Dealing with the cryptic survey: Processing labels and value labels with Mata

Alfonso Miranda

Institute of Education, University of London
(A.Miranda@ioe.ac.uk)

Data Management

- Research is done on the basis of complex survey data
- Putting together data in a format that is ready for analysis **is often a non trivial exercise**
- Researchers put lots of effort to solve their Data Administration problems and **often take the wrong decisions** and end up analysing **badly build data**
- This may lead to extrange results and significant bias
- However, most people would say that cleaning and preparing data is a boring, mostly mechanical, and undeserving activity

The problem

- Survey data comes often as a plain table containing cryptic variable names, numbers, and letters
- To make sense of the data, the researcher is given a questionnaire or a code book that contains a list of variable names, their description, and an interpretation of the values (either a number or a string) that each variable can take
- Code books are commonly provided as plain text or in PDF format. Hence, the researcher is left “free” to type labels and value labels one by one

Bad research habits. . .

There are two things you are better off not watching in the making: sausages and econometric estimates

Edward Leamer

Bad research habits. . .

- **Cutting and processing the piece of the survey that is needed in the short-run** and leave the rest for future processing
  - Never fully understand how the survey is structured
  - Reduce sample size more than strictly needed
  - Create false missing values and/or item non-response
  - Do not take into account sample design
  - Introduce potential selection bias
- **This leads to the creation of various versions of the same data**
  - Inability to track changes
  - Cannot reproduce research results

This talk. . .

- Here I discuss only one relatively small aspect that arise when preparing data for analysis
- Namely, I will show how to recover the information that is contained in questionnaires or code books that are in PDF format (not copy protected) and how to process this information in a clean, fast, and efficient way with Mata

The Agenda

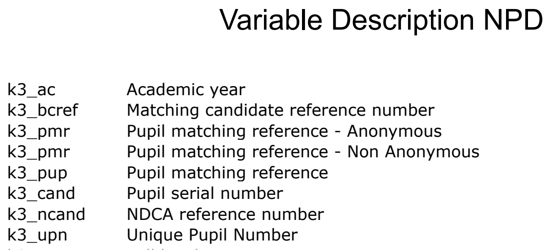
We have two pieces of information:

- Data in Stata format with variable names but no description (i.e., no variable labels)

```
--------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
--------------------------------------------------------------------------------
k3_ac           str9   %9s
k3_pmr          str18  %18s
k3_dob          str19  %19s
k3_age          byte   %8.0g
k3_mth          byte   %8.0g
k3_schid        long   %12.0g
k3_land         str1   %9s
k3_lang         str1   %9s
k3_ma           str1   %9s
k3_sc           str1   %9s
k3_engta        str1   %9s
```

- A list of variable names and their description in a PDF file

The Agenda



Variable Description NPD

k3_ac	Academic year
k3_bcref	Matching candidate reference number
k3_pmr	Pupil matching reference - Anonymous
k3_pmr	Pupil matching reference - Non Anonymous
k3_pup	Pupil matching reference
k3_cand	Pupil serial number
k3_ncand	NDCA reference number
k3_upn	Unique Pupil Number

AIM: To create variable labels using the information contained in the PDF

Current Stata capabilities to deal with variable labels

- Can use Stata's official `label` command

```
label variable varname ["label"]
```

For instance, we could type:

```
. label k3_ac ‘‘Academic year’’
. label k3_bcref ‘‘Matching candidate reference number’’
```

- But that will require to type one label at a time... Not very efficient
- It would be nice if one could write a program that takes two large strings, one containing variable names and the other containing all variable descriptors, and process all variable labels at the strike of a single return

The general idea

I seek to write a program that will be invoked as follows:

```
#delimit   ;
local varnames "k3_ac #  k3_bcref #  k3_pmr #  k3_pmr #  k3_pup ";

local vardes "Academic year  # Matching candidate reference number
# Pupil  matching reference - Anonymous
# Pupil matching reference - Non Anonymous  # Pupil matching reference";
#delimit cr

mata: Labelvar("varnames","vardes")
```

And will to exploit the ability, which I assume I have, of copying the data from the PDF document as plain text into a text editor (your favourite) and from the text editor into a spreadsheet (your favourite)

Live demostration

Time for a live demonstration. Hope everything goes well. . .

Live demostration

- Now, in the rest of the talk I will give details on the programming of `Labelvar` in Mata.
- So, those who are not that interested in the technical details please bear with me. . .

Mata: An overview

Mata is a full-fledged matrix programming language. Mata can be used interactively or called from Stata and a large number of functions (matrix, scalar, mathematical, statistical, equation solvers, optimiser) are provided. Mata can access Stata's variables and can work with virtual matrices (views) of the data in memory. Mata code is automatically compiled into byte-code and runs significantly faster than Stata

Mata can do strings...

Mata handles matrices that contain either numeric or string elements, though a single matrix may not mix strings and numbers. Here are some examples:

```
 . mata
:
: A = (1,2 \ 3,4)

: A
        1   2
    +---------+
  1 |  1   2  |
  2 |  3   4  |
    +---------+

: B = ("This","That" \ "These","Those")

: B
            1       2
    +-----------------+
  1 |    This    That  |
  2 |   These   Those  |
    +-----------------+
: end
```

Mata can do strings. . .

The sum of two string matrices is defined as:

```
: B = ("This","That" \ "These","Those")

: C = ("Hola","Si" \ "NO","QUE")

          1      2
   +---------------+
 1 |  Hola     Si  |
 2 |    NO    QUE  |
   +---------------+

: D = B + C

: D
              1          2
   +-----------------------+
 1 |  ThisHola     ThatSi  |
 2 |   TheseNO   ThoseQUE  |
   +-----------------------+
```

Here I used an **assignment** operator (the equals sign $=$ in the code) to define a new matrix D. Notice the sum operator was performed using the conformability rule that the usual numeric sum operator will require

Mata can do strings...

To summarize,

- In Mata "This" + "Hola" returns "ThisHola"
- This definition of the sum operator for strings may not sound that intuitive... But the operator does make sense given that product operator is not defined for strings
- So, "This" * "Hola" produces an error message
- Usual conformability of the sum operator applies

Hence, the idea is to exploit these capabilities of Mata and its ability to communicate with Stata to solve our labels problem

The code I

The code is written in a text editor into a do file `Labelvar.mata`, which will be compiled once it is ready

The first thing we need to do is call Mata and define the function we are programming

```
 mata:
mata clear
void function Labelvar(string scalar listvar, string scalar listdes)
{
```

The `void` says Mata that the function returns nothing. There are two arguments, one named `listvar` and the other named `listdes`. Both arguments are scalars (i.e., a matrix with a single cell) that contain a string value

```
/* Parsing relevant strings */

t = tokeninit("", "#", (‘""""’, ‘"‘""’"’), 0, 0)
```

`Tokeninit()` defines advanced parsing. First argument defines the character that will be treated as white space. Second argument defines the character that will define where a word begins and where it ends, here # (this is what we are after for parsing our label names and descriptors.) Remaining options control the way qoute characters behave and how large numeric values are displayed. Here we do not allow numbers and so the zeroes

The code II

Next `tokenset()` will be used to specify that our newly defined advanced parsing `t` will be used for processing the contents of the Stata locals `listvar` and `listdes`

```
 tokenset(t, st_local(listvar))
listvarT = tokengetall(t)
tokenset(t, st_local(listdes))
descriptorT = tokengetall(t)
```

Function `tokengetall()` will put all the elements of local `listvar` in the cells of a row vector, including the parsing character #

```
 /* get variables  */

for (i=1;i<=cols(listvarT);i++) {
 if (i==1) variables = strtrim(listvarT[i])
 if (i>1 & listvarT[i]!="#") variables = (variables,strtrim(listvarT[i]))
}
```

The lines above loop over the columns of `listvar` to define a new matrix `variables` that contains only the name of our variables, getting rid of the parsing character that were still present in matrix `listvar`. We do the same with the variable descriptors

```
 /* get descriptors */

for (i=1;i<=cols(descriptorT);i++) {
 if (i==1) descriptor = strtrim(descriptorT[i])
 if (i>1 & descriptorT[i]!="#") descriptor = (descriptor,strtrim(descriptorT[i]))
}
```

The code III

And this is a trick to make the quotation symbols be part of the strings that are deposited in `descriptorT`:

```
comma = ‘"""’
for (i=1;i<=cols(descriptor);i++) {
 descriptor[i] = comma+descriptor[i]+comma
}
```

So, for instance, if we were to apply the same thick to matrix C we will get something like this:

```
          1       2
    +-------------------+
  1 |  "Hola"     "Si"  |
  2 |    "NO"    "QUE"  |
    +-------------------+
```

Now, matrix `variables` contains the variable names and matrix `descriptors` contains the variable descriptors, with the quotation marks " " being part of the descriptions. We are almost done. . . Now we only need to manipulate these matrices to create our labels

The code IV

Next, we use the function `Stata()` to interact with Stata. Loop over the elements of matrix `variables` and summarise variable by variable, keeping record in scalar `_rc` if the variable we are working with was found in data — in that case `_rc` will equal zero. Then I bring the result of this operation into Mata using the *st_numscalar*() function

```
/* Create labels definitions in Stata */

for (i=1;i<=cols(variables);i++) {
 stata("capture su" + " " + variables[i])
 stata("scalar inlist=_rc")
 inlist=st_numscalar("inlist")
 if (inlist==0) {
  stata("label var" +" "+ variables[i]+" "+  descriptor[i])
 }
}
```

Finally, if the variable is found on current data, we use `Stata()` to interact with Stata and create the needed variable labels. Notice how **the definition of the sum operator in Mata is used** to build up, in each iteration, a string that contains the information in the relevant cell of `variables` and `descriptor`, and adds a set of "fixed" strings — one of which is an empty space. The resulting string will make sense as a command once it is issued to the Stata prompt

Last one on programming, I promise. . .

Now, just need to close the initial curly bracket and save the compiled file into a `mo-file`:

```
}
mata mosave Labelvar(), dir(PERSONAL) replace
mata clear
end
```

Ok, the do-file with the source code is ready. The only thing we still must do is to run`Labelvar.do`to compile the code. Now the new mata function `Labelvar()` will be available for use.

- Very similar code will deal with the problem of defining label values. The code is written in the appendix
- This code is also available at the ssc:

  ```
  . ssc install labelutil
  ```
- Many thanks!
- The End

Labels_v2() Function

```
 mata:
mata clear
void function Labels_v2(string scalar labelsS, string scalar valuesS,
string scalar lname, string scalar vtype)
{

 /* declarations */

 string matrix labels, values
 string scalar comma

 /* Parsing relevant strings */

 t = tokeninit("", "#", (‘""""’, ‘"‘""’"’), 0, 0)
 tokenset(t, st_local(labelsS))
 labelsT = tokengetall(t)
 tokenset(t, st_local(valuesS))
 valuesT = tokengetall(t)

 /* get labels */

 labels = J(1,1,"")
 for (i=1;i<=cols(labelsT);i++) {
  if (i==2) labels = strtrim(labelsT[i])
  if (i>2 & labelsT[i]!="#") labels = (labels,strtrim(labelsT[i]))
 }
 comma = ‘"""’
 for (i=1;i<=cols(labels);i++) {
  labels[i] = comma+labels[i]+comma
 }
```

Labels_ Function II

```
 /* get values */

valuesR = J(1,1,"")
for (i=1;i<=cols(valuesT);i++) {
 if (i==2) valuesR = strtrim(valuesT[i])
 if (i>2 & valuesT[i]!="#") valuesR = (valuesR,strtrim(valuesT[i]))
}
values = strtoreal(valuesR)
for (i=1;i<=cols(valuesR);i++) {
if (values[i]==.) values[i] = J(1,1,8800)+J(1,1,i)
}
for (i=1;i<=cols(valuesR);i++) {
 valuesR[i] = comma+valuesR[i]+comma
}

/* Create a verctor with new values as strings */

valuesNS = strofreal(values)
 for (i=1;i<=cols(valuesNS);i++) {
 valuesNS[i] = comma+valuesNS[i]+comma
}

/* Replace values in data */

if (vtype=="s") {
 /* trim string values in data */
 stata("qui replace "+lname+" = "+"rtrim("+lname+")")
 /* deal with blank records */
 stata("qui replace "+lname+" = "+comma+"9985"+comma+" if "+lname+"=="+comma+comma)
 stata("label def "+" "+lname+" "+"9985"+" "+comma+"Blank in data"+comma+", add")
```

Labels_ Function III

```
   /* replace new values in data */
  for (i=1;i<=cols(valuesR);i++) {
   stata("qui replace"+" "+lname+"="+valuesNS[i]+" if "+" "+lname+"=="+valuesR[i])
  }
  stata("qui destring "+lname+ ", replace")
 }

 /* reverse substitution --- variable is writen in data as label string description */

if (vtype=="rev") {
  /* trim string values in data */
  stata("qui replace "+lname+" = "+"rtrim("+lname+")")
  /* deal with blank records */
  stata("qui replace "+lname+" = "+comma+"9985"+comma+" if "+lname+"=="+comma+comma)
  stata("label def "+" "+lname+" "+"9985"+" "+comma+"Blank in data"+comma+", add")
  /* replace new values in data */
  for (i=1;i<=cols(valuesR);i++) {
   stata("qui replace"+" "+lname+"="+valuesNS[i]+" if "+" "+lname+"=="+labels[i])
  }
  stata("qui destring "+lname+ ", replace")
 }

 /* Create labels definitions in Stata */

 for (i=1;i<=cols(labels);i++) {
  stata("label def" +" "+lname+" "+strofreal(values[i])+" "+ labels[i]+", add")
 }
```

Labels_v2() Function IV

```
    /* label values */
  stata("label val "+lname+" "+lname)
}
mata mosave Labels(), dir(PERSONAL) replace
mata clear
end
```

- NB. `Labels_v2()` will code all blank records as 9985. This can changed as needed/preferred