Capsule 8. Regroupement des Données

Simon LaRue

14/11/2022

Introduction

Dans cette capsule, nous présenterons quelques concepts de base sur les types de données statistiques et comment ils sont représentés dans un programme R. Par la suite, nous aborderons quelques notions de regroupement des données, notamment les méthodes pour catégoriser nos variables, ainsi que les méthodes pour calculer des statistiques sur les agglomérations de nos données.

Base de données

La base de données utilisée dans cette capsule est celle de "eICU Collaborative Research Database" qui contient de l'information sur les admissions aux Unités de soins intensifs aux États-Unis durant la période de 2014-2015. Nous importons le jeu de données `apacheApsVar.csv` qui contient les variables utilisées pour calculer l' « Acute Physiology Score », un indicateur de la sévérité d'une maladie à l'admission aux soins intensifs.

```
set.seed(14112022)
library(dplyr)

# Importation des données
apache_variables <- read.csv("apacheApsVar.csv")

# Formatage des valeur manquantes
apache_variables <- na_if(apache_variables,-1)

# structure du jeu de données
str(apache_variables)
```

```
## 'data.frame':    2205 obs. of  26 variables:
##  $ apacheapsvarid    : int  92788 8893 79585 293242 154681 53115 2143574 119324 53149 4682 ...
##  $ patientunitstayid: int  141765 143870 144815 145427 147307 147784 148611 149433 149713 151179 ...
##  $ intubated         : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ vent              : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ dialysis          : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ eyes              : int  4 4 4 4 4 4 NA 4 4 4 ...
##  $ motor             : int  6 6 6 6 6 6 6 NA 6 6 ...
##  $ verbal            : int  5 5 5 5 3 5 NA 4 5 ...
##  $ meds              : int  0 0 0 0 0 0 0 NA 0 0 ...
##  $ urine             : num  NA NA NA NA NA NA NA NA NA NA ...
##  $ wbc               : num  10.2 11.7 7.9 21.1 NA 12.6 7.3 NA 9.9 49.8 ...
##  $ temperature       : num  36.2 36.4 36.7 36.2 36.8 36 36.2 NA 36.3 22.2 ...
##  $ respiratoryrate   : int  39 40 6 41 33 53 45 NA 54 52 ...
##  $ sodium            : num  139 133 141 141 NA 142 142 NA 133 139 ...
##  $ heartrate         : int  88 40 131 49 115 109 105 98 94 171 ...
##  $ meanbp            : int  108 47 61 72 107 130 108 NA 65 40 ...
##  $ ph                : num  NA NA NA NA NA 7.38 NA NA NA 7.23 ...
##  $ hematocrit        : num  37.8 34.1 36.6 40.4 NA 40.9 47.5 NA 38.7 37.7 ...
##  $ creatinine        : num  1.04 1.14 0.63 1.05 NA 0.54 0.82 NA 0.61 3.02 ...
##  $ albumin           : num  NA NA 3.6 NA NA 2.9 NA NA 3.8 1.1 ...
##  $ pao2              : num  NA NA NA NA NA 76 NA NA NA 91 ...
##  $ paco2             : num  NA NA NA NA NA 81 NA NA NA 26 ...
##  $ bun               : num  28 14 6 14 NA 17 7 NA 11 72 ...
##  $ glucose           : int  61 140 82 118 NA 151 112 NA 97 58 ...
##  $ bilirubin         : num  NA NA 0.5 NA NA 0.7 NA NA 0.5 0.2 ...
##  $ fio2              : int  NA NA NA NA NA 60 NA NA NA 28 ...
```

Principes

En statistique, nous pouvons identifier quatre principaux types de données: quantitatif discrète, quantitatif continue, qualitatif ordinale et qualitatif nominale.

Les variables quantitatives discrètes sont des entiers qui représentent généralement un dénombrement d'objets, de personnes ou encore d'événements. Par exemple, le nombre d'autobus qui passe à un arrêt de bus dans une journée serait une variable discrète.

Les variables continues sont des valeurs réelles qui peuvent prendre toutes les valeurs d'un intervalle. Par exemple, la taille d'un individu est une variable continue. D'un point de vue informatique, les variables quantitatives discrètes et continues sont représentées par des données de type entier (`"integer"`) et numérique (`"double"`) respectivement.

Les variables qualitatives peuvent être ordinales, dans quel cas l'ordre des modalités est important. Par exemple, un questionnaire de satisfaction demande si les participants sont « en désaccord », « plutôt en désaccord », « plutôt en accord » ou « en accord ». Ici, nous avons quatre modalités, valeur que peut prendre la variable. avec une certaine gradation. Nous pouvons dire que les participants ayant répondu « en accord » sont plus satisfaits que ceux « plutôt en accord », qui sont plus satisfaits que ceux « plutôt en désaccord », etc.

Une variable qualitative peut aussi être nominale, dans quel cas, il n'y aura pas d'ordre dans les modalités de la variable. Par exemple, nous pouvons penser à la couleur des yeux, celle des cheveux ou encore le genre des individus. D'un point de vue informatique, les données peuvent être représentées sous forme de chaîne de caractères (`"character"`) ou encodées avec des entiers. En R, des types d'objets intéressants pour représenter nos variables qualitatives sont les facteurs (`"factor"`).

Les facteurs vont permettre de stocker une variable qualitative sous le format d'un vecteur de nombre, mais avec un attribut de niveau (`" levels"`) qui conserve l'idée que notre variable est qualitative et non numérique. Ce format est propice à être utilisé dans des fonctions de calcul et des modèles statistiques. Pour créer un facteur, il suffit d'utiliser la fonction `factor()` sur un vecteur de données. L'argument `levels` permet de spécifier explicitement l'ordre dans lequel nous souhaitons que les facteurs soient et l'argument `ordered` permet de spécifier si notre variable est ordinale ou nominale.

```
str(apache_variables$dialysis)
```

```
##  int [1:2205] 0 0 0 0 0 0 0 0 0 0 ...
```

```
apache_variables$dialysis[67:77]
```

```
##  [1] 0 0 0 1 0 0 0 0 0 0 0
```

```
dialysis_factor <- factor(apache_variables$dialysis)
str(dialysis_factor)
```

```
##  Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
dialysis_factor[67:77]
```

```
##  [1] 0 0 0 1 0 0 0 0 0 0 0
## Levels: 0 1
```

```
dialysis_factor <- factor(apache_variables$dialysis,levels = c(0,1), ordered = TRUE)
str(dialysis_factor)
```

```
##  Ord.factor w/ 2 levels "0"<"1": 1 1 1 1 1 1 1 1 1 1 ...
```

```
dialysis_factor[67:77]
```

```
##  [1] 0 0 0 1 0 0 0 0 0 0 0
## Levels: 0 < 1
```

Catégorisation des données

mutate

Si nous voulons ajouter, modifier ou supprimer une nouvelle variable dans notre jeu de données, la fonction `mutate()` est notre meilleure amie. Cette fonction, présente dans la librairie `dplyr`, dispose de plusieurs utilités pratiques.

Nous pouvons aisément ajouter une nouvelle variable à notre jeu de données. Par exemple, nous voudrions ajouter le poids des individus en livres. Il est possible de créer une nouvelle variable, que nous pourrions nommer "poids", et lui attribuer un vecteur la représentant. Pour ajouter une nouvelle variable, il suffit d'attribuer le nom de la variable à l'objet décrivant le contenu.

```
poids_individus <- rnorm(nrow(apache_variables), mean = 161, sd = 39)
apache_variables <- apache_variables %>% mutate(poids = poids_individus)
names(apache_variables)
```

```
##  [1] "apacheapsvarid"    "patientunitstayid" "intubated"
##  [4] "vent"              "dialysis"          "eyes"
##  [7] "motor"             "verbal"            "meds"
## [10] "urine"             "wbc"               "temperature"
## [13] "respiratoryrate"   "sodium"            "heartrate"
## [16] "meanbp"            "ph"                "hematocrit"
## [19] "creatinine"        "albumin"           "pao2"
## [22] "paco2"             "bun"               "glucose"
## [25] "bilirubin"         "fio2"              "poids"
```

La fonction `mutate()` permet aussi d'effectuer facilement des opérations de conversion d'une variable à une autre. Par exemple, nous disposons du poids des individus en livres et nous voulons leur poids en kilogrammes dans notre jeu de données. Il est possible de créer une variable poids$kg qui viendra remplacer notre variable poids(lbs). Pour modifier ou ajouter une variable, il suffit d'attribuer le nom de la variable à l'expression de notre nouvelle variable.

```
head(apache_variables$poids)
```

```
## [1] 163.1304 182.0633 162.8314 131.7515 242.2162 113.6922
```

```
apache_variables <- apache_variables %>% mutate(poids_kg = poids/2.205)
head(apache_variables$poids_kg)
```

```
## [1]  73.98203  82.56838  73.84645  59.75123 109.84862  51.56110
```

Finalement, il est possible de supprimer efficacement une variable de notre jeu de données. L'argument `.keep` de la fonction permet aussi de contrôler les variables retirées ou conservées de notre jeu de données. Les options disponibles permettent de :

- "all" : conserver toute les variables,
- "used" : conserver uniquement les variables utilisées,
- "unused" : conserve seulement les variables inutilisées,
- "none" : conserve uniquement le résultat.

Pour supprimer une variable manuellement, il suffit d'attribuer le nom de la variable à `NULL`.

```
apache_variables <- apache_variables %>% mutate(poids = NULL)
names(apache_variables)
```

```
##  [1] "apacheapsvarid"    "patientunitstayid" "intubated"
##  [4] "vent"              "dialysis"          "eyes"
##  [7] "motor"             "verbal"            "meds"
## [10] "urine"             "wbc"               "temperature"
## [13] "respiratoryrate"   "sodium"            "heartrate"
## [16] "meanbp"            "ph"                "hematocrit"
## [19] "creatinine"        "albumin"           "pao2"
## [22] "paco2"             "bun"               "glucose"
## [25] "bilirubin"         "fio2"              "poids_kg"
```

Une autre manière de procéder, aurait été de supprimer le poids en lbs au même moment que la création de la variable poids en kg, en utilisant l'option "unused" dans l'argument `.keep` comme suit:

```
apache_variables <- apache_variables %>% mutate(poids_kg = poids_individus/2.205,
                                                .keep = "unused")
```

case_when

Si nous voulions créer une nouvelle variable, dont les valeurs sont conditionnelles aux valeurs ou aux modalités d'une autre, nous utiliserions des conditions logiques `if_else`. Cependant, l'utilisation des commandes de base pour les conditions logiques peut devenir lourde à travailler. La fonction `case_when()` de la librairie `dplyr` permet d'effectuer ce type de transformation tout en simplifiant le code nécessaire. Par exemple, nous disposons de la variable « âge » sous format continu et nous souhaitons la grouper sous un format catégoriel, tels que les individus âgés de moins de 30 ans sont dans le groupe jeune, ceux âgés entre 31 et 65 sont dans le groupe adulte et les plus âgés que 66 ans sont dans le groupe aîné. Il sera alors possible de créer une version catégorique de l'âge en utilisant des conditions sur l'âge continu.

Exemple

Nous allons d'abord créer une nouvelle variable avec la fonction `mutate()` et attribuer les valeurs de l'expression avec la fonction `case_when()`. Pour utiliser la fonction `case_when()`, la condition logique et la valeur à attribuer sont séparées par un tilde. Nous voulons généralement énumérer plusieurs conditions logiques avec leurs valeurs attribuées respectives. L'ordre des conditions énumérées est important. La première condition est vérifier avant la deuxième, qui est vérifiée avant la troisième et ainsi de suite. Dans cet exemple, nous allons catégoriser la variable « heartrate » : une catégorie d'intérêt avec les individus ayant un battement par minute supérieur à 80; une catégorie pour seul avec un battement par minute inférieur à 80.

```
head(apache_variables$heartrate,20)
```

```
##  [1]  88  40 131  49 115 109 105  98  94 171 100 120 113  90 100  98  96 115 108
## [20] 104
```

```
apache_variables <- apache_variables %>% mutate(heart_cat =
                          case_when(apache_variables$heartrate >= 80 ~ 1,
                                    apache_variables$heartrate < 80 ~ 0)
                          )
head(apache_variables$heart_cat,20)
```

```
##  [1] 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

Fonction d'agglomération

Les fonctions d'agglomération permettent de synthétiser l'information des sous-groupes d'individus de notre jeu de données. Autrement dit, nous produisons des statistiques sur une variable d'intérêt selon les modalités d'une autre variable de regroupement. Par exemple, nous nous intéressons à la taille des individus (variable d'intérêt) pour différentes catégories d'âge (variable de regroupement). Nous pouvons alors générer des statistiques pour chaque catégorie d'âge.

Les fonctions permettant de calculer des statistiques sur nos groupes de données incluent : les fonctions `min()` et `max()` qui sont utiles lorsque nous souhaitons connaître la plus petite valeur et la plus grande valeur d'une variable continue. La fonction `mean()` sert à calculer la moyenne, `median()` calcule la médiane et `sd()` l'écart-type échantillonnale d'une variable continue. La fonction `n()` calcule le nombre d'individus dans chaque modalité de notre variable de groupement. Une fonction raccourcie, `count()`, permet de compter simplement les effectifs des modalités d'une variable.

```
count(apache_variables, heart_cat)
```

```
##   heart_cat    n
## 1         0  499
## 2         1 1664
## 3        NA   42
```

De la librairie `dplyr`, les fonctions `summarize()` et `group_by()` combinées ensemble permettent de réaliser une agglomération sur nos données.

```
apache_variables %>%
  group_by(heart_cat) %>%
  summarise(moyenne_temperature = mean(temperature, na.rm = TRUE),
            ecart_type_glucose = sd(glucose, na.rm = TRUE),
            n = n())
```

```
## # A tibble: 3 x 4
##   heart_cat moyenne_temperature ecart_type_glucose     n
##       <dbl>               <dbl>               <dbl> <int>
## 1         0                36.3                99.8   499
## 2         1                36.5                102.  1664
## 3        NA                36.2                182.    42
```

```
# agrégation selon deux variables
apache_variables %>%
  group_by(heart_cat, dialysis) %>%
  summarise(moyenne_temperature = mean(temperature, na.rm = TRUE),
            ecart_type_glucose = sd(glucose, na.rm = TRUE),
            n = n())
```

```
## # A tibble: 5 x 5
## # Groups:   heart_cat [3]
##   heart_cat dialysis moyenne_temperature ecart_type_glucose     n
##       <dbl>    <int>               <dbl>              <dbl> <int>
## 1         0        0                36.3               101.   484
## 2         0        1                36.2                36.0   15
## 3         1        0                36.5               101.  1639
## 4         1        1                36.3               126.    55
## 5        NA        0                36.2               182.    42
```

Une fonction alternative de la librairie de base qui fait des agrégations est la fonction `aggregate()`. Nous l'utilisons en fournissant une formule R, où les éléments à gauche du tilde sont les variables d'intérêt dont nous souhaitons calculer l'information et les variables à droite sont les variables de groupement.

```
# agrégation selon une variable
aggregate(cbind(glucose, temperature) ~ heart_cat, data = apache_variables, FUN = median)
```

```
##   heart_cat glucose temperature
## 1         0     118        36.4
## 2         1     134        36.5
```

```
# aggrégation selon deux variables
aggregate(temperature ~ dialysis + heart_cat, data = apache_variables, FUN = sd)
```

```
##   dialysis heart_cat temperature
## 1        0         0   0.8117842
## 2        1         0   0.6224435
## 3        0         1   0.9246293
## 4        1         1   1.1073115
```