Description	Quick start	Menu	Syntax
Option	Remarks and examples	References	Also see

# Description

sort arranges the observations of the current data into ascending order based on the values of the variables in *varlist*. There is no limit to the number of variables in *varlist*. Missing numeric values are interpreted as being larger than any other number, so they are placed last with  $. < .a < .b < \cdots < .z$ . When you sort on a string variable, however, null strings are placed first and uppercase letters come before lowercase letters.

The dataset is marked as being sorted by *varlist* unless in *range* is specified. If in *range* is specified, only those observations are rearranged. The unspecified observations remain in the same place.

# **Quick start**

Sort dataset in memory by ascending values of v1

sort v1

Same as above, and order within v1 by ascending values of v2 and within v2 by v3

sort v1 v2 v3

Same as above, and keep observations with the same values of v1, v2, and v3 in the same presort order sort v1 v2 v3, stable

# Menu

Data > Sort

# Syntax

```
sort varlist [in] [, stable]
```

# Option

stable specifies that observations with the same values of the variables in *varlist* keep the same relative order in the sorted data that they had previously. For instance, consider the following data:

Typing sort x without the stable option produces one of the following six orderings:

x b	xb	x b	xb	x b	x b
12	12	11	11	13	13
11	13	13	12	11	12
13	11	12	13	12	11
24	24	24	24	24	24
31	31	31	31	31	31

Without the stable option, the ordering of observations with equal values of *varlist* is randomized. With sort x, stable, you will always get the first ordering and never the other five.

If your intent is to have the observations sorted first on x and then on b within tied values of x (the fourth ordering above), you should type sort x b rather than sort x, stable.

stable is seldom used and, when specified, causes sort to execute more slowly.

# **Remarks and examples**

Sorting data is one of the more common tasks involved in processing data. Often, before Stata can perform some task, the data must be in a specific order. For the merge command to create a new dataset that matches records from two datasets on a common key, both of those datasets must be sorted by that key. Either you will sort the data or merge will sort it for you. If you want to use the by *varlist*: prefix, the data must be sorted in order of *varlist*. You even sort data to put it into a more convenient order when using list.

Remarks are presented under the following headings:

Finding the smallest values (and the largest) Tracking sort order Sorting on multiple variables Descending sorts Sorting on string variables Sorting with ties

### Finding the smallest values (and the largest)

Sorting data can be informative. Suppose that we have data on automobiles, and each car's make and mileage rating (called make and mpg) are included among the variables in the data. We want to list the five cars with the lowest mileage rating in our data:

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)
. keep make mpg weight
. sort mpg, stable
. list make mpg in 1/5
       make
                            mpg
       Linc. Continental
 1.
                             12
 2.
       Linc. Mark V
                             12
 3.
       Cad. Deville
                             14
 4.
       Cad. Eldorado
                             14
 5.
       Linc. Versailles
                             14
```

We can also list the five cars with the highest mileage.

. list in -5/1

	make	mpg	weight
70.	Toyota Corolla	31	2,200
71.	Plym. Champ	34	1,800
72.	Datsun 210	35	2,020
73.	Subaru	35	2,050
74.	VW Diesel	41	2,040
	1		

## **Tracking sort order**

Stata keeps track of the order of your data. For instance, we just sorted the above data on mpg. When we ask Stata to describe the data in memory, it tells us how the dataset is sorted:

. describe								
Contains data from https://www.stata-press.com/data/r19/auto.dta								
Observations: Variables:		74 3		1978 automobile data				
				13 Apr 2024 17:45				
				(_dta has notes)				
Variable	Storage	Display	Value					
name	type	format	label	Variable label				
make	str18	%-18s		Make and model				
mpg	int	%8.0g		Mileage (mpg)				
weight	int	%8.0gc		Weight (lbs.)				
make mpg weight	str18 int int	%-18s %8.0g %8.0gc		Make and model Mileage (mpg) Weight (lbs.)				

Sorted by: mpg

Note: Dataset has changed since last saved.

Stata keeps track of changes in sort order. If we were to make a change to the mpg variable, Stata would know that the data are no longer sorted. Remember that the first observation in our data has mpg equal to 12, as does the second. Let's change the value of the first observation:

```
. replace mpg=13 in 1
(1 real change made)
. describe
Contains data from https://www.stata-press.com/data/r19/auto.dta
 Observations:
                           74
                                                 1978 automobile data
    Variables:
                            3
                                                 13 Apr 2024 17:45
                                                 (dta has notes)
Variable
                                     Value
               Storage
                         Display
                          format
                                     label
                                                 Variable label
    name
                  type
make
                 str18
                         %-18s
                                                 Make and model
mpg
                 int
                         %8.0g
                                                 Mileage (mpg)
weight
                 int
                         %8.0gc
                                                 Weight (lbs.)
```

Sorted by:

Note: Dataset has changed since last saved.

After making the change, Stata indicates that our dataset is "Sorted by:" nothing. Let's put the dataset back as it was:

```
. replace mpg=12 in 1
(1 real change made)
. sort mpg
```

#### Technical note

Stata is limited in how it tracks changes in the sort order and will sometimes decide that a dataset is not sorted when, in fact, it is. For instance, if we were to change the first observation of our automobile dataset from 12 miles per gallon to 10, Stata would decide that the dataset is "Sorted by:" nothing, just as it did above when we changed mpg from 12 to 13. Our change in example 2 did change the order of the data, so Stata was correct. Changing mpg from 12 to 10, however, does not really affect the sort order.

As far as Stata is concerned, any change to the variables on which the data are sorted means that the data are no longer sorted, even if the change actually leaves the order unchanged. Stata may be dumb, but it is also fast. It sorts already-sorted datasets instantly, so Stata's ignorance costs us little.

#### Sorting on multiple variables

Data can be sorted by more than one variable, and in such cases, the sort order is lexicographic. If we sort the data by two variables, for instance, the data are placed in ascending order of the first variable, and then observations that share the same value of the first variable are placed in ascending order of the second variable. Let's order our automobile data by mpg and within mpg by weight:

```
. sort mpg weight
```

```
. list in 1/8, sep(4)
```

	make	mpg	weight
1.	Linc. Mark V	12	4,720
2.	Linc. Continental	12	4,840
3.	Peugeot 604	14	3,420
4.	Linc. Versailles	14	3,830
5.	Cad. Eldorado	14	3,900
6.	Merc. Cougar	14	4,060
7.	Merc. XR-7	14	4,130
8.	Cad. Deville	14	4,330

The data are in ascending order of mpg, and within each mpg category, the data are in ascending order of weight. The lightest car that achieves 14 miles per gallon in our data is the Peugeot 604.

#### Technical note

The sorting technique used by Stata is fast, but the order of variables not included in *varlist* is not maintained. If you wish to maintain the order of additional variables, include them at the end of *varlist*. There is no limit to the number of variables by which you may sort.

#### **Descending sorts**

Sometimes, you may want to order a dataset by descending sequence of something. Perhaps we wish to obtain a list of the five cars achieving the best mileage rating. The sort command orders the data only into ascending sequences. Another command, gsort, orders the data in ascending or descending sequences; see [D] gsort. You can also create the negative of a variable and achieve the desired result:

```
. generate negmpg = -mpg
```

```
. sort negmpg
```

```
. list in 1/5
```

	make	mpg	weight	negmpg
1.	VW Diesel	41	2,040	-41
2.	Datsun 210	35	2,020	-35
З.	Subaru	35	2,050	-35
4.	Plym. Champ	34	1,800	-34
5.	Toyota Corolla	31	2,200	-31

We find that the VW Diesel tops our list.

#### Sorting on string variables

sort may also be used on string variables. The data are sorted alphabetically:

- . sort make
- . list in 1/5

make	mpg	weight	negmpg
AMC Concord	22	2,930	-22
AMC Pacer	17	3,350	-17
AMC Spirit	22	2,640	-22
Audi 5000	17	2,830	-17
Audi Fox	23	2,070	-23
	make AMC Concord AMC Pacer AMC Spirit Audi 5000 Audi Fox	makempgAMC Concord22AMC Pacer17AMC Spirit22Audi 500017Audi Fox23	make         mpg         weight           AMC Concord         22         2,930           AMC Pacer         17         3,350           AMC Spirit         22         2,640           Audi 5000         17         2,830           Audi Fox         23         2,070

#### Technical note

Bear in mind that Stata takes "alphabetically" to mean "in order by byte value". This means that all uppercase letters come before lowercase letters; for example, Z < a. As far as Stata is concerned, the following list is sorted alphabetically:

list, sep(0)				
	myvar			
1. 2. 3. 4. 5. 6.	ALPHA Alpha BETA Beta alpha beta			

For most purposes, this method of sorting is sufficient. It is possible to override Stata's sort logic. See [U] **12.4.2.5 Sorting strings containing Unicode characters** for information about ordering strings in a language-sensitive way. We do not recommend that you do this.

#### Sorting with ties

Sorting when your list of sort variables does not uniquely identify an observation, that is to say when you have ties, is usually dangerous and should be avoided. Consider using sort to find the average mpg for the five cars with the smallest gear ratio.

```
. use https://www.stata-press.com/data/r19/auto
(1978 automobile data)
. sort gear_ratio
. summarize mpg in 1/5
    Variable
                       Obs
                                   Mean
                                           Std. dev.
                                                             Min
                                                                        Max
                         5
                                     17
                                           3.674235
                                                             14
                                                                         21
         mpg
```

So the answer is 17.

We go on and do some other work.

We forgot to write down the answer from earlier, and silly us, we were not logging our results. So we run our commands again:

	mpg	5	15.8	2.949576	14	21
	Variable	Obs	Mean	Std. dev.	Min	Max
. :	summarize mp	pg in 1/5				
. :	sort gear_ra	atio				
. 1 (1	use https:// 978 automobi	/www.stata-press ile data)	.com/data/	r19/auto		

So the answer is 15.8.

What happened? The title of this section is a clue. Let's list some of the data:

```
. list gear_ratio mpg in 1/10
```

	gear_r~o	mpg
1. 2. 3. 4.	2.19 2.24 2.26 2.28 2.41	14 21 15 14 15
6. 7. 8	2.41 2.41 2.41 2.43	16 21 18
9. 10.	2.43 2.47 2.47	16 14

The first four observations look fine; each value of gear\_ratio is unique. But the fifth, sixth, and seventh observations all have a gear\_ratio of 2.41, whereas the values of mpg differ. Do we want mpg = 15, mpg = 16, or mpg = 21 in our mean?

There are not many things we can do after a sort that will produce unique results if the sort itself has observations with ties in *varlist*. The ordering is not unique. You must be sure that the ordering really does not matter. If that is the case, then why did you sort in the first place?

So what are we to do? We could rephrase our question as "What is the lowest possible average mpg for five cars with the smallest gear\_ratio?" Then we would type

	mpg	5	15.8	2.949576	14	21
	Variable	Obs	Mean	Std. dev.	Min	Max
•	summarize mpg	g in 1/5				
•	sort gear_rat	tio mpg				

Now we will always get the same answer—15.8.

How do you know your sort variables form a unique ordering? Ask

```
. isid gear_ratio mpg
variables gear_ratio and mpg do not uniquely identify the observations
r(459);
```

That is still not a unique ordering. Our analysis does not require a fully unique ordering. Because we are summarizing mpg, tied values of mpg will give the same answer. Even so, there would be no harm in adding another variable to make the ordering truly unique:

	isid gear_ra	atio mpg weight				
	sort gear_ra	atio mpg weight				
	summarize m	pg in 1/5				
	Variable	Obs	Mean	Std. dev.	Min	Max
_	mpg	5	15.8	2.949576	14	21

What if we did not want the lowest mpg? What if we preferred a randomized answer where the computer chooses one of the observations with tied gear\_ratio? The best approach is to use a good random-number generator to create a new variable with random values that you will also sort on:

	mpg	5	17	3.674235	14	21
	Variable	Obs	Mean	Std. dev.	Min	Max
•	. summarize mpg in 1/5					
•	sort gear_ratio rnd					
•	<pre>generate rnd = runiform()</pre>					
•	set seed 123	345				

Why did we set the seed? So that our randomized result is reproducible. That is not a contradiction.

A benefit of this approach is that regardless of any further transformations or manipulations we make on this dataset we can always recover the ordering by typing

. sort gear\_ratio rnd

Well, we cannot change the values of gear\_ratio or rnd, and we cannot add or insert observations, but any other manipulations are allowed.

Our example is rather artificial, but there are many cases where you do want a randomized order within tied values of a sorted variable. One such case is creating simulated datasets for panel data or multilevel data.

It turns out that our first results were also randomly ordered. That is true because sort performs a quick randomized jumbling before sorting. We were already getting a randomized order within the ties. Do not use this in practice. The randomization performed by sort is designed solely to make sort faster by preventing any possibility of an initial ordering that defeats the sort algorithm and makes the sorting much slower. If you want a random ordering within ties, then use a random-number generator with good properties like the one implemented in runiform(). For more about the random-number generator, see [R] set seed and the references therein.

If you do not want a random ordering within ties and you also do not want to use other variables from the dataset to define a unique ordering, you can add a sequence variable to the dataset and include it in your sort,

. generate id = \_n . sort gear\_ratio id

That sort will still depend on the order of your data when id is created, but you will always be able to recreate the ordering by typing

. sort gear\_ratio id

The ordering produced after this sort will be identical to the ordering had we instead typed

. sort gear\_ratio, stable

The advantage to creating the id variable is that we can recover this ordering at any time in the future by retyping

. sort gear\_ratio id

That cannot be said of

. sort gear\_ratio, stable

The ordering after this sort will depend on the order before the sort command. So if we sort on another variable between our two stable sorts, the ordering after those two stable sorts will be different.

One final note. If you ran the commands in this entry, you may have obtained different results from those printed here for the first several summarize commands and a different ordering from the first list command. That is yet another reminder not to perform order-dependent analyses when your current sort order is not unique. You got different results because the jumbler that sort preapplies started from a different point than it did when we ran the commands for this manual entry. Unless you start Stata immediately before running a sort with tied values or you set the state of the jumbler, you will rarely get the same ordering for tied keys. If you want to get the ordering we got in this entry, you should use Stata/SE and type

. set sortrngstate 12345

That's what we do so that this entry does not change every time we re-create the manuals. See [P] set sortrngstate. This is such an esoteric command that we warn you against using it. Regardless, unless your goal is to write a manual entry that describes how to deal with tied values in sorts, do not use set sortrngstate to create reproducible sorts. Think about your problem and sort on variables that create the unique ordering you need. Or decide you want a stable sort of the ties based on the current ordering. Or use the method described above that creates a good random number to randomly order the tied values.

# References

Royston, P. 2001. Sort a list of items. *Stata Journal* 1: 105–106. Schumm, L. P. 2006. Stata tip 28: Precise control of dataset sort order. *Stata Journal* 6: 144–146.

## Also see

- [D] describe Describe data in memory or in a file
- [D] gsort Ascending and descending sort
- [U] 11 Language syntax

Stata, Stata Press, Mata, NetCourse, and NetCourseNow are registered trademarks of StataCorp LLC. Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations. StataNow is a trademark of StataCorp LLC. Other brand and product names are registered trademarks or trademarks of their respective companies. Copyright © 1985–2025 StataCorp LLC, College Station, TX, USA. All rights reserved.



For suggested citations, see the FAQ on citing Stata documentation.