Stata How-to:
Conditions, subsetting data

Patrick Blanchenay

2021-12-27

Contents

1. Comparing wages for men and for women

When working with data, you will often need to apply commands only to a subset of observations contained in your dataset.

Suppose for instance that you have data on wages, and you want to compare wages for men and women in your dataset. You want to: keep observations for men only, compute the average wage among these observations; then keep observations for women only, compute the average wage among these.[1] We will use `CPS_2016.dta` to learn how to do this. This dataset is an excerpt from the Current Population Survey, a widely-used survey that records (among many things) respondents' income (variable `incwage`), education (variable `educ`), and so on.[2]

There are many, many ways to compute the average of a variable. Let's use the **`summarize`** command:[3]

```
// Summarize incwage for the whole dataset
summarize incwage
```

```
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
     incwage |      2,177     29907.7     50947.2          0    900000
```

So in our dataset, the average wage is $29,907.7. Is there a way to compute the average wage but for men only, without including observations for women, and then switch?

[1] We will also learn how to do this using regressions.

[2] In this dataset, we do not distinguish between wages and income.

[3] Check the document "Stata How-To #2: Describe Data" for more details.

2. Applying commands to a subset of observations using `if`

Suppose we want to compute the average wage, but only for men. One way to do this is to remove all women from our dataset, then compute the average as we did in section 1. But in general, researchers do not like erasing data. What if you saved the truncated data by mistake?

By using an **`if`** condition, we can apply commands selectively, only to the observations that satisfy a given condition, without having to erase or alter data. This works with almost any Stata command.

In this dataset, the sex of the respondent is recorded in the `sex` variable. It takes the value 1 for men, and 2 for women. If we want to compute a command only for men, these are observations for which the variable `sex` is equal to 1; we write the command and condition as follows:

```
// Summarize incwage, but only for men
summarize incwage if (sex == 1)
```

The use of parentheses is optional, but a good habit. You will see that now the average wage ($37,269.78) is different from the one computed in section 1: it is now computed only for men. And we did not have to erase observations from our dataset; the number of observations has not changed after performing this command.

> **Warning**
>
> To create an **`if`** condition where a variable is equal to a certain value, we must use a double equal sign **`==`**; for instance in the example above: **`if (sex == 1)`**.

What Stata did internally was to:

1. Check, *observation by observation*, whether the **`if`** condition is satisfied.
2. Keep only observations for which the **`if`** condition is satisfied. In this case, keep only observations for which `sex` is equal to 1, that is: men.
3. Perform the specified command for those observations only; in this case, summarize the `incwage` variable.
4. Return the dataset to its condition before the command was executed.

Step 2 of this procedure is sometimes referred to as *subsetting* since Stata internally keeps a subset of the observations.

2.1. Example: Counting observations

A very common use of the **`if`** condition is to count observations that satisfy a criteria using the command **`count`**. For instance, to count women in your dataset, you could type:

```
count if (sex == 2) // Counting women in the dataset
```

Watch out for missing values when doing so: see section 5, and the warning box in section 8.

2.2. Example: Graphing a subset of the data

The **`if`** condition works with most commands in Stata, including graphing commands.

For instance, if you want to graph the distribution of men's wages in a histogragram:

```
histogram incwage if (sex == 1) // distribution of men's wages
```

3. Placement of the `if` condition

The **`if`** condition must come *before* any options to the command. For instance, when using **`summarize`** with the **`detail`** option, the **`if`** condition comes *before* the option:

```
// Summarize incwage for women
summarize incwage if (sex==2), detail
```

Or to in our histogram example from section 2.2:

```
histogram incwage if (sex == 1), fraction // distribution of men's wages in fraction of total obs.
```

4. More complicated conditions

The **`if`** condition can be much more complicated than just an equality. For instance, you can summarize wages for people earning less than $100,000, by typing:

```
// Summarize incwage for people earning less or equal to 100,000.
summarize incwage if (incwage <= 100000)
```

Conditions can also be combined using the "and" operator & and the "or" operator |. For instance, average wage for men earning less than 100,000:

```
// Summarize incwage for MEN earning less or equal to 100,000.
summarize incwage if (incwage <= 100000) & (sex==1)
```

Table 1: Useful operators in conditions

Operator	Meaning	Remark
&	AND	to link two conditions that should both be true
|	OR	to link two conditions, at least one of which should be true
==	equal	
!=	not equal	
>= , <=	greater or equal, less or equal	
missing(*var*)	missing	checks whether *var* is equal to missing
!missing(*var*)	non-missing	checks whether *var* is any value except missing (`.`)

5. IMPORTANT: In comparisons, missing values are considered infinite

When doing comparisons, missing values are considered infinite. This is a very common source of mistakes in code, which can easily go undetected, so let me repeat:

Warning

In Stata, missing values are considered equal to $+\infty$.

Suppose for instance, you want to know how many individuals in your dataset earn more than $100,000. Our instinct would be to write:

```
count if (incwage >= 100000) // finds 748 observations
```

The command above would naturally count observations where the variable `incwage` takes a value higher than $100,000; but it would also include observations where `incwage` is missing (`.`), because missing (`.`) is considered $+\infty$ which is greater than $100,000. So the command counted observations for which `incwage` is either recorded and greater than $100,000, or unavailable; it would return a count of 748 observations. You would erroneously conclude that 27% of individuals in you dataset earn more than $100,000.

In 99.9% of cases this is not what you intended to do.

To count only observations for which `incwage` is recorded and greater than $100,000, you have to tell Stata to disregard missing values. There are several ways to do this; I recommend the following one (note the exclamation mark `!` to say *non*-missing).

```
//Excluding missing values (138 observations)
count if (incwage >= 100000) & (!missing(incwage))
```

You would more accurately conclude that, *among individuals for which a wage is recorded*, 5.9% earn more than $100,000, a much lower proportion.

6. Using an **`if`** condition with **`generate`** and **`replace`** commands

First, make sure to read "Stata How-To #3: Modify Data" on how to generate variables, or how to modify their values.

You will often have to create a new variable that takes different values based on one or more conditions. For instance, suppose we want to create a variable `evermarried` encoding whether the individual has ever been married. The marital status of respondents is recorded in variable `marst` as follows:

Value	Meaning
1	Married, spouse present
2	Married, spouse absent
3	Separated
4	Divorced
5	Widowed
6	Never married/single

From this, it is clear that `evermarried` should be equal to 1 if `marst` is either 1, 2, 3, 4 or 5; and `evermarried` should be equal to 0 if `marst` is equal to 6. The are many ways to do this, but one way is to use an **`if`** condition when creating the new variable in Stata using **`generate`** and another **`if`** condition with a **`replace`** command:

```
generate evermarried = 1 if (marst <= 5) // creates variable evermarried, and assigns value 1 whenever
    marital status is 5 or less
replace evermarried = 0 if (marst == 6)
label variable evermarried "Ever married" // labelling the newly created var
```

Important: Why do we need two commands (one **`generate`** and one **`replace`**) in the example above? The first command creates the variable `evermarried` equal to 1 but *only for observations where `marst` is less or equal to 5*. What happens to observations for which marst is not less than 5? The variable `evermarried` is left missing

(`.`). Since we want `evermarried` to equal 0 in those cases, we need an additional command **`replace`** which will assign `evermarried` the value 0 for these observations.

Good Practice

When creating or replacing values of a variable you should always check that your code behaved as intended using the data browser or data summary commands! A typo or mistake can easily result in the variable being assigned incorrect values. We can for instance use the **`tabulate`** command:

```
tabulate marst evermarried, missing // check values of evermarried assigned correctly
```

7. Removing observations

On occasions, you may need to perform many commands on the same subset of data. It is cumbersome to type an **`if`** condition after each of those; instead, you can keep only the observations of interest, and then perform all the commands on that restricted dataset. To do this, we can use the **`drop`** and **`keep`** commands with an appropriate **`if`** condition to selectively remove or keep certain observations:

```
/* Keep only observations for individuals aged 50 or over */
drop if (age < 49)
// Keep only women
keep if (sex==2)
```

Warning

The **`drop`** command *permanently* erase observations/variables from memory. The data cannot be recovered without either reloading the dataset and painstakingly recreating all your variables and data manipulation,[a] or using a **`preserve/restore`** combo (see below).

[a] Every time you do, a kitten dies.

Good Practice

If you need to erase some observations, perform commands, then revert to the full dataset, you can use the **`preserve`** and **`restore`** commands:

```
preserve // take a snapshot of the dataset as it currently is (2,787 obs.)
    keep if (sex ==2) // keep only women in the dataset (1,432 obs.)
    summarize incwage
restore // restore the dataset to how it was at the ‘‘preserve’’ (2,787 obs.)
```

8. Advanced: Creating a dummy variable based on a condition

In a lot of analyses, we need to create a dummy variable equal to 1 if some condition is met and 0 if the condition is not met. This is what we did when creating the variable `evermarried` in Section 6. The way we did this was to have first a **`generate`** command for observations where `evermarried` is equal to 1, followed by a **`replace`** for observations where `evermarried` is equal to 0.

There is however a way to do this faster to do this in one command. To do that, we use a trick: when Stata evaluates a condition to be true, it assigns it a value of 1; when the condition is false, it gets a value of 0.

```
// Creates a dummy equal to 1 for individuals ever married, 0 otherwise
 generate evermarried =  (marst <= 5)
```

In the right-hand side of the command above, the brackets contain a condition. For observations where `marst` is equal to 5 or less, the bracket will be evaluated to 1 because the condition is TRUE; for such observations, the variable `evermarried` will be assigned the value 1. For observations where `marst` is strictly more than 5, the condition is evaluated to FALSE; variable `evermarried` will be assigned the value 0.

This only works when creating variable that takes two values, since a condition is either TRUE or FALSE.

Warning

Pay attention to missing values when doing this. (See also section 5.) In the example above, for observations where `marst` is missing, the condition **`(marst <= 5)`** is evaluated to FALSE, and variable `evermarried` will be assigned the value 0. But if `marst` is missing, you do not know whether a person has been married before or not, and you probably should assign the variable `evermarried` a missing value (`.`). One way to do this is to only assign a value to `evermarried` if `marst` is not missing; `evermarried` would be left to missing otherwise:

```
// Creates a dummy equal to 1 for individuals ever married if marital status is recorded
 generate evermarried =  (marst <= 5) if (!missing(marst))
```

9. Advanced: Applying a command to distinct groups of observations using **`bysort`**

If you wanted to have the average wage separately for men and for women, you could do it in two commands:

```
summarize incwage if (sex == 1) // Summarize incwage, but only for men
summarize incwage if (sex == 2) // Summarize incwage, but only for women
```

In this case, `sex` is the variable that defines group, and it takes only two values, so it is pretty easy. If however the grouping variable takes many values, you have to type as many lines of codes as there are different values of the variable, something programmers usually care to avoid. For instance the variable coding marital status, `marst` takes 6 different values. To know the average wage for each marital status, we would have to type 6 different lines of code.

There is a shorter way: we can use the prefix command **`bysort`** followed by the subsetting variable, then colon (:), then the command you wish to run:

```
// Summarize incwage for each marital status
bysort marst: summarize incwage
```

This tells Stata: "For each distinct value of the variable `marst`, find observations that have this value, and for these observations, summarize the variable `incwage`." So Stata will find observations for which `marst` is equal to 1 (Married, spouse present), and compute the average wage among those; then do the same for individuals who are Married, spouse absent; then for those who are Separated, and so on.

NB: If your goal is to compute the average of a variable by groups, there are easier ways to do this.[4]

[4] For instance, using **`table`** or **`collapse`**, see "Stata How-To #2: Describe Data".

10. Practice

1. In this CPS dataset, do men aged 40 or above earn more than men below 40? (Yes, by $16.5*K*.)
2. Create a variable `female` equal to 1 for women, and 0 for men. Are there any missing values?
3. Create a variable `wdivsep` equal to 1 for women who are divorced or women who are separated. `wdivsep` should equal 0 for all other marital statuses. Use **`tabulate`** with the **`missing`** option, to check that:
   - `wdivsep` is equal to 1 for 149 observations;
   - is equal to 0 for 2,638 observations;
   - there are no missing values.
4. How many individuals earn non-zero income? Graph the histogram of their wages.
5. For each level of education `educ`, compute the average respondents' age.