Paper 2487-2018

Fun with Address Matching: Use of the COMPGED Function and the SQL Procedure

S. Bianca Salas, Alexandra Varga, and Elizabeth Shuster, Kaiser Permanente Center for Health Research, Portland, OR

ABSTRACT

Address matching is often a challenging task for a SAS® programmer. What seems like a relatively straightforward quest ends up amounting to hours of frustration and manual record review. Who knew there were more than five ways to spell the word "North"? Now, multiply this by 20+ words that might have varying naming conventions (I want to scream just thinking about it). This paper discusses some common data cleaning techniques used in address matching, including the TRANSWRD and COMPRESS functions. A roadmap for the use of the COMPGED function and the SQL procedure to identify matches is provided. The advantages and flexibility of this approach are sure to drive you on your way to address matching in no time.

INTRODUCTION

Addresses can serve as unique identifiers of individuals or observations within a data set. They can be linked to U.S. Census data to gather additional social and economic information on a population that was not available within the source data set. Address data can be used to identify members within a household as part of an infectious disease outbreak investigation, to categorize unique households for survey or mailing distributions, or to identify possible duplicates within a data set. Unfortunately, addresses are often housed in open-text fields which make them subject to human error and varying approaches to denoting a street name, direction or suffix. For instance, 222 W Elm Street can also be listed as:

- 222 West Elm Street
- 222 W E lm St
- 222 W. Elm St.
- 222 W ELM STREET

Without data cleaning, standardization, or fuzzy matching, SAS would have identified each of these as a separate address. To identify matches, each observation can be hard-coded to match the structure of the first address, an acceptable approach for very small data sets. However, manual review is not practical for larger data sets, with hundreds or even thousands of addresses to process.

This paper discusses several SAS techniques that can enhance the performance and efficiency of an address matching program. Specifically, we outline the advantages and limitations of the COMPRESS function and SQL procedure, and provide step-by-step processes for data cleaning, standardization, and transformation.

DATA CLEANING AND STANDARDIZATION

There are several SAS functions that can help simplify the data cleaning and transformation process. Our approach makes use of the UPCASE, CAT, COMPRESS, and TRANWRD functions. A general rule of thumb for any matching process is to standardize the case of the variable and concatenate the variables that will be used in the match. We use the UPCASE and CAT functions to accomplish this step, as outlined at the end of this section.

Table 1 is an example of a typically structured address data set.

ID	ADDRESS1	ADDRESS2	CITY	STATE	ZIP
1	9999 ash st		Portland	OR	91999
2	98765 NW DATABASE WY	APT 14	Beaverton	OR	92999
3	1407 Flowerey Dr		Salem	OR	93999
4	549 N. 7TH AVE		Portland	OR	94999
5	9999 ASH STREET		Portland	OR	91999
6	98765 nw database wy	#14	Beaverton	OR	92999
7	1407 FLOWEREY DT		Salem	OR	93999
8	549 north 7th avenue		Portland	OR	94999
9	749 n 7th ave		Portland	OR	94999

Table 1. Example of an Address Data Set

COMPRESS FUNCTION

A common address matching challenge is the use of additional spaces and tabs within a variable. The COMPRESS function, used with modifiers, removes extra spaces, trailing blanks, and punctuation that would otherwise hinder the performance of a matching operation. COMPRESS performs all these data cleaning techniques in one single line of code.

The syntax and modifier descriptions are as follows (SAS Institute Inc., 2018a):

- Syntax: COMPRESS(<source>,<chars>,<modifiers>)
- Modifier Descriptions:
    - s = removes space characters (blank, tab, feed, etc.)
    - t = removes trailing blanks
    - p = removes punctuation marks

TRANWRD FUNCTION

Notice that in Table 1, there were several potential matches that differed slightly on street name, direction, or suffix. The TRANWRD function can standardize these discrepancies by replacing or removing all occurrences of a given word (or set of characters) within a character string, making for a smoother match.

The syntax and a description of each element are listed below (SAS Institute Inc., 2018a):

- Syntax: TRANWRD(<source>,<target>,<replacement>)
- Element Descriptions:
    - source = source string
    - target = string of characters that are searched to be replaced
    - replacement = string of characters that will replace the target

CODE AND OUTPUT FOR ADDRESS STANDARDIZATION

The sample code and output below demonstrate the use of UPCASE and CAT to create a new address variable: *ADDRESS_FULL*. Following this, COMPRESS and TRANWRD are used to standardize common address components. An example of the output after the code has been submitted is provided in Table 2.

The code creates the standardized data set ***CLEANED_ADDRESS.*** Comments are inserted to detail the purpose of each function.

```
data CLEANED_ADDRESS;

/* Remove blank addresses so they do not show up as matches. */
```

```
set ORIGINAL (where= (address1 ne ''));

   /* Format address to upper case and concatenate into one field. */
   /* Remove spaces, trailing blanks, and punctuations. */
   address_full=compress (upcase(cat(address1, address2)), , 'stp');

   /* Standardize common street elements and suffixes by providing the
   three TRANWRD elements. */
   address_full=tranwrd(address_full,'STREET','ST');
   address_full=tranwrd(address_full,'AVENUE','AVE');
   address_full=tranwrd(address_full,'APT','');
   address_full=tranwrd(address_full,'POBOX','');

   /* Remove the replacement blanks created in TRANWRD (if blanks were used
   as replacements). */
   address_full=compress(address_full);

run;
```

Table 3 displays the transformation of the *ADDRESS1* and *ADDRESS2* variables from the original data set into the new variable *ADDRESS_FULL*.

ID	ADDRESS1	ADDRESS2	ADDRESS_FULL
1	9999 ash st		9999ASHST
2	98765 NW DATABASE WY	APT 14	98765NWDATABASEWY14
3	1407 Flowerey Dr		1407FLOWEREYDR
4	549 N. 7TH AVE		549N7THAVE
5	9999 ASH STREET		9999ASHST
6	98765 nw database wy	#14	98765NWDATABASEWY14
7	1407 FLOWEREY DT		1407FLOWEREYDT
8	549 north 7th avenue		549NORTH7THAVE
9	749 n 7th ave		749N7THAVE

Table 2. Address Variables Before and After Data Standardization

This example is not comprehensive and should be tailored to the unique issues identified in your address data set. One way to identify data inconsistencies is to sort the data set by ADDRESS_FULL or a combination of ZIP and ADDRESS_FULL. Visual inspection can be used to isolate additional terms to be included in the TRANWRD cleaning process.

USING THE COMPGED FUNCTION FOR ADDRESS MATCHING

The COMPGED function measures the dissimilarity between two strings using a variation of Levenshtein edit distance, which calculates the least number of edits needed to change one string into another string (SAS Institute Inc., 2018c). It returns a score (cost) based on the generalized edit distance (deletions, insertions, etc.) required to transform the string. The syntax for this function is provided below with the arguments in brackets being optional:

- Syntax: COMPGED (string-1, string-2, <cutoff>, <modifiers>)

Table 4 illustrates the COMPGED cost scoring system (SAS Institute Inc., 2018c).

Operation	Default Cost in Units	Description of Operation
APPEND	50	When the output string is longer than the input string, add any one character to the end of the output string without moving the pointer.
BLANK	10	Do any of the following: • Add one space character to the end of the output string without moving the pointer. • When the character at the pointer is a space character, advance the pointer by one position without changing the output string. • When the character at the pointer is a space character, add one space character to the end of the output string, and advance the pointer by one position. If the cost for BLANK is set to zero by the COMPCOST function, the COMPGED function removes all space characters from both strings before doing the comparison.
DELETE	100	Advance the pointer by one position without changing the output string.
DOUBLE	20	Add the character at the pointer to the end of the output string without moving the pointer.
FDELETE	200	When the output string is empty, advance the pointer by one position without changing the output string.
FINSERT	200	When the pointer is in position one, add any one character to the end of the output string without moving the pointer.
FREPLACE	200	When the pointer is in position one and the output string is empty, add any one character to the end of the output string, and advance the pointer by one position.
INSERT	100	Add any one character to the end of the output string without moving the pointer.
MATCH	0	Copy the character at the pointer from the input string to the end of the output string, and advance the pointer by one position.

Table 3. Example of COMPGED Operations and Cost Scoring

To perform a COMPGED match, the data needs to be structured across two variables, for example *ADDRESS_FULL1* and *ADDRESS_FULL2*. The following COMPGED code can be used to calculate the cost across these variables*:*

```
COM_COST= COMPGED(ADDRESS_FULL1,ADDRESS_FULL2)
```

As seen in Table 3, COMPGED assigns varying costs based on the location of the edit within the string. Deletions, insertions, and replacements made in the first position (Operations FDELETE, FINSERT, and FREPLACE) cost 200 units, while deletions, insertions or replacements elsewhere in the text string cost 100 units. Comparing rows 3 and 4 below, you can see there is only a single character difference between *ADDRESS_FULL1* and *ADDRESS_FULL2* in both rows. Row 3, however, is assigned a cost of 100 (change made at the end) and row 4 a cost of 200 (change made at the beginning). This feature makes COMPGED particularly sensitive for address matching since addresses that have the same street name (e.g. 7th Ave) but different street numbers should not be considered matching addresses.

Table 4 shows COMPGED cost scoring when used on addresses.

ROW #	ADDRESS_FULL1	ADDRESS_FULL2	COM_COST
1	9999 ASH ST	9999 ASH ST	0
2	98765 NW DATABASE WY 14	98765 NW DATABASE WY #14	20
3	1407 FLOWEREY D**R**	1407 FLOWEREY D**T**	100
4	**5**49 N 7TH AVE	**7**49 N 7TH AVE	200

Table 4. COMPGED Costs Associated with Addresses

Unlike the example above, our address data was structured in a single column. To perform the COMPGED match, we create a copy of the ***CLEANED_ADDRESS*** data set***.*** The copied data set will be referred to as ***ADDRESS_COPY*** in the next section.

STEPS TO IDENTIFY CLEAN ADDRESS MATCHES

THE SQL PROCEDURE AND COMPGED FUNCTIONS TO IDENTIFY MATCHES

The SQL procedure retrieves, modifies and creates tables, views, and indexes (SAS Institute Inc., 2018d). One of the many benefits of PROC SQL is the ability to join on multiple parameters to create new

tables. While this paper discusses two common parameters unique to address matching, *ZIP* and *ADDRESS_FULL*, the PROC SQL join could be further parameterized by a multitude of items including date or other group characteristics. PROC SQL also provides you flexibility to determine which variables you want to keep from each data set, the order in which you want each variable to appear, and which variables you would like to rename, all in one SQL procedure.

The following steps outline this process:

1. Perform a direct PROC SQL join on *ZIP*.
2. Calculate the costs between the *ADDRESS_FULL* variables in the data sets ***CLEANED_ADDRESS*** and ***ADDRESS_COPY***. Restrict to COMPGED costs less than or equal to 100.This cost threshold is subjective and will vary depending on your project specifications and the degree of certainty you wish to achieve with the match.
3. Limit to joins where ***CLEANED_ADDRESS*** ID and ***ADDRESS_COPY*** ID are not equal. This will prevent matches on the same participant (given that ID is unique in your data set).

The steps above along with examples of distinct variable selection and variable renaming are shown in the code below:

```
proc sql;
   create table Address_Matches as
   select distinct p1.*,
   p2.ID as ADDRESS_COPY_ID,
   p2.address_full as ADDRESS_COPY_ADDRESS,
   p2.zip as ADDRESS_COPY_ZIP,
   compged (p1.address_full, p2.address_full) as com_cost

   from CLEANED_ADDRESS p1 join ADDRESS_COPY p2

   on  p1.zip=p2.zip 1

   and (compged(p1.address_full,p2.address_full) le 100) 2

   and p1.id ne p2.id 3

   order by com_cost, id;
quit;
```

Table 5. Example of the Data Set Address_Matches

ID	ADDRESS1	ADDRESS2	ZIP	ADDRESS_FULL	ADDRESS_COPY_ID	ADDRESS_COPY_ADDRESS	ADDRESS_COPY_ZIP	COM_COST
1	9999 ash st		91999	9999ASHST	5	9999ASHST	91999	0
2	98765 NW DATABASE WY	APT 14	92999	98765NWDATABASEWY14	6	98765NWDATABASEWY14	92999	0
5	9999 ASH STREET		91999	9999ASHST	1	9999ASHST	91999	0
6	98765 nw database wy	#14	92999	98765NWDATABASEWY14	2	98765NWDATABASEWY14	92999	0
3	1407 Flowerey Dr		93999	1407FLOWEREYDR	7	1407FLOWEREYDT	93999	100
7	1407 FLOWEREYDT		93999	1407FLOWEREYDT	3	1407FLOWEREYDR	93999	100

Table 5. Example of the Data Set Address_Matches

REMOVE DUPLICATED MATCHES

Although all matches meeting our criteria are accounted for in Table 5, the procedure performed the join in both directions, so each match appears twice in the data set (i.e. matches on IDs 1 and 5 and 5 and 1). To eliminate these 'duplicate' matches, remove rows in which the ID field appears in the *ADDRESS_COPY_ID* field. Because the data set ***ADDRESS_MATCHES*** was ordered by the COMPGED cost variable *COM_COST*, the join is optimized leaving you with the best possible match.

Code to perform this de-duplication is as follows:

```
proc sql;
create table Final_Addresses as
select * from Address_Matches
where ID not in (select ADDRESS_COPY_ID from Address_Matches);
quit;
```

CONCLUSION

Analyses can be greatly enriched by utilizing address data. Household outreach and evaluation of socioeconomic factors and household behaviors can be improved by using a cleaned address data set. We demonstrated how you can use several SAS functions, including COMPRESS, TRANWRD, COMPGED and PROC SQL, to streamline the data cleaning and matching process. Although these techniques require some element of human review, particularly when dealing with open-text fields such as address and zip, the amount of time spent on processing and developing code to cover all address matches is significantly reduced.

REFERENCES

SAS Institute Inc. 2018a. "COMPRESS Function." Accessed February 8, 2018. http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000212246.htm

SAS Institute Inc. 2018b. "TRANWRD Function." Accessed February 8, 2018. http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a000215027.htm

SAS Institute Inc. 2018c. "COMPGED Function." Accessed February 8, 2018. http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a002206133.htm

SAS Institute Inc. 2018d. "Overview: SQL Procedure." Accessed February 8, 2018. http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#sql-overview.htm

ACKNOWLEDGMENTS

Scott Leslie, MedImpact Healthcare Systems, Inc.© provided critical guidance in the development of this paper as part of the SAS Mentorship Program. He assisted in the review of the abstract, paper and presentation for the 2018 SAS Global Forum submission.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Suzanne Bianca Salas
Kaiser Permanente Center for Health Research
suzanne.b.salas@kpchr.org