

THE STATA JOURNAL

Editor

H. Joseph Newton
Department of Statistics
Texas A & M University
College Station, Texas 77843
979-845-3142; FAX 979-845-3144
jnnewton@stata-journal.com

Associate Editors

Christopher F. Baum
Boston College

Rino Bellocco
Karolinska Institutet, Sweden and
Univ. degli Studi di Milano-Bicocca, Italy

A. Colin Cameron
University of California–Davis

David Clayton
Cambridge Inst. for Medical Research

Mario A. Cleves
Univ. of Arkansas for Medical Sciences

William D. Dupont
Vanderbilt University

Charles Franklin
University of Wisconsin–Madison

Joanne M. Garrett
University of North Carolina

Allan Gregory
Queen's University

James Hardin
University of South Carolina

Ben Jann
ETH Zürich, Switzerland

Stephen Jenkins
University of Essex

Ulrich Kohler
WZB, Berlin

Stata Press Production Manager

Stata Press Copy Editor

Editor

Nicholas J. Cox
Department of Geography
Durham University
South Road
Durham City DH1 3LE UK
n.j.cox@stata-journal.com

Jens Lauritsen
Odense University Hospital

Stanley Lemeshow
Ohio State University

J. Scott Long
Indiana University

Thomas Lumley
University of Washington–Seattle

Roger Newson
Imperial College, London

Marcello Pagano
Harvard School of Public Health

Sophia Rabe-Hesketh
University of California–Berkeley

J. Patrick Royston
MRC Clinical Trials Unit, London

Philip Ryan
University of Adelaide

Mark E. Schaffer
Heriot-Watt University, Edinburgh

Jeroen Weesie
Utrecht University

Nicholas J. G. Winter
University of Virginia

Jeffrey Wooldridge
Michigan State University

Lisa Gilmore
Gabe Waggoner

Copyright Statement: The Stata Journal and the contents of the supporting files (programs, datasets, and help files) are copyright © by StataCorp LP. The contents of the supporting files (programs, datasets, and help files) may be copied or reproduced by any means whatsoever, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the Stata Journal.

The articles appearing in the Stata Journal may be copied or reproduced as printed copies, in whole or in part, as long as any copy or reproduction includes attribution to both (1) the author and (2) the Stata Journal.

Written permission must be obtained from StataCorp if you wish to make electronic copies of the insertions. This precludes placing electronic copies of the Stata Journal, in whole or in part, on publicly accessible web sites, file servers, or other locations where the copy may be accessed by anyone other than the subscriber.

Users of any of the software, ideas, data, or other materials published in the Stata Journal or the supporting files understand that such use is made without warranty of any kind, by either the Stata Journal, the author, or StataCorp. In particular, there is no warranty of fitness of purpose or merchantability, nor for special, incidental, or consequential damages such as loss of profits. The purpose of the Stata Journal is to promote free communication among Stata users.

The *Stata Journal*, electronic version (ISSN 1536-8734) is a publication of Stata Press. Stata and Mata are registered trademarks of StataCorp LP.

Stata tip 39: In a list or out? In a range or out?

Nicholas J. Cox
Department of Geography
Durham University
Durham City, UK
n.j.cox@durham.ac.uk

Two simple but useful functions, `inlist()` and `inrange()`, were added in Stata 7, but users somehow still often overlook them. The manual entry [D] **functions** gives formal statements on definitions and limits. The aim here is to emphasize with examples how natural and helpful these functions can be.

The question answered by `inlist()` is whether a specified argument belongs to a specified list. That answered by `inrange()` is whether a specified argument falls in a specified range. We can ask the converse question, of not belonging to or falling outside a list or range, by simply negating the function. Thus `!inlist()` and `!inrange()` can be read as “not in list” and “not in range”.

These functions can reduce your typing, reduce the risk of small errors, and make your Stata code easier to read and maintain. Thus with the `auto` data in memory, consider the choice for the integer-valued variable `rep78` between older ways of getting a simple listing,

```
. list make rep78 if rep78 == 3 | rep78 == 4 | rep78 == 5
. list make rep78 if rep78 >= 3 & rep78 <= 5
. list make rep78 if rep78 > 2 & rep78 < 6
```

and newer ways of getting the same listing,

```
. list make rep78 if inlist(rep78, 3, 4, 5)
. list make rep78 if inrange(rep78, 3, 5)
```

The examples here are typical of a good way to use `inlist()` or `inrange()`: move directly from feeding arguments to each function to using the results of the calculation. If you wanted to keep the results, you could put them into a variable (or a macro). The result of `inlist()` or `inrange()` is either 1 when the value specified is in range or in list and 0 otherwise (and thus never missing). So, if you use a variable to store results, let it be a byte variable for efficiency in storage.

In more detail: so long as none of the arguments z, a, b is missing, `inrange(z, a, b)` is true whenever $z \geq a$ and $z \leq b$. Thus `inrange(60, 50, 70)` is true (numerically 1) because $60 \geq 50$ and $60 \leq 70$. However, `inrange(60, 70, 50)` is false (0) because 60 is not ≥ 70 and 60 is not ≤ 50 . Thus the order of a and b is crucial. There are situations when you are not sure in advance about the ordering of arguments, but you can always use devices such as `inrange($z, \min(a, b), \max(a, b)$)` (which tests whether one value is between two others).

The definition of `inrange()` is more complicated when any argument is numeric missing. See [D] **functions** for the precise definitions. The most important example is

that `inrange(z, a, .)` is interpreted as $z \geq a$ and $z < .$ (z greater than or equal to a , but not missing). This may look like a bug, but it is really a feature. Even experienced users sometimes forget that in Stata numeric missing is regarded as arbitrarily large. Hence, `z >= 42` will be true for all the missing values of `z`, as well as for all values that are greater than or equal to 42. The longstanding workaround when this is not what you want with regard to missing values is to add the extra condition that `z` is not missing, as in `z >= 42 & z < .`, but `inrange(z, 42, .)` is another way to do this.

The definitions that come into play when any argument is missing imply that `inrange()` is not a good tool to use when you want to test for numeric missings (including any comparisons with extended missing values). For that it is better to use `missing()`, `inlist()`, or combined statements using simple inequalities.

`inlist()` and `inrange()` can often be used with the in-built quantities `_n` and `_N` specifying, respectively, the current observation number and the current number of observations. Sometimes users wish to specify that a command should apply to an irregular set of observation numbers, and if `inlist(_n,17,42,99,217)` exemplifies how that could be done with a small set (the limit is 255 numbers and is unlikely to bite in sensible practice). A pitfall here is clearly that any sorting of the dataset will often imply that the observations concerned end up in different positions. Thus saving the results of this computation in a byte variable will often be a good idea. This approach is not better general practice than using criteria such as those based on variable values, but there may be occasions when you will want this feature.

Other examples of the same kind arise with longitudinal or panel data. Recently I wanted to identify the first and last values of a response in each panel, and

```
. by panelvar (timevar): gen y_ends = y if inlist(_n, 1, _N)
```

offers a way to do that. Conversely, `!inlist(_n, 1, _N)` identifies all the others. Whether you prefer that `if` condition to the more traditional `if _n == 1 | _n == _N` is admittedly a matter of taste. Using `in` is not an option here because `in` may not be combined with `by`.

The examples so far are all for numeric arguments. The arguments of either function can be all numeric or all string. Thus given one character, `c`, `inrange("c", "a", "z")` tests whether `c` is one of the 26 lowercase letters of the English alphabet; correspondingly, `inrange("c", "A", "Z")` tests whether `c` is one of the 26 uppercase letters of the same alphabet. More generally, `inrange("string", "a", "z")` tests whether `string` begins with a lowercase letter, and correspondingly for the arguments "A", "Z" and uppercase letters. Because lowercase and uppercase letters are typically not adjacent in your computer's character sets, be careful when working with both. If you were indifferent about the distinction between uppercase and lowercase, you could work with `lower("string")` or `upper("string")`.

These examples lead directly to a way of filtering a string variable to select characters that you want or ignore characters that you don't. Suppose that we wanted to select only the alphabetic characters in a string variable. Check the variable type to see its maximum length (18, or whatever), **generate** a new empty-string variable, and then loop over the characters, adding them to the end of the new variable only if they are as desired.

```
generate newvar = ""
quietly forvalues i = 1/18 {
    replace newvar = newvar + substr(oldvar,'i',1) ///
        if inrange(lower(substr(oldvar,'i',1)),"a","z")
}
```

Commands like this tend to become rather long, but they are not in principle complicated. The attraction of a low-level approach is that you can design exactly the filter you wish according to the precise problem it is intended to solve.

A further simple but general moral evident from various examples here is that the power of Stata functions often arises from how they can be combined.