You Can’t Get There From Here If You Don’t Know Where Here Is. The Importance of Profiling in Mapping Data

Patricia Hettinger, Oakbrook Terrace, IL

ABSTRACT

How many times have you been fooled by the name of a data element but found completely different values than you would expect? SAS Enterprise Guide has a characterization task that could give you better understanding but it has several drawbacks. One is that it will run frequencies on all character data regardless of length or number of distinct values. This can result in some variables being dropped from the output due to too many distinct values. It also results in frequencies not being run for numeric values at all. Another is that minimum, maximum, number of missing values and number of non-missing values will be calculated only for numeric variables when this information would be useful for any variable. A third issue is the likelihood of your system hanging when attempting to analyze large data sets with many variables. This paper details how you can overcome these obstacles as well as incorporating your profile results into a useful mapping document.

INTRODUCTION

It often happens that you need to look intensively at the data you used to take for granted. Perhaps there are changed procedures or new vendors. A critical source changes completely or worse yet, is discontinued altogether. There may be new rules and regulations affecting your industry, Dodd-Frank in the banking area for example. You might even find yourself taking back a process that was outsourced a few years ago. In any case, the documentation is scattered and no one’s really sure what’s there any more.

A well-thought out profiling methodology can help in several ways. First, of course, it shows what is there now and how far you may be from where you thought you were. Second, it gives several talking points with your users. What do these extra values mean? Why are some of the values that are supposed to be there missing? Why are there variables with 100% missing values? What might have changed in the 'real' world?

GENERAL DIFFERENCES BETWEEN DATABASES AND SAS DATA SETS

If you’ve ever worked on a database project, you may be familiar with the source to target mapping document. This should include definitions, code values and most importantly, the path between the source and the final variable. The path includes the source and target format, plus any changes that may have occurred in the transformation. Sometimes deciding the target name is the most difficult part. Should it be homeowner code, homeowner indicator, housing type code or…? The technical name may be even more difficult. The database naming convention is a limitation when you have only 32 positions or even fewer. There may be other rules too. Spell out ‘code’ or abbreviate as cd or cde? There are people whose main job is to name objects. Sometimes deciding on a name takes longer than creating or audit the data element. It’s amazing how many people have strong ideas about what something should be named.

The author has seen few source-to-target mapping documents for SAS data sets, even though many data sets are used by hundreds of people and considered the system of record. Perhaps this is because anyone with the proper system access could create them -- and did. There’s no database involved and thus no database administrator. We will discuss how you can adapt a standard mapping document to SAS data sets, including a baseline profile to measure other reiterations against if need be (and it probably will)

BASICS OF A MAPPING DOCUMENT

Even if you have access to a formal metadata repository, putting your mapping information into an Excel spreadsheet first is a good idea. Most business people understand them and can follow the information more easily. As far as naming conventions go, we will abbreviate ‘CODE’ as ‘CD’, ‘NUMBER’ as ‘NO’ and ‘DATE’ as ‘DT’.

The example in figure 1 shows the variable #, business name, target variable, definition, target variable format, source data name and source data element name for some variables relating to coupon redemption tracking.

Variable #	Business Name	Target Variable	Definition	Target Variable Format	Source Data Name	Source Data Element Name
1	**Account Number**	ACCT_NO	Number that specifically identifies an account	Numeric 12	Daily Pipeline Report	ACCOUNT-NUM
2	**Coupon Code**	COUPON_CD	The promotion associated with the coupon	CHAR8	Daily Pipeline Report	PROMO-CODE
3	**Coupon Redemption Method Code**	COUPON_REDEEM_ METH_CD	Code that indicates how a coupon was redeemed	Numeric 4	Daily Pipeline Report	PROMO-REDEEM-CODE
4	**Coupon Redemption Date**	COUPON_REDEEM_ DT	The date the coupon was redeemed	SAS date	Daily Pipeline Report	PROMO-REDEEM-DATE

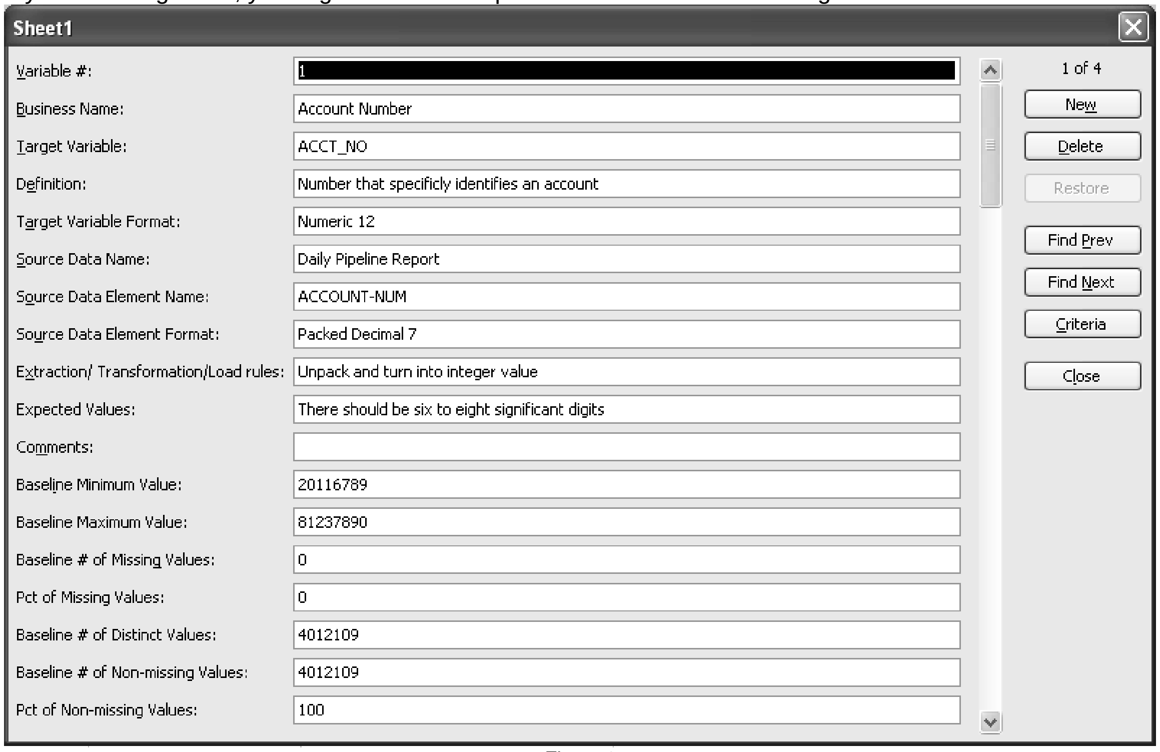
Figure 1

Figure 2 gives the source data element format, the extract/transformation/load rules, expected values and comments for some elements in the MarketingPromotion data set. If we have a large number of codes that do not change that often, we might want the coupon code values in its own section. Note the comments arose from discrepancies between what was expected and what was actually present in the baseline profile:

Target Variable	Source Data Element Format	Extraction/ Transformation/Load rules	Expected Values	Comments
ACCT_NO	Packed Decimal 7	Unpack and turn into integer value	There should be six to eight significant digits	
COUPON_CD	alphanumeric	direct move	Varies, see latest Marketing campaign list	
COUPON_REDEEM_METH_CD	Numeric integer	direct move	1-mailed in 2-redeemed online 3-inbound telemarketing	Mostly 2? Are we missing some in the baseline?
COUPON_REDEEM_DT	alphanumeric	Source format should be MM/DD/YYYY but use anydtdte10. format just in case	Valid calendar date	Why are we getting 0's in the source when there are redemption codes?

Figure 2

If you are using Excel, you might want to set up a data form like the one in Figure 3:



Variable #:	1
Business Name:	Account Number
Target Variable:	ACCT_NO
Definition:	Number that specificly identifies an account
Target Variable Format:	Numeric 12
Source Data Name:	Daily Pipeline Report
Source Data Element Name:	ACCOUNT-NUM
Source Data Element Format:	Packed Decimal 7
Extraction/ Transformation/Load rules:	Unpack and turn into integer value
Expected Values:	There should be six to eight significant digits
Comments:	
Baseline Minimum Value:	20116789
Baseline Maximum Value:	81237890
Baseline # of Missing Values:	0
Pct of Missing Values:	0
Baseline # of Distinct Values:	4012109
Baseline # of Non-missing Values:	4012109
Pct of Non-missing Values:	100

Figure 3

Adding the baseline profile shows us the minimum value, the maximum value, the number of distinct values, the number of missing values, the percent missing, the number of non-missing values and the percent non-missing (figure 4):

Target Variable	Baseline Minimum Value	Baseline Maximum Value	Baseline # of Missing Values	Pct of Missing Values	Baseline # of Distinct Values	Baseline # of Non-missing Values	Pct of Non-missing Values
ACCT_NO	20116789	81237890	0	0.00	4012109	4012109	100.00
COUPON_CD	A150	XX70	1512093	37.69	70	2500016	62.31
COUPON_REDEEM_METH_CD	1	3	1512093	37.69	3	2500016	62.31
COUPON_REDEEM_DT	1/1/2007	1/28/2011	1512093	37.69	1102	2500016	62.31

Figure 4

Another tab on the spreadsheet has the frequencies for those variables having 75 distinct values or less (Figure 5):

Variable	Value	Frequency Count	Frequency Percent
COUPON_REDEEM_METH_CD		1512093	37.69
COUPON_REDEEM_METH_CD	1	833339	20.77
COUPON_REDEEM_METH_CD	2	1250008	31.16
COUPON_REDEEM_METH_CD	3	416669	10.39
COUPON_CD		1512093	37.69
COUPON _CD	A150	28000	0.7
COUPON _CD	AA418	30000	0.75
COUPON _CD	AA583	28500	0.71
COUPON _CD	AA612	29500	0.74
COUPON _CD	and so on to largest value		
COUPON _CD	XX70	280	0.7

Figure 5

This profile gives us a good idea of what is in the target data set and maybe what should always be there. So how might we go about getting one? Do we have to code every time we do a load? No. We might not have any data profiling tools like Trillium Discovery or SAS Dataflux but we do have base SAS and SAS Enterprise Guide. This paper will detail how to set up a profiling application using just these two products. In fact, you don't even need Enterprise Guide but it is convenient for setting up prompts and stored processes.

THE CHARACTERIZATION TASK IN SAS ENTERPRISE GUIDE

The characterization task in Enterprise Guide is a fair way to understand a data set with frequencies on character data and statistics on numeric data. However it comes up a bit short with both types of data. There are no minimum and maximum values for character data, which would be particularly helpful for long variables. Frequencies are only run for character data, while we could benefit from frequencies on numeric variables too, especially numeric codes and dates.

A major performance issue is that the characterization task will run frequencies on all character data regardless of length or number of distinct values. This can result in some variables being dropped from the output due to too many distinct values. As stated before, it also results in frequencies not being run for numeric values at all. If you are attempting to use the task on large data sets with many variables, you may find yourself hanging, especially in the frequency step.

Fortunately, one can take the code generated in Enterprise Guide as a starting point for new and more customized routines. The author decided to address the characterization task deficiencies as well as add the number of distinct values to control the number of frequencies run. The performance improved quite a bit as well.

To illustrate, let us look at how the built-in characterization process works for our promotion and coupon redemption tracking data set. Figure 6 below shows the three numeric fields:

Dataset	Variable	Format	N	NMiss
INDATA.PROMOTRACK	ACCT_NO		4012109	0
INDATA.PROMOTRACK	COUPON_REDEEM_DT	MMDDYY10.	2500016	1512093
INDATA.PROMOTRACK	COUPON_REDEEM_METH_CD		2500016	1512093

Variable	Total	Min	Mean	Median	Max	StdMean
ACCT_NO	3.52E+16	20116789	4062198	33792109	81237890	3000.235
COUPON_REDEEM_DT	1.40E+09	17167	18346.75	18635.2	18655	0.72924498
COUPON_REDEEM_METH_CD	4583362	1	1.83333307	2	3	3.8777E-07

Figure 6

Some of these statistics don't make all that much sense. Do we really need a summation or a standard error of the mean for ACCT_NO or COUPON_REDEEM_DT? ACCT_NO might be a key but you certainly can't tell it from here. The minimum value is 1 for COUPON_REDEEM_METH_CD, the maximum value is 3 and the median is 2. Does that mean we only have three values? If so, what is their distribution? Where is this data set we're looking at anyway?

We did get one frequency for COUPON_CD. There were actually thirty distinct values reported (the default). Figure 7 shows the top five plus a general category, 'All other values' because that is the number of top values we wanted to see.

Dataset	Variable	Label	Format	Value	Count	Percent
INDATA.PROMOTRACK	COUPON_CD	COUPON_CD		***Missing***	1512093	37.69
INDATA.PROMOTRACK	COUPON_CD	COUPON_CD		AA418	30000	0.75
INDATA.PROMOTRACK	COUPON_CD	COUPON_CD		AA612	29500	0.74
INDATA.PROMOTRACK	COUPON_CD	COUPON_CD		AA583	28500	0.71
INDATA.PROMOTRACK	COUPON_CD	COUPON_CD		A150	28000	0.70
		and so on….				
INDATA.PROMOTRACK	COUPON_CD	COUPON_CD		***All other values***	940000	23.43

Figure 7

We could definitely get better information here. Seeing the range of values for any variable, whether character or numeric, would be useful. How about the acct_no? How many distinct vales are there? Should this be used as a key and indexed? We should find the number of distinct values and decide a cut-off point for frequencies, regardless of variable type. Even better, can we write the code once and reuse any number of times with variable substitution? We met all of these objects with output seen in figures 8 and 9:

New Summary

Dataset	Libpath	Variable	Format
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	ACCT_NO	NUM8
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_REDEEM_DT	DATE
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_REDEEM_METH_CD	NUM4
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_CD	CHAR8

Variable	Minimum Value	Maximum Value	# of Missing Values	# of Distinct Values	# of Non-missing Values
ACCT_NO	20116789	81237890	0	4012109	4012109
COUPON_REDEEM_DT	1/1/2007	1/28/2011	1512093	1102	2500016
COUPON_REDEEM_METH_CD	1	3	1512093	3	2500016
COUPON_CD	A150	XX70	1512093	70	2500016

Figure 8

New Frequency:

Dataset	Libpath	Variable	Value	Frequency Count	Frequency Percent
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_REDEEM_METH_CD		1512093	37.69
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_REDEEM_METH_CD	1	833339	20.77
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_REDEEM_METH_CD	2	1250008	31.16
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_REDEEM_METH_CD	3	416669	10.39
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_CD		1512093	37.69
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_CD	A150	28000	0.70
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_CD	AA418	30000	0.75
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_CD	AA583	28500	0.71
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_CD	AA612	29500	0.74
	and so on…				
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	COUPON_CD	XX70	280	0.70

Figure 9

ARCHITECTURE

The programs in this paper were originally set up in an Enterprise Guide 4.1 project stored on the local server. They were moved to SAS Enterprise Guide 4.2 and 4.3 with no code changes. Only new prompts had to be set up. They have profiled SAS data sets and relational database tables on UNIX and local servers. With a few simple modifications, they have run in batch jobs as well, even on Z/OS mainframes.

Our first point of departure from the standard characterization task is querying the dictionary column table instead of using PROC CONTENTS to determine the variable types. This gives us more information and flexibility with variable names containing special characters or embedded spaces. We will query the dictionary column tables for the number of variables to be profiled, their names and attributes and then sort by name.

Our second point of departure is to use PROC SQL to find the minimum, maximum, number of missing values and number of distinct values for each variable. Unlike procs univariate or means, this works for any type of variable. Once we know the number of distinct values, we can run a frequency on that variable if that number is greater than one (no point in running a frequency if the distribution is 100%) and at or under the maximum we previously specified. We will put then the numeric and character results together in one table by storing all the values in character columns. We will create a similar table for the results from PROC FREQ so that those values can be stored together as well.

Perhaps the most important addition is that of limiting or safety macro variables. One is the maximum number of variables we will profile in one step. If you have one thousand variables in a data set, you might want to profile just a few hundred at time to avoid a session hang-up. We will call this the &profmax variable. If we do want to profile in more than one pass, we will want to start the profiling at a different location for each step. We will call this variable &profstart. The final limiting variable will be on the number of distinct values for which we want to see the distribution. This variable is called &freqmax.

Therefore if we wanted to profile the first 500 variables and do a frequency on those having 2 to 75 different values, we would set &profmax to 500, &profstart to 1 and &freqmax to 75. If we wanted to do the rest or another 500, depending on how many variables are left, we would set &profmax to 500 again and 'start' to 501.

We will also use some location macro variables. They are the library name (&libname), the member name (&memname) and the external location where our results will be stored (&path).

In this process, we are storing our results in an XML document with two tabs, one for the summary and one for the frequency. The frequency tab shows the maximum number of values for which we ran them. In our example, that would be '2 to 75'.

WORK TABLES SETUP

We will store our results in four work tables, two for the summary and two for the frequencies. The TEMPRANGE table will be overwritten for each variable's summary information. The ALL_STATS table will hold the results for all of the variables. The structure for both is shown in Figure 10:

Column Name	dataset	libpath	variable	label	format	db_format
Population Rule	&libname..&memname	physical path of library	variable being analyzed	label, if present	format in source if present. Otherwise 20. if numeric, $100. if character	Native format. Either DATE, NUMw or CHARw

Column Name	min_value	max_value	miss_value	distinct_values	nmiss_value
Population Rule	Minimum value of variable	Maximum value of variable	number of missing values	number of distinct values	number of non-missing variables

Figure 10

We only need to build ALL_STATS once. We will have TEMPRANGE built as part of a loop in the getvar macro.

```
DATA   ALL_STATS (LABEL="Ranges for All Variables");
       ATTRIB dataset FORMAT=$41. Variable FORMAT=$32. Label FORMAT=$256. Format
       FORMAT=$31. Min_value FORMAT=$100. Max_value FORMAT=$100.
       Miss_value FORMAT=comma12.
       distinct_values FORMAT=comma12. Nmiss_value FORMAT=comma12. ;
       LABEL Min_value = 'Minimum Value' Max_value = 'Maximum Value'
       Miss_value = '# of Missing Values' Distinct_values = '# of Distinct
       Values' Nmiss_value = '# of Non-missing Values' ;
STOP;
RUN;
```

TEMPFREQ and STOREFREQS will have the structure in Figure 11. TEMPFREQ will be overwritten for each variable and STOREFREQS will contain the results for all the variables:

Column Name	dataset	libpath	Variable	Value	count	percent
Population Rule	&libname..&memname	physical path of library	variable being analyzed	Distinct Value	Count for this value	Percent of total values

Figure 11

This code will create STOREFREQS.   The TEMPFREQ build will be part of a loop in the dofreq macro.

```
DATA STOREFREQS(LABEL="Frequency Counts for Selected Variables");
       ATTRIB dataset FORMAT=$41. Variable FORMAT=$31. Value FORMAT=$100. Count
       FORMAT=8. Percent FORMAT=6.2;
       LABEL Count='Frequency Count' Percent='Percent of Total Frequency';
       STOP;
RUN;
```

The 'STOP' statements let us create empty data sets instead of those with just one observation each. The 'ATTRIB' sets up the length and format for the variables. Notice our minimum and maximum values in the summary table as well as the frequency values are formatted as character. This will let us store information for both numeric and character variables.

SAS'S DICTIONARY LIBRARY

Using SAS's dictionary library has a few advantages over PROC CONTENTS DATA=_all_; One is the PROC CONTENTS procedure will stop dead if it hits a member name in the library that has embedded spaces or special characters, quite likely if libnaming external data like SAP or Excel spreadsheets. The dictionary has no such restrictions. Here we are retrieving variable names from the COLUMN dictionary by using PROC SQL, getting the number of variable names returned by examining the automatic SAS variable SQLOBS. We were prompted for the library name (&libname) and member name (&memname) when we ran the program;

```
PROC SQL;
CREATE TABLE outlist as
SELECT * FROM
(SELECT * from dictionary.columns
WHERE memtype = 'DATA' and UPCASE(LIBNAME) = UPCASE("&libname")
and UPCASE(memname) = UPCASE("&memname"))
ORDER BY name; QUIT;
```

If you have RDBMS databases allocated to your session using the libname option, querying the dictionary tables will result in a dynamic call to the RDBMS for metadata. This can take some time.

If you aren't running this for a RDBMS table, clearing the libname will make the dictionary run a lot faster with a statement like this: `Libname clear` *`rdbms_name`* `;`

POPULATING THE SUMMARY

Once we have our list of variable names, we will call the macro %GETVAR, using the automatic variable SQLOBS as the actual number of variables, profstart as the variable position at which to start the profiling and profmax as the maximum number of variables to profile this time around:

```
%getvar(&sqlobs,&profstart,&profmax);
```

The %getvar macro does several things. It reads the work data set named ‘outlist’ to get each variable to process. It will start at the point specified in the &profstart variable and end after processing &profmax number of variables. It then creates SQL statements and executes them in a PROC SQL. Finally it formats the results and appends them to the summary ALL_STATS table. Here is the macro in its entirety:

```
%macro getvar(numobs,startlim,endlim);
%if %eval(&endlim+&startlim-1) lt %eval(&numobs)
%then %LET endctr = %eval(&startlim+&endlim);
%else %LET endctr = &numobs;
%do i = &startlim %to &endctr;

DATA _null_;
ATTRIB min_var FORMAT=$200. max_var FORMAT=$200. miss_var FORMAT=$200.
distinct_var FORMAT=$200. nomiss_var FORMAT=$200. search_name
FORMAT=$35. dataset FORMAT=$42. vlength FORMAT=$10. orig_format
FORMAT=$10.
path_name FORMAT=$100.;
;
      pointer=&i.;
      SET outlist point=pointer;
      vlength=length;
      search_name="'"||TRIM(name)||"'N";
      path_name = pathname(libname);
      dataset = TRIM(libname)||".'"||TRIM(memname)||"'N";
      if FORMAT=:'DATE' or FORMAT=:'JUL' or FORMAT=:'MM' or
      FORMAT=:'DD' or FORMAT=:'DAY' or FORMAT=:'MON' or FORMAT=:'YEAR'
      or FORMAT=:'WORDD' or FORMAT=:'EURD' or FORMAT=:'WEEK'
      then do;
            FORMAT='MMDDYY10.';
            orig_FORMAT='DATE';
      end;
      else if type = 'num' then do;
            orig_FORMAT=TRIM(UPCASE(type))||compress(vlength);
            FORMAT='20.';
      end;
      else if type = 'char' then do;
            FORMAT='$100.';
            orig_FORMAT=TRIM(UPCASE(type))||compress(vlength);
      end;
MIN_VAR = 'create table temprange as select
MIN('||TRIM(search_name)||') as min_value1';
max_var = ',MAX('||TRIM(search_name)||') as max_value1';
miss_var = ',NMISS('||TRIM(search_name)||') as miss_value';
distinct_var = ',COUNT(DISTINCT('||TRIM(search_name)||')) as
distinct_values';
nomiss_var = ',COUNT('||TRIM(search_name)||') as nmiss_value from
'||TRIM(dataset)||';';
```

```
      CALL SYMPUT('SQL1',TRIM(MIN_VAR));
      CALL SYMPUT('SQL2',TRIM(MAX_VAR));
      CALL SYMPUT('SQL3',TRIM(MISS_VAR));
      CALL SYMPUT('SQL4',TRIM(DISTINCT_VAR));
      CALL SYMPUT('SQL5',TRIM(NOMISS_VAR));

      CALL SYMPUT('var',TRIM(name));
      CALL SYMPUT('dataset',TRIM(dataset));
      CALL SYMPUT('var_n', quote(name)||"n");
      CALL SYMPUT('type',TRIM(type));
      CALL SYMPUT('label',label);
      CALL SYMPUT('format',format);
      CALL SYMPUT('orig_format',orig_format);
      CALL SYMPUT('path_name',path_name);
STOP;
RUN;

/*Note:  all above built the PROC SQL statement*/
PROC SQL;
&sql1
&sql2
&sql3
&sql4
&sql5
QUIT;

/*reformat the results*/
DATA temprange;
ATTRIB dataset FORMAT=$41. libpath FORMAT=$100. variable FORMAT=$32.
label FORMAT=$256. format FORMAT=$31.
db_format FORMAT=$31.
min_value FORMAT=$100. max_value FORMAT=$100.
;
SET temprange;
      dataset=TRIM("&libname..&memname");
      variable = TRIM("&var");
      format=TRIM("&format");
      db_format=TRIM("&orig_format");
      type = TRIM("&type");
      label=TRIM("&label");
      libpath=TRIM("&path_name");
      if type = 'num' then do;
            min_value=compress(put(min_value1,&format));
            max_value=compress(put(max_value1,&format));
      end;
      else do;
            min_value=min_value1;
            max_value=max_value1;
      end;
      DROP min_value1 max_value1 type;
RUN;

PROC APPEND base=work.ALL_STATS DATA=work.temprange force;
RUN;
%end;
%mend getvar;
```

Concepts

```
%if %eval(&endlim+&startlim-1) lt %eval(&numobs)
%then %LET endctr = %eval(&startlim+&endlim);
%else %LET endctr = &numobs;
%do i = &startlim %to &endctr;
```

If the maximum number of variables we want to process plus the starting position minus one is less than the total number of variables, we will process only the maximum number requested. If it is greater, we will just process all of the variables. This loop will be executed until we run out of variables to analyze. One thing that's nice about using the %DO … %TO kind of loop is that the counter increments automatically

In the DATA step:

```
POINTER=&i.;
SET outlist point=pointer;
```

Using point processing lets us read a data set using random access instead of sequential reads. This can save processing time with even a moderately sized data set. This works by setting a DATA step variable to the macro counter variable and using the POINT option on the set statement. Perhaps the most important statement in this DATA step is `STOP;` If omitted, the step will be trapped in an endless loop.

```
search_name="'"||TRIM(name)||"'N";
```

We're going to prefix the variable name with a single quote and add a suffix of single quote N. This will allow us to deal with variable names containing special characters or spaces, very common with Excel or SAP.

`FORMAT =: "YYQ" or FORMAT =:"MMYY"` etc…

The =: operator means 'begin with'. Here we are looking for all of the known SAS date formats. Considering the difficulty of reading SAS's internal date values, we want the output in a legible date format if the variable is indeed supposed to be a date. If the variable is any other numeric kind, we'll put it in a numeric format with a length of twenty. A character variable's length will be one hundred.

```
min_var = 'create table temprange as select MIN('||TRIM(search_name)||')
as min_value1';
```

Our min_var assignment will have a create table task as well as finding the minimum value of the variable. The max_var assignment will find the maximum value, miss_var the number of missing values, distinct_var the number of non-missing distinct values and nomiss_var the number of non-missing values. The nomiss_var variable assignment will also have the name of the data set or table we are querying.

```
CALL SYMPUT('SQL1',TRIM(MIN_VAR));
```

We use CALL SYMPUT to put the first statement into a macro variable for later use and continue until we have all five.

SQL constructed

When we ran this for our PROMOTRACK data set, the first SQL statement evaluated to:

```
CREATE TABLE TEMPRANGE AS SELECT MIN('ACCT_NO'N) as min_value1
,MAX('ACCT_NO'N) as max_value1 ,NMISS('ACCT_NO'N) as miss_value
,COUNT(DISTINCT('ACCT_NO'N)) as distinct_values ,COUNT('ACCT_NO'N) as
nmiss_value FROM WORK.'CUSTOMER'N;
```

```
DATA TEMPRANGE; ATTRIB dataset….SET TEMPRANGE;
```

We recreate the temporary data set to have the proper formats and to populate the data set, actual file path, and make sure the min_value and max_value variables are in character format.

```
if type = 'num' then do;
min_value="'"||compress(put(min_value1,&format));
max_value="'"||compress(put(max_value1,&format));
```

Put the minimum and maximum numeric results in either a date format or a numeric 20. Resist the temptation to put all fields seeming to be dates into date format. They may be dates but unless there is a date format formally attached with them, they are probably not SAS dates and will not be understandable if interpreted as such.

```
PROC APPEND BASE=WORK.ALL_STATS DATA=WORK.TEMPRANGE FORCE;
```

PROC APPEND will drop any variables not defined in the base (ALL_STATS). The FORCE option makes SAS append the data even if the lengths are different. Truncation will occur if needed. Once this is done, we are ready to summarize the next variable and so on until we're done.

After we're done collecting the statistics, we'll want to calculate percent missing and percent non-missing.

```
data all_stats;
      attrib percent_miss format=6.2 percent_nmiss format=6.2;
      set all_stats;
      percent_miss=100*(miss_value/(miss_value+nmiss_value));
      percent_nmiss=100*(nmiss_value/(miss_value+nmiss_value));
```

THE FREQUENCIES

Now that our summary table has been created, the next step is to run frequencies on all variables having from 2 to the number of specified distinct values by means of this code:

```
DATA set_freqs;
SET ALL_STATS;
WHERE distinct_values between 2 and &freqmax;
CALL SYMPUT('numobs',PUT(_n_, 12.));
RUN;
```

A value for &freqmax may be obtained at run time with either a SAS Enterprise Guide prompt or actually setting it in the code. We won't run a frequency on a variable having just one distinct non-missing value because we already know the distribution is 100%.

Much as we set up the summary table, we'll set up a table to hold the frequencies. Here it's limited to the count and the percent although other values obtained from the PROC FREQ procedure could be added too. The %dofreq macro is considerably simpler than %getvar because the work table structure is simpler:

```
%MACRO DOFREQ;
ATTRIB freqname FORMAT=$40.;
%DO i=1 %to &numobs.;
      DATA _NULL_;
      POINTER=&i.;
      SET SET_FREQS point=pointer;
      if format = '' OR FORMAT='$' then FORMAT='$32.';
      else FORMAT=TRIM(format);
      freqname="&libname..'&memname.'n";
      CALL SYMPUT('var', variable);
      CALL SYMPUT('var_n', QUOTE(variable) || "n");
      CALL SYMPUT('format', format);
      CALL SYMPUT('datast', dataset);
      STOP;
      RUN;
      PROC FREQ DATA=&freqname. NOPRINT;
      TABLES &var_n./MISSING OUT=TEMPFREQ;
      RUN;
      DATA TEMPFREQ ;
      SET TEMPFREQ ;
```

```
        ATTRIB value FORMAT=$32.;
        dataset = "&datast";
        Variable = "&var";
        value=TRIM(PUT(&var.,&format.));
        RUN;
        PROC APPEND BASE=WORK.STOREFREQS DATA=WORK.TEMPFREQ FORCE;
        RUN;
        %END;
%MEND DOFREQ;
```

Concepts

```
SET_FREQS point=pointer;
```

Use point processing here too.

```
PROC FREQ DATA=&datast. NOPRINT;
```

Do not print out the results of PROC FREQ.

```
TABLES &var_n./MISSING OUT=TEMPFREQ;
```

Our output will be a work data set named TEMPFREQ. The MISSING option will put in the number of missing values. Here we are running just one at a time. Note that if you wanted to run several variables in one PROC FREQ and store the results, you would need separate out data sets for each variable.

```
ATTRIB value FORMAT=$32.;
```

You may want to make this larger but thirty-two positions should be ample to hold most codes and dates.

```
PROC APPEND BASE=WORK.STOREFREQS DATA=WORK.TEMPFREQ FORCE;
```

PROC APPEND inserts the results to the STOREFREQS temporary data set and we loop through again.

EXCEL XML OUTPUT

Now that we have our tables built, it's time to move them to a more permanent location. Here we'll export them in Excel XML format in order to build separate workbooks for the summary and the frequency.. The frequency tab in the workbook will be labeled to show the values between 2 and &freqmax:

```
%let freqend = %unquote(%str(%'Freqs Value # 2 to &freqmax%'));
%let file = profile results_&libname..&memname..xml;
ods listing close;
ods tagsets.ExcelXP path = "&path"
 file="&file" style=analysis;
ods tagsets.ExcelXP options(sheet_name='Summary'
      Autofilter = 'yes'
      Frozen_Headers = '1'
      Frozen_rowheaders='2,3,4'
      Absolute_Column_Width="20,20 ,20, 20, 7.5, 20, 20, 7.5, 7.5, 7.5,
      7.5, 7.5"
Autofit_Height = 'YES');

proc report data=all_stats;
      column  dataSet libpath Variable label db_format min_value
max_value distinct_values miss_value percent_miss  nmiss_value
percent_nmiss
      ;
      define label/display width=100;
      define percent_miss/display "Pct Missing";
      define percent_nmiss/display "Pct Not Missing";
      define libpath/display "Physical Location";
      define min_value   / style (column)={tagattr="format:Text"};
      define max_value / style (column)={tagattr="format:Text"};
run;

ods tagsets.ExcelXP options(sheet_name=&freqend
      Autofilter = 'yes'
      Frozen_Headers = '1'
      Frozen_rowheaders='2,3,4'
      Absolute_Column_Width="20,20 ,20, 20,7.5,7.5 "
      skip_space='1,1,0,0,1' sheet_interval='none'
      suppress_bylines='no'
      Autofit_Height = 'YES');

proc print data=storefreqs label noobs;
      var dataSet libpath variable;
      var value / style (data)={tagattr="format:Text"};
      var count percent;
      by variable;
run;

ods tagsets.ExcelXP close;
```

Concepts

```
%LET file = profile results_&libname..&memname..xls;
```

For this example, the path and file name will /mktg/group1/u108/profile results_indata-promotrack.xls.

```
%LET freqend = %unquote(%str(%'Freqs Value # 2 to &freqmax%'));
```

This was the only form that named the frequency tab correctly.   Other forms such as using the %str function by itself or %superq either resulted in message `ERROR 22-322: Expecting a quoted string` or having the tab literally named Freqs Value # 2 to &freqmax (%quote, %unquote by itself, %bquote).

```
ODS LISTING CLOSE;
```

Close ODS.

```
ODS TAGSETS.EXCELXP path = "&path"
```

Use the ExcelXP tagset without overrides.

```
file="&file" style=analysis ;
```

The path was a variable entered in at runtime.

```
ODS TAGSETS.EXCELXP options(sheet_name='Summary'….Autofit_hight='YES');
```

Built-in options in this tagset let us name separate sheet names, and set some properties of the XML spreadsheet.

```
PROC REPORT DATA=ALL_;….
```

Use PROC REPORT to print out the summary table, ALL_STATS.

```
(column)={tagattr="format:Text"};
```

For the minimum and maximum values,
Format as if text. This keeps really large values from being displayed in scientific notation.

```
ODS TAGSETS.EXCELXP options(sheet_name=&freqend
```

Second tab in workbook.

```
PROC PRINT DATA=storefreqs noobs label;~~.
```

Print out frequencies

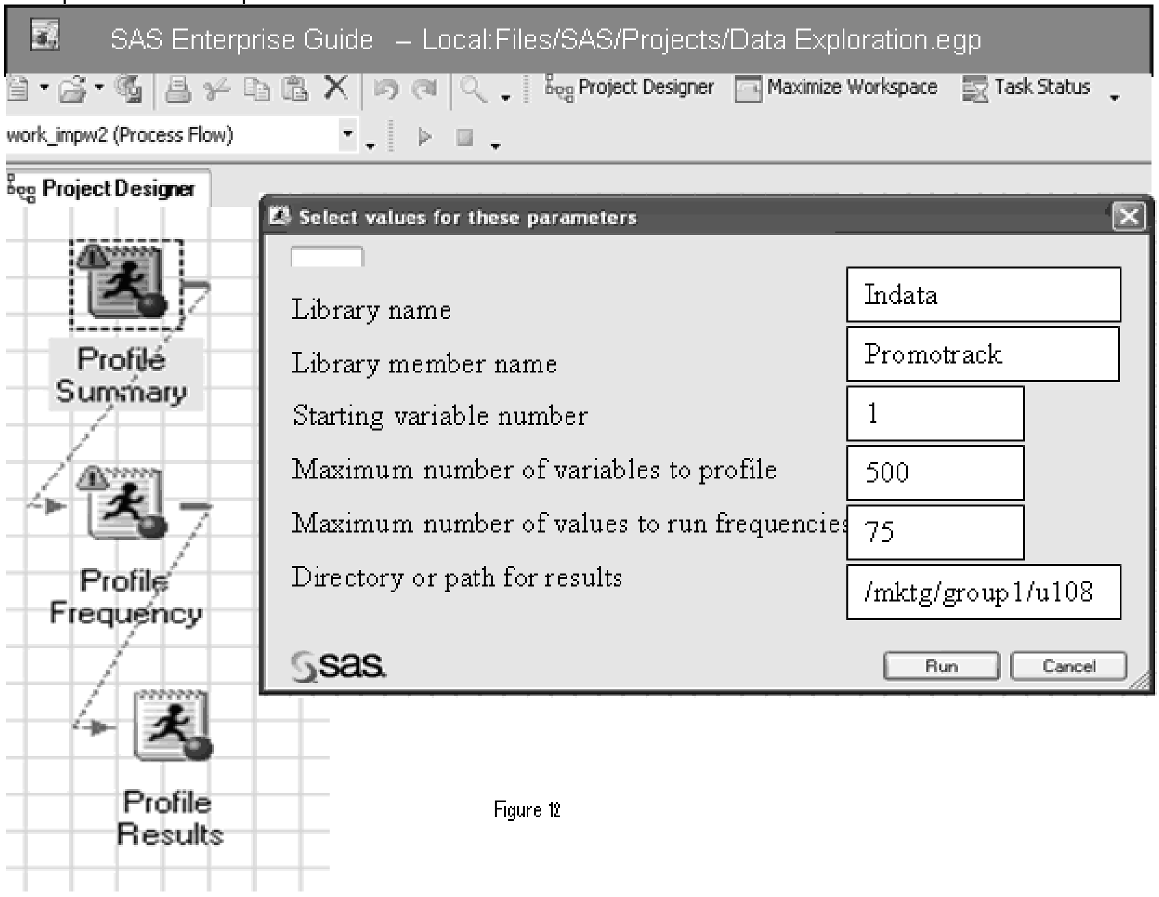
```
ODS TAGSETS.EXCELXP close;
```

Close tagset and finish writing to the XML file.

SETTING UP THE PROCESS

In Enterprise Guide, these were set up three programs linked together . Prompts were set up to obtain the different macro variables. Figure 12 shows the default values for &profstart, &profmax and &freqmax. If you are using Enterprise Guide and are allowed to do so, creating a stored process is very helpful. Anyone who has access to the process can use it without any prompt setup.

Enterprise Guide Setup:


SAS Enterprise Guide – Local:Files/SAS/Projects/Data Exploration.egp
Project Designer
Maximize Workspace
Task Status
work_impw2 (Process Flow)
Project Designer
Profile Summary
Profile Frequency
Profile Results
Select values for these parameters
Run
Cancel

Library name	Indata
Library member name	Promotrack
Starting variable number	1
Maximum number of variables to profile	500
Maximum number of values to run frequencies	75
Directory or path for results	/mktg/group1/u108

Figure 12

Another way is to set the macro variables in a driver program. This may be used in a batch job even on z/os:

```
%LET profstart=1;
Let profmax=500;
%LET libname = indata;
%LET mename = promotrack;
%LET freqmax=75;
%LET path=/mktg/group1/u108;
%include '/mktg/group1/u108/Profile Summary.sas';
%include '/mktg/group1/u108/Profile Frequency.sas';
%include '/mktg/group1/u108/Profile Results.sas';
```

CONCLUSION

A good mapping document with profile included may well be the most valuable thing that comes out of a SAS project. As with any other document, it's important to keep updated regularly. Even if there are no changes, putting a review date in the log will demonstrate that this is a living, breathing record.

The profile should encourage dialogue with your user community. It most probably will result in data quality improvement as well. The coupon redemption date is a good case in point. Once you know that you are getting a substantial number of unexpected values, you might want to petition change right at the source. If that proves difficult, you may want to transform the incoming data yourself, noting that transformation in the mapping document of course. Or in the case of missing data, It may be more practical to have a default, again reflected in the mapping document.

ACKNOWLEDGEMENTS:

Thanks to Joe and Paul Butkovich for your encouragement and support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. The author is often on the road but can be contacted at

Patricia Hettinger
Email: patricia_hettinger@att.net
Phone: 331-462-2142