A Step-by-Step Procedure for Matching Addresses/Names in SAS

-Pramod Sambidi and Himanshu Joshi, Houston-Galveston Area Council, Houston, TX

Abstract:
The Socioeconomic modeling group at the Houston-Galveston Area Council collects data on businesses from different sources and process it to develop a master buildings and businesses dataset with number of jobs per building. In the process of data development, one of the major issue was matching company name and address from different sources. The paper discusses the ways to standardize the addresses/names and match them successfully using SAS.

Introduction:
There are several sources that provide a list of businesses and their addresses in Houston area, but there is no specific source that provides data on commercial buildings and jobs per building. The Socioeconomic modeling group at the Houston-Galveston Area Council collects data on businesses from different sources and process it to develop a master buildings dataset with number of jobs per building. In the process of data development, one of the major issues was matching company name and address from different sources. This paper presents some of the SAS functions that are used in address matching step of our data development process. One of the major issues with address matching is misspelled or abbreviated street names or company names. Sometimes the misspelled words can be identified easily and corrected when they follow a certain pattern, however most of the time address field may have some errors that do not stand out and can only be processed or identified using the SAS character matching functions. This paper presents some of the SAS functions that can be used in standardizing and correcting the misspelled names in a given data.

Matching Process:
First, we assume that the address field in the data is in one column. The address filed may include street number, street prefix, street name, street type, street suffix and suite number.

For example 3555 timmons lane, suite 120

The first thing to do is capitalize all fields in the address column. This will help in address matching later on. We can use UPCASE function to capitalize all lower case letters. It is always a good thing to remove the leading and trialing blanks in a character variable, which can be achieved by using STRIP function.

```
example:
New_Address = strip(upcase(Old_address));
```

Once the address field is capitalized with leading and trailing blanks removed, the next step is to modify the address filed, so that it will be easy to parse the field into street prefix, street pretype, street number, street name, street type, and street suffix.
We will replace the following characters in the address field with space: **‘.’ ’-‘ ’;’ ’/’ ’,’** . In order to do this we use TRANWRD function, this replaces or removes all occurrences of a word in a

character string. At this point, we also replace character strings such as ‘suite’, ‘ste’, ‘room’ with ‘#’.

example:
```
New_address1 = tranwrd(new_address,".", " ");
```

In the next step, we will use the SCAN function to parse all the words from the address variable into new variables, this will us assign street prefix, street pretype, street number, street name, street type, and street suffix.

example:
```
new1 = scan (New_address1,1," . < ( + & ! $ * ) ; ^ , % |");
new2 = scan (New_address1,2," . < ( + & ! $ * ) ; ^ , % |");
new3 = scan (New_address1,3," . < ( + & ! $ * ) ; ^ , % |");....

newn= scan (New_address1,3," . < ( + & ! $ * ) ; ^ , % |");
```

After creating the new variables, we assume that first variable that was created contains the street number, so we create the *street number* variable and assign the contents of *new1* variable. In case the numbers are spelled out, it would be better to replace them with actual numbers using an IF statement.

example:
```
if new1 = 'ONE' then ST_NUMBER = '1';
```

Next we assume that *new2* variable contains the street prefix. We can use the USPS common address street types to test this and assign street prefix. We use a simple IF THEN DO statement to perform this. If *new2* variable contains character strings such as NORTH, SOUTH, EAST, WEST, N, S, E, or W, and if the *new3* variable do not contain any of the standard street types, then we assume that the *new2* variable is a street prefix. Therefore, we create a new variable called *street prefix* and assign the values of *new2* variable. To be consistent, change abbreviations such as N to NORTH and so on. However, if in fact *new3* variable contains one of the street types then we can assume that *new2* variable is *street name* and *new3* variable is *street type*. If this is the case then rest of the variables from *new4* to *newn* will be assumed as *street suffix*. Likewise, we compare each of these variables and assign them to *street number, street prefix, street name, street type, and street suffix*. Make sure that street types, street prefix, and street suffix are consistent (spelled correctly) all across the data.

Once addresses are standardized, the next step is to match them from different data sources in order to create a unique database of buildings and business. First, we match the addresses and create a unique buildings database. We use the Southeast Texas Addressing and Referencing Map (STAR Map) data created and maintained by the H-GAC and Geographic Data Committee. The street names from the STAR map data are used to correct the misspelled street name to create a unique buildings database. Once the buildings database is created, we match businesses from different sources to create a unique database of businesses. It looks like a straight forward procedure to match addresses and businesses, but most of the times names are misspelled or abbreviated, which prevents them from matching. Here we will present some of the SAS functions that are helpful in matching misspelled company names.

First we will capitalize names using the UPCASE function as mentioned earlier. After capitalizing, we will look for charactering string ‘THE’ at the beginning of company name and delete it. This is important because some data sources have ‘THE’ in front of the company name and some don’t have it. In order to do this, we can use SAS functions INDEXW and TRANWRD. INDEXW searches a character string for a sub-string pattern and returns the value indicating the location of that sub-string. Then, we will use TRANWRD to replace the sub-string that was found at the beginning of the character string.

example
```
index1=indexw(company_d1, 'THE');
if index1=1 then company_d1=tranwrd(company_d1,'THE','');
```

The first step in matching company names from two data sources is to create a cross product of the two, restricted by address so that a given company is compared with all other companies that are located at that company’s address. Once the cross product is created, we follow the following steps:

1. Match company names directly from two data sources based on their location and create a flag variable indicating that they are matched

```
if company_d1=company_d2 then Match='Exact Match';
```

2. Based on the flag variable select the unmatched companies and use SPEDIS function, which determines the likelihood of two words matching, expressed as the asymmetric spelling distance between the two words (SAS Help and Documentation).

```
spedis1 =spedis(company_d1,company_d2);
```

After getting the SPEDIS distance value, check for the minimum value for each company and how many times it occurs (SAS-L listserv.uga.edu). Based on visual observation select the minimum cut off value that occurs only once. Generally speaking, this procedure matches most of the unmatched names; however, in order to match all companies we need further processing.

3. In the next step we will select the unmatched data points and use the SPEDIS function with variables reversed. We will then select the matched data points based on a minimum cut off value as mentioned above.

```
spedis2 =spedis(company_d2,company_d1);
```

4. Next we will use SAS function COMPGED, which compares two strings by computing the generalized edit distance (SAS Help and Documentation). We will follow the same procedure to select the data points as we did for SPEDIS function. Remember cut off value always differs, so it is important that you look at the results and select the cut off value.

```
compged1=compged(company_d1,company_d2);
```

5. At the end of the 4rth step there will be few companies remaining for matching. We can match these by using some SAS character functions. We will list a few here:

i) Matching based on the ‘First Word’, which has a character length greater than three (SCAN function)

```
if length(scan(company_d1,1))ge 3;
where scan(company_d1,1)=scan(company_d2,1);
```

ii) Matching based on a ‘part’ of company name (INDEX function)

```
EXXON_d1=index (company_d1,'EXXON');
EXXON_d2=index (company_Td2,'EXXON');
if EXXON_d1~=0 and EXXON_d2~=0 ;
```

iii) Comparing and matching one word of a variable with all words in other variable (SCAN function)

```
if length(scan(company_d1,1))ge 3;
if scan(company_d1,1)= scan(company_d2,2)then match='scan_match_12_T'; else
if scan(company_d1,1)= scan(company_d2,3) then match='scan_match_13_T';else
if scan(company_d1,1)= scan(company_d2,4) then match='scan_match_14_T';else
if scan(company_d1,1)= scan(company_d2,5) then match='scan_match_15_T';else
if scan(company_d1,1)= scan(company_d2,6) then match='scan_match_16_T';else;

if scan(company_d1,2)= scan(company_d2,2)then match='scan_match_22_T'; else
if scan(company_d1,2)= scan(company_d2,3) then match='scan_match_23_T';else
if scan(company_d1,2)= scan(company_d2,4) then match='scan_match_24_T';else
if scan(company_d1,2)= scan(company_d2,5) then match='scan_match_25_T';else
if scan(company_d1,2)= scan(company_d2,6) then match='scan_match_26_T';else;
```

At the end of this matching process we can assume that there are no duplicate companies and hence, we have a unique data set of businesses. However, it is better to manually check for any duplicates provided you have few names that are left unmatched. We suggest users to look into other SAS functions and operators for matching, such as SOUNDS LIKE OPERATOR, SOUNDEX, COMPARE, CALL COMPCOST, and COMPLEV.

Conclusion:
The whole process of address and name matching seems to be laborious, but once the code is setup it will be easy for future matching and annual updates.

References

Ronald P. Cody. “SAS Functions by Example.” Section: ‘Functions That Compare Strings (Exact and “Fuzzy” Comparisons)’. 2004.

SAS-L listserv Discussion Forum, The University of Georgia. http://listserv.uga.edu
http://listserv.uga.edu/cgi-bin/wa?A2=ind0201d&L=sas-l&F=&S=&P=31978

CONTACT INFORMATION

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

Pramod R. Sambidi
Houston-Galveston Area Council
3555 Timmons Lane, Suite#120
Houston, TX-77027
Email:psambidi@h-gac.com