Title

stata.com

assert — Verify truth of claim

Description

`assert` verifies that *exp* is true. If it is true, the command produces no output. If it is not true, `assert` informs you that the "assertion is false" and issues a return code of 9; see **[U] 8 Error messages and return codes**.

Quick start

Confirm that `v1` only takes values 0 or 1

```
assert v1==0 | v1==1
```

Verify that `v2` is between 100 and 200 and never missing

```
assert inrange(v2,100,200)
```

Verify that `v2` is between 100 and 200 for all nonmissing values

```
assert inrange(v2,100,200) if !missing(v2)
```

Verify that `v2` is between 100 and 200 and never missing when `catvar` equals 2 or 3

```
assert inrange(v2,100,200) if (catvar==2 | catvar==3)
```

Verify that there are 5 observations per cluster identified by `cvar`

```
by cvar: assert _N==5
```

As above, but stop checking after the first cluster has fewer than or more than 5 observations

```
by cvar: assert _N==5, fast
```

Syntax

```
assert exp [if] [in] [, rc0 null fast]
```

by is allowed; see [D] **by**.

Options

rc0 forces a return code of 0, even if the assertion is false.

null forces a return code of 8 on null assertions. A null assertion occurs when an if condition excludes all observations from being checked by assert. By default, the return code is 0 for null assertions.

fast forces the command to exit at the first occurrence that *exp* evaluates to false.

Remarks and examples

stata.com

assert verifies that the expression provided is true. It is useful because it tells Stata not only what to do but also what you can expect to find. Groups of assertions are often combined in a do-file to certify data. If the do-file runs all the way through without complaining, every assertion in the file is true. Otherwise, assert will provide a count of the contradictions when an assertion is false. It will also issue an error message along with a return code of 9; see [U] **8 Error messages and return codes**.

assert is seldom used interactively because it is easier to use inspect, summarize, or tabulate to look for evidence of errors in the dataset. These commands, however, require you to review the output to spot the error.

▷ Example 1: Observation-level assertions

You and a colleague are analyzing union membership among women. Your colleague imported data from the National Longitudinal Survey of young women for the years 1968 to 1988. You plan to include the woman's age, total work experience, and whether or not she graduated from college in your model.

Your colleague tells you that the cleaned dataset is called nlswork and that the following things are true: that the variables recording union membership, age, total experience, and education level are not missing for any of the observations; that observations taken before a woman turned 18 have been removed; that total experience is always greater than or equal to 0; and that all college graduates have at least 14 years of education. Before you begin your analysis, you should verify the accuracy of these data. To test that the statements above are true, you create a do-file named check.do:

begin check.do, example 1

```
assert age>=18 & !missing(age)
assert !missing(union)
assert ttl_exp>=0 & !missing(ttl_exp)
assert grade>=14 & !missing(grade) if collgrad==1
```

end check.do, example 1

You save the above file, read in the data, and then issue the do command to check the assertions:

```
. use https://www.stata-press.com/data/r17/nlswork
(National Longitudinal Survey of Young Women, 14-24 years old in 1968)
. do check
```

The output is as follows:

```
. assert age>=18 & !missing(age)
159 contradictions in 28,534 observations
assertion is false
r(9);

end of do-file

r(9);
```

The do-file did not run to completion because it encountered a false assertion—that age is never missing and always at least 18 years.

You should resolve this and any other discrepancies before analyzing the data. You run the do-file again, this time with the `nostop` option, which tells Stata to continue executing the do-file despite any errors.

```
. do check, nostop
```

Once it runs in its entirety, you will have a list of all the data discrepancies to discuss with your colleague. The output is as follows:

```
. assert age>=18 & !missing(age)
159 contradictions in 28,534 observations
assertion is false
r(9);

. assert !missing(union)
9,296 contradictions in 28,534 observations
assertion is false
r(9);

. assert ttl_exp>=0 & !missing(ttl_exp)

. assert grade>=14 & !missing(grade) if collgrad==1
42 contradictions in 4,795 observations
assertion is false
r(9);

.
end of do-file
```

The output from the false assertions above is helpful. First, the number of contradictions can serve as a clue; a few contradictions may suggest data entry errors, whereas a large number may motivate further investigation. Second, you get a straightforward message that the assertion is false. Finally, you get a return code of 9, which makes it easy to write code based on whether or not an assertion is true.

◁

▷ Example 2: Speeding up assert

In example 1, we obtained a count of the number of observations where each assertion was false. However, if all you wanted to know was whether or not an assertion was true, you could reduce the amount of time required to check that assertion by specifying the `fast` option, as shown below:

```
. assert age>=18 & !missing(age), fast
assertion is false
r(9);
```

The `fast` option tells Stata to stop checking the assertion when it encounters the first case where it is false, which is why you do not get a count of the contradictions.

◁

▷ Example 3: Assertions by groups

Your assertions in the previous examples were tested in each observation. You spoke with your colleague regarding those assertions, and she has sent you a revised version of the dataset. The next goal is to make sure that age has been recorded correctly over time. Women in the study were observed once per year, and in some years, they were not observed at all. Therefore, you know that age must be increasing with every time period.

Thus, now you want to assess the characteristics of each woman over time, and you can do so with the by: prefix. You include the sort option with the by prefix because the data have not been sorted by woman (idcode) and year already; see **[U] 11.5 by varlist: construct**. Now you can assert that for each woman, the value of age is greater than it was in the previous year for all years except the first.

You add the following line to check.do:

begin check.do, example 3

```
by idcode (year), sort: assert age>=age[_n-1]+1 if _n>1
```

end check.do, example 3

Upon reissuing the the do check, nostop command, the following output is shown:

```
. by idcode (year), sort: assert age>=age[_n-1]+1 if _n>1
171 contradictions in 23,823 observations
assertion is false
r(9);

.
end of do-file
```

Again, we have found a few errors in the dataset. We might want to check the source of the dataset for any notes on data discrepancies.

◁

❑ Technical note

assert is smart in how it evaluates expressions. When you type something like assert _N==522 or assert work[_N]>0, assert knows that the expression needs to be evaluated only once. When you type assert female==1 | female==0, assert knows that the expression needs to be evaluated once for each observation in the dataset.

Here are some more examples demonstrating assert's intelligence.

```
by female: assert _N==100
```

asserts that there should be 100 observations for every unique value of female. The expression is evaluated once per by-group.

```
by female: assert work[_N]>0
```

asserts that the last observation on work in every by-group should be greater than zero. It is evaluated once per by-group.

```
by female: assert work>0
```

is evaluated once for each observation in the dataset and, in that sense, is formally equivalent to assert work>0.

❑

Reference

Gould, W. W. 2001. Statistical software certification. *Stata Journal* 1: 29–50.

Also see

[D] **assertnested** — Verify variables nested

[P] **capture** — Capture return code

[P] **confirm** — Argument verification

[U] 16 Do-files