PharmaSUG 2019 - Paper BP-057

The Power of the PROC FORMAT

Jonas V. Bilenas, Any Bank
Kajal Tahiliani, GlaxoSmithKline

ABSTRACT

The FORMAT procedure in SAS® is a very powerful and productive tool, yet many beginning programmers rarely make use of it. The FORMAT procedure provides a convenient way to do a table lookup in SAS. User-generated FORMATS can be used to assign descriptive labels to data values, create new variables, and find unexpected values. PROC FORMAT can also be used to generate data extracts and to merge data sets. This paper provides an introductory look at PROC FORMAT for the beginning user and provides sample code that illustrates the power of PROC FORMAT in a number of applications. Additional examples and applications of PROC FORMAT can be found in the SAS® Press book titled “The Power of PROC FORMAT."

INTRODUCTION

Building a user defined FORMAT can be viewed as a table lookup where VALUES are mapped to LABELS. Let us look at some table lookup examples.

Typical lookup tables use 1-to-1 or many-to-1 mappings. As an example of a 1-to-1 table lookup, we have a data set of character credit approval decision codes. When we generate reports of approval rates, we wish to map decision code values into literal labels. The 1-to-1 mapping is illustrated here:

```
‘a’ = ‘Approve’
‘d’ = ‘Decline’
```

If we have many approval codes and many decline codes, these can be mapped or assigned or grouped to the appropriate label in a many-to-1 mapping. For example;

```
‘a1’, ’a2’, ’a4’ =  ‘Approve’
‘d1’, ‘d6’       =  ‘Decline’
```

In the next section, we will look at how we might generate table lookups in SAS. The first method relies on the DATA step and the second approach will build the lookup using PROC FORMAT.

TABLE LOOKUP USING A DATA STEP

For this example, we will look at groupings of credit bureau risk scores. These scores are often used in credit decisions. One such score has integer values from 370 to 870 with exception scores outside this range. Higher values of the score relates to better credit quality and reduced risk.

We wish to run a frequency distribution on individuals with scores grouped into 3 classes; 370-670, 671-870, and un-scored. This is an example of a many-to-1 mapping where we will need to group scores into 3 categories. A beginning programmer would often handle this by creating another SAS data set where a new variable is generated to assign membership into categories. This new data is then used in a PROC FREQ to generate the report. Here is some code generated by the SAS programmer:

```
data stuff;
  set cb;
  if 370<= score <= 670 then group='670-';
  else if 670 < score <= 870 then group='671+';
  else group='unscored';
run;

proc freq data=stuff;
  tables group;
run;
```

Results from the code are illustrated in output 1.

```
                              Cumulative  Cumulative
GROUP   Frequency   Percent   Frequency   Percent
---------------------------------------------------
671+         623      10.7         623       10.7
670-        5170      89.2        5793       99.9
unsc           5       0.1        5798      100.0
```

Output 1. Output from PROC FREQ

The code did the job, but it required that a new DATASTEP be created and the label ‘unscored’ was truncated to ‘unsc’.

TABLE LOOKUP USING PROC FORMAT

Using a user defined FORMAT saves some processing time and resources. The same problem solved using PROC FORMAT is illustrated here.

```
proc format;
  value score 370 - 670 = '670-'
              670<- 870 = '671+'
              other     = 'unscored'
  ;
run;

proc freq data=cb;
  tables score;
  format score score.;
run;
```

Code returns output shown in Output 2.

```
                                Cumulative  Cumulative
  SCORE   Frequency   Percent   Frequency    Percent
------------------------------------------------------
670-           5170      89.2        5170       89.2
671+            623      10.7        5793       99.9
unscored          5       0.1        5798      100.0
```

Output 2. Output using user defined FORMATS in PROC FREQ

Some observations to make:

1. Assignment of the FORMAT occurs in PROC FREQ with a FORMAT statement.
2. I end the format definition with the ‘;’ on a new line. This is just my preference but I find it easier to debug PROC FORMAT code especially if we add more values to the FORMAT later on.
3. The ‘unscored’ label now appears without truncation and without modification of the spaces in the labels..

Syntax rules for PROC FORMAT will be reviewed in a later section. Let’s look at another application of PROC FORMAT to find unexpected values.

USING PROC FORMAT TO FIND UNEXPECTED VALUES

User defined formats can be used to list out unexpected values. If a range of values are not mapped in a PROC FORMAT, the original values will be returned as labels. Here is an example:

```
proc format;
  value looky 370-870 = '370-870'
  ;
run;

proc freq data=cb;
  tables score;
  format score looky.;
run;
```

Output 3 shows results from the above code.

```
                                          Cumulative  Cumulative
   SCORE    Frequency    Percent    Frequency     Percent
-----------------------------------------------------------------
370-870       30320       96.0        30320        96.0
   9003        1264        4.0        31584       100.0
```

Output 3. Finding Unexpected Data Values

With the above example we run the risk of truncating the output of values if the values have a width larger than the width of the FORMAT label. For this reason, it is better to use an embedded FORMAT as follows:

```
proc format;
  value looky 370-870 = '370-870'
              other   = [best.]
  ;
run;
```

GENERATING NEW VARIABLES WITH PROC FORMAT

New variables can be assigned within a data step using user defined FORMATS. A nice feature of using FORMATS for generating new variables is that the method can replace IF/THEN/ELSE code. By default, PROC FORMAT will not allow 1-to-many or many-to-many mapping. There are no checks for accidental 1-to-many or many-to-many mapping in IF/THEN/ELSE code within a data step.

In this example we wish to assign a credit line based on a risk score range.

```
proc format;
  value stx  low - < 160 = '1000'
             160 -   179 = '2500'
             180 -   199 = '5000'
             200 -   219 = '7500'
             220 -  high = '9500'
  ;
run;

data scores;
  set bbu.scores;
  line = input(put(score,stx.),best12.);
run;
```

With the above code, a new variable called LINE is generated from the call of the FORMAT STX in the PUT function. PUT function will always return a character, so the INPUT function was used to convert the variable to numeric since we required that the LINE variable be a numeric variable.

WHAT ABOUT A 2 DIMENSION TABLE LOOKUP?

Taking the last example, what if we wanted to offer different lines as a function of score for 20% of the records? Here is sample code:

```
 proc format;
      value use  low   -  0.8 = 'stx'
                 0.8 < - high = 'sty'
      ;
  value stx  low - < 160 = '1000'
             160 -   179 = '2500'
             180 -   199 = '5000'
             200 -   219 = '7500'
             220 -  high = '9500'
  ;
  value sty  low - < 160 =  '1500'
             160 -   179 =  '3200'
             180 -   199 =  '6500'
             200 -   219 =  '8000'
             220 -  high = '10000'
  ;
run;
data scores;
  set bbu.scores;
  fmtuse = put(ranuni(83),use.);                         ❶
  line = input(putn(score,fmtuse),best12.);              ❷
run;
```

Some comments about the code:

❶ I take a uniform random number and assign ‘stx’ 80% of the time and ‘sty’ 20% of the time as values to variable FMTUSE using a PUT function.

❷ I use the PUTN function which assigns, as the second argument, the value of FMTUSE to be used as the FORMAT. This will dynamically assign the format based upon values of the FMTUSE variable. Note that the second argument in the PUTN function does not end in a dot since the second argument is not a FORMAT but a variable name that contains the FORMAT designation. For character FORMATS use the PUTC function.

More examples of 2-dimensional and 3-dimensional table lookups using PROC FORMAT can be found in the SUGI31 paper by Perry Watts, “Using Database Principles to Optimize SAS® Format Construction from Tabular Data”.

USING PROC FORMAT TO EXTRACT DATA

User defined formats can be used to extract a subset of data from a larger DATASET. Here is an example:

```
proc format;
  value $key '06980'                 = 'Mail1'
             '06990','0699F','0699H' = 'Mail2'
             other                   = 'NG'
  ;
run;

data stuff;
  set large.stuff;
  where put(seqnum,$key.) ne 'NG';
run;
```

Note that for this example we are generating a character FORMAT that will map character values. Character FORMATS must start with a ‘$’.

Let’s review some syntax rules for setting up user defined FORMATS in the next sections.

SPECIFYING RANGES OF VALUES IN PROC FORMAT

Ranges of values can be specified in a number of ways and special keywords can be used in the expression of the range.

1. VALUES can be single values or values separated by commas:
   - ‘x’
   - ‘a’, ‘b’, ‘c’
   - 1, 22, 43
2. Ranges (numeric or character) can include intervals such as:
   - A – B. Interval includes both endpoints.
   - A <- B. Interval includes higher endpoint.
   - A - < B. Interval includes lower endpoint.
   - A <- < B. Interval does not include either endpoint.
3. Ranges can be specified with special keywords:
   - LOW, HIGH, OTHER, .,‘ ’
4. The LOW keyword does not format missing values for numeric formats. For character formats, LOW includes missing values.
5. The OTHER keyword does include missing values unless accounted for with specification of missing values.

OTHER FORMAT REQUIREMENTS

- For SAS8 and earlier, format names must be 8 characters or less. For SAS9, the number of characters a format name can have is 32. These lengths include the dollar sign required for character formats.
- FORMAT names cannot be identical to existing internal SAS format names.
- FORMAT names cannot end or begin with a number.
- Character FORMATS must begin with a “$”.
- INFORMATS can be created in PROC FORMAT with the INVALUE statement. This paper did not touch on the steps to create user informats. To review the INVALUE statement of PROC FORMAT, refer to SAS documentation and/or SAS PRESS book “The Power of PROC FORMAT” (2005, Bilenas).

USING PROC FORMAT FOR DATA MERGES OR EXTRACTS

PROC FORMAT also offers a method of merging large data sets (up to a few million, depending on memory resources) to very large (millions and millions) unsorted SAS data sets or flat files. The method first builds a user defined format from a special data set. The requirements for this data set are that it must not have any duplicates in key fields and have at least these variables:

- FMTNAME: name of format to create.
- TYPE: ‘C’ for character or ‘N’ for numeric, ‘I’ is for INFORMAT.
- START: the value you want to format into a label. If you are specifying a range, START specifies the lower end of the range and END specifies the upper end. End is not required when you have unique START variables mapping to LABELS.
- LABEL: the label you wish to generate.

Remember **LEFTS**: LABEL, END, FMTNAME, TYPE, and START. END is not required unless you have a range of VALUES to map to a single LABEL.

Once the data is generated, a FORMAT is generated from the data and then applied to match records from the larger unsorted SAS data set or flat file. Here is an example of code applied to a large unsorted SAS data set.

```
proc sort data=small out=temp nodupkey force;          ❶
  by seqnum;
run;

data fmt (rename=(seqnum=start));                      ❷
  retain fmtname 'key'                                 ❸
         type  'C'
         label 'Y';
  set temp end=eof;
  output;
  if eof then do;                                      ❹
    label = 'N';
    HLO   = 'O';                                       ❺
    output;
  End;
run;

proc format cntlin=fmt;                                ❻
run;

data match;                                            ❼
  set bigfile;
  where put(seqnum,$key.)= 'Y';
run;
```

Some observations on above code:

- The sort of the small DATASET (❶) was done to ensure no duplicates of the key variable, SEQNUM.
- In line ❷ we create the dataset that will be used by PROC FORMAT to generate the FORMAT.
- Also, on line ❷, we need to rename the key variable SEQNUM to START.
- Since FMTNAME, TYPE and LABEL will not change for each record we can use the RETAIN statement starting on line ❸ (RETAIN is more efficient than variable assignment statements).
- Note that we set FMTNAME to ‘key’. In this example, the format type is character since the key field on which to match is character. We also assign a value of ‘C’ to TYPE to indicate that we are setting up a character format. This code will work if we call FMTNAME ‘key’ or ‘$key’. Another interesting quirk of PROC FORMAT.
- Beginning with line ❹ we start a DO loop to handle specification of an ‘OTHER’ condition. The HLO variable specified in line❺ will handle the OTHER specification by setting HLO=’O’. It is a good idea to set the

START variable to missing to avoid a one-to-many mapping. At the end of the loop, we output another record where LABEL is set to ‘N’.

- In❻, we specify the **CNTLIN=** option on the PROC FORMAT statement. This will read in the data FMT and generate the FORMAT $KEY that we will then use in the merge-extract DATA step starting on line❼.

SAVING FORMATS IN FORMAT CATALOGS

Sometimes you may want to save your formats in permanent catalogs (named formats.sas7bcat) to use in other code or to be used by other users. Use the LIBRARY= option:

```
libname library ..
proc format library = library;
  value ..
  ;
   run;
```

The catalog generated by the above code can store many FORMATS in a single catalog. You can create many catalogs to hold different types of FORMATS. For example, you can create a catalog for FORMATS associated with finance data using the following code that generates a catalog named FINANCE.sas7bcat.

```
libname library ..
proc format <options> library = library.FINANCE;
/* additional code if required */
run;
```

To use the saved format catalogs in subsequent programs you don’t have to repeat the FORMAT code but you will need to specify the destination of the catalogs via LIBNAMES. Formats are searched, by default, in WORK and then LIBRARY LIBNAMES. You can add at FMTSEARCH in OPTIONS statement to specify additional directories to search for FORMATS stored in directories other than one specified by the LIBRARY.

Note that format catalogs cannot be FTPed to other operating systems. What you can do is to convert the catalog to a CNTLOUT data set that can be FTPed to other systems and then use the CNTLOUT as a CNTLIN to generate the formats in the system where the file was sent to. An example of using a CNTLIN option to generate the catalog is shown next.

LAGS AND LEADS FOR TIME SERIES DATA USING PROC FORMAT

Lately I have been storing time series data in SAS FORMAT CATALOGS to quickly do a table look-up using SAS DATES as VALUES and data as the LABELS. Let's take a look at some code to illustrate the application.

The example we will work on here is from sample data from https://datahub.io/collections/economic-data using CPI data as an example. Monthly CPI data is from 01JAN1913 to 01JAN2014.   1,213 observations loaded to a SAS data set. I used a data step to read in the data using DATALINES and the ANYDTDTE INFORMAT to read in the date values into the variable DATE. Part of the LOG output is shown here to verify that the DATE values are correct:

```
74       data stuff.cpi;
75         input date anydtdte10.
76                  index
77                  Inflation;
78       if _N_ < 10 then do;
79         put date date9. +5 index +5 inflation ;
80       end;
81       datalines;
01JAN1913 9.8 .
01FEB1913 9.8 0
01MAR1913 9.8 0
01APR1913 9.8 0
01MAY1913 9.7 -1.02
01JUN1913 9.8 1.03
```

OUTPUT 4. SECTION OF THE LOG.

Code to load the data to a permanent format catalog is displayed here:

```
options nocenter fullstimer;
libname  stuff '/somewhere/FORMATS';
libname library '/somewhere/FORMATS';
data fmt  / view=fmt;
  set stuff.cpi  (rename=(date=start index=label)) end=eof;
  retain fmtname 'CPI' type 'N';
  output;
  if eof then do;
    start=.;
    label=.;
    HLO='O';
    output;
  end;
run;
proc format cntlin=fmt library= library;
run;
```

Real run time was a bit more than a second. I have often stored hundreds of macroeconomic variables in permanent FORMAT catalogs. You need to run only one time and use the catalog data for multiple time series regression models. You can create a macro to store each FORMAT in one data step or run each format run separately. When you run multiple formats in a single CNTLIN data you will need to run a PROC SORT on the data using BY FMTNAME. The catalog data can be updated as needed when new data become available.

The formats are saved in the library LIBNAME as formats.sa7bcat. Let's look at 2 application of using the FORMAT catalog. As an example, say you have a data set with monthly dates anchored at the first of each month from 01JAN2000 to 01DEC2012. You want to pull in the CPI for each of the months. Code is shown below. By default, the format search will, by default look in the WORK directory followed by the LIBRARY directory. You can use a FMTSEARCH option to specify other CATALOGS to seek the FORMATS:

```
options nocenter fullstimer mprint symbolgen;
libname  stuff '/somewhere/FORMATS';
libname library '/somewhere/FORMATS';
data test2;
  set test;
  cpi = input(put(date,CPI.),best.);
run;
```

As another example, what if you wanted to also calculate a 12 month running average and a 48 month running average. You can use LAG functions in a data step. But wait that won't work since you don't have the historical months in the data. Have no fear, we can use the FORMAT CATALOG to pull in the data and calculate the running averages using a macro. Code is shown here. The macro assumes that the average starts at the specific DATE variable 11 back and runs to the current month. Note that we increment the date by months using the **INTNX** date function. By default, the month differences are anchored to the beginning of the month. To anchor to the end of the month use a fourth argument of INTNX set to 'e'. Use 'm' for middle of the month. By default, 'b' is assumed for the beginning. Note that you can go back to the past and into the future with the INTNX function. The first argument in INTNX is the interval that you want to move by. For date variables, the choices are; **DAY, WEEK, WEEKDAY, SEMIMONTH, MONTH, QTR, SEMIYEAR, YEAR.** The second argument is the start-from variable, here the variable name is date. The 3rd argument is the increment. By default, the anchor is the 4th argument; B (default) for beginning of the interval, M for middle, E for END, and S for SAME alignment as the input date.

Code:

```
%macro mv_avg (mnths) ;
  %let start = 1-&mnths.;
  roll&mnths= mean(
  %do i = &start %to 0;
    input(put(intnx('month',date,&i.),CPI.),best.)
    %if &i. ne 0 %then %do; , %end;
  %end;
  )
%mend;

data test2;
  set test;
  cpi = input(put(date,CPI.),best.);
  %mv_avg(12);
  %mv_avg(48);
run;

proc sgplot data=test2;
  series x=date y=cpi;
  series x=date y=roll12;
  series x=date y=roll48;
  xaxis discreteorder=formatted
        grid interval=YEAR;
  yaxis grid;
  format date year.;
  title Cool Graph, No?;
  inset "THE POWER OF PROC FORMAT" /
        Border position=bottomright;
 run;
```

The plot from SGPLOT is shown in figure 1.

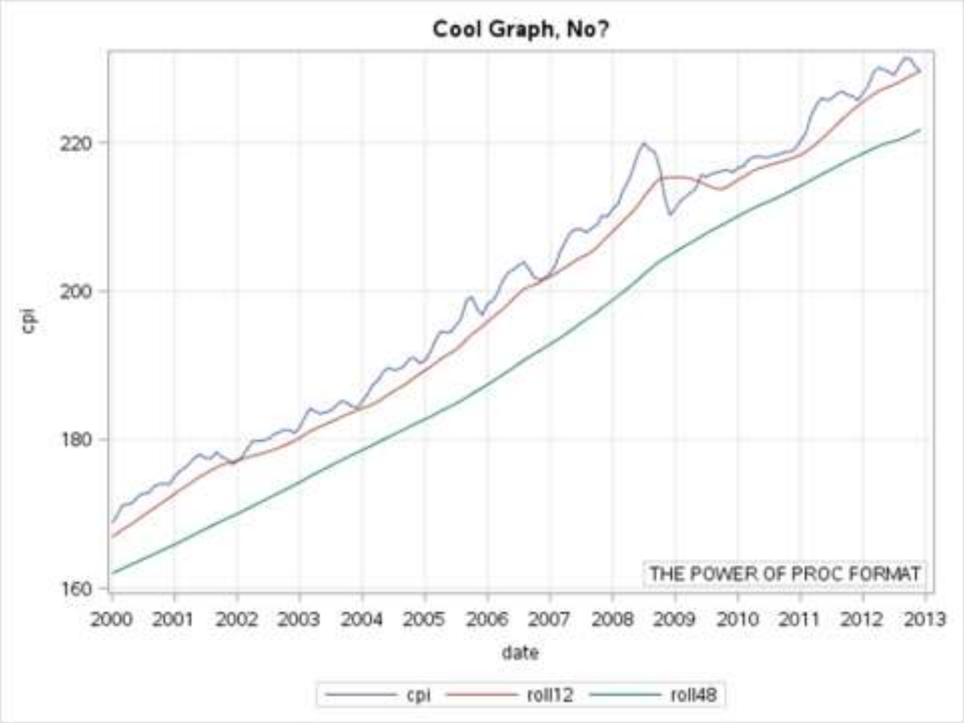

FIGURE 1. OUTPUT FROM SGPLOT CODE

PICTURE FORMATS

PICTURE FORMATS provide a template for printing numbers. The template can specify how numbers are displayed and provide a method to deal with:

- Leading zeros.
- Decimal and comma placement.
- Embedding characters within numbers.
- Prefixes.
- Truncation or rounding of numbers.

Many examples are included in (2005, Bilenas). One example of using PICTURE FORMATS is to add a trailing ‘%’ in PROC TABULATE output when a PCTSUM or PCTN calculation is specified. For this example, we also use the ROUND option so that the number is rounded rather than truncated. This code will print a leading 0 if the percentage is less than 1 (i.e., .67% displays as 0.67%) since we include a digit selector with a value other then 0 to the left of the decimal point. With the two ‘9’ values after the decimal, 2 digits will be displayed after the decimal even if one or more are 0:

```
proc format;
  picture p8r (round) 0-100 = '0009.99%'
  ;
run;
```

The above example will remove negative signs for negative values when applying the format. This may not be an issue in PROC TABULATE when using PCTSUM or PCTN statistics, but you may want to be safe and modify the code as follows:

```
proc format;
  picture p8r (round)
    low - < 0 = '0009.99%' (prefix='-')
    0 - high  = '0009.99%'
  ;
run;
```

CREATING MULTI-LABEL FORMATS

With the introduction of SAS8, we have a capability to map values to more than one label. Multi-label formats can be used in PROC SUMMARY and PROC TABULATE. Let us take a look at an example that we introduced earlier where we are mapping credit decision codes into labels:

‘a1’, ’a2’, ’a4’ = ‘Approve’
‘d1’, ‘d6’ = ‘Decline’

We wish to generate a frequency report for each decision and get totals for each category. Here is the code that generates the formats, using hypothetical data, and the summary report:

```
proc format;
value key low -    0.20 = 'a1'                        ❶
        0.20 < - 0.25 = 'a2'
        0.25 < - 0.35 = 'a4'
        0.35 < - 0.80 = 'd1'
        0.80 < - high = 'd6’
;
picture p8r (round)                                 ❷
   low - < 0 = '0009.99%' (prefix='-')
   0 - high  = '0009.99%'
;
```

```
  value $deccode (multilabel notsorted)                          ❸
         'a0' - 'a9' = 'APPROVE TOTALS'
         'a1'        = ' a1: Approval'
         'a2'        = ' a2: Weak Approval'
         'a4'        = ' a4: Approved Alternate Product'
         'd0' - 'd9' = 'DECLINE TOTALS'
         'd1'        = ' d1: Decline for Credit'
         'd6'        = ' d6: Decline Other'
   ;
run;


data decision;
  do id = 1 to 1000;
    decision = put(ranuni(7),key.);                             ❹
    output;
  end;
run;

proc tabulate data=decision noseps formchar='          ';
  class decision/mlf preloadfmt order=data;                     ❺
  format decision $deccode.;
  table (decision all)
        ,n*f=comma5.
        pctn='%'*f=p8r.
        /rts=33 row=float misstext=' ';
run;
```

In the PROC FORMAT section of the code we create 3 formats. In line ❶ we generate the format that will assign decision code to records in a SAS data set based on a uniform random number generated in a DO loop (❹).
The second format in line ❷ generates the PICTURE format for displaying percent signs in PROC TABULATE.

The final format in line ❸ is used to generate the multi-label format. Some comments on this format:

1. Note that we must specify the (multilabel) option when generating the format.
2. We can preserve the order of formatted values on tabulate output by specifying the NOTSORTED option in the generation of the FORMAT and PRELOADFMT ORDER=DATA options in line❺; the CLASS specification in TABULATE. Sorry, PHARMASUG folks mainly use PROC REPORT but it should work with that procedure. It will not work with the FREQ procedure since there is no CLASS statement in PROC FREQ. Give it a try and see what happens.
3. Note that we have labels for each of the 5 decision codes. We also map all codes beginning with the letter ‘a’ into ‘APPROVE TOTALS’ and all those beginning with the letter ‘d’ into ‘DECLINE TOTALS’
4. The CLASS statement in TABULATE or SUMMARY (❺) must include the MLF option to generate the multi-label output.

Output from TABULATE code is shown in Output 5.

```
                                   N      %

decision
APPROVE TOTALS                    314   31.40%
 a1: Approval                     163   16.30%
 a2: Weak Approval                 45    4.50%
 a4: Approved Alternate Product   106   10.60%
DECLINE TOTALS                    686   68.60%
 d1: Decline for Credit           453   45.30%
 d6: Decline Other                233   23.30%
All                             1,000  100.00%
```

Output 5. Illustrating Multi-Label FORMATS

CONCLUSION

The FORMAT procedure allows users to create their own formats that allow for a convenient table look up in SAS. Using PROC FORMAT will make your code more efficient and your output look more professional.

REFERENCES AND ADDITIONAL READING:

- Bilenas, J. “The Power of PROC FORMAT”, SAS Press, 2005. No longer in print.
- Bilenas, J.V. “The Power of PROC FORMAT”, SAS Press, 2005. No longer in print.
- Bilenas, J.V., Tahiliani, K. (2016). “The Power of Proc Format, http://analytics.ncsu.edu/sesug/2016/BB-103_Final_PDF.pdf
- Bilenas, J.V. (2007) “Using SAS® Dates and Times – A Tutorial “, SAS GLOBAL FORUM 2007, https://support.sas.com/resources/papers/proceedings/proceedings/forum2007/226-2007.pdf
- Carpenter, A. “Looking for a Date? A Tutorial on Using SAS Dates and Times”, SUGI30, 2005.
- Morgan, D. “The Essential Guide to SAS Dates and Times“, SAS Press. 2006
- Watts, P. Using Database Principles to Optimize SAS® Format Construction from Tabular Data, SUGI31, 2006

This work is an independent effort and does not necessarily represent the practices followed at current or previous employers.

KEYWORDS: **FORMAT, LEADS, LAGS, PICTURE, INFORMAT, DATE, MULTILABEL, TABULATE**