The Stata Journal

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*, *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		**Printed & electronic**	
1-year subscription	$ 98	1-year subscription	$138
2-year subscription	$165	2-year subscription	$245
3-year subscription	$225	3-year subscription	$345
1-year student subscription	$ 75	1-year student subscription	$ 99
1-year institutional subscription	$245	1-year institutional subscription	$285
2-year institutional subscription	$445	2-year institutional subscription	$525
3-year institutional subscription	$645	3-year institutional subscription	$765
Electronic only		**Electronic only**	
1-year subscription	$ 75	1-year subscription	$ 75
2-year subscription	$125	2-year subscription	$125
3-year subscription	$165	3-year subscription	$165
1-year student subscription	$ 45	1-year student subscription	$ 45

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

The Stata Journal (2013)
13, Number 4, pp. 867–875

group2: Generating the finest partition that is coarser than two given partitions

Christian H. Salas Pauliac
Department of Public Policy
University of Chicago
Chicago, IL
chsp@uchicago.edu

Abstract. In this article, I develop a useful interpretation of the function `group()` based on partitions belonging to mathematical set theory, an interpretation that in turn engenders a related command here called `group2`. In the context of the partitioning of sets, while the function `group()` creates a variable that generates the coarsest partition that is finer than the finest partition generated by the variables used as arguments, the `group2` command will create a variable that generates the finest partition that is coarser than the coarsest partition generated by the variables used as arguments. This latter operation has proven very useful in several problems of database management. An introduction of this new command in the context of mathematical partitions is provided, and two examples of its application are presented.

Keywords: dm0073, group2, partitions, group, egen

1 Introduction

The `egen` function `group()` generates a variable that takes numerical values that indicate "groups" of observations generated by the list of variables in the *varlist*. In this context, a group is understood to be observations that share the same value for every one of the variables in *varlist*. In other contexts, however, a group might be understood to be observations that share the same value for any one of the variables in *varlist*. An alternative interpretation of these two contexts is based on mathematical partitions, where the total number of observations in a database is understood to be a set and each variable is understood to be a partition whose cells are defined by the different values of each variable. Groups in the first context generate the coarsest partition that is finer than the finest partition generated by the variables in *varlist*; groups in the second context generate the finest partition that is coarser than the coarsest partition generated by the variables in *varlist*.

In this article, I will introduce a simple command called `group2`, which generates a variable that takes numerical values that indicate the groups as understood in the latter case. I will do so by relating both the `group()` function and the `group2` command to mathematical set theory and by motivating the use of the new command with two real-life examples.

2 A set-theory interpretation of the function group()

A brief overview of partitions in the context of set theory will come in handy. Let X be a set of elements. The set P of nonempty subsets $A, B, \ldots$ is a partition of X if and only if the following two conditions hold for any $A, B \in P$, where $A \neq B$:

$$\bigcup P = X$$
$$A \cap B = \emptyset$$

Subsets $A, B, \ldots$ of partition P are usually called cells of P. Simply, a partition of a set X is a fragmentation or grouping of the elements of X; different partitions will generate different fragmentation patterns. Let P_1 and P_2 be two different partitions of X. Partition P_1 is said to be finer than partition P_2 (and P_2 coarser than P_1) if every cell of P_1 is a subset of some cell of P_2. In other words, P_1 is a further fragmentation of P_2. For example,

$$X = \{a, b, c, d, e, f\}$$
$$P_1 = \{\{a, b\}, \{c\}, \{d, e, f\}\}$$
$$P_2 = \{\{a, b, c\}, \{d, e, f\}\}$$

Sometimes, such relation cannot be established between two partitions. For example, partitions P_3 and P_4 group the elements of X in a manner such that two cells of one partition overlap with more than one cell of the other partition. Two useful operations in this case are to find the coarsest partition that is finer than the finest of the two original partitions (call this P_5) and to find the finest partition that is coarser than the coarsest of the two original partitions (call this P_6). Intuitively, P_5 cuts through the cell overlaps, generating a partition whose cells are contained in no more than one cell of the original partitions, and P_6 combines the cell overlaps so that all the cells of the original partitions are contained in no more than one cell of P_6. For example,

$$P_3 = \{\{a, b\}, \{c, d\}, \{e, f\}\}$$
$$P_4 = \{\{a\}, \{b, c\}, \{d\}, \{e, f\}\}$$
$$P_5 = \{\{a\}, \{b\}, \{c\}, \{d\}, \{e, f\}\}$$
$$P_6 = \{\{a, b, c, d\}, \{e, f\}\}$$

How does this framework relate to the Stata function `group()`? Consider the total number of observations in a database to be the set X. Each variable of the database can be interpreted to be a partition P of the set X, where the cells of the partition are defined by the different values of each variable. For example, if one of the variables in the database takes the values 0 and 1, this variable generates a partition of X consisting of two cells, one that contains all observations for which this variable is equal to 0 and one that contains all observations for which it is equal to 1.

When the function `group()` takes two or more variables as arguments, it generates the coarsest partition that is finer than the finest partition generated by these arguments.

The following example serves as illustration. Say that we are working with a database containing survey information on households and that we are interested in two variables: `location`, indicating whether it is an urban or a rural household, and `gender`, indicating whether the head of the household is male or female. Each of these variables generates a different partition of the total number of observations, each according to a different criterion. When using these two variables as arguments, the `group()` function will generate a numerical variable identifying four groups containing all the possible types of households using these two variables: urban–male, urban–female, rural–male, and rural–female.[1] This new variable effectively generates the coarsest partition that is finer than the finest partition generated by the `location` and `gender` variables.

3 The group2 command

3.1 Syntax

The syntax for the `group2` command is as follows:

`group2` *varlist*

Exactly two variables must be specified in *varlist*. No options are allowed.

3.2 Description

If we consider the sample of observations in a dataset to be a particular set, the `group2` command creates a variable that generates the finest partition (of this set) that is coarser than the coarsest partition generated by the two variables in *varlist*.

Exactly two variables must be specified in *varlist*. Both numerical and string variables are allowed. Missing values in *varlist* (either `.` or `""`) are treated as if each one were a unique value, thereby indicating separate partitions. If n variables are needed in *varlist*, where $n > 2$, the command needs to be applied $n - 1$ times, where the third variable will be run with the outcome variable from applying the command to the first two variables, and so forth.

3.3 Remarks

The `group2` command generates a variable that takes numerical values, each one indicating a different group of observations, just as the `egen` function `group()` does. However, while `group()` understands a group to be all observations that share the same value for

1. Actually, the `group()` function may generate two, three, or four groups depending on the nature of the original partitions. For example, if all urban households have a male head of the family and all rural households have a female head of the family, then the two partitions generated by these variables are equivalent, and the `group()` function will only replicate such partitions. On the other hand, if all urban households have a male head of the family but rural households have both male and female heads of household, then the `group()` function will generate three groups.

every one of the variables in *varlist*, `group2` understands a group to be all observations that share the same value for any one of the variables in *varlist*. In other words, `group()` creates a variable that generates the coarsest partition (of the set of observations) that is finer than the finest partition generated by the variables in *varlist*; `group2` creates a variable that generates the finest partition (of the set of observations) that is coarser than the coarsest partition generated by the variables in *varlist*.

Table 1 compares the use of the `group()` function with the `group2` command as applied to two variables (`var1` and `var2`) of a fictitious sample.

Table 1. `group()` versus `group2`

`var1`	`var2`	`group()`	`group2`
1	A	1	1
1	B	2	1
2	B	3	1
2	C	4	1
3	D	5	2
3	D	5	2

4 Example

Several contexts in database management will require the finest partition that is coarser than the coarsest partition generated by two or more variables. The new command `group2` will perform such an operation. Let us illustrate two practical applications of this command with two examples that in fact motivated the creation of the program.

4.1 Example 1: Generating identification from several sources

Let us imagine a database containing information on employed individuals (henceforth called workers), where each observation is a worker and each variable is a different characteristic of the worker. We know that several workers are duplicated in the database; that is, several observations may be referring to the same worker. This might happen in several contexts. One example is when appending several databases containing different subsets of a population and where the intersection of these subsets might not always be empty. Another example happens in network databases in a panel form, where each principal declares several network members and where one network member might be declared by more than one principal. In a long panel form where the principal is duplicated as many times as the number of network members it has, several network members might be repeated.

We wish to create a unique ID for each worker; note that this ID variable will generate a partition of the database where each cell will contain observations corresponding to the same individual. A simple way of doing this is to identify duplicates in terms of all variables. Unfortunately, if some of the variables that would be useful to identify the worker (for example, home address) have missing values or are likely to be misspelled for some of the observations, none (or very few) of the observations will be an exact duplicate of any other in terms of all the variables.

An alternative way of generating this ID is to take a subset of the variables, one that is certain to uniquely identify a worker, and spot all duplicated observations in terms of this subset. The ID created in this exercise will generate a partition where each cell will contain observations corresponding to the same worker in terms of this subset of variables. The problem with this strategy is that there might be different subsets of variables uniquely identifying a worker that might generate ID assignments not equivalent to one another. If all ID assignments are right, we need an operation that uses the information contained in every assignment to produce an overall unique identification.

Such an operation is performed by the `group2` command. Each ID assignment generates a partition over the set of workers in the sample; we know that each cell in every partition contains observations corresponding to a unique individual. If partitions are not equivalent, then at least one cell of one partition will contain, be contained by, or cross one or more cells of another partition. For cells in this situation, one partition, P_a, will inform us that individuals contained in two different cells belonging to another partition, P_b, are the same worker. This will happen to several cells in every partition available. Therefore, the overall unique identification will emerge when all crossing cells are combined, which is precisely the finest partition that is coarser than the coarsest partition generated by the separate ID assignment.

Say we have four variables that provide information on workers: the name of the institution where the worker is employed (`workplace`), the worker's full name (`name`), the city where the worker lives (`city`), and the worker's home address (`homeaddress`). We know for certain that two subsets of these variables will uniquely identify every worker in the database: `workplace`–`name` and `city`–`homeaddress`. Using these two subsets, we generate the variables `id1` and `id2`, which uniquely identify workers in terms of each subset of variables.[2] Table 2 shows an extract of this database.

2. Variable `id1`, for example, may be generated by typing `egen id1 = group(workplace name)`.

Table 2. Generating identification from several sources

n	workplace	name	city	homeaddress	id1	id2	group()	group2
1	Arthur Guinness & Son	William Gosset	Dublin	100 A Street	1	1	1	1
2	Arthur Guinness & Son	William Gosset	Bublin	100 A Street	1	2	2	1
3	Federal Office for Intellectual Property	Albert Einstein	Bern	200 B Street	2	3	3	2
4	Federal Office for Intellectual Property	Albert Einstein	Bern	200 B Street	2	3	3	2
5	Federal Office for Intellectual Property	Albert Einstein	-	200 B Street	2	4	4	2
6	Guinness brewery	William Sealy Gosset	Dublin	100 A Street	3	1	5	1
7	Guinness brewery	William Sealy Gosset	Dublin	-	3	5	6	1

From table 2, it is clear that observations 1, 2, 6, and 7 are the same person, and observations 3, 4, and 5 are another person. The variable `id1` successfully identifies the second individual, yet it fails to fully identify the first one, instead indicating that observations 1 and 2 are one person and 6 and 7 are another because both the `workplace`'s name and the worker's `name` are spelled differently among those two subsets of observations. On the other hand, because `city` and `homeaddress` contain both misspellings and missing values, variable `id2` is only able to identify that observations 1 and 6 are the same worker and that observations 3 and 4 are the same worker.

The `group2` command combines the information provided from both `id1` and `id2` to generate a complete identification. First, because `id1` shows that observations 1 and 2 are the same worker and `id2` shows that observations 1 and 6 are the same worker, then `group2` knows that observations 1, 2, and 6 are the same worker. Yet `id1` also shows that observations 6 and 7 are the same worker; therefore, observations 1, 2, 6, and 7 must be the same worker. Second, `id1` shows that 3, 4, and 5 are the same worker, and even though `id2` shows that observation 5 is a different worker, `group2` ignores this because any of the matches are sufficient for identification.

The `group()` function, when using `id1` and `id2` as arguments, is also shown for comparison.

4.2 Example 2: Network identification

Let us illustrate the use of the `group2` command by looking at a different kind of problem: identification of network overlaps. Social networks are a major influence in today's economic activity, and promising research is taking place on this topic. A common challenge when managing these datasets is the identification of complete networks, where declarations of individual networks must be combined to calculate the size of the overall network.

Imagine a database that contains information on individuals, the places that they have visited in the last two days, and a variable indicating whether, before these two days, they were infected with a highly contagious virus. We are interested in knowing which places might have been left infected by the individuals carrying the virus and, therefore, which individuals were at risk of having been infected by visiting these places.

In this example, we understand a network to be different individuals connected through commonly visited places. Columns 1 to 3 of table 3 contain an extract of this database. These columns tell us that there are six individuals to be considered, that each individual visited three places in the last two days, and that only individual 1 was originally infected. From this small extract, we can identify the network manually: one of the places individual 1 visited, place C, was also visited by individual 2, so he or she is possibly infected; furthermore, individual 4 visited a place that individual 2 also visited, place D, so individual 4 may also be infected. In addition, we can see that individuals 3 and 6 both visited place H and that individual 5 visited three places no other individual visited.

In the presence of a larger dataset, the `group2` command will carry out the identification automatically: the command will group all observations that either share the same `individual_id` or share the same `place_id`. The outcome variable of this identification has been placed in the fourth column of table 3.

Table 3. Network identification

`individual_id`	`place_id`	`virus`	`group2`
1	A	1	1
1	B	1	1
1	C	1	1
2	C	0	1
2	D	0	1
2	E	0	1
3	F	0	3
3	G	0	3
3	H	0	3
4	D	0	1
4	I	0	1
4	J	0	1
5	K	0	2
5	L	0	2
5	M	0	2
6	H	0	3
6	N	0	3
6	G	0	3

5 Discussion

In this article, I presented an alternative interpretation of the `group()` function based on set theory, an interpretation that in turn engenders a related command, here called `group2`. While the `group()` function creates a variable that generates the coarsest partition that is finer than the finest partition generated by the variables used as arguments, the `group2` command will create a variable that generates the finest partition that is coarser than the coarsest partition generated by the variables used as arguments. The operation performed by this new command has been very useful in a number of database management problems, and sharing it became a natural step.

A future contribution will be to allow for this command to use more than two variables in *varlist*; in the meantime, the command has to be applied $n-1$ times for n variables. An additional contribution will be to add an option that allows the user to choose whether missing values are to be treated as different, unique values (the only current alternative) or as one particular value common to all missing values when partitioning the set of observations. To the extent that they are needed, other more sophisticated set operations might be translated to functions for database management.

About the author

Christian Salas Pauliac is studying for a PhD in Public Policy at the University of Chicago, Chicago, IL.