Title

stata.com

generate — Create or change contents of variable

Description

`generate` creates a new variable. The values of the variable are specified by =*exp*.

If no *type* is specified, the new variable type is determined by the type of result returned by =*exp*. A `float` variable (or a `double`, according to `set type`) is created if the result is numeric, and a string variable is created if the result is a string. In the latter case, if the string variable contains values greater than 2,045 characters or contains values with a binary 0 (`\0`), a `strL` variable is created. Otherwise, a `str#` variable is created, where # is the smallest string that will hold the result.

If a *type* is specified, the result returned by =*exp* must be a string or numeric according to whether *type* is string or numeric. If `str` is specified, a `strL` or a `str#` variable is created using the same rules as above.

See [D] **egen** for extensions to `generate`.

`replace` changes the contents of an existing variable. Because `replace` alters data, the command cannot be abbreviated.

`set type` specifies the default storage type assigned to new variables (such as those created by `generate`) when the storage type is not explicitly specified.

Quick start

Create numeric variable `newv1` equal to `v1` + 2

```
generate newv1 = v1 + 2
```

As above, but use type `byte` and label the values of `newv1` with value label `mylabel`

```
generate byte newv1:mylabel = v1 + 2
```

String variable `newv2` equal to "my text"

```
generate newv2 = "my text"
```

Variable `newv3` equal to the observation number

```
generate newv3 = _n
```

Replace `newv3` with observation number within each value of `catvar`

```
by catvar: replace newv3 = _n
```

Binary indicator for first observation within each value of `catvar` after sorting on `v2`

```
bysort catvar (v2): generate byte first = _n==1
```

As above, but for last observation

```
bysort catvar (v2): generate byte last = _n==_N
```

Combined datetime variable `newv4` from `%td` formatted `date` and `%tc` formatted `time`

```
generate double newv4 = cofd(date) + time
```

Menu

generate

Data > Create or change data > Create new variable

replace

Data > Create or change data > Change contents of variable

Syntax

Create new variable

```
generate [type] newvar[:lblname] =exp [if] [in]
  [, before(varname)|after(varname)]
```

Replace contents of existing variable

```
replace oldvar =exp [if] [in] [, nopromote]
```

Specify default storage type assigned to new variables

```
set type {float|double} [, permanently]
```

where *type* is one of `byte` | `int` | `long` | `float` | `double` | `str` | `str1` | `str2` | ... | `str2045`.

See *Description* below for an explanation of `str`. For the other types, see **[U] 12 Data**.

`by` is allowed with `generate` and `replace`; see [D] **by**.

Options

`before(`*varname*`)` or `after(`*varname*`)` may be used with `generate` to place the newly generated variable in a specific position within the dataset. These options are primarily used by the Data Editor and are of limited use in other contexts. A more popular alternative for most users is `order` (see [D] **order**).

`nopromote` prevents `replace` from promoting the variable type to accommodate the change. For instance, consider a variable stored as an integer type (`byte`, `int`, or `long`), and assume that you `replace` some values with nonintegers. By default, `replace` changes the variable type to a floating point (`float` or `double`) and thus correctly stores the changed values. Similarly, `replace` promotes `byte` and `int` variables to longer integers (`int` and `long`) if the replacement value is an integer but is too large in absolute value for the current storage type. `replace` promotes strings to longer strings. `nopromote` prevents `replace` from doing this; instead, the replacement values are truncated to fit into the current storage type.

`permanently` specifies that, in addition to making the change right now, the new limit be remembered and become the default setting when you invoke Stata.

Remarks and examples

stata.com

Remarks are presented under the following headings:

generate and replace

`generate` and `replace` are used to create new variables and to modify the contents of existing variables, respectively. Although the commands do the same thing, they have different names so that you do not accidentally replace values in your data. Detailed descriptions of expressions are given in **[U] 13 Functions and expressions**.

Also see [D] **edit**.

▷ Example 1

We have a dataset containing the variable `age2`, which we have previously defined as `age^2` (that is, age^2). We have changed some of the `age` data and now want to correct `age2` to reflect the new values:

```
. use http://www.stata-press.com/data/r15/genxmpl1
(Wages of women)
. generate age2=age^2
age2 already defined
r(110);
```

When we attempt to re-`generate` `age2`, Stata refuses, telling us that `age2` is already defined. We could `drop` `age2` and then re-`generate` it, or we could use the `replace` command:

```
. replace age2=age^2
(204 real changes made)
```

When we use `replace`, we are informed of the number of actual changes made to the dataset. ◁

You can explicitly specify the storage type of the new variable being created by putting the *type*, such as `byte`, `int`, `long`, `float`, `double`, or `str8`, in front of the variable name. For example, you could type `generate double revenue = qty * price`. Not specifying a type is equivalent to specifying `float` if the variable is numeric, or, more correctly, it is equivalent to specifying the default type set by the `set type` command; see below. If the variable is alphanumeric, not specifying a type is equivalent to specifying `str#`, where # is the length of the largest string in the variable.

You may also specify a value label to be associated with the new variable by including "`:`*lblname*" after the variable name. This is seldom done because you can always associate the value label later by using the `label values` command; see **[U] 12.6.3 Value labels**.

▷ Example 2

Among the variables in our dataset is `name`, which contains the first and last name of each person. We wish to create a new variable called `lastname`, which we will then use to sort the data. `name` is a string variable.

```
. use http://www.stata-press.com/data/r15/genxmpl2, clear
. list name

     +-----------------+
     |            name |
     |-----------------|
  1. |    Johanna Roman |
  2. |     Dawn Mikulin |
  3. |     Malinda Vela |
  4. |       Kevin Crow |
  5. | Zachary Bimslager |
     +-----------------+

. generate lastname=word(name,2)
. describe
Contains data from http://www.stata-press.com/data/r15/genxmpl2.dta
  obs:             5
 vars:             2                          18 Jan 2016 12:24
 size:           130
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
name            str17   %17s
lastname        str9    %9s
-------------------------------------------------------------------------------
Sorted by:
     Note: Dataset has changed since last saved.
```

Stata is smart. Even though we did not specify the storage type in our `generate` statement, Stata knew to create a `str9 lastname` variable, because the longest last name is `Bimslager`, which has nine characters.

◁

▷ Example 3

We wish to create a new variable, `age2`, that represents the variable `age` squared. We realize that because `age` is an integer, `age2` will also be an integer and will certainly be less than 32,740. We therefore decide to store `age2` as an `int` to conserve memory:

```
. use http://www.stata-press.com/data/r15/genxmpl3, clear
. generate int age2=age^2
(9 missing values generated)
```

Preceding `age2` with `int` told Stata that the variable was to be stored as an `int`. After creating the new variable, Stata informed us that nine missing values were generated. `generate` informs us whenever it produces missing values.

◁

See **[U] 13 Functions and expressions** and **[U] 25 Working with categorical data and factor variables** for more information and examples. Also see **[D] recode** for a convenient way to recode categorical variables.

❑ Technical note

If you specify the `if` or `in` qualifier, the `=`*exp* is evaluated only for those observations that meet the specified condition or are in the specified range (or both, if both `if` and `in` are specified). The other observations of the new variable are set to missing:

```
. use http://www.stata-press.com/data/r15/genxmpl3, clear
. generate int age2=age^2 if age>30
(290 missing values generated)
```

❑

▷ Example 4

`replace` can be used to change just one value, as well as to make sweeping changes to our data. For instance, say that we enter data on the first five odd and even positive integers and then discover that we made a mistake:

```
. use http://www.stata-press.com/data/r15/genxmpl4, clear
. list
```

	odd	even
1.	1	2
2.	3	4
3.	-8	6
4.	7	8
5.	9	10

The third observation is wrong; the value of `odd` should be 5, not −8. We can use `replace` to correct the mistake:

```
. replace odd=5 in 3
(1 real change made)
```

We could also have corrected the mistake by typing `replace odd=5 if odd==-8`.

◁

set type

When you create a new numeric variable and do not specify the storage type for it, say, by typing `generate y=x+2`, the new variable is made a `float` if you have not previously issued the `set type` command. If earlier in your session you typed `set type double`, the new numeric variable would be made a `double`.

Video examples

How to create a new variable that is calculated from other variables

How to identify and replace unusual data values

Methods and formulas

You can do anything with `replace` that you can do with `generate`. The only difference between the commands is that `replace` requires that the variable already exist, whereas `generate` requires that the variable be new. In fact, inside Stata, `generate` and `replace` have the same code. Because Stata is an interactive system, we force a distinction between replacing existing values and generating new ones so that you do not accidentally replace valuable data while thinking that you are creating a new piece of information.

References

Gleason, J. R. 1997a. dm50: Defining variables and recording their definitions. *Stata Technical Bulletin* 40: 9–10. Reprinted in *Stata Technical Bulletin Reprints*, vol. 7, pp. 48–49. College Station, TX: Stata Press.

——. 1997b. dm50.1: Update to defv. *Stata Technical Bulletin* 51: 2. Reprinted in *Stata Technical Bulletin Reprints*, vol. 9, pp. 14–15. College Station, TX: Stata Press.

Newson, R. B. 2004. Stata tip 13: generate and replace use the current sort order. *Stata Journal* 4: 484–485.

Royston, P. 2013. marginscontplot: Plotting the marginal effects of continuous predictors. *Stata Journal* 13: 510–527.

Weesie, J. 1997. dm43: Automatic recording of definitions. *Stata Technical Bulletin* 35: 6–7. Reprinted in *Stata Technical Bulletin Reprints*, vol. 6, pp. 18–20. College Station, TX: Stata Press.

Also see

[D] **compress** — Compress data in memory

[D] **corr2data** — Create dataset with specified correlation structure

[D] **drawnorm** — Draw sample from multivariate normal distribution

[D] **edit** — Browse or edit data with Data Editor

[D] **egen** — Extensions to generate

[D] **encode** — Encode string into numeric and vice versa

[D] **label** — Manipulate labels

[D] **recode** — Recode categorical variables

[D] **rename** — Rename variable

[U] **12 Data**

[U] **13 Functions and expressions**