GDATA PACKAGE

Nous présentons le package “gdata” que vous pouvez télécharger depuis CRAN, ou directement depuis R-studio en allant dans packages. Nous avons choisi ce package car il est intuitif, très pratique et simple d'utilisation. Il permet de manipuler à la fois une ou plusieurs base de données, notamment des fichiers excel, et faire de la data visualisation. On retrouve de nombreuses fonctions présentes dans des libraires python comme Pandas mais sous un nom différent. Nous allons vous montrer ses principales fonctions qu'on a jugé comme étant les plus pertinentes.

On commence par importer la librairie après avoir installé le package et importer la data que nous avons téléchargé sur Kaggle.

```
library(gdata)
```

```
## gdata: read.xls support for 'XLS' (Excel 97-2004) files ENABLED.
##
## gdata: read.xls support for 'XLSX' (Excel 2007+) files ENABLED.
##
## Attaching package: 'gdata'
## The following object is masked from 'package:stats':
##
##     nobs
## The following object is masked from 'package:utils':
##
##     object.size
## The following object is masked from 'package:base':
##
##     startsWith
```

```
df <- read.xls("/Users/akram/Downloads/e_commerce.xls")
# View(df) à exécuter si vous souhaitez voir votre tableau
```

On cast bien notre base de données afin de s'assurer qu'elle soit reconnue en tant que data frame (non obligatoire)

```
x <-as.data.frame(df)
```

Voici un moyen de visualiser globalement notre data frame

```
ll(x, dim = TRUE, sort = FALSE)
```

```
##                Class  KB   Dim
## InvoiceNo   character 707 27755
## StockCode   character 618 27755
## Description character 960 27755
## Quantity    character 625 27755
## InvoiceDate character 670 27755
## UnitPrice   character 580 27755
## CustomerID  character 622 27755
## Country     character 575 27755
```

Voici un second moyen de visualiser globalement notre data frame mais cette fonction n'est pas propre à notre package

```
summary(x)
```

```
##   InvoiceNo          StockCode         Description          Quantity
##  Length:27755       Length:27755       Length:27755       Length:27755
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##  InvoiceDate         UnitPrice          CustomerID          Country
##  Length:27755       Length:27755       Length:27755       Length:27755
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
```

On peut regarder les premières lignes de notre data frame, on peut choisir le nombre de lignes voulu (par exemple 4)

```
first(x, n = 4)
```

```
##   InvoiceNo StockCode                         Description Quantity  InvoiceDate
## 1    536365    85123A  WHITE HANGING HEART T-LIGHT HOLDER        6 12/1/10 8:26
## 2    536365     71053                 WHITE METAL LANTERN        6 12/1/10 8:26
## 3    536365    84406B      CREAM CUPID HEARTS COAT HANGER        8 12/1/10 8:26
## 4    536365    84029G KNITTED UNION FLAG HOT WATER BOTTLE        6 12/1/10 8:26
##   UnitPrice CustomerID        Country
## 1      2.55      17850 United Kingdom
## 2      3.39      17850 United Kingdom
## 3      2.75      17850 United Kingdom
## 4      3.39      17850 United Kingdom
```

On peut également choisir de visualiser les premières/dernières colonnes

left(x, n=2) #extremité gauche pour les premières, on utilise pour les dernières

On vérifie l'existence de doublons dans une colonne grâce à la fonction (renvoie un booléen)

duplicated2(x$CustomerID, bothWays = FALSE) #Plus adapté pour des petites tables

On peut supprimer les doublons grâce à la fonction

unique(x$CustomerID, incomparables = FALSE)

Il est possible de changer le nom d'une colonne

rename.vars(x, from = "CustomerID", to = "idclient")

Une carctéristique intéressante est de connaître le typage de l'objet qu'on manipule

```
is.what(x)
```

```
## [1] "is.data.frame" "is.list"       "is.object"     "is.recursive"
## [5] "is.unsorted"
```

Notre base de données de départ ne contenait que des valeurs propres, mais il arrive très souvent qu'une table puisse contenir des variables non prises en charge.Nous pouvons grâce à ce package, changer les variables inconnues en NA et inversement les valeurs NA en variables inconnues. Nous choisissons la condition pour laquelle une variable ne devrait pas être prise en compte ou être considerée comme NA.

unknownToNA(x$UnitPrice, unknown = "0.00") #Ici on a décidé que les valeurs nulles dans notre colonne UnitPrice devriaent être NA

Comme dans d'autres langages comme Python ou SQL, on peut effectuer des jointures entre plusieurs tables sur R et notamment avec la fonction ci-dessous

bindData(x, y, common) où x,y sont deux tables distinctes, ici la jointure par défaut est la jointure par une clé (colonne) commune aux deux tables

Grâce à ce package on peut mettre à jour des listes et colonnes

```
db1 <- list(âge = 26, taille = "grand", fumeur = "oui", 1, 16)
db2 <- list(taille = "petit", 3, fumeur = "non", 4)
update(db1, db2)
```

```
## $âge
## [1] 26
##
## $taille
## [1] "petit"
##
```

```
## $fumeur
## [1] "non"
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 16
```

```
#db1 ne prend en compte que les modifications pour les variables déclarées, cette liste garde toujours
```

Voilà c'est la fin de la présentation de ce package, si vous êtes amenés à travailler sur des tables sur R, cela pourrait vous être utile de connaître ce package notamment pour visualiser ou changer la structure de la table. Je vous mets à disposition sur mon github le fichier excel que j'ai utilisé pour ce tutorial.