



Atelier 2: Charger et manipuler des données

Série d'ateliers R du CSBQ

Centre de la science de la biodiversité du Québec



À propos de cet atelier

 REPO

 DEV

 WIKI

02

 5

DIAPOS

02

 2

DIAPOS

02

 SCRIPT

02

Packages requis

- `dplyr`
- `tidyr`
- `magrittr`

```
install.packages(c('dplyr', 'tidyr', 'magrittr'))
```

Objectifs d'apprentissage

1. Créer un projet RStudio
2. Écrire un script dans R
3. Charger, explorer et enregistrer des données
4. Manipuler des jeux de données avec `tidyverse`, `dplyr`, `magrittr`

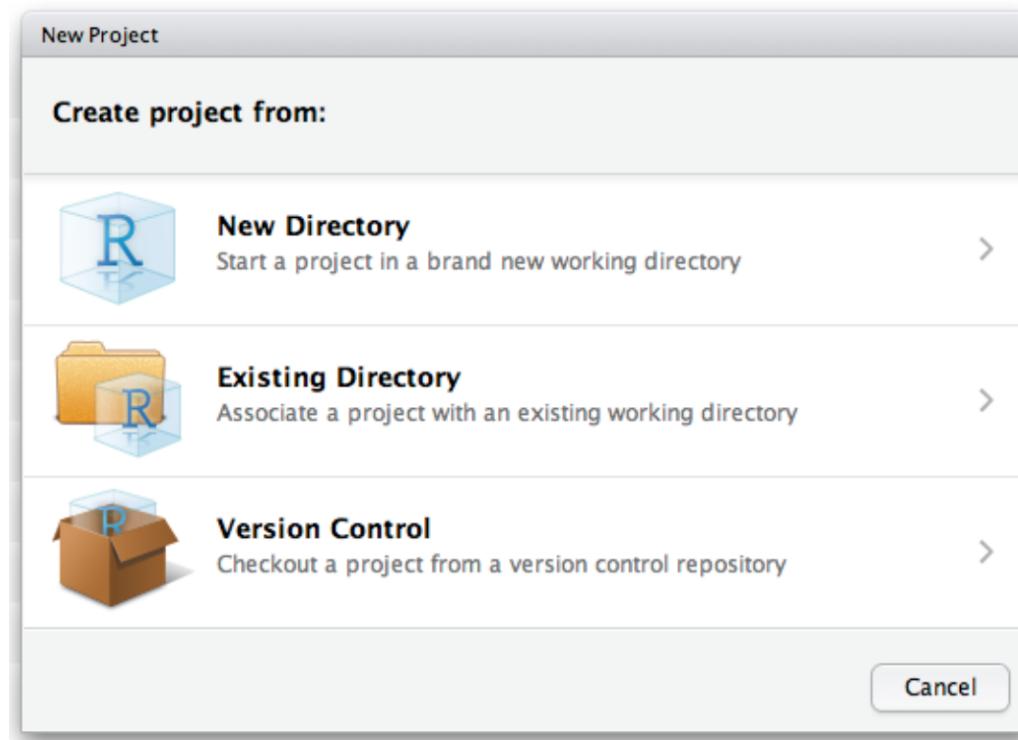
Créer un projet RStudio

Les projets RStudio

- Qu'est-ce qu'un projet RStudio ?
 - Les projets RStudio permettent l'organisation de son travail et l'accès facile à tous les fichiers requis pour une analyse.
 - Tous les fichiers, scripts, et documentation utilisés pour une analyse sont reliés ensemble dans un même projet par un fichier *.Rproj*.
- L'utilisation de projets RStudio facilite la **reproductibilité** et le partage de données, de scripts, et de leur documentation.

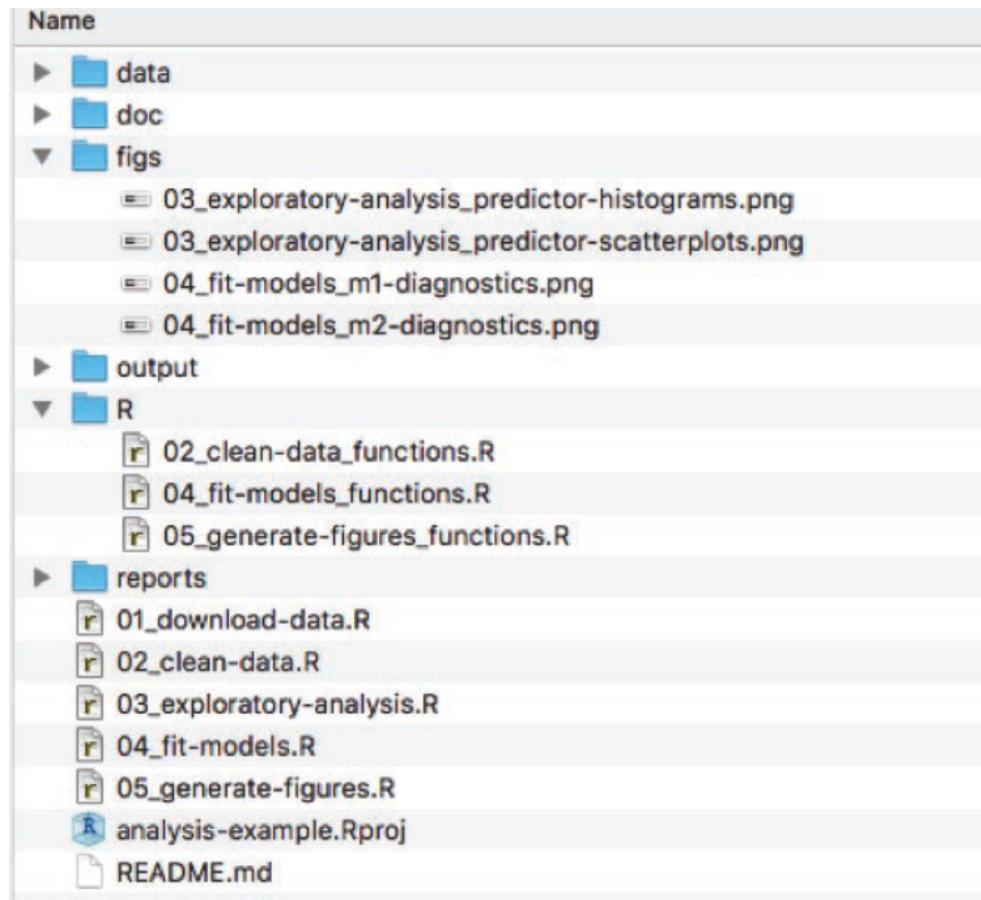
Créer un projet dans RStudio

Pour créer un projet dans RStudio, allez dans le menu Fichier puis sélectionnez **Nouveau projet** (*File -> New Project*).



Un projet = un dossier

Organizez vos fichiers !



Préparer des données pour l'importation dans R

- Vous devriez enregistrer les fichiers à importer dans R en tant que fichiers "comma separated values" (**.csv**)
- Les fichiers .csv peuvent être créés par presque toutes les applications (Excel, GoogleDocs, LibreOffice, etc.)
- Fichier -> Enregistrer sous **.CSV...**



Bien nommer les fichiers

Évitez d'utiliser des espaces, des accents ou des caractères spéciaux pour vos noms

Non:

- final.csv (*pas-informatif !*)
- safnnejs.csv (*C'est n'importe quoi!*)
- 1-4.csv (*Eviter d'utiliser des chiffres!*)
- Ne.pas.separer.par.des.points.csv
(*Peut causer des erreurs de lecture de fichier !*)

Oui:

- rawDatasetAgo2017.csv
- co2_concentrations_QB.csv
- 01_figIntro.R

Bien nommer les variables

- Utilisez des noms de variables courts et informatifs (i.e. "Temps_1" au lieu de "Temps de la première mesure")
- Les valeurs des colonnes doivent correspondre à l'usage prévu

Bien !

- Measurements
- SpeciesNames
- Site

Pas bien !

- a
- 3
- supernomunpeutroplong

Regardez le guide **tidyverse** pour plus de conseils.

Conseils pour préparer les données

- Pas de texte dans les colonnes de mode numérique
- Pas d'espace
- Identifiez les valeurs manquantes par NA ("not available")
- Faites attention aux erreurs typographiques!
- Évitez les valeurs numériques pour les variables n'ayant pas un sens numérique (i.e. individu, réplicat, traitement)
- Utilisez un format uniforme pour les dates, les chiffres, etc.
- N'ajoutez pas de notes, d'entêtes supplémentaires, ou de cellules fusionnées!
- Une variable par colonne!

Lire le paper de [Broman & Woo \(2017\)](#) pour plus de conseils sur l'organisation des données.

Exemples de mauvaises habitudes:

C	D	E	F
Quebec	chilled	175	24.1
Quebec	chilled	250	30.3
Quebec	chilled	350	34.6
Quebec	chilled	500	32.5
Quebec	chilled	675	35.4
Quebec	chilled	1000	38.7
Quebec	chilled	95	9.3
Quebec	chilled	175	27.3

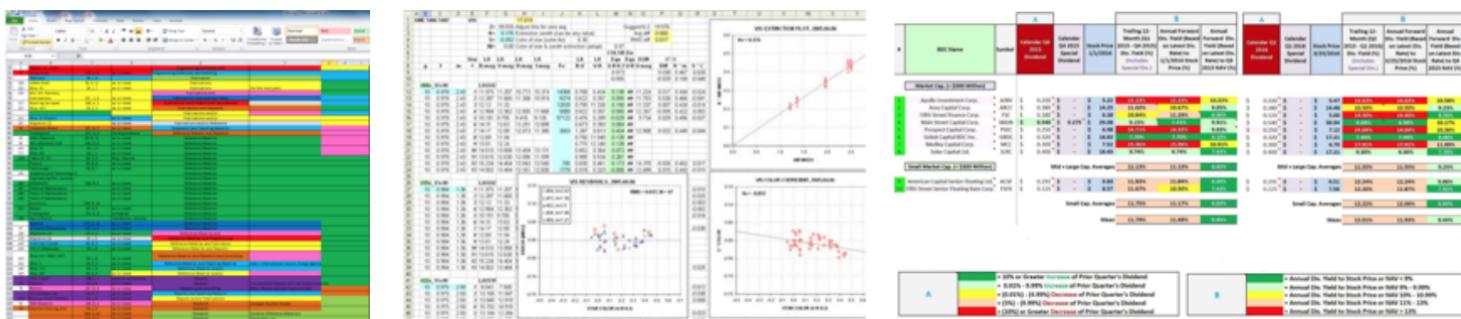
D	E	F	G
nonchilled	1000	39.7	
nonchilled	95	13.6	
nonchilled	175	27.3	
nonchilled	250	37.1	
nonchilled	350	41.8	
nonchilled	500	40.6	
nonchilled	675	cannot_read_notes	
nonchilled	1000	44.3	
nonchilled	95	16.2	



A	B	C	D	E	F	G	H	I	J	K
NOTE: It rain a lot in Quebec during sampling, due to excessive water falling on my notebook numerous values can't be read										
falling on my notebook numerous values can't be read										
4	Plant	Type	Treatment	conc	uptake					
5	1 Qn1	Quebec	nonchilled	95	16					
6	2 Qn1	Quebec	nonchilled	250	30.4					
7	4 Qn1	Quebec	nonchilled	350	37.2					
8	5 Qn1	Quebec	nonchilled	500	35.3					
9	6 Qn1	Quebec	nonchilled	cannot_reac	39.2	Quebec	429.15	17595		
10	7 Qn1	Quebec	nonchilled	1000	39.7	Mississippi	435.00	18270		
11	8 Qn2	Quebec	nonchilled	95	13.6					
12	9 Qn2	Quebec	nonchilled	175	27.3					
13	10 Qn2	Quebec	nonchilled	250	37.1					
14	11 Qn2	Quebec	nonchilled	350	41.8					
15	12 Qn2	Quebec	nonchilled	500	40.6					
16	13 Qn2	Quebec	nonchilled	675	cannot_read					
17	14 Qn2	Quebec	nonchilled	1000	44.3					
18	15 Qn3	Quebec	nonchilled	95	16.2					
19	16 Qn3	Quebec	nonchilled	175	32.4					
20	17 Qn3	Quebec	nonchilled	250	40.3					
21	18 Qn3	Quebec	nonchilled	350	42.1					
??	19 Qn3	Quebec	nonchilled	500	42.9					



Exemples de très mauvaises présentations



data.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Site	Date	Plot	Species	Weight	Adult		Rodent Trapping 3/15/2010						
2	DeepWell	2/13/2010		1 DIPO	12.1	j		Site	Plot	Adult	RodentSp	Weight		
3	Deep Well	Feb-10		2 Pero	13.22	j		DW		1 y	Pero	12		
4	rioSalado	2/13/2010	1a	pero	16	N		RS		2 j	PERO	escaped <15		
5	riuSladu	"	1+	CleGap	18.92	gut away		RS		3 n	Clegap	91		
6				Mean1	15.06									
7														
8														
9														
10														

Préparer ses données dans R

Il est possible de faire toute la préparation des données dans R. Les avantages sont :

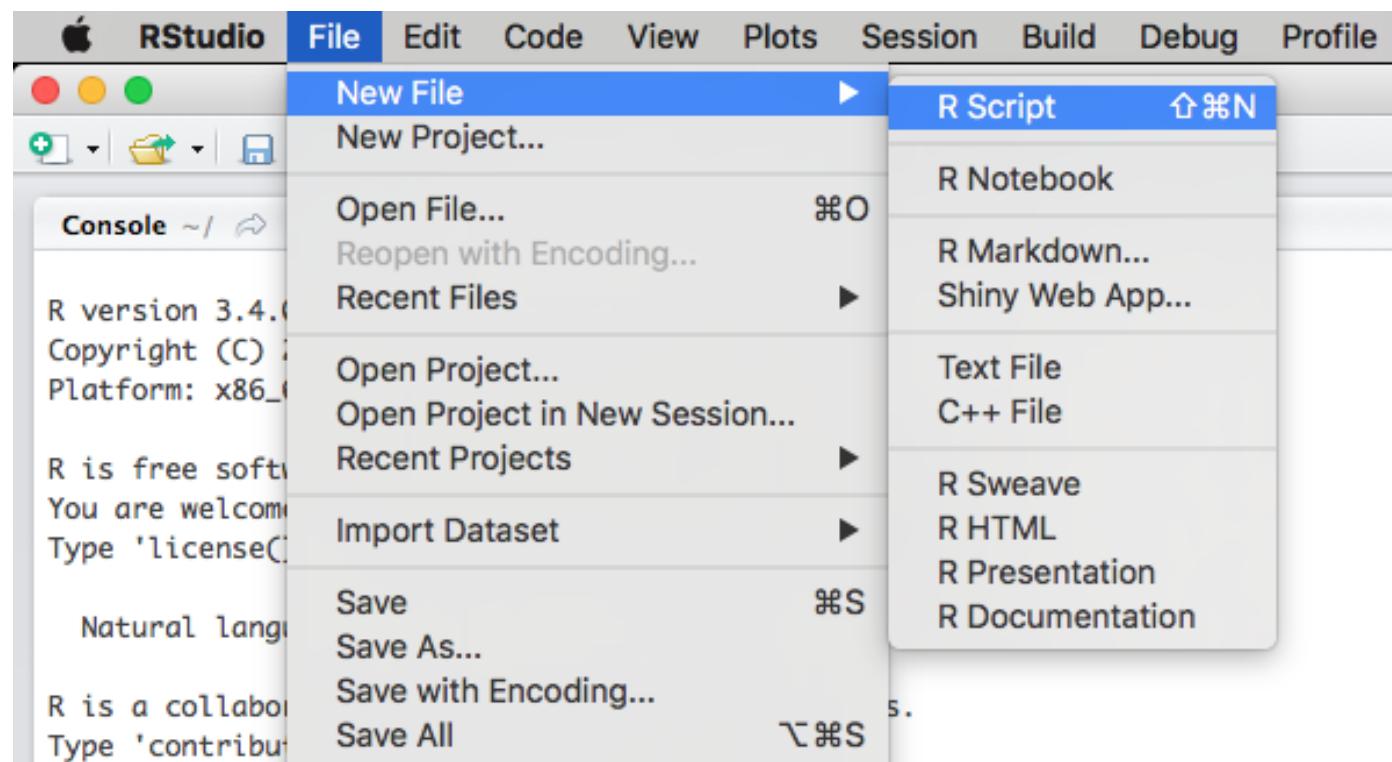
- On économise du temps pour les grosses bases de données
- On préserve les données d'origine
- On peut basculer entre les modes "long" et "large" très facilement (plus de détails plus tard)
- Pour des informations complémentaires, consultez la page suivante :
<https://www.zoology.ubc.ca/~schluter/R/data/>

Écrire un script dans R

Les scripts R

- Un script R c'est
 - un fichier texte contenant toutes les commandes nécessaires pour réaliser un projet.
 - Une fois écrit et enregistré, votre script R vous permettra d'apporter des changements et de refaire des analyses avec un minimum d'effort.
 - Sélectionnez simplement une commande et appuyez sur "Run" ou sur 'command-enter' (Mac) ou 'ctrl-enter' (PC).

Créer un script dans R



Écrire un script dans R

Commandes & Commentaires

Utilisez le symbole `#` pour insérer des commentaires au sein d'un script. Ceci indique à R d'ignorer tout ce qui se trouve à la suite du symbole `#` lors de l'exécution de commandes.

```
# Ceci est un commentaire pas une commande R!
```

Commandes & Commentaires

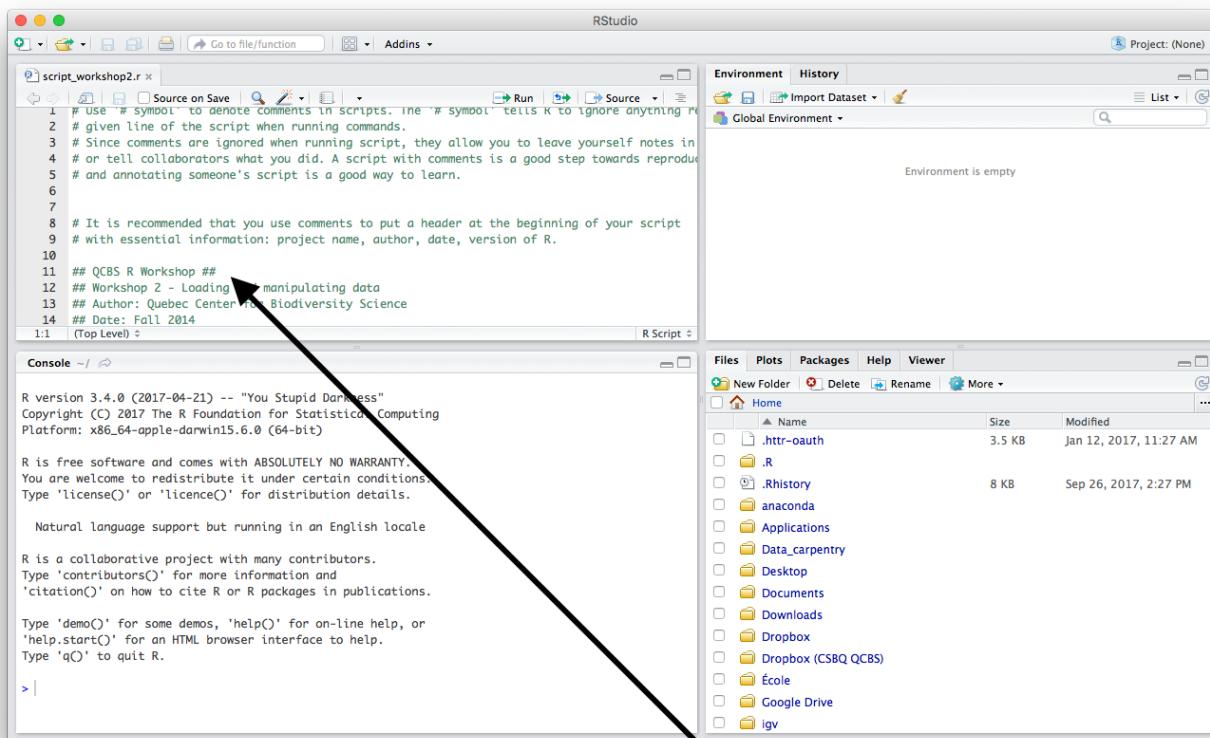
Annoter son script est un bon moyen de :

- se rappeler ce que vous avez fait
- dire aux collaborateurs ce que vous avez fait
- favoriser une science reproductible

Soyez aussi détaillé que possible !

Entêtes de section

Il est recommandé de commencer vos scripts avec un entête pour indiquer des infos importantes: nom du projet, auteur, date, version de R, etc.



The screenshot shows the RStudio interface with the following details:

- Script Editor:** The file `script_workshop2.r` is open. The code includes a header at the top:

```
1 # use '# symbol' to denote comments in scripts. The '#' symbol tells R to ignore anything written
2 # given line of the script when running commands.
3 # Since comments are ignored when running script, they allow you to leave yourself notes in
4 # or tell collaborators what you did. A script with comments is a good step towards reproducing
5 # and annotating someone's script is a good way to learn.
6
7
8 # It is recommended that you use comments to put a header at the beginning of your script
9 # with essential information: project name, author, date, version of R.
10
11 ## QCBS R Workshop ##
12 ## Workshop 2 - Loading and manipulating data
13 ## Author: Quebec Center for Biodiversity Science
14 ## Date: Fall 2014
```
- Console:** Displays the R startup message and basic help text.
- Environment:** Shows the global environment is empty.
- File Explorer:** Shows the local directory structure with files like `.httr-oauth`, `.R`, `.Rhistory`, `anaconda`, `Applications`, `Data_carpentry`, `Desktop`, `Documents`, `Downloads`, `Dropbox`, `Dropbox (CSBQ QCBS)`, `École`, `Google Drive`, and `igv`.

Header

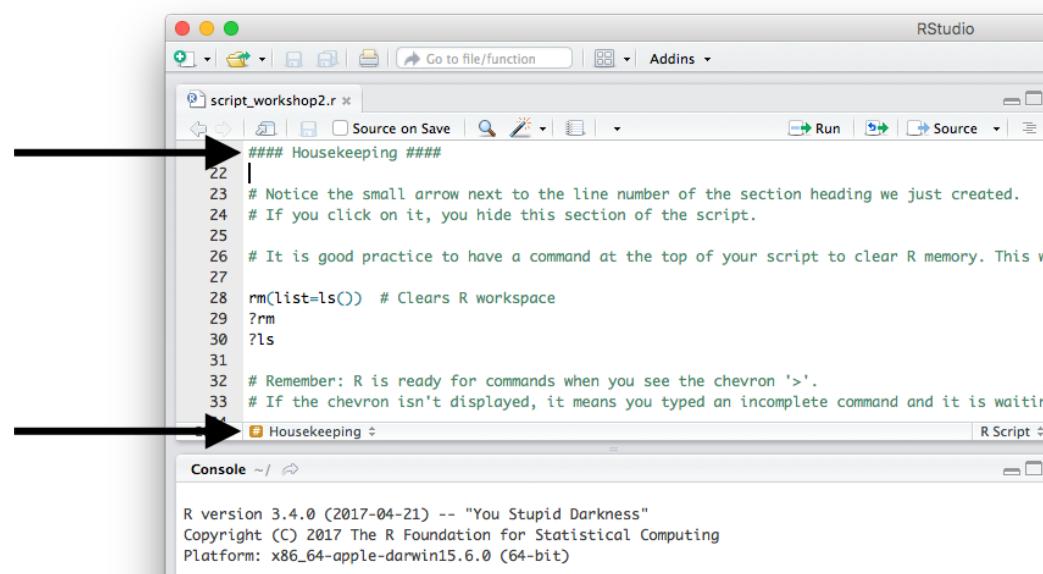
Entêtes de section

Sur R studio, Vous pouvez utiliser quatre symboles `#` de suite pour créer un entête de section.

Par exemple :

```
#### Chargement du fichier de données ####
```

Cela vous permet de passer rapidement d'une section à l'autre et de masquer des sections.



Nettoyage

C'est une bonne habitude de maintenir son espace de travail propre en effaçant la mémoire de R à l'aide de `rm(list=ls())`.

```
rm(list = ls()) # Efface ce qui se trouve dans l'espace de travail  
?rm  
?ls
```

Cette commande permet d'éviter d'utiliser un vieux jeu de données qui serait resté dans l'espace de travail.

Nettoyage

On peut tester cette commande en ajoutant des objets dans l'espace de travail pour mieux comprendre comment `rm(list=ls())` fonctionne.

```
A<- "Test" # On crée un objet "A".
A <- "Test" # Utilisez des espaces - plus facile à lire
A = "Test"

# Note: il est recommandé d'utiliser "<-" pour l'assigment au lieu de "="

# Visualiser des objets en mémoire
ls()
# [1] "A"

A
# [1] "Test"

# Nettoyer la mémoire
rm(list=ls())

A
# Error in eval(expr, envir, enclos): object 'A' not found
```

Petit rappel important

- R est prêt à exécuter une commande lorsque vous voyez le chevron `>` affiché dans la console. Si le chevron n'apparaît pas, c'est qu'une commande est incomplète. Appuyez sur 'ESC' pour sortir de cette commande.
- R est sensible à la casse, i.e. `A` est différent de `a`.

```
a <- 10
A <- 5
a
# [1] 10
A
# [1] 5
rm(list=ls()) # On nettoie l'espace de travail à nouveau !
```

Charger, explorer et enregistrer des données

Télécharger les données

Vous pouvez télécharger les données et le script depuis :

<http://qcbs.ca/wiki/r/workshop2>

Enregistrez les fichiers dans le dossier où vous avez créé votre projet R.

NOTE Il existe des données déjà disponibles sur R

```
# Liste complète de tous les données disponibles sur base R  
library(help = "datasets")
```

Répertoire de travail

Si vous n'utilisez pas un **projet RStudio**, vous devez indiquer à R le répertoire où se trouvent les fichiers de données afin de les charger.

Pour voir quel répertoire R utilise :

```
getwd()
```

Si ce n'est pas le répertoire avec lequel vous souhaitez travailler, vous pouvez définir le vôtre à l'aide de:

```
setwd("C:/Users/mon_repertoire")
```

Il est recommandé de créer un projet RStudio et ne pas utiliser `setwd()` pour faciliter la reproductibilité

Afficher le contenu du répertoire de travail

La fonction `dir()` affiche le contenu du répertoire de travail.

```
dir()  
# [1] "assets"                 "data"                  "images"  
# [4] "qcbsR-fonts.css"        "qcbsR-header.html"  "qcbsR-macros.js"  
# [7] "qcbsR.css"               "workshop02-fr_files" "workshop02-fr.html"  
# [10] "workshop02-fr.Rmd"
```

Vous pouvez vérifier:

- Si le fichier que vous voulez ouvrir se trouve dans le répertoire de travail
- L'orthographe du nom du fichier (e.g. 'monfichier.csv' au lieu de 'MonFichier.csv')

Importer un jeu de données

Utilisez la fonction `read.csv()` pour importer des données provenant d'un fichier .csv dans R.

```
co2 <- read.csv("data/co2_good.csv", header=TRUE)
```

- Cette commande va créer un objet nommé `co2`
- Le nom du fichier est écrit entre guillemets (`'file'` ou `"file"`)
- Si vous voulez charger un fichier d'un autre répertoire, vous devrez écrire l'extension complète: `read.csv("C:/Users/Mario/Downloads/co2_good.csv")`
- `header = TRUE` permet de spécifier que la première ligne du fichier contient le nom des colonnes

Importer un jeu de données

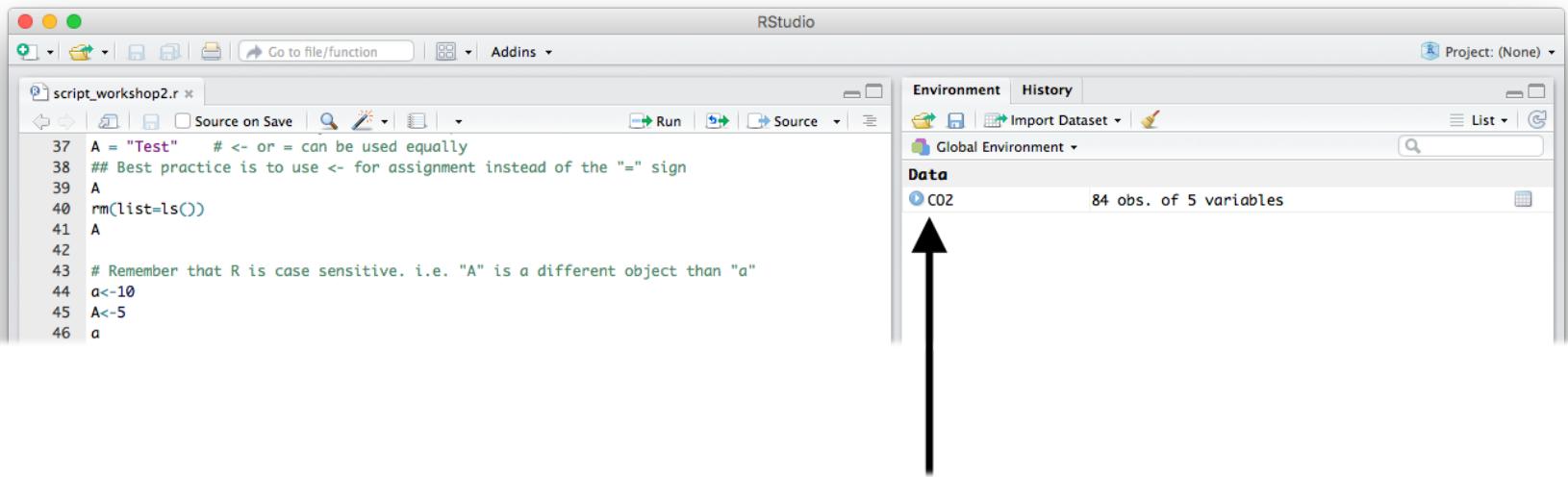
Rappelez-vous que vous pouvez obtenir de l'aide pour une fonction avec [?](#)

?read.csv

NOTE Si vous utilisez un système d'exploitation en français ou un éditeur CSV, il est possible que vous deviez utiliser la fonction `read.csv2()` pour importer correctement un fichier.

?read.csv2

Importer un jeu de données



Prenez note que RStudio montre maintenant le jeu de données CO2 dans votre **espace de travail**.

L'espace de travail inclut tous les objets créés pendant la session R.

Visualiser les données

R code	action
<code>c02</code>	Affiche le tableau de données complet dans la console
<code>head(c02)</code>	Affiche les premières lignes du tableau de données
<code>tail(c02)</code>	Affiche les dernières lignes du tableau de données
<code>names(c02)</code>	Affiche le nom des colonnes du tableau de données
<code>attributes(c02)</code>	Affiche les attributs du tableau de données
<code>dim(c02)</code>	Affiche la dimension du tableau de données
<code>ncol(c02)</code>	Affiche le nombre de colonnes du tableau de données
<code>nrow(c02)</code>	Affiche le nombre de lignes du tableau de données
<code>summary(c02)</code>	Calcule quelques statistiques de base sur les variables

NOTE Ces fonctions sont aussi utilisés pour d'autres objets tels que `vector`, `matrix`, `list`, `array`, etc.

Visualiser les données

```
str(CO2)
# 'data.frame': 84 obs. of 5 variables:
# $ Plant      : chr "Qn1" "Qn1" "Qn1" "Qn1" ...
# $ Type       : chr "Quebec" "Quebec" "Quebec" "Quebec" ...
# $ Treatment: chr "nonchilled" "nonchilled" "nonchilled" "nonchilled" ...
# $ conc       : int 95 175 250 350 500 675 1000 95 175 250 ...
# $ uptake     : num 16 30.4 34.8 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

La fonction `str()` est très utile pour identifier le type/mode de chaque colonne.

Note : Le jeu de données `CO2` contient des mesures répétées d'absorption de CO2 prises sur 6 plantes du Québec et 9 plantes du Mississippi à différentes concentrations de CO2 ambiant. La moitié des plantes de chaque région a subi un traitement de refroidissement la veille du début de l'expérience.

Visualiser les données

Problèmes fréquents lors de l'importation des données :

- Les facteurs apparaissent comme des chaînes de caractères (et vice versa)
- Les facteurs ont trop de niveaux à cause d'une erreur de frappe
- Les données numériques sont stockées sous forme de chaînes de caractères à cause d'une erreur de frappe

Exercice

Chargez les données de nouveau en utilisant le script suivant :

```
co2 <- read.csv("data/co2_good.csv", header = FALSE)
```

Vérifiez la structure des données avec la fonction `str()`.

Quel est le problème ?

N'oubliez pas de recharger les données avec l'argument `header=TRUE` avant de continuer.

Rappel de l'atelier 1 : Accéder aux données

Plusieurs façons d'extraire les données avec les crochets. Considérons un jeu de données nommé `mydata`.

Variable1	Variable2	Variable3	Variable4

```
mydata[1,] # Extrait la 1ère ligne  
mydata[2,3] # Extrait la 2ème ligne / 3ème colonne  
mydata[,1] # Extrait la 1ère colonne  
mydata[,1][2] # [...] peut être utilisé récursivement  
mydata$Variable1 # Extrait la colonne "Variable1"
```

Renommer les variables

On peut renommer les variables (colonnes) dans R.

```
# Créer une copie du jeu de données qu'on pourra modifier
CO2copy <- CO2

# names() donne les noms des variables présentes dans le jeu de données
names(CO2copy)
# [1] "Plant"      "Type"       "Treatment"  "conc"        "uptake"

# Changer des noms anglais pour des noms français
names(CO2copy) <- c("Plante", "Categorie", "Traitement", "conc", "absorption")
names(CO2copy)
# [1] "Plante"      "Categorie"   "Traitement"  "conc"        "absorption"
```

Créer des nouvelles variables

On peut facilement créer et produire des nouvelles variables. Par exemple, la fonction `paste()` permet la concaténation de chaînes de caractères et de variables. Consultez `?paste` et `?paste0`.

Créer un ID unique pour les échantillons avec la fonction `paste0()`

```
# N'oubliez pas d'utiliser "" pour les chaînes de caractères
CO2copy$uniqueID <- paste0(CO2copy$Plante,
                           "_", CO2copy$Categorie,
                           "_", CO2copy$Traitement)

# Observer les résultats
head(CO2copy$uniqueID)
# [1] "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchilled"
# [4] "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchilled"
```

Créer des nouvelles variables

On peut aussi créer des nouvelles variables à partir de chiffres et d'opérations mathématiques!

```
# Standardizer la variable "absorption" en valeurs relatives  
CO2copy$absorptionRel <- CO2copy$absorption/max(CO2copy$absorption)  
  
# Observer les résultats  
head(CO2copy$absorptionRel)  
# [1] 0.3516484 0.6681319 0.7648352 0.8175824 0.7758242 0.8615385
```

Sous-ensemble d'un data frame

Il existe plusieurs façons d'en faire :

```
# On continue à travailler avec notre jeux de données CO2copy  
  
# Extraire un sous-ensemble par un nom de variable  
CO2copy[, c("Plante", "absorptionRel")]  
  
# Extraire un sous-ensemble de rangées  
CO2copy[1:50, ]
```

Sous-ensemble d'un data frame

```
# Extraire les observations du traitement "nonchilled"
CO2copy[CO2copy$Traitement == "nonchilled", ]  
  
# Extraire selon une condition numérique
CO2copy[CO2copy$absorption >= 20, ]  
  
# Extraire selon plusieurs conditions numériques
CO2copy[CO2copy$Traitement == "nonchilled" & CO2copy$absorption >= 20, ]  
  
# Nous avons fini de modifier le data frame CO2copy, effaçons-le
rm(CO2copy)
```

Consultez [ici](#) pour voir les opérateurs logiques pouvant être utilisés pour extraire des sous-ensembles de données dans R.

Explorer les données

Un bon moyen de commencer votre exploration des données consiste à regarder des statistiques de base sur votre jeu de données.

Utilisez la fonction `summary()`.

```
summary(CO2)
```

C'est également utile pour repérer certaines erreurs que vous auriez peut-être manquées!

Explorer les données

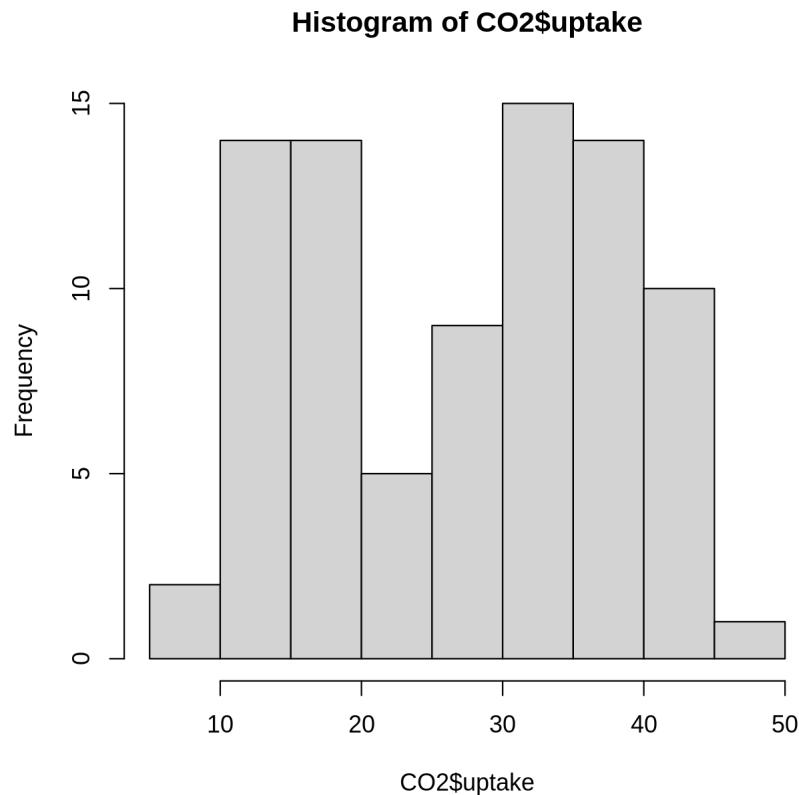
Vous pouvez également utiliser d'autres fonctions pour calculer des statistiques de base pour des parties spécifiques de votre trame de données.

Essayons les fonctions `mean()`, `sd()` et `hist()` :

```
# Calculer la moyenne et l'écart type de la concentration,  
# et les assigner à de nouvelles variables  
meanConc <- mean(CO2$conc)  
sdConc <- sd(CO2$conc)  
  
# print() imprime une valeur donnée dans la console R  
print(paste("the mean of concentration is:", meanConc))  
# [1] "the mean of concentration is: 435"  
  
print(paste("the standard deviation of concentration is:", sdConc))  
# [1] "the standard deviation of concentration is: 295.924119222056"
```

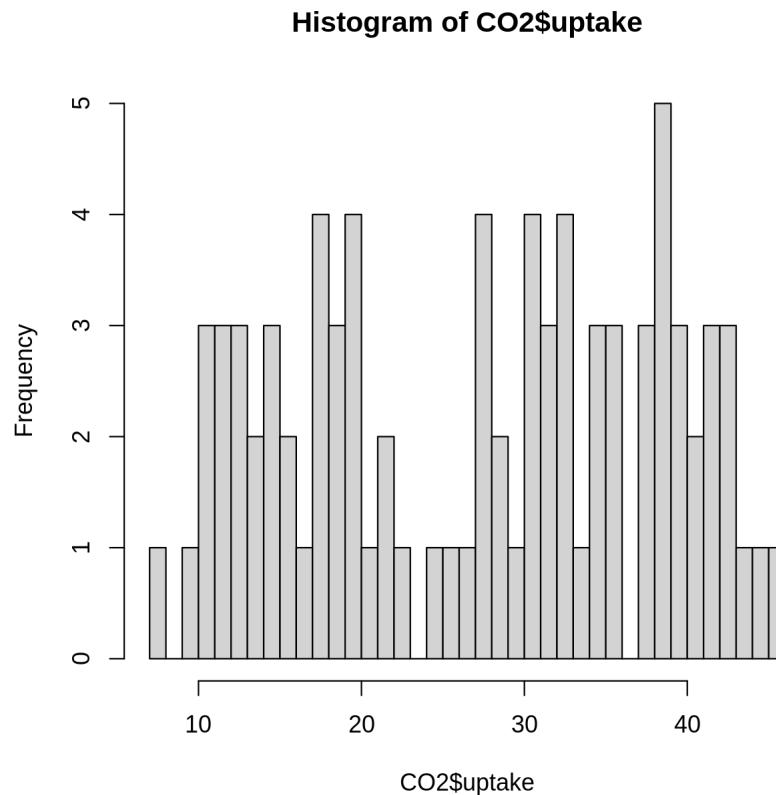
Explorer les données

```
# Créons un histogramme pour explorer la distribution de "uptake"  
hist(CO2$uptake)
```



Explorer les données

```
# Cr ons un histogramme pour explorer la distribution de "uptake"  
hist(CO2$uptake, breaks = 40) # breaks permet de changer le nombre de classes
```



Enregistrer son espace de travail

```
# Sauver l'espace de travail avec tous les objets
save.image(file="data/co2_project_Data.RData")

# Nettoyer l'espace de travail
rm(list = ls())

# Charger tout ce qui se trouvait dans l'espace de travail
load("data/co2_project_Data.RData")
head(CO2) # cela fonctionne! :)
```

```
#   Plant    Type Treatment conc uptake
# 1   Qn1 Quebec nonchilled  95   16.0
# 2   Qn1 Quebec nonchilled 175   30.4
# 3   Qn1 Quebec nonchilled 250   34.8
# 4   Qn1 Quebec nonchilled 350   37.2
# 5   Qn1 Quebec nonchilled 500   35.3
# 6   Qn1 Quebec nonchilled 675   39.2
```

Exporter des données

Pour enregistrer dans le répertoire de travail des données que vous avez créées ou modifiées dans R, utilisez la fonction `write.csv()`.

```
write.csv(co2, file = "data/co2_new.csv")
```

`co2` → Nom de l'objet dans R

`"co2_new.csv"` → Nom du nouveau fichier à enregistrer



Utilisez vos données

- Essayez de charger, explorer, et enregistrer vos propres données dans R
- *Si ce n'est pas le cas, essayez de corriger vos données dans Excel.*
- *Enregistrez vos données corrigées et ré-essayez de les ouvrir dans R.*
- *Si vous n'avez pas de données, travaillez avec vos voisins*
- *N'oubliez pas de nettoyer votre espace de travail*

Réparer un jeu de données

Réparer un jeu de données "endommagé"

Charger vos données peut être plus difficile que vous ne le pensez!

Les jeux de données peuvent être désordonnés et incompatibles entre certains systèmes (Mac, Windows) ou entre ordinateurs.

Pratiquons-nous à réparer certains problèmes communs.



Défi

Lire le fichier `co2_broken.csv`

```
CO2 <- read.csv("data/co2_broken.csv")
head(CO2) # C'est le bordel!
#           NOTE.. It.rain.a.lot.in.Quebec.during.sampling due.to.excessive X
# 1 falling on my notebook numerous values can't be read rain             NA NA
# 2                               Plant\tType\tTreatment\tconc\tuptake            NA NA
# 3                               Qn1\tQuebec\tnonchilled\t95\t16            NA NA
# 4                               Qn1\tQuebec\tnonchilled\t175\t30.4          NA NA
# 5                               Qn1\tQuebec\tnonchilled\t250\tcannot_read_notes    NA NA
# 6                               Qn1\tQuebec\tnonchilled\t350\t37.2          NA NA
#   X.1 X.2 X.3
# 1  NA  NA  NA
# 2  NA  NA  NA
# 3  NA  NA  NA
# 4  NA  NA  NA
# 5  NA  NA  NA
# 6  NA  NA  NA
```



Défi

Voici quelques fonctions qui peuvent vous aider :

- `read.csv()` - examinez les options permettant de charger un fichier .csv
- `head()` - montre les premières lignes
- `str()` - structure de données
- `class()` - classe de l'objet
- `unique()` - observations uniques
- `levels()` - niveaux d'un facteur
- `which()` - pose une question sur votre bloc de données
- `droplevels()` - supprime les niveaux indésirables après avoir déduit les facteurs

Indice Il y a quatre problèmes avec ce jeu de données!

Jeu de données "endommagé"

ERREUR 1

Les données sont contenues dans une seule colonne

```
head(CO2)
```

```
#           NOTE..It.rain.a.lot.in.Quebec.during.sampling due.to.excessive X
# 1 falling on my notebook numerous values can't be read rain             NA NA
# 2                               Plant\tType\tTreatment\tconc\tuptake            NA NA
# 3                               Qn1\tQuebec\tnonchilled\t95\t16            NA NA
# 4                               Qn1\tQuebec\tnonchilled\t175\t30.4          NA NA
# 5                               Qn1\tQuebec\tnonchilled\t250\tcannot_read_notes    NA NA
# 6                               Qn1\tQuebec\tnonchilled\t350\t37.2          NA NA
#
#   X.1 X.2 X.3
# 1  NA  NA  NA
# 2  NA  NA  NA
# 3  NA  NA  NA
# 4  NA  NA  NA
# 5  NA  NA  NA
# 6  NA  NA  NA
```

Jeu de données "endommagé"

ERREUR 1 - Solution

- Importez les données de nouveau, en spécifiant comment chaque valeur est séparée.
- L'argument `sep` indique à R quel type de caractère sépare les valeurs sur chaque ligne.
- Ici, une tabulation sépare les valeurs au lieu d'une virgule.

```
co2 <- read.csv("data/co2_broken.csv", sep = "")
```

Jeu de données "endommagé"

ERREUR 2

Les données ne commencent pas avant la 3ème ligne. Les entêtes de colonnes sont remplacés par des notes.

head(CO2)

```
#      NOTE.      It      rain      a          lot      in. Quebec during
# 1 falling    on      my notebook      numerous values can't      be
# 2 Plant     Type Treatment      conc      uptake
# 3 Qn1 Quebec nonchilled      95      16
# 4 Qn1 Quebec nonchilled      175      30.4
# 5 Qn1 Quebec nonchilled      250 cannot_read_notes
# 6 Qn1 Quebec nonchilled      350      37.2
# sampling. due to excessive X....
# 1      read rain,,, NA      NA      NA
# 2                      NA      NA      NA
# 3                      NA      NA      NA
# 4                      NA      NA      NA
# 5                      NA      NA      NA
# 6                      NA      NA      NA
```

Jeu de données "endommagé"

ERREUR 2 - Solution

Pour régler ce problème, vous devez indiquer à R de sauter les deux premières lignes avec l'argument "skip".

```
CO2 <- read.csv("data/co2_broken.csv", sep = "", skip = 2)
head(CO2)

#   Plant    Type Treatment      conc       uptake
# 1 Qn1 Quebec nonchilled      95          16
# 2 Qn1 Quebec nonchilled     175         30.4
# 3 Qn1 Quebec nonchilled    250  cannot_read_notes
# 4 Qn1 Quebec nonchilled    350         37.2
# 5 Qn1 Quebec nonchilled    500         35.3
# 6 Qn1 Quebec nonchilled cannot_read_notes      39.2
```

Jeu de données "endommagé"

ERREUR 3 Les variables `conc` et `uptake` sont considérées comme des facteurs au lieu de nombres, car il y a du texte dans ces colonnes.

```
str(CO2)
# 'data.frame': 84 obs. of 5 variables:
# $ Plant      : chr  "Qn1" "Qn1" "Qn1" "Qn1" ...
# $ Type       : chr  "Quebec" "Quebec" "Quebec" "Quebec" ...
# $ Treatment  : chr  "nonchilled" "nonchilled" "nonchilled" "nonchilled" ...
# $ conc        : chr  "95" "175" "250" "350" ...
# $ uptake      : chr  "16" "30.4" "cannot_read_notes" "37.2" ...
```



```
unique(CO2$conc)
# [1] "95"                  "175"                 "250"
# [4] "350"                 "500"                 "cannot_read_notes"
# [7] "1000"                "675"
```

Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

na.strings

a character vector of strings which are to be interpreted as [NA](#) values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields. Note that the test happens *after* white space is stripped from the input, so na.strings values may need their own white space stripped in advance.

Jeu de données "endommagé"

ERREUR 3 - Solution

Indiquez à R que tous les éléments NA, "na" et "cannot_read_notes" doivent être considérés comme des NA. Ensuite, comme toutes les autres valeurs de ces colonnes sont des nombres, `conc` et `uptake` seront chargés sous forme numérique / entier.

```
co2 <- read.csv("data/co2_broken.csv", sep = "", skip = 2,
                 na.strings = c("NA", "na", "cannot_read_notes"))
str(co2)
# 'data.frame': 84 obs. of 5 variables:
# $ Plant      : chr  "Qn1" "Qn1" "Qn1" "Qn1" ...
# $ Type       : chr  "Quebec" "Quebec" "Quebec" "Quebec" ...
# $ Treatment: chr  "nonchilled" "nonchilled" "nonchilled" "nonchilled" ...
# $ conc       : int  95 175 250 350 500 NA 1000 95 175 250 ...
# $ uptake     : num  16 30.4 NA 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

Jeu de données "endommagé"

ERREUR 4

En réalité, il y a seulement 2 traitements (chilled & non chilled), mais des erreurs d'orthographe créent 2 autres niveaux de traitement.

```
str(CO2)
```

```
levels(CO2$Treatment)
# NULL
unique(CO2$Treatment)
# [1] "nonchilled" "nnchilled"   "chilled"      "chiled"
```

Jeu de données "endommagé"

ERREUR 4 - Solution

```
# Identifier toutes les lignes contenant "nnchilled" et remplacer par "nonchilled"
CO2$Treatment[which(CO2$Treatment=="nnchilled")] <- "nonchilled"

# Faisons la même chose pour "chiled" :
CO2$Treatment[which(CO2$Treatment=="chiled")] <- "chilled"
```

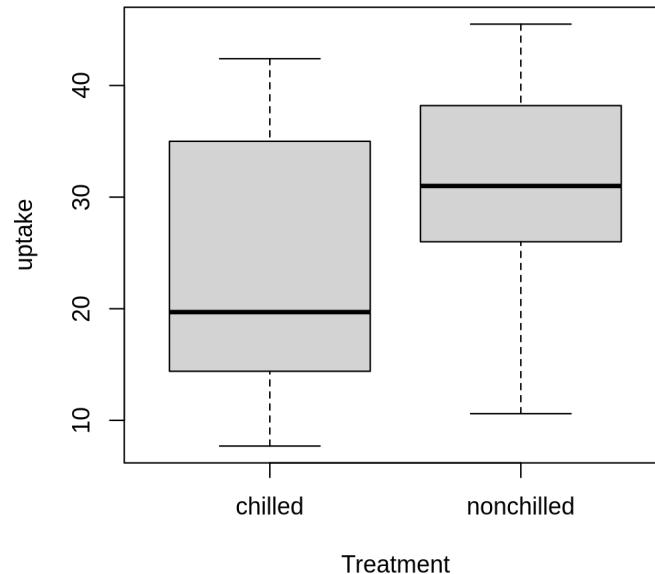
Jeu de données "endommagé"

ERREUR 4 - Solution

Après avoir réparé les facteurs, il faut enlever les niveaux de facteur non utilisés

Sinon :

```
boxplot(uptake ~ Treatment, data = c02)
```

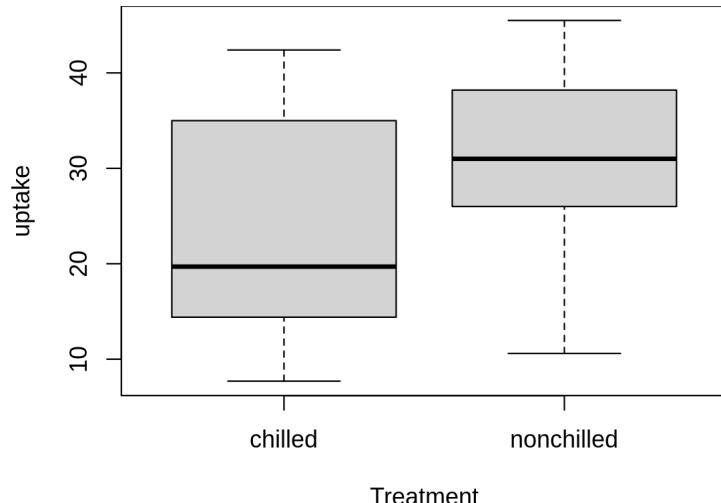


Jeu de données "endommagé"

ERREUR 4 - Solution

```
C02 <- droplevels(C02)
str(C02)
# 'data.frame': 84 obs. of 5 variables:
# $ Plant      : chr  "Qn1" "Qn1" "Qn1" "Qn1" ...
# $ Type       : chr  "Quebec" "Quebec" "Quebec" "Quebec" ...
# $ Treatment: chr  "nonchilled" "nonchilled" "nonchilled" "nonchilled" ...
# $ conc       : int  95 175 250 350 500 NA 1000 95 175 250 ...
# $ uptake     : num  16 30.4 NA 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

```
boxplot(uptake ~ Treatment, data = C02)
```



Bienvenue dans le monde de tidyverse !

Qu'est ce que Tidyverse?

Tidyverse est un ensemble de packages conçus pour la manipulation de données en science.

Tidyverse contient des outils plus efficaces et conviviales très utiles pour toutes sortes d'analyses.

Tous les packages inclus dans tidyverse sont automatiquement installés lors de l'installation de tidyverse : `install.packages("tidyverse")`

Par exemple :

- tidy: réorganiser les tables de données
- dplyr: manipuler les données
- magrittr: lier plusieurs opérations
- ggplot2: faire des graphiques
- readr: lire les données (plus rapide !)
- lubridate: manipuler des données de dates et de temps
- et plus !

Manipulation de données avec **tidyverse**

tidy pour réorganiser ses données

```
library(tidyr)
```



Format de données

Format large (wide)

Un data frame en format large contient une colonne pour chaque variable ou facteur

```
#   Species DHP Haut  
# 1  Chene  12  56  
# 2  Orme   20  85  
# 3  Frene   13  55
```

Format long

Un data frame en format long contient une colonne par variable, où chaque ligne est une observation

```
#   Species dimension cm  
# 1  Chene      DHP 12  
# 2  Orme      DHP 20  
# 3  Frene      DHP 13  
# 4  Chene      Haut 56  
# 5  Orme      Haut 85  
# 6  Frene      Haut 55
```

`ggplot2` peut utiliser le format large pour des visualisations simples, mais des visualisations plus complexes requièrent le format long.

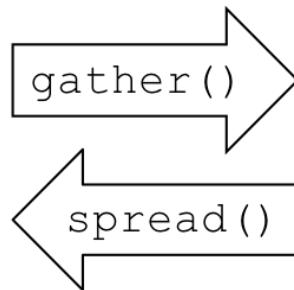
`dplyr`, `lm()`, `glm()`, `gam()` nécessitent le format long.

Manipuler vos données

Le paquet `tidyverse` permet de manipuler la structure d'un data frame en préservant les informations d'origine

`pivot_longer()` → "rassembler" les données (large -> long)

`pivot_wider()` → "dispercer" les données (long -> large)



Installation du paquet `tidyr`

```
install.packages("tidyr")  
library(tidyr)
```

Rassembler les colonnes en lignes

```
pivot_longer(data, cols, names_to, values_to, ...)
```

- `data` le jeu de données (e.g. 'large')
- `cols` les colonnes qu'on veut empiler dans le jeu de données (e.g. `DHP`, `Haut`)
- `names_to` le nom de la nouvelle colonne spécifiant la variable mesurée (e.g. `dimension`)
- `values_to` le nom de la nouvelle colonne spécifiant la mesure associée (e.g. `cm`)

Rassembler avec pivot_longer()

```
large <- data.frame(Species = c("Chene", "Orme", "Frene"),
                     DHP      = c(12, 20, 13),
                     Haut     = c(56, 85, 55))

large
#   Species DHP Haut
# 1 Chene    12   56
# 2 Orme     20   85
# 3 Frene    13   55

long <- pivot_longer(data      = large,
                      cols      = c("DHP", "Haut"),
                      names_to = "dimension",
                      values_to = "cm")

long
# # A tibble: 6 x 3
#   Species dimension   cm
#   <chr>    <chr>     <dbl>
# 1 Chene    DHP        12
# 2 Chene    Haut       56
# 3 Orme     DHP        20
# 4 Orme     Haut       85
# 5 Frene    DHP        13
# 6 Frene    Haut       55
```

Disperser avec les lignes en colonnes

```
pivot_wider(data, names_from, values_from, ...)
```

- `data` le jeu de données (e.g. `long`)
- `names_from` nom de la colonne contenant les noms des variables (e.g. `dimension`)
- `values_from` nom de la colonne contenant les mesures associées aux variables (e.g. `cm`)

Disperser avec pivot_wider()

```
long
# # A tibble: 6 x 3
#   Species dimension     cm
#   <chr>    <chr>      <dbl>
# 1 Chene    DHP         12
# 2 Chene    Haut        56
# 3 Orme    DHP         20
# 4 Orme    Haut        85
# 5 Frene   DHP         13
# 6 Frene   Haut        55

large2 <- pivot_wider(data           = long,
                      names_from = "dimension",
                      values_from = "cm")
large2
# # A tibble: 3 x 3
#   Species    DHP   Haut
#   <chr>    <dbl> <dbl>
# 1 Chene      12    56
# 2 Orme       20    85
# 3 Frene      13    55
```

La structure **tibble** pour des tables de données

Tibble est un format alternatif et plus pratique que celui d'un data frame. L'utilisation d'un tribble favorise de bonnes habitudes de programmation. Par exemple, il ne change pas une variable de chaîne de caractère en facteur.

```
tibble(x = 1:3, y = c("a", "b", "c"))
# # A tibble: 3 x 2
#       x     y
#   <int> <chr>
# 1     1     a
# 2     2     b
# 3     3     c
```

La structure `tibble` pour des tables de données

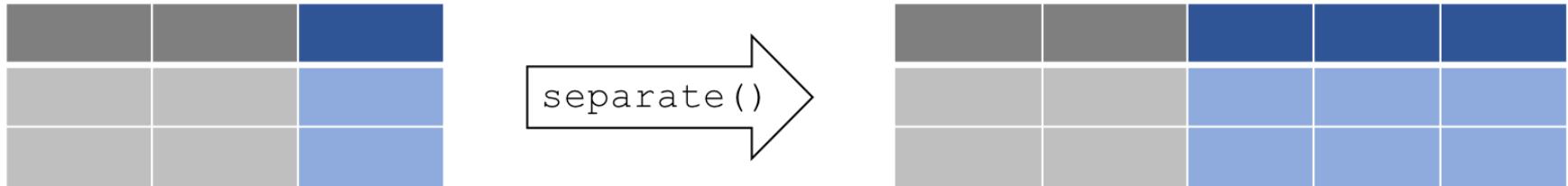
En plus, `tibble` simplifie l'utilisation de lists imbriqués dans la table de données.

Exemple:

```
tibble(x = 1:3, y = list(1:5, 1:10, 1:20))
# # A tibble: 3 x 2
#       x     y
#   <int> <list>
# 1     1 <int [5]>
# 2     2 <int [10]>
# 3     3 <int [20]>
```

Toutes les fonctions appliquées sur un data frame peuvent aussi être utilisées sur un tibble

separate() des colonnes



```
separate(data, col, into, sep)
```

- `data` → Un data frame (e.g. `long`)
- `col` → Nom de la colonne que nous voulons séparer
- `into` → Nom des nouvelles colonnes créées par la séparation
- `sep` → Caractère qui indique où séparer le contenu

Utiliser separate()

Créons un jeu de données fictif sur les poissons et le zooplancton :

```
set.seed(8)
degat <- data.frame(id = 1:4,
                      trt = sample(rep(c('control', 'culture'), each = 2)),
                      zooplankton.T1 = runif(4),
                      poisson.T1 = runif(4),
                      zooplankton.T2 = runif(4),
                      poisson.T2 = runif(4))

degat
#   id      trt zooplankton.T1 poisson.T1 zooplankton.T2 poisson.T2
# 1  1 culture     0.7189275  0.64449114    0.544962116  0.2644589
# 2  2 culture     0.2908734  0.45704489    0.138224346  0.2765322
# 3  3 control     0.9322698  0.08930101    0.927812252  0.5211070
# 4  4 control     0.7691470  0.43239137    0.001301721  0.2236889
```

Utiliser `separate()`

On peut commencer par convertir ce jeu de données vers le format long.

```
degat.long <- pivot_longer(degat,
                           names_to = "taxa",
                           cols      = c("zooplankton.T1",
                                         "poisson.T1",
                                         "zooplankton.T2",
                                         "poisson.T2"))

head(degat.long)
# # A tibble: 6 x 4
#       id trt     taxa     value
#   <int> <chr>   <chr>    <dbl>
# 1     1 culture zooplankton.T1 0.719
# 2     1 culture poisson.T1    0.644
# 3     1 culture zooplankton.T2 0.545
# 4     1 culture poisson.T2    0.264
# 5     2 culture zooplankton.T1 0.291
# 6     2 culture poisson.T1    0.457
```

Utiliser `separate()`

Ensute, on veut séparer les 2 temps d'échantillonnage (T1 et T2) dans la colonne "taxa" en utilisant le point comme caractère de séparation

```
debat.long.sep <- separate(debat.long, taxa,  
                           into = c("especies", "temps"), sep = "\\.")  
  
head(debat.long.sep)  
# # A tibble: 6 x 5  
#       id trt     especies     temps value  
#   <int> <chr>    <chr>      <chr> <dbl>  
# 1     1 culture zooplankton T1     0.719  
# 2     1 culture poisson     T1     0.644  
# 3     1 culture zooplankton T2     0.545  
# 4     1 culture poisson     T2     0.264  
# 5     2 culture zooplankton T1     0.291  
# 6     2 culture poisson     T1     0.457
```

L'argument `sep = " \\."` indique à R de scinder la chaîne de caractères autour du point (.). Nous ne pouvons pas écrire directement `sep = ". "` car il s'agit d'une expression régulière qui correspond à n'importe quel caractère (un joker).

Récapitulatif: `tidyr`

`tidyr` est un paquet qui réorganise la structure de jeux de données.

Convertir de format large en format long à l'aide de `pivot_longer()`

Convertir de format long en format large à l'aide de `pivot_wider()`

Séparer et regrouper des colonnes à l'aide de `separate()` et de son inverse `unite()`

Voici un aide-mémoire (en anglais) pour faciliter la manipulation de jeux de données avec `tidyr` et `dplyr`



Défi #1

Réorganisez le jeu de données `airquality` en format long (en rassemblant toutes les colonnes sauf "Month" et "Day").

```
?airquality
```

```
data(airquality)
```

Solution #1

Réorganisez le jeu de données `airquality` en format long (en rassemblant toutes les colonnes sauf "Month" et "Day").

```
air.long <- pivot_longer(airquality,
                         cols      = c("Ozone", "Solar.R", "Wind", "Temp"),
                         names_to = c("variable"))

head(air.long)
# # A tibble: 6 x 4
#   Month Day variable value
#   <int> <int> <chr>    <dbl>
# 1     5     1 Ozone     41
# 2     5     1 Solar.R  190
# 3     5     1 Wind      7.4
# 4     5     1 Temp      67
# 5     5     2 Ozone     36
# 6     5     2 Solar.R  118
```

Défi #2



Convertissez-le en format large pour retrouver le format original de [airquality](#)

Solution #2

Convertissez-le en format large pour retrouver le format original de `airquality`

```
air.wide <- pivot_wider(air.long,
                         values_from = "value",
                         names_from = "variable")
head(air.wide)
# # A tibble: 6 x 6
#   Month Day Ozone Solar.R Wind Temp
#   <int> <int> <dbl>    <dbl> <dbl> <dbl>
# 1     5     1     41      190    7.4    67
# 2     5     2     36      118     8     72
# 3     5     3     12      149   12.6    74
# 4     5     4     18      313   11.5    62
# 5     5     5     NA       NA   14.3    56
# 6     5     6     28       NA   14.9    66
```

Manipulation avec `dplyr`



Introduction à `dplyr`

La mission de `dplyr` est de simplifier nos tâches de manipulation.

- Package contenant un ensemble de fonctions (ou *verbes*) pour la manipulation de données, telles que le filtrage des lignes, la sélection de colonnes spécifiques, le réorganisation des lignes, l'ajout de nouvelles colonnes et la synthèse des données;
- Fonctions simples et intuitives;
- Rapide et efficace avec les grands jeux de données;
- Peut s'interfacer avec des bases de données externes et traduire votre code R en requêtes SQL

Certaines fonctions de base dans R ressemblent à des fonctions dans `dplyr`:
`split()`, `subset()`, `apply()`, `sapply()`, `lapply()`, `tapply()` et `aggregate()`

Introduction à `dplyr`

Commençons par installer et charger le paquet `dplyr`

```
library(dplyr)
```

Fonctions de bases de `dplyr`

Voici les 4 *verbes* principaux qui permettent d'exécuter les manipulations les plus communes:

- `select()`: sélectionne des colonnes dans un jeu de données
- `filter()`: filtre des rangées suivant les critères spécifiés
- `arrange()`: trie les données d'une colonne en ordre croissant ou décroissant
- `mutate()`: crée une nouvelle colonne de données (ou transforme une colonne existante)

select() - Sélection de colonnes



`select(data, ...)`

- `data` : le jeu de données
- `...` : noms ou positions de colonnes, ou expressions complexes (séparés par des virgules pour désigner les colonnes que l'on veut sélectionner)

Exemples :

```
select(données, colonne1, colonne2) # sélectionne colonne1 et colonne2
select(données, c(2:4,6) # sélectionne les colonnes 2 à 4, plus la 6ème colonne
select(données, -colonne1) # sélectionne toutes les colonnes sauf la 1ère
select(données, start_with(x.)) # sélectionne les colonnes ayant un nom qui commence
```

select() - Sélection de colonnes

Helper functions for select - ?select

select(iris, contains("."))

Select columns whose name contains a character string.

select(iris, ends_with("Length"))

Select columns whose name ends with a character string.

select(iris, everything())

Select every column.

select(iris, matches(".t."))

Select columns whose name matches a regular expression.

select(iris, num_range("x", 1:5))

Select columns named x1, x2, x3, x4, x5.

select(iris, one_of(c("Species", "Genus")))

Select columns whose names are in a group of names.

select(iris, starts_with("Sepal"))

Select columns whose name starts with a character string.

select(iris, Sepal.Length:Petal.Width)

Select all columns between Sepal.Length and Petal.Width (inclusive).

select(iris, -Species)

Select all columns except Species.

select() - Sélection de colonnes

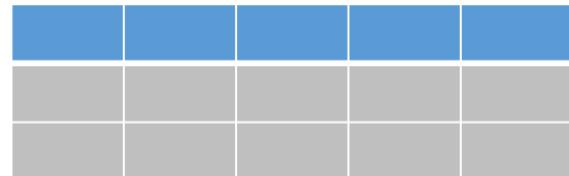
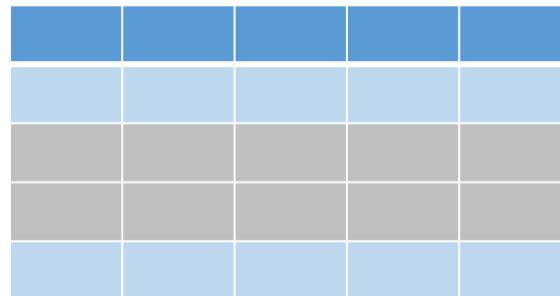
Par exemple, si on analyse la variation de la variable Ozone avec le temps.

```
ozone <- select(airquality, Ozone, Month, Day)
head(ozone)
#   Ozone Month Day
# 1    41      5    1
# 2    36      5    2
# 3    12      5    3
# 4    18      5    4
# 5    NA      5    5
# 6    28      5    6
```

filter() - sélection de lignes

Pour extraire un sous-ensemble de rangées selon une condition, on peut utiliser la fonction `filter()` avec la syntaxe suivante:

```
filter(dataframe, proposition logique 1, proposition logique 2, ...)
```



Logic in R - ?Comparison, ?base::Logic			
<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, , !, xor, any, all	Boolean operators

filter() - sélection de lignes

Par exemple, si on s'intéresse aux périodes de canicules du mois d'août dans le jeu de données `airquality`

```
aout <- filter(airquality, Month == 8, Temp >= 90)
# ou filter(airquality, Month == 8 & Temp >= 90)
head(aout)
#   Ozone Solar.R Wind Temp Month Day
# 1    89     229 10.3   90      8    8
# 2   110     207  8.0   90      8    9
# 3     NA     222  8.6   92      8   10
# 4    76     203  9.7   97      8   28
# 5   118     225  2.3   94      8   29
# 6    84     237  6.3   96      8   30
```

Ordonner des lignes avec `arrange()`

Réordonne les lignes selon une ou plusieurs colonnes, par défaut en ordre croissant

```
arrange(données, variable1, variable2, ...)
```

On peut également réordonner les lignes en ordre décroissant en utilisant la fonction `desc()` à l'intérieur de la fonction `arrange()`

```
arrange(data, variable1, desc(variable2), ...)
```

Ordonner des lignes avec `arrange()`

Exemple :

1) Commençons par créer une version désordonnée de `airquality`:

```
air_degat <- sample_frac(airquality, 1)
head(air_degat)
#   Ozone Solar.R Wind Temp Month Day
# 1    23     115  7.4   76     8   18
# 2    28     273 11.5   82     8   13
# 3     8      19 20.1   61     5    9
# 4   135     269  4.1   84     7    1
# 5    23     299  8.6   65     5    7
# 6    30     322 11.5   68     5   19
```

Ordonner des lignes avec `arrange()`

Exemple :

2) Maintenant, on réarrange le data frame en ordre chronologique, soit en ordre croissant selon `Month` et ensuite selon `Day`

```
air_chron <- arrange(air_degat, Month, Day)
```

```
head(air_chron)
```

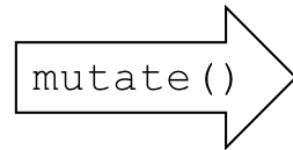
```
#   Ozone Solar.R Wind Temp Month Day
# 1    41     190  7.4   67      5    1
# 2    36     118  8.0   72      5    2
# 3    12     149 12.6   74      5    3
# 4    18     313 11.5   62      5    4
# 5    NA      NA 14.3   56      5    5
# 6    28      NA 14.9   66      5    6
```

Essayez `arrange(air_mess, Day, Month)` et voyez la différence.

`mutate()` - Créer de nouvelles colonnes

On peut utiliser la fonction `mutate()` pour créer et transformer des variables.

```
mutate(data, newVar1 = expression1, newVar2 = expression2, ...)
```

mutate() - Créer de nouvelles colonnes

Par exemple, on veut transformer la variable température `Temp` de degrés Fahrenheit vers degrés Celsius

```
airquality_C <- mutate(airquality,  
                      Temp_C = (Temp-32)*(5/9))
```

```
head(airquality_C)
```

```
#   Ozone Solar.R Wind Temp Month Day    Temp_C  
# 1   41     190 7.4   67      5   1 19.44444  
# 2   36     118 8.0   72      5   2 22.22222  
# 3   12     149 12.6   74      5   3 23.33333  
# 4   18     313 11.5   62      5   4 16.66667  
# 5   NA      NA 14.3   56      5   5 13.33333  
# 6   28      NA 14.9   66      5   6 18.88889
```

magrittr

Habituellement, la manipulation de données nécessite plusieurs étapes, le package `magrittr` propose l'opérateur `%>%` (*pipe operator*) qui nous permet de lier plusieurs opérations.



magrittr

Commençons par installer et charger le paquet :

```
library(magrittr)
```

magrittr

Supposons qu'on veut créer un sous-ensemble de airquality pour le mois de juin, et ensuite convertir la variable de la température en degrés Celsius. On peut créer ce data frame en combinant 2 *verbes* de dplyr

```
juin_C <- mutate(filter(airquality, Month == 6),  
                 Temp_C = (Temp - 32)*(5/9))
```

Plus on ajoute des opérations, plus ce code deviendra illisible. Mais, le faire étape par étape serait redondant et écrirait de nombreux objets dans l'espace de travail.

magrittr

Au lieu d'envelopper toutes les fonctions, on peut écrire les opérations en ordre d'exécutions et les relier à l'aide du *pipe* `%>%` :

```
juin_C <- airquality %>%
  filter(Month == 6) %>%
  mutate(Temp_C = (Temp-32)*(5/9))
```

Avantages :

- le code est moins redondant
- le code est facile à lire et à écrire parce que les fonctions sont exécutées dans l'ordre

dplyr::group_by et dplyr::summarise

Les verbes `dplyr` que nous avons appris dans cet atelier deviennent particulièrement puissants quand ils sont reliés par le "pipe" (%>%).

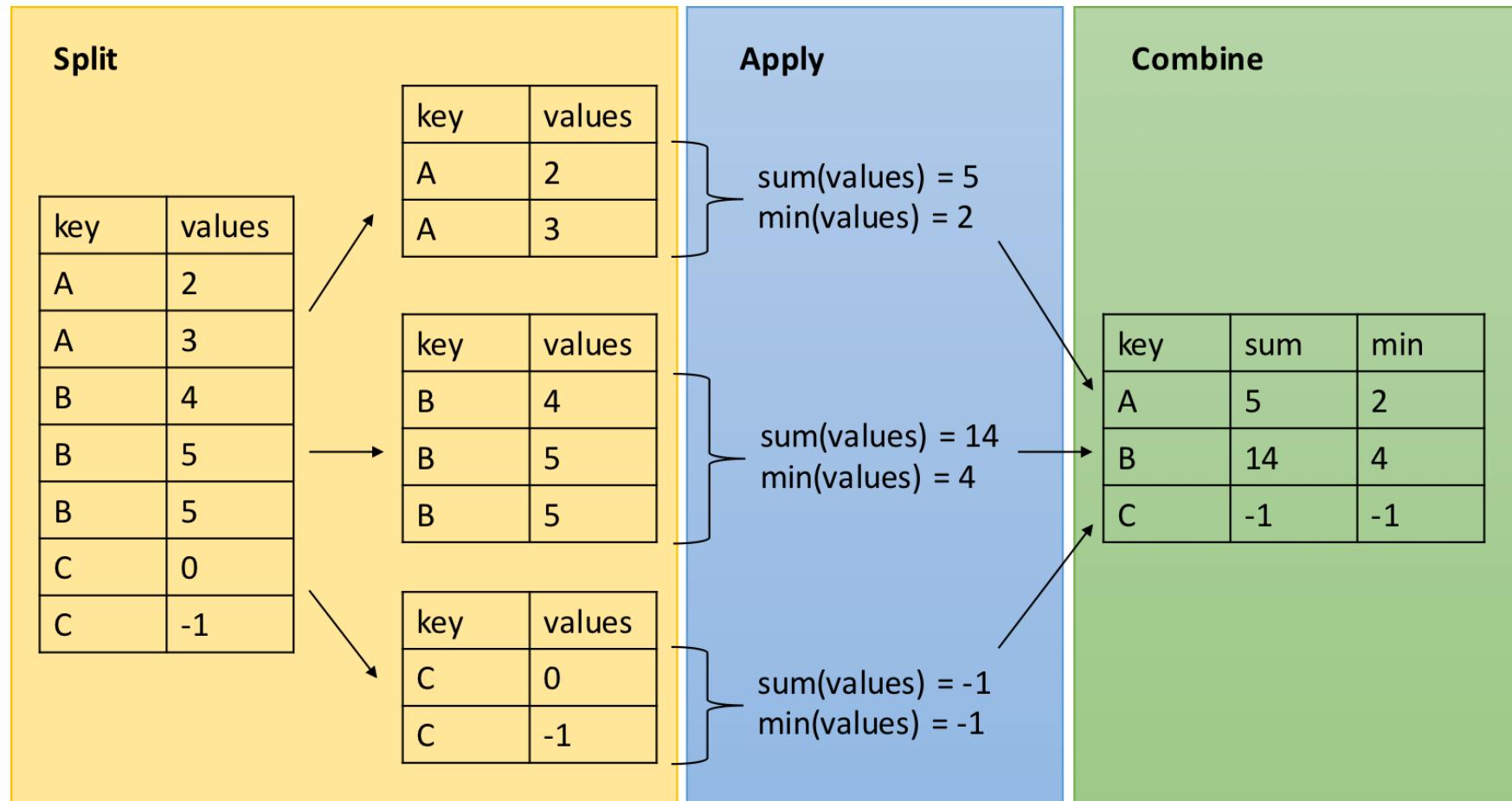
Les fonctions `dplyr` suivantes nous permettent de séparer nos jeu de données en groupes distincts sur lesquels on peut exécuter des opérations individuelles, comme des fonctions d'agrégation et de sommaire:

`group_by()` → regrouper le jeu de données par un facteur pour les opérations en aval (comme `summarise()`)

`summarise()` → créer un sommaire de variables au sein de groupes distincts dans un jeu de données en utilisant des fonctions d'aggrégation (e.g. `min()`, `max()`, `mean()`)

dplyr - Séparer-Appliquer-Combiner

La fonction `group_by` est la base de la stratégie Séparer-Appliquer-Combiner



dplyr - Séparer-Appliquer-Combiner

dplyr - Séparer-Appliquer-Combiner

Utilisons ces deux fonctions pour générer un sommaire du jeu de données `airquality` qui montre la température moyenne et l'écart type pour chaque mois:

```
mois_moy <- airquality %>%
  group_by(Month) %>%
  summarise(mean_temp = mean(Temp),
            sd_temp    = sd(Temp))

mois_moy
# # A tibble: 5 x 3
#   Month mean_temp sd_temp
#   <int>     <dbl>    <dbl>
# 1     5      65.5    6.85
# 2     6      79.1    6.60
# 3     7      83.9    4.32
# 4     8      84.0    6.59
# 5     9      76.9    8.36
```

Défi - dplyr et magrittr



En utilisant le jeu de données **ChickWeight**, créez un tableau sommaire dans lequel on retrouve la différence de masse entre le maximum et le minimum de la masse enregistré pour chaque poussin dans l'étude.

Utilisez les verbes dplyr et le "pipe" **%>%**.

```
## ?ChickWeight  
data(ChickWeight)
```

Solution

1. Utilisez `group_by()` pour grouper le jeu de données `ChickWeight`
2. Utilisez `summarise()` pour calculer la gain de poids par groupe

```
mass_diff <- ChickWeight %>%
  group_by(Chick) %>%
  summarise(mass_diff = max(weight) - min(weight))
head(mass_diff)
# # A tibble: 6 x 2
#   Chick mass_diff
#   <ord>     <dbl>
# 1 18          4
# 2 16         16
# 3 15         27
# 4 13         55
# 5 9          58
# 6 20         76
```

Défi R base



Maintenant essayez de refaire le même exercice mais **seulement avec les fonctions de bases de R !**

Prenez note qu'il existe plusieurs solutions.

Indice: Les fonction `?aggregate()` ou `?by()` pourraient s'avérer utiles.

Défi R base - Solution

```
mass_diff_rbase <- aggregate(formula = weight ~ Chick,
                               data     = ChickWeight,
                               FUN      = function(x) weight_diff = max(x) - min(x))
names(mass_diff_rbase) <- c("Chick", "weight_diff")

# Est ce que les deux résultats sont identiques (c-a-d avec et sans dplyr)

table(mass_diff_rbase == as.data.frame(mass_diff))
#
# TRUE
# 100
```

Ressources supplémentaires

Pour en savoir plus sur dplyr

dplyr et tidyr anti-sèche

Merci pour votre participation! :)

