

Bulle d'R. Manipulation de données dans l'univers du tidyverse

Sandrine LYSER

18 février 2025

Table des matières

Préambule. Jeu de données utilisé dans cette session	3
1 Quelques rappels	3
1.1 L'univers du tidyverse	3
1.2 La commande pipe	4
1.3 Les packages utiles pour la manipulation des données	4
1.4 Qu'est-ce-qu'un {tibble} ?	4
2 Manipulations des données	6
2.1 Manipulations avec le package {dplyr}	6
2.1.1 Sélectionner des lignes avec la fonction slice()	6
2.1.2 Filtrer des observations avec la fonction filter()	7
2.1.3 Supprimer les doublons avec la fonction distinct()	9
2.1.4 Trier les données avec la fonction arrange()	10
2.1.5 Sélectionner des colonnes avec select()	11
2.1.6 Créer/ajouter de nouvelles variables avec mutate()	13
2.1.7 Réordonner les variables avec la fonction relocate()	16
2.1.8 Modifier le nom d'une variable avec rename()	17
2.1.9 Combiner des jeux de données	17
2.1.10 Ajouter une colonne d'identifiants avec rownames_to_column()	20
2.1.11 Calculer des statistiques récapitulatives	21
2.1.12 Effectuer des opérations groupées avec la fonction group_by()	23
2.1.13 Fiche aide-mémoire {dplyr}	25
2.2 Manipulations avec le package {tidyr}	25
2.2.1 Transformer des colonnes en lignes avec pivot_longer()	26
2.2.2 Transformer des lignes en colonnes avec pivot_wider()	26
2.2.3 Séparer une colonne en plusieurs colonnes avec separate()	27
2.2.4 Regrouper plusieurs colonnes en une seule avec unite()	27
2.2.5 Supprimer les lignes contenant des valeurs manquantes avec drop_na()	28
2.2.6 Remplacer les valeurs manquantes par des valeurs spécifiées avec replace_na()	28
2.2.7 Fiche aide-mémoire {tidyr}	29
2.3 Manipulations des chaînes de caractères avec le package {stringr}	29
2.3.1 Détection d'une expression avec la fonction str_detect()	29
2.3.2 Détection d'une expression avec la fonction str_starts()	30
2.3.3 Remplacement d'une expression avec la fonction str_replace()	30
2.3.4 Conversion du texte en majuscules avec la fonction str_to_upper()	31
2.3.5 Conversion du texte en minuscules avec la fonction str_to_lower()	31
2.3.6 Conversion du texte en minuscules avec la fonction str_to_title()	32
2.3.7 Fiche aide-mémoire {stringr}	32
2.4 Manipulation des facteurs avec le package {forcats}	32
2.4.1 Réorganiser les niveaux d'un facteur selon leur importance avec fct_infreq()	33
2.4.2 Réorganiser les niveaux des facteurs manuellement avec fct_relevel()	33
2.4.3 Réorganiser les niveaux en fonction des valeurs d'une variable numérique avec fct_reorder()	34
2.4.4 Inverser l'ordre des niveaux avec fct_rev()	35

2.4.5	Changer le niveau d'un facteur avec <code>fct_recode()</code>	35
2.4.6	Modifier le nom des niveaux de manière globale avec <code>fct_relabel()</code>	36
2.4.7	Regrouper les niveaux d'un facteur en catégories manuellement définies avec <code>fct_collapse()</code>	36
2.4.8	Conserver le(s) niveau(x) le(s) plus commun(s) et grouper les autres avec <code>fct_lump()</code> .	37
2.4.9	Anonymiser les niveaux avec <code>fct_anon()</code>	38
2.4.10	Ajouter de nouveaux niveaux à un facteur avec <code>fct_expand()</code>	38
2.4.11	Supprimer les niveaux de facteurs inutilisés avec <code>fct_drop()</code>	39
2.4.12	Lister les niveaux d'un facteur <code>fct_unique()</code>	40
2.4.13	Compter la fréquence des niveaux d'un facteur <code>fct_count()</code>	40
2.4.14	Combiner les niveaux de plusieurs facteurs <code>fct_cross()</code>	40
2.4.15	Fiche aide-mémoire <code>{forcats}</code>	41

3 Ressources et références bibliographiques 41

Aides et ressources en ligne	41
3.1 Packages	41

Préambule. Jeu de données utilisé dans cette session

hdv2003

- disponible dans le package `{questionr}`
- échantillon tiré de l'enquête Histoire de vie, réalisée en 2003 en France, par l'Insee, auprès de la population âgée de 18 ans et plus
- 2000 lignes et 20 variables

Dictionnaire des variables

Variable	Description	Type
id	Identifiant unique de l'individu	entier
age	Âge de l'individu	entier
sexe	Sexe de l'individu	facteur (Femme, Homme)
nivetud	Niveau d'études	facteur (8 modalités)
poids	Poids de l'individu dans l'échantillon	numérique
occup	Occupation principale	facteur (7 modalités)
qualif	Qualification professionnelle	facteur (7 modalités)
freres.soeurs	Nombre de frères et soeurs	entier
clso	Classe sociale	facteur (3 modalités)
relig	Religion	facteur (6 modalités)
trav.imp	Importance accordée au travail	facteur (4 modalités)
trav.satisf	Satisfaction au travail	facteur (3 modalités)
hard.rock	Écoute du hard rock	facteur (Oui, Non)
lecture.bd	Lecture de bandes dessinées	facteur (Oui, Non)
peche.chasse	Pratique de la pêche ou de la chasse	facteur (Oui, Non)
cuisine	Pratique de la cuisine	facteur (Oui, Non)
bricol	Pratique du bricolage	facteur (Oui, Non)
cinema	Fréquentation du cinéma	facteur (Oui, Non)
sport	Pratique d'un sport	facteur (Oui, Non)
heures.tv	Nombre d'heures passées devant la télévision	numérique

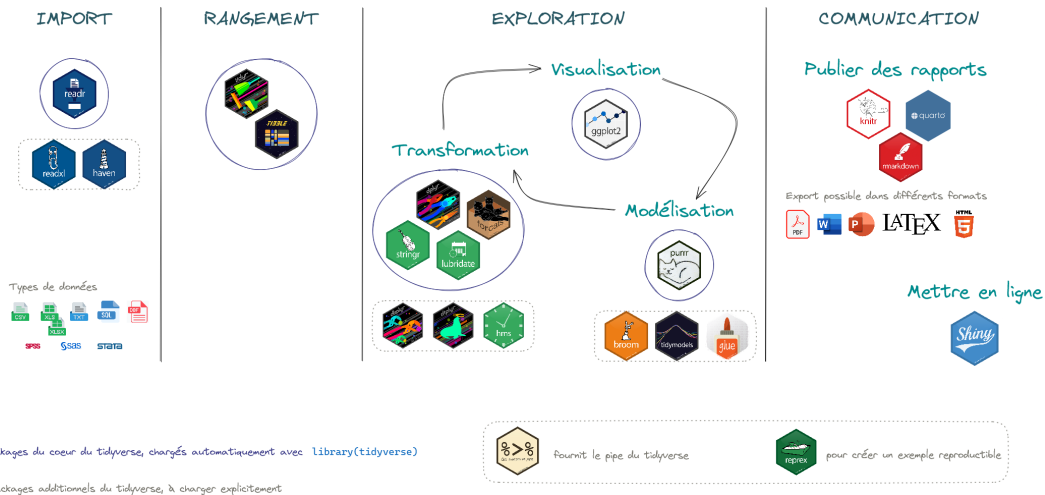
1 Quelques rappels

1.1 L'univers du `tidyverse`

Ensemble de packages partageant une même philosophie, structure et approche de la manipulation des données, conçus pour faciliter la manipulation, l'analyse et la visualisation des données

```
install.packages("tidyverse")  
# installation simultanée de plusieurs packages  
  
library("tidyverse")  
# chargement des 9 packages au coeur du tidyverse
```

LES DIFFÉRENTES ÉTAPES DE LA DATA SCIENCE Les packages du Tidyverse & leurs principaux objectifs



1.2 La commande pipe

- Le pipe `%>%`, introduit dès 2014 dans le package `{magrittr}`, permet d'effectuer des opérations successives de traitement
- Principe** : le pipe prend la sortie d'une fonction (ou d'une expression) et la passe automatiquement comme **premier** argument à la fonction suivante

```
data %>%
  fonction1() %>%
  fonction2() %>%
  fonction3()
# équivaut à
fonction3(fonction2(fonction1(data)))
```

- En 2021 (R version 4.1.0), sortie du pipe natif `|>`, dont le fonctionnement reste identique à l'autre pipe

```
data |>
  fonction1() |>
  fonction2() |>
  fonction3()
```

⇒ Il est conseillé d'utiliser la commande `|>`

1.3 Les packages utiles pour la manipulation des données

- Les indispensables**
 - `{dplyr}` : manipulation des données
 - `{tidyr}` : nettoyage/remise en forme/formatage des données
 - `{stringr}` : manipulation des chaînes de caractères
 - `{forcats}` : traitement des variables qualitatives
- Pour les formats dates et heures**
 - `{lubridate}` : manipulation des dates et heures
 - `{hms}` : manipulation des heures du jour

1.4 Qu'est-ce-qu'un `{tibble}` ?

= Une version modernisée des *dataframes*, plus pratique à utiliser que les *dataframes* "classiques"

- *dataframe* vs *tibble*

Critère	dataframe	tibble
Affichage des données	Toutes par défaut	Les premières lignes (pas de nom de lignes) ; les colonnes s'ajustent à l'écran avec des informations supplémentaires (type de colonne, dimension)
Noms des colonnes	Restrictions	Caractères spéciaux, accents ou espaces autorisés (mais pas recommandés !)
Sélection d'une colonne	La sélection d'une colonne devient un vecteur	La sélection d'une colonne reste un <i>tibble</i>

- Les *tibbles* sont plus rapides à manipuler (et plus stables) que les *dataframes*
- Possibilité de transformer un *dataframe* en *tibble* avec `tibble::as_tibble()`

Aperçu des 2 formats

Format *dataframe*

```
hdv2003 |>
  head(2)
```

```

id age  sexe                                nivetud  poids                                occup
1  1  28  Femme Enseignement superieur y compris technique superieur 2634.398 Exerce une profession
2  2  23  Femme                                <NA> 9738.396 Etudiant, eleve
  qualif freres.soeurs clso                                relig  trav.imp  trav.satisf hard.rock
1 Employe                                8 Oui Ni croyance ni appartenance Peu important Insatisfaction Non
2 <NA>                                2 Oui Ni croyance ni appartenance <NA> <NA> Non
  lecture.bd peche.chasse cuisine bricol cinema sport heures.tv
1 Non Non Oui Non Non Non 0
2 Non Non Non Non Oui Oui 1
```

Format *tibble*

```
hdv2003 |>
  as_tibble() |>
  head(2)
```

```

# A tibble: 2 x 20
  id age sexe nivetud poids occup qualif freres.soeurs clso relig trav.imp trav.satisf
<int> <int> <fct> <fct> <dbl> <fct> <fct> <int> <fct> <fct> <fct> <fct>
1 1 28 Femme Enseignement ~ 2634. Exer~ Emplo~ 8 Oui Ni c~ Peu imp~ Insatisfac~
2 2 23 Femme <NA> 9738. Etud~ <NA> 2 Oui Ni c~ <NA> <NA>
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
# bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```

Note

Pour la suite de la présentation

- on utilise le jeu de données **hdv2003** au format *tibble*

```
# Transformation au format tibble
hdv2003 |>
  as_tibble() -> hdv2003
```

- on considère les variables 'sexe' et 'clso' comme des vecteurs de caractères, et non pas des facteurs

2 Manipulations des données

2.1 Manipulations avec le package {dplyr}

Les fonctions utiles pour

- **manipuler des lignes/observations**
 - `slice()` : sélectionner des lignes suivant leur position
 - `filter()` : sélectionner/filtrer des observations
 - `distinct()` : supprimer les doublons
 - `arrange()` : trier les données par ordre croissant (ou décroissant)
- **manipuler des colonnes/variables**
 - `select()` : sélectionner/extraire des variables
 - `mutate()` : créer des variables
 - `relocate()` : changer l'ordre d'une colonne
 - `rename()` : changer le nom des variables
 - `left_join()`, `right_join()`, `inner_join()` et `full_join()` : pour combiner les tables
 - `rownames_to_column()` : ajouter le nom des lignes dans une variable
- **faire des résumés et des opérations groupées**
 - `summarise()` : faire un résumé des données
 - `count()` : compter le nombre d'observations
 - `group_by()` : regroupement de données

2.1.1 Sélectionner des lignes avec la fonction `slice()`

» On veut afficher les 10 premières et les 10 dernières lignes du jeu de données

```
hdv2003 |>
  dplyr::slice(1:10, 1991:2000)
```

A tibble: 20 x 20

	id	age	sexe	nivetud	poids	occup	qualif	freres	soeurs	clso	relig	trav.imp	trav.satisf
	<int>	<int>	<chr>	<fct>	<dbl>	<fct>	<fct>			<int>	<chr>	<fct>	<fct>
1	1	28	Femme	Enseignemen~	2634.	Exer~	Emplo~	8	Oui	Ni c~	Peu imp~	Insatisfac~	
2	2	23	Femme	<NA>	9738.	Etud~	<NA>	2	Oui	Ni c~	<NA>	<NA>	
3	3	59	Homme	Derniere an~	3994.	Exer~	Techn~	2	Non	Ni c~	Aussi i~	Equilibre	
4	4	34	Homme	Enseignemen~	5732.	Exer~	Techn~	1	Non	Appa~	Moins i~	Satisfacti~	
5	5	71	Femme	Derniere an~	4329.	Retr~	Emplo~	0	Oui	Prat~	<NA>	<NA>	
6	6	35	Femme	Enseignemen~	8675.	Exer~	Emplo~	5	Non	Ni c~	Le plus~	Equilibre	
7	7	60	Femme	Derniere an~	6166.	Au f~	Ouvri~	1	Oui	Appa~	<NA>	<NA>	
8	8	47	Homme	Enseignemen~	12892.	Exer~	Ouvri~	5	Non	Ni c~	Peu imp~	Insatisfac~	
9	9	20	Femme	<NA>	7809.	Etud~	<NA>	4	Oui	Appa~	<NA>	<NA>	
10	10	28	Homme	Enseignemen~	2277.	Exer~	Autre	2	Non	Prat~	Moins i~	Satisfacti~	
11	1991	52	Femme	Derniere an~	2801.	Exer~	Emplo~	10	Oui	Prat~	Moins i~	Equilibre	
12	1992	42	Femme	Enseignemen~	2039.	Chom~	Ouvri~	2	Oui	Prat~	<NA>	<NA>	
13	1993	50	Homme	Enseignemen~	12382.	Exer~	<NA>	3	Non	Prat~	Moins i~	Equilibre	
14	1994	41	Homme	Enseignemen~	1114.	Exer~	Profe~	7	Non	Prat~	Moins i~	Satisfacti~	
15	1995	46	Femme	Enseignemen~	8408.	Au f~	Autre	0	Non	Appa~	<NA>	<NA>	
16	1996	45	Homme	Enseignemen~	8092.	Exer~	Techn~	3	Non	Ni c~	Moins i~	Equilibre	
17	1997	46	Homme	Enseignemen~	956.	Exer~	Ouvri~	4	Oui	Prat~	Le plus~	Equilibre	
18	1998	24	Homme	Enseignemen~	5320.	Exer~	Techn~	1	Oui	Rejet	Moins i~	Satisfacti~	
19	1999	24	Femme	Enseignemen~	13741.	Exer~	Emplo~	2	Non	Appa~	Moins i~	Equilibre	

```
20 2000 66 Femme Enseignemen~ 7710. Au f~ Emplo~ 3 Non Appa~ <NA> <NA>
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
# bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```


2.1.2 Filtrer des observations avec la fonction filter()

2.1.2.1 Filtre sur un seul critère

» On filtre les individus dont l'occupation principale est "Exerce une profession"

```
hdv2003 |>
  filter(occup == "Exerce une profession")

# A tibble: 1,049 x 20
   id age sexe nivetud      poids occup qualif freres.soeurs clso relig trav.imp trav.satisf
<int> <int> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>      <fct>
1     1  28 Femme Enseignemen~ 2634. Exer~ Emplo~      8 Oui  Ni c~ Peu imp~ Insatisfac~
2     3  59 Homme Derniere an~ 3994. Exer~ Techn~      2 Non  Ni c~ Aussi i~ Equilibre
3     4  34 Homme Enseignemen~ 5732. Exer~ Techn~      1 Non  Appa~ Moins i~ Satisfacti~
4     6  35 Femme Enseignemen~ 8675. Exer~ Emplo~      5 Non  Ni c~ Le plus~ Equilibre
5     8  47 Homme Enseignemen~ 12892. Exer~ Ouvri~      5 Non  Ni c~ Peu imp~ Insatisfac~
6    10  28 Homme Enseignemen~ 2277. Exer~ Autre      2 Non  Prat~ Moins i~ Satisfacti~
7    12  47 Homme 2eme cycle  6698. Exer~ Ouvri~      4 Oui  Appa~ Moins i~ Satisfacti~
8    14  67 Femme Enseignemen~  587. Exer~ <NA>      5 Oui  Prat~ Moins i~ Satisfacti~
9    16  49 Femme Enseignemen~ 9958. Exer~ Emplo~      3 Non  Prat~ Moins i~ Equilibre
10   20  39 Femme Enseignemen~ 27196. Exer~ Ouvri~      5 Non  Appa~ Moins i~ Satisfacti~
# i 1,039 more rows
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
# bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```

 Mise en garde

R est sensible à la casse !

Bien faire attention aux majuscules/minuscules

2.1.2.2 Filtre sur plusieurs critères

Se fait grâce aux opérateurs & (ET) et | (OU)

» On filtre les femmes dont l'occupation principale est "Exerce une profession"

```
hdv2003 |>
  filter(occup == "Exerce une profession" & sexe == "Femme")

# A tibble: 529 x 20
   id age sexe nivetud      poids occup qualif freres.soeurs clso relig trav.imp trav.satisf
<int> <int> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>      <fct>
1     1  28 Femme Enseignemen~ 2634. Exer~ Emplo~      8 Oui  Ni c~ Peu imp~ Insatisfac~
2     6  35 Femme Enseignemen~ 8675. Exer~ Emplo~      5 Non  Ni c~ Le plus~ Equilibre
3    14  67 Femme Enseignemen~  587. Exer~ <NA>      5 Oui  Prat~ Moins i~ Satisfacti~
4    16  49 Femme Enseignemen~ 9958. Exer~ Emplo~      3 Non  Prat~ Moins i~ Equilibre
5    20  39 Femme Enseignemen~ 27196. Exer~ Ouvri~      5 Non  Appa~ Moins i~ Satisfacti~
6    21  30 Femme Enseignemen~ 14648. Exer~ Emplo~      6 Oui  Appa~ Moins i~ Satisfacti~
7    23  37 Femme Enseignemen~ 1282. Exer~ Emplo~      4 Non  Appa~ Moins i~ Satisfacti~
8    25  20 Femme <NA>      8780. Exer~ Ouvri~      0 Non  Appa~ Moins i~ Equilibre
9    27  31 Femme Enseignemen~ 6663. Exer~ Ouvri~      1 Non  Appa~ Moins i~ Equilibre
10   36  41 Femme 1er cycle  7872. Exer~ Emplo~      3 Oui  Rejet Moins i~ Insatisfac~
```

```
# i 519 more rows
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
#   bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```

» On souhaite filtrer les personnes avec la qualification professionnelle "Ouvrier specialise" ainsi que "Ouvrier qualifie"

2.1.2.3 | Avec l'opérateur |

```
hdv2003 |>
  filter(qualif == "Ouvrier specialise" | qualif == "Ouvrier qualifie")

# A tibble: 495 x 20
   id  age sexe nivetud      poids occup qualif freres.soeurs clso  relig trav.imp trav.satisf
   <int> <int> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>      <fct>
1     7    60 Femme Derniere an~ 6166. Au f~ Ouvri~      1 Oui  Appa~ <NA>      <NA>
2     8    47 Homme Enseignemen~ 12892. Exer~ Ouvri~      5 Non  Ni c~ Peu imp~ Insatisfac~
3    12    47 Homme 2eme cycle   6698. Exer~ Ouvri~      4 Oui  Appa~ Moins i~ Satisfacti~
4    19    70 Homme Derniere an~ 3141. Retr~ Ouvri~      2 Non  Appa~ <NA>      <NA>
5    20    39 Femme Enseignemen~ 27196. Exer~ Ouvri~      5 Non  Appa~ Moins i~ Satisfacti~
6    25    20 Femme <NA>         8780. Exer~ Ouvri~      0 Non  Appa~ Moins i~ Equilibre
7    27    31 Femme Enseignemen~ 6663. Exer~ Ouvri~      1 Non  Appa~ Moins i~ Equilibre
8    28    35 Homme 1er cycle   3359. Exer~ Ouvri~      6 Non  Ni c~ Moins i~ Equilibre
9    29    35 Homme Enseignemen~ 8536. Exer~ Ouvri~      2 Non  Appa~ Moins i~ Equilibre
10   30    30 Homme Enseignemen~ 10621. Exer~ Ouvri~      1 Oui  Ni c~ Aussi i~ Satisfacti~

# i 485 more rows
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
#   bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```

2.1.2.4 | Avec l'opérateur %in%

```
hdv2003 |>
  filter(qualif %in% c("Ouvrier specialise", "Ouvrier qualifie"))

# A tibble: 495 x 20
   id  age sexe nivetud      poids occup qualif freres.soeurs clso  relig trav.imp trav.satisf
   <int> <int> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>      <fct>
1     7    60 Femme Derniere an~ 6166. Au f~ Ouvri~      1 Oui  Appa~ <NA>      <NA>
2     8    47 Homme Enseignemen~ 12892. Exer~ Ouvri~      5 Non  Ni c~ Peu imp~ Insatisfac~
3    12    47 Homme 2eme cycle   6698. Exer~ Ouvri~      4 Oui  Appa~ Moins i~ Satisfacti~
4    19    70 Homme Derniere an~ 3141. Retr~ Ouvri~      2 Non  Appa~ <NA>      <NA>
5    20    39 Femme Enseignemen~ 27196. Exer~ Ouvri~      5 Non  Appa~ Moins i~ Satisfacti~
6    25    20 Femme <NA>         8780. Exer~ Ouvri~      0 Non  Appa~ Moins i~ Equilibre
7    27    31 Femme Enseignemen~ 6663. Exer~ Ouvri~      1 Non  Appa~ Moins i~ Equilibre
8    28    35 Homme 1er cycle   3359. Exer~ Ouvri~      6 Non  Ni c~ Moins i~ Equilibre
9    29    35 Homme Enseignemen~ 8536. Exer~ Ouvri~      2 Non  Appa~ Moins i~ Equilibre
10   30    30 Homme Enseignemen~ 10621. Exer~ Ouvri~      1 Oui  Ni c~ Aussi i~ Satisfacti~

# i 485 more rows
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
#   bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```

2.1.2.5 | Avec la fonction str_detect() du package {stringr}

```
hdv2003 |>
  filter(str_detect(qualif, "Ouvrier"))
```



```
# A tibble: 495 x 20
  id   age sexe   nivetud      poids occup qualif freres.soeurs clso  relig trav.imp trav.satisf
  <int> <int> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>      <fct>
1     7    60 Femme Derniere an~ 6166. Au f~ Ouvri~      1 Oui  Appa~ <NA>      <NA>
2     8    47 Homme Enseignemen~ 12892. Exer~ Ouvri~      5 Non  Ni c~ Peu imp~ Insatisfac~
3    12    47 Homme 2eme cycle  6698. Exer~ Ouvri~      4 Oui  Appa~ Moins i~ Satisfacti~
4    19    70 Homme Derniere an~ 3141. Retr~ Ouvri~      2 Non  Appa~ <NA>      <NA>
5    20    39 Femme Enseignemen~ 27196. Exer~ Ouvri~      5 Non  Appa~ Moins i~ Satisfacti~
6    25    20 Femme <NA>      8780. Exer~ Ouvri~      0 Non  Appa~ Moins i~ Equilibre
7    27    31 Femme Enseignemen~ 6663. Exer~ Ouvri~      1 Non  Appa~ Moins i~ Equilibre
8    28    35 Homme 1er cycle  3359. Exer~ Ouvri~      6 Non  Ni c~ Moins i~ Equilibre
9    29    35 Homme Enseignemen~ 8536. Exer~ Ouvri~      2 Non  Appa~ Moins i~ Equilibre
10   30    30 Homme Enseignemen~ 10621. Exer~ Ouvri~      1 Oui  Ni c~ Aussi i~ Satisfacti~
# i 485 more rows
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
#   bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```

i Opérateurs logiques les plus courants

Opérateur	Description
<	Inférieur à
<=	Inférieur ou égal à
>	Supérieur à
>=	Supérieur ou égal à
==	Exactement égal à
!=	Différent de
	Ou
&	Et
%in%	Appartient à
is.na()	Est une donnée manquante
!is.na()	N'est pas une donnée manquante

2.1.3 Supprimer les doublons avec la fonction distinct()

» On supprime les doublons à partir de la variable 'sexe'

```
hdv2003 |>
  distinct(sexe)
```

```
# A tibble: 2 x 1
  sexe
  <chr>
1 Femme
2 Homme
```

Il est possible de supprimer les doublons à partir de plusieurs critères

» On supprime les doublons à partir de la variable 'sexe' et 'relig'

```
hdv2003 |>
  distinct(sexe, relig)
```

```
# A tibble: 12 x 2
  sexe  relig
```

```

<chr> <fct>
1 Femme Ni croyance ni appartenance
2 Homme Ni croyance ni appartenance
3 Homme Appartenance sans pratique
4 Femme Pratiquant regulier
5 Femme Appartenance sans pratique
6 Homme Pratiquant occasionnel
7 Femme Pratiquant occasionnel
8 Homme Pratiquant regulier
9 Femme Rejet
10 Homme NSP ou NVPR
11 Homme Rejet
12 Femme NSP ou NVPR

```

2.1.4 Trier les données avec la fonction arrange()

2.1.4.1 Tri sur un seul critère

» On trie les données par ordre croissant de 'age'

```

hdv2003 |>
  arrange(age)

```

A tibble: 2,000 x 20

	id	age	sexe	nivetud	poids	occup	qualif	freres.soeurs	clso	relig	trav.imp	trav.satisf
	<int>	<int>	<chr>	<fct>	<dbl>	<fct>	<fct>		<int>	<chr>	<fct>	<fct>
1	162	18	Homme	<NA>	4983.	Etud~	<NA>		2	Non	Appa~	<NA>
2	215	18	Homme	<NA>	4631.	Etud~	<NA>		2	Oui	Ni c~	<NA>
3	346	18	Femme	<NA>	1725.	Etud~	<NA>		9	Non	Prat~	<NA>
4	377	18	Homme	<NA>	11213.	Etud~	<NA>		1	Oui	Rejet	<NA>
5	511	18	Homme	<NA>	2448.	Etud~	<NA>		6	Non	Prat~	<NA>
...												

i La fonction arrange() trie les valeurs par ordre croissant par défaut, mais on peut trier par ordre décroissant grâce à la fonction desc()

```

hdv2003 |>
  arrange(desc(age))

```

A tibble: 2,000 x 20

	id	age	sexe	nivetud	poids	occup	qualif	freres.soeurs	clso	relig	trav.imp	trav.satisf
	<int>	<int>	<chr>	<fct>	<dbl>	<fct>	<fct>		<int>	<chr>	<fct>	<fct>
1	1916	97	Femme	Derniere an~	2163.	Autr~	Autre		5	Non	Prat~	<NA>
2	270	96	Femme	Derniere an~	9993.	Retr~	<NA>		1	Oui	Ni c~	<NA>
3	1542	93	Femme	Derniere an~	7108.	Reti~	<NA>		7	Non	Prat~	<NA>
4	235	92	Homme	N'a jamais ~	827.	Retr~	Ouvri~		0	Ne s~	Prat~	<NA>
5	1151	91	Homme	Derniere an~	4291.	Reti~	<NA>		3	Oui	Appa~	<NA>
6	96	90	Femme	Derniere an~	5875.	Au f~	Emplo~		7	Non	Prat~	<NA>
7	1122	90	Femme	1er cycle	14812.	Retr~	Emplo~		3	Oui	Prat~	<NA>
8	1767	90	Femme	Derniere an~	12266.	Reti~	<NA>		2	Non	Appa~	<NA>
9	759	89	Femme	Derniere an~	9555.	Au f~	<NA>		1	Non	Prat~	<NA>
10	971	89	Homme	Derniere an~	2374.	Retr~	Ouvri~		5	Oui	Prat~	<NA>
# i	1,990 more rows											
# i	8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,											
#	bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>											

2.1.4.2 Tri sur plusieurs critères

» On trie les données par ordre alphabétique croissant pour la variable 'sexe' et par valeurs décroissantes pour la variable 'age'

```
hdv2003 |>
  arrange(sexe, desc(age))

# A tibble: 2,000 x 20
   id   age sexe nivetud      poids occup qualif freres.soeurs clso  relig trav.imp trav.satisf
   <int> <int> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>      <fct>
1  1916   97 Femme Derniere an~ 2163. Autr~ Autre          5 Non  Prat~ <NA>      <NA>
2   270   96 Femme Derniere an~ 9993. Retr~ <NA>          1 Oui  Ni c~ <NA>      <NA>
3  1542   93 Femme Derniere an~ 7108. Reti~ <NA>          7 Non  Prat~ <NA>      <NA>
4    96   90 Femme Derniere an~ 5875. Au f~ Emplo~    7 Non  Prat~ <NA>      <NA>
5  1122   90 Femme 1er cycle 14812. Retr~ Emplo~    3 Oui  Prat~ <NA>      <NA>
6  1767   90 Femme Derniere an~ 12266. Reti~ <NA>          2 Non  Appa~ <NA>      <NA>
7   759   89 Femme Derniere an~ 9555. Au f~ <NA>          1 Non  Prat~ <NA>      <NA>
8  1117   89 Femme A arrete se~ 5677. Au f~ Ouvri~    5 Oui  Appa~ <NA>      <NA>
9  1127   89 Femme A arrete se~ 25986. Autr~ <NA>          2 Non  Appa~ <NA>      <NA>
10 1516   88 Femme Derniere an~ 15686. Reti~ <NA>          2 Non  Prat~ <NA>      <NA>
# i 1,990 more rows
# i 8 more variables: hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>,
#   bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>
```

2.1.5 Sélectionner des colonnes avec select()

2.1.5.1 Sélection d'une seule variable

2 possibilités, avec la fonction select() ou avec la fonction pull()

» On sélectionne la variable 'sexe'

```
hdv2003 |>
  select(sexe)

# A tibble: 2,000 x 1
  sexe
  <chr>
1 Femme
2 Femme
3 Homme
4 Homme
5 Femme
6 Femme
7 Femme
8 Homme
9 Femme
10 Homme
# i 1,990 more rows

hdv2003 |>
  pull(sexe)

[1] "Femme" "Femme" "Homme" "Homme" "Femme" "Femme" "Femme" "Homme" "Femme" "Homme" "Femme"
[12] "Homme" "Femme" "Femme" "Femme" "Femme" "Homme" "Femme" "Homme" "Femme" "Femme" "Homme"
[23] "Femme" "Femme" "Femme" "Homme" "Femme" "Homme" "Homme" "Homme" "Homme" "Homme" "Homme"
[34] "Homme" "Femme" "Femme" "Homme" "Femme" "Femme" "Homme" "Femme" "Homme" "Homme" "Femme"
```

```
[45] "Femme" "Homme" "Femme" "Femme" "Femme" "Femme" "Femme" "Homme" "Femme" "Homme" "Femme" "Homme"
[56] "Femme" "Femme" "Femme" "Homme" "Femme" "Femme" "Femme" "Homme" "Homme" "Homme" "Homme" "Femme"
[67] "Homme" "Homme" "Femme" "Femme" "Homme" "Homme" "Femme" "Femme" "Femme" "Femme" "Femme" "Homme"
[78] "Femme" "Femme" "Femme" "Femme" "Femme" "Femme" "Femme" "Homme" "Homme" "Femme" "Homme" "Homme"
[89] "Homme" "Homme" "Homme" "Femme" "Homme" "Femme" "Femme" "Femme" "Femme" "Homme" "Homme" "Femme"
[100] "Femme" "Femme" "Homme" "Femme" "Homme" "Homme" "Femme" "Femme" "Femme" "Femme" "Femme" "Homme"
[111] "Homme" "Homme" "Homme" "Homme" "Femme" "Homme" "Homme" "Femme" "Homme" "Homme" "Homme" "Femme"
[122] "Femme" "Femme" "Homme" "Femme" "Femme" "Homme" "Femme" "Femme" "Homme" "Femme" "Femme" "Homme"
[133] "Femme" "Femme" "Femme" "Homme" "Homme" "Homme" "Homme" "Homme" "Homme" "Homme" "Homme" "Homme"
...
```

💡 Quelle différence entre `pull()` et `select()` ?

- La fonction `pull()` renvoie une seule colonne sous forme de vecteur.
- La fonction `select()` renvoie une ou plusieurs colonnes sous forme de *dataframe*.

2.1.5.2 Sélectionner plusieurs variables avec `select()`

» On sélectionne UNIQUEMENT les variables 'sexe' et 'age'

```
hdv2003 |>
  select(sexe, age)
```

A tibble: 2,000 x 2

```
  sexe    age
  <chr> <int>
1 Femme    28
2 Femme    23
3 Homme    59
4 Homme    34
5 Femme    71
6 Femme    35
7 Femme    60
8 Homme    47
9 Femme    20
10 Homme    28
```

i 1,990 more rows

» On sélectionne toutes les variables SAUF les variables 'sexe' et 'age'

```
hdv2003 |>
  select(-c(sexe, age))
```

A tibble: 2,000 x 18

```
  id nivetud      poids occup qualif freres.soeurs clso  relig trav.imp trav.satisf hard.rock
  <int> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>    <fct>    <fct>
1     1 Enseignement ~ 2634. Exer~ Emplo~      8 Oui  Ni c~ Peu imp~ Insatisfac~ Non
2     2 <NA>          9738. Etud~ <NA>      2 Oui  Ni c~ <NA>    <NA>    Non
3     3 Derniere anne~ 3994. Exer~ Techn~      2 Non  Ni c~ Aussi i~ Equilibre Non
4     4 Enseignement ~ 5732. Exer~ Techn~      1 Non  Appa~ Moins i~ Satisfacti~ Non
5     5 Derniere anne~ 4329. Retr~ Emplo~      0 Oui  Prat~ <NA>    <NA>    Non
6     6 Enseignement ~ 8675. Exer~ Emplo~      5 Non  Ni c~ Le plus~ Equilibre Non
7     7 Derniere anne~ 6166. Au f~ Ouvri~      1 Oui  Appa~ <NA>    <NA>    Non
8     8 Enseignement ~ 12892. Exer~ Ouvri~      5 Non  Ni c~ Peu imp~ Insatisfac~ Non
9     9 <NA>          7809. Etud~ <NA>      4 Oui  Appa~ <NA>    <NA>    Non
10    10 Enseignement ~ 2277. Exer~ Autre      2 Non  Prat~ Moins i~ Satisfacti~ Non
```

i 1,990 more rows

```
# i 7 more variables: lecture.bd <fct>, peche.chasse <fct>, cuisine <fct>, bricol <fct>,  
#   cinema <fct>, sport <fct>, heures.tv <dbl>
```

» On sélectionne toutes les variables dont le nom contient "trav"

```
hdv2003 |>  
  select(contains("trav"))
```

```
# A tibble: 2,000 x 2  
  trav.imp      trav.satisf  
  <fct>         <fct>  
1 Peu important   Insatisfaction  
2 <NA>            <NA>  
3 Aussi important que le reste Equilibre  
4 Moins important que le reste Satisfaction  
5 <NA>            <NA>  
6 Le plus important Equilibre  
7 <NA>            <NA>  
8 Peu important   Insatisfaction  
9 <NA>            <NA>  
10 Moins important que le reste Satisfaction  
# i 1,990 more rows
```



Les tests sur le nom des variables peuvent se faire avec différentes fonctions :

- contains()
- starts_with()
- ends_with()
- matches()

Ce ne sont pas les seules fonctions pour sélectionner des variables, il existe bien d'autres façons, qui sont détaillées dans la page d'aide de la fonction `select()` du package `{dplyr}`.

2.1.6 Créer/ajouter de nouvelles variables avec `mutate()`

» On ajoute dans le jeu de données `hdv2003`, la variable 'an_nais'

```
hdv2003 |>  
  mutate(an_nais = 2003 - age) -> hdv2003
```



Penser à affecter le résultat à un dataframe/tibble !

»> Vérification

```
hdv2003 |>  
  names()
```

```
[1] "id"      "age"      "sexe"      "nivetud"    "poids"    "occup"  
[7] "qualif"  "freres.soeurs" "clso"      "relig"      "trav.imp"  "trav.satisf"  
[13] "hard.rock" "lecture.bd" "peche.chasse" "cuisine"    "bricol"    "cinema"  
[19] "sport"    "heures.tv" "an_nais"
```

- La fonction `mutate()` peut également
 - modifier une variable existante si le nom est le même que celui d'une colonne existante
 - supprimer une variable en fixant sa valeur à `NULL`

```
hdv2003 |>
  mutate(age = NULL) |>
  names()
```

```
[1] "id"          "sexe"          "nivetud"        "poids"          "occup"          "qualif"
[7] "freres.soeurs" "clso"          "relig"          "trav.imp"        "trav.satisf"    "hard.rock"
[13] "lecture.bd"    "peche.chasse"  "cuisine"        "bricol"          "cinema"          "sport"
[19] "heures.tv"     "an_nais"
```

💡 Le package `{fastDummies}` est très utile pour créer rapidement des colonnes *dummy* à partir de variables catégorielles, avec la fonction `dummy_cols()`

```
hdv2003 |>
  select(sexe) |>
  fastDummies::dummy_cols()
```

```
# A tibble: 2,000 x 3
  sexe  sexe_Femme sexe_Homme
<chr>    <int>    <int>
1 Femme         1         0
2 Femme         1         0
3 Homme         0         1
4 Homme         0         1
...
```

Le nom des *dummies* créées automatiquement sont de la forme *variable_modalite*

2.1.6.1 Effectuer des transformations sur plusieurs variables, en utilisant la fonction `across()`

» On souhaite transformer tous les facteurs en variables "character"

```
hdv2003 |>
  mutate(across(where(is.factor), as.character))
```

»> Vérification

```
# Variables initiales
hdv2003 |>
  str()
```

```
tibble [2,000 x 21] (S3: tbl_df/tbl/data.frame)
 $ id          : int [1:2000] 1 2 3 4 5 6 7 8 9 10 ...
 $ age         : int [1:2000] 28 23 59 34 71 35 60 47 20 28 ...
 $ sexe        : chr [1:2000] "Femme" "Femme" "Homme" "Homme" ...
 $ nivetud     : Factor w/ 8 levels "N'a jamais fait d'etudes",...: 8 NA 3 8 3 6 3 6 NA 7 ...
 $ poids       : num [1:2000] 2634 9738 3994 5732 4329 ...
 $ occup       : Factor w/ 7 levels "Exerce une profession",...: 1 3 1 1 4 1 6 1 3 1 ...
 $ qualif      : Factor w/ 7 levels "Ouvrier specialise",...: 6 NA 3 3 6 6 2 2 NA 7 ...
 $ freres.soeurs: int [1:2000] 8 2 2 1 0 5 1 5 4 2 ...
 $ clso        : chr [1:2000] "Oui" "Oui" "Non" "Non" ...
 $ relig       : Factor w/ 6 levels "Pratiquant regulier",...: 4 4 4 3 1 4 3 4 3 2 ...
 $ trav.imp    : Factor w/ 4 levels "Le plus important",...: 4 NA 2 3 NA 1 NA 4 NA 3 ...
 $ trav.satisf : Factor w/ 3 levels "Satisfaction",...: 2 NA 3 1 NA 3 NA 2 NA 1 ...
 $ hard.rock   : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 $ lecture.bd  : Factor w/ 2 levels "Non","Oui": 1 1 1 1 1 1 1 1 1 1 ...
 ...
```

```
# Variables après modification
hdv2003 |>
  str()
```

```
tibble [2,000 x 21] (S3: tbl_df/tbl/data.frame)
 $ id          : int [1:2000] 1 2 3 4 5 6 7 8 9 10 ...
 $ age         : int [1:2000] 28 23 59 34 71 35 60 47 20 28 ...
 $ sexe        : chr [1:2000] "Femme" "Femme" "Homme" "Homme" ...
 $ nivatud     : chr [1:2000] "Enseignement superieur y compris technique superieur" NA "Derniere annee d'etudes pr
 $ poids       : num [1:2000] 2634 9738 3994 5732 4329 ...
 $ occup       : chr [1:2000] "Exerce une profession" "Etudiant, eleve" "Exerce une profession" "Exerce une professi
 $ qualif      : chr [1:2000] "Employe" NA "Technicien" "Technicien" ...
 $ freres.soeurs: int [1:2000] 8 2 2 1 0 5 1 5 4 2 ...
 $ clso        : chr [1:2000] "Oui" "Oui" "Non" "Non" ...
 $ relig       : chr [1:2000] "Ni croyance ni appartenance" "Ni croyance ni appartenance" "Ni croyance ni appartenan
 $ trav.imp     : chr [1:2000] "Peu important" NA "Aussi important que le reste" "Moins important que le reste" ...
 $ trav.satisf : chr [1:2000] "Insatisfaction" NA "Equilibre" "Satisfaction" ...
 $ hard.rock    : chr [1:2000] "Non" "Non" "Non" "Non" ...
 $ lecture.bd   : chr [1:2000] "Non" "Non" "Non" "Non" ...
 $ peche.chasse : chr [1:2000] "Non" "Non" "Non" "Non" ...
 $ cuisine      : chr [1:2000] "Oui" "Non" "Non" "Oui" ...
 $ bricol       : chr [1:2000] "Non" "Non" "Non" "Oui" ...
 $ cinema       : chr [1:2000] "Non" "Oui" "Non" "Oui" ...
 $ sport        : chr [1:2000] "Non" "Oui" "Oui" "Oui" ...
 $ heures.tv    : num [1:2000] 0 1 0 2 3 2 2.9 1 2 2 ...
 $ an_nais      : num [1:2000] 1975 1980 1944 1969 1932 ...
```

2.1.6.2 Effectuer des transformations en utilisant des conditions

» On ajoute dans le jeu de données **hdv2003**, la variable variable 'cl_age'

2.1.6.3 | Avec la fonction `if_else()` |

```
hdv2003 |>
  mutate(cl_age = if_else(age >= 15 & age <= 29, "15 à 29 ans",
                          if_else(age >= 30 & age <= 44, "30 à 44 ans",
                                  if_else(age >= 45 & age <= 59, "45 à 59 ans",
                                          if_else(age >= 60 & age <= 75, "60 à 74 ans",
                                                  "75 ans et plus"))))) -> hdv2003
```

»> Vérification

```
# Distribution de la variable cl_age
hdv2003 |>
  select(cl_age) |>
  questionr::freq(total = TRUE)
```

	n	%	val%
15 à 29 ans	302	15.1	15.1
30 à 44 ans	573	28.6	28.6
45 à 59 ans	609	30.4	30.4
60 à 74 ans	384	19.2	19.2
75 ans et plus	132	6.6	6.6
Total	2000	100.0	100.0

2.1.6.4 | Avec la fonction `case_when()` |

```
hdv2003 |>
  mutate(cl_age = case_when(age >= 15 & age <= 29 ~ "15 à 29 ans",
                             age >= 30 & age <= 44 ~ "30 à 44 ans",
                             age >= 45 & age <= 59 ~ "45 à 59 ans",
                             age >= 60 & age <= 75 ~ "60 à 74 ans",
                             age >= 75 ~ "75 ans et plus",
                             TRUE ~ "ERREUR")) -> hdv2003
```

»> Vérification

```
# Distribution de la variable cl_age
hdv2003 |>
  select(cl_age) |>
  questionr::freq(total = TRUE)
```

	n	%	val%
15 à 29 ans	302	15.1	15.1
30 à 44 ans	573	28.6	28.6
45 à 59 ans	609	30.4	30.4
60 à 74 ans	384	19.2	19.2
75 ans et plus	132	6.6	6.6
Total	2000	100.0	100.0

2.1.7 Réordonner les variables avec la fonction relocate()

Les nouvelles variables ajoutées avec `mutate()` se positionnent toujours à la fin du jeu de données

Il est possible de les positionner où l'on souhaite, avec la fonction `relocate()`

» On positionne la variable 'an_nais' juste après la variable 'age' dans le jeu de données **hdv2003**

```
hdv2003 |>
  relocate(an_nais, .after = age) -> hdv2003
```

»> Vérification

```
# Variables après modification
hdv2003 |>
  names()
```

[1] "id"	"age"	"an_nais"	"sexe"	"nivetud"	"poids"
[7] "occup"	"qualif"	"freres.soeurs"	"clso"	"relig"	"trav.imp"
[13] "trav.satisf"	"hard.rock"	"lecture.bd"	"peche.chasse"	"cuisine"	"bricol"
[19] "cinema"	"sport"	"heures.tv"	"cl_age"		

On aurait également pu obtenir le même résultat en utilisant `.before` dans la fonction

```
hdv2003 |>
  relocate(an_nais, .before = sexe) -> hdv2003
```

i La fonction `relocate()` s'utilise soit avec les paramètres `.before =` ou `.after =` et dans ces paramètres, on peut utiliser les fonctions `last_col()` (= dernière colonne) ou `everything()` (= toutes les variables)

2.1.8 Modifier le nom d'une variable avec `rename()`

» On renomme la variable 'an_nais' en 'an.naissance' dans **hdv2003**

```
hdv2003 |>
  rename(an.naissance = an_nais) -> hdv2003
```

»> Vérification

```
# Variables initiales
hdv2003 |>
  names()
```

```
[1] "id"          "age"          "an_nais"      "sexe"         "nivetud"      "poids"
[7] "occup"       "qualif"       "freres.soeurs" "clso"         "relig"        "trav.imp"
[13] "trav.satisf" "hard.rock"    "lecture.bd"   "peche.chasse" "cuisine"      "bricol"
[19] "cinema"      "sport"        "heures.tv"    "cl_age"
```

```
# Variables après modification
hdv2003 |>
  names()
```

```
[1] "id"          "age"          "an.naissance" "sexe"         "nivetud"      "poids"
[7] "occup"       "qualif"       "freres.soeurs" "clso"         "relig"        "trav.imp"
[13] "trav.satisf" "hard.rock"    "lecture.bd"   "peche.chasse" "cuisine"      "bricol"
[19] "cinema"      "sport"        "heures.tv"    "cl_age"
```

- 💡 Le package `{janitor}` (qui suit les principes du tidyverse et est parfaitement compatible avec le pipe), est utile pour nettoyer
- le nom des colonnes, avec la fonction `clean_names()`
 - convertit tous les noms en minuscules
 - remplace les espaces et les caractères spéciaux par des `_`
 - supprime les caractères non alphanumériques
 - assure la cohérence du format des noms
 - les vecteurs (et donc le contenu des données), avec la fonction `make_clean_names()`, plus générale, qui permet de
 - choisir le format de casse (`snake_case`, `camelCase`, etc.)
 - remplacer certains caractères spécifiques

2.1.9 Combiner des jeux de données

Pour combiner les données de 2 (ou plusieurs) jeux de données/tables, on effectue une **jointure**

- i** Il existe différentes jointures :
- **jointure à gauche** (*left join*) : conserve toutes les observations de la table de gauche même si elles ne sont pas présentes dans la table de droite
 - **jointure à droite** (*right join*) : conserve toutes les observations de la table de droite même si elles ne sont pas présentes dans la table de gauche
 - **jointure interne** (*inner join*) : conserve les observations qui sont présentes dans les 2 tables
 - **jointure externe** (*full join*) : conserve toutes les observations, présentes dans l'une ou l'autre des tables

! Pour pouvoir faire une jointure, il faut une **clé**, c'est à dire une variable commune aux 2 jeux de données, qui peut avoir ou non, le même nom

» On suppose que l'on souhaite regrouper les données de deux jeux de données, nommés **data1** et **data2**

```
# Données data1
data1 <- tibble(id = c(1, 2, 3, 4, 5),
                sexe = c('F', 'H', 'H', 'F', 'F'),
                age = c(25, 57, 88, 32, 44))

print(data1)
```

```
# A tibble: 5 x 3
  id sexe  age
<dbl> <chr> <dbl>
1     1 F     25
2     2 H     57
3     3 H     88
4     4 F     32
5     5 F     44
```

```
# Données data2
data2 <- tibble(ident = c(1, 3, 4, 6),
                 localisation = c("Bordeaux/33", "Cestas/33", "Bordeaux/33", "Dax/40"),
                 code = c("063", "122", "063", "088"),
                 revenu = c(2500, 3000, 3700, 1850))

print(data2)
```

```
# A tibble: 4 x 4
  ident localisation code  revenu
<dbl> <chr>          <chr> <dbl>
1     1 Bordeaux/33 063    2500
2     3 Cestas/33   122    3000
3     4 Bordeaux/33 063    3700
4     6 Dax/40      088    1850
```

2.1.9.1 Jointure à gauche

Cette jointure va ajouter à la suite des colonnes de **data1**, les colonnes de **data2**, en ne conservant que les lignes de **data1**

```
data1 |>
  left_join(data2, by = c("id" = "ident")) -> data_leftjoin
```

»> Vérification

```
data_leftjoin
```

```
# A tibble: 5 x 6
  id sexe  age localisation code  revenu
<dbl> <chr> <dbl> <chr>          <chr> <dbl>
1     1 F     25 Bordeaux/33 063    2500
2     2 H     57 <NA>          <NA>     NA
3     3 H     88 Cestas/33   122    3000
4     4 F     32 Bordeaux/33 063    3700
5     5 F     44 <NA>          <NA>     NA
```



- Les observations du jeu de données de gauche (**data1**) qui n'existent pas dans le jeu de données de droite (**data2**) sont conservées
- Pour ces observations, on a des données manquantes dans les colonnes de **data2**



Dans le cas où la clé de jointure a le même nom dans les deux tables (par exemple "cle"), la fonction s'écrit comme suit

```
data1 |>
  left_join(data2, by = "cle") -> data_leftjoin
```

2.1.9.2 Jointure à droite

Cette jointure va ajouter à la suite des colonnes de **data1**, les colonnes de **data2**, en ne conservant que les lignes de **data2**

```
data1 |>
  right_join(data2, by = c("id" = "ident")) -> data_rightjoin
```

»> Vérification

```
data_rightjoin
```

```
# A tibble: 4 x 6
   id sexe    age localisation code  revenu
<dbl> <chr> <dbl> <chr>         <chr> <dbl>
1     1 F      25 Bordeaux/33 063    2500
2     3 H      88 Cestas/33   122    3000
3     4 F      32 Bordeaux/33 063    3700
4     6 <NA>    NA Dax/40     088    1850
```



- Les observations du jeu de données de droite (**data2**) qui n'existent pas dans le jeu de données de gauche (**data1**) sont conservées
- Pour ces observations, on a des données manquantes dans les colonnes de **data1**

2.1.9.3 Jointure interne

Cette jointure va ajouter à la suite des colonnes de **data1**, les colonnes de **data2**, en ne conservant que les lignes communes à **data1** et **data2**

```
data1 |>
  inner_join(data2, by = c("id" = "ident")) -> data_innerjoin
```

»> Vérification

```
data_innerjoin
```

```
# A tibble: 3 x 6
   id sexe    age localisation code  revenu
<dbl> <chr> <dbl> <chr>         <chr> <dbl>
1     1 F      25 Bordeaux/33 063    2500
```

2	3	H	88	Cestas/33	122	3000
3	4	F	32	Bordeaux/33	063	3700

- Seules les observations communes aux deux jeux de données **data1** et **data2** sont conservées
- Il n'y a donc aucune valeur manquante dans le résultat de la jointure

2.1.9.4 Jointure externe

Cette jointure va ajouter à la suite des colonnes de **data1**, les colonnes de **data2**, en conservant toutes les lignes de **data1** et toutes les lignes de **data2**

```
data1 |>
  full_join(data2, by = c("id" = "ident")) -> data_fulljoin
```

»> Vérification

```
data_fulljoin
```

```
# A tibble: 6 x 6
  id sexe    age localisation code  revenu
<dbl> <chr> <dbl> <chr>      <chr> <dbl>
1     1 F      25 Bordeaux/33 063    2500
2     2 H      57 <NA>         <NA>    NA
3     3 H      88 Cestas/33   122    3000
4     4 F      32 Bordeaux/33 063    3700
5     5 F      44 <NA>         <NA>    NA
6     6 <NA>    NA Dax/40      088    1850
```

- Toutes les observations sont conservées, à la fois celles de **data1** mais aussi celles de **data2**
- Il y a donc des valeurs manquantes dans
 - les colonnes de **data1** pour les observations de **data2** qui n'existent pas dans **data1**
 - les colonnes de **data2** pour les observations de **data1** qui n'existent pas dans **data2**

2.1.10 Ajouter une colonne d'identifiants avec `rownames_to_column()`

Pour les cas où il n'y a pas d'identifiant dans le jeu de données ou que l'identifiant n'est pas dans une colonne mais dans *rownames*, on utilise la fonction `rownames_to_column()` qui va ajouter les noms de lignes dans une colonne, qui contiendra ainsi un identifiant unique pour chaque observation

```
hdv2003 |>
  rownames_to_column(var = "identifiant")
```

```
# A tibble: 2,000 x 23
  identifiant id age an.naissance sexe nivetud poids occup qualif freres.soeurs clso relig
  <chr>      <int> <int>      <dbl> <chr> <fct> <dbl> <fct> <fct>      <int> <chr> <fct>
1 1          1 28 1975 Femme Enseign~ 2634. Exer~ Emplo~ 8 Oui Ni c~
2 2          2 23 1980 Femme <NA> 9738. Etud~ <NA> 2 Oui Ni c~
3 3          3 59 1944 Homme Dernier~ 3994. Exer~ Techn~ 2 Non Ni c~
4 4          4 34 1969 Homme Enseign~ 5732. Exer~ Techn~ 1 Non Appa~
5 5          5 71 1932 Femme Dernier~ 4329. Retr~ Emplo~ 0 Oui Prati~
6 6          6 35 1968 Femme Enseign~ 8675. Exer~ Emplo~ 5 Non Ni c~
7 7          7 60 1943 Femme Dernier~ 6166. Au f~ Ouvri~ 1 Oui Appa~
```

```

8 8          8  47          1956 Homme Enseign~ 12892. Exer~ Ouvri~          5 Non  Ni c~
9 9          9  20          1983 Femme <NA>      7809. Etud~ <NA>          4 Oui  Appa~
10 10        10  28          1975 Homme Enseign~ 2277. Exer~ Autre          2 Non  Prat~
# i 1,990 more rows
# i 11 more variables: trav.imp <fct>, trav.satisf <fct>, hard.rock <fct>, lecture.bd <fct>,
#   peche.chasse <fct>, cuisine <fct>, bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>,
#   cl_age <chr>

```

2.1.11 Calculer des statistiques récapitulatives

2.1.11.1 avec la fonction `count()`

Compte le nombre d'occurrences des modalités de(s) variable(s) spécifiée(s) = tableau de fréquences qui contient uniquement les effectifs

Peut être utile en amont d'un `geom_bar()`

2.1.11.1.1 Pour une variable

» On souhaite compter le nombre d'occurrences pour la variable 'clso'

```

hdv2003 |>
  count(clso)

# A tibble: 3 x 2
  clso          n
  <chr>      <int>
1 Ne sait pas    27
2 Non           1037
3 Oui             936

```

2.1.11.1.2 Pour plusieurs variables

» On souhaite à présent compter le nombre d'occurrences pour les variables 'sexe' et 'clso'

```

hdv2003 |>
  count(sexe, clso)

# A tibble: 6 x 3
  sexe  clso          n
  <chr> <chr>      <int>
1 Femme Ne sait pas    19
2 Femme Non           592
3 Femme Oui           490
4 Homme Ne sait pas     8
5 Homme Non           445
6 Homme Oui           446

```

Note

Par défaut, les modalités sont affichés selon l'ordre alphanumérique pour les variables de type "character" ou l'ordre déterminé lors de leur création pour les facteurs. Pour afficher les données par ordre décroissant des fréquences, on ajoute `sort = TRUE` dans la fonction `count()` (fonctionne avec une ou plusieurs variables)

```

hdv2003 |>
  count(clso, sort = TRUE)

# A tibble: 3 x 2
  clso          n
  <chr>      <int>
1 Non        1037
2 Oui         936
3 Ne sait pas   27

```

2.1.11.2 avec la fonction summarise()

2.1.11.2.1 Pour une variable

» On souhaite calculer différentes statistiques pour connaître la distribution de la variable 'age'

```

hdv2003 |>
  summarise(moyenne_age = mean(age, na.rm = TRUE),
            ecarttype_age = sd(age, na.rm = TRUE),
            minimum_age = min(age, na.rm = TRUE),
            mediane_age = median(age, na.rm = TRUE),
            maximum_age = max(age, na.rm = TRUE))

# A tibble: 1 x 5
  moyenne_age ecarttype_age minimum_age mediane_age maximum_age
  <dbl>      <dbl>      <int>      <dbl>      <int>
1    48.2      16.9        18        48        97

```

2.1.11.2.2 Pour plusieurs variables simultanément

» On souhaite calculer différentes statistiques pour connaître la distribution de toutes les variables numériques du jeu de données (sauf 'id' et 'an.naissance')

```

hdv2003 |>
  select(-c(id, an.naissance)) |>
  summarise(across(where(is.numeric),
                    list(moyenne = mean,
                          ecarttype = sd,
                          mediane = median)))

# A tibble: 1 x 12
  age_moyenne age_ecarttype age_mediane poids_moyenne poids_ecarttype poids_mediane
  <dbl>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
1    48.2      16.9        48    5536.    4375.    4631.
# i 6 more variables: freres.soeurs_moyenne <dbl>, freres.soeurs_ecarttype <dbl>,
# freres.soeurs_mediane <dbl>, heures.tv_moyenne <dbl>, heures.tv_ecarttype <dbl>,
# heures.tv_mediane <dbl>

```

i Avec `summarise(across())`, les noms des nouvelles variables sont créés en concaténant les noms des variables initiales et les noms des fonctions indiqués dans `list()`, séparés par un _

2.1.12 Effectuer des opérations groupées avec la fonction `group_by()`

La fonction `group_by()` permet de définir des groupes pour effectuer des fonctions par groupe plutôt que sur l'ensemble des données

- On peut utiliser une ou plusieurs variables pour regrouper les données
- Cette fonction est souvent utilisée avec d'autres fonctions comme `mutate()`, `filter()` ou `summarise()`

» On souhaite calculer l'âge moyen, minimum et maximum des personnes qui écoutent ou non du hard rock

```
hdv2003 |>
  group_by(hard.rock) |>
  summarise(moyenne_age = mean(age, na.rm = TRUE),
            minimum_age = min(age, na.rm = TRUE),
            maximum_age = max(age, na.rm = TRUE))
```

A tibble: 2 x 4

	hard.rock	moyenne_age	minimum_age	maximum_age
	<fct>	<dbl>	<int>	<int>
1	Non	48.3	18	97
2	Oui	27.6	19	44

» On souhaite faire des groupes selon l'écoute de hard rock et la pratique de la pêche-chasse, pour calculer l'âge moyen, minimum et maximum de chacun des groupes constitués

```
hdv2003 |>
  group_by(hard.rock, peche.chasse) |>
  summarise(moyenne_age = mean(age, na.rm = TRUE),
            minimum_age = min(age, na.rm = TRUE),
            maximum_age = max(age, na.rm = TRUE))
```

A tibble: 4 x 5

Groups: hard.rock [2]

	hard.rock	peche.chasse	moyenne_age	minimum_age	maximum_age
	<fct>	<fct>	<dbl>	<int>	<int>
1	Non	Non	48.4	18	97
2	Non	Oui	47.4	18	78
3	Oui	Non	26.5	19	43
4	Oui	Oui	31.7	25	44

- On peut cumuler `group_by()` sur plusieurs variables avec `summarise()` de plusieurs variables

```
hdv2003 |>
  select(-c(id, an.naissance)) |>
  group_by(hard.rock, peche.chasse) |>
  summarise(across(where(is.numeric),
                    list(moyenne = mean,
                         ecarttype = sd)))
```

A tibble: 4 x 10

Groups: hard.rock [2]

	hard.rock	peche.chasse	age_moyenne	age_ecarttype	poids_moyenne	poids_ecarttype
	<fct>	<fct>	<dbl>	<dbl>	<dbl>	<dbl>
1	Non	Non	48.4	17.1	5464.	4317.
2	Non	Oui	47.4	15.3	6057.	4806.
3	Oui	Non	26.5	7.23	6663.	4520.

```

4 Oui      Oui      31.7      10.7      5323.      2350.
# i 4 more variables: freres.soeurs_moyenne <dbl>, freres.soeurs_ecarttype <dbl>,
#   heures.tv_moyenne <dbl>, heures.tv_ecarttype <dbl>

```

- Pour annuler la structuration en groupes, on utilise la fonction `ungroup()`

```

hdv2003 |>
  select(-c(id, an.naissance)) |>
  ungroup() |>
  summarise(across(where(is.numeric),
                    list(moyenne = mean,
                        ecarttype = sd)))

```

```

# A tibble: 1 x 8
  age_moyenne age_ecarttype poids_moyenne poids_ecarttype freres.soeurs_moyenne
      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
1    48.2      16.9      5536.      4375.      3.28
# i 3 more variables: freres.soeurs_ecarttype <dbl>, heures.tv_moyenne <dbl>,
#   heures.tv_ecarttype <dbl>

```


⚠ `group_by()` fonctionne pour tous les verbes vus précédemment, sauf `arrange()`, qui trie la table sans tenir compte des groupes

```
hdv2003 |>
  group_by(sexe) |>
  arrange(desc(age))

# A tibble: 2,000 x 22
# Groups:   sexe [2]
   id age an.naissance sexe nivetud poids occup qualif freres.soeurs clso relig trav.imp
<int> <int>      <dbl> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>
1  1916   97      1906 Femme Derniere a~ 2163. Autr~ Autre          5 Non  Prat~ <NA>
2   270   96      1907 Femme Derniere a~ 9993. Retr~ <NA>          1 Oui  Ni c~ <NA>
3  1542   93      1910 Femme Derniere a~ 7108. Reti~ <NA>          7 Non  Prat~ <NA>
4   235   92      1911 Homme N'a jamais~ 827. Retr~ Ouvri~          0 Ne s~ Prat~ <NA>
5  1151   91      1912 Homme Derniere a~ 4291. Reti~ <NA>          3 Oui  Appa~ <NA>
6    96   90      1913 Femme Derniere a~ 5875. Au f~ Emplo~          7 Non  Prat~ <NA>
7  1122   90      1913 Femme 1er cycle 14812. Retr~ Emplo~          3 Oui  Prat~ <NA>
8  1767   90      1913 Femme Derniere a~ 12266. Reti~ <NA>          2 Non  Appa~ <NA>
9   759   89      1914 Femme Derniere a~ 9555. Au f~ <NA>          1 Non  Prat~ <NA>
10  971   89      1914 Homme Derniere a~ 2374. Retr~ Ouvri~          5 Oui  Prat~ <NA>
# i 1,990 more rows
# i 10 more variables: trav.satisf <fct>, hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>,
# cuisine <fct>, bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>, cl_age <chr>
Pour que le tri soit pris en compte avec la fonction arrange(), il faut ajouter .by_group = TRUE dans
cette dernière
```

```
hdv2003 |>
  group_by(sexe) |>
  arrange(desc(age), .by_group = TRUE)

# A tibble: 2,000 x 22
# Groups:   sexe [2]
   id age an.naissance sexe nivetud poids occup qualif freres.soeurs clso relig trav.imp
<int> <int>      <dbl> <chr> <fct>      <dbl> <fct> <fct>      <int> <chr> <fct> <fct>
1  1916   97      1906 Femme Derniere a~ 2163. Autr~ Autre          5 Non  Prat~ <NA>
2   270   96      1907 Femme Derniere a~ 9993. Retr~ <NA>          1 Oui  Ni c~ <NA>
3  1542   93      1910 Femme Derniere a~ 7108. Reti~ <NA>          7 Non  Prat~ <NA>
4    96   90      1913 Femme Derniere a~ 5875. Au f~ Emplo~          7 Non  Prat~ <NA>
5  1122   90      1913 Femme 1er cycle 14812. Retr~ Emplo~          3 Oui  Prat~ <NA>
6  1767   90      1913 Femme Derniere a~ 12266. Reti~ <NA>          2 Non  Appa~ <NA>
7   759   89      1914 Femme Derniere a~ 9555. Au f~ <NA>          1 Non  Prat~ <NA>
8  1117   89      1914 Femme A arrete s~ 5677. Au f~ Ouvri~          5 Oui  Appa~ <NA>
9  1127   89      1914 Femme A arrete s~ 25986. Autr~ <NA>          2 Non  Appa~ <NA>
10  1516   88      1915 Femme Derniere a~ 15686. Reti~ <NA>          2 Non  Prat~ <NA>
# i 1,990 more rows
# i 10 more variables: trav.satisf <fct>, hard.rock <fct>, lecture.bd <fct>, peche.chasse <fct>,
# cuisine <fct>, bricol <fct>, cinema <fct>, sport <fct>, heures.tv <dbl>, cl_age <chr>
```

2.1.13 Fiche aide-mémoire {dplyr}

Toutes les fonctions du package {dplyr} sont disponibles dans la [cheatsheet](#) du package

2.2 Manipulations avec le package {tidyr}

Les fonctions utiles pour

- réorganiser les données

- `pivot_longer()` : transformer des colonnes en lignes
- `pivot_wider()` : transformer des lignes en colonnes
- **manipuler les colonnes**
 - `separate()` : séparer une colonne en plusieurs colonnes
 - `unite()` : regrouper plusieurs colonnes en une seule
- **gérer les données manquantes**
 - `drop_na()` : supprimer les lignes contenant des valeurs manquantes
 - `replace_na()` : remplacer les valeurs manquantes par des valeurs spécifiées

2.2.1 Transformer des colonnes en lignes avec `pivot_longer()`

Cette fonction convertit plusieurs colonnes en deux nouvelles colonnes : une pour les noms des variables, une pour leurs valeurs correspondantes

⇒ réduit le nombre de colonnes mais augmente celui des lignes, rendant ainsi le jeu de données plus "long"

» On transforme les colonnes des différentes activités (hard.rock, lecture.bd, peche.chasse, cuisine, bricol) en un format long

```
hdv2003 |>
  select(sexe, hard.rock, lecture.bd, peche.chasse, cuisine, bricol) |>
  pivot_longer(cols = c(hard.rock, lecture.bd, peche.chasse, cuisine, bricol),
               names_to = "activite",
               values_to = "pratique") -> hdvlong

hdvlong
```

```
# A tibble: 10,000 x 3
  sexe activite pratique
<chr> <chr>      <fct>
1 Femme hard.rock Non
2 Femme lecture.bd Non
3 Femme peche.chasse Non
4 Femme cuisine Oui
5 Femme bricol Non
6 Femme hard.rock Non
7 Femme lecture.bd Non
8 Femme peche.chasse Non
9 Femme cuisine Non
10 Femme bricol Non
# i 9,990 more rows
```

Elle est particulièrement utile pour restructurer les données pour certaines analyses statistiques et représentations graphiques qui nécessitent un format long

2.2.2 Transformer des lignes en colonnes avec `pivot_wider()`

Cette fonction sert à transformer des données du format long au format large ("wide")

» On veut créer un tableau croisé montrant le nombre de personnes pratiquant chaque activité pour chaque modalité de la variable 'sexe'

```
hdvlong |>
  filter(pratique == "Oui") |>
  group_by(sexe, activite) |>
  summarise(count = n()) |>
```

```

pivot_wider(names_from = activite,
             values_from = count,
             values_fill = 0)

```

```

# A tibble: 2 x 6
# Groups:   sexe [2]
  sexe bricol cuisine hard.rock lecture.bd peche.chasse
  <chr> <int>   <int>    <int>    <int>    <int>
1 Femme   338    611        7        32        52
2 Homme   515    270        7        15       172

```

Cette fonction est utile pour créer des tableaux croisés permettant de visualiser et comparer des catégories

2.2.3 Séparer une colonne en plusieurs colonnes avec separate()

» Dans le jeu de données **data_fulljoin** créé précédemment, on va séparer les valeurs contenues dans la variable 'localisation' en 2 variables, l'une que l'on nommera 'ville', l'autre 'departement'

```

# Jeu de données data_fulljoin
data_fulljoin

```

```

# A tibble: 6 x 6
  id sexe  age localisation code  revenu
  <dbl> <chr> <dbl> <chr>    <chr> <dbl>
1     1 F    25 Bordeaux/33 063    2500
2     2 H    57 <NA>      <NA>     NA
3     3 H    88 Cestas/33  122    3000
4     4 F    32 Bordeaux/33 063    3700
5     5 F    44 <NA>      <NA>     NA
6     6 <NA>   NA Dax/40    088    1850

```

```

# Séparation des valeurs de la variable localisation
data_fulljoin |>
  separate(col = localisation,
           into = c("ville", "departement"),
           sep = "/",
           remove = FALSE) -> data_fulljoin
data_fulljoin

```

```

# A tibble: 6 x 8
  id sexe  age localisation ville  departement code  revenu
  <dbl> <chr> <dbl> <chr>    <chr>    <chr>    <chr> <dbl>
1     1 F    25 Bordeaux/33 Bordeaux 33      063    2500
2     2 H    57 <NA>      <NA>    <NA>    <NA>     NA
3     3 H    88 Cestas/33  Cestas  33      122    3000
4     4 F    32 Bordeaux/33 Bordeaux 33      063    3700
5     5 F    44 <NA>      <NA>    <NA>    <NA>     NA
6     6 <NA>   NA Dax/40    Dax     40      088    1850

```



Pour supprimer la variable d'origine dans le jeu de données, il faut indiquer `remove = TRUE`

2.2.4 Regrouper plusieurs colonnes en une seule avec unite()

» Dans **data_fulljoin**, on va regrouper les valeurs de "departement" et "code" pour reconstituer le code insee de la commune

```
data_fulljoin |>
  unite("code_insee", departement:code, sep= "", remove = FALSE)
```

A tibble: 6 x 9

	id	sexe	age	localisation	ville	code_insee	departement	code	revenu
	<dbl>	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<chr>	<dbl>
1	1	F	25	Bordeaux/33	Bordeaux	33063	33	063	2500
2	2	H	57	<NA>	<NA>	NANA	<NA>	<NA>	NA
3	3	H	88	Cestas/33	Cestas	33122	33	122	3000
4	4	F	32	Bordeaux/33	Bordeaux	33063	33	063	3700
5	5	F	44	<NA>	<NA>	NANA	<NA>	<NA>	NA
6	6	<NA>	NA	Dax/40	Dax	40088	40	088	1850



Pour supprimer les variables d'origine dans le jeu de données, il faut indiquer `remove = TRUE`

2.2.5 Supprimer les lignes contenant des valeurs manquantes avec `drop_na()`

» On supprime les valeurs manquantes du jeu de données `data_fulljoin`

```
data_fulljoin |>
  drop_na()
```

A tibble: 3 x 8

	id	sexe	age	localisation	ville	departement	code	revenu
	<dbl>	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<dbl>
1	1	F	25	Bordeaux/33	Bordeaux	33	063	2500
2	3	H	88	Cestas/33	Cestas	33	122	3000
3	4	F	32	Bordeaux/33	Bordeaux	33	063	3700

Il est possible de supprimer les valeurs manquantes pour une variable en particulier, en précisant son nom dans `drop_na()`

```
data_fulljoin |>
  drop_na(age)
```

A tibble: 5 x 8

	id	sexe	age	localisation	ville	departement	code	revenu
	<dbl>	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<dbl>
1	1	F	25	Bordeaux/33	Bordeaux	33	063	2500
2	2	H	57	<NA>	<NA>	<NA>	<NA>	NA
3	3	H	88	Cestas/33	Cestas	33	122	3000
4	4	F	32	Bordeaux/33	Bordeaux	33	063	3700
5	5	F	44	<NA>	<NA>	<NA>	<NA>	NA

2.2.6 Remplacer les valeurs manquantes par des valeurs spécifiées avec `replace_na()`

```
data_fulljoin |>
  mutate(revenu = replace_na(revenu, -999))
```

A tibble: 6 x 8

	id	sexe	age	localisation	ville	departement	code	revenu
	<dbl>	<chr>	<dbl>	<chr>	<chr>	<chr>	<chr>	<dbl>
1	1	F	25	Bordeaux/33	Bordeaux	33	063	2500
2	2	H	57	<NA>	<NA>	<NA>	<NA>	-999
3	3	H	88	Cestas/33	Cestas	33	122	3000

4	4 F	32 Bordeaux/33	Bordeaux	33	063	3700
5	5 F	44 <NA>	<NA>	<NA>	<NA>	-999
6	6 <NA>	NA Dax/40	Dax	40	088	1850

💡 Il faut compléter avec une valeur du même type. Ici, pour revenu, il faut une valeur numérique. Si l'on souhaite ajouter du texte dans une variable numérique, il faut modifier le type de la variable

```
data_fulljoin |>
  mutate(revenu = replace_na(as.character(revenu), "Refus de répondre"))

# A tibble: 6 x 8
   id sexe  age localisation ville  departement code  revenu
<dbl> <chr> <dbl> <chr>      <chr>    <chr>      <chr> <chr>
1     1 F    25 Bordeaux/33 Bordeaux 33      063  2500
2     2 H    57 <NA>      <NA>    <NA>    <NA> Refus de répondre
3     3 H    88 Cestas/33  Cestas  33      122  3000
4     4 F    32 Bordeaux/33 Bordeaux 33      063  3700
5     5 F    44 <NA>      <NA>    <NA>    <NA> Refus de répondre
6     6 <NA>   NA Dax/40    Dax     40      088  1850
```

2.2.7 Fiche aide-mémoire {tidyr}

Toutes les fonctions du package {tidyr} sont disponibles dans la [cheatsheet](#) du package

2.3 Manipulations des chaînes de caractères avec le package {stringr}

Les fonctions utiles pour

- la recherche et correspondance d'expression
 - `str_detect()` : détecter la présence d'une expression (*pattern*) dans une chaîne de caractères
 - `str_starts()` : détecter la présence d'une expression au début d'une chaîne de caractères
- la transformation de chaînes
 - `str_replace()` : remplacer une expression dans une chaîne de caractères spécifiée
 - `str_to_upper()` : convertir une chaîne en majuscules
 - `str_to_lower()` : convertir une chaîne en minuscules
 - `str_to_title()` : convertir une chaîne en ne mettant que la 1re lettre des mots en majuscules

2.3.1 Détection d'une expression avec la fonction `str_detect()`

» On souhaite sélectionner les lignes qui contiennent l'expression "qualifie" dans la variable 'qualif'

```
hdv2003 |>
  filter(str_detect(qualif, "qualifie")) |>
  # pour la vérification, on affiche seulement la variable qualif
  select(qualif)
```

```
# A tibble: 292 x 1
  qualif
<fct>
1 Ouvrier qualifie
2 Ouvrier qualifie
3 Ouvrier qualifie
4 Ouvrier qualifie
5 Ouvrier qualifie
6 Ouvrier qualifie
```

```

7 Ouvrier qualifie
8 Ouvrier qualifie
9 Ouvrier qualifie
10 Ouvrier qualifie
# i 282 more rows

```

2.3.2 Détection d'une expression avec la fonction str_starts()

» On souhaite sélectionner les lignes qui commencent par l'expression "Ouvrier" dans la variable 'qualif'

```

hdv2003 |>
  filter(str_starts(qualif, "Ouvrier")) |>
  # pour la vérification, on affiche seulement la variable qualif
  select(qualif)

# A tibble: 495 x 1
  qualif
  <fct>
1 Ouvrier qualifie
2 Ouvrier qualifie
3 Ouvrier qualifie
4 Ouvrier specialise
5 Ouvrier qualifie
6 Ouvrier specialise
7 Ouvrier qualifie
8 Ouvrier specialise
9 Ouvrier qualifie
10 Ouvrier qualifie
# i 485 more rows

```

2.3.3 Remplacement d'une expression avec la fonction str_replace()

» On souhaite remplacer l'expression "Ouvrier" par "Ouv." dans la variable 'qualif'

```

hdv2003 |>
  mutate(qualif = str_replace(qualif, "Ouvrier", "Ouv.")) -> hdv2003

```

»> Vérification

```

# Variable après modification
hdv2003 |>
  select(qualif) |>
  questionr::freq()

```

	n	%	val%
Autre	58	2.9	3.5
Cadre	260	13.0	15.7
Employe	594	29.7	35.9
Ouv. qualifie	292	14.6	17.7
Ouv. specialise	203	10.2	12.3
Profession intermediaire	160	8.0	9.7
Technicien	86	4.3	5.2
NA	347	17.3	NA

2.3.4 Conversion du texte en majuscules avec la fonction `str_to_upper()`

» On souhaite convertir les modalités de la variable 'hard.rock' en majuscules

```
hdv2003 |>
  mutate(hard.rock = str_to_upper(hard.rock)) -> hdv2003
```

»> Vérification

```
# Variable initiale
hdv2003 |>
  select(hard.rock) |>
  questionr::freq()
```

```
      n    % val%
Non 1986 99.3 99.3
Oui   14   0.7  0.7
```

```
# Variable après modification
hdv2003 |>
  select(hard.rock) |>
  questionr::freq()
```

```
      n    % val%
NON 1986 99.3 99.3
OUI   14   0.7  0.7
```

2.3.5 Conversion du texte en minuscules avec la fonction `str_to_lower()`

» On souhaite convertir les modalités de la variable 'hard.rock' en minuscules

```
hdv2003 |>
  mutate(hard.rock = str_to_lower(hard.rock)) -> hdv2003
```

»> Vérification

```
# Variable initiale
hdv2003 |>
  select(hard.rock) |>
  questionr::freq()
```

```
      n    % val%
NON 1986 99.3 99.3
OUI   14   0.7  0.7
```

```
# Variable après modification
hdv2003 |>
  select(hard.rock) |>
  questionr::freq()
```

```
      n    % val%
non 1986 99.3 99.3
oui   14   0.7  0.7
```

2.3.6 Conversion du texte en minuscules avec la fonction `str_to_title()`

» On souhaite convertir le texte de la variable 'qualif' avec la première lettre des mots en majuscules

```
hdv2003 |>
  mutate(qualif = str_to_title(qualif)) -> hdv2003
```

»> Vérification

```
# Variable initiale
hdv2003 |>
  select(qualif) |>
  questionr::freq()
```

	n	%	val%
Autre	58	2.9	3.5
Cadre	260	13.0	15.7
Employe	594	29.7	35.9
Ouv. qualifie	292	14.6	17.7
Ouv. specialise	203	10.2	12.3
Profession intermediaire	160	8.0	9.7
Technicien	86	4.3	5.2
NA	347	17.3	NA

```
# Variable après modification
hdv2003 |>
  select(qualif) |>
  questionr::freq()
```

	n	%	val%
Autre	58	2.9	3.5
Cadre	260	13.0	15.7
Employe	594	29.7	35.9
Ouv. Qualifie	292	14.6	17.7
Ouv. Specialise	203	10.2	12.3
Profession Intermediaire	160	8.0	9.7
Technicien	86	4.3	5.2
NA	347	17.3	NA

2.3.7 Fiche aide-mémoire {stringr}

Toutes les fonctions du package {stringr} sont disponibles dans la [cheatsheet](#) du package

2.4 Manipulation des facteurs avec le package {forcats}

Les fonctions utiles pour

- **changer l'ordre des modalités (niveaux)**
 - `fct_infreq()`: réorganiser les niveaux du plus fréquent au moins fréquent
 - `fct_relevel()` : réorganiser les niveaux d'un facteur selon un ordre spécifique défini par l'utilisateur
 - `fct_reorder()` : réorganiser les niveaux d'un facteur selon les valeurs d'une autre variable (numérique)
 - `fct_rev()` : inverser l'ordre des niveaux d'un facteur
- **modifier les valeurs des modalités**
 - `fct_recode()` : changer le niveau d'un facteur

- `fct_relabel()` : renommer les niveaux d'un facteur en appliquant une fonction à chaque niveau
- `fct_collapse()` : regrouper les niveaux d'un facteur
- `fct_lump()` : regrouper les niveaux peu fréquents d'un facteur en un nouveau niveau appelé "autre"
- `fct_anon()` : anonymiser les niveaux
- **ajouter ou supprimer des modalités**
 - `fct_expand()` : ajouter des niveaux supplémentaires à un facteur
 - `fct_drop()` : supprimer les niveaux inutilisés d'un facteur
- **faire des opérations sur les facteurs**
 - `fct_unique()` : lister les niveaux d'un facteur
 - `fct_count()` : compter le nombre d'observation pour chacun des niveaux d'un facteur
 - `fct_cross()` : combiner les niveaux de plusieurs facteurs

2.4.1 Réorganiser les niveaux d'un facteur selon leur importance avec `fct_infreq()`

» On souhaite réorganiser les niveaux de 'occup' selon leur importance

```
hdv2003 |>
  mutate(occup = forcats::fct_infreq(occup)) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(occup) |>
  questionr::freq()
```

	n	%	val%
Exerce une profession	1049	52.4	52.4
Chomeur	134	6.7	6.7
Etudiant, eleve	94	4.7	4.7
Retraite	392	19.6	19.6
Retire des affaires	77	3.9	3.9
Au foyer	171	8.6	8.6
Autre inactif	83	4.2	4.2

```
# Modalités après modification
hdv2003 |>
  pull(occup) |>
  questionr::freq()
```

	n	%	val%
Exerce une profession	1049	52.4	52.4
Retraite	392	19.6	19.6
Au foyer	171	8.6	8.6
Chomeur	134	6.7	6.7
Etudiant, eleve	94	4.7	4.7
Autre inactif	83	4.2	4.2
Retire des affaires	77	3.9	3.9

2.4.2 Réorganiser les niveaux des facteurs manuellement avec `fct_relevel()`

» On veut avoir les niveaux de la variable 'trav.satisf' dans l'ordre suivant : "Insatisfaction", "Equilibre" et "Satisfaction"

```
hdv2003 |>
  mutate(trav.satisf = fct_relevel(trav.satisf,
                                   c("Insatisfaction", "Equilibre", "Satisfaction")) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(trav.satisf) |>
  levels()
```

```
[1] "Satisfaction" "Insatisfaction" "Equilibre"
```

```
# Modalités après modification
hdv2003 |>
  pull(trav.satisf) |>
  levels()
```

```
[1] "Insatisfaction" "Equilibre" "Satisfaction"
```

i Pour rappel

Par défaut, les modalités d'une variable qualitative sont triées par ordre alphanumérique

2.4.3 Réorganiser les niveaux en fonction des valeurs d'une variable numérique avec `fct_reorder()`

» On réorganise les niveaux de 'relig' selon l'âge moyen de chacune des modalités

```
hdv2003 |>
  mutate(relig = fct_reorder(relig, age, .fun = median, .desc = FALSE)) |>
  group_by(relig) |>
  summarise(age_median = median(age))
```

»> Vérification

```
# Ordre initial
hdv2003 |>
  group_by(relig) |>
  summarise(age_median = median(age))
```

```
# A tibble: 6 x 2
  relig          age_median
<fct>          <dbl>
1 Praticant regulier      55
2 Praticant occasionnel   51
3 Appartenance sans pratique 48
4 Ni croyance ni appartenance 39
5 Rejet                   43
6 NSP ou NVPR             48.5
```

```
# Ordre après modification
hdv2003 |>
  mutate(relig = fct_reorder(relig, age, .fun = median, .desc = FALSE)) |>
  group_by(relig) |>
  summarise(age_median = median(age))
```

```
# A tibble: 6 x 2
  relig                age_median
  <fct>                <dbl>
1 Ni croyance ni appartenance    39
2 Rejet                          43
3 Appartenance sans pratique     48
4 NSP ou NVPR                   48.5
5 Pratiquant occasionnel         51
6 Pratiquant regulier            55
```

i

- Le paramètre `.fun` définit la fonction d'agrégation. Elle peut être `mean`, `median`, `sd`, etc.
- Pour ordonner les valeurs par ordre décroissant, il faut définir le paramètre `.desc = TRUE`

Cette fonction est particulièrement utile pour adapter l'ordre des modalités dans les visualisations graphiques ou pour comparer des groupes selon une variable quantitative

2.4.4 Inverser l'ordre des niveaux avec `fct_rev()`

» Dans la variable 'lecture.bd', on veut que les niveaux soient dans l'ordre suivant : "Oui" et "Non"

```
hdv2003 |>
  mutate(lecture.bd = fct_rev(lecture.bd)) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(lecture.bd) |>
  levels()
```

```
[1] "Non" "Oui"
```

```
# Modalités après modification
hdv2003 |>
  pull(lecture.bd) |>
  levels()
```

```
[1] "Oui" "Non"
```

2.4.5 Changer le niveau d'un facteur avec `fct_recode()`

» On modifie le niveau "2eme cycle" de 'nivetud' en "2e cycle"

```
hdv2003 |>
  mutate(nivetud = forcats::fct_recode(nivetud, "2e cycle" = "2eme cycle")) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(nivetud) |>
  levels()
```

```
[1] "N'a jamais fait d'etudes"
```

```
[2] "A arrete ses etudes, avant la derniere annee d'etudes primaires"
```

```
[3] "Derniere annee d'etudes primaires"
```

```
[4] "1er cycle"
[5] "2eme cycle"
[6] "Enseignement technique ou professionnel court"
[7] "Enseignement technique ou professionnel long"
[8] "Enseignement superieur y compris technique superieur"
```

```
# Modalités après modification
hdv2003 |>
  pull(nivetud) |>
  levels()
```

```
[1] "N'a jamais fait d'etudes"
[2] "A arrete ses etudes, avant la derniere annee d'etudes primaires"
[3] "Derniere annee d'etudes primaires"
[4] "1er cycle"
[5] "2e cycle"
[6] "Enseignement technique ou professionnel court"
[7] "Enseignement technique ou professionnel long"
[8] "Enseignement superieur y compris technique superieur"
```

2.4.6 Modifier le nom des niveaux de manière globale avec `fct_relabel()`

» On passe les niveaux de la variable 'cuisine' en majuscules

```
hdv2003 |>
  mutate(cuisine = forcats::fct_relabel(cuisine, toupper)) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(cuisine) |>
  levels()
```

```
[1] "Non" "Oui"
```

```
# Modalités après modification
hdv2003 |>
  mutate(cuisine = forcats::fct_relabel(cuisine, toupper)) -> hdv2003
```

```
# Modalités après modification
hdv2003 |>
  pull(cuisine) |>
  levels()
```

```
[1] "NON" "OUI"
```

2.4.7 Regrouper les niveaux d'un facteur en catégories manuellement définies avec `fct_collapse()`

» Dans la variable 'relig', on veut regrouper les niveaux "Pratiquant regulier" et "Pratiquant occasionnel" dans la modalité "Pratiquant"

```
hdv2003 |>
  mutate(relig = fct_collapse(relig,
    "Pratiquant" = c("Pratiquant regulier", "Pratiquant occasionnel"))) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(relig) |>
  levels()
```

```
[1] "Pratiquant regulier"      "Pratiquant occasionnel"  "Appartenance sans pratique"
[4] "Ni croyance ni appartenance" "Rejet"                  "NSP ou NVPR"
```

```
# Modalités après modification
hdv2003 |>
  pull(relig) |>
  levels()
```

```
[1] "Pratiquant"              "Appartenance sans pratique" "Ni croyance ni appartenance"
[4] "Rejet"                  "NSP ou NVPR"
```

2.4.8 Conserver le(s) niveau(x) le(s) plus commun(s) et grouper les autres avec `fct_lump()`

» On veut conserver le niveau le plus courant de la variable 'occup' et regrouper les autres niveaux dans un même niveau

```
hdv2003 |>
  mutate(occup = fct_lump(occup, n = 1)) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(occup) |>
  questionr::freq()
```

	n	%	val%
Exerce une profession	1049	52.4	52.4
Retraite	392	19.6	19.6
Au foyer	171	8.6	8.6
Chomeur	134	6.7	6.7
Etudiant, eleve	94	4.7	4.7
Autre inactif	83	4.2	4.2
Retire des affaires	77	3.9	3.9

```
# Modalités après modification
hdv2003 |>
  pull(occup) |>
  questionr::freq()
```

	n	%	val%
Exerce une profession	1049	52.4	52.4
Other	951	47.5	47.5



- On peut nommer le niveau `other_level` (ex : `other_level = "Autre"`).
- Avec `n` négatif, la fonction conserve les `n` niveaux les moins courants.
- On peut donner une fréquence d'apparition, en changeant `n` par `prop` (ex : `prop = 0.2` conserve les niveaux qui représentent au moins 20% du vecteur).
- Avec un `prop` négatif, on définit une marge maximale d'apparition.

2.4.9 Anonymiser les niveaux avec fct_anon()

Cette fonction est utile pour transformer le texte en codes numériques

NB1 : la variable anonymisée est toujours un facteur

NB2 : ni les valeurs ni l'ordre des niveaux ne sont conservés, les identifiants numériques sont arbitraires

» On remplace les niveaux de la variable 'bricol' par des identifiants numériques aléatoires

```
hdv2003 |>
  mutate(bricol_anon = fct_anon(bricol)) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(bricol) |>
  levels()
```

```
[1] "Non" "Oui"
```

```
# Modalités après modification
hdv2003 |>
  pull(bricol_anon) |>
  levels()
```

```
[1] "1" "2"
```

💡 Si on le souhaite, il est possible d'ajouter un préfixe aux nouveaux labels

```
hdv2003 |>
  mutate(bricol_anon = fct_anon(bricol, prefix = "bricol_")) |>
  pull(bricol_anon) |>
  levels()

[1] "bricol_1" "bricol_2"
```

2.4.10 Ajouter de nouveaux niveaux à un facteur avec fct_expand()

» On veut ajouter les niveaux "Ne se reconnaît pas dans ces catégories" et "Refus de répondre" dans la variable 'sexe'

```
hdv2003 |>
  mutate(sexe = fct_expand(sexe,
                           "Ne se reconnaît pas dans ces catégories", "Refus de répondre")) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(sexe) |>
  questionr::freq()
```

```
      n % val%
Femme 1101 55  55
Homme   899 45  45
```

```
# Modalités après modification
hdv2003 |>
  pull(sexe) |>
  questionr::freq()
```

	n	%	val%
Femme	1101	55	55
Homme	899	45	45
Ne se reconnaît pas dans ces catégories	0	0	0
Refus de répondre	0	0	0

2.4.11 Supprimer les niveaux de facteurs inutilisés avec fct_drop()

» On veut supprimer les niveaux inutilisés, pour lesquels on n'a pas d'observations, dans la variable 'sexe'

```
hdv2003 |>
  mutate(sexe = fct_drop(sexe)) -> hdv2003
```

»> Vérification

```
# Modalités initiales
hdv2003 |>
  pull(sexe) |>
  questionr::freq()
```

	n	%	val%
Femme	1101	55	55
Homme	899	45	45
Ne se reconnaît pas dans ces catégories	0	0	0
Refus de répondre	0	0	0

```
# Modalités après modification
hdv2003 |>
  pull(sexe) |>
  questionr::freq()
```

	n	%	val%
Femme	1101	55	55
Homme	899	45	45

💡 Avec `fct_drop(only = "...")`, on peut préciser les niveaux que l'on souhaite supprimer

```
hdv2003 |>
  mutate(sexe = fct_drop(sexe, only = "Refus de répondre")) |>
  pull(sexe) |>
  questionr::freq()
```

	n	%	val%
Femme	1101	55	55
Homme	899	45	45
Ne se reconnaît pas dans ces catégories	0	0	0

2.4.12 Lister les niveaux d'un facteur `fct_unique()`

Cette fonction permet de vérifier l'ordre des modalités d'une variable facteur, ce qui peut s'avérer utile avant analyse, notamment pour les graphiques

» On veut connaître les niveaux de la variable 'occup'

```
hdv2003 |>
  pull(occup) |>
  fct_unique()
```

```
[1] Exerce une profession Other
Levels: Exerce une profession Other
```

2.4.13 Compter la fréquence des niveaux d'un facteur `fct_count()`

Cette fonction donne le tableau de fréquence des niveaux du facteur avec le nombre d'occurrences pour chacun d'eux

» On souhaite compter le nombre d'occurrences pour la variable 'relig'

```
hdv2003 |>
  pull(relig) |>
  fct_count(sort = FALSE, prop = FALSE)
```

```
# A tibble: 5 x 2
```

f	n
<fct>	<int>
1 Pratiquant	708
2 Appartenance sans pratique	760
3 Ni croyance ni appartenance	399
4 Rejet	93
5 NSP ou NVPR	40

Note

Il est possible de trier les niveaux par ordre décroissant des fréquences avec le paramètre `sort = TRUE`. Avec le paramètre `prop = TRUE`, il est possible d'ajouter une colonne avec les

```
hdv2003 |>
  pull(relig) |>
  fct_count(sort = TRUE, prop = TRUE)
```

```
# A tibble: 5 x 3
```

f	n	p
<fct>	<int>	<dbl>
1 Appartenance sans pratique	760	0.38
2 Pratiquant	708	0.354
3 Ni croyance ni appartenance	399	0.200
4 Rejet	93	0.0465
5 NSP ou NVPR	40	0.02

2.4.14 Combiner les niveaux de plusieurs facteurs `fct_cross()`

» On veut créer une variable qui combine les niveaux des variables 'cinema' et 'lecture.bd'


```
hdv2003 |>
  mutate(cinebd = fct_cross(cinema, lecture.bd, sep = ":")) -> hdv2003
```

»> Vérification

```
# Variables initiales - "Tableau croisé"
hdv2003 |>
  count(cinema, lecture.bd)
```

```
# A tibble: 4 x 3
  cinema lecture.bd     n
  <fct>   <fct>     <int>
1 Non    Oui         18
2 Non    Non        1156
3 Oui    Oui         29
4 Oui    Non        797
```

```
# Variable croisée
hdv2003 |>
  pull(cinebd) |>
  questionr::freq()
```

	n	% val%
Non:Oui	18	0.9 0.9
Oui:Oui	29	1.5 1.5
Non:Non	1156	57.8 57.8
Oui:Non	797	39.9 39.9



- On peut conserver les combinaisons qui n'ont aucune observation, en ajoutant le paramètre `keep_empty = TRUE` dans la fonction `fct_cross()`
- On peut paramétrer le séparateur entre les niveaux (ex. : `_`, `.`, etc.)
- Il est possible de croiser plus de deux facteurs

2.4.15 Fiche aide-mémoire {forcats}

Toutes les fonctions du package {forcats} sont disponibles dans la [cheatsheet](#) du package

3 Ressources et références bibliographiques

Aides et ressources en ligne

-  La page [Bulle d'R](#)
-  La cheatsheet [Bonnes pratiques en R à ETTIS](#)
-  Pour le package [forcats](#)

3.1 Packages

- Bache, S. M., et H. Wickham. 2022. *magrittr: A Forward-Pipe Operator for R*. <https://CRAN.R-project.org/package=magrittr>.
- Firke, S. 2024. *janitor: Simple Tools for Examining and Cleaning Dirty Data*. <https://sfirke.github.io/janitor/index.html>.
- Kaplan, J. 2025. *fastDummies: Fast Creation of Dummy (Binary) Columns and Rows from Categorical Variables*. <https://jacobkap.github.io/fastDummies/>.
- Wickham, H. 2023a. *forcats: Tools for Working with Categorical Variables (Factors)*. <https://forcats.tidyverse.org/>.

- . 2023b. *stringr: Simple, Consistent Wrappers for Common String Operations*. <https://stringr.tidyverse.org>.
- Wickham, H., R. François, L. Henry, K. Müller, et D. Vaughan. 2023. *dplyr: A Grammar of Data Manipulation*. <https://dplyr.tidyverse.org>.
- Wickham, H., D. Vaughan, et M. Girlich. 2024. *tidyr: Tidy Messy Data*. <https://tidyr.tidyverse.org>.