R et Python
Performances comparées

Chargement de données, filtres, tris, statistiques descriptives

Packages pour R et Python

Ricco Rakotomalala

R et Python – Performances comparées

Comparer les performances – principalement le temps d'exécution, mais aussi l'occupation mémoire – de R et Python lors du chargement et de calculs de statistique descriptive simple (filtrage, moyennes, calculs conditionnels, croisement de variables) sur un grand ensemble de données (9796862 observations et 42 variables).

R version 3.5.2 (de grandes améliorations depuis 3.5.0), avec les objets de base (data.frame), et ceux des packages spécialisés dans l'appréhension des fortes volumétries : ''data.table'' (1.12.0) ; ''readr'' (1.3.1) et ''dplyr'' (0.7.8) (qui font partie de ''tidyverse'').

Anaconda Python 3.7.1, avec le package ''pandas'' (0.23.4).

Machine Core i7-4770S (3.10 Ghz) sous Windows 10 64 bits (version éducation).

Fichiers CSV

CHARGEMENT DES DONNÉES

R – Package ‘‘Base’’

data.frame est le type ensemble de données (tableau individus x variables).
Fondamental pour le traitement statistique sous R. Intégré dans le package ‘‘base’’.

```
#chargement --> 99.06 sec.
system.time(D <-
read.csv("kddcup99twice.txt",header=FALSE))

#type --> 'data.frame'
print(class(D))

#description
print(str(D))

#occupation mémoire --> 2130.2 MB
#gestionnaire de tâches : 2259.6 Mo
print(object.size(D)/(1024*1024))
```

```
'data.frame':  9796862 obs. of  42 variables:
 $ V1 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V2 : Factor w/ 3 levels "icmp","tcp","udp": 2 2 2 2 2 2 2 2 2 2 ...
 $ V3 : Factor w/ 70 levels "aol","auth","bgp",..: 22 22 22 22 22 22 ...
 $ V4 : Factor w/ 11 levels "OTH","REJ","RSTO",..: 10 10 10 10 10 10 10 ...
 $ V5 : int  215 215 162 162 236 236 233 233 239 239 ...
 $ V6 : int  45076 45076 4528 4528 1228 1228 2032 2032 486 486 ...
 $ V7 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V8 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V9 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V10: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V11: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V12: int  1 1 1 1 1 1 1 1 1 1 ...
 $ V13: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V14: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V15: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V16: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V17: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V18: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V19: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V20: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V21: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V22: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V23: int  1 1 2 2 1 1 2 2 3 3 ...
 $ V24: int  1 1 2 2 1 1 2 2 3 3 ...
 $ V25: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V26: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V27: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V28: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V29: num  1 1 1 1 1 1 1 1 1 1 ...
 $ V30: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V31: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V32: int  0 0 1 1 2 2 3 3 4 4 ...
 $ V33: int  0 0 1 1 2 2 3 3 4 4 ...
 $ V34: num  0 0 1 1 1 1 1 1 1 1 ...
 $ V35: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V36: num  0 0 1 1 0.5 0.5 0.33 0.33 0.25 0.25 ...
 $ V37: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V38: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V39: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V40: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V41: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V42: Factor w/ 23 levels "back.","buffer_overflow.",..: 12 12 12 12 ...
```

R – Package ‘’data.table’’

Le package ‘’data.table’’ (version 1.12.0) propose des structures et des opérateurs optimisés pour la manipulation des grands ensembles de données, avec une grammaire spécifique.

```
#chargement – 13.24 sec.
system.time(D <-
data.table::fread("kddcup99twice.txt",header=FALSE))

#type --> "data.table" "data.frame"
print(class(D))

#description - 9796862 obs. of  42 variables
print(str(D))

#occupation mémoire - 2279.7 MB
#gestionnaire de tâches : 2521.3 Mo
print(object.size(D)/(1024*1024))
```

```
Classes ‘data.table’ and 'data.frame':      9796862 obs. of  42
variables:
 $ V1 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V2 : chr  "tcp" "tcp" "tcp" "tcp" ...
 $ V3 : chr  "http" "http" "http" "http" ...
 $ V4 : chr  "SF" "SF" "SF" "SF" ...
 $ V5 : int  215 215 162 162 236 236 233 233 239 239 ...
 $ V6 : int  45076 45076 4528 4528 1228 1228 2032 2032 486 486 ...
 $ V7 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V8 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V9 : int  0 0 0 0 0 0 0 0 0 0 ...
 $ V10: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V11: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V12: int  1 1 1 1 1 1 1 1 1 1 ...
 $ V13: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V14: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V15: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V16: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V17: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V18: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V19: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V20: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V21: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V22: int  0 0 0 0 0 0 0 0 0 0 ...
 $ V23: int  1 1 2 2 1 1 2 2 3 3 ...
 $ V24: int  1 1 2 2 1 1 2 2 3 3 ...
 $ V25: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V26: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V27: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V28: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V29: num  1 1 1 1 1 1 1 1 1 1 ...
 $ V30: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V31: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V32: int  0 0 1 1 2 2 3 3 4 4 ...
 $ V33: int  0 0 1 1 2 2 3 3 4 4 ...
 $ V34: num  0 0 1 1 1 1 1 1 1 1 ...
 $ V35: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V36: num  0 0 1 1 0.5 0.5 0.33 0.33 0.25 0.25 ...
 $ V37: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V38: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V39: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V40: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V41: num  0 0 0 0 0 0 0 0 0 0 ...
 $ V42: chr  "normal." "normal." "normal." "normal." ...
 - attr(*, ".internal.selfref")=<externalptr>
```

R – Packages ‘’dplyr’’ et ‘’readr’’

Les packages ‘’dplyr’’ et ‘’readr’’ font partie de l’univers ‘’tidyverse’’. Ils sont spécialisés dans l’appréhension et la manipulation des ensembles de données. Ici aussi, la syntaxe est spécifique.

```
#dplyr et readr font tous deux partie de tidyverse
#dplyr --> installe automatiquement tibble
library(dplyr)

#readr
library(readr)

#chargement - 51.37 sec.
system.time(D <-
read_csv("kddcup99twice.txt",col_names=FALSE))

#type --> "spec_tbl_df" "tbl_df" "tbl" "data.frame"
print(class(D))

#pour ne pas limiter l'affichage
options(tibble.print_min=Inf)

#description - 9796862 obs. of  42 variables
print(str(D))

#occupation mémoire - 3139.3 MB
#gestionnaire de tâches - 3742.6 Mo
print(object.size(D)/(1024*1024))
```

```
Classes ‘spec_tbl_df’, ‘tbl_df’, ‘tbl’ and 'data.frame':   9796862
obs. of  42 variables:
 $ X1 : num  0 0 0 0 0 0 0 0 0 0 ...
 $ X2 : chr  "tcp" "tcp" "tcp" "tcp" ...
 $ X3 : chr  "http" "http" "http" "http" ...
 $ X4 : chr  "SF" "SF" "SF" "SF" ...
 $ X5 : num  215 215 162 162 236 236 233 233 239 239 ...
 $ X6 : num  45076 45076 4528 4528 1228 ...
 $ X7 : num  0 0 0 0 0 0 0 0 0 0 ...
 $ X8 : num  0 0 0 0 0 0 0 0 0 0 ...
 $ X9 : num  0 0 0 0 0 0 0 0 0 0 ...
 $ X10: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X11: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X12: num  1 1 1 1 1 1 1 1 1 1 ...
 $ X13: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X14: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X15: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X16: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X17: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X18: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X19: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X20: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X21: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X22: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X23: num  1 1 2 2 1 1 2 2 3 3 ...
 $ X24: num  1 1 2 2 1 1 2 2 3 3 ...
 $ X25: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X26: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X27: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X28: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X29: num  1 1 1 1 1 1 1 1 1 1 ...
 $ X30: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X31: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X32: num  0 0 1 1 2 2 3 3 4 4 ...
 $ X33: num  0 0 1 1 2 2 3 3 4 4 ...
 $ X34: num  0 0 1 1 1 1 1 1 1 1 ...
 $ X35: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X36: num  0 0 1 1 0.5 0.5 0.33 0.33 0.25 0.25 ...
 $ X37: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X38: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X39: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X40: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X41: num  0 0 0 0 0 0 0 0 0 0 ...
 $ X42: chr  "normal." "normal." "normal." "normal." ...
```

Python – Package ‘’pandas’’

Le package ‘’pandas’’ est spécialisé dans l’appréhension et la manipulation des ensembles de données sous Python.

```
#mesurer le temps
import time

#specifier les noms
noms = ["V"+str(i) for i in range(1,43)]

#importer les données - 124.42 sec.
import pandas
tag = time.perf_counter()
D = pandas.read_csv("kddcup99twice.txt",header=None,names=noms)
print(time.perf_counter()-tag)

#type --> <class 'pandas.core.frame.DataFrame'>
print(type(D))

#info - RangeIndex: 9796862 entries, 0 to 9796861
#Data columns (total 42 columns):
#gestionnaire de tâches : 3646.8 Mo
print(D.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9796862 entries, 0 to 9796861
Data columns (total 42 columns):
V1     int64
V2     object
V3     object
V4     object
V5     int64
V6     int64
V7     int64
V8     int64
V9     int64
V10    int64
V11    int64
V12    int64
V13    int64
V14    int64
V15    int64
V16    int64
V17    int64
V18    int64
V19    int64
V20    int64
V21    int64
V22    int64
V23    int64
V24    int64
V25    float64
V26    float64
V27    float64
V28    float64
V29    float64
V30    float64
V31    float64
V32    int64
V33    int64
V34    float64
V35    float64
V36    float64
V37    float64
V38    float64
V39    float64
V40    float64
V41    float64
V42    object
dtypes: float64(15), int64(23), object(4)
memory usage: 3.1+ GB
```

Temps d'exécution pour une série de manipulation

CALCULS

Filtrage des lignes

Effectuer une requête avec des conditions sur les variables.

Data.Frame (R)

```
#restriction - 382696 - 0.44 sec.
system.time(print(nrow(D[D$V42=="normal." & D$V2=="udp",])))
```

Data.Table (R)

```
#restriction - 382696 - 0.12 sec.
system.time(print(D[V42=="normal." & D$V2=="udp",.(.N)]))
```

Dplyr (R)

```
#restriction - 382696 - 0.18 sec.
system.time(print(nrow(filter(D, X42=="normal.", X2=="udp"))))
```

Pandas (Python)

```
#restriction - 382696 - 1.44 sec.
tag = time.perf_counter()
print(D.loc[(D.V42=="normal.") & (D.V2=="udp"),:].shape[0])
print(time.perf_counter()-tag)
```

Calcul de moyennes

Identifier les variables numériques, calculer leurs moyennes.

Data.Frame (R)

```
#colonnes numériques
numeriques <- which(sapply(D,is.numeric))
#calculer les moyennes - 0.51 sec.
system.time(print(sapply(D[,numeriques],mean)))
```

Data.Table (R)

```
#calculer les moyennes - 0.5 sec.
system.time(print(D[,sapply(.SD,mean),,.SDcols=numeriques]))
```

Dplyr (R)

```
#calculer les moyennes - 0.74 sec.
system.time(print(sapply(D %>% select(numeriques),mean)))
```

Pandas (Python)

```
#moyennes pour les variables numeriques - 1.75 sec.
print(D.mean(axis=0,numeric_only=True))
```

Résultats

```
 v1           v5           v6           v7           v8           v9          v10          v11
4.834243e+01 1.834621e+03 1.093623e+03 5.716116e-06 6.487792e-04 7.961733e-06 1.243766e-02 3.205108e-05
         v12          v13          v14          v15          v16          v17          v18          v19
1.435290e-01 8.088304e-03 6.818510e-05 3.674646e-05 1.293496e-02 1.188748e-03 7.430951e-05 1.021143e-03
         v20          v21          v22          v23          v24          v25          v26          v27
0.000000e+00 4.082940e-07 8.351654e-04 3.349734e+02 2.952671e+02 1.779703e-01 1.780370e-01 5.766509e-02
         v28          v29          v30          v31          v32          v33          v34          v35
5.773010e-02 7.898842e-01 2.117961e-02 2.826080e-02 2.329811e+02 1.892142e+02 7.537132e-01 3.071111e-02
         v36          v37          v38          v39          v40          v41
6.050520e-01 6.464107e-03 1.780911e-01 1.778859e-01 5.792780e-02 5.765941e-02
```

Calcul de moyennes (bis)

Calculer les moyennes des variables numériques sur un sous-ensemble d'observations.

Data.Frame (R)

```
#calculer les moyennes avec une restriction - 0.72 sec.
system.time(print(sapply(D[D$V42=="normal.",numeriques],mean)))
```

Data.Table (R)

```
#calculer les moyennes avec une restriction - 0.25 sec.
system.time(print(D[V42=="normal.",sapply(.SD,mean),,.SDcols=numeriques]))
```

Dplyr (R)

```
#calculer les moyennes avec une restriction - 0.49 sec.
system.time(print(sapply(D %>% filter(X42=="normal.") %>% select(numeriques),mean)))
```

Pandas (Python)

```
#calculer les moyennes avec une restriction - 1.42 sec.
print(D.loc[D.V42=="normal.",:].mean(axis=0,numeric_only=True))
```

Résultats

```
 V1           V5           V6           V7           V8           V9          V10          V11
2.178247e+02 1.477846e+03 3.234650e+03 7.195864e-06 0.000000e+00 3.597932e-05 4.953530e-02 9.868614e-05
         V12          V13          V14          V15          V16          V17          V18          V19
7.192677e-01 3.838891e-02 3.104501e-04 1.840085e-04 6.497043e-02 5.887245e-03 3.628772e-04 5.130651e-03
         V20          V21          V22          V23          V24          V25          V26          V27
0.000000e+00 2.055961e-06 3.881655e-03 8.159029e+00 1.091279e+01 1.483057e-03 1.724581e-03 5.594070e-02
         V28          V29          V30          V31          V32          V33          V34          V35
5.620641e-02 9.852575e-01 1.853534e-02 1.324944e-01 1.484984e+02 2.020148e+02 8.448792e-01 5.650019e-02
         V36          V37          V38          V39          V40          V41
1.349402e-01 2.434030e-02 2.039328e-03 1.050185e-03 5.778443e-02 5.601557e-02
```

Trier l'ensemble de données selon les valeurs d'une variable numérique.

Data.Frame (R)

```
#trier selon la variable V34 - 3.23 sec.
system.time(print(head(D[order(D$V34),])))
```

Data.Table (R)

```
#trier selon la variable V34 - 1.14 sec.
system.time(print(head(D[order(V34),])))
```

Dplyr (R)

```
#trier selon la variable V34 - 6.56 sec.
system.time(print(D %>% arrange(X34) %>% head))
```

Pandas (Python)

```
#trier selon la variable V34 - 3.59 sec.
print(D.sort_values(by='V34').head())
```

Résultats

```
 V1  V2    V3 V4  V5    V6 V7 V8 V9 V10 V11 V12 V13 V14 V15 V16 V17 V18 V19 V20 V21 V22 V23 V24 V25 V26 V27
1      0 tcp  http SF 215 45076  0  0  0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0
2      0 tcp  http SF 215 45076  0  0  0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0
67727  0 tcp  http SF 162  4528  0  0  0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0
67728  0 tcp  http SF 162  4528  0  0  0   0   0   1   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0
83933  0 udp ntp_u SF  48    48  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0
83934  0 udp ntp_u SF  48    48  0  0  0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   1   0   0   0
       V28 V29 V30 V31 V32 V33 V34  V35 V36 V37 V38 V39 V40 V41     V42
1        0   1   0   0   0   0   0 0.00   0   0   0   0   0   0 normal.
2        0   1   0   0   0   0   0 0.00   0   0   0   0   0   0 normal.
67727    0   1   0   0   0   0   0 0.00   0   0   0   0   0   0 normal.
67728    0   1   0   0   0   0   0 0.00   0   0   0   0   0   0 normal.
83933    0   1   0   0 255   1   0 0.01   0   0   0   0   0   0 normal.
83934    0   1   0   0 255   1   0 0.01   0   0   0   0   0   0 normal.
```

Comptage

Fréquences absolues des valeurs d'une variable catégorielle.

Data.Frame (R)

```
#comptage de V42 - 0.20 sec.
system.time(print(table(D$V42)))
```

Data.Table (R)

```
#comptage de V42 - 0.13 sec.
system.time(print(D[,.(.N),by=.(V42)]))
```

Dplyr (R)

```
#comptage de V42 - 0.38 sec.
system.time(print(D %>% group_by(X42) %>% summarise(n())))
```

Pandas (Python)

```
#comptage de V42 - 1.07 sec.
print(D.V42.value_counts())
```

Résultats

	V42	N
1:	normal.	1945562
2:	buffer_overflow.	60
3:	loadmodule.	18
4:	perl.	6
5:	neptune.	2144034
6:	smurf.	5615772
7:	guess_passwd.	106
8:	pod.	528
9:	teardrop.	1958
10:	portsweep.	20826
11:	ipsweep.	24962
12:	land.	42
13:	ftp_write.	16
14:	back.	4406
15:	imap.	24
16:	satan.	31784
17:	phf.	8
18:	nmap.	4632
19:	multihop.	14
20:	warezmaster.	40
21:	warezclient.	2040
22:	spy.	4
23:	rootkit.	20

Moyennes conditionnelles (1)

Moyenne d'une variable numérique conditionnellement aux valeurs d'une variable catégorielle.

Data.Frame (R)

```
#moyenne de V34 selon V42 - 0.12 sec.
system.time(print(tapply(X=D$V34,INDEX=D$V42,mean)))
```

Data.Table (R)

```
#moyenne de V34 selon V42 - 0.18 sec.
system.time(print(D[,.(mean(V34)),by=.(V42)]))
```

Dplyr (R)

```
#moyenne de V34 selon V42 - 0.44 sec.
system.time(print(D %>% group_by(X42) %>% summarise(mean(X34))))
```

Pandas (Python)

```
#moyenne de V34 selon V42 - 0.47 sec.
print(D.groupby('V42')['V34'].mean())
```

Résultats

	V42	mean(V34)
1:	normal.	0.844879197
2:	buffer_overflow.	1.000000000
3:	loadmodule.	0.835555556
4:	perl.	0.013333333
5:	neptune.	0.043371318
6:	smurf.	0.999691383
7:	guess_passwd.	1.000000000
8:	pod.	0.659734848
9:	teardrop.	0.246833504
10:	portsweep.	0.002928071
11:	ipsweep.	0.930468712
12:	land.	0.870000000
13:	ftp_write.	0.875000000
14:	back.	1.000000000
15:	imap.	0.916666667
16:	satan.	0.014214699
17:	phf.	0.972500000
18:	nmap.	0.526735751
19:	multihop.	0.715714286
20:	warezmaster.	0.900000000
21:	warezclient.	0.735441176
22:	spy.	0.185000000
23:	rootkit.	0.306000000

Moyennes conditionnelles (2)

Moyenne d'une variable numérique conditionnellement aux combinaisons de deux variables catégorielles.

Data.Frame (R)

```
#moyenne de V34 selon (V42, V2) - 0.20 sec.
system.time(print(tapply(X=D$V34,INDEX=list(D$V42,D$V2),mean)))
```

Data.Table (R)

```
#moyenne de V34 selon (V42, V2) - 0.21 sec.
system.time(print(D[,.(mean(V34)),by=.(V42,V2)]))
```

Dplyr (R)

```
#moyenne de V34 selon (V42, V2) - 0.59 sec.
system.time(print(D %>% group_by(X42,X2) %>% summarise(mean(X34))))
```

Pandas (Python)

```
#moyenne de V34 selon (V42, V2) - 1.52 sec.
print(pandas.crosstab(index=D.V42,columns=D.V2,values=D.V34,aggfunc=pandas.Series.mean))
```

Résultats

	V42	V2	mean(V34)
1:	normal.	tcp	0.9048007988
2:	normal.	udp	0.6258512239
3:	normal.	icmp	0.5197696466
4:	buffer_overflow.	tcp	1.0000000000
5:	loadmodule.	tcp	0.8355555556
6:	perl.	tcp	0.0133333333
7:	neptune.	tcp	0.0433713178
8:	smurf.	icmp	0.9996913835
9:	guess_passwd.	tcp	1.0000000000
10:	pod.	icmp	0.6597348485
11:	teardrop.	udp	0.2468335036
12:	portsweep.	tcp	0.0027346978
13:	ipsweep.	tcp	0.0684090909
14:	land.	tcp	0.8700000000
15:	ftp_write.	tcp	0.8750000000
16:	back.	tcp	1.0000000000
17:	imap.	tcp	0.9166666667
18:	satan.	icmp	0.2097297297
19:	satan.	udp	0.1214988290
20:	satan.	tcp	0.0007506892
21:	phf.	tcp	0.9725000000
22:	ipsweep.	icmp	0.9993917107
23:	nmap.	icmp	0.9960077519
24:	nmap.	tcp	0.0035589942
25:	multihop.	tcp	0.7157142857
26:	nmap.	udp	0.7534400000
27:	warezmaster.	tcp	0.9000000000
28:	warezclient.	tcp	0.7354411765
29:	spy.	tcp	0.1850000000
30:	rootkit.	tcp	0.1514285714
31:	portsweep.	icmp	0.3383333333
32:	rootkit.	udp	0.6666666667

Performances comparées

RÉCAPITULATIF

Récapitulatif des durées d'exécution – Tous ont mené à bien les calculs !

Voilà un tableau qui ne manquera pas de faire réagir…

	Temps d'exécution en secondes			
Action	R (Base)	R (Data.Table)	R (Readr / Dplyr)	Python (Pandas)
Chargement du fichier	99.06	13.24	51.37	124.42
Filtrage	0.44	0.12	0.18	1.44
Moyennes	0.51	0.50	0.74	1.75
Filtrage + Moyennes	0.72	0.25	0.49	1.42
Tri sur un critère	3.23	1.14	6.56	3.59
Comptage	0.20	0.13	0.38	1.07
Moyennes conditionnelles (1)	0.12	0.18	0.44	0.47
Moyennes conditionnelles (2)	0.20	0.21	0.59	1.52

Mesures des durées d'exécution :

- Pour le chargement de fichier, la durée est mesurée une seule fois, après un redémarrage de l'EDI (RStudio / Spyder)
- Pour les autres actions, il s'agit de la médiane pour 10 exécutions (on note que le premier appel est toujours plus long – parfois significativement – que les suivants).

RÉFÉRENCES

Références

The R Project for Statistical Computing (R 3.5.2) -- https://www.r-project.org/

''Introduction to data.table'' - https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

''dplyr tutorial'' -- http://genomicsclass.github.io/book/pages/dplyr_tutorial.html

''Tidyverse packages'' -- https://www.tidyverse.org/packages/

Anaconda Distribution (Python 3.7) -- https://www.anaconda.com/distribution/

Python Data Analysis Library – Pandas -- https://pandas.pydata.org/