Paper 073-2011
A Better Characterization Routine

Patricia Hettinger, Oakbrook Terrace, IL

ABSTRACT:

One of the great things about SAS® Enterprise Guide is the ability to get usable results without a lot of coding. Another great thing is the ability to take the code generated by an Enterprise Guide task and improve upon it. The standard data characterization task in SAS® Enterprise Guide gives some important information about your data but 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 modify this task to overcome these obstacles.

INTRODUCTION:

The characterization task in Enterprise Guide is a good 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. The author decided to address these 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 some results from the built-in characterization process for a promotion tracking data set. Figure 1 below shows three numeric fields:

Dataset	Variable	Label	Format	N	NMiss
INDATA.PROMOTRACK	ACCT_NO	ACCT_NO		4012109	0
INDATA.PROMOTRACK	REDEEM_DATE	REDEEM_DATE	MMDDYY10.	2500016	1512093
INDATA.PROMOTRACK	REDEEM_CODE	REDEEM_CODE		2500016	1512093

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

Figure 1

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 REDEEM_DATE? ACCT_NO might be a key but you certainly can't tell it from here. The minimum value is 1 for REDEEM_CODE, 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 PROMO_CODE. There were actually thirty distinct values reported (the default). Figure 2 shows the top five plus a general category, 'All other values'.

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

Figure 2

There is definitely room for improvement here. Seeing the range of values for any variable, whether character or numeric would be useful. We can find the number of distinct values and decide a cut-off point for frequencies, regardless of variable type. Best of all, we can write the code once and reuse any number of times with variable substitution. The new output may be seen in figures 3 and 4:

New Summary

Dataset	Libpath	Variable	Label	Format
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	ACCT_NO	ACCT_NO	NUM8
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	REDEEM_DATE	REDEEM_DATE	DATE
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	REDEEM_CODE	REDEEM_CODE	NUM4
INDATA.PROMOTRACK	/DATA/MKT/PROMOS	PROMO_CODE	PROMO_CODE	CHAR8

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

Figure 3

New Frequency:

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

Figure 4

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 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 there is just one) 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 5:

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 5

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 6. 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 6

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.

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.

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..xls;
ODS LISTING CLOSE;
ODS TAGSETS.EXCELXP path="&path"
file="&file" style=analysis;
ODS TAGSETS.EXCELXP options(sheet_name='Summary'
```

```
Autofilter = 'yes'
Frozen_Headers='1'
absolute_column_width='15'
Autofit_height = 'YES');
PROC PRINT DATA=ALL_STATS noobs label;
var dataset libpath Variable db_format min_value max_value miss_value
distinct_values nmiss_value;
RUN;
ODS TAGSETS.EXCELXP options(sheet_name = &freqend
Autofilter = 'yes'
Frozen_Headers='1'
absolute_column_width='15'
Autofit_height = 'YES');
PROC PRINT DATA=storefreqs noobs label;
var dataset libpath variable value count percent;
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 PRINT DATA=ALL_STATS noobs label;….` Print out the summary table, ALL_STATS.
`RUN;`
`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 are set as three programs linked together . Prompts were set up to obtain the different macro variables. Figure 7 shows the default values for &profstart, &profmax and &freqmax.

Enterprise Guide Setup:

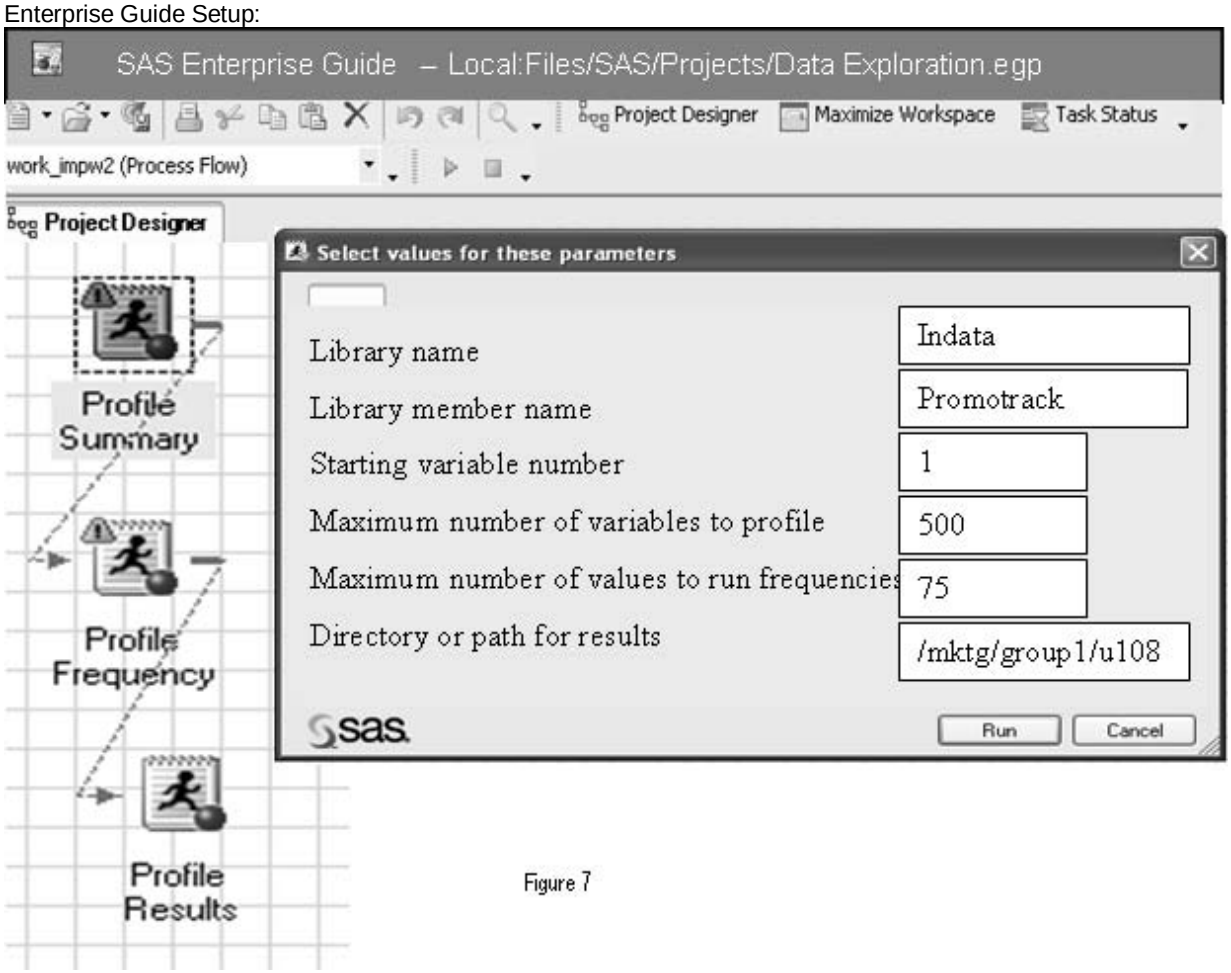

Figure 7

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

This routine should make it much easier to examine your data in a systematic way. In fact , some of you may have ideas for further improvement. However you use these concepts, it's recommended that you profile your data regularly; even those you don't think change much. You will often be surprised but then that's what so much fun about being a data analyst and SAS user!

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