Title

stata.com

sortpreserve — Sort within programs

Syntax Description Option Remarks and examples Also see

Syntax

```
program [define] program_name [, ... sortpreserve ...]
```

Description

This entry discusses the use of `sort` (see [D] **sort**) within programs.

Option

`sortpreserve` specifies that the program, during its execution, will re-sort the data and that therefore Stata itself should take action to preserve the order of the data so that the order can be reestablished afterward.

`sortpreserve` is in fact independent of whether a program is `byable()` but `byable()` programs often specify this option.

Pretend you are writing the program `myprog` and that, in performing its calculations, it needs to sort the data. It is very jolting for a user to experience,

```
. by pid: myprog ...
. by pid: sum newvar
not sorted
r(5);
```

Specifying `sortpreserve` will prevent this and still allow `myprog` to sort the data freely. `byable()` programs that sort the data should specify `sortpreserve`. It is not necessary to specify `sortpreserve` if your program does not change the sort order of the data and, in that case, things are a little better if you do not specify `sortpreserve`.

`sortpreserve` takes time, although less than you might suspect. `sortpreserve` does not actually have to re-sort the data at the conclusion of your program—an O(n ln n) operation—it is able to arrange things so that it can reassert the original order of the data in O(n) time, and `sortpreserve` is, in fact, very quick about it. Nonetheless, there is no reason to waste the time if the data never got out of order.

Concerning sort order, when your `byable()` program is invoked for the first time, it will be sorted on `_byvars` but, in subsequent calls (in the case of `byable(recall)` programs), the sort order will be just as your program leaves it even if you specify `sortpreserve`. `sortpreserve` restores the original order after your program has been called for the last time.

Remarks and examples

stata.com

Remarks are presented under the following headings:

Introduction

Properly written programs do one of three things:

1. Report results
2. Add new variables to the dataset
3. Modify the data in memory

However, you do not want to get carried away with the idea. A properly written program might, for instance, report results and yet still have an option to add a new variable to the dataset, but a properly written program would not do all three. The user should be able to obtain reports over and over again by simply retyping the command, and if a command both reports results and modifies the data, that will not be possible.

Properly written programs of the first two types should also not change the sort order of the data. If the data are sorted on `mpg` and `foreign` before the command is given, and all the command does is report results, the data should still be sorted on `mpg` and `foreign` at the conclusion of the command. Yet the command might find it necessary to `sort` the data to obtain the results it calculates.

This entry deals with how to easily satisfy both needs.

sortpreserve

You may include `sort` commands inside your programs and leave the user's data in the original order when your program concludes by specifying the `sortpreserve` option on the `program` definition line:

```
program whatever, sortpreserve
        ...
end
```

That is all there is to it. `sortpreserve` tells Stata when it starts your program to first record the information about how the data are currently sorted and then later use that information to restore the order to what it previously was. Stata will do this no matter how your program ends, whether as you expected, with an error, or because the user pressed the *Break* key.

The cost of sortpreserve

There is a cost to `sortpreserve`, so you do not want to specify the option when it is not needed, but the cost is not much. `sortpreserve` will consume a little computer time in restoring the sort order at the conclusion of your program. Rather than talking about this time in seconds or milliseconds, which can vary according to the computer you use, let's define our unit of time as the time to execute:

```
. generate long x = _n
```

Pretend that you added that command to your program, just as we have typed it, without using temporary variables. You could then make careful timings of your program to find out just how much extra time your program would take to execute. It would not be much. Let's call that amount of time one *genlong* unit. Then

- `sortpreserve`, if it has to restore the order because your program has changed it, takes 2 *genlong* units.
- `sortpreserve`, if it does not need to change the order because your program has not changed it yet, takes one-half a *genlong* unit.

The above results are based on empirical timings using 100,000 and 1,000,000 observations.

How sortpreserve works

`sortpreserve` works by adding a temporary variable to the dataset before your program starts, and if you are curious about the name of that variable, it is recorded in the macro `‘_sortindex’`. Sometimes you will want to know that name. It is important that the variable `‘_sortindex’` still exist at the conclusion of your program. If your program concludes with something like

```
keep ‘id’ ‘varlist’
```

you must change that line to read

```
keep ‘id’ ‘varlist’ ‘_sortindex’
```

If you fail to do that, Stata will report the error message "could not restore sort order because variables were dropped". Actually, even that little change may be insufficient because the dataset in its original form might have been sorted on something other than `‘id’` and `‘varlist’`. What you really need to do is add, early in your program and before you change the sort order,

```
local sortvars : sort
```

and then change the `keep` statement to read

```
keep ‘id’ ‘varlist’ ‘sortvars’ ‘_sortindex’
```

This discussion concerns only the use of the `keep` command. Few programs would even include a `keep` statement because we are skirting the edge of what is a properly written program.

`sortpreserve` is intended for use in programs that report results or add new variables to the dataset, not programs that modify the data in memory. Including `keep` at the end of your program really makes it a class 3 program, and then the idea of preserving the sort order makes no sense anyway.

Use of sortpreserve with preserve

`sortpreserve` may be used with `preserve` (see [P] **preserve** for a description of `preserve`). We can imagine a complicated program that re-sorts the data, and then, under certain conditions, discovers it has to do real damage to the data to calculate its results, and so then `preserve`s the data to boot:

```
program ..., sortpreserve
        ...
        sort ...
        ...
        if ... {
                preserve
                ...
        }
        ...
end
```

The above program will work. When the program ends, Stata will first restore any `preserve`d data and then reestablish the sort of the original dataset.

Use of sortpreserve with subroutines that use sortpreserve

Programs that use `sortpreserve` may call other programs that use `sortpreserve`, and this can be a good way to speed up code. Consider a calculation where you need the data first sorted by `‘i’ ‘j’`, then by `‘j’ ‘i’`, and finally by `‘i’ ‘j’` again. You might code

```
program ..., sortpreserve
        ...
        sort ‘i’ ‘j’
        ...
        sort ‘j’ ‘i’
        ...
        sort ‘i’ ‘j’
        ...
end
```

but executing

```
program ..., sortpreserve
        ...
        sort ‘i’ ‘j’
        mysubcalculation ‘i’ ‘j’ ...
        ...
end
program mysubcalculation, sortpreserve
        args i j ...
        sort ‘j’ ‘i’
        ...
end
```

will be faster.

Also see

[P] **byable** — Make programs byable

[P] **program** — Define and manipulate programs