AA18.1: Découverte de Rstudio 2

Premiers Travaux

Import des Jeux de données d'exemple

Dans cette partie nous allons (enfin) travailler sur des "vraies" données, et utiliser deux jeux de données archéologiques.

R n'est pas prévu pour la saisie de données, mais il bénéficie de nombreuses fonctions et packages permettant l'import de données depuis un grand nombre de formats.

Toutes les données doivent de préférence se trouver dans le même dossier.

Il y a 2 façon de travailler:

- travailler avec les Projets (en haut à droite de l'interface) Pour créer et parametrer un projet voir la section dédiée ici (https://juba.github.io/tidyverse/05-organiser.html#projets)
- définir le répertoire de travail en début de script via le panneau console: cliquer sur […] pour parcourir et choisir son répertoire de travail puis cliquer sur la roue crantée *More > Set as working directory*

Nous allons pour aujourd'hui choisir la deuxième solution ! Créer un dossier AA18 dans lequel vous avez copié les données `culot_blignicourt.csv` et `Amboise_ceram.xls`

Parcourir les dossier grâce au bouton `[...]` à droite du panneau Files et sélectionner le dossier **AA18**. Cliquer sur la roue crantée `More` puis choisir `Set as working directory`

> Dorénavant le logiciel reconnait ce dossier comme étant le dossier de travail par défaut c'est à dire que si l'on appelle un fichier il ira le chercher dans ce dossier. Aussi, si on enregistre un document depuis Rstudio (un graphique, un tableau, un rapport, etc) il sera enregistré dans ce dossier.

Notez la commande inscrite dans la console du genre:

```
> setwd("~/USER/blabla/AA18_1_Presentation/AA18")
```

> **Exercice: (https://slides.com/archeomatic/aa181/#/3/5)**
> 1. Créer un nouveau script **paleometallo.R**
> 2. Copier cette première ligne de commande qui commence par `setwd(...)`.
> 3. Commenter

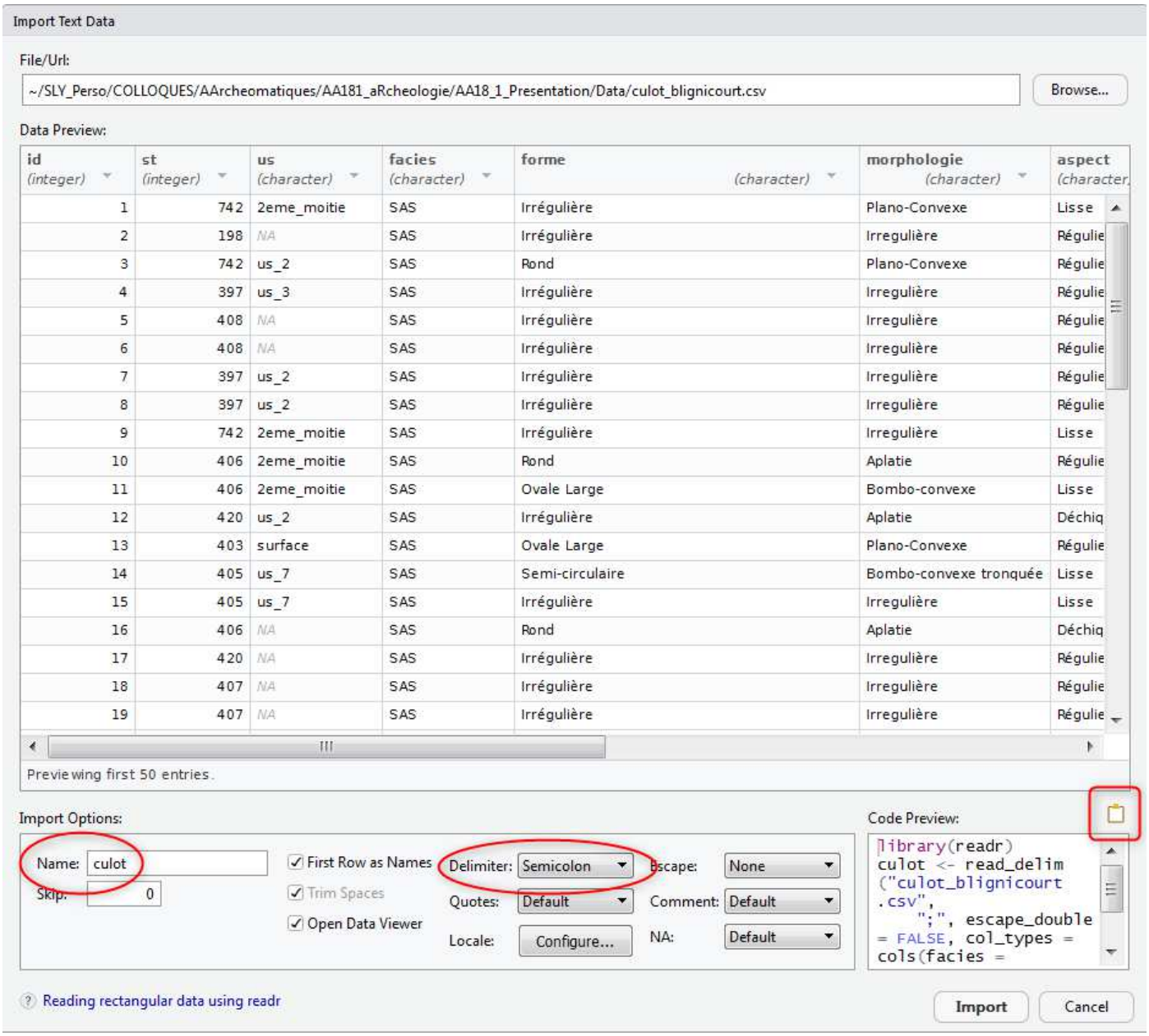
Import de fichiers textes

> Le premier jeu de données que nous allons utiliser est un extrait d'une étude paléometalurgique réalisée par Benjamin Jagou (Inrap-HDF) suite à un diagnostic dans la commune de Blignicourt(10) sur lequel plus de 100kg de rebuts siderurgiques ont été ramassés. Cet extrait concerne l'inventaire des scories de culots. Il contient 546 individus et 11 variables. Ce jeu de données est au format CSV.

Le plus simple pour importer un fichier tabulé dans R est d'utiliser le bouton [Import Dataset] du panneau **Environment**

Choisir l'option `from text (readr)`

Une fenêtre de dialogue apparait:



Import dataset

1. Cliquer sur le bouton [Browse] et parcourir pour trouver le fichier `culot_blignicourt.csv`
2. donner un nom court à l'objet qui va être créé: **culot**
3. Définir le séparateur de colonnes à *semicolon* (point virgule)
4. cliquer sur l'icone coller dans le presse papier (en bas à droite ou est affichée la fonction)
5. Coller dans la console et executer
6. Compléter votre script

Cette commande ne renvoit aucun résultat particulier (sauf en cas d'erreur), mais vous devriez voir apparaître dans l'onglet **Environement** de RStudio un nouvel objet nommé `culot` :

Cet objet est d'un type nouveau : il s'agit d'un tableau de données.

Tableau de données (*data frame*)

Un *data frame* (ou tableau de données, ou table) est un type d'objet dans R qui contient des données au format tabulaire, avec les observations en ligne et les variables en colonnes, comme dans une feuille de tableur de type LibreOffice ou Excel.

Si on se contente d'exécuter le nom de notre tableau de données :

```
culot
```

R va, comme à son habitude, nous l'afficher dans la console, ce qui est tout sauf utile.

Une autre manière d'afficher le contenu du tableau est de cliquer sur l'icône en forme de tableau à droite du nom de l'objet dans l'onglet *Environment*.

Ou d'utiliser la fonction `View` :

```
View(culot)
```

Dans les deux cas votre tableau devrait s'afficher dans RStudio avec une interface de type tableur.

Il est important de comprendre que l'objet `culot` contient *l'intégralité* des données du tableau. On voit donc qu'un objet peut contenir des données de types très différents (simple nombre, texte, vecteur, tableau de données entier), et être potentiellement de très grande taille

Un *data frame* peut être manipulé comme les autres objets vus précédemment. On peut par exemple faire :

```
c <- culot
```

ce qui va entraîner la copie de l'ensemble de nos données dans un nouvel objet nommé `c` . Ceci peut paraître parfaitement inutile mais a en fait l'avantage de fournir un objet avec un nom beaucoup plus court, ce qui diminuera la quantité de texte à saisir par la suite.

Pour résumer, comme nous avons désormais décidé de saisir nos commandes dans un script et non plus directement dans la console, les premières lignes de notre fichier de travail sur les données de l'enquête *Histoire de vie* pourraient donc ressembler à ceci :

```
## Definition du répertoire de travail
setwd("~/USER/blaba/AA18")

## Import du jeu de données (fichier csv)
culot <- read_delim("culot_blignicourt.csv",";", escape_double = FALSE, trim_ws = TRUE)
```

Import de fichiers Excel

> Le deuxième jeu de données est un extrait d'une étude céramologique de lots de mobiliers de l'Age du Fer réalisée par Francesca Di Napoli (INrap-CIF). Le mobilier céramique est issu d'une fouille sur l'oppidum d'Amboise. Outre des données de localisation (Fait, US, SD, Carré) ce tableau contient des informations de caractérisation typologique (groupe technique, forme,…) ainsi que la provenance et des élements de datation dont l'attribution a des horizons chronologiques. L'idée serait au final de pouvoir transformer ce tableau dans un format acceptable pour faire l'atelier archéomatique 18.2 (http://isa.univ-tours.fr/spip.php?article374). Il contient 2292 individus et 16 variables. Il est au format Excel (xls).

> **Exercice: (https://slides.com/archeomatic/aa181/#/3/6)**
> Créer un nouveau script **ceramologie.R**
> Définir le répertoire de travail (c'est le même)
> Importer ce tableau et le "stocker" dans un objet ceram

Structure du tableau

> Revenons a nos moutons.. ou plutôt nos culots

Un tableau étant un objet comme un autre, on peut lui appliquer des fonctions. Par exemple, `nrow` et `ncol` retournent le nombre de lignes et de colonnes du tableau :

```
nrow(c)
```

```
ncol(c)
```

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

```
dim(c)
```

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

```
names(c)
```

Enfin, la fonction `str` renvoit un descriptif plus détaillé de la structure du tableau. Elle liste les différentes variables, indique leur type et liste les premières valeurs :

```
str(c)
```

Sous RStudio, on peut afficher à tout moment la structure d'un objet en cliquant sur l'icône de triangle sur fond bleu à gauche du nom de l'objet dans l'onglet *Environment*.

Accéder aux variables d'un tableau

Une opération très importante est l'accès aux variables du tableau (à ses colonnes) pour pouvoir les manipuler, effectuer des calculs, etc. On va pour cela utiliser l'opérateur `$` , qui permet d'accéder aux colonnes du tableau. Ainsi, si l'on tape :

```
c$aspect
```

R va nous afficher l'ensemble des valeurs de notre variable `aspect` dans la console, ce qui est à nouveau fort peu utile. Mais cela nous permet de constater que `c$aspect` est un vecteur de chaînes de caractères tels qu'on en a déjà rencontré précédemment.

La fonction `table$colonne` renvoit donc la colonne nommée `colonne` du tableau `table` , c'est-à-dire un vecteur, en général de nombres ou de chaînes de caractères.

Si on souhaite afficher seulement les premières ou dernières valeurs d'une variable, on peut utiliser les fonctions `head` et `tail` :

```
head(c$poids)
```

```
tail(c$poids, 10)
```

Le deuxième argument numérique permet d'indiquer le nombre de valeurs à afficher.

Créer une nouvelle variable

On peut aussi utiliser l'opérateur `$` pour créer une nouvelle variable dans notre tableau : pour cela, il suffit de lui assigner une valeur.

Les scories de culot ont été mesurées dans les 3 dimensions qui sont contenues dans les champs `longueur` , `largeur` et `epaisseur` . On pourrait imaginer créer une variable `volume.glob` qui serait le volume.englobant c'est à dire la multiplication des 3 variables: Attention: ceci est une fiction ! tout calcul et création de variables n'engage personne !

```
c$volume.glob <- c$longueur * c$largeur * c$epaisseur
```

On peut alors constater, soit visuellement soit dans la console, qu'une nouvelle variable (une nouvelle colonne) a bien été ajoutée au tableau :

```
head(c$volume.glob)
```

Analyse univariée

On a donc désormais accès à un tableau de données `d` , dont les lignes sont des observations (des individus enquêtés), et les colonnes des variables (des caractéristiques de chacun de ces individus), et on sait accéder à ces variables grâce à l'opérateur `$` .

Si on souhaite analyser ces variables, les méthodes et fonctions utilisées seront différentes selon qu'il s'agit d'une variable *quantitative* (variable numérique pouvant prendre un grand nombre de valeurs comme le poids, des dimensions…) ou d'une variable *qualitative* (variable pouvant prendre un nombre limité de valeurs appelées modalités comme l'aspect, la forme, etc.).

Analyser une variable quantitative

Une variable quantitative est une variable de type numérique (un nombre) qui peut prendre un grand nombre de valeurs. On en a plusieurs dans notre jeu de données, notamment l'âge (variable `poids`) ou les dimensions mesurées des cories `longueur`, `largeur`, `epaisseur`.

Nous alons nous interesser à la variable `poids`. En premier lieu calculons la masse totale de scories de culot:

```
sum(c$poids)
```

soit 37598 grammes

On peut l'obtenir en kilogrammes

```
sum(c$poids) / 1000
```

soit 37,598 kg

Ou encore mieux: en kilogralme, arrondi à 1 chiffre dérrière la virgule, avec la fonction `round`

```
round(sum(c$poids) / 1000, 1)
```

Indicateurs de centralité

Caractériser une variable quantitative, c'est essayer de décrire la manière dont ses valeurs se répartissent, ou se distribuent.

Pour cela on peut commencer par regarder les valeurs extrêmes, avec les fonctions `min`, `max` ou `range` :

```
min(c$poids)
max(c$poids)
range(c$poids)
```

On peut aussi calculer des indicateurs de *centralité* : ceux-ci indiquent autour de quel nombre se répartissent les valeurs de la variable. Il y en a plusieurs, le plus connu étant la moyenne, qu'on peut calculer avec la fonction `mean` :

```
mean(c$poids)
```

Il existe aussi la médiane, qui est la valeur qui sépare notre population en deux : on a la moitié de nos observations en-dessous, et la moitié au-dessus. Elle se calcule avec la fonction `median` :

```
median(c$poids)
```

Une différence entre les deux indicateurs est que la médiane est beaucoup moins sensible aux valeurs "extrêmes" : on dit qu'elle est plus *robuste*. Ainsi, en 2013, le salaire net *moyen* des salariés à temps plein en France était de 2202 euros, tandis que le salaire net *médian* n'était que de 1772 euros. La différence étant due à des très hauts salaires qui "tirent" la moyenne vers le haut.

Indicateurs de dispersion

Les indicateurs de dispersion permettent de mesurer si les valeurs sont plutôt regroupées ou au contraire plutôt dispersées.

L'indicateur le plus simple est l'étendue de la distribution, qui décrit l'écart maximal observé entre les observations :

```
max(c$poids) - min(c$poids)
```

Les indicateurs de dispersion les plus utilisés sont la variance ou, de manière équivalente, l'écart-type (qui est égal à la racine carrée de la variance). On obtient la première avec la fonction `var`, et le second avec `sd` (abbréviation de *standard deviation*) :

```
var(c$poids)
```

```
sd(c$poids)
```

Plus la variance ou l'écart-type sont élevés, plus les valeurs sont dispersées autour de la moyenne. À l'inverse, plus ils sont faibles et plus les valeurs sont regroupées.

Une autre manière de mesurer la dispersion est de calculer les quartiles :

- le premier quartile est la valeur pour laquelle on a 25% des observations en dessous et 75% au dessus
- le deuxième quartile est la valeur pour laquelle on a 50% des observations en dessous et 50% au dessus (c'est donc la médiane)
- le troisième quartile est la valeur pour laquelle on a 75% des observations en dessous et 25% au dessus

On peut les calculer avec la fonction `quantile` :

```
quantile(c$poids)
```

`quantile` prend deux arguments principaux : le vecteur dont on veut calculer le quantile, et un argument `prob` qui indique quel quantile on souhaite obtenir. `prob` prend une valeur entre 0 et 1 : 0.5 est la médiane, 0.25 le premier quartile, 0.1 le premier décile, etc.

Notons enfin que la fonction `summary` permet d'obtenir d'un coup plusieurs indicateurs classiques :

```
summary(c$poids)
```

Représentation graphique

L'outil le plus utile pour étudier la distribution des valeurs d'une variable quantitative reste la représentation graphique.

Une représentation graphique efficace pour les variables continues est la boîte à moustache:

```
boxplot(c$poids)
```

Elle est d'autant plus efficace quand elle est présentée à l'horizontal grâce à l'argument `horizontal = TRUE`

```
boxplot(c$poids,
        horizontal = TRUE)
```

Les arguments de `boxplot` permettent également de modifier la présentation du graphique. On peut ainsi changer la couleur des barres avec `col` , le titre avec `main` , les étiquettes des axes avec `xlab` et `ylab` , etc. :

```
boxplot(c$poids,
        horizontal = TRUE,
        col = "grey",
        main = "Répartition des poids de scories de culot",
        xlab = "Poids (en grammes)")
```

> Note: ses arguments sont des arguments graphiques, ils sont compatibles avec toutes les fonctions graphiques…

La représentation la plus courante est sans doute l'histogramme. On peut l'obtenir avec la fonction `hist` :

```
hist(c$poids)
```

Cette fonction n'a pas pour effet direct d'effectuer un calcul ou de nous renvoyer un résultat : elle génère un graphique qui va s'afficher dans l'onglet *Plots* de RStudio.

On peut personnaliser l'apparence de l'histogramme en ajoutant des arguments supplémentaires à la fonction `hist` . L'argument le plus important est `breaks` , qui permet d'indiquer le nombre de classes que l'on souhaite.

```
hist(c$poids, breaks = 10)
```

```
hist(c$poids, breaks = 70)
```

Le choix d'un "bon" nombre de classes pour un histogramme n'est pas un problème simple : si on a trop peu de classes, on risque d'effacer quasiment toutes les variations, et si on en a trop on risque d'avoir trop de détails et de masquer les grandes tendances.

Ajoutons un peu d'arguments graphiques: de la couleur des barres avec `col` , un titre avec `main` , des étiquettes des axes avec `xlab` et `ylab` .

```
hist(c$poids, col = "skyblue",
     main = "Répartition des poids de scories de culot",
     xlab = "Poids",
     ylab = "Effectif")
```

Bonus: Il existe d'autres façon de gérer la couleur, nottament avec la fonction `ColorRampPalette` qui permet d'annoncer la palette de couleur que l'on va utiliser. Cette fonction accepte plusieurs forme de dénomination des couleurs: sous forme de texte (en anglais et entre “”), en rgv, en html ainsi qu'à partir de palettes pré-implentée dans R.

Par exemple on va déclarer une palette de dégradé (et la stocker dans un objet) allant du vert au rouge en passant par le jaune:

```
jolie_palette <- colorRampPalette(c("green","yellow","red"))(10)
```

l'argument utilisé est c(couleur de départ, couleur centrale, couleur d'arrivée) puis on ajoute le nombre de couleur a générer dans cette gamme. On aurait pu aussi identifier les couleurs avec du code html en s'aidant par exemple du site color brewer (http://colorbrewer2.org)

Nous pouvons donc l'utiliser dans notre histogramme:

```
hist(c$poids,
     col = jolie_palette,
     breaks = 70,
     main = "Répartition des poids de scories de culot",
     xlab = "Poids",
     ylab = "Effectif")
```

La fonction `hist` fait partie des fonctions graphique de base de R. Nous verrons par la suite l'extension `ggplot2` , qui fait partie du *tidyverse* et qui permet la production et la personnalisation de graphiques complexes sous R.

Analyser une variable qualitative

Une variable qualitative est une variable qui ne peut prendre qu'un nombre limité de valeurs, appelées modalités. Dans notre jeu de données on trouvera par exemple l'aspect (`aspect`), la forme (`forme`), la morphologie (`morphologie`)…

À noter qu'une variable qualitative peut tout-à-fait être numérique, et que certaines variables peuvent parfois être traitées soit comme quantitatives, soit comme qualitatives : ce n'est pas le cas dans ce jeu de données.

Tri à plat

L'outil le plus utilisé pour représenter la répartition des valeurs d'une variable qualitative est le *tri à plat* : il s'agit simplement de compter, pour chacune des valeurs possibles de la variable (pour chacune des modalités), le nombre d'observations ayant cette valeur. Un tri à plat s'obtient sous R à l'aide de la fonction `table` :

```
table(c$aspect)
```

Ce tableau nous indique donc que parmi nos enquêtés on trouve 38 culots d'aspect déchiqueté, 182 d'aspect irrégulier, 73 lisses et 253 réguliers.

Un tableau de ce type peut être affiché ou stocké dans un objet, et on peut à son tour lui appliquer des fonctions. Par exemple, la fonction `sort` permet de trier le tableau selon la valeur de l'effectif. On peut donc faire :

```
tab <- table(d$aspect)
sort(tab)
```

Attention, par défaut la fonction `table` n'affiche pas les valeurs manquantes (`NA`). Si on souhaite les inclure il faut utiliser l'argument `useNA = “always”` , soit : `table(c$us, useNA = “always”)` .

À noter qu'on peut aussi appliquer `summary` à une variable qualitative. Le résultat est également le tri à plat de la variable, avec en plus le nombre de valeurs manquantes éventuelles :

```
summary(c$aspect)
```

Représentations graphiques

On peut représenter graphiquement le tri à plat d'une variable qualitative avec un diagramme en barres, obtenu avec la fonction `barplot` .

Attention, contrairement à `hist` cette fonction ne s'applique pas directement à la variable mais au résultat du tri à plat de cette variable, calculé avec `table` . Il faut donc procéder en deux étapes :

```
tab <- table(d$aspect)
barplot(tab)
```

On peut aussi trier le tri à plat avec la fonction `sort` avant de le représenter graphiquement, ce qui peut faciliter la lecture du graphique :

```
barplot(sort(tab))
```

Une alternative au graphique en barres est le *diagramme de Cleveland*, qu'on peut obtenir avec la fonction `dotchart` . Celle-ci s'applique elle aussi au tri à plat de la variable calculé avec `table` .

```
dotchart(table(c$aspect))
```

Là aussi, pour améliorer la lisibilité du graphique il est préférable de trier le tri à plat de la variable avant de le représenter :

```
dotchart(sort(table(c$aspect)))
```

Exercices

Exercice (https://slides.com/archeomatic/aa181/#/3/7)
A la suite du script **Paleometallo.R**
Analyser la variable quantitative `longueur` du jeu de donnée `culot`

- Indicateurs de centralité
- Indicateurs de dispersion
- Représentations graphiques

```
l <- c$longueur
summary(l)
hist(l)
boxplot(l, horizontal = T)
```

Exercice (https://slides.com/archeomatic/aa181/#/3/8)
A la suite du script **Paleometallo.R**
Analyser la variable qualitative `longueur` du jeu de donnée `culot`

- Faire un tri à plat
- Y a t'il des données manquantes (NA)?
- Représentations graphiques

```
m <- c$morphologie
t <- table(m)
table(m, useNA = "always")
barplot(sort(t), las=2)
dotchart(sort(t))
```

Notez que toutes les étiquettes n'apparaissent pas sur le diagramme en barre. On peut utiliser l'argument `las = 2` (0 = étiquettes parrallèles aux graduations, 1= horizontales)