The Stata Journal (2010)
10, Number 3, pp. 458–481

Translation from narrative text to standard codes variables with Stata

Federico Belotti
University of Rome "Tor Vergata"
Rome, Italy
federico.belotti@uniroma2.it

Domenico Depalo
Bank of Italy
Rome, Italy
domenico.depalo@bancaditalia.it

Abstract. In this article, we describe `screening`, a new Stata command for data management that can be used to examine the content of complex narrative-text variables to identify one or more user-defined keywords. The command is useful when dealing with string data contaminated with abbreviations, typos, or mistakes. A rich set of options allows a direct translation from the original narrative string to a user-defined standard coding scheme. Moreover, `screening` is flexible enough to facilitate the merging of information from different sources and to extract or reorganize the content of string variables.

Editors' note. This article refers to undocumented functions of Mata, meaning that there are no corresponding manual entries. Documentation for these functions is available only as help files; see `help regex`.

Keywords: dm0050, screening, keyword matching, narrative-text variables, standard coding schemes

1 Introduction

Many researchers in varied fields frequently deal with data collected as narrative text, which are almost useless unless treated. For example,

- Electronic patient records (EPRs) are useful for decision making and clinical research only if patient data that are currently documented as narrative text are coded in standard form (Moorman et al. 1994).
- When different sources of data use different spellings to identify the same unit of interest, the information can be exploited only if codes are made uniform (Raciborski 2008).
- Because of verbatim responses to open-ended questions, survey data items must be converted into nominal categories with a fixed coding frame to be useful for applied research.

These are only three of the many critical examples that motivate an ad hoc command.

Recoding a narrative-text variable into a user-defined standard coding scheme is currently possible in Stata by combining standard data-management commands (for example, `generate` and `replace`) with regular expression functions (for example, `regexm()`).

However, many problems do not yield easily to this approach, especially problems containing complex narrative-text data. Consider, for example, the case when many source variables can be used to identify a set of keywords; or the case when, looking at different keywords, one is within a given source variable but not necessarily at the beginning of that variable, whereas the others are at the beginning, the end, or within that or other source variables. Because no command jointly handles all possible cases, these cases can be treated with existing Stata commands only after long and tedious programming, increasing the possibility of introducing errors. We developed the `screening` command to fill this gap, simplifying data-cleaning operations while being flexible enough to cover a wide range of situations.

In particular, `screening` checks the content of one or more string variables (sources) to identify one or more user-defined regular expressions (keywords). Because string variables are not flexible, to make the command easier and more useful, a set of options reduces your preparatory burden. You can make the matching task wholly case insensitive or set matching rules aimed at matching keywords at the beginning, the end, or within one or more sources. If source variables contain periods, commas, dashes, double blanks, ampersands, parentheses, etc., it is possible to perform the matching by removing such undesirable content. Moreover, if the matching task becomes more difficult because of abbreviations or even pure mistakes, `screening` allows you to specify the number of letters to screen in a keyword. Finally, the command allows a direct translation of the original string variables in a user-defined standard coding scheme.

All these features make the command simple, extremely flexible, and fast, minimizing the possibility of introducing errors. It is worth emphasizing that we find Mata more convenient to use than Stata, with advantages in terms of time execution.

The article is organized as follows. In section 2, we describe the new `screening` command, and we provide some useful tips in section 3. Section 4 illustrates the main features of the command using EPR data, while section 5 details some critical cases in which the use of `screening` may aid your decision to merge data from different sources or to extract and reorder messy data. In the last section, section 6, we offer a short summary.

2 The screening command

String variables are useful in many practical circumstances. A drawback is that they are not so flexible: for example, in EPR data, coding `CHOLESTEROL` is different from coding `CHOLESTEROL LDL`, although the broad pathology is the same. Stata and Mata offer many built-in functions to handle strings. In particular, `screening` extensively uses the Mata regular-expression functions `regexm()`, `regexr()`, and `regexs()`.

(*Continued on next page*)

2.1 Syntax

`screening` [*if*] [*in*], `sources(`*varlist*[, *sourcesopts*]`)` `keys(`[*matching_rule*] `"`*string*`"` [...]`)` [`letters(`*#*`)` `explore(`*type*`)` `cases(`*newvar*`)` `newcode(`*newvar* [, *newcodeopts*]`)` `recode(`*recoding_rule* `"`*user_defined_code*`"` [*recoding_rule* `"`*user_defined_code*`"` ...]`)` `checksources` `tabcheck` `memcheck` `nowarnings` `save` `time`]

2.2 Options

`sources(`*varlist*[, *sourcesopts*]`)` specifies one or more *string* source variables to be screened. `sources()` is required.

sourcesopts	description
`lower`	perform a case-insensitive match (lowercase)
`upper`	perform a case-insensitive match (uppercase)
`trim`	match keywords by removing leading and trailing blanks from sources
`itrim`	match keywords by collapsing sources with consecutive internal blanks to one blank
`removeblank`	match keywords by removing from sources all blanks
`removesign`	match keywords by removing from sources the following signs: `* + ? / \ % ( ) [ ] { } | . ^ - _ # $`

`keys(`[*matching_rule*] `"`*string*`"` ...`)` specifies one or more regular expressions (keywords) to be matched with source variables. `keys()` is required.

matching_rule	description
`begin`	match keywords at beginning of string
`end`	match keywords at end of string

`letters(`*#*`)` specifies the number of letters to be matched in a keyword. The number of letters can play a critical role: specifying a high number of letters may cause the number of matched observations to be artificially low because of mistakes or abbreviations in the source variables; on the other hand, matching a small number of letters may cause the number of matched observations to be artificially high because of the inclusion of uninteresting cases containing the "too short" keyword. The default is to match keywords as a whole.

`explore(`*type*`)` allows you to explore `screening` results.

type	description
`tab`	tabulate all matched cases for each keyword within each source variable
`count`	display a table of frequency counts of all matched cases for each keyword within each source variable

`cases(`*newvar*`)` generates a set of categorical variables (as many as the number of keywords) showing the number of occurrences of each keyword within all specified source variables.

`newcode(`*newvar*[`,` *newcodeopts*]`)` generates a new (numeric) variable that contains the position of the keywords or the regular expressions in `keys()`. The coding process is driven by the order of keywords or regular expressions.

newcodeopts	description
`replace`	replace *newvar* if it already exists
`add`	obtain *newvar* as a concatenation of subexpressions returned by `regexs(`*n*`)`, which must be specified as a *user_defined_code* in `recode`
`label`	attach keywords as value labels to *newvar*
`numeric`	convert *newvar* from string to numeric; it can be specified only if the `recode()` option is specified

`recode(`*recoding_rule* `"`*user_defined_code*`"` [*recoding_rule* `"`*user_defined_code*`"` `...`]`)` recodes the `newcode()` *newvar* according to a user-defined coding scheme. `recode()` must contain at least one *recoding_rule* followed by one *user_defined_code*. When you specify `recode(1 "`*user_defined_code*`")`, the `"`*user_defined_code*`"` will be used to recode all matched cases from the first keyword within the list specified via the `keys()` option. If `recode(2,3 "`*user_defined_code*`")` is specified, the `"`*user_defined_code*`"` will be used to recode all cases for which second and third keywords are simultaneously matched, and so on. This option can only be specified if the `newcode()` option is specified.

`checksources` checks whether source variables contain special characters. If a matching rule is specified (`begin` or `end` via `keys()`), `checksources` checks the sources' boundaries accordingly.

`tabcheck` tabulates all cases from `checksources`. If there are too many cases, the option does not produce a table.

`memcheck` performs a "preventive" memory check. When `memcheck` is specified, the command will exit promptly if the allocated memory is insufficient to run `screening`. When memory is insufficient and `screening` is run without `memcheck`, the command could run for several minutes or even hours before producing the message `no room to add more variables`.

`nowarnings` suppresses all warning messages.

`save` saves in `r()` the number of cases detected, matching each source with each keyword.

`time` reports elapsed time for execution (seconds).

3 Tips

The low flexibility of string variables is a reason for concern. In this section, we provide some tips to enhance the usefulness of `screening`. Some tips are useful to execute the command, while other tips are useful to check the results.

Most importantly, capitalization matters: this means that screening for `KEYWORD` is different from screening for `keyword`. If source variables contain `HEMINGWAY` and you are searching for `Hemingway`, screening will not identify such keyword. If suboption `upper` (`lower`) is specified in `sources()`, keywords will be automatically matched in uppercase (lowercase).

Choose an appropriate *matching_rule*. The `screening` default is to match keywords over the entire content of source variables. By specifying the *matching_rule* `begin` or `end` within the `keys()` option, you may switch accordingly the matching rule on string boundaries. For example, if sources contain `HEMINGWAY ERNEST` and `ERNEST HEMINGWAY` and you are searching `begin HEMINGWAY`, the `screening` command will identify the keyword only in the former case. Whether the two cases are equivalent must be evaluated case by case.

Another issue is how to choose the optimal number of letters to be screened. For example, with EPR data, different physicians might use different abbreviations for the same pathologies. And so talking about a "right" number of letters is nonsense. As a rule of thumb, the number of letters should be specified as the minimum number that uniquely identifies the case of interest. Using many letters can be too exclusive, while using few letters can be too inclusive. In all cases, but in particular when the appropriate number of letters is unknown, we find it useful to tabulate all matched cases through the `explore(tab)` option. Because it tabulates all possible matches between all keywords and all source variables, it is the fastest way to explore the data and choose the best matching strategy (in terms of keywords, matching rule, and letters).

Advanced users can maximize the potentiality of `screening` by mixing keywords with Stata regular-expression operators. Mixing in operators allows you to match more-complex patterns, as we show later in the article.[1] For more details on regular-expression syntaxes and operators, see the official documentation at http://www.stata.com/support/faqs/data/regex.html.

1. The `letters()` option does not work if a keyword contains regular-expression operators.

screening displays several messages to inform you about the effects of the specified options. For example, consider the case in which you are searching some keywords containing regular-expression operators. screening will display a message with the correct syntax to search a keyword containing regular-expression operators. The nowarnings option allows you to suppress all warning messages.

screening generates several temporary variables (proportional to the number of keywords you are looking for and to the number of sources you are looking from). So when you are working with a big dataset and your computer is limited in terms of RAM, it might be a good idea to perform a "preventive" memory check. When the memcheck option is specified and the allocated memory is insufficient, screening will exit promptly rather than running for several minutes or even hours before producing the message no room to add more variables.

We conclude this section with an evaluation of the command in terms of time execution using different Stata flavors and different operating systems. In particular, we compare the latest version of screening written using Mata regular-expression functions with its beta version written entirely using the Stata counterpart. We built three datasets of 500,000 (A), 5 million (B), and 50 million (C) observations with an ad hoc source variable containing 10 different words: HEMINGWAY, FITZGERALD, DOSTOEVSKIJ, TOLSTOJ, SAINT-EXUPERY, HUGO, CERVANTES, BUKOWSKI, DUMAS, and DESSI. Screening for HEMINGWAY (50% of total cases) gives the following results (in seconds):

Stata flavor and operating system	Mata A	Mata B	Mata C	Stata A	Stata B	Stata C
Stata/SE 10 (32-bit) and Mac OS X 10.5.8 (64-bit)*	0.66	6.67	na	0.93	9.24	na
Stata/MP 11 (64-bit) and Mac OS X 10.5.8 (64-bit)*	0.60	5.66	na	0.85	7.73	na
Stata/MP 11 (64-bit) and Window Server 2003 (64-bit)+	0.37	3.70	37.22	0.70	7.06	70.59

* Intel Core 2 Duo 2.2 GHz (dual core) with 4 GB RAM

\+ AMD Opteron 2.2 GHz (quad core) with 20 GB RAM

The table speaks for itself!

4 Example

To illustrate the command, we use anonymized patient-level data from the Health Search database, a nationally representative panel of patients run by the Italian College of General Practitioners (Italian Society of General Medicine). Our sample contains freely inputted EPRs concerning the prescription of diagnostic tests.[2] A list of 15 observations

2. The original data are in Italian. Where necessary for comprehension, we translate to English.

from the "uppercase" source variable `diagn_test_description` provides an overview of cases at hand:

```
. list diagn_test_descr in 1/15, noobs separator(20)

  ┌──────────────────────────────────┐
  │                  diagn_test_descr │
  ├──────────────────────────────────┤
  │                      TRIGLICERIDI │
  │                   EMOCROMO FORMULA │
  │                 COLESTEROLO TOTALE │
  │                            ALTEZZA │
  │               PT TEMPO PROTROMBINA │
  │      VISITA CARDIOLOGICA CONTROLLO │
  │                   HCV AB EPATITE C │
  │            COMPONENTE MONOCLONALE │
  │                   ATTIVITA´ FISICA │
  │  PSA ANTIGENE PROSTATICO SPECIFICO │
  │                     RX CAVIGLIA SN │
  │               FAMILIARITA´ K UTERO │
  │                       TRIGLICERIDI │
  │               URINE ESAME COMPLETO │
  │               URINE PESO SPECIFICO │
  └──────────────────────────────────┘
```

As you can see, this is a rich EPR dataset that is totally useless unless treated. If data were collected for research purposes, physicians would be given a finite number of possible options. There is much agreement in the scientific community that the cost to leave the burden of inputting standard codes directly to physicians at the time of contact with the patient is higher than the relative benefit: the task is extremely onerous, it is unrelated to the physician's primary job, and most importantly, it requires extra effort. Therefore, the common view supports the implementation of data-entry methods that do not disturb the physician's workflow (Yamazaki and Satomura 2000).

From the above list of observations, it is also clear that free-text data entry provides physicians with the freedom to determine the order and detail at which they want to input data. Even if the original free-text data were complete, it would still be difficult to extract standardized and structured data from this kind of record because of abbreviations, typos, or mistakes (Moorman et al. 1994). Extracting data in the presence of abbreviations and typos is exactly what `screening` allows you to do.

As a practical example, we focus on the identification of different types of cholesterol tests. In particular, our aim is to create a new variable (`diagn_test_code`) containing cholesterol test codes according to the Italian National Health System coding scheme. Because at least three types of cholesterol test exist, namely, `hdl`, `ldl`, and `total`, our matching strategy must take into account that a physician can input 1) only the types of the test, 2) only its broad definition (`cholesterol`), or 3) both, without considering abbreviations, typos, mistakes, and further details.

Thus we first explore the data by running `screening` with the `explore(tab)` option:

```
. screening, sources(diagn_test_descr, lower) keys(colesterolo) explore(tab)
Cases of colesterolo found in diagn_test_descr
```

colesterolo	Freq.	Percent	Cum.
colesterolo totale	2,954	51.86	51.86
hdl colesterolo	1,854	32.55	84.41
ldl colesterolo	617	10.83	95.24
colesterolo hdl	117	2.05	97.30
colesterolo ldl	37	0.65	97.95
colesterolo tot	28	0.49	98.44
colesterolo	24	0.42	98.86
colesterolo hdl sangue	16	0.28	99.14
colesterolo totale sangue	16	0.28	99.42
colesterolo esterificato	4	0.07	99.49
colesterolo tot.	4	0.07	99.56
colesterolo hdl 90.14.1	3	0.05	99.61
colesterolo totale 90143	3	0.05	99.67
colesterolo libero	2	0.04	99.70
colesterolo stick	2	0.04	99.74
colesterolo tot hdl	2	0.04	99.77
colesterolo totale 90.143	2	0.04	99.81
ultima misurazione colesterolo	2	0.04	99.84
colesterolo hdl	1	0.02	99.86
colesterolo ldl 90.14.2	1	0.02	99.88
colesterolo non ldl	1	0.02	99.89
colesterolo t. mg/dl	1	0.02	99.91
colesterolo tot. c	1	0.02	99.93
colesterolo tot. hdl	1	0.02	99.95
colesterolo tot.,	1	0.02	99.96
colesterolo totale h	1	0.02	99.98
rich,specialistica colesterolo trigl	1	0.02	100.00
Total	5,696	100.00	

Here the `lower` suboption makes the matching task case insensitive. Apart from the `explore(tab)` option, the syntax above is compulsory and performs what we call a default matching, that is, an exact match of the keyword `colesterolo` over the entire content of the source variable `diagn_test_descr`. The tabulation above (notice the lowercase) informs you that the keyword `colesterolo` is encountered in 5,696 cases. What do these cases contain? Because you did not instruct the command to match a shorter length of the keyword, the only possible case is the keyword itself; all the cases contain the keyword `colesterolo`.

Given the nature of the data, it might be convenient to run `screening` with a shorter length of the keyword so as to find possible partial matching in the presence of abbreviations or mistakes. The `letters(#)` option instructs `screening` to perform the match on a shorter length:

(*Continued on next page*)

```
. screening, sources(diagn_test_descr, lower) keys(colesterolo) letters(5)
> explore(tab)

Cases of coles found in diagn_test_descr

                                 coles |      Freq.     Percent        Cum.
---------------------------------------+-----------------------------------
                     colesterolo totale |      2,954       37.25       37.25
                        hdl colesterolo |      1,854       23.38       60.62
                              coles ldl |      1,343       16.93       77.56
                             hdl colest |        853       10.76       88.31
                        ldl colesterolo |        617        7.78       96.09
                        colesterolo hdl |        117        1.48       97.57
                        colesterolo ldl |         37        0.47       98.03
                        colesterolo tot |         28        0.35       98.39
                            colesterolo |         24        0.30       98.69
                 colesterolo hdl sangue |         16        0.20       98.89
              colesterolo totale sangue |         16        0.20       99.09
                        colesterolemia |         14        0.18       99.27
                            hdl colest. |          5        0.06       99.33
                            colest.tot. |          4        0.05       99.38
                colesterolo esterificato |          4        0.05       99.43
                       colesterolo tot. |          4        0.05       99.48
                azotemia glicemia colest |          3        0.04       99.52
                            colest. hdl |          3        0.04       99.56
                colesterolo hdl 90.14.1 |          3        0.04       99.60
               colesterolo totale 90143 |          3        0.04       99.63
                     colesterolo libero |          2        0.03       99.66
                      colesterolo stick |          2        0.03       99.68
                    colesterolo tot hdl |          2        0.03       99.71
              colesterolo totale 90.143 |          2        0.03       99.74
                            ldl colest. |          2        0.03       99.76
           ultima misurazione colesterolo |          2        0.03       99.79
                            colest. ldl |          1        0.01       99.80
                           colest. tot. |          1        0.01       99.81
                             colest.tot |          1        0.01       99.82
                      colester.tot.hdl, |          1        0.01       99.84
                        colesterolo hdl |          1        0.01       99.85
                colesterolo ldl 90.14.2 |          1        0.01       99.86
                    colesterolo non ldl |          1        0.01       99.87
                  colesterolo t. mg/dl |          1        0.01       99.89
                     colesterolo tot. c |          1        0.01       99.90
                   colesterolo tot. hdl |          1        0.01       99.91
                      colesterolo tot., |          1        0.01       99.92
                   colesterolo totale h |          1        0.01       99.94
                   emocromo c. colester |          1        0.01       99.95
                glicemia colesterolemia- |          1        0.01       99.96
 got gpt colest / trigli/creat/emocromo |          1        0.01       99.97
    rich,specialistica colesterolo trigl |          1        0.01       99.99
                 uricemia uricuria colest |          1        0.01      100.00
---------------------------------------+-----------------------------------
                                  Total |      7,931      100.00
```

By specifying a five-letter partial match, `screening` detects 2,235 new cases of cholesterol tests. By further reducing the number of letters, we get the following result:[3]

3. Because of space restrictions, we deliberately omit the complete tabulation obtainable with the `explore(tab)` option. It is available upon request.

```
. screening, sources(diagn_test_descr, lower) keys("colesterolo") letters(3)
> explore(tab)
Cases of col found in diagn_test_descr
                                    col |      Freq.     Percent        Cum.
----------------------------------------+-----------------------------------
                      colesterolo totale |      2,954       23.45       23.45
                                 col tot |      2,034       16.15       39.60
                         hdl colesterolo |      1,854       14.72       54.32
                               coles ldl |      1,343       10.66       64.99
                               hdl colest |        853        6.77       71.76
                         ldl colesterolo |        617        4.90       76.66
              urinocoltura coltura urina |        326        2.59       79.25
                          v.ginecologica |        161        1.28       80.52
           eco tiroide eco capo e collo |        150        1.19       81.71
                         colesterolo hdl |        117        0.93       82.64
  (output omitted)
                         colesterolo ldl |         37        0.29       90.77
  calcolo rischio cardiovascolare (iss) |         35        0.28       91.04
             coprocoltura coltura feci |         33        0.26       91.31
                                  colore |         32        0.25       91.56
        ecocolordoppler arti inf. art. |         32        0.25       91.81
                            urinocoltura |         32        0.25       92.07
                             colposcopia |         31        0.25       92.31
                         colesterolo tot |         28        0.22       92.54
                            reticolociti |         28        0.22       92.76
 ecodoppler a.inferiori ecocolor venosa |         27        0.21       92.97
                        eco ginecologica |         25        0.20       93.17
                             colesterolo |         24        0.19       93.36
       rischio cardio vascolare nota 13 |         23        0.18       93.55
   rischio cardiovascolare % a 10 anni |         22        0.17       93.72
 ecodoppler a.inferiori ecocolor arter. |         19        0.15       93.87
  (output omitted)
                                 col hdl |          3        0.02       97.13
                             colest. hdl |          3        0.02       97.16
                 colesterolo hdl 90.14.1 |          3        0.02       97.18
                 colesterolo totale 90143 |          3        0.02       97.21
conta batt.,urinocoltura, antibiogramma |          3        0.02       97.23
eco cardiaca con doppler e colordoppler |          3        0.02       97.25
                 eco color/doppl.car. ver |          3        0.02       97.28
                  eco(color)dopplergrafia |          3        0.02       97.30
           ecocardiografia colordoppler |          3        0.02       97.32
               ecocolordoppler art.aa.inf. |          3        0.02       97.35
ecocolordoppler arterioso arti inferior |          3        0.02       97.37
    ecocolordoppler tronchi sovraortici |          3        0.02       97.40
         ecocolordopplergrafia cardiaca |          3        0.02       97.42
             ecografia muscolotendinea |          3        0.02       97.44
      ecografia tiroide eco capo e collo |          3        0.02       97.47
familiarita´ ev.cerebrovascol.( 72m 74f |          3        0.02       97.49
              immunocomplessi circolanti |          3        0.02       97.51
          rx digerente (tenue e colon) |          3        0.02       97.54
   test broncodilatazione farmacologica |          3        0.02       97.56
test cardiovascolare da sforzo con cicl |          3        0.02       97.59
test sforzo cardiovascol. pedana mobile |          3        0.02       97.61
                   urinocoltura atb+mic |          3        0.02       97.63
          urinocoltura con antibiogramma |          3        0.02       97.66
 urinocoltura identificazione batt.+ ab |          3        0.02       97.68
                     che colinesterasi |          2        0.02       97.70
                                     col |          2        0.02       97.71
```

```
                       colangio rm |          2        0.02       97.73
                colesterolo libero |          2        0.02       97.75
                 colesterolo stick |          2        0.02       97.76
               colesterolo tot hdl |          2        0.02       97.78
         colesterolo totale 90.143 |          2        0.02       97.79
```
(*output omitted*)
```
                       ldl colest. |          2        0.02       98.11
```
(*output omitted*)
```
          col tot 216 hdl 58 fibri |          1        0.01       98.48
               col=245ldl=193tr=91 |          1        0.01       98.48
       colangiografia intravenosa |          1        0.01       98.49
                   colecistografia |          1        0.01       98.50
          colecistografia per os c |          1        0.01       98.51
                       colest. ldl |          1        0.01       98.52
                      colest. tot. |          1        0.01       98.52
                        colest.tot |          1        0.01       98.53
                 colester.tot.hdl, |          1        0.01       98.54
                 colesterolo hdl   |          1        0.01       98.55
         colesterolo ldl 90.14.2 |          1        0.01       98.55
               colesterolo non ldl |          1        0.01       98.56
              colesterolo t. mg/dl |          1        0.01       98.57
                colesterolo tot. c |          1        0.01       98.58
              colesterolo tot. hdl |          1        0.01       98.59
                 colesterolo tot., |          1        0.01       98.59
              colesterolo totale h |          1        0.01       98.60
             colloquio psicologico |          1        0.01       98.61
```
(*output omitted*)
```
                           hdl col |          1        0.01       99.22
```
(*output omitted*)
```
visita specialistica colonscopia con bi |     1        0.01       99.99
             yersinia coltura feci |          1        0.01      100.00
-----------------------------------+-----------------------------------
                             Total |     12,595      100.00
```

Again `screening` detects new cases: 2,034 cases characterized by the abbreviation `col tot` (that is, total cholesterol) that are impossible to identify without further reducing the number of letters. The problem is that, among all matched cases (12,595), there are also a number of unwanted cases, that is, cases containing the same spelling of the keyword but related to another type of diagnostic test. Despite this incorrect identification, we will show later in the section how to obtain a new "recoded variable" by specifying the appropriate *recoding_rule* as an argument of the `recode()` option.

The number of letters you match plays a critical role: specifying a high number of letters may cause the number of matched observations to be artificially low due to mistakes or abbreviations in the source variables; on the other hand, matching a small number of letters may cause the number of matched observations to be artificially high due to the inclusion of uninteresting cases containing the "too short" keyword.

As mentioned above, we are interested in the identification of three types of cholesterol tests. To achieve this objective, in what follows we focus on a set of four keywords (`totale`, `colesterolo`, `ldl`, `hdl`) with three identifying letters. We also specify the `newcode()` option to generate a new variable recoding the observations that match the specified keywords.

At this point, we describe more deeply the recoding mechanism of `screening`:

- If `newcode()` is specified, a new variable is generated, taking as values the position of the keywords or regular expressions specified through the `keys()` option. The coding process is driven by the order of keywords or regular expressions.

- If `recode()` is specified, the `newcode()` *newvar* suboption is recoded according to the user-defined coding scheme.

Thus a first recoding of the source variable can be obtained as follows:

```
. screening, sources(diagn_test_descr, lower)
> keys("totale" "colesterolo" "ldl" "hdl") letters(3 3 3 3) explore(count)
> newcode(tmp_diagn_test_code)
          Source   Key        Freq.   Percent

diagn_test_descr   tot         7304     29.47
                   col        12595     50.81
                   ldl         2015      8.13
                   hdl         2872     11.59

                   Total      24786    100.00
. tabulate tmp_diagn_test_code
tmp_diagn_t
   est_code       Freq.     Percent        Cum.

          1       7,304       49.15       49.15
          2       7,535       50.70       99.85
          3          12        0.08       99.93
          4          11        0.07      100.00

      Total      14,862      100.00
```

The `explore(count)` option instructs `screening` to display a table of frequency counts of all matched cases. The `newcode()` option creates `tmp_diagn_test_code`, which is a new variable that takes as values the position of the keywords or regular expressions specified through the `keys()` option. The coding process is driven by the order of keywords or regular expressions: the number `1` is associated with the 7,304 observations matching the first keyword, `tot`; the number `2` is associated with the 7,535 observations matching the second keyword, `col`; and so on. Hence, by specifying `keys("totale" "colesterolo" "ldl" "hdl")` together with `letters(3 3 3 3)`, `tot` takes precedence over `col` in the recoding process. This means that if some observations are recoded according to the first keyword match, they will not be recoded according to the following keywords in the `keys()` list, even if they match.

For this reason, the best recoding strategy is to first specify keywords that uniquely identify the cases of interest. Because keywords `hdl` and `ldl` each uniquely identify a cholesterol test, they must have priority in the recoding process over `totale`, which is an extension common to other pathologies.

Indeed, when we reverse the order of the keywords and specify the `replace` suboption in the `newcode()` option, `screening` produces

```
. screening, sources(diagn_test_descr, lower)
> keys("hdl" "ldl" "colesterolo" "totale") letters(3 3 3 3)
> newcode(tmp_diagn_test_code, replace)
WARNING: By specifying -replace- sub-option you are overwriting the -newcode()-
> variable.
. tabulate tmp_diagn_test_code

tmp_diagn_t
   est_code |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |      2,872       19.32       19.32
          2 |      2,015       13.56       32.88
          3 |      7,731       52.02       84.90
          4 |      2,244       15.10      100.00
------------+-----------------------------------
      Total |     14,862      100.00
```

where the `newcode()` variable now identifies all `hdl` and `ldl` cases. Notice that here we followed the correct approach, from specific to general. Moreover, as shown by the following code, when we specify the `newcode()` suboption `label`, `screening` attaches the specified keywords as value labels to the `newcode()` variable.

```
. screening, sources(diagn_test_descr, lower)
> keys("hdl" "ldl" "colesterolo" "totale") letters(3 3 3 3)
> newcode(tmp_diagn_test_code, replace label)
WARNING: By specifying -replace- sub-option you are overwriting the -newcode()-
> variable.
. tabulate tmp_diagn_test_code

tmp_diagn_t
   est_code |      Freq.     Percent        Cum.
------------+-----------------------------------
        hdl |      2,872       19.32       19.32
        ldl |      2,015       13.56       32.88
colesterolo |      7,731       52.02       84.90
     totale |      2,244       15.10      100.00
------------+-----------------------------------
      Total |     14,862      100.00
```

The last step toward recoding is achieved by using the `recode()` option. This option allows you to recode the `newcode()` variable according to a user-defined coding scheme. When you specify this option, the coding process is completely under your control. The `recode()` option requires a *recoding_rule* followed by a "*user_defined_code*" (the "*user_defined_code*" must be enclosed within double quotes).

When we specify `recode(1 "90.14.1" ...)`, the standard code `"90.14.1"` will be used to recode all matched cases from the first keyword (`hdl`); when we specify recode(... 2 "90.14.2" ...), the standard code "90.14.2" will be used to recode all matched cases from the second keyword (ldl); and so on. The third and forth keywords deserve special attention. totale (which was specified as the forth keyword, hence position 4) is a common extension that we want to identify only when it is matched simultaneously with colesterolo (which was specified as the third keyword, hence position 3). Thus the appropriate syntax in this case will be recode(... 3,4 "90.14.3" ...). Finally, when we specify recode(... 3 "not class. tests"), the code "not class. tests" will be used to recode all matched cases from the third keyword (colesterolo) that are not classified because they do not contain any further specification.

The final syntax of our example is

```
. screening, sources(diagn_test_descr, lower)
> keys("hdl" "ldl" "colesterolo" "totale") letters(3 3 3 3)
> newcode(diagn_test_code)
> recode(1 "90.14.1" 2 "90.14.2" 3,4 "90.14.3" 3 "not class. tests")
. tabulate diagn_test_code
```

diagn_test_code	Freq.	Percent	Cum.
90.14.1	2,872	22.76	22.76
90.14.2	2,015	15.97	38.73
90.14.3	5,055	40.06	78.79
not class. tests	2,676	21.21	100.00
Total	12,618	100.00	

As the tabulate command shows, the new variable diagn_test_code is created according to the user-defined codes. Notice that only 5,055 cases are coded as "total cholesterol" (90.14.3). A two-way tabulate command (below) helps to highlight that 2,244 cases have to be considered incorrect identifications—that is, cases containing the same spelling of the keywords (totale) but related to other types of diagnostic tests[4] —whereas 2,676 are incomplete because they contain only colesterolo without further specification.

```
. tabulate diagn_test_code tmp_diagn_test_code if tmp_diagn_test_code !=., m
```

	tmp_diagn_test_code				
diagn_test_code	hdl	ldl	colestero	totale	Total
	0	0	0	2,244	2,244
90.14.1	2,872	0	0	0	2,872
90.14.2	0	2,015	0	0	2,015
90.14.3	0	0	5,055	0	5,055
not class. tests	0	0	2,676	0	2,676
Total	2,872	2,015	7,731	2,244	14,862

This example shows that screening is a simple tool to manage complex string variables. Once you have obtained structured data (in our example, a categorical variable indicating cholesterol tests), you can finally start your statistical analysis.

4. Because of space restrictions, we deliberately omit the tabulation of such cases. It is available upon request.

5 Extensions

Although the main utility of `screening` is the direct translation of complex narrative-text variables in a user-defined coding scheme, the command is flexible enough to cover a wide range of situations. In section 5.1, we present an example of how to use the command to facilitate the merging of information from different sources, while in section 5.2, we show how to use `screening` to extract or rearrange a portion of a string variable.

5.1 Merging from different sources

In applied studies, a classic problem comes from trying to merge information from different sources that use different codes for the same units. A recently released command, `kountry` (Raciborski 2008), is an important step toward a solution.

The `kountry` command can be used to facilitate the merging of information from different sources by recoding a string variable into a standardized form. This recoding is possible using a custom dictionary created through a helper command.[5] In this section, we show an alternative way to merge information from different sources by using the `screening` command.

As an example, we try to merge two Italian datasets, one provided by the National Statistical Office (National Institute of Statistics in Italy) and the other provided by the Italian Ministry of the Interior. The two datasets contain, for each Italian municipality, the complete name and an alphanumeric code, the latter being different across sources. In theory, with the (uniquely identified) name of each municipality, it should be easy to merge the two datasets.

We first proceed by matching the two original datasets:

```
. use istat, clear
. sort comune
. merge m:m comune using ministero
  (output omitted)
. tabulate _merge
```

_merge	Freq.	Percent	Cum.
master only (1)	288	3.43	3.43
using only (2)	290	3.46	6.89
matched (3)	7,812	93.11	100.00
Total	8,390	100.00	

5. See `help kountryadd` (if `kountry` is installed).

As you can see, there are 288 inconsistencies.[6] When we tabulate the unmatched cases, we would realize that unconventional expressions, like apostrophes, accents, double names, etc., are responsible for this imperfect result:

```
. preserve
. sort comune
. drop if _merge==3
(7812 observations deleted)
. list comune _merge in 1/20, separator(20) noobs

  ┌─────────────────────────────────────────┐
  │                      comune    _merge   │
  ├─────────────────────────────────────────┤
  │                      AGLIE´         2   │
  │                       AGLI          1   │
  │              ALA´ DEI SARDI         2   │
  │              ALBISOLA MARINA        2   │
  │           ALBISOLA SUPERIORE        2   │
  │             ALBISSOLA MARINA        1   │
  │          ALBISSOLA SUPERIORE        1   │
  │                        ALI´         2   │
  │                  ALI´ TERME         2   │
  │            ALLUVIONI CAMBIO´        2   │
  │             ALLUVIONI CAMBI         1   │
  │                       ALME´         2   │
  │                        ALM          1   │
  │                AL DEI SARDI         1   │
  │                         AL          1   │
  │                    AL TERME         1   │
  │           ANTEY-SAINT-ANDRE´        2   │
  │            ANTEY-SAINT-ANDR         1   │
  │       APPIANO SULLA STRADA DEL      2   │
  │  APPIANO SULLA STRADA DEL VINO      1   │
  └─────────────────────────────────────────┘

. restore
```

If you wish to recover all 288 unmatched municipalities, the proposed command is a simple and fast solution. Indeed, when you take advantage of the available options, you can (almost) completely recover unmatched cases with only one command. As an example, we recover nine cases (it is possible to recover all cases with this procedure), with a loop running on values of `_merge` equal to 1 or 2, that is, running only on unmatched cases:

6. The number of unmatched cases is different between the master (288) and the using (290) datasets because of aggregation and separation of municipalities. Solving this kind of problem is beyond the illustrative scope of this example.

```
. forvalues i=1/2 {
  2.         preserve
  3.         keep if _merge==`i´
  4.
. screening, sources(comune) keys("ALBISSOLA" "AQUILA D´ARROSCIA" "BAJARDO"
> "BARCELLONA" "BARZAN" "BRIGNANO" "CADERZONE" "CAVAGLI" "MARINA" "SUPERIORE")
> cases(cases) newcode(comune, replace)
> recode(1,9 "ALBISOLA MARINA" 1,10 "ALBISOLA SUPERIORE" 2 "AQUILA DI ARROSCIA"
> 3 "BAIARDO" 4 "BARCELLONA POZZO DI GOTTO" 5 "BARZANO´" 6 "BRIGNANO FRASCATA"
> 7 "CAVAGLIA" 8 "CADERZONE TERME")
  5.         if `i´==1 drop codice_ente
  6.         if `i´==2 drop codice
  7.         keep  comune codice
  8.         sort comune
  9.         save new_`i´,replace
 10.        restore
 11. }
(8102 observations deleted)
WARNING: By specifying -replace- sub-option you are overwriting the -newcode()-
> variable.
(note: file new_1.dta not found)
file new_1.dta saved
(8100 observations deleted)
WARNING: By specifying -replace- sub-option you are overwriting the -newcode()-
> variable.
(note: file new_2.dta not found)
file new_2.dta saved
. keep if _merge==3
(578 observations deleted)
. save perfect_match, replace
(note: file perfect_match.dta not found)
file perfect_match.dta saved
. use new_1, clear
. merge 1:1 comune using new_2
  (output omitted)
. tabulate _merge

                 _merge |      Freq.     Percent        Cum.
------------------------+-----------------------------------
        master only (1) |        279       49.03       49.03
         using only (2) |        281       49.38       98.42
            matched (3) |          9        1.58      100.00
------------------------+-----------------------------------
                  Total |        569      100.00
. append using perfect_match
. tabulate _merge

                 _merge |      Freq.     Percent        Cum.
------------------------+-----------------------------------
        master only (1) |        279        3.33        3.33
         using only (2) |        281        3.35        6.68
            matched (3) |      7,821       93.32      100.00
------------------------+-----------------------------------
                  Total |      8,381      100.00
```

Because we deliberately recovered only nine cases, the number of unmatched cases before the execution of `screening` is improved by nine cases, from 7,812 to 7,821 exact matches.

5.2 Extracting a piece of a string variable

In this section, we show through three examples how `screening` can be used to extract or rearrange a portion of a string variable.[7]

Example 1

Imagine you have the string variable `address`, and you want to create a new variable that contains just the zip codes. Here is what the source variable `address` may look like:

```
. list, noobs sep(10)

  +------------------------------------------------------+
  |                                              address |
  |------------------------------------------------------|
  | 4905 Lakeway Drive, College Station, Texas 77845 USA |
  |            673 Jasmine Street, Los Angeles, CA 90024 |
  |               2376 First street, San Diego, CA 90126 |
  |                66666 West Central St, Tempe AZ 80068 |
  |              12345 Main St. Cambridge, MA 01238-1234 |
  |            12345 Main St  Sommerville  MA 01239-2345 |
  |             12345 Main St  Watertwon  MA 01239   USA |
  +------------------------------------------------------+
```

To find the zip code, you have to use `screening` with specific regular expressions, allowing it to exactly match all cases in the source variable `address`. Some examples of specific regular expressions are the following:

- `([0-9][0-9][0-9][0-9][0-9])` to find a five-digit number, the zip code
- `[\-]*` to match zero or more dashes, - or --
- `[0-9]*` to match zero or more numbers, that is, the zip code plus any other numbers
- `[ a-zA-Z]*` to match zero or more blank spaces and (lowercase or uppercase) letters

Once the correct regular expression(s) is found, to use `screening` to create a new variable containing the zip codes, you have to do the following:

7. The following examples have been taken from the UCLA website resources to help you learn and use Stata. See http://www.ats.ucla.edu/stat/stata/faq/regex.htm.

1. Use the `newcode()` option to create the new variable `zipcode`.

2. Combine the above regular expressions and use them as a unique keyword.

3. Use the `regexs(`*n*`)` function as a `"`*user_defined_code*`"` in the `recode()` option. `regexs(`*n*`)` returns the subexpression n from the respective keyword match, where $0 \leq n \leq 10$. Stata regular-expression syntaxes use parentheses, `()`, to denote a subexpression group. In particular, $n = 0$ is reserved for the entire string that satisfied the regular expression (keyword); $n = 1$ is reserved for the first subexpression that satisfied the regular expression (keyword); and so on.

Hence, you may code

```
. screening, sources(address)
> keys("([0-9][0-9][0-9][0-9][0-9])[\-]*[0-9]*[ a-zA-Z]*$")
> cases(c) newcode(zipcode) recode(1 "regexs(1)")
WARNING! You are SCREENING some keywords using regular-expression operators
> like ^ . ( ) [ ] ? *
         Notice that:
 1) Option -letter- doesn´t work IF a keyword contains regular-expression operators
 2) Unless you are looking for a specific regular-expression, regular-expression
    operators must be preceded by a backslash \ to ensure keyword-matching
    (e.g. \^ \. )
 3) To match a keyword containing $ or \, you have to specify them as [\$] [\\]

. tabulate zipcode
```

zipcode	Freq.	Percent	Cum.
01238	1	14.29	14.29
01239	2	28.57	42.86
77845	1	14.29	57.14
80068	1	14.29	71.43
90024	1	14.29	85.71
90126	1	14.29	100.00
Total	7	100.00	

where `recode(1 "regexs(1)")` indicates that

1. `1` is the *recoding_rule*; that is, the coding process is related to the first (and unique) keyword match.

2. `regexs(1)` is used to recode. Indeed, it returns the string related to the first (and unique) subexpression match.[8]

As a result, the new variable `zipcode` is created by using only one line of code. Notice that `screening` warns you that you are matching a keyword containing one or more regular-expression operators.

8. Remember that subexpressions are denoted by using `()`. In the considered syntax, the only subexpression is represented by `([0-9][0-9][0-9][0-9][0-9])`. This means that, in this case, you cannot specify $n > 1$.

Example 2

Suppose you have a variable containing a person's full name. Here is what the variable `fullname` looks like:

```
. list, noobs sep(10)

  ┌──────────────┐
  │     fullname │
  ├──────────────┤
  │   John Adams │
  │  Adam Smiths │
  │  Mary Smiths │
  │ Charlie Wade │
  └──────────────┘
```

Our goal is to swap first name with last name, separating them by a comma. The regular expression to reach the target is `(([a-zA-Z]+)[ ]*([a-zA-Z]+))`. It is composed of three parts:

1. `([a-zA-Z]+)` to capture a string consisting of letters (lowercase and uppercase), that is, the first name
2. `[ ]*` to match with a space(s), that is, the blank between first and last name
3. `([a-zA-Z]+)` again to capture a string consisting of letters, this time the last name

The following is a way to proceed using `screening`:

```
. screening, sources(fullname)
> keys("([a-zA-Z]+)[ ]*([a-zA-Z]+)" "[ ]" "([a-zA-Z]+)[ ]*([a-zA-Z]+)")
> newcode(fullname, add replace) recode(1 "regexs(2)," 2 "regexs(0)"
> 3 "regexs(1)")
WARNING! You are SCREENING some keywords using regular-expression operators
> like ^ . ( ) [ ] ? *
         Notice that:
 1) Option -letter- doesn´t work IF a keyword contains regular-expression operators
 2) Unless you are looking for a specific regular-expression, regular-expression
    operators must be preceded by a backslash \ to ensure keyword-matching
    (e.g. \^ \. )
 3) To match a keyword containing $ or \, you have to specify them as [\$] [\\]
. list fullname, noobs sep(10)

  ┌───────────────┐
  │      fullname │
  ├───────────────┤
  │   Adams, John │
  │  Smiths, Adam │
  │  Smiths, Mary │
  │ Wade, Charlie │
  └───────────────┘
```

Notice the `newcode()` suboption `add`. It can be specified only when a `regexs(`*n*`)` function is specified as a `"`*user_defined_code*`"` in the `recode()` option. The `add` suboption allows for the creation of the `newcode()` variable as a concatenation of subexpressions returned by `regexs(`*n*`)`. In the example above,

1. `recode(1 "regexs(2)," ...` returns the second subexpression from the first keyword match (the last name) plus a comma.
2. `...2 "regexs(0)" ...` returns the blank matched by the second keyword;
3. `...3 "regexs(1)")` returns the first subexpression from the third keyword match (the first name).

As a result, the variable `fullname` is replaced (note the suboption `replace`) sequentially by the concatenation of subexpressions returned by 1, 2, and 3 above.

Example 3

Imagine that you have the string variable `date` containing dates:

```
. list date, noobs sep(20)
```

date
20jan2007
16June06
06sept1985
21june04
4july90
9jan1999
6aug99
19august2003

The goal is to produce a string variable with the appropriate four-digit year for each case, which Stata can easily convert into a date. You can achieve the target by coding something like the following:

```
. generate day = regexs(0) if regexm(date, "^[0-9]+")
. generate month = regexs(0) if regexm(date, "[a-zA-Z]+")
. generate year = regexs(0) if regexm(date, "[0-9]*$")
. replace year = "20"+regexs(0) if regexm(year, "^[0][0-9]$")
(2 real changes made)
. replace year = "19"+regexs(0) if regexm(year, "^[1-9][0-9]$")
(2 real changes made)
. generate date1 = day+month+year
```

```
. list, noobs sep(10)
```

date	day	month	year	date1
20jan2007	20	jan	2007	20jan2007
16June06	16	June	2006	16June2006
06sept1985	06	sept	1985	06sept1985
21june04	21	june	2004	21june2004
4july90	4	july	1990	4july1990
9jan1999	9	jan	1999	9jan1999
6aug99	6	aug	1999	6aug1999
19august2003	19	august	2003	19august2003

Alternately, you can obtain the same result by using `screening`:

```
. screening, sources(date) keys("^[0-9]+" "[a-zA-Z]+" "[0][0-9]$" "[1-9][0-9]$")
> newcode(date1, add)
> recode(1 "regexs(0)" 2 "regexs(0)" 3 "20+regexs(0)" 4 "19+regexs(0)")
WARNING! You are SCREENING some keywords using regular-expression operators
> like ^ . ( ) [ ] ? *
         Notice that:
 1) Option -letter- doesn´t work IF a keyword contains regular-expression operators
 2) Unless you are looking for a specific regular-expression, regular-expression
    operators must be preceded by a backslash \ to ensure keyword-matching
    (e.g. \^ \. )
 3) To match a keyword containing $ or \, you have to specify them as [\$] [\\]
. list date date1, noobs sep(10)
```

date	date1
20jan2007	20jan2007
16June06	16June2006
06sept1985	06sept1985
21june04	21june2004
4july90	4july1990
9jan1999	9jan1999
6aug99	6aug1999
19august2003	19august2003

Also in this case, as in the previous example, we specify the `newcode()` suboption `add` because we need to create the `newcode()` variable as a concatenation of subexpressions from keyword matching. The same result can be obtained using the following syntax:

(*Continued on next page*)

```
. screening, sources(date)
> keys(begin "[0-9]+" "[a-zA-Z]+" end "[0][0-9]" end "[1-9][0-9]")
> newcode(date1, add)
> recode(1 "regexs(0)" 2 "regexs(0)" 3 "20+regexs(0)" 4 "19+regexs(0)")
WARNING! You are SCREENING some keywords using regular-expression operators
> like ^ . ( ) [ ] ? *
         Notice that:
 1) Option -letter- doesn´t work IF a keyword contains regular-expression operators
 2) Unless you are looking for a specific regular-expression, regular-expression
    operators must be preceded by a backslash \ to ensure keyword-matching
    (e.g. \^ \. )
 3) To match a keyword containing $ or \, you have to specify them as [\$] [\\]
. list date date1, noobs sep(10)
```

date	date1
20jan2007	20jan2007
16June06	16June2006
06sept1985	06sept1985
21june04	21june2004
4july90	4july1990
9jan1999	9jan1999
6aug99	6aug1999
19august2003	19august2003

where the only difference is represented by the way in which the *matching_rule* is specified: `begin` instead of `^` and `end` instead of `$`.

6 Summary

In this article, we introduced the new `screening` command, a data-management tool that helps you examine and treat the content of string variables containing free, possibly complex, narrative text. `screening` allows you to build new variables, to recode new or existing variables, and to build a set of categorical variables indicating keyword occurrences (a first step toward textual analysis). Considerable efforts were devoted to making the command as flexible as possible; thus `screening` contains a rich set of options that is intended to cover the most frequently encountered problems and necessities. Because of this flexibility, the command can be used in many different fields, like EPR data, data from different sources, or survey data. The execution of `screening` is fast, thanks to Mata programming; its syntax is simple and common to many other Stata commands, thus it is useful for all users regardless of their levels of experience in Stata. We especially recommend that you use the `explore()` option; it makes the command a useful data-mining tool. Nevertheless, expert users can exploit a more complicated syntax that substantially eases the preparatory burden for data cleaning.

Acknowledgments

We would like to thank Alice Cortignani, Rossana D'Amico, Andrea Piano Mortari, and Riccardo Zecchinelli who tested the command, Vincenzo Atella who read an earlier version of the article, Iacopo Cricelli who provided us with EPR data, and Rafal Raciborski for useful discussions. We are also grateful to David Drukker and all participants at the 2009 Italian Stata Users Group meeting. Finally, the suggestions made by the referee and the editor were useful to improve the command. We are responsible for any remaining errors.

7 References

Moorman, P. W., A. M. van Ginneken, J. van der Lei, and J. H. van Bemmel. 1994. A model for structured data entry based on explicit descriptional knowledge. *Methods of Information in Medicine* 33: 454–463.

Raciborski, R. 2008. kountry: A Stata utility for merging cross-country data from multiple sources. *Stata Journal* 8: 390–400.

Yamazaki, S., and Y. Satomura. 2000. Standard method for describing an electronic patient record template: Application of XML to share domain knowledge. *Methods of Information in Medicine* 39: 50–55.

About the authors

Federico Belotti is a PhD student in econometrics and empirical economics at the University of Rome Tor Vergata.

Domenico Depalo is a researcher in the Economic Research Department of the Bank of Italy in Rome. He received his PhD in econometrics and empirical economics from the University of Rome Tor Vergata and was enrolled in a Post Doc program at the University of Rome La Sapienza.