Séance 1 : Import et manipulation de données

Introduction à la sociologie quantitative, niveau 2

Samuel Coavoux

Deux philosophies

R est un vieux langage développé de façon décentralisé. Il a accumulé un certain nombre d'idiosyncratismes et d'incohérences.

Depuis quelques années, sous l'impulsion du développeur Hadley Wickam, une entreprise de refonte et d'harmonisation du système de manipulation des objets dans R a été entreprise, sous le nom générique de Tidyverse (parce qu'elle s'appuie sur le concept de données propres, "tidy data").

Dans ce cours, ne sont enseignées que les techniques propres au Tidyverse, plus cohérent et plus aisé. Pour avoir accés à ces fonctions, tous les scripts doivent inclure l'instruction suivante :

```
library(tidyverse)
```

Pour un cours plus détaillé sur le base-R, voir : https://github.com/scoavoux/R-ENS/blob/master/S02_import_index.pdf

Section 1

Importer des données

Sous-section 1

Localiser ses données

Notion de working directory

working directory = le répertoire de l'ordinateur considéré par la session courante de R comme sa "base". C'est là qu'il va aller chercher les fichiers lorsqu'on lui demande d'importer des données. C'est par rapport à ce répertoire qu'il définit les chemins relatifs.

`getwd()` renvoie le working directory actuel. `setwd("PATH")` permet de fixer le working directory.

ATTENTION : `setwd()` ne doit pas être utilisé avec Rstudio. En effet, le working directory est fixé, avec Rstudio, à la racine du **projet**.

```
getwd()
```

```
## [1] "/home/vvxf6766/PortKnox/Cours/2016-2019_Stat@P1"
```

Chemins relatifs et absolus

En informatique, on appelle :

- **absolu** le chemin vers un fichier qui part de la racine de l'ordinateur. Par exemple :
  - unix (linux ou mac) : /home/user/Documents/R/data.csv
  - windows : c :\\Users\user\documents\R\data.csv
- **relatif** un chemin vers un fichier qui part du **répertoire actuel** (en l'occurence du working directory) :
  - ./data/data.csv (ici, **.** signifie répertoire actuel)

Les chemins relatifs sont **toujours préférables** parce qu'ils sont plus pérennes : si vous copiez le dossier sur un autre ordinateur ou à un autre endroit, ils fonctionneront tant que la structure interne du répertoire ne change pas.

Chemins : bonnes pratiques

Nous allons faire en sorte de :

- toujours travailler dans un projet Rstudio (de sorte que le working directory est fixe)
- toujours localiser nos fichier par rapport à ce working directory
- ranger données, scripts, etc. proprement au sein de chaque projet de recherche statistique :
  - un sous-répertoire pour les données
  - un sous-répertoire par grande famille d'analyse préliminaire (un seul pour un petit projet)
  - un sous-répertoire par publication/chapitre/sous-projet d'analyse
- ne jamais dupliquer de fichier : privilégier l'usage de git pour faire des sauvegarde d'un état intermédiaire de l'analyse.

Repérer le format de données

En gros, il y a deux grandes familles de format de données, que l'on repère principalement à leur extension :

- les données en texte brut (généralement .txt, .csv, .dlm)
- les données dans un format binaire, généralement propres à un logiciel :
  - R : .RData
  - SAS : .sasb7dat
  - STATA : .dta
  - SPSS : .sav, .por
  - Excel : .xls, .xlsx

Que faut-il utiliser

.RData => fonction `load()`

Texte brut => famille de fonctions `read.*()` en base-R, `read_*()` en tidyverse

Autre format => regarder dans les packages `foreign()` (R-base), `haven()` et `readxl()` (tidyverse).

Sous-section 2

Importer des données

Import de données

La famille `read_*()` est un ensemble de fonctions pour lire les données au format texte. L'usage d'une fonction en particulier dépend du format exact des données.

Le cas le plus fréquent est que les données soient en csv :

```
d <- read_csv("./data/data.csv")
# Csv avec ";" pour séparateur
d <- read_csv2("./data/data.csv")
```

Haven

On utilise le package haven pour lire des données de format binaire.

```
library(haven)
```

- `read_stata()` STATA
- `read_spss()` SPSS
- `read_sas()` SAS

Excel

Pour lire des données directement depuis les formats binaires d'Excel, .xls et .xlsx, il existe plusieurs packages. Le plus performant pour le moment est `readxl`.

```
library(readxl)
read_excel("./data/myfile.xlsx")
# Si les données ne sont pas dans la première
# feuille du document, on peut préciser sheet
# l'argument header s'appelle col_names et il
# est TRUE par défaut
read_excel("./data/myfile.xlsx", sheet=2)
```

Noms d'objets

Les bases de données importées sont l'objet le plus fréquemment utilisé dans une analyse. Pour plus de facilité :

- donner un nom court aux bases de données, pour réduire l'effort pour les écrire. Personnellement, lorsqu'une analyse porte sur une base unique, mes données sont toujours stockée dans un objet `d` (pour data)
- s'il y a plusieurs bases, donner des noms descriptifs, clairs, et cohérents.

Exploration d'une base de données

- `str()`, `summary()`
- `dim()`, `length()`, `nrow()`, `ncol()`
- `head()`, `tail()`: afficher les cinq premières/cinq dernières lignes

Sous-section 3

Indexation

Indexation

On appelle indexation l'opération qui consiste à sélectionner un sous-ensemble restreint des valeurs d'un vecteurs :

- seulement certaines valeurs d'un vecteur unidimensionnel ;
- seulement certains lignes ou certaines colonnes d'un vecteur à deux dimensions.

Il y a trois opérateurs d'indexation en base-R. Dans ce cours, nous ne verrons que $ qui permet de sélectionner un objet au sein d'une liste (comme une variable dans un data.frame).

- `$`
- `[[`
- `[`

$

```
library(questionr)
data("hdv2003")
```

```
# hdv2003$age
head(hdv2003$age)
```

```
## [1] 28 23 59 34 71 35
```

Indexation

Pour des détails sur le fonctionnement de $, [et [[, cf. https://github.com/scoavoux/R-ENS/blob/master/S02_import_index.pdf

Section 2

Manipulation de données en tidyverse

Principe

Les packages du tidyverse simplifient notamment :

- l'import : les fonctions `read_*()`, par défaut, produisent des data frame propres dans les règles de l'art "tidy data" ;
- l'indexation de données : on déclare travailler sur un data_frame en particulier et l'on a pas besoin de répéter son nom pour appeler des variables ; on appelle les variables par leur nom, sans guillemets ;
- la manipulation de données : on utilise des *verbes d'action* pour décrire les transformations à apporter à un data_frame, par exemple `select`, `filter`, `mutate` alors que base-r requiert pour ces opérations des symboles plus abstraits ;
- l'enchaînement d'actions, grâce à %>%.

Magrittr

Dans les langages de shell (`sh`, `bash`, `zsh`, etc.), le signe | est appelé "pipe" (tuyau). Il permet d'enchaîner plusieurs fonctions en passant le résultat de la fonction de gauche comme premier argument de la fonction de droite.

Le package, `magrittr` ("Ceci n'est pas un pipe") contient principalement la fonction %>%, adaptant le "pipe" dans R.

Sous R-Studio, le raccourci clavier ctrl+shift+M insert un pipe.

Usages

En pratique, un %>% permet d'enchaîner des fonctions sans avoir besoin de stocker le résultat dans des objets intermédiaires

```
x <- 1:10
mean(x)
```

Peut être écrit :

```
1:10 %>% mean()
```

Usages du pipe : enchaîner des opérations

On peut ainsi réécrire des opérations complexes en les enchaînant chacune sur une ligne plutôt qu'en les imbriquant les unes dans les autres.

```
x <- factor(c("43", "56", "78"))
# Transformer en caractere, puis en numérique,
# puis faire la moyenne
mean(as.numeric(as.character(x)))
```

```
## [1] 59
```

```
# Alternativement :
x %>% as.character() %>%
  as.numeric() %>%
  mean()
```

```
## [1] 59
```

Sous-section 1

Sélection et recodage

Sélectionner des lignes : filter

`filter()` sélectionne des lignes à partir d'un vecteur logique (condition).

```
library(dplyr)
load("data/ACS_artists.Rdata")
dt <- tbl_df(dt) # facultatif, facilite l'impression
# Sélectionner les hommes
filter(dt, sexe == "Male", income > 100000)
```

```
## # A tibble : 7,575 x 8
##    sexe    age state     income dipl                cit       en
##    <chr> <dbl> <fct>      <dbl> <fct>               <fct>     <f
##  1 Male     52 (39) Ohi~ 125000 (20) Associate's~ (1) Bor~ <N
##  2 Male     40 (06) Cal~ 110000 (21) Bachelor's ~ (1) Bor~ <N
##  3 Male     53 (26) Mic~ 239000 (21) Bachelor's ~ (1) Bor~ <N
##  4 Male     58 (36) New~ 110000 (17) GED or alte~ (1) Bor~ <N
##  5 Male     64 (53) Was~ 281600 (22) Master's de~ (1) Bor~ <N
##  6 Male     38 (48) Tex~ 130670 (19) 1 or more y~ (1) Bor~ <N
##  7 Male     60 (36) New~ 114000 (21) Bachelor's ~ (1) Bor~ <N
##  8 Male     37 (06) Cal~ 250000 (21) Bachelor's ~ (1) Bor~ <N
```

Sélectionner des lignes : slice

`slice()` sélectionne des lignes à partir d'un index (rang).

```
# les 10 premiers individus
slice(dt, 1:10)
```

```
## # A tibble : 10 x 8
##    sexe      age state       income dipl                cit
##    <chr>   <dbl> <fct>        <dbl> <fct>               <fct>
##  1 Female     24 (37) Nort~    7600 (19) 1 or more ~ (1) Bor~
##  2 Female     42 (36) New ~   18000 (22) Master's d~ (1) Bor~
##  3 Female     64 (41) Oreg~   72100 (22) Master's d~ (1) Bor~
##  4 Female     66 (55) Wisc~   19400 (18) Some colle~ (1) Bor~
##  5 Male       50 (18) Indi~   60000 (21) Bachelor's~ (1) Bor~
##  6 Female     40 (37) Nort~   24500 (19) 1 or more ~ (1) Bor~
##  7 Female     45 (34) New ~   -9999 (21) Bachelor's~ (1) Bor~
##  8 Male       52 (08) Colo~   75000 (19) 1 or more ~ (1) Bor~
##  9 Female     42 (48) Texa~    4000 (16) Regular hi~ (1) Bor~
## 10 Male       52 (39) Ohio~  125000 (20) Associate'~ (1) Bor~
```

Sélectionner des colonnes : select

`select` permet de restreindre les colonnes d'un data.frame à un sous-ensemble. On peut également les renommer directement dans select. Toutes les colonnes qui ne sont pas nommées ne sont pas sélectionnées.

```
select(dt, sexe, age, d = dipl_c)
```

```
## # A tibble : 124,023 x 3
##    sexe      age d
##    <chr>  <dbl> <fct>
##  1 Female    24 HS degree
##  2 Female    42 Graduate ed.
##  3 Female    64 Graduate ed.
##  4 Female    66 HS degree
##  5 Male      50 College
##  6 Female    40 HS degree
##  7 Female    45 College
##  8 Male      52 HS degree
##  9 Female    42 HS degree
```

Sélectionner des colonnes : select

`select()` permet l'indexation négative. un signe - devant un nom de colonne la supprime du data.frame. On ne peut pas mixer indexation normale et négative : soit on choisit des colonnes, soit on en élimine.

```
select(dt, -dipl_c, -dipl, -cit)
```

```
## # A tibble : 124,023 x 5
##    sexe      age state                          income eng
##    <chr>  <dbl> <fct>                           <dbl> <fct>
##  1 Female    24 (37) North Carolina/NC           7600 <NA>
##  2 Female    42 (36) New York/NY                18000 <NA>
##  3 Female    64 (41) Oregon/OR                  72100 <NA>
##  4 Female    66 (55) Wisconsin/WI               19400 <NA>
##  5 Male      50 (18) Indiana/IN                 60000 <NA>
##  6 Female    40 (37) North Carolina/NC          24500 <NA>
##  7 Female    45 (34) New Jersey/NJ              -9999 <NA>
##  8 Male      52 (08) Colorado/CO                75000 <NA>
##  9 Female    42 (48) Texas/TX                    4000 <NA>
```

Sélectionner des colonnes : selecteurs

On atteint là la grande force de dplyr. On peut sélectionner des colonnes par leur nom avec trois fonctions qui prennent toutes trois comme premier argument une chaîne de characteres :

- `starts_with()`: les colonnes dont le nom commence par cette chaîne ;
- `ends_with()`: les colonnes dont le nom se termine par cette chaîne ;
- `contains()`: les colonnes dont le nom contient cette chaîne.

Cf. `?select_helpers`

Il est possible d'utiliser des *expressions régulières* pour faire des sélections plus complexes – par exemple toutes les variables qui prennent la forme "Deux lettres puis quatre chiffres". Cf. https://github.com/scoavoux/R-ENS/blob/master/S05_manipulation_donnee_avancee.pdf

Sélectionner des colonnes : selecteurs

```
# Toutes les variables qui commencent par "dipl"
select(dt, starts_with("dipl"))
```

```
## # A tibble : 124,023 x 2
##    dipl                                                  dipl_c
##    <fct>                                                 <fct>
##  1 (19) 1 or more years of college credit, no degree HS degre
##  2 (22) Master's degree                                Graduate
##  3 (22) Master's degree                                Graduate
##  4 (18) Some college, but less than 1 year             HS degre
##  5 (21) Bachelor's degree                              College
##  6 (19) 1 or more years of college credit, no degree HS degre
##  7 (21) Bachelor's degree                              College
##  8 (19) 1 or more years of college credit, no degree HS degre
##  9 (16) Regular high school diploma                    HS degre
## 10 (20) Associate's degree                             College
## # ... with 124,013 more rows
```

Sélectionner des colonnes : rename

`rename()` renvoie toutes les colonnes du data.frame, nommées ou non, mais change le nom de certaines.

```
# Renommer la variable diplome
rename(dt, diplome = dipl)
```

```
## # A tibble : 124,023 x 8
##    sexe      age state      income diplome            cit
##    <chr>  <dbl> <fct>       <dbl> <fct>              <fct>
##  1 Female    24 (37) Nor~    7600 (19) 1 or more y~ (1) Bor~
##  2 Female    42 (36) New~   18000 (22) Master's de~ (1) Bor~
##  3 Female    64 (41) Ore~   72100 (22) Master's de~ (1) Bor~
##  4 Female    66 (55) Wis~   19400 (18) Some colleg~ (1) Bor~
##  5 Male      50 (18) Ind~   60000 (21) Bachelor's ~ (1) Bor~
##  6 Female    40 (37) Nor~   24500 (19) 1 or more y~ (1) Bor~
##  7 Female    45 (34) New~   -9999 (21) Bachelor's ~ (1) Bor~
##  8 Male      52 (08) Col~   75000 (19) 1 or more y~ (1) Bor~
##  9 Female    42 (48) Tex~    4000 (16) Regular hig~ (1) Bor~
## 10 Male      52 (39) Ohi~  125000 (20) Associate's~ (1) Bor~
```

Combiner sélection et renommage

On peut renommer des colonnes directement avec `select`. Seules les colonnes mentionnées sont conservées.

```
# Sélectionner sexe et diplome, renommer diplome
select(dt, sexe, diplome = dipl)
```

```
## # A tibble : 124,023 x 2
##    sexe   diplome
##    <chr>  <fct>
##  1 Female (19) 1 or more years of college credit, no degree
##  2 Female (22) Master's degree
##  3 Female (22) Master's degree
##  4 Female (18) Some college, but less than 1 year
##  5 Male   (21) Bachelor's degree
##  6 Female (19) 1 or more years of college credit, no degree
##  7 Female (21) Bachelor's degree
##  8 Male   (19) 1 or more years of college credit, no degree
##  9 Female (16) Regular high school diploma
## 10 Male   (20) Associate's degree
```

Transformer des colonnes : mutate

`mutate()` permet de recoder ou de créer une variable à partir des variables existantes.

```
mutate(dt, sexe = factor(sexe, levels = c("Female", "Male"),
                         labels = c("Femme", "Homme")))
```

```
## # A tibble : 124,023 x 8
##    sexe    age state       income dipl                cit
##    <fct> <dbl> <fct>        <dbl> <fct>               <fct>
##  1 Femme    24 (37) Nort~    7600 (19) 1 or more y~ (1) Bor~
##  2 Femme    42 (36) New ~   18000 (22) Master's de~ (1) Bor~
##  3 Femme    64 (41) Oreg~   72100 (22) Master's de~ (1) Bor~
##  4 Femme    66 (55) Wisc~   19400 (18) Some colleg~ (1) Bor~
##  5 Homme    50 (18) Indi~   60000 (21) Bachelor's ~ (1) Bor~
##  6 Femme    40 (37) Nort~   24500 (19) 1 or more y~ (1) Bor~
##  7 Femme    45 (34) New ~   -9999 (21) Bachelor's ~ (1) Bor~
##  8 Homme    52 (08) Colo~   75000 (19) 1 or more y~ (1) Bor~
##  9 Femme    42 (48) Texa~    4000 (16) Regular hig~ (1) Bor~
## 10 Homme    52 (39) Ohio   125000 (20) Associate's  (1) Bor~
```

Transformer des colonnes : mutate

```
mutate(dt, age_classe = cut(age, 5))
```

```
## # A tibble : 124,023 x 9
##    sexe      age state      income dipl        cit     eng   dipl
##    <chr>  <dbl> <fct>       <dbl> <fct>       <fct>   <fct> <fct
##  1 Female    24 (37) No~    7600 (19) 1 or~ (1) B~ <NA>  HS d
##  2 Female    42 (36) Ne~   18000 (22) Mast~ (1) B~ <NA>  Grad
##  3 Female    64 (41) Or~   72100 (22) Mast~ (1) B~ <NA>  Grad
##  4 Female    66 (55) Wi~   19400 (18) Some~ (1) B~ <NA>  HS d
##  5 Male      50 (18) In~   60000 (21) Bach~ (1) B~ <NA>  Coll
##  6 Female    40 (37) No~   24500 (19) 1 or~ (1) B~ <NA>  HS d
##  7 Female    45 (34) Ne~   -9999 (21) Bach~ (1) B~ <NA>  Coll
##  8 Male      52 (08) Co~   75000 (19) 1 or~ (1) B~ <NA>  HS d
##  9 Female    42 (48) Te~    4000 (16) Regu~ (1) B~ <NA>  HS d
## 10 Male      52 (39) Oh~  125000 (20) Asso~ (1) B~ <NA>  Coll
## # ... with 124,013 more rows
```

Transformer des colonnes : mutate_all,

On peut transformer plusieurs variables en une seule fois avec `mutate_all()` (toutes les variables du data frame), `mutate_at()` (seulement les variables que l'on indique) et `mutate_if()` (seulement les variables qui remplissent une conditions). La fonction de recodage à appliquer doit être incluse dans `funs()` ; dans cette fonction, le point (`.`) est employé pour appeler le vecteur originel.

```
# Tout transformer en character
mutate_all(dt, funs(as.character(.)))
```

```
## # A tibble : 124,023 x 8
##    sexe   age   state       income dipl                 cit
##    <chr>  <chr> <chr>       <chr>  <chr>                <chr>
##  1 Female 24    (37) Nort~ 7600   (19) 1 or more y~ (1) Bor~
##  2 Female 42    (36) New ~ 18000  (22) Master's de~ (1) Bor~
##  3 Female 64    (41) Oreg~ 72100  (22) Master's de~ (1) Bor~
##  4 Female 66    (55) Wisc~ 19400  (18) Some colleg~ (1) Bor~
##  5 Male   50    (18) Indi~ 60000  (21) Bachelor's ~ (1) Bor~
##  6 Female 40    (37) Nort~ 24500  (19) 1 or more y~ (1) Bor~
```

Transformer des colonnes : mutate_at,

`mutate_at()` permet de sélectionner les variables à recoder par des selecteurs ; il s'agit du second argument, `.cols` qui doit être inclu dans `vars()`.

```
# Transformer dipl et dipl_c en character
mutate_at(dt, vars(starts_with("dipl")), funs(as.character(.)))
```

```
## # A tibble : 124,023 x 8
##    sexe     age state       income dipl              cit
##    <chr>  <dbl> <fct>        <dbl> <chr>             <fct>
##  1 Female    24 (37) Nort~    7600 (19) 1 or more ~ (1) Bor~
##  2 Female    42 (36) New ~   18000 (22) Master's d~ (1) Bor~
##  3 Female    64 (41) Oreg~   72100 (22) Master's d~ (1) Bor~
##  4 Female    66 (55) Wisc~   19400 (18) Some colle~ (1) Bor~
##  5 Male      50 (18) Indi~   60000 (21) Bachelor's~ (1) Bor~
##  6 Female    40 (37) Nort~   24500 (19) 1 or more ~ (1) Bor~
##  7 Female    45 (34) New ~   -9999 (21) Bachelor's~ (1) Bor~
##  8 Male      52 (08) Colo~   75000 (19) 1 or more ~ (1) Bor~
##  9 Female    42 (48) Texa~    4000 (16) Regular hi~ (1) Bor~
```

Transformer des colonnes : mutate_if,

`mutate_if()` permet de sélectionner les variables à recoder ; il s'agit du second argument, `.predicate`, qui doit être un test de condition.

```
# Transformer les factor en character
mutate_if(dt, is.factor, funs(as.character(.)))
```

```
## # A tibble : 124,023 x 8
##    sexe      age state       income dipl              cit
##    <chr>  <dbl> <chr>        <dbl> <chr>             <chr>
##  1 Female    24 (37) Nort~    7600 (19) 1 or more ~ (1) Bor~
##  2 Female    42 (36) New ~   18000 (22) Master's d~ (1) Bor~
##  3 Female    64 (41) Oreg~   72100 (22) Master's d~ (1) Bor~
##  4 Female    66 (55) Wisc~   19400 (18) Some colle~ (1) Bor~
##  5 Male      50 (18) Indi~   60000 (21) Bachelor's~ (1) Bor~
##  6 Female    40 (37) Nort~   24500 (19) 1 or more ~ (1) Bor~
##  7 Female    45 (34) New ~   -9999 (21) Bachelor's~ (1) Bor~
##  8 Male      52 (08) Colo~   75000 (19) 1 or more ~ (1) Bor~
##  9 Female    42 (48) Texa~    4000 (16) Regular hi~ (1) Bor~
## 10 Male      52 (39) Ohio~  125000 (20) Associate's~ (1) Bor~
```

Sous-section 2

Groupes

group_by

`group_by()` permet de grouper les données par modalités d'une variable catégorielle pour leur appliquer des opérations. Le premier argument est toujours le data.frame que l'on exploite. Ensuite, on liste les variables de ce data.frame par lesquelles on souhaite regrouper les valeurs.

Regroupe signifie découper les individus en autant de groupe que le produit des modalités des variables indiqués. Avec une variable à k_1 modalités, k_1 groupes, avec deux variables à k_1, k_2 modalités, $k_1 * k_2$ groupes, etc.

group_by renvoie un `tbl_df` groupé, mais ne fait pas d'autres changements ; il modifie par contre le résultat des verbes suivants.

group_by et filter/slice

```
# Sélectionner l'homme et la femme
# les mieux payés
group_by(dt, sexe) %>%
  filter(income == max(income))
```

```
## # A tibble : 2 x 8
## # Groups :   sexe [2]
##   sexe     age state        income dipl              cit
##   <chr>  <dbl> <fct>         <dbl> <fct>             <fct>
## 1 Female    62 (11) Distr~ 1.47e6 (21) Bachelor'~ (4) U.S. c~
## 2 Male      80 (36) New Y~ 1.10e6 (19) 1 or more~ (1) Born i~
```

group_by et filter/slice

```
# Sélectionner l'homme et la femme
# les mieux payés dans chaque région
group_by(dt, sexe, state) %>%
  filter(income == max(income))
```

```
## # A tibble : 106 x 8
## # Groups :   sexe, state [102]
##    sexe      age state      income dipl           cit          e
##    <chr>  <dbl> <fct>       <dbl> <fct>          <fct>        <
##  1 Male      52 (34) New~ 7.25e5 (21) Bachelor~ (4) U.S. c~ (
##  2 Male      42 (23) Mai~ 3.10e5 (19) 1 or mor~ (1) Born i~ <
##  3 Female    56 (10) Del~ 1.10e5 (20) Associat~ (1) Born i~ <
##  4 Female    57 (09) Con~ 6.35e5 (21) Bachelor~ (1) Born i~ <
##  5 Male      48 (37) Nor~ 5.06e5 (21) Bachelor~ (1) Born i~ <
##  6 Female    49 (35) New~ 2.54e5 (21) Bachelor~ (1) Born i~ (
##  7 Female    34 (13) Geo~ 4.02e5 (21) Bachelor~ (1) Born i~ <
##  8 Male      55 (25) Mas~ 1.08e6 (22) Master's~ (1) Born i~ <
##  9 Female    75 (28) Mis~ 2.90e5 (18) Some col~ (1) Born i~ (
```

group_by et mutate

```
# Pour chaque individu, calculer l'écart de son salaire
# à la moyenne des salaires des personnes de même sexe
group_by(dt, sexe) %>%
  mutate(mean_inc = sum(income) / n(),
         ecart_inc = income - mean_inc) %>%
  select(sexe, income, ecart_inc,
         mean_inc)
```

```
## # A tibble : 124,023 x 4
## # Groups :   sexe [2]
##    sexe    income ecart_inc mean_inc
##    <chr>    <dbl>     <dbl>    <dbl>
##  1 Female    7600   -26178.   33778.
##  2 Female   18000   -15778.   33778.
##  3 Female   72100    38322.   33778.
##  4 Female   19400   -14378.   33778.
##  5 Male     60000     4249.   55751.
##  6 Female   24500    -9278.   33778.
```

Summarize

`summarize()` permet de réduire une base de données, groupée ou non, à des indicateurs agrégés. Particulièrement utile avec `group_by()`.

```
# Calculer des indicateurs par sexe
group_by(dt, sexe) %>%
  summarise(effectif = n(),
            inc_m = mean(income),
            inc_sd = sd(income))
```

```
## # A tibble : 2 x 4
##   sexe   effectif  inc_m inc_sd
##   <chr>     <int>  <dbl>  <dbl>
## 1 Female    61185 33778. 44219.
## 2 Male      62838 55751. 66508.
```

Section 3

Recodage

Enquête emploi en continu

```
load("./data/eec2015.RData")
```

Transformer une classe de variable

La famille de fonction `as.*()` permet de transformer un objet d'une classe à l'autre. On utilisera principalement `as.numeric()`, `as.character()`, et `as.data.frame`.

as.character()

Fonctionne avec tous les types de vecteurs unidimensionnels.

- factor -> character : chaque valeur prend le level correspondant
- numeric/integer -> character : valeur numérique devient une chaîne de caractères (1 devient "1")
- logical -> character : TRUE devient "TRUE"

as.numeric

- character -> numeric : les valeurs qui sont composées uniquement de chiffres et de séparateur de décimale sont converties en numérique ; toutes les autres valeurs deviennent NA.

```
x <- c("12", "14", "50ml")
as.numeric(x)
```

```
## Warning : NAs introduits lors de la conversion automatique
```

```
## [1] 12 14 NA
```

- logical -> numeric : `TRUE` devient `1`, `FALSE` devient `0`

as.numeric appliqué aux factor

factor -> numeric : le vecteur est converti, chaque valeur est l'index du level correspondant

```
x <- factor(c("CAP", "BEP", "BAC", "BEP"))
levels(x)
```

```
## [1] "BAC" "BEP" "CAP"
```

```
as.numeric(x)
```

```
## [1] 3 2 1 2
```

as.numeric appliqué aux factor (2)

Par conséquent, si un vecteur numérique est encodé par erreur sous la forme d'un factor, il faut faire attention à transformer le factor en character avant

```
x <- factor(c(12,17,20))
as.numeric(x)
```

```
## [1] 1 2 3
```

```
as.numeric(as.character(x))
```

```
## [1] 12 17 20
```

```
# Solution plus efficace
as.numeric(levels(x))[x]
```

```
## [1] 12 17 20
```

Découper une variable numérique en classes

On utilise `cut()` avec comme argument `x` (le vecteur à découper) et `breaks` (soit un nombre de classes d'amplitude égales, soit les limites elles-mêmes).

```
head(cut(eec$HHCE, breaks = 6))
```

```
## [1] (30,40] (40,50] <NA>    <NA>    <NA>    (40,50]
## Levels : (-0.06,10] (10,20] (20,30] (30,40] (40,50] (50,60.1]
```

```
head(cut(eec$HHCE, breaks = c(0, 20, 35, 42, 70)))
```

```
## [1] (20,35] (42,70] <NA>    <NA>    <NA>    (35,42]
## Levels : (0,20] (20,35] (35,42] (42,70]
```

cut()

Par défaut, l'intervalle est fermé à droite (changer avec `right = FALSE`) et exclut la valeur minimale (changer avec `include.lowest=TRUE`)

```
cut(c(0, 10, 20, 12), breaks=c(0, 10, 20))
```

```
## [1] <NA>    (0,10]  (10,20] (10,20]
## Levels : (0,10] (10,20]
```

```
cut(c(0, 10, 20, 12),
    breaks=c(0, 10, 20),
    include.lowest = TRUE,
    right = FALSE)
```

```
## [1] [0,10)  [10,20] [10,20] [10,20]
## Levels : [0,10) [10,20]
```

cut()

```
eec <- mutate(eec, HHCE = cut(HHCE,
                              breaks = c(0, 20, 35, 42, 70),
                              labels = c("0-19", "20-34",
                                         "35-41", "42-70")))
```

Recoder des valeurs

```
eec <- mutate(eec, HORAIC = ifelse(HORAIC == 4, NA, HORAIC))
```

Renommer des modalités

```
eec$SEXE <- factor(eec$SEXE,
                   levels = c("1", "2"),
                   labels = c("Hommes", "Femmes"))
```

Réordonner des modalités

L'ordre dans lequel on déclare les levels et labels est l'ordre dans lequel les levels seront stockés. Pour le modifier, on peut employer à nouveau `factor()`

```
eec <- mutate(eec,
              SEXE = factor(SEXE,
                            # Changer ordre
                            levels = c("Femmes", "Hommes")))
```

Regrouper des modalités

```
eec <- mutate(eec, DIP11 = case_when(
  DIP11 %in% c("10", "11") ~ "Bac+3 et plus",
  DIP11 %in% c("10", "11")        ~ "Bac+3 et plus",
  DIP11 %in% c("30", "31", "33")  ~ "Bac à bac+2",
  DIP11 %in% c("41", "42")        ~ "Bac",
  DIP11 == "50"                   ~ "CAP, BEP",
  DIP11 %in% c("60", "70")        ~ "BEPC, CEP",
  DIP11 == "71"                   ~ "Sans diplôme"))
```

Regrouper des modalités

```
eec <- mutate(eec,
              DIP11 = factor(DIP11,
                             levels = c("Sans diplôme",
                                        "BEPC, CEP",
                                        "CAP, BEP",
                                        "Bac",
                                        "Bac à bac+2",
                                        "Bac+3 et plus")))
```

Regrouper des modalités : avec un dictionnaire

```
dic <- c(`71` = "Sans diplôme",
         `70` = "BEPC, CEP",
         `60` = "BEPC, CEP",
         `50` = "CAP, BEP",
         `41` = "Bac",
         `42` = "Bac",
         `33` = "Bac à bac+2",
         `31` = "Bac à bac+2",
         `30` = "Bac à bac+2",
         `11` = "Bac+3 et plus",
         `10` = "Bac+3 et plus")
```

Regrouper des modalités : avec un dictionnaire

```
eec <- mutate(eec,
              diplome_d = dic[DIP11],
              diplome_d = factor(diplome_d, levels = unique(dic)
```

Concaténer deux vecteurs caractère

On utilise `paste()` pour coller deux vecteurs caractères l'un à l'autre ; l'argument `sep` permet de définir ce qui les séparera (par défaut un espace, `" "`) ; `paste0(x)` est défini comme `paste(x, sep="")`.

```
eec <- mutate(eec, dipl_sexe = paste(diplome_d, SEXE))
```

Sous-section 1

Appariements

Appariements

Un appariement est une opération qui consiste à fusionner deux bases de données à partir d'un ou plusieurs identifiants communs. En base R, on utilise le plus souvent `merge()`. C'est particulièrement utile quand les données dont on dispose consistent en un ensemble de bases dont les observations sont de nature différentes.

Dans l'exemple suivant, on a une base d'ouvrages et une base d'écrivains. Tous les ouvrages n'ont pas leur écrivains dans l'autre bases ; tous les écrivains n'ont pas d'ouvrages ; certains en ont plusieurs. Les variables ne sont pas les mêmes : on ne dispose du pays que de l'écrivain, pas de l'ouvrage. Comment réinsérer le pays dans la base ouvrages ?

```
ouvrages  <- read_csv("./data/ouvrages.csv")
```

```
## Parsed with column specification :
## cols(
##   auteur = col_character(),
##   titre = col_character(),
##   annee = col_integer()
```

Ouvrages

```
ouvrages
```

```
## # A tibble : 4 x 3
##   auteur titre                annee
##   <chr>  <chr>                <int>
## 1 Balzac La peau de chagrin  1831
## 2 Woolf  Ms. Dalloway         1925
## 3 Woolf  To the lighthouse    1927
## 4 Sand   La mare au diable    1846
```

Écrivains

```
ecrivains
```

```
## # A tibble : 3 x 3
##   nom    prenom   pays
##   <chr>  <chr>    <chr>
## 1 Balzac Honoré   FR
## 2 Woolf  Virginia UK
## 3 Proust Marcel   FR
```

Fonctions

La série de fonction `*_join()` du tidyverse permet d'appareiller deux data frame. Le choix de la fonction dépend du genre d'appareillement que l'on souhaite. Dans tous les cas, la fonction renvoie un data.frame composé des colonnes de x et de celles de y.

- `inner_join(x, y)` : toutes les observations qui sont à la fois dans x et dans y ;
- `left_join(x, y)`: toutes les observations de x, seulement les observations de y qui sont dans x,
- `right_join(x, y)`: toutes les observations de y, seulement les observations de x qui sont dans y,
- `full_join(x, y)`: toutes les observations de x et toutes celles de y ;

On utilise l'argument `by` pour préciser les colonnes identifiant chaque obseration.

Inner join

Noter la duplication des informations pour V. Woolf, comme elle a deux ouvrages.

```
inner_join(ouvrages, ecrivains, by = c("auteur" = "nom"))
```

```
## # A tibble : 3 x 5
##   auteur titre                annee prenom   pays
##   <chr>  <chr>                <int> <chr>    <chr>
## 1 Balzac La peau de chagrin   1831 Honoré   FR
## 2 Woolf  Ms. Dalloway         1925 Virginia UK
## 3 Woolf  To the lighthouse    1927 Virginia UK
```

Inner join : noms identiques

Pour simplifier, nous pouvons renommer la variable d'appareiement dans ouvrages.

```
ouvrages <- rename(ouvrages, nom = "auteur")
inner_join(ouvrages, ecrivains, by = "nom")
```

```
## # A tibble : 3 x 5
##   nom    titre                annee prenom   pays
##   <chr>  <chr>                <int> <chr>    <chr>
## 1 Balzac La peau de chagrin   1831 Honoré   FR
## 2 Woolf  Ms. Dalloway         1925 Virginia UK
## 3 Woolf  To the lighthouse    1927 Virginia UK
```

Left_join

Tous les ouvrages, pas tous les écrivains.

```
left_join(ouvrages, ecrivains, by = "nom")
```

```
## # A tibble : 4 x 5
##   nom    titre                annee prenom   pays
##   <chr>  <chr>                <int> <chr>    <chr>
## 1 Balzac La peau de chagrin   1831 Honoré   FR
## 2 Woolf  Ms. Dalloway         1925 Virginia UK
## 3 Woolf  To the lighthouse    1927 Virginia UK
## 4 Sand   La mare au diable    1846 <NA>     <NA>
```

Left_join

Tous les écrivains, pas tous les ouvrages.

```
right_join(ouvrages, ecrivains, by = "nom")
```

```
## # A tibble : 4 x 5
##   nom    titre                annee prenom   pays
##   <chr>  <chr>                <int> <chr>    <chr>
## 1 Balzac La peau de chagrin   1831 Honoré   FR
## 2 Woolf  Ms. Dalloway         1925 Virginia UK
## 3 Woolf  To the lighthouse    1927 Virginia UK
## 4 Proust <NA>                   NA Marcel   FR
```

```
## NB : strict équivalent de :
## left_join(ecrivains, ouvrages, by = "nom")
```

Full_join

Tous les ouvrages, tous les écrivains.

```
full_join(ouvrages, ecrivains, by = "nom")
```

```
## # A tibble : 5 x 5
##   nom    titre                annee prenom   pays
##   <chr>  <chr>                <int> <chr>    <chr>
## 1 Balzac La peau de chagrin   1831 Honoré   FR
## 2 Woolf  Ms. Dalloway         1925 Virginia UK
## 3 Woolf  To the lighthouse    1927 Virginia UK
## 4 Sand   La mare au diable    1846 <NA>     <NA>
## 5 Proust <NA>                   NA Marcel   FR
```

bind_rows

Si l'on a deux data.frame contenant les mêmes variables, on peut les joindre avec `bind_rows()`.

```
auteurs <- data_frame(nom = c("Woolf", "Proust"),
                      pays = c("UK", "FR"))
auteurs2 <- data_frame(nom = c("Thackeray", "Sand"),
                       pays = c("UK", "FR"))
```

bind_rows

```
bind_rows(auteurs, auteurs2)
```

```
## # A tibble : 4 x 2
##   nom       pays
##   <chr>     <chr>
## 1 Woolf     UK
## 2 Proust    FR
## 3 Thackeray UK
## 4 Sand      FR
```