The Stata Journal (2022)
22, Number 4, pp. 998–1003  DOI: 10.1177/1536867X221141068

Stata tip 148: Searching for words within strings

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

1 The problem: Looking for words

Searching for particular text within strings is a common data management problem. One frequent context is whenever various possible answers to a question are bundled together in values of a string variable. Suppose people are asked which sports they enjoy or something more interesting, like which statistical software they use routinely. To keep the matter simple, we will first imagine just lists of one or more numbers that are concise codes for distinct answers, say, `"42"` for `"cricket"` or `"1"` for `"Stata"`. Nonnumeric codes will also be considered in due course. For more on handling such questions, sometimes called multiple response, see Cox and Kohler (2003) or Jann (2005).

The precise problem discussed in this tip is finding text in strings whenever such text is a word in Stata's sense, or something close to that. This needs a little explanation.

Here is a tiny sandbox dataset that will be enough to show the problem and some devices that can yield solutions. By way of example, we will focus mainly on a goal of generating indicator variables, sometimes known as dummy variables. For one overview of generating such variables, see Cox and Schechter (2019). We will also touch on the problem of counting instances of a word.

```
. input str8 mytext
        mytext
  1. "1"
  2. "1 2"
  3. "1 2 11"
  4. "11 12 13"
  5. "11 12 13 111"
  6. end
```

Searching for `"1"` or `"2"`, say, starts with looking for either character with a string function. The function `strpos()` is useful for that. For a rapid personal survey of especially useful functions, see Cox (2011a).

Finding such single characters is easy and unproblematic if the possible answers are one character long at most. More generally, searches are easy if there is no ambiguity. Consider

```
. generate byte is1 = strpos(mytext, "1") > 0
```

The function `strpos()` looks for particular text within other text. It returns 0 if that particular text is not found and a positive number, the position of that particular text, if that text is found. Thus, the position of `"1"` in `"1 2"` is 1, the position of `"2"` in

"1 2" is 3, and so on. Hence, an indicator variable like is1 will be returned as 1 if there are observations in which strpos() returns a positive result. Otherwise, the indicator variable will be returned as 0. If you are new to the idea that an expression like

```
strpos(mytext, "1") > 0
```

returns 1 if true and 0 if false, see Cox and Schechter (2019) or, more directly, Cox (2005, 2016).

If you look again at the sandbox, you should see what is coming next. Looking for "1" with

```
strpos("1 2 11", "1")
```

will still work, fortunately, but looking for "1" with

```
strpos("11 12 13", "1")
```

will yield a false positive. The problem is that we want to find "1" only if it occurs by itself, namely, as a separate word. Stata's primary sense of a word within a string is that words are separated by spaces.

In some Stata contexts, double quotation marks bind together more strongly than spaces separate, so "Stata is subtle" would be treated as a single word if the quotation marks were explicit. For present purposes, we will leave that complication aside.

2 A solution: Looking for spaces too

Let's carry forward the idea that we need to look for spaces too. At first sight, this is a beautiful idea that just does not work very well because there are too many possibilities to catch. Thus, looking for "1 " catches "1"—as part of "1 "—and not "11" within "1 2 11", which is as intended. But it catches the first "1 "—as part of "11 "—within "11 12 13", which is not what we want. Other way round, looking for " 1" catches correctly sometimes and incorrectly other times. Looking for " 1 "—with spaces before and after—will not work if "1" is the first word or the last word, so without a previous space or a following space, respectively.

But that last idea can be made to work with a simple twist. Congratulations if you thought of this directly!

```
. generate byte is1 = strpos(" " + mytext + " ", " 1 ")
. list
```

	mytext	is1
1.	1	1
2.	1 2	1
3.	1 2 11	1
4.	11 12 13	0
5.	11 12 13	0

So we solve the problem of initial and following spaces by supplying them on the fly. Note that we do not need to `generate` a new variable or `replace` an existing variable; we just get Stata to work with a version of the variable with extra spaces. Extra spaces that go beyond our need are harmless, because `" 1 "`, in which "1" has two spaces before it and two after it, is treated the same way as `" 1 "`, in which "1" has one space before it and one after it.

3 What about other separators?

Suppose our string variable used another separator, say, commas, which could just be a different convention or a good idea anyway if spaces occur naturally. Someone's favorite sport might be `"water polo"` or `"debugging code"`. Then whatever the commas separate are not words in Stata's technical sense, but they are still words for us or atoms we wish to seek as such.

We could still use a similar idea of looking for `",1,"` within `"," + mytext + ","`. We just need to watch for gratuitous extra spaces so that `"1 ,"` is not missed. If strings could be moderately complicated, we might need a different method. More positively, if spaces have no meaning and we have values like `"1,2 ,3"`, then changing all commas to spaces allows the method of the previous section to be used.

4 A solution: What would change if we deleted words?

Here is another solution. This time around, an example comes before the explanation.

```
. generate byte IS1 = strlen(mytext) > strlen(subinword(mytext, "1", "", 1))
. list
```

	mytext	is1	IS1
1.	1	1	1
2.	1 2	1	1
3.	1 2 11	1	1
4.	11 12 13	0	0
5.	11 12 13	0	0

We get the same answer, so how did that work?

The function `strlen()` measures the length of strings by counting characters. Although no longer documented, the older name `length()` still works if you remember or prefer that.

The function `subinword()` replaces text with other text if and only if that text occurs as a word in Stata's primary sense. The function knows how to handle words at the beginning and end of strings. However, `subinword()` does not follow Stata's extended sense that a word can be defined (meaning, delimited) by explicit double quotation marks.

But how does replacing text help? We do not want to change text; we are just searching for it. Yet, if the result of replacing text by an empty string (deleting it, to put it plainly) would be to reduce the length of the string, then evidently we did find that text.

Notice "would be". As before, we do not have to `generate` a new variable or `replace` an existing variable. We just get Stata to tell us what the result would be if the text existed and so would be deleted.

Whether the length of the string is greater than the length of the string with the word removed is a true or false question. Either the first length is greater because there is at least one instance of the word or the two lengths are the same because there is no such instance. If the expression is true, 1 is returned; and if it is false, 0 is returned, giving us an indicator variable.

This method is of interest for another reason: you may want to count instances of a word. We could have written

```
. generate byte IS1 = strlen(mytext) > strlen(subinword(mytext, "1", "", .))
```

The difference is in the last argument fed to `subinword()`, namely, system missing `.` rather than 1. That different syntax instructs Stata to delete all instances of the word `"1"` rather than the first only. For detecting whether the word exists, you need know only that it exists at least once.

If the problem is counting instances instead of checking for existence, then the difference in lengths

```
. generate count1 = strlen(mytext) - strlen(subinword(mytext, "1", "", .))
```

is precisely the number of times `"1"` occurs as a word. If you are looking for instances of `"11"` or `"111"`, remember to divide by 2 or 3—the lengths of the words in question, respectively—or you will get the number of characters notionally deleted, not the number of words.

For more on counting substrings, see Cox (2011b).

5 Nonnumeric words

Datasets may include one or more nonnumeric words bundled in a string variable. Suppose there was a survey question about which programming languages are routine for Stata users, with possible answers such as one or more of `Python`, `Julia`, `C++`, and `C`.

Handling such nonnumeric words can be both easier and more difficult than handling numeric words. The possibility of ambiguity is less but still present, as witness checking for mentions of `C` and finding them within mentions of `C++`. Hence, insisting on searching for a word, and not just a substring, can be necessary using one of the devices just explained.

Greater difficulty can arise because of variations in spelling and punctuation, depending sensitively on how such data were entered and collated. Suppose that `none` was expected as an answer when true but that there are also instances of `None`, `NONE`, and so forth. This particular variability is easily handled by looking for `none` within `strlower()` or—according to taste—looking for `NONE` within `strupper()`. The older function names `lower()` and `upper()` are equivalent and still work. Other variations in spelling may be harder to handle, but the first step is always to find out exactly which names were used.

6 A list of tricks

We have covered two main ideas:

- Words are separated by spaces, so look for a word together with previous and following spaces, remembering how to catch words at the beginning or the end of a string (sections 2 and 3).
- If we ask Stata to tell us whether and how the length of a string would change if we were to delete a word, we have ways to detect the occurrence of that word, either yes or no, or the number of occurrences if that is what we seek (section 4).

That is not a complete treatise, even on this small topic. A longer account might mention other possibilities, complications that may arise, or possible solutions.

First, I will mention other problems:

- I have focused on plain ASCII characters, but searching for Unicode needs more care and different functions.
- I have mentioned but not fully solved the complication of "words" that include spaces. But the more complicated the string we are searching for, the less likely ambiguity is to bite.
- I have focused on simple searching of string variables, but string manipulation is needed in other contexts, such as parsing user input if you are writing Stata programs.

Now, I will signal other solutions:

- Many readers will already know about regular expression syntax.
- Sometimes, we cannot solve a problem with one command line. We may need to use the `gettoken` (see [P] **gettoken**) command or the `split` (see [D] **split**) command. We may need to loop over words with a construct like `foreach` or `forvalues` (see [P] **foreach** or [P] **forvalues**).

All of these matters deserve detailed treatment, which is left to other accounts.

7 Acknowledgment

William Lisowski made helpful comments on a draft.

References

Cox, N. J. 2005. FAQ: What is true or false in Stata? http://www.stata.com/support/faqs/data/trueorfalse.html.

———. 2011a. Speaking Stata: Fun and fluency with functions. *Stata Journal* 11: 460–471. https://doi.org/10.1177/1536867X1101100308.

———. 2011b. Stata tip 98: Counting substrings within strings. *Stata Journal* 11: 318–320. https://doi.org/10.1177/1536867X1101100212.

———. 2016. Speaking Stata: Truth, falsity, indication, and negation. *Stata Journal* 16: 229–236. https://doi.org/10.1177/1536867X1601600117.

Cox, N. J., and U. Kohler. 2003. Speaking Stata: On structure and shape: The case of multiple responses. *Stata Journal* 3: 81–99. https://doi.org/10.1177/1536867X0300300106.

Cox, N. J., and C. B. Schechter. 2019. Speaking Stata: How best to generate indicator or dummy variables. *Stata Journal* 19: 246–259. https://doi.org/10.1177/1536867X19830921.

Jann, B. 2005. Tabulation of multiple responses. *Stata Journal* 5: 92–122. https://doi.org/10.1177/1536867X0500500113.