# The Stata Journal



The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*, *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

**Subscriptions** are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

**Subscription rates** listed below include both a printed and an electronic copy unless otherwise mentioned.

| U.S. and Canada | | Elsewhere | |
|---|---|---|---|
| **Printed & electronic** | | **Printed & electronic** | |
| 1-year subscription | $ 98 | 1-year subscription | $138 |
| 2-year subscription | $165 | 2-year subscription | $245 |
| 3-year subscription | $225 | 3-year subscription | $345 |
| 1-year student subscription | $ 75 | 1-year student subscription | $ 99 |
| 1-year institutional subscription | $245 | 1-year institutional subscription | $285 |
| 2-year institutional subscription | $445 | 2-year institutional subscription | $525 |
| 3-year institutional subscription | $645 | 3-year institutional subscription | $765 |
| **Electronic only** | | **Electronic only** | |
| 1-year subscription | $ 75 | 1-year subscription | $ 75 |
| 2-year subscription | $125 | 2-year subscription | $125 |
| 3-year subscription | $165 | 3-year subscription | $165 |
| 1-year student subscription | $ 45 | 1-year student subscription | $ 45 |

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.



The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata, STATA, Stata Press, Mata, MATA, and NetCourse are registered trademarks of StataCorp LP.

The Stata Journal (2013)
**13**, Number 4, pp. 862–866

# cmpute: A tool to generate or replace a variable

Patrick Royston
Hub for Trials Methodology Research
MRC Clinical Trials Unit at UCL
London, UK
pr@ctu.mrc.ac.uk

**Abstract.** I provide a new programming tool, `cmpute`, to manage conveniently the creation of a new variable or the replacement of an existing variable interactively or within a Stata program.

**Keywords:** dm0072, cmpute, data management, create variable, replace variable, ado-file programming

## 1 Introduction

In Stata programs that I write, I am often faced with safely managing the creation of new variables to be stored in the workspace alongside user data. For example, I do not wish to overwrite existing user variables without warning. An obvious precaution is to include a `replace` option in the program so that the user can sanction overwriting a variable when appropriate. However, accurately handling the presence or absence of `replace` and the creation of a new variable is programmatically cumbersome.

In this short article, I describe a new tool, `cmpute`, to streamline the "regeneration" (creation or replacement) of a variable subject to certain sensible constraints. `cmpute` has some features in common with an earlier program, `defv` (Gleason 1997, 1999). However, the aims of `defv` are different. A key goal of `defv` is to enable the documentation of changes to an important variable by accumulating characteristics (as `note`s), possibly over many sessions with a particular dataset. My main goal with `cmpute` is to streamline the creation or replacement of variables within a Stata program. While it is fine that `cmpute` may be found useful interactively, that's not my goal.

From its original release, Stata has separated the creation of new variables (done with `generate`) from the revision of the contents of existing variables (done with `replace`). Furthermore, while you can abbreviate `generate` all the way down to `g` if you wish (in practice, most people use `gen`), you cannot abbreviate `replace`. These decisions all flow from Stata's underlying philosophy of protecting your data and of making it as difficult as possible for you to change your data unless you spell out explicitly that this is your intention.

 

In proposing to do what Stata's designers in their wisdom cast asunder, I am consciously favoring programmer convenience while also reducing any element of risk by protecting users against inadvertent changes to their data. (Note: If you specify the `force` option of `cmpute`, be aware that it means what it says. The effects of a force may be drastic.)

`cmpute` has a loose connection with the official command `clonevar`, which precisely reproduces the data and all other features of an existing variable in a new variable.

## 2 Example

Consider the following simple program:

```
program define mylog
// Program to safely create a log transformation of a single variable
version 12.1
syntax varlist(min=1 max=1 numeric), generate(string) [replace]
capture confirm var `generate´
// `generate´ does not exist; it´s safe to create it and finish
if c(rc) != 0 {
    generate `generate´ = ln(`varlist´)
    exit
}
// `generate´ does exist; it must be handled correctly
if "`replace´" == "replace" {
    replace `generate´ = ln(`varlist´)
}
else {
    display as error "`generate´ already defined"
    error 110
}
end
```

The program accepts a variable supplied in `varlist` and creates a new variable called *string*, stored in a local macro called `generate`, containing the logarithmically transformed values of `` `varlist´ ``. `mylog` replaces the contents of the variable *string* if it already exists—provided that the `replace` option is specified. If the `replace` option is not specified, an error message must be issued because we do not wish to wipe out the existing *string* without permission. The above program is not completely foolproof, but on the whole, it does a reasonable job of handling various possible inputs and the existence or otherwise of the variable *string*. There must be thousands of programs out there containing lines of code that do something similar. If more than one variable is to be handled, the code can get quite bulky (and ugly).

We could replace chunks of code like that in `mylog` with a single call to the new program, `cmpute`. For example,

```
. cmpute logx = ln(x), replace
```

does essentially the same thing as

```
. mylog x, generate(logx) replace
```

Of course, `cmpute` is much more general; within Stata's limits, it can handle an arbitrarily complex expression after the `=` sign.

# 3 Syntax

The syntax of `cmpute` is as follows:

`cmpute` [ *type* ] {*existing_var* | *newvar*} `=` *exp* [ *if* ] [ *in* ] [ `, force` `label(`*string*`)` `replace` ]

## 3.1 Description

`cmpute` replaces an existing variable, *existing_var*, or creates a new variable, *newvar*, from an expression in *exp*. An error message occurs if an attempt is made to change *existing_var* without specifying `replace`. If *type* is specified, `cmpute` sets the storage type of *existing_var* or *newvar* to *type* (see also the `force` option). *type* must be one of `byte`, `int`, `long`, `float`, `double`, `str`#, or in Stata 13 or higher, `strL`.

Note that `cmpute` leaves formats, value labels, and characteristics as they were, so a programmer wanting to alter any of those needs to make the changes separately.

Although `cmpute` is envisaged primarily as a programmer's tool, users may also find it convenient in interactive use as a shortcut to creating and labeling a new (or existing) variable in one step.

## 3.2 Options

`force` applies `recast` to force a change in the storage type of an *existing_var* to *type*. This option should be used with caution because it could result in loss of data. See help on `recast` for further information. `force` has no effect on a *newvar*.

`label(`*string*`)` labels the new or regenerated variable "*string*".

`replace` replaces *existing_var*. Using `cmpute` with an existing variable but omitting `replace` raises an error message. `replace` has no effect on a *newvar*.

## 3.3 Examples: Interactive use

The examples given below are of interactive use. See section 4 to get an idea of `cmpute`'s utility in programming.

```
. cmpute str6 make = substr(make, 1, 6), replace label("Make (trunc)")
. cmpute int gear_ratio = int(100 * gear_ratio), replace force
. cmpute logx = ln(x), label("log(x)")
```

# 4 Example: Programming use

Here is a simple program, an extension of `mylog`, that uses `cmpute` to manage the creation of new variables:

```
program define mylog2
version 12.1
syntax varlist(min=1 numeric) [if] [in], generate(string) [replace]
marksample touse
local nvar : word count `varlist´
tokenize `varlist´
forvalues i = 1 / `nvar´ {
    cmpute double `generate´`i´ = ln(``i´´) if `touse´, `replace´ ///
       label("ln(``i´´)")
}
end
```

`mylog2` log-transforms a list of variables in `` `varlist´ ``. As you can see, the aim here is to implement an option whose syntax is `generate(`*name*`)`. The option saves permanently a bunch of new or replaced variables whose names begin with *name*. If the `replace` option is omitted, the `cmpute` ... line will raise an error if a variable called `` `generate´`i´ `` already exists for some `i`. If `replace` is used, all such variables are silently overwritten.

I have requested that the log-transformed variables `` `generate´1 ``, `` `generate´2 ``, ..., `` `generate´`nvar´ `` be stored in double precision, and I have simultaneously labeled them meaningfully. The local macro ``` ``i´´ ``` evaluates to the `i`th token (element) in `` `varlist´ ``, that is, to the `i`th variable name.

Note: I have written `mylog2` such that if any member of `varlist` has a missing value in a given observation not due to the `if` and `in` qualifiers, that observation becomes missing in all the generated variables. The reason is that `marksample` has automatically incorporated missingness of members of `varlist` in the indicator temporary variable `touse`. I could easily change such behavior if that is not what is wanted. For example, the `cmpute` ... line could instead be coded

```
cmpute double `generate´`i´ = ln(``i´´) `if´ `in´, `replace´ label("ln(``i´´)")
```

which would preserve all original values of variables in `` `varlist´ `` except where filtered by either the `if` or the `in` qualifier or of course by an attempt to log transform a nonpositive value.

# 5 Summary

`cmpute` is meant as an interactive command or a programming tool. In a program, you often wish to create a new variable or replace an existing one, and you also have implemented a `replace` option to allow an existing variable to be overwritten. `cmpute` handles the necessary coding and (critically) the error checking in a single call. Doing this properly line by line within your program is cumbersome. `cmpute` also supports expressions via `=`*exp* and supports labeling and recasting a regenerated variable.

# 6 Acknowledgment

I am most grateful to Nick Cox for clarifying the original presentation of `cmpute` and for providing helpful comments on the manuscript, which have led me to significant improvements.

# 7 References

Gleason, J. R. 1997. 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.

———. 1999. 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.

### About the author

Patrick Royston is a medical statistician with more than 30 years of experience, with a strong interest in biostatistical methods and in statistical computing and algorithms. He works largely in methodological issues in the design and analysis of clinical trials and observational studies. He is currently focusing on alternative outcome measures in trials with a time-to-event outcome; on problems of model building and validation with survival data, including prognostic factor studies and treatment-covariate interactions; on parametric modeling of survival data; and on novel clinical trial designs.