Title

stata.com

gsort — Ascending and descending sort

Description

`gsort` arranges observations to be in ascending or descending order of the specified variables and so differs from `sort` in that `sort` produces ascending-order arrangements only; see [D] **sort**.

Each *varname* can be numeric or a string.

The observations are placed in ascending order of *varname* if `+` or nothing is typed in front of the name and are placed in descending order if `-` is typed.

Quick start

Sort dataset in memory by ascending values of `v1`, equivalent to `sort`

```
gsort v1
```

Sort dataset in memory by descending values of `v1`

```
gsort -v1
```

Sort dataset by ascending values of `v1` and descending values of `v2`

```
gsort v1 -v2
```

Create `newv` for use in subsequent `by` operations

```
gsort v1 -v2, generate(newv)
```

Place missing values of descending-order `v2` at the top of the dataset instead of the end

```
gsort v1 -v2, mfirst
```

Menu

Data > Sort

Syntax

```
gsort [+|-] varname [ [+|-] varname ...] [, generate(newvar) mfirst]
```

Options

`generate(`*newvar*`)` creates *newvar* containing 1, 2, 3, ... for each group denoted by the ordered data. This is useful when using the ordering in a subsequent `by` operation; see **[U] 11.5 by varlist: construct** and examples below.

`mfirst` specifies that missing values be placed first in descending orderings rather than last.

Remarks and examples

stata.com

`gsort` is almost a plug-compatible replacement for `sort`, except that you cannot specify a general *varlist* with `gsort`. For instance, `sort alpha-gamma` means to sort the data in ascending order of `alpha`, within equal values of `alpha`; sort on the next variable in the dataset (presumably `beta`), within equal values of `alpha` and `beta`; etc. `gsort alpha-gamma` would be interpreted as `gsort alpha -gamma`, meaning to sort the data in ascending order of `alpha` and, within equal values of `alpha`, in descending order of `gamma`.

▷ Example 1

The difference in *varlist* interpretation aside, `gsort` can be used in place of `sort`. To list the 10 lowest-priced cars in the data, we might type

```
. use http://www.stata-press.com/data/r15/auto
. gsort price
. list make price in 1/10
```

or, if we prefer,

```
. gsort +price
. list make price in 1/10
```

To list the 10 highest-priced cars in the data, we could type

```
. gsort -price
. list make price in 1/10
```

`gsort` can also be used with string variables. To list all the makes in reverse alphabetical order, we might type

```
. gsort -make
. list make
```

◁

▷ Example 2

`gsort` can be used with multiple variables. Given a dataset on hospital patients with multiple observations per patient, typing

```
. use http://www.stata-press.com/data/r15/bp3
. gsort id time
. list id time bp
```

lists each patient's blood pressures in the order the measurements were taken. If we typed

```
. gsort id -time
. list id time bp
```

then each patient's blood pressures would be listed in reverse time order.

◁

❑ Technical note

Say that we wished to attach to each patient's records the lowest and highest blood pressures observed during the hospital stay. The easier way to achieve this result is with `egen`'s `min()` and `max()` functions:

```
. egen lo_bp = min(bp), by(id)
. egen hi_bp = max(bp), by(id)
```

See [D] **egen**. Here is how we could do it with `gsort`:

```
. use http://www.stata-press.com/data/r15/bp3, clear
. gsort id bp
. by id: generate lo_bp = bp[1]
. gsort id -bp
. by id: generate hi_bp = bp[1]
. list, sepby(id)
```

This works, even in the presence of missing values of `bp`, because such missing values are placed last within arrangements, regardless of the direction of the sort.

❑

❑ Technical note

Assume that we have a dataset containing `x` for which we wish to obtain the forward and reverse cumulatives. The forward cumulative is defined as $F(X) =$ the fraction of observations such that `x` $\leq X$. Again let's ignore the easier way to obtain the forward cumulative, which would be to use Stata's `cumul` command,

```
. set obs 100
. generate x = rnormal()
. cumul x, gen(cum)
```

(see [R] **cumul**). Eschewing `cumul`, we could type

```
. sort x
. by x: generate cum = _N if _n==1
. replace cum = sum(cum)
. replace cum = cum/cum[_N]
```

That is, we first place the data in ascending order of `x`; we used `sort` but could have used `gsort`. Next, for each observed value of `x`, we generated `cum` containing the number of observations that take on that value (you can think of this as the discrete density). We summed the density, obtaining the distribution, and finally normalized it to sum to 1.

The reverse cumulative $G(X)$ is defined as the fraction of data such that `x` $\geq X$. To obtain this, we could try simply reversing the sort:

```
. gsort -x
. by x: generate rcum = _N if _n==1
. replace rcum = sum(rcum)
. replace rcum = rcum/rcum[_N]
```

This would work, except for one detail: Stata will complain that the data are not sorted in the second line. Stata complains because it does not understand descending sorts (`gsort` is an ado-file). To remedy this problem, `gsort`'s `generate()` option will create a new grouping variable that is in ascending order (thus satisfying Stata's narrow definition) and that is, in terms of the groups it defines, identical to that of the true sort variables:

```
. gsort -x, gen(revx)
. by revx: generate rcum = _N if _n==1
. replace rcum = sum(rcum)
. replace rcum = rcum/rcum[_N]
```

❑

Also see

[D] **sort** — Sort data