The Stata Journal

The *Stata Journal* publishes reviewed papers together with shorter notes or comments, regular columns, book reviews, and other material of interest to Stata users. Examples of the types of papers include 1) expository papers that link the use of Stata commands or programs to associated principles, such as those that will serve as tutorials for users first encountering a new field of statistics or a major new technique; 2) papers that go "beyond the Stata manual" in explaining key features or uses of Stata that are of interest to intermediate or advanced users of Stata; 3) papers that discuss new commands or Stata programs of interest either to a wide spectrum of users (e.g., in data management or graphics) or to some large segment of Stata users (e.g., in survey statistics, survival analysis, panel analysis, or limited dependent variable modeling); 4) papers analyzing the statistical properties of new or existing estimators and tests in Stata; 5) papers that could be of interest or usefulness to researchers, especially in fields that are of practical importance but are not often included in texts or other journals, such as the use of Stata in managing datasets, especially large datasets, with advice from hard-won experience; and 6) papers of interest to those who teach, including Stata with topics such as extended examples of techniques and interpretation of results, simulations of statistical concepts, and overviews of subject areas.

The *Stata Journal* is indexed and abstracted by *CompuMath Citation Index*, *Current Contents/Social and Behavioral Sciences*, *RePEc: Research Papers in Economics*, *Science Citation Index Expanded* (also known as *SciSearch*), *Scopus*, and *Social Sciences Citation Index*.

For more information on the *Stata Journal*, including information for authors, see the webpage

http://www.stata-journal.com

Subscriptions are available from StataCorp, 4905 Lakeway Drive, College Station, Texas 77845, telephone 979-696-4600 or 800-STATA-PC, fax 979-696-4601, or online at

http://www.stata.com/bookstore/sj.html

Subscription rates listed below include both a printed and an electronic copy unless otherwise mentioned.

U.S. and Canada		Elsewhere	
Printed & electronic		**Printed & electronic**	
1-year subscription	$115	1-year subscription	$145
2-year subscription	$210	2-year subscription	$270
3-year subscription	$285	3-year subscription	$375
1-year student subscription	$ 85	1-year student subscription	$115
1-year institutional subscription	$345	1-year institutional subscription	$375
2-year institutional subscription	$625	2-year institutional subscription	$685
3-year institutional subscription	$875	3-year institutional subscription	$965
Electronic only		**Electronic only**	
1-year subscription	$ 85	1-year subscription	$ 85
2-year subscription	$155	2-year subscription	$155
3-year subscription	$215	3-year subscription	$215
1-year student subscription	$ 55	1-year student subscription	$ 55

Back issues of the *Stata Journal* may be ordered online at

http://www.stata.com/bookstore/sjj.html

Individual articles three or more years old may be accessed online without charge. More recent articles may be ordered online.

http://www.stata-journal.com/archives.html

The *Stata Journal* is published quarterly by the Stata Press, College Station, Texas, USA.

Address changes should be sent to the *Stata Journal*, StataCorp, 4905 Lakeway Drive, College Station, TX 77845, USA, or emailed to sj@stata.com.

The *Stata Journal* (ISSN 1536-867X) is a publication of Stata Press. Stata, Stata, Stata Press, Mata, Mata, and NetCourse are registered trademarks of StataCorp LP.

The Stata Journal (2015)
15, Number 3, pp. 672–697

Record linkage using Stata: Preprocessing, linking, and reviewing utilities

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

Aaron Flaaen
Division of Research and Statistics
Federal Reserve Board of Governors
Washington, DC
aaron.b.flaaen@frb.gov

Abstract. In this article, we describe Stata utilities that facilitate probabilistic record linkage—the technique typically used for merging two datasets with no common record identifier. While the preprocessing 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 to improve the match quality when linking. The `reclink2` command is a generalized version of Blasnik's `reclink` (2010, Statistical Software Components S456876, Department of Economics, Boston College) that allows for many-to-one matching. 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 (for example, Excel), inputting clerical reviews, and importing back into Stata, one can use the `clrevmatch` tool to conduct all of these steps within Stata. This helps improve the speed and flexibility of matching, which often involves multiple runs.

Keywords: dm0082, reclink2, clrevmatch, reclink, stnd_compname, stnd_address, 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 (for example, ReferenceUSA, Securities and Exchange Commission filings, LexisNexis, the Business Register of the U.S. Census Bureau). This information can be collected at several different levels of aggregation, including a plant (establishment or branch), firm, or tax-identifying unit. Many household surveys also ask respondents to report name, address, and other characteristics of their employers. Because these databases contain specific information based on the purpose of their construction, researchers often need to combine data from multiple sources to facilitate their analyses. 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's history database to assess how private equity acquisitions affect labor market outcomes of workers.

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

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

Respondent ID	Name	Street address
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

1. The examples in this article contain no actual respondent data from any survey.

Table 2. An example of records from a firm database

Firm ID	Name	Street address
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

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. Implementing Stata's `merge` command using name and street address will yield only one match (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 are 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) preprocessing, 2) probabilistic linking, and 3) clerical review of machine-generated matched pairs. The preprocessing step ensures that both datasets have the same formats and that chosen fields are meaningful in matching. Typically, this first step consists of two substeps: 1) parsing a field into the relevant subcomponents and 2) standardizing common character strings. This often helps researchers achieve higher quality matches in the linking step. For example, consider the employer of respondent #2 in table 1. Without preprocessing, firm #2 `"AT&T INC."` (an incorrect match) in table 2 will look more similar to `"BT&T INC."` than `"BB & T FKA COASTAL FEDERAL BANK"` (firm #14 in table 2, 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 whitespace

before and after the & character. Preprocessing to parse entity types and alternative names into separate fields and then 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 (for example, standardized name, standardized address) to input into a probabilistic matching algorithm. For each record from the first dataset, the algorithm selects candidates from the second dataset. These candidates may include all records from the second dataset or may be selected based on certain criteria (for example, 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 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.

In this article, we introduce a set of utilities that facilitates the preprocessing and clerical review steps of record linking. We also briefly explain 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 article, although actual record-linkage tasks often involve very large databases. In sections 2 and 3, we 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 with the commands. In section 4, we explain how advanced users can modify these pattern files to construct specialized preprocessing rules for individual matching exercises. In section 5, we describe the new `reclink2` command. 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, as in our last example where multiple respondents may work for the same employer. Other examples include matching establishments with firms and matching customer location of sale with establishment directories. Finally, in section 6, we explain the `clrevmatch` command, which is an interactive tool that allows 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 (for example, Excel), input clerical reviews, and then import back into Stata.

2 The stnd_compname command

2.1 Syntax

`stnd_compname` *varname* [*if*] [*in*], `gen(`*newvarnames*`)`
[`patpath(`*directory_of_pattern_files*`)`]

2.2 Description

The `stnd_compname` command standardizes and parses a string variable containing company names into five components; `gen(`*newvarnames*`)` is required. The generated outputs are in the following order: 1) official name; 2) doing-business-as (DBA) name; 3) 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 ancillary 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 Options

`gen(`*newvarnames*`)` generates five variables corresponding to components of *varname*. `gen()` is required.

`patpath(`*directory_of_pattern_files*`)` specifies the directory of the pattern files.

2.4 Examples

The following examples apply `stnd_compname` to the company names listed in the introduction section. The `respondent_employers.dta` 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)
stnd_compname version p1.3
pattern files from ~/ado/plus/p/
. 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		

`firm_dataset.dta` 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)
stnd_compname version p1.3
pattern files from ~/ado/plus/p/
```

```
. 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.	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.	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*] [*in*], `gen`(*newvarnames*)
[`patpath`(*directory_of_pattern_files*)]

3.2 Description

The `stnd_address` command standardizes and parses a string variable specified as a street address into five components; `gen(`*newvarnames*`)` is required. The generated outputs are in the following order: 1) street number and street; 2) post office (P.O.) box; 3) unit, apartment, or suite 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. Like `stnd_compname`, `stnd_address` relies on several subcommands and ancillary 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 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 pattern file will be skipped. The default pattern files are based on U.S. addresses. See section 4 for details.

3.3 Options

`gen(`*newvarnames*`)` generates five variables corresponding to components of *varname*. `gen()` is required.

`patpath(`*directory_of_path_name*`)` specifies the directory of the pattern files.

3.4 Examples

As in the previous section, we now apply the `stnd_address` command to the street addresses in the two databases used previously. 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)
stnd_address version p1.3
pattern files from ~/ado/plus/p/
```

```
. 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)
stnd_address version p1.3
pattern files from ~/ado/plus/p/
```

```
. 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 files. 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 alternative pattern files or modify the rules in the existing files for standardizers that are customized for a particular matching project. To do this, users must first understand how these subcommands work and how they depend 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 because 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 the associated pattern files.

Table 3. Subcommands used in `stnd_compname`

	Subcommands	Pattern filenames
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 filenames
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 (for example, `"Y M C A"` is changed to `"YMCA"`), and it 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 whether the specified field actually contains more than one 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's 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 (for example, `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), and 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 because 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"` into `"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, and column 2 attempts to limit parsing when keywords are actually a part of the company name. If the string characters in column 2 are found in addition to those in column 1, that parsing will be skipped.[2] Note that this pattern file does not include all possible words for entity types, because it is used in the later stage of `stnd_compname`, where some standardizations have been done earlier. For instance, the pattern file includes only `"INC"` and not `"INCORP"` or `"INCORPORATION"`, because these two words have already been standardized to `"INC"` in an earlier stage.

The `parsing_pobox` command parses P.O. box information into another field. Each row of its pattern file, `P120_pobox_patterns`, lists a keyword possibly describing P.O. box information (`PO BOX`, `PO DRAWER`, etc). The command `parsing_add_secondary` parses secondary information often found in the string containing the street address into separate fields. Its pattern file, `P132_secondaryadd_patterns.csv`, is more complicated because this command searches over different combinations of address formats. This pattern file consists of three columns. Column 1 contains a string pattern to search for, using regular expressions. Column 2 is the associated information type (for instance, `"STNUM_ST_APT"` refers to the "street number-street-apt" format, and `"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 with the regular expression pattern from column 1.

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"`.

4.2 Standardizing commands

The `stnd_specialchar` command deals with special characters, using three 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 one pattern file. The `stnd_smallwords` command is also based on word substitution but takes an action only if that word does not constitute the whole string. It has two associated pattern files: `P81_std_smallwords_all.csv` (which is always used) and `P82_std_smallwords_address.csv` (which is used only in the `stnd_address` command).

The pattern files associated with the standardizers described above (with the exception of the `stnd_specialchar` subcommand) consist of two 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 (for example, ∼, !, and #). Characters that tend to be typographical errors are removed. Characters that tend to separate words are replaced with a whitespace.[4] There are three 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 above, where column 1 contains a string to be substituted (original) and column 2 contains its standardized form. It is relevant for `stnd_compname` only.

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

3. `stnd_commonwrd_name`, `stnd_commonwrd_all`, `stnd_numbers`, `stnd_nesw`, `stnd_secondaryadd`, and `stnd_smallwords` search everywhere within the string for the word specified in their pattern files. `stnd_entitytype` searches for only the word at the end of the string because its presence in the middle of the string could have other purposes. For 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 whitespace. Consider `"L.L.BEAN"`, `"LL BEAN"`, and `"LL BEAN,INCORP"`. Simply removing both "." and "," gives `"LLBEAN"`, `"LL BEAN"`, and `"LL BEANINCORP"`. This causes two 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 because the last string contains `"LL"` and `"BEANINCORP"`.

`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 (for example, "179-184").

4.3 Examples

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

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

In the second run, the user wants to further standardize common words in company names. The user must create a new pattern file, `P40_std_commonwrd_name.csv`. Assume the user puts this pattern file in `c:/ado/personal/mypattern_pass2/`. This directory may contain only this pattern file. In this second run, the user 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 warning messages indicating that some pattern files are not found. In this case, however, they may be safely ignored because 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 (UK). While these UK firm listings are in English and share many common word abbreviations as U.S. listings, there are some differences. For instance, UK 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. 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 the 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 must update the other pattern files, such as those corresponding to street types, P.O. box, and common word abbreviations. For example, in much of Latin America, common street types include *calle* (or CLL), *camino* (or CAM), *paseo* (or PSO), and *avenida* (or AVE). The term "P.O. box" is not always used in non-English countries. Mexico uses both "P.O. box" and *casilla de correos* or *apartado postal*. Most European countries use different words (for example, *postfach* [or PF] for Germany and *boite postale* [or BP] for France). For these cases, the user would edit the pattern files `P110_std_streetetype.csv` and `P120_pobox_patterns.csv`.[5]

5 The reclink2 command

5.1 Syntax

`reclink2` *varlist* `using` *filename*`,` `idmaster(`*varname*`)` `idusing(`*varname*`)`
`gen(`*newvarname*`)` [`wmatch(`*match_weight_list*`)` `wnomatch(`*nonmatch_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 Blasnik (2010). 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 nonmatch weights, and their bigram scores.[6] The `reclink2` command introduces two new options, `manytoone` and `npairs()`. All other options were taken from the original `reclink` command (Blasnik 2010).

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 must include the terms *strasse* (or Str) in Germany, *rue* (or R) in France, and *via* (or 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 in 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 distance (edit distance), and Q-gram (see Christen [2012] for details).

5.3 Options

`idmaster(`*varname*`)` specifies the name of a variable in the master dataset that uniquely identifies the observations. This variable is used to track observations. If a unique identifier does not exist, one can be created using `generate idmaster=_n`. The `idmaster()` option is required.

`idusing(`*varname*`)` specifies the name of a variable in the using dataset that uniquely identifies the observations analogous to `idmaster()`. `idusing()` is required.

`gen(`*newvarname*`)` specifies the name of a new variable created by `reclink2` to store the matching scores (scaled 0–1) for the linked observations. `gen()` is required.

`wmatch(`*match_weight_list*`)` specifies the weights given to matches for each variable in *varlist*. Each variable requires a *weight*. The default is `wmatch(1)`. *weight*s must be ≥ 1 and are typically integers from 1–20. The values should reflect the relative likelihood of a variable match indicating a true observation match. For example, a `name` variable will often have a large *weight* such as 10, but a `city` variable, where many duplicates are expected, may have a *weight* of just 2.

`wnomatch(`*nonmatch_weight_list*`)` specifies the *weight*s given to mismatches for each variable in the *varlist*. These *weight*s are analogous to `wmatch()`s but instead reflect the relative likelihood that a mismatch on a variable indicates that the observations do not match—a small value indicates that mismatches are expected even if the observations truly match. A variable such as telephone number may have a large `wmatch()` but a small `wnomatch()`, because matches are unlikely to occur randomly, but mismatches may be fairly common because of changes in phone numbers over time or multiple phone numbers owned by the same person or entity.

`orblock(`*varlist*`)` is used to speed up the record linkage by providing a method for selecting only subsets of observations from the using dataset to search for matches. Only observations that match on at least one variable in the or-block are examined. Or-blocking on the full *varlist* is the default behavior if four or more variables are specified. This default can be overridden by specifying `orblock(none)`, which is advised if all the variables are expected to be unique. New variables are sometimes created in the master and using datasets to assist with or-blocking, such as initials of first and last names, street numbers extracted from addresses, and telephone area codes. Or-blocking can dramatically improve the speed of `reclink2`.

`required(`*varlist*`)` specifies one or more variables that must match exactly for the observation to be considered a match. The variables must also be in the main *varlist* and are included in the matching score. `required()` could have been named `andblock()` to make its function clear in relation to `orblock()`.

`exactstr(`*varlist*`)` specifies one or more string variables where the bigram string comparator is not used to assess the degree of agreement, but instead, the agreement is simply 0 or 1.

`exclude(`*filename*`)` specifies the name of a file that contains previously matched observations, providing a convenient way to use `reclink` repeatedly with different specifications. The `exclude()` file must include the variables specified in `idmaster()` and `idusing()`. Any observation with nonmissing values for ID variables is considered matched and is excluded from the datasets for the current matching. Results from each run of `reclink` can be appended together and specified as the `exclude()` file. This approach can speed up matching by starting with a more restrictive `orblock()` setting and `required()` specifications that work quickly, followed by a more exhaustive and slower search for the more difficult observations.

`_merge(`*newvarname*`)` specifies the name of the variable that will mark the source of each observation. The default is `_merge(_merge)`.

`uvarlist(`*varlist*`)` allows the using dataset to have different variable names than the master dataset for the variables to be matched. If specified, `uvarlist()` must have the same number of variables in the same order as the master *varlist*.

`uprefix(`*text*`)` changes the prefix used for renaming the variables in the matching *varlist* that are brought into the master dataset from the using dataset. The default is `uprefix(U)`. For example, if the matching variables are `name` and `address`, then the resulting dataset will have variables `Uname` and `Uaddress` added from the using dataset for the matching observations.

`minscore(#)` specifies the minimum overall matching score value (0–1) used to declare two observations a match. The default is `minscore(0.6)`. Observations in the using dataset are merged into the master dataset only if they have a match score $\geq$ `minscore()` and are the highest match score in the using dataset. Lower values of `minscore()` will expand the number of matches but may lead to more false matches.

`minbigram(#)` specifies the bigram value needed to declare two strings as possibly matched. The default is `minbigram(0.6)`. Each raw bigram score is transformed into match and nonmatch weight multipliers that vary from 0 to 1, with a sharp change at `minbigram()`. A higher value of `minbigram()` may be useful when matching longer strings.

`manytoone` specifies that `reclink2` will allow records from the using dataset to be matched to multiple records from the master dataset (a many-to-one 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.

`npairs(#)` specifies that the program retain the top # 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 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, an incorrect

record sometimes gets a higher score than a correct record and is selected by `reclink` as the best match. Typically, such matches must be removed during clerical review; then, in subsequent "passes", the *varlist* and weights are altered in an attempt to find the more appropriate match. `npairs()` allows the user to review and find additional matches that would have otherwise required multiple "passes" and, hence, multiple stages of clerical review. Because there is no increase in computation time for `npairs()`, it should help improve efficiency for large-scale matching problems that typically rely on multiple passes for optimal accuracy and coverage.[7] Note, however, that while `npairs(#)` can capture a correct match that does not yield the highest score, incorrect matches that pass the minimum-score threshold will also be included in the output. Therefore, we recommend that users keep # small (typically 2 or 3) and use `npairs(#)` in conjunction with `minscore()`.[8]

If `manytoone` and `npairs()` are not specified, `reclink2` produces exactly the same results as `reclink` in most cases.[9]

5.4 Examples

Using the example from previous sections, we take the now standardized datasets of a 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 P.O. box (`pobox`) variables, the matching will also use the `city` 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 0.6. 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
```

7. The computation time is unaltered because `reclink` must compute scores for all pairwise record combinations regardless of whether multiple pairs are retained as output.
8. In an extreme case, if # is infinity and `minscore()` and `minbigram()` are zero, all candidates that 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.
10. City and state names should also be standardized so that 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.

```
. 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, so 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) manytoone
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
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

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.

Now, we see that the `rid` #6 and `rid` #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 misspelling 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 one 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 `npairs(2)` tells the program to retain the top two matches that satisfy the score threshold. The result now shows that for `rid` #2 and `rid` #3, two candidates meet this score criterion. Here we see that the correct match for `rid` #2 is indeed the second-highest match score for that record. The `npairs()` option enables the researcher to catch this alternative match within one `reclink2` procedure. Typically, the researcher would be required to reject the high-score match for this record during clerical review and then attempt to use an alternative matching specification (a 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, each matched pair must still be approved or rejected. 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 generates matched pairs in the linking step, users typically are required to export results to a different program, reformat, record manual reviews, and then

import back into Stata. Records with accepted matches are then saved separately and the matching process is continued for records without accepted matches. This set of steps can become particularly cumbersome in a large, multistage linking project. However, `clrevmatch` offers 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 (for example, `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 whether multiple matches are found for a given record in the master dataset. If so, the program 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. Some of the required inputs are explained below.

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

6.3 Options

`idmaster(`*varname*`)` specifies the record identifiers from the master dataset. `idmaster()` is required.

`idusing(`*varname*`)` specifies the record identifiers from the using dataset. `idusing()` is required.

`varM(`*varlist*`)` and `varU(`*varlist*`)` specify the set of variables in the master and using datasets, respectively, that will be displayed during the review. The user can specify

not only the set of variables used when matching but also other existing variables in the dataset that may help assess the candidates. `varM()` and `varU()` are required.

`clrev_result(`*newvarname*`)` specifies a new variable name to record the user's clerical review input. `clrev_result()` is required.

`clrev_note(`*newvarname*`)` specifies a new variable name for the user to enter a note associated with each pair of records. Because clerical review is often a lengthy and time-consuming component of record linking, this program periodically saves the results as the user progresses. If the reviewer does not finish reviewing the whole dataset in one session, he or she can continue to 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. `clrev_note()` is required.

`reclinkscore(`*varname*`)` specifies the variable containing the machine-generated score from the matching step. If this option is not specified, all other score-related options are disabled.

`rlscoremin(#)` and `rlscoremax(#)` 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]

`rlscoredisp(on|off)` is set to `on` by default, such that the display includes the machine-generated score from the `reclinkscore()` option. If the user does not want the score to influence the clerical review decision, specify `rlscoredisp(off)` so that the score will not be displayed.

`fast` speeds up the review process. By default, the reviewer is asked to confirm the clerical input, and then the program allows the reviewer to enter any additional notes for later review or editing. Specifying `fast` will cause the program to skip these steps.

`clrev_label(`*label*`)` allows the user to specify the 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` for `"definitely a match"`. The labels can be user specified using Stata's label format. For example, an alternative label could be a simpler one like `clrev_label(0 "not match" 1 "match")` or a more specific one like `clrev_label(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 five records.

`replace` overwrites the dataset specified in *filename*.

12. The default is `rlscoremin(0)` and `rlscoremax(1)`. 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.

`newfilename(`*newfilename*`)` outputs results as a new dataset, as specified by `newfilename(`*newfilename*`)`. Note that either `replace` or `newfilename()` must be chosen for the program to proceed.

`saveold` saves the dataset in an older Stata format.

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 following example, we 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`, respectively. This review will use the default review label.

```
. clrevmatch using reclinking_forreview, idm(rid) idu(firm_id)
> varM(stn_name add1 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: .978
------------------------------------------------------------------------------
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:.
```

Now, we can input a clerical review label for this potential match (most likely a `3` for `"definitely a match"`). Then, because `fast` is not specified, the program will 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, two candidates have a 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 first match pertaining to the `rid` #2 record and approve the second match. Recall that this record has two candidates to be reviewed because we 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. For example, the user could type

```
. 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 thousands 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(0.8)` and `rlscoremax(0.97)` and assume that all pairs with scores higher than 0.97 are true matches.

7 Conclusions

We have provided a new set of commands that facilitates 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 that they allow advanced users to 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. `clrevmatch` helps researchers interactively review the generated matched pairs without exporting to another software and importing back to Stata. These utilities can also be used independently. For example, the `stnd_compname` and `stnd_address` commands may be used in one 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 Acknowledgments

The Summer Working Group for Employer List Linking—a collaboration between researchers at the U.S. Census Bureau, University of Michigan, and Cornell University—provided several useful suggestions for the `stnd_compname` command. Ann Rodgers contributed to the `agg_acronym` subcommand. We gratefully acknowledge support by the Alfred P. Sloan Foundation for the University of Michigan's Census-Enhanced Health and Retirement Study project and the National Science Foundation (SES1131500) for the University of Michigan node of the National Science Foundation-Census Research Network.

9 References

Abowd, J. M., and M. H. 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. Working paper. http://www.usc.edu/schools/business/FBE/FOM/paper/FOM-Session5_2.20p_AGRAWAL.pdf.

Blasnik, M. 2010. reclink: Stata module to probabilistically match records. Statistical Software Components S456876, Department of Economics, Boston College. https://ideas.repec.org/c/boc/bocode/s456876.html.

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

Fellegi, I. P., and A. B. Sunter. 1969. A theory for record linkage. *Journal of the American Statistical Association* 64: 1183–1210.

About the authors

Nada Wasi is an assistant research scientist at the Survey Research Center, Institute for Social Research at the University of Michigan.

Aaron Flaaen is an economist in the Division of Research and Statistics at the Federal Reserve Board of Governors in Washington, DC.