Statistiques : TP1

Université de Tours

`Département informatique de Blois`

Statistiques descriptives et visualisation d'information

*
* *

Contents

1 Démarrer avec R

R est un langage conçu pour l'analyse statistique de données. Il est disponible gratuitement et fonctionne sur les différents systèmes d'exploitation. Vous pouvez le télécharger à l'adresse :

⇒ https://cran.r-project.org/

Téléchargez ensuite l'environnement graphique RStudio Desktop à l'adresse suivante :

⇒ https://www.rstudio.com/products/rstudio/download/

2 Pour débuter

La finalité de ce TP est d'apprendre les commandes R indispensables pour la manipulation des données et la production d'outils de visualisation.

N'oubliez pas, la meilleure façon d'avoir de l'aide sur une commande en R, c'est en tapant :

```
help(nom_de_la_commande)
```

2.1 Les vecteurs

Pour créer un vecteur, on utilise la commande `c()`. Attention, toutes les données doivent être du même type.

```
X = c(0, 4, 10, 2, 5, 12, 16, 13, 6, 8, 5, 10, 11)
```

La fonction `c()` permet aussi de concaténer des vecteurs :

```
Y = c(20, 18, 17, 18, 19)
Z = c(X, Y)

Z
```

```
##  [1]  0  4 10  2  5 12 16 13  6  8  5 10 11 20 18 17 18 19
```

Dans la suite, on note $|X|$ la taille du vecteur X. Quelques commandes supplémentaires:

- Accéder à un élément à l'élément $i \in [\![1, |X|]\!]$ de X [1] : `X[i]`

```
X[2]
```

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

- Retirer l'élément i de X : `X[-i]`

```
X[-3]
```

```
##  [1]  0  4  2  5 12 16 13  6  8  5 10 11
```

- Accéder aux éléments entre i et j (avec $i > j$) de X : `X[i:j]`

```
X[2:5]
```

```
## [1]  4 10  2  5
```

- Filtre sur X par une condition logique φ : `X[phi]` > Les opérateurs logiques ET `&`, OU `|`, NON `!`

```
X[X < 10]
```

```
## [1] 0 4 2 5 6 8 5
```

- Taille $|X|$: `length(X)`

```
length(X)
```

```
## [1] 13
```

- Création du vecteur $V = (i, ..., n)$: `seq(i,n)`

[1] Attention, l'indiciçage commence à 1 et non 0 !

```
seq(3, 14)
```

```
##  [1]  3  4  5  6  7  8  9 10 11 12 13 14
```

- Création du vecteur $V = (i, i, ..., i)$ de taille n : `rep(i,n)`

```
rep(1, 5)
```

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

- On peut aussi faire les opérations arithmétiques de base :
  - Addition de i à tous les élements de X : `X+i`
  - Multiplication par i à tous les élements de X : `X*i`
  - etc...

2.2 Les tableaux de données (ou data frame)

Un tableau de données est une matrice où chaque colonne correspond à un attribut (âge, taille, poids par exemple) et chaque ligne à un individu. Typiquement, un data frame correspond à une base de données dé-normalisée. Les data frames R sont très semblables à ceux que l'on peut manipuler en Python.

2.2.1 Importer un CVS

Dans la suite de ce TP nous allons nous servir du fichier `poke.csv` qui répertorie tous les Pokémons sur les 7 premières générations de la franchise.

Lorsque l'on est sur **RStudio**, l'importation se fait très simplement dans l'onglet : `file -> import dataset -> From CSV` [2].

1. Sélectionnez l'emplacement du fichier,
2. Le type de séparateur,
3. N'oubliez pas non plus de vérifier le bon format des données,
4. Le type de valeur manquante `NA`,
5. Importez-le.

On peut aussi taper la commande :

```
data = read.table("path", sep="type_sep", header={TRUE, FALSE})
```

On donne la description suivante des colonnes:

Colonne	Description	Value
`name`	Nom du pokémon (unique)	String
`generation`	Génération où le pokémon est apparu	{1,...,7}
`type1`	Type du pokémon	String
`type2`	Deuxième type (optionnel) du pokémon	String
`color`	Couleur principale du pokémon	String
`evolving_stade`	Stade d'évolution du pokémon: 0 (bébé) à 3 (adulte)	{0,1,2,3}
`hp`	Nombre de points de vie (PV) du pokémon	Double
`atk`	Nombre de points d'Attaque du pokémon	Double
`def`	Nombre de points de Défense du pokémon	Double
`sp_atk`	Nombre de points d'Attaque Spéciale du pokémon	Double

[2] On peut parfois trouver l'option `From txt` à la place.

Colonne	Description	Value
`sp_def`	Nombre de points de Défense Spéciale du pokémon	Double
`spd`	Nombre de points de Vitesse du pokémon	Double
`base_stats`	Somme des statistiques du pokémon	Double
`is_lengendary`	Booléen si le pokémon est légendaire	{0,1}
`capt_rate`	Taux de capture du pokémon. Plus le taux est faible, plus le pokémon est difficile à capturer	Int
`exp_growth`	Nombre de points total d'expérience du pokémon	Double
`egg_steps`	Nombre de pas avant éclosion d'un oeuf du pokémon	Double
`height`	Taille du pokémon (en m)	Double
`weight`	Poids du pokémon (en kg)	Double
`hapiness`	Base de bonheur du pokémon	Int
`classif`	Classification du pokémon	String
`percentage_male`	Pourcentage de mâles moyen pour 100 individus	Double
`against__Type_`	Efficacité du type `_Type_` sur le pokémon	Double

On précise également qu'un type de Pokémon peut appartenir à la liste suivante :

- bug
- dark
- dragon
- electric
- fairy
- fighting
- fire
- flying
- ghost
- grass
- ground
- ice
- normal
- poison
- psychic
- rock
- steel
- water

On compte donc autant de colonnes `against__Type_` que de types ci-dessus.

Les valeurs dans ces colonnes correspondent à l'efficacité du types `_Type_` sur le pokémon. Par exemple, pour le pokémon *Bulbasaur* (première ligne) la valeur 2 dans la colonne `against_fire` signifie que le Pokémon est doublement affecté par le type feu. À l'inverse, la valeur 0.5 dans la colonne `against_water` signifie que le pokémon est faiblement affecté par l'eau. Ces colonnes sont étroitement liées au type du pokémon considéré.

2.2.2 Premières manipulations

Les noms des colonnes sont disponibles via la fonction `names()` :

```
names(poke)
```

Soit T, un data frame. Il est possible d'accéder à des données précises de T par les commandes suivantes:

- Nombre de lignes de T : `nrow(T)`
- Nombre de colonnes de T : `ncol(T)`
- Accés aux données de la colonne de nom X : `T$X`
- Accés à la i-ème ligne de T : `T[i,]`
- Accés aux lignes entre i et k de T : `T[i:k,]`
- Accés à la j-ème colonne de T : `T[,j]`
- Accés aux colonnes entre j et k de T : `T[,j:k]`

- Accés à la colonne de nom X de T : `T[,"X"]`
- Accés aux colonnes de T dont les noms appartiennent à un vecteur V : `T[,V]`

```
poke[, c("name", "generation")]
```

- Accés à la donnée à la i-ème ligne, colonne j de T : `T[i,j]`
- Supprimer la ligne i de T : `T[-i,]`
- Supprimer la colonne j de T : `T[,-j]`
- Filtrage selon une condition logique φ de T : `subset(T,phi)`

```
subset(poke, type1 == "fire" | type2 == "fire")
```

Enfin, si vous voulez sauvegarder des résultats d'un data frame T, vous pouvez l'exporter au format `.csv` via la commande :

```
write.table(T, "mes_resultats.csv", sep = ";", row.names = {TRUE,FALSE})
```

2.2.3 Utilisation de DPLYR

La bibliothèque **dplyr** est un pilier de R et de la manipulation des dataframes. Elle est très utile pour l'utilisation et la mise en oeuvre des opérations d'algèbre relationnel entre dataframes (sélection, projection, agrégation et jointures).

```
library(dplyr)
```

```
##
## Attachement du package : 'dplyr'
## Les objets suivants sont masqués depuis 'package:stats':
##
##     filter, lag
## Les objets suivants sont masqués depuis 'package:base':
##
##     intersect, setdiff, setequal, union
```

2.2.3.1 Opérateurs de l'algèbre relationnelle

Nous allons la mettre en oeuvre pour établir quelques résultats statistiques sur notre jeu de données.

Commençons par la description des opérateurs classiques de l'algèbre relationnelle :

- $\sigma_{\varphi}(T)$: `filter(T, \varphi)` – Sélection par la condition φ.
- $\pi_{X}(T)$: `select(T,X)` – Projection sur l'ensemble de colonnes X.
- $\rho_{X \to Y}(T)$: `rename` – Renomage de la colonne X en Y
- $\delta(T)$: `distinct(T)` – Supression des duplicats du dataframe T.
- $\gamma_{X,f(Y)}(T)$: `group_by(T, X) %>% summarise(... = f(Y))` – Agrégation par la fonction f de la variable Y selon les modalités X.

Supposons que nous souhaitons obtenir *La liste de tous les pokémons de type Plante.*

En algèbre relationnelle, on écrirait : $\pi_{amount}(\sigma_{type1="grass" \vee type2="grass"}(Poke))$.

Ce qui peut s'écrire en R grâce à dplyr :

```
T_ <- filter(poke, type1 == "grass" | type2 == "grass")
T_ <- select(T_, name)

T_
```

```
## # A tibble: 98 x 1
##    name
##    <chr>
##  1 Bulbasaur
##  2 Ivysaur
##  3 Venusaur
##  4 Oddish
##  5 Gloom
##  6 Vileplume
##  7 Paras
##  8 Parasect
##  9 Bellsprout
## 10 Weepinbell
## # ... with 88 more rows
```

Cela oblige à passer par une variable intermédiaire `T_`, ce qui est peu lisible et fastidieux. On peut pour éviter cela utiliser l'opérateur pipe `%>%` qui permet de passer en argument de la fonction à droite de l'opérateur le résultat obtenu à gauche. On a ainsi :

```
filter(poke, type1 == "grass" | type2 == "grass") %>% select(name)
```

```
## # A tibble: 98 x 1
##    name
##    <chr>
##  1 Bulbasaur
##  2 Ivysaur
##  3 Venusaur
##  4 Oddish
##  5 Gloom
##  6 Vileplume
##  7 Paras
##  8 Parasect
##  9 Bellsprout
## 10 Weepinbell
## # ... with 88 more rows
```

Supposons à présent que je souhaite connaitre *Le nombre de HP maximal du pokemon pour chaque génération.*

En algèbre relationnelle, on écrirait : $\gamma_{generation,\max(hp)}(Poke)$.

```
group_by(poke, generation) %>%
    summarise(max_hp = max(hp))
```

```
## # A tibble: 7 x 2
##   generation max_hp
##        <dbl>  <dbl>
## 1          1    250
## 2          2    255
## 3          3    170
## 4          4    150
```

```
## 5          5    165
## 6          6    216
## 7          7    223
```

Exercices

Donner le code R des requêtes suivantes :

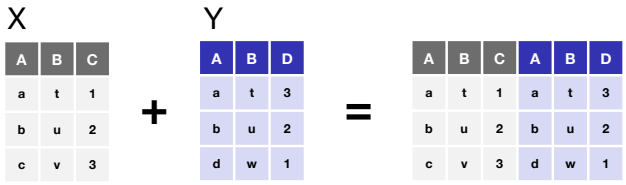
1. L'ensemble des pokémons de la génération 1.
2. L'ensemble des pokémons faible au feu mais pas à la glace.
3. Le nombre moyen de HP moyen.
4. Le nombre de pokemons pour chaque génération.
5. La base stats maximal des pokémons légendaires et non légendaires par génération.

2.2.3.2 Jointures de tables La combinaison de différents dataframes est une des clé de l'analyse de données. Nous allons voir ici les différentes opérations pour bien "coller" nos tables comme on le souhaite.

Bind columns

On accole les deux dataframes, en colonnes :

```
X <- data.frame(A=c('a','b','c'), B=c('t','u','v'),C=c(1,2,3))
Y <- data.frame(A=c('a','b','d'), B=c('t','u','w'),D=c(3,2,1))
```



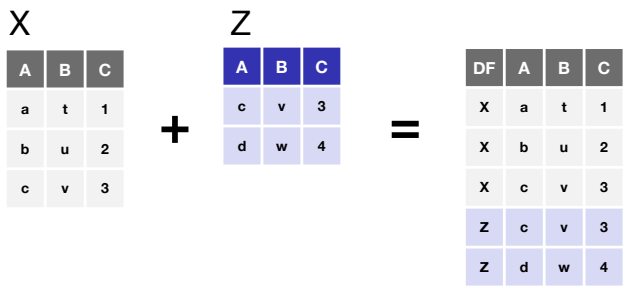

```
bind_cols(X,Y)
```

```
## New names:
## * A -> A...1
## * B -> B...2
## * A -> A...4
## * B -> B...5
```

```
##   A...1 B...2 C A...4 B...5 D
## 1     a     t 1     a     t 3
## 2     b     u 2     b     u 2
## 3     c     v 3     d     w 1
```

Bind rows

On accole les deux dataframes en ajoutant les lignes du second à la suite de l'autre. Attention, les différentes variables (et l'ordre de celles-ci) doivent correspondre.




```
Z <- data.frame(A=c('c', 'd'), B=c('v','w'),C=c(3,4))

bind_rows(X,Z)
```

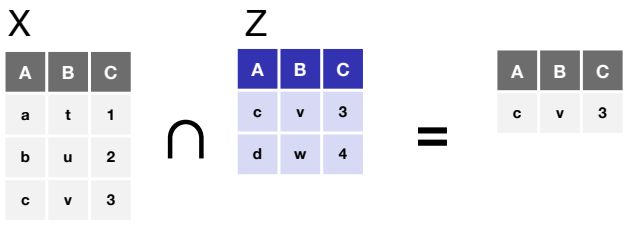
```
##   A B C
## 1 a t 1
## 2 b u 2
## 3 c v 3
## 4 c v 3
## 5 d w 4
```

L'ajout de l'option `.id="df_origin"` permet d'identifier le dataframe d'origine du tuple.

Opérations ensemblistes

Il est possible d'utiliser les opérations ensemblistes usuelles pour manipuler les tableaux :

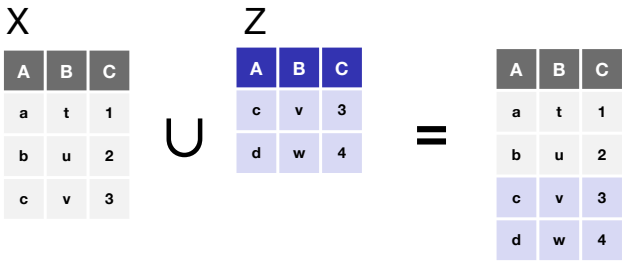
Intersection




```
intersect(X,Z)
```

```
##   A B C
## 1 c v 3
```

Union

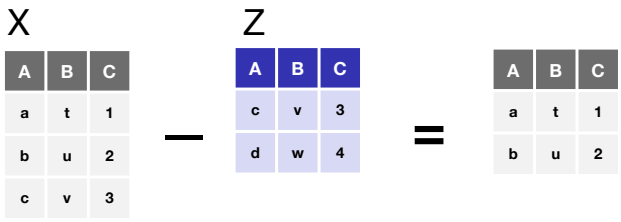



```
union(X,Z)
```

```
##   A B C
```

```
## 1 a t 1
## 2 b u 2
## 3 c v 3
## 4 d w 4
```

Différence




```
setdiff(X,Z)
```

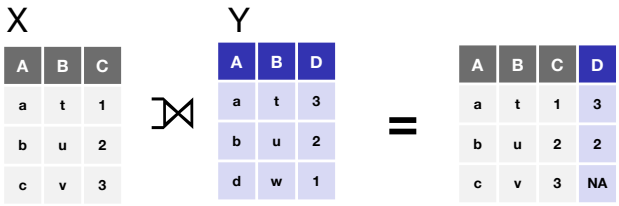
```
##   A B C
## 1 a t 1
## 2 b u 2
```

Les jointures

Les jointures permettent de reconstituer un unique tableau à partir de différentes tables et de leurs clés étrangères.

Left Join

La jointure à gauche permet d'adjoindre aux informations de la table de gauche les informations issues de la table de droite.




```
left_join(X,Y)
```

```
## Joining, by = c("A", "B")
```

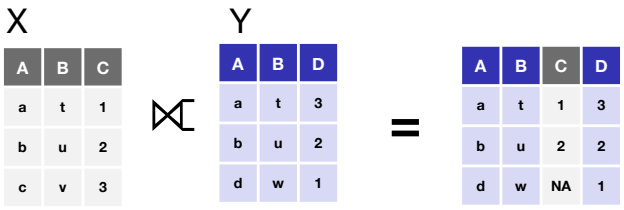
```
##   A B C  D
## 1 a t 1  3
## 2 b u 2  2
## 3 c v 3 NA
```

Ainsi, c'est le tableau de gauche qui définit les individus du tableau en sortie quand bien-même il n'y a pas d'information à leur sujet dans le tableau de droite: remarquez ainsi la donnée manquante dans la colonne D pour le 3ème individu.

Il arrive qu'une variable identifiant les individus n'ait pas le même nom dans les deux tableaux... Dans ce cas, on peut préciser (et c'est recommander) de faire correspondre à une variable du premier tableau, une autre variable du deuxième tableau, par exemple, de la manière suivante: `left_join(X,Y,by=c("A"="A", "C"="D"))`

Right Join

Ici, on adjoint aux informations de la table de droite les informations issues de la table de gauche.




```
right_join(X,Y)
```

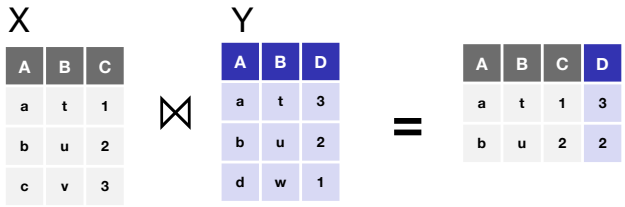
```
## Joining, by = c("A", "B")
```

```
##   A B  C D
## 1 a t  1 3
## 2 b u  2 2
## 3 d w NA 1
```

Ainsi, c'est le tableau de droite qui définit les individus du tableau en sortie.

Inner Join

C'est la jointure la plus commune, elle ne conserve que les individus communs aux deux jeux de données



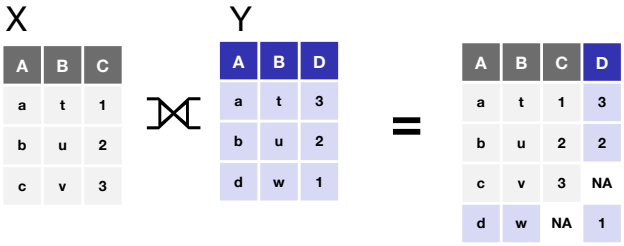

```
inner_join(X,Y)
```

```
## Joining, by = c("A", "B")
```

```
##   A B C D
## 1 a t 1 3
## 2 b u 2 2
```

Full Join

Au contraire de la jointure interne, la jointure complète conserve l'ensemble de tous les individus.




```
full_join(X,Y)
```

```
## Joining, by = c("A", "B")
```

```
##   A B  C  D
## 1 a t  1  3
```

```
## 2 b u  2  2
## 3 c v  3 NA
## 4 d w NA  1
```

Exercices

Donner le code R des requêtes suivantes :

1. Le nom et le poids du/des pokémon.s avec le.s plus lourd.s par génération.
2. Pour chaque pokémon, la différence entre sa base stats et la base stats moyenne.
3. Pour chaque pokémon, la différence entre sa base stats et la base stats moyenne des pokémon de sa génération et du même stade évolutif.

3 Analyse univariée

3.1 Variables quantitatives

Pour une variable quantitative (ou numérique), les statistiques de base que l'on peut calculer sont le minimum, le maximum, la moyenne, la variance et l'écart type, la médiane et les autres quantiles (on rappelle que le quantile d'ordre p est la valeur q t.q. p est la fraction des valeurs observées inférieures à q).

Soit la variable quantitative X, on considère les fonctions suivantes :

- Valeur minimale : `min(X)`
- Valeur maximale : `max(X)`
- Domaine = [min, max] : `range(X)`
- Moyenne arithmétique : `mean(X)`
- Variance : `var(X)`
- Écart-type : `sd(X)`
- Médiane : `median(X)`
- Quantiles principaux 0, 25, 50, 75 et 100% : `quantile(X)`
- Quantile pour le percentile $p \in [0, 1]$: `quantile(X,p)`
- Résumé synthétique de la distribution de X : `summary(X)`

En cas de données manquantes, on les exclue grâce à l'option:

```
na.rm = TRUE
```

Exercice

1. Donner la moyenne, l'écart-type et la médiane de la variable `against_fire` et `against_water`. Lequel de ces deux types (fire ou water) semble le plus efficace contre les pokémons ?
2. Déterminer le type le plus efficace au global sur l'ensemble des pokémons. Détaillez votre raisonnement.

3.1.1 Boîte à moustaches

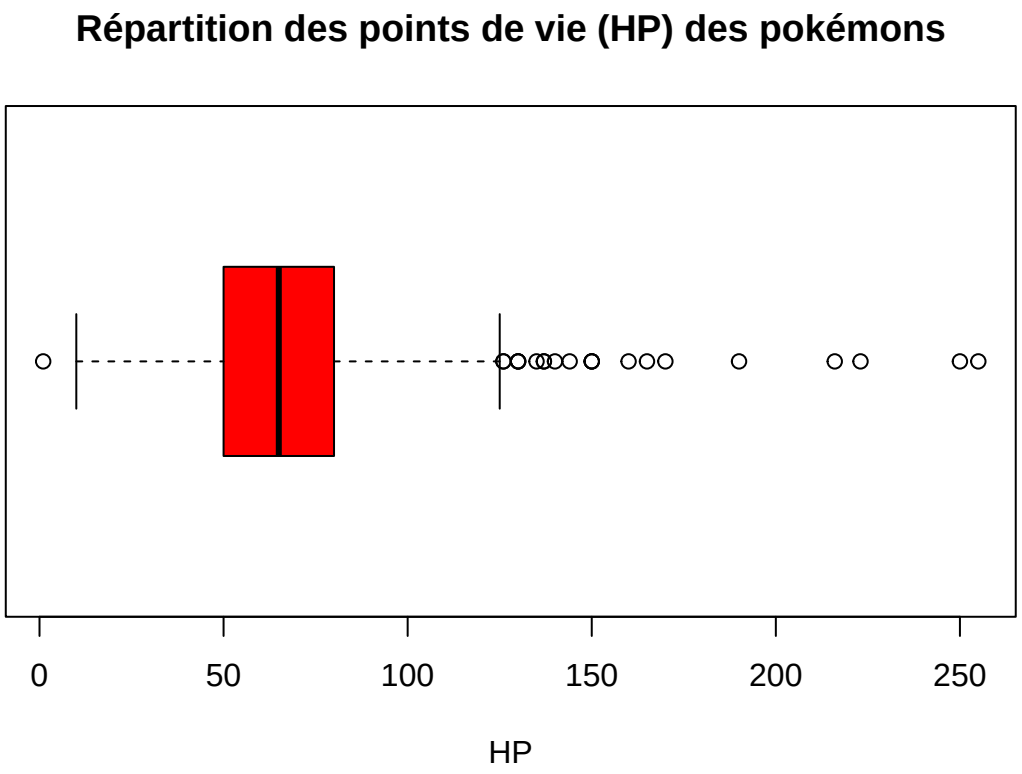
La commande :

```
boxplot(T$X)
```

Permet de visualiser la *boîte à moustaches* d'une série de données quantitatives X en provenance d'un data frame T.

On peut rajouter quelques options, par exemple:

```
boxplot(poke$hp,
        horizontal = TRUE,
        xlab = "HP",
        col = "red",
        main = "Répartition des points de vie (HP) des pokémons")
```



On rappelle l'analyse d'une boite à moustaches :

- La hauteur (ou largeur si on choisit l'option `horiz = TRUE`) est définie par le premier et troisième quartile q_1 et q_3.
- Le trait horizontal (ou vertical) correspond à la médiane $m = q_2$.
- La moustache en bas (ou à gauche) part de q_1 et va jusqu'au min de l'échantillon s'il n'y a pas de points extrêmes en bas (à gauche), c'est-à-dire des valeurs inférieures à $q_1 - 1.5 \times (q_3 - q_1)$.

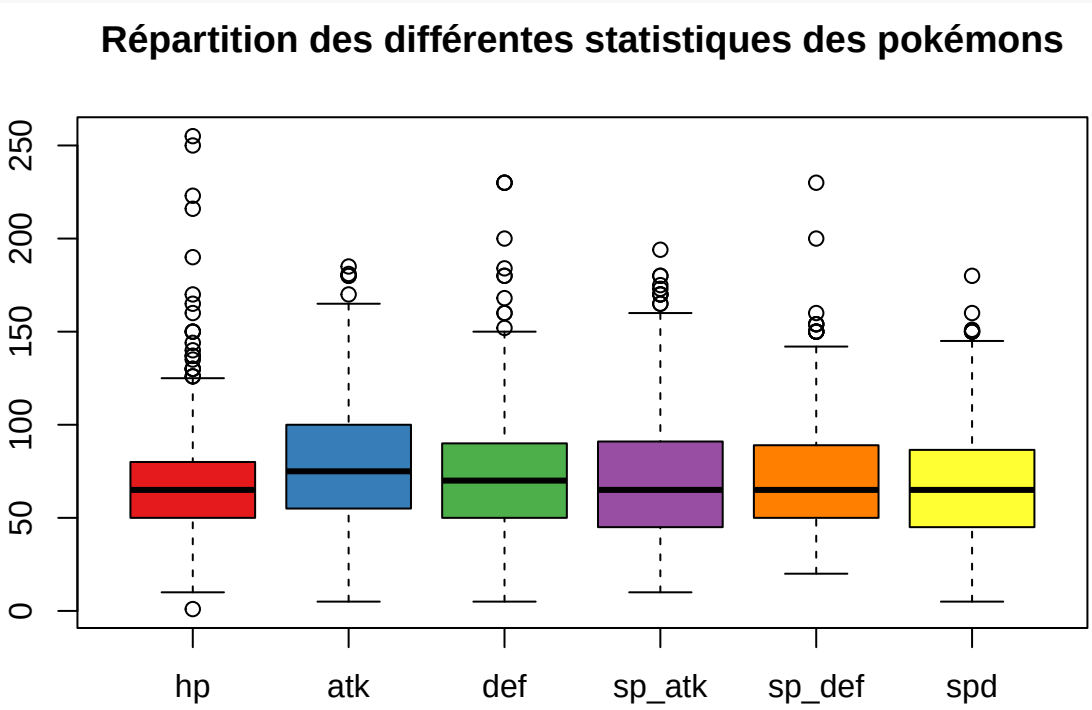
S'il y a de tels points, la moustache s'arrête en $q_1 - 1.5 \times (q_3 - q_1)$ et on les place au-dessous (à gauche)/

- La moustache en haut (ou à droite) part de q_3 et va jusqu'au maximum de l'échantillon s'il n'y a pas de points extrêmes à droite, c'est-à-dire des valeurs supérieures à $q_3 - 1.5 \times (q_3 - q_1)$. S'il y a de tels points, la moustache s'arrête en $q_3 - 1.5 \times (q_3 - q_1)$ et on les place au-dessus (à droite) de celle-ci.
- Les éléments aberrants sont alors représentés en dehors des moustaches par des points. Ils peuvent être filtrés par l'option `outline = FALSE`.

On peut également faire figurer plusieurs boîtes à moustaches sur un même graphique, par exemple :

```
library(RColorBrewer)
```

```
boxplot(poke[8:13],
        names(poke)[8:13],
        main = "Répartition des différentes statistiques des pokémons",
        col = brewer.pal(n = 6, name = "Set1"))
```




```
# col est la couleur assignée boxplot et brewer.pal est la palette utilisée.
# Ici on utilise le package RBrewer et 6 échantillons de la palette "Set1"
```

3.1.2 Histogramme et graphique à densité

L'autre outil graphique indispensable pour représenter la distribution d'une variable quantitative est l'*histogramme*.

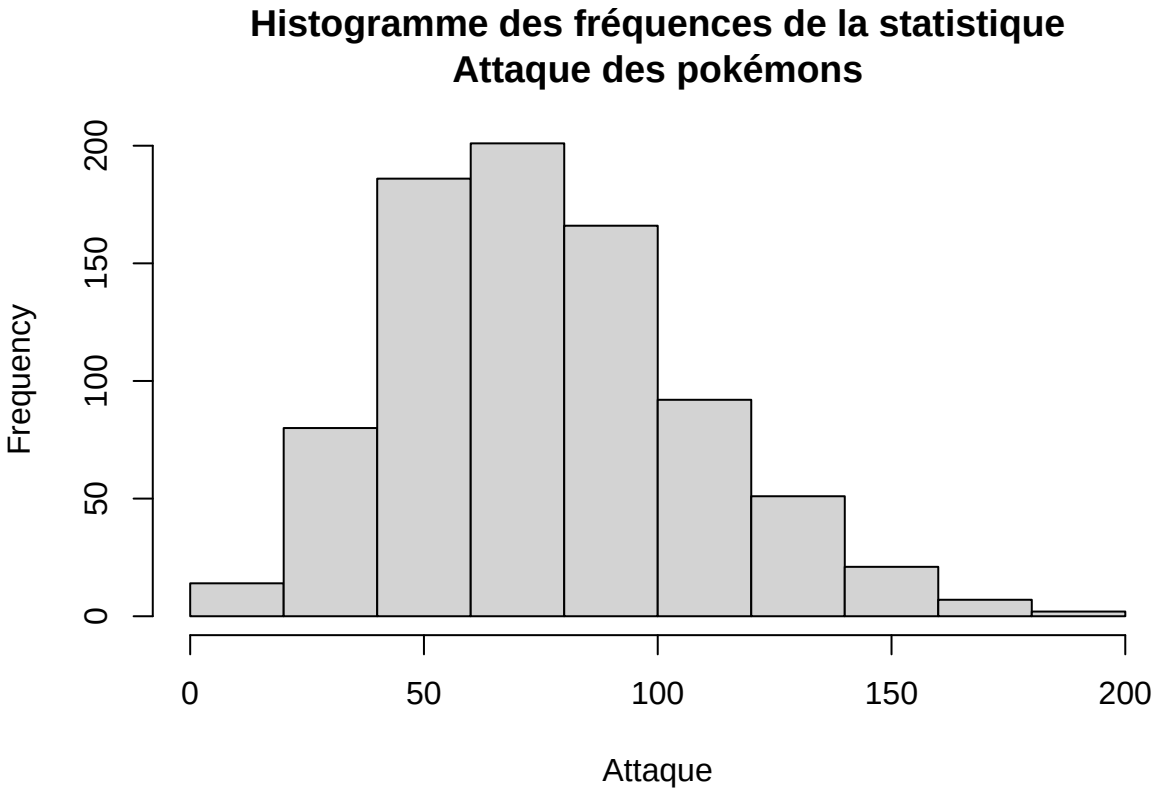
```
hist(T$X)
```

Par défaut, `hist()` affiche l'histogramme des fréquences, ce qui correspond au paramètre `freq=TRUE`.

On peut également régler le nombre de classes à l'aide de la commande `breaks = n` , où *n* est le nombre de classes sur l'histogramme.

Cet outil statistique est très efficace pour estimer la loi de distribution suivi par la variable que l'on analyse.

```
hist(poke$atk,
     main = "Histogramme des fréquences de la statistique\nAttaque des pokémons",
     xlab = "Attaque")
```



À noter que les éléments aberrants peuvent être ignorés lors de l'analyse.

Pour afficher l'histogramme avec la densité on utilisera `freq = FALSE`. Dans ce cas, l'aire de chaque rectangle sera égale à la proportion d'observations dans la classe correspondante (de façon à que l'intégrale de l'histogramme soit égale à 1). Par défaut, les classes sont définies sur des intervalles égaux.

Cette option permet de visualiser la densité estimée par noyau grâce à la commande `density`.

Un noyau $K : \mathbb{R} \to \mathbb{R}^+$ est une fonction de pondération, positive, intégrable et à valeurs réelles qui respecte les deux propriétés suivantes :

- $\int_{-\infty}^{+\infty} K(u)du = 1$ (Densité de probabilité)
- $\forall u \in \mathbb{R}, K(u) = K(-u)$ (Fonction paire)

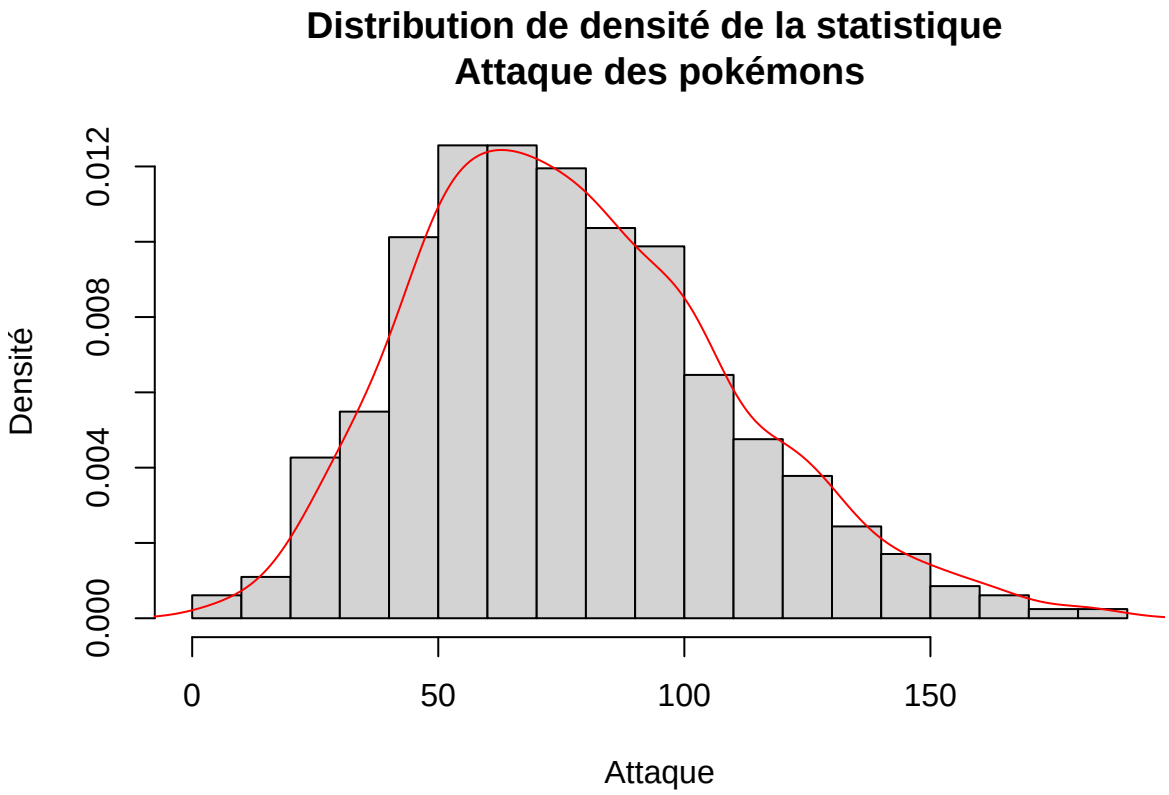
Il existe de nombreuses fonctions noyau, la plus connue et utilisée est celle du noyau gaussien $K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$.

Ainsi, si l'on dispose d'un échantillon $x_1, ..., x_n \hookrightarrow f$ de variables aléatoires i.i.d. alors l'estimateur non-paramétrique par la méthode du noyau de la densité est donnée par :

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

où h est un paramètre de lissage de la courbe de densité estimée. Il correspond au paramètre `bw` (bandwidth) de la méthode `density`.

```
hist(poke$atk,
     main = "Distribution de densité de la statistique \nAttaque des pokémons",
     breaks = 15, freq = FALSE, xlab = "Attaque", ylab = "Densité") ;
lines(density(poke$atk, kernel = "gaussian", bw = "nrd0"), col = "Red")
```



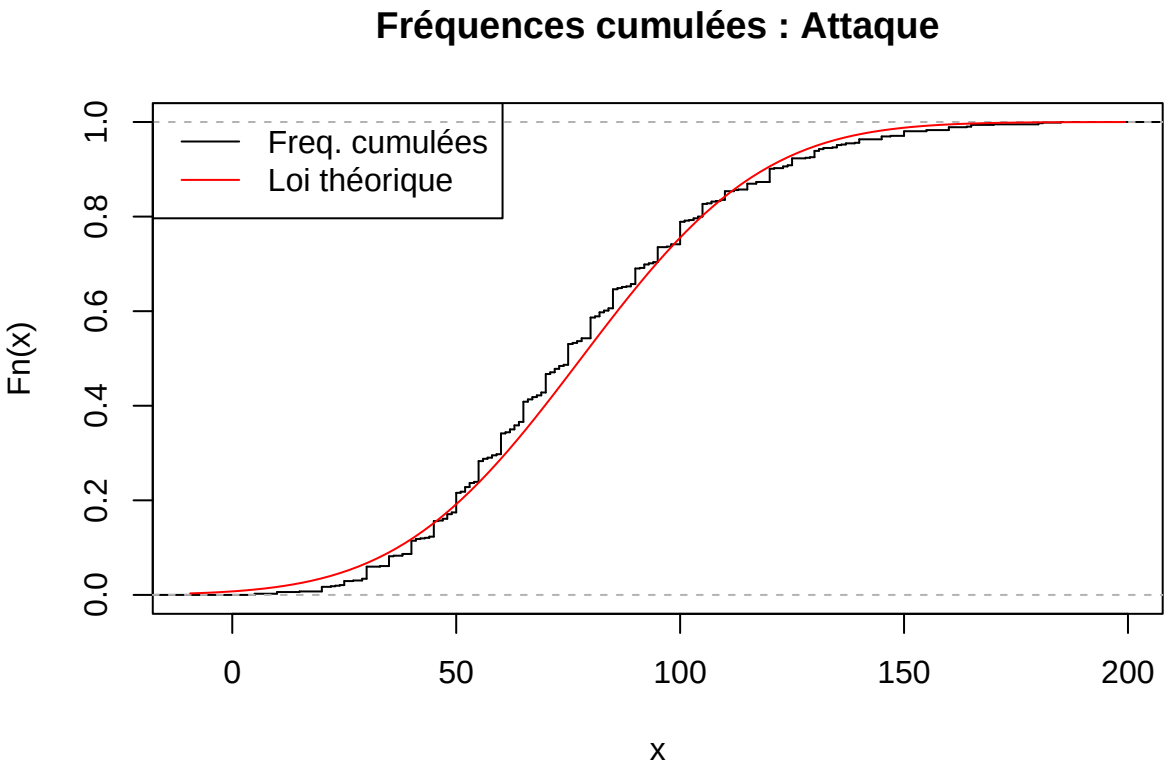
Le graphique des *effectifs cumulés croissants* peut-être utile également. Il sert à :

- Estimer la probabilité des observations.
- Constater les éventuelles ruptures caractéristiques au sein de la distribution, ce qui peut-être utile en cas de discrétisation.

```
# Fréquences cumulées croissantes
plot(ecdf(poke$atk),
     verticals = TRUE,
     do.points = FALSE,
     main = "Fréquences cumulées : Attaque")

# Courbe théorique (ici pnorm -> loi normale)
curve(pnorm(x,
            mean(poke$atk),
            sd=sd(poke$atk)
      ),
      col= "red", add = TRUE)

# Legende du graphe
legend("topleft", legend=c("Freq. cumulées", "Loi théorique"),
      col=c("black", "red"), lty=1, cex=1)
```



Notons au passage que le nombre retourné par la commande `sd` *correspond à l'estimateur sans biais* $\hat{\sigma}$ *de l'écart type, soit:*

$$\hat{\sigma} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x})}$$

Dans la suite de ce TP, sauf indication contraire, on considérera que $\sigma = \hat{\sigma}$.

3.1.3 Diagramme Quantile-Quantile

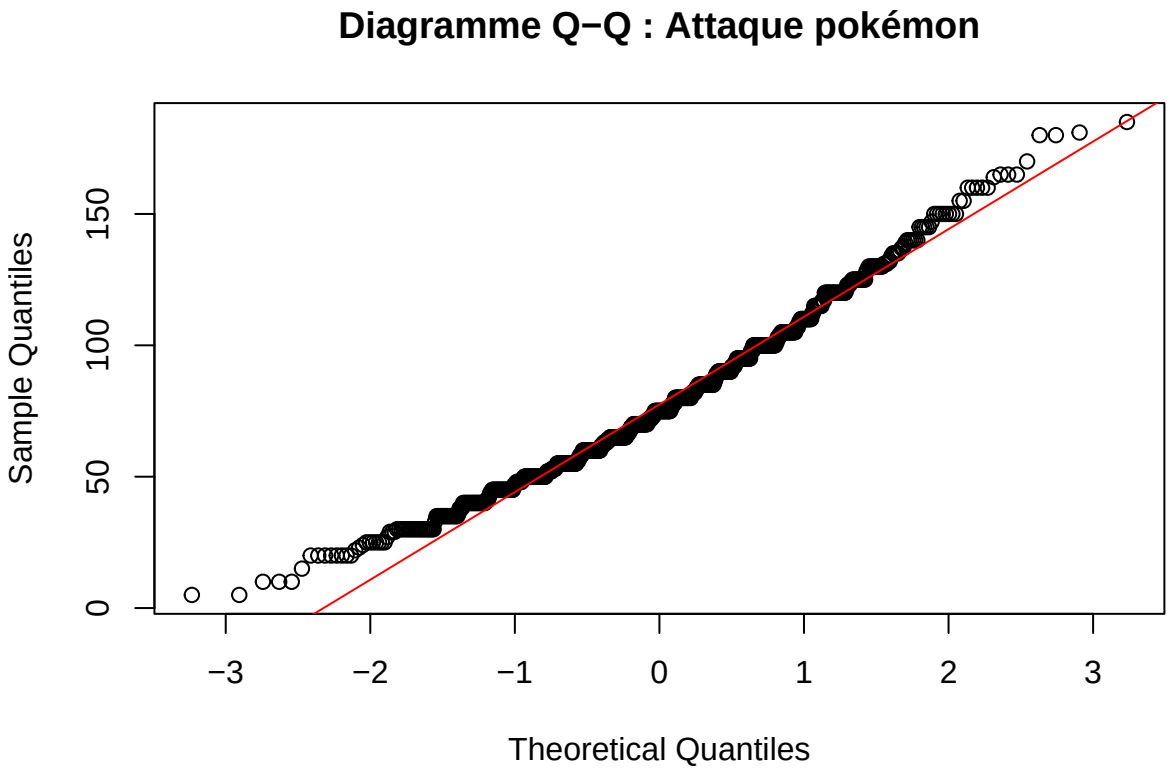
Pour évaluer la pertinence de l'ajustement d'une distribution donnée X à un modèle théorique, on utilise un *diagramme Quantile-Quantile* (ou diagramme Q-Q). Le plus souvent, on teste l'hypothèse d'une distribution normale.

```
qqnorm(T$X)
qqline(T$X) # la droite sample quantiles = theoretical quantiles
```

Les commandes suivantes permettent de générer le graphe ci-dessous :

```
qqnorm(poke$atk, main = "Diagramme Q-Q : Attaque pokémon")

# Impression de la ligne rouge
qqline(poke$atk, col = "red")
```



En abscisse se trouvent les quantiles théoriques x_i^* et en ordonnée les quantiles observés x_i. Si la distribution théorique choisie est pertinente, les points doivent s'aligner suivant la droite d'équation $x_i = \sigma_x x_i^* + \overline{x}$.

On peut ainsi confirmer que le modèle suit une loi normale d'espérance estimée $\overline{x}$ et d'écart type σ.

Un avantage de ce diagramme et que la normalisation Centrée-Réduite de x telle que

$$x \leftarrow \frac{x - \overline{x}}{\sigma_x}$$

n'a pas d'influence sur le graphique.

Exercice

1. Dresser l'histogramme de la variable `capt_rate`. Cette variable suit-elle une loi normale ? Expliquer votre réponse.
2. Dresser les graphiques des fréquences cumulées croissantes et la boite à moustaches de `capt_rate`. Que constatez-vous ? Quelles sont les similitudes entre ces deux graphiques ?

3.2 Variables qualititatives

3.2.1 Résumer des données et manipulations

Les fonctions `summary()` et `table()` appliquées à une variable qualitative (ou non numérique) X de T comptent les occurrences des différentes modalités :

```
table(T$X)
```

Par exemple :

```
table(poke$color)
```

```
##
##  Black   Blue  Brown  Green   Grey   Pink Purple    Red  White Yellow
##     43    149    124     87     82     47     75     83     61     69
```

Pour obtenir les proportions :

```
prop.table(table(poke$color))
```

```
##
##       Black        Blue       Brown       Green        Grey        Pink      Purple
## 0.05243902 0.18170732 0.15121951 0.10609756 0.10000000 0.05731707 0.09146341
##         Red       White      Yellow
## 0.10121951 0.07439024 0.08414634
```

Quand plusieurs colonnes possèdent des données liées ou similaires, il peut être intéressant d'effectuer des opérations de jointures ou d'agrégation de type SQL pour les manipuler.

Par exemple, les variables `type1` et `type2` qui représentent le ou les types d'un pokémon. Ces variables doivent être interprétées conjointement si on souhaite représenter les proportions de pokémons pour chaque type.

On procède ainsi :

```
# On dispose les deux analyses des variables dans un data frame
T1 <- data.frame(poke$type1) ; names(T1) <- c("type")

# On supprime les blancs à l'aide de la fonction subset
T2 <- subset(poke, !is.na(type2))[,"type2"] ; names(T2) <- c("type")

T <- data.frame(table(rbind(T1, T2)))

# On trie selon la fréquence d'apparition
T <- T[order(T$Freq),]
```

3.2.2 Graphique à barres

Les *graphiques à barres* (ou bar plot) sont utilisés pour représenter le nombre d'occurrences d'une valeur au sein d'une variable qualitative. La commande :

```
barplot(table(T$X))
```

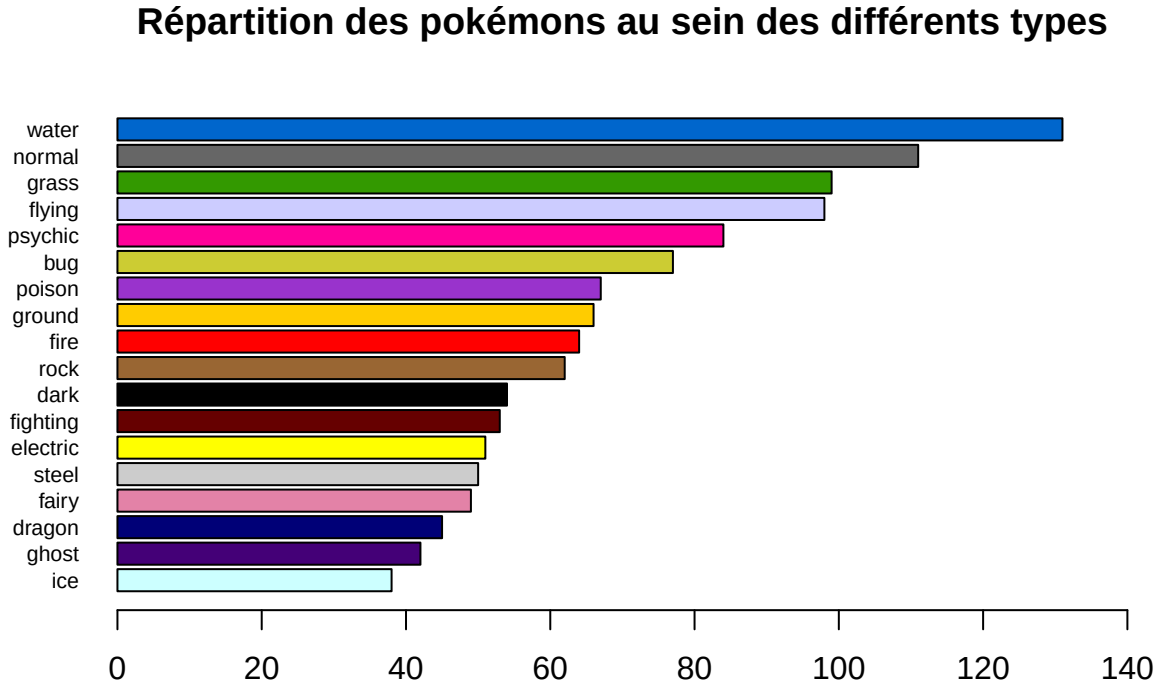
Permet de visualiser le graphique à barres de la variable X en provenance d'un data frame T. Parfois il est préférable, pour des raisons de visualisation, d'ordonner les valeurs. Dans ce cas, on peut suivre la procédure établie dans la section précédente.

Ainsi, si l'on considère le data frame T précédent, on peut construire le graphique à barres représentant l'occurrence des types de pokémon tel que :

```
# Fonction pour gérer les couleurs
col_type <- function(a) {
    if(a == "bug") {
        return("#CCCC33")
    }
    if(a == "dark") {
        return("#000000")
    }
    if(a == "dragon") {
        return("#000077")
```

```
    }
    if(a == "electric") {
        return("#FFFF00")
    }
    if(a == "fairy") {
        return("#E382A7")
    }
    if(a == "fighting") {
        return("#660000")
    }
    if(a == "fire") {
        return("#FF0000")
    }
    if(a == "flying") {
        return("#CCCCFF")
    }
    if(a == "ghost") {
        return("#440077")
    }
    if(a == "grass") {
        return("#339900")
    }
    if(a == "ground") {
        return("#FFCC00")
    }
    if(a == "ice") {
        return("#CCFFFF")
    }
    if(a == "normal") {
        return("#666666")
    }
    if(a == "poison") {
        return("#9933CC")
    }
    if(a == "psychic") {
        return("#FF0099")
    }
    if(a == "rock") {
        return("#996633")
    }
    if(a == "steel") {
        return("#CCCCCC")
    }
    if(a == "water") {
        return("#0066CC")
    }
    if (is.na(a)) {"#FFFFFF"}
    else {
        return("#FFFFFF")
    }
}
```

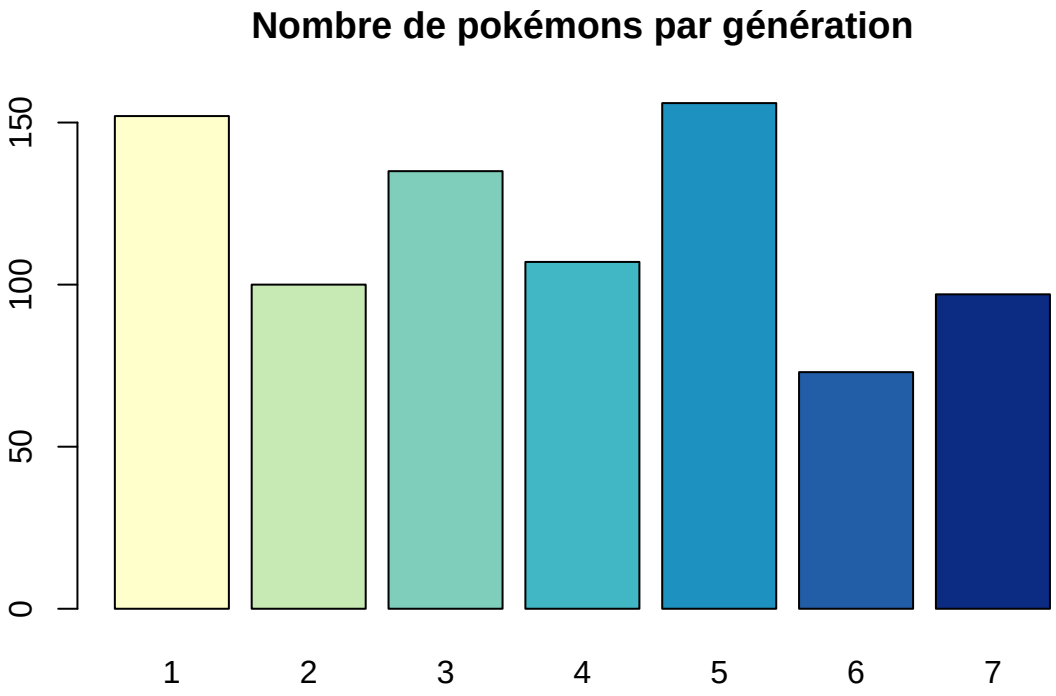
```r
barplot(names.arg = T[,1], T[,2], horiz = TRUE,
        col = unlist(lapply(T[,1], col_type)), # Couleurs en fonction du type
        las=1,  # Précise que la légende est à l'horizontale
        xlim=c(0,140), # Donne la limite de l'axe des x
        cex.names = 0.7, # Taille étiquettes réduites
        main = "Répartition des pokémons au sein des différents types")
```



Exercice

Donner le code R permettant de générer le graphique à barres suivant.

On précise que la palette utilisée est `YlGnBu`.



4 Analyse bivariée

Dans la suite de ce TP, nous allons utiliser la bibliothèque *ggplot2* pour générer de nouveaux graphiques.

Rendez-vous dans l'onglet Packages qui se trouve en bas à droite de la fenêtre de RStudio, puis recherchez le `ggplot2` ou installez-le si besoin.

4.1 Variables quantitatives vs variables quantitatives

On considère deux variables quantitatives X et Y mesurées sur les mêmes individus.

La covariance des deux variables quantitatives $\text{Cov}(X, Y)$ mesure les écarts conjoints par rapport à leur moyenne respective. On rappelle que:

$$\text{Cov}(X, Y) = \overline{X \times Y} - \overline{X} \times \overline{Y} = \frac{1}{n} \sum_{i=1}^{n} (x_i - \overline{x})(y_i - \overline{y})$$

Le coefficient de corrélation linéaire de Pearson

$$\rho_{XY} = \frac{\text{Cov}(X, Y)}{\sigma_X \sigma_Y}$$

mesure l'intensité de dépendance linéaire entre les deux variables quantitatives.

La covariance $\text{Cov}(X, Y)$ et la corrélation ρ_{XY} peuvent être obtenues à l'aide des commandes :

```
cov(X, Y)
```

```
cor(X, Y)
```

À noter que la commande `cov` retourne l'estimateur sans biais de la covariance avec pour dénominateur $n - 1$.

On rappelle les règles d'interprétation du coefficient de corrélation.

On a $\rho_{XY} \in [-1, 1]$, si $\rho \to 1$ (resp. -1) il y a une dépendance linéaire positive (resp. négative) et 0 s'il n'y a pas de dépendance linéaire. Globalement :

- Si $|\rho| \leq 0.5 \Rightarrow$ Corrélation faible voire nulle.
- Si $0.5 < |\rho| \leq 0.8 \Rightarrow$ Corrélation forte des données.
- Si $|\rho| > 0.8 \Rightarrow$ Corrélation très forte des données.

Dans le cas où les données ne suivent pas une distribution normale bivariée, il est recommandé d'utiliser le coefficient de corrélation de Spearman qui est plus robuste.

```
cor(poke$capt_rate, poke$base_stats)
```

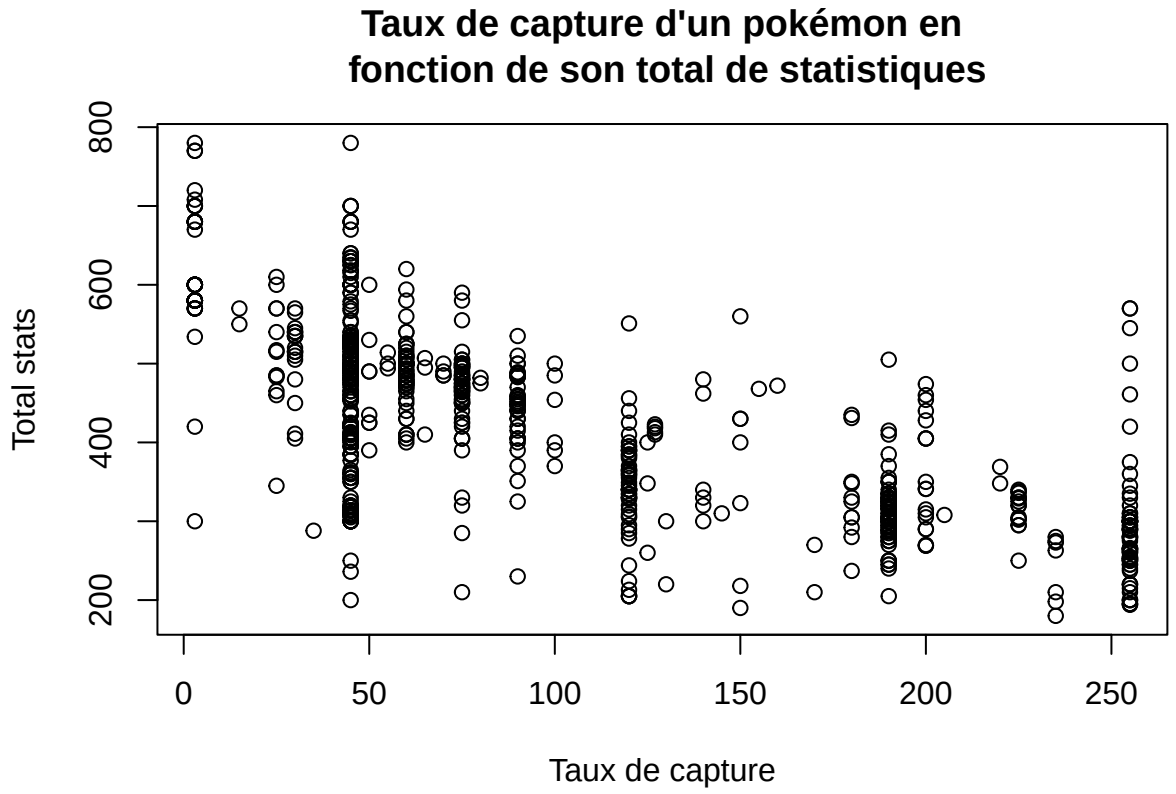
```
## [1] -0.7144634
```

Cette corrélation indique que plus un pokémon à une base totale de statistiques élevée, plus il est difficile à capturer.

On peut visualiser cet effet en traçant le nuages de point des variables quantitatives X et Y en provenance d'un data frame T à l'aide de la commande :

```
plot(X, Y)
```

```
plot(poke$capt_rate, poke$base_stats,
     xlab = "Taux de capture",
     ylab = "Total stats",
     main = "Taux de capture d'un pokémon en\n fonction de son total de statistiques")
```



À noter que l'on peut zoomer sur une zone du graphique en changeant les échelles grâce aux mots-clés `xlim` et `ylim`.

La librairie `ggplot2` va nous permettre d'embellir quelque peu notre graphique.

```
library(ggplot2)
library(viridis)
```

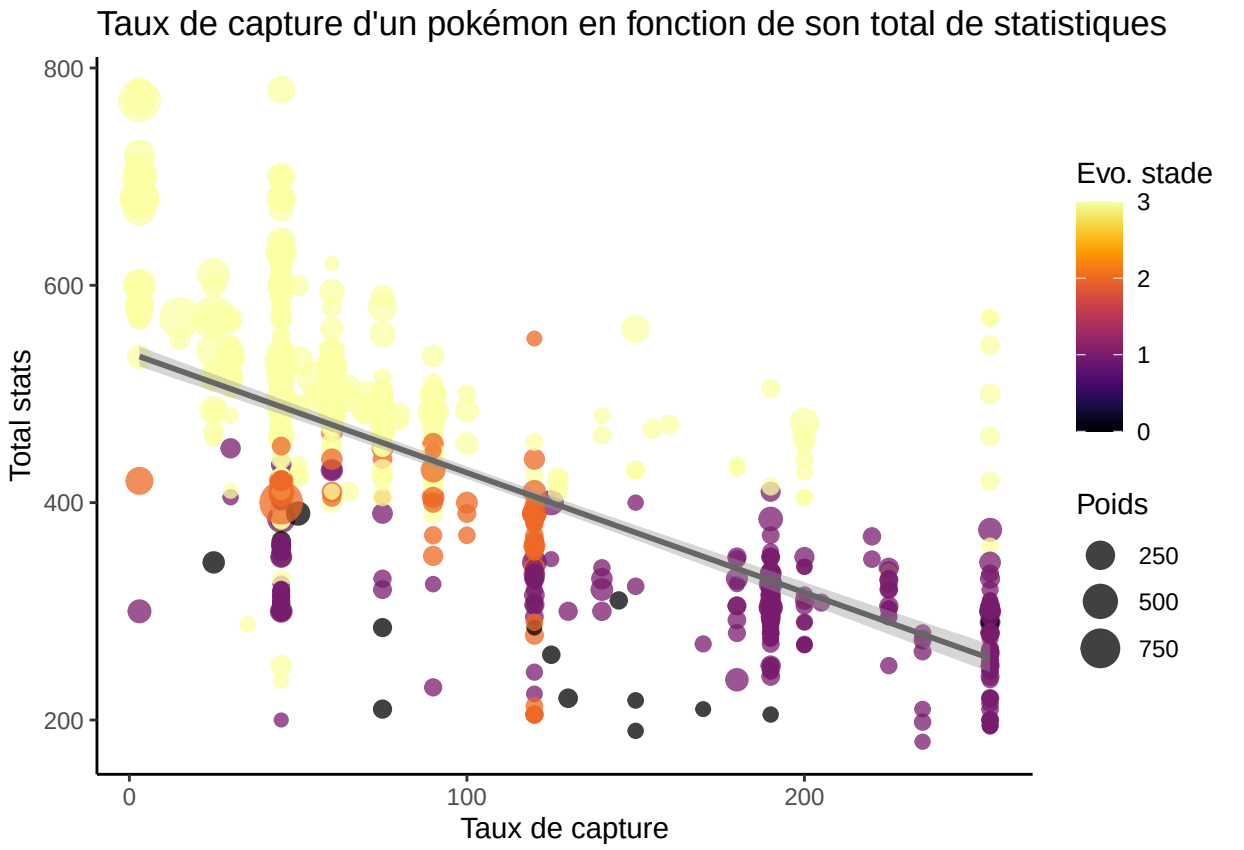
```
## Le chargement a nécessité le package : viridisLite
```

```
evo_stade <- as.numeric(poke$evolving_stade)

p <- ggplot(poke, aes(x = capt_rate, y = base_stats)) +
     geom_point(aes(color = evo_stade, size = weight), alpha = 0.75) +
     scale_size(range = c(2, 7))  + # Réglage de la plage de tailles des points
     scale_color_viridis(option = "inferno") + # Pour les échelles de couleurs continues
     geom_smooth(color = "#666666", method = "lm") +
     theme_classic() +
     labs(color = "Evo. stade",
          size = "Poids",
          x = "Taux de capture",
          y = "Total stats",
          title = "Taux de capture d'un pokémon en fonction de son total de statistiques")

p
```

```
## `geom_smooth()` using formula 'y ~ x'
```



À propos de la courbe de régression `geom_smooth()` l'option choisie par défaut est la régression locale. Comme ce type de régression est non paramètrique, elle a l'avantage de permettre d'obtenir un indicateur de regression non linéaire. Une régression plus connue est celle utilisée ici qui correspond à celle des moindres carrés obtenue par l'option `geom_smooth(method = lm)`.

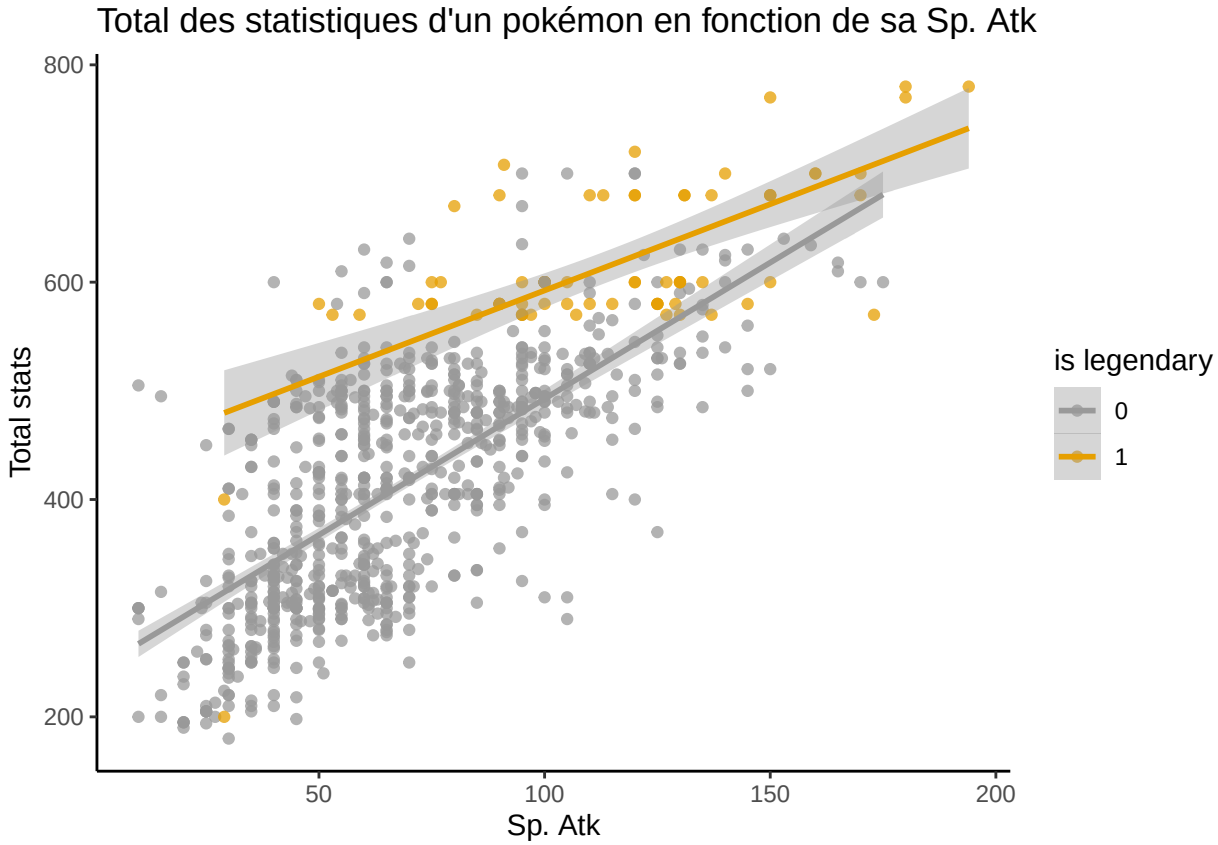
Exercice

1. Quel est le coefficient de corrélation entre les variables `sp_atk` et `base_stats`. Pouvez-vous expliquez

pourquoi ce score est élevé ?

2. Donner le code R permettant de générer le graphique ci-dessous. On veillera à utiliser les couleurs `'#999999'` et `'#E69F00'`.

```
## `geom_smooth()` using formula 'y ~ x'
```



4.2 Variables qualitatives vs variables qualitatives

4.2.1 Discrétisation d'une variable quantitative

Il peut parfois être utile de réduire un ensemble de valeurs $x_1, ..., x_n$ à certains intervalles $I_1, ..., I_q$ de sorte que $(I_1, ..., I_q)$ forme une partition (au sens large) de Dom(x).

Cette étape de *discrétisation* d'une variable X peut être assurée par la fonction `cut` telle que :

```
cut (X,breaks = c(n_0, ..., n_q),labels = seq(q-1))
```

Ainsi, `cut` crée les intervalles I_k tels que $I_1 =]n_0, n_1]$, $I_2 =]n_1, n_2]$, ..., $I_q =]n_{q-1}, n_q]$ de sorte que $cut(x_i) = k \Leftrightarrow x_i \in I_k$.

Par exemple, on peut discrétiser la variable `capt_rate` selon les différents quantiles comme suit:

```
quant <- quantile(poke$capt_rate)
quant[1] <- quant[1] - 1 # Pour prendre en compte la borne inférieure
capt_rate_disc <- cut(poke$capt_rate, breaks = quant, labels = seq(length(quant)-1))
```

À noter qu'il est possible aussi de discrétiser une variable par la méthode des *ruptures naturelles* (méthode de Jenks). Cette méthode regarde la distribution de la X et cherche les points de rupture afin de la séparer en différents seuils.

```
library (cartography)

seuils <- getBreaks (X, nclass = n, method = "fisher-jenks")
```

4.2.2 Histogramme empilé

Lorsque les variables disposent de nombreuses modalités, il peut être judicieux de représenter non pas un histogramme (afin de limiter le nombre de barres), mais plutôt un *histogramme empilé* (ou stack plot).

Attention, le stack plot s'affranchit du nombre d'individus et représente des proportions. Ce qui peut-être trompeur si les classes sont déséquilibrées...

Supposons ici que l'on souhaite visualiser la proportion de chaque type de pokémon pour chacune des générations. On procède ainsi:

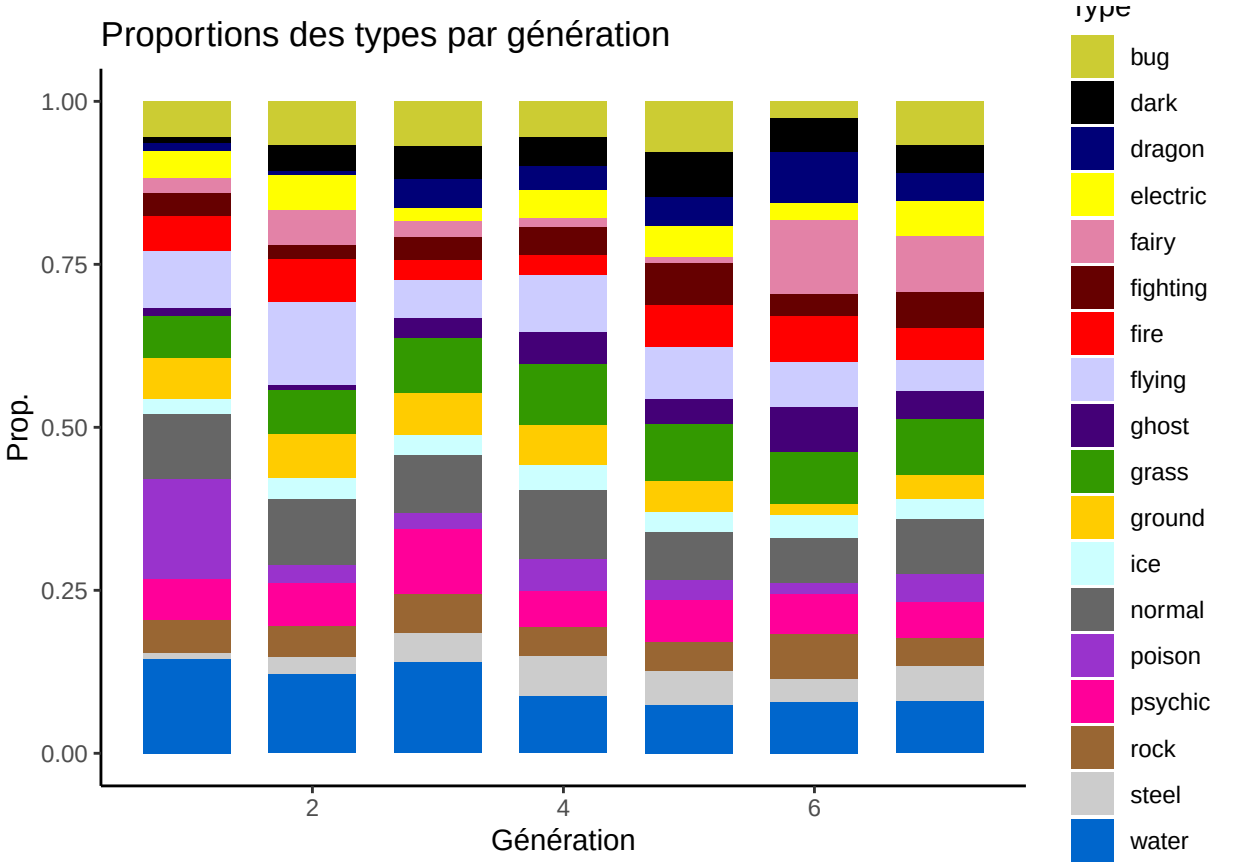
On rassemble les colonnes type1 et type2 comme vu précédemment.

```
T1 <- data.frame(poke[, c("generation", "type1")]) ; names(T1) <- c("gen", "type")

T2 <- subset(poke, !is.na(type2))[, c("generation", "type2")] ; names(T2) <- c("gen", "type")
T <- data.frame(rbind(T1, T2))
```

On dresse le stack plot à l'aide de la commande:

```
p <- ggplot(T) +
  geom_bar(width=0.7, mapping = aes(x = gen, fill = type), position = "fill") +
  scale_fill_manual(values= unlist(lapply(sort(unique(T$type)), col_type))) +
  theme_classic()  +
  labs(fill = "Type",
       x = "Génération",
       y = "Prop.",
       title = "Proportions des types par génération")
p
```



Exercice

Supposons que l'on veuille visualiser, pour chaque type, la proportion de chaque autre type de Pokémon. Par exemple, pour le type `grass`, on souhaite connaitre la proportion de pokémons de types `grass / t` où `t` est un type quelconque.

Plus formellement, on souhaite visualiser la proportion de chaque combinaison de types (t_1, t_2).

1. Créer un dataframe doté de deux colonnes `t1` et `t2` où chaque ligne correspond à un pokémon et où `t1` correspond au type primaine du pokémon et `t2` son type secondaire. Dans le cas où `type2 = NA`, `t2` prendra la valeur de `type1`. Par exemple, si on a `type1 = grass` et `type2 = NA`, on affectera à `t2` également la valeur `grass` ce qui correspond à dire que le pokémon est de type "plante" pur.
2. Donner le code R permettant de générer le graphique des proportions des types secondaires pour chaque type de pokémon.

4.2.3 Table de contingence et test du χ^2

Soient deux variables qualitatives X et Y de modalités respectives $x_1, ..., x_p$ et $y_1, ..., y_q$ et soit (n_{ij}), la fréquence d'observation des modalités x_i et y_j conjointe.

Le tableau des n_{ij}, $\forall i, j, 1 \leq i \leq p, 1 \leq j \leq q$ est appelé *table de contingence* et est donné par la commande :

```
T <- table(X,Y)
```

Par exemple, soient les deux variables B et W qui indiquent, respectivement, si un pokémon est bleu et si un pokémon est de type eau.

```
B <- ifelse(poke$color == "Blue", 1, 0)
W <- ifelse(poke$type1 == "water" |
            !is.na(poke$type2) & poke$type2 == "water", 1, 0)
```

```
T <- table(B,W)
T
```

```
##    W
## B     0   1
##   0 611  60
##   1  78  71
```

La commande `prop.table(T)` permet d'obtenir les probabilités estimées $\mathbb{P}(X = x_i \cap Y = y_j) \approx \frac{n_{ij}}{N}$

```
prop.table(T)
```

```
##    W
## B            0          1
##   0 0.74512195 0.07317073
##   1 0.09512195 0.08658537
```

Pour obtenir les probabilités conditionnelles, on utilise l'option *margin*. L'option `margin = 1`, permet d'obtenir les $\mathbb{P}(Y = y_j | X = x_i)$.

```
prop.table(T, margin = 1)
```

```
##    W
## B            0          1
##   0 0.91058122 0.08941878
##   1 0.52348993 0.47651007
```

```
# P(W=1|B=1) = 0.47, c'est-à-dire que la probabilité qu'un pokémon
# soit de type eau sachant qu'il est bleu est de 47%.
```

L'option `margin = 2`, permet d'obtenir les $\mathbb{P}(X = x_i | Y = y_j)$:

```
prop.table(T, margin = 2)
```

```
##    W
## B           0         1
##   0 0.8867925 0.4580153
##   1 0.1132075 0.5419847
```

```
# P(B=1|W=1) = 0.54, c'est-à-dire que la probabilité qu'un pokémon
# soit de type bleu sachant qu'il est de type eau est de 54%.
```

La commande `table()` utilisée avec deux variables qualitatives permet également d'effectuer un test du χ^2 d'indépendance.

```
chisq.test(table(X,Y))
```

Ce test permet de vérifier l'absence de lien statistique entre deux variables X et Y. Les deux sont dites indépendantes lorsqu'il n'existe aucun lien statistique entre elles. Dit autrement, la connaissance de X ne permet en aucune manière de se prononcer sur Y. L'hypothèse nulle (H_0) de ce test est la suivante : les deux variables X et Y sont indépendantes.

```
chisq.test(table(B,W))
```

```
##
```

```
##  Pearson's Chi-squared test with Yates' continuity correction
##
## data:  table(B, W)
## X-squared = 133.23, df = 1, p-value < 2.2e-16
```

En termes de valeur p, l'hypothèse nulle est généralement rejetée lorsque $p \leq 0,05$.

Ici, la p-valeur est bien inférieure à 0.05. On peut donc rejeter l'hypothèse (H_0) et affirmer que les variables B et W ne sont pas indépendantes. (On peut penser que les pokémons Eau sont plus souvent bleus...).

4.2.4 Diagramme mosaïque et résidus de Pearson

Un *diagramme mosaïque* est un modèle de représentation des tables de contingence. La commande :

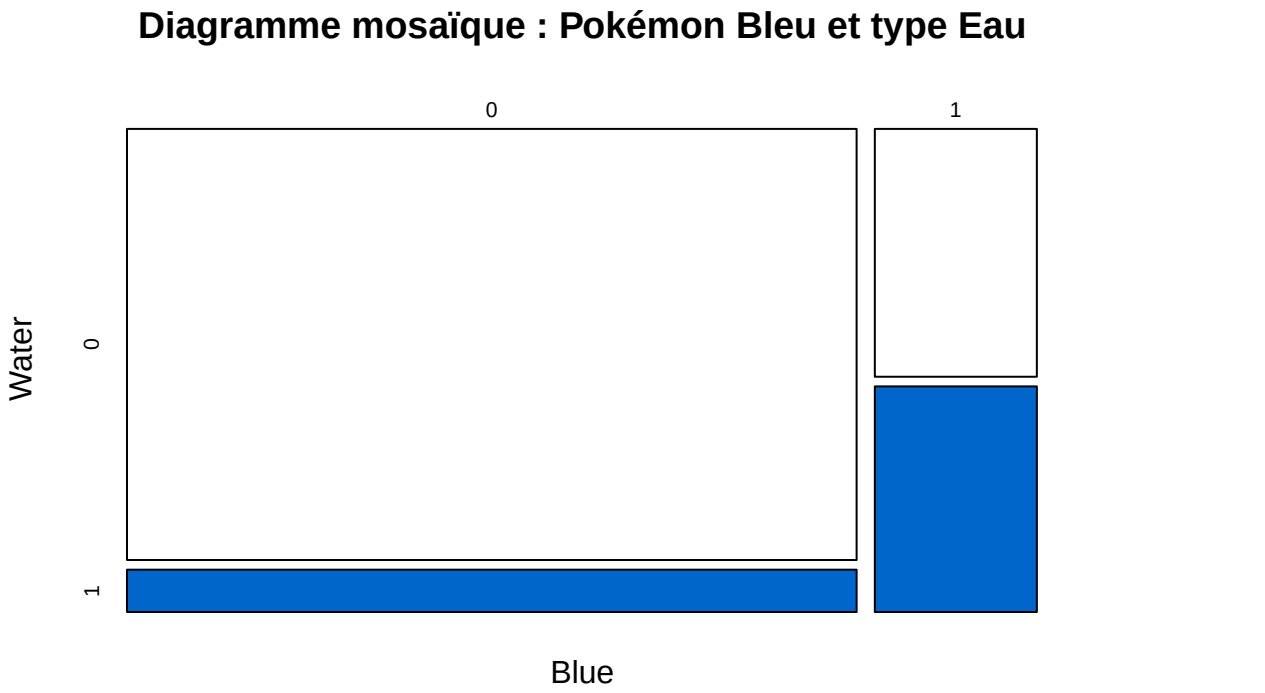
```
mosaicplot(table(Y,X))
```

Retourne un digramme de représentation des probabilités $\mathbb{P}(X|Y = x_1), ..., \mathbb{P}(X|y = y_q)$. On note c_{ij} la cellule ligne i, colonne j, alors:

- la largeur de c_{ji} est proportionnelle à la fréquence d'observation de y_j: $width(c_{ij}) \propto n_{+j}$
- la hauteur de c_{ji} est proportionnelle à la fréquence d'observation de x_i sous l'hypothèse de y_j: $height(c_{ij}) \propto \frac{n_{ij}}{n_{+j}}$
- l'aire de c_{ij} est proportionnelle à la fréquence d'observation conjointe de x_i et y_j: $area(c_{ij}) \propto n_{ij}$.

Attention cette commande n'est pas symétrique, `mosaicplot(table(Y, X))` $\neq$ `mosaicplot(table(X, Y))`.

```
mosaicplot(table(B,W),
        col = c("white", "#0066CC"),
        xlab = "Blue",
        ylab = "Water",
        main = "Diagramme mosaïque : Pokémon Bleu et type Eau")
```



L'option `shade = TRUE` du diagramme mosaïque permet de visualiser les *résidus de Pearson* r_{ij} des variables X, Y pour chaque modalité.

$$r_{ij} = \frac{n_{ij} - n^*_{ij}}{\sqrt{n^*_{ij}}}$$

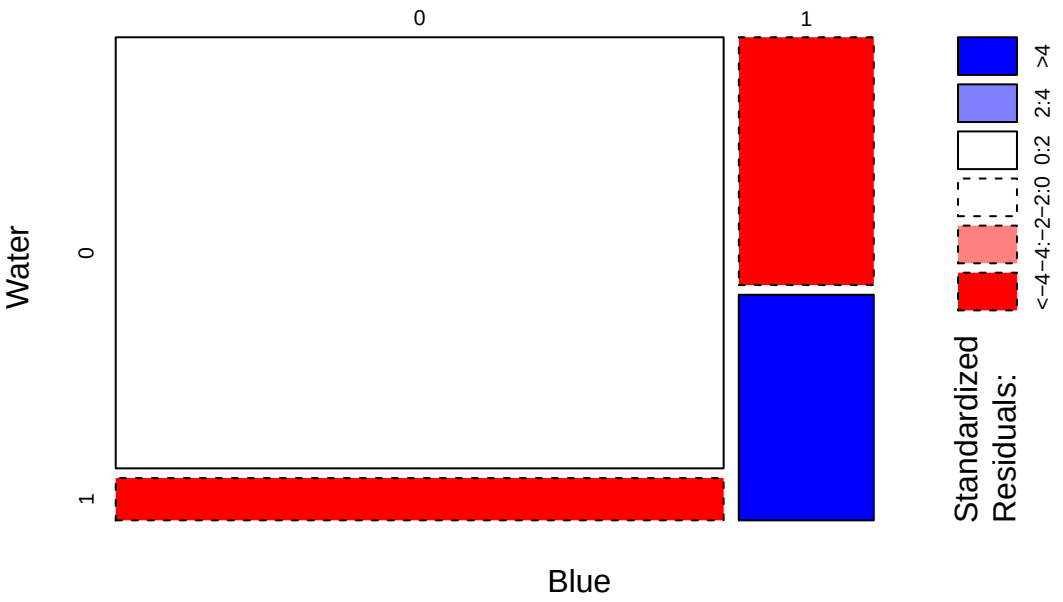
où $n^*_{ij} = \frac{n_{i+} \times n_{+j}}{N}$ et représente la fréquence théorique d'apparition conjointe des modalités x_i et y_j .

Les résidus de Pearson montrent la force et la direction de l'association entre x_i et y_j. La force est donnée par la valeur absolue du résidu ; la direction par son signe. Les unités sont en écart-type de sorte qu'un résidu supérieur à 2 ou inférieur à -2 représente un écart significatif par rapport à l'indépendance au niveau de 95%.

La valeur su χ^2 est obtenue par la somme des résidus de Pearson :

$$\chi^2 = \sum_{i=1}^{p} \sum_{j=1}^{q} r_{ij}$$

```
mosaicplot(table(B,W),
        shade = TRUE,
        xlab = "Blue",
        ylab = "Water",
        main = "Résidus de Pearson : Pokémon Bleu et type Eau")
```



Par exemple, on voit sur la figure que la cellule colonne 2, ligne 2 est colorée en bleu foncé. Ceci indique que l'on a effectivement une forte dépendance entre le fait qu'un Pokémon soit de type Eau et Bleu, comme nous l'avons vérifié précédemment par un test du χ^2 et la p-valeur.

Enfin, pour quantifier la taille d'effet, c'est-à-dire la force d'association entre nos variables X et Y, on peut calculer la *statistique du V de Cramér* telle que :

$$V = \sqrt{\frac{\chi^2}{N \times (\min\{p, q\} - 1)}}$$

Il est aussi obtenu par la commande :

```
    cramersV(table(X, Y))
```

Ce nombre représente la force de la relation entre les variables et non une statistique inférentielle qui permettrait de conclure ou non si ladite relation observée dans les données existe bien dans la réalité. En ce sens, la taille de l'effet est complémentaire d'autres mesures statistiques.

```
library(lsr)
cramersV(table(capt_rate_disc, poke$evolving_stade))
```

```
## Warning in stats::chisq.test(...): Chi-squared approximation may be incorrect
```

```
## [1] 0.4154502
```

On a $V \in [0, 1]$, avec l'interprétation suivante

- Si $V < 0.2 \Rightarrow$ relation faible, voire nulle
- Si $0.2 \leq V < 0.3 \Rightarrow$ relation moyenne.
- Si $V \geq 0.3 \Rightarrow$ relation forte.

On a donc ici une relation très forte entre nos variables étudiées.

Exercice

On souhaite vérifier l'hypothèse que les pokémons roses sont davantage genrés comme femelle (`percentage_male` < 50) que les autres pokémons.

1. Dresser l'histogramme de la variable `percentage_male` et proposer une méthode de discrétisation pour cette variable.
2. Créer une nouvelle variable `is_pink` qui vaut 1 si le pokémon est rose et 0 sinon.
3. Proposer une méthode permettant vérifier l'hypothèse puis conclure.

4.3 Variables quantitatives vs variables qualitatives

Ce type d'analyse est très fréquente lorsque l'on cherche à dresser un graphique de données agrégées par un Group by par exemple. Cependant, le nombre d'options en terme de visualisation est assez limité.

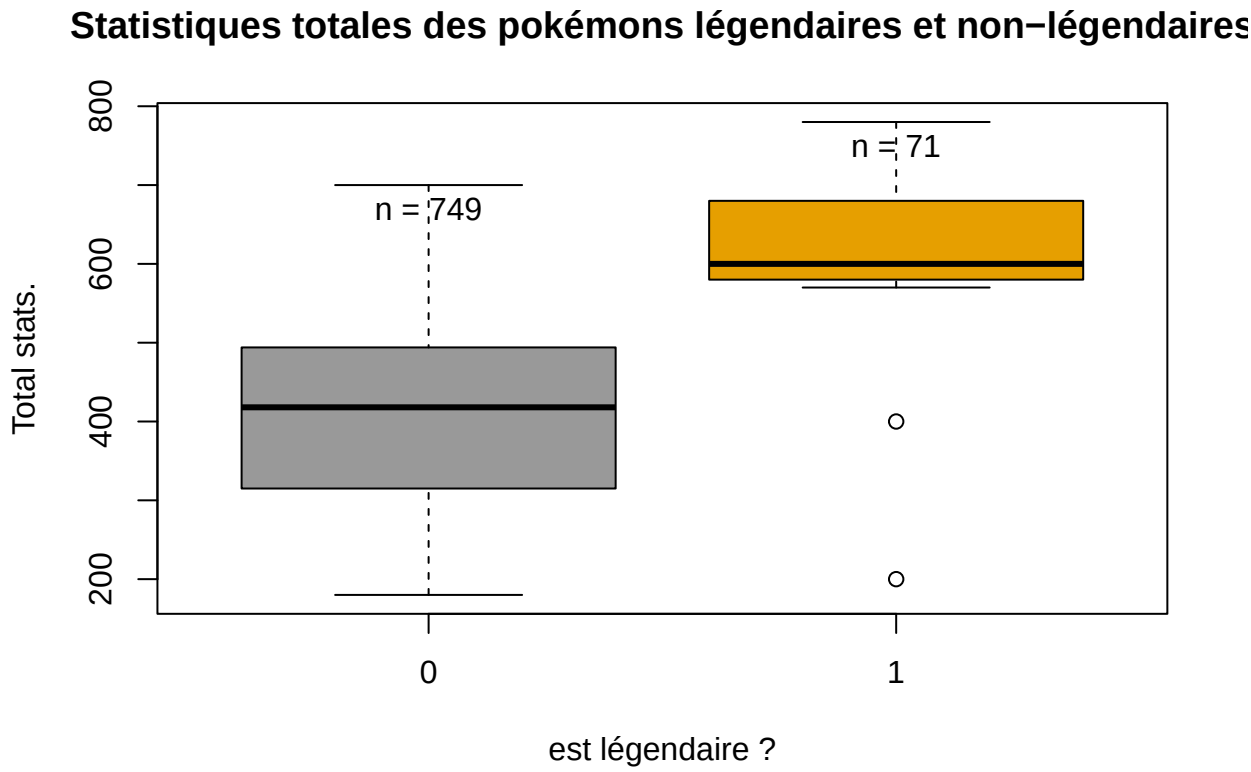
Un des choix les plus simples est celui d'une série de boîtes à moustaches grâce à la commande : `> boxplot(X ~ Y,data=T)`

Où X est une variable quantitative et Y, une variable qualitative.

Supposons que l'on souhaite connaitre la base totale de stats des pokémons en fonction de s'ils sont légendaires ou non. On peut utiliser le code suivant :

```
b <- boxplot(base_stats ~ is_legendary, data = poke,
        col = c('#999999','#E69F00'),
        xlab = "est légendaire ?",
        ylab = "Total stats.",
        main = "Statistiques totales des pokémons légendaires et non-légendaires")

# Permet d'ajouter le nombre d'éléments n
nbGroup <- length(unique(poke$is_legendary))
text(x=c(1:nbGroup), y=b$stats[nrow(b$stats),] - 30,
     paste("n = ", table(poke$is_legendary), sep=""))
```



Lorsque l'on souhaite agréger nos résultats par une variable supplémentaire, on peut imaginer la mise en place d'un diagramme à multi-barres (multi-bar plot).

Par exemple, supposons désormais que l'on souhaite connaitre la statistique totale (`base_stats`) des pokémons légendaires et non-légendaires, par génération.

On considère le code suivant qui permet de calculer la moyenne et l'écart-type d'une variable selon une liste de variables d'agrégation :

```
# Obtention de la moyenne de l'écart-type de `varname` group by `groupnames`
data_summary <- function(data, varname, groupnames){
  require(plyr)
  summary_func <- function(x, col){
    c(mean = mean(x[[col]], na.rm=TRUE),
      sd = sd(x[[col]], na.rm=TRUE))
  }
  data_sum <- ddply(data, groupnames, .fun=summary_func,
                    varname)
  data_sum <- plyr::rename(data_sum, c("mean" = varname))
 return(data_sum)
}
```

On peut dresser le graphique suivant :

```
summary <- data_summary(poke,
                        varname = "base_stats",
                        groupnames = c("is_legendary","generation"))
```

```
## Le chargement a nécessité le package : plyr
```

```
## -------------------------------------------------------------------------------
```

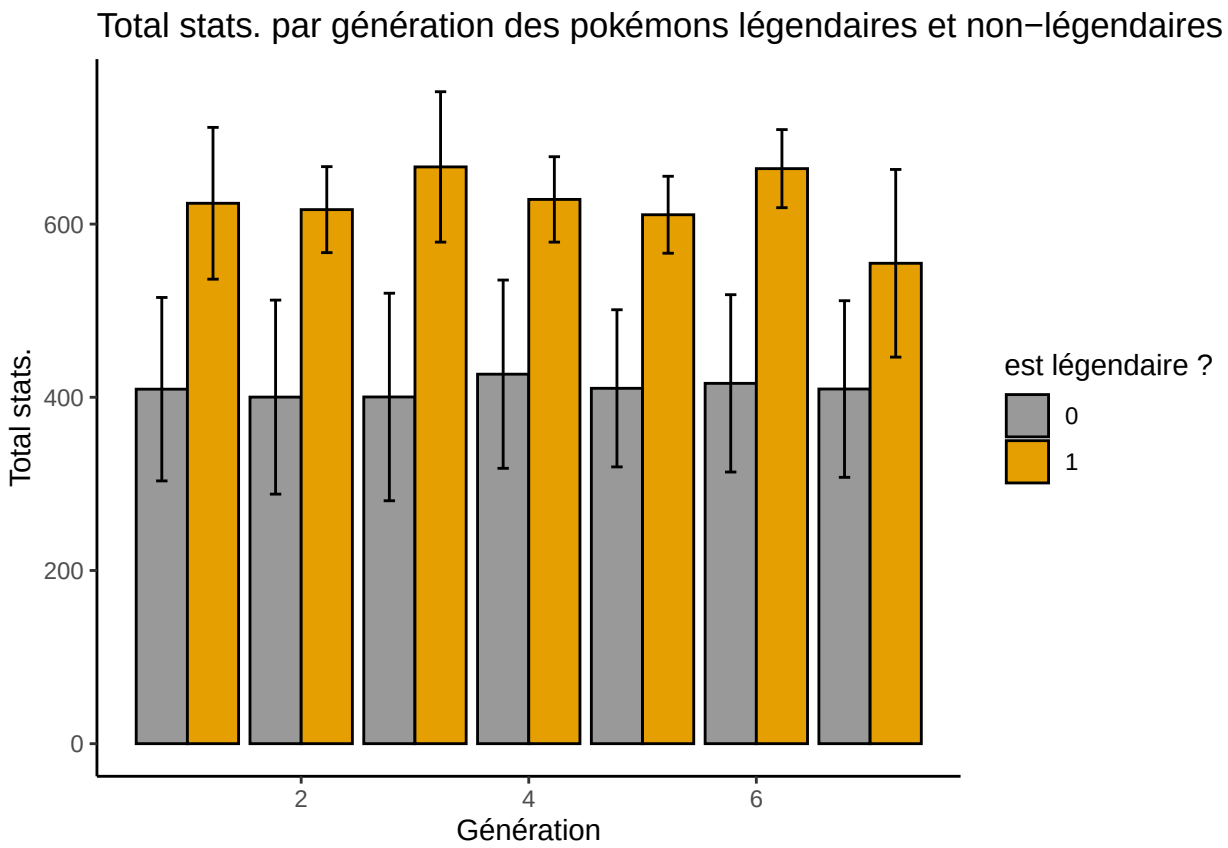
```
## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attachement du package : 'plyr'

## Les objets suivants sont masqués depuis 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize
```

```
p <- ggplot(summary, aes(x=generation, y=base_stats, fill=is_legendary)) +
  geom_bar(stat="identity", color="black",
           position=position_dodge()) +
  geom_errorbar(aes(ymin=base_stats-sd, ymax=base_stats+sd), width=.2,
                position=position_dodge(.9))  +
  theme_classic() +
  labs(title="Total stats. par génération des pokémons légendaires et non-légendaires",
       x="Génération",
       y = "Total stats.",
       fill = "est légendaire ?")+
  scale_fill_manual(values=c('#999999','#E69F00'))
p
```

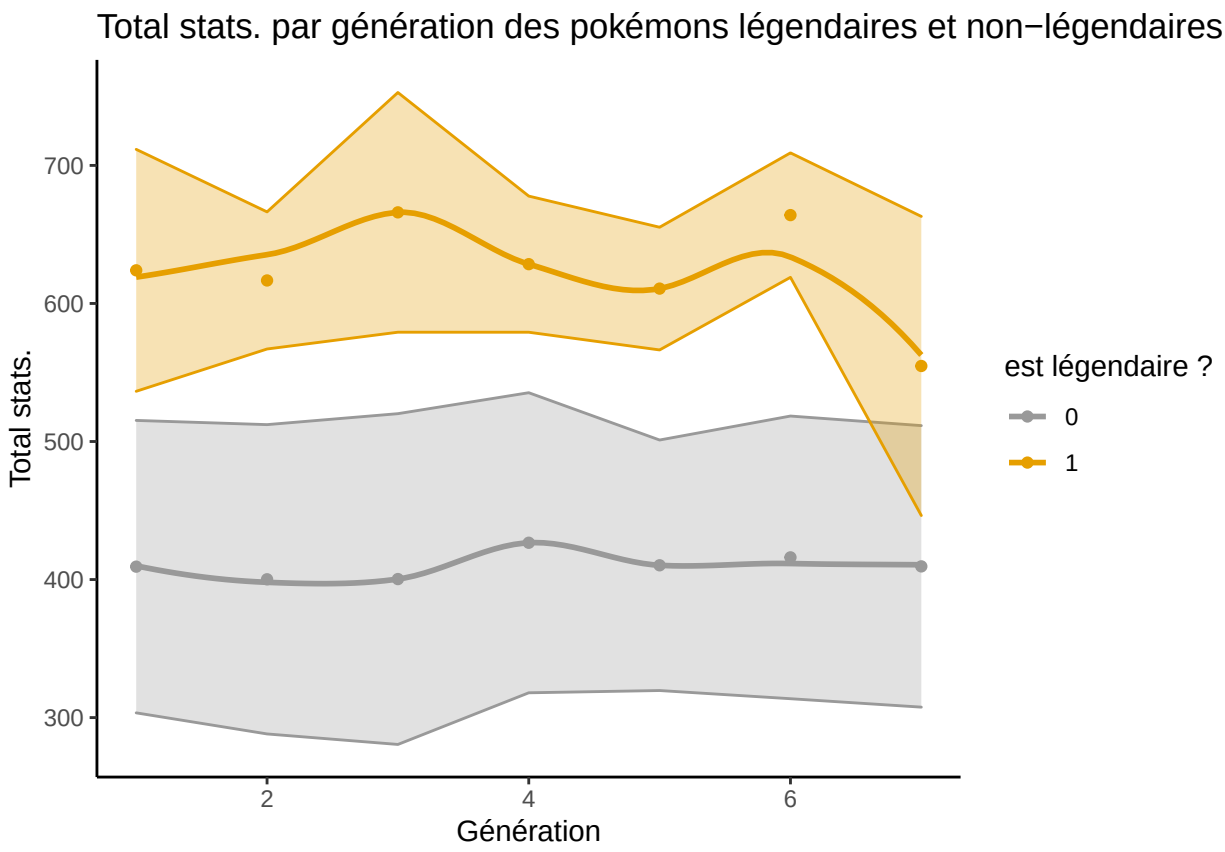


Comme la variable `generation` peut être considérée comme une variable temporelle, on peut aussi tracer une

évolution au cours du temps comme dans l'exemple ci-dessous. Un ruban est alors ajouté afin de représenter l'écart-type à la moyenne :

```
p <- ggplot(summary, aes(x=generation, y=base_stats, color=is_legendary)) +
  geom_point()+
  geom_ribbon(aes(ymin=base_stats-sd, ymax=base_stats+sd, fill = is_legendary), alpha=0.3,
              show.legend = FALSE)  +
  scale_color_manual(values=c('#999999','#E69F00')) +
  scale_fill_manual(values=c('#999999','#E69F00')) +
  geom_smooth(se = FALSE) +
  theme_classic() +
  labs(title="Total stats. par génération des pokémons légendaires et non-légendaires",
       x="Génération",
       y = "Total stats.",
       color = "est légendaire ?")
p
```

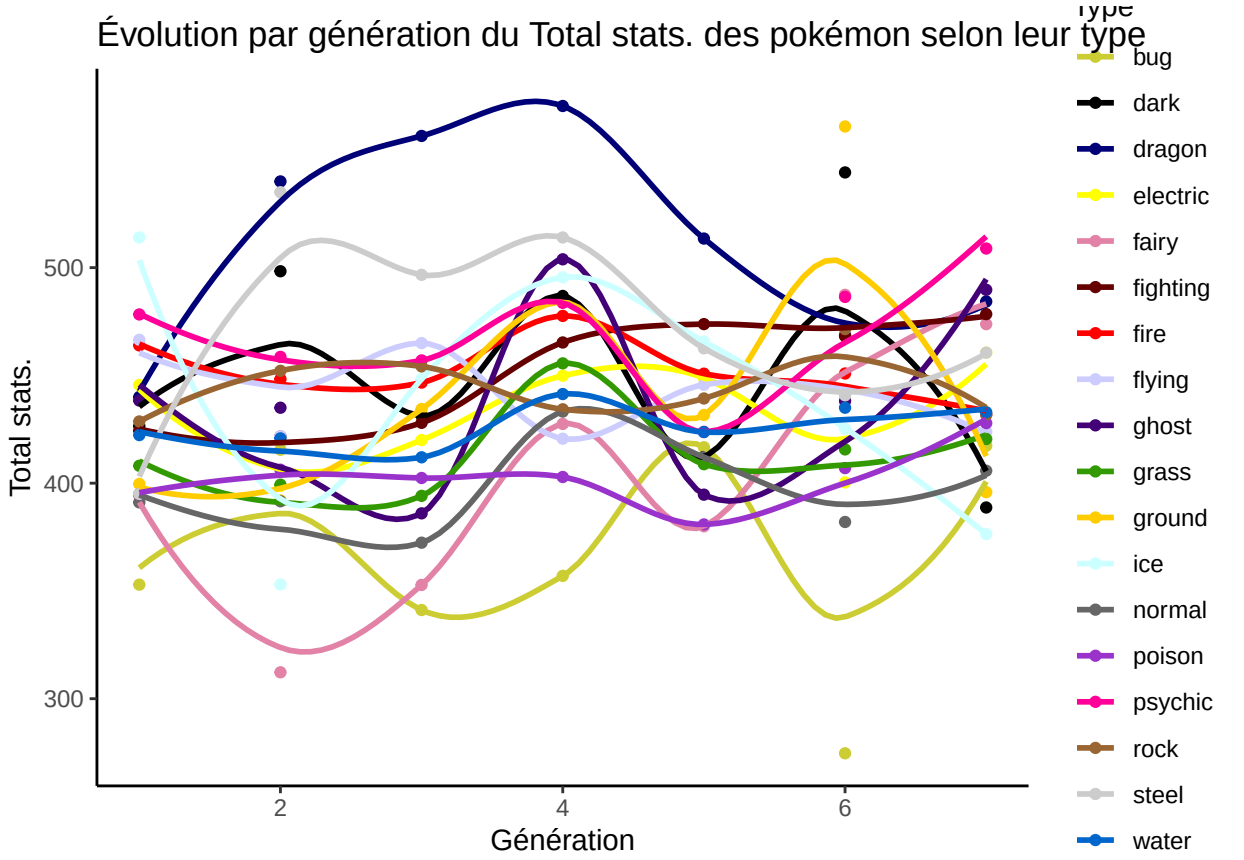
```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



Exercice

1. Donner le code R nécessaire afin de générer le graphique suivant:

```
## `geom_smooth()` using formula 'y ~ x'
```



Quelles analyses peut-on en tirer ? Quel est son défaut majeur ?

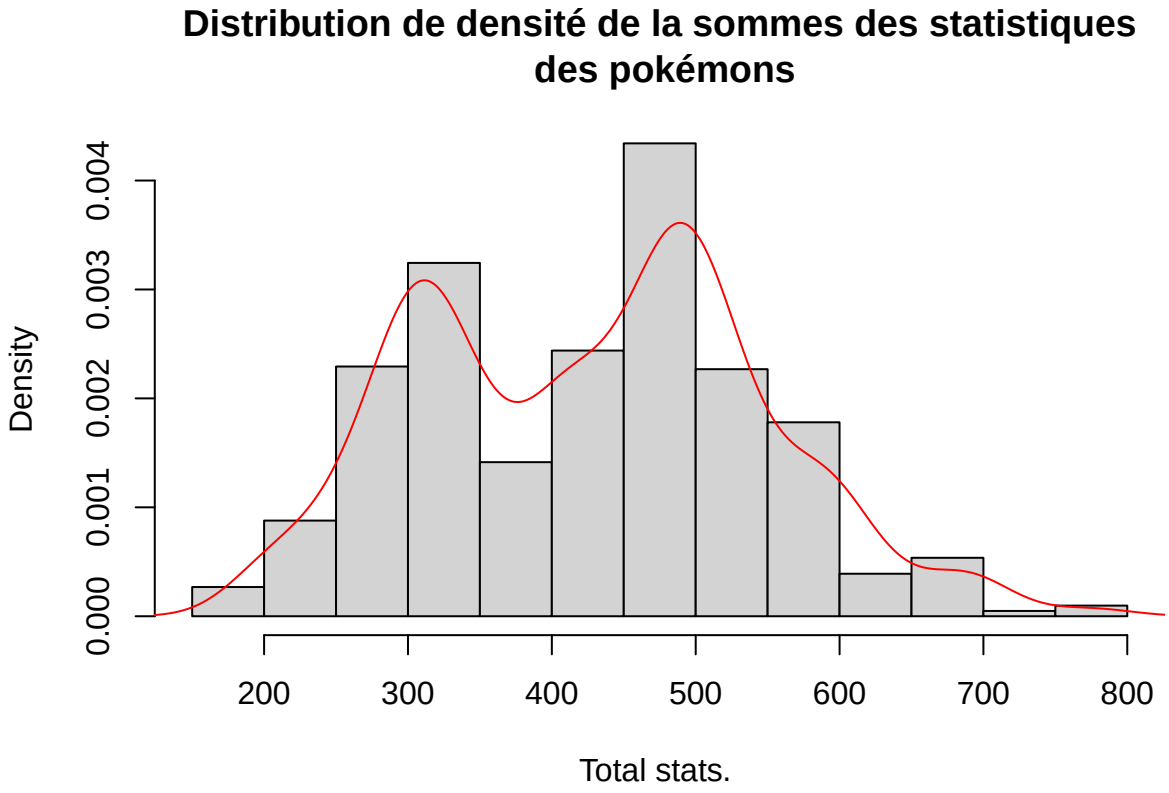
2. Reprendre le graphique précédent et faire figurer uniquement les courbes des types `dark` et `fairy`. On ajoutera également un ruban pour générer les écarts-types.

5 Quelques éléments avancés

5.1 Mélanges gaussiens

Il arrive parfois que les données observées ne suivent pas une loi normale stricte mais que l'on semble plutôt avoir une sorte de mélange de lois normales sur l'histogramme de densité. Par exemple, on peut remarquer ce phénomène si l'on observe la somme des statistiques d'un pokémon.

```
hist(poke$base_stats, freq = FALSE,
     xlab = "Total stats. ",
     main ="Distribution de densité de la sommes des statistiques\n des pokémons") ;
lines(density(poke$base_stats), col = "Red")
```



Ce type de phénomène est caractéristique de ce que l'on appelle les *mélanges gaussiens*. Il s'agit, pour faire simple, d'estimer paramétriquement la distribution de variables aléatoires en les modélisant comme une somme de plusieurs noyaux gaussiens. On cherche alors à déterminer la variance et la moyenne de chaque gaussienne.

Ces paramètres sont optimisés selon un critère de maximum de vraisemblance i.e. log-likelihood pour approcher le plus possible la distribution recherchée. Cette procédure se fait le plus souvent itérativement via l'algorithme espérance-maximisation (EM).

Visualiser et détecter des mélanges gaussiens peut-être fait à l'aide du package *mixtools*.

La commande pour générer les lois normales relatives d'un mélange gaussien en provenance d'une variable X est :

```
normalmixEM(X, mu = c(mu1, mu2,...mun), sigma=c(sd1,..., sdn), k=n)
```

À noter que les listes mu et sigma peuvent être omises si l'on n'a pas idée quant à l'initialisation des paramètre suivis.

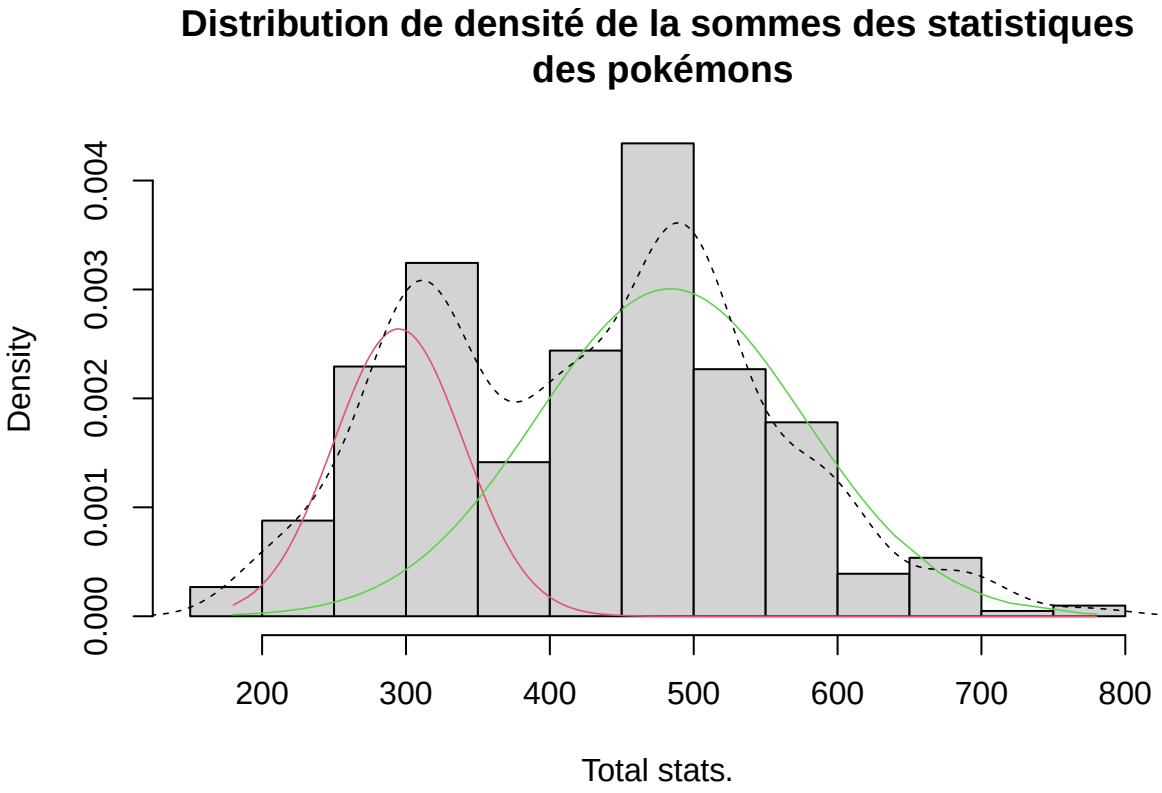
Dans notre cas d'analyse de `base_stats`, on peut estimer que l'on a deux lois normales cachées : Une centrée sur $\mu_1 \approx 320$ et l'autre $\mu_2 \approx 480$. Les écarts-types semblent autour de $\sigma_1 = 50$ et $\sigma_2 = 100$.

```
library("mixtools")
```

```
## mixtools package, version 1.2.0, Released 2020-02-05
## This package is based upon work supported by the National Science Foundation under Grant No. SES-0518
```

```
EM_poke <- normalmixEM(poke$base_stats, mu = c(320, 480), sigma=c(50,100), k=2)

plot(EM_poke, which=2,
     xlab2 = "Total stats.",
     main2 = "Distribution de densité de la sommes des statistiques\n des pokémons",
     lwd2=0.8)
lines(density(EM_poke$x), lty=2, lwd=0.8)
```



Exercice

1. Quel phénomène selon vous peut expliquer l'apparition de mélanges gaussiens ?
2. Déterminer une méthodologie d'analyse, que vous détaillerez, permettant d'expliquer la présence des deux gaussiennes sur le graphique précédent.
3. Conclure votre analyse et expliquer à quoi correspond, en terme d'individus, chacune de ces gaussiennes.

5.2 Analyse multivariée

Bien qu'il ne soit pas évident de représenter un grand nombre d'informations sur un même graphique, il existe des méthodes graphiques pour visualiser les informations sur plusieurs séries de données $X_1, X_2, ..., X_p$ d'un seul coup.

On note qu'ici les $(X_i)_{i\in[\![1,p]\!]}$ sont des variables quantitatives.

En premier lieu, installez le package *corrplot*.

```
install.packages("corrplot")
```

Comme il est parfois laborieux de détecter les corrélations entre variables, une méthode couramment employée est le calcul d'une matrice des corrélations entre une série de variables $\mathcal{X} = (X_1, ..., X_p)$ et la visualisation par un corrélogramme.

Supposons que l'on s'intéresse au colonne des types `against__Type_`.

On extrait les colonnes qui nous intéressent :

```
num_var <- poke[,24:ncol(poke)]
```

Pour utiliser la fonction `rquery.cormat`, vous pouvez la sourcer comme suit à l'aide du package *corrplot* :

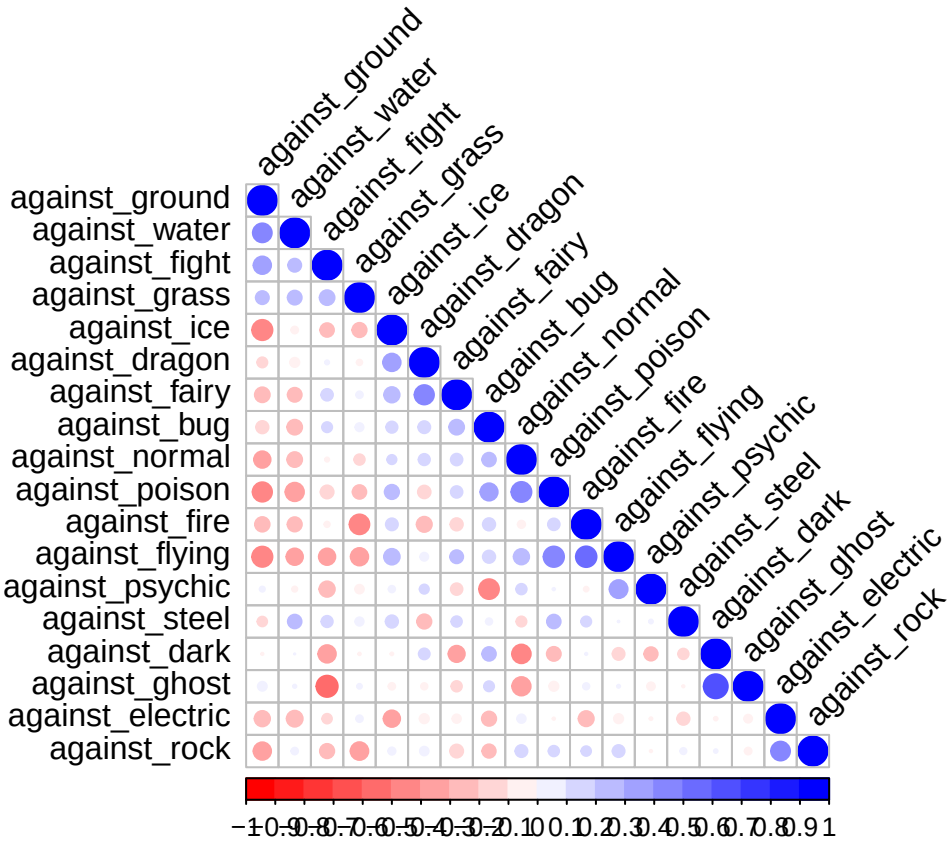
```
library("corrplot")
```

```
## corrplot 0.90 loaded
```

```
source("http://www.sthda.com/upload/rquery_cormat.r")
```

On obtient la matrice de corrélation par la commande :

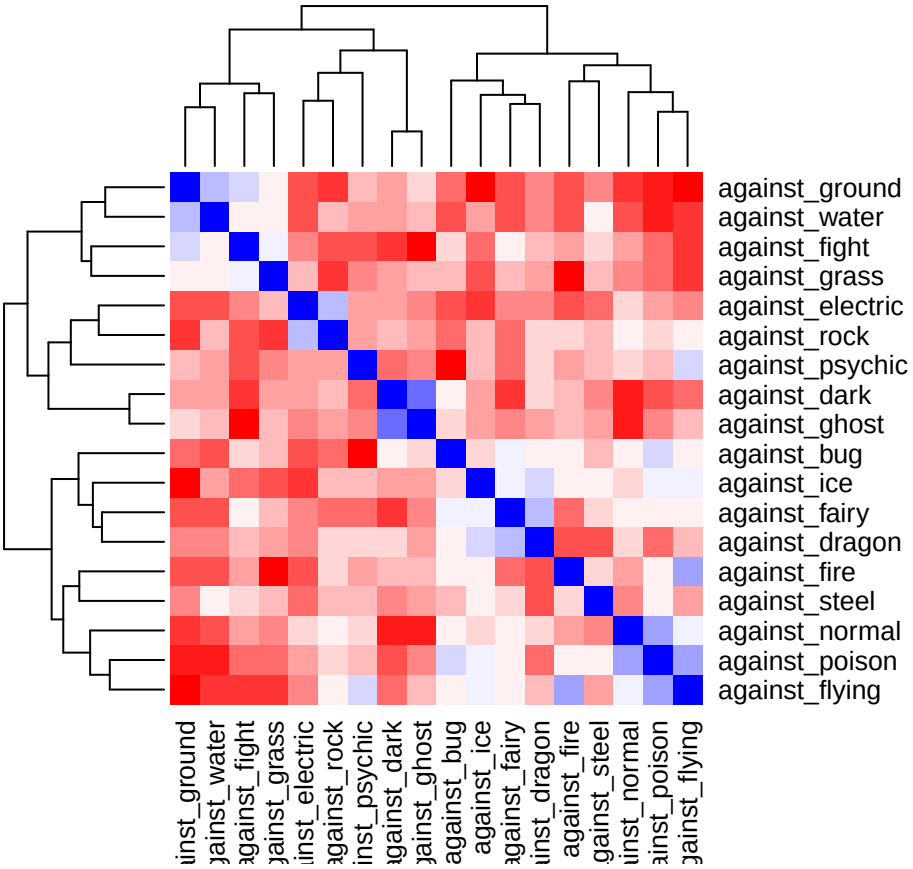
```
col<- colorRampPalette(c("red", "white", "blue"))(20)
rquery.cormat(num_var, col = col)
```



La taille du point correspond à la p-valeur entre les deux variables et la couleur au coefficient de corrélation [par défaut celui de Pearson].

On peut aussi dresser une heatmap (ou carte de chaleur) visible à droite de la matrice des corrélation et qui permet de regrouper facilement les différentes variables entre elles à l'aide d'un dendrogramme.

```
rquery.cormat(num_var, graphType="heatmap", col = col)
```



Exerice

Donner une analyse du dernier diagramme (carte de chaleur et dendrogramme).

6 Pour finir...

Exerice

Proposer 3 scénarios d'analyse que vous détaillerez et illustrerez selon votre choix. En particulier, vous préciserez et justifierez :

- Votre scénario d'analyse (but et démarche),
- Les outils statistiques que vous metterez en oeuvre,
- Votre choix de méthode de visualisation.
- Analyse et conclusion quant à vos observations.