Checking data with the *datacheck* utility

Krishnan Bhaskaran
Krishnan.bhaskaran@lshtm.ac.uk

London School of Hygiene and Tropical Medicine, UK

Abstract

datacheck is a simple utility for carrying out data checks and reporting on problems or errors in a dataset. A *datacheck* statement specifies a condition which one or more variables in the dataset must meet. If the data "fail" the check, the program displays meaningful and customizable output identifying the problem observation(s). In contrast to *assert*, *datacheck* does not halt Stata or produce any errors, so a number of *datacheck* statements can be included in a do-file to run in sequence. This allows a suite of data checks to be performed in a single run, with the customized output serving as a user-friendly log detailing problems with the data.

Keywords: data management, data check, consistency, consistency check, data cleaning, check, verify assert, assertk.

1. Introduction

Data checking and "cleaning" is a substantial part of the work of many data managers, statisticians, and other data users. *datacheck* is a simple utility that aims to make checking and reporting on data quality easier. The main functions of the command are to check that one or more variables in a dataset meet a specified condition, and to report on any observations in the data where that condition is not satisfied.

Stata already has a built-in command, *assert,* which can be used to check data against specified conditions and thus verify the truth of a claim (Gould, 2003). *assert* is useful for checking important assumptions about the data. It can be used as a safety check following complicated code: for example, after running code to produce stabilized weights, you could check that the weights generated always lie between -1 and +1. The command can also be used to check for more basic errors in a dataset; for example, you could check that "sex" is always coded as male or female:

```
. assert sex=="m"| sex=="f"
. 3 contradictions in 2740 observations
assertion is false
r(9);
```

As the above output shows, a key feature of Stata's built-in *assert* command is that if a false claim is made, the do-file or ado-file halts, and Stata delivers an error message. This can be a useful feature: if the file runs without errors, the user is reassured that the data have passed all the specified checks. If, on the other hand, one of the checks fails, he or she is forced to intervene and deal with the problem observation(s), before carrying out any further processing of data that may not meet important assumptions.

However, *assert* does have its limitations as a data checking tool. Firstly, if a data check produces an error, further coding is needed to identify the offending observations. Secondly, it may not be convenient to deal with data problems one by one, and in the order in which they are identified.

datacheck aims to address these limitations. Data checks are specified in the same way as with *assert*, but when one or more observations fail a check, customizable output is produced, immediately identifying the observation(s) that have failed the check. In contrast with the *assert* command, the do-file or ado-file containing the data check then continues without error; a key advantage of this behavior is that a sequence of *datacheck* statements, corresponding to a suite of data checks, can be included in a single do-file and run without interruption. All problems with the data will be output in a meaningful and readable report which can then be used as a basis for querying, correcting, or otherwise dealing with apparent data errors.

2. Syntax

```
datacheck true_or_false_condition [if condition] [in range] [ , by(byvarlist)
    message(string) varshow(varlist) previous next flag nolist list_options ]
```

3. Options

`by(byvarlist)` makes the assertion by `byvarlist`. This allows, for example, conditions referring to `_n` and `_N` defined within distinct groups of `byvarlist`. The dataset must be previously sorted by `byvarlist`. If this option is specified, list output is by default separated by `byvarlist`.

`message(string)` displays the given message string if any contradictions are found.

`varshow(varlist)` restricts list output to the variables in `varlist`. If this option is not specified, all variables in the dataset are listed.

`previous` and `next` list the previous and/or following observation as well as any observation contradicting the assertion. This can be especially useful when data are in time order.

`flag` leaves behind a binary flag variable in the dataset named _contra taking the value 1 for observations failing the check and 0 otherwise. This flag variable will automatically be dropped the next time `datacheck` is run, and a new variable will be generated the next time `datacheck` is run with this option.

`nolist` suppresses output of list. Output is restricted to a brief report on contradictions.

`list_options` are options of `list`.

4. Uses and examples

The following example illustrates the use of *datacheck* for a simple data check. Suppose that patients in Arm 1 of a clinical trial are supposed to be on aspirin; the check is as follows:

```
. datacheck drug=="aspirin" if arm==1, varshow(id drug arm) message(Wrong drug)
```

```
Wrong drug (3 obs)
    +----------------------+
    | id        drug   arm |
    |----------------------|
 1. |  1   ibuprofen     1 |
36. | 36   ibuprofen     1 |
41. | 41   ibuprofen     1 |
    +----------------------+
```

The single-line command produces a list of all patients taking the “wrong” drug. The *varshow* option allows the output to be limited to useful variables only, and the header message can be chosen to make the output of a large log file of checks easy to follow.

More complex checks are possible, for example here we check that hospital visits occur in chronological order, which involves checking across observations:

```
    . datacheck date>date[_n-1] if _n>1, varshow(visit date) message(Dates do not follow) prev

Dates do not follow (1 obs)
    +-------------------+
    | visit        date |
    |-------------------|
 3. |     3   13aug2003 |
 4. |     4   25jan2003 |
    +-------------------+
```

One data item fails: Visit 4 appeared to occur before after Visit 3. Using the `prev` option illustrates the problem clearly by showing not only the offending observation, but also the preceding one.

We may also take advantage of by-group processing using the `by` option. After sorting by *id* and *time*, here we check that each patient has their first record at time 0:

```
    . datacheck time==0 if _n==1, by(id) varshow(id time) message(Patient's 1st record not at time 0)

Patient's first record is not at time 0 (1 obs)
    +-----------+
    | id   time |
    |-----------|
55. | 19      3 |
    +-----------+
```

Again there is one offending observation in the dataset, in which the first observation for a patient is at time 3.

Summary

datacheck provides an easy way of performing and reporting data checks. Checks are specified in an intuitive way, and there are advantages over the *assert* command since data failing the check are immediately identified without further programming. Furthermore, a series of checks may be run in a do-file without “assertion is false” errors halting the program, and a customized and meaningful log of data errors will be produced.

About the author

Krishnan Bhaskaran is a lecturer in statistical epidemiology at the London School of Hygiene and Tropical Medicine, and a regular user of Stata for the last ten years.

References

Gould, W. 2003. Stata tip 3: How to be assertive. *The Stata Journal,* 3**,** 448.