Création d'un jeu de données agrégées en calculant des statistiques descriptives pour une variable selon les niveaux d'une autre variable

par Chedzer-Clarc Clément et Claude-Alla Joseph

2016-05-07

Table des matières

---

1. Introduction

Dans nos calculs quotidiens, on a souvent recours à la subdivision d'un jeu de données en sous-ensembles afin d'effectuer des calculs statistiques sur les sous-ensembles obtenus. Cette façon de faire permet de tirer le maximum d'informations possibles sur le jeu de données en question. Les packages de base en R permettent de faire une telle opération. Par exemple, si dans un jeu de données, on veut calculer des statistiques descriptives

pour une variable selon les niveaux d'une autre variable, les packages de base en R offrent la fonction *aggregate* qui permet d'effectuer cette tâche. Il existe aussi d'autres packages alternatifs en R qui, une fois installés, permettent d'agréger un jeu de données. Le package *dplyr* est l'un des plus courants. Cette fiche a pour but d'expliquer comment calculer des statistiques descriptives en R dans le but d'obtenir un jeu de données agrégées. À cet effet, deux méthodes seront utilisées pour illustrer le calcul des statistiques. La première méthode utilisera des fonctions de R de base et la deuxième méthode n'utilisera que des fonctions du package alternatif *dplyr*.

2. Objectifs du travail

2.1. Objectif général

> Créer un jeu de données agrégées en calculant des statistiques descriptives pour une variable selon les niveaux d'une autre variable.

2.2. Objectifs spécifiques

- Calculer les statistiques descriptives du jeu de données par la méthode de base
- Agréger le jeu de données par la méthode alternative (utilisation du package *dplyr*)
- Comparer les deux méthodes d'agrégation

3. Méthodologie

3.1. Présentation du jeu de données `Wimbledon.men.2013`

Pour effectuer la comparaison entre les deux méthodes, un jeu de données concernant tous les matchs de tennis en simple masculin ayant eu lieu lors du tournoi de Wimbledon en 2013 sera utilisé pour faire les démontrations. Ce jeu de données contient des informations telles que : les adversaires du matchs (les variables `player1` et `player2`), la ronde de la compétition à laquelle les joueurs se sont affrontés (variable `Round`), le nombre de sets gagnés par les deux joueurs (variables `FNL.1` et `FNL.2`) etc. Pour télécharger et obtenir plus d'informations sur le jeu de donnée, allez à l'adresse suivante :

https://archive.ics.uci.edu/ml/machine-learning-databases/00300/

Pour importer le jeu de données dans R, on procède comme suit :

```
Wimbledon.men.2013 <- read.csv("Wimbledon-men-2013.csv")
```

Pour visualiser le jeu de données sous forme de tableau, on tape dans la console :

```
View(Wimbledon.men.2013)
```

Et pour visualiser les dix premières lignes (seulement 5 variables selectionnées)

```
head(Wimbledon.men.2013[,c("Player1","Player2","Round","FNL.1","FNL.2")],10)
```

```
##          Player1         Player2 Round FNL.1 FNL.2
## 1       B.Becker        A.Murray     1     0     3
## 2         J.Ward          Y-H.Lu     1     1     3
## 3        N.Mahut         J.Hajek     1     3     0
```

```
## 4      T.Robredo A.Bogomolov Jr.     1     3     0
## 5        R.Haase       M.Youzhny     1     0     3
## 6      M.Gicquel      V.Pospisil     1     0     3
## 7    A.Kuznetsov      A.Montanes     1     3     1
## 8   J.Tipsarevic       V.Troicki     1     0     3
## 9    M.Baghdatis         M.Cilic     1     0     3
## 10 K.De Schepper       P.Lorenzi     1     3     0
```

De plus, pour connaitre la structure interne du jeu de données, il suffit d'entrer le code suivant dans la console R :

```
str(Wimbledon.men.2013)
```

```
## 'data.frame':    114 obs. of  42 variables:
##  $ Player1: Factor w/ 77 levels "A.Haider-Maurer",..: 7 40 58 72 65 50 2 39 49 42 ...
##  $ Player2: Factor w/ 75 levels "A.Bedene","A.Bogomolov Jr.",..: 8 74 33 2 52 71 7 72 46 57 ...
##  $ Round  : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Result : int  0 0 1 1 0 0 1 0 0 1 ...
##  $ FNL.1  : int  0 1 3 3 0 0 3 0 0 3 ...
##  $ FNL.2  : int  3 3 0 0 3 3 1 3 3 0 ...
##  $ FSP.1  : int  59 62 72 77 68 59 63 61 61 67 ...
##  $ FSW.1  : int  29 77 44 40 61 41 56 47 31 56 ...
##  $ SSP.1  : int  41 38 28 23 32 41 37 39 39 33 ...
##  $ SSW.1  : int  14 35 10 12 15 27 21 21 16 21 ...
##  $ ACE.1  : int  5 18 17 6 7 7 21 3 4 22 ...
##  $ DBF.1  : int  1 4 3 0 2 6 3 1 5 6 ...
##  $ WNR.1  : int  26 60 41 25 32 22 56 28 20 61 ...
##  $ UFE.1  : int  18 28 18 11 29 28 32 16 18 29 ...
##  $ BPC.1  : int  5 13 8 14 2 6 16 4 1 8 ...
##  $ BPW.1  : int  1 1 5 5 0 1 4 0 1 3 ...
##  $ NPA.1  : int  28 27 26 14 29 11 21 33 14 47 ...
##  $ NPW.1  : int  19 19 17 11 20 6 15 24 9 35 ...
##  $ TPW.1  : logi  NA NA NA NA NA NA ...
##  $ ST1.1  : int  4 7 6 6 4 3 6 3 3 7 ...
##  $ ST2.1  : int  3 4 6 6 5 2 6 4 4 6 ...
##  $ ST3.1  : int  2 6 6 6 5 6 3 6 4 6 ...
##  $ ST4.1  : int  NA 6 NA NA NA NA 6 NA NA NA ...
##  $ ST5.1  : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ FSP.2  : int  57 67 70 79 67 70 73 71 70 54 ...
##  $ FSW.2  : int  39 85 34 35 53 56 59 55 45 40 ...
##  $ SSP.2  : int  43 33 30 21 33 30 27 29 30 46 ...
##  $ SSW.2  : int  20 31 14 8 17 11 14 16 16 22 ...
##  $ ACE.2  : int  11 12 4 1 9 25 7 15 16 4 ...
##  $ DBF.2  : int  2 3 0 4 3 3 8 2 2 2 ...
##  $ WNR.2  : int  38 57 24 16 40 53 33 40 41 22 ...
##  $ UFE.2  : int  16 32 13 27 26 30 28 26 19 15 ...
##  $ BPC.2  : int  10 15 1 0 21 12 9 10 6 6 ...
##  $ BPW.2  : int  5 2 0 0 3 4 2 2 4 0 ...
##  $ NPA.2  : int  23 46 19 22 44 33 11 38 11 23 ...
##  $ NPW.2  : int  17 39 12 13 30 26 10 27 8 15 ...
##  $ TPW.2  : logi  NA NA NA NA NA NA ...
##  $ ST1.2  : int  6 6 2 2 6 6 3 6 6 6 ...
##  $ ST2.2  : int  6 6 4 2 7 6 4 6 6 4 ...
##  $ ST3.2  : int  6 7 3 4 7 7 6 7 6 2 ...
##  $ ST4.2  : int  NA 7 NA NA NA NA 3 NA NA NA ...
```

```
##  $ ST5.2  : int  NA NA NA NA NA NA NA NA NA NA ...
```

Il s'agit donc d'un dataframe à 114 observations de 42 variables. Ses éléments sont des sous-objets de type vecteur. Parmi ces sous-objets de type vecteur, 2 sont de type facteur, 2 sont des vecteurs logiques et 38 sont numériques.

3.2. Calcul sur le jeu de données

À ce jeu de données, on ajoutera une nouvelle variable qui contiendra le nombre de sets dans chacun des matchs puis pour démontrer la création du jeu de données agrégées, on calculera par **Ronde** (Variable `Round`) :

1. le nombre de matchs à 3 sets,
2. le nombre de matchs à 4 sets,
3. le nombre de matchs à 5 sets et
4. le nombre moyen de sets par match ;

de façon à obtenir le résultat suivant :

Round	N3ST	N4ST	N5ST	MeanNST
1	41	13	10	3.515625
2	15	6	2	3.434783
3	6	4	2	3.666667
4	4	2	2	3.750000
5	3	0	1	3.500000
6	0	1	1	4.500000
7	1	0	0	3.000000

Ces calculs seront réalisés dans un premier temps par la méthode de base en utilisant la fonction `aggregate` pour agréger le jeu de données. Dans un second temps, on utilisera la méthode alternative avec le package `dplyr` pour effectuer ces mêmes calculs. Enfin, on effectuera une comparaison entre les deux méthodes.

4. Méthode de base en R : Fonction `aggregate`

4.1. Présentation de la fonction `aggregate`

Dans R la fonction `aggregate` se trouve dans le package `stats`, l'un des pakages de R de base. Elle permet de subdiviser un jeu de données en sous-ensembles en calculant des statistiques descriptives pour chaque sous-ensemble d'observations.

La forme générale de cette fonction est : `aggregate(x,by,FUN)` où `x`, `by` et `FUN` sont les arguments principaux. D'autres options peuvent être aussi ajoutées comme `na.rm` si on désire enlever ou non les valeurs manquantes du calcul de la statistiques descriptive et que la fonction fournie à l'argument `FUN` accepte l'argument `na.rm` en entrée. L'argument `by` permet d'inclure les combinaisons des modalités des variables qui détermineront les sous-ensembles d'observations.

L'argument `FUN` permet d'appeler la fonction qui calculera la statistique descriptive. Dans notre cas, les statistiques seront calculés par la fonction `mean` pour calculer la moyenne des sets et la fonction créée `getNmatchs` pour calculer le nombre de matchs contenant un certain nombre de sets. Celle-ci s'écrit comme suit :

```
getNmatchs <- function(y, set = 3){ sum(y == set) }
```

Dans cette fonction, l'argument `y` est un vecteur de valeurs numériques (contenant des nombres de sets pour des matchs de tennis) et l'argument `set` est un paramètre pour spécifier le nombre de sets qui nous intéresse.

4.2. Utilisation de aggregate sur Wimbledon.men.2013

Avant de commencer à utiliser `aggregate`, on doit d'abord ajouter une nouvelle variable à notre jeu de données. Cette variable contiendra le nombre de sets dans chacun des matchs (somme des variables FNL.1 et FNL.2) et sera nommée `NST`.

```
Wimbledon.men.2013$NST<-Wimbledon.men.2013$FNL.1+Wimbledon.men.2013$FNL.2
```

On visualise maintenant une partie du jeu de données avec la nouvelle variable ajoutée :

```
head(Wimbledon.men.2013[,c("Player1","Player2","Round","FNL.1","FNL.2","NST")],10)
```

```
##          Player1          Player2 Round FNL.1 FNL.2 NST
## 1       B.Becker        A.Murray     1     0     3   3
## 2         J.Ward          Y-H.Lu     1     1     3   4
## 3        N.Mahut         J.Hajek     1     3     0   3
## 4      T.Robredo A.Bogomolov Jr.     1     3     0   3
## 5        R.Haase       M.Youzhny     1     0     3   3
## 6      M.Gicquel      V.Pospisil     1     0     3   3
## 7    A.Kuznetsov      A.Montanes     1     3     1   4
## 8   J.Tipsarevic       V.Troicki     1     0     3   3
## 9    M.Baghdatis         M.Cilic     1     0     3   3
## 10 K.De Schepper       P.Lorenzi     1     3     0   3
```

On veut calculer le nombre de matchs respectivement à 3, 4 et 5 sets ainsi que le nombre moyen de sets par match en fonction de la ronde de la compétition. En faisant la correspondance à chaque argument de `aggregate` :

- `x` est la nouvelle variable créée `NST`
- `by` est la variable `Round` pour déterminer les sous-ensembles d'observations
- `FUN` est la fonction permettant de faire les calculs (`getNmatchs <- function(y, set = 3){ sum(y == set) }`et `mean`)

4.2.1. Calcul du nombre de matchs à 3 sets

Pour calculer le nombre de matchs à 3 sets, on fera appel à la fonction créée `getNmatchs` dans l'argument `FUN` de `aggregate`. L'argument `set` de `getNmatchs` sera dans ce cas égal à 3.

```
N3ST<-aggregate(Wimbledon.men.2013$NST,by=list(Wimbledon.men.2013$Round),
                FUN=getNmatchs, set=3)
```

Pour afficher le résultat, on fait :

```
print(N3ST)
```

```
##   Group.1  x
## 1       1 41
```

```
## 2       2 15
## 3       3  6
## 4       4  4
## 5       5  3
## 6       6  0
## 7       7  1
```

On constate que les noms des colonnes (variables) ne sont pas ceux qu'on voulait. Par défaut, la fonction `aggregate` donne des noms aux colonnes du jeu de données. Toutefois,on peut la forcer à nommer les colonnes comme on le désire.

```
N3ST<-aggregate(list(N3ST=Wimbledon.men.2013$NST),by=list(Round=Wimbledon.men.2013$Round),
                FUN=getNmatchs, set=3)
```

```
#Nombre de matchs à 3 sets par ronde
print(N3ST)
```

```
##   Round N3ST
## 1     1   41
## 2     2   15
## 3     3    6
## 4     4    4
## 5     5    3
## 6     6    0
## 7     7    1
```

L'ajout de la fonction `list` permet de nommer la variable de l'argument `x`. Sans cet ajout, il ne serait pas possible de nommer cette variable dans `aggregate`.

4.2.2. Calcul du nombre de matchs à 4 sets

Dans ce cas, l'argument `set` de `getNmatchs` prend la valeur 4. Ainsi, on obtient :

```
N4ST<-aggregate(list(N4ST=Wimbledon.men.2013$NST),by=list(Round=Wimbledon.men.2013$Round),
                FUN=getNmatchs, set=4)
```

```
#Nombre de matchs à 4 sets par ronde
print(N4ST)
```

```
##   Round N4ST
## 1     1   13
## 2     2    6
## 3     3    4
## 4     4    2
## 5     5    0
## 6     6    1
## 7     7    0
```

4.2.3. Calcul du nombre de matchs à 5 sets

Dans ce cas, l'argument `set` de `getNmatchs` prend la valeur 5.

```
N5ST<-aggregate(list(N5ST=Wimbledon.men.2013$NST),by=list(Round=Wimbledon.men.2013$Round),
                FUN=getNmatchs, set=5)
```

```
#Nombre de matchs à 5 sets par ronde
print(N5ST)
```

```
##   Round N5ST
## 1     1   10
## 2     2    2
## 3     3    2
## 4     4    2
## 5     5    1
## 6     6    1
## 7     7    0
```

4.2.4. Calcul du nombre moyen de sets par match

Dans ce cas, l'argument `FUN` de `aggregate` est égal à la fonction `mean`.

```
MeanNST<-aggregate(list(MeanNST=Wimbledon.men.2013$NST),by=list(Round=Wimbledon.men.2013$Round),
                   FUN=mean)
```

```
#Nombre moyen de sets par match par ronde
print(MeanNST)
```

```
##   Round  MeanNST
## 1     1 3.515625
## 2     2 3.434783
## 3     3 3.666667
## 4     4 3.750000
## 5     5 3.500000
## 6     6 4.500000
## 7     7 3.000000
```

```
#Structure des nouveaux jeux de données obtenus
str(N3ST)
```

```
## 'data.frame':    7 obs. of  2 variables:
##  $ Round: int  1 2 3 4 5 6 7
##  $ N3ST : int  41 15 6 4 3 0 1
```

```
str(N4ST)
```

```
## 'data.frame':    7 obs. of  2 variables:
##  $ Round: int  1 2 3 4 5 6 7
##  $ N4ST : int  13 6 4 2 0 1 0
```

```
str(N5ST)
```

```
## 'data.frame':    7 obs. of  2 variables:
##  $ Round: int  1 2 3 4 5 6 7
##  $ N5ST : int  10 2 2 2 1 1 0
```

```
str(MeanNST)
```

```
## 'data.frame':    7 obs. of  2 variables:
```

```
##  $ Round  : int  1 2 3 4 5 6 7
##  $ MeanNST: num  3.52 3.43 3.67 3.75 3.5 ...
```

Avec la fonction `aggregate`, on a obtenu 4 dataframes à 7 observations de 2 variables, contenant des sous-objets de type vecteur numérique.

4.2.5. Fusion des jeux de données obtenus

Pour avoir exactement le même résultat que celui recherché, il nous faut fusionner les 4 jeux de données obtenus en fonction de la variable commune `Round` de façon à avoir un jeu de données unique contenant les 4 statistiques calculées par `Round`. Pour ce faire, on va utiliser la fonction `merge` qui est conçue pour ne prendre que deux objets en arguments. Pour cela, il faudra imbriquer deux `merge` dans un autre `merge`.

```
#Fusion des jeux de données obtenus
merge(N3ST,merge(N4ST,merge(N5ST,MeanNST,by="Round"),by="Round"),by="Round")
```

```
##   Round N3ST N4ST N5ST  MeanNST
## 1     1   41   13   10 3.515625
## 2     2   15    6    2 3.434783
## 3     3    6    4    2 3.666667
## 4     4    4    2    2 3.750000
## 5     5    3    0    1 3.500000
## 6     6    0    1    1 4.500000
## 7     7    1    0    0 3.000000
```

Pour avoir plus d'informations sur la fonction `merge`, tapez `help(merge)`dans la console R.

Il faut noter qu'il aurait été possible d'obtenir les 4 statistiques désirées dans un seul appel de la fonction `aggregate`. En effet, on aurait pu donner en entrée à l'argument `FUN` une fonction créée qui calcule toutes les statistiques.

```
getNmatchs <- function(y, set = 3){ sum(y == set) }
aggregate(Wimbledon.men.2013$NST,by=list(Round=Wimbledon.men.2013$Round),
          FUN=function(y) c(N3ST=getNmatchs(y,3),N4ST=getNmatchs(y,4),
                            N5ST=getNmatchs(y,5),MeanNST=mean(y)))
```

```
##   Round    x.N3ST    x.N4ST    x.N5ST x.MeanNST
## 1     1 41.000000 13.000000 10.000000  3.515625
## 2     2 15.000000  6.000000  2.000000  3.434783
## 3     3  6.000000  4.000000  2.000000  3.666667
## 4     4  4.000000  2.000000  2.000000  3.750000
## 5     5  3.000000  0.000000  1.000000  3.500000
## 6     6  0.000000  1.000000  1.000000  4.500000
## 7     7  1.000000  0.000000  0.000000  3.000000
```

Le résultat obtenu n'a cependant pas l'allure souhaitée. Les noms des colonnes des statistiques calculées ne correspondent pas et le jeu de données est uniformisé à 6 chiffres après la virgule. Il aurait fallu une mise en forme pour obtenir le résultat tel que souhaité.

5. Méthode alternative : Package dplyr

5.1. Présentation du Package dplyr

dplyr est un package créé par Hadley Whickam et qui permet de manipuler facilement des données. D'où son surnom "A grammar of data manipulation" (Réf : https://cran.r-project.org/web/packages/dplyr/dplyr.pdf). Pour utiliser dplyr, il faut d'abord l'installer dans R. L'installation de dplyr se fait de la manière suivante :

```
#Installation de dplyr
install.packages("dplyr")
```

Il faut ensuite charger le package dans la session de travail. Pour le faire, on écrit la commande suivante :

```
library("dplyr")
```

dplyr fournit une fonction pour chaque manipulation de base des données. Parmi les fonctions de manipulation de base, on peut retenir :

- filter() et slice() qui permettent de sélectionner un sous-ensemble de lignes dans un jeu de données ;
- mutate() pour ajouter de nouvelles colonnes en fonction de colonnes existantes ;
- select() permet de sélectionner les colonnes qui nous interessent dans un jeu de données ;
- summarise() transforme un jeu de données en une seule rangée ;
- distinct() retourne les valeurs uniques dans un tableau de données ;
- Etc.

Pour plus d'informations sur les fonctions du package dplyr , consultez l'aide de R en faisant :

```
help(dplyr)
```

ou visitez le manuel de référence sur https://cran.r-project.org/web/packages/dplyr/dplyr.pdf.
A noter que pour calculer les statistiques descriptives en fonction d'un sous-ensemble d'observations, les fonctions mutate, group_by, select, summarize seront utilisées.

5.2. Fonction mutate

Pour bien illustrer l'utilisation de dplyr, on va importer de nouveau le fichier de données en R et on l'appellera Wimbledon.

```
Wimbledon <- read.csv("Wimbledon-men-2013.csv")
```

La fonction mutate permet d'ajouter de nouvelles colonnes dans un jeu de données en fonction des colonnes existantes. La forme générale de cette fonction est mutate(.data, ...), où :

- .data est le jeu de données auquel on veut ajouter les nouvelles colonnes ;
- ... correspond aux colonnes que l'on veut ajouter. Le prochain exemple permettra de mieux comprendre.

On l'utilisera donc pour ajouter la nouvelle variable NST qui est fonction des variables FNL.1 et FNL.2 (NST=FNL.1+FNL.2).

```
Wimbledon<-mutate(Wimbledon,NST=FNL.1+FNL.2)
```

5.3. Fonction `select` et `slice`

Pour afficher les 10 premières lignes d'un groupe de colonnes sélectionnées du jeu de données `Wimbledon` modifié, on utilisera les fonctions `select` et `slice`.
`select` permettra de séléctionner les colonnes qu'on veut visualiser et `slice` nous permettra de séléctionner la position des lignes qui nous intéressent.

```
select(slice(Wimbledon,1:10),Player1,Player2,Round,FNL.1,FNL.2,NST)
```

```
## # A tibble: 10 x 6
##          Player1         Player2 Round FNL.1 FNL.2   NST
##           <fctr>          <fctr> <int> <int> <int> <int>
##  1      B.Becker        A.Murray     1     0     3     3
##  2        J.Ward          Y-H.Lu     1     1     3     4
##  3       N.Mahut         J.Hajek     1     3     0     3
##  4     T.Robredo A.Bogomolov Jr.     1     3     0     3
##  5       R.Haase       M.Youzhny     1     0     3     3
##  6     M.Gicquel      V.Pospisil     1     0     3     3
##  7   A.Kuznetsov      A.Montanes     1     3     1     4
##  8  J.Tipsarevic       V.Troicki     1     0     3     3
##  9   M.Baghdatis         M.Cilic     1     0     3     3
## 10 K.De Schepper       P.Lorenzi     1     3     0     3
```

5.4. Fonction `group_by` et `summarise`

`group_by` permet de subdiviser un jeu de données en sous-ensembles. Elle est de la forme `group_by(.data,by_vars)`, où :

- `.data` est le fichier de données que l'on veut grouper ;
- `by_vars` correspond à la variable en fonction de laquelle on veut grouper le jeu de données. Noter qu'il peut s'agir d'une ou de plusieurs variables.

La fonction `summarise` permet de calculer des statistques sur l'ensemble des observations d'un jeu de données. Elle est de la forme : `summarise(.data,...)`, avec :

- `.data` est le jeu de données pour lequel on veut calculer les statistiques ;
- `...` correspond aux variables que l'on désire obtenir. A titre d'exemple, on aurait **Minimum=min(NST)**.

Voici un exemple d'utilisation de la fonction `summarise`.

```
essai<-summarise (Wimbledon, N3ST=getNmatchs(NST,3),N4ST=getNmatchs(NST,4),
          N5ST=getNmatchs(NST,5), MeanNST=mean(NST))
str(essai)
```

```
## 'data.frame':    1 obs. of  4 variables:
##  $ N3ST   : int 70
##  $ N4ST   : int 26
##  $ N5ST   : int 18
##  $ MeanNST: num 3.54
```

`summarise` retourne un *dataframe* avec une seule observation et 4 variables, soit une variable par statistique calculée. Cette fonction permet donc de calculer plusieurs statistiques lors d'un seul appel de fonction et de nommer chacune des variables. Cependant, on ne peut pas l'appliquer sur des sous-ensembles d'un jeu de données. Pour contourner cet obstacle, `dplyr` permet d'utiliser conjointement les fonctions `group_by` et `summarise`.

5.5. Utilisation conjointe de `group_by` et `summarise`

Dans cette conjonction, la fonction `group_by` subdivise le jeu de données en sous-ensembles et la fonction `summarise` calcule les statistiques pour chaque sous-ensemble obtenu avec `group_by`. *Quel beau travail d'équipe ! !*

Appliquons conjointement ces deux fonctions sur notre jeu de données pour calculer les statistiques qu'on nommera N3ST, N4ST, N5ST et MeanNST respectivement nombre de matchs à 3 sets, à 4 sets, à 5 sets et nombre moyen de sets par match par ronde.

```
groupe<-group_by(Wimbledon,Round)# On groupe le jeu de données en fonction de la ronde de la compétitio
selection<-select(groupe,Round,NST)#On sélectionne les variables qui nous intéressent
result<-summarise(selection,
              N3ST=getNmatchs(NST,3),
              N4ST=getNmatchs(NST,4),
              N5ST=getNmatchs(NST,5),
              MeanNST=mean(NST,na.rm=TRUE)) # On calcule les statistiques
print(result)
```

```
## # A tibble: 7 x 5
##   Round  N3ST  N4ST  N5ST  MeanNST
##   <int> <int> <int> <int>    <dbl>
## 1     1    41    13    10 3.515625
## 2     2    15     6     2 3.434783
## 3     3     6     4     2 3.666667
## 4     4     4     2     2 3.750000
## 5     5     3     0     1 3.500000
## 6     6     0     1     1 4.500000
## 7     7     1     0     0 3.000000
```

Si on ne souhaite pas stocker les résultats intermédiaires, on peut imbriquer les fonctions les unes dans les autres. Dans ce cas, le nouveau code s'écrit :

```
summarise(
  select(
    group_by(Wimbledon,Round),
    Round,NST
  ),
  N3ST=getNmatchs(NST,3),
  N4ST=getNmatchs(NST,4),
  N5ST=getNmatchs(NST,5),
  MeanNST=mean(NST,na.rm=TRUE))
```

```
## # A tibble: 7 x 5
##   Round  N3ST  N4ST  N5ST  MeanNST
##   <int> <int> <int> <int>    <dbl>
## 1     1    41    13    10 3.515625
## 2     2    15     6     2 3.434783
## 3     3     6     4     2 3.666667
## 4     4     4     2     2 3.750000
## 5     5     3     0     1 3.500000
## 6     6     0     1     1 4.500000
## 7     7     1     0     0 3.000000
```

6. Comparaison des deux méthodes

En résumé, les deux méthodes (la méthode de base avec `aggregate` et la méthode alternative avec les fonctions `group_by` et `summarise` de `dplyr`) permettent de calculer le nombre de matchs à 3 sets, le nombre de matchs à 4 sets, le nombre de matchs à sets et le nombre de moyen de sets par match en fonction de la ronde (`Round`) de la compétition de tennis Wimbledon en 2013. Toutefois, il existe des avantages et des inconvénients pour chacune des méthodes.

6.1. Avantages et inconvénients de la méthode de base (`aggregate`)

L'utilisation de `aggregate` présente l'avantage de ne pas exiger l'installation préalable de packages vu que cette fonction est issue du R de base. De plus, son utilisation pour calculer des statistiques pour des sous-ensembles d'un jeu de données ne requiert pas de conjonction avec une autre fonction comme c'est le cas de `summarise` qui nécessite `group_by`.
Toutefois, il n'est pas possible d'effectuer le calcul des statistiques et de nommer chacune des variables en un seul appel de fonction. Il faut autant d'appels de fonctions que de statistiques à calculer pour accomplir une telle tâche. Ce qui a pour conséquence l'utilisation de `merge` pour combiner les résultats obtenus. Ce qui n'est pas pratique dans le cas où on aurait un grand nombre de statistiques à calculer sur un jeu de données.

6.2. Avantages et inconvénients de la méthode alternative

L'utilisation conjointe de `group_by` et de `summarise` permet d'effectuer les calculs en un seul appel de fonction et de nommer les variables issues des calculs statistiques. De plus, le résultat est obtenu sous forme d'un tableau de données sans avoir besoin d'utiliser `merge`. Ce qui est pratique dans le cas d'un grand nombre de statistiques à calculer.

Toutefois, cette méthode n'est pas sans inconvénients. Elle exige l'installation préalable du package `dplyr` et la combinaison de `summarise` et `group_by` pour pouvoir calculer des statistiques sur des sous-ensembles d'observations.

De plus, l'affichage du tableau des résultats contient des informations supplémentaires sur le type des sous-objets du tableau. Cela peut être considéré comme un avantage ou un inconvénient selon l'angle sous lequel on l'analyse.

7. Références bibliographiques

- Aide intéractive de R
- Introduction to dplyr : https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html
- Informations sur le jeu de données Wimbledon.men.2013 :https://archive.ics.uci.edu/ml/datasets/Tennis+Major+Tournament+Match+Statistics