Paper 3104-2015

Data Management Techniques for Complex Healthcare Data

Gabriela Cantu-Perez MPH, Christopher Klekar MPH, MBA, Baylor Scott&White Health

ABSTRACT

Data sharing through healthcare collaboratives and national registries creates opportunities for secondary data analysis projects. These initiatives provide data for quality comparisons as well as endless research opportunities to external researchers across the country. The possibilities are bountiful when you join data from diverse organizations and look for common themes related to illnesses and patient outcomes. With these great opportunities comes great pain for data analysts and health services researchers tasked with compiling these data sets according to specifications. Patient care data is complex, and, particularly at large healthcare systems, might be managed with multiple electronic health record (EHR) systems. Matching data from separate EHR systems while simultaneously ensuring the integrity of the details of that care visit is challenging. This paper demonstrates how data management personnel can use traditional SAS® PROCs in new and inventive ways to compile, clean, and complete data sets for submission to healthcare collaboratives and other data sharing initiatives. Traditional data matching methods such as SPEDIS are uniquely combined with iterative SQL joins using the SAS® functions INDEX, COMPRESS, CATX, and SUBSTR to yield the most out of complex patient and physician name matches. Recoding, correcting missing items, and formatting data can be efficiently achieved by using traditional functions such as MAX, PROC FORMAT, and FIND in new and inventive ways.

INTRODUCTION

Data sharing through healthcare collaboratives and national registries creates opportunities for secondary data analysis projects. These initiatives provide data for quality comparisons as well as endless research opportunities to external researchers across the country. The possibilities are boundless when you join data from diverse organizations and look for common themes related to illnesses and patient outcomes. With these great opportunities comes great pain for data analysts and health services researchers tasked with compiling these data sets according to specifications. Patient care data is complex, and, particularly at large healthcare systems might be managed with multiple electronic health record (EHR) systems.

Additionally, electronic health record systems were designed to capture clinical data for the purpose of patient care and may not be ideally formatted for analytic needs. Therefore, data analysts are tasked with compiling these complex data to be used for these initiatives.

This paper is designed for midlevel SAS programmers and analyst and will address new and innovate ways to use a series of SAS® favorite functions such as, COMPRESS, SUBSTR, INDEX, CATX, SPEDIS, MAX, and FIND. It will also highlight the use of SAS PROC SQL and PROC FORMAT.

SERIAL SQL MATCHES ON NAME

The match demonstrated in this paper will unite physician's information obtained from one EHR to their national identifier stored in another EHR. Since the two systems do not have common keys that can be matched upon, it is required to match the physicians to themselves using names alone. As this example will show, matching on text strings will pose many challenges. Some issues shown in this paper are:

1. Extensions on the last name, such as "Jr", "II"
2. Use of nicknames or shortened first names like first initial only
3. Special characters included in the last name, such as "-", " ' ", " ` "
4. Misspellings in the first or last name

A successful match will be achieved using serial Structured Query Language (SQL) joins that take advantage of SAS functions like SPEDIS, a familiar fuzzing matching favorite, as well as a combination of SUBSTR, INDEX, and COMPRESS.

Figure 1 contains a schematic of the match sequence that this paper will demonstrate:

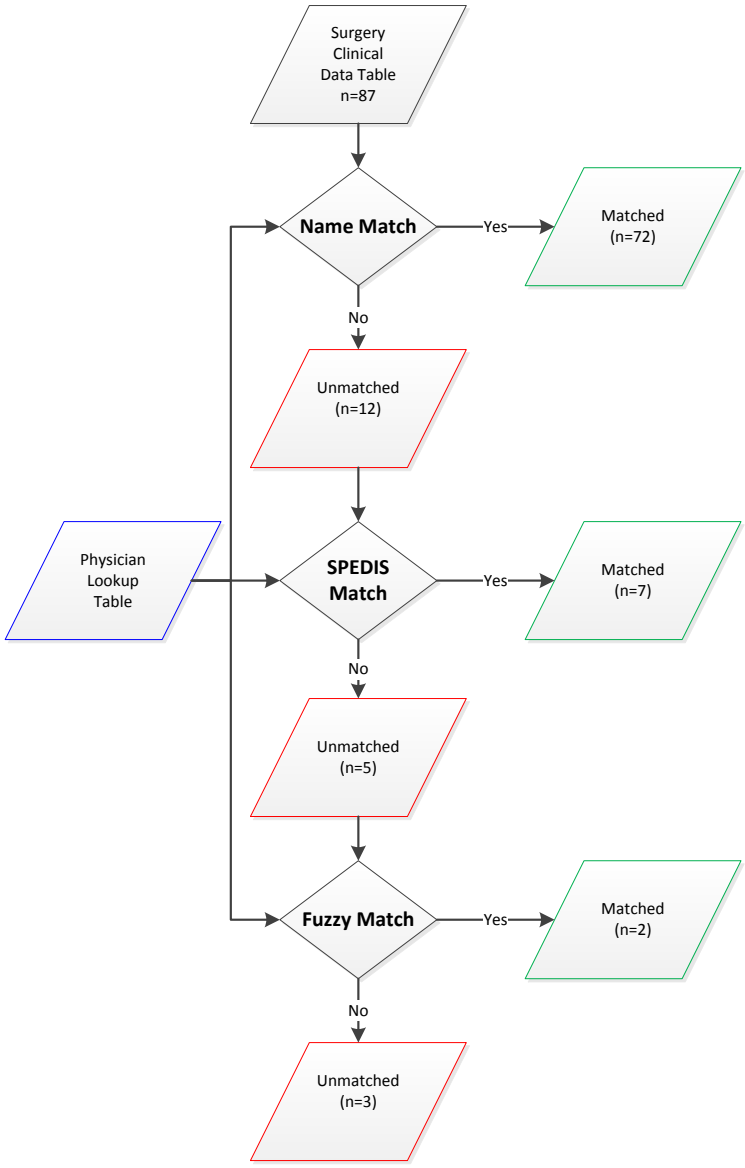

Figure 1 Schematic for SQL serial match sequence

SQL JOIN 1: NAME ALONE

The first join in this example will match strictly on name. This is the simplest join; one that will hopefully yields the most results as it is a simple one-to-one match on first and last name. However, in this example there are twelve physicians' that do not match using the name match. The individuals in Table 1 without the variables "PHYSICIAN_FIRST_NAME" and "PHYSICIAN_LAST_NAME" and the national identifier "NPI" populated are those that did not match to our physician lookup table. They will require subsequent matching strategies.

The NPI is populated for some of the physicians in the lookup table, but not all. The second red boxed area shows an example of other data quality issues to be detailed in the subsequent paper. Here the same provider has two records that have matched to the surgical clinical data, one with a missing national identifier (NPI) and a second record that contains the desired information.

Table 1 shows the results of the SQL join on name alone with highlighted non-matched records and other data quality issues:

FIRSTNAME	LASTNAME	PHYSICIAN_LAST_NAME	PHYSICIAN_FIRST_NAME	NPI
RON	ANDERSON-TODD			.
SAM	BOONE			.
CHARLES	GAYDEN			.
GRIFIN	GRAEHL			.
R MARK	LANCASTER			.
NATHAN	O`BRIEN			.
G HILL	ROBERTS			.
BEN	SNOW			.
JACK	STASIKOWSKI			.
BRYAN	TODD			.
STEVEN	WILLIAMS II			.
KIRK	WINTER			.
EDWARD	PETERS	PETERS	EDWARD	2835913061
NATHAN	TABA	TABA	NATHAN	3440049305
ALAN	GLOGAU	GLOGAU	ALAN	3702769401
LINDSEY	YOUNG	YOUNG	LINDSEY	3096688151
RICK	ELLIS	ELLIS	RICK	3925111077
JOSHUA	DONEGAN	DONEGAN	JOSHUA	3945020071
LANCE	ROUX	ROUX	LANCE	3401642733
PHILLIP	MEHTA	MEHTA	PHILLIP	3825863345
ERIC	GLIDDEN	GLIDDEN	ERIC	2854254063
ADAM	JONES	JONES	ADAM	.
ROGER	ROYE	ROYE	ROGER	2613698979
SIMON	SANDERS	SANDERS	SIMON	2692565697
RICHARD	TAYLOR	TAYLOR	RICHARD	3722906413
TOM	WESTKAEMPER	WESTKAEMPER	TOM	3864300939
BRODY	BOLLINGER	BOLLINGER	BRODY	3603674823
JOSEPH	LEMMON	LEMMON	JOSEPH	3298997589
JEFF	MOORE	MOORE	JEFF	2026218553
VIRGIL	COLEMAN	COLEMAN	VIRGIL	2248089419
HENRY	KAZEWYCH	KAZEWYCH	HENRY	2126907135
ALEXANDER	MOLLABASHY	MOLLABASHY	ALEXANDER	3500743515
PETER	PETERS JR	PETERS JR	PETER	2712997845
PETER	FREUDIGMAN JR	FREUDIGMAN JR	PETER	2894528469
THOMAS	MEDLOCK	MEDLOCK	THOMAS	.
THOMAS	MEDLOCK	MEDLOCK	THOMAS	2167210163
BARRETT	SCHUBERT	SCHUBERT	BARRETT	.

Table 1 SQL join on name results

Those twelve that highlighted in red will be re-matched using the SPEDIS function. In order to do that, it is necessary to subset the above SQL join result table to contain only the unmatched data and then perform the second join in the serial SQL join process described in this paper.

SQL JOIN 2: SPEDIS JOIN

The dataset shown in Table 2, is the subset of those physicians that did not match on name alone. The SPEDIS function will be used to create a new variable named “VALUE” which contains a SAS calculated name difference between the surgery clinical systems name and that of the physician lookup dataset. A threshold of 25 has been set using a where statement, that will ensure no erroneous matches will be included. The threshold for the variable “VALUE” was derived iteratively by first setting the threshold value high at 100 and performing a match. The results were reviewed to determine a reasonable number for “VALUE” based on the level of error in the name matches.

Table 2 is the subset of the data matched on name alone; it contains 12 unmatched records to be matched using the SPEDIS function:

	FIRSTNAME	LASTNAME
1	RON	ANDERSON-TODD
2	SAM	BOONE
3	CHARLES	GAYDEN
4	GRIFIN	GRAEHL
5	R MARK	LANCASTER
6	NATHAN	O'BRIEN
7	G HILL	ROBERTS
8	BEN	SNOW
9	JACK	STASIKOWSKI
10	BRYAN	TODD
11	STEVEN	WILLIAMS II
12	KIRK	WINTER

Table 2 Data to be joined using SPEDIS function

The code below demonstrates the SPEDIS match using the CATX function to combine the "LASTNAME" and "FIRSTNAME" variables from the surgical clinical dataset and compare that value to the combined "PHYSICIAN_LAST_NAME" and "PHYSICIAN_FIRST_NAME" from the physician lookup table:

```
proc sql;
create table physician_match as
   select distinct
      a.firstname
      ,a.lastname
      ,b.physician_last_name
      ,b.physician_first_name
      ,b.NPI
      ,SPEDIS(CATX(',',a.lastname, a.firstname),
       CATX(',',b.physician_last_name, b.physician_first_name)) as value
from physician_straight_match a
   left join physician_lookup b
       on a.lastname=b.physician_last_name
       and b.physician_first_name ne ''
    where calculated value <=25
    order by calculated value;
quit;
```

The results of the SPEDIS join produced 6 new matches. The errors in the first names range from missing letters to the use of short name forms. After verifying the national identifiers using the internet, the matches obtained were correct and the threshold for that variable was appropriate. All NPI numbers do not need to be verified, for the purpose of this project only those with the two highest values were verified.

Table 3 shows the results following the SPEDIS match, including the "VALUE" variable that shows the maximum value necessary to achieve a successful match:

FIRSTNAME	LASTNAME	PHYSICIAN_LAST_NAME	PHYSICIAN_FIRST_NAME	NPI	VALUE
GRIFIN	GRAEHL	GRAEHL	GRIFFIN	2046216949	1
JACK	STASIKOWSKI	STASIKOWSKI	J	3239844031	6
SAM	BOONE	BOONE	SAMSON	3278444761	16
BRYAN	TODD	TODD	B	2066274501	14
R MARK	LANCASTER	LANCASTER	RICK	3581554889	23
G HILL	ROBERTS	ROBERTS	G. HILL	3805694885	3
CHARLES	GAYDEN	GAYDEN	CHARLE	3844010175	2

Table 3 Results of the SPEDIS match

SQL JOIN 3: FUZZY MATCH

There will be one more match attempt to successfully find the NPI data for the last 4 records. Once again the data was reduced to just the unmatched data.

Table 4 contains the unmatched data following the SPEDIS match:

	FIRSTNAME	LASTNAME
1	RON	ANDERSON-TODD
2	NATHAN	O`BRIEN
3	STEVEN	WILLIAMS II
4	BEN	SNOW
5	KIRK	WINTER

Table 4 Data table for the fuzzy match

The last match sequence will focus on stripping the first and last names of any unusual characters using the compress function in the code snippet below (two last names contain characters that may prohibit matching "-" and "`").

Surgery table code:

```
SUBSTR((compress(A.LASTNAME,"'`-" )),1,INDEX(compress(A.LASTNAME,"'`-" ),
' ')-1)
```

Lookup table code:

```
SUBSTR((compress(B.PHYSICIAN_LAST_NAME,"'`-" )),1,INDEX(compress
(B.PHYSICIAN_LAST_NAME,"'`-" ),' ')-1)
```

Additionally, the code will also remove any extension on the last names using the combined functions of SUBSTR and INDEX. The SUBSTR function argument one, is the cleaned "PHYSICIAN_LAST_NAME" with the special characters removed, the second argument is the number "1" telling the SUBSTR function to begin at the first character in this clean last name, the final argument in the SUBSTR function uses the INDEX function and returns a value to SAS of the first space found "-1". The SUBSTR function used in this combination truncates the last name to the last character before the space, leading to the removal of "II".

Lastly, a SPEDIS calculation was included to determine the differences in the physician's first name. The combination of these cleaning maneuvers performed both on the surgery clinical data table and the physician lookup table will optimize my last attempt to match data between the two sources.

The following code was used to create a fuzzy match on clean versions of the physician's last name from each dataset while calculating the difference in the first name variables from both sources:

```
proc sql;
create table physician_match_last as
   select distinct
       a.firstname
      ,a.lastname
      ,b.physician_last_name
      ,b.physician_first_name
      ,b.NPI
       ,SUBSTR((COMPRESS(a.lastname,"'`-" )),1,INDEX(COMPRESS
       (a.lastname,"'`-" ),' ')-1) as clean_demo_last
      ,SUBSTR((COMPRESS(b.physician_last_name,"'`-" )),1,INDEX(COMPRESS
       (b.physician_last_name,"'`-" ),' ')-1) as clean_lu_last
      ,SPEDIS(firstname, physician_first_name)as value
from physician_for_fuzzy a
   left join physician_lookup b
      on SUBSTR((COMPRESS(a.lastname,"'`-" )),1,INDEX(COMPRESS
```

```
          (a.lastname,"'`-" ),' ')-1)
                              =
          SUBSTR((COMPRESS(b.physician_last_name,"'`-" )),1,INDEX(COMPRESS
          (b.physician_last_name,"'`-" ),' ')-1)

    where calculated value <=25
    order by value;
quit;
```

Table 5 contains the result of the fuzzy match SQL join, showing two new records that were matched to the physician lookup table:

FIRSTNAME	LASTNAME	PHYSICIAN_LAST_NAME	PHYSICIAN_FIRST_NAME	NPI	CLEAN_DEMO_LAST	CLEAN_LU_LAST	VALUE
STEVEN	WILLIAMS II	WILLIAMS	STEVEN	3460410155	WILLIAMS	WILLIAMS	0
NATHAN	O'BRIEN	O'BRIEN	NATHAN	3359078461	OBRIEN	OBRIEN	0

Table 5 Results of the fuzzy match

The variables created using the complex SUBSTR statement “CLEAN_DEMO_LAST” (the last name from the surgery clinical data) and “CLEAN_LU_LAST” (the last name from the physician lookup table) yield clean truncated physician last names that match exactly. Additionally the SPEDIS “VALUE” for the differences in the first names were both 0, meaning that with these alterations they yield perfect matches.

The remaining dataset contains 3 unmatched providers who do not have data in the physician lookup dataset and therefore will remain unmatched.

UNIQUE USES OF THE MAX FUNCTION

Once data has been matched, it may be necessary to clean up the correctly identified records. Of particular importance is the missing NPI numbers, as shown by the red box in Table 1. The MAX function can be used to remove duplicates, while simultaneously correcting missing NPI’s.

The code below ensures that there are no missing NPI numbers if they are available for at least one of the matched records:

```
proc sql;
create table physician_cleanup as
   select distinct
      max(NPI) as clean_NPI
      ,lastname
      ,firstname
from physician_final_file
   group by lastname, firstname;
quit;
```

Table 6 shows the results of the MAX function and the de-duplication of the final physician matched dataset. It replaces previously missing NPIs with those listed for the same physician elsewhere in the dataset. It uses the GROUP BY statement to ensure that only the NPI for the same physician are evaluated together:

CLEAN_NPI	LASTNAME	FIRSTNAME
2026218553	MOORE	JEFF
2046216949	GRAEHL	GRIFIN
2066274501	TODD	BRYAN
2066534407	STEHLY	DARREN
2086446691	SAZY	TOM
2107054313	HEINRICH	TOM
2126907135	KAZEWYCH	HENRY
2126926341	VO	DONALD
2147066827	SCHUBERT	BARRETT
2167210163	MEDLOCK	THOMAS
2167231693	OZUMBA	DANIEL
2248089419	COLEMAN	VIRGIL
2288518133	LAIS	DERRICK
2288566299	WENGER	RANDY
2308913089	GILBERT	STEVEN
2349084813	KHAN	ASHTON
2511061251	CRAWFORD	SEAN
2551441669	FLEAGER	SANDRA
2613698979	ROYE	ROGER
2633944727	BERRY	ROGER
2692565697	SANDERS	SIMON
2712646459	JENNINGS	DEBRA
2712997845	PETERS JR	PETER
2753449127	VINEYARD	JAY
2773389199	WINTER	KIRK
2835913061	PETERS	EDWARD
2854254063	GLIDDEN	ERIC
2894517217	MOFFETT	JOHN
2894528469	FREUDIGMAN JR	PETER
2894935013	HONIG	CRAIG
2914753655	KLEUSER	TERRY
2934832605	GOLDMAN	ALAN
2975365879	JONES	ADAM
2995746795	CATES	BRUCE

Table 6 Data cleaning results using the MAX function

Using the MAX function to aggregate records can be useful on different occasions. For example, another use can be flagging a patient's "ever/never" encounter. If there are multiple measurements for a patient taken over the course of their stay in the hospital, it may be necessary to find out if they have ever had an elevated heart rate or blood pressure, for example. First it will be necessary to create a flag variable for each encounter as "positive" or "elevated" using a CASE or IF/THEN statement. Then using the MAX function and GROUP BY option it is possible to look at the patient test results and see if there was an occurrence.

See the code below, which was used to identify whether a patient ever had a positive confusion assessment measurement (CAM), a test used to detect delirium:

```
proc sql;
create table cam_positive as
   select distinct
      casecode
      ,value
      ,case when value='positive' then 1
            when value='negative' then 0
            ends as cam_score
from vitals_tests;
quit;
```

Error! Reference source not found. the recording of the value variable into either 0 for negative results or 1 for positive results:

CASECODE	Value	CAM_SCORE
129246	negative	0
129246	positive	1
129590	negative	0
129650	negative	0
129774	negative	0
129790	negative	0
129826	negative	0
129830	negative	0
129842	negative	0
129858	negative	0
129868	negative	0
130004	negative	0
130034	negative	0
130052	negative	0
130080	negative	0
130080	positive	1
130254	negative	0
130280	negative	0
130282	negative	0
130348	negative	0
130498	negative	0
130506	negative	0
130510	negative	0
130700	negative	0
130702	negative	0
130898	negative	0
495870	negative	0
495870	positive	1
503218	negative	0
503218	positive	1
503308	negative	0
503806	negative	0
504114	negative	0
504134	negative	0
504234	negative	0

Table 7 Recoding data as dichotomous outcomes

The code below uses the MAX statement to group patient information by “CASECODE” and will determine if the patient has ever had a positive CAM test:

```
proc sql;
create table ever_never_pos as
  select distinct
      casecode
     ,MAX(cam_score) as ever_pos
from cam_positive
   group by casecode;
quit;
```

The boxed “CASECODE” in Tables 7 and 8 demonstrate the deduplication of records from one positive and one negative result to one result with a value of 1 or “yes” for the variable “EVER_POS” used to denote whether this patient has ever had a positive test result.

Table 8 shows the output from the MAX statement:

CASECODE	ever_pos
129246	1
129650	0
129774	0
129790	0
129826	0
129830	0
129842	0
129858	0
129868	0
130004	0
130034	0
130052	0
130080	1
130254	0
130280	0
130282	0
130348	0
130498	0
130506	0
130510	0
130700	0
130702	0
130898	0
495870	1
503218	1
503308	0
503806	0
504114	0
504134	0
504234	0
504308	0
504376	1
504386	0
504404	0

Table 8 Results of MAX function used to flag positive outcomes

COMBINING FIND AND LOGIC STATEMENTS

Next this paper will demonstrate the use of the SAS FIND function in combination with traditional IF/THEN statements to mine text fields. In this instance, it is necessary to ascertain the laterality (or side of the body) of a surgery. This EHR captures this information in a text field that will need to be mined in order to determine procedure side.

Figure 2 demonstrates the free text data that will need to be searched:

CASECODE	PRIM_LATERALITY	PRIM_PROCEDURE_DESCRP	PRIM_SURG_DESCRP
693677		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
698632		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-LEFT
699236		ARTHROPLASTY,TOTAL HIP	ARTHROPLASTY,TOTAL HIP-LEFT
699638		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
700664		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
701769		ARTHROPLASTY,TOTAL HIP	TOTAL RIGHT HIP REPLACEMENT (STRYKER).
701819		ARTHROPLASTY,TOTAL KNEE	LEFT TOTAL KNEE REPLACEMENT (STRYKER).
703103		ARTHROPLASTY,TOTAL HIP	RIGHT TOTAL HIP REPLACEMENT (STRYKER).
703362		ARTHROPLASTY,TOTAL HIP	RIGHT TOTAL HIP REPLACEMENT (STRYKER).
703381		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
703841		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
703916		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT WITH BONE MARROW ASPIRATION FOR PLATELET RICH PLASMA GEL
704249		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
704515		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
705253		ARTHROPLASTY,TOTAL KNEE	RIGHT TOTAL KNEE REPLACEMENT (STRYKER).
705534		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
706810		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
706857		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
707234		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
708792		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-LEFT
709015		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-LEFT AND LATERAL RETINACULAR RELEASE
709254		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-LEFT
709602		ARTHROPLASTY,TOTAL HIP	ARTHROPLASTY,TOTAL HIP-LEFT
710770		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-LEFT
711626		ARTHROPLASTY,TOTAL HIP	ARTHROPLASTY,TOTAL HIP-LEFT
712069		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
712868		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-LEFT
712876		ARTHROPLASTY,TOTAL KNEE	LEFT TOTAL KNEE REPLACEMENT (STRYKER).
714096		ARTHROPLASTY,TOTAL KNEE	LEFT TOTAL KNEE REPLACEMENT.
714843		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-LEFT
716730		ARTHROPLASTY,TOTAL HIP	RIGHT TOTAL HIP REPLACEMENT (STRYKER).
716803		ARTHROPLASTY,TOTAL KNEE	ARTHROPLASTY,TOTAL KNEE-RIGHT
887052	RIGHT	ARTHROPLASTY,TOTAL HIP	RIGHT ARTHROPLASTY,TOTAL HIP
887048	RIGHT	ARTHROPLASTY,TOTAL KNEE	RIGHT ARTHROPLASTY,TOTAL KNEE
887058	RIGHT	ARTHROPLASTY,TOTAL KNEE	RIGHT ARTHROPLASTY,TOTAL KNEE
887058	RIGHT	ARTHROPLASTY,TOTAL KNEE	RIGHT TOTAL KNEE ARTHROPLASTY
887371	LEFT	ARTHROPLASTY,TOTAL HIP	LEFT ARTHROPLASTY,TOTAL HIP (EXACTECH)
887342		ARTHROPLASTY,TOTAL KNEE,B...	ARTHROPLASTY,TOTAL KNEE,BILATERAL UNASSISTED W NAVIGATION
887420	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT ARTHROPLASTY,TOTAL KNEE
887420	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT ARTHROPLASTY,TOTAL KNEE ()
887405		ARTHROPLASTY,TOTAL KNEE,B...	ARTHROPLASTY,TOTAL KNEE,BILATERAL ASSISTED (STRYKER)
887414		ARTHROPLASTY,TOTAL KNEE,B...	ARTHROPLASTY,TOTAL KNEE,BILATERAL ASSISTED (STRYKER)
887414	BILATERAL	ARTHROPLASTY,TOTAL KNEE,B...	BILATERAL TOTAL KNEE ARTHROPLASTY, ASSISTED
887202	LEFT	ARTHROPLASTY,TOTAL HIP	LEFT ARTHROPLASTY,TOTAL HIP
887865	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT ARTHROPLASTY,TOTAL KNEE
887865	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT TOTAL KNEE ARTHROPLASTY
887844		ARTHROPLASTY,TOTAL KNEE,B...	ARTHROPLASTY,TOTAL KNEE,BILATERAL UNASSISTED
887844	LEFT	ARTHROPLASTY,TOTAL KNEE,B...	LEFT TOTAL KNEE ARTHROPLASTY UNASSISTED
887844	RIGHT	ARTHROPLASTY,TOTAL KNEE	RIGHT TOTAL KNEE ARTHROPLASTY
887855	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT ARTHROPLASTY,TOTAL KNEE
887855	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT TOTAL KNEE ARTHROPLASTY
887861	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT ARTHROPLASTY,TOTAL KNEE
887861	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT TOTAL KNEE ARTHROPLASTY
888194	LEFT	ARTHROPLASTY,TOTAL HIP	LEFT ARTHROPLASTY,TOTAL HIP
888335	RIGHT	ARTHROPLASTY,TOTAL HIP	RIGHT ARTHROPLASTY,TOTAL HIP EXACTECH
888333	RIGHT	ARTHROPLASTY,TOTAL HIP	RIGHT ARTHROPLASTY,TOTAL HIP
888343	LEFT	ARTHROPLASTY,TOTAL HIP	LEFT ARTHROPLASTY,TOTAL HIP EXACTECH
888338	BILATERAL	ARTHROPLASTY,TOTAL KNEE,B...	BILATERAL ARTHROPLASTY,TOTAL KNEE,BILATERAL UNASSISTED EXACTECH
888887	LEFT	ARTHROPLASTY,TOTAL HIP	LEFT ARTHROPLASTY,TOTAL HIP (SMITH & NEPHEW)
888887	LEFT	ARTHROPLASTY,TOTAL HIP	LEFT TOTAL HIP ARTHROPLASTY
888891	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT ARTHROPLASTY,TOTAL KNEE
888891	LEFT	ARTHROPLASTY,TOTAL KNEE	LEFT ARTHROPLASTY,TOTAL KNEE ()
889248	RIGHT	ARTHROPLASTY,TOTAL KNEE	RIGHT ARTHROPLASTY,TOTAL KNEE

Figure 2 Free text surgical information

The dataset contains some information in the variable“PRIM_LATERALITY” that is intended to designate surgical side, however it is not well populated and the remainder will have to be pulled from the other two fields.

Using a combination of the FIND and IF/THEN function will easily locate the patient’s laterality and code it into a new field, while identifying those without laterality specified in either field, see the code below:

```
data laterality;
set find_example;
   right=FIND(prim_surg_descrp, 'RIGHT','I',1);
   left=FIND(prim_surg_descrp, 'LEFT','I',1);
```

```
   bi=FIND(prim_procedure_descrp, 'BILATERAL','I',1);
   bi2=FIND(prim_surg_descrp, 'BILATERAL','I',1);
      IF right >=1 or prim_laterality='RIGHT' THEN lat_right=1;
         ELSE lat_right =0;
      IF left >=1 or prim_laterality='LEFT' THEN lat_left=1;
         ELSE lat_left =0;
      IF bi >=1 or bi2>=1 or prim_laterality='BILATERAL' THEN lat_bi=1;
         ELSE lat_bi=0;

/*CHECKING TO BE SURE EVERYONE HAS ATLEAST ONE LATERALITY FILLED OUT*/
   Total_lat=lat_right+lat_left+lat_bi;
/*IF THERE ARE SPELLING ERRORS OR NO KNOWN LATERALITY THEN THEY ARE
MISSING*/
   IF total_lat=0 THEN lat_unknown=1; ELSE lat_unknown =0;

/*LAT VARIABLE CREATION*/
   IF lat_bi=1 THEN lat='B';
    ELSE IF lat_right=1 AND lat_left=1   THEN lat='B';
    ELSE IF lat_right=1 AND lat_left=0   THEN lat='R';
    ELSE IF lat_left =1 AND lat_right=0  THEN lat='L';
    ELSE IF lat_unknown=1 THEN lat='U';

DROP right left bi bi2 prim_procedure_descrp prim_surg_descrp lat_right
lat_left lat_bi lat_unknown prim_laterality;

run;
```

Table 9 shows the results of the data clean up following the FIND function and IF/THEN statements

	CASECODE	LAT
1	693677	R
2	698632	L
3	699236	L
4	699638	R
5	700664	R
6	701769	R
7	701819	L
8	703103	R
9	703362	R
10	703381	R
11	703841	R
12	703916	R
13	704249	R
14	704515	R
15	705253	R
16	705534	R
17	706810	R
18	706857	R
19	707234	R
20	708792	L
21	709015	L
22	709254	L
23	709602	L
24	710770	L
25	711626	L
26	712069	R
27	712868	L
28	712876	L
29	714096	L
30	714843	L
31	716730	R
32	716803	R
33	717509	L

Table 9 Results of the FIND function

The FIND function correctly identifies the patient surgical laterality and leads to record de-duplication that is a result of the text fields which contained the previous data. Now, even records with missing lateralities have been coded as “U” for unknown.

USING PROC FORMAT TO CATEGORIZE DATA

Data may need to be categorized using very specific definitions; instead of using a series of IF/THEN statements, a PROC FORMAT statement coupled with a PUT statement in PROC SQL can quickly re-categorize the data. In this example, the text “TESTNAME” variable to be categorized using logical observation identifiers names and codes (LOINCs) which are a universal code system for tests, measurements and observations. For example, the test names “Weight (kg)” and “Weight: kg” would be assigned the same LOINC value. Data with different measurement units have been manipulated in a previous step to the desired unit of measure.

Table 10 shows patient vital signs information that will need to be standardized prior to use.

TestName	TESTVALUE	CASECODE
Noninvasive Systolic Blood Pressure	151	120502
Noninvasive Systolic Blood Pressure	152	120502
Height cm	152.4	120502
Temperature in Celsius	36.7	120502
Temperature in Fahrenheit	36.8	120502
Temperature in Celsius	37.1	120502
Temperature in Fahrenheit	37.2	120502
Noninvasive Diastolic Blood Pressure	73	120502
Weight (kg)	75.3	120502
Weight (lbs)	75.3	120502
Weight: kg	75.3	120502
Noninvasive Diastolic Blood Pressure	78	120502
Pulse	90	120502
Pulse	93	120502
Noninvasive Systolic Blood Pressure	102	121034
Height cm	160	121034
Noninvasive Systolic Blood Pressure	172	121034
Temperature in Celsius	36.6	121034
Temperature in Fahrenheit	36.7	121034
Temperature in Celsius	36.9	121034
Temperature in Fahrenheit	36.9	121034
Pulse	48	121034
Weight (kg)	48.5	121034
Weight (lbs)	48.5	121034
Weight: kg	48.534	121034
Pulse	54	121034
Noninvasive Diastolic Blood Pressure	66	121034
Noninvasive Diastolic Blood Pressure	87	121034
Noninvasive Systolic Blood Pressure	145	121142
Height cm	157.4	121142
Noninvasive Systolic Blood Pressure	177	121142
Noninvasive Diastolic Blood Pressure	31	121142
BMI	35.4	121142
BMI	35.5	121142

Table 10 Display of vitals data prior to formatting

The PROC FORMAT code below will define the specific categorizations to be made:

```
proc format;
VALUE $LOINC_CLIN
'BMI'='39156-5'
'BP diastolic','BP Diastolic','Diastolic BP (mmHg)','Noninvasive Diastolic
Blood Pressure'='8462-4'
'BP Systolic','Systolic BP (mmHg)','Noninvasive Systolic Blood
Pressure'='8480-6'
'Height cm','Height In Cm','Height: cm','HEIGHT'='8302-2'
'Initial Diastolic Blood Pressure'='11377-9'
'Initial Systolic Blood Pressure'='11378-7'
```

```
'Current Weight (kg)','Daily Weight','Daily Weight in lbs ozs','Daily
Weight in pounds',
'Weight (kg)','Weight kg','Weight Kilograms','Weight: kg','Weight
(lbs)','Weight lbs','WEIGHT', 'WEIGHT - DAILY'='29463-7'
'Estimated Current Weight: kg','Estimated Current Weight: lbs'='8335-2'
'Temperature', 'Temperature Celsius','Temperature in Celsius','Temperature
in Fahrenheit','Temperature(Degrees Celsius)'='8310-5'
'Pulse','Pulse (beats/min)','Heart Rate','Heart Rate (Beats/Minute)','Heart
Rate (beats/minute)'='8867-4'
;
run;
```

The following PROC SQL code uses a PUT statement to translate the “TESTNAME” to a standardized code stored in the variable “LOINC_CODE” for all tests meeting the criteria:

```
PROC SQL;
CREATE TABLE FORMAT_CATEGOR AS
    SELECT DISTINCT
    CASECODE
    ,PUT(TESTNAME,$LOINC_CLIN.)  AS LOINC_CODE
    ,TESTVALUE  .
    FROM  FORMATS_TORECODE
ORDER BY CASECODE;
QUIT;
```

It is important to remember that names of the “TESTNAME” variable not found in the PROC FORMAT statement will result in truncated versions of the “TESTNAME” variable to meet the length of the resultant formatted data. It is always important to run a data quality check using a PROC FREQ (or other method) following data management techniques.

Table 11 provides the output from the PROC SQL statement and shows the standardization of the “TESTNAME” variable in the new “LOINC_CODE” variable:

CASECODE	LOINC_CODE	TESTVALUE
120502	29463-7	75.3
120502	8302-2	152.4
120502	8310-5	36.7
120502	8310-5	36.8
120502	8310-5	37.1
120502	8310-5	37.2
120502	8462-4	73
120502	8462-4	78
120502	8480-6	151
120502	8480-6	152
120502	8867-4	90
120502	8867-4	93
121034	29463-7	48.5
121034	29463-7	48.534
121034	8302-2	160
121034	8310-5	36.6
121034	8310-5	36.7
121034	8310-5	36.9
121034	8462-4	66
121034	8462-4	87
121034	8480-6	102
121034	8480-6	172
121034	8867-4	48
121034	8867-4	54
121142	29463-7	85.2
121142	29463-7	87.8
121142	29463-7	88.1
121142	39156-5	35.4
121142	39156-5	35.5
121142	8302-2	157.4
121142	8310-5	36.6
121142	8310-5	37
121142	8462-4	31
121142	8462-4	49

Table 11 shows the formatted patient vitals' information

CONCLUSION

SAS functions are very powerful and can be used on a variety of data types. When trying to reformat or clean particularly messy data, sometimes we focus on very few SAS PROCs or functions that we are most familiar with. This paper was intended to take old SAS functions, combine them in new ways in order to create a new approach in the cleaning and prepping of healthcare analytic datasets.

As healthcare continues to evolve, the use of clinical data will only increase. The demand for near real-time data to support decisions regarding evidence based practices and clinically effective healthcare will continue to grow. Analyst and programmers will have to prepare themselves and their SAS toolbox with a variety of methods to overcome these obstacles.

REFERENCES

Gershteyn, Yefim PhD. April 2000. Paper 86-25 "Use of SPEDIS Function in Finding Specific Values." Indianapolis, IN : http://www2.sas.com/proceedings/sugi25/25/cc/25p086.pdf.

Huang, Choa; Fu, Yu. April 2013. Paper 257-2013 "Top 10 Most Powerful Functions for PROC SQL." San Francisco, CA: http://support.sas.com/resources/papers/proceedings13/257-2013.pdf.

CONTACT INFORMATION

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

Gabriela Cantu-Perez
Baylor Scott&White Health
Email: Gabriela.Cantu@baylorhealth.edu

Christopher Klekar
Baylor Scott&White Health
Email: Christopher.Klekar@baylorhealth.edu