23 Working with strings

Contents

Please read **[U] 12 Data** before reading this entry.

23.1 Description

The word *string* is shorthand for a string of characters. “Male” and “Female”, “yes” and “no”, and “R. Smith” and “P. Jones” are examples of strings. The alternative to strings is numbers—0, 1, 2, 5.7, and so on. Variables containing strings—called *string variables*—occur in data for a variety of reasons. Four of these reasons are listed below.

A variable might contain strings because it is an identifying variable. Employee names in a payroll file, patient names in a hospital file, and city names in a city data file are all examples of this. This is a proper use of string variables.

A variable might contain strings because it records categorical information. “Male” and “Female” and “yes” and “no” are examples of such use, but this is not an appropriate use of string variables. It is not appropriate because the same information could be coded numerically, and, if it were, it would take less memory to store the data and the data would be more useful. We will explain how to convert categorical strings to categorical numbers below.

Also, a variable might contain strings because of a mistake. For example, the variable contains things like 1, 5, 8.2, but because of an error in reading the data, the data were mistakenly put into a string variable. We will explain how to fix such mistakes.

Finally, a variable might contain strings because the data simply could not be coerced into being stored numerically. “15 Jan 1992”, “1/15/92”, and “1A73” are examples of such use. We will explain how to deal with such complexities.

23.2 Categorical string variables

A variable might contain strings because it records categorical information.

Suppose that you have read in a dataset that contains a variable called `sex`, recorded as “male” and “female”, yet when you attempt to run a linear regression, the following message is displayed:

```
. use http://www.stata-press.com/data/r13/hbp2
. regress hbp sex
no observations
r(2000);
```

There are no observations because `regress`, along with most of Stata’s “analytic” commands, cannot deal with string variables. Commands want to see numbers, and when they do not, they treat the variable as if it contained numeric missing values. Despite this limitation, it is possible to obtain tables:

```
. encode sex, gen(gender)
. regress hbp gender
      Source |       SS       df       MS              Number of obs =    1128
-------------+------------------------------           F(  1,  1126) =   14.04
       Model |  .644485682     1  .644485682           Prob > F      =  0.0002
    Residual |  51.6737767  1126  .045891454           R-squared     =  0.0123
-------------+------------------------------           Adj R-squared =  0.0114
       Total |  52.3182624  1127  .046422593           Root MSE      =  .21422

------------------------------------------------------------------------------
         hbp |      Coef.   Std. Err.      t    P>|t|     [95% Conf. Interval]
-------------+----------------------------------------------------------------
      gender |   .0491501   .0131155     3.75   0.000     .0234166    .0748837
       _cons |  -.0306744   .0221353    -1.39   0.166    -.0741054    .0127566
------------------------------------------------------------------------------
```

The magic here is to convert the string variable `sex` into a numeric variable called `gender` with an associated value label, a trick accomplished by `encode`; see [U] **12.6.3 Value labels** and [D] **encode**.

23.3 Mistaken string variables

A variable might contain strings because of a mistake.

Suppose that you have numeric data in a variable called `x`, but because of a mistake, `x` was made a string variable when you read the data. When you `list` the variable, it looks fine:

```
. list x

     +-----+
     |   x |
     |-----|
  1. |   2 |
  2. | 2.5 |
  3. |  17 |
    (output omitted)
     +-----+
```

Yet, when you attempt to obtain summary statistics on `x`,

```
. summarize x
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
           x |         0
```

If this happens to you, type `describe` to confirm that `x` is stored as a string:

```
. describe
Contains data
  obs:            10
 vars:             3
 size:           160
--------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
--------------------------------------------------------------------------------
x               str4   %9s
y               float  %9.0g
z               float  %9.0g
--------------------------------------------------------------------------------
Sorted by:
```

`x` is stored as a `str4`.

The problem is that summarize does not know how to calculate the mean of string variables—how to calculate the mean of “Joe” plus “Bill” plus “Roger”—even when the string variable contains what could be numbers. By using the destring command ([D] **destring**), the variable mistakenly stored as a str4 can be converted to a numeric variable.

```
. destring x, replace
. summarize x
    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
        newx |        10        1.76    .8071899         .7          3
```

An alternative to using the destring command is to use generate with the real() function; see [D] **functions**.

23.4 Complex strings

A variable might contain strings because the data simply could not be coerced into being stored numerically.

A complex string is a string that contains more than one piece of information. Complex strings may be very long and may contain binary information. Stata can store strings up to 2-billion characters long and can store strings containing binary information, including binary 0 (\0). You can read more about this in [U] **12.4 Strings**. The most common example of a complex string, however, is a date: “15 Jan 1992” contains three pieces of information—a day, a month, and a year. If your complex strings are dates or times, see [U] **24 Working with dates and times**.

Although Stata has functions for dealing with dates, you will have to deal with other complex strings yourself. Assume that you have data that include part numbers:

```
. list partno

     +--------+
     | partno |
     |--------|
  1. | 5A2713 |
  2. | 2B1311 |
  3. | 8D2712 |
   (output omitted)
     +--------+
```

The first digit of the part number is a division number, and the character that follows identifies the plant at which the part was manufactured. The next three digits represent the major part number and the last digit is a modifier indicating the color. This complex variable can be decomposed using the substr() and real() functions described in [D] **functions**:

```
. gen byte div = real(substr(partno,1,1))
. gen str1 plant = substr(partno,2,1)
. gen int part = real(substr(partno,3,3))
. gen byte color = real(substr(partno,6,1))
```

We use the substr() function to extract pieces of the string and use the real() function, when appropriate, to translate the piece into a number.

For an extended discussion of numeric and string data types and how to convert from one kind to another, see Cox (2002).

23.5 Reference

Cox, N. J. 2002. Speaking Stata: On numbers and strings. *Stata Journal* 2: 314–329.