The Stata Journal (*yyyy*) *vv*, Number *ii*, pp. 1–21

Record Linkage using STATA: Pre-processing, Linking and Reviewing Utilities

Nada Wasi
Survey Research Center
Institute for Social Research
University of Michigan
nwasi@umich.edu

Aaron Flaaen
Department of Economics
University of Michigan
aflaaen@umich.edu

Abstract.

This article describes STATA utilities which facilitate several steps in conducting probabilistic record linkage – the technique typically employed for merging two datasets with no common record identifier. While the pre-processing tools are developed specifically for linking two company databases, the other tools can be used for many different types of linkage. Specifically, the `stnd_compname` and `stnd_address` commands parse and standardize company names and addresses in order to improve the match quality in the linking step. The `reclink2` command is a generalized version of `reclink` that allows for a many-to-one matching procedure. Finally, `clrevmatch` is an interactive tool that allows the user to review matched results in an efficient and seamless manner. Rather than exporting results to another file format (e.g., Excel), inputting clerical reviews, and importing back into STATA, the clrevmatch tool conducts all of these steps within STATA. This helps improve the speed and flexibility of the whole matching process which often involves multiple runs.

Keywords: record linkage, fuzzy matching, string standardization

1 Introduction

Businesses, government agencies and academic researchers increasingly collect information about companies, their profiles and various business activities (e.g., ReferenceUSA, SEC filings, LexisNexis, the Business Register of the U.S. Census Bureau). This information can be collected at several different levels of aggregation: plant (establishment or branch), firm or tax identifying unit. Several household surveys also ask respondents to report name, address, and other characteristics of their employers. As these databases contain specific information based on the purpose of their construction, researchers often need to combine data from multiple sources to facilitate their analysis. For instance, Abowd and Stinson [2013] link employers from the Survey of Income and Program Participation to those in the Social Security Administration's Detailed Earnings Record to study measurement errors from self-reported earnings. Agrawal and Tambe [2013] match employers from workers' resumes to a firm history database to assess how private equity acquisitions impact labor market outcomes of workers.

When two datasets have a common unit identifier (e.g., firm's identification number), merging datasets is a trivial exercise. However, in many cases no common identifier exists – making it challenging to join corresponding observations from different datasets. Probabilistic record linkage (also known as data matching and fuzzy merge) is typically employed in this situation. Entities are linked based on other partial-identifiers such as names and addresses. Linking via such fields is complicated by a number of factors: different databases likely record data in different formats, the potential for misspellings and alternate name conventions, and so on. An example below illustrates the difficulty in this form of matching. Here a researcher would like to match self-reported employers from a household survey (Table 1) to a firm database (Table 2).[1]

Usually, company records obtained from household surveys do not always contain full official company names whereas records from a firm database often do. Even within the same dataset, the abbreviations used may vary across records. Using STATA's `merge` command based on name and street address will yield only one match pair (respondent #5 and firm #8). Unlike `merge`, probabilistic record linkage relies on an approximate string comparison function so that records with the most "similar" strings are joined as a match. The formal mathematics of probabilistic record linkage is developed by Fellegi and Sunter [1969]. Christen [2012] provides a comprehensive review of issues and methods related to record linkage.

In practice, the process involves three key steps: (1) pre-processing; (2) probabilistic linking; and (3) clerical review of machine-generated matched pairs. The pre-processing step assures both datasets have the same formats and chosen fields are meaningful in matching. Typically, the pre-processing step itself consists of two substeps: parsing a field into the relevant sub-components, and standardizing common character strings. This often helps researchers achieve higher quality matches in the linking step. As an example, consider the employer of respondent #2. Without pre-processing, firm #2 "AT&T INC." (an incorrect match) will look more similar to "BT&T INC." than "BB & T FKA COASTAL FEDERAL BANK" (a correct match) would. Other than the typo, this is because (1) the "INC." characters make respondent #2's employer and firm #2 more similar; (2) the record of firm #14 contains extra information about its formerly known as (FKA) name; and (3) the presence of a white space before and after the & character. Pre-processing that parses its entity type and alternate name into separate fields and standardizing the fields to ensure format consistency would solve these problems.

The second step involves linking records from two datasets. In this step, researchers choose a set of fields (e.g., standardized name, standardized address) as inputs into a probabilistic matching algorithm. For each record from the first dataset, the algorithm selects candidates from the second data set. These candidates may be all records from the second dataset or are selected based on certain criteria (e.g., only records from the same state). Then, for each pair consisting of a record from the first dataset and a corresponding candidate from the second dataset, the program uses a string comparison function to calculate field-similarity scores. This is accomplished for each input field individually, and then a (composite) pair-similarity score is constructed as the sum of

[1] The examples presented in this paper contain no actual respondent data from any survey.

Table 1: An example of employer records from a household survey

Respondent id	**Name**	**Street add**
1	7-11	ROUTH STREET
2	BT&T INC.	P.O. BOX 345
3	AT & T	208 S. AKARD ST
4	KROGER	
5	WAL-MART STORES, INC.	508 SW 8TH STREET
6	WLAMART	508 8TH STREET
7	WALMART	508 8TH ST

Table 2: An example of records from a firm database

Firm id	**Name**	**Street add**
1	7-ELEVEN, INC	1722 ROUTH STREET
2	AT&T INC.	P.O. BOX 132160
3	DISH NETWORK CORPORATION	9601 SOUTH MERIDIAN BOULEVARD
4	HVM L.L.C. D/B/A EXTENDED STAY HOTELS	11525 N. COMMUNITY HOUSE ROAD
5	RHEEM MANUFACTURING COMPANY	1100 ABERNATHY RD NE STE 1400
6	STARBUCKS CORPORATION	2401 UTAH AVENUE SO., 8TH FLOOR
7	THE KROGER CO	1014 VINE ST
8	WAL-MART STORES, INC.	508 SW 8TH STREET
9	KMART CORPORATION	3333 BEVERLY ROAD
10	PROFESSIONAL PHARMACIES INC DBA PLAZA PHARMACY	11 BRIDGEWAY PLAZA
11	MADISON HOLDINGS, INC. C/O WORLD FINANCIAL	270 PARK AVENUE, SUITE 1503
12	RESORTS U.S.A. T/A SEASIDE RESORT	18 W. JIMMIE ROAD
13	PG INDUSTRIES ATTN JOHN SMITH	PO BOX 2706
14	BB & T FKA COASTAL FEDERAL BANK	POB 345

all field-similarity scores, adjusted by specified weights. The candidate with the highest pair-similarity score is chosen as “a match”.

Although the pair-similarity scores are correlated with correct matches, they are an imperfect metric. A manual clerical review of machine-generated matched pairs is usually necessary, especially for pairs with low scores. Typical record linking processes require several runs (often called passes) where researchers try different combinations of fields, criteria for choosing candidates (blocking strategies) and their associated weights. Results from each run are reviewed and unmatched records go to the next run to be tried again with different matching specifications.

This paper introduces a set of utilities which facilitate the pre-processing and clerical review steps. It also briefly explains a modification of an existing record linkage command (`reclink`, see Blasnik [2010]) to make it more flexible. The example above will be used throughout the paper, although the actual record linkage tasks often involve very large databases. Sections 2 and 3 explain the `stnd_compname` and `stnd_address` commands which parse and standardize company names and addresses, respectively. These parsers and standardizers are based on a set of default rule-based pattern files, which are installed in conjunction with the commands. Section 4 explains how advanced users can modify these pattern files to construct specialized pre-processing rules for an individual matching exercise. The new `reclink2` command is described in Section 5. Unlike `reclink` which assumes a one-to-one relationship between two datasets, `reclink2` allows for many-to-one matching. Although a minor modification, it represents a substantial increase in the versatility of the command. Many record-linking exercises are by nature a many-to-one match. This is the case of our example above where more than one respondent may work for the same employer. Other examples include matching establishments to firms, and matching customer location of sale with establishment directories. Finally, Section 6 explains the `clrevmatch` command: an interactive tool allowing a researcher to review and assess each matched pair generated by a record-linking program. This utility increases the efficiency of the clerical review procedure, typically one of the most time-intensive tasks. It also helps improve the speed of the whole matching process which often involves multiple runs. Without `clrevmatch`, users usually need to export results to another file format (e.g., Excel), input clerical reviews, and then import back into STATA.

2 The stnd_compname command

2.1 Syntax

`stnd_compname` *varname* [`if` *exp*] [`in` *range*]`,` `gen(`*newvarname*`)`
  [`patpath(`*directory of pattern files*`)`]

2.2 Description

The `stnd_compname` command standardizes and parses a string variable containing company names into 5 components; `gen(`*newvarname*`)` is required. The generated outputs are in the following order: (1) official name; (2) Doing-Business-As (DBA) name; (3) Formerly-Known-As (FKA) name; (4) business entity type; and (5) attention name. Each component is standardized. If a given name cannot be parsed, the original value is recorded in the official name field. `stnd_compname` relies on several subcommands and rule-based pattern files. These subcommands and pattern files must also be installed. The default directory of the pattern files is /ado/plus/p/. If the pattern files are installed in a different directory, the user must specify the directory in the `patpath()` option. If a particular pattern file is not found, the program will display a warning message and the standardizing or parsing step associated with that file will be skipped. The default pattern files are based on U.S. business names. See Section 4 for details.

2.3 Examples

The following examples apply `stnd_compname` to the company names listed in the introduction section. The respondent_employers dataset contains the employer names from the household survey in Table 1. The variable firm_name is the original variable containing company names to be standardized.

```
. use respondent_employers, clear

. stnd_compname firm_name, gen(stn_name stn_dbaname stn_fkaname entitytype attn
> _name)

. list firm_name stn_name stn_dbaname entitytype

     +---------------------------------------------------------------+
     |             firm_name          stn_name   stn_db~e   entity~e |
     |---------------------------------------------------------------|
  1. |                  7-11              7 11                       |
  2. |             BT&T INC.            BT & T                   INC |
  3. |                AT & T            AT & T                       |
  4. |                KROGER            KROGER                       |
  5. | WAL-MART STORES, INC.   WAL MART STORES                   INC |
     |---------------------------------------------------------------|
  6. |               WLAMART           WLAMART                       |
  7. |               WALMART           WALMART                       |
     +---------------------------------------------------------------+
```

The firm_dataset dataset contains the firm listing in Table 2.

```
. use firm_dataset, clear

. list firm_name

     +------------------------------------------------+
     |                                      firm_name |
     |------------------------------------------------|
  1. |                                  7-ELEVEN, INC |
  2. |                                      AT&T INC. |
  3. |                       DISH NETWORK CORPORATION |
  4. |          HVM L.L.C. D/B/A EXTENDED STAY HOTELS |
  5. |                    RHEEM MANUFACTURING COMPANY |
     |------------------------------------------------|
  6. |                          STARBUCKS CORPORATION |
  7. |                                  THE KROGER CO |
  8. |                          WAL-MART STORES, INC. |
  9. |                             KMART CORPORATION  |
 10. | PROFESSIONAL PHARMACIES INC DBA PLAZA PHARMACY |
     |------------------------------------------------|
 11. |     MADISON HOLDINGS, INC. C/O WORLD FINANCIAL |
 12. |              RESORTS U.S.A. T/A SEASIDE RESORT |
 13. |                  PG INDUSTRIES ATTN JOHN SMITH |
 14. |                BB & T FKA COASTAL FEDERAL BANK |
     +------------------------------------------------+

. stnd_compname firm_name, gen(stn_name stn_dbaname stn_fkaname entitytype attn
> _name)
```

```
. list stn_name stn_dbaname entitytype

     +-----------------------------------------------------+
     |          stn_name            stn_dbaname   entity~e |
     |-----------------------------------------------------|
  1. |              7 11                               INC |
  2. |            AT & T                               INC |
  3. |      DISH NETWORK                              CORP |
  4. |               HVM   EXTENDED STAY HOTELS        LLC |
  5. |         RHEEM MFG                                CO |
     |-----------------------------------------------------|
  6. |         STARBUCKS                              CORP |
  7. |        THE KROGER                                CO |
  8. |   WAL MART STORES                               INC |
  9. |             KMART                              CORP |
 10. |   PROF PHARMACIES           PLZ PHARMACY        INC |
     |-----------------------------------------------------|
 11. |  MADISON HOLDINGS                               INC |
 12. |       RESORTS USA         SEASIDE RESORT            |
 13. |            PG IND                                   |
 14. |            BB & T                                   |
     +-----------------------------------------------------+

. list stn_name stn_fkaname attn_name

     +------------------------------------------------------------+
     |          stn_name            stn_fkaname         attn_name |
     |------------------------------------------------------------|
  1. |              7 11                                          |
  2. |            AT & T                                          |
  3. |      DISH NETWORK                                          |
  4. |               HVM                                          |
  5. |         RHEEM MFG                                          |
     |------------------------------------------------------------|
  6. |         STARBUCKS                                          |
  7. |        THE KROGER                                          |
  8. |   WAL MART STORES                                          |
  9. |             KMART                                          |
 10. |   PROF PHARMACIES                                          |
     |------------------------------------------------------------|
 11. |  MADISON HOLDINGS                          WORLD FINANCIAL |
 12. |       RESORTS USA                                          |
 13. |            PG IND                               JOHN SMITH |
 14. |            BB & T   COASTAL FEDERAL BANK                   |
     +------------------------------------------------------------+
```

3 The stnd_address command

3.1 Syntax

`stnd_address` *varname* [`if` *exp*] [`in` *range*] , `gen(`*newvarname*`)`
[`patpath(`*directory of pattern files*`)`]

3.2 Description

The `stnd_address` command standardizes and parses a string variable specified as a street address into 5 components; `gen`(*newvarname*) is required. The generated outputs are in the following order: (1) street number and street; (2) PO Box; (3) Unit, Apt or STE number; (4) building information; and (5) floor or level information. If a given input cannot be parsed, the original value is recorded in the first field. Similar to `stnd_compname`, `stnd_address` relies on several subcommands and rule-based pattern files being installed. The default directory of the pattern files is /ado/plus/p/. If the pattern files are installed in a different directory, the user needs to specify the directory in the `patpath()` option. If a particular pattern file is not found, the program will display a warning message and the standardizing or parsing step associated with that pattern file will be skipped. The default pattern files are based on U.S. addresses. See Section 4 for details.

3.3 Examples

Analogous to the previous section, we now apply the `stnd_address` command to the street address in the two databases used above. The original variable containing street addresses is streetadd.

```
. use respondent_employers, clear

. list streetadd

     +-------------------+
     |         streetadd |
     |-------------------|
  1. |      ROUTH STREET |
  2. |      P.O. BOX 345 |
  3. |   208 S. AKARD ST |
  4. |                   |
  5. | 508 SW 8TH STREET |
     |-------------------|
  6. |    508 8TH STREET |
  7. |        508 8TH ST |
     +-------------------+

. stnd_address streetadd, gen(add1 pobox unit bldg floor)

. list add1-floor

     +------------------------------------------------+
     |           add1     pobox   unit   bldg   floor |
     |------------------------------------------------|
  1. |       ROUTH ST                                 |
  2. |                  BOX 345                       |
  3. | 208 S AKARD ST                                 |
  4. |                                                |
  5. |  508 SW 8TH ST                                 |
     |------------------------------------------------|
  6. |     508 8TH ST                                 |
  7. |     508 8TH ST                                 |
     +------------------------------------------------+

. use firm_dataset, clear
. list streetadd

     +---------------------------------+
     |                       streetadd |
     |---------------------------------|
  1. |               1722 ROUTH STREET |
  2. |                 P.O. BOX 132160 |
  3. |   9601 SOUTH MERIDIAN BOULEVARD |
  4. |   11525 N. COMMUNITY HOUSE ROAD |
  5. |   1100 ABERNATHY RD NE STE 1400 |
     |---------------------------------|
  6. | 2401 UTAH AVENUE SO., 8TH FLOOR |
  7. |                    1014 VINE ST |
  8. |               508 SW 8TH STREET |
  9. |               3333 BEVERLY ROAD |
 10. |              11 BRIDGEWAY PLAZA |
     |---------------------------------|
 11. |    270 PARK AVENUE, SUITE 1503  |
 12. |               18 W. JIMMIE ROAD |
 13. |                  PO BOX 2706    |
 14. |                         POB 345 |
     +---------------------------------+

. stnd_address streetadd, gen(add1 pobox unit bldg floor)

. list add1-floor

     +-------------------------------------------------------------------+
     |                       add1        pobox       unit   bldg   floor |
     |-------------------------------------------------------------------|
  1. |              1722 ROUTH ST                                        |
  2. |                               BOX 132160                          |
  3. |       9601 S MERIDIAN BLVD                                        |
  4. | 11525 N COMMUNITY HOUSE RD                                        |
  5. |       1100 ABERNATHY RD NE                STE 1400                |
     |-------------------------------------------------------------------|
  6. |            2401 UTAH AVE S                                   FL 8 |
  7. |               1014 VINE ST                                        |
  8. |              508 SW 8TH ST                                        |
  9. |            3333 BEVERLY RD                                        |
 10. |           11 BRIDGEWAY PLZ                                        |
     |-------------------------------------------------------------------|
 11. |                 270 PK AVE                STE 1503                |
 12. |             18 W JIMMIE RD                                        |
 13. |                                 BOX 2706                          |
 14. |                                  BOX 345                          |
     +-------------------------------------------------------------------+
```

4 Options: Specifying alternative pattern files

The stnd_compname and stnd_address commands are wrappers of a sequence of several subcommands. Each subcommand parses or standardizes a string based on its associated rule-based pattern file(s). In general, parsers use the string characters specified in the pattern files to guide how to split the original string variables into two or more variables. Standardizers map a set of strings to their standardized forms. There are some variations across these subcommands. Advanced users may want to specify alternate pattern files, or modify the rules in the existing files for standardizing that is customized for a particular matching project. To do this, users must first understand how these subcommands work, and their dependencies on each other.

The subcommands used for the `stnd_compname` and `stnd_address` commands are listed in order in Tables 3 and 4, respectively. The sequence is critically important as some subcommands and their associated pattern files are conditional on certain characters being removed or standardized in earlier stages. While users may apply any of these subcommands directly, it is not recommended without carefully inspecting its associated pattern file(s).

Table 3: Subcommands used in `stnd_compname`

	Subcommands	Pattern file names
4.1	parsing_namefield	P10_namecomp_patterns.csv
4.2	stnd_specialchar	P21_spchar_specialcases.csv P22_spchar_remove.csv P23_spchar_rplcwithspace.csv
4.3	stnd_entitytype	P30_std_entity.csv
4.4	stnd_commonwrd_name	P40_std_commonwrd_name.csv
4.5	stnd_commonwrd_all	P50_std_commonwrd_all.csv
4.6	stnd_numbers	P60_std_numbers.csv
4.7	stnd_NESW	P70_std_NESW.csv
4.8	stnd_smallwords	P81_std_smallwords_all.csv
4.9	parsing_entitytype	P90_entity_patterns.csv
4.10	agg_acronym	

Table 4: Subcommands used in `stnd_address`

	Subcommands	Pattern file names
4.2	stnd_specialchar	P22_spchar_remove.csv P23_spchar_rplcwithspace.csv
4.5	stnd_commonwrd_all	P50_std_commonwrd_all.csv
4.6	stnd_numbers	P60_std_numbers.csv
4.7	stnd_NESW	P70_std_NESW.csv
4.8	stnd_smallwords	P81_std_smallwords_all.csv P82_std_smallwords_address.csv
4.11	stnd_streettype	P110_std_streettypes.csv
4.12	parsing_pobox	P120_pobox_patterns.csv
4.13	stnd_secondaryadd	P131_std_secondaryadd.csv
4.14	parsing_add_secondary	P132_secondaryadd_patterns.csv

Below we provide details of the required format of pattern files used in the parsing and standardizing subcommands. As shown in Tables 3 and 4, some subcommands are used for both stnd_compname and stnd_address, while others are command-specific. The agg_acronym command removes a space between one-letter words in a string (e.g., "Y M C A" is changed to "YMCA"), and does not rely on a pattern file.

4.1 Parsing commands

The subcommands listed in the tables include four parsers. The stnd_compname command relies on parsing_namefield and parsing_entitytype. The stnd_address command uses parsing_pobox and parsing_add_secondary.

The parsing_namefield command is the first step in the stnd_compname command. It checks if the specified field actually contains more than a single name. Some company listings include both official names and trade names or former names in the same field. Other listings include ATTN or C/O followed by a person name (see examples in Table 2). Applying parsing_namefield to "[Official Name] [keyword] [Alternative Name]" will split the official name from its alternative name without retaining the keyword (e.g., DBA). Each row of the pattern file P10_namecomp_pattern.csv associated with this command consists of two columns: column 1 is a string pattern to search for (keyword); column 2 is the associated name component type. For example, "PROFESSIONAL PHARMACIES INC DBA PLAZA PHARMACY" will be split into "PROFESSIONAL PHARMACIES INC" and "PLAZA PHARMACY".

The parsing_entitytype command works slightly differently as it keeps the word in its associated pattern file and places it under the new entity-type variable. Following the example above, this subcommand further splits "PROFESSIONAL PHARMACIES INC" to "PROFESSIONAL PHARMACIES" and "INC" given that "INC" exists in its pattern file, P90_entity_patterns.csv. This pattern file also consists of two columns. Column 1 is a string pattern containing the search keywords of entity types. Column 2 attempts to limit parsing when keywords are actually a part of the company name.[2] If the string characters in column 2 are found in addition to those in column 1, that parsing will be skipped. It should be noted that this pattern file does not include all possible words for entity types as it is used in the later stage of stnd_compname where some standardizations have been done earlier. For instance, the pattern file only includes "INC" but neither "INCORP" nor "INCORPORATION" because these two words have already been standardized to "INC" in an earlier stage.

The parsing_pobox command parses PO Box information into another field. Each row of its pattern file, P120_pobox_patterns, lists a keyword possibly describing PO Box information (e.g., PO BOX, PO DRAWER, etc). The parsing_add_secondary command parses secondary information often found in the string containing street address into separate fields. Its pattern file, P132_secondaryadd_patterns.csv is more complicated as this command searches over different combinations of address formats. This pattern file consists of 3 columns. Column 1 contains a string pattern to search for, uti-

[2] For example, if a row lists "CO INC, & CO", parsing_entitytype will treat "CO INC" as an entity type only if it does not find "& CO". This avoids parsing "TIFFANY & CO INC" into "TIFFANY &" and "CO INC".

lizing Regular Expressions. Column 2 is the associated information type. For instance, "STNUM_ST_APT" refers to the "street number-street-apt" format. "BLDG_FL" refers to the "building name-floor" format. Columns 3 and 4 contain information on the location of key address components that are used in conjunction with the Regular Expression pattern from column 1.

4.2 Standardizing commands

The `stnd_specialchar` command deals with special characters, using 3 associated pattern files. The `stnd_entitytype`, `stnd_commonwrd_name`, `stnd_commonwrd_all`, `stnd_numbers`, `stnd_NESW` and `stnd_secondaryadd` commands are all based on word substitution. Each uses a single pattern file. The `stnd_smallwords` command is also based on word substitution but only takes an action if that word does not constitute the whole string. It has two associated pattern files: P81_std_smallwords_all.csv is always used; and P82_std_smallwords_address.csv is only used in the `stnd_address` command.

The pattern files associated with the standardizers described above (with the exception of the `stnd_specialchar` subcommand) consist of 2 columns: column 1 contains a string to be substituted (original form) and column 2 contains its standardized form. All default pattern files use a short form of standardization ("STREET" is changed to "ST"; "East" is changed to "E".) Shorter forms are chosen for two reasons. First, abbreviating a word is less risky than expanding a word. For example, expanding "E" to "East" may end up wrongly expanding "JOHN E SMITH" to "JOHN EAST SMITH". Second, these words tend to have small distinguishing power. The longer they are, the more they contribute to a field-similarity score. Most word standardization subcommands rely on STATA's `subinword` command to ensure that the string is not a part of a larger string.[3] This prevents replacing "Eastern Michigan University" with "Eern Michigan University".

The `stnd_specialchar` command standardizes special characters (e.g., ∼ ! #). Characters which tend to be typographical errors are removed. Characters which tend to separate words are replaced with a whitespace.[4] There are 3 associated pattern files. P21_spchar_specialcases.csv is an initial standardization to perform with company names before removing or replacing any special characters. For instance, we may want to replace ".COM" with "DOTCOM" before removing "."; or replace "A+" with "APLUS" before changing "+" to "&". This pattern file is similar to other standardizers listed

[3] `stnd_commonwrd_name`, `stnd_commonwrd_all`, `stnd_numbers`, `stnd_NESW`, `stnd_secondaryadd` and `stnd_smallwords` search for the word specified in their pattern files everywhere within the string. `stnd_entitytype` only searches for the word at the end of the string because its presence in the middle of the string could have other purposes. As an example, "PC" at the end of the string tends to stand for "PROFESSIONAL CORPORATION" but "PC" in the middle is likely to indicate a business related to "PERSONAL COMPUTER".

[4] It does matter whether a character is removed or replaced with a white space. Consider "L.L.BEAN", "LL BEAN" and "LL BEAN,INCORP". Simply removing both "." and "," gives "LLBEAN", "LL BEAN", "LL BEANINCORP". This causes 2 problems: (a) "LLBEAN" will not appear the same as "LL BEAN"; and (b) a pattern file that looks for a word "INCORP" to standardize to "INC" will not find it as the last string contains "LL" and "BEANINCORP".

above where column 1 contains a string to be substituted (original) and column 2 contains its standardized form. It is only relevant for `stnd_compname`.

The pattern files P22_spchar_remove.csv and P23_spchar_rplcwithspace.csv contain characters to be removed, and to be replaced with a whitespace, respectively. The `stnd_specialchar` command itself has an option for characters to be excluded. While `stnd_compname` uses all characters listed in the pattern files, `stnd_address` specifies the program to initially retain "#" and "-", because "#" is often a prefix to apartment numbers and "-" may indicate street numbers (e.g., "179-184").

4.3 Examples

Case 1: A user wants to use the default pattern files in her first run. In the second run, she wants to further standardize the already-standardized variable from the first run. Assume that she has all default pattern files installed in the default directory "c:/ado/plus/p/". In the first run, she applies `stnd_compname` to a variable "orig_name" and specifies the output variables as: "name_stn1" "dba" "fka" "entity" "attn":

```
. stnd_compname orig_name, gen(name_stn1 dba fka entity attn)
```

In the second run, she wants to standardize common words in company names further. She will need to create a new pattern file P40_std_commonwrd_name.csv. Assume she puts this pattern file in "c:/ado/personal/mypattern_pass2/". This directory may contain only this pattern file. In this second run, she applies `stnd_compname` to the standardized variable from the first stage:

```
. stn_compname name_stn1, gen(name_stn2)
>patpath(c:/ado/personal/mypattern_pass2/)
```

The program will display a series of warning messages indicating that some pattern files are not found, but in this case they may be safely ignored as the relevant steps were already accomplished in the first run.

Case 2: A user wants to remove or edit some rules listed in the default pattern files. For example, the user wants to standardize business names and addresses from a database of firms from the United Kingdom. While these U.K. firm listings are in English and share many common word abbreviations as U.S. listings, there are some differences. For instance, U.K. legal entity naming conventions include "Public Limited Corporation" (or "PLC"), "Community Interest Company" (or "CIC"), and "Royal Charter" (or "RC"). To incorporate these patterns, we suggest the user first copy all default pattern files into a different directory, say "c:/ado/personal/UKpatterns". Next, the user would edit the pattern file P30_std_entity.csv to enable the `stnd_compname` program to standardize these words. And to make the program parse these words into an entity type field, the user would edit P90_entity_patterns.csv. Finally, the user would need to specify that `stnd_compname` use pattern files in this directory:

```
. stnd_compname orig_name, gen(name_stn dba fka entity attn)
>patpath(c:/ado/personal/UKpatterns/)
```

In this case, the program should not display any warning messages.

It is also possible to apply these standardizing utilities to non-English business databases, provided the language uses the Roman alphabet. In these cases, however, the user must collect a full set of compatible standardization files that apply to the country-specific business name and address system. In addition to specifying different entity naming conventions, the user will need to update the other pattern files, such as those corresponding to street types, PO Box, and common word abbreviations. For example, in much of Latin America common street types include: "Calle (CLL)", "Camino (CAM)", "Paseo (PSO)", and "Avenida (AVE)". The term "PO Box" is not always used in non-English countries. Mexico uses both "PO Box" and "Casilla de Correos" or "Apartado Postal". Most European countries use different words (e.g., Postfach or PF for German, Boite Postale or BP for French). For these cases, the user would edit the pattern files P110_std_streetetype.csv and P120_pobox_patterns.csv, respectively.[5]

5 The record linkage command: reclink2

5.1 Syntax

```
reclink2 varlist using filename, idmaster(varname) idusing(varname)
   gen(newvarname) [wmatch(match weight list) wnomatch(non-match weight
   list) orblock(varlist) required(varlist) exactstr(varlist)
   exclude(filename) _merge(newvarname) uvarlist(varlist) uprefix(text)
   minscore(#) minbigram(#) manytoone npairs(#) ]
```

5.2 Description

reclink2 performs probabilistic record linkage between two datasets that have no joint identifier necessary for standard merging. The command is an extension of the reclink command originally written by Michael Blasnik. The two datasets are called the "master" and "using" datasets where the "master" dataset is the dataset in memory. For each observation in the "master" dataset, the program tries to find the best match from the "using" dataset based on the specified list of variables, their associated match and non-match weights, and bigram scores.[6] The reclink2 command introduces two new options, manytoone and npairs().

The manytoone option specifies that the command will allow records from the using dataset to be matched to more than one record from the master dataset (a many-to-one

[5] Other examples include the use of the term "AG" in Germany, "SA" in France , and "SpA" in Italy to indicate an incorporated firm. Among many others, users would need to include the terms "Strasse (Str)" in Germany, "Rue (R)" in France, and "Via (V)" in Italy as street types for address standardization. All English directional words also would require translation.

[6] Bigram is an approximate string comparator, which is computed from the ratio of the number of common two consecutive letters of the two strings and their average length minus one. The bigram score used in reclink is a modified version where a pair of strings with up to four common prefix letters also gets extra credit. Other common string comparators include the Jaro-Winkler string comparator, the Levenshtein edit distance, and Q-gram (see Christen (2012) for details).

linking procedure). In the base version of `reclink`, the first step finds and removes perfectly matched pairs from both datasets. Hence, a record in the using dataset that is perfectly matched to a record in the master dataset cannot be subsequently linked to an additional record in the master dataset for which it is an adequate, though not perfect, match. This option effectively allows for sampling with replacement from the using dataset. The examples below illustrate the problem of using a program assuming a one-to-one match on an inherently many-to-one match setting.

The `npairs()` option specifies that the program retain the top n potential matches (above the minimum score threshold) from the using dataset that correspond to a given record in the master dataset. In the base version of `reclink`, only the single candidate with the highest match score is retained as a match – unless the top match scores are identical. Because the approximate string comparator is imperfect, there can be situations where an incorrect record gets a higher score than a correct record, and hence is selected by `reclink` as the best match. Typically, such matches must be removed in the clerical review process, and then in subsequent "passes" the *varlist* and/or weights are altered in an attempt to find the more appropriate match. The `npairs` option allows the user to review and find additional matches that would have otherwise required multiple "passes" and hence multiple stages of clerical review. As there is no increase in computation time for this option, it should help improve efficiency for large-scale matching problems which typically rely on multiple passes for optimal accuracy and coverage.[7]

It should be noted, however, that while the `npairs(`n`)` option allows one to capture a correct match that does not yield the highest score, incorrect matches which pass the minimum score threshold will also be included in the output. Therefore, it is recommended to keep n small (typically 2 or 3) and use the `npairs(`n`)` option in conjunction with the `minscore` option.[8]

If `manytoone` and `npairs` are not specified, `reclink2` produces exactly the same results as `reclink` in most cases.[9] The existing set of options in `reclink` are also retained. See help `reclink` for further explanation of other inputs.

5.3 Examples

Continuing with the example from previous sections, we take the now-standardized datasets of respondent's employer and firm data and illustrate how match results differ across specifications. Our master and using datasets are "respondent_employers_stn.dta" and "firm_dataset_stn.dta", respectively. Besides the standardized name (stn_name), street address(add1), and PO box (pobox) variables, the matching will also use the city

[7] The computation time is unaltered because `reclink` must compute scores for all pair-wise record combinations regardless of whether multiple pairs are retained as output.

[8] In an extreme case, if n is infinity and `minscore minbigram` are zero, all candidates which meet the criteria of the blocking strategy will be output.

[9] `reclink2` also corrects for several minor bugs in the original program such as preventing the `required()` blocking on missing values.

and state variables.[10]

In this first example, we attempt to match via the default one-to-one matching. Hence, the program will output one potential match (the maximum score per record) provided the pair-similarity score is above the default minimum of .6. We specify the variable name containing the generated scores as “rlsc”.

```
. use respondent_employers_stn, clear
. reclink2 stn_name add1 pobox city state using firm_dataset_stn, idm(rid) idu(
> firm_id) wmatch(10 8 6 5 5) gen(rlsc)
1 perfect matches found
Added: firm_id= identifier from firm_dataset_stn   rlsc = matching score
Observations:  Master N = 7    firm_dataset_stn N= 14
  Unique Master Cases: matched = 5 (exact = 1), unmatched = 2
. sort rid
. list rid stn_name add1 Ustn_name Uadd1 rlsc, sep(4) noobs

  +--------------------------------------------------------------------------------+
  | rid          stn_name             add1          Ustn_name           Uadd1   rlsc |
  |--------------------------------------------------------------------------------|
  |   1              7 11         ROUTH ST               7 11   1722 ROUTH ST  0.978 |
  |   2            BT & T                              AT & T                  0.944 |
  |   3            AT & T   208 S AKARD ST             AT & T                  0.826 |
  |   4            KROGER                              KROGER    1014 VINE ST  0.893 |
  |--------------------------------------------------------------------------------|
  |   5  WAL MART STORES     508 SW 8TH ST  WAL MART STORES   508 SW 8TH ST  1.000 |
  |   6           WLAMART        508 8TH ST                                       . |
  |   7           WALMART        508 8TH ST                                       . |
  +--------------------------------------------------------------------------------+
```

There is one obvious problem here. The program does not find a match for the employers of respondents `rid`#6 and `rid`#7 despite the existence of a record of Wal-Mart with a similar address in the firm dataset. This is an inherent feature of the one-to-one matching assumption of `reclink` when perfectly matched records exist. The employer of `rid`#5 appears to match perfectly with the firm record for Wal-Mart, and hence this firm record cannot be subsequently matched with the other respondents identifying Wal-Mart. (In this case no other pairwise score reached the minimum score threshold of 0.6, but if the threshold is set to a lower value, it could show false matches for these records).[11]

Next, we call the same `reclink2` command, but specify the `manytoone` option:

```
. use respondent_employers_stn, clear
. reclink2 stn_name add1 pobox city state using firm_dataset_stn, idm(rid) idu(
> firm_id) wmatch(10 8 6 5 5) gen(rlsc) many
1 perfect matches found
Added: firm_id= identifier from firm_dataset_stn   rlsc = matching score
Observations:  Master N = 7    firm_dataset_stn N= 14
  Unique Master Cases: matched = 7 (exact = 1), unmatched = 0
```

[10] City and state names should also be standardized so their formats are consistent in both datasets. That task, however, is easier relative to standardizing company names and street addresses and is not illustrated here.

[11] `reclink` does not assume a one-to-one matching in a strict sense. As shown in this example, the record AT & T from the firm dataset can be used twice for `rid`#2 and `rid`#3 because neither constitutes a perfectly matched pair.

```
. sort rid
. list rid stn_name add1 Ustn_name Uadd1 rlsc, sep(4) noobs

  +---------------------------------------------------------------------------+
  | rid          stn_name           add1          Ustn_name           Uadd1     rlsc |
  |---------------------------------------------------------------------------|
  |   1              7 11       ROUTH ST               7 11   1722 ROUTH ST    0.978 |
  |   2            BT & T                            AT & T                    0.944 |
  |   3            AT & T   208 S AKARD ST           AT & T                    0.826 |
  |   4            KROGER                            KROGER    1014 VINE ST    0.893 |
  |---------------------------------------------------------------------------|
  |   5   WAL MART STORES    508 SW 8TH ST   WAL MART STORES   508 SW 8TH ST    1.000 |
  |   6           WLAMART       508 8TH ST   WAL MART STORES   508 SW 8TH ST    0.638 |
  |   7           WALMART       508 8TH ST   WAL MART STORES   508 SW 8TH ST    0.943 |
  +---------------------------------------------------------------------------+
```

Now we see that the `rid` #6 and #7 are matched with the correct record from the firm dataset. Next, we draw attention to `rid`#2 where the self-reported employer BT & T is incorrectly matched to AT & T from the firm dataset. With these small datasets, we know that the true match is firm #14 but a mis-spelling of the employer name (BT&T rather than BB&T) complicates the task. The next example demonstrates one potential strategy for matching this record in a single run using the **`npairs`** option:

```
. use respondent_employers_stn, clear
. reclink2 stn_name add1 pobox city state using firm_dataset_stn, idm(rid) idu(
> firm_id) wmatch(10 8 6 5 5) gen(rlsc) many npairs(2)
1 perfect matches found

Added: firm_id= identifier from firm_dataset_stn   rlsc = matching score
Observations:  Master N = 7    firm_dataset_stn N= 14
  Unique Master Cases: matched = 7 (exact = 1), unmatched = 0
. sort rid
. list rid stn_name add1 Ustn_name Uadd1 rlsc, sep(4) noobs

  +---------------------------------------------------------------------------+
  | rid          stn_name           add1          Ustn_name           Uadd1     rlsc |
  |---------------------------------------------------------------------------|
  |   1              7 11       ROUTH ST               7 11   1722 ROUTH ST    0.978 |
  |   2            BT & T                            AT & T                    0.944 |
  |   2            BT & T                            BB & T                    0.937 |
  |   3            AT & T   208 S AKARD ST           AT & T                    0.826 |
  |---------------------------------------------------------------------------|
  |   3            AT & T   208 S AKARD ST           BB & T                    0.659 |
  |   4            KROGER                            KROGER    1014 VINE ST    0.893 |
  |   5   WAL MART STORES    508 SW 8TH ST   WAL MART STORES   508 SW 8TH ST    1.000 |
  |   6           WLAMART       508 8TH ST   WAL MART STORES   508 SW 8TH ST    0.638 |
  |---------------------------------------------------------------------------|
  |   7           WALMART       508 8TH ST   WAL MART STORES   508 SW 8TH ST    0.943 |
  +---------------------------------------------------------------------------+
```

Specifying **`npair(2)`** tells the program to retain the top 2 matches which satisfy the score threshold. The result now shows that for `rid`#2 and #3, two candidates meet this score criteria. Here we see that the correct match for `rid`#2 is indeed the 2nd highest match score for that record. The **`npairs`** option enables the researcher to catch this alternate match within one **`reclink2`** procedure. Typically, the researcher would be required to reject the high-score match for this record in the clerical review stage, and then attempt to utilize an alternative matching specification (different set of variables or weighting schemes) in an additional **`reclink`** pass to capture the correct match.

We now save this post-reclink2 dataset as "reclink_forreview.dta". After the machine generates the matched pairs, there is still the work of approving or rejecting each matched pair. To a large degree, this requires the input of human reviewers. The next section discusses the clerical review utility that expedites this time-intensive task.

6 The clrevmatch command

After a program generated matched pairs in the linking step, users typically were required to export results to a different program, re-format, record manual reviews, and then import back into STATA. Records with accepted matches were then saved separately and the matching process continued for records without accepted matches. This set of steps can become particularly cumbersome in a large, multi-stage linking project. The clrevmatch program creates a seamless reviewing tool that is efficient, flexible, and user-friendly.

6.1 Syntax

```
clrevmatch using filename, idmaster(varname) idusing(varname)
   varM(varlist) varU(varlist) clrev_result(newvarname)
   clrev_note(newvarname) [reclinkscore(varname) rlscoremin(#)
   rlscoremax(#) rlscoredisp(on|off) fast clrev_label(label) nobssave(#)
   replace newfilename(newfilename) saveold]
```

6.2 Description

clrevmatch provides an interactive tool to assist in the clerical review of matched pairs generated from a record linkage program (e.g., reclink, reclink2). The program displays a potential match such that the pair of records constituting the match are easily assessed by the user. The user then inputs a clerical review indicator on whether the matched pair is accepted, rejected, or left as uncertain. Alternative labels can be specified. clrevmatch also checks if multiple matches are found for a given record in the master dataset. If this is the case, the program first indicates how many matches exist for that record and then displays all potential candidates. The user can then assign a clerical decision for each candidate. The required inputs are explained below.

filename specifies the name of the dataset to be reviewed. This dataset must contain machine-generated matched pairs from two datasets (called master and using datasets) along with their record identifiers, idmaster() and idusing(). The user must either specify replace to save the clerical decisions into the existing dataset, or newfilename() to generate results in a new file.

varM() and varU() specify the set of variables in the master and using datasets that will be displayed during the review process. The user can specify not only the set of variables used in the matching process, but also other existing variables in the dataset which may help assess the candidates.

clrev_result() specifies a (new) variable name to record the user's clerical review input. clrev_note() specifies a (new) variable name for the user to enter a note associated with each pair of records. Because the clerical review process is often a lengthy and time-consuming component of the record linking process, this program periodically saves the results as the user progresses. If the reviewer does not finish reviewing the whole dataset in one session, she can continue to do her work in the next session by entering the same clrev_result() and clrev_note() variables. A different reviewer may want to use different variable names for these two variables.

6.3 Options

reclinkscore() specifies the variable containing the machine-generated score from the matching step to enable other score-related options.

rlscoredisp() is set to "on" by default, such that the display includes the machine-generated score from the reclinkscore() option. In some situations, the user may not want the score to influence the clerical review decision. By setting rlscoredisp(off), the score will not be displayed.

rlscoremin() and rlscoremax() options allow the user to specify the range of machine-generated scores so that only those pairs matching the specified criteria will appear for clerical review.[12]

fast is an option to help speed up the review process. By default, the reviewer is asked to confirm the clerical input, and then the program provides the opportunity for the reviewer to enter any additional notes for later review or editing. Specifying fast will cause the program to skip these steps.

clrev_label() allows the user to specify their own labels for the clerical review results. By default, the program asks for the reviewer to enter 0 for "not a match", 1 for "maybe a match", 2 for "very likely a match", and 3 "definitely a match". The user can specify their own label using STATA's label format. For example, an alternative label could be a simpler one "0 "not match" 1 "match"" or a more specific one "1 "only names matched" 2 "only addresses matched" 3 "both matched" 4 "neither matched"". The program will attach the specified label to the clrev_result() variable.

nobssave() specifies how often the program will save the results. By default, the program will save the file after every 5 records. The option saveold will save the dataset in an older STATA format.

[12] The default values for rlscoremin() and rlscoremax() are set to 0 and 1, respectively. This is based on the range of scores generated by the reclink algorithm. If clrevmatch is to be used with a dataset generated by some other probabilistic record linkage algorithm, rlscoremin() and rlscoremax() should be set based on the range of scores generated by that algorithm.

6.4 Examples

We continue with the matching example from previous sections. Here, we demonstrate how `clrevmatch` can assist the user in the review and clerical edits of the matches after the record linking step. In the example below, we will review the file "reclinking_forreview.dta" saved in the previous step. Recall that the pair-similarity score variable in that dataset is "rlsc". We specify two new variables to contain the clerical result and note as "crev" and "crnote". This review will use the default review label.

```
. clrevmatch using reclinking_forreview,  idm(rid) idu(firm_id) varM(stn_name a
> dd1 pobox city state) varU(Ustn_name Uadd1 Upobox Ucity Ustate) reclinkscore(
> rlsc) clrev_result(crev) clrev_note(crnote) replace

Total # pairs to be reviewed = 9
----------------------------------------------------------------------
 File 1
 ------
 stn_name:  7 11
 add1:      ROUTH ST
 pobox:
 city:      DALLAS
 state:     TX
----------------------------------------------------------------------
 File 2
 ------
 Ustn_name: 7 11
 Uadd1:     1722 ROUTH ST
 Upobox:
 Ucity:     DALLAS
 Ustate:    TX
                                              match score: .987
______________________________________________________________________

How would you describe the pair?
clrevlbl:
           0 not a match
           1 maybe a match
           2 very likely a match
           3 definitely a match
please enter a clerical review indicator:.
```

At this point the user would input a clerical review label for this potential match (most likely a 3 for "definitely a match"), and then as `fast` is not specified, the program would offer the option to go back to change the answer or to enter a manual note (this display is omitted.) Next, the program will move to the second record.

```
# pairs left to be reviewed = 8/9
 There are 2 potential candidates for this record.
 All candidate profiles will be first displayed.
 We will then ask you to describe the match quality of each candidate.
----------------------------------------------------------------------
 File 1
 ------
 stn_name:  BT & T
 add1:
 pobox:     BOX 345
 city:      DALLAS
 state:     TX
----------------------------------------------------------------------
 File 2
 ------
 candidate # 1
 Ustn_name: AT & T
 Uadd1:
 Upobox:    BOX 132160
 Ucity:     DALLAS
 Ustate:    TX
                                         match score: .944
 ----
 candidate # 2
 Ustn_name: BB & T
 Uadd1:
 Upobox:    BOX 345
 Ucity:     DALAS
 Ustate:    TX
                                         match score: .937
----------------------------------------------------------------------

How would you describe candidate # 1?
clrevlbl:
           0 not a match
           1 maybe a match
           2 very likely a match
           3 definitely a match
please enter a clerical review indicator:. 0
How would you describe candidate # 2?
clrevlbl:
           0 not a match
           1 maybe a match
           2 very likely a match
           3 definitely a match
please enter a clerical review indicator:. 2
```

In this case, there are two candidates with the score above the threshold for `rid` #2. The program first displays all candidates and asks the reviewer to judge each candidate. The user can now throw out the 1st match pertaining to the `rid` #2 record and approve the 2nd match. Recall that this record has two candidates to be reviewed because we have used the `npairs(2)` option in `reclink2`. If we were to use the baseline `reclink`, the matched file to be reviewed would contain only the first candidate. The reviewer would then reject this candidate, and another run of record linkage would be needed to find the match for `rid` #2.

To specify alternative labels, the user can set up a local macro in the STATA label format and enter this variable in the `clrev_label()` option, e.g.,
. local mylabel “0 “not match” 1 “match””
and then add the term `clrev_label(‘mylabel’)` in the clrevmatch command line.

In practice, datasets with machine-generated pairs may contain several thousand or even millions of pairs. Researchers may accept a small margin of error by reviewing only pairs within some middle range of scores. For example, one may specify `rlscoremin(.8)` and `rlscoremax(.97)` and assume that all pairs with scores higher than .97 are true matches.

7 Conclusions

We have provided a new set of STATA commands which facilitate several stages of probabilistic record linkage. The `stnd_compname` and `stnd_address` commands help researchers properly prepare the data files before linking them. The commands are flexible in the sense that advanced users can modify the default pattern files. The `reclink2` command is a generalized version of the existing record linkage command (`reclink`). This new command introduces an option for many-to-one linking and an option to output more than one potential matched candidate. Finally, the `clrevmatch` helps researchers interactively review the generated matched pairs without exporting and importing to another software. These utilities can also be used independently. For example, the `stnd_compname` and `stnd_address` commands may be used in a single dataset to standardize the record formats before applying the built-in `duplicates` command. The `reclink2` and `clrevmatch` commands are not limited to linking firm databases. They can be used with other types of databases such as those containing lists of patients, customers, or benefit plans. The `clrevmatch` command can also be used with any dataset containing potential match-paired records.

8 References

Abowd, J., and M. Stinson. 2013. Estimating measurement error in annual job earnings: A comparison of survey and administrative data. *Review of Economics and Statistics* 95: 1451–1467.

Agrawal, A., and P. Tambe. 2013. Private Equity, Technological Investment, and Labor Outcomes. Available at SSRN: http://ssrn.com/abstract=2286802.

Blasnik, M. 2010. RECLINK: Stata module to probabilistically match records. Statistical Software Components.

Christen, P. 2012. *Data Matching: Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer.

Fellegi, I., and A. Sunter. 1969. A Theory of Record Linkage. *Journal of the American Statistical Association* 64: 1183–1210.

9 Acknowledgments

The Summer Working-group for Employer List Linking(SWELL)– a collaboration between researchers at the US Census Bureau, University of Michigan, and Cornell University – provided several useful suggestions for the stnd_compname command. Ann Rodgers contributed to the agg_acronym.ado subcommand. We gratefully acknowledge support by the Alfred P. Sloan Foundation for the University of Michigan's Census-Enhanced HRS (CenHRS) Project and by the National Science Foundation (SES1131500) for the University of Michigan node of the NSF-Census Research Network (NCRN).