JANITOR

Janitor a pour fonction principale d'examiner et de nettoyer des jeux de données de facon rapide.

Il convient a des utilisateurs d'un niveau entre débutant et intermédiaire sous R. Des utilisateurs plus aguerris peuvent coder facilement ce que JANITOR propose. Ce package permet neanmoins de gagner en rapidité. Il se rapproche aussi beaucoup des fonctions d'analyse proposées par SPSS ou encore EXCEL. — lien Github utile => https://github.com/sfirke/janitor.

Nous cherchons ici a lire un jeu de donnees Excel.

```
library(readxl)
```

RECUPERATION DU JEU DE DONNEES.
J'indique donc ou aller chercher mon jeu de données sur le Cloud. J'utise la fonction **read_excel()** pour lire mes données.

```
mymsa = read_excel("/cloud/project/mymsa.xlsx")
```

ETUDE DES FONCTIONS PRINCIPALES.
Nous vous proposons d'étudier les fonctions qui ont retenu notre attention.

```
# NETTOYONS LES NOMS AVEC **clean_names()**
x = janitor::clean_names(mymsa)
data.frame(mymsa = colnames(mymsa), x = colnames(x))
```

```
##                mymsa                    x
## 1               RFID                 rfid
## 2              Plant                plant
## 3           KillDate            kill_date
## 4             BodyNo              body_no
## 5   LeftSideScanTime  left_side_scan_time
## 6  RightSideScanTime right_side_scan_time
## 7         HangMethod          hang_method
## 8                Hgp                  hgp
## 9                Sex                  sex
## 10          LeftHscw            left_hscw
## 11         RightHscw           right_hscw
## 12         TotalHscw           total_hscw
## 13             P8Fat                p8fat
## 14               Lot                  lot
## 15          Est % BI       est_percent_bi
## 16          HumpCold            hump_cold
## 17               Ema                  ema
## 18  OssificationCold    ossification_cold
## 19       AusMarbling         aus_marbling
## 20       MsaMarbling         msa_marbling
## 21        MeatColour          meat_colour
## 22         FatColour           fat_colour
## 23        RibfatCold          ribfat_cold
## 24                Ph                   ph
## 25          LoinTemp            loin_temp
```

```
## 26          FeedType            feed_type
## 27      NoDaysOnFeed      no_days_on_feed
## 28          MSAIndex            msa_index
## 29             spare                spare
```

Les :: nous servent ici a appeler la fonction clean_names du package Janitor. R va ici nous fournir en sortie un tableau avec les intitules de colonnes remis au meme format ici tout en minuscules. Nous avons gardé a gauche les colonnes de mysa et a droite celles modifiées. —

```
#OBTENIR LA FREQUENCE D'UN VECTEUR EN UTLISANT **tabyL()**
#Sur ce meme data frame remis au bon format nous allons chercher a étudier la frequende d'un vecteur.
#Faisons un test sur meat_colour.
tabyl(x, meat_colour)
```

```
##  meat_colour    n percent
##           1B   87 0.02175
##           1C  657 0.16425
##            2 1730 0.43250
##            3 1478 0.36950
##            4   30 0.00750
##            5   14 0.00350
##            6    4 0.00100
```

Nous pouvons appeler la fonction **tabyl()** sur un data frame qui est “piped in” ce qui permet de gagner en rapidite et flexibilité.
%>% represente ici le “pipe” = par cette manipulation nous n’avons pas a rappeler le data frame.

```
# Load dplyr for the %>% pipe
library(dplyr)
```

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

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

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

```
x %>% tabyl(meat_colour)
```

```
##  meat_colour    n percent
##           1B   87 0.02175
##           1C  657 0.16425
##            2 1730 0.43250
##            3 1478 0.36950
##            4   30 0.00750
##            5   14 0.00350
##            6    4 0.00100
```

Faisons un focus sur une des colonnes, ici spare.

```
x %>% tabyl(spare)
```

```
##  spare    n percent valid_percent
##     NA 4000       1            NA
```

Nous constatons que cette colonne ne contient que des valeurs manquantes (NA).

```
#ENLEVER LES COLONNES VIDES
#Ici nous avons besoin de la fonction **remove_empty()**.
x = remove_empty(x, which = c("rows","cols"))
```

Cette fonction peut s'utiliser directement lors de l'import en utilisant une pipeline, comme nous pouvons le voir ci-dessous:

```
#UTILISATION D'UNE PIPELINE A L'IMPORT
x = read_excel("/cloud/project/mymsa.xlsx") %>%
  clean_names() %>% remove_empty()
```

```
## value for "which" not specified, defaulting to c("rows", "cols")
```

Dans ce cas pas besoin de specifier la valeur de which, elle est attribuée par defaut.

```
#CROSSTABULATION
#Commencons par regarder le distribution de meat_colours
x %>% tabyl(meat_colour, plant)
```

```
##  meat_colour    1   2
##           1B    0  87
##           1C   87 570
##            2 1443 287
##            3 1477   1
##            4   27   3
##            5    9   5
##            6    1   3
```

Passons au total par colonne.
Nous changeons ici where = "row" par where = "col".

```
#AJOUT DE TOTAUX PAR COLONNE **adorn_totals()**
x %>%
  tabyl(meat_colour, plant) %>%
  adorn_totals(where = "col")
```

```
##  meat_colour    1   2 Total
##           1B    0  87    87
##           1C   87 570   657
##            2 1443 287  1730
##            3 1477   1  1478
##            4   27   3    30
##            5    9   5    14
##            6    1   3     4
```

La encore il nous reste des options pour formatter les données avec des pourcentages.

```
#FORMATTAGE DES DONNEES AVEC **adorn_pct_formatting()**
x %>%
  tabyl(meat_colour, plant) %>%
  adorn_totals(where = c("row","col")) %>%
  adorn_percentages(denominator = "col") %>%
  adorn_pct_formatting(digits = 0)
```

```
##  meat_colour    1    2 Total
##           1B   0%   9%    2%
##           1C   3%  60%   16%
##            2  47%  30%   43%
##            3  49%   0%   37%
##            4   1%   0%    1%
##            5   0%   1%    0%
##            6   0%   0%    0%
##        Total 100% 100%  100%
```

Passons maintenant a l'étude des doublons

```
#GET DUPES
#Afin d'etudier les doublons nous regardons pour chaque donnée si nous
#disposons d'un identifiant unique et nous allons chercher s'il existe une #version doublonnée de cet i
x %>% get_dupes(rfid)
```

```
## No duplicate combinations found of: rfid
```

```
## # A tibble: 0 x 29
## # ... with 29 variables: rfid <chr>, dupe_count <int>, plant <dbl>,
## #   kill_date <dttm>, body_no <dbl>, left_side_scan_time <dbl>,
## #   right_side_scan_time <dbl>, hang_method <chr>, hgp <chr>, sex <chr>,
## #   left_hscw <dbl>, right_hscw <dbl>, total_hscw <dbl>, p8fat <dbl>,
## #   lot <dbl>, est_percent_bi <chr>, hump_cold <dbl>, ema <dbl>,
## #   ossification_cold <dbl>, aus_marbling <dbl>, msa_marbling <dbl>,
## #   meat_colour <chr>, fat_colour <dbl>, ribfat_cold <dbl>, ph <dbl>,
## #   loin_temp <dbl>, feed_type <chr>, no_days_on_feed <dbl>, msa_index <dbl>
```

Il n 'existe ici aucun doublon.

Janitor est particulierement utile lorsque nous disposons de données Excel.
Il suffit d'y rentrer l'équivalent numérique de la date que nous souhaitons convertir.

```
#LE PETIT PLUS
#CONVERTIR DES FORMATS DE DATE SOUS EXCEL **excel_numeric_to_date**
#1er exemple.
excel_numeric_to_date(42223)
```

```
## [1] "2015-08-07"
```

```
#2nd exemple.
excel_numeric_to_date(41103)
```

```
## [1] "2012-07-13"
```

Janitor peut ici clairement servir a nettoyer notre jeu de données avant de l'importer et d'utiliser le package StatsModels dessus.