7. Programmation fonctionnelle avec purrr

Malick SENE

2025-04-11

```
library(purrr)
library(haven)
library(tidyverse)
library(tidyr)
wf <- read_dta("ehcvm_welfare_sen2021.dta")
```

7. Programmation fonctionnelle avec purrr :

La programmation fonctionnelle vise à traiter les données de manière concise et expressive en appliquant des fonctions à des objets (vecteurs, listes, data frames) sans recourir à des boucles explicites. Le package **purrr** (faisant partie du tidyverse) fournit une suite de fonctions pour itérer sur vos données. Ci-dessous, nous passons en revue les principales fonctions et leurs applications.

7.1 map()

La fonction `map()` permet d'appliquer une fonction à chaque élément d'une liste ou d'un vecteur et retourne une liste des résultats. Par exemple, on peut afficher la classe de chaque colonne dans wf (data frame). Pour cela, on passe en parametre à la fonction le dataframe ainsi que la fonction appliquée.

```
# Afficher les classes des différentes colonnes
classes <- map(wf, class)
classes
```

```
## $grappe
## [1] "numeric"
##
## $menage
## [1] "numeric"
##
## $country
## [1] "character"
##
## $year
## [1] "numeric"
##
## $hhid
## [1] "numeric"
##
## $vague
```

```
## [1] "numeric"
##
## $month
## [1] "Date"
##
## $zae
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $region
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $milieu
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hhweight
## [1] "numeric"
##
## $hhsize
## [1] "numeric"
##
## $eqadu1
## [1] "numeric"
##
## $eqadu2
## [1] "numeric"
##
## $hgender
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hage
## [1] "numeric"
##
## $hmstat
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hreligion
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hnation
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hethnie
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $halfa
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $halfa2
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $heduc
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hdiploma
```

```
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hhandig
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hactiv7j
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hactiv12m
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hbranch
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hsectins
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $hcsp
## [1] "haven_labelled" "vctrs_vctr"     "double"
##
## $dali
## [1] "numeric"
##
## $dnal
## [1] "numeric"
##
## $dtot
## [1] "numeric"
##
## $pcexp
## [1] "numeric"
##
## $zzae
## [1] "numeric"
##
## $zref
## [1] "numeric"
##
## $def_spa
## [1] "numeric"
##
## $def_temp
## [1] "numeric"
##
## $def_temp_prix2021m11
## [1] "numeric"
##
## $def_temp_cpi
## [1] "numeric"
##
## $def_temp_adj
## [1] "numeric"
##
## $zali0
```

```
## [1] "numeric"
##
## $dtet
## [1] "numeric"
##
## $monthly_cpi
## [1] "numeric"
##
## $cpi2017
## [1] "numeric"
##
## $icp2017
## [1] "numeric"
##
## $dollars
## [1] "numeric"
```

7.2 map_dbl()

map_dbl() est similaire à map(), mais il force le résultat à être un vecteur numérique (double). On l'utilise par exemple pour calculer rapidement une statistique sur chacune des colonnes numériques.

Exemple : Calculer la moyenne de toutes les colonnes numériques (en ignorant les NA)

```
# Sélectionner les colonnes numériques de wf
numeric_cols <- wf %>% select(where(is.numeric))
# Calculer la moyenne de chaque colonne numérique
moyennes <- map_dbl(numeric_cols, mean, na.rm = TRUE)
moyennes
```

```
##              grappe              menage                year
##        3.007435e+02        7.186517e+00        2.021000e+03
##                hhid               vague                 zae
##        3.008154e+04        1.503230e+00        6.711517e+00
##              region              milieu            hhweight
##        6.799017e+00        1.449157e+00        2.974399e+02
##              hhsize              eqadu1              eqadu2
##        8.746770e+00        6.589008e+00        4.625000e+00
##             hgender               hage              hmstat
##        1.284410e+00        5.408132e+01        2.805197e+00
##           hreligion             hnation             hethnie
##        1.058427e+00        1.295435e+01        3.096050e+00
##               halfa              halfa2               heduc
##        5.116573e-01        4.946629e-01        2.111517e+00
##            hdiploma             hhandig            hactiv7j
##        5.622191e-01        8.918539e-02        2.066713e+00
##           hactiv12m             hbranch            hsectins
##        1.381882e+00        6.359712e+00        2.936817e+00
##                hcsp                dali                dnal
##        7.716776e+00        2.276262e+06        2.029004e+06
##                dtot               pcexp                zzae
##        4.305266e+06        6.211984e+05        3.714872e+05
##                zref             def_spa            def_temp
```

```
##         3.696655e+05         1.004928e+00         9.980761e-01
## def_temp_prix2021m11         def_temp_cpi         def_temp_adj
##         1.019655e+00         9.934770e-01         1.014037e+00
##                zali0                 dtet          monthly_cpi
##         1.962331e+05         6.415379e+05         1.195515e+02
##              cpi2017              icp2017              dollars
##         1.097094e+00         2.385777e+02         6.552606e+00
```

7.3 map_dfr()

`map_dfr()` applique une fonction à chaque élément d'une liste (ou d'un vecteur) puis combine (bind) les résultats par ligne pour renvoyer un data frame.
Le suffixe **_dfr** signifie "data frame, row-wise".

Principaux paramètres :

- **.x** : La liste (ou le vecteur) sur lequel la fonction est appliquée.
- **.f** : La fonction qui doit retourner un data frame ou un tibble pour chaque élément.
- **.id** (optionnel) : Un nom de colonne qui contiendra les noms (ou indices) d'origine de **Exemple : Résumer chaque colonne numérique avec quelques statistiques**

Dans cet exemple, nous allons appliquer une fonction qui calcule la moyenne, le minimum et le maximum pour chaque colonne numérique.
Chaque colonne est résumé(e) par un petit tibble, puis `map_dfr()` combine ces tibbles en un data frame unique avec une colonne d'identifiant.

```
summary_stats <- map_dfr(numeric_cols, function(x) {
  tibble(
    mean = mean(x, na.rm = TRUE),
    min = min(x, na.rm = TRUE),
    max = max(x, na.rm = TRUE)
  )
}, .id = "variable")
```

```
## Warning: ‘grappe$min‘ and ‘region$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 3, 5, 7, 9, and 11

## Warning: ‘grappe$max‘ and ‘region$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 3, 5, 7, 9, and 11

## Warning: ‘grappe$min‘ and ‘milieu$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1 and 2

## Warning: ‘grappe$max‘ and ‘milieu$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1 and 2

## Warning: ‘grappe$min‘ and ‘hgender$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1 and 2
```

```
## Warning: ‘grappe$max‘ and ‘hgender$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1 and 2

## Warning: ‘grappe$min‘ and ‘hmstat$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, 6, and 7

## Warning: ‘grappe$max‘ and ‘hmstat$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, 6, and 7

## Warning: ‘grappe$min‘ and ‘hreligion$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, and 5

## Warning: ‘grappe$max‘ and ‘hreligion$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, and 5

## Warning: ‘grappe$min‘ and ‘hnation$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, ..., 13, and 14

## Warning: ‘grappe$max‘ and ‘hnation$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, ..., 13, and 14

## Warning: ‘grappe$min‘ and ‘hethnie$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, ..., 12, and 13

## Warning: ‘grappe$max‘ and ‘hethnie$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, ..., 12, and 13

## Warning: ‘grappe$min‘ and ‘halfa$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1

## Warning: ‘grappe$max‘ and ‘halfa$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1

## Warning: ‘grappe$min‘ and ‘halfa2$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1

## Warning: ‘grappe$max‘ and ‘halfa2$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1
```

```
## Warning: ‘grappe$min‘ and ‘heduc$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, and 9

## Warning: ‘grappe$max‘ and ‘heduc$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, and 9

## Warning: ‘grappe$min‘ and ‘hdiploma$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 0, 1, 2, 3, 4, 5, 6, 7, ..., 9, and 10

## Warning: ‘grappe$max‘ and ‘hdiploma$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 0, 1, 2, 3, 4, 5, 6, 7, ..., 9, and 10

## Warning: ‘grappe$min‘ and ‘hhandig$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1

## Warning: ‘grappe$max‘ and ‘hhandig$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1

## Warning: ‘grappe$min‘ and ‘hactiv7j$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, and 6

## Warning: ‘grappe$max‘ and ‘hactiv7j$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, and 6

## Warning: ‘grappe$min‘ and ‘hactiv12m$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, and 4

## Warning: ‘grappe$max‘ and ‘hactiv12m$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, and 4

## Warning: ‘grappe$min‘ and ‘hbranch$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, ..., 10, and 11

## Warning: ‘grappe$max‘ and ‘hbranch$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, ..., 10, and 11

## Warning: ‘grappe$min‘ and ‘hsectins$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, and 6
```

```
## Warning: ‘grappe$max‘ and ‘hsectins$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, and 6
```

```
## Warning: ‘grappe$min‘ and ‘hcsp$min‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$min‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10
```

```
## Warning: ‘grappe$max‘ and ‘hcsp$max‘ have conflicting value labels.
## i Labels for these values will be taken from ‘grappe$max‘.
## x Values: 1, 2, 3, 4, 5, 6, 7, 8, 9, and 10
```

```
summary_stats
```

```
## # A tibble: 45 x 4
##    variable     mean min               max
##    <chr>       <dbl> <dbl+lbl>         <dbl+lbl>
##  1 grappe     301.      2 [ziguinchor]   598
##  2 menage       7.19    1 [Kédougou]      19
##  3 year      2021    2021               2021
##  4 hhid     30082.    201              59812
##  5 vague        1.50    1 [Kédougou]       2 [ziguinchor]
##  6 zae          6.71    1 [Kédougou]      11 [Dakar]
##  7 region       6.80    1 [Kédougou]      14 [sedhiou]
##  8 milieu       1.45    1 [Kédougou]       2 [ziguinchor]
##  9 hhweight   297.     17.7             3082.
## 10 hhsize       8.75    1 [Kédougou]      53
## # i 35 more rows
```

7.4 map2()

La fonction `map2()` permet d'itérer **simultanément sur deux vecteurs ou listes** et d'appliquer une fonction à chaque paire d'éléments.
Elle renvoie une **liste** (ou un vecteur si vous utilisez une variante comme `map2_dbl()` ou `map2_chr()`) contenant les résultats de l'application de la fonction à chaque couple d'éléments.

Principaux paramètres :

- **.x** et **.y** : Les deux vecteurs ou listes sur lesquels itérer en parallèle.
  Par exemple, si vous voulez itérer sur deux colonnes d'un data frame.
- **.f** : La fonction à appliquer à chaque paire d'éléments (.x

  $$i$$

  et .y

  $$i$$

).
  Cette fonction doit accepter deux arguments (ou plus, en combinant avec `...`).

Exemple : Calcul d'un produit pondéré

Dans la base de données `wf` (EHCVM),nous disposons de deux colonnes :
- `hhsize` : représentant la taille du ménage,
- `hhweight` : représentant la pondération du ménage.

On souhaite calculer pour chaque ménage le **produit de la taille et de la pondération** (par exemple, pour créer une mesure pondérée).

```
weighted_values <- map2_dbl(wf$hhsize, wf$hhweight, function(size, weight) {
  size * weight
})
# Affichage des quelques premiers résultats
head(weighted_values)
```

```
## [1] 1158.6219  772.4146 1158.6219 1931.0366  290.0678 1931.0366
```

7.5 pmap()

La fonction `pmap()` est la **généralisation de `map()` pour plus de deux arguments**.
Elle prend en entrée une liste de vecteurs (tous de même longueur) et applique une fonction aux éléments correspondants de chacun de ces vecteurs.

Principaux paramètres :

- **.l** : Une liste dont chaque élément est un vecteur.
  Tous les vecteurs doivent avoir la même longueur.
- **.f** : La fonction à appliquer aux éléments correspondants.
  Cette fonction doit accepter autant d'arguments que de vecteurs dans la liste.

Exemple : Calculer un score combiné à partir de plusieurs colonnes dans la base `wf`

Dans EHCVM, on dispose de trois colonnes : - `hhsize` : la taille du ménage, - `pcexp` : l'indicateur de bien-être par tête, - `hage` : l'âge du chef de ménage.

On souhaite créer une nouvelle variable `score` qui combine ces informations. Par exemple, on peut définir un score qui divise l'indicateur de bien-être par la taille du ménage et qui applique un coefficient multiplicateur si l'âge du chef de ménage est supérieur à 50 ans.

```
# Calculer le score avec pmap_dbl() pour obtenir un vecteur numérique
score <- pmap_dbl(
  list(wf$hhsize, wf$pcexp, wf$hage),
  function(size, pcexp, age) {
    # Si le chef du ménage a plus de 50 ans, augmenter légèrement le score
    pcexp / size * ifelse(age > 50, 1.1, 1)
  }
)

# Affichage des premiers scores calculés
head(score)
```

```
## [1]  221635.2  393679.9  329057.0  185845.7 2441456.1  196876.6
```

7.6 imap()

La fonction `imap()` fonctionne comme `map()`, mais en plus de traiter chaque élément d'un vecteur ou d'une liste,
elle fournit également l'indice ou le nom correspondant à chaque élément.
Cela est particulièrement utile lorsqu'on souhaite connaître la **position** ou l'**étiquette** associée à chaque élément lors de l'itération.

Principaux paramètres :

- **.x** : Le vecteur ou la liste sur lequel on itère.
- **.f** : La fonction à appliquer. Cette fonction reçoit généralement deux arguments :
  - Le premier correspondant à l'**élément** (la valeur)
  - Le deuxième à son **indice** (ou son nom).

Exemple : Afficher pour chaque colonne numérique son nom et sa moyenne

Dans cet exemple, nous allons extraire les colonnes numériques de notre data frame `wf` (issu de l'EHCVM) et utiliser `imap()`
pour créer un vecteur de chaînes de caractères indiquant pour chaque colonne son nom et sa moyenne.

```
# Sélectionner les colonnes numériques de wf
numeric_cols <- wf %>% select(where(is.numeric))
# Utiliser imap() pour créer un vecteur avec le nom de la colonne et sa moyenne
stats <- imap_chr(numeric_cols, function(column, name) {
  # Calcul de la moyenne en ignorant les valeurs manquantes
  mean_val <- mean(column, na.rm = TRUE)
  paste0(name, " : moyenne = ", round(mean_val, 2))
})

# Afficher le résultat
stats
```

```
##                                grappe                                menage
##             "grappe : moyenne = 300.74"              "menage : moyenne = 7.19"
##                                  year                                  hhid
##                 "year : moyenne = 2021"            "hhid : moyenne = 30081.54"
##                                 vague                                   zae
##                 "vague : moyenne = 1.5"                "zae : moyenne = 6.71"
##                                region                               milieu
##               "region : moyenne = 6.8"              "milieu : moyenne = 1.45"
##                              hhweight                              hhsize
##           "hhweight : moyenne = 297.44"              "hhsize : moyenne = 8.75"
##                              eqadu1                               eqadu2
##              "eqadu1 : moyenne = 6.59"              "eqadu2 : moyenne = 4.63"
##                              hgender                                hage
##              "hgender : moyenne = 1.28"               "hage : moyenne = 54.08"
##                               hmstat                           hreligion
##               "hmstat : moyenne = 2.81"          "hreligion : moyenne = 1.06"
##                              hnation                             hethnie
##             "hnation : moyenne = 12.95"              "hethnie : moyenne = 3.1"
##                               halfa                               halfa2
##                "halfa : moyenne = 0.51"              "halfa2 : moyenne = 0.49"
```

```
##                                   heduc                                   hdiploma
##                "heduc : moyenne = 2.11"             "hdiploma : moyenne = 0.56"
##                                  hhandig                                 hactiv7j
##              "hhandig : moyenne = 0.09"             "hactiv7j : moyenne = 2.07"
##                                 hactiv12m                                 hbranch
##            "hactiv12m : moyenne = 1.38"              "hbranch : moyenne = 6.36"
##                                 hsectins                                   hcsp
##             "hsectins : moyenne = 2.94"                 "hcsp : moyenne = 7.72"
##                                    dali                                   dnal
##            "dali : moyenne = 2276261.7"           "dnal : moyenne = 2029004.42"
##                                    dtot                                  pcexp
##           "dtot : moyenne = 4305266.12"           "pcexp : moyenne = 621198.38"
##                                    zzae                                   zref
##            "zzae : moyenne = 371487.21"             "zref : moyenne = 369665.5"
##                                 def_spa                               def_temp
##                "def_spa : moyenne = 1"                "def_temp : moyenne = 1"
##                     def_temp_prix2021m11                              def_temp_cpi
## "def_temp_prix2021m11 : moyenne = 1.02"          "def_temp_cpi : moyenne = 0.99"
##                                def_temp_adj                               zali0
##         "def_temp_adj : moyenne = 1.01"            "zali0 : moyenne = 196233.07"
##                                    dtet                           monthly_cpi
##            "dtet : moyenne = 641537.95"         "monthly_cpi : moyenne = 119.55"
##                                 cpi2017                                icp2017
##               "cpi2017 : moyenne = 1.1"              "icp2017 : moyenne = 238.58"
##                                 dollars
##              "dollars : moyenne = 6.55"
```

7.7 keep() et discard()

Les fonctions `keep()` et `discard()` du package **purrr** permettent de filtrer les éléments d'un vecteur ou d'une liste en fonction d'un prédicat (une fonction logique).

- **`keep()`** conserve les éléments pour lesquels le prédicat renvoie `TRUE`.
- **`discard()`** élimine (rejette) les éléments pour lesquels le prédicat renvoie `TRUE`.

Principaux paramètres :

- **.x** : Le vecteur ou la liste à filtrer.
- **.predicate** : Une fonction logique qui sera appliquée à chaque élément de `.x`.
  Si pour un élément donné, `.predicate` retourne `TRUE`, alors :
  - avec `keep()`, l'élément est conservé,
  - avec `discard()`, l'élément est éliminé.

Exemple : Sélectionner les colonnes en fonction du taux de valeurs non manquantes

Nous souhaitons conserver uniquement les colonnes où **le ratio de valeurs non manquantes est d'au moins 70 %**.

```
# Définir le seuil minimal de valeurs non manquantes (ici 70%)
seuil <- 0.7

# Utilisation de keep() pour ne conserver que les colonnes qui respectent ce critère
```

```r
colonnes_pertinentes <- keep(wf, function(x) {
  taux_non_na <- mean(!is.na(x))
  taux_non_na >= seuil
})

# Afficher les noms des colonnes retenues
names(colonnes_pertinentes)
```

```
##  [1] "grappe"              "menage"              "country"
##  [4] "year"                "hhid"                "vague"
##  [7] "month"               "zae"                 "region"
## [10] "milieu"              "hhweight"            "hhsize"
## [13] "eqadu1"              "eqadu2"              "hgender"
## [16] "hage"                "hmstat"              "hreligion"
## [19] "hnation"             "hethnie"             "halfa"
## [22] "halfa2"              "heduc"               "hdiploma"
## [25] "hhandig"             "hactiv7j"            "hactiv12m"
## [28] "hbranch"             "hsectins"            "hcsp"
## [31] "dali"                "dnal"                "dtot"
## [34] "pcexp"               "zzae"                "zref"
## [37] "def_spa"             "def_temp"            "def_temp_prix2021m11"
## [40] "def_temp_cpi"        "def_temp_adj"        "zali0"
## [43] "dtet"                "monthly_cpi"         "cpi2017"
## [46] "icp2017"             "dollars"
```