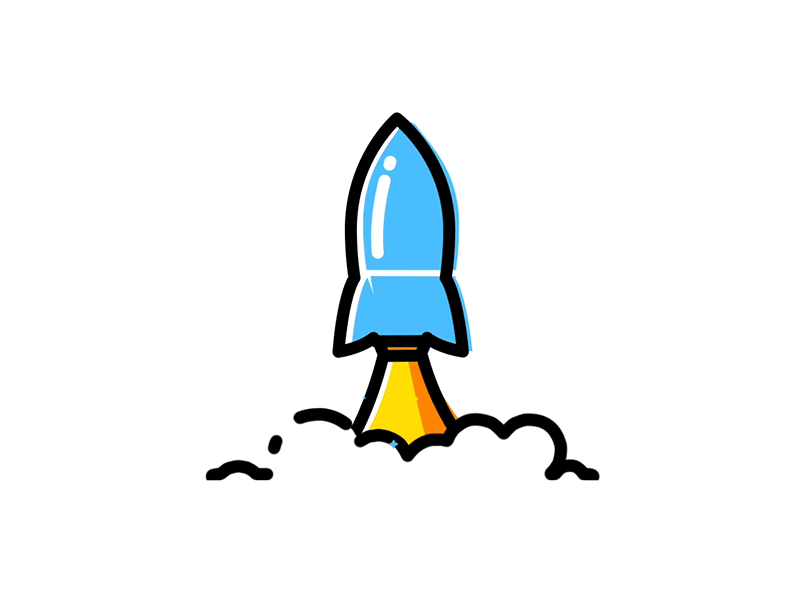
Gtools

Table of contents

 Version 1 avec **gtools** [Aout 2022]

* Stata 17: grosse amélioration du temps d’exécution de certaines commandes, en particulier **sort** et **collapse**.
* Depuis de nombreuses années des packages ou commandes standalone amélioraientt le temps d’exécution, en particulier les packages **ftools** de *Sergio Correa* ou les commandes fastxtile/fastwpctile d’**egenmisc**.
* Le package **gtools** de *Mauricio Caceres Bravo* donne des résultats vraiment très intéressants lorsqu’on atteint un seuil d’un million d’observations pour les commandes suivantes: greshape, gquantiles, gegen, glevelof avec une variable caractère, et dans une moindre mesure gcollapse.
* Au delà des durées d’exécution, ces packages et commandes peuvent avoir quelques options propres, par exemple, l’option by() de *gquantiles* ou la possibilité d’enregistrer les valeurs en ordre décroissant avec glevelsof.

**Benchmarks**

* Version Stata 17 SE. Les benchmarks réalisés par *Mauricio Caceres* sont en version MP.
* Configuration PC: i5-10210U CPU et 16GO de RAM.
* Volumétries: 10k, 100k, 1M, 10M.
* Comparaisons avec R si fonctions équivalentes.
* Programme Stata: [programme](programme.do)
* To do: faire les tests sur la version serveur-linux (toujours Stata 17 SE)

**Sources**:

* **Stata 17 faster**: <https://www.stata.com/new-in-stata/faster-stata-speed-improvements/>
* **ftools**: <https://github.com/sergiocorreia/ftools>
* **gtools**:
  + <https://gtools.readthedocs.io/en/latest/index.html>
  + <https://github.com/mcaceresb/stata-gtools>

# 1. **Le package**

Auteur: ***Mauricio Caceres Bravo***

***Installation***:

* <https://gtools.readthedocs.io/en/latest/index.html>
* <https://github.com/mcaceresb/stata-gtools>

Les Benchmarks réalisés par l’auteur ont été exécutés avec Stata MP. J’ai fait tourné son programme ([lien](https://raw.githubusercontent.com/mcaceresb/stata-gtools/master/docs/benchmarks/quick.do)) avec Stata 17 SE sous windows. Les résultats sont les suivants:

Versus | Native | gtools | % faster   
 ---------- | ------ | ------ | --------   
 collapse | 1.53 | 1.25 | 18.51%   
 collapse | 1.68 | 1.17 | 29.91%   
 reshape | 31.63 | 6.90 | 78.19%   
 reshape | 60.26 | 10.95 | 81.83%   
 xtile | 17.74 | 1.12 | 93.67%   
 pctile | 18.20 | 0.77 | 95.76%   
 egen | 2.13 | 0.64 | 69.77%   
 contract | 4.52 | 1.74 | 61.54%   
 isid | 18.71 | 0.68 | 96.35%   
 duplicates | 10.07 | 0.86 | 91.42%   
 levelsof | 2.75 | 0.44 | 83.94%   
 distinct | 7.24 | 0.44 | 93.88%   
 winsor | 16.09 | 0.65 | 95.99%   
 sum\_detail | 17.09 | 1.22 | 92.86%   
 tabstat | 11.18 | 0.67 | 94.03%   
 range\_stat | 67.37 | 2.82 | 95.81%

Pour mon propre benchmark, plus gourmand (10 variables quanti et une variable binaire), les données sont générées de la manière suivante:

**Création de la base de données (N=10M)**

clear   
set obs 10000000  
tempvar x  
gen `x' = runiform()  
gen g = `x'>.5  
  
forv i=1/10 {  
gen y`i' = rnormal()   
   
gen id = \_n   
}

Pour récupérer les durées d’exécution, j’utilise un fragment du programme de *M.Caceres*. Les commandes sont exécutées avec le prefixe bench 1:

capture program drop bench  
program bench  
 gettoken timer call: 0, p(:)  
 gettoken colon call: call, p(:)  
 cap timer clear `timer'  
 timer on `timer'  
 `call'  
 timer off `timer'  
 qui timer list  
 c\_local r`timer' `=r(t`timer')'  
end

* Les tests sont réalisés avec les équivalents de xtile, reshape, collapseet levelsof. L’équivalent à tabstat sera ajouté rapidement.
* Pour information, les programmes des fonctions R sont également rapidement décris. Les durées d’exécution ont été récupérés avec la librairie tictoc.

# 2. **gquantiles**

* Commande usine xtile et pctile (help xtile). Le benchmark est seulement effectué pour xtile (affectation d’un quantile à une valeur) qui est plus gourmant que pctile (calcul et report des quantiles).
* En termes d’options, l’autre intérêt de **gquantile** est de stratifier l’opération avec l’option by().

***Syntaxe courte***

\*xtile  
gquantiles nouvelle\_var = var1 , xtile nq(#) [by(var2)]  
  
\*pctile  
gquantiles nouvelle\_var = var1 , pctile nq(#) [by(var2)]

***Programme***

\* Fonction bench (voir plus haut)  
  
qui forv i=1/10 {  
   
\*\* XTILE  
   
tempvar yg`i'  
bench 1: xtile `yg`i'' = y`i' , nq(10)   
local rt1 = `rt1' + `r1'   
 }  
di "XTILE runtime =" `rt1'  
  
\*\*\* GQUANTILES  
qui forv i=1/10 {  
capt drop `yg`i''   
tempvar yg`i'  
bench 1: gquantiles `yg`i'' = y`i' , xtile nq(10)   
local rt2 = `rt2' + `r1'   
 }  
di "GQUANTILES runtime =" `rt2'

***Résultats*** (secondes)

| Stata | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| xtile | 0.12 | 1.65 | 16.03 | 196.56 |
| **gquantiles** | 0.06 | 0.22 | 1.24 | 14.75 |

| R | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| quantcut | 0.04 | 0.24 | 2.38 | 29.11 |
| ntile | 0.06 | 0.16 | 1.54 | 15.51 |

|  |
| --- |
| Fonctions R |
| * **quantcut**   + librairie gtools   + Syntaxe pour la variable y1: **df$gy1=quantcut(df$y1,10)** * **ntile**   + librairie dplyr   + Syntaxe pour la variable y1: **df=df %>% mutate(gy1 = ntile(y1, 10))** |

# 3. **greshape**

* Niveau syntaxe peu de différence avec la commande usine, si ce n’est pour les arguments i() et j()
  + i() = id()
  + j() = key()
* Pour R:
  + Fonction de base reshape.
    - Avantage: syntaxe très proche de Stata
    - Inconvénients: temps d’exécution pas optimal. Pour 10M d’observations, j’ai arrêté l’exécution au bout de 10 minutes.
  + Fonctions pivot\_longer et pivot\_wider de \*\*tydir.

Si **greshape** est nettement plus performant que **reshape**, il reste nettement en deçà des deux fonctions de la librairie **tydir** de R.

***Programme***

\* Fonction bench (voir plus haut)  
  
\*\*RESHAPE  
qui bench 1: reshape long y, i(id) j(j)  
di "RESHAPE LONG runtime =" `r1'  
qui bench 1: reshape wide y, i(id) j(j)  
di "RESHAPE WIDE runtime =" `r1'  
  
\*\*GRESHAPE  
qui bench 1: greshape long y, by(id) keys(j)  
di "GRESHAPE LONG runtime =" `r1'  
qui bench 1: greshape wide y, by(id) keys(j)  
di "GRESHAPE WIDE runtime =" `r1'

***Résultats*** (secondes)

## **Long**

| Stata | 10K | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| reshape long | 0.14 | 1.22 | 12.36 | 245.18 |
| **greshape long** | 0.04 | 0.21 | 3.22 | 61.23 |

| R | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| reshape | 0.1 | 1.19 | 11.9 | /// |
| pivot\_longer | 0.01 | 0.12 | 0.6 | 13.39 |

## **Wide**

| Stata | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| reshape wide | 0.37 | 2.18 | 26.58 | 338.10 |
| **greshape wide** | 0.06 | 0.30 | 2.79 | 55.86 |

| R | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| reshape | 0.37 | 3.69 | 34.93 | /// |
| pivot\_wider | 0.01 | 0.24 | 1.98 | 38.97 |

|  |
| --- |
| Fonctions R |
| * **reshape**   + Installé avec R   + Long: long = reshape(gtools, idvar = "id", timevar="j", varying = list(2:11), v.names = "y", direction = "long")   + Wide: wide = reshape(long, idvar = "id", timevar="j", v.names = "y", sep = "", direction = "wide") * **pivot\_longer/pivot\_wider**   + librairie tydir   + long: long = pivot\_longer(gtools, cols = starts\_with("y"))   + wide: wide = pivot\_wider(long, names\_from = c("name"), values\_from = c("value")) |

# 4. **gcollapse**

* Syntaxe identique à celle de collapse. Par défaut, c’est également la moyenne qui est calculée.
* Ajout d’une option merge replace qui remplace la valeur des observations par l’indicateur séléctionné.
* On ajouté l’option by() sur la variable g (deux groupes).

***Programme***

\*\*\* COLLAPSE  
preserve  
qui bench 1: collapse y1-y10, by(g)  
local col `r1'  
restore  
  
\*\*\* GCOLLAPSE  
preserve  
qui bench 1: gcollapse y1-y10, by(g)  
local gcol `r1'   
restore  
  
di "N=`N"  
di "COLLAPSEruntime =" `col'  
di "GCOLLAPSEruntime =" `gcol'

***Résultats*** (secondes)

| Stata | 10K | 100K | 1M | 10 M |
| --- | --- | --- | --- | --- |
| collapse | 0.007 | 0.041 | 0.461 | 7.846 |
| **gcollapse** | 0.021 | 0.049 | 0.219 | 2.559 |
| **R** | **10K** | **100K** | **1M** | **10 M** |
| summarise | 0.03 | 0.06 | 0.3 | 1.91 |

*Note*: pour Stata le programme exécute preserve/restore, ce qui augmente légèrement un temps d’exécution

|  |
| --- |
| Fonction R |
| * **summarise()**   + librairie dplyr   + Syntaxe : **collapse= gtools %>% group\_by(g) %>% summarise(across(y1:y10, ~ mean(.x, na.rm = TRUE)))** |

***Programme***

# 5. **gegen**

* Syntaxe identique à celle d’egen. On a choisi comme fonction la moyenne.
* On ajouté l’option by() sur la variable g (deux groupes).

forv i=1/10 {  
qui bench 1: egen my`i' = mean(y`i'), by(g)  
local egen = `egen' + `r1'   
}  
  
drop my\*  
  
forv i=1/10 {  
qui bench 1: gegen my`i' = mean(y`i'), by(g)  
local gegen = `gegen' + `r1'   
}  
  
di "N=`N"  
di "EGEN runtime =" `egen'  
di "GEGEN runtime =" `gegen'

| Stata | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| egen | 0.23 | 0.41 | 4.82 | 73.6 |
| **gegen** | 0.69 | 0.20 | 0.83 | 8.88 |
| **R** | **10k** | **100k** | **1M** | **10M** |
| mutate + mean | 0.03 | 0.05 | 0.17 | 1.74 |

|  |
| --- |
| Fonction R |
| * **mutate()** associée à la fonction mean   + librairie dplyr   + Syntaxe :     - **var <- c("y1", "y2", "y3", "y4", "y5", "y6", "y7", "y8", "y9", "y10")**     - **gtools = gtools %>% group\_by(g) %>% mutate(across(var, mean, .names = "m{col}"))** |

# 6. **glevelsof**

|  |
| --- |
| Rappel |
| * La commande **levelsof** (help levelsof) permet de récupérer automatiquement les valeurs d’une variable pour les transformer sous forme de macro. Par défaut la macro enregistrée est nommée r(levels), il est possible de l’appeler différemment avec l’option local(). Elle est particulièrement utile en amont d’une opération en boucle de type foreach. La macro générée r(r) permet de récupérer le nombre de valeurs enregistrés, et peut donc être utile pour des instructions en boucle de type forvalue (et évite de programmer une macro avec la fonction word count plus loin). * Les valeurs sont enregistrées par ordre croissant numérique ou alphabétique selon le type de variable. |

**glevelsof**

* Autorise plusieurs variables. la macro enregistrée concaténera les valeurs et/ou expression avec un séparateur (espace par défaut).
* Permet de trier les valeurs en ordre décroissant en ajoutant **-** devant le nom de la variable.

**benchmark**

* Bien évidemment, pas de comparaison possible avec R
* Programme d’origine différent: on va générer une variable qui affecte aléatoirement une lettre de l’alphabet (une version caractère et une version numérique générée avec encode). Le programme a été écrit par *Paul Picard* sur le forum *Statalist* ([lien](https://www.statalist.org/forums/forum/general-stata-discussion/general/1421124-generate-random-strings-containing-letters-and-numbers))

clear  
set obs 10000  
local c2use ABCDEFGHIJKLMNPQRSTUVWXYZ  
gen random\_string = substr("`c2use'", runiformint(1,length("`c2use'")),1) + ///  
 string(runiformint(0,9)) + ///  
 char(runiformint(65,90)) + ///  
 char(runiformint(65,90)) + ///  
 string(runiformint(0,9)) + ///  
 char(runiformint(65,90))  
  
gen xchar = substr(random\_string,1,1)  
encode xchar, gen(xnum)  
drop random\_string

Levelsof :

levelsof xchar  
  
/\*  
`"A"' `"B"' `"C"' `"D"' `"E"' `"F"' `"G"' `"H"' `"I"' `"J"' `"K"' `"L"' `"M"' `"N"' `"P"' `"Q"' `  
> "R"' `"S"' `"T"' `"U"' `"V"' `"W"' `"X"' `"Y"' `"Z"'  
\*/  
  
levelsof xnum  
  
/\*  
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25  
\*/

Glevelsof avec valeurs enregistrées en ordre décroissant:

glevelsof -xchar  
  
`"Z"' `"Y"' `"X"' `"W"' `"V"' `"U"' `"T"' `"S"' `"R"' `"Q"' `"P"' `"N"' `"M"' `"L"' `"K"' `"J"'   
` "I"' `"H"' `"G"' `"F"' `"E"' `"D"' `"C"' `"B"' `"A"'  
  
glevelsof -xnum  
  
25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1

| **Variable caractère** | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| levelsof | 0.01 | 0.10 | 2.64 | 42.51 |
| **glevelsof** | 0.01 | 0.01 | 0.11 | 0.62 |

| **Variable numerique** | 10k | 100k | 1M | 10M |
| --- | --- | --- | --- | --- |
| levelsof | 0.01 | 0.01 | 0.09 | 1.04 |
| **glevelsof** | 0.00 | 0.01 | 0.04 | 0.32 |