Transformer les données

avec dplyr et janitor

Anicet Ebou

Abidjan R users

2020-04-25

Situation

- Très souvent, nous sommes confrontés à ce genre de fichiers Excel:

	A	B	C	D	E	F	G	H	I	J
1		patient	44	44	10	10	3	3	53	53
2		IDS	T5/T6	T5/T6	T5/T6	T5/T6	T5/T6	T5/T6	T5/T6	T5/T6
3	angles		A	B	A	B	A	B	A	B
4	60		31.83133	1	31.52867	1	32.92768	0	31.88682	0
5	65		31.66967	1	31.33477	1	32.23012	0	32.36918	0
6	70		31.45108	1	31.09272	0.202002	31.73647	0	32.51658	0
7	75		31.0815	1	30.96078	0.448317	31.20458	0.086414	32.33217	1

- Ou alors on a de beaux fichiers mais on a besoin de réordonner les variables, ou créer de nouvelles variables etc.

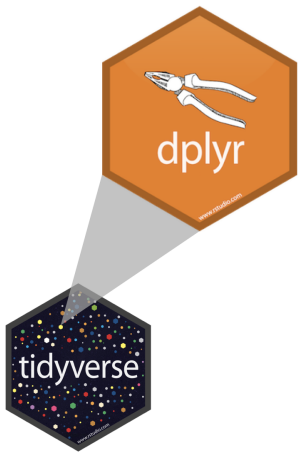
- La vérité c'est que la majorité du processus de l'analyse de données consiste en nettoyer les données (50 à 80 % du temps).

Transformer les données ?

- C'est créer des sous-ensemble de la base de donnée en présence.
- C'est créer de nouvelles variables, ordonner d'autres
- Effectuer des calculs sur les données

dplyr pour la manipulation des données

dplyr est basé sur des verbes (fonctions) qui facilitent la manipulation des données.



- `filter()`: sélectionne les observations par leurs valeurs.
- `arrange()`: réorganise les lignes.
- `select()`: sélectionne les variables par leurs noms.
- `mutate()`: crée de nouvelles variables en fonction des variables existantes.
- `summarise()`: Résume les valeurs en un seul summary.
- etc.

Janitor

- Janitor contient des fonction de base pour la manipulation et l'exploration des données.

Janitor aide en particulier à:

- Formater correctement les nom des colonnes des tableaux
- Créer des tables de contingences plus facilement
- Retirer ou isoler les doublons

Une grammaire pour la manipulation des données

Tout les verbes (fonctions) **dplyr** et **janitor** fonctionnent de la même manière:

- Le premier argument est un data frame.

```
verb(dataframe)
```

- Les arguments des verbes décrivent **quoi faire** avec le data frame, en utilisant **les noms des variables (sans les griffes)**.

```
verb(dataframe, var1 == value)
```

- Le résultat est un nouveau data frame (Il faut donc sauver ce nouveau data frame).

```
df <- verb(dataframe, var1 == value)
```

Filtrer les observations avec `filter()`

Nous utiliserons la base de donnees diamants disponible dans le package ggplot2. Je préconise de charger le package tidyverse avant de commencer: `library(tidyverse)`.

Pour filtrer les diamants de 1000 dollars uniquement:

```
filter(diamonds, price == 1000)
```

```
## # A tibble: 25 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.38  Very Good E     VVS2     61.8    56  1000  4.66  4.68  2.88
##  2 0.39  Very Good F     VS1      57.1    61  1000  4.86  4.91  2.79
##  3 0.38  Very Good E     VS1      61.5    58  1000  4.64  4.69  2.87
##  4 0.38  Premium   E     VS1      60.7    59  1000  4.65  4.7   2.84
##  5 0.38  Ideal     E     VS1      61.6    56  1000  4.65  4.67  2.87
##  6 0.53  Very Good G     SI2      62.5    55  1000  5.14  5.19  3.23
##  7 0.570 Very Good I     SI2      62.1    57  1000  5.29  5.33  3.3
...
```

Sélectionner des variables avec `select()`

- Sélectionner pour garder les variables

```
select(diamonds, price, carat)
```

```
## # A tibble: 53,940 x 2
##    price carat
##    <int> <dbl>
##  1   326 0.23
##  2   326 0.21
...
```

- Exclure des variables

```
select(diamonds, -carat)
```

```
## # A tibble: 53,940 x 9
##    cut       color clarity depth table price     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
...
```

Sélectionner des variables avec `select()`

Sélectionner une suite variables

```
select(diamonds, carat:price)
```

```
## # A tibble: 53,940 x 7
##    carat cut       color clarity depth table price
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int>
##  1 0.23  Ideal     E     SI2      61.5    55   326
##  2 0.21  Premium   E     SI1      59.8    61   326
##  3 0.23  Good      E     VS1      56.9    65   327
##  4 0.290 Premium   I     VS2      62.4    58   334
##  5 0.31  Good      J     SI2      63.3    58   335
##  6 0.24  Very Good J     VVS2     62.8    57   336
##  7 0.24  Very Good I     VVS1     62.3    57   336
...
```

Introduisons l'opérateur pipe: %>%

Ecrire

```
filter(diamonds, price == 1000)
```

Equivaut à (avec la pipe)

```
diamonds %>% filter(price == 1000)
```

```
## # A tibble: 25 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1 0.38  Very Good E     VVS2     61.8    56  1000  4.66  4.68  2.88
##  2 0.39  Very Good F     VS1      57.1    61  1000  4.86  4.91  2.79
...
```

Introduisons l'opérateur pipe: %>%

`%>%` permet d'effectuer des opérations multiples sur les mêmes données plus facilement tout en gardant le code lisible.

Par exemple

```
diamants_1000 <- filter(diamonds, price == 1000)
diamant_1000_carat_prix <- select(diamants_1000, carat, price)
```

est plus simplement effectué par

```
diamonds %>%
  filter(price == 1000) %>%
  select(carat, price)
```

Filtrer sur plusieurs conditions à la fois

Pour les diamants de 1000 dollars, de carat 0.1 et 0.5:

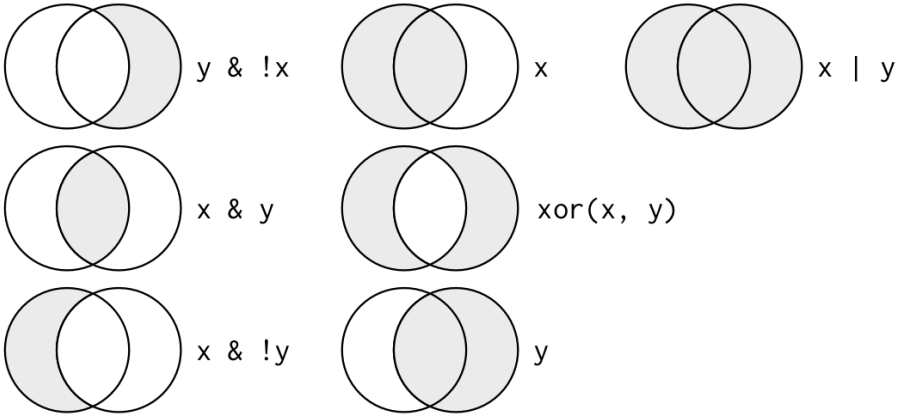
```
diamonds %>%
  filter(price == 1000, carat %in% c(0.1,0.5))
```

```
## # A tibble: 3 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1   0.5 Good      E     SI2      63.2    61  1000  5.02  5.05  3.18
## 2   0.5 Premium   G     SI2      62.5    57  1000  5.04  5.01  3.14
## 3   0.5 Very Good J     SI1      63.2    55  1000  5.07  5.03  3.19
```

Les opérateurs logiques

opérateur	définition	opérateur	définition
<	plus petit	x | y	x OU y
<=	plus petit ou égale	is.na(x)	test si x est NA
>	plus grand que	!is.na(x)	test si x n'est pas NA
>=	plus grand ou égal à	x %in% y	test si x est contenue dans y
==	égale à	!(x %in% y)	test si x n'est pas contenue dans y
!=	différent de	!x	différent de x
x & y	x ET y		

Comment les opérateurs logiques fonctionnent




```
diamonds %>%
  filter(price == 1000, carat == 0.1 | carat == 0.5)
```

```
## # A tibble: 3 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1   0.5 Good      E     SI2      63.2    61  1000  5.02  5.05  3.18
## 2   0.5 Premium   G     SI2      62.5    57  1000  5.04  5.01  3.14
...
```

Ordonner les lignes avec `arrange()`

`arrange()` permet d'ordonner les lignes de facon croissante ou décroissante.

```
diamonds %>% arrange(carat)
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1   0.2 Premium   E     SI2      60.2    62   345  3.79  3.75  2.27
##  2   0.2 Premium   E     VS2      59.8    62   367  3.79  3.77  2.26
...
```

```
diamonds %>% arrange(desc(carat))
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  5.01 Fair      J     I1       65.5    59 18018 10.7  10.5   6.98
##  2  4.5  Fair      J     I1       65.8    58 18531 10.2  10.2   6.72
...
```

Sélectionner un certain nombre de ligne avec `slice()`

Pour les 10 premières lignes:

```
diamonds %>% slice(1:10)
```

```
## # A tibble: 10 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  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 0.290 Premium   I     VS2      62.4    58   334  4.2   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
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
...
```

Ajouter de nouvelles variables avec `mutate()`

`mutate()` permet d'ajouter de nouvelles colonnes (variables).

```
diamonds %>%
  mutate(prix_cfa = price * 610)
```

```
## # A tibble: 53,940 x 11
##    carat cut       color clarity depth table price     x     y     z pri
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>   
##  1 0.23  Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43   19
##  2 0.21  Premium   E     SI1      59.8    61   326  3.89  3.84  2.31   19
##  3 0.23  Good      E     VS1      56.9    65   327  4.05  4.07  2.31   19
##  4 0.290 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63   20
##  5 0.31  Good      J     SI2      63.3    58   335  4.34  4.35  2.75   20
##  6 0.24  Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48   20
##  7 0.24  Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47   20
...
```

Mesures groupées avec `summarise()`

Combiné avec `group_by()`

```
diamonds %>%
  group_by(carat) %>%
  summarise(prix_moyen = mean(price, na.rm = TRUE))
```

```
## # A tibble: 273 x 2
##    carat prix_moyen
##    <dbl>      <dbl>
##  1 0.2         365.
##  2 0.21        380.
##  3 0.22        391.
##  4 0.23        486.
##  5 0.24        505.
##  6 0.25        551.
##  7 0.26        551.
##  8 0.27        575.
##  9 0.28        580.
## 10 0.290       601.
## # … with 263 more rows
```

Autres fonctions utiles

- `pull()` pour extraire une variable comme vecteur

```
diamonds %>% pull(price)
```

```
##     [1]   326   326   327   334   335   336   336   337   337   338   339
##    [13]   342   344   345   345   348   351   351   351   351   352   353
##    [25]   353   354   355   357   357   357   402   402   402   402   402
##    [37]   402   402   403   403   403   403   403   403   403   403   403
##    [49]   404   404   404   404   404   404   404   405   405   405   405
...
```

- `distinct()` pour avoir les lignes uniques

```
diamonds %>% distinct(color)
```

```
## # A tibble: 7 x 1
##   color
##   <ord>
## 1 E
## 2 I
...
```

Autres fonctions utiles

- `count()` pour le nombre d'observation par groupe

```
diamonds %>% count(color)
```

```
## # A tibble: 7 x 2
##   color     n
##   <ord> <int>
## 1 D      6775
## 2 E      9797
## 3 F      9542
## 4 G     11292
## 5 H      8304
## 6 I      5422
## 7 J      2808
```

formater correctement les nom des variables avec clean_names()

```
colnames(iris)
```

```
## [1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
```

```
iris %>% clean_names()
```

```
##     sepal_length sepal_width petal_length petal_width    species
## 1            5.1         3.5          1.4         0.2     setosa
## 2            4.9         3.0          1.4         0.2     setosa
## 3            4.7         3.2          1.3         0.2     setosa
## 4            4.6         3.1          1.5         0.2     setosa
## 5            5.0         3.6          1.4         0.2     setosa
## 6            5.4         3.9          1.7         0.4     setosa
## 7            4.6         3.4          1.4         0.3     setosa
## 8            5.0         3.4          1.5         0.2     setosa
## 9            4.4         2.9          1.4         0.2     setosa
...
```

Plusieurs dizaines d'autres fonctions

Il existe encore plusieurs dizaine d'autrs fonction a explorer:

- https://dplyr.tidyverse.org
- http://sfirke.github.io/janitor

Merci pour votre attention

Slides: bit.ly/2VAcJgL