Efficient Stata For Big Data

Stata is built over a matrix-oriented language called Mata. When these matrices (datasets) get large, there is significant overhead that both slows down the processes and increases memory load. This overhead can appear in a variety of ways, such as memory preservation (backups), loops over variables or values, and hidden sorts.

Mauricio Caceres's `gtools` library can speed up many processes by running them in C. This reduces overhead and even outperforms Sergio Correia's `ftools`. This speedup is even larger for processes performed over groups using `by()` or `groupby()`, which native Stata commands sometimes do not even support. Most speedups are visible in Stata MP with data containing more than 10 million observations.

This document provides a mapping of native Stata commands to the analogous efficient `gtools` suite, as well as other helpful suggestions for using Stata with big data. The syntax for the `gtools` analog is generally the same as the Stata command, with a few notable exceptions.

See all available commands in the docs at: https://gtools.readthedocs.io/en/latest/

Stata Command	Gtools	Improvement
collapse	**gcollapse** clist [, options]	New statistic functions.
isid	**gisid** varlist [,missok]	
levelsof	**glevelsof** varlist	Custom sorts.
sort, gsort	**hashsort** [+|-] varname [[+|-] varname ...]	
egen	**gegen** [type] newvar = fcn(arguments)	New functions.
xtile, pctile	**gquantiles** [newvar=] exp {pctile|xtile|_pctile}]	
winsor	**gstats winsor** varlist [, by(varlist) options]	
tabstat	**gstats tab,** by(varlist)	New statistic functions.
tabulate	**gstats tab** id, s(count) by(var)	New statistic functions.
summarize	**gstats tab** varlist	Customizable for faster calculation
summarize, detail	**gstats sum** varlist	Group support
reshape	**greshape** long stubnames, by(varlist) **greshape** gather varlist, keys(varlist) values(varname) **greshape** wide stubnames, by(varlist) keys(varname) **greshape** spread varlist, keys(varname)	Supports R syntax (shown). Customizable: can skip sorts, preserves and missing value checks for faster processing.
unique	**gunique** varlist [if] [in] [, detail]	

Additional Suggestions

Be intentional about data types. The default data type for `generate` is `float`. If the data is certain to fit into something smaller, set the type when generating the variable rather than compressing later. Be careful, however, as placing a value in a smaller data type truncates the bits and returns a much different value. This is a safe approach for indicators (which fit in `byte`) and years (which fit in `int`).

```
gen byte red_apple = (fruit==2 & color==3)
```

Refrain from using strings. Strings are bit-hungry. If the data contain a categorical string, consider encoding instead. This replaces the string with a number (usually a `byte`) and places a value label over the number in the metadata. This significantly reduces the space requirements for the variable.

```
encode fruit_s, gen(fruit)

encode color_s, gen(color)
```

If the data are already represented by a categorical number:

```
label define fruit_label 1 "apple"  2 "pear"

label define color_label 1 "red" 2 "yellow" 3 "green"

label values fruit fruit_label

label values color color_label
```

Pre-sort the data. If the `by()` operations tend to group the same observations, pre-sort the data first. `hashsort` (the sorting function under the hood of every `gtools` function) checks if the data are already sorted prior to starting the algorithm.

Save intermediate files to disk. These are good checkpoints if something goes awry and may reduce the amount of data needed in memory at a single time.

Don't use preserve. `preserve` makes a second copy of the data and is used in a lot of Stata commands like `reshape` and `collapse`. `Gtools` allows the user to bypass this step, which is a significant speed and memory improvement. For temporary multitasking, use `frames`. This allows for quick multitasking, viewing, and editing of data within the same Stata instance.

```
frame create f1

frame copy default f1

frame drop f1

frame f1: gen byte red_apple = (fruit==2 & color==3)
```

Only use variables as needed. Don't load a full dataset if not necessary. Use only the variables needed for the analysis so transformations don't include superfluous data, which increases overhead.

```
use id year amount using data.dta if year > 2000
```