Creating Numeric Formats Based on Precision, Not Magnitude

John R. Gerlach, IMS America; Plymouth Meeting, PA

Abstract

Suppose you need to know the natural format, indicating full precision,of one or more continuous numeric variables. Perhaps you need to produce a codebook that indicates length and format, *not* like those attributes found in the CONTENTS procedure. Or, perhaps, you need to write a raw data file having a record layout that requires optimal precision. This paper discusses numeric data in SAS and explains a robust tool that creates a picture format for each numeric variable in a SAS data set.

Numeric Data

The SAS System stores all numeric values using floating-point (real binary) representation, which allows for numbers of very large magnitude and high precision, that is, the digits to the right of the decimal place. Although this representation provides sufficient means for numeric computation, the user should always consider issues and anamolies, such as storage length and comparison of numeric values.

The descriptor portion of a SAS data set, including the attributes *length* and *format*, does not contain information about the natural length (in digits), that is, the true precision of numeric variables. The length attribute specifies the number of bytes the SAS System uses for storing data, which is by default eight bytes for numeric variables. The format attribute simply tells the SAS System how to writes values.

The Issue of Precision

Sadly, the analyst often has a vague notion about the magnitude of a numeric variable and no idea about its precision. The more experienced analyst wisely performs a univariate analysis of such data, but that does not reveal much about its precision.

Assume you need to produce a raw data file for someone who intends to perform some statistical analysis that will help determine the efficacy of a drug. One of the variables, for example, ranges from 9.0 to 15.0, and its highest value is 14.5. Based on that information the format for the variable would be *99.9*, (where *9* denotes a single digit). In fact, however, most of the variables have two or three digits of precision (e.g., 13.572), albeit less than 14.5. Consequently, assuming that you do not know *a priori* the precision of the data, you can lose substantial information that might adversely affect the analyses.

Of course, when writing numeric data to a flat file, you can use the BEST*w* format and let SAS worry about preserving the precision of the data. However, the data will be written right-justified, not aligned with respect to the decimal. Even when using the *-c* alignment format modifier (e.g., put x best10.5 -c), the PUT statement only centers the values with respect to the width specified in the format, not centered around the decimal point. So, if you intend to produce a viable data file that contains many numeric data, you must consider precision in order to attain a reasonable record layout, as well as preserving fulllest precision. Otherwise, the data file could pose problems that might adversely affect the analysis.

Obtaining Precision

The SAS code presented here processes a SAS data set and produces a picture format for all numeric variables. You need not know the names of the numeric variables or how many there are in the data set. The resulting format name follows a simple convention, that is, as many as the first seven characters of the variable name with a f-suffix. For example, given the variables *systolic*, and *diast* (systolic and diastolic blood pressure) the code creates picture formats called *systolif* and *diastf*, respectively.

Initially, if necessary, you should exclude any numeric variables that are not suitable for processing, for example, nominal type data such as a patient number or social security number. However, the code ignores character variables. Finally, you must specify the name of the input data set, denoted by the macro variable ***dsn***.

```
data cohort;
   set orig.cohort(drop=patno ssn);
run;

%let dsn = cohort;
```

Using Dictionary tables, the SQL procedure defines a macro variable that denotes the number of numeric variables in the input data set. Because the COUNT function produces a right justified number, the next step adjusts a (left justifies) the value of the macro variable for subsequent use. The %sysfunc function allows you to use Data step functions in the %let statement, which accomplishes the task.

```
proc sql noprint;
   select count(*) into: nvars
      from dictionary.columns
      where memname eq "&dsn." and
         substr(type,1,1) eq 'n';
quit;

%let nvars = %sysfunc(
   trim(%sysfunc(left(&nvars.))));
```

The following Data step determines the highest precision of each numeric variable in the input data set. Notice that parallel arrays define both the numeric variables and the respective pattern variables. For each observation, a DO loop traverses the array of numeric variables preserving the highest precision of each. Then, after processing the data set, the OUTPUT statement writes a single observation to the output data set *pats*.

```
data pats;
   retain to '000000000' from '123456789';
   set &dsn.(keep=_numeric_) end=eof;
   array nvars{*} _numeric_;
   array pat{*}$12 pat1-pat&nvars.;
   retain pat1-pat&nvars.;
   do i = 1 to &nvars.;
      tmp = left(put(nvars{i},best12.));
      if (length(tmp) gt length(pat{i}))
         or pat{i} eq ''
         then pat{i} =
            translate(tmp,to,from);
      end;
   if eof then output;
   drop i tmp;
run;
```

The DO loop plays a crucial role in processing the numeric data. First, the PUT function produces a text version of the value using the BEST12 format, which means that SAS will write as many digits as possible, including a decimal point if necessary, using twelve bytes of space. Keep in mind that precision will be lost for the sake of magnitude. So, in order to ensure optimal precision you may need to increase the width of the BEST*w* format, which has a maximum of thirty-two bytes. Secondly, using the LENGTH function, the IF statement compares the text version of the *i*th numeric variable with its preserved respective pattern.

Notice that the IF statement also checks whether the *i*th element of the pattern array contains a missing value. Why? Well, consider the situation where a numeric variable contains only a single digit. In SAS, the length of a character variable whose value is missing has a length of one byte, the same length as the text version of a single digit numeric value. Consequently, the code would fail, without the added expression, to update the pattern in the process of obtaining the highest precision. In fact, the code would not create a format for numeric variables consisting of single digits.

The IF statement determines when to update the pattern array with more accurate information, that is, a pattern denoting greater precision. The TRANSLATE function performs the update using the text version of the numeric value and translating its digits to all zeroes, which emulates the syntax required for writing values without any leading zeroes. The constants *to* and *from* specify the one-to-one mapping of digits that the function uses to translate the digits one through nine to the digit zero.

The final Data step creates a control data set that contains the information needed to create as many formats as there are numeric variables. Keep in mind that the input data set *pats* contains only one observation, representing all the numeric variables, from which all the necessary formats are created. Moreover, all the needed formats are contained in a single data set *picfmts*.

```
data picfmts;
   length fmtname $8 label $12 nvar $7;
   retain start 'LOW' end 'HIGH'
      hlo 'LH' type 'P';
   set pats;
   array nvars{*} _numeric_;
   array pat{*}$12 pat1-pat&nvars.;
   do i = 1 to &nvars.;
      call vname(nvars{i},nvar);
      fmtname = compress(nvar) || 'F';
      label   = pat{i};
      output;
      end;
   keep fmtname type start end hlo label;
run;
```

The resulting control data set, used to create the format(s), contains six variables: *fmtname*, *type*, *start*, *end*, *hlo*, and *label*. The variables *start*, *end*, *hlo*, and *type* indicate the range, from the lowest to the highest value, and the type of format, namely, a picture format used specifically for writing numeric data. The RETAIN statement assigns these constant variables their appropriate values.

For each numeric variable, denoted by its respective pattern, the DO loop assigns the variables *fmtname* and *label* accordingly and outputs an observation. The CALL routine VNAME obtains the variable identifier, which becomes part of the format name, along with the suffix f. The variable *label* contains simply the pattern which contains the proper syntax for creating the desired picture format. Consequently, if there are 20 numeric variables, the data set *picfmts* contains twenty observations, where each format is uniquely defined by the variable *fmtname*.

Using the CNTLIN option of the FORMAT procedures allows you to create multiple formats from a single SAS data set; thus, the code below accomplishes the task. The option *fmtlib* produces a listing of all the formats just created.

```
proc format cntlin=picfmts fmtlib;
run;
```

Notice that the codes makes an assumption concerning variable names, that is, the names are unique up to seven of the eight characters. If not, the FORMAT procedure generates an error message and fails to create the formats.

Using the Picture Formats

As suggested, you might use the formats for writing a more suitable raw data file or, perhaps, for generating a codebook. In creating a data file, via the DATA _null_ step, you would simply use the PUT statement followed by the respective format, which takes its own name from the variable, along with a suffix f. Of course, you decide whether or not to use the format, depending on the intended design of the data file. The codebook application affords an opportunity to obtain better information that indicates the natural length of a numeric variable. Here, you need to export the format from its library, creating a CNTLOUT data set, and use it accordingly.

Conclusion

Processing continouous numeric data poses interesting issues and, sometimes, creates unusual circumstances. Thus, understanding how the SAS System stores and processes numeric data becomes imperative. Also, it is possible to determine the natural format of numeric data, in order to avoid losing any information.

Author Information

John R. Gerlach
IMS America
600 West Germantown Pike
Plymouth Meeting, PA 19462-1048
610.832.5493