Manipulation de données avec dplyr

Malick SENE

2025-04-13

```
library(dplyr)#Chargement du package dplyr
library(haven)
wf <- read_dta("ehcvm_welfare_sen2021.dta")
```

#3. Manipulation de données avec dplyr

Une fois les données importées, l'étape suivante consiste à les préparer et les transformer en vue de l'analyse. Le package **dplyr**, composant central du **tidyverse**, fournit un ensemble d'outils puissants et cohérents pour manipuler efficacement les données contenues dans un `data.frame` ou un `tibble`. Il repose sur une syntaxe intuitive basée sur des **verbes** clairs (tels que `filter()`, `select()`, `mutate()`, etc.), qui permettent d'enchaîner les opérations de manière lisible et structurée. Conçu pour optimiser les performances, `dplyr` offre des fonctions généralement plus rapides que celles de base R, ce qui le rend particulièrement adapté au traitement de **données de grande dimension**. Il adopte également le paradigme des **données tidy**(abordé dans l'introduction), facilitant ainsi une approche rigoureuse, reproductible et fluide de la manipulation des données.

3.1 Les fonctions de dplyr

La manipulation de données avec dplyr se fait en utilisant un nombre réduit de fontions, qui correspondent chacun à une action différente appliquée à un tableau de données.

3.1.1 slice

La fonction slice() permet de sélectionner des lignes spécifiques de votre jeu de données en fonction de leur position. C'est un peu comme zoomer sur une section particulière d'un tableau. On lui passe en parametre un chiffre ou un vecteur de chiffre.

```
# Exemple : Sélectionner les 5 premières lignes de la base welfare
wf %>% slice(1:5)
```

```
## # A tibble: 5 x 47
##   grappe menage country  year  hhid vague month      zae        region   milieu
##    <dbl>  <dbl> <chr>   <dbl> <dbl> <dbl> <date>     <dbl+lbl>  <dbl+lb> <dbl+l>
## 1      2      5 SEN      2021   205     2 2022-05-01 11 [Dakar] 1 [daka~ 1 [Urb~
## 2      2     15 SEN      2021   215     2 2022-05-01 11 [Dakar] 1 [daka~ 1 [Urb~
## 3      2      3 SEN      2021   203     2 2022-05-01 11 [Dakar] 1 [daka~ 1 [Urb~
## 4      2     13 SEN      2021   213     2 2022-05-01 11 [Dakar] 1 [daka~ 1 [Urb~
## 5      2      8 SEN      2021   208     2 2022-06-01 11 [Dakar] 1 [daka~ 1 [Urb~
## # i 37 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
```

```
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>,
## #   pcexp <dbl>, zzae <dbl>, zref <dbl>, def_spa <dbl>, def_temp <dbl>, ...
```

3.1.2 filter

La fonction `filter()` permet de **sélectionner des lignes** d'un tableau de données en fonction d'une ou plusieurs conditions logiques.
Elle conserve uniquement les lignes pour lesquelles le test spécifié renvoie `TRUE`.

```
# Exemple : filtrer les ménages dont la taille est supérieur à 5 individu
wf %>% filter(hhsize > 5)
```

```
## # A tibble: 5,152 x 47
##    grappe menage country  year  hhid vague month      zae         region  milieu
##     <dbl>  <dbl> <chr>   <dbl> <dbl> <dbl> <date>     <dbl+lbl>  <dbl+l> <dbl+l>
##  1      2      1 SEN      2021   201     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  2      2     12 SEN      2021   212     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  3      3     17 SEN      2021   317     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  4      3      4 SEN      2021   304     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  5      4      3 SEN      2021   403     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  6      4     12 SEN      2021   412     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  7      4     11 SEN      2021   411     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  8      5      6 SEN      2021   506     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  9      5      7 SEN      2021   507     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
## 10      5      5 SEN      2021   505     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
## # i 5,142 more rows
## # i 37 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>, ...
```

3.1.3 select et rename

Parfois, vous ne voulez pas manipuler toutes les colonnes. Vous pouvez alors sélectionner ou renommer celles qui vous intéressent. select() vous permet de choisir des colonnes spécifiques, tandis que rename() vous aide à modifier leur nom :

```
# Sélectionner les colonnes "country" et "hactiv12m"
wf %>% select(country,hactiv12m)
```

```
## # A tibble: 7,120 x 2
##    country hactiv12m
##    <chr>   <dbl+lbl>
##  1 SEN     1 [Occupe]
##  2 SEN     1 [Occupe]
##  3 SEN     1 [Occupe]
##  4 SEN     1 [Occupe]
```

```
##  5 SEN     1 [Occupe]
##  6 SEN     1 [Occupe]
##  7 SEN     1 [Occupe]
##  8 SEN     3 [Non occupe]
##  9 SEN     1 [Occupe]
## 10 SEN     1 [Occupe]
## # i 7,110 more rows
```

```
# Renommer la colonne "age" en "âge"

wf %>% rename(country_21=country) ##La colonne à renommé se met à droite de l'égalité et le nouveau nom
```

```
## # A tibble: 7,120 x 47
##    grappe menage country_21  year  hhid vague month      zae         region
##     <dbl>  <dbl> <chr>      <dbl> <dbl> <dbl> <date>     <dbl+lbl>  <dbl+lbl>
##  1      2      5 SEN         2021   205     2 2022-05-01 11 [Dakar] 1 [dakar]
##  2      2     15 SEN         2021   215     2 2022-05-01 11 [Dakar] 1 [dakar]
##  3      2      3 SEN         2021   203     2 2022-05-01 11 [Dakar] 1 [dakar]
##  4      2     13 SEN         2021   213     2 2022-05-01 11 [Dakar] 1 [dakar]
##  5      2      8 SEN         2021   208     2 2022-06-01 11 [Dakar] 1 [dakar]
##  6      2     16 SEN         2021   216     2 2022-06-01 11 [Dakar] 1 [dakar]
##  7      2      7 SEN         2021   207     2 2022-06-01 11 [Dakar] 1 [dakar]
##  8      2      4 SEN         2021   204     2 2022-05-01 11 [Dakar] 1 [dakar]
##  9      2      1 SEN         2021   201     2 2022-05-01 11 [Dakar] 1 [dakar]
## 10      2     12 SEN         2021   212     2 2022-05-01 11 [Dakar] 1 [dakar]
## # i 7,110 more rows
## # i 38 more variables: milieu <dbl+lbl>, hhweight <dbl>, hhsize <dbl>,
## #   eqadu1 <dbl>, eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>,
## #   hmstat <dbl+lbl>, hreligion <dbl+lbl>, hnation <dbl+lbl>,
## #   hethnie <dbl+lbl>, halfa <dbl+lbl>, halfa2 <dbl+lbl>, heduc <dbl+lbl>,
## #   hdiploma <dbl+lbl>, hhandig <dbl+lbl>, hactiv7j <dbl+lbl>,
## #   hactiv12m <dbl+lbl>, hbranch <dbl+lbl>, hsectins <dbl+lbl>, ...
```

Dans la fonction select, si on fait précéder le nom d'un "-", la colonne est éliminée plutôt que sélectionnée: En outre, la syntaxe `colonne1:colonne2` permet de sélectionner **toutes les colonnes comprises entre `colonne1` et `colonne2`**, incluses.

La fonction `select()` peut également être utilisée pour **réorganiser l'ordre des colonnes** dans un tableau. La fonction auxiliaire `everything()` permet de sélectionner **toutes les colonnes restantes** qui n'ont pas encore été explicitement mentionnées.

Par exemple, si l'on souhaite placer la colonne `hhid` en première position dans le tableau `wf`, on peut écrire :

```
select(wf, hhid, everything())
```

```
## # A tibble: 7,120 x 47
##     hhid grappe menage country  year vague month      zae         region  milieu
##    <dbl>  <dbl>  <dbl> <chr>   <dbl> <dbl> <date>     <dbl+lbl>  <dbl+l> <dbl+l>
##  1   205      2      5 SEN      2021     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  2   215      2     15 SEN      2021     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  3   203      2      3 SEN      2021     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  4   213      2     13 SEN      2021     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  5   208      2      8 SEN      2021     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  6   216      2     16 SEN      2021     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
```

```
##  7   207      2      7 SEN      2021     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  8   204      2      4 SEN      2021     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  9   201      2      1 SEN      2021     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
## 10   212      2     12 SEN      2021     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
## # i 7,110 more rows
## # i 37 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>, ...
```

3.1.4 arrange

La fonction `arrange()` du package **dplyr** permet de **trier les lignes** d'un tableau de données en fonction des valeurs d'une ou plusieurs colonnes.
Elle est particulièrement utile pour organiser les données avant une visualisation, une analyse, ou simplement pour faciliter leur lecture.

Le tri peut se faire :

- par ordre croissant (par défaut),
- par ordre décroissant en combinant avec la fonction `desc()`.

```
# Trier les données de la base wf par la variable hage par ordre décroissant
wf%>% arrange(desc(hage))
```

```
## # A tibble: 7,120 x 47
##    grappe menage country  year  hhid vague month      zae       region   milieu
##     <dbl>  <dbl> <chr>   <dbl> <dbl> <dbl> <date>     <dbl+lbl> <dbl+lb> <dbl+l>
##  1    219     12 SEN      2021 21912     2 2022-06-01  9 [Zigu~  5 [tam~ 2 [Rur~
##  2    581      5 SEN      2021 58105     2 2022-05-01  9 [Zigu~ 14 [sed~ 2 [Rur~
##  3     17      4 SEN      2021  1704     1 2021-12-01 11 [Daka~  1 [dak~ 1 [Urb~
##  4    219     14 SEN      2021 21914     2 2022-06-01  9 [Zigu~  5 [tam~ 2 [Rur~
##  5    125      9 SEN      2021 12509     1 2021-12-01  9 [Zigu~  2 [zig~ 2 [Rur~
##  6    130      9 SEN      2021 13009     2 2022-05-01  5 [Thie~  3 [dio~ 2 [Rur~
##  7    269      7 SEN      2021 26907     1 2021-11-01  7 [Kaol~  6 [kao~ 1 [Urb~
##  8    475      4 SEN      2021 47504     2 2022-05-01  3 [Sain~ 11 [mat~ 2 [Rur~
##  9    124     10 SEN      2021 12410     2 2022-06-01  9 [Zigu~  2 [zig~ 2 [Rur~
## 10    146      6 SEN      2021 14606     2 2022-05-01  5 [Thie~  3 [dio~ 2 [Rur~
## # i 7,110 more rows
## # i 37 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>, ...
```

```
# Trier les données de la base wf par la variable hhsize par ordre croissant
wf %>% arrange(hhsize)
```

```
## # A tibble: 7,120 x 47
##    grappe menage country  year  hhid vague month      zae         region  milieu
##     <dbl>  <dbl> <chr>   <dbl> <dbl> <dbl> <date>     <dbl+lbl>   <dbl+l> <dbl+l>
##  1      2      8 SEN      2021   208     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  2      2      7 SEN      2021   207     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  3      3      7 SEN      2021   307     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  4      3     16 SEN      2021   316     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  5      3      3 SEN      2021   303     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  6      3     15 SEN      2021   315     1 2021-11-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  7      4      8 SEN      2021   408     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  8      4      1 SEN      2021   401     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  9      4      5 SEN      2021   405     2 2022-07-01 11 [Dakar] 1 [dak~ 1 [Urb~
## 10      4     13 SEN      2021   413     2 2022-07-01 11 [Dakar] 1 [dak~ 1 [Urb~
## # i 7,110 more rows
## # i 37 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>, ...
```

On peut aussi trier selon plusieurs colonnes. Par expemple dabord par hhsize puis par hage

```
arrange(wf,hhsize,hage)
```

Combinée à `arrange()`, la fonction `slice()` permet, par exemple, de **sélectionner 5 ménages ayant la plus grande taille**.

```
wf %>%
  arrange(desc(hhsize)) %>%
  slice(1:5)
```

```
## # A tibble: 5 x 47
##   grappe menage country  year  hhid vague month      zae         region   milieu
##    <dbl>  <dbl> <chr>   <dbl> <dbl> <dbl> <date>     <dbl+lbl>   <dbl+lb> <dbl+l>
## 1    316      9 SEN      2021 31609     2 2022-06-01 5 [Thies-~  7 [thi~ 1 [Urb~
## 2    215      5 SEN      2021 21505     1 2021-11-01 9 [Ziguin~  5 [tam~ 1 [Urb~
## 3    349      1 SEN      2021 34901     2 2022-05-01 5 [Thies-~  8 [lou~ 2 [Rur~
## 4    441      3 SEN      2021 44103     2 2022-06-01 9 [Ziguin~ 10 [kol~ 1 [Urb~
## 5    391      3 SEN      2021 39103     2 2022-05-01 7 [Kaolac~  9 [fat~ 2 [Rur~
## # i 37 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>,
## #   pcexp <dbl>, zzae <dbl>, zref <dbl>, def_spa <dbl>, def_temp <dbl>, ...
```

3.3.5 mutate

La fonction `mutate()` permet de **créer de nouvelles colonnes** dans un tableau de données, généralement à partir de **variables existantes**.

Supposons que l'on souhaite calculer la dépense totale par adulte équivalent dans le ménage à partir des variables `dtot` (dépense totale du ménage) et `eqadu1` (nombre d'adultes équivalents selon la méthode FAO) :

```
wf <- wf %>%
  mutate(depense_par_eq = dtot / eqadu1)

# Afficher les premières lignes avec la nouvelle variable
select(wf, dtot, eqadu1, depense_par_eq) %>% head()
```

```
## # A tibble: 6 x 3
##       dtot eqadu1 depense_par_eq
##      <dbl>  <dbl>          <dbl>
## 1 2288874.  2.28        1003892.
## 2 1806940.  1.66        1088518.
## 3 3398241.  2.82        1205050.
## 4 4846636.  4           1211659.
## 5 2546811.  0.790       3223812.
## 6 5647738.  3.83        1474605.
```

3.2 Enchaîner les opérations avec le pipe

Quand on manipule un tableau de données, il est très fréquent d'enchaîner plusieurs opérations.
On va par exemple filtrer pour extraire une sous-population, sélectionner des colonnes puis trier selon une variable.

Pour simplifier et améliorer encore la lisibilité du code, on va utiliser un nouvel opérateur, baptisé *pipe*.
Le pipe se note `%>%` ou encore `|>`, et son fonctionnement est le suivant :
si j'exécute `expr %>% f`, alors le résultat de l'expression `expr`, à gauche du pipe, sera passé comme premier argument à la fonction `f`, à droite du pipe,
ce qui revient à exécuter `f(expr)`.

Supposons que l'on veuille effectuer les opérations suivantes sur la base `wf` :

1. **Filtrer** les ménages dont le chef a plus de 60 ans (`hage > 60`)
2. **Sélectionner** uniquement les colonnes `hhid`, `hage` et `pcexp`
3. **Trier** ces ménages par niveau de bien-être (`pcexp`) décroissant

```
wf %>%
  filter(hage > 60) %>%
  select(hhid, hage, pcexp) %>%
  arrange(desc(pcexp))
```

```
## # A tibble: 2,366 x 3
##     hhid  hage    pcexp
##    <dbl> <dbl>    <dbl>
##  1  2308    72 6355519
##  2  8415    68 5515465
##  3  8414    72 4854511
##  4  3003    68 4742818
##  5  6113    63 4599493
```

```
##  6  2311    73 4483286.
##  7  1608    73 4218752.
##  8  2915    62 3891998.
##  9  3106    67 3856486.
## 10 42606    69 3811420
## # i 2,356 more rows
```

3.3 Les opérations groupées

3.3.1 group_by

La fonction `group_by()` du package **dplyr** permet de **regrouper les observations** d'un jeu de données selon une ou plusieurs variables catégorielles.
Ce regroupement constitue une étape clé avant l'application de fonctions d'agrégation comme `summarise()`, `mean()`, `sum()`, `count()`, etc.

Autrement dit, `group_by()` structure les données en **sous-groupes**, sur lesquels on peut ensuite appliquer des opérations statistiques ou de transformation.

L'extrait ci-dessous montre comment regrouper les données de la base `wf` en fonction de la variable `region` :

```
wf %>% group_by(region)
```

```
## # A tibble: 7,120 x 48
## # Groups:   region [14]
##    grappe menage country  year  hhid vague month      zae         region  milieu
##     <dbl>  <dbl> <chr>   <dbl> <dbl> <dbl> <date>     <dbl+lbl>  <dbl+l> <dbl+l>
##  1      2      5 SEN      2021   205     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  2      2     15 SEN      2021   215     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  3      2      3 SEN      2021   203     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  4      2     13 SEN      2021   213     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  5      2      8 SEN      2021   208     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  6      2     16 SEN      2021   216     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  7      2      7 SEN      2021   207     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  8      2      4 SEN      2021   204     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  9      2      1 SEN      2021   201     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
## 10      2     12 SEN      2021   212     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
## # i 7,110 more rows
## # i 38 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>, ...
```

À ce stade, aucune transformation n'est encore effectuée, mais le jeu de données est désormais marqué comme regroupé selon les valeurs de region. Cela signifie que toute fonction appliquée par la suite (comme summarise()) agira séparément pour chaque groupe(région) identifié.

On peut aussi grouper selon plusieurs variables :

```
wf %>%
  group_by(region, milieu)
```

```
## # A tibble: 7,120 x 48
## # Groups:   region, milieu [28]
##    grappe menage country  year  hhid vague month      zae         region  milieu
##     <dbl>  <dbl> <chr>   <dbl> <dbl> <dbl> <date>     <dbl+lbl>   <dbl+l> <dbl+l>
##  1      2      5 SEN      2021   205     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  2      2     15 SEN      2021   215     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  3      2      3 SEN      2021   203     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  4      2     13 SEN      2021   213     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  5      2      8 SEN      2021   208     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  6      2     16 SEN      2021   216     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  7      2      7 SEN      2021   207     2 2022-06-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  8      2      4 SEN      2021   204     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
##  9      2      1 SEN      2021   201     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
## 10      2     12 SEN      2021   212     2 2022-05-01 11 [Dakar] 1 [dak~ 1 [Urb~
## # i 7,110 more rows
## # i 38 more variables: hhweight <dbl>, hhsize <dbl>, eqadu1 <dbl>,
## #   eqadu2 <dbl>, hgender <dbl+lbl>, hage <dbl>, hmstat <dbl+lbl>,
## #   hreligion <dbl+lbl>, hnation <dbl+lbl>, hethnie <dbl+lbl>, halfa <dbl+lbl>,
## #   halfa2 <dbl+lbl>, heduc <dbl+lbl>, hdiploma <dbl+lbl>, hhandig <dbl+lbl>,
## #   hactiv7j <dbl+lbl>, hactiv12m <dbl+lbl>, hbranch <dbl+lbl>,
## #   hsectins <dbl+lbl>, hcsp <dbl+lbl>, dali <dbl>, dnal <dbl>, dtot <dbl>, ...
```

Cela permet par exemple de calculer des statistiques par région et par milieu de résidence (urbain/rural).

Après certaines opérations groupées, il peut être utile d'utiliser la fonction ungroup() afin de réinitialiser la structure du tableau. Cela permet d'éviter des comportements inattendus lors des manipulations suivantes, qui pourraient sinon continuer à s'appliquer à chaque groupe.

```r
# Exemple d'utilisation
wf %>%
  group_by(region) %>%
  summarise(nb = n()) %>%
  ungroup()
```

3.3.2 summarise et count

Lors de l'analyse de données, il est souvent utile de résumer les informations en calculant des statistiques globales pour différents sous-groupes d'observations. Le package dplyr fournit deux fonctions essentielles pour ce type de tâches : summarise() (ou summarize()) et count().

3.3.2.1. summarise() – Calcul de statistiques agrégées par groupe La fonction summarise() permet de produire des résumés statistiques comme la moyenne, la somme, le maximum ou le minimum d'une variable, pour chaque sous-groupe défini par une ou plusieurs variables catégorielles. Pour cela, on utilise group_by() en amont, qui divise les données selon les groupes souhaités.

Exemple : Calcul de l'âge moyen par genre des chefs dde menages de la bases wf

```r
# Calcul de l'âge moyen par genre
wf %>%
  group_by(hgender) %>%
  summarise(age_moyen = mean(hage, na.rm = TRUE))
```

```
## # A tibble: 2 x 2
##   hgender      age_moyen
##   <dbl+lbl>        <dbl>
## 1 1 [Masculin]      54.0
## 2 2 [Féminin]       54.2
```

Dans cet exemple :

group_by(hgender) indique que les données doivent être regroupées selon le genre de l'enquêté (hgender).

summarise(age_moyen = mean(hage, na.rm = TRUE)) calcule la moyenne de l'âge (hage) pour chaque groupe.

na.rm = TRUE signifie que les valeurs manquantes (NA) seront ignorées lors du calcul, ce qui est essentiel pour éviter des résultats erronés ou manquants.

Le résultat est un tableau contenant deux colonnes : une pour chaque modalité de hgender, et une autre indiquant l'âge moyen pour chaque groupe.

3.3.2.2 count() – Comptage d'observations par groupe La fonction count() permet de compter rapidement le nombre d'observations dans chaque catégorie d'une ou plusieurs variables. C'est une version simplifiée et directe de l'enchaînement group_by() %>% summarise(n = n()).Par exemple :

```r
# Compter le nombre de lignes par genre
wf %>% count(hgender)
```

```
## # A tibble: 2 x 2
##   hgender          n
##   <dbl+lbl>    <int>
## 1 1 [Masculin]  5095
## 2 2 [Féminin]   2025
```

Ici :

count(hgender) calcule automatiquement le nombre d'observations pour chaque modalité de hgender dans le tableau wf.

Le résultat est un tableau de deux colonnes : la modalité (hgender) et le nombre d'observations (n) correspondantes.

Remarques à noter :

-Différence entre summarise() et count() :

summarise() est utilisé pour produire des statistiques calculées (moyennes, sommes, proportions, etc.).

count() est une fonction rapide pour connaître la fréquence d'apparition des modalités d'une variable.

-Importance de group_by() : Cette fonction est essentielle avec summarise() pour obtenir des résultats par groupe. Sans group_by(), summarise() calculera une statistique globale sur tout le jeu de données.

-Utilisation de na.rm = TRUE : Très important dans les fonctions comme mean(), sum(), etc., car des valeurs manquantes peuvent faire échouer le calcul ou donner un résultat NA.

-Lisibilité du code : L'utilisation du pipe (%>%) permet de rendre le code plus lisible et plus fluide, en enchaînant les opérations de manière logique.

Extensions possibles :

Il est possible de calculer plusieurs statistiques en même temps avec summarise() :

```r
wf %>%
  group_by(hgender) %>%
  summarise(
    age_moyen = mean(hage, na.rm = TRUE),
    age_median = median(hage, na.rm = TRUE),
    nb = n()
  )
```

```
## # A tibble: 2 x 4
##   hgender      age_moyen age_median    nb
##   <dbl+lbl>        <dbl>      <dbl> <int>
## 1 1 [Masculin]      54.0         53  5095
## 2 2 [Féminin]       54.2         54  2025
```

Et avec count(), on peut aussi trier les résultats :

```r
wf %>% count(hgender, sort = TRUE)
```

```
## # A tibble: 2 x 2
##   hgender          n
##   <dbl+lbl>    <int>
## 1 1 [Masculin]  5095
## 2 2 [Féminin]   2025
```

3.3.3 Grouper selon plusieurs variables

Il est tout à fait possible de regrouper vos données selon plusieurs critères. Par exemple, ici, nous avons choisi de grouper les données de la base wf selon le genre (hgender) et l'ethnie (hethnie). Le regroupement nous permet de calculer l'effectif pour chaque combinaison de ces deux variables. Cela permet de mieux comprendre la répartition des données en fonction de ces dimensions spécifiques.

```r
# Grouper par genre et par ethnie
wf %>%
  group_by(hgender, hethnie) %>%
  summarise(effectif = n())
```

```
## `summarise()` has grouped output by 'hgender'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 26 x 3
## # Groups:   hgender [2]
##    hgender      hethnie             effectif
##    <dbl+lbl>    <dbl+lbl>              <int>
##  1 1 [Masculin]  1 [Wolof/Lébou]        1553
##  2 1 [Masculin]  2 [Sérère]             600
##  3 1 [Masculin]  3 [Poular]            1819
##  4 1 [Masculin]  4 [Soninké]             77
##  5 1 [Masculin]  5 [Diola]              266
##  6 1 [Masculin]  6 [Mandingue/Socé]     328
##  7 1 [Masculin]  7 [Balante]             37
##  8 1 [Masculin]  8 [Bambara]             87
##  9 1 [Masculin]  9 [Malinké]             60
## 10 1 [Masculin] 10 [Autres ethnies]      87
## # i 16 more rows
```

3.4 Autres fontions utiles

dplyr contient d'autres fonctions utiles pour la manipulation de données.

3.4.1 sample_n, sample_frac

Si vous voulez obtenir un échantillon aléatoire de vos données, sample_n() et sample_frac() sont parfaits pour créer des sous-ensembles :

```
# Prendre 5 lignes aléatoires
wf %>% sample_n(5)

# Prendre 10% des lignes aléatoires
wf %>% sample_frac(0.1)
```

3.4.2 lead et lag Les fonctions lead() et lag() du package dplyr sont particulièrement utiles lorsqu'on travaille avec des données ordonnées, notamment dans le cadre d'une analyse de séries temporelles ou de comportements séquentiels. Elles permettent d'accéder respectivement à la valeur suivante (lead) ou à la valeur précédente (lag) d'une variable.

Autrement dit, ces fonctions offrent un moyen simple de "regarder en avant ou en arrière" dans un vecteur de données, tout en conservant l'alignement ligne par ligne.

Dans l'exemple ci-dessous, on ajoute une nouvelle colonne à la table menage, qui contiendra la valeur de la variable hhid (identifiant du ménage) à la ligne précédente :

```
Ajout_col_men<-wf %>% mutate(id_chef= lag(hhid))
```

Ici, mutate() est utilisé pour créer une nouvelle variable id_chef qui prend, pour chaque ligne, la valeur de hhid de la ligne précédente. Cela peut être utile, par exemple, pour détecter des changements d'identifiants, comparer un ménage à celui qui le précède, ou analyser des transitions.

3.4.3 tally La fonction tally() du package dplyr est un outil simple mais puissant qui permet de compter le nombre d'observations dans un tableau, en particulier après un regroupement avec group_by(). Elle offre une alternative rapide à summarise(n = n()), en produisant une table des effectifs par groupe.

Dans l'extrait suivant, on utilise tally() pour compter le nombre d'observations selon le genre (hgender) du chef de ménage :

```
wf %>%
  group_by(hgender) %>%
  tally()
```

```
## # A tibble: 2 x 2
##   hgender          n
##   <dbl+lbl>    <int>
## 1 1 [Masculin]  5095
## 2 2 [Féminin]   2025
```

Ici, group_by(hgender) sert à regrouper les données par genre, puis tally() compte le nombre de lignes (c'est-à-dire d'individus ou de ménages) dans chaque groupe.

Remarques

-tally() est équivalent dans une syntaxe plus concise. à :

```r
wf %>%
  group_by(hgender) %>%
summarise(n = n())
```

```
## # A tibble: 2 x 2
##   hgender          n
##   <dbl+lbl>    <int>
## 1 1 [Masculin]  5095
## 2 2 [Féminin]   2025
```

-Si l'on veux pondérer le comptage avec une variable (ex. un poids d'enquête), tally() accepte l'argument wt = :

```r
wf %>%
  group_by(milieu) %>%
  tally(wt = hhweight)
```

```
## # A tibble: 2 x 2
##   milieu            n
##   <dbl+lbl>     <dbl>
## 1 1 [Urbain] 1150181.
## 2 2 [Rural]   967592.
```

Le résultat produit une nouvelle colonne appelée n, représentant le nombre de cas dans chaque groupe.

3.4.5 distinct La fonction distinct() du package dplyr est utilisée pour éliminer les doublons dans un jeu de données. Elle permet de ne conserver que les lignes uniques, c'est-à-dire celles qui ne sont pas entièrement identiques à d'autres lignes dans la table.

Autrement dit, distinct() permet de simplifier un tableau en supprimant les répétitions inutiles, ce qui peut s'avérer très utile lors du nettoyage des données.

Dans l'exemple suivant, on applique distinct() à l'ensemble du tableau menage pour en extraire uniquement les lignes uniques :

```r
men_lig_uni<- wf %>% distinct()
```

Le nouvel objet men_lig_uni contient les mêmes colonnes que menage, mais sans les doublons. Si une ligne est répétée plusieurs fois dans le tableau d'origine, elle n'apparaîtra qu'une seule fois dans le résultat.

Par défaut, distinct() considère toutes les colonnes pour déterminer l'unicité d'une ligne.

Il est possible de spécifier une ou plusieurs colonnes pour ne conserver que les valeurs uniques sur ces variables :

```r
mr <-  wf %>%
  distinct(milieu,region)#Conserve de la combinaison unique des variables milieu et mstat
```

Cela permet, par exemple, de connaître les différentes combinaisons de régions et de milieux de résidence dans la base.

3.5 Concaténation : bind_rows et bind_cols

Lorsque vous avez plusieurs jeux de données à combiner, bind_rows() et bind_cols() sont utiles. bind_rows() empile les données verticalement (lignes), tandis que bind_cols() les combine horizontalement (colonnes).

-La fonction bind_rows() permet de combiner plusieurs DataFrames par les lignes, même s'ils n'ont pas le même nombre ou les mêmes noms de colonnes. Lors de cette opération, les colonnes sont appariées par nom, et toute colonne manquante dans un DataFrame sera remplie avec des valeurs NA dans le résultat final.

```
# Exemple Combinons les DataFrames wf et menage par lignes
base_combined_lig <- bind_rows(wf, menage)
```

-Contrairement à la fonction bind_rows,la fonction bind_cols() de dplyr nécessite que les DataFrames aient le même nombre de lignes pour les combiner par colonnes.

3.4 Jointures

Lorsque les données sont réparties dans plusieurs tableaux (ou dataframes), il est souvent nécessaire de les fusionner pour effectuer des analyses complètes. C'est le rôle des jointures (joins), qui permettent de réunir les tableaux en fonction d'une ou plusieurs clés communes (généralement un identifiant, comme hhid pour le ménage).Le package dplyr facilite cette opération grâce à des fonctions dédiées, à la fois puissantes et faciles à lire : inner_join(), left_join(), right_join() et full_join().

Principales fonctions de jointure

-inner_join() : conserve uniquement les lignes présentes dans les deux tableaux.

-left_join() : conserve toutes les lignes du tableau de gauche, et complète avec les données du tableau de droite si les clés correspondent.

-right_join() : l'inverse de left_join() : conserve toutes les lignes du tableau de droite.

-full_join() : conserve toutes les lignes des deux tableaux, même si la clé ne correspond pas (les valeurs manquantes seront remplies avec NA).

Cependant, avant de réaliser une jointure entre deux jeux de données, certaines préparations essentielles doivent être effectuées pour garantir une fusion cohérente et sans erreur. Voici quelques étapes à suivre :

-Identifier une clé de jointure fiable : Crée ou sélectionne une variable (souvent un identifiant, comme id, hhid, ou code) qui existe dans les deux bases et permet de relier chaque observation de manière unique. Cette clé doit être commune et logiquement comparable.

-Harmoniser les variables (recode ou labellisation) : Les mêmes concepts doivent être codés de manière identique dans les deux bases. Par exemple, si la variable sexe est codée 1 = Femme et 2 = Homme dans la base A, mais 1 = Homme et 2 = Femme dans la base B, il est impératif de recode ou labelliser les valeurs pour qu'elles soient uniformes avant la fusion.

-Gérer les valeurs manquantes : Repère et traite les valeurs manquantes dans les variables clés ou importantes. Une valeur manquante dans la clé de jointure peut empêcher une observation de s'apparier correctement.

-Vérifier et traiter les doublons : La clé de jointure doit être unique dans au moins une des deux bases. Si une base contient des doublons sur la clé, cela entraînera une multiplication des lignes après fusion. Utilise distinct() ou duplicated() pour détecter et corriger cela.

-Contrôler les types de données : Assure-toi que les variables sur lesquelles repose la jointure sont du même type dans les deux bases (par exemple, character vs. integer). Une différence de type peut empêcher la jointure de fonctionner correctement, même si les valeurs semblent identiques.

En résumé,une jointure réussie repose sur une préparation rigoureuse des données : clé commune fiable, codage cohérent, traitement des valeurs manquantes, suppression des doublons et cohérence des types. Ce n' est qu'après ce travail en amont que l' on peut baser a la fusion.

Exemples d'utilisation des differentes jointures: Jointure interne (inner join)

```
# Jointure interne
data_joined <- inner_join(welfare_18, wf, by = "hhid")
```

Cette commande conserve uniquement les individus qui apparaissent à la fois dans les bases welfare_18 et wf, en les fusionnant selon l'identifiant hhid.

```
# Jointure gauche
data_joined <- left_join(individu,menage, by = "hhid")
```

Ici, on garde tous les individus de la base individu, et on leur associe les informations issues de la base menage quand l'identifiant hhid est présent dans les deux. Si un individu ne trouve pas de correspondance dans menage, ses colonnes seront remplies avec NA.

```
# Jointure droite : garder tous les ménages
jointure_droite <- right_join(individu, menage, by = "hhid")
```

Ici,on garde toutes les informations de la base menage, même si certains ménages n'ont pas encore d'individu recensé.

```
# Jointure complète : garder toutes les observations des deux années
welfare_complet <- full_join(welfare_18, wf, by = "hhid")
```

Elle conserve toutes les lignes des deux bases, qu'il y ait correspondance ou non. C'est la plus exhaustive des jointures : aucune donnée n'est perdue.