JANITOR

JANITOR PERMET DE NETTOYER UN SET DE DONNEES.
Il convient a des utilisateurs d'un niveau entre debutant et intermediaire sous R. Des utilisateurs plus aguerris peuvent coder ce que JANITOR propose, mais ce package permet de le faire plus rapidement.

lien Github utile => https://github.com/sfirke/janitor. Installons le Package pour voir de quoi il s'agit.

```
#install.packages("janitor")
library(janitor)
```

```
##
## Attaching package: 'janitor'
```

```
## The following objects are masked from 'package:stats':
##
##     chisq.test, fisher.test
```

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

# NETTOYER LES NOMS : 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
```

```
#R va ici nous fournir en sortie un tableau avec les intitules de colonnes.
# remis au meme format (tout en minuscule)
```

```
#OBTENIR LA FREQUENCE D'UN VECTEUR EN UTLISANT TABYL()
#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
```

```
#Comparons tabyl() a Table()
```

```
table(x$meat_colour)
```

```
##
##   1B   1C    2    3    4    5    6
##   87  657 1730 1478   30   14    4
```

Nous pouvons appeler la fonction tabyl() sur un data frame qui est “piped in” ce qui permet de gagner en rapidite et flexibilite.

```
# 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
```

```
#Nous avons la possibilite de changer le format,
#en ajoutant les pourcentages avec **adorn_pct_formatting**
```

```
x %>%
  tabyl(meat_colour) %>%
  adorn_pct_formatting(digits = 0, affix_sign = TRUE)
```

```
##  meat_colour    n percent
##           1B   87      2%
##           1C  657     16%
##            2 1730     43%
##            3 1478     37%
##            4   30      1%
##            5   14      0%
##            6    4      0%
```

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
x = remove_empty(x, which = c("rows","cols"))
```

Cette fonction peut s'utiliser directement lors de l'import en utilisant une pipeline

```
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 attribuee 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
```

Nous pouvons ajouter des sous totaux de ligne ou de coolonne en utilisant **adorn_totals()**. Faisons le test ici sur le total de lignes

```
x %>%
  tabyl(meat_colour, plant) %>%
  adorn_totals(where = "row")
```

```
##  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
##        Total 3044 956
```

Passons au total par colonne

```
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
```

Nous avons la possibilite d'afficher a la fois le total par ligne et par colonne au sein d'une meme fonction

```
x %>%
  tabyl(meat_colour, plant) %>%
  adorn_totals(where = c("row","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
##        Total 3044 956  4000
```

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

```
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%
```

Et nous pouvons de nouveau inserer des totaux avec adorn()

```
x %>%
  tabyl(meat_colour, plant) %>%
  adorn_totals(where = c("row","col")) %>%
  adorn_percentages(denominator = "col") %>%
```

```
  adorn_pct_formatting(digits = 1) %>%
  # Nous pouvons aujuster ici le nombre de chiffres apres la virgule du pourcentage
  adorn_ns(position = "front")
```

```
##  meat_colour            1            2         Total
##           1B    0   (0.0%)  87   (9.1%)   87   (2.2%)
##           1C   87   (2.9%) 570  (59.6%)  657  (16.4%)
##            2 1443  (47.4%) 287  (30.0%) 1730  (43.2%)
##            3 1477  (48.5%)   1   (0.1%) 1478  (37.0%)
##            4   27   (0.9%)   3   (0.3%)   30   (0.8%)
##            5    9   (0.3%)   5   (0.5%)   14   (0.4%)
##            6    1   (0.0%)   3   (0.3%)    4   (0.1%)
##        Total 3044 (100.0%) 956 (100.0%) 4000 (100.0%)
```

Passons maintenant a l'etude des doublons

```
#GET DUPES
#Afin d'etudier les doublons nous regardons pour chaque donnee si nous disposons d'un identifiant uniqu
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. Nous allons donc creer artificiellement des doublons

```
x1 = x %>% slice(1:3)
x2 = bind_rows(x1,x)
x2 %>% get_dupes(rfid)
```

```
## # A tibble: 6 x 29
##   rfid  dupe_count plant kill_date           body_no left_side_scan_~
##   <chr>      <int> <dbl> <dttm>                <dbl>            <dbl>
## 1 201 ~          2     1 2016-08-15 00:00:00     193              423
## 2 201 ~          2     1 2016-08-15 00:00:00     193              423
## 3 253 ~          2     1 2016-08-15 00:00:00     257              542
## 4 253 ~          2     1 2016-08-15 00:00:00     257              542
## 5 818 ~          2     1 2016-08-02 00:00:00      99              445
## 6 818 ~          2     1 2016-08-02 00:00:00      99              445
## # ... with 23 more variables: 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>
```

```
#CONVERTIR DES FORMATS DE DATE SOUS EXCEL
#Janitor est particulierement utile lorsque nous disposons de donnees Excel
#Il suffit d'y rentrer l'equivqlent numerique de la date que nous souhaitons convertir
```

```r
excel_numeric_to_date(42223)
```

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

```r
excel_numeric_to_date(41103)
```

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