Science des Données - Lab 1

Nolwenn Le Meur & Pascal Crépey

05 March, 2024

Contents

L'objectif de cet atelier est d'utiliser différentes méthodes de manipulation (importation, structure) de données pour une (meilleure) préparation à leur analyse.

Une partie de cet atelier a été construit en collaboration avec Clément Massonnaud.

1 Installation et chargement des *packages* R utiles pour l'analyse

Pour augmenter les fonctionnalités disponibles, R utilise un système de *packages* (*modules* ou *libraries*) qui peuvent être automatiquement téléchargés et installés sur votre ordinateur. La fonction `install.packages()` permet de réaliser cette opération.

Cette fonction s'occupe de récupérer la dernière version disponible du package pour votre système et d'installer les *packages* additionnels qui pourraient être nécessaires au bon fonctionnement du package que vous installez.

Tapez le code suivant:

```
?install.packages # pour obtenir l'aide sur cette fonction
#puis
install.packages("readxl")
```

Dans le cadre de cet atelier, nous aurons besoin des *packages* suivants:

- readxl (déjà installé avec la commande précédente)
- data.table
- dplyr
- ggplot2
- DataExplorer

Si vous ne les avez jamais téléchargés sur votre ordinateur depuis votre dernière installation de R et RStudio, vous devez les télécharger depuis Internet. Cette installation n'est à faire qu'une seule fois sauf mises à jour de *R*, *RStudio* ou du *packages* lui-même.

Installez ces packages.

solution

```
install.packages("readxl")
install.packages("data.table")
install.packages("dplyr")
install.packages("ggplot2")
install.packages("DataExplorer")
```

Pour utiliser ces *packages* dans une de vos sessions, vous devez les charger avec la fonction `library()`. Ce chargement doit être fait à chaque nouvelle session.

Chargez les packages.

solution

```
library("readxl")
library("data.table")
library("dplyr")
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library("ggplot2")
library("DataExplorer")
```

2 Manipuler une base de données

2.1 Charger des données

Nous allons charger la base des nouvelles admissions hospitalières quotidiennes par département. Celle-ci est disponible à l'adresse https://www.data.gouv.fr/fr/datasets/r/6fadff46-9efd-4c53-942a-54aca783c30c. Pour cela vous pouvez directement utiliser la fonction read.csv afin de charger la base via l'URL dans la variable *baseAllHosp*. Notez que le séparateur de colonnes est un ";" et qu'il faut l'indiquer à `read.csv` (cf. `?read.csv`).

Chargez les données provenant de data.gouv.fr dans la variable baseAllHosp.

solution

```r
URL <- "https://www.data.gouv.fr/fr/datasets/r/6fadff46-9efd-4c53-942a-54aca783c30c"

# data provenant de data.gouv.fr
baseAllHosp <- read.csv(URL, sep = ";")

# si vous utiliser le package data.table
#baseAllHosp <- fread(URL)
```

Vous pouvez aussi charger des données à partir d'autres sources comme des fichiers TXT ou CSV mais aussi des fichiers propriétaires come SAS, STATA ou encore Excel. Pour Excel par exemple, nous pouvons utiliser la fonction `read_xlsx()` du package *readxl*.

Pour notre atelier, vous devez charger les fichiers Geodes_SoinsCritiques_Covid19.xlsx et Geodes_Hospit_Covid19.xlsx dans les variables *baseSC* et *baseHosp*, mais attention, les données se trouvent dans la feuille (sheet) "Graphiques", les 3 premières lignes de cette feuille n'ont pas d'intérêt et il n'y a que 667 lignes à lire. Il vous faut consulter l'aide pour voir comment s'utilise la fonction `read_xlsx()`.

Chargez les données provenant d'Excel dans les variables baseSC et baseHosp.

solution

```r
?read_xlsx

baseSC <- read_xlsx("Geodes_SoinsCritiques_Covid19.xlsx",
                    skip = 3, n_max = 667, sheet = "Graphiques")
baseHosp <- read_xlsx("Geodes_Hospit_Covid19.xlsx",
                      skip = 3, n_max = 667, sheet = "Graphiques")
```

2.2 Décrire une base de données

Vous pouvez utiliser la fonction `str()` (*structure*) pour un premier aperçu du contenu de la base, c'est à dire le nom des variables, leur type et les premières valeurs de ces variables. Et ainsi vous pouvez effectuer un premier contrôle qualité du type:

- Est-ce que la baseAllHosp a les dimensions (lignes x colonnes) attendues ?
- Est-ce que le type des variables est correct pour l'analyse ?
- Est-ce que les décimales sont au bon format?

Quel est le type de la variable *jour* dans baseAllHosp ?

solution

```r
str(baseAllHosp)

# 'data.frame': 67320 obs. of  6 variables:
#  $ dep       : chr  "01" "01" "01" "01" ...
#  $ jour      : chr  "2020-03-19" "2020-03-20" "2020-03-21" "2020-03-22" ...
#  $ incid_hosp: int  1 0 3 3 14 11 13 14 14 7 ...
#  $ incid_rea : int  0 0 0 1 1 1 2 3 2 3 ...
#  $ incid_dc  : int  0 0 0 0 0 0 0 2 0 1 ...
#  $ incid_rad : int  0 1 0 1 5 4 5 2 0 3 ...

# La variable jour est de type character
```

Lorsque vous importez des données dans R, vous pouvez aussi utiliser des *packages* développés pour le *Data Profiling* avand de vous lancez dans *l'analyse exploratoire (EDA)*. Ces outils proposent des fonctions pour la caractérisation et le contrôle qualité des variables d'une base de données.

Pour exemple, vous allez utliser quelques fonctions du *package DataExplorer*.

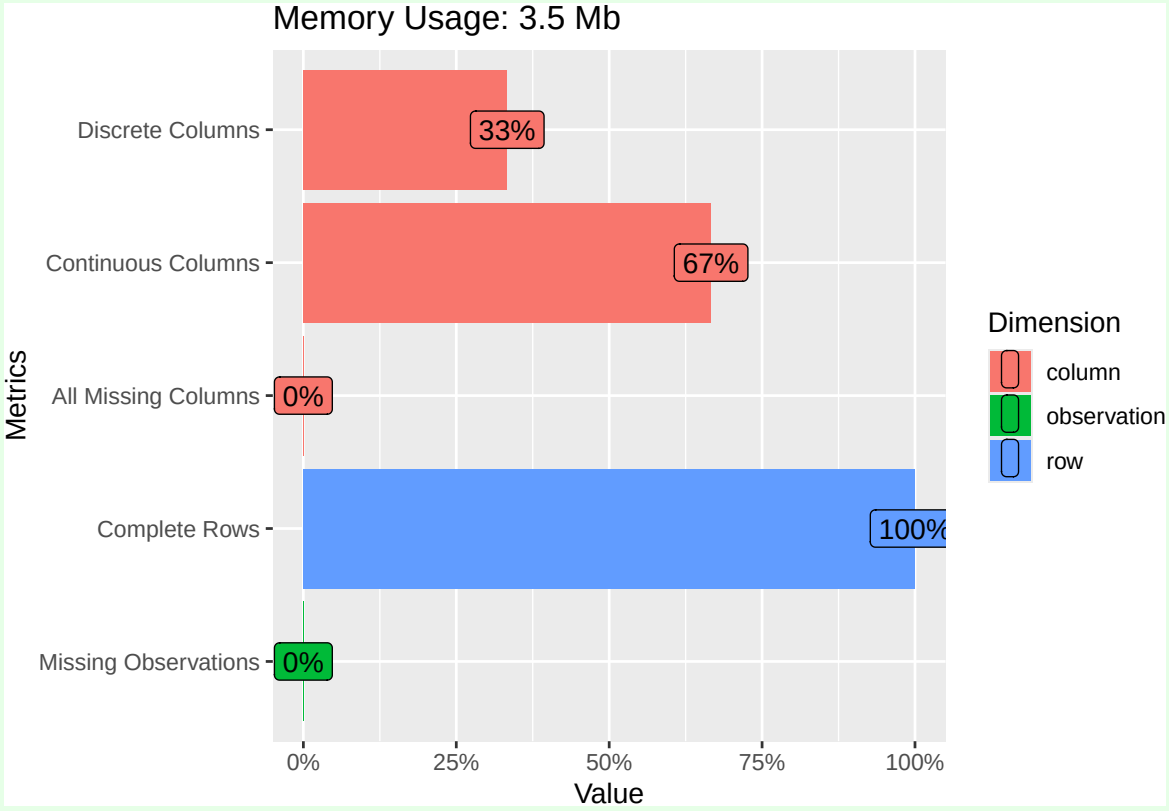
Explorer l'utilité des fonction *introduce()* et *plot_intro()* sur la base baseAllHosp

solution

```
# Utilisation de DataExplorer pour premier QA
introduce(baseAllHosp)
```

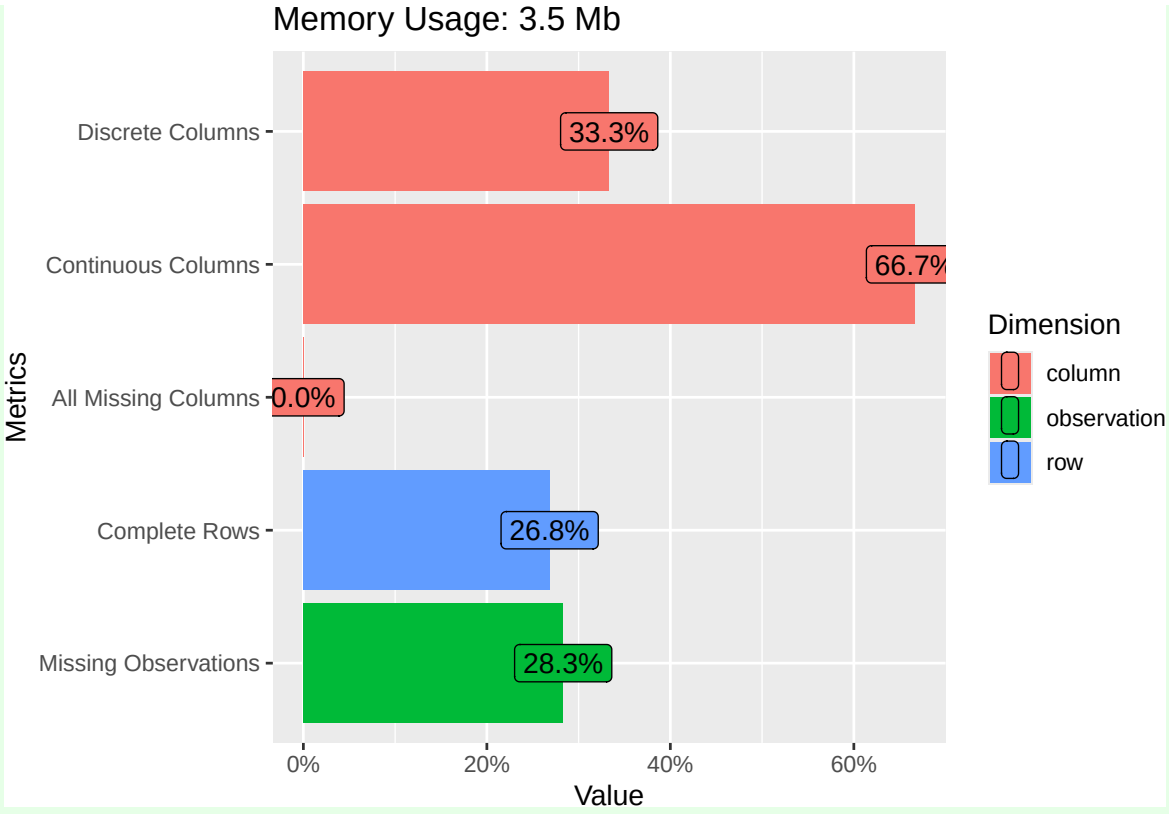
```
##     rows columns discrete_columns continuous_columns all_missing_columns
## 1 113016       6                2                  4                   0
##   total_missing_values complete_rows total_observations memory_usage
## 1                    0        113016             678096      3695272
```

```
plot_intro(baseAllHosp)
```

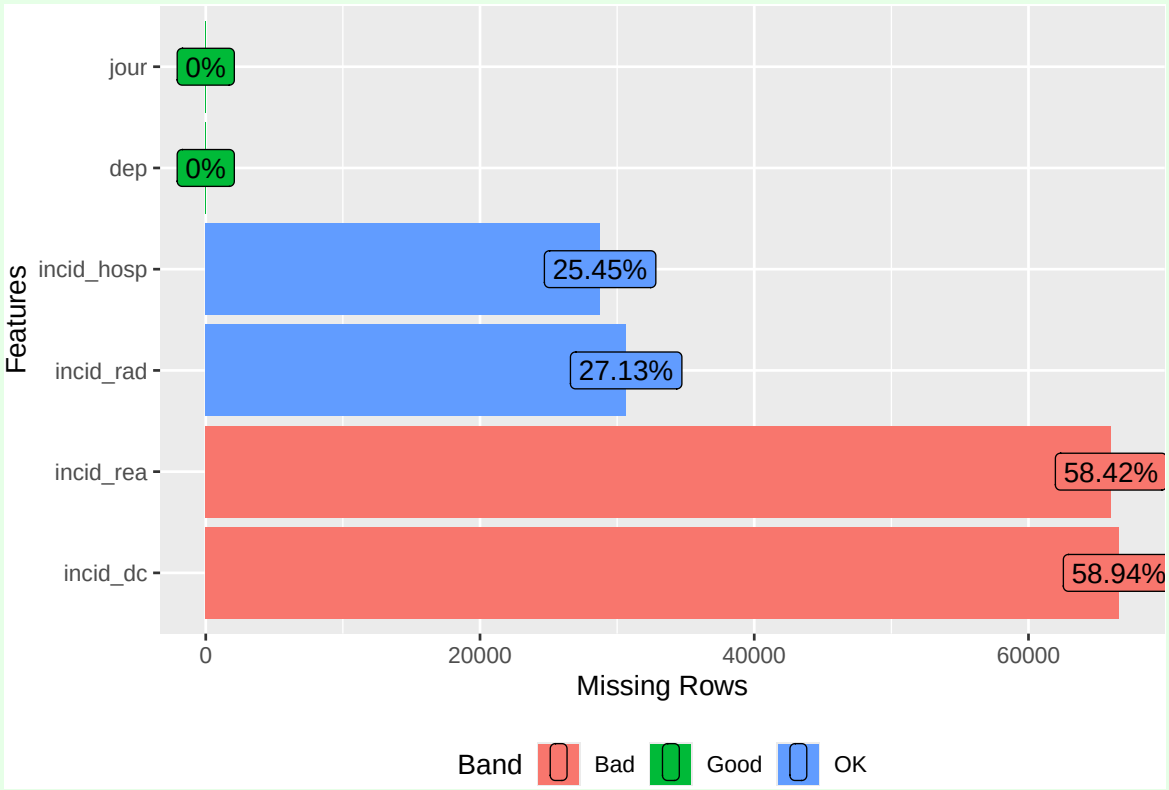



```
# Nous n'avons pas de données manquantes
# mais si tel était le cas vous pourriez savoir dans quelles variables
# avec la fonction plot_missing()

# Juste pour exemple ici nous remplacons les 0 par des NA
baseAllHosp2 <- baseAllHosp
baseAllHosp2[baseAllHosp2==0] <- NA
plot_intro(baseAllHosp2)
```




```
plot_missing(baseAllHosp2)
```



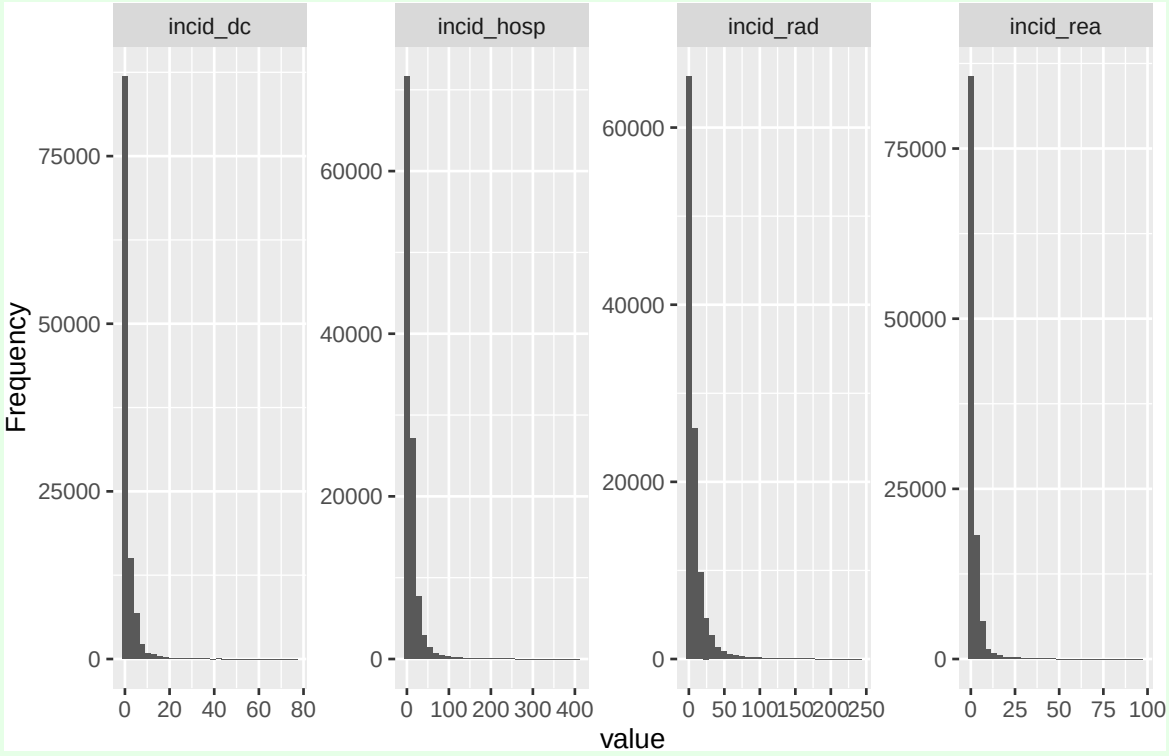
Il est souvent utile d'apprécier la distribution de vos variables quantitatives pour savoir si cela correspond

bien à ce que vous attendiez.

Utliser la fonction *plot_histogram()* sur la base baseAllHosp

solution

```
plot_histogram(baseAllHosp)
```



Enfin, *DataExplorer* possède des fonctions similaires pour les variables qualitatives et explorer la corrélation entre variables.

3 Manipuler des variables

3.1 Changer le nom des variables

Si vous utilisez `str()` sur *baseSC* et *baseHosp* vous constaterez que les noms des variables ne sont pas très explicites. Vous pouvez accéder au nom des variables grâce à la fonction `names()` et ainsi voir ou changer ainsi les noms de ces variables. Pour la base *baseHosp* il suffit de faire comme suit:

```
#pour voir les noms
names(baseHosp)
#-> [1] "Indicateurs" "Valeur"
#pour les changer
names(baseHosp) <- c("jour", "hospitalisation")
```

Faites la même chose pour la base baseSC en donnant le nom 'soins_critiques' à la variable 'Valeur'.

solution

```r
#pour voir les noms
names(baseSC)
```

```
## [1] "Indicateurs" "Valeur"
```

```r
#-> [1] "Indicateurs" "Valeur"
#pour les changer
names(baseSC) <- c("jour", "soins_critiques")
#pour baseHosp
names(baseHosp) <- c("jour", "hospitalisation")
```

3.2 Changer le type d'une variable

Comme vous pouvez le constater en utilisant *str()*, le type des variables *hospitalisation* et *soins_critiques* est *character* alors qu'il devrait être *numeric* puisque ce sont des taux pour 100 000 personnes. Nous devons donc convertir ces variables.

Le problème ici est que dans le fichier source les décimales sont marquées par des ","/virgules mais *R* fonctionne avec des "."/points. Si à la lecture du fichier nous ne l'avons pas spécifié, il faut le corriger après. Pour cela nous pouvons utliser la fonction `sub()` pour changer les "," en "." puis la fonction `as.numeric()` pour que le système comprenne que ce sont des chiffres et non pas des caractères. Pour la variable *hospitalisation* dans *baseHosp*, nous faisons ainsi:

```r
baseHosp$hospitalisation <- as.numeric(
  sub(pattern = ",",
      replacement = ".",
      x = baseHosp$hospitalisation)
  )

str(baseHosp)
# tibble [667 × 2] (S3: tbl_df/tbl/data.frame)
#  $ jour           : chr [1:667] "2020-03-07" "2020-03-08" "2020-03-09" "2020-03-10" ...
#  $ hospitalisation: num [1:667] 0 0 0 0 0 0 2.9 3.6 4.4 5.7 ...
```

Faites la même chose pour baseSC.

solution

```r
baseSC$soins_critiques = as.numeric(
  sub(pattern = ",",
      replacement = ".",
      x = baseSC$soins_critiques)
  )
```

3.3 Générer ou recoder une variable

3.3.1 Créer une variable

Pour créer une nouvelle variable (ou colonne) dans une base (*data.frame*) il suffit d'assigner une série de valeurs à une variable qui n'existe pas déjà (sinon, vous changez la variable initiale).

La variable *jour* est aussi une chaine de caractères et pas une *'vraie'* date pouvant être comprise par R. Nous allons donc générer une nouvelle variable de type *Date* à partir de cette variable. Pour cela, nous utilisons simplement la fonction `as.Date()`.

```r
baseHosp$date <- as.Date(baseHosp$jour)
```

Faites la même chose pour baseSC et baseAllHosp.

solution

```r
baseSC$date <- as.Date(baseSC$jour)

baseAllHosp$date <- as.Date(baseAllHosp$jour)
```

L'avantage du type *Date* c'est que vous pouvez ensuite faire des calculs sur ces dates, comme ajouter ou retirer des jours (+/-), récupérer le numéro de la semaine (`week()`), ou le numéro du jour dans la semaine (`wday()`).

Générer une nouvelle variable *jour_semaine* dans baseHosp

solution

```r
baseHosp$date <- as.Date(baseHosp$jour)
baseHosp$jour_semaine <- wday(baseHosp$date)
```

3.3.2 Discrétiser une variable

Nous voulons maintenant catégoriser les incidences d'hospitalisation en trois niveaux: basse, moyenne, haute. Ces niveaux doivent correspondre aux 33% des incidences les plus basses pour la catégorie "basse", aux 33% les plus hautes pour la catégorie "haute", et "moyenne" pour le reste.

Utilisez la fonction `quantile` pour trouver les bornes.

solution

```r
quantile(baseHosp$hospitalisation, probs = c(0, 0.33, 0.66, 1))
```

Utilisez la fonction `cut` pour créer une nouvelle variable catégorielle *hospit_category*.

solution

```r
bornes <- quantile(baseHosp$hospitalisation, probs = c(0, 0.33, 0.66, 1))

baseHosp$hospit_category <- cut(baseHosp$hospitalisation,
                                breaks = bornes,
                                include.lowest = TRUE,
                                labels= c("basse", "moyenne", "haute"))
```

3.4 Joindre deux bases de données

Dans un projet Data Science, nous faisons souvent appel à plusieurs sources de données que nous sommes amenées à fusionner pour intégrer l'information.

Nous allons maintenant voir comment joindre les deux bases *baseHosp* et *baseSC*.

Il y a deux façons de joindre les 2 bases :

- coller les colonnes : les bases sont "collées horizontalement", ce qui donne une base finale de dimension : `colonnes = ncol(baseSC) + ncol(baseHosp)` et `lignes = nrow(baseSC) = nrow(baseHosp)`. On

dit que la base ainsi formée est au format “wide”
- coller les lignes : les bases sont “collées verticalement”, ce qui donne une base finale de dimension : `colonnes = ncol(baseSC) = ncol(baseHosp)` et `lignes = nrow(baseSC) + nrow(baseHosp)`. On dit que la base ainsi formée est au format “long”

3.4.1 Coller les colonnes

Pour “coller” des colonnes à une base, nous pouvons utiliser la fonction `cbind()` (*c* pour *column*).

```
baseWide <- cbind(baseHosp, baseSC)
head(baseWide)
```

Cependant, ce n’est pas exactement ce que nous souhaitons car nous avons collé toutes les colonnes de baseSC sur baseHosp, et nous avons donc des colonnes dupliquées, ce qui n’est pas utile et encombre la mémoire.

Collez les colonnes qui nous intéressent, en l’occurrence la seule colonne “soins_critiques”.

solution

```
baseWide <- cbind(baseHosp, baseSC$soins_critiques)
head(baseWide)
```

```
##         jour hospitalisation       date jour_semaine baseSC$soins_critiques
## 1 2020-03-07             0,0 2020-03-07            7                      0
## 2 2020-03-08             0,0 2020-03-08            1                      0
## 3 2020-03-09             0,0 2020-03-09            2                      0
## 4 2020-03-10             0,0 2020-03-10            3                      0
## 5 2020-03-11             0,0 2020-03-11            4                      0
## 6 2020-03-12             0,0 2020-03-12            5                      0
```

Nous avons finalement bien les colonnes que nous souhaitons. **Cependant il y a un problème fondamental** avec cette méthode. Les colonnes sont collées “telles quelles”, c’est à dire que les lignes sont mises bout-à-bout dans l’ordre où elles se présentent. Or, le plus souvent, nous voulons coller les colonnes en s’assurant que chaque ligne soit alignée avec la ligne correspondante de l’autre base, par exemple s’il s’agit d’un même patient, ou comme ici, d’une même date.

Pour cela nous pourrions d’abord trier les colonnes de chaque base pour s’assurer que les lignes sont dans le même ordre. Mais c’est fastidieux et le risque d’erreur est non négligeable. C’est pour ça qu’il est *toujours* préférable de **réaliser une jointure**.

3.4.2 Réaliser une jointure

Une jointure est, comme son nom l’indique, une façon de joindre deux bases, mais en fixant des conditions sur la façon de joindre les bases. On parle de **clés de jointures** (*foreign key*).

Par exemple, cela peut nous permettre de spécifier que nous voulons “coller les colonnes”, mais en s’assurant que les lignes correspondent à la même date dans les deux bases. Concrètement, la fonction de jointure va joindre les bases ligne par ligne, en appariant les dates identiques.

Le package `dplyr` nous met à disposition tous les outils nécessaires pour réaliser tout type de jointure (c’est un peu le Brico Dépôt du data management).

La fonction pour joindre deux bases sur la base ensembliste de leur **intersection** est `inner_join()`.

```
# Jointure sur la variable "jour"
baseWide <- dplyr::inner_join(baseHosp, baseSC, by = "jour")
head(baseWide)
```

Réalisez une jointure sur la variable date.

solution

```
# Jointure sur la variable "date"
baseWide <- dplyr::inner_join(baseHosp, baseSC, by = "date")
head(baseWide)
```

```
## # A tibble: 6 x 6
##   jour.x     hospitalisation date       jour_semaine jour.y     soins_critiques
##   <chr>      <chr>           <date>            <int> <chr>                <dbl>
## 1 2020-03-07 0,0             2020-03-07            7 2020-03-07               0
## 2 2020-03-08 0,0             2020-03-08            1 2020-03-08               0
## 3 2020-03-09 0,0             2020-03-09            2 2020-03-09               0
## 4 2020-03-10 0,0             2020-03-10            3 2020-03-10               0
## 5 2020-03-11 0,0             2020-03-11            4 2020-03-11               0
## 6 2020-03-12 0,0             2020-03-12            5 2020-03-12               0
```

```
# Il est possible de joindre sur plusieurs variables à la fois
baseWide <- dplyr::inner_join(baseHosp, baseSC, by = c("jour", "date"))
head(baseWide)
```

```
## # A tibble: 6 x 5
##   jour       hospitalisation date       jour_semaine soins_critiques
##   <chr>      <chr>           <date>            <int>           <dbl>
## 1 2020-03-07 0,0             2020-03-07            7               0
## 2 2020-03-08 0,0             2020-03-08            1               0
## 3 2020-03-09 0,0             2020-03-09            2               0
## 4 2020-03-10 0,0             2020-03-10            3               0
## 5 2020-03-11 0,0             2020-03-11            4               0
## 6 2020-03-12 0,0             2020-03-12            5               0
```

Il est possible de joindre sur une variable qui a un nom différent dans chaque base

```
baseWide <- dplyr::inner_join(baseHosp, baseSC, by = c("jour" = "date"))
head(baseWide)
```

Cette dernière commande produit une erreur car les colonnes “jour” et “date” n’ont pas le type et donc ne peuvent pas être comparées pour joindre les bases.

Dans notre cas, la jointure est facile car les deux ont exactement le même nombre de lignes et chaque date est unique. Dans le cas contraire, `dplyr` dispose des fonctions `full_join()`, `left_join()`, `right_join()` selon ce que vous souhaitez garder l’entièreté des enregistrements des deux bases ou de l’une ou l’autre.

Nous vous invitons à regarder les documentations (aide et *Cheat Sheet*) de ces fonctions pour apprendre comment elles fonctionnent. Plus généralement, vous pouvez vous référer au très bon livre de Hadley Wickham, créateur de dplyr, “R for Data Science”.

3.4.3 Joindre les lignes

Puisque nous voulons coller les bases verticalement, il faut s’assurer qu’elles ont des colonnes identiques.

Vérifiez si les colonnes des deux bases sont identiques en utilisant la fonction *`identical()`*

solution

```
identical(names(baseHosp), names(baseSC))
```

```
## [1] FALSE
```

Pour savoir quelles colonnes diffèrent entre les bases, vous pouvez utiliser des opérations ensemblistes telle que la différence (`setdiff()`)

```
# Différence entre les colonnes de "baseSC" et "baseHosp"
setdiff(names(baseSC), names(baseHosp))

# Différence entre les colonnes de "baseHosp" et "baseSC"
setdiff(names(baseHosp), names(baseSC))
```

Nous voyons qu'il y a une différence sur le nom des colonnes "soins_critiques" et "hospitalisation", et que baseHosp a une colonne "jour_semaine" supplémentaire.

Nous allons redonner le même nom de "valeur" aux colonnes "soins_critiques" et "hospitalisation".

```
names(baseHosp)[names(baseHosp) == "hospitalisation"] <- "valeur"
names(baseSC)[names(baseSC) == "soins_critiques"] <- "valeur"
```

Pour la colonne "jour_semaine", nous pouvons décider, soit de la supprimer de la base "baseHosp", soit de l'ajouter à la base "baseSC".

Pour supprimer une colonne, il suffit de lui attribuer la valeur spéciale "NULL".

Supprimez la colonne "*jour_semaine*".

solution

```
baseHosp$jour_semaine <- NULL
```

On peut vérifier que maintenant les deux bases ont les mêmes colonnes

```
identical(names(baseHosp), names(baseSC))
```

Il reste une dernière étape importante avant de joindre les bases. Si nous les joignons telles quelles, nous ne pourrons plus faire la différence entre les valeurs de baseHosp et de baseSC dans la base jointe.

Il faut donc créer une variable dans chaque base qui permettra de les identifier une fois jointes.

Créez une variable "type" qui aura pour valeur "hospitalisation" pour la base "baseHosp" et "soins_critiques" pour la base "baseSC"

solution

```
baseHosp$type <- "hospitalisation"
baseSC$type <- "soins_critiques"
```

Enfin, avant de pouvoir joindre, il faut également s'assurer que les colonnes de chaque base sont du même type. Si vous essayez de joindre deux colonnes qui ont le même nom, mais qui en réalité sont de type différent, R essayera de convertir automatiquement le type d'une des colonnes, **SANS VOUS AVERTIR**, ce qui peut avoir des effets très indésirables (et donner lieu de joyeuses heures de débugage...)

Soyez toujours très vigilants sur le type de vos variables!!

Vous pouvez vérifier le type (ou la classe) d'une colonne à l'aide des fonctions "class()" ou "typeof()". Pour éviter de vérifier les colonnes une par une, vous pouvez utiliser la fonction "sapply"

```
# Vérification une à une
identical(class(baseHosp$jour), class(baseSC$jour))
identical(class(baseHosp$valeur), class(baseSC$valeur))

# Vérification automatique
```

```r
sapply(baseHosp, class)
sapply(baseSC, class)

identical(sapply(baseHosp, class), sapply(baseSC, class))
```

Nous pouvons finalement joindre les deux bases verticalement avec la fonction `rbind()` (r pour *row*)

Par convention, on dira que la base jointe est au format "long".

Joignez les deux bases au format long.

solution

```r
baseLong <- rbind(baseHosp, baseSC)
```

Vérifiez que le nombre de lignes de baseLong est bien égale à la somme des lignes des deux bases initiales.

solution

```r
identical(nrow(baseLong), nrow(baseHosp) + nrow(baseSC))
```

```
## [1] TRUE
```

Sauvegardez les bases *baseLong* et *baseAllHosp* dans les fichiers *baseLong.rds* et *baseAllHosp.rds* grâce à la fonction `saveRDS()`

solution

```r
saveRDS(baseLong, file = "baseLong.rds")

saveRDS(baseAllHosp, file = "baseAllHosp.rds")
```