Paper DM-126

Using PROC FCMP to Improve Fuzzy Matching

Christine L. Warner, Integrity Management Services, LLC, Alexandria, VA

October 17, 2016

ABSTRACT

Fuzzy matching is the art and science of matching inexact phrases, names, addresses, numbers, and other text. The need for fuzzy matching to reconcile disparate databases is one of the most common problems in master data management, entity resolution, and data management in general. Aggregating financials for one entity or one person is nearly impossible when there are multiple records for that entity or person. Fuzzy matching is the foundation for solving the golden record problem, also known as entity resolution, record linkage, and master data management. Entity resolution is not covered in this paper but covered very well by Glenn Wright in his SAS paper titled "Probabilistic Record Linkage in SAS". My paper presents some creative ways to conduct fuzzy matching on person names, entity names, addresses, and other relevant fields commonly utilized during matching. Hopefully you will find some of the code useful in your fuzzy matching endeavors!

INTRODUCTION

I have been a fuzzy matching enthusiast for years (aka "geek"). The nuances in different edit distance functions intrigue me and the problems I have faced as a programmer have inspired me to develop my own user-defined fuzzy matching functions. In the recent past with the addition of PROC FCMP, SAS has allowed me the flexibility to develop user-defined functions that have proved invaluable when conducting large matching and entity resolution projects. Some of the challenges I have faced in my career that have warranted enhanced fuzzy matching and entity resolution include:

- Resolving 8,000 different spellings of "Buenos Aires" for Customs and Border Protection (CBP), so that CBP could identify whether a package was sent to an urban or rural area, in the context of homeland security.
- Matching Medicaid providers and recipients to Medicare providers and beneficiaries on a national level, for the Centers for Medicare and Medicaid Services (CMS).
- Reconciling a client's financial transaction database – payee names - with the OFAC (Office of Foreign Assets Control) Terrorist Watch List, also known as the Specially Designated Nationals (SDN) and Blocked Persons List.
- Matching vendor master files against a company's employee file, matching upon name, address, phone, SSN/EIN, and email, for the purposes of identifying potential fraud and fictitious vendors or employees.
- Matching vendor master files against the Mail Drop File, which contains a list of all UPS Stores, Fedex, and Parcel Posts across the United States, for the purposes of identifying potential fictitious or fraudulent entities.

COMPARING NAMES AND PHRASES

Using fuzzy matching to compare names and phrases can be complicated when a person's name can have so many different spellings, nicknames, prefixes, suffixes, or presented in different orders. SAS provides several functions such as COMPLEV (Levenshtein function) and COMPGED (Generalized Edit Distance) function that are very useful for matching names. I often lean on COMPLEV and COMPGED to compare first names and last names for individuals. I also utilize a SAS version of the Jaro-Winkler function that I discovered online that has been fit to PROC FCMP by several SAS experts (see References). But sometimes data is not always clean, or ordered or parsed in the way you need it to be to match. For example, matching the two names below does not lend itself to COMPLEV, COMPGED, or the Jaro-Winkler function very easily because these functions are best suited for *word* matching, not *phrase* matching.

```
Example:

"Theresa L. Smith-Hernandez"
"Hernandez, Teresa"

data temp;
  length name1 name2 $40;
  name1='Theresa L. Smith-Hernandez';
  name2='Hernandez, Teresa';
  lev = complev(name1,name2);
```

```
  ged = compged(name1,name2);
  jaro = jaroink(name1,name2,.1);
run;
```

Match Results Table 1:

Name1:	Theresa L. Smith-Hernandez
Name2:	Teresa Hernandez
COMPLEV Score	21
COMPGED Score	1610
Jaro-Winkler Score	0.5385

The COMPLEV score is 21, and the COMPGED score as 1610, and the Jaro-Winkler score is 0.5385: all VERY inaccurate scores because these functions were not designed to match phrases. The Jaro-Winkler score is a score from 0 to 1, with 0 being no match, and 1 being a perfect match. COMPLEV and COMPGED work the other way: the lower the score, the more accurate the match.

There are solutions to this dilemma: 1) utilize COMPLEV, COMPGED, and/or the Jaro-Winkler function in a word by word comparison using a function I called “PHRASE_MATCH”, and/or 2) utilize the PCTTRIGRAM function to match strings and phrases.

PHRASE_MATCH FUNCTION

The PHRASE_MATCH function takes as parameters two strings or phrases; they could be individual names, entity names, or text phrases that you would like to match upon. The output is a percentage between 0 and 1 that represents the percentage of characters in phrase1 (p1) that match the characters in phrase2 (p2). The function matches each word of phrase1 with each word of phrase2 using 3 methods; then, if the word is considered a match, the length of the word is added to the variable CHARS_MATCH. After each word of phrase1 has been compared to each word in phrase2, CHARS_MATCH is then divided by the total length of phrase1, minus the spaces. There are many ways this could be achieved, this is just one approach, shown below:

Example 1: PHRASE_MATCH Function

```
proc fcmp outlib = work.funcs.test;
  function PHRASE_MATCH(p1$,p2$);
       length s1 s2 $100;
       length word1 word2 $20;
       PHRASE_MATCH=0;
       s1=upcase(p1);
       s2=upcase(p2);
       numwords1=numwords(s1); /* user-defined function numwords */
       numwords2=numwords(s2);
       /* compare every word in one phrase with every word in other phrase: */
       /* denominator is based upon length of P1, PHRASE1, S1 */
        chars_match=0;
        do i = 1 to numwords1;
          found=0;
          do j = 1 to numwords2;
            lev_hit=0;
               jw_hit=0;
               found=0;
               word1=scan(s1,i);
                word2=scan(s2,j);
                len1=length(word1);
                len2=length(word2);
                lev_score = complev(word1,word2);
                jw_score = jarowink(word1,word2,.1);
                if word1=word2 then found=1;
                if len1 ge 5 and len2 ge 5 then
                do;
                  if find(word1,word2,'it') ne 0 then found=1;
                  if find(word2,word1,'it') ne 0 then found=1;
```

```
                end;
                if not found then
                do;
                  if lev_score le 1 and (len1 ge 4 and len2 ge 4) then found=1;
                  if lev_score le 2 and (len1 ge 8 and len2 ge 8) then found=1;
                end;
                if jw_score ge .9 then found=1;
                if found then chars_match = chars_match + length(word1);
            end;  /* j loop */
        end;  /* i loop */
        /* compress spaces and punctuation before calculating length */
        denom=length(compress(s1,,'sp'));
        PHRASE_MATCH = chars_match/denom;
  return(PHRASE_MATCH);
  endsub;
quit;

options cmplib=(work.funcs); /* This writes the function to the functions library */
```

Now if we repeat the name matching using the PHRASE_MATCH function, we will see some higher scores showing a match:

Match Results Table 2: Match Results Using PHRASE_MATCH

Name1:	Theresa L. Smith-Hernandez
Name2:	Teresa Hernandez
COMPLEV Score	21
COMPGED Score	1610
Jaro-Winkler Score	0.53
PHRASE_MATCH(name1,name2)	0.72 (match words in name1 with words in name2)
PHRASE_MATCH(name2,name1)	1.00 (match words in name2 with words in name1)

When PHRASE_MATCH(name1,name2) is called, we are matching the words in name1 with the words in name2. Because the word "Smith" is not in name2, and the "L" initial does not match in name2, the score is somewhat lower than perfect: .7272. However, when we call the function with the parameters reversed, PHRASE_MATCH(name2,name1), the score is 1.0, a perfect score, because both "Teresa" and "Hernandez" match the words in name1.

Let's look at another example, this time with corporate entity names.

Name1:	Greenville Pharmacy and Clinic, Inc.
Name2:	The Pharmacy at Greenville
COMPLEV Score	31
COMPGED Score	2680
Jaro-Winkler Score	0.4646
PHRASE_MATCH(name1,name2)	0.60 (match words in name1 with words in name2)
PHRASE_MATCH(name2,name1)	0.78 (match words in name2 with words in name1)

Here again we see that, with a high score of 0.78 (out of 1.0), while other functions consider the two names a clear non-match, PHRASE_MATCH shows promise in identifying some disparate matches. To put the other matches in perspective, a COMPLEV score of less than 3 is generally considered a close match, as is a COMPGED score of less than 100-200, or a Jaro-Winkler score greater than 0.9.

PCTTRIGRAM FUNCTION

I would like to share the PCTTRIGRAM function with you, as I have found it invaluable in matching strings and in blocking data prior to more in-depth matching and scoring. The PCTTRIGRAM function calculates the percentage of tri-grams that match between phrases. A trigram is a 3-character substring of a phrase; for example, ‘CHR’ is the first trigram of ‘CHRISTINE’. Specifically, PCTTRIGRAM calculates the percentage of trigrams in phrase2 (p2) that are

found in phrase1 (p1). To achieve the reverse, you must call the function as PCTTRIGRAM(p2,p1) instead of PCTTRIGRAM(p1,p2).

Example 2: PCTTRIGRAM Function

```
proc fcmp outlib = work.funcs.test;
  function pcttrigram(p1$,p2$);
     length s1 s2 $150;
     nummatches=0;
        *change to upcase;
        s1=upcase(p1);
        s2=upcase(p2);
        len1=length(s1);
        len2=length(s2);
        max=length(s2)-2;
        do i = 1 to max;
           sub=substr(s2,i,3);
           if index(s1,trim(sub)) ne 0 then nummatches+1;
        end;
        if max ne 0 then pcttrigram=nummatches/max;
                    else pcttrigram=0;
  return(pcttrigram);
  endsub;
quit;

options cmplib=(work.funcs);
```

NICKNAME COMPARISON USING HASH TABLES

One of the problems that fuzzy matching does not solve is the nickname problem. There just is no way to fuzzy match "Dick" with "Richard" or "Betty" and "Elizabeth". One solution is to find a good nickname database, either open source or through Peacock Data (www.peacockdata.com), and then merge this database with your data. The quickest, most efficient way to conduct nickname matching is using SAS hash tables.

After learning more about SAS hash tables, I eager to apply it to the nickname problem. If you aren't familiar with hash tables, I highly recommend the paper "How Do I Love Hash Tables? Let Me Count The Ways!" by Judy Loren, Paper 029-2008 (See References). In the code example below, I create a temporary SAS data set called "nicknames", and another SAS data set called "all". Both data sets have the columns name1 and name2. In the code, I create a hash table called "lookup"

```
data nicknames;
  length name1 name2 $20;
  name1='CHRISTY'; name2='CHRISTINE'; output;
  name1='ROBERT'; name2='BOB'; output;
  name1='THERESA'; name2='TERRY'; output;
  name1='MICHAEL'; name2='MIKE'; output;
  name1='MIKE'; name2='MICHAEL'; output;
  name1='STEPHEN'; name2='STEVE'; output;
  name1='ELIZABETH'; name2='LIZ'; output;
  name1='LIZ'; name2='ELIZABETH'; output;
run;

data all;
  length name1 name2 $20;
  input name1 $ name2 $;
  datalines;
SUSIE SUE
CHRISTY CHRISTINE
JOHN JONATHAN
MICHAEL MIKE
MIKE MICHAEL
CYNTHIA CINDY
JOHN JOHNNY
ELIZABETH LIZ
```

```
LIZ ELIZABETH
;
run;

data nicknames_found;
   declare hash lookup(dataset: 'nicknames');
   rc = lookup.Definekey('name1','name2');
   rc = lookup.Definedata(all: 'yes');
   rc = lookup.Definedone();
   do until (eof) ;
     set all end = eof;
     rc = lookup.find();
     if rc=0 then output;
   end;
run;
```

If the return code is 0, then the nickname pair (name1,name2) was found, and is then output to the data set "nicknames_found".

ADDRESS MATCHING FUNCTIONS

Address matching could either be a simple SUBSTR function or an art. I choose art. One could simply compare the first 5 characters of the address and be done with the job. But when fuzzy matching is a passion, you cannot stop there! The following are functions you can utilize to increase the accuracy of address matching. These functions are particularly useful when addresses are non-standard, when the addresses are international, or when the primary address is stored in the "street2" column instead of "street1".

One function that I find helpful is one that I created, called "Firstnum", which returns the first number found in the street address.

FIRSTNUM FUNCTION

Firstnum function has one input parameter, a string, and returns the first number of that string. The resulting string is a character string of digits.

```
proc fcmp outlib = work.funcs.test;
  function firstnum(phrase$) $;
  length word $15;
  cnt=1;
  word='x';
  found=0;
  do until(word=' ' or found=1);
     word=scan(phrase,cnt);
        * if word has a # in it- take out everything else and keep only digits:;
        if indexc(word,'123456789') NE 0 then
        do;
           found=1;
           firstnum=compress(word,,'kd');
        end;
        cnt+1;
  end;
  return(firstnum);
  endsub;
run;

options cmplib=(work.funcs);
```

The modifiers 'kd' in the 3rd position of the COMPRESS function tell SAS to keep only the digits in the word found.

Table #: FIRSTNUM Function Results

Street1:	Street2:	Firstnum(Street1):	Firstnum(Street2):
6789 East Main Street	6789-E.Main Street	6789	6789
1972A South Hampton Drive	1792 S. Hampton St. Apt#A	1972	1972
The Cleveland Clinic 2049 E 100th St	2049 East 100th St	2049	2049

In the table above, taking the first 5 characters of street1 and street2 and comparing them would yield no matches. However, using the FIRSTNUM function, all three of the above examples would be considered matches because their firstnum function values are the same for street1 and street2. Messy data is indeed our friend where fuzzy matching is considered!

LONGEST FUNCTION

Another user-defined function that might prove useful is the "LONGEST" function. This function returns the longest word in a phrase. It is particularly useful for address matching because oftentimes, the longest word is the primary word of the address. If you combine this function with a data cleansing function to remove noise words, the LONGEST function becomes even more effective. The code is shown below:

```
proc fcmp outlib = work.funcs.test;
  function longest(phrase$) $;
    length longest word $20;
       Cnt=1;
       max_len=0;
       word='x';
       Do while(word ne ' ');
         Word=scan(phrase,cnt);
         len=length(word);
         if len gt max_len and indexc(word,'123456789')=0 then
          do;
               max_len=len;
               pos_max=cnt;
          end;
         Cnt+1;
       End;
       longest=scan(phrase,pos_max);
       return(longest);
       endsub;
run;

options cmplib=(work.funcs);
```

The diagram below shows how the LONGEST function works. It identifies the longest word in a phrase, and if there are multiple words with the same length, it chooses the first occurring "longest word". The SAS code to call the function is:

longest_word = LONGEST(name); /* where name is your input parameter, a phrase, address, or string */

Figure #. LONGEST Function Results

VIEWTABLE: Work.Temp2

	name	longest_word
1	1234 south lilac st	south
2	4092 breckinridge ln	breckinridge
3	123 east ridgefield rd	ridgefield
4	789-E harbor dr	harbor
5	5678 Pennsylvania Ave	Pennsylvania
6	4243 north Evans drive	north

In Figure # you can see that the function defaults to the first longest word when other words have the max length as well. If the longest word is a "noise" word such as "north" or "south", this can be taken care of by data cleansing functions prior to calling the LONGEST function.

CLEANSEADDR FUNCTION

The following CLEANSEADDR function is one approach to cleansing typical American street address data. It is not meant to be an all-inclusive list but a guide and starting point to cleanse your data prior to matching. The CLEANSEADDR function relies heavily on the TRANWRD function, which is a default function offered by SAS to conduct a "find" and "replace". Below is the code for the address cleansing function:

```
proc fcmp outlib=work.funcs.test;
  function cleanseaddr(s1$) $;
  length temp_address $100;
  temp_address=upcase(s1);
  temp_address=left(trim(temp_address));


temp_address=tranwrd(temp_address,' ONE ', ' 1 ');
temp_address=tranwrd(temp_address,' TWO ',' 2 ');
temp_address=tranwrd(temp_address,' THREE ',' 3 ');
temp_address=tranwrd(temp_address,' FOUR ',' 4 ');
temp_address=tranwrd(temp_address,' FIVE ',' 5 ');
temp_address=tranwrd(temp_address,' SIX ',' 6 ');
temp_address=tranwrd(temp_address,' SEVEN ',' 7 ');
temp_address=tranwrd(temp_address,' EIGHT ',' 8 ');
temp_address=tranwrd(temp_address,' NINE ',' 9 ');
temp_address=tranwrd(temp_address,' TEN ',' 10 ');

temp_address=tranwrd(temp_address,' 1ST',' FIRST');
temp_address=tranwrd(temp_address,' 2ND',' SECOND');
temp_address=tranwrd(temp_address,' 3RD',' THIRD');
temp_address=tranwrd(temp_address,' 4TH',' FOURTH');
temp_address=tranwrd(temp_address,' 5TH',' FIFTH');
temp_address=tranwrd(temp_address,' 6TH',' SIXTH');
temp_address=tranwrd(temp_address,' 7TH',' SEVENTH');
temp_address=tranwrd(temp_address,' 8TH',' EIGHTH');
temp_address=tranwrd(temp_address,' 9TH',' NINTH');
temp_address=tranwrd(temp_address,' 10TH',' TENTH');

temp_address=tranwrd(temp_address,'EAST',' E ');
temp_address=tranwrd(temp_address,'NORTH',' N ');
temp_address=tranwrd(temp_address,'WEST',' W ');
temp_address=tranwrd(temp_address,'SOUTH',' S ');
temp_address=tranwrd(temp_address,' STREET ',' ST ');
temp_address=tranwrd(temp_address,'SAINT ','ST ');
if index(temp_address,'BROAD')=0 then temp_address=tranwrd(temp_address,'ROAD','RD');
temp_address=tranwrd(temp_address,'BOULEVARD','BLVD');
temp_address=tranwrd(temp_address,'PLZ','PLAZA');
temp_address=tranwrd(temp_address,'AVENUE','AVE');
temp_address=tranwrd(temp_address,'HIGHWAY','HWY');
temp_address=tranwrd(temp_address,' DRIVE ',' DR ');
temp_address=tranwrd(temp_address,'P.O.','PO');
temp_address=tranwrd(temp_address,'SUITE','STE');
temp_address=tranwrd(temp_address,'SUIT','STE');
temp_address=tranwrd(temp_address,'ST RT','STATE ROUTE');
temp_address=tranwrd(temp_address,' RR ', ' RURAL ROUTE ');
temp_address=tranwrd(temp_address,' SR ', ' STATE ROUTE ');
temp_address=tranwrd(temp_address,' RR#',' RURAL ROUTE #');
temp_address=tranwrd(temp_address,' SR#',' STATE ROUTE #');
temp_address=tranwrd(temp_address,'PLACE','PL');
temp_address=tranwrd(temp_address,'SQUARE','SQ');
temp_address=tranwrd(temp_address,'LANE','LN');
temp_address=tranwrd(temp_address,'TRAIL','TR');
temp_address=tranwrd(temp_address,'BUILDING','BLDG');
temp_address=tranwrd(temp_address,'CIRCLE','CIR');
temp_address=tranwrd(temp_address,'RTE ','ROUTE ');
temp_address=tranwrd(temp_address,'COURT ','CT ');
temp_address=tranwrd(temp_address,'PARKWAY','PKWY');
temp_address=tranwrd(temp_address,'APARTMENT','APT');
temp_address=tranwrd(temp_address,'CENTER ','CTR ');
temp_address=tranwrd(temp_address,'CENTRE ','CTR ');
temp_address=tranwrd(temp_address,'CENTR ','CTR ');
temp_address=tranwrd(temp_address,'CNTER ','CTR ');
temp_address=tranwrd(temp_address,'MC ','MC');
```

```
temp_address=tranwrd(temp_address,"/"," ");
temp_address=tranwrd(temp_address,","," ");
temp_address=tranwrd(temp_address,"&"," ");
temp_address=tranwrd(temp_address,"-"," ");
temp_address=tranwrd(temp_address,"."," ");
temp_address=compress(temp_address,",.+-/\_()*&!@$%^'");
temp_address=tranwrd(temp_address,'POBOX','PO BOX');
temp_address=tranwrd(temp_address,'P O BOX','PO BOX');
temp_address=tranwrd(temp_address,'PO BOX','PO BOX ');
temp_address=tranwrd(temp_address,'  ',' ');

cleanseaddr=temp_address;

return(cleanseaddr);
endsub;
quit;
```

PCTTRIGRAM FUNCTION

I introduced the PCTTRIGRAM function earlier, but in this section I describe how to apply its power to complex address matching, particularly international addresses.

Even with COMPGED, COMPLEV, the Jaro-Winkler function, and others such as SPEDIS, COMPARE, and SOUNDEX, we still lack a more accurate string comparison function, especially when the order of the string could be presented in different ways. Using PCTTRIGRAM allows programmers to identify potential address matches and more accurate matches, regardless of the order of the phrase.

The code to call the function looks like this:

```
DATA ADDRESSES2;
  SET ADDRESSES;
  PCTTRI1=PCTTRIGRAM(ADDR1,ADDR2);
  PCTTRI2=PCTTRIGRAM(ADDR2,ADDR1);
  GED_SCORE=COMPGED(ADDR1,ADDR2);
  LEV_SCORE=COMPLEV(ADDR1,ADDR2);
  JARO_SCORE=JAROWINK(ADDR1,ADDR2,.1);
RUN;
```

Figure # depicts an address matching example using the PCTTRIGRAM function.

Figure #. Using PCTTRIGRAM To Match Addresses

VIEWTABLE: Work.Addresses2

	ADDR1	ADDR2	PCTTRI1	PCTTRI2	GED_SCORE	LEV_SCORE	JARO_SCORE
1	NO. 9, JALAN KEDIDI 2 SUNGAI ARA BAYAN LEPAS 9	JALAN SG ARA 2, DESA ARA, BAYAN LEPAS	0.5769230769	0.4242424242	1790	22	0.5299873737
2	P.O. BOX 932816	GP STRATEGIES CORPORATION PO BOX 932816	0.28125	1	1610	26	0.43125
3	NO 1 YISHUN AVENUE 7 AGILENT BUILDING	1 YISHUN AVENUE 7	1	0.4137931034	1180	20	0.6183048977
4	81400 SENAI, JOHOR DARUL TAKZI TAMAN TEKNOLOGI JOHOR, PLO 55, JALAN PERSIARAN TEKNOLOGI	TMN TEKNOLOGI JOHOR, 81400 SENAI, PLO 55, JALAN PERSIARAN TEKNOLOGI	0.8867924528	0.7428571429	3860	41	0.4368686869

Figure # illustrates results when comparing addresses using different SAS functions, and the PCTTRIGRAM function, which is a user-defined function. Note, the first observation is a likely match but does have some slight differences "Sungai Ara" instead of "Desa Ara", lowering the score to 0.57 for PCTTRI1. PCTTRI1 is the call of PCTTRIGRAM(addr1,addr2), and PCTTRI2 is the call of PCTTRIGRAM(addr2,addr1). The function is not symmetrical and therefore should be called with the parameters reversed. In the second observation, ADDR1 is completely contained in ADDR2, minus the punctuation. This is why the PCTTRI2 score is 1.0: all of the trigrams in ADDR1 are found in ADDR2. Likewise, for the third observation, ADDR2 is completely contained in ADDR1, resulting in a PCTTRI1 score of 1.0. Where this function really shines is the fourth observation. The two addresses are clearly equal upon human inspection, but the order is different. The PCTTRI1 score is 0.88 (high) and the PCTTRI2 score is 0.74, also rather high. Both scores indicate a likely match between the two addresses.

Threshold setting: As the programmer, you can set the match minimum threshold anywhere: generally, 0.5 will return somewhat likely matches but not very exact matches. Of course, a threshold of 1.0 will return *only* exact

matches, which is not our desired goal here. Setting the threshold at 0.7 is a good start for matching, and you may want to do a conditional threshold:

IF PCTTRI1 GE .7 OR PCTTRI2 GE .7 THEN MATCH=1;

If you plan on utilizing the PCTTRIGRAM function for blocking by address, I would recommend a slightly lower threshold of 0.5.

PHONE NUMBER MATCHING FUNCTION

Matching on phone number may seem rather easy at first, but if the phone numbers are presented with different punctuation, spacing, or the international country code (i.e. +1 for U.S.), then matching phone numbers can be a challenge. Also to consider are changes in area code where the last 7 digits are the same, or cases where all digits are the same except the last digit, perhaps indicating the persons started cell service the same day, or perhaps work for the same corporation. The following phone function is designed to identify difficult matches and assign a score, with a max of 10 points indicating a perfect match. A score of 6 indicates the area code is different, but the last 7 digits are the same; and a score of 4 indicates the Levenshtein distance between the phone numbers is less than or equal to 2. A phone score of 3 indicates the first six digits are equal.

PHONESCORE FUNCTION

The following is the PHONESCORE function, developed using SAS's user-defined function capability:

```
proc fcmp outlib=work.funcs.test;
  function phonescore(s1$,s2$);
   phonescore=0;
   p1=tranwrd(s1,'+1','');
   p2=tranwrd(s2,'+1','');
   p1=compress(p1,,'sp');
   p2=compress(p2,,'sp');
   p1=compress(p1,,'kd');
   p2=compress(p2,,'kd');
   put p1= p2=;
   if p1 not in('',' ') and p2 not in('',' ') then
   do;
     if substr(p1,1,6)=substr(p2,1,6) then phonescore=3;
     if complev(p1,p2) le 2 then phonescore=4;
     if substr(p1,4,7)=substr(p2,4,7) then phonescore=6;
     if p1=p2 then phonescore=10;
   end;
   return(phonescore);
   endsub;
quit;
```

I created a mock data set to display how PHONESCORE works, with two columns, phone1 and phone2. I then called the function as follows:

phone_score = PHONESCORE(phone1,phone2);

The results of the PHONESCORE matching are shown in Figure #:

Figure #. PHONESCORE Function Match Results

	phone1	phone2	phone_score
1	703-228-3456	(703) 228-3456	10
2	571-989-2144	571 989 2143	4
3	814-237-1234	(814)237-1234	10
4	302-525-8998	312-525-8998	6
5	703-237-8159	571-237-8159	6
6	604-567-0012	+1 604-567-0013	4

MATCHING ON DATES OF BIRTH

Matching on date of birth is a common task for SAS programmers, but as we know, data integrity and lack of standardization can prove to be a problem when matching. The following user-defined function handles cases where the day, month, and/or year are different, or the month and day are transposed. Scores range from 0 to a max of 12, where the dates of birth are exactly the same. A score of 4 indicates only the years are equal; a score of 6 indicates the month and day were likely transposed; a score of 8 indicates one of three situations: 1) the year and month are equal, but the day is different, 2) the year and day are equal, but the month is different, or 3) the month and day are equal but the year is different. I created a function called DOB_SCORE that inputs two dates of birth as parameters, and returns a score from 0 to 12, with 12 indicating a perfect match. Below is the code for this function:

DOB_SCORE FUNCTION

```
proc fcmp outlib=work.funcs.test;
function dob_score(dob1,dob2) ;
  dob_score=0;
  if dob1 ne . and dob2 ne . then
  do;
    if year(dob1)=year(dob2) then dob_score=4;
    if year(dob1)=year(dob2) and month(dob1)=day(dob2) then dob_score=6;
    if year(dob1)=year(dob2) and month(dob2)=day(dob1) then dob_score=6;
    if year(dob1)=year(dob2) and month(dob1)=month(dob2) then dob_score=8;
    if year(dob1)=year(dob2) and day(dob1)=day(dob2) then dob_score=8;
    if month(dob1)=month(dob2) and day(dob1)=day(dob2) then dob_score=8;
    if dob1=dob2 then dob_score=12;
  end;
return(dob_score);
endsub;
quit;

options cmplib=(work.funcs);

data dob1;
format dob1 dob2 mmddyy10.;
informat dob1 dob2 mmddyy10.;
infile datalines dlm=',' dsd;
input dob1 dob2 ;
datalines;
08/01/1970, 08/01/1970
08/01/1970, 08/02/1970
08/01/1970, 08/01/1972
08/01/1970, 01/08/1970
08/01/1870, 08/01/1970
;
run;

data dob2;
  set dob1;
  dob_score = dob_score(dob1,dob2);
run;
```

Figure #. Date of Birth Matching Results

VIEWTABLE: Work.Dob2

	dob1	dob2	dob_score
1	08/01/1970	08/01/1970	12
2	08/01/1970	08/02/1970	8
3	08/01/1970	08/01/1972	8
4	08/01/1970	01/08/1970	6
5	08/01/1870	08/01/1970	8

CONCLUSION

Since the addition of PROC FCMP, SAS programmers have the capability to develop user-defined functions. In this paper, I described some user-defined functions that may be useful for comparing names, strings, addresses, phone numbers, and dates of birth. In one example, I used the SAS hash object to conduct a nickname lookup, but the majority of this paper focuses on applying PROC FCMP to solve fuzzy matching challenges. Happy programming!

REFERENCES

Wright, Glenn. Paper 76-128. *Probabilistic Record Linkage in SAS*
http://www.wuss.org/proceedings11/Papers_Wright_G_76128.pdf

Loren, Judy. Paper 029-2008. *How Do I Love Hash Tables? Let Me Count The Ways!*
http://www2.sas.com/proceedings/forum2008/029-2008.pdf

Jaro-Winkler: http://stackoverflow.com/questions/6865019/jaro-winkler-string-comparison-function-in-sas

RECOMMENDED READING

- Wright, Glenn. Paper 76-128. *Probabilistic Record Linkage in SAS*
  http://www.wuss.org/proceedings11/Papers_Wright_G_76128.pdf
- Loren, Judy. Paper 029-2008. *How Do I Love Hash Tables? Let Me Count The Ways!*
  http://www2.sas.com/proceedings/forum2008/029-2008.pdf
- Wright, Glenn. Paper 2997-2 *Transitive Record Linkage Using SAS Hash Objects*
  http://www.wuss.org/proceedings10/databases/2997_2_DDI-Wright.PDF
- Carpenter, Arthur Paper 139-2013 *Using PROC FCMP To The Fullest: Getting Started and Doing More*
  http://support.sas.com/resources/papers/proceedings13/139-2013.pdf

CONTACT INFORMATION

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

Name:	Christine L. Warner
Enterprise:	Integrity Management Services, LLC and Automated Auditors, LLC
City, State ZIP:	Alexandria, VA 22315
Work Phone:	571-606-7743
E-mail:	cwarner.autoaudit@gmail.com and cwarner@integritym.com
Web:	http://www.integritym.com and http://www.autoaudit.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.