Stata Community Resource

Contents:

Written on 1/27/2019 (examples shown running version 15).

Data storage types: Numeric

Unlike some other statistical packages (e.g., SPSS or SAS) Stata stores data as different "types." Each data storage type has a different level of accuracy and therefore different memory storage requirements.

For example, if we have a dataset loaded already, and want to examine our storage types:

```
. describe
```

```
Contains data from Schaffer_Homework3_SOC561.dta
  obs:         2,867
 vars:           961                          25 Jan 2019 11:30
 size:     3,331,454
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
mar1            byte    %8.0g      LABA       marital status of 1st person
mar2            byte    %8.0g      LABA       marital status of 2nd person
mar3            byte    %8.0g      LABA       marital status of 3rd person
mar4            byte    %8.0g      LABA       marital status of 4th person
```

Here we can see that the storage type is listed as "byte." Byte indicates that the variable is stored as an integer between -127 and 100.

The default data storage type for Stata is "float."

By inquiring with Stata using the `help` command, we see that the float variable type is much larger relative to byte:

```
. help datatypes
```

Storage type	Minimum	Maximum	Closest to 0 without being 0	Bytes
byte	−127	100	±1	1
int	−32,767	32,740	±1	2
long	−2,147,483,647	2,147,483,620	±1	4
float	$-1.70141173319 \times 10^{38}$	$1.70141173319 \times 10^{38}$	$\pm 10^{-38}$	4
double	$-8.9884656743 \times 10^{307}$	$+8.9884656743 \times 10^{307}$	$\pm 10^{-323}$	8

Also note that byte, int, and long storage types can only hold integers, whereas float and double are floating point data storage types.

Another difference between the various numerical data storage types is in their levels of precision. For example, float variable type data will be stored with a 7-digit level of accuracy. This means that Stata will not round a number stored as a float type, as long as the number is 7 total digits or fewer. If precision is needed past the 7-digit limit, however, the "double" storage type should be used.

One potential complication of storing a >7 digit variable as a "float" or smaller is that Stata may round down the integer without advertising that it is doing so. We can observe that this occurs by requesting for Stata to display an 8-digit integer as "float":

```
. display float(16777217)
16777216
```

We can observe that Stata does default to "float" (as described above) when we ask Stata to create a new variable based on an existing variable that is stored as "byte":

```
. generate variable4 = mar1
(71 missing values generated)

. describe variable4

              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
variable4       float   %9.0g
```

If you would like to specify a different data storage type when creating a variable, you may do so:

```
. generate double variable3 = mar1
```

Here, *double* indicates that the new variable "variable3" should be stored as the double data type.

Since the double storage type does not compromise precision, and since other storage types might, one approach is to set the default storage type to double:

```
. set type double, permanently
```

Note that the tradeoff for setting the default storage type to double is in increased memory and processing time. However, it is possible to reverse the data storage type later, if this becomes a problem. We will discuss changing storage types in a later section.

Data storage types: String

Strings are variables that Stata does not treat as numeric, and therefore Stata cannot run computations on strings. Strings can be letters only, numbers only, or a combination of letters and numbers. Strings may also be composed of keyboard symbols:

```
. generate variable = "$#|**"
```

When adding a string variable to Stata, note that it must be referred to in quotation marks:

```
. generate variable20 = "numbers"
```

Note that this command tells Stata that the string variable should be stored as a 7-character string (because the word "numbers" has 7 letters).

Changing data storage types

Sometimes it might be necessary to change the data storage type of an existing variable:

```
. recast str25 variable20
```

Here, we asked Stata to convert the data storage type of our variable20 to a string with 25 characters.

We might also need to convert a variable stored as a string to one that is stored as a numeric variable. For example, if a variable that you would like to run computations on is stored as a string, it would need to first be converted to a numeric variable:

```
. destring variable20, generate variable21
```

Conversely, we can ask Stata to convert a variable stored as numeric to a string variable:

```
. generate variable22 = tostring(age)
```

Here, we have asked Stata to produce a new variable from the existing variable "age."

Compressing Data

As we discussed above, data storage types have various implications for processing time and memory demands. In the event that data storage decisions are adversely affecting processing time, the command "compress" can be applied globally, to the entire dataset. Stata will evaluate each variable and apply a more economical data storage solution whenever possible. For example, if I have stored a variable as str30, but none of the entries for that variable require more than 5 characters, running the compress command will result in Stata "demoting" that variable to str5. The same process will occur for the numeric variables, as well:

```
. compress
  variable variable4 was float now byte
  variable variable3 was double now byte
  (28,670 bytes saved)
```

Here, Stata demoted the variable4 to byte, as Stata determined that storing it as "float" was a waste of data for this particular variable.

Variable Naming

Stata makes a distinction between variable names, variable labels, and value labels. When running commands in Stata, variables are called by their variable names (rather than their variable labels).

To see a list of all of the existing variable names, their value labels, and their variable labels:

```
. describe
```

We can also run the *describe* command on just one of the variables:

```
. describe spsector

              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
spsector        byte    %8.0g      LABCH      type of college spouse attended
```

Here we observe the 3 distinct pieces of information that Stata is storing about this variable: its name, its value label, and its variable label.

Naming Decisions

The names of variables should be intuitive (communicating meaning) and consistently applied across the dataset. While Stata allows massively long variable names (up to 32 characters), the presence of extraneous characters might increase the risk of copying errors. Variable names should also be sufficiently distinct from one another to avoid conflating two different variables.

Variable naming decisions should also be considered as part of your data organization scheme. Adding a sequence count number to the end of the variable name will enable Stata to display the variables in whatever order makes logical sense for your analyses.

When new variables are created, Stata requires that their names be specified. For example, when using the command "generate," Stata will return an error if the variable name has not been specified:

```
. generate byte = mar1
too few variables specified
r(102);
```

Above, we asked Stata to produce a byte-stored variable type from the existing variable named mar1, but Stata refused to heed our request since the variable name was not specified in the command.

We will now provide the name "mar23" and ask Stata again:

```
. generate byte mar23 = mar1
(71 missing values generated)
```

This time, Stata complies with the request.

If we want to rename the variable mar23 a different name, we can do that as well:

```
. rename mar23 marital_status_1
```

Above, we have used underscore to delineate the words *marital*, *status*, and *1,* to help with readability and interpretability, and have used a name that is more indicative of what this variable is measuring (in this case, the marital status of the first survey respondent).

Labels and Notes

Stata labels can be applied to a variable, to the values of a variable, or to the entire dataset. We will discuss each of these in turn.

Variable labels

Unlike variable names, Stata does not require that variable labels be specified at the point of new variable creation.

To provide a variable label, we can use "label variable":

```
. label variable marital_status_1 "marital status of the first survey respondent"
```

Here we can observe that the variable label provides useful information about what the variable measures.

Value Labels

Another distinct aspect of Stata compared to other stats packages is that the value labels are specified prior to their assignment to a variable, and, once defined, a set of value labels can be applied to multiple variables. Now we will define a new set of value labels for employment status:

```
. label define employment_status 1 "employed" 0 "not employed"
```

This set of value labels can be applied to variables:

```
. label values employed_current employment_status
```

Above, we direct Stata to label values of a variable named "employed_current" with the employment status value labels we just created.

To see what value labels have been defined for your dataset:

```
. label list
```

To see a specific label value list:

```
. label list employment_status
employment_status:
           0 not employed
           1 employed
```

In the event that you need to modify a value label list by adding another value:

```
. label define employment_status 2 "disabled", add
```

Here, we have told Stata to add a value, 2, which we label with "disabled." In our dataset, a "2" will indicate that the participant responded as "disabled" to the question about employment status.

Note that when creating a new variable based on an existing variable (e.g., "generate") Stata will not copy the existing data value labels to the new variable. If these should be copied to the new variable from the old, use:

```
. clonevar marital_status_5 = mar1
```

This command instructs Stata to copy the existing values, value labels, variable label, and any notes from "mar1" to a new variable that we named "marital_status_5."

Notes

Notes can be attached to the dataset currently in use, or to specific variables. To attach a "note to self" in the dataset, in this case about missing variables:

```
note: check missing variables
```

To attach a note to a specific variable, such as v1**:**

```
notes v1: the recode needs to be checked
```

One useful piece of information to attach in a note is the "data signature": a data signature is part of the Stata meta-data (data about data). The "data signature" is a numerical representation of the current characteristics of the dataset, such as the number of variables. When one of these characteristics changes in any way, the data signature changes. This function is useful for tracking whether a dataset has been changed, or verifying that it has not been changed.

To add the data signature to the log file, first display the number (if not currently displayed) and then add it to a note:

```
. datasig
  2867:969(101857):3870183183:1367648850

. note:  2867:969(101857):3870183183:1367648850
```

The data signature can also be associated with the dataset itself. To set the data signature as a characteristic of the dataset (once the dataset is saved):

```
. datasignature set
```

Using "datasignature confirm" performs a verification function: if the data have changed since the data signature was set, Stata will generate an error code. This function provides a fail-safe option for ensuring data integrity if included in a do-file. If "datasignature confirm" is written into the "do-file," the processing of the do-file would stop in the event that the file had changed in the interim since the signature had been set and recorded in the do-file.

Sources and Resources

Bardsley, P., Chantala, K., & Blanchette, D. Precision and data storage. *Stata Tutorial.* Retrieved from https://www.cpc.unc.edu/research/tools/data_analysis/statatutorial/misc/precision

Cox, N. J., & Newton, H. J. (2014). *One hundred nineteen stata tips* (Third edition. ed.). College Station, Texas: A Stata Press Publication/Stata Corp LP.

Long, J. S. (2009). *The workflow of data analysis using Stata*. College Station, Tex.: Stata Press.

Ludwig-Mayerhofer, W. (2018, 12/28/2018). Data and Storage Types. *Internet Guide to Stata.* Retrieved from http://wlm.userweb.mwn.de/Stata/wstatvar.htm

Statistical Consulting Group. Labeling data *Stata learning modules.* Retrieved from https://stats.idre.ucla.edu/stata/modules/labeling-data/