Prise en main du logiciel R

Préambule

Ce document présente les principaux éléments permettant de prendre en main le logiciel Rstudio. Les possibilités offertes par ce logiciel vont bien au-delà de ce qui est présenté ici. Nous aborderons uniquement le fonctionnement de Rstudio sous Windows et nous ne garantissons pas que les choses soient reproductibles sous Mac ou Linux.

Installation

- Etape 1 : installer R : https://cran.r-project.org/
- Etape 2 : installer Rstudio : https://posit.co/download/rstudio-desktop/ . Rstudio est une interface graphique qui rend plus agréable l'utilisation de R.

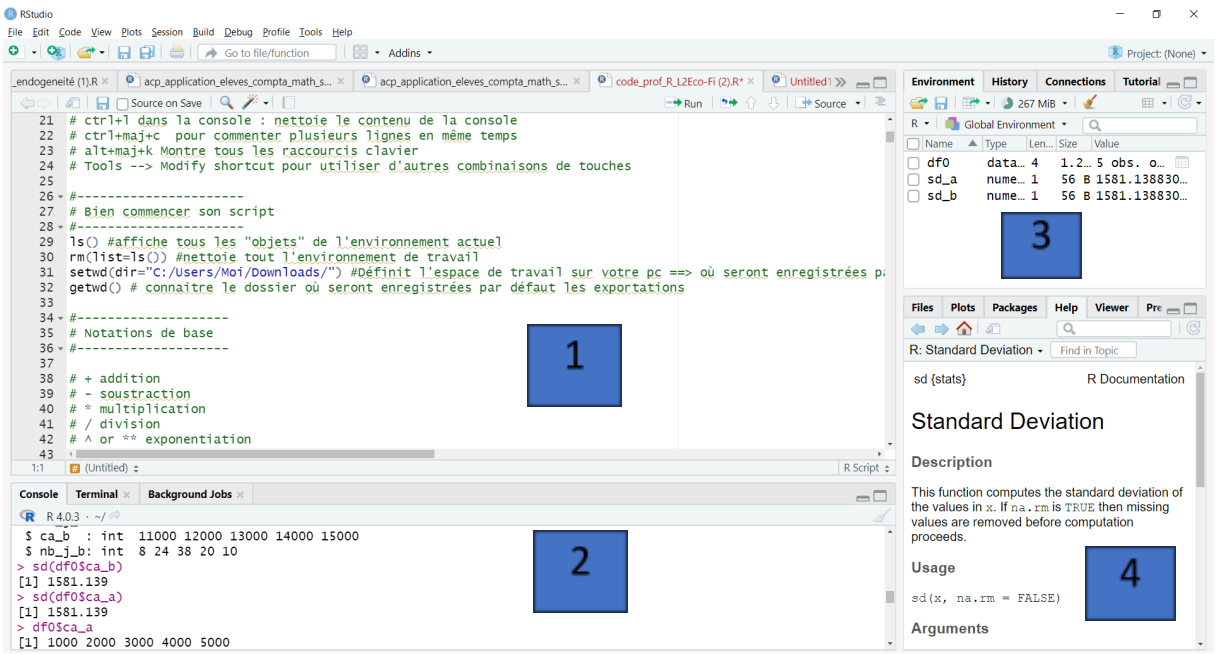
ATTENTION : Il est important de suivre ces deux étapes dans l'ordre. En effet, Rstudio ne fonctionne que si R "normal" est installé. Si vous avez des difficultés pour installer le logiciel, nous pourrons regarder cela rapidement ensemble lors de la première séance.

En dernier recours, vous pouvez éventuellement la version de R utilisable sur Google Colab (tutoriel introductif)

Présentation de l'interface Rstudio

L'interface de Rstudio est divisée en 4 grandes zones :

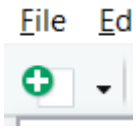
1. Editeur de code : fonctionne comme un traitement de texte
2. Console : c'est ici qu'apparaissent les résultats des instructions exécutées
3. Environnement de travail : apparaissent ici les informations sur les objets en mémoire
4. Affichage de l'aide sur les fonctions, affichage des graphiques créés, interface de gestion des *packages*



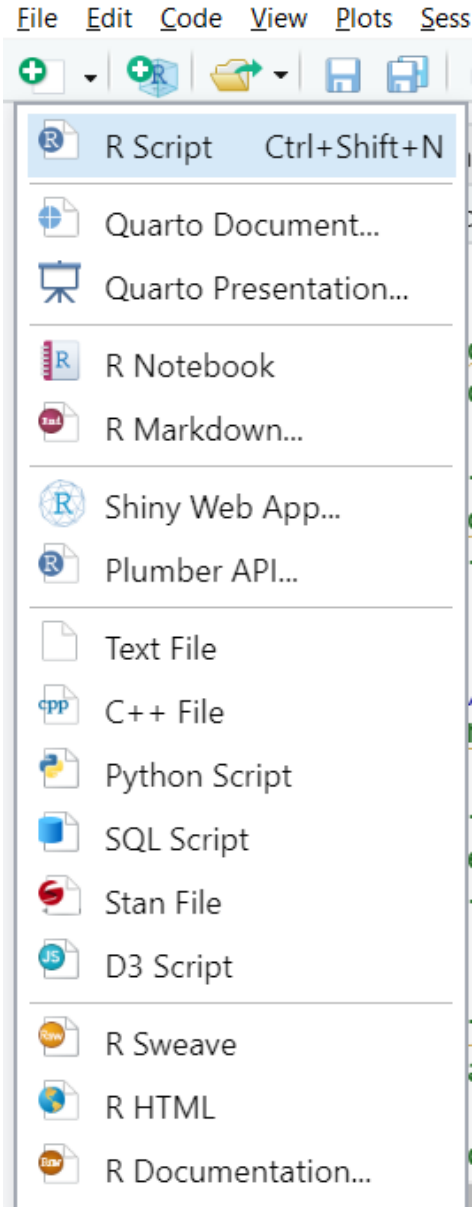
Premières lignes de code

Ouvrir un nouvel éditeur de code

En haut à gauche de l'interface, cliquer sur le « + » vert :

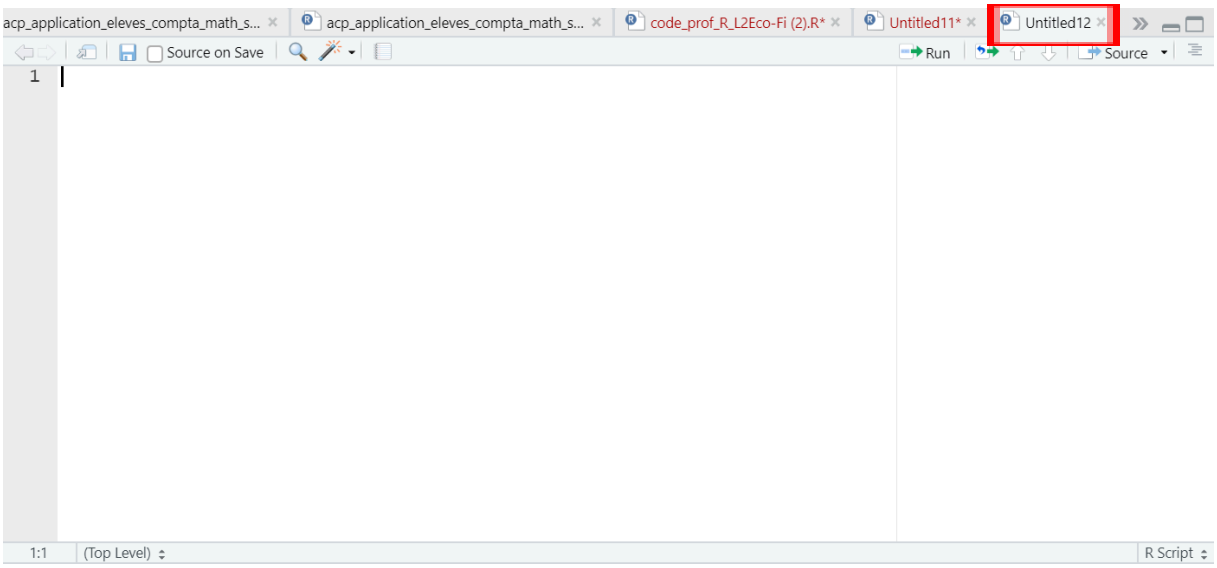


S'affiche alors une liste, dans cette liste sélectionner Rscript (il existe un raccourci clavier : Ctrl+Shift+N. NB : Shift correspond à la touche « Maj »)



Ouvrez votre premier éditeur de code

Dans la zone 1 vous allez alors obtenir une zone de traitement de texte vierge :



Par défaut, le fichier où vous allez saisir votre code se nomme « Untitled » suivi d'un numéro en fonction des fichiers sans noms déjà existants. Dans la capture d'écran ci-dessus, le fichier est nommé « Untitled12 ».

Nous vous invitons dans un premier temps à enregistrer ce document en lui donnant un nom explicite. De manière classique, pour enregistrer vous utiliser le raccourci clavier « Ctrl+s ». Vous pouvez également cliquer sur la disquette à droite du « + » de tout à l'heure :

Sauvegarder votre éditeur de code sur votre ordinateur en lui donnant un nom explicite, par exemple « mon_premier_code_R »

Bien commencer son code

Afin de pouvoir travailler sereinement, il est fortement recommandé de commencer votre code avec les lignes suivantes :

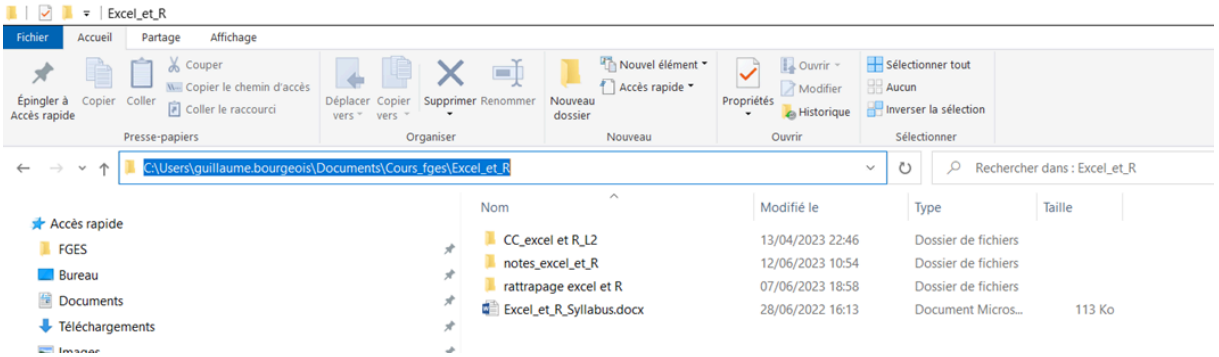
```
ls() #affiche tous les "objets" de l'environnement actuel

rm(list=ls()) #nettoie tout l'environnement de travail
```

La fonction `ls()` vous permet de savoir si vous avez encore des objets en mémoire dans votre session R. Si vous avez encore des objets en mémoire, il est préférable de les supprimer au début de votre session de travail afin d'éviter toute confusion. Pour nettoyer votre espace de travail, utilisez la commande `rm(list=ls())`.

Si vous le souhaitez, vous avez la possibilité de spécifier votre dossier de travail (*working directory - wd*), c'est-à-dire le dossier dans lequel votre travail sera enregistré par défaut. Pour cela, il faut utiliser la fonction `setwd()`. A l'intérieur des parenthèses, vous devez spécifier le chemin adéquat sur votre ordinateur en utilisant l'argument `dir` comme présenté ci-dessous. Par exemple, si je

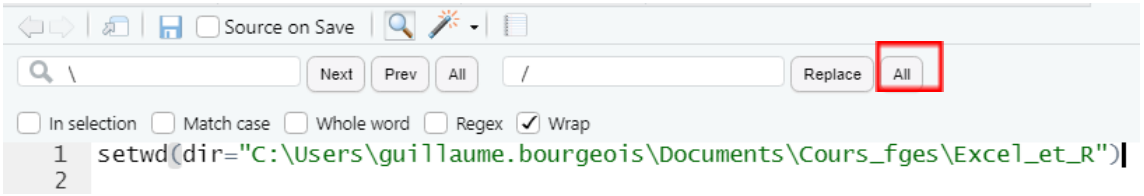
souhaite enregistrer mes travaux dans le dossier “R-L2” qui est sur mon bureau : `setwd(dir="c:/Users/guillaume.bourgeois/Desktop/R-L2/")`. Attention : ne recopiez pas “bêtement” ce code! Vous devez évidemment adapter ce chemin à l’arborescence de votre ordinateur (votre nom d’utilisateur n’est pas guillaume.bourgeois, sous mac le chemin ne commence pas par `"c:/"`, plus d’infos ici pour les utilisateurs de macOS). Voici quelques indications/astuces pour vous aider :

Astuce 1 : sous windows, pour connaître le chemin vers un dossier, ouvrez ce dossier et utilisez le raccourci ctrl+l, le chemin est alors surligné en bleu et vous avez juste à utiliser ctrl+c pour le copier :

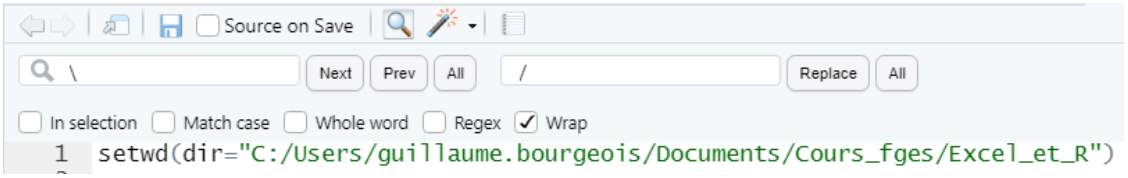


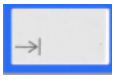
⇒ C:\Users\guillaume.bourgeois\Documents\Cours_fges\Excel_et_R

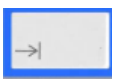
ATTENTION : windows utilise des “\” alors que R utilise des “/” il faut donc faire les modifications à la main ou bien utiliser l’outil *find and replace* dans R (ctrl+f) puis cliquer sur all :



On obtient alors :



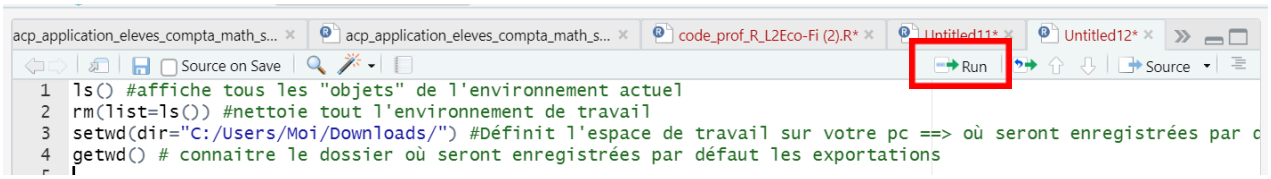
Astuce 2 : Quand vous êtes en train de saisir votre code sur R vous pouvez utiliser la touche tabulation   (en haut à gauche du clavier) et R vous donnera les dossiers disponibles.

- `setwd(dir="")`
- entre les guillemets écrire : c:/Users/ ⇒ `setwd(dir="c:/Users/")` si vous travaillez sous Windows. Si vous utilisez un Mac, écrire directement /Users/ ⇒ `setwd(dir="/Users/")`
- Appuyer sur la touche tabulation  

- R vous propose alors les dossiers disponibles ⇒ Sélection le dossier qui correspond à votre nom d'utilisateur sur votre ordinateur, en naviguant dans la liste avec les flèches du clavier, puis en appuyant sur entrée ⇒ `setwd(dir="c:/Users/guillaume.bourgeois/",` ou `setwd(dir="/Users/guillaume.bourgeois/")` si vous êtes sous Mac.
- Appuyer à nouveau sur la touche tabulation et ainsi de suite afin de naviguer dans l'arborescence de vos dossiers jusqu'à trouver celui où vous souhaitez enregistrer par défaut vos fichiers en lien avec votre travail sur R ⇒ `setwd(dir="c:/Users/guillaume.bourgeois/Desktop/R-L2/")`, ici pour l'exemple le dossier "R-L2" qui se trouve sur le bureau (Desktop en anglais)

La fonction `getwd()` fonctionne sans rien indiquer entre les parenthèses. Elle vous permet simplement de savoir quel est votre dossier de travail, cela apparaissant dans la console (zone 2).

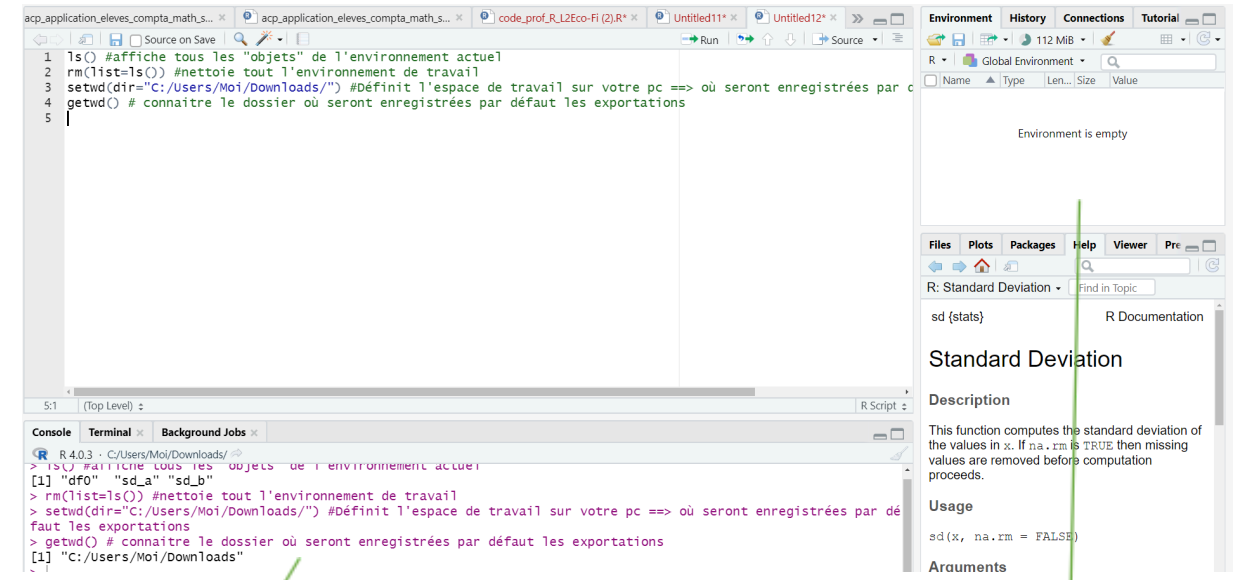
Pour exécuter votre code, vous devez vous positionner avec le curseur sur la ligne de code que vous souhaitez exécuter et appuyer sur Ctrl+Entrée. Pour exécuter toute la ligne vous n'avez pas besoin de la sélectionner toute entière, il suffit juste de positionner votre curseur n'importe où sur celle-ci. Vous n'êtes pas obligé d'utiliser ce raccourci clavier, vous avez un bouton « run » :



Les résultats de l'exécution du code apparaissent dans la console (zone 2)

Copier-Coller les lignes de codes dans l'encadré ci-dessus(p. 5) et exécuter ces lignes de code.

Votre interface doit alors ressembler à :



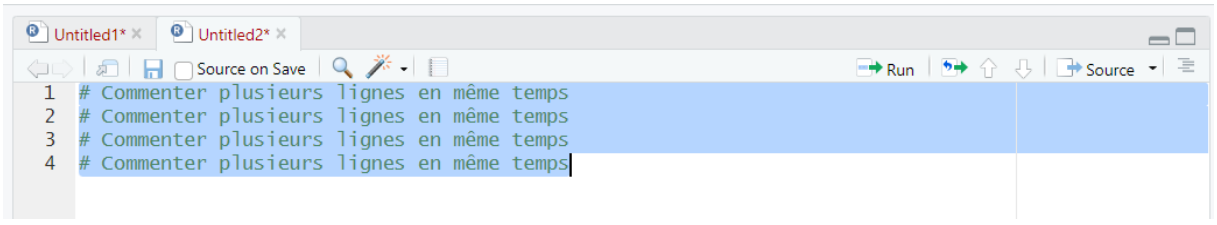
Les commandes que vous avez exécutées ainsi que leurs **éventuels** résultats apparaissent dans la console

Votre environnement de travail est vide. Vous n'avez rien en mémoire dans votre session de travail.

Commenter ses lignes de code :

Vous avez déjà pu observer dans ces premières lignes de code l'utilisation du symbole **"#".** Celui-ci permet d'introduire un commentaire dans votre code, c'est–à-dire du texte qui ne sera pas considéré comme du code par R. Cela est extrêmement utile pour décrire ce que vous êtes en train de faire, si par exemple vous devez reprendre votre code avec plusieurs jours sans l'avoir touché ou si vous travaillez sur un projet à plusieurs. Attention : le commentaire commence avec le "#", sur une ligne tout le texte placé avant sera bel et bien considéré comme du code par R.

Astuce : utiliser le raccourci clavier ctrl+shift+c pour mettre un # au début de la ligne. Vous pouvez faire cela sur plusieurs lignes en même temps après les avoir toutes sélectionnées :



Les objets dans Rstudio

Manipulations simples

Dans Rstudio, vous pouvez enregistrer dans votre session de travail des valeurs dans des objets. Cela peut être particulièrement utile lorsque vous souhaitez garder en mémoire un résultat, où lorsque vous aurez besoin de ces valeurs ultérieurement dans votre code.

Par exemple :

Rappel : # permet d'introduire un commentaire, pour R tout ce qui suit ce symbole ne doit pas être considéré comme tu code à exécuter

a=5 # crée un objet numérique : j'associe au nom d'objet "a" la valeur 5, observez que l'objet s'affiche dans votre environnement en haut à droite, avec différentes informations

a # il faut saisir le nom de l'objet pour l'afficher dans la console (en bas à gauche)

a*3 #Vous pouvez demander à R d'exécuter des calculs en utilisant directement le nom de l'objet

a=12 # ecrit par dessus, maintenant a n'est plus égal à 5 mais à 12

a # Affichage du contenu de l'objet dans la console

rm(a) # supprimer l'objet "a" de votre environnement de travail (n'apparaît plus en haut à droite). Vous utilisez ici la fonction `rm()`

a=24 # on peut recréer un objet a, associé à une nouvelle valeur

A # **R est sensible aux majuscules** : a est différent de A, cela vaut pour l'ensemble du code (nom des fonctions, etc...). Dans la console vous voyez alors "Error: object 'A' not found"

b="Guillaume" #objet caractère

b

Copier-Coller les lignes de codes dans l'encadré ci-dessus et exécuter ces lignes de code. Prenez soin d'observer ce qu'il se passe dans votre zone « Environnement de travail » et dans la console.

⇒ il existe différents types de données (textuelles, numériques, integer, booléennes, logique,...) (plus de détails)

Fonction c()

La fonction `c()` est l'une des fonctions les plus importantes dans Rstudio, si ce n'est la plus importante. Ses applications sont multiples et nous l'utilisons tout le temps !

Dans cette sous-section, nous allons uniquement présenter son utilisation pour la création d'objets. Nous aborderons par la suite quelques-unes de ses autres utilisations.

Le nom de cette fonction comporte uniquement la lettre « c », première lettre du mot « concaténer »[1]. Cette fonction vise donc à regrouper plusieurs valeurs. Elle va ainsi nous permettre de créer des vecteurs de valeurs.

Par exemple :

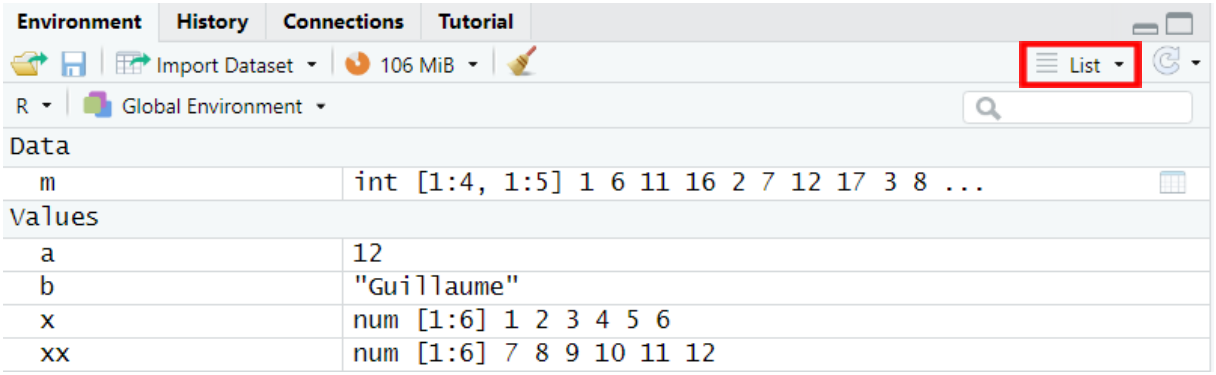
```
x=c(1,2,3,4,5,6) # vecteur de valeurs, remarquez ici que nous créons un vecteur de valeurs numériques, il est donc ici inutile (et malvenu) d'utiliser des guillemets. L'objet s'appellera x, vous être libre de lui associer un autre nom

x # Il faut saisir et exécuter le nom de l'objet pour qu'il s'affiche dans la console


xx= c(7,8,9,10,11,12) # vecteur de valeurs

xx # Il faut saisir et exécuter le nom de l'objet pour qu'il s'affiche dans la console


x=c(x, -1, -2, -3, -4, -5, -6) # Je peux venir modifier un vecteur déjà existant, en lui ajoutant des éléments par exemple. En gros ic je dis à R : je veux que la nouvelle version du vecteur x, soit construite en partant de la version actuelle, à laquelle j'ajoute les valeurs -1, -2, -3, -4, -5 et -6.

x

x_xx=c(x, xx) # Je peux concaténer deux vecteurs de valeurs déjà existants


length(x) # donne la longueur du vecteur x

length(xx) # donne la longueur du vecteur xx

length (x_xx) # donne la longueur du vecteur x_xx


# Nous pouvons également créer des matrices

m=matrix(1:20, # valeur à inclure dans la matrice, ici tous les entiers entre 1 et 20
```

[1] Selon le Larousse, concaténer signifie : " Dans certains langages de programmation, enchaînement de deux listes ou de deux chaînes de caractères mises bout à bout." (source)

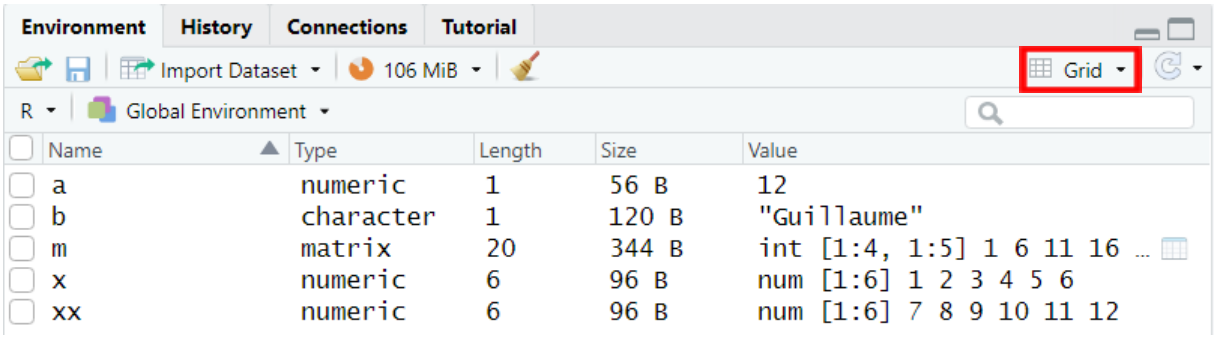
```
          ncol=5, # nombre de colonnes
          nrow=4, # nombre de lignes
               byrow=TRUE) #"sens" de lecture, les valeurs seront
ordonnées par ligne
m # Afficher la matrice dans la console
```

Copier-Coller les lignes de codes dans l'encadré ci-dessus et exécuter ces lignes de code. Prenez soin d'observer ce qu'il se passe dans votre zone « Environnement de travail » et dans la console.

Voici ce qui doit s'afficher dans votre environnement de travail :



Vous avez la possibilité de changer le mode d'affichage en cliquant sur list (encadré rouge) et passer en mode grid :



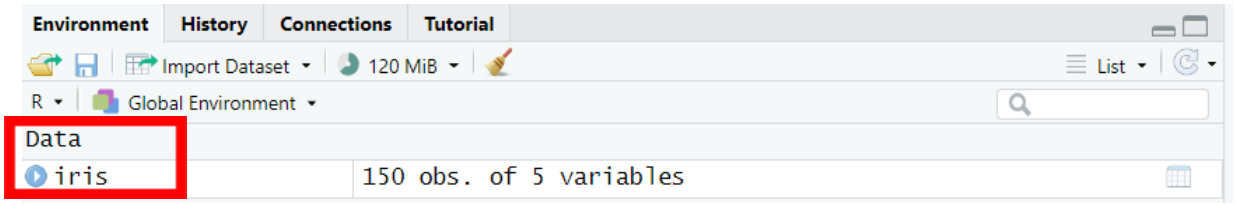
⇒ FAIRE LE PREMIER EXERCICE (lien), vous serez invité.e à réaliser l'exercice 2 plus tard, poursuivez la lecture du document

Data frames

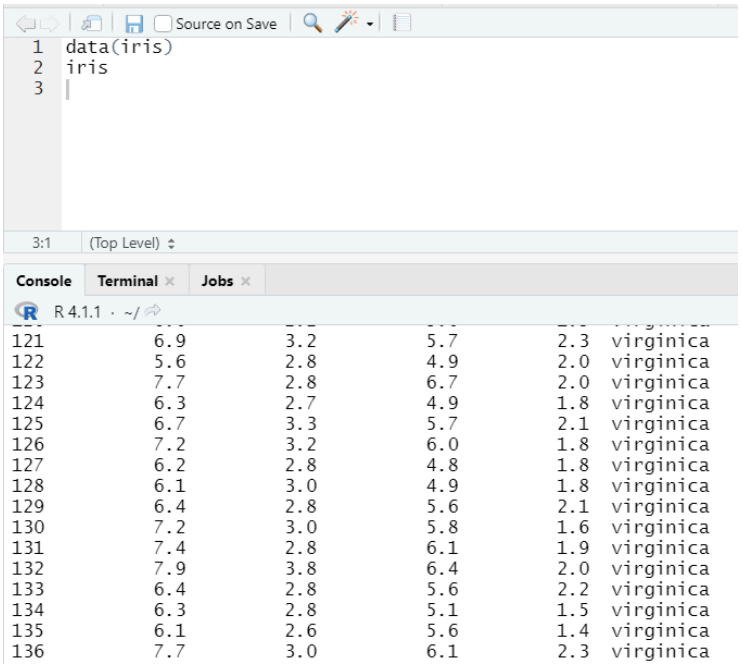
Pour R, un data frame est un ensemble de vecteurs. Dit plus simplement, il s'agit d'un tableau de données en deux dimensions. Ainsi, **le jeu de données iris** préchargé dans R est pour le logiciel un data frame, car il s'agit d'un tableau en deux dimensions où les lignes représentent les individus (fleurs) et les colonnes représentent les caractéristiques de ces individus. La liste des jeu de données préchargés dans R est visible avec la fonction `data()`.

Une fois que vous aurez appelé ce jeu de données, il apparaîtra bien comme un jeu de données pour R :

```
data("iris") # Attention :  iris est ici le nom du data frame à importer, on pourrait aussi faire data(CO2), car CO2 est un autre jeu de données préchargé, ou encoreexécuter data() pour avoir la liste complète des jeux de données préchargés dans R.


iris


?iris # "aide" sur le jeu de donnée : informations notamment sur la signification des variables notamment
```

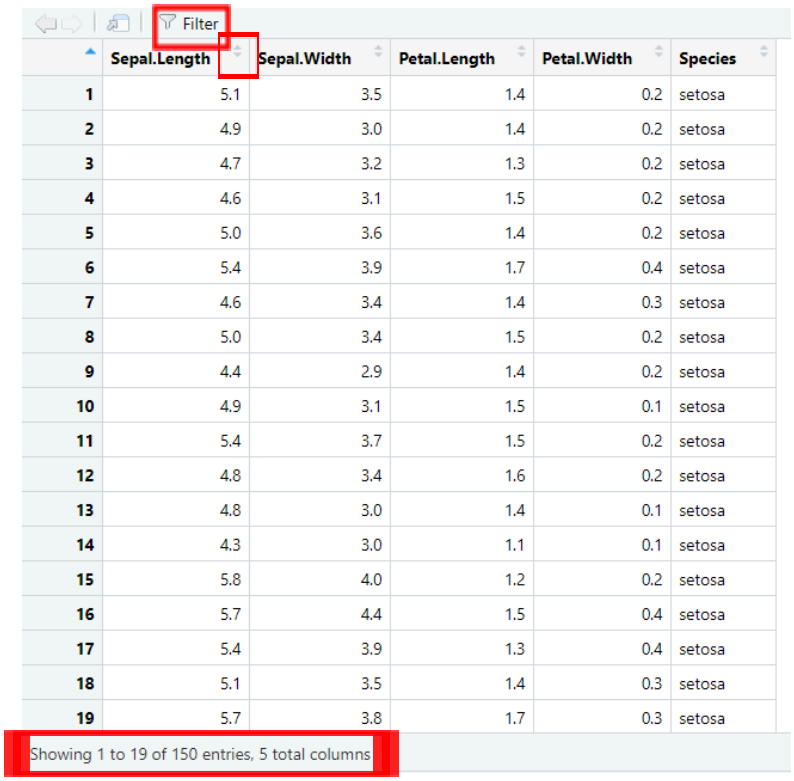


Afin d'afficher le contenu d'un data frame vous pouvez simplement écrire son nom (ici : iris) et exécuter cette commande. Le data frame s'affiche alors dans la console :



Néanmoins, ce mode de visualisation n'est pas très adapté à partir du moment où le jeu de données comporte un certain nombre de lignes et de colonnes. Dans ce cas, il est plutôt conseillé d'utiliser une nouvelle fenêtre de visualisation de données. Pour ouvrir une telle fenêtre vous devez utiliser la fonction `View()` (faîtes attention au V majuscule : R est TOUJOURS sensible aux majuscules).

```
data("iris")

iris

View(iris)
```



Filter

	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

Showing 1 to 19 of 150 entries, 5 total columns

Les flèches à côté des entêtes de colonne vous permettent de ranger les valeurs de la variable par ordre croissant ou décroissant. R affiche également en bas les dimensions du jeu de données (ici 150 lignes et 5 colonnes). Enfin, vous pouvez filtrer les valeurs affichées en cliquant sur l'icône idoine.

Fonction data.frame()

Dans R, grâce à la fonction `data.frame()`, vous pouvez construire votre propre tableau de données. Par exemple, si vous voulez créer un jeu de données comportant la taille (en pouces) et le poids (en livres) d'un échantillon de femmes :

```
femmes = data.frame(

taille = c(58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71,
72),

poids = c(115, 117, 120, 123, 126, 129, 132, 135, 139, 142, 146,
150, 154, 159, 164)

)


femmes # Il faut saisir le nom de l'objet et exécuter pour afficher
son contenu dans la console
```

Obtenir des informations sur les vecteurs et les data frames

Une fois le data frame importé / créé dans votre environnement de travail, certaines fonctions vous permettent d'obtenir des informations basiques sur celui-ci :

```
data(iris)
iris


class(iris) # type d'objet
dim(iris) # dimensions : nombre de lignes et de colonnes
ncol(iris) # nombre de colonnes
nrow(iris) # nombre de lignes
colnames(iris) # nom des colonnes
row.names(iris) # nom des lignes
head(iris, 10) # affiche les 10 premières lignes du jeu de données


str(iris) # structure du jeu de données ==> dimensions, type des variables


table(is.na(iris)) # vous indique si le jeu de données comporte des valeurs manquantes. Nous reviendrons ultérieurement sur la fonction table()


summary(iris) # Donne des informations basiques sur chacune des variables du jeu de données, les informations fournies dépendent évidemment du type de variable : vous ne pouvez pas calculer une moyenne sur une variable textuelle par exemple
```

Utilisation des crochets []

Pour tout utilisateur de R, il est fondamental de savoir utiliser parfaitement les crochet `[]`. Ils servent à extraire une partie d'un objet, que celui-ci soit en une dimension (vecteur) ou en deux dimensions (data frame, matrice).

Imaginez que vous ayez la liste de noms des enfants d'une famille, de l'aîné au plus jeune :

```
nom_enfants=c("Gabriel","Justine","Jean","Laura")    #création    du
vecteur de valeurs

# Si je veux connaître le prénom du cadet :

nom_enfants[2]


# Si je veux connaître le prénom de l'aîné et du cadet :

nom_enfants[c(1,2)] # on remarque ici l'utilisation de la fonction
c() à l'intérieur des crochets afin d'extraire plus d'un élément


# Si je veux connaître le prénom de l'aîné et du plus jeune :

nom_enfants[c(1,4)]


# Si je veux extraire tout le vecteur SAUF le deuxième élément, en créant un nouvel objet
"nom_enfants_court" :

nom_enfants_court=nom_enfants[-2]


# Si je veux extraire tout le vecteur SAUF le 2e et le 3e élément

nom_enfants[-c(2,3)] # On doit placer le signe "-" avant le c
```

Dans le cas d'un objet en deux dimensions, nous devons indiquer à R quelle ligne et quelle colonne nous souhaitons extraire. L'indice de ligne et l'indice de colonne sont séparés par une virgule au sein des crochets : `nom_data_frame[indice de ligne,indice de colonne]`. Par exemple, pour le jeu de données iris (qui est préchargé dans R) :

```
data("iris")

iris

iris[1,3]  # va afficher la valeur à l'intersection entre la
première ligne et de la troisième colonne
```

Il est possible d'extraire toute une ligne, en ne précisant pas d'indice de colonne après la virgule. De même, il est possible d'extraire toute une colonne en ne précisant aucun indice de ligne avant la virgule. Par exemple, toujours sur le data frame `iris` :

```
data("iris")
```

```
iris

iris[2,] # extrait les valeurs de la deuxième ligne (et de toutes les colonnes)


iris[,4] # extrait les valeurs de la quatrième colonne (et de toutes les lignes
```

Enfin, vous pouvez extraire plusieurs éléments grâce à la fonction `c()` et également demander à R d'extraire tout l'objet hormis l'élément qui se trouve à une certaine position, une certaine ligne, une certaine colonne. Par exemple :

```
data("iris")

iris

iris[c(2,4),2] # extrait l'intersection entre [les lignes 2 et 4] avec la [colonne 2]

iris[,c(1,2)] # extrait les deux premières colonnes (et toutes les lignes)


iris[-1,] # Extrait tout le data frame SAUF la première ligne

iris[-c(1,3),] # Extrait tout le data frame SAUF la première ligne et la troisième ligne
```

Astuce : Si par exemple vous souhaitez extraire les lignes 2,3,4,5,6 du data frame `iris`, au lieu de saisir `iris[c(2,3,4,5,6),]` vous pouvez utiliser les ":" `iris[2:6,]`

Utilisation du symbole du dollar

Le symbole "$" n'est pas du tout utilisé de la même manière dans Excel et dans Rstudio. Néanmoins, son utilisation dans Rstudio est tout aussi fondamentale et vous devez parfaitement utiliser l'utilisation de ce symbole.

Dans Rstudio, le $ permet de faire référence à /extraire une variable dans un data frame à partir de son nom. La fonction `colnames()` vous donne le nom des variables d'un data frame.

```
data("iris")

iris

colnames(iris)

iris$Species
```

⇒ FAIRE L'EXERCICE 2 (lien)

Importation de données

Séance 2 : REPRENDRE LA LECTURE ICI

L'importation des données est très souvent le préalable à la réalisation d'une étude statistique avec un logiciel tel que Rstudio. Il est donc important de maîtriser cette partie, d'en comprendre les subtilités et de pouvoir vérifier si une importation s'est bien passée.

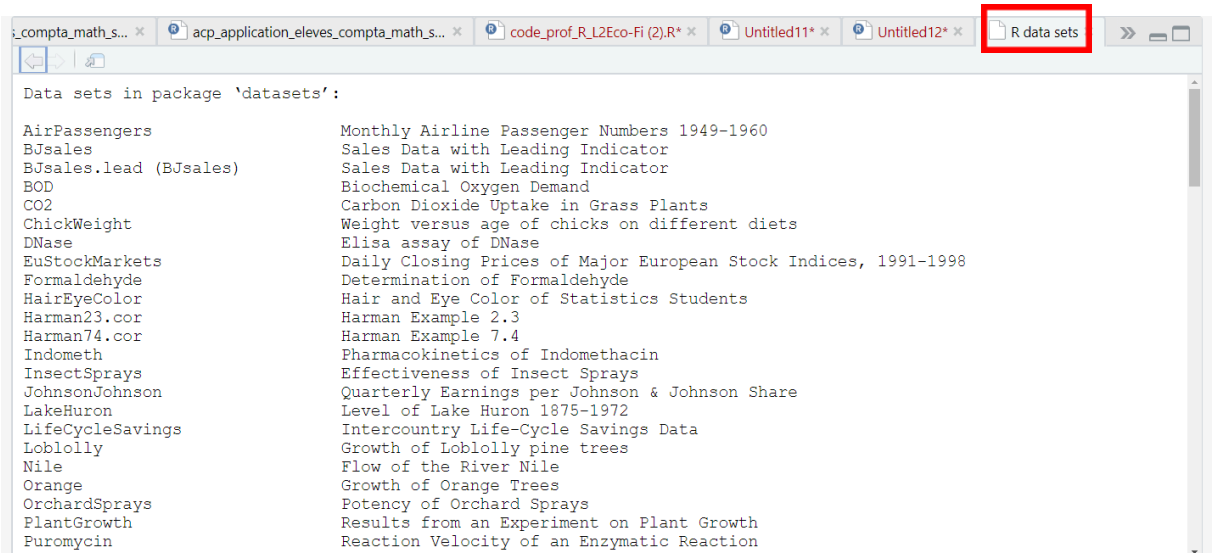
Importation des données pré-enregistrées dans Rstudio

En réalité, lorsque vous ouvrez Rstudio, vous n'ouvrez pas une coquille vide, des données sont par défaut préchargées dans votre logiciel. Pour qu'elles apparaissent dans votre environnement de travail il faut simplement les activer.

La liste des jeux de données préchargés ainsi qu'une description succincte de chacun d'eux est disponible en exécutant la commande `data()`, en n'écrivant rien entre les parenthèses.

Exécuter la commande `data()`

Une nouvelle fenêtre nommée « R data sets » va alors s'ouvrir :



ATTENTION : vous avez quitté votre éditeur de code (qui se nomme toujours Untitled12 dans la capture d'écran ci-dessus).

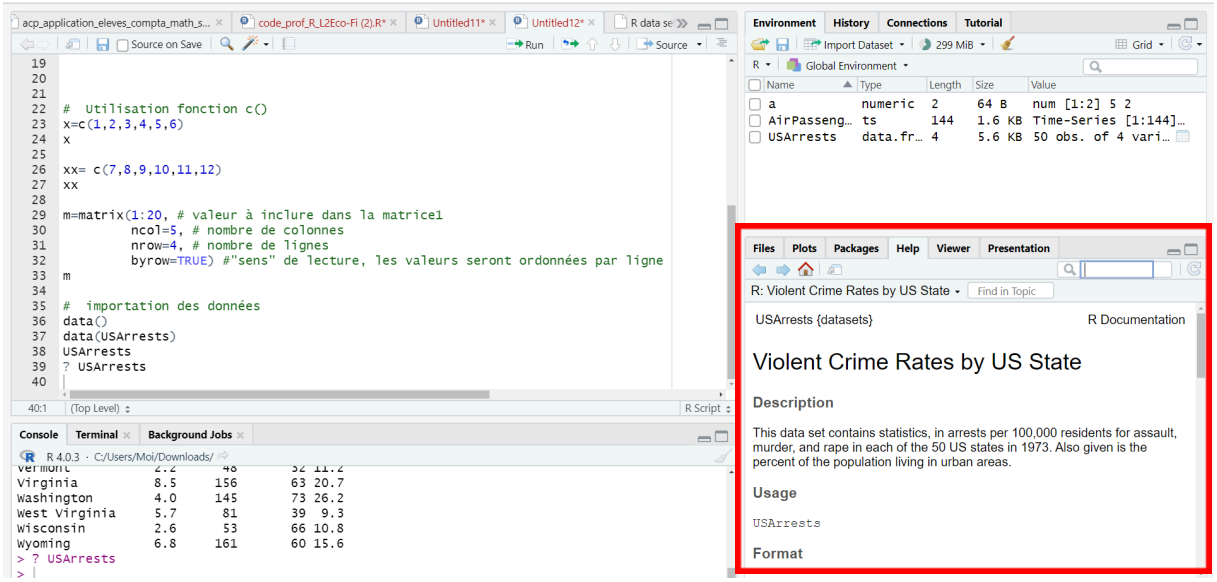
Un de ces jeux de données se nomme USArrests (notez l'importance des majuscules pour R). Activez ce jeu de données pour qu'il apparaisse dans votre espace de travail en exécutant :

```
data(USArrests)

USArrests
```

L'intérêt de ces jeux de données préchargés est qu'ils disposent d'une description détaillée de leur contenu. Pour afficher cette description, exécuter la commande :

```
? USArrests
```

Les informations apparaissent alors en bas à droite dans la fenêtre dédiée à l'aide (zone 4).



Importation depuis un lien internet

Vous pouvez évidemment importer des données qui ne sont pas préchargées dans Rstudio. C'est d'ailleurs ce que vous ferez la plupart du temps. Ces données peuvent être stockées sur un support physique (votre propre ordinateur, une clé USB, …) ou bien être stockées en ligne. Nous verrons dans un premier temps comment importer des données stockées en ligne.

Les données que nous allons tenter d'importer dans Rstudio sont disponibles avec le lien suivant :

https://raw.githubusercontent.com/guillaume-bourgeois92/ACP/main/ex_notes_compta_compta_stats_gfi_pour_import.csv

Ce lien fournit déjà quelques informations :

- Les données sont stockées sur GitHub, ce qui ne posera aucun problème pour l'importation dans Rstudio
- Le fichier se nomme « ex_notes_compta_compta_stats_gfi_pour_import.csv ». Il est donc au format csv (format textuel, avec un séparateur point-virgule).

De manière générale, il est préférable que les données que vous souhaitez importer dans Rstudio soient au format .csv ou .txt, cela limitant considérablement les potentiels problèmes d'importation.

R peut importer des données dans d'autres formats que .csv ou .txt mais nous ne verrons pas ces fonctions dans le cadre du cours ⇒ plus d'informations ici

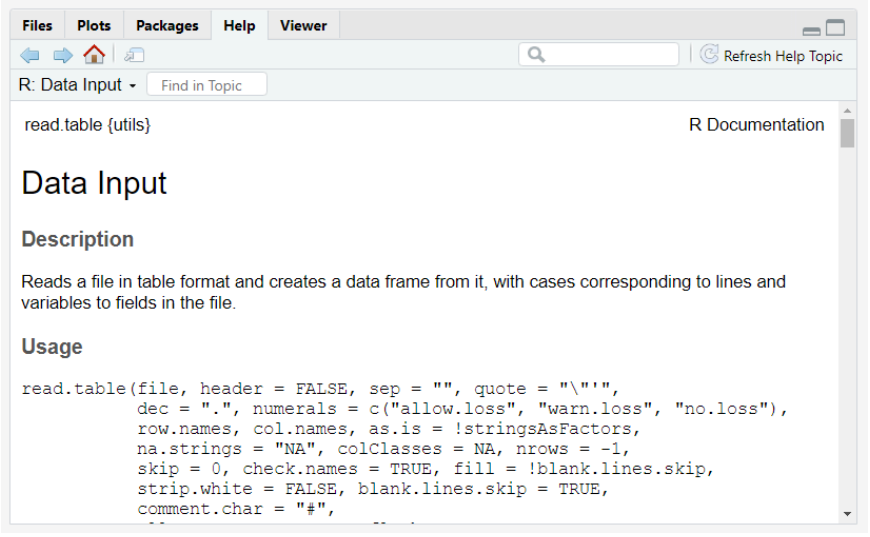
Cliquer sur le lien ci-dessus et visualiser les données

Voici ce que vous devez visualiser, remarquez les points-virgules afin de séparer les différentes valeurs

```
Individu;Statistiques;Math;Compta;Gestion_fi;Moyenne;ecart-type
Ind1;19;14;8;18;11.50;4.32
Ind2;20;12;4;4;9.00;6.63
Ind3;10;10;32;38;13.75;12.68
Ind4;13;17;4;4;8.50;5.68
Ind5;6;8;26;24;9.75;9.06
Ind6;6;3;28;32;9.75;12.87
Ind7;19;16;8;20;12.25;4.71
Ind8;15;18;6;6;9.75;5.36
Ind9;9;2;32;30;10.50;13.01
Ind10;8;7;20;20;8.75;6.26
```

Pour importer des données en format csv nous utilisons la fonction `read.csv2()`. R a la particularité de vous fournir une aide sur chacune des fonctions. Par exemple, afin d'accéder à la l'aide sur la fonction `read.csv2`, vous devez saisir et exécuter le code `?read.csv2`. L'aide apparaît alors dans la zone d'aide, en bas à droite. Vous pouvez également chercher de l'aide directement sur internet, il existe des ressources francophones (https://thinkr.fr/abcdr/ ; http://www.sthda.com/french/wiki/logiciel-r ; https://forums.cirad.fr/logiciel-R/) et anglophones (https://stackoverflow.com/questions/tagged/r). Vous trouverez également facilement une multitude de documents de référence au format .pdf en ligne ainsi que des tutos sur YouTube.

Lorsque vous exécutez `?read.csv2` l'aide apparaît en bas à droite (zone 4) :

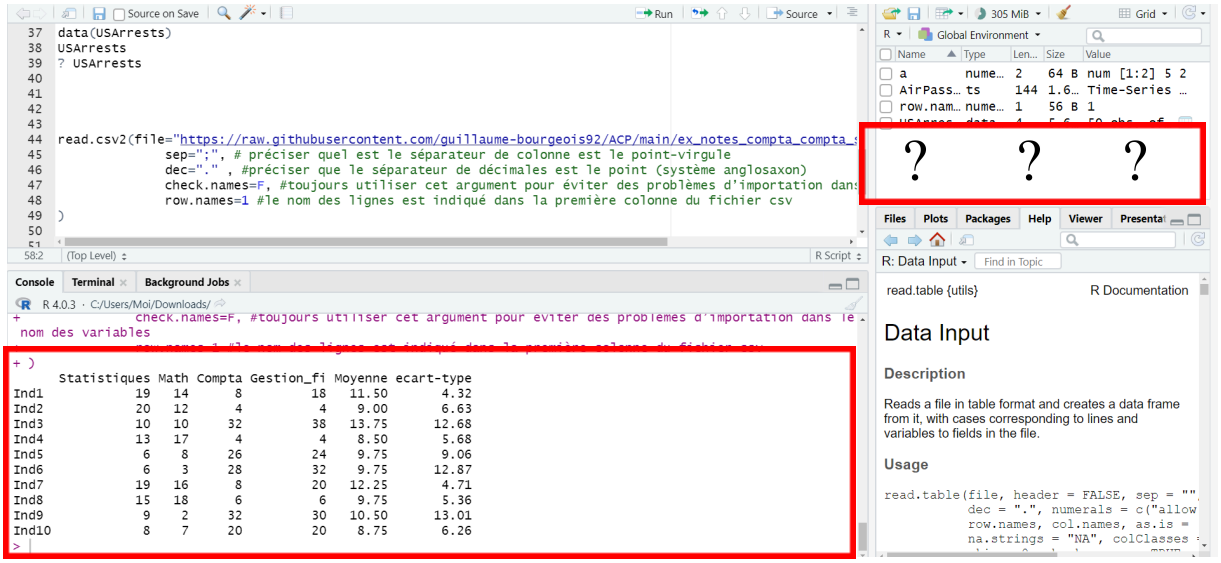


Voici le code qui vous permet d'importer les données stockées sur Github au format csv. Vous remarquerez l'utilisation de plusieurs arguments dans cette fonction (`file`, `sep`, `dec`, `check.names`, `row.names`). Ces arguments sont séparés par une virgule (et non par un point-virgule comme dans Excel). Chaque ligne comporte un commentaire (après le #) pour donner l'utilité de chaque argument

```
read.csv2(file="https://raw.githubusercontent.com/guillaume-bourgeois92/ACP/main/ex_notes_compta_compta_stats_gfi_pour_import.csv",  # on précise dans l'argument file le lien où sont stockées les données

                sep=";", # préciser que le séparateur de colonne est le point-virgule

                 dec="." , #préciser que le séparateur de décimales est le point (système anglosaxon)

                 check.names=F, #toujours utiliser cet argument pour éviter des problèmes d'importation dans le nom des variables

                 row.names=1 #le nom des lignes est indiqué dans la première colonne du fichier csv, ne pas utiliser cet argument si ce n'est pas le cas!!

               )
```

⍰ Copier-Coller les lignes de codes dans l'encadré ci-dessus et les exécuter. Prenez soin d'observer ce qu'il se passe dans votre zone « Environnement de travail » et dans la console. Attention à bien tout sélectionner, et ne pas oublier la dernière parenthèse sur la dernière ligne!

Problème : les données apparaissent dans la console mais ne sont pas sauvegardées dans votre espace de travail :

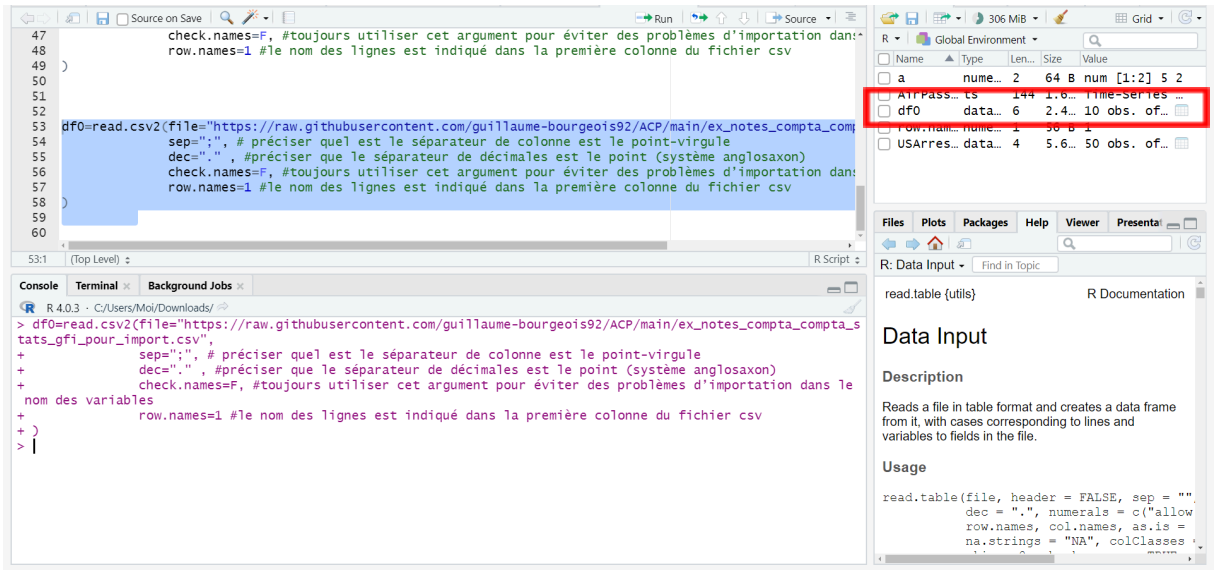


C'est normal, il faut préciser lors de l'importation que vous souhaitez créer un objet dans votre espace de travail où seront enregistrées les données que vous importez. Appelons par exemple cet objet `df0` (le reste du code reste inchangé) :

```
df0=read.csv2(file="https://raw.githubusercontent.com/guillaume-bourgeois92/ACP/main/ex_notes_compta_compta_stats_gfi_pour_import.csv"
,
```

```
                sep=";", # préciser quel est le séparateur de colonne
est le point-virgule

                 dec="." , #préciser que le séparateur de décimales
est le point (système anglo saxon)

                check.names=F, #toujours utiliser cet argument pour
éviter des problèmes d'importation dans le nom des variables

                 row.names=1 #le nom des lignes est indiqué dans la
première colonne du fichier csv

)
```

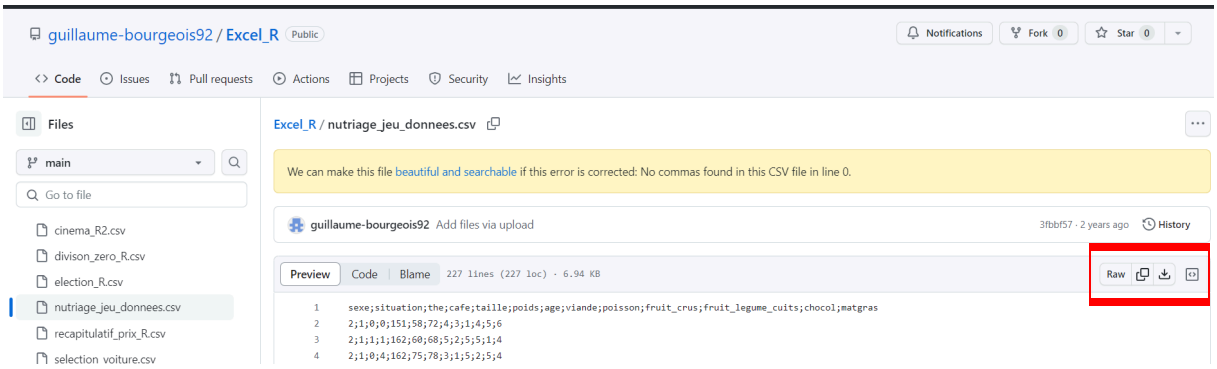
Le jeu de données apparaîtra alors bien dans votre environnement de travail. Pour qu'il s'affiche dans la console il suffira de saisir son nom (comme vu plus tôt pour afficher un objet simple) :



Importation depuis votre ordinateur

L'importation depuis votre ordinateur suit le même principe que celle avec un lien internet. La différence est qu'à la place d'utiliser un lien url vous devez préciser le chemin sur votre ordinateur dans l'argument `file`.

Afin d'illustrer l'importation d'un jeu de données enregistré sur votre ordinateur je vais utiliser le fichier disponible avec ce lien et que vous pouvez télécharger en cliquant sur la flèche qui pointe vers le bas :

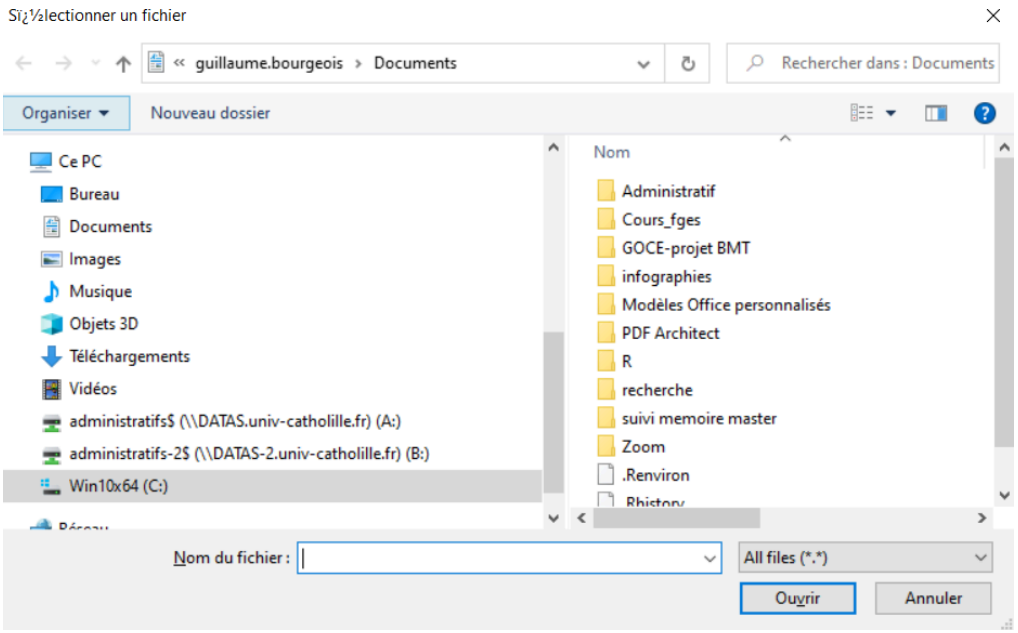


Par défaut, ce fichier nommé nutriage_jeu_donnees.csv est enregistré dans mon dossier de téléchargements : “c:\Users\guillaume.bourgeois\Downloads\”. Pour l’importer dans R j'exécute donc la commande :

```
df1=read.csv2(file="c:/Users/guillaume.bourgeois/Downloads/nutriage_jeu_donnees.csv",
              sep=";", # préciser quel est le séparateur de colonne est le point-virgule
                dec="." , #préciser que le séparateur de décimales est le point (système anglo saxon)
                 check.names=F #toujours utiliser cet argument pour éviter des problèmes d’importation dans le nom des variables
              )
```

Remarque

Au lieu de préciser le chemin sur votre ordinateur, vous pouvez utiliser l’instruction `file.choose()` en remplacement de l’argument file. R ouvre alors une nouvelle fenêtre qui vous permet d’explorer vos dossiers et d'aller sélectionner le document que vous souhaitez importer :




```
df1=read.csv2(file.choose(),
              sep=";", # préciser quel est le séparateur de colonne est le point-virgule
                dec="." , #préciser que le séparateur de décimales est le point (système anglo saxon)
                 check.names=F, #toujours utiliser cet argument pour éviter des problèmes d’importation dans le nom des variables
```

```
)
```

Vérifications de l'importation

Bien que dans la majorité des cas l'importation des données se passe bien, il est possible que vous rencontriez quelques difficultés une fois les données importées. Cela peut être lié au fait que Rstudio n'arrive pas à lire certaines données et les remplace par des *NAs* (*Not available*), que le logiciel ne reconnait pas le bon format de variable (une variable numérique qui n'est pas comprise comme telle par le logiciel), etc…

Ainsi, une bonne habitude à prendre est de systématiquement vérifier vos données une fois qu'elles ont été importées.

Les lignes de code suivantes vous permettent de vérifier en détails votre importation (données importées enregistrées dans l'objet `df0`) . L'argument df0 sera évidemment à adapter selon le nom de votre jeu de données dans votre environnement de travail.

```
dim(df0) # dimensions : nombre de lignes et de colonnes

colnames(df0) # nom des colonnes

row.names(df0) # nom des lignes

head(df0, 10) # affiche les 10 premières lignes du jeu de données


str(df0) # structure du jeu de données ==> dimensions, type des variables, lettre est une variable caractère et nombre une variable de type integer càd des nombres strictement entiers


table(is.na(df0)) # vous indique si le jeu de données comporte des valeurs manquantes


summary(df0) # Donne des informations sur chacune des variables du jeu de données, les informations fournies dépendent évidemment du type de variable : vous ne pouvez pas calculer une moyenne sur une variable textuelle par exemple
```

 Copier-Coller les lignes de codes dans l'encadré ci-dessus et exécuter ces lignes de code : l'importation des données depuis internet s'est-elle bien déroulée ?

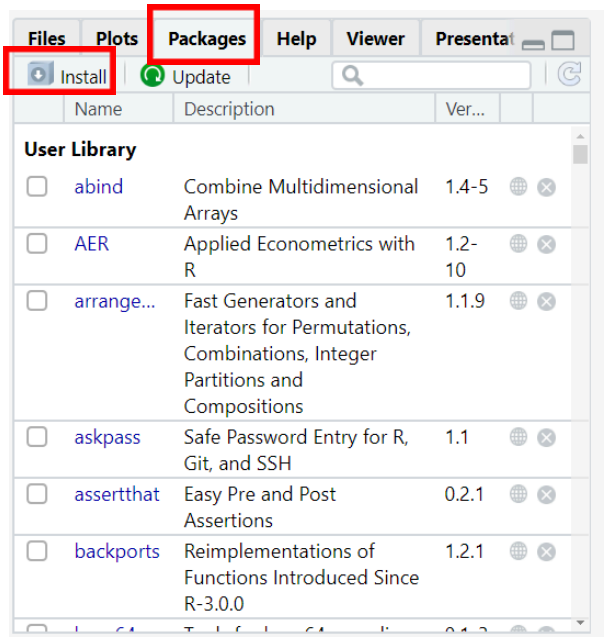
FAIRE LES EXERCICES 3,4,5,6

Packages dans Rstudio

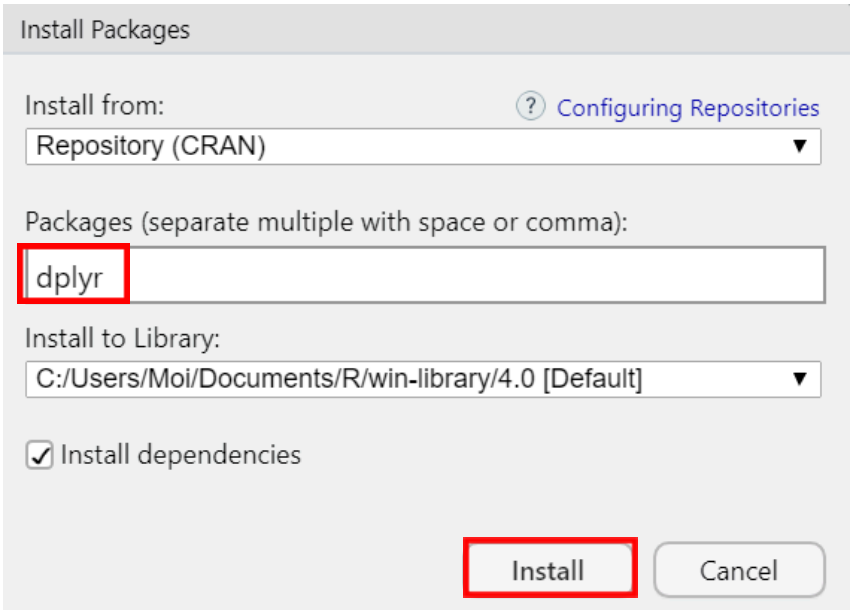
Séance 3 : (reprendre la lecture ici pour la séance 3)

Lorsque vous installez Rstudio sur votre ordinateur, un grand nombre de fonctions dites “de base” sont déjà installées et paramétrées. Néanmoins, pour certaines analyses statistiques comme par exemple construire des graphiques plus jolis ou manipuler efficacement des jeux de données, vous pouvez avoir besoin de fonctions qui ne sont pas présentes par défaut dans le logiciel.

Rstudio a l’avantage d’être un logiciel libre de droits où la communauté d’utilisateurs a la possibilité de développer des fonctions supplémentaires et de les mettre à la disposition de tous les utilisateurs. Ces fonctions supplémentaires, créées par la communauté, peuvent être importées dans le logiciel via les *packages*. Pour importer un *package* vous devez l’installer. L’installation de *packages* se fait via l’interface dédiée dans Rstudio (zone 4, en bas à droite) où vous devez dans un premier temps sélectionner l’onglet *Packages*, puis cliquer sur « *install* » :



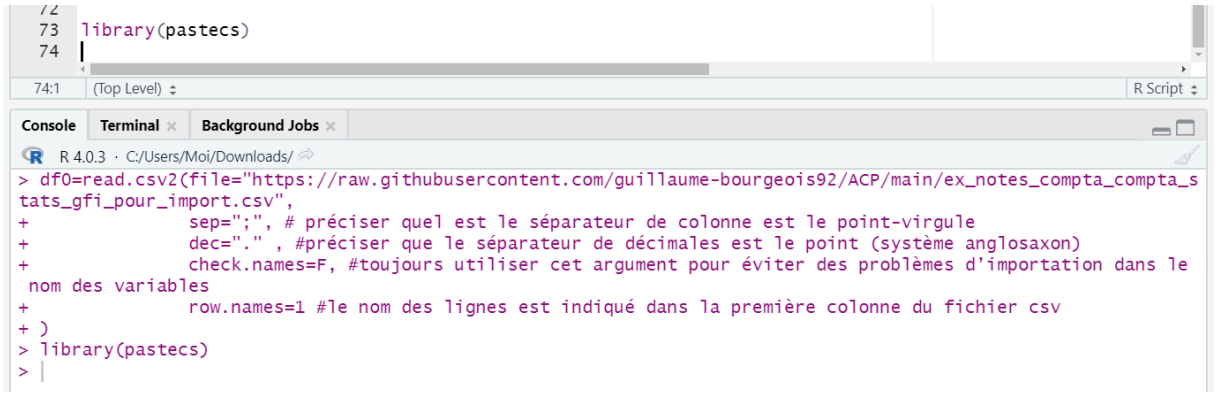
Une nouvelle fenêtre apparaît alors, où vous pouvez rechercher le package avec son nom, puis l’installer. La capture ci-dessous donne un exemple pour le package `dplyr` qui fournit des fonctions avancées pour manipuler des jeux de données. Ce package est utilisé par toutes les personnes ayant un niveau au moins intermédiaire sur R, il n’est pas conseillé de l’utiliser si vous êtes débutant.



⍰ Installer les *packages* FactoMineR, ggplot2 et pastecs (attention aux majuscules/minuscules !) dans votre code

Concrètement, lors de l'installation des *packages*, Rstudio télécharge des fichiers et les stocke dans votre ordinateur à l'endroit indiqué par la case « *install to Library* » que vous voyez dans la capture d'écran ci-dessus. Ainsi, vous avez besoin d'installer un package une seule et unique fois. Une fois le package installé, il sera toujours disponible sur votre ordinateur.

Les *packages* que vous avez installés sont ainsi ceux que vous pouvez directement utiliser dans vos sessions de travail pour réaliser vos analyses. Pour demander à Rstudio d'utiliser les fonctions d'un *package* vous devez obligatoirement appeler ce *package* dans votre session de travail. Cela se fait avec la fonction `library()`. Par exemple, si vous souhaitez utiliser les fonctions du *package* `pastecs` vous devez indiquer dans votre code : `library(pastecs)`



Le *package* pastecs fournit notamment la fonction `stat.desc()` qui permet de calculer un ensemble de statistiques descriptives pour toutes les variables quantitatives d'un tableau de données (*i.e data frame)*. Chaque *package* possède son propre manuel d'utilisation que vous pouvez consulter en ligne sur le site officiel de R, par exemple pour le package pastecs.

Vous devez appeler les packages dont vous avez besoin à chaque nouvelle session de travail. Ainsi, si vous fermez et rouvrez R vous ne pourrez plus utiliser directement les fonctions du *package*. Il vous faudra à nouveau exécuter la fonction `library(pastecs)` par exemple si vous souhaitez utiliser les fonctions du *package* `pastecs`.

Statistiques descriptives basiques

Utilisation des fonctions de base

Pour illustrer cette section nous allons utiliser le jeu de données que nous avons utilisé lors de l'importation.

Pour rappel vous pouvez l'importer dans votre espace de travail avec le code suivant :

```
df0=read.csv2(file="https://raw.githubusercontent.com/guillaume-bourgeois92/ACP/main/ex_notes_compta_compta_stats_gfi_pour_import.csv",

              sep=";", # préciser quel est le séparateur de colonne est le point-virgule

                  dec="." , #préciser que le séparateur de décimales est le point (système anglosaxon)

                   check.names=F, #toujours utiliser cet argument pour éviter des problèmes d'importation dans le nom des variables

              row.names=1 #le nom des lignes est indiqué dans la première colonne

)
```

De base, dans Rstudio, vous avez à disposition plusieurs fonctions qui vous permettent de calculer des statistiques descriptives :

Nom de la fonction	Indicateur statistique calculé
`mean()`	Moyenne
`sd()`	Ecart-type
`var()`	Variance
`cov()`	Covariance
`median()`	Médiane
`min()`	Minimum
`max()`	Maximum
`range()`	Etendue
`sum()`	Somme des valeurs

Dans ces fonctions, le principal argument à indiquer entre les parenthèses est la série sur laquelle vous souhaitez appliquer la formule. Cela peut prendre différentes formes :

- En indiquant directement les chiffres dans la parenthèses : `mean(c(1,50,25,66,88))`. On remarquera ici l'utilisation de la fonction `c()` qui permet de dire à Rstudio que je veux la moyenne de cet ensemble de nombre
- Nous aurions pu créer le vecteur de valeurs dans un premier temps `vecteur=c(1,50,25,66,88)` puis appliquer la fonction mean sur ce vecteur : `mean(vecteur)`

- En indiquant une colonne située dans un tableau de données : `mean(df0$Math)`. Nous demandons ainsi à Rstudio de calculer la moyenne de la variable `Math` située dans le tableau de données `df0`.

On notera l'utilisation du symbole $ pour faire référence à une variable située dans un jeu de données. Cette notation est fondamentale et vous devez la maîtriser.

Calculer l'ensemble des indicateurs statistiques pour la variable Math du tableau de données df0

Fonction `summary()`

La fonction `summary()` vous permet de calculer plusieurs statistiques descriptives sur l'ensemble d'un jeu de données ou sur une variable en particulier.

Si la variable est numérique, R va calculer : le minimum, le 1er quartile, la médiane, la moyenne, le 3e quartile, le maximum :

```
> summary(df0)
  Statistiques        Math           Compta       Gestion_fi      Moyenne
 Min.   : 6.00   Min.   : 2.00   Min.   : 4.0   Min.   : 4.0   Min.   : 8.500
 1st Qu.: 8.25   1st Qu.: 7.25   1st Qu.: 6.5   1st Qu.: 9.0   1st Qu.: 9.188
 Median :11.50   Median :11.00   Median :14.0   Median :20.0   Median : 9.750
 Mean   :12.50   Mean   :10.70   Mean   :16.8   Mean   :19.6   Mean   :10.350
 3rd Qu.:18.00   3rd Qu.:15.50   3rd Qu.:27.5   3rd Qu.:28.5   3rd Qu.:11.250
 Max.   :20.00   Max.   :18.00   Max.   :32.0   Max.   :38.0   Max.   :13.750
   ecart-type
 Min.   : 4.320
 1st Qu.: 5.440
 Median : 6.445
 Mean   : 8.058
 3rd Qu.:11.775
 Max.   :13.010
```

Si la variable est textuelle, R fera un dénombrement de chaque occurrence. Par exemple, avec la variable `Species` du data frame `iris` :

```
> summary(iris$Species)
    setosa versicolor  virginica
        50         50         50
```

Utilisation de la fonction stat.desc du package pastecs

La syntaxe de la fonction `stat.desc()` est similaire à celle des fonctions de statistiques descriptives de base. La différence majeure est que cette fonction vous permet de calculer directement un grand nombre d'indicateurs statistiques. Si j'exécute la commande `stat.desc(c(1,50,25,66,88))`, j'obtiens les résultats suivants :

```
> stat.desc(c(1,50,25,66,88))
    nbr.val     nbr.null       nbr.na          min          max        range          sum       median
  5.0000000    0.0000000    0.0000000    1.0000000   88.0000000   87.0000000  230.0000000   50.0000000
       mean      SE.mean CI.mean.0.95          var      std.dev     coef.var
 46.0000000   15.2413910   42.3168855 1161.5000000   34.0807864    0.7408867
```

Nous verrons juste après comment réduire le nombre de décimales. Attardons-nous pour l'instant sur la signification de chacun de ces indicateurs :

Nom de l'indicateur	Signification
`Nbr.val`	Nombre de valeurs
`Nbr.null`	Nombre de valeurs égales à 0
`Nbr.na`	Nombre de valeurs manquantes
`min`	Minimum
`max`	Maximum
`range`	Etendue
`median`	Médiane
`mean`	Moyenne
`SE.mean`	Erreur type de la moyenne
`CI.mean.0.95`	Borne inférieur de l'intervalle de confiance à 95% de la moyenne
`var`	Variance
`Std.dev`	Ecart-type
`Coef.var`	Coefficient de variation (écart type divisé par la moyenne)

Astuce : utilisez les crochets pour ne calculer que certains indicateurs avec la fonction `stat.desc()`. Pour n'obtenir que le nombre de valeurs manquantes (3e élément), le minimum (4e), le maximum (5e), la médiane (7) et la moyenne (8) : `stat.desc(c(1,50,25,66,88))[c(3,4,5,7,8)]`

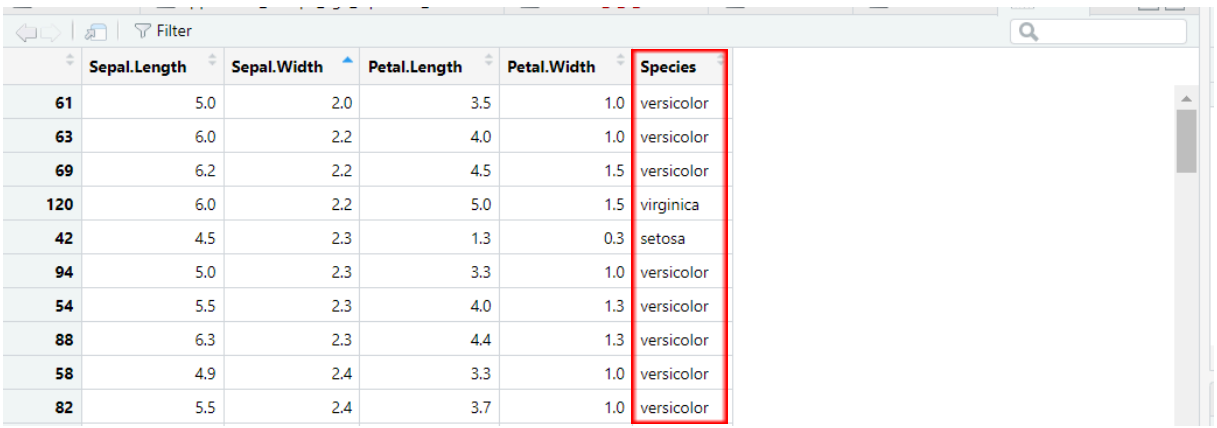
```
> stat.desc(c(1,50,25,66,88))[c(3,4,5,7,8)]
nbr.na    min    max    sum median
     0      1     88    230     50
```

Observez d'ailleurs que comme aucune valeur ne comporte de décimale, aucune décimale apparaît dans les résultats de la console.

Statistiques descriptives sur un sous-échantillon

Dans R, vous pouvez bien sûr calculer des statistiques descriptives sur un sous-échantillon. Pour illustrer cela, reprenons le jeu de données iris (préchargé dans le logiciel).

```
rm(list=ls())

data(iris)

iris

View(iris)
```



	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
61	5.0	2.0	3.5	1.0	versicolor
63	6.0	2.2	4.0	1.0	versicolor
69	6.2	2.2	4.5	1.5	versicolor
120	6.0	2.2	5.0	1.5	virginica
42	4.5	2.3	1.3	0.3	setosa
94	5.0	2.3	3.3	1.0	versicolor
54	5.5	2.3	4.0	1.3	versicolor
88	6.3	2.3	4.4	1.3	versicolor
58	4.9	2.4	3.3	1.0	versicolor
82	5.5	2.4	3.7	1.0	versicolor

La variable Species de ce jeu de données contient 3 modalités : `unique(iris$Species)==> setosa versicolor virginica`. Nous pouvons calculer la moyenne de la longueur des sépales pour chacune de ces espèces, en précisant entre crochets que l'on souhaite filtrer les données une espèce particulière :

```
mean(iris$Sepal.Length[iris$Species=="setosa"])

mean(iris$Sepal.Length[iris$Species=="versicolor"])

mean(iris$Sepal.Length[iris$Species=="virginica"])
```

Cependant, cette méthode nécessite de créer une ligne de code pour chaque modalité différente de la variable Species (ce qui n'est pas très pratique!). Pour éviter cela, il faut utiliser la fonction `aggregate()`. La syntaxe générale de cette fonction est :

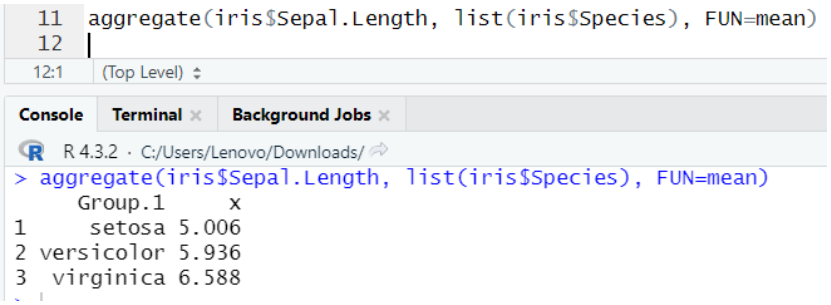
```
aggregate(df$col_to_aggregate, list(df$col_to_group_by), FUN=mean)
```

- df désigne le nom du data frame (iris dans notre exemple)
- col_to_aggregate désigne le nom de la variable sur laquelle calculer la moyenne (Sepal.Length dans notre exemple)
- col_to_group_by désigne le nom de la variable dont les modalités vont permettre de construire les sous-groupes (Species dans notre exemple).

Ainsi, en exécutant :

```
aggregate(iris$Sepal.Length,

        list(iris$Species),

        FUN=mean)
```

J'obtiens :



Nous pouvons calculer d'autres indicateurs que la moyenne :

```
aggregate(iris$Sepal.Length, list(iris$Species), FUN=sd) # écart-type

aggregate(iris$Sepal.Length, list(iris$Species), FUN=median) # médiane

aggregate(iris$Sepal.Length, list(iris$Species), FUN=min) # minimum

…
```

Compléments

Comment réduire le nombre de décimales ?

Pour réduire le nombre de décimales affichées par R, nous pouvons utiliser la fonction `round()` qui permet d'arrondir les valeurs avec un nombre de chiffres après la virgule défini. Ainsi, la commande `stat.desc(c(1,50,25,66,88))`

devient

`round(``stat.desc(c(1,50,25,66,88))`**`, 2)`** si nous souhaitons deux chiffres après la virgule.

Matrice de corrélation

La matrice de corrélation permet de représenter sous forme de tableau les coefficients de corrélation entre les variables prises deux à deux. Pour calculer une matrice de corrélation nous devons utiliser la fonction `cor()` :

```
cor(df0)
```

```
> cor(df0)
             Statistiques        Math     Compta Gestion_fi    Moyenne ecart-type
Statistiques    1.0000000  0.72645198 -0.8186291 -0.6083936 0.15985101 -0.7038429
Math            0.7264520  1.00000000 -0.8488758 -0.7069364 0.05300547 -0.8050402
Compta         -0.8186291 -0.84887580  1.0000000  0.9124336 0.31703187  0.8998517
Gestion_fi     -0.6083936 -0.70693638  0.9124336  1.0000000 0.60886073  0.7845192
Moyenne         0.1598510  0.05300547  0.3170319  0.6088607 1.00000000  0.2446502
ecart-type     -0.7038429 -0.80504022  0.8998517  0.7845192 0.24465018  1.0000000
```

Comment exporter ses jeux de données?

Lorsque Rstudio fournit des résultats, ceux-ci ne sont pas du tout mis en forme et ne peuvent pas être utilisés comme tels dans vos rapports. Vous devez les exporter en csv pour les mettre en forme dans Excel. Les exportations se font avec la fonction `write.csv2()`.

La fonction `write.csv2()` fonctionne ainsi : `write.csv2(ce que je veux exporter, file="chemin où je souhaite enregistrer/nom_fichier.csv")`

Par exemple, si je souhaite exporter les statistiques descriptives de la variable Math du jeu de données importé depuis Github, dans mon dossier de téléchargements (Downloads) et en créant un fichier que je nomme `export_stat_desc.csv` :

```
library(pastecs)


df0=read.csv2(file="https://raw.githubusercontent.com/guillaume-bourgeois92/ACP/main/ex_notes_compta_compta_stats_gfi_pour_import.csv",

              sep=";", # préciser quel est le séparateur de colonne est le point-virgule

                  dec="." , #préciser que le séparateur de décimales est le point (système anglo saxon)

                   check.names=F, #toujours utiliser cet argument pour éviter des problèmes d'importation dans le nom des variables

                   row.names=1 #le nom des lignes est indiqué dans la première colonne du fichier csv

)
```

```
write.csv2(stat.desc(df0$Math),
file="c:/Users/guillaume.bourgeois/Downloads/export_stat_desc.csv")
```

	A	B
1		x
2	nbr.val	10
3	nbr.null	0
4	nbr.na	0
5	min	2
6	max	18
7	range	16
8	sum	107
9	median	11
10	mean	10,7
11	SE.mean	1,7953644
12	CI.mean.0.95	4,06139644
13	var	32,2333333
14	std.dev	5,67744074
15	coef.var	0,53060194

Vous pouvez également sauvegarder vos données dans le format spécifique à R : `.RData`. ATTENTION : vous ne pouvez sauvegarder qu'un objet que vous avez créé (vous ne pouvez pas demander à R d'enregistrer directement le résultat de `stat.desc(df0$Math)` par exemple)

```
library(pastecs)


df0=read.csv2(file="https://raw.githubusercontent.com/guillaume-bourgeois92/ACP/main/ex_notes_compta_compta_stats_gfi_pour_import.csv",

              sep=";", # préciser quel est le séparateur de colonne est le point-virgule

                  dec="." , #préciser que le séparateur de décimales est le point (système anglo saxon)

                   check.names=F, #toujours utiliser cet argument pour éviter des problèmes d'importation dans le nom des variables

                   row.names=1 #le nom des lignes est indiqué dans la première colonne du fichier csv

)


write.csv2(stat.desc(df0$Math), file="c:/Users/Lenovo/Downloads/export_stat_desc.csv")


statistiques_math=stat.desc(df0$Math) # on enregistre les résultats dans un objet dans R

save(statistiques_math,file="c:/Users/Lenovo/Downloads/export_stat_desc.RData"   )   # on exporte cet objet pour réaliser une sauvegarde du résultat au format .RData
```

 calculer la matrice de corrélation sur le data frame df0 et enregistrer cette matrice de corrélation dans l'objet mat_cor. Exporter ensuite cette matrice de corrélation en format csv, avec la fonction `write.csv2()` et la mettre en forme dans Excel.

Dénombrement et tableau de contingence

- La fonction `table()` permet de réaliser un dénombrement des valeurs uniques prises par une variable.

Prenons l'exemple du jeu de données disponible avec ce lien : https://raw.githubusercontent.com/guillaume-bourgeois92/-conom-trie_l3/main/dm_2023_data_assurance.csv

```
df0=read.csv2(file="https://raw.githubusercontent.com/guillaume-bourgeois92/-conom-trie_l3/main/dm_2023_data_assurance.csv",
              check.names=F,
              sep=";",
              dec=",")
```

Filter

	age	sex	bmi	children	smoker	region	charges	age2	bmi2	lage	lbmi	lcharges
1	19	female	27.900	0	yes	southwest	16884.924	361	778.4100	2.944439	3.328627	9.734176
2	18	male	33.770	1	no	southeast	1725.552	324	1140.4129	2.890372	3.519573	7.453302
3	28	male	33.000	3	no	southeast	4449.462	784	1089.0000	3.332205	3.496508	8.400538
4	33	male	22.705	0	no	northwest	21984.471	1089	515.5170	3.496508	3.122585	9.998092
5	32	male	28.880	0	no	northwest	3866.855	1024	834.0544	3.465736	3.363149	8.260197
6	31	female	25.740	0	no	southeast	3756.622	961	662.5476	3.433987	3.248046	8.231275
7	46	female	33.440	1	no	southeast	8240.590	2116	1118.2336	3.828641	3.509753	9.016827
8	37	female	27.740	3	no	northwest	7281.506	1369	769.5076	3.610918	3.322875	8.893093

Ce jeu de données est composé de 1 338 individus (vérifiable avec `dim(df0)`). Nous pouvons calculer grâce à la fonction `table()`, combien de femmes et d'hommes composent cet échantillon :

```
table(df0$sex)
```

```
> table(df0$sex)

female   male
   662    676
```

Dans cet échantillon, nous avons donc 662 femmes et 676 hommes.

- Avec la fonction `table()`, on donne l'effectif de chacune des modalités. La fonction `prop.table()` permet de donner les fréquences. **ATTENTION : vous ne pouvez utiliser la fonction `prop.table()` que sur un résultat de `table()` :**

```
prop.table(table(df0$sex))
```

```
> prop.table(table(df0$sex))

   female      male
0.4947683 0.5052317
```

R donne trop de décimales dans les résultats, cela se solutionne avec la fonction `round()` qui permet d'arrondir :

```
round(  prop.table( table(df0$sex)),2)
```

⇒ Résultats arrondis à 2 chiffres après la virgule (2 avant la dernière parenthèse dans le code ci-dessus)

```
> round(  prop.table( table(df0$sex)),2)

female   male
  0.49   0.51
```

- Enfin, il est possible de construire des tableaux de contingence grâce à la fonction `table();` en indiquant deux variables. Par exemple, si nous souhaitons savoir combien d'hommes fument, combien ne fument pas (et idem pour les femmes) :

```
table(df0$sex, df0$smoker)
```

```
> table(df0$sex, df0$smoker)

         no yes
female  547 115
male    517 159
```

Ainsi, parmi les femmes de l'échantillon, 547 ne fument pas et 115 fument.

Pour obtenir les fréquences partielles, on utilisera la fonction `prop.table()` :

```
round(prop.table(table(df0$sex, df0$smoker)),2)
```

```
> round(prop.table(table(df0$sex, df0$smoker)),2)

         no  yes
female 0.41 0.09
male   0.39 0.12
```

Vous pouvez bien vérifier que la somme des valeurs fait ici 1.

Les fréquences conditionnelles se calculent avec l'option `margin` à l'intérieur de la fonction `prop.table()`. On utilisera margin=1 pour les fréquences conditionnelle en ligne, et margin=2 pour les fréquences conditionnelles en colonne:

```
round(prop.table(table(df0$sex, df0$smoker), margin=1),2)
```

```
> round(prop.table(table(df0$sex, df0$smoker), margin=1),2)

         no  yes
female 0.83 0.17
male   0.76 0.24
```

⇒ 83% des femmes ne fume pas, 17% des femmes fume

⇒ 76% des hommes ne fume pas, 24% des hommes fume

```
round(prop.table(table(df0$sex, df0$smoker), margin=2),2)
```

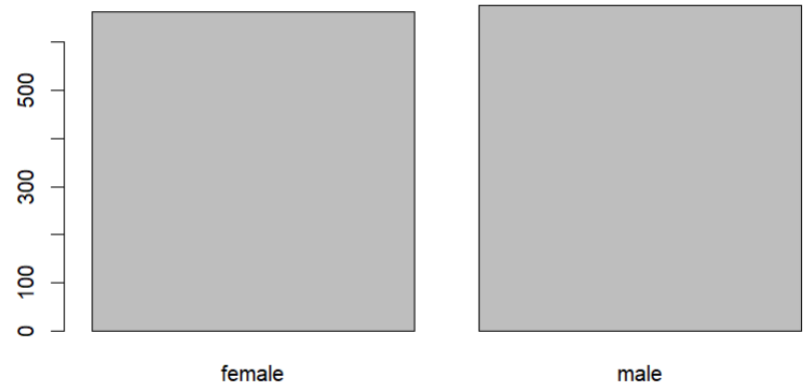
```
> round(prop.table(table(df0$sex, df0$smoker), margin=2),2)

           no  yes
female 0.51 0.42
male   0.49 0.58
```

Graphiques

Barplot (diagramme en tuyau d'orgue)

A partir des résultats de la fonction table(), nous pouvons tracer un barplot avec la fonction éponyme :

```
barplot(table(df0$sex))
```

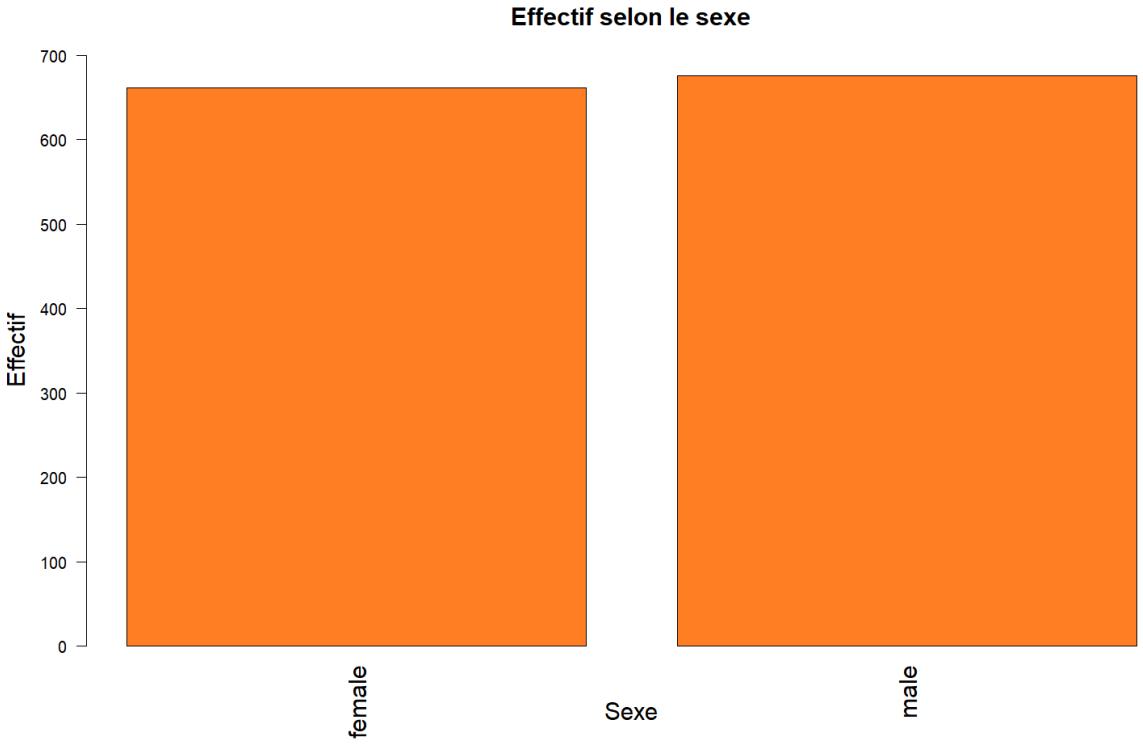


R n'est pas le pro de la mise en forme et on le voit bien avec ce graphique qui n'est pas très esthétique (!). Nous aimerions :

- Changer la couleur ⇒ argument `col` (voir toutes les couleurs de base dans R : lien)
- Que l'axe des ordonnées aille au delà des barres ⇒ argument `ylim`
- Changer l'orientation des graduations ⇒ argument `las`
- Mettre un titre ⇒ argument `main`,
- Puis mettre le titre en plus gros ⇒ argument `cex.main`
- Mettre un titre à l'axe des ordonnées ⇒ argument `ylab`
- Mettre un titre à l'axe des abscisses ⇒ argument `xlab`
- Mettre le titre des axes en plus gros ⇒ argument `cex.lab`
- Mettre les graduations en plus gros ⇒ argument `cex.names`

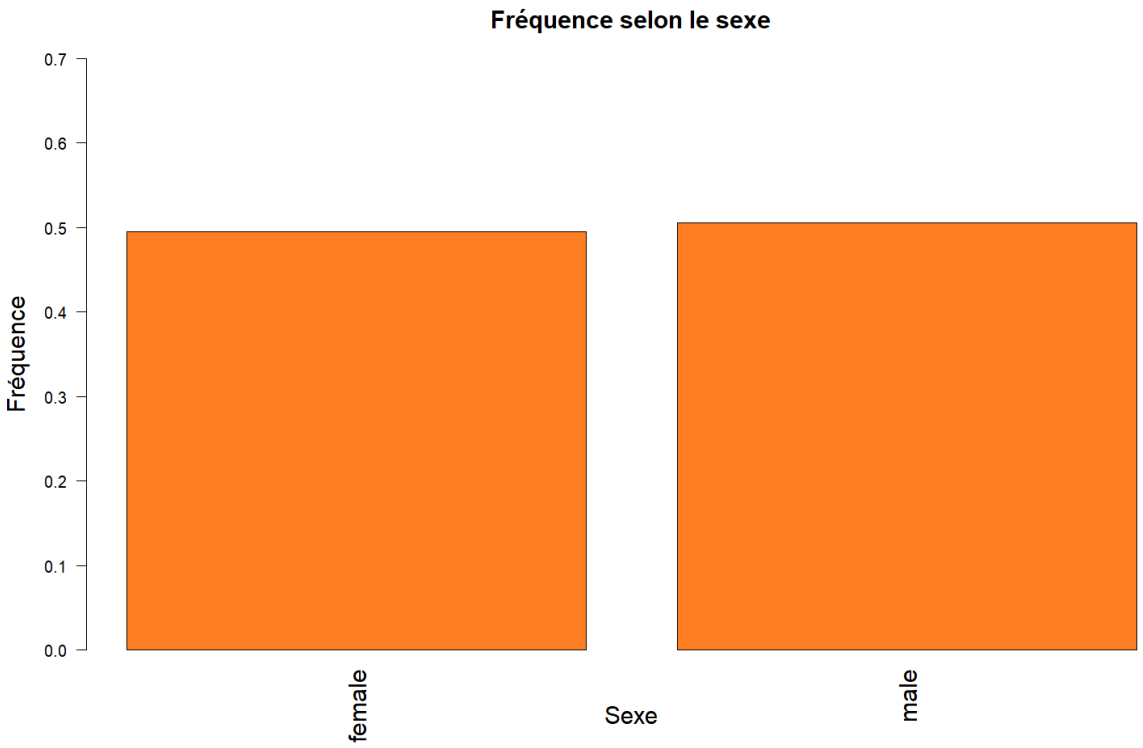
```
barplot(table(df0$sex),
        col="chocolate1",
        ylim=c(0,700),
        las=2,
        main="Effectif selon le sexe",
        cex.main=1.5,
        ylab="Effectif",
        xlab="Sexe",
       cex.lab=1.5,
        cex.names=1.5
```

```
        )
```



Nous pouvons également construire un barplot sur les résultats de la fonction `prop.table()` : ATTENTION ici à bien changer les limites de l'axe des ordonnées et à changer les titres le cas échéant

```
barplot(prop.table(table(df0$sex)),
        col="chocolate1",
        ylim=c(0,0.7),
        las=2,
        main="Fréquence selon le sexe",
        cex.main=1.5,
        ylab="Fréquence",
        xlab="Sexe",
        cex.lab=1.5,
        cex.names=1.5
)
```

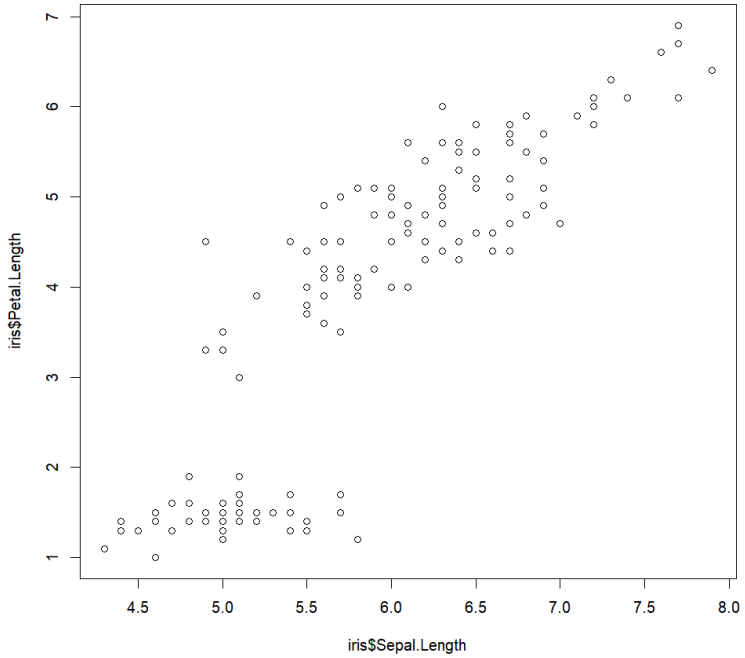


⇒ Faire l'exercice 7

Plot (nuages de points, courbes)

A partir du jeu de données préchargé iris nous pouvons tracer le nuage de points de la longueur des pétales en fonction de la longueur des sépales :

```
plot(x=iris$Sepal.Length, y=iris$Petal.Length)
```



Pour mettre en forme ce graphique, les arguments sont identiques à ceux d'un barplot :

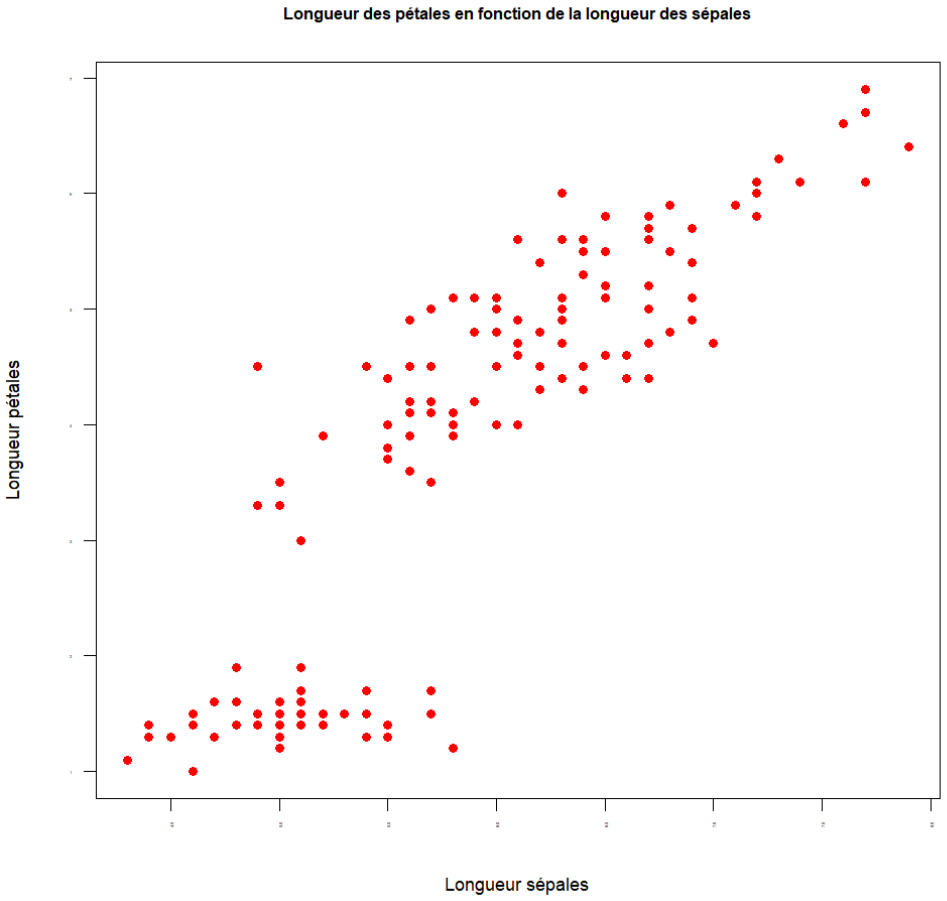
- Changer la couleur ⇒ argument `col` (voir toutes les couleurs de base dans R : lien)
- Que l'axe des ordonnées aille au delà des barre ⇒ argument `ylim`
- Changer l'orientation des graduations ⇒ argument `las`
- Mettre un titre ⇒ argument `main,`
- Puis mettre le titre en plus gros ⇒ argument `cex.main`
- Mettre un titre à l'axe des ordonnées ⇒ argument `ylab`
- Mettre un titre à l'axe des abscisses ⇒ argument `xlab`
- Mettre le titre des axes en plus gros ⇒ argument `cex.lab`
- ATTENTION : cex.names ne fonctionne pas ⇒ utiliser à la place `cex.axis` pour modifier la taille des graduations

On peut néanmoins rajouter :

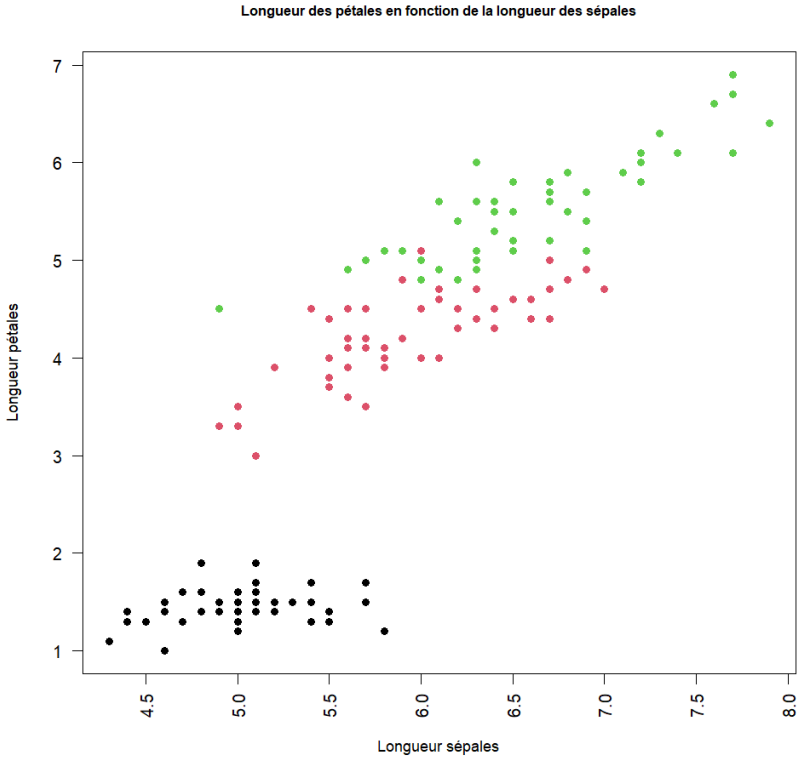
- Changer le style des points ⇒ argument `pch`. cet argument prend les valeurs de 1 à 25 (aide à lire)

```
plot(x=iris$Sepal.Length, # variable en abscisses, variable Sepal.Length du data frame iris
     y=iris$Petal.Length,
     col="red",
     pch=16,
     main="Longueur des pétales en fonction de la longueur des sépales",
     xlab="Longueur sépales",
     ylab="Longueur pétales",
```

```
    cex.main=0.8,
    cex.lab=0.9,
    las=2,
    cex.axis=0.2
    )
```

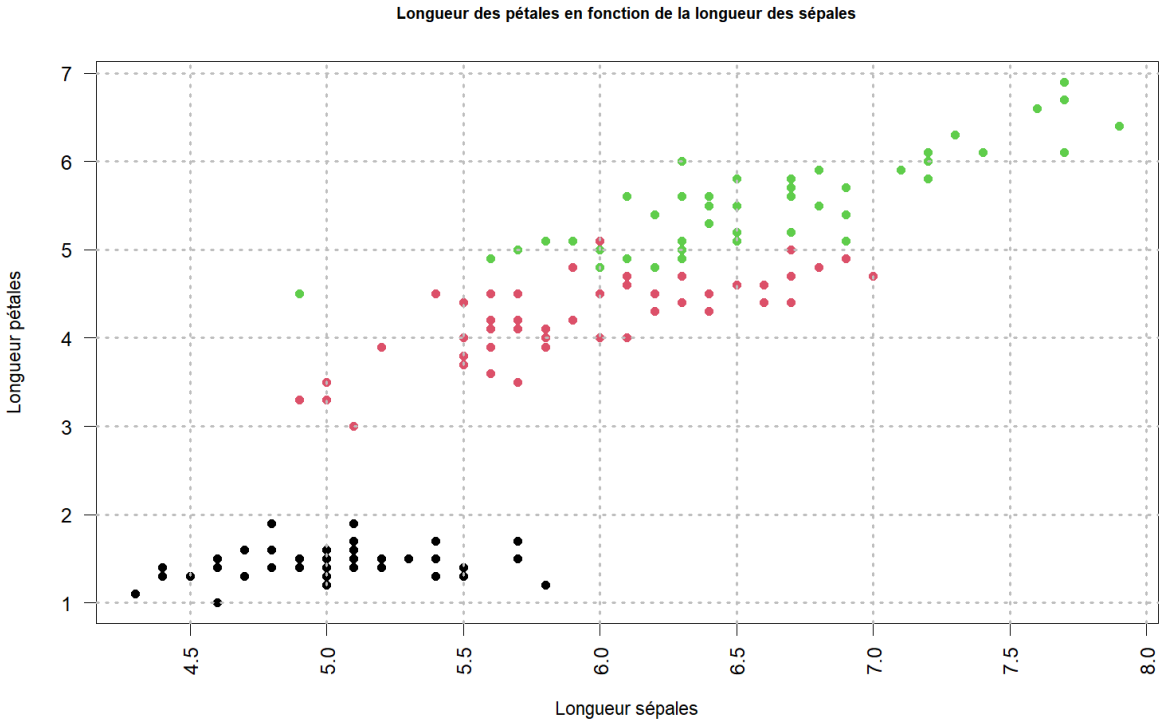


En précisant `col=iris$Species` (au lieu de `col="red"`), on peut colorier les points en fonction des modalités de la variable Species :



Nous pouvons rajouter une grille qui peut faciliter la lecture du graphique. Pour cela, il suffit de rajouter une nouvelle ligne en dessous du code ci-dessus :

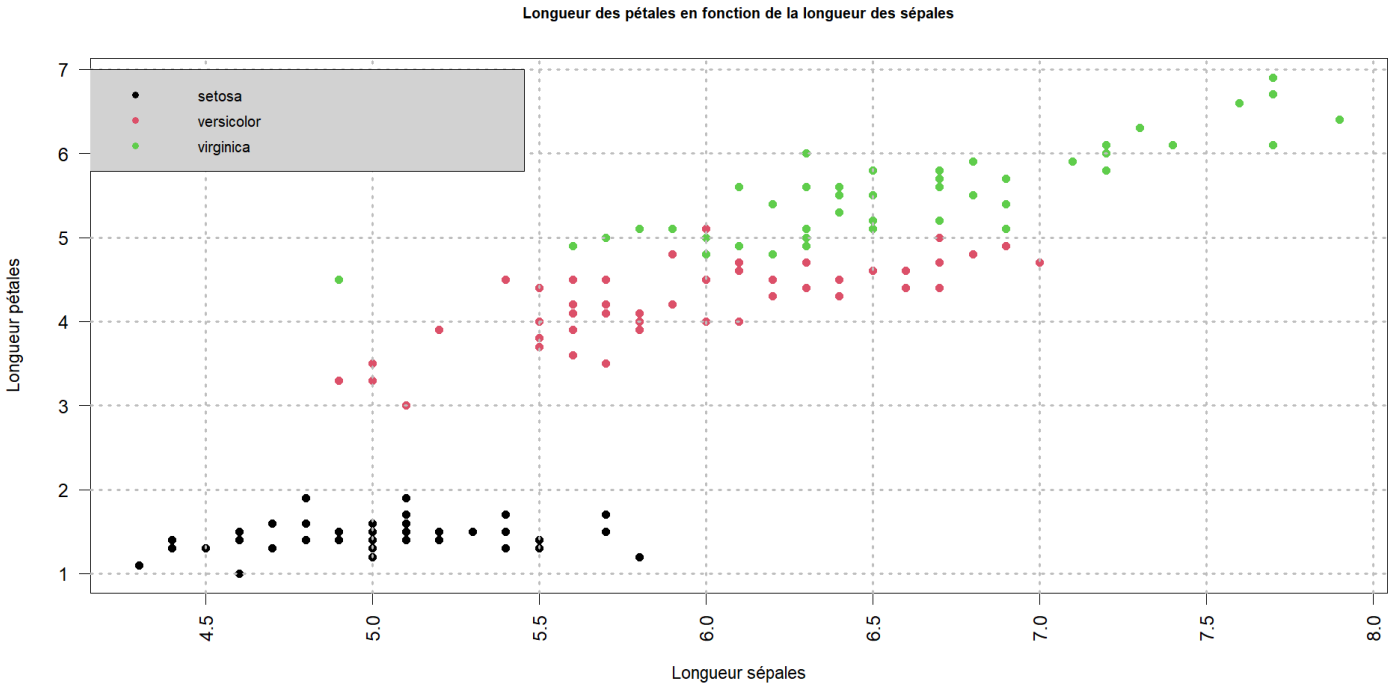
```
grid(lwd=2, col="grey")
```



A noter qu'il est possible de modifier les paramètres de la grille (nombre de lignes, type de ligne, couleur, uniquement des lignes horizontales,...). Pour plus d'informations vous pouvez consulter cette page internet.

Pour une mise en forme plus avancée, il est possible de rajouter une légende sur ce graphique. Pour cela, il faut indiquer le code suivant juste après la fonction `plot()` :

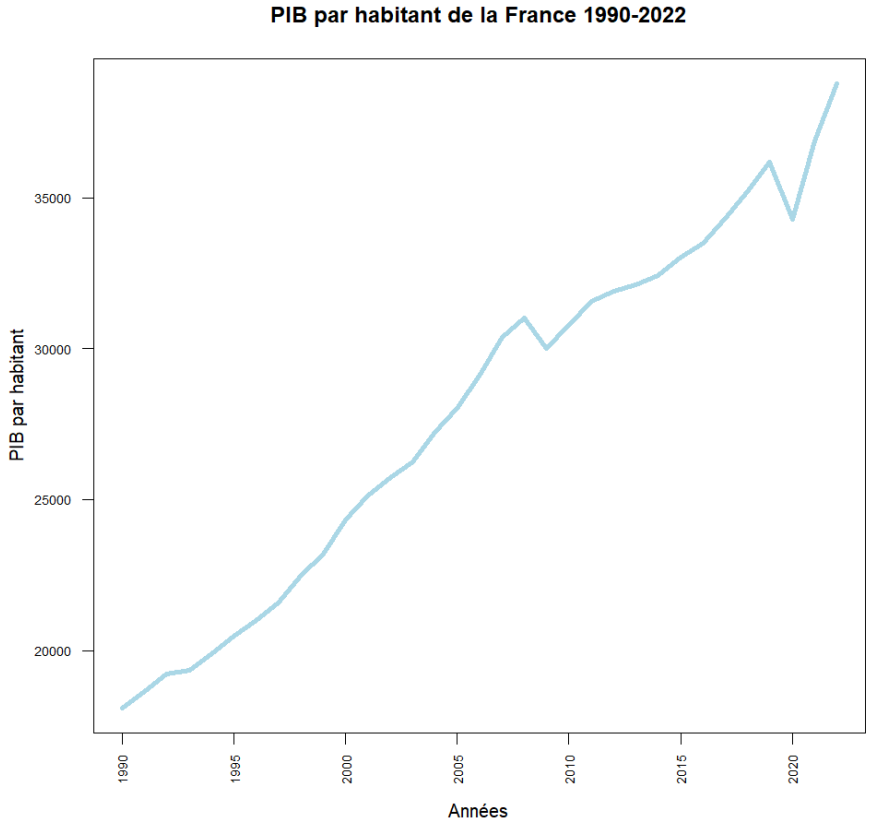
```
plot(x=iris$Sepal.Length,
       y=iris$Petal.Length,
       col=iris$Species,
       pch=16,
       main="Longueur des pétales en fonction de la longueur des sépales",
       xlab="Longueur sépales",
       ylab="Longueur pétales",
       cex.main=0.8,
       cex.lab=0.9,
       las=2,
       cex.axis=1
  )
grid(lwd=2, col="grey") # pour tracer une grille sur le graphique
legend(4.1, # position sur x de la légende
 7, # position sur y de la légende ⇒ Commence au point x=4.1 et y=7
 legend=levels(iris$Species), # étiquettes
 pch=16, # type de points (utiliser le même chiffre que dans la fonction plot)
 col=unique(iris$Species), # couleur en fonction des modalités de la variable Species
 cex=0.8, # taille du texte
 bg="lightgrey") # couleur de l'arrière plan de la zone de légende
```



La fonction plot permet également de tracer des courbes. Cherchons par exemple à tracer l'évolution du PIB par habitant de la France entre 1990 et 2002. Les données peuvent être importées avec le code suivant :

```
df0=read.csv2(file="https://raw.githubusercontent.com/guillaume-bourgeois92/R/main/pib_france.csv",
     sep=";",
     check.names=F)
```

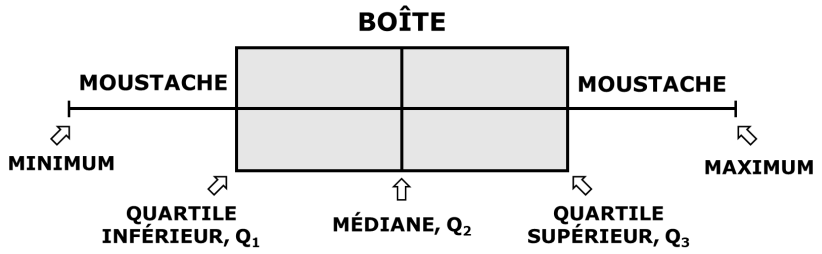
```
plot(x=df0$Annee,
     y=df0$PIB_per_capita,
     main="PIB par habitant de la France 1990-2002",
     xlab="Années",
     ylab="PIB par habitant",
     las=2, # orientation des graduations,
     cex.axis=0.7, # taille de la police des graduations
     type="l", # tracer une ligne
     col="lightblue", # couleur de la ligne
     lwd=4, # largeur de la ligne
       lty=1 # type de trait :
        #http://www.sthda.com/french/wiki/les-differents-types-de-traits-dans-r-lty
     )
```



Boxplot

La fonction `boxplot` permet de tracer des boîtes à moustaches. Pour rappel, une boîte à moustaches se présente ainsi :

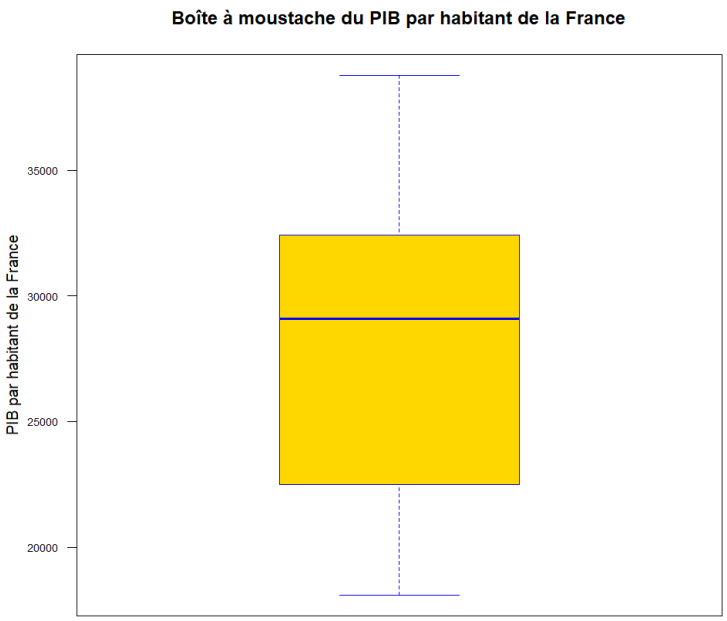
Figure 4.5.2.1 Construction d'une boîte à moustaches



C'est donc d'une représentation graphique de plusieurs indicateurs de statistique descriptive. Les arguments ne sont pas beaucoup différents que dans les autres fonctions graphiques. Vous pouvez changer la couleur des bordures : `border="Blue"` par exemple.

```
boxplot(df0$PIB_per_capita,
        las=2,
        col="Gold",
        main="Boîte à moustache du PIB par habitant de la France",
        ylab="PIB par habitant de la France",
        cex.axis=0.7,
        border="Blue" # Changer la couleur des bordures
        )
```

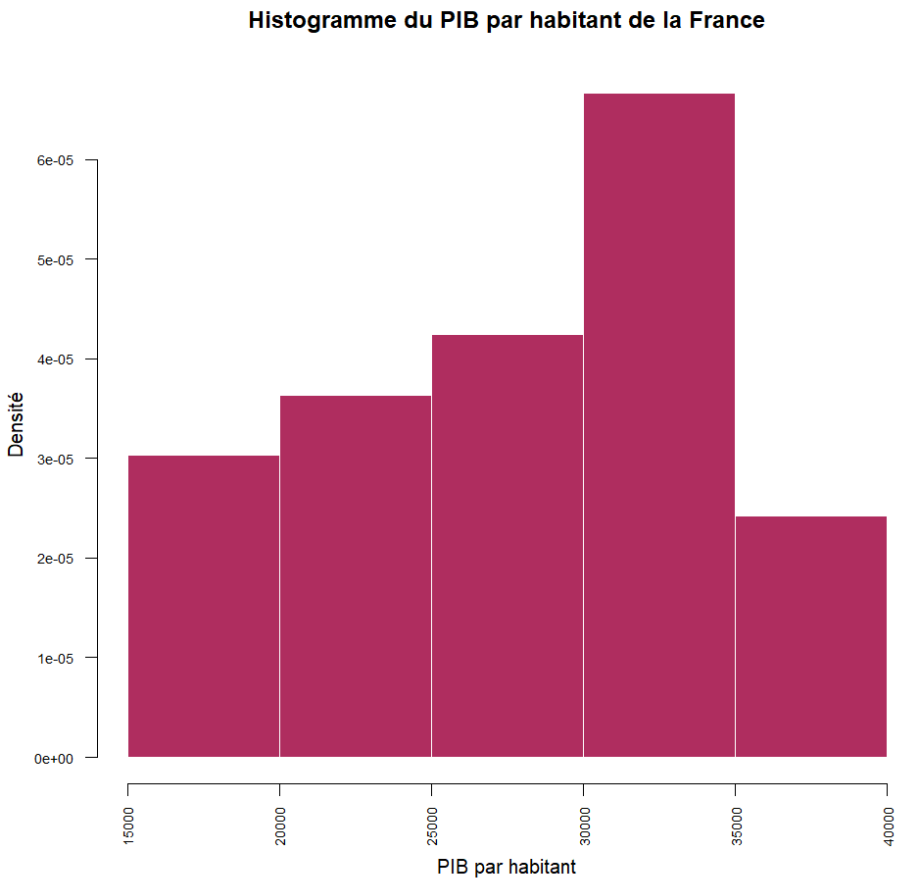
Plus d'informations sur les boxplot dans R ici et ici (plus avancé, avec l'utilisation des packages dplyr et ggplot2).



Histogrammes

Il faut utiliser la fonction `hist()` :

```
hist(df0$PIB_per_capita,
     col="maroon", # couleur à l'intérieur des barres
     border="White", # Couleur des bordures
     main="Histogramme du PIB par habitant de la France",
     xlab="PIB par habitant",
     ylab="Densité",
    freq=FALSE, # représenter la densité et non les fréquences sur l'axe des ordonnées
          las=2, #orientation des graduations
     cex.axis=0.7)
```



Compléments :

https://bookdown.org/ael/rexplor/chap7.html

http://www.thibault.laurent.free.fr/cours/R_intro/chapitre_4.html

Manipulation des données

Création de nouvelles variables dans un data frame

Il est facile de rajouter une variable dans un data frame. Pour cela, la structure du code est **nom_data_frame$nom_nouvelle_variable = formule pour créer la nouvelle variable.**

Prenons l'exemple d'un petit jeu de données, que je nomme articles, avec des articles de sport et leur prix hors taxe :

```
articles=data.frame("article"=c("Maillot", "Chaussures", "ballon foot", "balllon rugby",
                                "raquette tennis"),
                    "prix_ht"=c(25, 80, 25, 30, 150))
```

	article	prix_ht
1	Maillot	25
2	Chaussures	80
3	ballon foot	25
4	balllon rugby	30
5	raquette tennis	150

Rajoutons une variable "prix_ttc" égale au prix_ht *1.2, dans ce data frame "articles" :

```
articles$prix_ttc = articles$prix_ht * 1.2
```

	article	prix_ht	prix_ttc
1	Maillot	25	30
2	Chaussures	80	96
3	ballon foot	25	30
4	balllon rugby	30	36
5	raquette tennis	150	180

NB : voici les symboles pour saisir des formules dans R

- + : addition
- - : soustraction
- * : multiplication
- / : division
- ^ ou ** : exponentiation
- log() : fonction logarithme (ex : `log(iris$eSpal.Length)`)
- exp() : fonction exponentielle (ex : `exp(iris$Sepal.Length))`

Compléments : une manière plus avancée de créer de nouvelles variables est de passer par des fonctions. Par exemple, dans R, la fonction `ifelse()` vous permet de réaliser des traitements

conditionnels (usage identique à la fonction SI dans Excel). Vous trouverez plus d'informations sur cette fonction ici.

```
# Par exemple pour créer une variable dans le jeu de données iris, que l'on nommera "grande sépale", égale à "oui" si la fleur possède une longueur de sépale supérieur à 6 et "non" sinon :

iris$grande_sepale=ifelse(iris$Sepal.Length>6, "oui","non")
```

Filtrer les données dans un data frame

Nous avons vu précédemment que nous pouvons utiliser les crochets [] afin d'extraire un élément d'un vecteur, plusieurs d'un coup, tout un vecteur sauf un élément, etc… Les crochets peuvent être utilisés pour filtrer les données. Par exemple, si je reprends le petit data frame `articles` ci-dessus et que je souhaite afficher uniquement les objets dont le prix ht est supérieur à 50€ :

```
articles[articles$prix_ht>50,]
```

Nous remarquons ici que la syntaxe globale ne change pas à l'intérieur des [] : avant la virgule ce sont les lignes que nous souhaitons conserver et après la virgule les colonnes. Ici, nous filtrons uniquement les lignes pour lesquelles la valeur de la variable prix_ht est supérieure à 50€.

Voici les symboles de comparaisons à utiliser dans R :

- > : supérieur à
- >= : supérieur ou égal à
- < : inférieur à
- <= : inférieur ou égal à
- == : égal à
- != : différent de

Autre exemple, à partir du jeu de données iris, nous pouvons extraires les lignes pour les fleurs de l'espèce (variable Species) virginica :

```
data(iris)

iris2=iris[iris$Species=="virginica",]

View(iris2)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
101	6.3	3.3	6.0	2.5	virginica
102	5.8	2.7	5.1	1.9	virginica
103	7.1	3.0	5.9	2.1	virginica
104	6.3	2.9	5.6	1.8	virginica
105	6.5	3.0	5.8	2.2	virginica
106	7.6	3.0	6.6	2.1	virginica
107	4.9	2.5	4.5	1.7	virginica
108	7.3	2.9	6.3	1.8	virginica
109	6.7	2.5	5.8	1.8	virginica
110	7.2	3.6	6.1	2.5	virginica
111	6.5	3.2	5.1	2.0	virginica
112	6.4	2.7	5.3	1.9	virginica

Ou bien extraire toutes les lignes SAUF celle de l'espèce virginica :

```
data(iris)

iris3=iris[iris$Species!="virginica",]

View(iris3)
```

Nous pouvons associer plusieurs conditions :

- le signe & (esperluette) permet d'indiquer que toutes les conditions doivent être respectées en même temps. Par exemple, si je souhaite extraire dans le jeu de données iris les plantes appartenant à l'espèce "Virginica" **ET** dont la longueur des sépales est supérieure à 2 :

```
data(iris)

iris3=iris[iris$Species=="virginica" & iris$Sepal.Length>2,]

View(iris3)
```

- le signe | (altgr+6 sous windows ; Maj + option + L sous mac) permet d'indiquer que l'une des conditions doit être respectée (une seule suffit). Par exemple, si je souhaite extraire dans le jeu de données iris les plantes appartenant à l'espèce "Virginica" **OU** celles dont la longueur des sépales est supérieure à 2 :

```
data(iris)

iris3=iris[iris$Species=="virginica" | iris$Sepal.Length>2,]

View(iris3)
```

Modifier le nom des variables

La fonction `colnames()` renvoie le vecteur du nom des colonnes du data frame :

```
data(iris)

colnames(iris)
```

⇒ `[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"`

`colnames(iris)[1]` renvoie alors au premier élément de ce vecteur : `"Sepal.Length"`. Il est alors facile de changer le nom de la première variable, par exemple si je souhaite l'appeler "Longueur_sepale" :

```
colnames(iris)[1]="Longueur_sepale"
```

ATTENTION : ne jamais mettre d'accent ou d'espace dans le nom d'une variable! Généralement, nous représentons un espace avec l'*underscore* (le "tiret du 8" : _).

Séance 4 : arrêter la lecture ici, et réaliser l'exercice 8 puis l'exercice NUTRIAGE

A COMPLETER

- extraction sous chaine (substr)
- package dplyr, jointures, …

Calculs matriciels

```
# création d'une matrice avec les entiers naturels de 1 à 20, comprenant 5 colonnes et 4 lignes

m=matrix(1:20,

          ncol=5,

          nrow=4,

          byrow=TRUE)


m # afficher cette matrice


mm = matrix(c(1,2,3, 11,12,13),

          nrow = 2,

          ncol=3,

          byrow=TRUE,

          dimnames = list(c("row1", "row2"),

                                  c("C.1", "C.2", "C.3"))) # precise le nom des lignes et des colonnes dans cette matrice


mm


m0=c(1:25)

m0 # vecteur ligne avec les valeurs des entiers naturels 1 à 25


m1=matrix(m0)

m1 # on a transformé ce vecteur en vecteur colonne


m1=matrix(m0,ncol=5,

      nrow=5,

      byrow=TRUE) #on crée une matrice à partir du vecteur m0 (attention au nb de lignes et de colonnes : il faut que nombre lignes * nombre colonnes soit égal au nombre d'éléments à inclure dans la matrice, ici tous les entiers de 1 à 25 donc 25 éléments)

m1 # nous avons maintenant une matrice 5x5, avec les valeurs de 1 à 25
```

```
# on peut créer une matrice en collant deux vecteurs ou plus :
rbind(c(1, 2, 3), c(4, 5, 6)) #collage en ligne
cbind(c(1, 2, 3), c(4, 5, 6)) # collage en colonne

#transposer une matrice (inverser les lignes et les colonnes)
t(m1)

#informations sur les matrices
dim(m1) # dimensions matrice
ncol(m1) # nombre de colonnes
nrow(m1) # nombre de lignes

#determinant, inversion, trace, valeurs propres
m2=matrix(c(5, 1, 0,
            3,-1, 2,
            4, 0,-1),
          nrow=3,
          byrow=TRUE)
m2

det(m2) #déterminant de la matrice
solve(m2) #rappel il faut det !=0 ==> inversion de la matrice
sum(diag(m2)) #trace de la matrice, c'est à dire la somme des éléments dans sa première diagonale (5-1-1)
eigen(m2) #valeurs propres et vecteurs propres de la matrice (ici racines imaginaires du polynôme caractéristique)

#opérations sur les matrices
o_m1=matrix(c(1:25), nrow=5, ncol=5)
o_m1

o_m2=matrix(c(26:50), nrow=5, ncol=5, byrow=TRUE)
o_m2
```

```
o_m1+o_m2

o_m1-o_m2

o_m1*o_m2 # Attention toutes les opérations fonctionnent car 2 matrices de même dimensions...

o_m1/o_m2
```

Boucles (for, while)

A COMPLETER

Graphiques

https://bookdown.org/ael/rexplor/chap7.html

http://www.thibault.laurent.free.fr/cours/R_intro/chapitre_4.html

R pour l'analyse de données (ACP, AFC, …)

Prérequis

Avant de poursuivre votre lecture, vous devez vous assurer que vous êtes à l'aise avec les points suivants :

- Interface du logiciel
- Comment obtenir de l'aide sur une fonction
- L'installation et l'utilisation de packages (*dplyr*, *FactoMiner*, *pastecs*, …)
- La création d'objets, leur manipulation (objets "simples", data frame, utilisation *$* et des *[ ]*)
- Le calcul de statistiques descriptives (fonction stat.desc du package *pastecs*, fonction *stargazer* du package éponyme, …)
- La création de graphiques (notamment `barplot`)
- Exporter un data frame en csv afin de mettre le tableau en forme puis l'insérer dans un document Word

Réalisation d'une ACP

Support de cours ppt : lien

L'analyse en composante principale (ACP) est utilisée pour explorer et analyser les données d'un ensemble de variables quantitatives prises par plusieurs individus. Des variables qualitatives peuvent être prises en compte comme variables supplémentaires afin d'aider à l'interprétation des résultats.

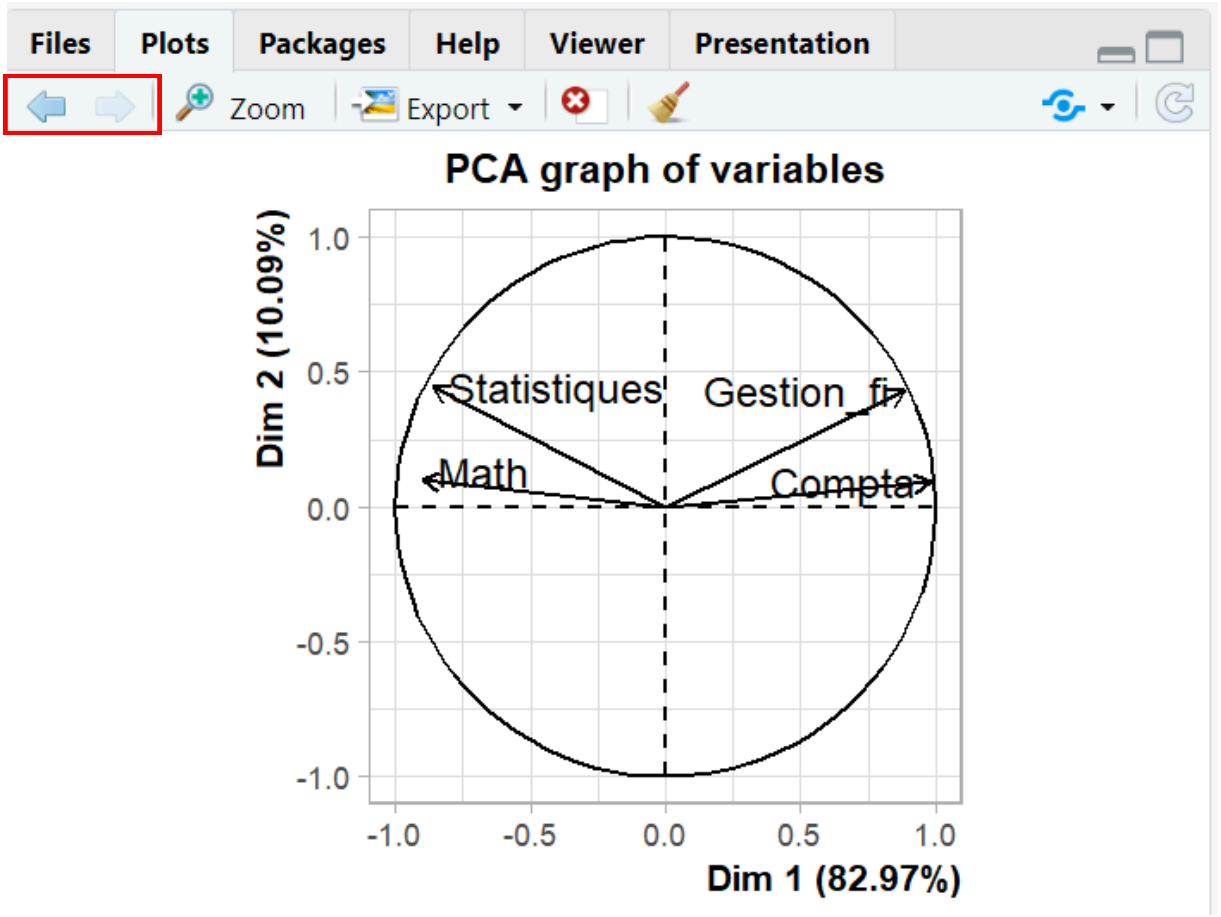
La première étape va donc être d'importer un jeu de données. Dans le cadre du module d'analyse de données, le code permettant d'importer les données vous sera toujours fourni afin que personne ne soit bloqué à cette étape. Les données étant la plupart du temps stockées en ligne, sur le Github de votre enseignant, vous devrez veiller à être connecté à internet pour que l'importation fonctionne.

Avant tout premiers calculs assurez vous que le jeu de données a bien été importé dans R (vous pouvez par exemple utiliser les fonction *dim, head, colnames, str, summary*)

Le calcul de statistiques descriptives constitue une étape préalable à l'ACP afin de calculer les variances des variables et ainsi justifier le centrage et réduction des données (si les variances sont très différentes entre les variables on se retrouve avec des variables qui auraient un poids trop important dans l'inertie du nuage de point initial). Ici, vous pouvez utiliser la fonction *stat.desc* du package *pastecs* (explication plus haut dans ce document) ou bien la fonction `stargazer` du package éponyme (video YouTube).

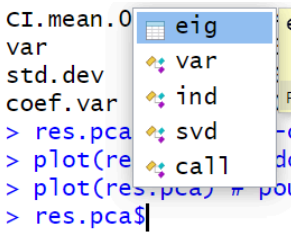
Il est maintenant temps de passer à la réalisation de l'ACP. Nativement, il n'existe pas de fonction de R qui permette de réaliser une ACP. Il faut donc utiliser un package : le package *FactoMineR* (attention à bien respecter les majuscules, R y étant sensible). Vous devez donc l'installer (à ne faire qu'une fois pour toutes), puis à chaque session de travail vous devrez appeler le package afin de charger les fonctions associées dans votre espace de travail : `library(FactoMineR)`. La fonction PCA (pour *principal component analysis* - le nom anglais de l'ACP) de ce package permet de lancer l'ACP. Indiquez comme seul argument de cette fonction le nom de votre jeu de données, en utilisant si besoin les crochets si vous ne souhaitez pas que l'ACP se fasse sur l'ensemble du jeu de données (peut être utile par exemple s'il comporte des indicateurs statistiques comme des moyennes, ou des variables qualitatives). Par exemple, si votre jeu de données se nomme `df0` dans votre environnement de travail et que vous souhaitez réaliser l'ACP uniquement sur les 5 premières colonnes : `res.pca = PCA(df0[,1:5])`, les résultats seront alors enregistrés dans l'objet `res.pca`. R affichera directement le graphique des variables dans le premier plan factoriel. Pour obtenir le graphique des

individus dans le premier plan factoriel, vous avez juste à naviguer vers l'arrière grâce à la flèche prévue à cet effet :



Vous pouvez également utiliser le code : `plot(res.pca)`.

Tous les résultats sont enregistrés dans l'objet res.pca qui est un objet de type 'list', et il contient 5 éléments différents



- `res.pca$eig` : informations sur les valeurs propres associées à chacun des axes (*i.e.* quantité d'information captée par l'axe) et pourcentage d'information capté par chaque axe, pourcentage cumulé.

- `res.pca$var` : informations sur les variables. Si vous faites tourner cela vous observez que R fournit 4 tableaux :
  - `$coord` : coordonnées des variables sur chacune des composantes principales
  - `$cor` : matrice des coefficients de corrélations entre les variables

- `$cos2` : cosinus carré des variables sur chacune des dimensions (= qualité de la représentation). NB : pensez à bien sommer les cos2 sur la dimension 1 + celui sur la dimension 2 afin d'étudier la qualité de représentation dans le premier plan factoriel
- `$contrib` : contribution des variables sur chacune des dimensions (permet de mesurer leur importance dans la construction de la composante principale)

⇒ Pour n'afficher qu'un seul des tableaux il faut donc utiliser un second $: `res.pca$var$cos2` pour le tableau des cos2 par exemple.

- `res.pca$ind` : informations sur les individus. Si vous faites tourner cela vous observez que R fournit 4 tableaux :
  - `$coord` : coordonnées des individus sur chacune des composantes principales
  - `$cos2` : cosinus carré des individus sur chacune des dimensions (= qualité de la représentation). NB : pensez à bien sommer les cos2 sur la dimension 1 + celui sur la dimension 2 afin d'étudier la qualité de représentation dans le premier plan factoriel
  - `$contrib` : contribution des individus sur chacune des dimensions (permet de mesurer leur importance dans la construction de la composante principale)
  - `$dist` : dans le repère initial (avant projection sur les composantes principales) distance au carré entre l'individu et le centre de gravité G (information peu utile pour vous).

- `res.pca$svd` : informations sur la décomposition en valeurs singulières (nous n'avons fait que survoler les choses en CM).
  - `$vs` : racines carrées des valeurs propres
  - `$U` : coordonnées des individus sur les composantes principales, sans tenir compte des valeurs propres. Ces valeurs viendront ensuite déformer ces coordonnées en fonction de l'importance relative qu'elles donnent à chacun des axes (`res.pca$ind$coord` donne les coordonnées sur les axes, en prenant en compte cette "correction")
  - `$V` : Vecteurs propres, qui donnent donc comment sont formées les combinaisons linéaires de variables afin de construire les composantes principales.

- `res.pca$cal` : informations sur les paramètres considérés et les données utilisées pour réaliser l'ACP. Quelques détails :
  - `$row.w` : poids associé à chacune des lignes (=individu). Dans une ACP, par défaut, tous les individus ont le même poids. On pourrait néanmoins remettre en cause cela si par exemple chaque individu est en réalité un groupe d'individus (une tranche d'âge typiquement) dont les effectifs sont différents d'un individu à un autre. Le poids est égal à 1/nombre d'individus. Notons qu'en AFC des poids différents sont appliqués.
  - `$col.w` : poids des variables, il sera toujours de 1 en ACP si les variables ont été centrées réduites
  - `$ncp` : nombre de composantes principales à considérer dans la présentation des résultats
  - `$X` : tableau des données brutes (ce que l'on a donc indiqué comme argument de la fonction PCA)

Afin d'automatiser les choses, on peut regrouper les informations dans un même tableau. Par exemple, pour les informations sur les individus (coordonnées, cos2, contribution), en séparant bien dimension 1 et dimension 2 :

```
infos_ind=round(cbind(res.pca$ind$coord[,1:2],
                      res.pca$ind$contrib[,1:2],
                      res.pca$ind$cos2[,1:2]),
                2)


colnames(infos_ind)=c("coord.axe1","coord.axe2",
                      "contrib.axe1","contrib.axe2",
                      "cos2.axe1","cos2.axe2")
```

Vous pouvez ensuite exporter en format csv cet objet *infos_ind* (fonction `write.csv2`) afin d'améliorer sa présentation dans Excel pour ensuite le coller en mode point dans votre document Word.

Complément

Si vous souhaitez visualiser une matrice des correlations, vous pouvez utiliser une représentation sous forme de carte thermique (*heatmap*). Plus d'informations ICI.

Réalisation d'une AFC

A venir…

Régressions économétriques

A compléter

Importer les données

recup le nom des variables

calculer la variance et la covariance

plot

tracer droite de tendance : abline(lm())

fonction lm

package stargazer affichage ds R et export

Ressources externes

Tutoriels YouTube

1. **3 heures** de tuto qui couvrent les principaux éléments (**à regarder en priorité**) : partie1 + partie2
2. Tutoriel introduction (anglais)
3. Importation données CSV et txt (fonction read.table), il existe aussi une fonction read.csv et read.csv2
4. Exportation de données
5. Principales fonctions
6. Fonction plot (anglais) – barplot et piechart (anglais)

Documentations pdf

- https://math.unice.fr/~malot/manuelR.pdf (20p)
- https://egallic.fr/Enseignement/R/m1_stat_eco_logiciel_R.pdf (232p)
- https://web.maths.unsw.edu.au/~lafaye/textes/livreR2014.pdf (710p)