Title

stata.com

assert — Verify truth of claim

Description Quick start Syntax Options Remarks and examples
Also see

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.

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

Remarks and examples

stata.com

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. assert 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.

```
. do myassert
. use trans, clear
(xplant data)
. assert sex=="m" | sex=="f"
. assert packs==0 if !smoker
. assert packs>0 if smoker
. sort patient date
. by patient: assert sex==sex[_n-1] if _n>1
. by patient: assert abs(bp-bp[_n-1]) < 20 if bp< . & bp[_n-1]< .
. by patient: assert died==0 if _n!=_N
. by patient: assert died==0 | died==1 if _n==_N
. by patient: assert n_xplant==0 | n_xplant==1 if _n==_N
. assert inval==int(inval)
.
.
end of do-file
```

▷ Example 1

You receive data from Bob, a coworker. He has been working on the dataset for some time, and it has now been delivered to you for analysis. Before analyzing the data, you (smartly) verify that the data are as Bob claims. In Bob's memo, he claims that 1) the dataset reflects the earnings of 522 employees, 2) the earnings are only for full-time employees, 3) the variable female is coded 1 for female and 0 otherwise, and 4) the variable exp contains the number of years, or fraction thereof, on the job. You assemble the following do-file:

```
use frombob, clear
assert _N==522
assert sal>=6000 & sal<=125000
assert female==1 | female==0
```

```
gen work=sum(female==1)
assert work[_N]>0
replace work=sum(female==0)
assert work[_N]>0
drop work
assert exp>=0 & exp<=40
```

Let's go through these assertions one by one. After using the data, you assert that _N equals 522. Remember, _N reflects the total number of observations in the dataset; see **[U] 13.4 System variables (_variables)**. Bob said it was 522, so you check it. Bob's second claim was that the data are for only full-time employees. You know that everybody in your company makes a salary between $6,000 and $125,000, so you check that the salary figures are within this range. Bob's third assertion was that the `female` variable was coded zero or one.

You add something more. You know that your company employs both males and females, so you check that there are some of each. You create a variable called `work` equal to the running sum of female observations and then verify that the last observation of this variable is greater than zero. You then repeat the process for males and discard the `work` variable. Finally, you verify that the `exp` variable is never negative and is never larger than 40.

You save the above file as `check.do`, and here is what happens when you run it:

```
. do check

. use frombob, clear
(5/21 data)

. assert _N==522

. assert sal>6000 & sal<=125000
14 contradictions in 522 observations
assertion is false
r(9);

end of do-file
r(9);
```

Everything went fine until you checked the salary variable, when Stata told you that there were 14 contradictions to your assertion and stopped the do-file. Seeing this, you now interactively `summarize` the `sal` variable and discover that 14 people have missing salaries. You dash off a memo to Bob asking him why these data are missing.

◁

▷ Example 2

Bob responds quickly. There was a mistake in reading the salaries for the consumer relations division. He says it's fixed. You believe him but check with your do-file again. This time you type `run` instead of `do`, suppressing all the output:

```
. run check
. _
```

Even though you suppressed the output, if there had been any contradictions, the messages would have printed. `check.do` ran fine, so all its assertions are true.

◁

❑ Technical note

assert is especially useful when you are processing large amounts of data in a do-file and wish to verify that all is going as expected. The error here may not be in the data but in the do-file itself. For instance, your do-file is rolling along, and it has just merged two datasets that it created by subsetting some other data. If everything has gone right so far, every observation should have merged. Include the line

```
assert _merge==3
```

to verify the correctness of the merge. If all the observations did not merge, the assertion will be false, and your do-file will stop.

As another example, you are combining data from many sources, and you know that after the first two datasets are combined, every individual's sex should be defined. So, you include the line

```
assert sex< .
```

in your do-file. Experienced Stata users include many assertions in their do-files when they process data.

❑

❑ 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.

❑

Also see

[P] **capture** — Capture return code

[P] **confirm** — Argument verification

[U] 16 Do-files