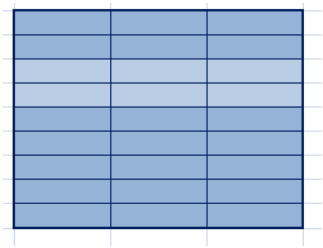
Optimiser la manipulation de data frames avec le package dplyr



Réunion du Groupe de travail
« Ingénieurs statisticiens de Toulouse »
2 Octobre 2015

Sophie LAMARRE
Equipe Biopuces Bionanotechnologies et Plateforme GeT-Biopuces
LISBP Toulouse
Email: sophie.lamarre@insa-toulouse.fr

1. INTRODUCTION SUR DPLYR

- Créé en Janvier 2014 (successeur du package plyr)
- dplyr: « d » pour dataframes
- Facile à utiliser, à comprendre
- Ecrit en C++ => rapidité d'exécution
- Package développé par Hadley Wickham, statisticien senior de R Studio et qui fait partie de l'équipe de développeurs de R (il a notamment développé le package plyr et ggplot2)

1. INTRODUCTION SUR DPLYR

- Principe:

Resultat <- Nom_fonction(data_frame, action_a_realiser)

Toujours un verbe d'action

- 5 Verbes:
  - Select
  - Filter
  - Arrange
  - Mutate
  - Summarise

2. FILTRER LES DONNEES

- Fonction ‘filter’:

```
> head(diamonds)
  carat       cut color clarity depth table price    x    y    z
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48

> # Dimensions initiales
> dim(diamonds)
[1] 53940    10
>
> # Nombre de diamants dont la couleur est "E" et dont le prix est < 330
> filtrage1 <- filter(diamonds, color == "E", price < 330)
> dim(filtrage1)
[1]  3 10
>
> # Nombre de diamants dont la clarte est "SI2" ou dont la longueur (x) < 4
> filtrage2 <- filter(diamonds, clarity == "SI2" | x < 4)
> dim(filtrage2)
[1] 9641   10
```

3. TRIER LE DATA FRAME

- Fonction ‘arrange’:

```
> # On trie le dataframe par:
> # - qualite du decoupage
> # - prix
> # - couleur
> tri1 <- arrange(diamonds, cut, price, color)
> head(tri1)
  carat  cut color clarity depth table price    x    y    z
1  0.22 Fair     E     VS2  65.1    61   337 3.87 3.78 2.49
2  0.25 Fair     E     VS1  55.2    64   361 4.21 4.23 2.33
3  0.23 Fair     G    VVS2  61.4    66   369 3.87 3.91 2.39
4  0.27 Fair     E     VS1  66.4    58   371 3.99 4.02 2.66
5  0.30 Fair     J     VS2  64.8    58   416 4.24 4.16 2.72
6  0.30 Fair     F     SI1  63.1    58   496 4.30 4.22 2.69
>
> # On trie le dataframe par ordre decroissant du prix
> tri2 <- arrange(diamonds, desc(price))
> head(tri2)
  carat       cut color clarity depth table price    x    y    z
1  2.29   Premium     I     VS2  60.8    60 18823 8.50 8.47 5.16
2  2.00 Very Good     G     SI1  63.5    56 18818 7.90 7.97 5.04
3  1.51     Ideal     G      IF  61.7    55 18806 7.37 7.41 4.56
4  2.07     Ideal     G     SI2  62.5    55 18804 8.20 8.13 5.11
5  2.00 Very Good     H     SI1  62.8    57 18803 7.95 8.00 5.01
6  2.29   Premium     I     SI1  61.8    59 18797 8.52 8.45 5.24
```

4. SELECTIONNER CERTAINES VARIABLES

- Fonction ‘select’:

```
> # On cree un tableau ne comportant que les variables "carat", "price" et "color"
> selection1 <- select(diamonds, carat, price, color)
> head(selection1)
  carat price color
1  0.23   326     E
2  0.21   326     E
3  0.23   327     E
4  0.29   334     I
5  0.31   335     J
6  0.24   336     J
>
> # On cree un tableau ne comprenant que les variables entre "carat" et "depth"
> selection2 <- select(diamonds, carat:depth)
> head(selection2)
  carat       cut color clarity depth
1  0.23     Ideal     E     SI2  61.5
2  0.21   Premium     E     SI1  59.8
3  0.23      Good     E     VS1  56.9
4  0.29   Premium     I     VS2  62.4
5  0.31      Good     J     SI2  63.3
6  0.24 Very Good     J    VVS2  62.8
```

4. SELECTIONNER CERTAINES VARIABLES

- Fonction ‘select’:

```
# On cree un tableau comprenant toutes les variables sauf celles
# entre "carat" et "depth"
selection3 <- select(diamonds, -(carat:depth))
head(selection3)
table price    x    y    z
   55   326 3.95 3.98 2.43
   61   326 3.89 3.84 2.31
   65   327 4.05 4.07 2.31
   58   334 4.20 4.23 2.63
   58   335 4.34 4.35 2.75
   57   336 3.94 3.96 2.48

# On cree un tableau ne comprenant que les variables contenant dans leur intitule
# un "y"
selection4 <- select(diamonds, contains("y"))
head(selection4)
clarity    y
    SI2 3.98
    SI1 3.84
    VS1 4.07
    VS2 4.23
    SI2 4.35
   VVS2 3.96
```

4. SELECTIONNER CERTAINES VARIABLES

- Fonction ‘select’:

```
> # On cree un tableau ne comprenant que les variables dont l'intitutle commence
> # par "c"
> selection5 <- select(diamonds, starts_with("c"))
> head(selection5)
  carat       cut color clarity
1  0.23     Ideal     E     SI2
2  0.21   Premium     E     SI1
3  0.23      Good     E     VS1
4  0.29   Premium     I     VS2
5  0.31      Good     J     SI2
6  0.24 Very Good     J    VVS2
```

5. RENOMMER UNE VARIABLE

- Fonction ‘rename’:

```
> renom <- rename(diamonds, couleur_carat = color)
> head(renom)
  carat       cut couleur_carat clarity depth table price    x    y    z
1  0.23     Ideal             E     SI2  61.5    55   326 3.95 3.98 2.43
2  0.21   Premium             E     SI1  59.8    61   326 3.89 3.84 2.31
3  0.23      Good             E     VS1  56.9    65   327 4.05 4.07 2.31
4  0.29   Premium             I     VS2  62.4    58   334 4.20 4.23 2.63
5  0.31      Good             J     SI2  63.3    58   335 4.34 4.35 2.75
6  0.24 Very Good             J    VVS2  62.8    57   336 3.94 3.96 2.48
```

6. AJOUTER UNE NOUVELLE COLONNE

- Fonction ‘mutate’:

```
> # On calcule le prix au kilo pour chaque diamant
> # puis on convertit le prix en euro
> calcul1 <- mutate(diamonds,
+                   prix_au_kilo = price / carat,
+                   prix_kilo_euro = prix_au_kilo * 0.90354642)
> head(calcul1)
  carat       cut color clarity depth table price    x    y    z prix_au_kilo prix_kilo_euro
1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43     1417.391      1280.6788
2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31     1552.381      1402.6483
3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31     1421.739      1284.6073
4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63     1151.724      1040.6362
5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75     1080.645       976.4131
6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48     1400.000      1264.9650
>
```

7. NE GARDER QUE LES NOUVELLES COLONNES

- Fonction ‘transmute’:

```
> # On calcule le prix au kilo pour chaque diamant
> # puis on convertit le prix en euro
> # On ne garde que ces deux nouvelles colonnes
> keep1 <- transmute(diamonds,
+                    prix_au_kilo = price / carat,
+                    prix_kilo_euro = prix_au_kilo * 0.90354642)
> head(keep1)
  prix_au_kilo prix_kilo_euro
1     1417.391      1280.6788
2     1552.381      1402.6483
3     1421.739      1284.6073
4     1151.724      1040.6362
5     1080.645       976.4131
6     1400.000      1264.9650
```

8. EFFECTUER QUELQUES CALCULS

- Fonction ‘summarise’:

```
> # On calcule le prix moyen au kilo des diamants
> summarise(calcul1,
+           moy_prix = mean(prix_kilo_euro, na.rm = TRUE))
  moy_prix
1 3621.771
>
> # Calculer le prix median par qualite du decoupage du diamant
> summarise(group_by(diamonds, cut), mediane_prix = median(price))
Source: local data frame [5 x 2]

        cut mediane_prix
1      Fair       3282.0
2      Good       3050.5
3 Very Good       2648.0
4   Premium       3185.0
5     Ideal       1810.0
```

8. EFFECTUER QUELQUES CALCULS

- Fonction ‘summarise’:

```
> # On compte combien on a de diamants par couleur
> summarise(group_by(calcul1,color), compteur = n())
Source: local data frame [7 x 2]

  color compteur
1     D     6775
2     E     9797
3     F     9542
4     G    11292
5     H     8304
6     I     5422
7     J     2808
>
> # equivalent avec syntaxe >%> (pour que ce soit plus lisible)
> calcul1 %>% group_by(color) %>% summarise(compteur = n())
Source: local data frame [7 x 2]

  color compteur
1     D     6775
2     E     9797
3     F     9542
4     G    11292
5     H     8304
6     I     5422
7     J     2808
```

9. EFFECTUER UN TIRAGE ALEATOIRE

- Fonction ‘sample_...’:

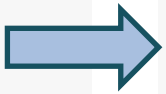
Par défaut, replace=FALSE

```
> # On tire aleatoirement 10 diamants
> tirage1 <- sample_n(diamonds, 10)
> dim(tirage1)
[1] 10 10
> head(tirage1)
      carat       cut color clarity depth table price    x    y    z
52598  0.53 Very Good     E      IF  59.8    60  2542 5.28 5.35 3.18
15141  1.12     Ideal     F     SI1  61.6    56  6087 6.66 6.72 4.12
17107  1.74   Premium     F      I1  59.9    58  6821 7.88 7.82 4.70
4670   1.01      Good     D     SI2  63.8    61  3671 6.13 6.06 3.89
19367  0.40      Good     D     SI2  63.1    57   622 4.72 4.75 2.99
21158  1.54   Premium     I     SI1  60.1    58  9314 7.53 7.48 4.51
>
> # On tire aleatoirement 1% des diamants
> tirage2 <- sample_frac(diamonds, 0.01)
> dim(tirage2)
[1] 539  10
> head(tirage2)
      carat       cut color clarity depth table price    x    y    z
11824  1.33   Premium     I     SI2  59.7    59  5094 7.21 7.15 4.29
4005   0.90 Very Good     I     VS2  63.3    58  3519 6.07 6.13 3.86
34858  0.30   Premium     G    VVS2  62.6    58   878 4.29 4.27 2.68
24675  1.22     Ideal     F    VVS1  61.2    57 13029 6.88 6.92 4.22
20179  1.03   Premium     E     VS1  62.8    58  8629 6.46 6.41 4.04
52157  0.80 Very Good     I     SI1  61.9    56  2473 5.94 5.98 3.69
>
```

10. FONCTIONS COMBINEES: CHAINING OU PIPELINING

- >%>
  - Nouvelle fonctionnalité du package dplyr, issu du package maggrit (qui a créé >%>)
  - On peut le prononcer « then »
  - Permet d'effectuer plusieurs opérations en une seule ligne

```
filter(
  summarise(
    select(
      group_by(flights, year, month, day),
      arr_delay, dep_delay
    ),
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ),
  arr > 30 | dep > 30
)
```



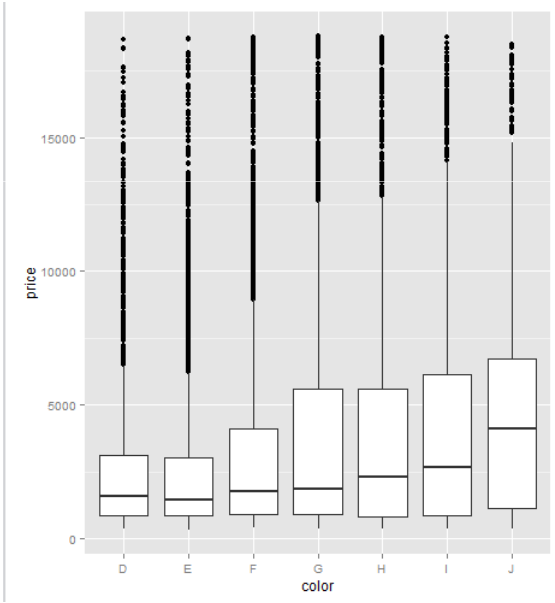
```
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
```

```
#> Source: local data frame [49 x 5]
#> Groups: year, month [11]
#>
#>     year month   day      arr      dep
#>    (int) (int) (int)    (dbl)    (dbl)
#> 1   2013     1    16 34.24736 24.61287
#> 2   2013     1    31 32.60285 28.65836
#> 3   2013     2    11 36.29009 39.07360
#> 4   2013     2    27 31.25249 37.76327
#> ..   ...   ...   ...      ...      ...
```

10. EXEMPLE DE FONCTIONS COMBINEES

- L'art d'effectuer simplement un graphique en faisant un calcul en quelques lignes:

```
> # On realise un graphique avec ggplot2 montrant la distribution
> # du prix selon la couleur du diamant
> diamonds %>%                          # On selectionne le jeu de donnees "diamonds"
+   filter(cut == "Ideal") %>%          # Onfiltre: cut == Ideal
+   ggplot(aes(x=color,y=price)) +      # On utilise ggplot2
+   geom_boxplot()                      #  et on cree les boxplots
> |
```



11. DPLYR vs PLYR

- dplyr est beaucoup plus rapide que plyr
- dplyr ne s'utilise qu'avec des dataframes
- plyr possède des fonctionnalités facilitant la manipulation de data frames, arrays, listes)

12. CONCLUSION

- D'autres fonctionnalités existent comme réaliser le merge entre deux dataframes, …
- Plus de documentations:
  - https://github.com/hadley/dplyr
  - https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf