



Formation-R-initiation

Formation R initiation

Sommaire

- 1 Présentation**
- 2 Les objets de R**
- 3 Manipulation de données**
- 4 Outils DPLYR & TIDYR**



0.1 Avant-propos

Ce diaporama de formation a été rédigé dans le but d'être le support visuel des formations dispensées au [MASA](#).
Ces formations s'adressent à des agents qui ont suivi **les fondamentaux de la programmation**.



0.2 Avant-propos

Elles sont données en présentiel sur une durée de 2 jours.

Champ couvert par cette formation

Ce support couvre une initiation au langage R avec R-studio et une découverte de l'environnement de production du MASA.

Pour information, les thèmes abordés sont:

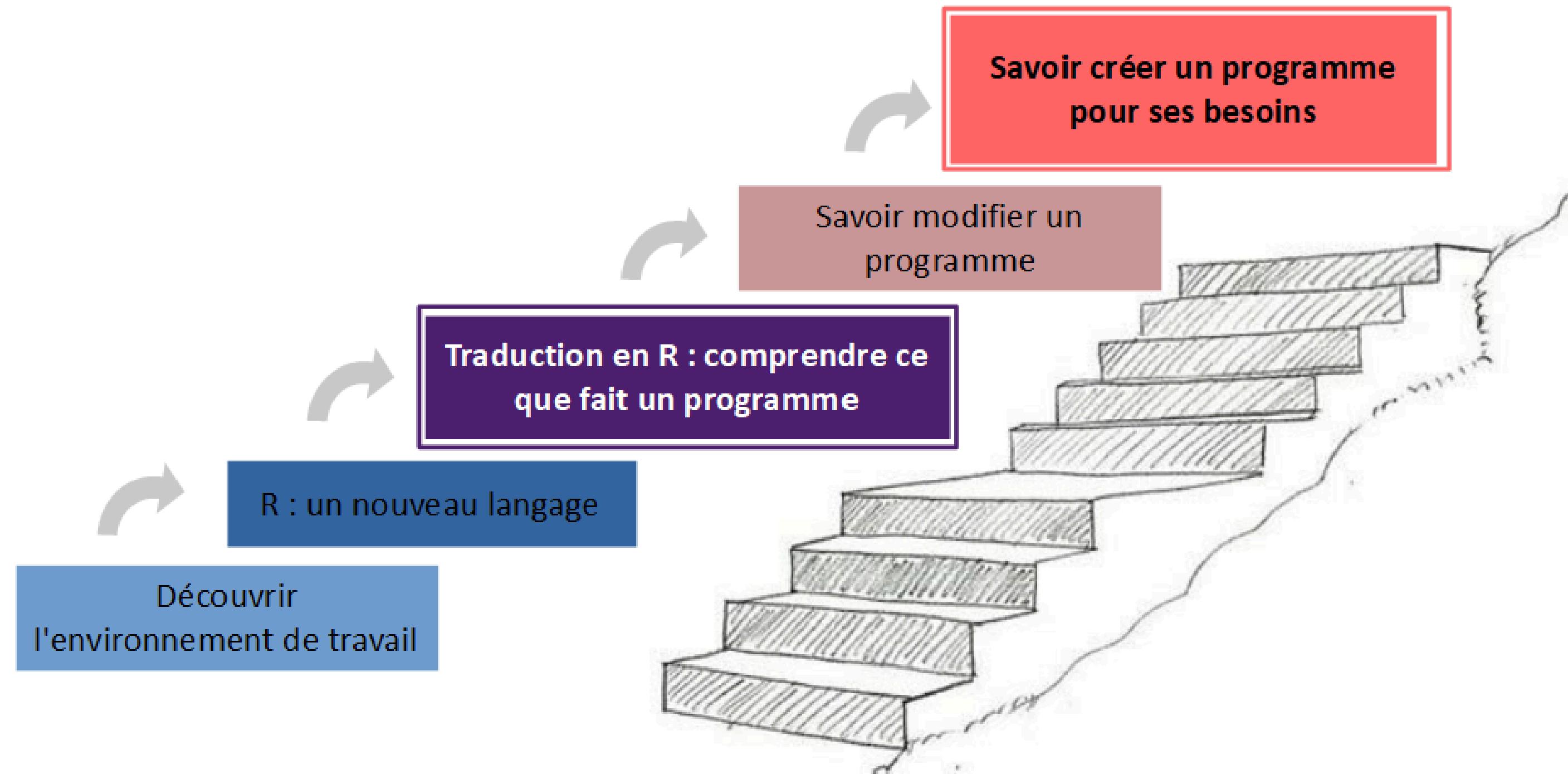
- 01 - Présentation
- 02 - Les objets de R
- 03 - Manipulation de données
- 04 - Outils DPLYR & TIDYR

Ils sont orientés pour être utile aux agents du SSM Agriculture et se concentrent sur une utilisation de R via [RStudio](#) qui est mise à disposition des agents sur la plateforme interne Cerise basée sur RStudio Workbench.



1 Présentation

1.1 Objectifs de la formation



1.2 RStudio

Logiciel de **traitement de données et analyse statistique** :

- offre un environnement interactif de développement statistique, analytique et graphique ; est doté d'un langage de programmation R ;
- permet d'accéder à des données, de les manipuler et les analyser ;
- S'interface avec les bases de données : Oracle, SYBASE, PostgreSQL, SQLITE...

Remplace **SAS ou SPSS**

Logiciel IDE : *Integrated Development Environment*



1.3 Pourquoi utiliser R

Avantages	Inconvénients
- logiciel gratuit et open source	- Langage informatique nouveau à apprêhender
- développement actif : communauté d'utilisateurs de plus en plus importante	- pas de génération de code par « clic-bouton »
- nombreux packages (« outils ») scientifiques	- compatibilité des versions
- très bonnes capacités graphiques	- calculs sur de grosses volumétries plus « gourmand » en CPUs et Mémoire
- rapidité des traitements (R travaille en mémoire)	- documentation et aide souvent disponibles qu'en anglais

1.4 Comment travailler avec R au MASA

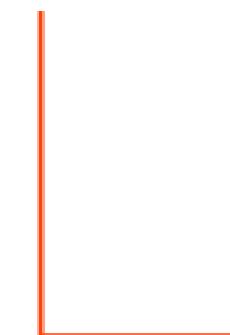
- Depuis la plate-forme CERISE :
 - ⇒ Adresse : <https://rstudio.agriculture.rie.gouv.fr>
 - ⇒ PISTACHE : Pistache > Traitements statistiques et Diffusion > R > Migration SPSS et SAS vers R > Accès au WIKI Cerise - R > Accéder à Cerise

Cerise permet l'accès aux ressources déposées sur le serveur, la sauvegarde, le partage de code et le travail simultané. C'est l'usage recommandé.

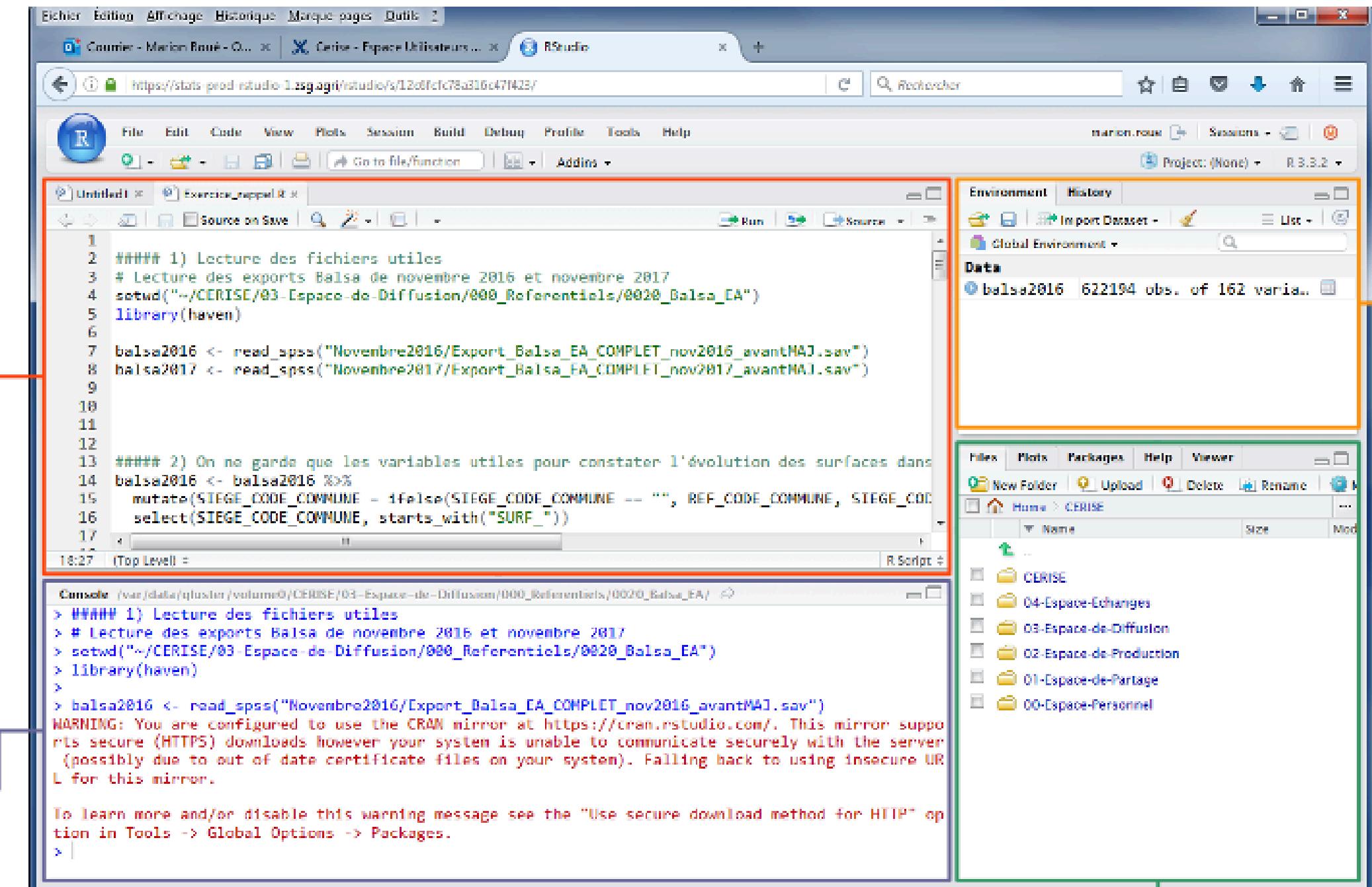
- En local sur son poste, sans intervention de Pastel
 - Non préconisée
- Existence d'une plate-forme CERISE de Préproduction :
 - <https://rstudio-pprd.agriculture.rie.gouv.fr>

1.5 Fenêtres dans RStudio

Vue des scripts et
des données



Console



Vue des éléments
en mémoire et de
l'historique des
commandes

Vue des
répertoires, des
graphiques, des
packages et de
l'aide

1.6 La Console

⇒ EN BAS A GAUCHE

- permet de saisir des lignes de commande
- affiche la log et les résultats

The screenshot shows the RStudio interface with the 'Console' tab selected. The console window displays the following R session:

```
C:/java/RStudio/
>
>
> 2+6
[1] 8
> mean(1:10)
[1] 5.5
> |
```

Two arrows point from the text 'Programme saisi' to the command '2+6' in the console. Another arrow points from the text 'Log' to the command 'mean(1:10)'.

Log : historique des commandes, messages, etc.



1.7 La Console



- Si la première ligne de la console commence par le caractère **>**

→ cela signifie que R est disponible et en attente de la prochaine commande

- Si la première ligne commence par le caractère **+**

→ cela signifie que R considère que la commande de la ligne précédente n'est pas terminée et qu'il « attend la suite »

Il faut alors :

- compléter la commande

ou

- sortir et recommencer en appuyant sur **Echap**

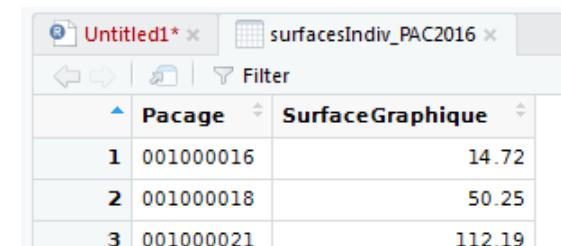
1.8 La Console

- Possibilité de naviguer dans la console avec les flèches ↑ ou ↓
- Appuyer sur Entrée pour exécuter une commande, ou cliquer sur 
- Pour effacer le contenu de la console : menu **Edit > Clear console** ou **Ctrl+L**

1.9 Visualisation

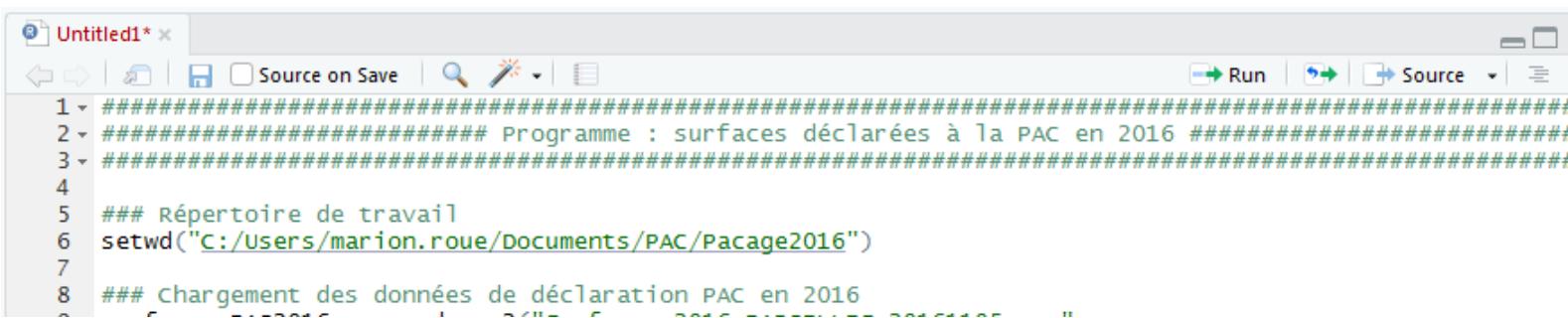
⇒ EN HAUT A GAUCHE

- les données :



	Pacage	SurfaceGraphique
1	001000016	14.72
2	001000018	50.25
3	001000021	112.19

- les scripts (programmes) :



```
1 ##### Programme : surfaces déclarées à la PAC en 2016 #####
2 #### Répertoire de travail
3 setwd("C:/users/marion.roue/Documents/PAC/Pacage2016")
4
5 #### Chargement des données de déclaration PAC en 2016
6 f = read.csv("C:/users/marion.roue/Documents/PAC/Pacage2016/surfacesIndiv_PAC2016.csv")
```

→ Pour soumettre la ligne ou la sélection depuis le script, il faut cliquer sur Run ou Ctrl + Entrée



1.10 Entrer une ligne de commande

Deux possibilités pour entrer une ligne de commande :

- depuis le **Script** ⇒ plutôt pour les commandes à conserver
- directement dans la **Console** ⇒ plutôt pour les commandes de vérifications

Je besoin de ...

Trouver pour vérification le minimum d'une variable

J'écris ...

Console

Faire un tableau qui sera à refaire chaque mois

Script

Créer un référentiel qu'il faudra compléter/modifier

Script

Afficher une ligne d'un tableau pour vérifier l'exactitude de données

Console

Afficher une ligne d'un tableau pour écrire un primeur

Script



1.11 Environnement et historique

→ EN HAUT A DROITE

Historique des commandes

The screenshot shows the RStudio interface with the 'Environment' tab selected. The 'Data' section lists three objects: 'surfaces...', 'surfacesA...', and 'surfacesI...'. The 'History' tab is also visible at the top.

visualiser les éléments en mémoire

Différents icônes :

- charger une table de données R
- importer des données externes
- Sauvegarder des données R
- Vider les éléments en mémoire

Liste des options :

- List
- List
- Grid



1.12 Fichiers, graphiques, aides, packages et Viewer

→ EN BAS A DROITE

Plusieurs onglets :

Files (Fichiers)

Accès à « Home » qui correspond à son espace personnel

Arborescence des fichiers, comme dans l'explorateur de fichier de votre PC

Plot

Visualisation des graphiques

Packages

Gestion des paquets

Help

Accéder à l'aide

Connaître les paramètres fonctions

Viewer

Navigateur internet

À utiliser très souvent !





1.13 Organisation sous Cerise

Cerise est organisé en plusieurs répertoires :

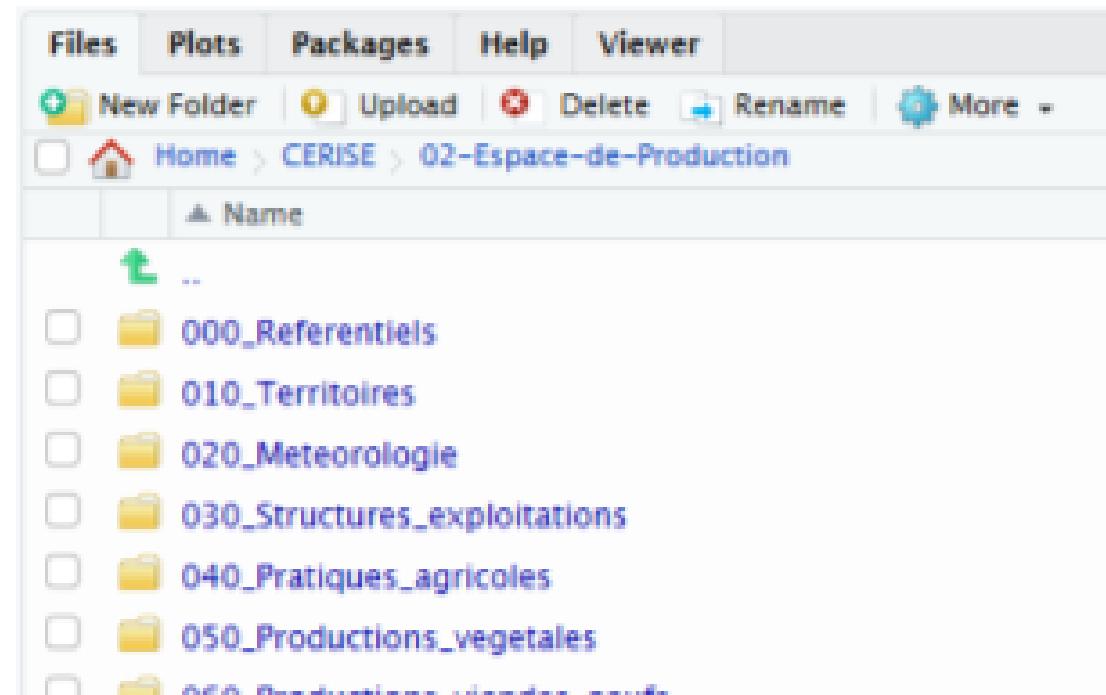
- **00-Espace-Personnel** => espaces personnels des agents, accessible par l'agent uniquement
- **01-Espace-de-Partage** => lieu de partage général entre les différents acteurs

The screenshot shows a software interface titled 'CERISE'. At the top, there is a menu bar with tabs: 'Files', 'Plots', 'Packages', 'Help', and 'Viewer'. Below the menu bar, there is a toolbar with icons for 'New Folder', 'Upload', 'Delete', 'Rename', and 'More'. A breadcrumb navigation bar shows the path: 'Home > CERISE > 01-Espace-de-Partage'. The main area is a file list table with columns for selection, icon, name, and date modified. The table contains the following entries:

	Name	Date Modified
<input type="checkbox"/>	..	2023-09-11 14:45:00
<input type="checkbox"/>	Formations	2023-09-11 14:45:00
<input type="checkbox"/>	MOE	2023-09-11 14:45:00
<input type="checkbox"/>	Outils_Communs	2023-09-11 14:45:00
<input type="checkbox"/>	Partage-de-code	2023-09-11 14:45:00
<input type="checkbox"/>	SRISE	2023-09-11 14:45:00
<input type="checkbox"/>	SSP	2023-09-11 14:45:00

1.14 Organisation sous Cerise

- **02-Espace-de-Production** => plateforme de stockage des données brutes collectées, ainsi que des fichiers de données et programmes issus des traitements statistiques réalisés par l'équipe projet



- **03-Espace-de-Diffusion** => mise à disposition au sein du SSM des données issues des traitements statistiques réalisés en amont

=> Ces deux derniers espaces sont découplés par opérations statistiques



1.15 Organisation sous Cerise

- **04-Espace-Echanges** => stockage des fichiers de données à transmettre aux autres applications du SI CASSIS (par exemple Agreste) ainsi qu'aux SI des partenaires extérieurs

The screenshot shows a file management interface for the Cerise system. The top navigation bar includes 'Files', 'Plots', 'Packages', 'Help', and 'Viewer'. Below the bar are buttons for 'New Folder', 'Upload', 'Delete', 'Rename', and 'More'. The breadcrumb navigation shows the path: Home > CERISE > 04-Espace-Echanges. The main area displays a list of files and folders:

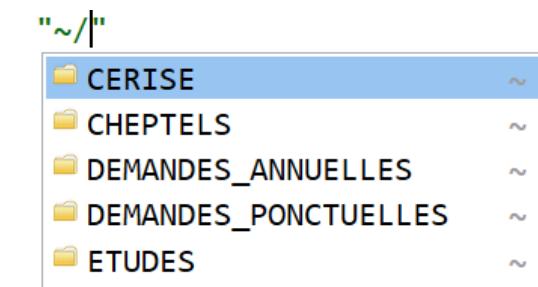
- ..
- 2022_05_20_questionnaires_pre-initialises_ajoutes.csv
- 2022_05_20_questionnaires_pre-initialises_ajoutes_code_Acces2022-
- base_2018_exploitants.dbf
- base_2018_parcelle.dbf
- depot
- données_enquêtes
- donnees_hyperion
- DPRS
- EML
- eric
- FICHES TERRITORIALES SRISE

1.16 Cheminer sous Cerise

Sous Cerise, le chemin “~/” fait référence à son espace personnel

On peut reconstituer facilement un chemin complexe de l’arborescence Cerise en utilisant **l’auto-complétion** de R :

- Je commence par taper le chemin pointant vers l’espace personnel “~/”
- Je place le curseur après le slash, et j’appuie sur la touche tabulation du clavier : l’arborescence est proposée



- Avec les flèches haut et bas, je sélectionne l’élément qui m’intéresse. Si je veux descendre plus bas dans l’arborescence, j’appuie sur tabulation pour que l’arborescence du dossier choisi s’affiche. Je continue jusqu’à avoir construit le chemin complet :

“~/CERISE/03-Espace-de-Diffusion/030_Structures_exploitations/”

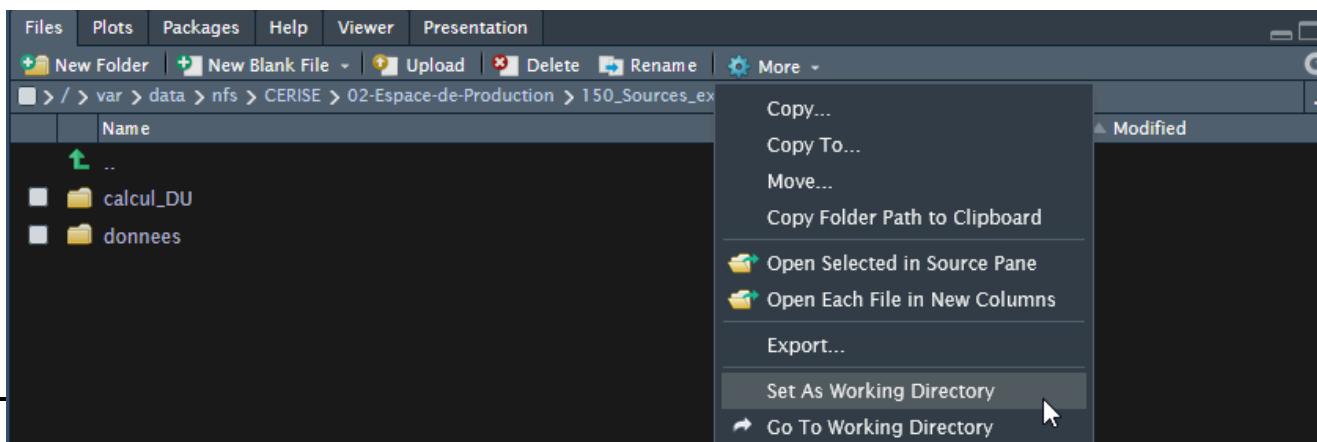
3010_Enquetes_Structures	.../03-Espace-de-Diffusion/030_Structures_exploitations
3020_Recensements	.../03-Espace-de-Diffusion/030_Structures_exploitations

1.17 Répertoire de travail

- R a par défaut un répertoire de travail. Cela signifie que si aucun chemin n'est spécifié, R va lire ou écrire les fichiers dans ce répertoire.

⇒ Sur le serveur, le répertoire de travail est l'espace personnel

- Pour connaître le répertoire de travail : commande `getwd()`
- Pour changer le répertoire de travail :
 - commande `setwd("chemin du rép souhaité")`
 - Ou dans l'onglet **Files**





1.18 Maafluence

Dans la page d'accueil Maafluence de Cerise, on peut trouver des liens vers divers outils utiles :

https://orion.agriculture/confluence/display/CER/CERISE_Espace+Utilisateurs

Aide et documentation

- Foire Aux Questions
- La documentation dont :
 - La plateforme Cerise
 - Les ressources de formation pour R
 - Le guide des bonnes mani'R
 - Les fiches méthodologiques R
- Le forum du langage R
- Le groupe utilisateur R et sa newsletter *Comment faiR*

- Une FAQ sur Cerise avec les questions récurrentes, les actualités
- Des documents d'aide : le guide des bonnes mani'R, les fiches méthodologiques, les documents de formation
- Le lien vers le forum R du ministère, toujours vif à répondre à toutes les questions
- Le lien vers les actualités et les présentations du **Groupe Utilisateurs R** :

<https://orion.agriculture/confluence/display/CER/Le+groupe+utilisateur+R>



1.19 Les scripts (programmes)

Quand on programme, il est préférable d'écrire les commandes dans un fichier texte plutôt que dans la console

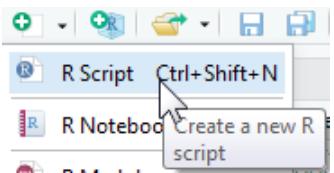
Avantages:

- **Sauvegarde des opérations effectuées**
 - → lorsqu'on redémarre R, tout ce qui a été effectué dans la console est perdu (contrairement aux scripts qui peuvent être enregistrés)
- **Obligation de commenter son code**
- **Code reproductible**
 - → en exécutant à nouveau les commandes d'un script, on peut reproduire les mêmes données et les mêmes résultats
- **Possibilité de partager son code**

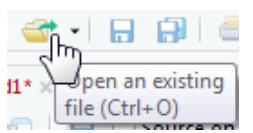
1.20 Les scripts (programmes)

Les scripts sont des fichiers texte - ils portent l'extension .R

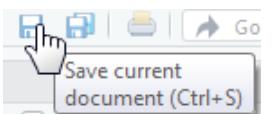
- → Créer un script : **File > New File > R Script** ou



- → Travailler avec un script existant : **File > Open File...** ou

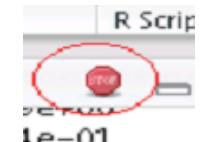


- → Sauvegarder un script : **File > Save** ou **File > Save as...** ou

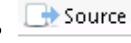


1.21 Les scripts (programmes)

- Pour exécuter des commandes d'un script, se positionner sur la ligne ou bien sélectionner l'ensemble des commandes à exécuter puis **Ctrl+Entrée** ou **Code > Run Selected line(s)** ou 
- Pour interrompre une exécution, on utilise le bouton « stop » de la console :



*Pour exécuter tout le code d'un script préprogrammé, on utilise la fonction
`source(file = "V:/FormationR/Prog/mon_fichier.R")`*

ou sous RStudio : 

(dans ce cas, rien ne s'affiche dans la console lors de l'exécution)

1.22 Les scripts (programmes)



R est sensible à la casse

⇒ Les caractères en minuscules ou en MAJUSCULES sont différents



Les commentaires s'écrivent après le symbole #



Dans les chemins Windows, les \ doivent être remplacés par des / ou des \\

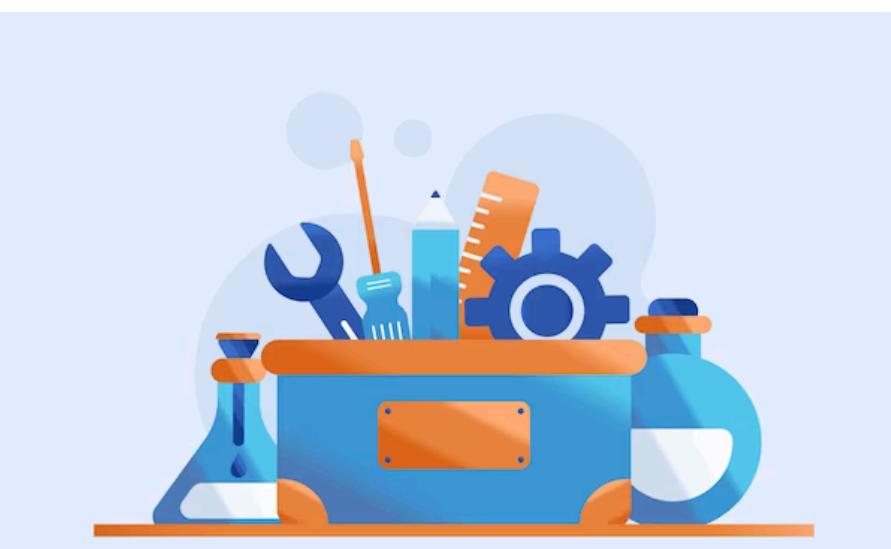
1.23 Les packages

Nombreuses possibilités avec installation de base de R ...

... Mais parfois besoin de nouvelles fonctionnalités

⇒ Utilisation de **packages** :

- **Boîte à outils** : fonctions spécifiques (et parfois des données) relatives à un domaine particulier
- Développés et maintenus par la communauté des utilisateurs de R, et diffusés via le CRAN (Comprehensive R Archive Network = réseau de serveurs)



1.24 Les packages

- Pour pouvoir utiliser un package, il doit être installé et chargé *en mémoire* :
 - un package n'a besoin d'être installé qu'une seule fois : « achat de la boîte à outils »
 - il faut le charger à chaque nouvelle session pour l'utiliser : « prendre la boîte du placard »
- **En local**

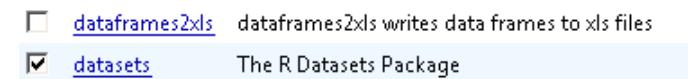
Un certain nombre de packages (base) sont fournis lors de l'installation de R et n'ont pas besoin d'être installés.

- **Sur Cerise**
 - Les « packages du socle commun » sont déjà installés → onglet Packages : [System Library](#)
 - Possibilité d'installer des packages par les utilisateurs → onglet Packages : [User Library](#)
- L'installation d'un package se fait de l'une des façons suivantes :
 - *En cliquant sur les menus en haut de la fenêtre* : Tools → Install packages
 - Via une ligne de syntaxe dans le code : `install.packages("nom_package")`
 - *Via l'onglet packages dans la fenêtre en bas à droite* : bouton **Install**

1.25 Les packages

Charger un package (à chaque utilisation)

- utiliser la ligne de commande **library(nom_package)** (*conseillé pour la reproductibilité du programme*)
- cocher sur RStudio (*déconseillé car si quelqu'un reprend le programme, il ne saura pas qu'il faut cocher*)



RStudio: l'onglet **Packages** de la fenêtre en bas à droite indique les packages installés.

Les packages cochés sont ceux chargés, et donc utilisables.

Lorsqu'on essaie d'utiliser une fonction d'un package sans l'avoir chargé, un message d'erreur s'affiche

```
> freq(tab_ind$nom)
Error in freq(tab_ind$nom) : could not find function "freq"
> |
```



1.26 Les Raccourcis clavier utiles

En plus des raccourcis claviers usuels (Enregistrer, copier-coller, annuler l'action précédente), un certain nombre de raccourcis clavier peuvent être très utiles au quotidien dans R :

Raccourci clavier	Action
Ctrl + shift + n	Ouvrir un nouveau script
Ctrl + shift + m	Écrire un pipe %>% ou avec la nouvelle écriture >
Ctrl + shift + c	Passer les lignes de code sélectionnées en commentaires
Ctrl + shift + r	Introduire un titre de section en commentaires
Ctrl + l	Vider la console
Ctrl + f	Ouvrir un module pour du chercher-replacer dans le script
Ctrl + i	Indenter automatiquement les lignes de code sélectionnées

1.27 EXERCICE 1: Découverte de RStudio

1. **Découverte du serveur** : accéder à son répertoire personnel et créer des dossiers

Dossier pour la formation initiation avec 3 sous-dossiers :

- un pour les données à importer : **1-Donnees**
- un pour enregistrer les programmes : **2-Programmes**
- un pour exporter les résultats : **3-Resultats**

2. Répertoire de travail

Dans quel répertoire êtes vous ? Le modifier pour que ce soit le dossier consacré à la formation



2 Les objets de R



2.1 Introduction

Quand on lance une commande (dans la console ou depuis un script), R fait le calcul (ou l'opération demandée) et affiche le résultat dans la console

```
> 3+2  
[1] 5  
>
```

```
> round(sqrt(2), 3)  
[1] 1.414  
>
```

- ⇒ Pour réutiliser, appeler ou modifier un résultat dans la suite du programme : **stocker le résultat dans un objet**
⇒ c'est l'affectation
- ⇒ **Tout ce qui n'est pas stocké dans un objet n'est pas gardé en mémoire !**

C'est comme si je voulais modifier un fichier Excel, il faut bien que je stocke mon travail !



2.2 Stockage d'objet

Affectation : affecter à un objet une valeur Opérateur : <-



Affecter une valeur à un objet le crée en même temps

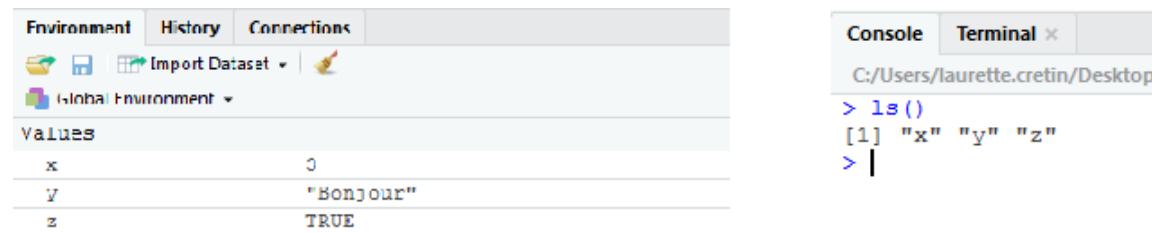
Pour afficher la valeur d'un objet, il suffit de saisir son nom

```
Console Terminal x
C:/Users/laurette.cretin/Desktop/F
> x
[1] 3
> y
[1] "Bonjour"
```

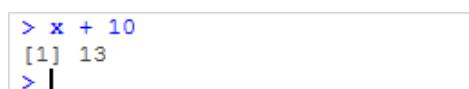


2.3 Stockage d'objet

- Possibilité de créer autant d'objets que l'on souhaite
 - Les objets créés apparaissent dans la fenêtre en haut à droite
 - La commande `ls()` liste tous les objets en mémoire



Si un objet est utilisé dans une opération, R le remplace par sa valeur





2.4 Stockage d'objet

- **Les objets n'ont pas de mémoire**

⇒ lorsqu'on assigne une nouvelle valeur à un objet déjà existant, la valeur précédente est perdue

```
Console Terminal ×
C:/Users/laurette.cretin/Desktop/Pratiques culturelles
> x <- 5
> x
[1] 5
> |
```

Comme dans un tableau : si dans une cellule je colle une valeur, la précédente est écrasée

- **Pas de lien entre les objets**

⇒ lorsqu'on assigne un objet à un autre, cela ne crée pas de lien entre eux : modifier le premier objet ne va pas modifier le second

```
Console Terminal ×
C:/Users/laurette.cretin/Desktop/Pratiques culturelles
> a <- 10
> b <- a
> b
[1] 10
> a <- 11
> b
[1] 10
> |
```

2.5 Gestion de la mémoire

- `rm(nom_objet)` supprime l'objet nom_objet de la mémoire.
- `rm(list=ls())` supprime tous les objets en mémoire.
- RStudio : L'icône  de l'onglet **Environment** supprime aussi les objets. En utilisant l'affichage Grid, on peut effectuer une sélection.
- R travaille en mémoire vive :
 - Environnement « trop plein » = impact sur les temps d'exécution et les performances de R
 - ⇒ Supprimer régulièrement de la mémoire les objets qui ne sont plus utiles
- Cerise : mémoire partagée entre tous les utilisateurs
 - ⇒ Gérer son environnement pour ne pas impacter négativement les autres utilisateurs



2.6 Règles de nommage des objets

Les noms des objets doivent commencer par une lettre

Il est possible :

- d'utiliser des majuscules et des minuscules
- d'utiliser des chiffres, l'underscore (_) et le point (.)

Il est déconseillé :

- d'utiliser les caractères accentués
- de choisir des noms de variables non explicites, trop longs ou trop courts
- d'écraser les noms de fonctions existantes

Il est impossible :

- d'utiliser un espace
- d'utiliser certains noms réservés au système (else, for, T, F, ...)
- d'écraser les noms de fonctions existantes

2.7 Les différents objets en R

En R, les données sont stockées dans différents objets :

- **Vecteur** : ensemble de valeurs de même nature
- **Data.frame** : concaténation de plusieurs vecteurs (=colonnes), de natures potentiellement différentes

⇒ *similaire à un tableau de données d'un tableur*

Avancé:

- **Matrice** : ensemble de valeurs de même nature (que des nombres, que des chaînes de caractères, que des booléens...)
- **Liste** : objet regroupant différents objets



2.8 Le vecteur

- Exemple de vecteur : 1 objet (le vecteur) avec différents éléments

```
[1] "Janvier"   "Février"    "Mars"        "Avril"       "Mai"        "Juin"        "Juillet"     "Aout"        "Septembre"  "Octobre"  
[11] "Novembre"  "Decembre"
```

- Créer un vecteur : `c()`

c() est une fonction qui concatène plusieurs informations de même nature

```
Source on Save | 🔎 ⚡ | 📁
1 poli <- c("Bonjour", "Merci", "Au revoir", "Hello", "Bye")
2 serie<- c(3, -5, 20/3, pi, 3.25, 4*8)
```

poli est un vecteur de type caractère et de longueur 5
serie est un vecteur de type numérique de longueur 6

Attention : Si des éléments sont de modes différents, R les convertit automatiquement au même mode :

```
> nature <- c("Hello","Toto",1)
> nature
[1] "Hello" "Toto" "1"
```

Rappel: `c()` permet de créer un objet, pour l'utiliser **il faut le stocker et donc l'affecter !**

2.9 Le vecteur

- Un vecteur peut être de quatre types :
 - numérique
 - caractère
 - logique (vrai ou faux)
 - facteur : vecteur dont les seules valeurs possibles sont les modalités d'une variable quantitative (ex : régions, départements...)
- Valeur particulière : **NA** lorsqu'une variable est en valeur manquante
- On détermine le type d'un vecteur avec **mode()**, on peut tester les types avec les fonctions **is.numeric()**, **is.character()**, **is.logical()**, **is.na()** etc :

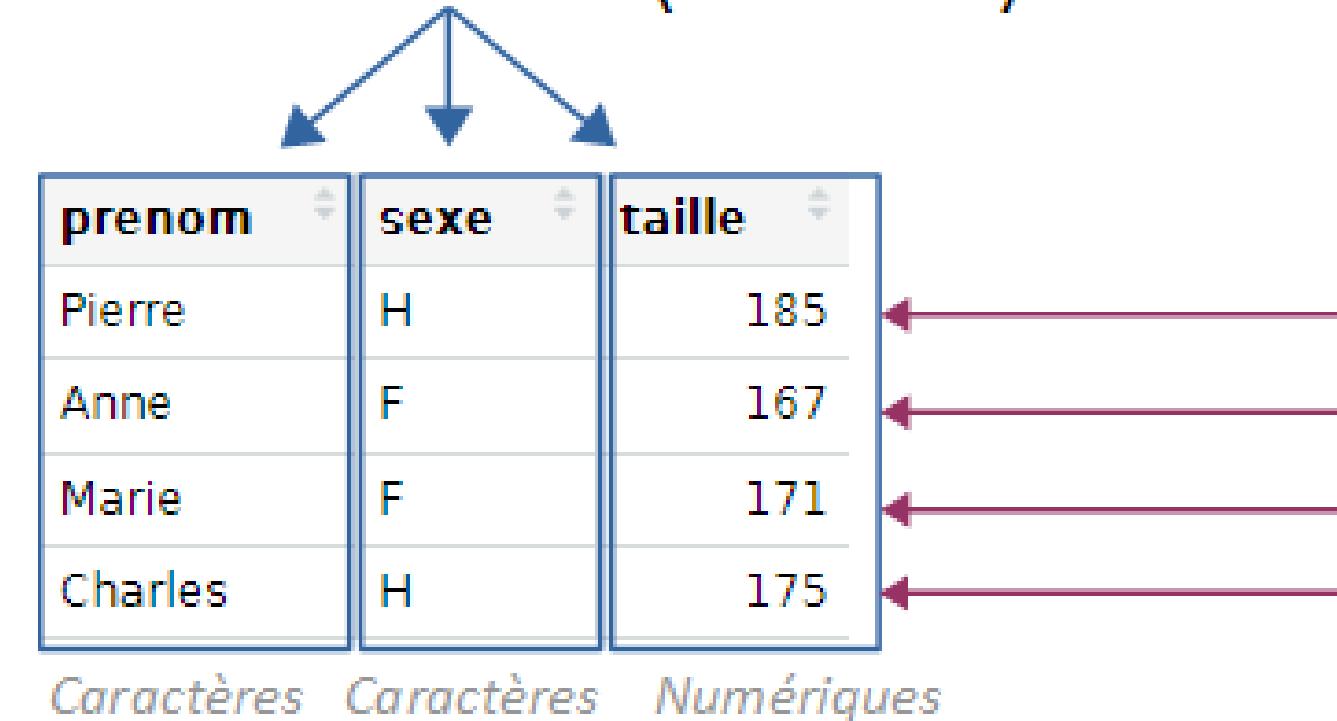
```
> x <- c(1,2,3,4,5)
> mode(x)
[1] "numeric"
> is.numeric(x)
[1] TRUE
```

2.10 Les data.frame

data.frame: plusieurs vecteurs (colonnes) de même longueur et de mode potentiellement différents

- ⇒ C'est ce que l'on utilise le plus souvent
- ⇒ C'est un **tableau de données**, avec les observations en lignes et les variables en colonnes

Exemple : **data.frame tab_ind avec 3 vecteurs (variables) de 4 éléments chacun (observations)**



Caractères	Caractères	Numériques
prenom	sexe	taille
Pierre	H	185
Anne	F	167
Marie	F	171
Charles	H	175

2.11 Les data.frame

- Création de 3 vecteurs de longueur 4 : nom, sexe et taille

```
nom <- c("Pierre", "Anne", "Marie", "Charles")
sexe <- c("H", "F", "F", "H")
taille <- c(185, 167, 171, 175)|
```

- La fonction **data.frame()** permet la création d'un data frame à partir de vecteurs

```
tab_ind <- data.frame(nom, sexe, taille, stringsAsFactors = FALSE)
```

Ce dernier paramètre permet de ne pas considérer les variables caractères en facteurs



- Visualisation du data frame : tab_ind ou **View(tab_ind)**

Dans la console, en tapant juste le nom de l'objet :

```
> tab_ind
   nom sexe taille
1 Pierre H 185
2 Anne F 167
3 Marie F 171
4 Charles H 175
```

Dans le **visualiseur**, en cliquant sur l'objet :

	nom	sexe	taille
1	Pierre	H	185
2	Anne	F	167
3	Marie	F	171
4	Charles	H	175



2.12 Les data.frame : Informations

Pour connaître les caractéristiques d'un data.frame, il y a plusieurs outils utiles :

- **str()** donne la structure de la table (liste des colonnes, avec le type et les premières valeurs)
- **names()** renvoie le vecteur des noms de colonnes
- **head()** et **tail()** donnent respectivement les 6 premières et les 6 dernières lignes de la table (6 par défaut, peut être modifié)
- **nrow()** et **ncol()** donnent respectivement le nombre de lignes et de colonnes de la table

Pour accéder aux différentes variables d'un data.frame, on utilise l'opérateur **\$**

```
> tab_ind$nom
[1] "Pierre"  "Anne"    "Marie"   "Charles"
```

2.13 Exercice1 (suite) : découverte De rStudio

3. Programmes et manipulation des données

Mise en place :

- Créer un nouveau programme et l'enregistrer
- Charger les librairies : dplyr, tidyr, readr

Travail sur un data.frame : iris est un jeu de données fourni avec R

- Combien de colonnes ? De lignes ?
- Quel est le nom des variables ?
- Afficher les 10 premières lignes
- Quel est le type de chacune des variables ?

Il y a plusieurs bonnes manière de faire : il faut expérimenter !

2.14 Les fonctions

Sous R, les “objets” sont des données (vecteurs, data.frames...) ou des fonctions

⇒ Les fonctions permettent de transformer les données.

- Comme sous Excel ou calc, une fonction reçoit en entrée des données et des arguments (=paramètres)
- Une fonction renvoie des données et peut avoir des effets collatéraux (ex : modification d'une colonne)

resultat ← **fonction**(donnee, arg=TRUE,...)

Fonctions disponibles avec R de base et différents packages

Ou possibilité de créer des fonctions (module R perfectionnement)



2.15 Les fonctions

Une multitude de fonctions : voir fiche mémo

- Retourne des informations sur les objets : **names()**, **ncol()**, **mode()** ...
- Numériques : **sum()**, **mean()** ...
- Traitements des chaînes de caractères : **substr()**, **toupper()**, ...

Il n'est pas nécessaire de connaître par cœur toutes les fonctions et leurs arguments : habitudes, mémo, aide de R, nombreuses ressources en ligne

The screenshot shows the RStudio interface. At the top, there is a menu bar with options: Files, Plots, Packages, Help, Viewer, and Presentation. Below the menu bar is a toolbar with icons for back, forward, home, and help. To the right of the toolbar is a search bar containing the word "mean". Underneath the search bar, there is a status bar displaying "R: Arithmetic Mean" and a "Find in Topic" button. The main content area is titled "Arithmetic Mean" and contains the text "mean {base}". In the bottom right corner of the content area, the text "R Documentation" is visible.



2.16 Les fonctions

On peut appliquer une fonction à un vecteur et donc à une colonne d'un data.frame, mais aussi à un élément

```
> tab_ind$nom <- str_to_lower(tab_ind$nom)
> tab_ind$nom
[1] "pierre"  "anne"    "marie"   "charles"
> mean(tab_ind$taille)
[1] 174.5
.
```

On peut modifier une colonne avec une fonction ou créer une nouvelle

```
> tab_ind$taille_m <- tab_ind$taille/100
> tab_ind$taille_m
[1] 1.85 1.67 1.71 1.75
> tab_ind$prenom_MAJ <- toupper(tab_ind$prenom)
> tab_ind$prenom_MAJ
[1] "PIERRE"  "ANNE"    "MARIE"   "CHARLES"
```

2.17 Exercice 1 (suite) : Découverte de RStudio

3) Programmes et manipulation des données

- Calculer la moyenne de la variable Sepal.Length
- Mettre en majuscule la variable Species

Est-ce que les modifications ont été stockées ?

AVANT LA SUITE DE LA FORMATION / PENSER A NETTOYER VOTRE ENVIRONNEMENT



3 Manipulation de données

3.1 Introduction

- La partie *Manipulation de données* va fournir les outils nécessaires aux traitements classiques à opérer sur une table de données.
- A part pour la lecture et l'écriture de fichiers, les outils proposés relèvent des packages **dplyr**) et **tidyverse**) : ils sont d'un usage plus aisné que les outils R de base et couvrent beaucoup des fonctionnalités les plus courantes.

Objectif: découvrir et utiliser les outils (*packages*) nécessaires aux traitements classiques à opérer sur une table de données

- Lecture et écriture de fichier : **readr**, **readxl**, etc.
- Fonctionnalités les plus courantes de manipulation des données : **dplyr** et **tidyverse**

3.2 Lecture de données

Créer un data.frame :

- `data.frame()`
- importer des données de différents formats

Format du fichier source	Fonction *
format de données « plats » (csv, txt...)	<code>read_delim, read_csv, read_csv2</code>
RDS	<code>readRDS</code>
XLSX, XLS	<code>read.xlsx, read_excel</code>
ODS	<code>read_ods</code>
SAS, SAV	<code>read_sas, read_spss</code>
parquet	<code>read_parquet</code>

*Le détail des fonctions est dans les diapos masquées ou facilement trouvable en ligne ou dans l'onglet help

3.3 Lecture de données

- Format des colonnes

Pour la plupart des formats (csv, txt,...), il faut spécifier le type des colonnes :

```
exploitations <- read_csv2("mes_donnees_exploitations.csv",
col_types = cols(.default = col_character(),
SAU = col_double(),
UGB = col_double()))
```

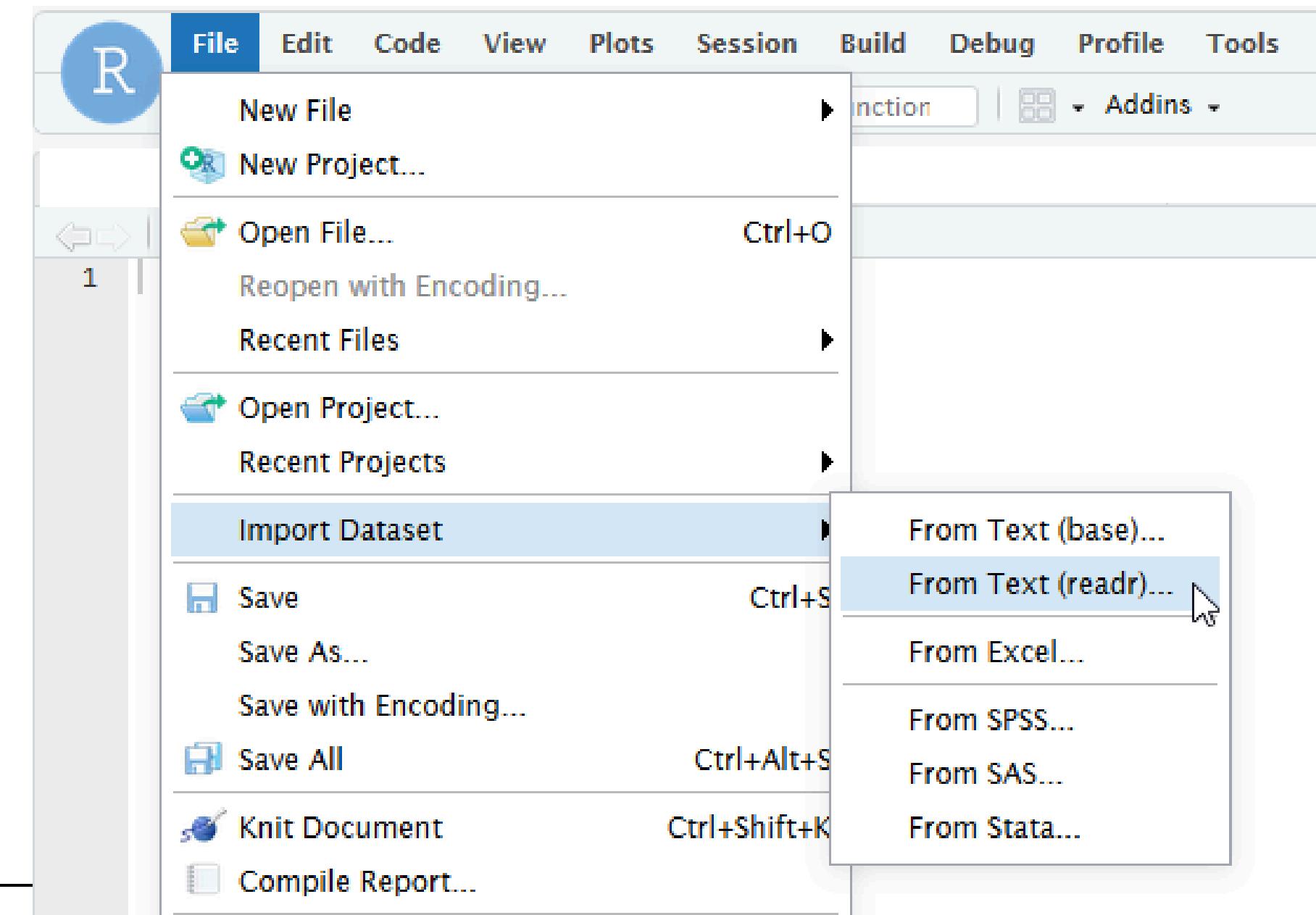
Avantage du format RDS :

- Conserve l'image exacte d'un objet : type de variable, encodage, etc.
- Format plus léger

C'est le format recommandé pour travailler sous R (enregistrement de tables de travail, de tables de mise à disposition, etc.)

3.4 Lecture de données

Rstudio dispose également d'un assistant d'import pour les fichiers text



3.5 Lecture de données

Import Text Data

File/Url: ~/CERISE/03-Espace-de-Diffusion/040_Pratiques_agricoles/4030_Pratiques_maraichage/PhytoLegumes2018/Données_définitives/20200629_Phyt...

Data Preview: répertoire des données

NOM_DOSSIER (character)	REG (character)	LIB_REG (character)	DEP (integer)	STRATE (character)	ESPECENQ (integer)	LIB_ESPECE (character)	COEF1 (double)	COEF2 (double)	PLEINTERIORSOL (integer)	CSPMODCULT (character)	PIYTOI (integer)
0197100001810	01	Guadeloupe	971	6_971_0	10	Salade	3.329386	1.42921348	1	SMA11	
0197100001810	01	Guadeloupe	971	6_971_0	10	Salade	3.329386	1.42921348	1	SAA11	
0197100001810	01	Guadeloupe	971	6_971_0	10	Salade	3.329386	1.42921348	1	SAA11	
0197100001810	01	Guadeloupe	971	6_971_0	10	Salade	3.329386	1.42921348	1	SAA11	

Previewing first 50 entries

prévisualisation du résultat de l'import

Import Options:

Name: X20200629_Phytolégi	<input checked="" type="checkbox"/> First Row as Names	Delimiter: Semicolon	Escape: None
Skip: 0	<input checked="" type="checkbox"/> Trim Spaces	Quotes: Default	Comment: Default
<input checked="" type="checkbox"/> Open Data Viewer		Locale: Configure...	NA: Default

paramètres à renseigner / modifier

Code Preview:

```
library(readr)
X20200629_Phytolégumes2018_TTMT_phyto_def <- 
read_delim("~/CERISE/03-Espace-de-Diffusion/040_Pratiques_agricoles/4030_Pratiques_maraichage/PhytoLegumes2018/Données_définitives/20200629_Phytolégumes2018_TTMT_phyto_def.csv",
";", escape_double = FALSE, trim_ws = TRUE)
```

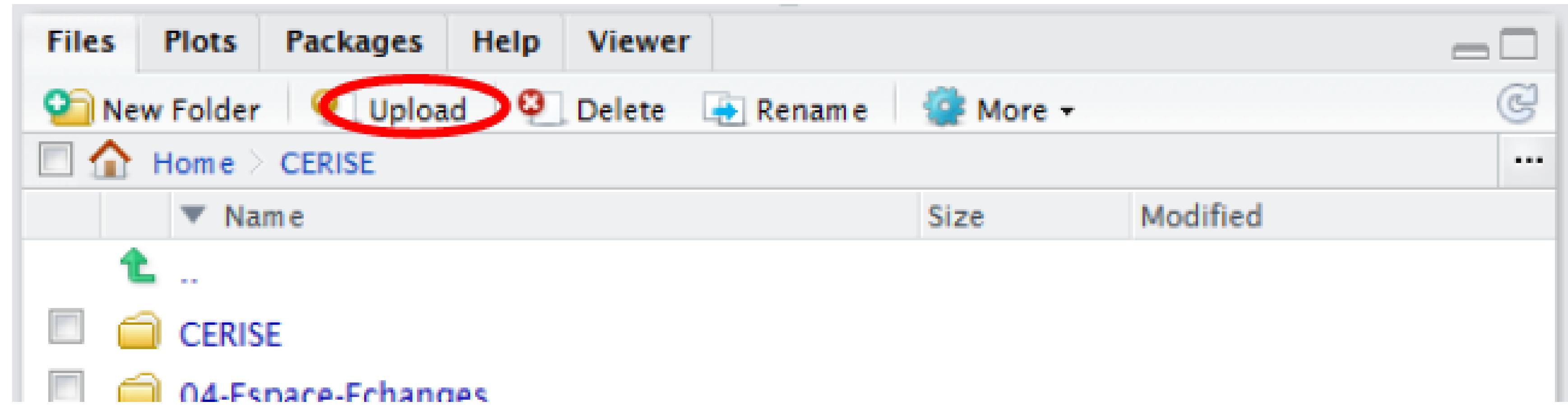
code généré pour l'import

À copier dans le programme, pour une bonne reproductibilité du code

3.6 Lecture de données

Sur le serveur, on ne peut importer des données que si elles sont sauvegardées sur le serveur. Pour importer des données enregistrées en local, il faut auparavant les enregistrer sur le serveur.

⇒ pour importer une table de l'ordinateur local sur Cerise, on utilise le bouton *Upload* de l'onglet *Files* en bas à droite de l'interface.



3.7 Ecriture de données

Enregistrer le **data.frame** résultat dans un fichier externe

Format du fichier source	Fonction *
format de données « plats » (csv, txt...)	<i>write_delim, write_csv, write_csv2 format recommandé pour l'utilisation dans d'autres logiciels que R</i>
RDS	<i>SaveRDS format recommandé pour l'utilisation dans R</i>
XLSX, XLS	<i>write.xlsx</i>
ODS	<i>write_ods</i>
SAS, SAV	NON RECOMMANDÉ
parquet	<i>write_parquet</i>

* Le détail des fonctions est dans les diapos masquées ou facilement trouvable en ligne ou dans l'onglet help



4 Outils DPLYR & TIDYR

4.1 Principes des outils

2 packages à charger en début de programme

- Objectif de *dplyr* : rassembler dans un seul package les outils de manipulation de données les plus importants pour l'analyse des données
 - ⇒ ensemble de fonctions correspondant à un **ensemble d'opérations élémentaires**
- Deux principes pour les packages *tidyr* et *dplyr* :
 - **Usage de fonctions « verbe »** toutes construites sur le même principe : le premier paramètre est la table de données sur laquelle on travaille.
 - **Usage de l'opérateur pipe (issu du package magrittr)**

Dans R-Studio, le raccourci clavier pour cet opérateur est : **Ctrl + Shift + M**

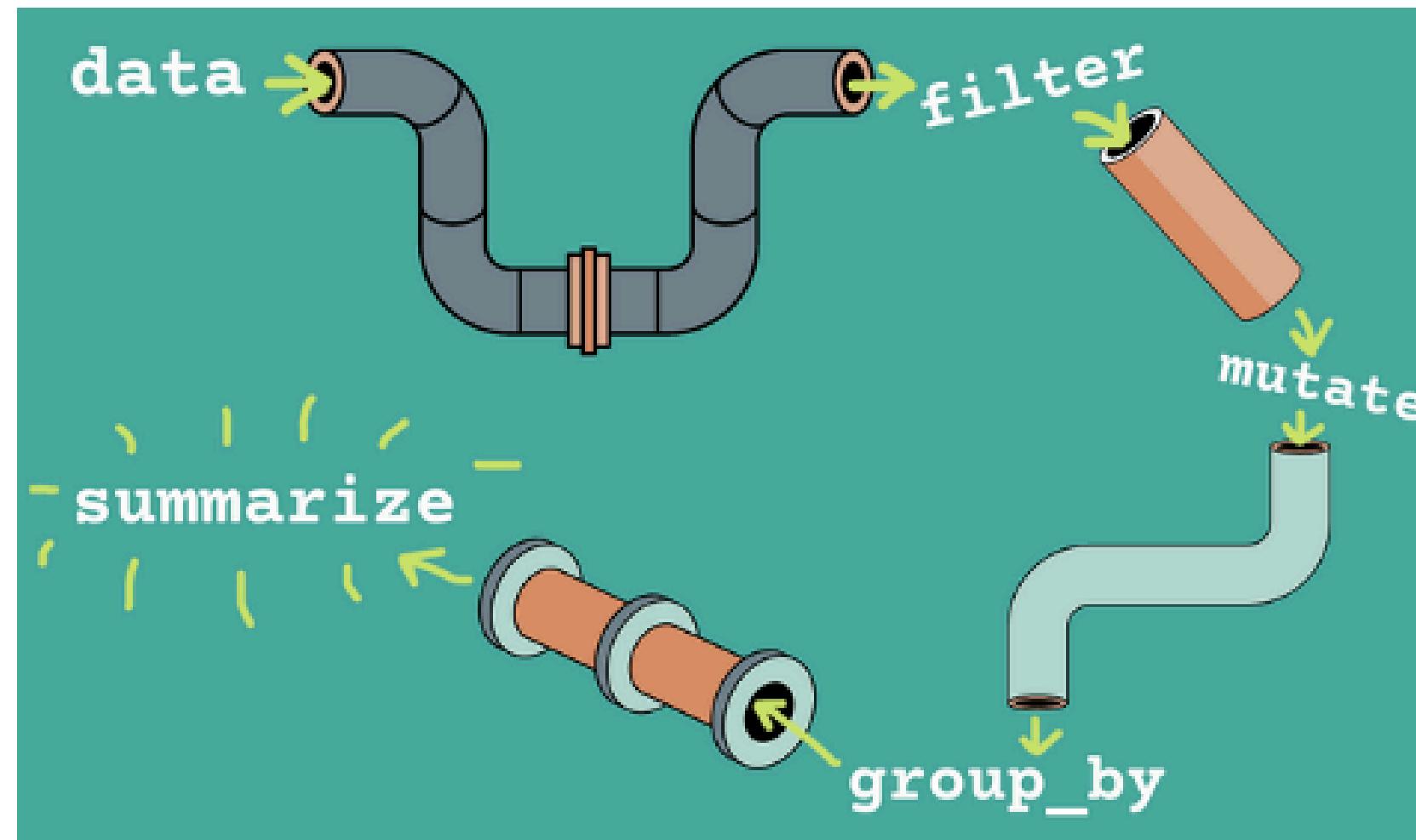


Nouvelle écriture

4.2 Principes des outils

L'opérateur pipe `%>%` ou `|>` permet d'enchaîner les traitements dans une seule commande.

- => Tout ce qui suit le pipe est appliqué à tout ce qui le précède



Exemple d'application :

```
data |>  
  filter[filtre de ligne](...) |>  
  mutate[création de colonne](...) |>  
  group_by[regroupement](...) |>  
  summarise[agrégat](...)
```



4.3 Principes des outils

- Tout ce qui suit le pipe est appliqué à tout ce qui le précède, appliquée à une situation du quotidien :

```
Compote ← Avec 1kg de sucre, peser 10 g %>%
  Verser dans la casserole %>% (c'est bien juste les 10 grammes de sucre qu'on verse et non le paquet complet!)
  Couper des pommes %>%
  Ajouter à la casserole %>% (on ajoute les pommes coupées et non pas complètes, car le pipe prend en compte l'étape précédente)
  Faire cuire %>% (On fait bien cuire l'ensemble pommes coupées + 10g de sucre)
  Écraser (On écrase pommes coupées + sucre CUIST)
```

- Un programme avec des enchaînements de pipes et de verbes peut se « traduire »

<pre>data_prenom %>% select(prenom, annee, region, occurence) %>% filter(region=="Occitanie") %>% mutate(prenom=toupper(prenom)) </pre>		<p>Je pars de ma base appelée <i>data_prenom</i>, je sélectionne les colonnes qui m'intéressent. Cette base réduite à 4 colonnes est ensuite filtrée pour n'avoir que la région Occitanie. Enfin, je viens modifier ma variable prénom.</p>
--	--	--



4.4 Principes des outils

Dans le cas d'enchaînement de traitements avec l'opérateur pipe, on peut aérer le code en allant à la ligne **après** chaque pipe.

ATTENTION : Si on passe à la ligne avant le pipe, la suite des traitements ne s'effectue pas, la commande s'arrête.

```
# On filtre les données de la Bretagne, on trie par surfaces décroissantes
# et on ajoute le libellé de la région
grand_ouest %>% select(starts_with("Numero"), AFO) %>%
  filter(Numero_Region == "53") %>%
  arrange(desc(AFO)) %>%
  mutate(Libelle_Region = "Bretagne")
```



*Liberté
Égalité
Fraternité*

4.5 Sélectionner des colonnes

Pour sélectionner des colonnes dans une table, on utilise la fonction **select()**

```
maTable %>% select(noms_des_colonnes)
```

- → on renseigne l'ensemble des colonnes à conserver, séparés par une virgule
- → pour anti-sélectionner, on fait précéder le nom de la colonne par le signe -
- → L'ordre des colonnes dans les parenthèses sera l'ordre dans la table en sortie



4.6 Sélectionner des colonnes

- Pour créer une nouvelle table :

```
nouvelle_table <- ma_table %>% select(nom_des_colonnes)
```

- Pour agir directement sur la même table :

```
ma_table <- ma_table %>% select(nom_des_colonnes)
```

Rappel: Si on ne fait pas l'affectation, on aura la table réduite affichée dans la console mais le changement ne sera pas stocké

```
prenom_reduit <- prenom %>% select(preusuel, nombre, dpt)
```

sexe	preusuel	annais	dpt	nombre
1	A	NA	XX	27
1	AADAM	NA	XX	38
1	AADEL	NA	XX	56
1	AADHIRAN	2023	93	5



preusuel	nombre	dpt
A	27	XX
AADAM	38	XX
AADEL	56	XX
AADHIRAN	5	93



4.7 Sélectionner des colonnes

Quelques outils supplémentaires :

Description	Syntaxe
Sélectionner la 2ème, la 3ème et les 6ème colonne de ma_table	ma_table %> % select(2, 3, 6)
Sélectionner l'ensemble des colonnes comprises entre les colonnes VarDeb et VarFin de ma_table	ma_table %> % select(VarDeb:VarFin)
Sélectionner les colonnes qui contiennent exactement « Surface » dans leur nom	ma_table %> % select(contains("Surface", ignore.case = FALSE))
Sélectionner les colonnes dont le nom commence par « SURF » ou « surf » ou « Surf »...	ma_table %> % select(starts_with("SURF", ignore.case = TRUE))
Sélectionner les colonnes dont le nom termine par « _1 »	ma_table %> % select(ends_with("_1"))



4.8 Filtrer des lignes

- Pour filtrer les lignes dans une table à l'aide de conditions logiques,

on utilise la fonction **filter()**

```
maTable %>% filter(conditions)
```

→ Les conditions peuvent être séparées par des virgules ou par les séparateurs de conditions habituelles &, | ou xor.



4.9 Filtrer des lignes

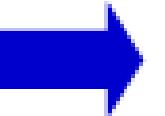
- Pour créer une nouvelle table :

```
nouvelle_table <- ma_table %>% filter(conditions)
```

- Pour agir directement sur la même table :

```
ma_table <- ma_table %>% filter(conditions)
```

sexe	preusuel	annais	dpt	nombre
1	A	NA	XX	27
1	AADAM	NA	XX	38
1	AADEL	NA	XX	56
1	AADHIRAN	2023	93	5
1	AADHIRAN	NA	XX	15
1	AADIL	1983	84	3
-	AADIL	1992	92	3



sexe	preusuel	annais	dpt	nombre
1	AADIL	1983	84	3
1	AADIL	1992	92	3
1	AADIL	NA	XX	187



4.10 Filtrer des lignes

Opérateur	Exemple syntaxe	Signification
<code>==</code>	<code>ma_table %>% filter(region == "53")</code>	On ne garde que les lignes pour lesquelles la variable région est égale à 53 → Attention au double égal !!!
<code>!=</code>	<code>ma_table %>% filter(region != "53")</code>	On garde toutes les lignes pour lesquelles la variable région est différente de 53
<code>></code>	<code>ma_table %>% filter(SAU > 10)</code>	On garde les lignes pour lesquelles la variable SAU est supérieure stricte à 10
<code><</code>	<code>ma_table %>% filter(SAU < 10)</code>	On garde les lignes pour lesquelles la variable SAU est inférieure stricte à 10



4.11 Filtrer des lignes

Opérateur	Exemple syntaxe	Signification
<code><=</code>	<code>ma_table %>% filter(SAU <= 10)</code>	On garde les lignes pour lesquelles la variable SAU est inférieure ou égale à 10
<code>%in%</code>	<code>ma_table %>% filter(reg %in% c("52", "53"))</code>	On garde les lignes pour lesquelles la variable SAU est inférieure ou égale à 10
<code>&</code>	<code>ma_table %>% filter(reg == "53" & SAU > 10)</code>	On garde les lignes pour lesquelles la variable reg vaut 53 et la SAU est supérieure stricte à 10 (les deux conditions en même temps)
<code> </code>	<code>ma_table %>% filter(SAU > 10 EFF > 50)</code>	On garde les lignes pour lesquelles la variable SAU est supérieure stricte à 10 ou la variable EFF est supérieure stricte à 50 (l'une ou



4.12 Filtrer des lignes

- On peut aussi filtrer les lignes avec la fonction **slice()**

```
maTable %>% slice(paramètres)
```



Exemples :

maTable %>% slice(1:10) ⇒ renvoie les lignes 1 à 10 de maTable
maTable %>% slice_max(nombre, n=10) ⇒
renvoie les 10 lignes de *maTable* avec les valeurs max de la variable *nombre*



4.13 Gérer les doublons

- Pour gérer les doublons dans une table (ex enlever les doublons), on utilise la fonction **distinct()**

```
maTable %>% distinct(var_doublon, .keep_all = FALSE)
```

var_doublon correspond aux variables qui, combinées, permettent d'identifier les doublons.

si *.keep_all = FALSE*, la table résultat ne contiendra que les variables citées dans var_doublon. Si *.keep_all = TRUE*, toutes les variables de la table sont conservées.

- On peut supprimer les lignes doublons d'une table facilement en écrivant :

```
maTable %>% distinct()
```

4.14 Gérer les doublons

name	homeworld	species
Luke Skywalker	Tatooine	Human
C-3PO	Tatooine	Human
R2-D2	Naboo	Droid
Darth Vader	Tatooine	Human
Leia Organa	Alderaan	Human
Owen Lars	Tatooine	Human
Beru Whitesun Lars	Tatooine	Human
RS-D4	Tatooine	Droid
Biggs Darklighter	Tatooine	Human
Obi-Wan Kenobi	Stewjon	Human

```
# Avoir la liste des especes des personnages de starwars
starwars %>% distinct(species)
```



```
species
<chr>
Human
Droid
Wookiee
Rodian
Hull
Yoda's species
Trandoshan
Mon Calamari
Ewok
...
```



Par défaut, seules les variables entre parenthèses sont conservées

```
# vérifier si il n'y pas de doublons dans prenom
> nrow(prenom)
[1] 3884324
> nrow(prenom %>% distinct())
[1] 3884324
```

4.15 Exercice 2 : Traitement de données

Objectif de l'exercice : à partir des données du RA2020, reproduire ce tableau

Tableau	Le recensement agricole en quelques chiffres		
	2010	2020	Évolution 2020/2010
Exploitations (mille)	490	389	- 21 %
dont à spécialisation végétale (mille)	221	202	- 9 %
à spécialisation animale (mille)	209	145	- 31 %
mixtes (polyculture, polyélevage) (mille)	58	41	- 30 %
Exploitations sous statut individuel (mille)	342	227	- 33 %
Part des exploitations en agriculture biologique ¹ (%)	4	12	+ 8 points
Part des exploitations sous autres signes officiels de qualité ou d'origine ² (%)	24	27	+ 3 points
Part des exploitations vendant en circuit court ³ (%)	18	23	+ 6 points
Chefs d'exploitation, coexploitants et associés actifs (millier de personnes)	604	496	- 18 %
dont ayant 60 ans ou plus (%)	20	25	+ 5 point
femmes (%)	27	26	- 1 point
Travail agricole (millier d'ETP) ⁴	740	659	- 11 %
SAU moyenne (ha) ⁵	55	69	+ 25 %
SAU totale (millier d'ha)	26 963	26 730	- 1 %
dont céréales, oléagineux, protéagineux (millier d'ha)	11 854	11 444	- 3 %
prairies (artificielles, temporaires, permanentes) (millier d'ha)	11 107	11 075	0 %
cultures permanentes (millier d'ha)	1 001	1 013	+ 1 %
Cheptel (millier d'UOB)	26 462	24 630	- 7 %

À faire :

- 1) Préparer l'environnement
- 2) Importer les données
- 3) Après avoir découvert les données, réduire la table aux informations (colonnes et lignes) nécessaires
- 4) Existe-t-il des doublons ?
- 5) Identifier les différentes orientations technico-économiques



4.16 Trier une table

Pour trier une table selon une ou plusieurs variables, on utilise la fonction **arrange()**

```
maTable %>% arrange(variables_de_tri)
```

→ possibilité de trier selon plusieurs colonnes : les variables de tri doivent être séparées par une virgule.

→ on encadre les variables qu'il faut trier de façon décroissante par la fonction **desc()**.



4.17 Trier une table

- Pour créer une nouvelle table :

```
nouvelle_table_triee <- ma_table %>% arrange(variables_de_tri)
```

- Pour agir directement sur la même table

```
ma_table <- ma_table %>% arrange(variables_de_tri)
```

name	height	mass	hair_color
Luke Skywalker	172	77.0	blond
C-3PO	167	75.0	NA
R2-D2	96	32.0	NA
Darth Vader	202	136.0	none
Leia Organa	150	49.0	brown
Owen Lars	178	120.0	brown, grey
Beru Whitesun Lars	165	75.0	brown
R5-D4	97	32.0	NA

```
# Trier en fonction de la taille du personnage
starwars_trie <- starwars %>%
  arrange(desc(height))|
```

name	height	mass	hair_color
Yarrel Poot	204	NA	none
Tarfful	234	136.0	brown
Lama Su	229	88.0	none
Chewbacca	228	112.0	brown

```
starwars_trie <- starwars %>%
  arrange(height)|
```

name	height	mass	hair_color
Yoda	65	17.0	white
Ratts Tyerell	79	15.0	none
Wicket Systri Warrick	88	20.0	brown



4.18 Renommer des variables

Pour renommer une ou plusieurs colonnes dans une table, on utilise la fonction **rename()**

```
maTable %>% rename(nouveau_nom = ancien_nom)
```

On peut renommer plusieurs variables dans un seul rename, en séparant les instructions par des virgules.



4.19 Renommer des variables

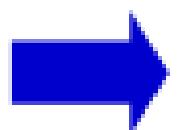
- Pour créer une nouvelle table :

```
nouvelle_table <- ma_table %>% rename(nouveau_nom = ancien_nom)
```

- Pour agir directement sur la même table

```
ma_table <- ma_table %>% rename(nouveau_nom = ancien_nom)
```

sexe	preusuel	annais	dpt	nombre
1	A	NA	XX	27
1	AADAM	NA	XX	38
1	AADEL	NA	XX	56



```
prenom <- prenom %>%  
rename(departement=dpt,annee=annais)
```

sexe	preusuel	annee	departement	nombre
1	A	NA	XX	27
1	AADAM	NA	XX	38
1	AADEL	NA	XX	56



4.20 Renommer des variables - Avancé !

- Pour renommer plusieurs variables de la même façon en même temps, on utilise la fonction **rename_with** sur le modèle suivant *rename_with(table, fonction de renommage, variables à renommer)*. Quelques exemples :

Description	Syntaxe
Passer tous les noms de colonnes de majuscules	<code>ma_table %>% rename_with(toupper)</code>
Passer tous les noms de colonnes de minuscules	<code>ma_table %>% rename_with(tolower)</code>
Suffixer tous les noms de colonnes par "_1"	<code>ma_table %>% rename_with(paste0, ... = "_1")</code>
Passer les noms des colonnes finissant par "CODE" en minuscules	<code>ma_table %>% rename_with(tolower, ends_with("CODE"))</code>
Suffixer par "_1" les noms des colonnes commençant par "REF"	<code>ma_table %>% rename_with(paste0, ... = "_1", starts_with("REF"))</code>
Passer en majuscules tous les noms des variables comprises entre var1 et var10	<code>ma_table %>% rename_with(toupper, var1:var10)</code>
Passer en minuscules tous les noms des variables de la 2 ^e à la 5 ^e colonne	<code>ma_table %>% rename_with(tolower, 2:5)</code>
Remplacer "Surf" par "S" dans tous les noms des variables	<code>ma_table %>% rename_with(str_replace, pattern = "Surf", replacement = "S")</code>

4.21 Exercice 2 : Traitement de données

À faire :

- Trier la table par département, OTEFDA_COEF17 et SAU_TOT décroissante
- Renommer la variable OTEFDA_COEF17 en OTEFDA, la variable UGBIFS.TOT en UGB et la variable SAU_TOT en SAU
- Supprimer le préfixe “SIEGE_” dans les variables géographiques.



Différents niveaux : variable par variable,
ou plus avancé mais recommandé : en une seule commande



`str_replace()` ou `str_remove()`



4.22 Créer et modifier des variables

Pour créer une nouvelle variable ou modifier une variable déjà existante dans une table, on utilise la fonction **mutate()**

```
maTable %>% mutate(variables_crées)
```

→ possibilité de créer plusieurs variables en une seule instruction, en séparant les différentes variables par des virgules

4.23 Créer et modifier des variables

On peut modifier ou créer une variable :

- À partir d'une constante
- À partir d'une ou plusieurs autres variables de la table
- À partir d'une ou plusieurs autres variables d'une autre table

Les expressions successives prennent en compte les résultats des calculs précédents

→ il est donc possible de créer une variable à partir d'une variable créée précédemment dans la même instruction

```
ma_table %>% mutate(ma_somme = varA + varB + varC,  
                      ma_moyenne = ma_somme/3)
```



Liberté
Égalité
Fraternité

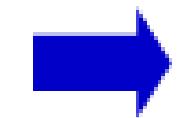
4.24 Créer et modifier des variables

Créer une variable se fait selon cette formule :

nom_nouvelle_variable = instructions

```
# Créer une variable IMC
# On utilise dans l'instruction IMC, la variable taille en mètre créée juste avant dans le même mutate
```

name	height	mass	hair_color
Luke Skywalker	172	77.0	blond
C-3PO	167	75.0	NA
R2-D2	96	32.0	NA
Darth Vader	202	136.0	none
Leia Organa	150	49.0	brown
Owen Lars	178	120.0	brown, grey
Beru Whitesun Lars	165	75.0	brown



```
starwars_data <- starwars_data %>%
  mutate(height_m = height/100,
        IMC = mass/(height_m*height_m),
        .after = mass)
```

name	height	mass	height_m	IMC	hair_color
Luke Skywalker	172	77.0	1.72	26.0258	blond
C-3PO	167	75.0	1.67	26.89232	NA
R2-D2	96	32.0	0.96	34.72222	NA
Darth Vader	202	136.0	2.02	33.33007	none
Leia Organa	150	49.0	1.50	21.77778	brown

4.25 Créer et modifier des variables

Par défaut : nouvelle variable ajoutée comme dernière colonne

⇒ possibilité d'indiquer la position des variables à créer dans la table. On utilise les paramètres `.before` et `.after` sur le modèle des paramètres de la fonction `relocate` :

```
> ma_table <- data.frame(V1 = 1, V2 = 2, V3 = 3, V4 = 4)
> # Je crée la variable V0 avant la variable V1
> ma_table <- ma_table %>% mutate(V0 = 0, .before = V1)
> ma_table
   V0 V1 V2 V3 V4
1  0  1  2  3  4
> # Je crée la variable V12 après la variable V2
> ma_table <- ma_table %>% mutate(V12 = V1+V2, .after = V2)
> ma_table
   V0 V1 V2 V12 V3 V4
1  0  1  2    3  3  4
```



4.26 Créer et modifier des variables

- Pour (re)coder une variable, on peut utiliser la fonction **case_when()**, qui s'écrit de la manière suivante :

**case_when(Condition1 ~ valeur_a,
Condition2 ~ valeur_b,
...,
TRUE ~ valeur_par_defaut)**

prenom_ls <- prenom_ls %>%
mutate(génération = case_when(annais <1990 ~ "Autres",
annais<1995 ~"Millenial",
annais <2010 ~"GenZ",
TRUE ~ "Autres"))

Les conditions s'exclues : Condition2 est vérifiée
avec comme acquis que 1 n'est pas vérifiée

prénom	annais	dpt	nombre	génération
LILIANE	1981	20	5	Autres
ALOIS	2011	60	3	Autres
STACY	2004	972	4	GenZ
FLORIAN	1993	52	19	Millenial

- Pour un (re)codage à deux conditions (si / sinon), on peut utiliser la fonction **if_else()** :

if_else(Condition1, valeur_a, valeur_b)

sexes <- sexes %>%
mutate(lib_sexe = if_else(sexe == 1,"masculin","feminin"))

sexe	prénom	lib_sexe
2	LILIANE	feminin
1	ALOIS	masculin



4.27 Exercice 2 : Traitement de données

À vous de créer ou modifier les variables nécessaires à la réalisations du tableau final !

Le tableau final devra contenir de “beaux” libellés de région et de département.

On va donc commencer par corriger les variables LIB_REG et LIB_DEP.

str_to_title() du package stringr

Le tableau résultat contient un décompte des exploitations selon qu’elles aient une spécialisation végétale, animale ou bien mixte. La variable OTEFDA est trop détaillée

⇒ créer la variable SPECIALISATION



4.28 Tris à plats

- Pour visualiser la répartition d'une variable catégorielle, on utilise la fonction **count()**

```
maTable %>% count(variable)
```

- → la fonction compte le nombre d'occurrences pour chaque modalité de la variable
- → la table en sortie contient deux variables : la variable pour laquelle on compte les modalités, et une variable appelée **n** qui contient les effectifs



4.29 Tris à plats

- Possibilité de calculer facilement des pourcentages, en ajoutant l'instruction suivante :

```
mutate(part = n/sum(n) * 100)
```

```
starwars_data %>%  
  count(sex) %>%  
  mutate(part = 100 * n/sum(n))
```

The diagram illustrates the execution of a dplyr pipeline. It starts with the code `starwars_data %>% count(sex) %>% mutate(part = 100 * n/sum(n))`. A green box highlights the `count(sex)` step, and a blue box highlights the `mutate(part = 100 * n/sum(n))` step. An arrow points from the blue box to a resulting data frame. This data frame has three columns: `sex` (a character vector), `n` (an integer vector), and `part` (a double vector). The data is as follows:

sex	n	part
<chr>	<int>	<dbl>
female	16	18.4
hermaphroditic	1	1.15
male	60	69.0
none	6	6.90
NA	4	4.60



4.30 Tris à plats

- Possibilité de compter les croisements de plusieurs variables : liste des variables à croiser, séparées par une virgule

```
diamonds %>% count(clarity, cut)

# A tibble: 40 x 3
#   clarity     cut     n
#   <ord>     <ord> <int>
# 1 I1       Fair    210
# 2 I1       Good     96
# 3 I1       Very Good  84
# 4 I1       Premium   205
# 5 I1       Ideal    146
# 6 SI2      Fair    466
# 7 SI2      Good   1081
```

- Possibilité d'ajouter une pondération, à l'aide du paramètre **wt**

```
diamonds %>% count(cut, wt = x)

# A tibble: 5 x 2
#   cut           n
#   <ord>     <dbl>
# 1 Fair      10058.
# 2 Good      28645.
# 3 Very Good 69359.
# 4 Premium   82386.
# 5 Ideal     89113.
```



4.31 Agréger des données

- Pour résumer les données d'une table en une seule statistique, on utilise la fonction **summarise()**

```
maTable %>% summarise(fonctions_stat(variable))
```

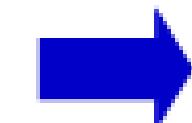
- → possibilité de calculer plusieurs statistiques, en séparant les instructions par une virgule
- → on peut utiliser les fonctions statistiques de base telles que max, min, mean, sd, n...

4.32 Agréger des données

- Résumer une variable se fait selon cette formule :

```
summarise(nom_variable_résumé = fonction_stat)
```

name	height	mass
Luke Skywalker	172	77.0
C-3PO	167	75.0
R2-D2	96	32.0
Darth Vader	202	136.0
Leia Organa	150	49.0
Owen Lars	178	120.0
Beru Whitesun Lars	165	75.0
R5-D4	97	32.0
Biggs Darklighter	183	84.0
Obi-Wan Kenobi	182	77.0



```
# Données agrégées de la masse des personnages
starwars_data %>%
  summarise(min = min(mass, na.rm=T),
            moyenne = mean(mass, na.rm=T),
            max= max(mass, na.rm=T))
```

	min	moyenne	max
	<dbl>	<dbl>	<dbl>
	15	97.3	1358



4.33 Agréger des données - Avancé !

Pour calculer des statistiques sur plusieurs variables à la fois et/ou selon plusieurs fonctions, on utilisera **across()**.
Quelques exemples :

Description	Syntaxe
Calculer la moyenne de toutes les variables de la table	<code>maTable %>% summarise(across(everything(), mean))</code>
Calculer la somme de chacune des variables numériques de la table	<code>maTable %>% summarise(across(where(is.numeric), sum))</code>
Calculer la somme de chacune des variables numériques de la table	<code>maTable %>% summarise(across(everything(), ~sum(is.na(.))))</code>
Calculer la moyenne et la somme, d'une variable x de la table	<code>maTable %>% summarise(across(x, list(moy = mean, somme = sum)))</code>
Calculer la moyenne et la somme de toutes les variables de la table	<code>maTable %>% summarise(across(everything(), list(moy = mean, somme = sum)))</code>



4.34 Agréger des données par groupe

- Pour agréger les données d'une table par groupe d'une variable catégorielle on utilise la fonction **group_by()** avant la fonction **summarise()** :

```
maTable %>%  
  group_by(var_groupe) %>%  
  summarise(fonctions_stat(variable))  
%>% ungroup()
```

- → On peut regrouper selon plusieurs variables, séparées par des virgules



4.35 Agréger des données par groupe

```
# taille moyenne selon l'espèce
starwars_data %>%
  group_by(species) %>%
  summarise(taille_moyenne = mean(height,na.rm=T))
```

species	taille_moyenne
<chr>	<dbl>
Aleena	79
Besalisk	198
Cerean	198
Chagrian	196
Clawdite	168
Droid	131.
Dug	112
Ewok	88

Rappel: pour cet exemple, pas d'affectation, le résultat s'affiche dans la console mais ne sera pas stocké



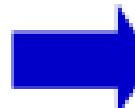
4.36 Agréger des données par groupe

On peut aussi combiner la fonction `group_by()` avec la fonction `mutate()` : dans ce cas les statistiques par groupe sont directement intégrées à la table de données individuelles

```
# On ajoute la taille moyenne de l'espèce à la table individuelle
# cela pourrait par exemple servir à créer une variable : écart de la moyenne de son espèce

starwars_data <- starwars_data %>%
  group_by(species) %>%
  mutate(taille_moyenne = mean(height,na.rm=T), .after = height)
```

name	height	mass
Luke Skywalker	172	77.0
C-3PO	167	75.0
R2-D2	96	32.0
Darth Vader	202	136.0
Leia Organa	150	49.0



name	height	taille_moyenne	mass
Luke Skywalker	172	176.6452	77.0
C-3PO	167	131.2000	75.0
R2-D2	96	131.2000	32.0
Darth Vader	202	176.6452	136.0
Leia Organa	150	176.6452	49.0



4.37 Exercice 2 : Traitement de données

Données agrégées de la région

⇒ Penser à affecter le résultat !

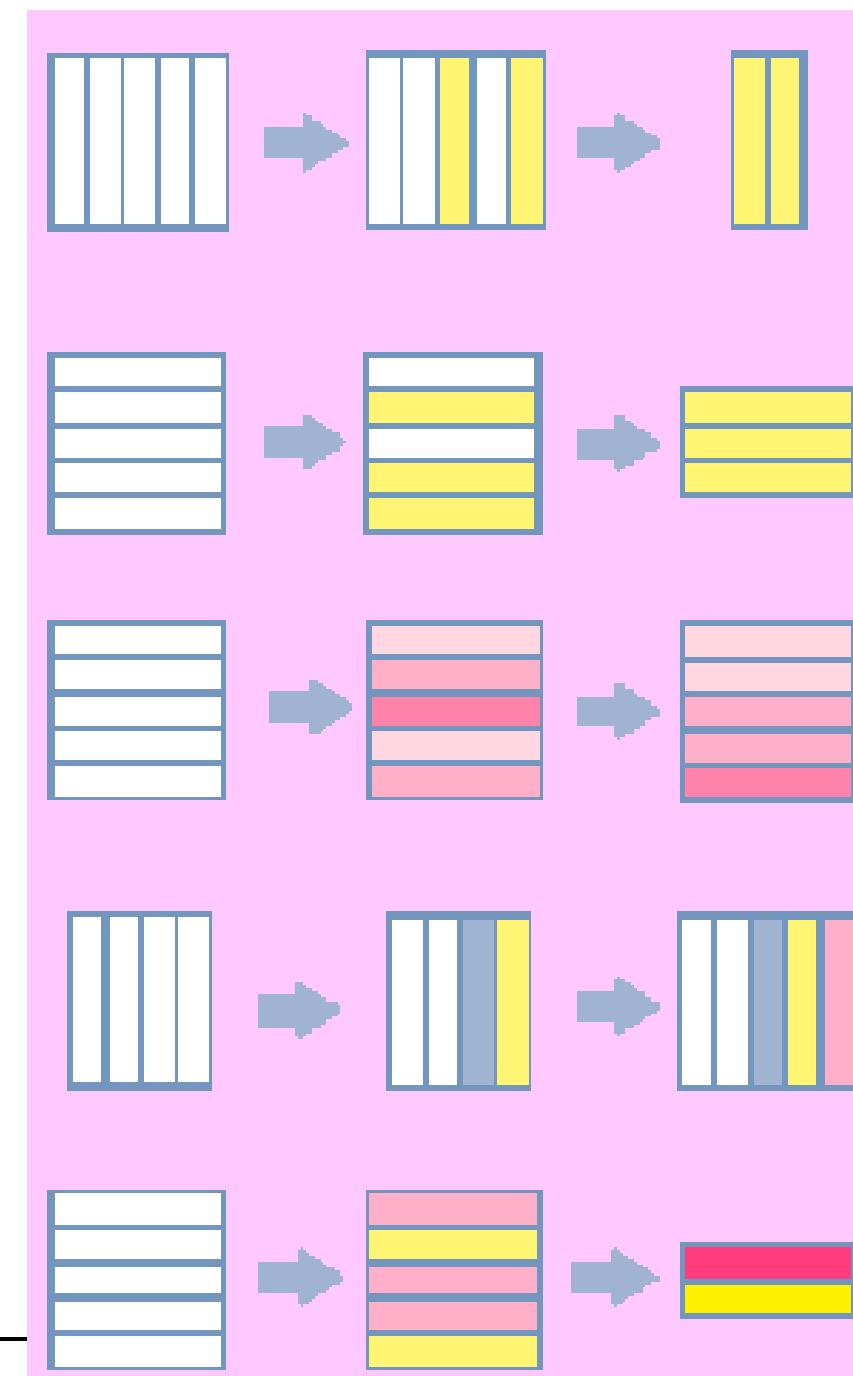
Dénombrer le nombre d'exploitations relevant de chaque spécialisation

⇒ Penser à affecter le résultat !

Et par département ? Faire les calculs agrégés à ce niveau



4.38 Exercice 2 : Aide mémoire



`select()`

`filter()`

`arrange()`

`mutate()`

`group_by() +
summarise()`



4.39 Concaténer des tables

Pour accolter deux tables horizontalement (rajouter des lignes), on utilise la fonction **bind_rows()**

```
bind_rows(maTable1, maTable2)
```

ou précédé d'un pipe

```
maTable1 %>% bind_rows(maTable2)
```

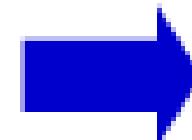
⇒ Les colonnes ayant le même nom vont s'empiler. Les autres vont se compléter avec des valeurs manquantes.

La fonction **bind_cols()** permet de la même façon de rajouter des **colonnes** à une table sans passer par une variable de jointure.

Attention: sans variable de jointure signifie que rien ne garantit que les lignes correspondent

ville	region
Toulouse	70
Bordeaux	75
Lyon	84

region	lib
70	Occitanie
94	Corse
75	Nouvelle Aquitaine



ville	region...2	region...3	lib
Toulouse		70	Occitanie
Bordeaux		75	Corse
Lyon	84	75	Nouvelle Aquitaine

4.40 Concaténer des tables

prenom_Camille_fille

Occurrence du prénom Camille chez les filles dans l'Ain à partir de 2016

sexe	preusuel	annais	dpt	nombre
2	CAMILLE	2016	01	12
2	CAMILLE	2017	01	11
2	CAMILLE	2018	01	10
2	CAMILLE	2019	01	8
2	CAMILLE	2020	01	9
2	CAMILLE	2021	01	11
2	CAMILLE	2022	01	8
2	CAMILLE	2023	01	4

prenom_Camille_garcon

Occurrence du prénom Camille chez les garçons dans l'Ain à partir de 2016

sexe	preusuel	annais	dpt	nombre
1	CAMIILL	2016	01	9
1	CAMIILL	2017	01	5
1	CAMIILL	2018	01	10
1	CAMIILL	2019	01	8
1	CAMIILLE	2020	01	8
1	CAMIILLE	2021	01	8
1	CAMIILLE	2022	01	5
1	CAMIILLE	2023	01	6

`prenom_camille <- prenom_camille_fille %>% bind_rows(prenom_camille_garcon)`



sexe	preusuel	annais	dpt	nombre
2	CAMIILL	2016	01	12
2	CAMIILLE	2017	01	11
2	CAMIILLE	2018	01	10
2	CAMIILLE	2019	01	8
2	CAMIILL	2020	01	9
2	CAMIILL	2021	01	11
2	CAMIILLE	2022	01	8
2	CAMIILLE	2023	01	4
1	CAMIILLE	2016	01	9
1	CAMIILL	2017	01	5
1	CAMIILLE	2018	01	10
1	CAMIILLE	2019	01	8
1	CAMIILLE	2020	01	8
1	CAMIILLE	2021	01	8
1	CAMIILL	2022	01	5
1	CAMIILL	2023	01	6



*Liberté
Égalité
Fraternité*

4.41 Joindre des tables avec une variable de jointure

Pour fusionner deux tables en utilisant une ou plusieurs variables de jointure, on utilise les fonctions **XXX_join()**

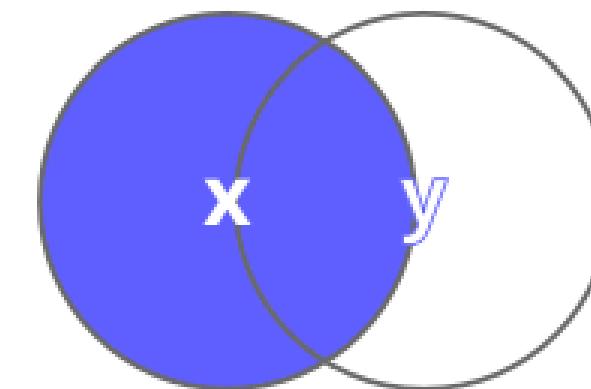
```
XXX_join(maTable1, maTable2, by = variables_de_fusion) # ou précédé d'un pipe maTable1 %>%  
XXX_join(maTable2, by = variables_de_fusion)
```

Les noms des variables de jointures doivent être indiqués entre guillemets **Il n'est pas nécessaire de trier les tables avant de faire la jointure**

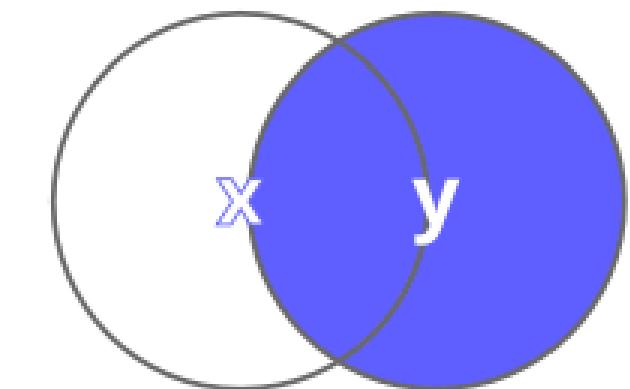


4.42 Joindre des tables avec une variable de jointure

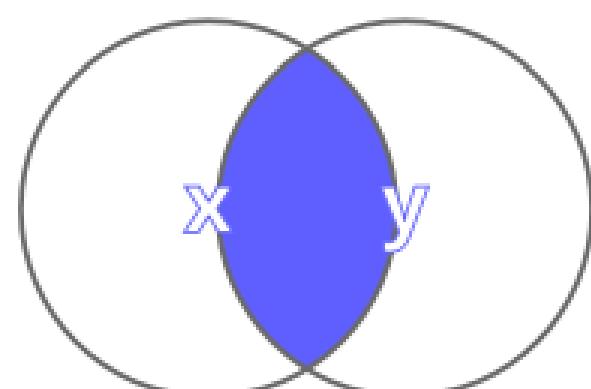
`left_join(x, y)`



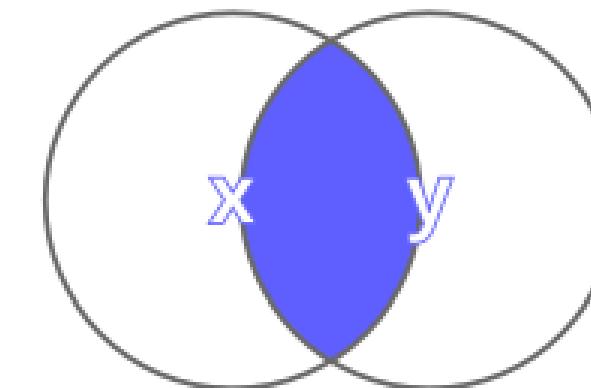
`right_join(x, y)`



`inner_join(x, y)`

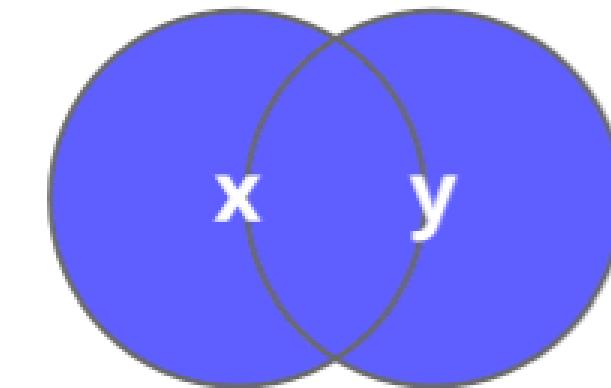


`semi_join(x, y)`

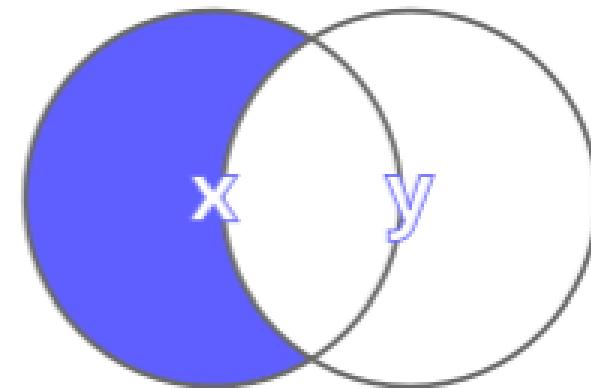


(never duplicate rows of x)

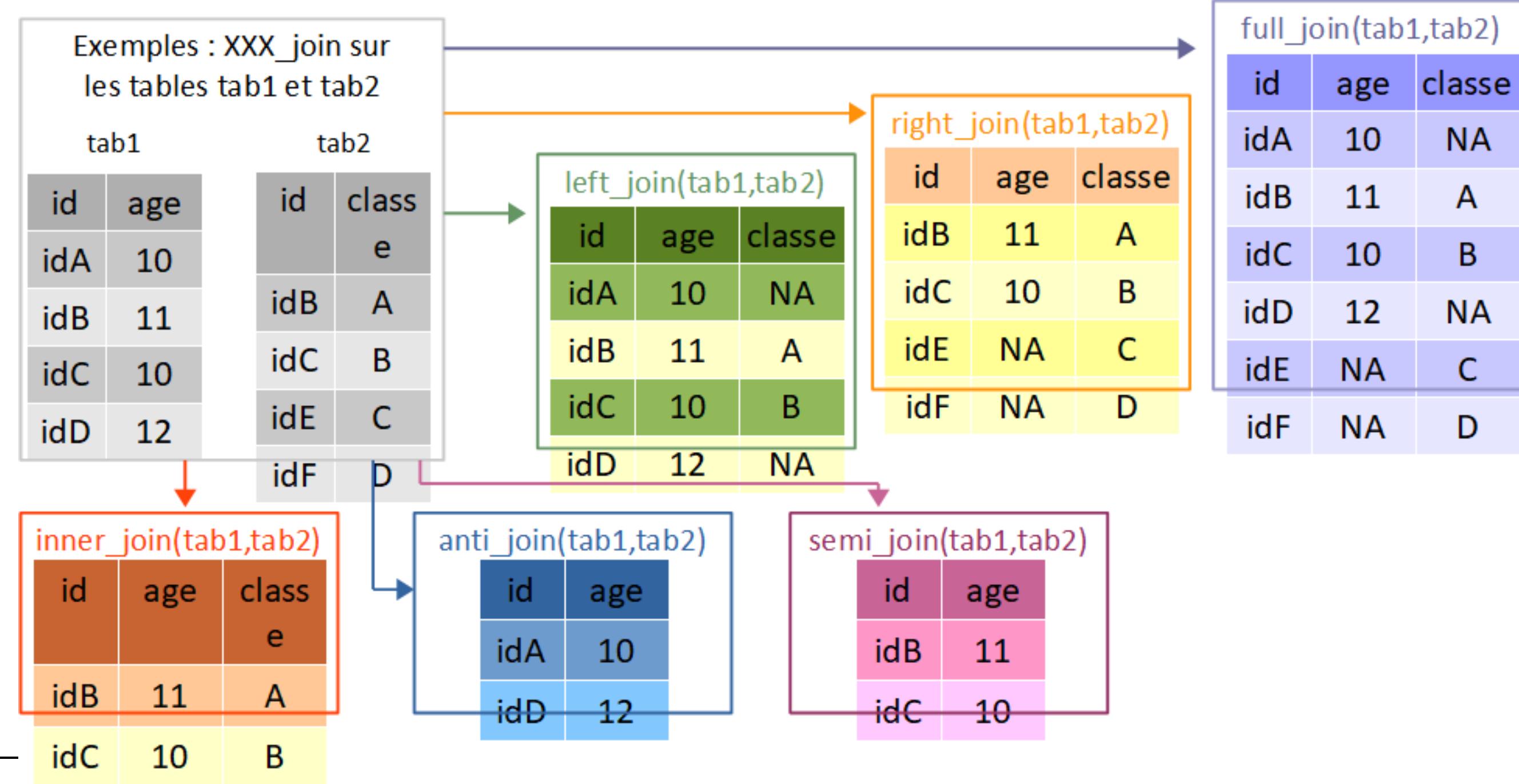
`full_join(x, y)`



`anti_join(x, y)`



4.43 Joindre des tables avec une variable de jointure





4.44 Joindre des tables avec une variable de jointure

- Il est possible de réaliser une jointure à l'aide de plusieurs variables identifiantes : l'argument **by** s'écrit alors `c("id1","id2", ...)`
- Lorsque les variables de jointure ont des noms différents dans les deux tables, l'argument **by** prend comme paramètre un vecteur du type `c("id1_tab1" = "id1_tab2","id2_tab1" = "id2_tab2",...)`
- Si rien n'est précisé, la fusion se fait sur l'ensemble des variables portant le même nom dans les deux tables



4.45 Joindre des tables avec une variable de jointure

- Lorsque des variables ont le même nom dans deux tables différentes mais ne sont pas des variables de jointure, on peut choisir le suffixe à leur donner dans la table résultat avec l'argument **suffix**. Il s'écrit sous la forme :

```
suffix = c('suffix_tab1', 'suffix_tab2')
```

Remarque:

- Toutes les variables ne seront pas suffixées selon leur table d'origine, seulement celles possédant le même nom entre les deux tables.



4.46 Joindre des tables avec une variable de jointure

surfaces_2010

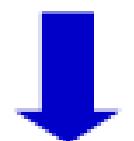
Numero_region	Numero_Dept	Surface
53	22	9793
53	29	12625
53	35	7536
53	56	8895

surfaces_2020

Numero_region	Numero_Dept	Surface
53	22	10034
53	29	13728
53	35	6368
53	56	8095

```
surfaces_2010 %>% full_join(surfaces_2020,
  by = c("Numero_region", "Numero_Dept"),
  suffix = c("_10", "_20"))
```

|



Numero_region	Numero_Dept	Surface_10	Surface_20
53	22	9793	10034
53	29	12625	13728
53	35	7536	6368
53	56	8895	8095



4.47 Réorganiser des tables

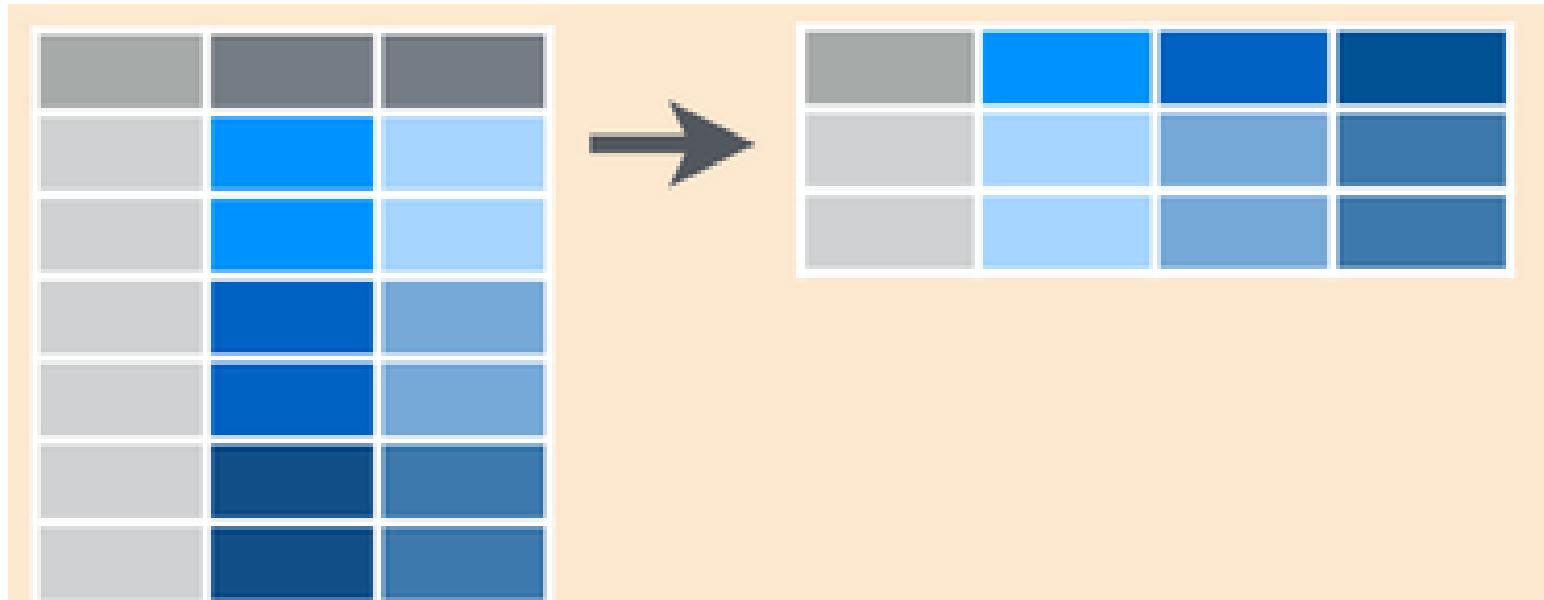
- Pour réorganiser une table, c'est à dire passer des lignes en colonnes ou inversement, on utilise les fonctions **pivot_wider()** et **pivot_longer()** du package tidyverse.

```
pivot_longer(maTable, cols, names_to, values_to) # ou précédé d'un pipe
maTable %>% pivot_longer(cols, names_to, values_to)
pivot_wider(maTable, names_from, values_from) # ou précédé d'un pipe
maTable %>% pivot_wider(names_from, values_from)
```



4.48 Réorganiser des tables

- `pivot_wider()` permet de disperser les lignes dans les colonnes



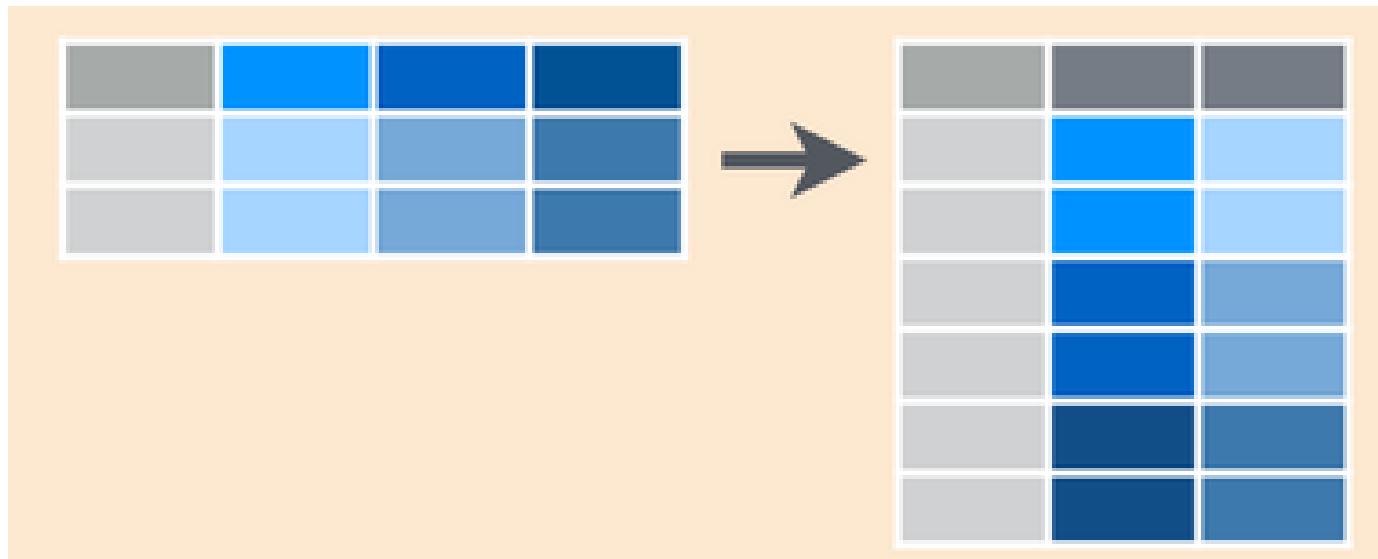
*maTable %>%
pivot_wider(names_from, values_from)*

names_from = nom de la variable dont les modalités deviendront les intitulés de colonne **values_from** = nom de la variable à utiliser pour remplir les colonnes



4.49 Réorganiser des tables

- `pivot_longer()` est l'opération inverse : elle permet de rassembler les colonnes dans les lignes



```
maTable %>%  
pivot_longer(cols, names_to, values_to)
```

cols = colonnes pivotantes

names_to = nom de la première colonne à créer et qui recevra les actuels noms des colonnes pivotantes

values_to = nom de la deuxième colonne à créer et qui recevra les valeurs des colonnes pivotantes

Moyen mnémotechnique pour différencier les deux fonctions : avec pivot_longer, la table devient plus longue

4.50 Réorganiser des tables

```
# Passage de tab1 à tab2 avec la fonction pivot_wider()
tab2 <- tab1 %>% pivot_wider(names_from = mesure, values_from = temperature)
```

```
# Passage de la tab2 à la tab1 avec la fonction pivot_longer()
tab1 <- tab2 %>% pivot_longer(cols=min:moyenne, names_to="mesure", values_to = "temperature")
```

Bien mettre des guillemets

annee	mois	mesure	temperature
2020	janvier	min	-10
2020	janvier	max	12
2020	janvier	moyenne	3
2020	fevrier	min	-6
2020	fevrier	max	14
2020	fevrier	moyenne	5

pivot_longer()



pivot_wider()

annee	mois	min	max	moyenne
2020	janvier	-10	12	3
2020	fevrier	-6	14	5



4.51 Réorganiser des tables - Avancé !

Quelques paramètres supplémentaires et utiles de la fonction `pivot_longer`:

	family_id	age_mother	birth_child1	birth_child2	birth_child3
1	1	30	1998-11-26	2000-01-29	<NA>
2	2	27	1996-06-22	<NA>	<NA>
3	3	26	2002-07-11	2004-04-05	2007-09-02
4	4	32	2004-10-10	2009-08-27	2012-07-21
5	5	29	2000-12-05	2005-02-28	<NA>

	family_id	age_mother	child_number	date_of_birth
	<i><int></i>	<i><dbl></i>	<i><chr></i>	<i><chr></i>
1	1	30	1	1998-11-26
2	1	30	2	2000-01-29
3	2	27	1	1996-06-22
4	3	26	1	2002-07-11
5	3	26	2	2004-04-05
6	3	26	3	2007-09-02
7	4	32	1	2004-10-10
8	4	32	2	2009-08-27
9	4	32	3	2012-07-21
10	5	29	1	2000-12-05
11	5	29	2	2005-02-28

`ma_df %>% pivot_longer(`

`cols = birth_child1:birth_child3, ← colonnes à pivoter (ou starts_with("birth_child"))`

`names_to = "child_number", ← nom de la colonne recevant les noms des colonnes à`

`pivoter values_to = "date_of_birth", ← nom de la colonne & recevant les valeurs des colonnes pivotées`

`names_prefix = "birth_child", ← préfixe à enlever des noms de colonnes à pivoter`

`values_drop_na = TRUE ← le résultat ne contient pas des lignes avec des NA dans la colonne indiquée dans le paramètre values_to`

)



4.52 Exercice 2 : Traitement de données

Transposer les tables spécialisation pour avoir une ligne par département et une ligne total régional

1. Fusionner les tables départementales
2. Accoler les tables régionales
3. => Concaténer ces deux tables

Selon le temps :

Denier chipotage pour une belle table à exporter :

- Déplacer les variables de SAU et d'UGB en fin de table
- Exprimer les variables SAU_tot et UGB_tot en milliers.
- Arrondir les variables SAU_tot, SAU_moy et UGB_tot à l'unité.
- Renommer les colonnes “en français”.
- **Export du résultat !**



4.53 Aide mémoire

Nom de la fonction	Usage
<code>select()</code>	sélectionner des variables
<code>relocate()</code>	réorganiser des variables
<code>rename()</code>	renommer des variables
<code>filter()</code>	sélectionner des observations selon une ou plusieurs conditions
<code>mutate()</code>	créer ou modifier des variables
<code>summarise()</code>	agréger les observations en effectuant une fonction « résumé » sur une ou plusieurs variables
<code>arrange()</code>	ordonner les observations selon une ou plusieurs variables
<code>group_by()</code>	grouper les observations selon une ou plusieurs variables (avant d'effectuer des calculs)
<code>pivot_wider()</code>	réorganiser les variables dans les lignes
<code>pivot_longer()</code>	réorganiser les variables dans les lignes ^{112 / 113}

4.54 Liens utiles

- Introduction à R et au tidyverse : <https://juba.github.io/tidyverse/> (en français)
- Formations R aux MTES & MCTRCT : <https://mtes-mct.github.io/parcours-r/> (en français)
- Formations sous GEDSI :

<https://ged.national.agri/gedsi/nxpath/default/SSP/workspaces/Partage%20SRISE/Support%20Formations/Formations/tabIds=%3A&conversationId=0NXMAIN2>

- Forum CIRAD : <http://forums.cirad.fr/logiciel-R/> (forum francophone)
- Les aides-mémoires : <https://www.rstudio.com/resources/cheatsheets/> (aides-mémoire en anglais)
- Sans oublier les recherches Google qui renvoient très souvent une aide en français ou Chat GPT