Patient

Pascal

2024-03-13

Contents

1 Objectif **5**

2 Architecture de R **7**
2.1 Paquets 8

3 Le langage R **11**
3.1 Introduction 11
3.2 Un exemple de traitement de données 11
3.3 Vecteurs 12
3.4 Sélection des individus 19

4 Données de Parcoursup **25**

5 Résumés rapides descriptives **31**
5.1 Les premières tables 31
5.2 Tableaux croisés 33
5.3 Themes 33
5.4 Exportation 34
5.5 Personnalisation des statistiques 34
5.6 Modèles 37

6 Graphiques et ggplot **39**
6.1 Les graphiques de base 39
6.2 Liens 70

7 Manipulation avancée (mettre en forme vos données) **73**
7.1 Transformation de plusieurs variables 73
7.2 Opérateurs et case_when 77
7.3 Réutilisation de statistiques 80
7.4 Chargement 81

Chapter 1

Objectif

Ce document constitue la trame du cours sur R pour les doctorants “orientation” du CRTD du CNAM.

Il n’est pas exhaustif et se place plutôt en appui du cours de M. Kilani. Il est construit sur les bases et sur les difficultés du langage.

Chapter 2

Architecture de R

```
## New names:
## * `Filière de formation` -> `Filière de formation...10`
## * `Filière de formation` -> `Filière de formation...14`
```

R est à l'origine un logiciel de statistiques. Pour en faciliter l'usage il y avait une interface très rugueuse qui était livré avec sous Windows.

Pour l'utiliser, en fait, il fallait écrire le code dans un bloc-notes (le code est du texte brut) et le coller dans la console de R.

Vous pouvez toujours voir ce que ça donne en lançant R et non pas RStudio sur votre bureau. En prenant la mesure qu'il y a eu beaucoup de progrès de fait.

Maintenant il existe RStudio, racheté par Posit. En fait c'est éditeur de texte grandement amélioré.

Vous tapez le script dans la fenêtre en haut à gauche et vous l'exécutez en cliquant sur l'icone ou CTRL+ENTREE. Vous pouvez limiter ce que vous exécutez en sélectionnant du texte dans l'éditeur, seul le texte sélectionné sera soumis.

En clair, comme RStudio est un éditeur amélioré : - le code qui n'est pas soumis, R ne le reconnait pas. Si vous créez une variable en ligne 4 mais que vous ne le soumettez pas, quand vous aurez besoin de la variable en ligne 30, elle n'existera pas. - l'éditeur de texte peut contenir autre chose que du texte simple. Par exemple du RMarkdown comme c'est le cas pour ce document. C'est un langage qui produit de l'HTML c'est-à-dire des pages web. Le Markdown est très simple, le langage tient sur une feuille : ici. - ou du LaTeX avec le mode knitr - On peut des graphiques R dans la fenêtre RStudio (en bas à droite) - RStudio peut servir pour la gestion de paquets - …

RStudio fait donc beaucoup de choses.

A noter que Jamovi et JASP utilisent aussi R sauf que pour eux ce n'est presque

plus visible. A part dans les extensions et/ou les modules. Il y a des modules Jamovi pour écrire du code R ou pour faire des modèles structuraux directement en R dans Jamovi.

Vous allez avoir une démonstration enregistré pour l'utilisation de RStudio.

Les autres choses à rappeler, c'est que R est un langage cassee dépendant : - les noms de fonctions de R sont à écrire en **minuscules** - un objet n'est pas le même si une ou plusieurs lettres sont en majuscule au lieu d'être en minuscule.

```
aAa <- 2
AAa <- 4
```

donc

```
AAa
```

```
## [1] 4
```

et

```
aAa
```

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

Les commentaires dans R sont à noter avec des #. Tout ce qui suit le # est un commentaire.

Comme RStudio vous l'aurez remarqué vous aide en analysant et en colorant le code, n'utilisez pas un script R pour mettre des remarques (et non du code) en dehors de commentaires. Sinon c'est la catastrophe…

Utiliser toujours RStudio en mode Projet

2.1 Paquets

Les fonctionnalités de R sont finalement assez limitées. Il fait un certain nombre de statistiques et surtout fourni des outils mathématiques puissants mais ça grande force est de proposer des paquets qui ajoute des fonctionnalités.

Ces paquets sont développés par des gens (qui font ça sur leur temps personnel ou professionnel), des entreprises, des fondations, …

La liste des paquets officiels est là

Ils sont maintenant pléthoriques et je vous conseille de vous reportez à cette page.

Elles répertorient les paquets utiles par domaine d'application. De plus les paquets un peu douteux en qualité n'y figure pas. Donc c'est du solide.

Pour cela je vous recommande de l'utiliser pour se faire, suivez les instructions ci-dessous:

```
install.packages("ctv")
```

Puis quand vous voulez installer une vue (SocialSciences par exemple) :

```
library(ctv)
install.views("SocialSciences")
```

On vient de voir comment appeler un paquet :

```
library(tidyverse)
```

ou

```
require(tidyverse)
```

Cela revient presque au même. Presque. Pour commencer vous pouvez utiliser **library** en priorité.

Essayez d'installer un paquet :

```
install.packages("readxl")
```

Maintenant vous pouvez lire les fichiers Excel. Vous n'avez besoin d'installer le paquet qu'une fois mais vous devez le réclamer avec **library** la première fois que vous l'utilisez dans un script. La politique est de charger tous les paquets que vous utilisez **en tête du script**.

Maintenant on va faire des statistiques.

Chapter 3

Le langage R

3.1 Introduction

Le but est d'aborder des notions et de voir quelques exemples.

R ne fonctionne pas comme JASP, Jamovi, SAS ou SPSS. Par exemple SPSS, vous ouvrez une source de données et vous voyez vos données sur un tableur.

Quand vous passez une commande sur SPSS, il n'y a pas d'ambiguité, le traitement se fait sur le tableur actif.

Avec SAS, on ajoute une dose de complexité, car vous avez des bibliothèques et des tables.

Dans les deux cas, quand vous lancez une procédure statistique vous récupérez les résultats dans une fenêtre dédié car il y a une séparation des données et des résultats (dans la quasi-totalité des cas).

Avec R, c'est différent. R est un langage de programmation comme Python, Pascal, Rust, etc.

La force de R et ce qui le rend compliqué est qu'il n'y a pas de séparation aussi stricte entre données et résultats.

Vous avez des objets en mémoire dans R et ces objets peuvent servir aussi bien de sources de données, d'arguments pour sélectionner une partie des résultats ou bien être des résultats d'une opérations statistiques.

3.2 Un exemple de traitement de données

On va travailler sur une base de données qui sont les iris de Fisher. C'est plus simple car c'est un jeu de données qui est en mémoire dans R, on verra comment charger une source de données plus tard.

```
data(iris)
```

Vous pouvez cliquer sur **iris** qui est apparu dans la fenêtre en haut à droite de RStudio. Elle ouvre un tableur assez frustre mais qui permet de visualiser les données.

Mais c'est une très mauvaise habitude d'utiliser ce tableur pour visualiser les données.

Il vaut mieux taper :

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 .
```

str c'est pour **structure**. Enfin je crois. Elle vous décrit quelle est la nature de l'objet et de quoi il est composé. Ca peut être rudemment complexe.

Mais là non. Il nous dit que c'est une **data.frame**. Le type **data.frame** est ce qui se rapproche le plus d'un tableau de données comme dans SPSS ou Jamovi.

R nous dit que la data.frame a 150 observations et 5 variables. La structure est tabulaire comme dans Jamovi ou JASP: on a 150 relevés de plante (individus) et on a gardé 5 éléments caractérisant l'individu.

- Sepal.Length : **num** veut dire **numeric**, c'est une taille de Sépale.
- Sepal.Width : **num** veut dire **numeric**, c'est une taille de Sépale.
- ...
- Species : c'est l'espèce, qui peut prendre 3 valeurs. Dans un type appelé **Factor**

On voit que R sépare bien chacune des variables. Pour schématiser dans notre cas nous avons des individus en ligne et des observations en colonne. Comme Jamovi.

3.3 Vecteurs

3.3.1 Exemple de vecteurs

Là où ça devient différent c'est que la **data.frame** est en fait un aggrégat d'élements plus simples.

Vous pouvez extraire par exemple la longueur des sépales pour tous les individus:

```
iris[,"Sepal.Length"]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1
##  [28] 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6
##  [82] 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2
## [136] 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

Vous regardez par colonne, vous obtenez les valeurs pour les 150 individus.

Si vous regardez la structure de ce que vous avez obtenu :

```
str(iris[,"Sepal.Length"])
```

```
##  num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
```

Vous voyez que ce qui s'affichait tout à l'heure sur la **data.frame**.

Vous pouvez calculer la moyenne des longueurs:

```
mean(iris[,"Sepal.Length"])
```

```
## [1] 5.843333
```

Vous venez de faire une opération sur un vecteur. C'est un ensemble qui est typé ici des numériques mais ça peut être du texte, des entiers, etc. respectivement **character**, **integer**, etc.

Vous pouvez extraire ce vecteur :

```
longueur.sepale <- iris[,"Sepal.Length"]
```

Vous remarquez que R n'affiche pas le résultat de l'opération car on ne lui demande pas de résultat. On affecte la partie à droite de "<-" à la partie gauche.

Si on fait:

```
longueur.sepale
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5.1 5.4 5.1
##  [28] 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5.1 4.6 5.3
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6.1 6.4 6.6
##  [82] 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6.3 5.8 7.1
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6.1 6.4 7.2
## [136] 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

On retrouve bien nos longueurs.

On peut en calculer la moyenne :

```
mean(longueur.sepale)
```

```
## [1] 5.843333
```

Qu'est ce qui se passe ? En fait on a indexé notre **data.frame**.

On a dit à R, renvoie nous la variable "Sepal.Length" et nous avons décidé de le stocker dans une variable autre.

3.3.2 Création de vecteurs

Pour créer un vecteur, il faut utiliser la fonction **c** pour **concatenate**.

Exemple :

```
c("Sepal.Length","Petal.Length")
```

```
## [1] "Sepal.Length" "Petal.Length"
```

On peut l'affecter à une variable:

```
longueurs <- c("Sepal.Length","Petal.Length")
```

C'est un vecteur :

```
str(longueurs)
```

```
##  chr [1:2] "Sepal.Length" "Petal.Length"
```

Maintenant on peut faire:

```
iris[,longueurs]
```

```
##     Sepal.Length Petal.Length
## 1            5.1          1.4
## 2            4.9          1.4
## 3            4.7          1.3
## 4            4.6          1.5
## 5            5.0          1.4
## 6            5.4          1.7
## 7            4.6          1.4
## 8            5.0          1.5
## 9            4.4          1.4
## 10           4.9          1.5
## 11           5.4          1.5
## 12           4.8          1.6
## 13           4.8          1.4
## 14           4.3          1.1
## 15           5.8          1.2
## 16           5.7          1.5
## 17           5.4          1.3
```

```
## 18          5.1         1.4
## 19          5.7         1.7
## 20          5.1         1.5
## 21          5.4         1.7
## 22          5.1         1.5
## 23          4.6         1.0
## 24          5.1         1.7
## 25          4.8         1.9
## 26          5.0         1.6
## 27          5.0         1.6
## 28          5.2         1.5
## 29          5.2         1.4
## 30          4.7         1.6
## 31          4.8         1.6
## 32          5.4         1.5
## 33          5.2         1.5
## 34          5.5         1.4
## 35          4.9         1.5
## 36          5.0         1.2
## 37          5.5         1.3
## 38          4.9         1.4
## 39          4.4         1.3
## 40          5.1         1.5
## 41          5.0         1.3
## 42          4.5         1.3
## 43          4.4         1.3
## 44          5.0         1.6
## 45          5.1         1.9
## 46          4.8         1.4
## 47          5.1         1.6
## 48          4.6         1.4
## 49          5.3         1.5
## 50          5.0         1.4
## 51          7.0         4.7
## 52          6.4         4.5
## 53          6.9         4.9
## 54          5.5         4.0
## 55          6.5         4.6
## 56          5.7         4.5
## 57          6.3         4.7
## 58          4.9         3.3
## 59          6.6         4.6
## 60          5.2         3.9
## 61          5.0         3.5
## 62          5.9         4.2
## 63          6.0         4.0
```

```
## 64          6.1         4.7
## 65          5.6         3.6
## 66          6.7         4.4
## 67          5.6         4.5
## 68          5.8         4.1
## 69          6.2         4.5
## 70          5.6         3.9
## 71          5.9         4.8
## 72          6.1         4.0
## 73          6.3         4.9
## 74          6.1         4.7
## 75          6.4         4.3
## 76          6.6         4.4
## 77          6.8         4.8
## 78          6.7         5.0
## 79          6.0         4.5
## 80          5.7         3.5
## 81          5.5         3.8
## 82          5.5         3.7
## 83          5.8         3.9
## 84          6.0         5.1
## 85          5.4         4.5
## 86          6.0         4.5
## 87          6.7         4.7
## 88          6.3         4.4
## 89          5.6         4.1
## 90          5.5         4.0
## 91          5.5         4.4
## 92          6.1         4.6
## 93          5.8         4.0
## 94          5.0         3.3
## 95          5.6         4.2
## 96          5.7         4.2
## 97          5.7         4.2
## 98          6.2         4.3
## 99          5.1         3.0
## 100         5.7         4.1
## 101         6.3         6.0
## 102         5.8         5.1
## 103         7.1         5.9
## 104         6.3         5.6
## 105         6.5         5.8
## 106         7.6         6.6
## 107         4.9         4.5
## 108         7.3         6.3
## 109         6.7         5.8
```

```
## 110          7.2          6.1
## 111          6.5          5.1
## 112          6.4          5.3
## 113          6.8          5.5
## 114          5.7          5.0
## 115          5.8          5.1
## 116          6.4          5.3
## 117          6.5          5.5
## 118          7.7          6.7
## 119          7.7          6.9
## 120          6.0          5.0
## 121          6.9          5.7
## 122          5.6          4.9
## 123          7.7          6.7
## 124          6.3          4.9
## 125          6.7          5.7
## 126          7.2          6.0
## 127          6.2          4.8
## 128          6.1          4.9
## 129          6.4          5.6
## 130          7.2          5.8
## 131          7.4          6.1
## 132          7.9          6.4
## 133          6.4          5.6
## 134          6.3          5.1
## 135          6.1          5.6
## 136          7.7          6.1
## 137          6.3          5.6
## 138          6.4          5.5
## 139          6.0          4.8
## 140          6.9          5.4
## 141          6.7          5.6
## 142          6.9          5.1
## 143          5.8          5.1
## 144          6.8          5.9
## 145          6.7          5.7
## 146          6.7          5.2
## 147          6.3          5.0
## 148          6.5          5.2
## 149          6.2          5.4
## 150          5.9          5.1
```

On vient d'indexer iris avec un vecteur composé de deux noms qui sont les noms des variables.

R lit la partie droite de la virgule et comprends que nous voulons les deux variables. Quelle est la structure de ce que l'on récupère :

```
str(iris[,longueurs])
```

```
## 'data.frame':    150 obs. of  2 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
```

C'est une **data.frame** les informations sur nos 150 individus pour les longueurs.

on peut faire:

```
iris.longueurs <- iris[,longueurs]
str(iris.longueurs)
```

```
## 'data.frame':    150 obs. of  2 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
```

```
iris.longueurs[,"Sepal.Length"]
```

```
##   [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8 5.7 5.4 5.1 5.7 5
##  [28] 5.2 5.2 4.7 4.8 5.4 5.2 5.5 4.9 5.0 5.5 4.9 4.4 5.1 5.0 4.5 4.4 5.0 5.1 4.8 5
##  [55] 6.5 5.7 6.3 4.9 6.6 5.2 5.0 5.9 6.0 6.1 5.6 6.7 5.6 5.8 6.2 5.6 5.9 6.1 6.3 6
##  [82] 5.5 5.8 6.0 5.4 6.0 6.7 6.3 5.6 5.5 5.5 6.1 5.8 5.0 5.6 5.7 5.7 6.2 5.1 5.7 6
## [109] 6.7 7.2 6.5 6.4 6.8 5.7 5.8 6.4 6.5 7.7 7.7 6.0 6.9 5.6 7.7 6.3 6.7 7.2 6.2 6
## [136] 7.7 6.3 6.4 6.0 6.9 6.7 6.9 5.8 6.8 6.7 6.7 6.3 6.5 6.2 5.9
```

```
mean(iris.longueurs[,"Sepal.Length"])
```

```
## [1] 5.843333
```

3.3.3 Les types de vecteurs

les vecteurs en résumé peuvent être : - des numéros entiers, **int** - des chaines de caractères, **chr** - des logiques, **logi** - des réels, **num** - ...

On peut créer un vecteur d'entiers

```
c(1,3)
```

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

Un vecteur de logique: (T pour vrai, F pour faux)

```
c(T,F,T,F,F)
```

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

Et là surprise : si on demande à R de nous retourner la première et la troisième variable de iris

```
str(iris[,c(1,3)])
```

```
## 'data.frame':    150 obs. of  2 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
```

Plus compliqué. On sait qu'il y a 5 variables dans iris ? On est d'accord ? Donc si on lui demande de nous renvoyer la variable quand c'est vrai et de ne pas nous la renvoyer quand c'est faux ?

```
str(iris[,c(T,F,T,F,F)])
```

```
## 'data.frame':    150 obs. of  2 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
```

En fait les **data.frame**s sont des agrégats de vecteurs que l'on peut indexer avec des vecteurs.

Pourquoi on ne peut pas faire :

```
mean(iris.longueurs)
```

parce qu'on a deux variables ? R refuse de faire ce qui n'a pas de sens.

Après tout on voudrait faire la moyenne de sépale et de pétale. Déjà mais ça pourrait être pire :

```
mean(iris[,c("Sepal.Length","Species")])
```

C'est la catastrophe. Vous essayez de faire une moyenne sur une variable texte et une variable continue. C'est faux.

Pour faire le résumer d'une variable texte:

```
table(iris["Species"])
```

```
##
##     setosa versicolor  virginica
##         50         50         50
```

3.3.4 Pour résumé

On a les **data.frame**, on a les **vector** de différents types. On sait qu'on peut sélectionner les variables par l'intermédiaire de vecteurs.

3.4 Sélection des individus

Intuitivement, comment sélectionner des individus ?

Ca marche comme pour les variables, on utilise des vecteurs ?

On veut les individus de 1 et 5.

```
str(iris[c(1,5),])
```

```
## 'data.frame':    2 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 5
##  $ Sepal.Width : num  3.5 3.6
##  $ Petal.Length: num  1.4 1.4
##  $ Petal.Width : num  0.2 0.2
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1
```

Attention à la place de la virgule. Cette fois on sélectionne des lignes. A gauche de la virgule pour des lignes et à droite pour les colonnes.

C'est tout bon.

On a vu qu'il y avait trois espèces. Si on veut sélectionner ceux qui sont du type **versicolor** ?

On se rappele des vecteurs de logique: là où iris[,"Species"] vaudra **versicolor** on sélectionne et là où ce n'est pas **versicolor** on ne sélectionne pas.

On ne va pas le faire à la main. On ne fait rien à la main sous R.

```
especes <- c("versicolor","truc","versicolor","setosa")
especes
```

```
## [1] "versicolor" "truc"       "versicolor" "setosa"
```

```
especes=="versicolor"
```

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

On l'adapte pour notre cas :

```
iris[,"Species"]=="versicolor"
```

```
##   [1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [19] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [37] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
##  [55]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [73]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
##  [91]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE
## [109] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [127] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [145] FALSE FALSE FALSE FALSE FALSE FALSE
```

Donc on indexe les individus :

```
versicolor <- iris[iris[,"Species"]=="versicolor",]
```

Faites un point pour voir si tout est conforme dans votre esprit sur la place des accolades, etc. En fait c'est le **old-fashioned** R.

En fait ça commence à devenir compliqué, alors des gens on fait des fonctions qui génère des vecteurs… à partir de mots anglais.

On va créer ainsi de gauche à droite des sous espaces pour ne retenir que ce qui nous intéresse.

Exemple : from iris, filter Species==“versicolor”,

```
require(tidyverse)
```

```
iris |> filter(Species=="setosa")
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1         3.5          1.4         0.2  setosa
## 2           4.9         3.0          1.4         0.2  setosa
## 3           4.7         3.2          1.3         0.2  setosa
## 4           4.6         3.1          1.5         0.2  setosa
## 5           5.0         3.6          1.4         0.2  setosa
## 6           5.4         3.9          1.7         0.4  setosa
## 7           4.6         3.4          1.4         0.3  setosa
## 8           5.0         3.4          1.5         0.2  setosa
## 9           4.4         2.9          1.4         0.2  setosa
## 10          4.9         3.1          1.5         0.1  setosa
## 11          5.4         3.7          1.5         0.2  setosa
## 12          4.8         3.4          1.6         0.2  setosa
## 13          4.8         3.0          1.4         0.1  setosa
## 14          4.3         3.0          1.1         0.1  setosa
## 15          5.8         4.0          1.2         0.2  setosa
## 16          5.7         4.4          1.5         0.4  setosa
## 17          5.4         3.9          1.3         0.4  setosa
## 18          5.1         3.5          1.4         0.3  setosa
## 19          5.7         3.8          1.7         0.3  setosa
## 20          5.1         3.8          1.5         0.3  setosa
## 21          5.4         3.4          1.7         0.2  setosa
## 22          5.1         3.7          1.5         0.4  setosa
## 23          4.6         3.6          1.0         0.2  setosa
## 24          5.1         3.3          1.7         0.5  setosa
## 25          4.8         3.4          1.9         0.2  setosa
## 26          5.0         3.0          1.6         0.2  setosa
## 27          5.0         3.4          1.6         0.4  setosa
## 28          5.2         3.5          1.5         0.2  setosa
## 29          5.2         3.4          1.4         0.2  setosa
## 30          4.7         3.2          1.6         0.2  setosa
## 31          4.8         3.1          1.6         0.2  setosa
## 32          5.4         3.4          1.5         0.4  setosa
## 33          5.2         4.1          1.5         0.1  setosa
## 34          5.5         4.2          1.4         0.2  setosa
```

```
## 35          4.9         3.1          1.5         0.2  setosa
## 36          5.0         3.2          1.2         0.2  setosa
## 37          5.5         3.5          1.3         0.2  setosa
## 38          4.9         3.6          1.4         0.1  setosa
## 39          4.4         3.0          1.3         0.2  setosa
## 40          5.1         3.4          1.5         0.2  setosa
## 41          5.0         3.5          1.3         0.3  setosa
## 42          4.5         2.3          1.3         0.3  setosa
## 43          4.4         3.2          1.3         0.2  setosa
## 44          5.0         3.5          1.6         0.6  setosa
## 45          5.1         3.8          1.9         0.4  setosa
## 46          4.8         3.0          1.4         0.3  setosa
## 47          5.1         3.8          1.6         0.2  setosa
## 48          4.6         3.2          1.4         0.2  setosa
## 49          5.3         3.7          1.5         0.2  setosa
## 50          5.0         3.3          1.4         0.2  setosa
```

3.4.1 Calcul de la moyenne et nouveaux générateurs

Pour la moyenne des longueurs de sépales ?

```
iris |> summarise(moyenne=mean(Sepal.Length))
```

```
##    moyenne
## 1 5.843333
```

Ce qui devient :

```
iris |> filter(Species=="setosa") |> summarise(moyenne=mean(Sepal.Length))
```

```
##   moyenne
## 1   5.006
```

Mais y'a des choses plus pratique.

```
iris |> group_by(Species) |> summarise(moyenne=mean(Sepal.Length))
```

```
## # A tibble: 3 x 2
##   Species    moyenne
##   <fct>        <dbl>
## 1 setosa        5.01
## 2 versicolor    5.94
## 3 virginica     6.59
```

```
quantile(iris[,"Sepal.Length"])
```

```
##   0%  25%  50%  75% 100%
##  4.3  5.1  5.8  6.4  7.9
```

```
iris |> group_by(Species) |> summarise(moy.Sepal.Length=mean(Sepal.Length),ec=sd(Sepal.Length),me
```

```
## # A tibble: 3 x 4
##   Species    moy.Sepal.Length    ec mediane
##   <fct>                 <dbl> <dbl>   <dbl>
## 1 setosa                 5.01 0.352     5
## 2 versicolor             5.94 0.516     5.9
## 3 virginica              6.59 0.636     6.5
```

```
iris |> group_by(Species) |> summarise(moy.Sepal.Length=mean(Sepal.Length),ec=sd(Sepal.Length),me
```

```
## # A tibble: 3 x 5
##   Species    moy.Sepal.Length    ec mediane   q90
##   <fct>                 <dbl> <dbl>   <dbl> <dbl>
## 1 setosa                 5.01 0.352     5    5.41
## 2 versicolor             5.94 0.516     5.9  6.7
## 3 virginica              6.59 0.636     6.5  7.61
```

Etc...

Ah au fait c'est quoi comme vient de retourner ?

```
str(iris |> group_by(Species) |> summarise(sepal.length=mean(Sepal.Length),
                                           sepal.width=mean(Sepal.Width)))
```

```
## tibble [3 x 3] (S3: tbl_df/tbl/data.frame)
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 2 3
##  $ sepal.length: num [1:3] 5.01 5.94 6.59
##  $ sepal.width : num [1:3] 3.43 2.77 2.97
```

Calculer la moyenne de toutes les colonnes sauf Species ?

```
iris |> group_by(Species) |> summarise(across(Sepal.Length:Petal.Width,mean))
```

```
## # A tibble: 3 x 5
##   Species    Sepal.Length Sepal.Width Petal.Length Petal.Width
##   <fct>             <dbl>       <dbl>        <dbl>       <dbl>
## 1 setosa             5.01        3.43         1.46       0.246
## 2 versicolor         5.94        2.77         4.26       1.33
## 3 virginica          6.59        2.97         5.55       2.03
```

Chapter 4

Données de Parcoursup

Les données de ParcourSup viennent de là : Parcoursup 2023 - vœux de poursuite d'études et de réorientation dans l'enseignement supérieur et réponses des établissements

sinon vous avez les métiers en tensions : Taux de pression et d'emploi pour les diplômes de la voie professionnelle

En suivant les liens vous pouvez télécharger les fichiers Excel.

Déplacer le fichier Excel à la racine de votre projet. puis

```
library(readxl)
```

```
parcours <- read_excel("data/fr-esr-parcoursup.xlsx")
```

```
## New names:
## * `Filière de formation` -> `Filière de formation...10`
## * `Filière de formation` -> `Filière de formation...14`
```

Je viens de créer une data.frame du nom de parcours avec le contenu du fichier Excel.

Pour sélectionner les variables, utiliser le raccourci TAB.

Pour charger un fichier SPSS, il faut aussi un paquet supplémentaire :

```
require(haven)
```

```
patient <- read_sav("data/patient.sav")
```

C'est le même paquet pour les formats **SAS** (**sas7bdat**) et **STATA**. On trouve le chargement des mêmes types de fichier dans le paquet **foreign** mais attention ce sont pour les vieux formats de fichiers.

Par exemple pour SAS, il suffit de changer de fonction :

```
patient <- read_sas("patient.sas7bdat")
```

Lors de l'import, de SAS, SPSS, il conserve le type de la variable. Quand on veux importer un fichier Excel ou un fichier texte, cela est différent.

On va prendre l'exemple de fichier texte : l'importation se fait en fait en trois temps.

```
library(readr)
patient <- read_csv("data/patient.csv")
```

```
## Rows: 200 Columns: 17
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (5): UID, Hopital, sexe, scoliose, drepano
## dbl (12): poids, vitaux, CIM2, age, dureeopmin, postopj, ACP, peridurale, periACP, r
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
table(patient$scoliose)
```

```
##
##  ante autre  post
##     3     2    20
```

Dans un premier temps, la fonction **read_csv** va parcourir les 1000 premières lignes du fichier à la découverte de : - du séparateur entre les champs - du séparateur de décimales - le type de chaque colonne.

Si par exemple il trouve que des chiffres dans une colonne, le type sera **dbl**. Par contre s'il trouve un mélange de charactères et de chiffres, là rien ne va plus. Ca peut se produire par exemple lorsque vous avez des chiffres mélangés à des valeurs manquantes qui sont représentés par des valeurs textes ou bien des symboles textuels.

Les petites machines qui transforment les données en données typées sont des parser. Elles sont d'ailleurs disponible à part :

```
str(parse_double(c("1.56", "NA", "NA")))
```

```
##  num [1:3] 1.56 NA NA
```

NA est reconnu comme valeur manquante alors pas de souci, le 1.56 est reconnu.

et si on mettait 1,56 ?

```
str(parse_double(c("1,56", "NA", "NA")))
```

```
## Warning: 1 parsing failure.
## row col               expected actual
```

```
##   1  -- no trailing characters   1,56
```

```
##  num [1:3] NA NA NA
##  - attr(*, "problems")= tibble [1 x 4] (S3: tbl_df/tbl/data.frame)
##   ..$ row     : int 1
##   ..$ col     : int NA
##   ..$ expected: chr "no trailing characters"
##   ..$ actual  : chr "1,56"
```

pas terrible ce qui suit :

```
str(parse_double(c("1.56", "NR", "NR")))
```

```
## Warning: 2 parsing failures.
## row col expected actual
##   2  -- a double     NR
##   3  -- a double     NR
```

```
##  num [1:3] 1.56 NA NA
##  - attr(*, "problems")= tibble [2 x 4] (S3: tbl_df/tbl/data.frame)
##   ..$ row     : int [1:2] 2 3
##   ..$ col     : int [1:2] NA NA
##   ..$ expected: chr [1:2] "a double" "a double"
##   ..$ actual  : chr [1:2] "NR" "NR"
```

On rétablit la situation normale en mettant na = NR :

```
str(parse_double(c("1.56", "NA", "NA"),na = "NR"))
```

```
## Warning: 2 parsing failures.
## row col expected actual
##   2  -- a double     --
##   3  -- a double     --
```

```
##  num [1:3] 1.56 NA NA
##  - attr(*, "problems")= tibble [2 x 4] (S3: tbl_df/tbl/data.frame)
##   ..$ row     : int [1:2] 2 3
##   ..$ col     : int [1:2] NA NA
##   ..$ expected: chr [1:2] "a double" "a double"
##   ..$ actual  : chr [1:2] "NA" "NA"
```

Dans la jungle des parsers, on a les parsers : - parse_logical() - parse_integer() - parse_double() - parse_character() - parse_number() - parse_factor() - parse_datetime() (may be the force with you) - ...

Si vous avez bien suivi, la machine va lire les n premières lignes et à chaque colonne essayer de deviner le type de variable et appeler le parser qui va bien : ce sont les fonctions **guess_**.

Ceux sont eux qui vont décider du type de variable que vous importez. Pour le faire vous même, il suffit de spécificier chaque à la main :

```
read_csv("data/iris.csv", col_types = list(
  Sepal.Length = col_double(),
  Sepal.Width = col_double(),
  Petal.Length = col_double(),
  Petal.Width = col_double(),
  Species = col_factor(c("setosa", "versicolor", "virginica"))
))
```

```
## Warning: The following named parsers don't match the column names: Sepal.Length, Sep
## Petal.Width, Species
```

```
## Warning: One or more parsing issues, call `problems()` on your data frame for detail
##   dat <- vroom(...)
##   problems(dat)
```

```
## # A tibble: 150 x 1
##    `Sepal.Length;Sepal.Width;Petal.Length;Petal.Width;Species`
##    <chr>
##  1 5,1;3,5;1,4;NA;setosa
##  2 4,9;3;1,4;NA;setosa
##  3 4,7;3,2;1,3;NA;setosa
##  4 4,6;3,1;1,5;NA;setosa
##  5 5;3,6;1,4;NA;setosa
##  6 5,4;3,9;1,7;0,4;setosa
##  7 4,6;3,4;1,4;0,3;setosa
##  8 5;3,4;1,5;0,2;setosa
##  9 4,4;2,9;1,4;0,2;setosa
## 10 4,9;3,1;1,5;0,1;setosa
## # i 140 more rows
```

Sur cette ligne, c'est un peu complexe et surtout cela fait appel à deux élements que vous connaissez pas. Les **list**s et les **factor**s.

Un **factor** est un ensemble de valeurs fini : c'est comme ça que vous pouvez coder un ensemble de valeurs que vous pouvez énumérer et que vous utiliseriez par exemple dans une expérience. Par exemple, on peut avoir comme facteur :

- le nombre de cylindres de mtcars 4, 6 ou 8
- les médicaments dans une expérience en double aveugle: A, B, C
- ...

L'idée est qu'un facteur est à utiliser dans une ANOVA (un test de différences de moyennes sur 1 à k groupes).

La liste est une **data.frame** libertaire : par libértaire j'entends qu'on peut mettre n'importe objet et l'indexer (presque) comme une **data.frame**.

```
a <- list(iris,c(1,2,3),LETTERS,mtcars[,c("cyl","mpg")])
str(a)
```

```
## List of 4
##  $ :'data.frame':    150 obs. of  5 variables:
##   ..$ Sepal.Length: num [1:150] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##   ..$ Sepal.Width : num [1:150] 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##   ..$ Petal.Length: num [1:150] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##   ..$ Petal.Width : num [1:150] 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##   ..$ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ : num [1:3] 1 2 3
##  $ : chr [1:26] "A" "B" "C" "D" ...
##  $ :'data.frame':    32 obs. of  2 variables:
##   ..$ cyl: num [1:32] 6 6 4 6 8 6 8 4 4 6 ...
##   ..$ mpg: num [1:32] 21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
```

On voit que c'est un type **list**, les **data.frame**.

Chapter 5

Résumés rapides descriptives

```
library(gtsummary)
```

```
tbl_summary(mtcars)
```

5.1 Les premieres tables

Caractéristique	**N = 32**
mpg	19,2 (15,4 – 22,8)
cyl	
4	11 (34%)
6	7 (22%)
8	14 (44%)
disp	196 (121 – 326)
hp	123 (97 – 180)
drat	3,70 (3,08 – 3,92)
wt	3,33 (2,58 – 3,61)
qsec	17,71 (16,89 – 18,90)
vs	14 (44%)
am	13 (41%)
gear	
3	15 (47%)
4	12 (38%)
5	5 (16%)
carb	
1	7 (22%)
2	10 (31%)
3	3 (9,4%)
4	10 (31%)
6	1 (3,1%)
8	1 (3,1%)

```
tbl_summary(iris)
```

Caractéristique	**N = 150**
Sepal.Length	5,80 (5,10 – 6,40)
Sepal.Width	3,00 (2,80 – 3,30)
Petal.Length	4,35 (1,60 – 5,10)
Petal.Width	1,30 (0,30 – 1,80)
Species	
setosa	50 (33%)
versicolor	50 (33%)
virginica	50 (33%)

Plus fort,

```
tbl_summary(iris,by="Species")
```

Caractéristique	**setosa**, N = 50	**versicolor**, N = 50	**virginica**, N = 50
Sepal.Length	5,00 (4,80 – 5,20)	5,90 (5,60 – 6,30)	6,50 (6,23 – 6,90)
Sepal.Width	3,40 (3,20 – 3,68)	2,80 (2,53 – 3,00)	3,00 (2,80 – 3,18)
Petal.Length	1,50 (1,40 – 1,58)	4,35 (4,00 – 4,60)	5,55 (5,10 – 5,88)
Petal.Width	0,20 (0,20 – 0,30)	1,30 (1,20 – 1,50)	2,00 (1,80 – 2,30)

On peut rajouter un test statistique:

```
tbl_summary(iris,by="Species" ) %>% add_p() %>% add_overall()
```

Caractéristique	**Total**, N = 150	**setosa**, N = 50	**versicolor**, N = 50	**virginica**, N = 50
Sepal.Length	5,80 (5,10 – 6,40)	5,00 (4,80 – 5,20)	5,90 (5,60 – 6,30)	6,50 (6,23 – 6,90)
Sepal.Width	3,00 (2,80 – 3,30)	3,40 (3,20 – 3,68)	2,80 (2,53 – 3,00)	3,00 (2,80 – 3,18)
Petal.Length	4,35 (1,60 – 5,10)	1,50 (1,40 – 1,58)	4,35 (4,00 – 4,60)	5,55 (5,10 – 5,88)
Petal.Width	1,30 (0,30 – 1,80)	0,20 (0,20 – 0,30)	1,30 (1,20 – 1,50)	2,00 (1,80 – 2,30)

```
trial %>%
  tbl_cross(row = stage, col = trt, percent = "cell") %>%
  add_p() %>%
  bold_labels()
```

	Drug A	Drug B	**Total**	**p-valeur**
___T Stage___				0,9
T1	28 (14%)	25 (13%)	53 (27%)	
T2	25 (13%)	29 (15%)	54 (27%)	
T3	22 (11%)	21 (11%)	43 (22%)	
T4	23 (12%)	27 (14%)	50 (25%)	
___Total___	98 (49%)	102 (51%)	200 (100%)	

5.2 Tableaux croisés

On peut faire des tris croisés et choisir le sens des pourcentages : par cellule, par ligne ou par colonne.

```
patient %>%
  tbl_cross(row = Hopital, col = sexe, percent = "col") %>%
  add_p() %>%
  bold_labels()
```

	Feminin	Masculin	**Total**	**p-valeur**
___Hopital___				0,007
A	48 (42%)	52 (61%)	100 (50%)	
B	67 (58%)	33 (39%)	100 (50%)	
___Total___	115 (100%)	85 (100%)	200 (100%)	

5.3 Themes

```
theme_gtsummary_compact(set_theme = TRUE, font_size = NULL)

patient %>%
  tbl_cross(row = Hopital, col = sexe, percent = "col") %>%
  add_p() %>%
  bold_labels()
```

	Feminin	Masculin	**Total**	**p-valeur**
___Hopital___				0,007
A	48 (42%)	52 (61%)	100 (50%)	
B	67 (58%)	33 (39%)	100 (50%)	
___Total___	115 (100%)	85 (100%)	200 (100%)	

```
reset_gtsummary_theme()
```

5.4 Exportation

Pour les exporter avec le paquet **writexl** :

```
iris %>% tbl_summary(by=Species) %>%
  gtsummary::as_tibble() %>%
  writexl::write_xlsx(., "example_gtsummary1.xlsx")
```

Sinon

```
iris %>% tbl_summary(by=Species) %>%
  gtsummary::as_gt() %>%
  gt::gtsave(., "example_gtsummary3.rtf")
```

5.5 Personnalisation des statistiques

Sans qu'il soit nécessaire de fonction, vous pouvez personnaliser les sortie.

Il suffit de changer la valeur de **statistic**.

Les mots-clefs sont, pour les variables quantitatives et qualtitatives : - **{N_obs}** total number of observations

- **{N_miss}** number of missing observations
- **{N_nonmiss}** number of non-missing observations
- **{p_miss}** percentage of observations missing
- **{p_nonmiss}** percentage of observations not missing

Pour les variables quantitatives : - **{mean}**

- **{median}**
- **{sd}**
- les quantiles **{pDD}** avec DD un chiffre sur 100 comme **{p25}**
- **{min}**
- **{max}**

Pour les variables qualitatives : - **{p}** le pourcentage

- **{n}**, nombre d'observations dans la cellule

- nombre total **{N}**

Il suffit alors de préciser les statistiques à afficher :

```
iris %>% tbl_summary(statistic = list(
  all_continuous() ~ "{mean} ({sd}) {min} {max}",
  all_categorical() ~ "{n} / {N} ({p}%)")) %>%
  modify_footnote( all_stat_cols() ~ "Moyenne (EC) Min. Max.; Effectif cellule / Total (%)")
```

Characteristic	**N = 150**
Sepal.Length	5.84 (0.83) 4.30 7.90
Sepal.Width	3.06 (0.44) 2.00 4.40
Petal.Length	3.76 (1.77) 1.00 6.90
Petal.Width	1.20 (0.76) 0.10 2.50
Species	
setosa	50 / 150 (33%)
versicolor	50 / 150 (33%)
virginica	50 / 150 (33%)

Vous pouvez également le faire sur plusieurs lignes en spécifiant que vous voulez les statistiques sur plusieurs lignes avec un argument supplémentaire :

```
iris %>%
  tbl_summary(
    type = all_continuous() ~ "continuous2",
    statistic = list(all_continuous() ~ c(
      "{N_nonmiss}",
      "{mean} ({sd})",
      "{median} ({p25}, {p75})",
      "{min}, {max}"
    ), all_categorical() ~ "{n} / {N} ({p}%)"))
```

Characteristic	**N = 150**
Sepal.Length	
N	150
Mean (SD)	5.84 (0.83)
Median (IQR)	5.80 (5.10, 6.40)
Range	4.30, 7.90
Sepal.Width	
N	150
Mean (SD)	3.06 (0.44)
Median (IQR)	3.00 (2.80, 3.30)
Range	2.00, 4.40
Petal.Length	
N	150
Mean (SD)	3.76 (1.77)
Median (IQR)	4.35 (1.60, 5.10)
Range	1.00, 6.90
Petal.Width	
N	150
Mean (SD)	1.20 (0.76)
Median (IQR)	1.30 (0.30, 1.80)
Range	0.10, 2.50
Species	
setosa	50 / 150 (33%)
versicolor	50 / 150 (33%)
virginica	50 / 150 (33%)

Vous pouvez de même personnaliser en adaptant la langue :

```
theme_gtsummary_language(language = "fr", decimal.mark = ",", big.mark = " ")
iris %>%
  tbl_summary(
    type = all_continuous() ~ "continuous2",
    statistic = list(all_continuous() ~ c(
      "{N_nonmiss}",
      "{mean} ({sd})",
      "{median} ({p25}, {p75})",
      "{min}, {max}"
    ), all_categorical() ~ "{n} / {N} ({p}%)"))
```

Caractéristique	**N = 150**
Sepal.Length	
N	150
Moyenne (ET)	5,84 (0,83)
Médiane (EI)	5,80 (5,10, 6,40)
Étendue	4,30, 7,90
Sepal.Width	
N	150
Moyenne (ET)	3,06 (0,44)
Médiane (EI)	3,00 (2,80, 3,30)
Étendue	2,00, 4,40
Petal.Length	
N	150
Moyenne (ET)	3,76 (1,77)
Médiane (EI)	4,35 (1,60, 5,10)
Étendue	1,00, 6,90
Petal.Width	
N	150
Moyenne (ET)	1,20 (0,76)
Médiane (EI)	1,30 (0,30, 1,80)
Étendue	0,10, 2,50
Species	
setosa	50 / 150 (33%)
versicolor	50 / 150 (33%)
virginica	50 / 150 (33%)

5.6 Modèles

5.6.1 Exemple de modèle de régression linéaire

On peut faire aussi des modèles avec gtsummary :

```
tbl_merge(list(
  tbl_regression(
    lm(totalechelle~sexe+vitaux+dureeopmin,data=patient[patient$Hopital=="A",])
  ),
  tbl_regression(
    lm(totalechelle~sexe+vitaux+dureeopmin,data=patient[patient$Hopital=="B",])
  ))
)
```

Caractéristique	**Beta**	**95% IC**	**p-valeur**	**Beta**	**95% IC**	**p-valeur
sexe						
Feminin	—	—		—	—	
Masculin	-345	-528 – -162	<0,001	-314	-634 – 6,3	0,055
vitaux	18	11 – 25	<0,001	4,0	-7,8 – 16	0,5
dureeopmin	0,29	-0,58 – 1,2	0,5	1,2	-0,16 – 2,5	0,084

Une liste des fonctionnalités est disponible là : Ici

Chapter 6

Graphiques et ggplot

Les graphiques sont une composante de R qui est en partie à l'origine de son succès car on peut de très beaux et ce depuis la création de R.

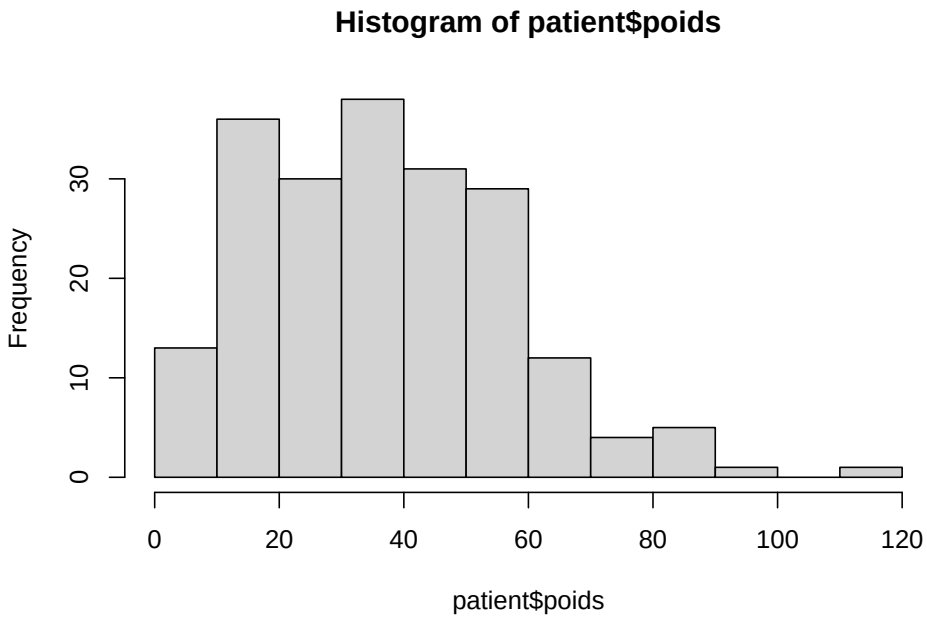
6.1 Les graphiques de base

Certaines sont très simples d'autres un peu plus compliquées. Nous verrons dans un premier temps les graphiques de base c'est-à-dire qui ne nécessitent pas de charger un **package**.

6.1.1 Pour les graphiques de chiffres

Les premières fonctions présentées sont les plus usuelles comme les histogrammes.

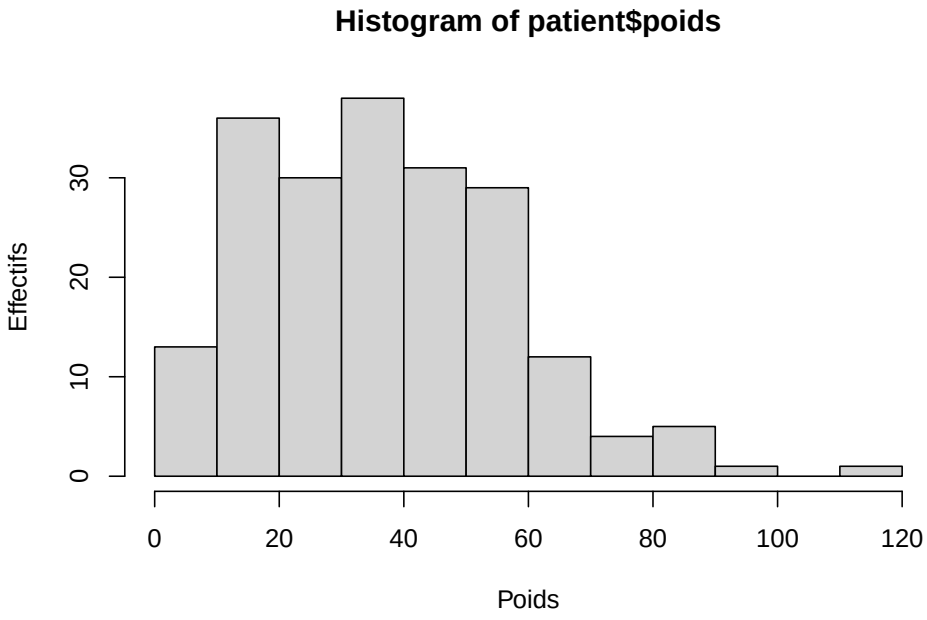
```
hist(patient$poids)
```



Ce n'est pas très esthétique. Il y a des arguments aux fonctions qui permettent d'améliorer les choses.

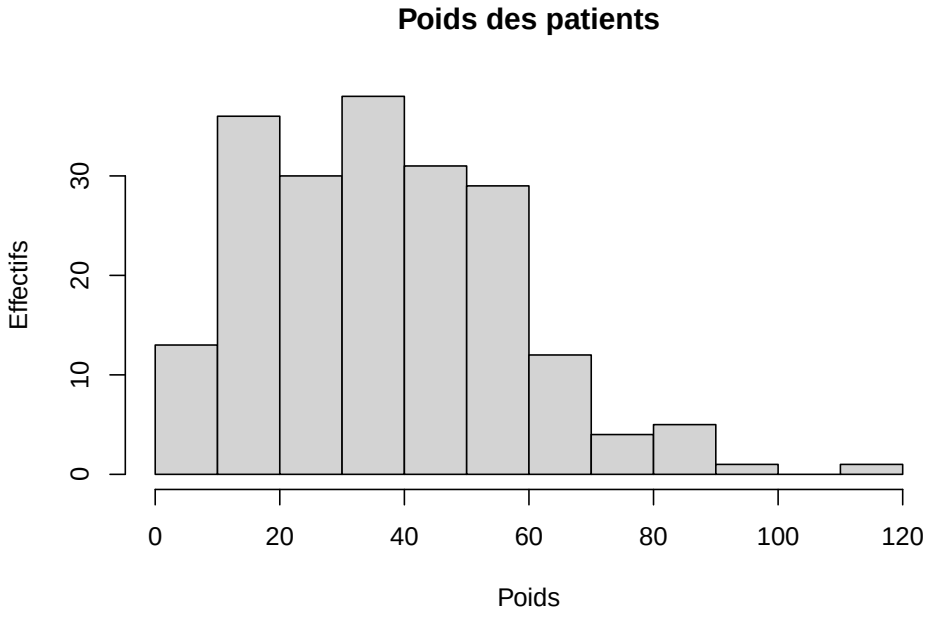
Déjà changer les noms des axes X et y, notamment se débarasser de **Frequency** qui est un faux ami en français.

```
hist(patient$poids,xlab="Poids",ylab="Effectifs")
```



Ensuite le titre :

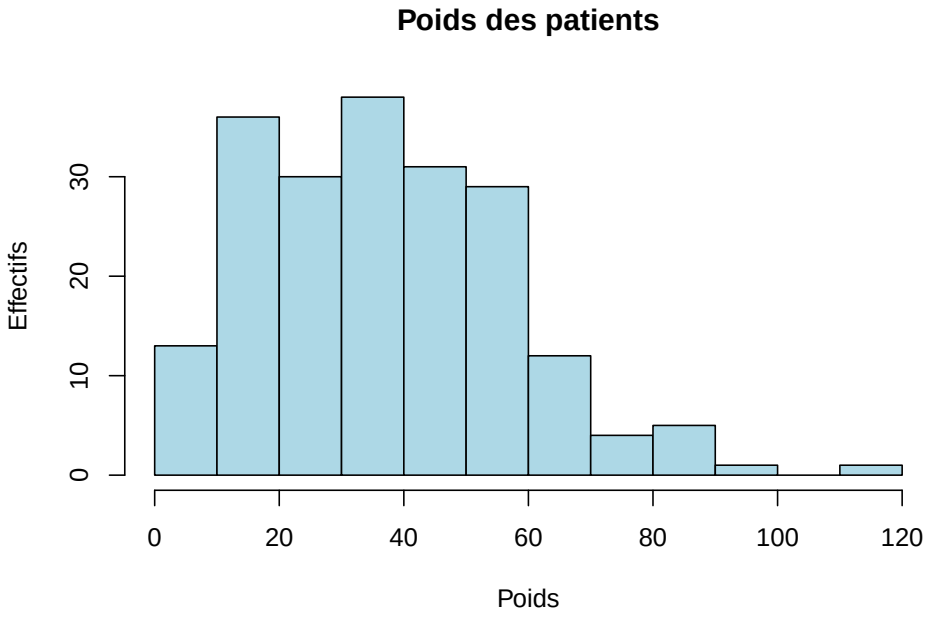
```
hist(patient$poids,xlab="Poids",ylab="Effectifs",main="Poids des patients")
```



Pour la couleur, c'est un peu plus compliqué. En effet il y a simple des couleurs qui répondent à leurs mots en anglais, la liste est là.

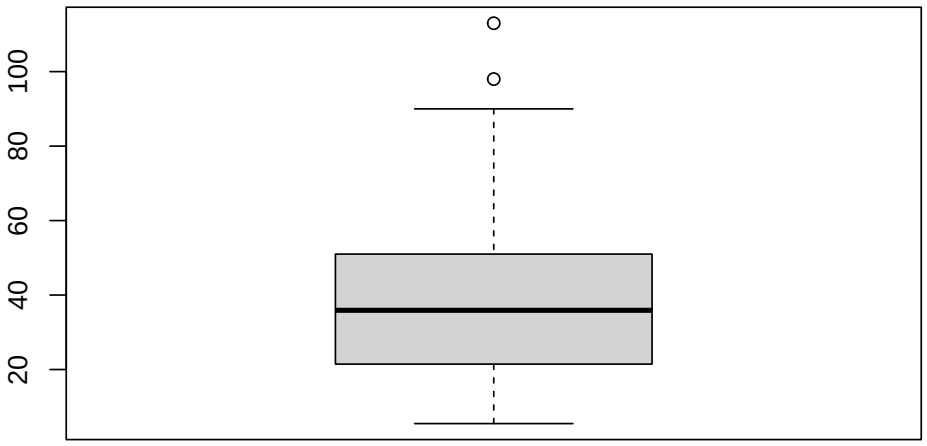
Mais les couleurs correspondent au codage web des couleurs qui sont en fait des hexadécimaux. Si vous voulez personnaliser plus les couleurs, je vous conseille le paquet **RColorBrewer** qui possède de jolis (et intelligents) assortiments de couleurs et de la lecture

```
hist(patient$poids,xlab="Poids",ylab="Effectifs",main="Poids des patients",col="lightblue")
```



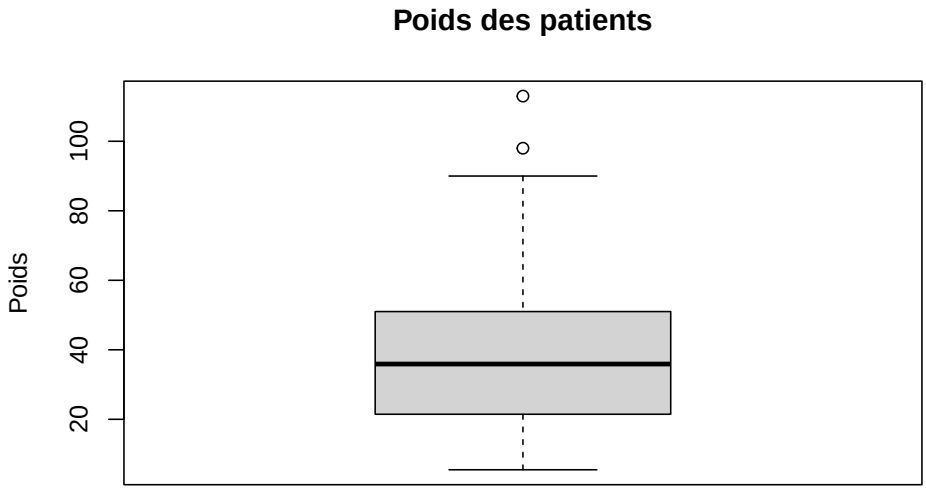
Ensuite il y a les boxplots ou boîtes à moustache pour les variables continues.

```
boxplot(patient$poids)
```



De même on arrange un peu :

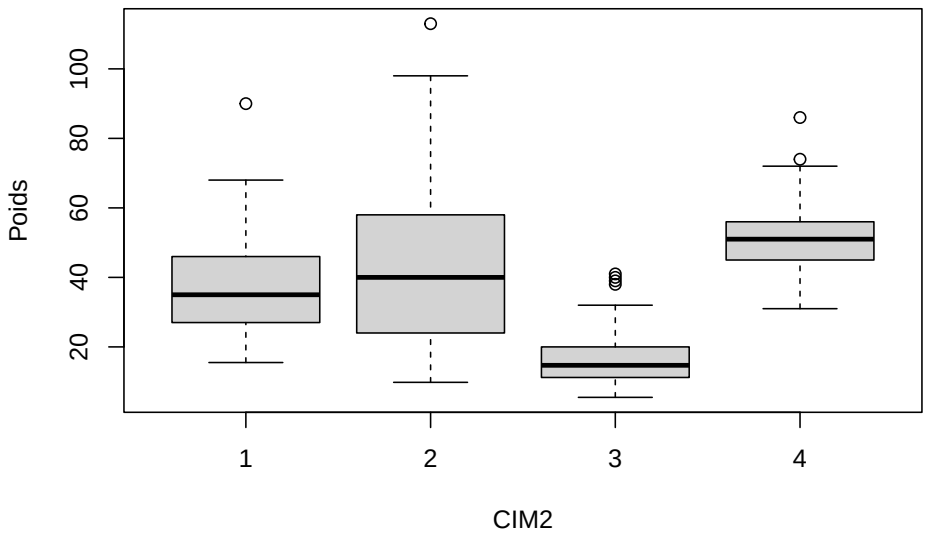
```
boxplot(patient$poids,ylab="Poids",main="Poids des patients")
```



Pour les boxplots on peut faire un peu mieux, par exemple pour segmenter par type de pathologies.

On passe la **data.frame** patient et on précise le nom de la variable qualitative qui doit "séparer" les tracés.

```
boxplot(poids ~ CIM2, data = patient,ylab="Poids",main="Poids des patients en fonction du CIM2")
```



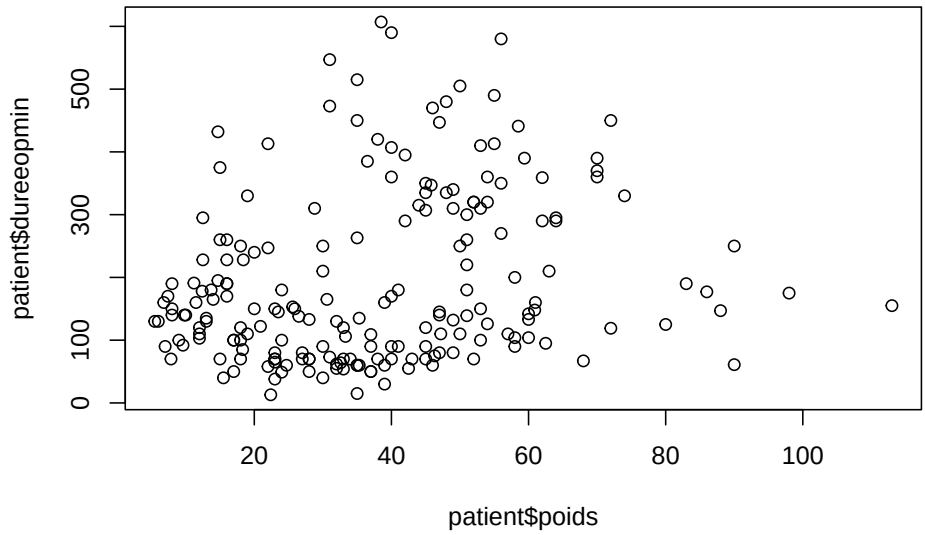
Le premier argument doit vous paraître un peu abstrait. En fait c'est une formule sous R. C'est l'équivalent de "patient=CIM2".

A gauche du ~ on place la variable à expliquer et à droite la ou les variables explicatives. Ici on en a une de chaque côté.

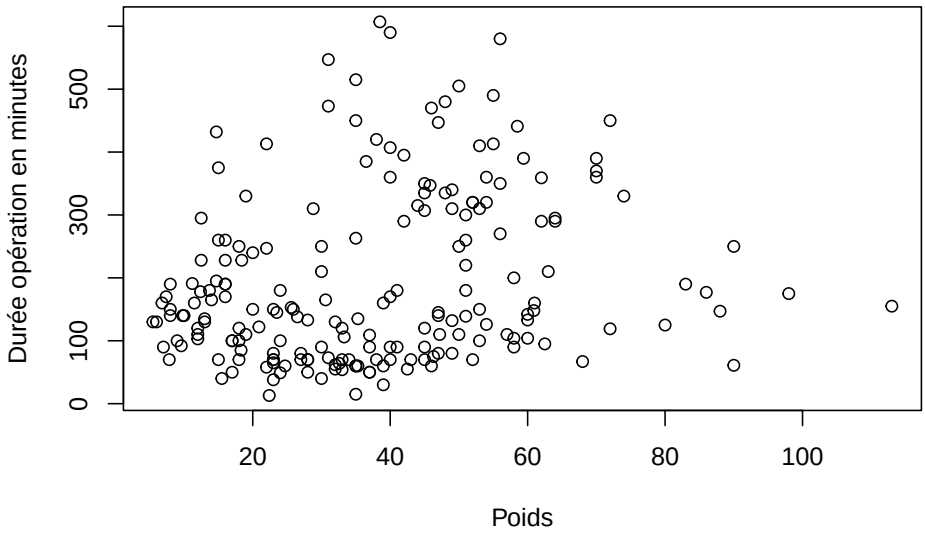
Le graphique le plus simple serait le **scatterplot**. On aurait pu commencer par lui :

Cette fois on a deux arguments qui sont la variable numérique des x en premier et la variable numérique des y en second.

```
plot(patient$poids,patient$dureeopmin)
```

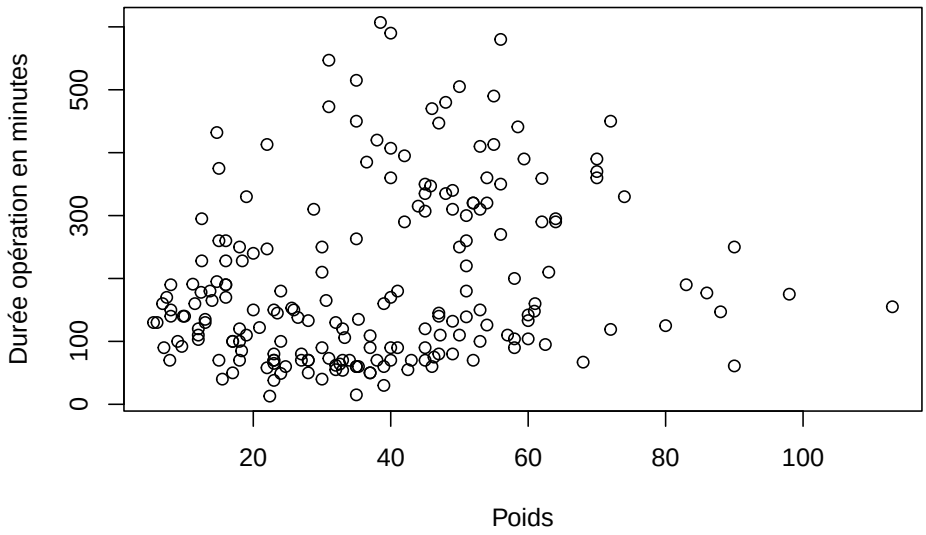



```
plot(patient$poids,patient$dureeopmin,xlab="Poids",ylab="Durée opération en minutes")
```



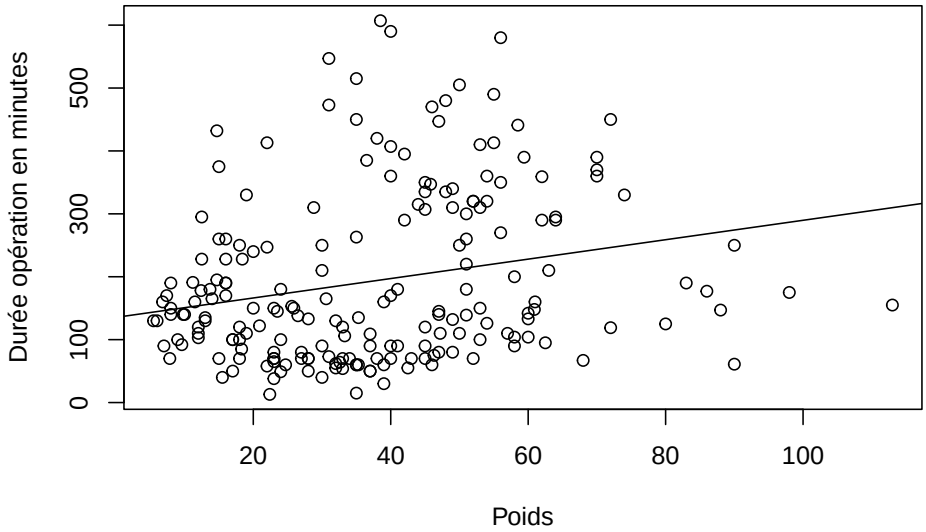
Qui aurait pu s'écrire :

```
plot(dureeopmin ~ poids,data=patient,xlab="Poids",ylab="Durée opération en minutes")
```



Si on veut tracer une ligne pour la régression linéaire, il faut faire appel à la fonction **lm** qui calcule la régression et R se charge du reste.

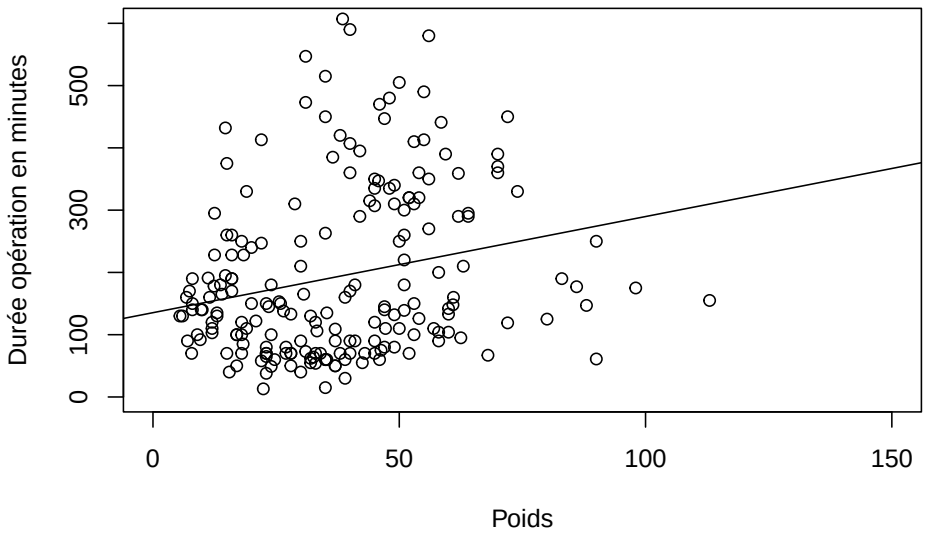
```
coefficients <- lm(dureeopmin~poids,data=patient)
plot(patient$poids,patient$dureeopmin,xlab="Poids",ylab="Durée opération en minutes")
abline(coefficients)
```



On voit ici que j'ai appellé **abline** après le plot. En effet, il est nécessaire de faire un **plot**, un **hist** ou une **boxplot** avant pour que R initialise le graphique notamment le calcul des coordonnées maximales et minimales.

D'ailleurs on peut les spécifier nous mêmes :

```
coefficients <- lm(dureeopmin~poids,data=patient)
plot(patient$poids,patient$dureeopmin,xlab="Poids",ylab="Durée opération en minutes",
     xlim=c(0,150),ylim=c(0,600))
abline(coefficients)
```

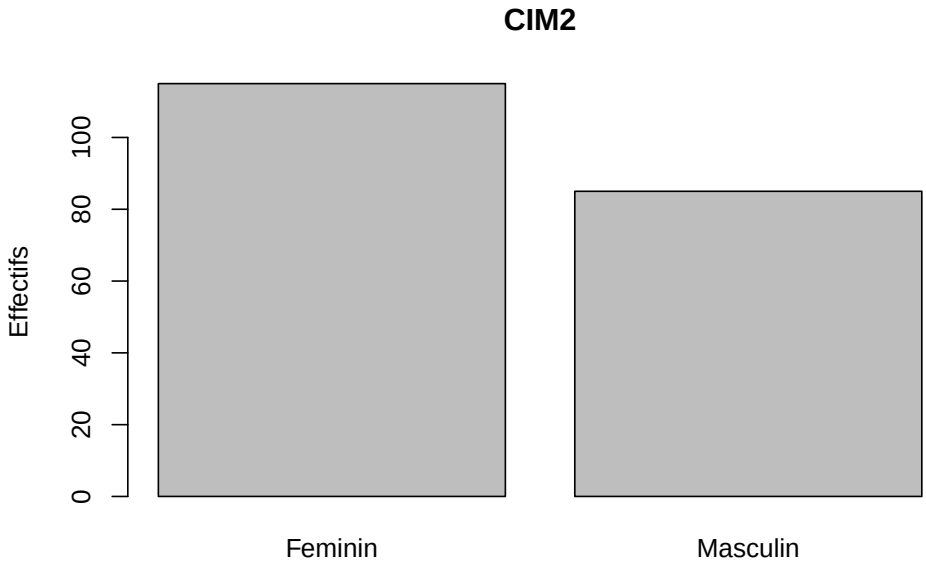


Pour sauvegarder un graphique, on doit le faire avant d'appeler la fonction **principale** et refermer le fichier avec la commande **dev.off**:

La dernière fonction à connaître pour les graphiques de base est le **barplot**.

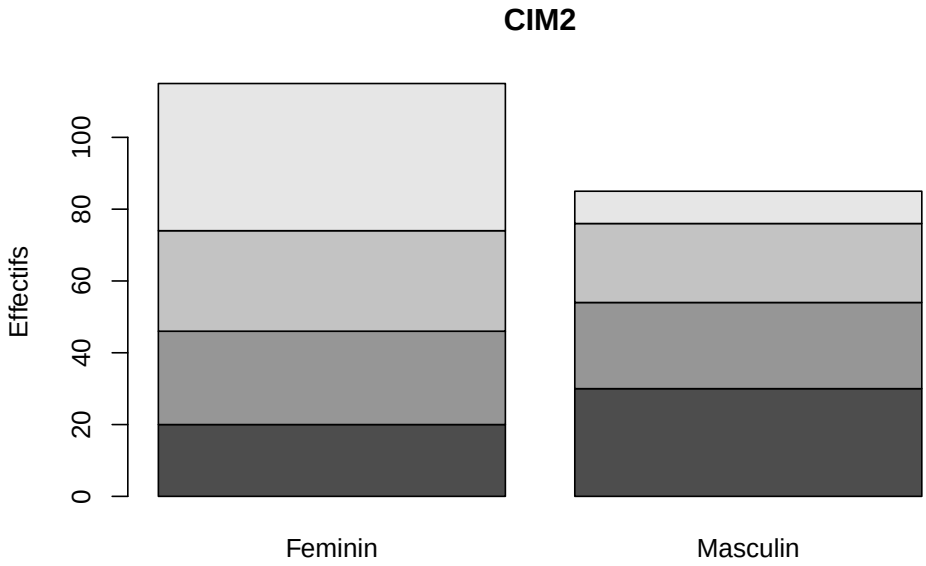
Il s'agit de représenter des tableaux de contingence, le plus simple étant à une dimension :

```
tableau <- table(patient$sexe)
barplot(tableau, main = "CIM2", ylab = "Effectifs")
```



On peut lui passer un argument à deux dimensions mais la table devient tout de suite difficile à lire.

```
tableau <- table(patient$CIM2,patient$sexe)
barplot(tableau, main = "CIM2", ylab = "Effectifs")
```



le tidyverse et ggplot

6.1.2 Introduction

Vous pourrez comme précédemment entendre parler des graphiques de base de même que des graphiques **lattice** mais le choucou du **tidyverse** c'est **ggplot2**.

C'est un éco-système de **packages** qui permet de faire la plupart des graphiques plus simplement et qui est basé sur le paquet **gplot2**.

Un livre gratuit lui est consacré là et une page en français là

On va reprendre notre grammaire. Il faut saisir que **ggplot2** fonctionne par couche. Sur une base, vous additionner des couches qui apporte la personnalisation des graphiques.

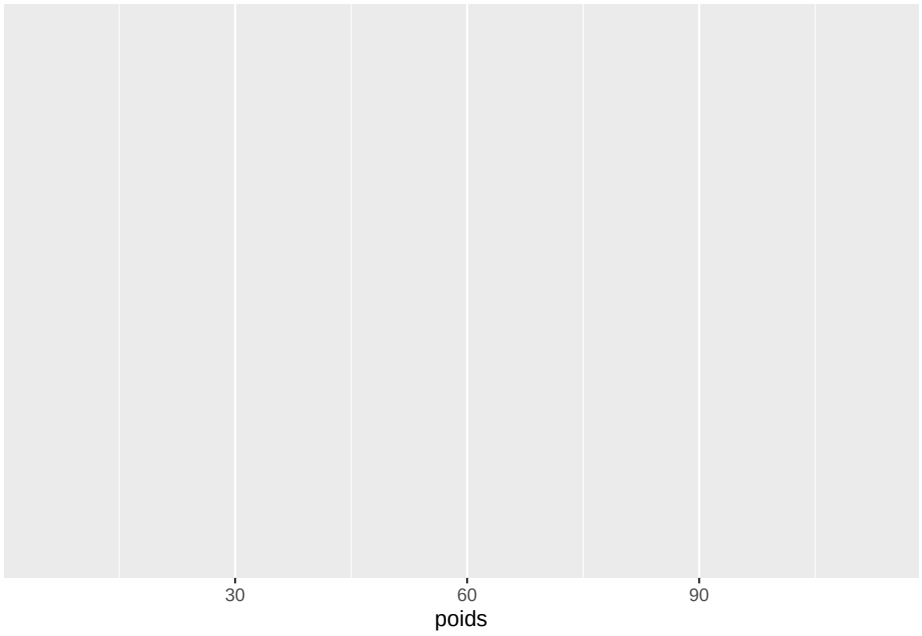
6.1.3 La base

Au tout départ, il faut lui passer une **data.frame**, c'est le passage obligé.

```
ggplot(patient)
```


Ensuite on précise les variables de travail. Pour l'histogramme, on en a qu'une :

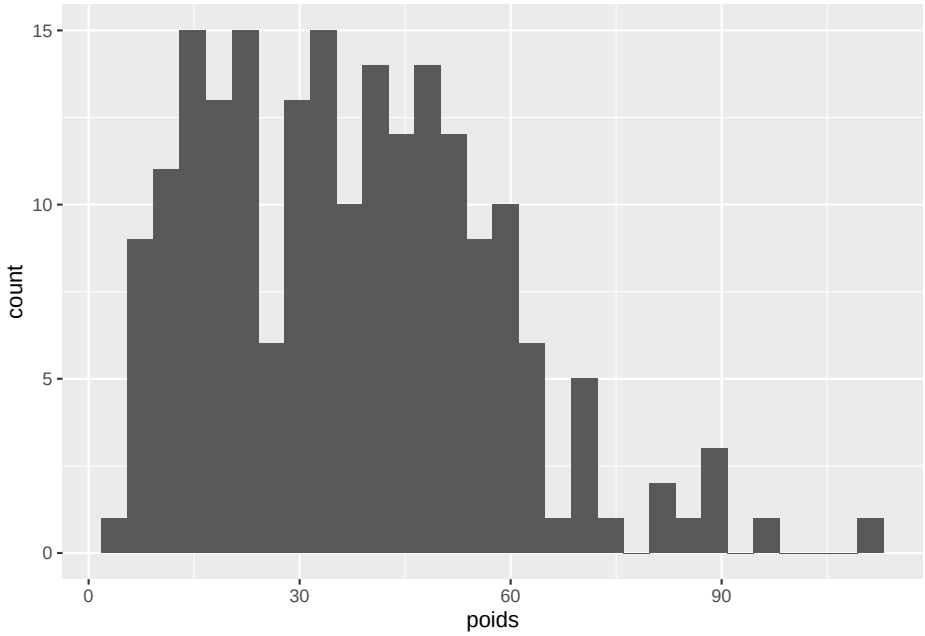
```
ggplot(patient,aes(poids))
```



Vous pouvez constater, que le logiciel a calculé et positionner les légendes pour créer un graphique avec poids comme variable des abscisses (horizontal).

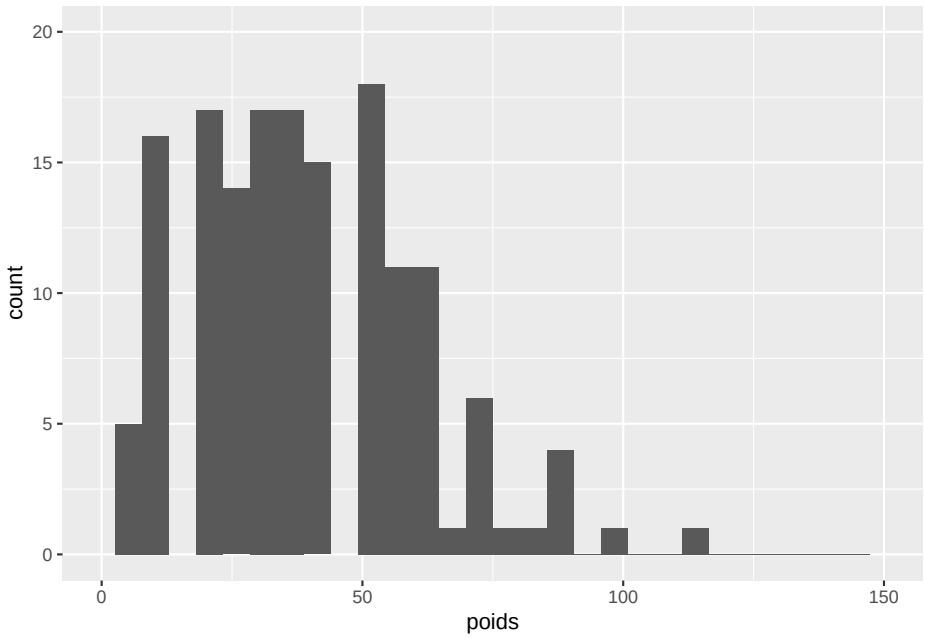
On personnalise en demandant un graphique de type histogramme. En additionnant littéralement:

```
ggplot(patient,aes(poids))+geom_histogram()
```



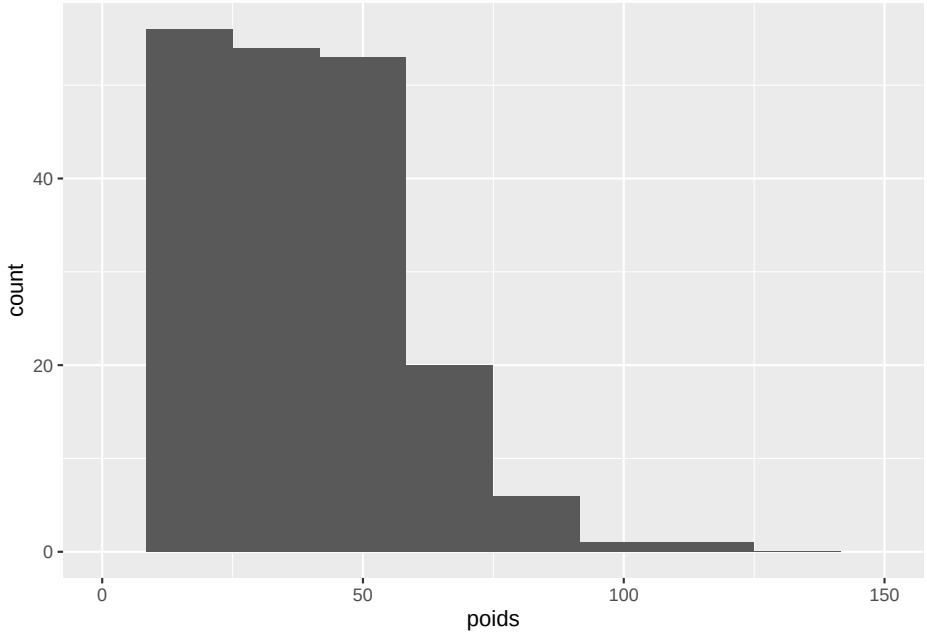
Pour modifier les limtes du graphiques, on rajoute :

```
ggplot(patient,aes(poids))+geom_histogram()+
  scale_x_continuous(limits = c(0,150)) +
  scale_y_continuous(limits = c(0,20))
```



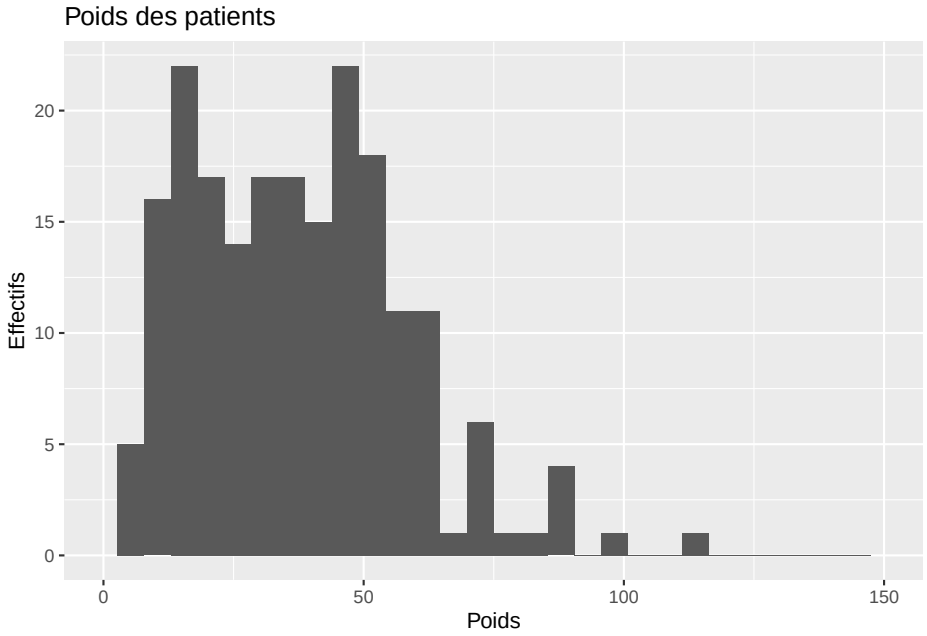
Si on veut modifier le nombre de barres verticales (la précision de l'histogramme), on précise l'option dans la couche de l'histogramme :

```
ggplot(patient,aes(poids))+geom_histogram(bins=10)+
  scale_x_continuous(limits = c(0,150))
```



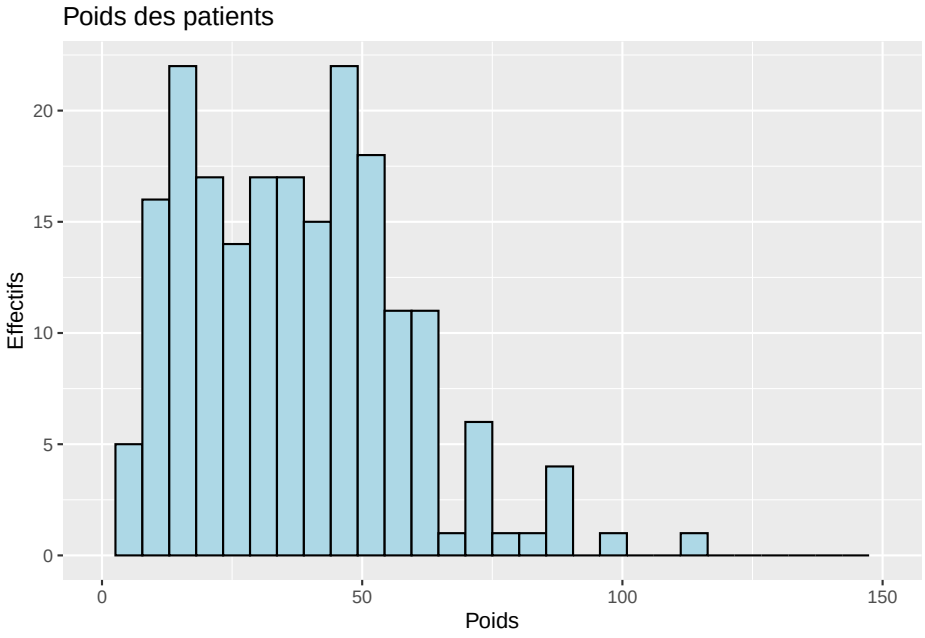
Pour les titres, c'est pareil, on ajoute des couches :

```
ggplot(patient,aes(poids))+geom_histogram()+
  scale_x_continuous(limits = c(0,150)) +
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Effectifs")
```



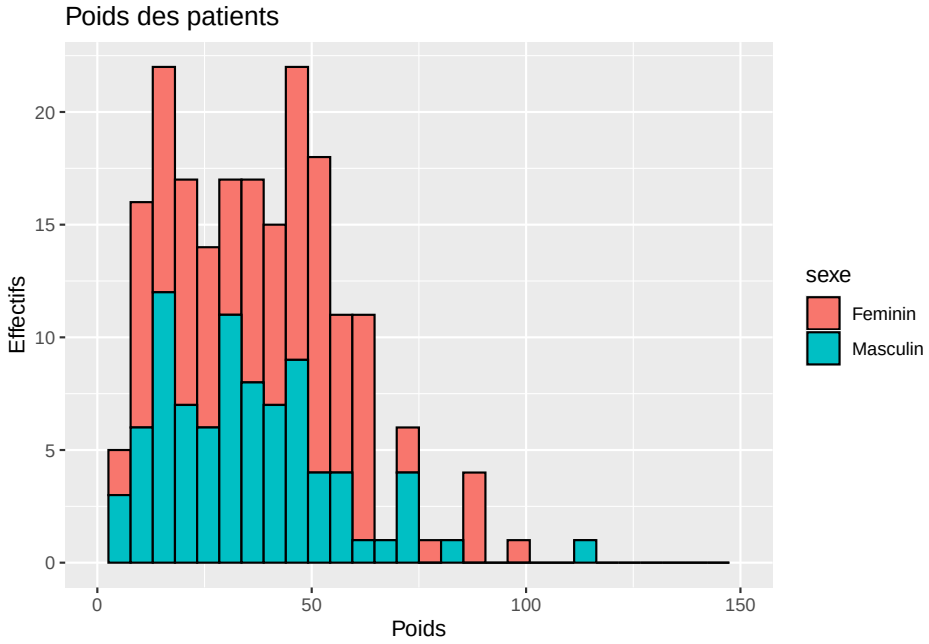
On peut ajouter des propriétés esthétiques comme la couleur, par exemple :

```
ggplot(patient,aes(poids))+geom_histogram(fill ="lightblue", colour = "black")+
  scale_x_continuous(limits = c(0,150)) +
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Effectifs")
```



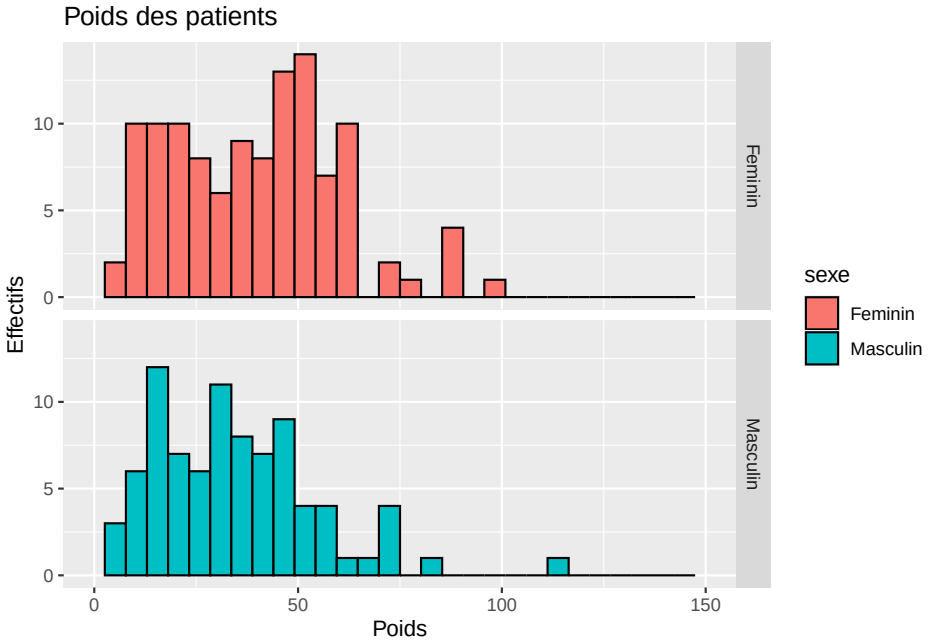
Là où **ggplot2** sort du lot, c'est sa capacité à segmenter et à représenter avec une bonne grammaire graphique

```
ggplot(patient,aes(poids,fill=sexe))+geom_histogram(color="black")+
  scale_x_continuous(limits = c(0,150)) +
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Effectifs")
```



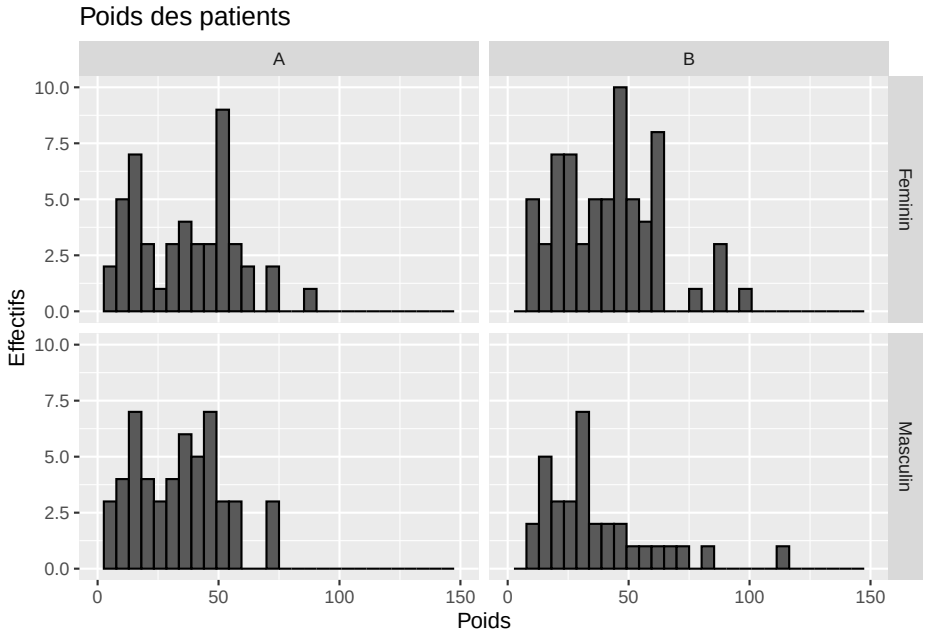
On a l'ajout de couleurs ou alors deux graphiques avec des unités bien choisies:

```
ggplot(patient,aes(poids,fill=sexe))+geom_histogram(color="black")+
  scale_x_continuous(limits = c(0,150)) +
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Effectifs") +
  facet_grid(sexe ~ .)
```



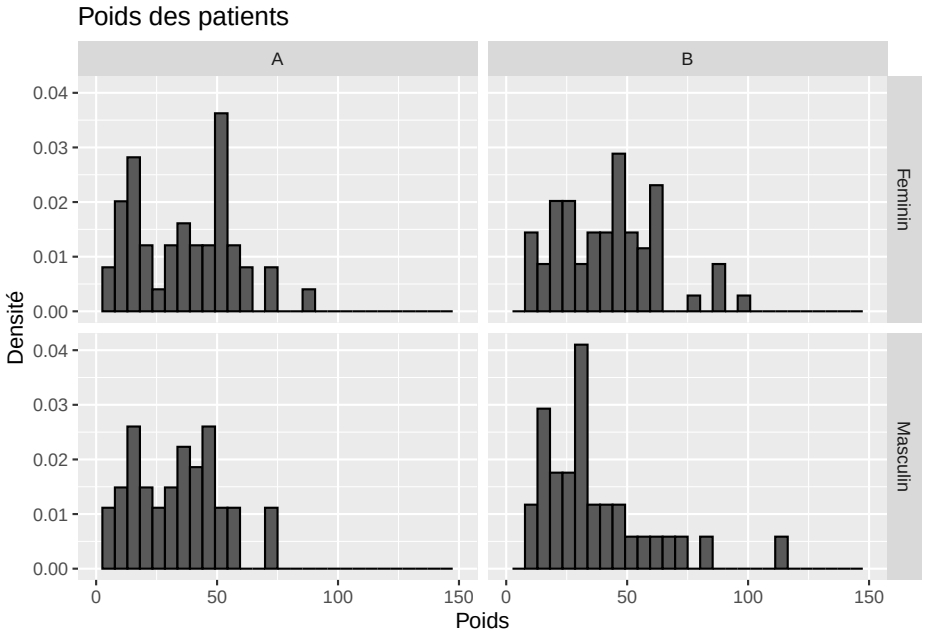
On a de nouveau une formule. Cette fois, c'est à gauche du ~ les lignes et à droite les colonnes :

```
ggplot(patient,aes(poids))+geom_histogram(color="black")+
  scale_x_continuous(limits = c(0,150)) +
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Effectifs") +
  facet_grid(sexe ~ Hopital)
```



On peut vouloir calculer la **densité** et non les effectifs dans ce cas :

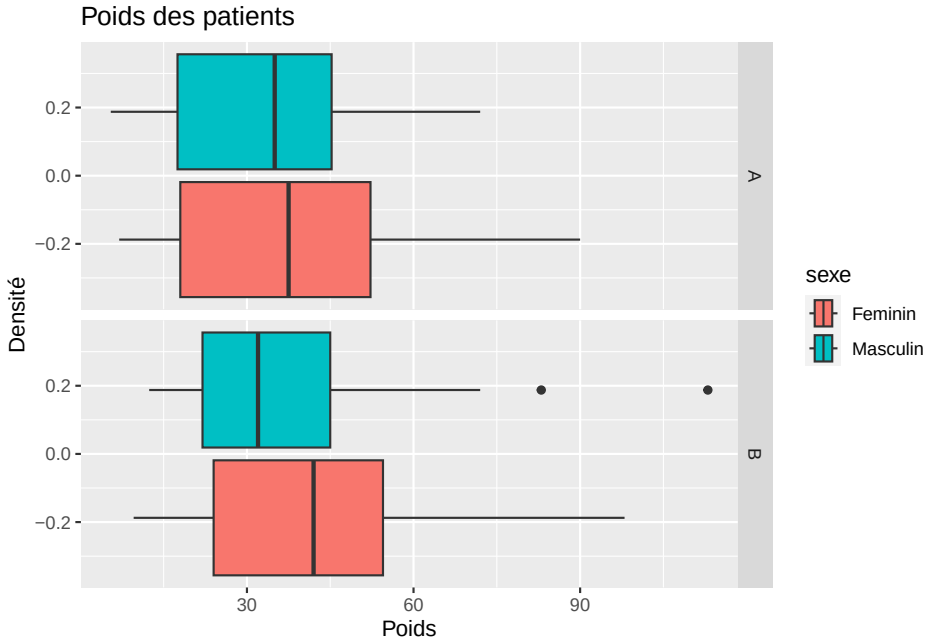
```
ggplot(patient,aes(poids))+geom_histogram(aes(y = ..density..),color="black")+
  scale_x_continuous(limits = c(0,150)) +
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Densité") +
  facet_grid(sexe ~ Hopital)
```



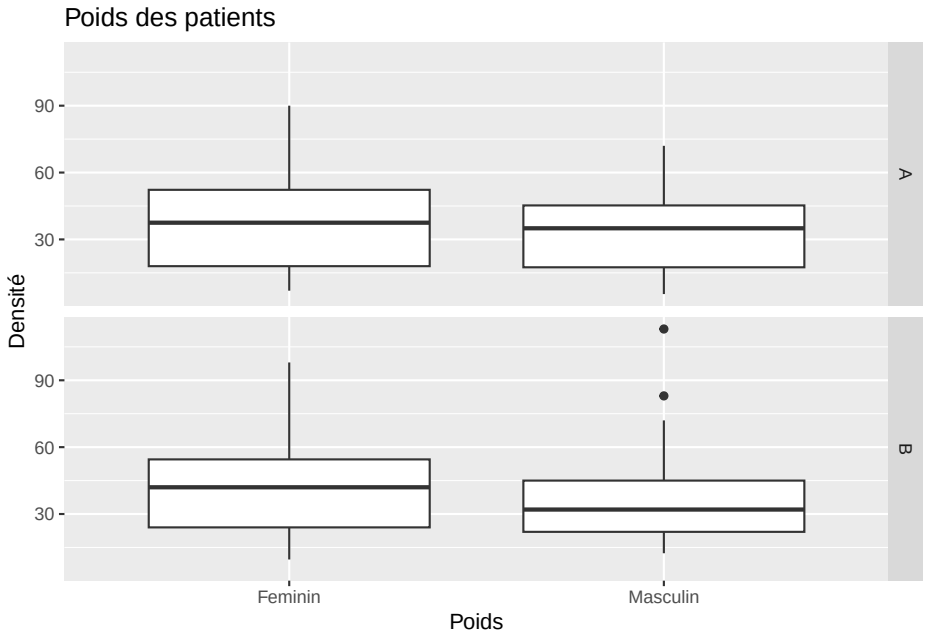
6.1.4 Les autres graphiques

Le boxplot :

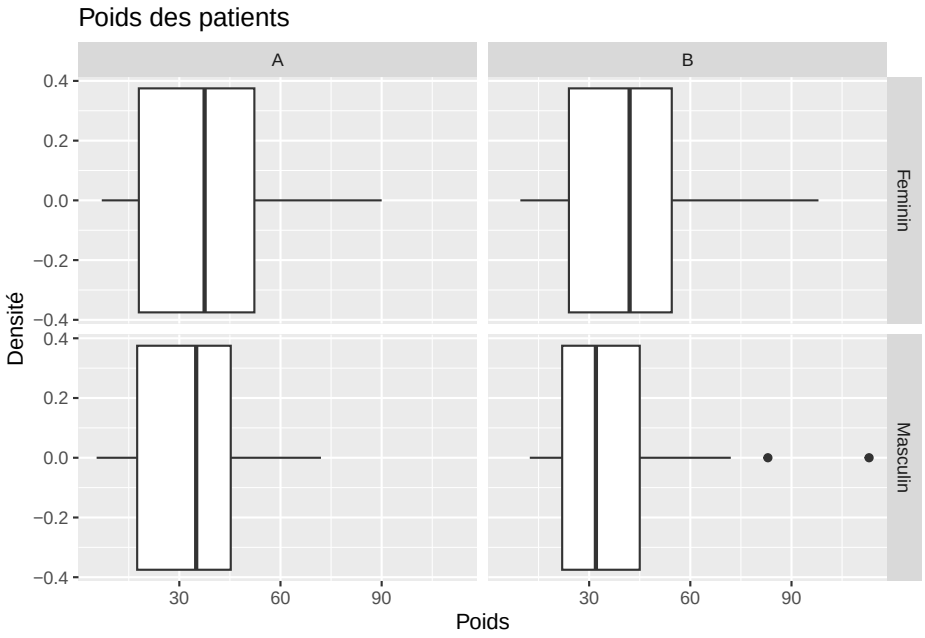
```
ggplot(patient,aes(x=poids,fill=sexe))+geom_boxplot()+
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Densité") +
  facet_grid(Hopital ~ .)
```




```
ggplot(patient,aes(x=sexe,y=poids))+geom_boxplot()+
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Densité") +
  facet_grid(Hopital ~ .)
```

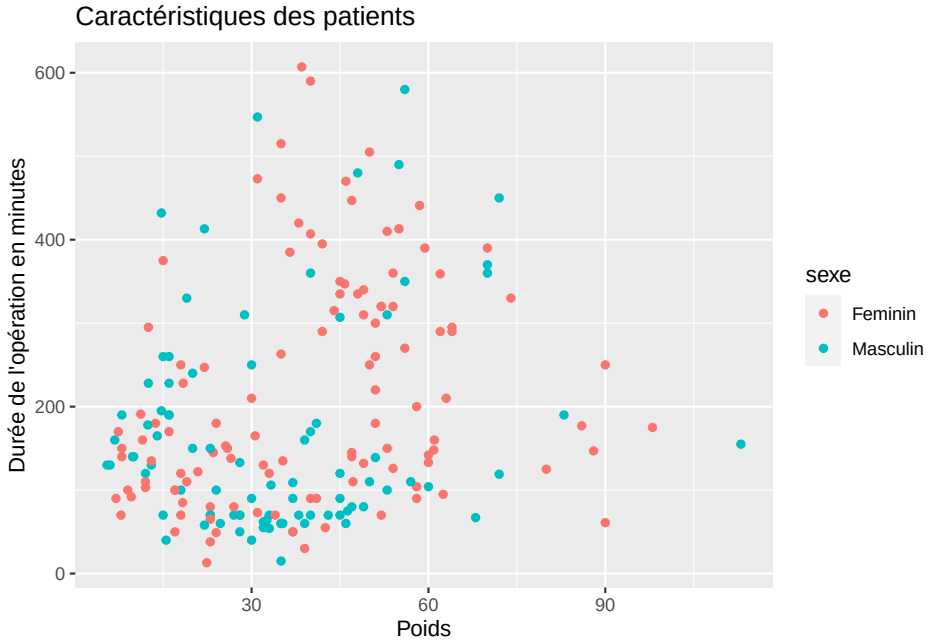


```
ggplot(patient,aes(poids))+geom_boxplot()+
  ggtitle("Poids des patients") +
  xlab("Poids") +
  ylab("Densité") +
  facet_grid(sexe ~ Hopital)
```



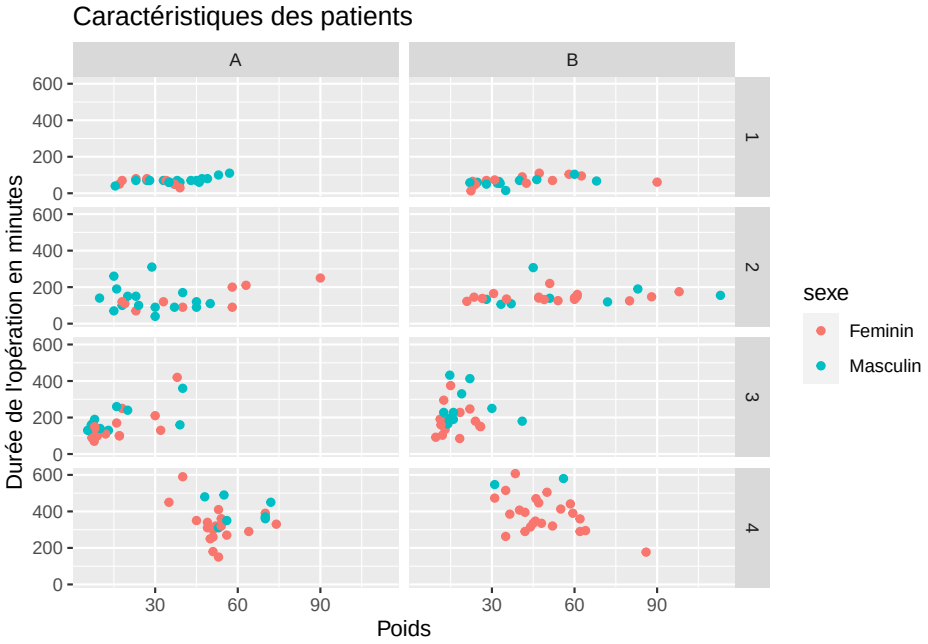
D'où des graphiques en **scatterplot** comme :

```
ggplot(patient,aes(x=poids,y=dureeopmin))+geom_point(aes(col=sexe))+
  ggtitle("Caractéristiques des patients") +
  xlab("Poids") +
  ylab("Durée de l'opération en minutes")
```



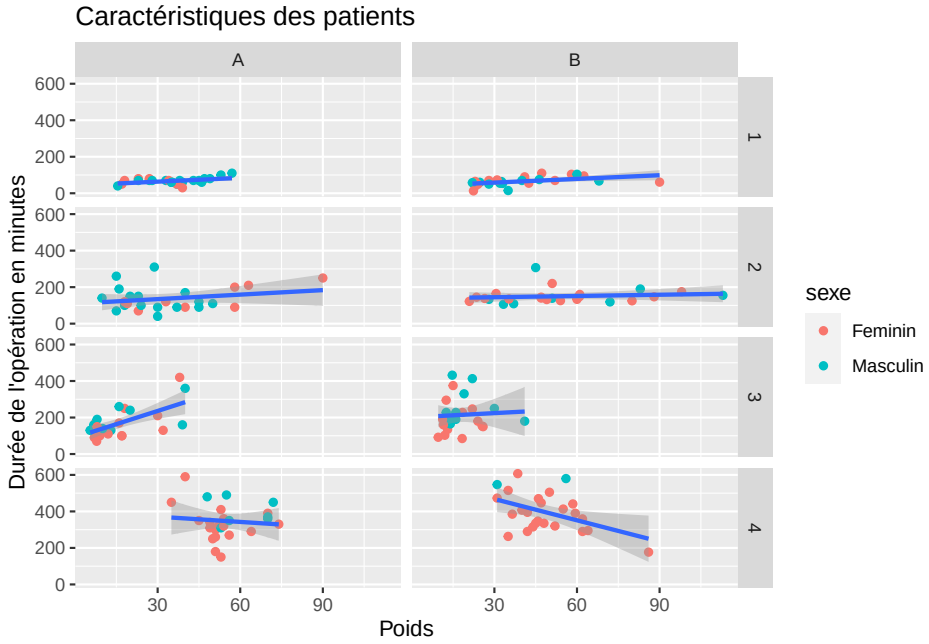
ou en rajoutant plein de trucs :

```
ggplot(patient,aes(x=poids,y=dureeopmin))+geom_point(aes(col=sexe))+
  ggtitle("Caractéristiques des patients") +
  xlab("Poids") +
  ylab("Durée de l'opération en minutes") +
  facet_grid(CIM2 ~ Hopital)
```



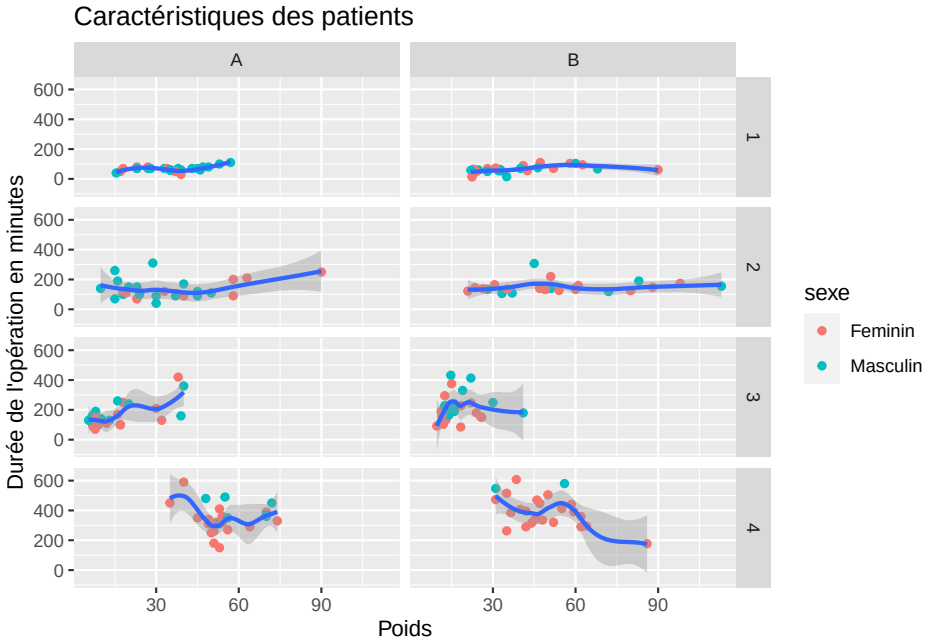
Pour rajouter une droite de régression :

```
ggplot(patient,aes(x=poids,y=dureeopmin))+geom_point(aes(col=sexe))+
  geom_smooth(method="lm") +
  ggtitle("Caractéristiques des patients") +
  xlab("Poids") +
  ylab("Durée de l'opération en minutes") +
  facet_grid(CIM2 ~ Hopital)
```



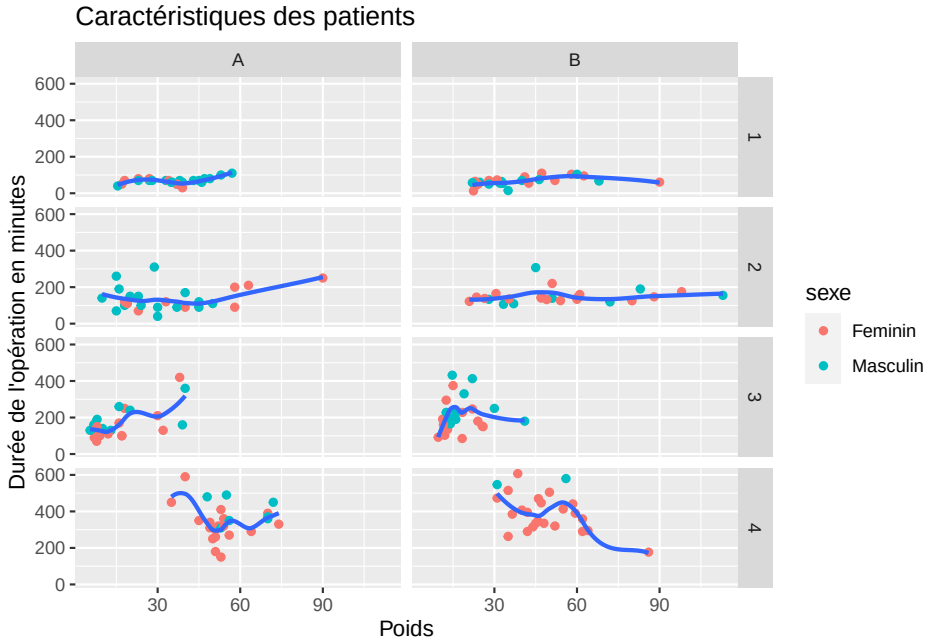
Des courbes de tendances :

```
ggplot(patient,aes(x=poids,y=dureeopmin))+geom_point(aes(col=sexe))+
  geom_smooth() +
  ggtitle("Caractéristiques des patients") +
  xlab("Poids") +
  ylab("Durée de l'opération en minutes") +
  facet_grid(CIM2 ~ Hopital)
```



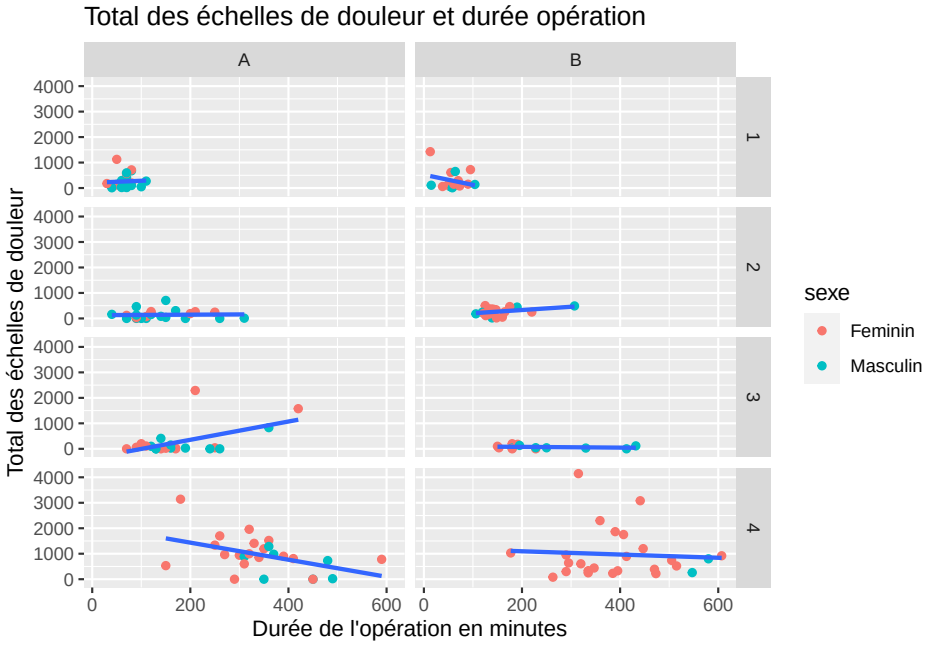
Sans l'intervalle de confiance :

```
ggplot(patient,aes(x=poids,y=dureeopmin))+geom_point(aes(col=sexe))+
  geom_smooth(se=FALSE) +
  ggtitle("Caractéristiques des patients") +
  xlab("Poids") +
  ylab("Durée de l'opération en minutes") +
  facet_grid(CIM2 ~ Hopital)
```



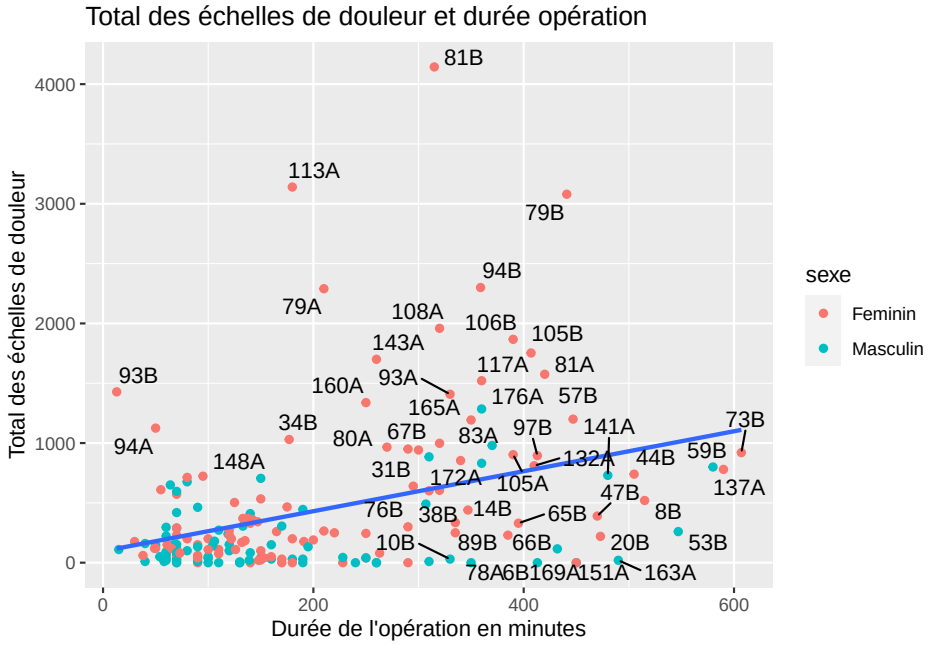
Avec l'intervalle de confiance et la droite de régression :

```
ggplot(patient,aes(x=dureeopmin,y=totalechelle))+geom_point(aes(col=sexe))+
  geom_smooth(method="lm",se=FALSE) +
  ggtitle("Total des échelles de douleur et durée opération") +
  xlab("Durée de l'opération en minutes") +
  ylab("Total des échelles de douleur") +
  facet_grid(CIM2 ~ Hopital)
```



Pour ajouter des étiquettes, il existe la librairie **ggrepel** qui permet de faire en sorte que la superposition des étiquettes soit minimale :

```
ggplot(patient,aes(x=dureeopmin,y=totalechelle,label=UID))+geom_point(aes(col=sexe))+
  geom_smooth(method="lm",se=FALSE) +
  geom_text_repel() +
  ggtitle("Total des échelles de douleur et durée opération") +
  xlab("Durée de l'opération en minutes") +
  ylab("Total des échelles de douleur")
```

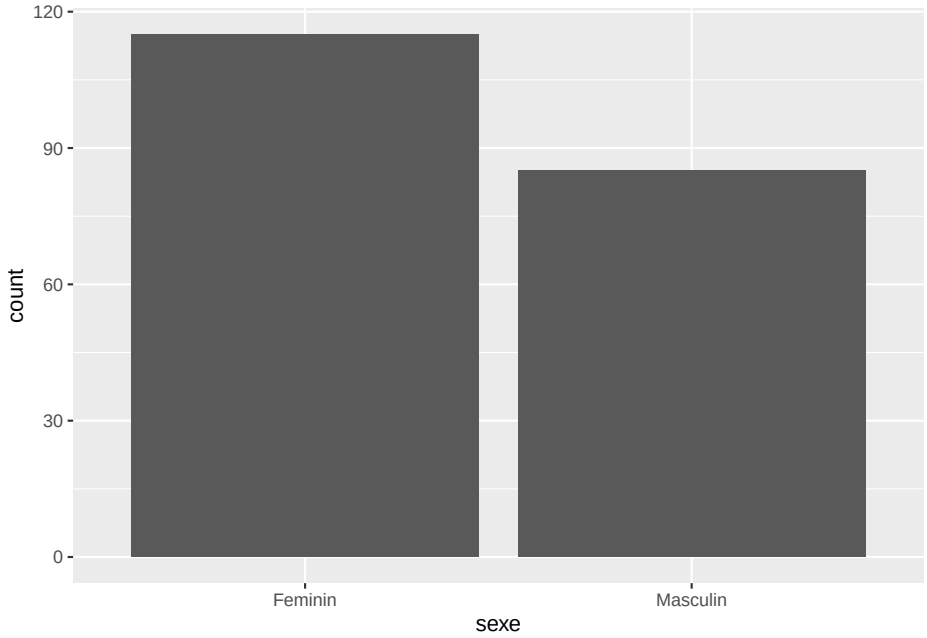


Evidemment toutes les étiquettes ne sont pas dessinés car il y a trop d'individus mais cela permet de repérer les individus atypiques.

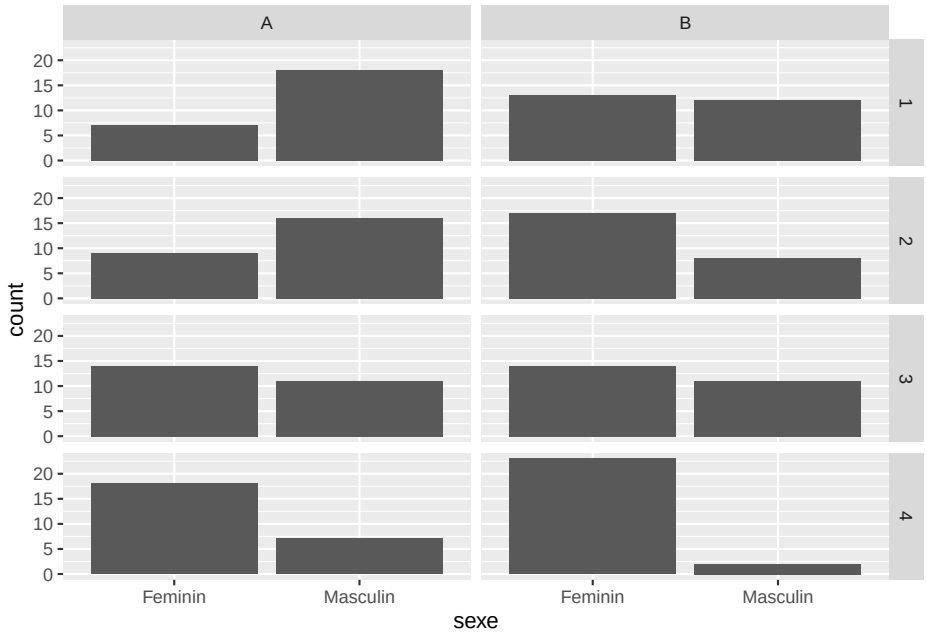
6.1.5 Tableaux de contingences

Pour les tableaux de fréquences, on peut faire très simple :

```
ggplot(patient,aes(sexe))+geom_bar()
```




```
ggplot(patient,aes(sexe))+geom_bar()+
  facet_grid(CIM2~Hopital)
```



Là où **ggplot2** commence à devenir compliqué, c'est que **geom_bar** ne va pas marcher car il faut lui fournir la **data.frame** avec les statistiques **en ligne**.

Soit :

```
##       sexe CIM2 value
## 1  Feminin    1    20
## 2 Masculin    1    30
## 3  Feminin    2    26
## 4 Masculin    2    24
## 5  Feminin    3    28
## 6 Masculin    3    22
## 7  Feminin    4    41
## 8 Masculin    4     9
```

Pour faire ce tableau, il faut faire appel au **package reshape2**.

```
require(reshape2)
tableau <- table(patient$sexe,patient$CIM2)
tableau
```

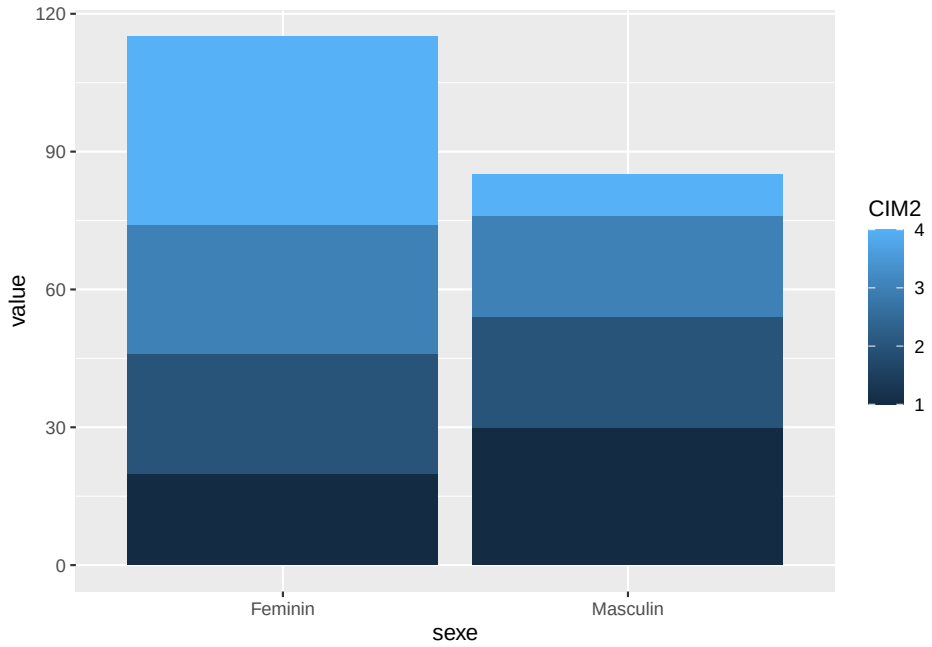
```
##
##             1  2  3  4
##   Feminin  20 26 28 41
##   Masculin 30 24 22  9
```

De ce tableau on passe au long en une commande :

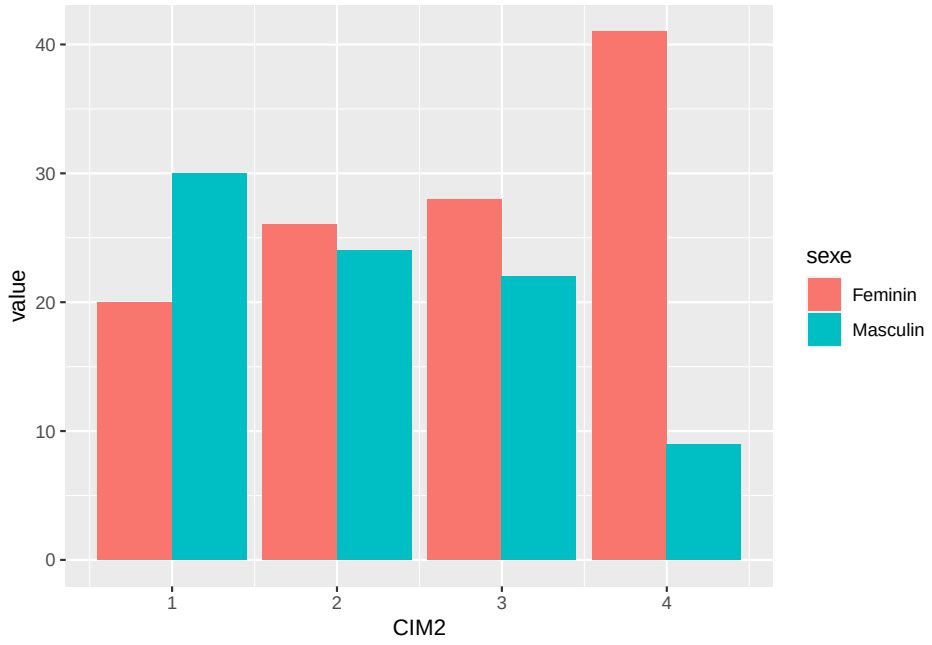
```
long <- melt(tableau,varnames = c("sexe","CIM2"),value.name = "value")
```

Il faut spécifier le nom à donner aux deux variables et spécifier le résultat du croisement des deux variables qui est le nombre d'observations c'est-à-dire le contenu de chaque cellule de **tableau**.

```
ggplot(long,aes(x=sexe,y=value,fill=CIM2))+geom_bar(position = "stack",stat="identity")
```

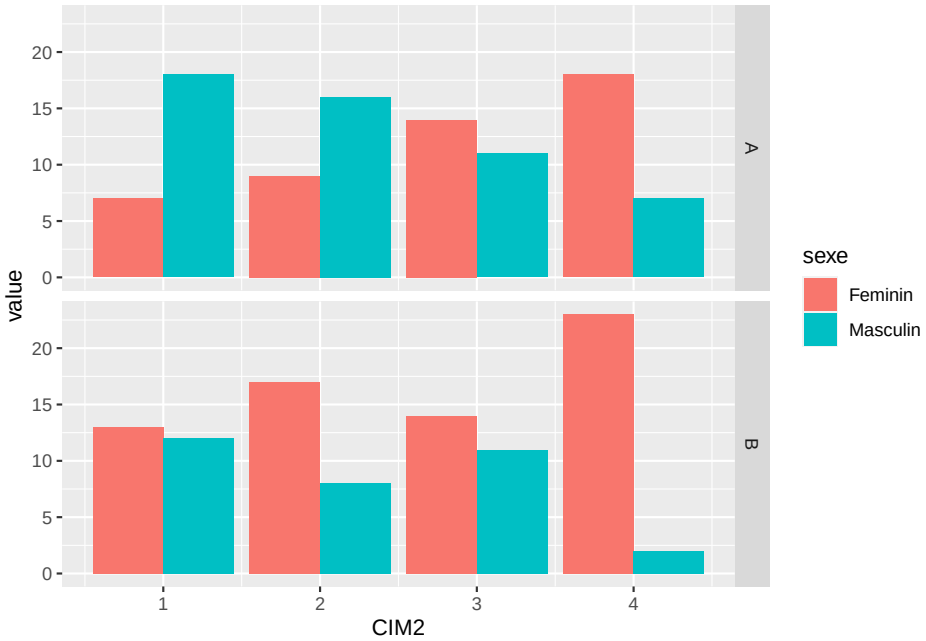



```
ggplot(long,aes(x=CIM2,y=value,fill=sexe))+geom_bar(position = "dodge",stat="identity")
```



D'où le graphique :

```
tableau <- table(patient$Hopital,patient$sexe,patient$CIM2)
long <- melt(tableau,varnames = c("Hopital","sexe","CIM2"),value.name = "value")
ggplot(long,aes(x=CIM2,y=value,fill=sexe))+
  geom_bar(position = "dodge",stat="identity")+
  facet_grid(Hopital ~ . )
```



Pour sauvegarder un graphique **ggplot2**, la syntaxe est différente et surtout on l'appelle une fois que le graphique est terminé, c'est-à-dire en dernier :

Et ainsi de suite...

6.2 Liens

Il y a de nombreuses galleries sur le web avec toutes les possiblités offertes par les graphiques de base comme les graphiques avec **ggplot2**.

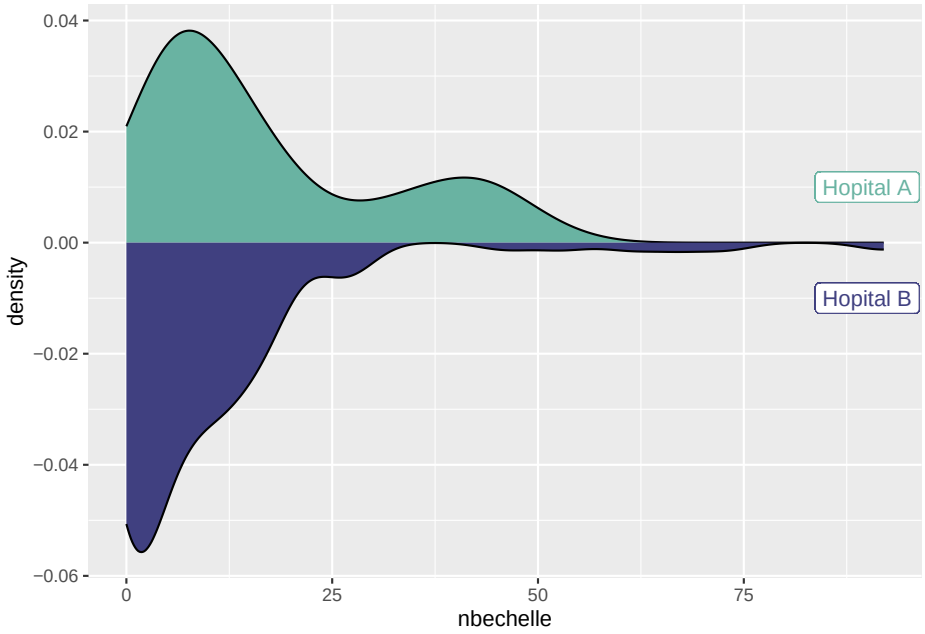
- r-graph-gallery
- r-chart
- ...

On peut en parcourir ensemble...

```
tableau <- dcast(patient, UID + sexe ~ Hopital, value.var = "nbechelle")
tableau$poids <- ifelse(!is.na(tableau$A),tableau$A,tableau$B)

ggplot(tableau, aes(x=nbechelle) ) +
```

```
  geom_density( aes(x = A, y = ..density..), fill="#69b3a2" ) +
  geom_label(aes(x=90, y=0.01, label="Hopital A"), color="#69b3a2") +
  geom_density( aes(x = B, y = -..density..), fill= "#404080") +
  geom_label(aes(x=90,y=-0.01, label="Hopital B"), color="#404080")
```



Chapter 7

Manipulation avancée (mettre en forme vos données)

Le but de cette partie qui ne sera pas très longue sur le papier mais beaucoup plus à l'apprentissage est de vous montrer comment automatiser les choses.

Pour prendre un exemple qui nous est familier, transformer en facteur ou en variable ordinale des items de plusieurs échelles. Sous SPSS ou sous R, c'est pénible si vous n'utilisez pas les macros pour SPSS ou la programmation sous R.

On peut automatiser beaucoup de choses sous R. Il y a des paquets sous R qui permettent d'analyser des modèles structuraux ou faire des pages web interactives. Sans aller jusque là, on peut se rendre la vie plus facile.

7.1 Transformation de plusieurs variables

On va charger le fichier de données en ligne :

```
file.name <- "https://personality-project.org/r/psych/HowTo/scoring.tutorial/small.msq.txt"
msq <- read_table(file.name)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   `"active"` = col_double(),
##   `"alert"` = col_double(),
##   `"aroused"` = col_double(),
```

```
##   `"sleepy"` = col_double(),
##   `"tired"` = col_double(),
##   `"drowsy"` = col_double(),
##   `"anxious"` = col_double(),
##   `"jittery"` = col_double(),
##   `"nervous"` = col_double(),
##   `"calm"` = col_double(),
##   `"relaxed"` = col_double(),
##   `"at.ease"` = col_double(),
##   `"gender"` = col_double(),
##   `"drug"` = col_double()
## )
```

Oui au passage on peut lire des fichiers de données directement en ligne.

Dans ce cas, on a utilisé la commande **read_table** du **package readr**, car le séparateur de champs est l'espace.

```
str(msq)
```

```
## spc_tbl_ [200 x 14] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ "active" : num [1:200] 2 1 1 1 1 3 1 0 3 1 ...
##  $ "alert"  : num [1:200] 2 1 1 2 2 3 1 1 3 1 ...
##  $ "aroused": num [1:200] 0 0 1 0 1 2 0 0 3 0 ...
##  $ "sleepy" : num [1:200] 0 2 3 1 1 0 2 1 0 1 ...
##  $ "tired"  : num [1:200] 1 2 3 1 1 0 2 1 0 1 ...
##  $ "drowsy" : num [1:200] 1 2 2 0 0 0 3 1 0 1 ...
##  $ "anxious": num [1:200] 1 1 2 0 0 1 0 0 3 0 ...
##  $ "jittery": num [1:200] 1 0 0 0 1 0 0 0 3 0 ...
##  $ "nervous": num [1:200] 0 1 1 0 0 0 0 0 1 0 ...
##  $ "calm"   : num [1:200] 2 2 2 2 3 2 1 3 0 1 ...
##  $ "relaxed": num [1:200] 2 2 2 2 2 3 2 2 0 2 ...
##  $ "at.ease": num [1:200] 2 2 2 2 3 3 1 2 0 2 ...
##  $ "gender" : num [1:200] 2 2 1 2 1 1 2 2 1 1 ...
##  $ "drug"   : num [1:200] 2 1 1 2 1 2 2 1 2 1 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   `"active"` = col_double(),
##   ..   `"alert"` = col_double(),
##   ..   `"aroused"` = col_double(),
##   ..   `"sleepy"` = col_double(),
##   ..   `"tired"` = col_double(),
##   ..   `"drowsy"` = col_double(),
##   ..   `"anxious"` = col_double(),
##   ..   `"jittery"` = col_double(),
##   ..   `"nervous"` = col_double(),
##   ..   `"calm"` = col_double(),
```

```
##   ..   `"relaxed"` = col_double(),
##   ..   `"at.ease"` = col_double(),
##   ..   `"gender"` = col_double(),
##   ..   `"drug"` = col_double()
##   .. )
```

On voit que les guillemets ont été importés et parasite la ligne des noms de colonnes. Pour ça on utilise **gsub**, une fonction qui remplace les caractéères dans le premier argument par les caractères dans le deuxième. Le troisième argument est **vecteur** dans lequel on veut remplacer le texte.

Ici on va remplacer un guillemet double, partout, par aucun caractère. Et on applique ça aux noms de variables de msq, ce qui nous donne :

```
colnames(msq) <- gsub('"','',colnames(msq),fixed=T)
msq
```

```
## # A tibble: 200 x 14
##    active alert aroused sleepy tired drowsy anxious jittery nervous  calm relaxed at.ease gend
##     <dbl> <dbl>   <dbl>  <dbl> <dbl>  <dbl>   <dbl>   <dbl>   <dbl> <dbl>   <dbl>   <dbl>  <db
##  1      2     2       0      0     1      1       1       1       0     2       2       2
##  2      1     1       0      2     2      2       1       0       1     2       2       2
##  3      1     1       1      3     3      2       2       0       1     2       2       2
##  4      1     2       0      1     1      0       0       0       0     2       2       2
##  5      1     2       1      1     1      0       0       1       0     3       2       3
##  6      3     3       2      0     0      0       1       0       0     2       3       3
##  7      1     1       0      2     2      3       0       0       0     1       2       1
##  8      0     1       0      1     1      1       0       0       0     3       2       2
##  9      3     3       3      0     0      0       3       3       1     0       0       0
## 10      1     1       0      1     1      1       0       0       0     1       2       2
## # i 190 more rows
```

Les commentaires de read.table indique que les colonnes sont toutes des nombres réels de double précision. C'est très bien pour les analyses psychométriques primaires avec **psych** mais pas avec lavaan qui réclame des facteurs.

On va utiliser la machine à automatiser **mutate_at** qui permet d'appliquer une transformation sur une série de variable.

Dans le premier argument, on mets **vars()** et à l'intérieur quelque chose pour définir une ou des variables comme avec un **select**.

Le deuxième argument est la fonction à appliquer et après les arguments optionnels.

```
msq_fact <- msq
msq_fact <- msq_fact %>% mutate(across(active:at.ease,~ factor(.x,ordered=T)))
```

Pour être vraiment propre, on spécifierait les niveaux :

```
msq_fact <- msq_fact %>% mutate(across(active:at.ease, ~ factor(.x,levels=c(0,1,2,3),or
```

```
str(msq_fact)
```

```
## tibble [200 x 14] (S3: tbl_df/tbl/data.frame)
##  $ active : Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 3 2 2 2 2 4 2 1 4 2 ...
##  $ alert  : Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 3 2 2 3 3 4 2 2 4 2 ...
##  $ aroused: Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 1 1 2 1 2 3 1 1 4 1 ...
##  $ sleepy : Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 1 3 4 2 2 1 3 2 1 2 ...
##  $ tired  : Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 2 3 4 2 2 1 3 2 1 2 ...
##  $ drowsy : Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 2 3 3 1 1 1 4 2 1 2 ...
##  $ anxious: Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 2 2 3 1 1 2 1 1 4 1 ...
##  $ jittery: Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 2 1 1 1 2 1 1 1 4 1 ...
##  $ nervous: Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 1 2 2 1 1 1 1 1 2 1 ...
##  $ calm   : Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 3 3 3 3 4 3 2 4 1 2 ...
##  $ relaxed: Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 3 3 3 3 3 4 3 3 1 3 ...
##  $ at.ease: Ord.factor w/ 4 levels "0"<"1"<"2"<"3": 3 3 3 3 4 4 2 3 1 3 ...
##  $ gender : num [1:200] 2 2 1 2 1 1 2 2 1 1 ...
##  $ drug   : num [1:200] 2 1 1 2 1 2 2 1 2 1 ...
```

Il y a des variantes à **mutate_at** comme **mutate_if**.

Par exemple, pour centrer/réduire les variables numériques :

```
iris <- iris %>% mutate(across(where(is.numeric),scale))
```

Si la variable est numérique alors R va centrer/réduire la variable. Donc pas de problème avec **Species** qui est un facteur :

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num [1:150, 1] -0.898 -1.139 -1.381 -1.501 -1.018 ...
##   ..- attr(*, "scaled:center")= num 5.84
##   ..- attr(*, "scaled:scale")= num 0.828
##  $ Sepal.Width : num [1:150, 1] 1.0156 -0.1315 0.3273 0.0979 1.245 ...
##   ..- attr(*, "scaled:center")= num 3.06
##   ..- attr(*, "scaled:scale")= num 0.436
##  $ Petal.Length: num [1:150, 1] -1.34 -1.34 -1.39 -1.28 -1.34 ...
##   ..- attr(*, "scaled:center")= num 3.76
##   ..- attr(*, "scaled:scale")= num 1.77
##  $ Petal.Width : num [1:150, 1] -1.31 -1.31 -1.31 -1.31 -1.31 ...
##   ..- attr(*, "scaled:center")= num 1.2
##   ..- attr(*, "scaled:scale")= num 0.762
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 .
```

Soit :

```
iris %>% tbl_summary(statistic = list(
  all_continuous() ~ "{mean} ({sd})"),type = c(Sepal.Length:Petal.Width) ~ "continuous")
```

Caractéristique	**N = 150**
Sepal.Length	0,00 (1,00)
Sepal.Width	0,00 (1,00)
Petal.Length	0,00 (1,00)
Petal.Width	0,00 (1,00)
Species	
setosa	50 (33%)
versicolor	50 (33%)
virginica	50 (33%)

7.2 Opérateurs et case_when

On appelle opérateur des mots-clefs généralement symbolique comme les +,/,==, etc.

Par exemple de très utile, il y a l'opérateur **%in%**.

Il prends un vecteur à gauche et un vecteur à droite.

Dans le cas simple, avec un élement dans un vecteur à gauche, il renvoie vrai si l'élement à gauche est présent dans le vecteur de droite :

```
3 %in% 1:5
```

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

```
-1 %in% 1:5
```

```
## [1] FALSE
```

Quand il y a plusieurs élements à gauche, l'opérateur renvoie une réponse pour chaque élement à gauche :

```
c(1,3) %in% 1:5
```

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

```
c(-1,3) %in% 1:5
```

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

Si on veut transformer msq en items dichotomiques, c'est-à-dire coder 0 ou 1 en 0 et 2 et 3 en 1. Alors ça devient facile.

En fait il y a deux façons de l'écrire. la première est **old school**.

On utilise la fonction **ifelse** pour renvoyer 0 ou 1 selon la réponse :

Pour comprendre **ifelse** un exemple :

```
ifelse(c(TRUE,FALSE,TRUE,FALSE,FALSE),1,0)
```

```
## [1] 1 0 1 0 0
```

ifelse renvoie 1 quand c'est vrai et 0 quand c'est faux. Ce qui nous donne associé à notre nouvel opérateur:

```
active <- ifelse(msq$active %in% c(2,3),1,0)
table(active)
```

```
## active
##   0   1
## 156  44
```

C'est un peu brutal car on ne précise pas explicitement ce que va prendre les valeurs 0 et 1.

En plus élégant, il y a une variante à privilégier avec le **tidyverse**:

```
msq2 <- msq %>% mutate(active=case_when(
      active %in% c(0,1) ~ 0,
      active %in% c(2,3) ~ 1,
      .default = NA
))
table(msq2$active)
```

```
##
##   0   1
## 156  44
```

Mais là, on ne fait qu'un variable à la fois, il faudrait appliquer une fonction pour avoir le résultat sur toutes les variables.

Une fonction se définit par un corps de fonction, des arguments et un nom.

```
ma.fonction <- function(x) {
  return(x)
}
```

Ce qui donne :

```
ma.fonction(1)
```

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

Ce qui est en dernière ligne ou bien (c'est mieux) ce qui est indiqué entre parenthèses pour la fonction **return** est renvoyée.

Donc notre fonction devient :

```
dichotomiser <- function(x) {
  resultat <- case_when(
      x %in% c(0,1) ~ 0,
```

```
      x %in% c(2,3) ~ 1,
      .default = NA
  )
  return(resultat)
}
```

```
table(dichotomiser(msq$active))
```

```
##
##   0   1
## 156  44
```

Pour l'appliquer, il faut se rappeler de **mutate_at**:

```
msq2 <- msq %>% mutate(across(active:at.ease,dichotomiser))
```

Ce qui donne bien :

```
tbl_summary(msq2)
```

Caractéristique	**N = 200**
active	44 (22%)
alert	49 (25%)
Manquant	1
aroused	24 (12%)
sleepy	101 (51%)
Manquant	2
tired	123 (62%)
drowsy	98 (49%)
anxious	26 (13%)
jittery	31 (16%)
nervous	20 (10%)
calm	106 (53%)
relaxed	119 (60%)
Manquant	1
at.ease	91 (46%)
Manquant	2
gender	
1	62 (46%)
2	74 (54%)
Manquant	64
drug	
1	68 (50%)
2	68 (50%)
Manquant	64

7.3 Réutilisation de statistiques

Contrairement aux autres logiciels, les résultats statistiques sont la plupart du temps réutilisable par l'utilisateur.

Par exemple, sous Jamovi, il faut créer des variables **à la main** pour créer des variables avec les quantiles.

Pour créer ces variables sous R, c'est beaucoup plus simple, il n'y a pas besoin de faire de tests (if):

```
quantile(iris$Sepal.Length)
```

```
##          0%         25%         50%         75%        100%
## -1.86378030 -0.89767388 -0.05233076  0.67224905  2.48369858
```

La fonction **quantile** nous renvoie les quartiles sous la forme d'un vecteur qu'il suffit de combiner avec une autre fonction **cut**. Cette dernière crée des variables facteurs à partir d'une variable quantitative et de points de césure.

Les points de césure sont fournis par la fonction **quantile**, par conséquent :

```
table(cut(iris$Sepal.Length, breaks=quantile(iris$Sepal.Length)))
```

```
##
##   (-1.86,-0.898] (-0.898,-0.0523]  (-0.0523,0.672]     (0.672,2.48]
##               40               39               35               35
```

Pour généraliser :

```
quartiles <- function(x) {
  cut(x, breaks=quantile(x))
}
iris2 <- iris %>% mutate(across(where(is.numeric),quartiles))
```

```
tbl_summary(iris2)
```

Caractéristique	**N = 150**
Sepal.Length	
(-1.86,-0.898]	40 (27%)
(-0.898,-0.0523]	39 (26%)
(-0.0523,0.672]	35 (23%)
(0.672,2.48]	35 (23%)
Manquant	1
Sepal.Width	
(-2.43,-0.59]	46 (31%)
(-0.59,-0.132]	36 (24%)
(-0.132,0.557]	30 (20%)
(0.557,3.08]	37 (25%)
Manquant	1
Petal.Length	
(-1.56,-1.22]	43 (29%)
(-1.22,0.335]	31 (21%)
(0.335,0.76]	41 (28%)
(0.76,1.78]	34 (23%)
Manquant	1
Petal.Width	
(-1.44,-1.18]	36 (25%)
(-1.18,0.132]	37 (26%)
(0.132,0.788]	38 (26%)
(0.788,1.71]	34 (23%)
Manquant	5
Species	
setosa	50 (33%)
versicolor	50 (33%)
virginica	50 (33%)

7.4 Chargement

```
## spc_tbl_ [200 x 17] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ UID         : chr [1:200] "100A" "100B" "101A" "101B" ...
##  $ Hopital     : chr [1:200] "A" "B" "A" "B" ...
##  $ sexe        : chr [1:200] "Masculin" "Feminin" "Masculin" "Masculin" ...
##  $ poids       : num [1:200] 45 80 37 28 18 13.7 16 35 40 32 ...
##  $ vitaux      : num [1:200] 22 8 17 28 6 21 9 10 17 11 ...
##  $ CIM2        : num [1:200] 2 2 1 2 3 3 3 1 2 1 ...
##  $ age         : num [1:200] 13.54 15.72 10.94 11.14 4.05 ...
##  $ dureeopmin  : num [1:200] 90 125 50 133 250 180 260 15 90 55 ...
##  $ postopj     : num [1:200] 2.208 1.014 4.667 2.24 0.938 ...
##  $ scoliose    : chr [1:200] NA NA NA NA ...
##  $ drepano     : chr [1:200] NA "rien" NA "drepanocytose" ...
```

```
##  $ ACP         : num [1:200] 0 0 0 0 0 0 0 0 0 0 ...
##  $ peridurale  : num [1:200] 0 0 0 0 0 1 0 0 0 0 ...
##  $ periACP     : num [1:200] 0 0 0 0 0 2 0 0 0 0 ...
##  $ nbttt       : num [1:200] 3 4 4 5 3 10 3 2 3 7 ...
##  $ totalechelle: num [1:200] 463 503 146 306 40 200 0 110 0 NA ...
##  $ nbechelle   : num [1:200] 16 17 12 17 6 27 4 7 6 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   UID = col_character(),
##   ..   Hopital = col_character(),
##   ..   sexe = col_character(),
##   ..   poids = col_double(),
##   ..   vitaux = col_double(),
##   ..   CIM2 = col_double(),
##   ..   age = col_double(),
##   ..   dureeopmin = col_double(),
##   ..   postopj = col_double(),
##   ..   scoliose = col_character(),
##   ..   drepano = col_character(),
##   ..   ACP = col_double(),
##   ..   peridurale = col_double(),
##   ..   periACP = col_double(),
##   ..   nbttt = col_double(),
##   ..   totalechelle = col_double(),
##   ..   nbechelle = col_double()
##   .. )
##  - attr(*, "problems")=<externalptr>
```

Ouch… Remettre en factor, les variables qui doivent l'être. Lequelles ?

Scoliose ça sera plus propre avec les différents champs et "non" en l'absence de scoliose…

```
patient$scoliose[is.na(patient$scoliose)] <- "Non"
table(patient$scoliose)
```

```
##
##  ante autre   Non  post
##     3     2   175    20
```

Ca serait bien de faire **scoliose2**

Et pareil avec les drépanocytose donc avec **drepano** et **drepano2**:

Calculer la variable **moyechelle** qui fait la moyenne des échelles de douleur :

```
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -8.31250 -1.94531 -0.79167 -0.08893  0.46875 40.60417
```

Juste pour voir **case** : faire des catégories quand nbttt < 0, entre 0 et 10 et est supérieur à 10

```
##
## Entre 0 et -5      Inf à -5       Sup à 0
##           127             7            66
```