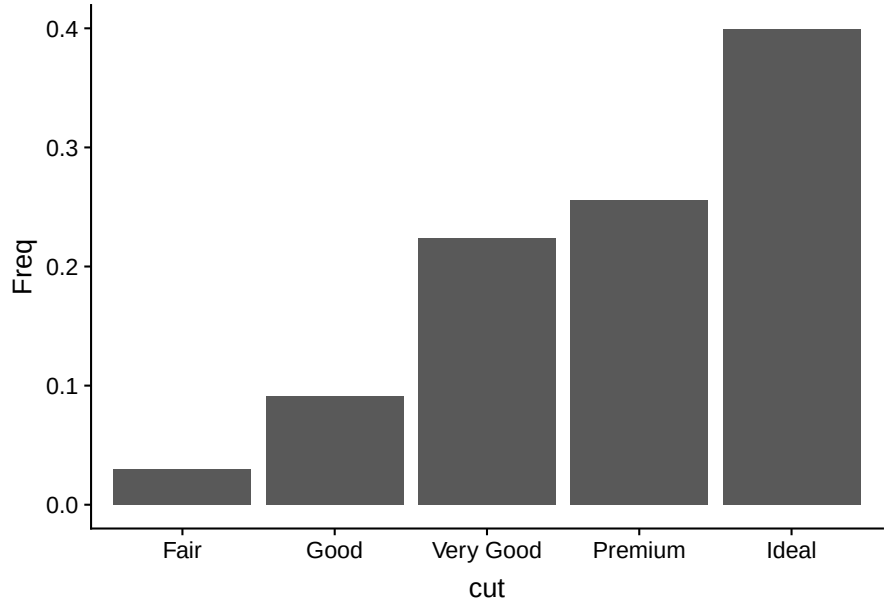


En colonne

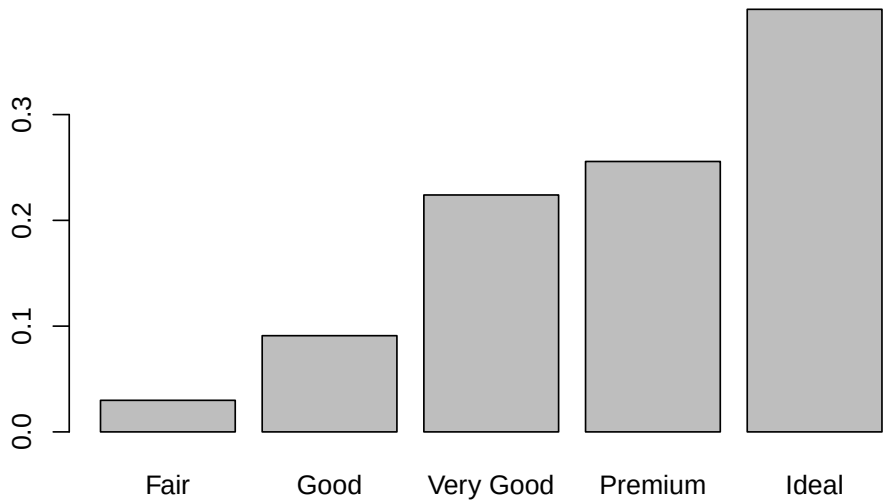
```
FM_C =margin.table(F,2)
knitr::kable(FM_C,digits=2)
```

Var1	Freq
Fair	0.03
Good	0.09
Very Good	0.22
Premium	0.26
Ideal	0.40

```
# Si on dispose de la table d'origine
ggplot(diamonds) + aes(x = cut,after_stat(count / sum(count))) +geom_bar() +
  xlab("cut") +ylab("Freq")
```




```
# Si on ne dispose que de FM_C
FM_C %>% barplot()
```



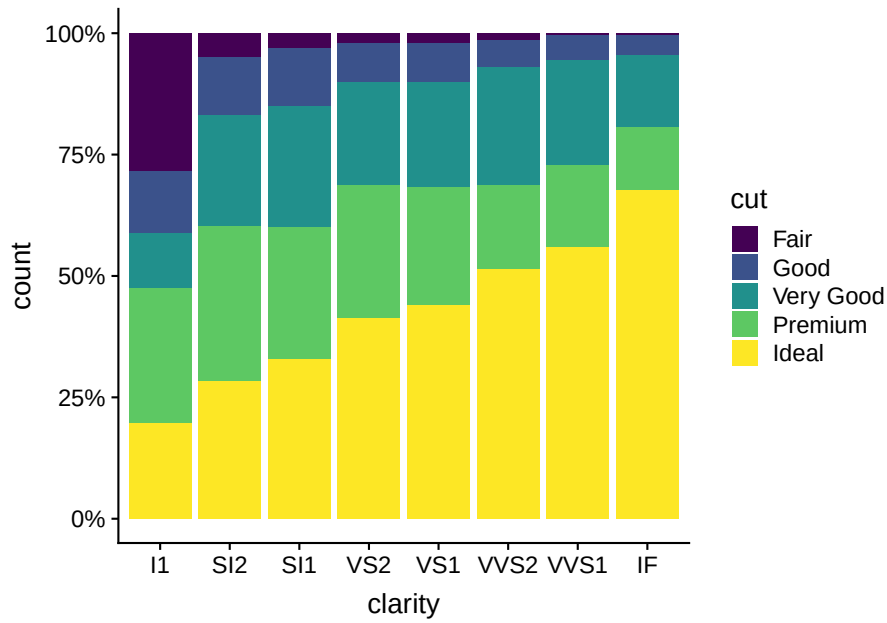
4. Fréquences conditionnelles et représentation

Profils lignes

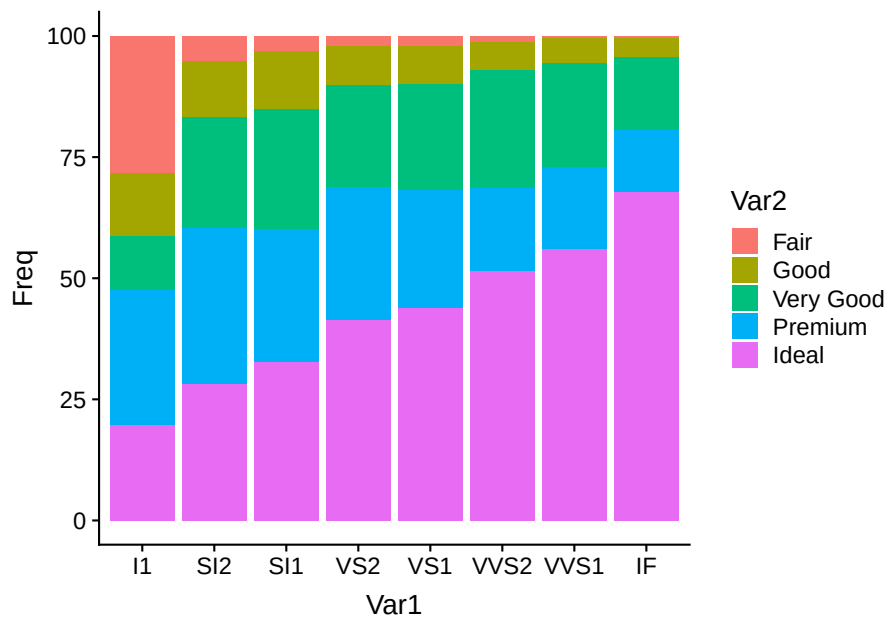
```
Profil.Ligne =rprop(Eff,digits = 2)
knitr::kable(Profil.Ligne)
```

	Fair	Good	Very Good	Premium	Ideal	Total
I1	28.3400810	12.955466	11.33603	27.66532	19.70310	100
SI2	5.0685229	11.757668	22.84098	32.07527	28.25756	100
SI1	3.1228473	11.940299	24.79908	27.36318	32.77459	100
VS2	2.1292217	7.978463	21.13722	27.38620	41.36890	100
VS1	2.0805287	7.930486	21.72317	24.34219	43.92363	100
VVS2	1.3620213	5.645480	24.37821	17.17331	51.44098	100
VVS1	0.4651163	5.088919	21.58687	16.85363	56.00547	100
IF	0.5027933	3.966480	14.97207	12.84916	67.70950	100
Ensemble	2.9847979	9.095291	22.39896	25.56730	39.95365	100

```
# Si on dispose de la table d'origine
ggplot(diamonds) +aes(x = clarity, fill = cut) +
  geom_bar(position = "fill") +
  scale_y_continuous(labels = percent)
```




```
# Si on dispose que de Profil.Ligne
rprop(Eff,total=FALSE) %>% as.data.frame() %>% ggplot(aes(x=Var1,fill=Var2))+geom_bar(aes(y=Freq), stat=
```



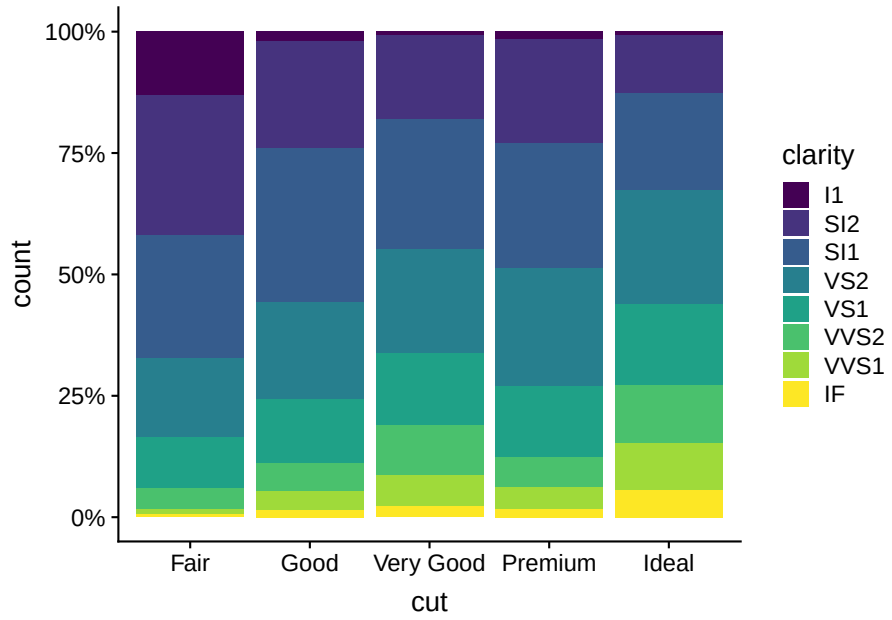
Profils colonnes

```
Profil.Colonne =cprop(Eff,digits = 2)
knitr::kable(Profil.Colonne)
```

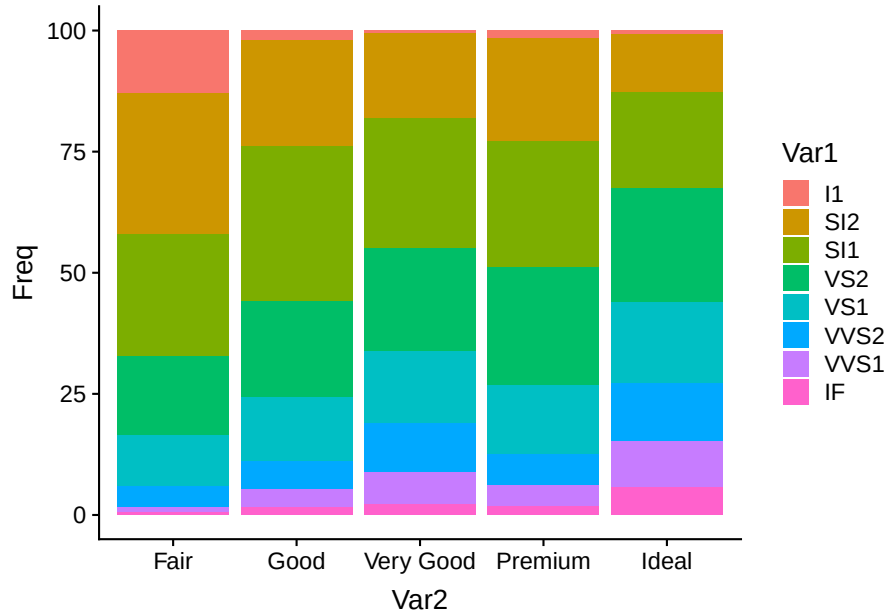
	Fair	Good	Very Good	Premium	Ideal	Ensemble
I1	13.0434783	1.956788	0.6952491	1.486477	0.6774628	1.373749
SI2	28.9440994	22.034244	17.3812283	21.383511	12.0551251	17.044865
SI1	25.3416149	31.797799	26.8167522	25.922703	19.8691476	24.221357
VS2	16.2111801	19.934774	21.4451250	24.341962	23.5302306	22.725250
VS1	10.5590062	13.208316	14.6912763	14.422449	16.6535196	15.148313
VVS2	4.2857143	5.829596	10.2218176	6.308462	12.0922463	9.391917

	Fair	Good	Very Good	Premium	Ideal	Ensemble
VVS1	1.0559006	3.791276	6.5303758	4.466681	9.4983991	6.776047
IF	0.5590062	1.447207	2.2181758	1.667754	5.6238690	3.318502
Total	100.0000000	100.000000	100.0000000	100.000000	100.0000000	100.000000

```r
# Si on dispose de la table d'origine
ggplot(diamonds) +aes(x = cut, fill = clarity) +
  geom_bar(position = "fill") +
  scale_y_continuous(labels = percent)
```

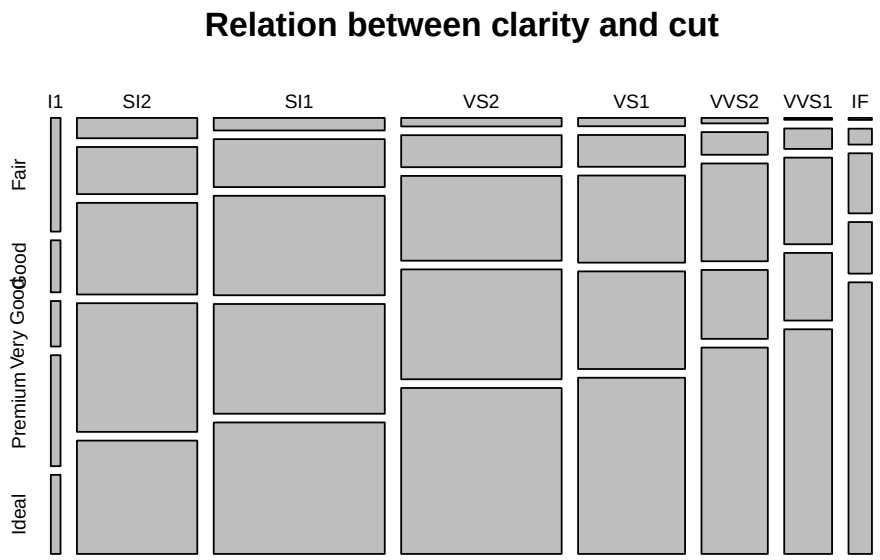



```r
# Si on dispose que de Profil.Ligne
cprop(Eff,total=FALSE) %>% as.data.frame() %>% ggplot(aes(x=Var2,fill=Var1))+geom_bar(aes(y=Freq), stat=
```

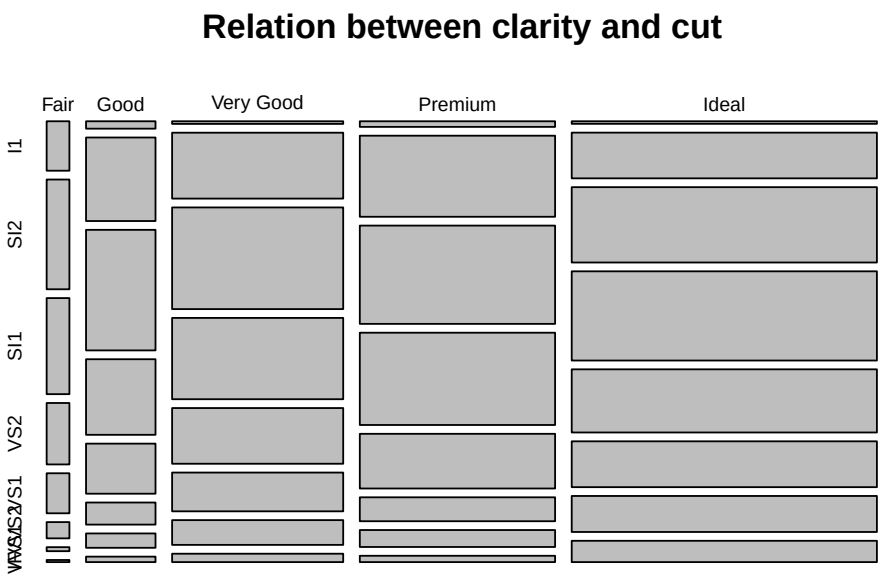


Graphe mosaic

```r
mosaicplot(Eff,main="Relation between clarity and cut")
```




```r
mosaicplot(t(Eff),main="Relation between clarity and cut")
```



Pour le premier graphe, chaque rectangle représente une case du tableau. La largeur des barres = fréquences marginales par ligne et la longueur au sein de chaque barre = fréquences conditionnelles par ligne.