Paper 277-2013

Speed it Up: Using SAS® to Automate Initial Discovery Practices

Mariya Karimova, AdvanceMed Corporation, an NCI Company; Christine John, AdvanceMed Corporation, an NCI Company

ABSTRACT

Healthcare investigations frequently begin with a tip containing very little provider information. This presentation attempts to use SAS® to automate the initial discovery process, turning a name into a full overview of the provider. Multiple data sources are combined, which oftentimes require fuzzy matching to resolve conflicting identifiers. The program utilizes INFILE URL and SAS text functions to obtain meaningful information from various websites. It further utilizes SAS ODS and SAS/GRAPH® to create a single standard PDF report; which provides a visualization of provider billing patterns, summarizes their affiliations, and embeds hyperlinks to original web-based resources.

Additional topics that are discussed include: creating a script for multiple users, paramaterization, utilizing system variables, and SAS® 9.2 to SAS® 9.3 conversion.

INTRODUCTION

FBI Special Agent in Charge David Welker stated, "The United States spends more than $2.5 trillion on health care annually and rough estimates indicate that anywhere from 3 to 10 percent of all health care expenditures are attributed to fraud. Fraud in turn drives up the cost of health care and health care insurance which impacts you, the consumer, in the cost and quality of your health care"

"Health care fraud costs the country an estimated $80 billion a year. And it's a rising threat, with national health care spending topping $2.7 trillion and expenses continuing to outpace inflation."

Fbi.gov

Oftentimes, at the very beginning of an investigation, we don't have much more than a name and an NPI (National Provider Identifier) to begin our search. Finding all the additional information can be a daunting task for a busy analyst. Sometimes the decision between opening an investigation or closing a complaint without further actions must be made within the first few days. We know that most medical professionals are honest, but in the rare case of a dishonest provider, the goal is to identify an issue in near-real time.

Provider identifiers are merged from each multiple enrollment data sources and compared with the identifiers available in Medicare and Medicaid claims data in order to create the most exhaustive list of identifiers possible. Over a period of time, we noticed many of the providers involved in our investigations had previously been brought before their State Medical Board. According to the Federation of State Medical Boards, the number of serious actions per 1000 Physicians from 2008-2010 was 2.98. Another potential clue may be affiliation with other entities. One of the websites that allows quick and easy social network search is Corporation Wiki.

WEB-CRAWLING

There is a wealth of information available through public websites. As SAS programmers, we couldn't stop wondering if there exists a method to extract information in an automated way and present it to our end customer in a compact form.

One of the underused capabilities of SAS is the INFILE URL statement, which allows us to bring in HTML code of a specified webpage. The example shown here is from Corporation Wiki. As a warning, this excersice is purely academic in nature and is designed to show the capabilities of SAS. A programmer should consider copyrights and other legal implication before utilizing any web crawling techniques for a commercial use. (Please note, all indiviudals' names in these examples are ficticious.)

The ultimate goal here was to parse out useful information presented in the picture below and transform it into a blurb about known relations of the person of interest. In this case, the fact that 'Jane Practitioner' is affiliated with 'Jan Tower' may be quite telling if we have a previous complaint on 'Jan Tower'.

In order to get from picture on the left to the one on the right (Figure 1), we need to look at the underlying HTML code. But first, we need to address the issue of identifying the appropriate URL. On Figure 2, you can see the correlation between the search page and the underlying code. We want to find the URL of the actual webpage related to the person of interest. Luckily, most webpages have addresses that are easily parsable. Corporation Wiki search

page has URL in a form **http://www.corporationwiki.com/search/results?term=jane+practitioner**, where the last two terms represent the name to be searched.

The following code will do the job:

```
FILENAME SEARCH URL "&SLINK";

DATA RWORK.PROFILE_&NPI._WIKI2;
INFILE SEARCH LRECL=32767 ;
LENGTH LINK3 $270;
INPUT @ &LINK1.;
LINK3=CAT('"http://www.corporationwiki.com','/',COMPRESS(&LINK1.,'"
'),SCAN(LINK1,1,'>'));
RUN;
```

SLINK in line 1 refers to a macro variable containing the concatenation of the base web address and known name. LRECL=32767 is a crucial piece to the code above. This allows us to infile as much text as possible, since there is a good chance that the string we are looking for is going to be longer than default 200 characters. Finding the string with the URL in the text massive may be a difficult task. It may not have a fixed length or position, and it definitely doesn't use any type of standard delimiter. However, in most cases we can find an exact string that would preface the hyperlink. In line 4, we are using input **@ 'string'** to input all characters **after** the specified string.

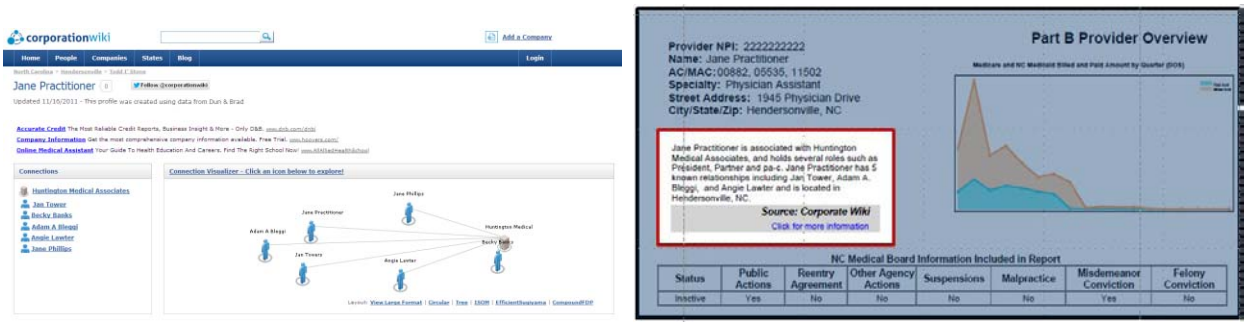

Figure 1. Corporation Wiki – Website and the Output

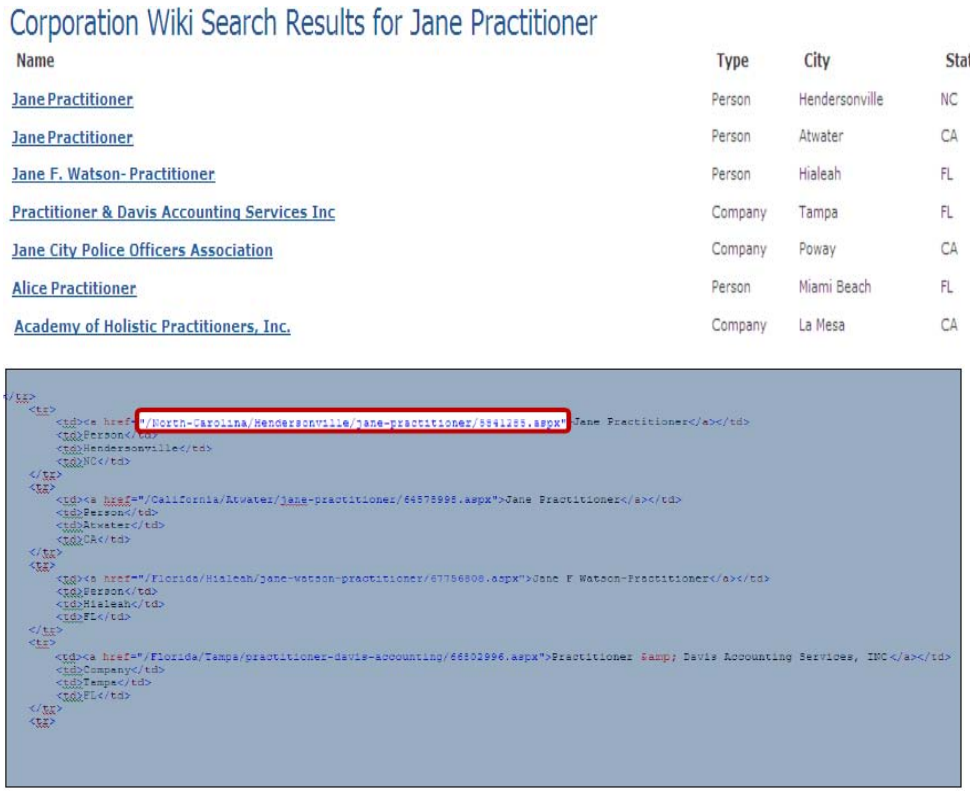

Figure 2. Corporation Wiki Search Page

As you may suspect, a search by name usually returns more than one result. Even adding city and state may not be enough to resolve the identity. Certain decision rules have to be applied to establish the true match. In our case, we have started by intaking all possible results (block 1) and identifying anybody who may be related to the medical profession, by searching for words like Doctor, MD, Physician, etc. (block 2) If we have a common name and a larger city, this still may not be specific enough, so we had to go through a secondary check of any known associates who may be mentioned on Corporation Wiki (block 3). Finally, we wrote a macro around the code in order to infile all matches identified on a search page (block 0)

Block 0

```
%MACRO MULTIPLE;

I = 1 %TO &N_OBS.;

PROC SQL NOPRINT;
       SELECT LINK3 INTO: LINK3 FROM RWORK.PROFILE_&NPI._WIKI2
       WHERE N_OBS=%EVAL(&I.);
QUIT;
```

Block 1

```
FILENAME SEARCH URL &LINK3.;

/* OUTPUT - LINK3*/

       DATA RWORK.PROFILE_&NPI._WIKI3_&I.;
              INFILE SEARCH LRECL=32767 ;
              LENGTH LINK3 $270. LINK2 $270.;
              INPUT @ '"DESCRIPTION" CONTENT="' LINK2 $270;
                     LINK3=SCAN(LINK2,1,'"');
                     POSITION=FINDW(LINK3,'META','<')-1;
       KEEP LINK3 POSITION;
       RUN;
```

Block 2

```
       DATA RWORK.PROFILE_&NPI._WIKI3_&I.;
              SET RWORK.PROFILE_&NPI._WIKI3_&I.;
              WHERE UPCASE(LINK3) LIKE '%MED%' OR UPCASE(LINK3) ?
              'DOCTOR' OR LINK3 ? 'MD' OR UPCASE(LINK3) LIKE '%HEALTH%' OR
UPCASE(LINK3)
              LIKE '%PHYS%';
       RUN;
```

Block 3

```
       PROC SQL;
       CREATE TABLE RWORK.PROFILE_&NPI._WIKI3A_&I. AS SELECT DISTINCT A.*,
              CASE WHEN UPCASE(LINK3) ? UPCASE(STRIP(TRIM(PAYEE_NAME))) THEN 'Y'
              END AS IND,
              CASE WHEN POSITION = -1 THEN LINK3
                     ELSE SUBSTR(LINK3,1,POSITION)
              END AS LINK
       FROM RWORK.PROFILE_&NPI._WIKI3_&I. AS A,
       RWORK.PROFILE_&NPI._WIKIA;
       QUIT;

.....
OTHER CODE HERE
.....
```

Block 0

```
RUN;
%END;
%MEND MULTIPLE;
```

In short, the algorithm is as follows:

1. Study the structure of website HTML
2. Infile the search page using the search phrase (INFILE URL)
3. Use Input @ to identify string containing URL of the requried webpage

4. Use text functions to clean it up and do the INFILE again
5. Write a macro to repeat step 4 until all possible webpages are checked
6. Apply decision rules to identify the matching page
7. Repeat steps 1-3 to get the information of interest

OUTPUT

Identifying, summarizing, and analyzing data represents only part of the job for a healthcare data analyst. After the analysis is complete, it is important to make the information both accessible and understandable for the end-user. Although Excel is an excellent delivery format for technical or data-savvy users, multi-tabbed workbooks can be both intimidating and difficult to navigate for someone who does not deal regularly in numbers and spreadsheets. The healthcare field is rapidly changing, with more data becoming available every day. In addition to producing analysis that is easily digestible, it is also important that we are able to rapidly produce reports on demand, automating as much of the process as is possible.

Deliverables must be produced in a print-ready format with column headers formatted, columns sized with appropriate widths, and graphs readily available. Although much of the column and row formatting can be automated and returned in Microsoft Excel through the use of ODS XML and Microsoft tagsets, we have experienced difficulty with that method as our data contains character fields that are made of all numbers, which are varied lengths, with leading zeros. In addition, delivering a report with automated graphs not only makes the data more digestible for the end user, but saves the analyst the time of making and formatting graphs for each deliverable. Exporting directly to a PDF using SAS ODS PDF enables us to address all of these issues, and rapidly produce structured reports that require no additional "cleaning" after being exported.

The critical elements we used to create our PDF report are detailed below, with sections of SAS code included.

INLINE FORMATTING – NOT JUST FOR TEXT, BUT LOGOS TOO

Defining an ODS escape character allows for inline formatting of text written directly to the PDF, as well as some formatting of columns of individual tables. The ODS escape character essentially tells SAS that you want to stop typing text that will be printed, and you want to give the ODS instructions regarding our text. At the beginning of the output section of our script, we defined the escape character to be a carrot (^), using the command:

```
ODS ESCAPECHAR='^';
```

After defining this as the escape character, we were then free to define the style of text we wanted to use at each individual section of the report. As an example, the header was printed using the following code:

```
ODS PDF TEXT="^S={JUST=CENTER FONT_SIZE=12PT}Part B Provider Overview";
```

In the example above, ^S={} informs SAS that we are going to define a text style. In this particular case, we wanted to alter the justification and size that would normally be produced.

The usefulness of inline formatting and escape characters is not limited to simply modifying the size or type of font. Inline formatting can also be used to insert an image from a saved image file. This is of particular use to us when adding a company logo to our finished product. Using inline formatting to add our logo also means that we are able to place our logo anywhere you could insert text. Thus, we had the option of either explicitly defining where in the report the logo should appear, or placing it every header or footer of the report.

```
ODS PDF TEXT="^S={JUST=LEFT FONT_SIZE=13PT
PREIMAGE='C:\LOGO.bmp?WIDTH=1.5IN&HEIGHT=0.5IN'}";
```

FORMATTING INDIVIDUAL TABLES

Individual tables were created using PROC REPORT. Special row and column formatting, such as the shading of specific columns, was achieved through the use of COMPUTE blocks. PROC REPORT also allows us to define the amount of space allocated to each individual column through the define statement as well as create a spanned header across the entire table as shown in the sample billing summary below (figure 3). This example also uses inline formatting with an escape character, as described in the section above, to force the border around the header to be white. The code written to produce this table can be found in block 4.

This simple table is very helpful to show trends in a provider's billing pattern. An additional benefit to this table is that it has combined two different payers, both Medicare and North Carolina Medicaid, which would otherwise only be looked at individually. Our report includes multiple simple summaries of this form, to create an overall picture of the provider's billing. This specific example is of particular interest as it shows a gap in billing during 2010. If you were to look through the other information available in the profile (including enrollment and Medical Board information), it would become apparent that this provider's license was suspended near the beginning of 2009 and was not reinstated until 2011, causing the observed gap in billing.

Overall Summary for Medicare & North Carolina Medicaid Claims

Year	Medicare HIC Count	Medicare Provider Billed	Medicare Provider Paid	Medicaid Recipient ID Count	Medicaid Billed	Medicaid Paid	Total Provider Billed	Total Provider Paid
2008	61	$34,655.08	$9,112.00	205	$153,300.59	$24,853.51	$187,955.67	$33,965.51
2009	15	$4,235.00	$715.70	24	$9,139.55	$1,295.12	$13,374.55	$2,010.82
2010								
2011	382	$112,808.00	$73,323.09	531	$244,786.10	$77,332.30	$357,594.10	$150,655.39
2012	331	$62,422.00	$8,852.87				$62,422.00	$8,852.87
TOTAL	475	$214,120.08	$92,003.66	742	$407,226.24	$103,480.93	$621,346.32	$195,484.59

Figure 3. Sample Billing Summary

Block 4

```
/*OVERALL BILLING SUMMARY*/
ODS REGION
       X=0.0 IN
       Y=4.5 IN
       WIDTH=6.20 IN
       HEIGHT=2.0 IN
;
PROC REPORT
       DATA=RWORK.PROFILE_&NPI._TOT_SUM1EXP NOWD STYLE(COLUMN)=[JUST=CENTER];
       COLUMN ("^S={BORDERTOPCOLOR=WHITE BORDERLEFTCOLOR=WHITE
BORDERRIGHTCOLOR=WHITE} Overall Summary for Medicare & North Carolina Medicaid
Claims" _ALL_);
       DEFINE YEAR / DISPLAY STYLE(COLUMN)=[CELLWIDTH=7%];
              COMPUTE YEAR;
                     IF YEAR = "TOTAL"
                     THEN CALL DEFINE(_ROW_, "STYLE",
"STYLE=[BACKGROUND=CXCFE9FF]");
              ENDCOMP;
       DEFINE HIC_COUNT / DISPLAY STYLE(COLUMN)=[CELLWIDTH=9%];
       DEFINE CLAIM_COUNT / NOPRINT;
       DEFINE SUM_BILLED / DISPLAY STYLE(COLUMN)=[CELLWIDTH=12%];
       DEFINE SUM_PAID / DISPLAY STYLE(COLUMN)=[CELLWIDTH=12%];
       DEFINE HIC_COUNT_MCAID / DISPLAY STYLE(COLUMN)=[CELLWIDTH=9%];
       DEFINE CLAIM_COUNT_MCAID / NOPRINT;
       DEFINE SUM_BILLED_MCAID / DISPLAY STYLE(COLUMN)=[CELLWIDTH=12%];
       DEFINE SUM_PAID_MCAID / DISPLAY STYLE(COLUMN)=[CELLWIDTH=12%];
       DEFINE TOTAL BILLED /DISPLAY STYLE(COLUMN)=[CELLWIDTH=13%
```

Block 5

```
PROC REPORT
…
       DEFINE LINK3_EXP / DISPLAY;
           COMPUTE GOOGLE;
                  URLSTRING = http://www.google.com/;
                  CALL DEFINE (_COL_, "URL", URLSTRING);
              ENDCOMP;
…
RUN:
```

Block 6

```
/*OUTPUT AREA LINE GRAPH OF BILLED&PAID USING GREPLAY*/
ODS REGION
       X=2.75 IN
       Y=0.35 IN
       WIDTH=4.125 IN
       HEIGHT=2.125 IN;
PROC GREPLAY IGOUT=WORK.BILLING_GRAPH NOFS;
       TC SASHELP.TEMPLT;
       TEMPLATE=WHOLE;
       TPLAY 1:PAID;
QUIT;
```

EMBEDDING HYPERLINK—COMPUTE BLOCKS AREN'T JUST FOR STYLE

Although one of the main goals of creating this type of report is that it would be print-ready, in cases where it is distributed electronically, we wanted the end-user to be able to directly click on hyperlinks to return them to the source of any web-based information that was identified. Luckily for us, if you are simply printing the full website address onto the PDF, it is immediately understood as a link, and will be an active hyperlink automatically. However, it is not always convenient to print the full website address.

In cases where you would rather display text that is then clickable (IE: "Click for More Information"), one option is to put this text into a table that is displayed using PROC REPORT, and use COMPUTE blocks to embed a hyperlink (block 5).

PUTTING IT ALL TOGETHER, ON THE SAME PAGE – ODS REGIONS

To create a well-organized and structured report, we wanted to have multiple tables, graphs, and text blocks print on a single page. ODS REGIONS allows us complete control over the location of each object printed within the page of the PDF by defining individual areas of the page, measured from the upper left-hand corner of the page.

Prior to each PROC REPORT, text block, or graph, we first defined the region where will be printed (block 4). To successfully define individual regions, you must not only decide the location of the object, but also the maximum amount of space that can be allocated to that particular object. This can become difficult in summaries where there could be an unknown number of rows. To work around this problem, we frequently chose to only display the top 5 rows of interest.

All graphs were created prior to the beginning of the ODS PDF code using either PROC GCHART or PROC GPLOT. To publish the graphs within the PDF, we used PROC GREPLAY inside of a defined ODS REGION (block 6).

After the initial two pages of structured summaries in our report, we wanted to include additional information printed on subsequent pages, without any limitation to the size of the individual tables. These tables needed to be able to expand to the size necessary to contain all additional information. To accomplish this, we simply forced a page break using *STARTPAGE=NOW*. We then continued to use the PROC REPORT procedure outside of any ODS REGIONS specification, allowing the table to have as many rows as necessary, and even break onto multiple pages if needed.

LESSONS LEARNED

Up to this point, we have been focused on ensuring that the final PDF report exported from SAS is easily digestible and useful to the end-user, who will never see or interact with the SAS program. However, there is another user that we must also keep in mind; the individual SAS analyst who will be running the program in order to produce reports. The ultimate goal in the creation of this report was to develop a standard script that individual analysts could run on demand, with essentially no modifications needed other than specifying the healthcare provider of interest.

Writing a SAS script for other users with varying levels of SAS experience presents its own unique set of challenges. Detailed below are just a few of the best-practices that were identified during the development of this report.

NOT ALL USERS REGULARLY CLEAR THEIR LIBRARIES

Developing this report required the creation of a large number of intermediate SAS datasets. We had anticipated this, and thus the script was written such that the majority of these intermediate datasets are written in the work or temporary library. That way, the datasets would be cleared each time the analyst closed the SAS program. However, in production of these reports, it became quite common for an analyst to run several reports in a single day, which was rapidly filling temporary libraries with hundreds of datasets. This prompted the addition of a PROC DATASETS delete statement at the end of the standard report, to automatically clear all intermediate tables from the users SAS library. We also added a *DM 'ODSRESULTS; CLEAR;';* statement to clear all information that was written to the user's output window, as well as a PROC GREPLAY delete statement to delete the graphs.

USER-DEFINED MACRO VARIABLES RUN AMUCK

There are several occasions throughout our program where information is identified from the results of a query and is assigned to a macro variable. It is normal for there to be instances where the initial query would return no results, and thus nothing would be assigned to the macro variable. In initial testing, this process seemed to work fine. However, when the program was placed in limited production, where it was being run multiple times in a row, we immediately ran into trouble.

In cases where the first run on 'Provider A' assigned a value to the macro variable, and the second run on 'Provider B' returned no results and thus could not assign a value to the macro, then there was nothing to overwrite the prior variable assignment. This resulted in macro variables from the first run of the program appearing in the second run of the program, creating somewhat of a Franken-profile, with information merged from two different providers. To rectify this problem, we added a *%SYMDEL* macro to the end of the profile script, to delete all macros defined during the program.

FORWARD PROGRESS—CONVERTING FROM SAS 9.2 TO SAS 9.3

Just when we thought that we had worked all of the bugs out of our program, our data management team upgraded us from SAS 9.2 to SAS 9.3. SAS 9.3 understands the specified location of the ODS REGIONS differently than SAS 9.2. This prompted an immediate revision of the output section of the script, where each of the locations of the individual regions had to be re-defined.

CONCLUSION

Techiques described above were applied to all available inhouse datasources as well as multiple web-based resources. Please note that all names, addresses, and identifiers included in this sample profile are fictitious.

CONTACT INFORMATION

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

Name: Mariya Karimova
Enterprise: AdvanceMed, an NCI Company
Address: 520 Royal Pkwy, Suite 100
City, State ZIP: Nashville, TN 37214
E-mail: karimovm@admedcorp.com
Web: http://www.nciinc.com/
http://www.linkedin.com/pub/mariya-karimova/1/661/b36

Name: Christine John
Enterprise: AdvanceMed, an NCI Company
Address: 520 Royal Pkwy, Suite 100
City, State ZIP: Nashville, TN 37214
E-mail: johnc@admedcorp.com
Web: http://www.nciinc.com/