CODERS' CORNER

CSAE
CENTRE FOR THE STUDY OF AFRICAN ECONOMIES

DATA CLEANING ROUTINE FOR STRING VARIABLES

Many raw data sets – survey as well as administrative data – contain string variables that need to be cleaned before they can be processed and analysed. Stata has a lot of functions that greatly facilitate working with string variables.

Here is a collection of my favourite lines of code to clean string variables:

Let's imagine we have a data set from a firm survey where firms report the destination country for their exports. The variable that captures this information is called "destination". As a first step we tabulate the values of the variable and we notice that there are six distinct values for Burkina Faso alone.

"Burkina Faso"	=> two spaces between "Burkina" and "Faso"
" Burkina Faso"	=> First letters capitalized and a leading space
"BURKINA FASO"	=> All letters capitalised
"burkina faso "	=> All letters are lower case and a trailing space at the end
"Burkina/Faso"	=> Special characters
"Burkina"	=> The value was truncated after the first word

1. Removing spaces

Leading, trailing, and internal spaces can easily be removed using the *strtrim()* and *stritrim()* function. The two can even be combined in one line:

```
replace destination = strtrim(stritrim(destination))
```

Spaces are often hard to detect when browsing data so this has become my default first line of code whenever I start working with a string variable.

2. Harmonising capitalisation

Harmonising capitalisation is just as easy. The three main options are:

- *upper()*, which capitalises all letters => "BURKINA FASO"
- *lower()*, which converts all letters to lower case => "burkina faso"
- *proper()*, which capitalizes the first letter of a word => "Burkina Faso"

You can even combine them with the functions for removing spaces in one line:

```
replace destination = proper(strtrim(stritrim(destination)))
```

3. Removing (or altering) special characters

By tabbing or browsing the data, we might not be able to immediately detect all special characters that are hidden somewhere in the data. The user written command *charlist* conveniently lists all special characters that occur in any of the values the string variable takes (ssc install *charlist*):

```
charlist destination                    // lists all characters
di `r(ascii)’                           // lists the ascii code for each character
```

Stata might not be able to correctly read all those characters and some might thus appear as boxes in the list provided by *charlist*. We therefore add another line to list the characters as ascii codes. This will come in handy in the next step. Beware that *charlist* can take a few minutes to run when working with large data as it searches all values of a string variable.

We can now remove or convert all those characters using the *subinstr()* function.

```
replace destination = subinstr(destination, “/” , “ “ ,  .)      // replaces the / with a space
replace destination = subinstr(destination, “&” , ““ ,  .)       // simply removes an & sign
replace destination = subinstr(destination, “;” ,  “,“ ,  .)     // replaces a ; with a ,
```

As you might have noticed we first list the variable name, then the character we want to replace, and finally the character it is replaced with. We can even specify how many of the characters we want to replace, e.g. if it is only the first “&” of a value, we replace the . after the final “,” with a 1:

```
replace destination = subinstr(destination, “&” , ““ ,  1)      // removes first & only
```

If we can’t find a certain character on our keyboard or Stata is not able to print it correctly, we can also use the corresponding ascii code – a popular character encoding standard. To give an example, the ascii code for “&” is 38. You can try this by typing

```
display char(38)
```

in the command window.

Charlist automatically stores the ascii code of all characters that occur in any value taken by our string variable and we retrieved it using r(ascii).

```
replace destination = subinstr(destination, char(38) , ““ ,  .)                // removes all & signs
replace destination = subinstr(destination, char(32) , char(44) ,  .)      // replaces all spaces
with commas
```

The ascii code for a space is 32 and the code for a comma is 44.

4. Streamlining values

If values got truncated or there are different spellings for the same country name, the function *strops()* can sometimes provide a quick fix. What *strops()* usually does is to detect the first occurrence of a sequence of characters in a string variable. If it takes a value greater than zero this means the sequence of characters we searched for is contained in the value – no matter the exact position. Let’s say we want to harmonize all occurrences of “Burkina Faso” and also capture those cases where the “Faso” has been truncated when the data was processed. We use the line

```
replace destination = “Burkina Faso” if strpos(destination, “Burkina”)>0
```

Everything this line does is to replace all values that contain “Burkina” in some form with “Burkina Faso”. Note that *strpos()* is case sensitive and we thus always want to run it after having harmonized capitalisation. To give another example, some firms might have abbreviated their entries for the Democratic Republic of the Congo. Instead of iterating through all possibilities (e.g. Dem. Rep. Congo or Democratic Republic of Congo), we simply look for entries that contain both the sequence of characters “Congo” and “Dem”.

replace *destination* = “Democratic Republic of the Congo” if strpos(*destination,* “Congo”)>0 & strpos(*destination,* “Dem”)>0

5. The last few lines of code

At the end of the data cleaning routine we might want to re-run the first line of code for removing spaces. This ensures no new excess spaces have been introduced in the process, e.g. when removing special characters.

I have mentioned this in a previous post on working with large data sets, but wanted to highlight this here as well: the *compress* command. The *compress* command is particularly useful when working with string variables as it can greatly reduce the amount of memory used by your data. It converts the storage type of all your variable to the smallest type possible without losing any information AND it conveniently coalesces values of string variables that are stored as strL (string variables with an arbitrary length). Both steps can make a big difference. Make sure to run it just after cleaning your

Verena Wiedemann, DPhil candidate in Economics, St Antony’s College

08 December 2020