Stata How-to: Clean and Modify Data

Patrick Blanchenay

2021-12-27

Contents

1. Creating new variables with `generate` 1
- 1.1. Missing values when creating a variable . . . 2
- 1.2. See also . . . 2

2. Creating a variable equal to some summary statistic, using `egen` 3
3. Modifying/cleaning data 3
4. Practice 4

1. Creating new variables with `generate`

New variables can be created using the **`generate`** command. Once again, let's use the `nlsw88` system dataset:

```
sysuse nlsw88, clear
generate logwage = ln(wage)
```

The **`generate`** command does two things:

1. it creates a new variable called `logwage` (a new column in the dataset);
2. observation by observation, it assigns to `logwage` a value equal to the natural logarithm of the `wage` variable for that observation.

Most data creation or modification is done observation by observation, or row-wise; that is: for the first observation, look at the value of `wage`, take its log and assign to the newly created variable (column) called `logwage`; move to the second observation, look at the value of `wage`, take its log, assign to `logwage`, and so on.[1]

In general, the expression following the = sign can include mathematical operations (addition, subtraction, division, multiplication, exponentiation: $+ - / * \hat{}$) and standard functions such as ln(), exp(), sin(), etc.

Good Practice

When generating a new variable, it is good practice to give the variable a label, which will be used by Stata in various outputs. This is done with **`label variable`**:

```
// Generate a log-wage and label it
generate logwage = ln(wage)
label variable logwage "Log(wage)" // the label is enclosed in double quotes
```

[1] In reality, Stata uses matrix operations to do this much faster that computing row by row, but from a logical point of view, the value of the newly created variable for a given observation is computed based on values of other variables for that same observation.

1.1. Missing values when creating a variable

If for an observation, the variable being created cannot be computed, Stata assigns a missing value to that observation. Whenever Stata is asked to create or modify data, it will report how many missing values were created in the process.

Missing values usually happen for one of two reasons:

- The value cannot be mathematically computed, for example dividing by zero, taking the logarithm of zero, etc.
- The new variable is computed based on a variable for which the value is missing.

See for instance these commands below:

```
. gen logtenure= ln(tenure)
(66 missing values generated)

. gen yearhours = hours * 52
(4 missing values generated)
```

The first commands create a new variable `logtenure` equal to the logarithm of `tenure`. 66 missing values are generated because these observations are individuals with 0 years of job tenure, and it is not possible to compute the ln() of zero. The second command creates a variable `yearhours` equal to the number of hours work per year, by multiplying weekly hours `hours` by 52. The command generated 4 missing values because the variable `hours` is missing for 4 observations.

I include another fictitious example to show that the problem is compounded when the new variables depends on several existing variables. Suppose we have two variables `var1` and `var2`, and we compute a new variable `newvar` = $\frac{1}{var1} + var2$. The `newvar` will be missing when either variable is missing (or when the mathematical expression cannot be computed).

var1	var2	newvar
1	1	2
.	1	.
3	2	2.333
0	3	.
2	4	4.5
1	.	.
.	.	.

This can be important in certain contexts: if your analysis focusses on log-wages, you should remember that your analysis will implicitly exclude all individuals that have wages equal to 0.

1.2. See also

See also Stata How-to: Conditions, subsetting data on how to use conditions when creating new variables.

2. Creating a variable equal to some summary statistic, using **egen**

On occasions you may need to create a variable equal to the mean of a group, or the max of a group, etc. To do we use the more complicated command **egen**, combined with a **by()** clause that will specify how we define groups.

Let's use the system dataset `nlsw88`. Suppose we want to find out how much more (or less) individuals earn compared to the average wage within their industry. We will first create a variable equal to the mean wage within each industry, using **egen**; we then create a variable equal to the difference between the individual's wage and their industry mean wage, using **generate**.

```
// First compute the mean wage in each industry
egen industrywage = mean(wage), by(industry)
label variable industrywage "Avg. industry wage"
// Then compute the difference
generate wage_inddiff = wage - industrywage
label variable wage_inddiff "Wage diff with industry avg."
```

The first command computes the average wage, industry by industry. For all individuals that work in the same industry, the variable `industrywage` will take the same value, even though the individuals differ in their actual wages. The **generate** command then compute the difference.

If instead we wanted to compute the average wage by industry-occupation combinations, we would use a more complicated **by()** clause:

```
egen indoccwage = mean(wage), by(industry occupation) // notice the by() clause
label variable indoccwage "Avg. wage by industry-occupation"
```

If no **by()** clause is specified, the overall average would be computed, that is, the average wage in the whole dataset. The new variable will take the same value for all observations:

```
egen meanwage = mean(wage)  // no by() clause
label variable meanwage "Avg. wage"
```

We can use **egen** command to compute other summary statistics than averages, for instance **sd()**, **min()**, **p50()** (median), etc. The list of accepted functions is available in **help egen**.

3. Modifying/cleaning data

In ECO372, you will generally receive already prepared datasets. But sometimes you need to "clean" your data, i.e. correct or remove erroneous values (that might have been mistyped, or misreported). This is done using the **replace** command, which assigns new values to already existing variables. This is usually used in combination with **if** conditions, so you should first read Stata How-to: Conditions, subsetting data on how to use **if** conditions.

Example 1: Explicit missing values. In the original dataset, missing data is actually recorded as a specific value, e.g. 999, when unavailable. For instance, in the raw CPS data , missing wages are recorded as $9,999,999. These observations should be set to missing (**.**) to let Stata know that these are not actual values.

```
replace incwage = . if(incwage == 9999999) // missing income
replace educ = . if (educ == 1) // missing level of education
```

See the section on value labels in Stata How-To: Explore and Describe Data to spot these.

Example 2: Topcoding. In some datasets, some variables are "topcoded": values higher than a certain threshold are coded as an artificially very high value, or as the average of values above that threshold. (This often used to preserve confidentiality.) Solution 1: ignore the problem (and some incomes will be underestimated). Solution 2: set all of these averaged incomes to missing, to reflect that fact that you do not exactly know these incomes.

Example 3: Missing as zeroes. In some datasets, missing values should actually be coded as zeros. For instance, in a dataset of exam grades, missing values represent students who did not take the test, and they should receive a zero.[2]

```
  replace grade = 0 if (missing(grade))
```

Example 4: Misrecording. An observation is obviously badly recorded. In a dataset recording heights of individuals in centimeters, values will typically be 153, 187, 171, etc. If for an observation, height is recorded as 5.9, you may want to set it to missing, or if you think this is height in feet, convert it to 180 centimeters.

```
  replace grade = 0 if (missing(grade))
```

Example 4: Misrecording. An observation is obviously badly recorded. In a dataset recording heights of individuals in centimeters, values will typically be 153, 187, 171, etc. If for an observation, height is recorded as 5.9, you may want to set it to missing, or if you think this is height in feet, convert it to 180 centimeters.

```
replace height = 167 if (height == 1670) //  misrecorded height
replace height = 180 if (height == 5.9) //  problem of units
```

Example 5: Spelling mistake. A string variable is mistyped for an observation. For instance, the `address_street` variable records "St-Goerge st.", and this prevents you from grouping all observations located on St-George street.

```
replace address_street = "St-George st." if (address_street == "St-Goerge st.") // correcting typo
```

4. Practice

Create a folder for this project, and a do-file in that folder. Download the raw CPS data into that folder. Use the do-file to:

1. ☐ Add a comment description of the do-file at the top; and set the working directory.
2. ☐ Load the untreated CPS data.
3. ☐ For the `educ` variable, find what values recording missing or unavailable data. Set `educ` to missing for these observations. (You should have 609 missing out of 2787.)
4. ☐ For the `incwage` variable, find what values recording missing or unavailable data, or just weird values. Set `incwage` to missing for these observations. (You should have 610 missing out of 2787.)
5. ☐ For the `marst` variable, find what values recording missing or unavailable data. Set `marst` to missing for these observations. (You should have no missing.)
6. ☐ For the `race` variable, find what values recording missing or unavailable data. Set `race` to missing for these observations. (You should have no missing.)
7. ☐ Check that your do-file can run from top to bottom without generating any error.

[2] This is fictitious; but make sure to check the ECO372 syllabus for the course grading policy.