Paper 116-2007

Win the Pennant! Use PROC FORMAT

Marge Scerbo, National Study Center, UMB

Abstract

Mr. Torre is the manager of a fictitious New York major league baseball team. Torre needs to produce some reports for the team's owner. He's been discussing these reports with his analysts:

1. The first set of reports show the frequencies for the various positions on the active roster. The owner has requested that these frequencies report the description rather than the codes.
2. Torre needs reports based on the players' ages. The results may differ depending on how the age breaks are defined.
3. For clarity, some of the reports need the players' names and not their team uniform number.
4. A new variable is required that is based on the team uniform number table.

All of these requests can be accomplished with basic SAS® DATA step code. Often, the DATA step is the best and simplest solution. But, consider that PROC FORMAT may provide easy resolution to these problems. This hands-on workshop will introduce you to the basics of this procedure. We will step through the DATA step code and then produce the same answers using Proc FORMAT.

Introduction

The FORMAT procedure can be a programmer's best friend once the initial hurdles of understanding its usage have been overcome. No procedure can replace the power of the SAS DATA step, but learning to use procedures in the right place and at the right time may provide new and efficient methods to the same end. There are many uses for Proc FORMAT including the ability to add labels to data values. This paper and workshop will specifically cover the VALUE statement to label values or groups of values.

Proc FORMAT is part of the Base SAS language. It can create formats that in turn can be used by either another DATA or PROC step.

DATA

Throughout this presentation, we will be using different datasets containing information about the Yankee players. The first table (YANKEES) has some basic information about the players, their heights, weights, birth dates, and position. This tables contains a key variable, *number*, that will be used for identification of each player. Here is a sample:

Number	B/T	Ht	Wt	DOB	Age	Position
26	R/R	74	245	8/10/1971	35	C
20	S/R	74	205	8/17/1971	35	C
22	L/R	72	190	10/22/1982	23	IF
25	L/R	75	230	1/8/1971	35	IF
17	R/R	74	175	9/10/1978	27	IF
2	R/R	75	195	6/26/1974	32	IF
13	R/R	85	225	7/27/1975	30	IF
53	L/R	72	210	3/11/1974	32	OF
28	S/L	69	170	8/11/1984	22	OF
18	L/L	74	205	11/5/1973	30	OF

The next dataset (BATTINGORDER) is the batting order of the team at one point in the season, as well as sample batting statistics. Again, each player is identified with his number.

BO	Number	Games	AB	R	H	2B	3B	HR	RBI	BB	SO	AVG
1	2	113	452	81	153	27	3	10	71	54	74	0.338
2	18	111	440	85	127	26	4	18	59	53	62	0.289
3	13	116	439	82	125	19	1	25	86	67	107	0.285
4	25	108	359	74	91	17	0	34	93	87	84	0.253
5	20	105	345	43	93	17	1	15	59	47	71	0.270
6	51	100	332	47	93	23	0	8	46	24	42	0.280
7	28	86	319	53	93	17	2	7	38	35	36	0.292
8	17	79	309	37	101	22	1	7	34	12	34	0.327
9	53	88	228	27	55	10	3	7	27	12	48	0.241
10	22	67	197	25	46	8	3	0	25	10	29	0.234

The last dataset (NAMES) is a file containing the players' names and numbers. This file will be used in constructing a format table:

Number	Name
26	Sal Fasano
20	Jorge Posada
22	Robinson Cano
25	Jason Giambi
17	Nick Green
2	Derek Jeter
13	Alex Rodriguez
53	Bobby Abreu
28	Melky Cabrera

With these three datasets, we will be able to create the reports the Yankees' owner has requested.

DATA Step Variable Labels

First, it is important to clearly understand the difference between a variable label and a value label.

- By using a LABEL statement in either a DATA or PROC step, a variable label can be attached to a variable. This label adds descriptive information to the name of the variable, for example the variable *SSN* labeled as 'Social Security Number'.
- A value label for a variable is usually used for one of two purposes:
  - to describe the codes or other values of the variable
  - to group values for study

Grouping is very important in many studies:

- Many analyses are reported by groups, not by individual values
- Groups often show patterns that individual analyses cannot
- In frequency studies where there are continuous numbers or a large number of values to be reported, grouping limits the amount of output

Beginning SAS programmers learn to add descriptive labels to their variable values by using IF/THEN/ELSE statements and creating new variables. This process is completely correct but may be inefficient if the new variable is not needed on a regular basis and/or the dataset is very large. Often, these new variables are needed for one-time reports and not for any analytic process.

In the example below (Report 1), IF/THEN/ELSE statements are used to create values for the field, *position*, by creating a new variable called *positionname*. Note that the IF statement is preceded by a LENGTH statement to define the new variable, *positionname*. This length is equal to the longest possible value of the variable. The IF statement queries the value of *position* until a match for the variable value is found. Since the variable is a character variable, note that the values are placed in quotes:

```
*Report 1;
*data step labels;
 data positions;
       set pap.yankees;
       length positionname $8;
       if position = 'C'    then positionname = 'Catcher';
       else if position = 'IF' then positionname = 'Infield';
       else if position = 'OF' then positionname = 'Outfield';
       else if position = 'P'  then positionname = 'Pitcher';
run;
proc freq data = positions;
       tables positionname / nocum;
run;
```

In this example, a new dataset, *positions*, is created with a new variable, *positionname*. This process produces a clean LOG, and the output requested, but it is always important to assess efficiency, especially when large datasets are involved.

The output created from this DATA and PROC step is shown below:

Position	Frequency	Percent
Catcher	2	8.00
Infield	5	20.00
Outfield	5	20.00
Pitcher	13	52.00

This DATA step process is correct, but Proc FORMAT can also provide the same output.

PROC FORMAT Syntax

There are a number of rules for creating and using formats. As is the case in most SAS code, the variable type is extremely important and, as such, defines which format type should be used:

- To format a character variable, create a character format
- To format a numeric variable, create a numeric format

Format names are defined in the VALUE statement. The rules for format names are:

- Character formats must begin with a $
- Numeric formats cannot begin with a $
- A valid SAS format name can be up to 32 characters in length (the $ counts as one of the 32 characters in a character format name)
- A format name must start with a letter or underscore (the first character after the $ in a character format name)
- Format names can contain only letters, numbers, and underscores
- A format name cannot end in a number
- A format name cannot be the name of a SAS function

Variable values on the left side of the equal sign can be:

- An individual value
- Multiple values separated by commas
- A range of values separated by a dash
- Combinations of individual values, multiple values, and ranges
- Values should not be repeated or overlapped
- Numeric values cannot be not enclosed in quotes

- Character values can be enclosed in quotes
- The keywords LOW and HIGH can be used to define the lowest values and highest values of the variable when the format is associated with a variable
- The keyword OTHER can be used to capture other non-missing values
- The keywords LOW, HIGH, and OTHER do not categorize missing values.

The rules for the values on the right side of the equal sign include:

- A format label may be up to 256 characters in length
- Quotation marks should be placed around format labels

Multiple formats can be created with one Proc FORMAT by using multiple VALUE statements. Remember, variable value(s) on the left side of the equal sign print as the character value(s) on the right.

Formatted values are always character, regardless of whether the variable and the format is character or numeric. These formatted values cannot be used as numbers, but the internal value is unchanged, so it can still be used in procedures.

You can never go wrong by first running a frequency on the raw values and then on the corresponding formatted values to see if they make sense.

Now that the rules are laid out, let's create the Report 1 using a Proc FORMAT rather than a DATA step:

```
*Report 1 ;
proc format;
      value $posname
            C  = 'Catcher'
            IF = 'Infield'
            OF = 'Outfield'
            P  = 'Pitcher' ;
run;
proc freq data = pap.yankees;
      tables position / nocum;
      format position $posname.;
run;
```

The first step is to create a new format:

- In this case a temporary format is created. If the format is temporary, it must be created in the same session (and prior to) any PROC or DATA step accessing the format:
  ```
  proc format;
  ```
- The VALUE statement names the format to be created, in this case a character format called *$posname*. Note that this format does NOT end in a period and is NOT followed by a semicolon:
  ```
  value $posname
  ```
- Now the variable values and their labels are listed. Note that the labels are enclosed in quotes and that the last variable description is followed by a semicolon:
  ```
  C = 'Catcher'
  /* many more here if needed*/
  IF = 'Infield' ;
  ```
- Remember when dealing with character values, that the match must be exact. If the value to be formatted is uppercase, make sure the format is also uppercase!

Numeric Formats

The creation of user defined numeric formats is very similar to the character format creation. There are several important issues to remember:

- The format name does not begin with a dollar sign
- The labels created are character, not numeric, and therefore can not be used in calculations
- The internal value of the number is not changed

In the following example, we will characterize the height of each player as Short, Medium, or Tall. We can do this using DATA step IF/THEN/ELSE statements. Note that the variable values are not placed in quotes:

```
*numeric data step labels;
data yankeesheight;
      set pap.yankees;
      length height $8;
      if ht lt 72 then height = 'Short';
      else if ht lt 78 then height = 'Medium';
      else height = 'Tall';
run;
proc freq data = yankeesheight;
      tables height / nocum;
run;
```

Results in this output:

height	Frequency	Percent
Medium	21	84.00
Short	1	4.00
Tall	3	12.00

We can accomplish the same value labeling using Proc FORMAT. Note that neither a DATA step nor a new dataset is required in order to produce the same report. In the format below, the keywords LOW and HIGH are used:

```
*numeric format;
proc format;
      value height
            low - 71  = 'Short'
            72 - 77   = 'Medium'
            78 - high = 'Tall' ;
run;
proc freq data = pap.yankees;
      tables ht / nocum;
      format ht height.;
run;
```

In this example, we created a numeric format called *height* and used this format in the Proc FREQ of the players' heights (variable *ht*). The resulting table is the same as the DATA step process.

Age Formats

Sometimes the study of a field or fields may require multiple views in order to obtain a reasonable and/or useful result. Age is a good example of that type of analytic field. The abstract discusses the need for a report (Report 2) studying the players' age, although no specific categorization has been identified. It makes sense to group age in different ways to identify which best fits the needs of that particular study.

```
*Report 2;
data ages;
      set pap.yankees;
      length age1 age2 $10;
```

```
        if age lt 23 then age1 = 'Youngster';
        else if age lt 29 then age1 = 'Great Age';
        else if age lt 35 then age1 = 'Watch Them';
        else age1 = 'Retire?';

        if age lt 25 then age2 = 'Toddlers';
        else if age lt 30 then age2 = 'Kids';
        else if age lt 35 then age2 = 'Juniors';
        else age2 = 'Seniors';
run;

proc freq data = ages;
        tables age1 age2;
run;
```

Producing these listings:

age1	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Great Age	5	20.00	5	20.00
Retire?	9	36.00	14	56.00
Watch Them	10	40.00	24	96.00
Youngster	1	4.00	25	100.00

age2	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Juniors	7	28.00	7	28.00
Kids	6	24.00	13	52.00
Seniors	9	36.00	22	88.00
Toddlers	3	12.00	25	100.00

Now we will accomplish the same results using Proc FORMAT.

```
*Report 2;
proc format;
        value agea
                low - 22    = 'Youngster'
                23 - 28     = 'Great Age'
                29 - 34     = 'Watch Them'
                35 - high   = 'Retire?'     ;
        value ageb
                low - 24    = 'Toddlers'
                25 - 29     = 'Kids'
                30 - 34     = 'Juniors'
                35 - high   = 'Seniors'     ;
run;
```

```
proc freq data = pap.yankees;
       tables age;
       format age agea.;
       title 'Age Grouping 1';
run;
proc freq data = pap.yankees;
       tables age;
       format age ageb.;
       title 'Age Grouping 2';
run;
```

Formats for Crosstabs

One descriptive mechanism for studying data is to create cross tabulation tables to see where the data values intersect. Creating crosstabs with Proc FREQ can produce volumes of results, often obscuring patterns. This is particularly a problem with continuous data. There may be times where creating formats to study these crossing might prove efficient.

The following code will produce a table with little (or maybe too much) information:

```
proc freq data = pap.battingavg;
       tables r * ab / nopercent norow nocol nocum;
       label  r  = 'Runs'
              ab = 'At Bat';
run;
```

We may be better served by first creating two separate formats, one for runs and one for at bats. Note that one Proc FORMAT can create any number of formats. Using these formats, we will then run the same frequency procedure to create output that is easier to read.

```
proc format;
       value run
              low - 10  = 'Mediocre'
              11 - 25   = 'Average'
              26 - 50   = 'Good'
              51 - high = 'Best'  ;
  *note that there is no new proc format or run statement needed here;
       value ab
              0 - 100    = 'Few'
              101 - 200  = 'So So'
              201 - 300  = 'Good'
              301 - 400  = 'Better'
              401 - high = 'Best'  ;
run;
proc freq data = pap.battingavg;
       tables r * ab / nopercent norow nocol nocum;
       format r run. ab ab.;
       label r  = 'Runs'
             ab = 'At Bat';
run;
```

The resulting table is quite specific. Only three players are rated best in both at bats and runs while 9 have not had many runs. This type of output provides a pretty clear view for the team owner.

Table of R by AB						
R(Runs)	AB(At Bat)					Total
	Few	So So	Good	Better	Best	
Mediocre	9	0	0	0	0	9
Average	1	3	0	0	0	4
Good	0	0	1	3	0	4
Best	0	0	0	2	3	5
Total	10	3	1	5	3	22

Missing Data

When you use Proc FORMAT, take care when dealing with missing values. The keywords, HIGH, LOW, and OTHER, do not capture missing data. If the analytic requires precise tables that include missing values, include the missing values within the VALUE statement (. for numeric and ' ' for character). Additionally, make sure to use the MISSING option (if available) correctly within the Proc step. Below note that the FORMAT creates a format for age that includes missing data. The Proc FREQ includes the MISSING option on the TABLES statement. This option can only be used for numeric variables.

```
proc format;
      value ageb
            low - 24   = 'Toddlers'
            25 - 29    = 'Kids'
            30 - 34    = 'Juniors'
            35 - high  = 'Seniors'
                  .    = 'Missing';
run;
proc freq data = pap.missingage;
      tables age/missing;
      format age ageb.;
run;
```

for this output:

age	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Missing	***1***	***4.00***	***1***	***4.00***
Toddlers	3	12.00	4	16.00
Kids	6	24.00	10	40.00
Juniors	6	24.00	16	64.00
Seniors	9	39.00	25	100.00

The percentages in this table have changed by including the missing values.

Multiple Methods

SAS often provides multiple solutions to the same problem. For example, the team reports need to include the players' names. The datasets, YANKEES and BATTINGORDER, have player numbers but no names. IF/THEN/ELSE

statements can be written in a DATA step where each number is queried and the name is attached to the observation based on that number. This process is not only time consuming but would require rewrites of the DATA step every time a player is added (or released):

```
*First DATA step method;
data yankeenames;
       set pap.yankees;
       if number = 13 then name = 'Alex Rodriguez';
       else if number = 2 then name = 'Derek Jeter';
       *add more names here;
run;
proc print data = yankeesnames;
       var name avg;
run;
```

Assuming two separate datasets both had the key variable *number*, another way to accomplish the same task is to merge the datasets:

```
*Second DATA Step method;
proc sort data = pap.names out = names;
       by number;
run;
proc sort data = pap.battingorder out = battingorder;
       by number;
run;
data finalorder;
       merge battingorder (in = order) names;
       by number;
       if order = 1;
run;
proc print data = finalorder;
       var name avg;
run;
```

This process will create the correct reports, but sorting and merging large datasets can be inefficient.

You can also use Proc FORMAT to create a label for each player number. This will produce the same output and can be easily updated to create the reports as players come and go. No new dataset is created. Note that we are creating a numeric format for the variable *number*.

```
proc format;
       value names
              2      = 'Derek Jeter'
              13     = 'Alex Rodriguez'
              25     = 'Jason Giambi'
              /*add more here*/   ;
run;
proc print data = pap.battingorder;
       var number avg;
       format number names.;
run;
```

There's another way to solve this problem and that is by creating a format from the player names dataset.

Datasets for Formatting

Using the CNTLIN option within a Proc FORMAT allows you to create a user-defined format from values stored in a SAS dataset. This is a very efficient method to add value labels from a 'lookup' dataset, rather than merging two datasets, a master and a lookup. The syntax needed to create this format-based dataset is very specific. It requires the following variables:

- start this variable contains the codes or the values to be formatted
- label this variable contains the descriptions or labels for the values
- fmtname this variable is the actual name of the format

In the example below, the list of players' names and numbers is stored in a dataset called NAMES. This process creates a format from the NAMES dataset with the number of the player (*number*) becoming the **start** and the player's name (*name*) becoming the **label**. The new dataset FORMATNAMES contains these variables plus a new variable called *fmtname* that is equal to the value *playernames* for every observation. Note that the variable *fmtname* is always character and must be placed in quotes. This variable is initialized in the RETAIN statement and carried forward in the following observations. If a character format is being created, include the dollar sign within the quotes. Note also that the format name does not include the period!

Proc FORMAT then uses this dataset in the CNTLIN option:

```
*report 3;
data formatnames;
       retain fmtname "playernames";
       set pap.names;
       start = number;
       label = name;
       keep start label fmtname;
run;

proc format cntlin = formatnames;
run;
proc print data = pap.battingorder;
       var number avg;
       format number playernames.;
run;
```

This is what a portion of the dataset FORMATNAMES looks like

start	label	fmtname
26	Sal Fasano	playernames
20	Jorge Posada	playernames
22	Robinson Cano	playernames
25	Jason Giambi	playernames
17	Nick Green	playernames
2	Derek Jeter	playernames
13	Alex Rodriguez	playernames
53	Bobby Abreu	playernames

Creating New Variables

There may be times when you want to create a variable from a format. Though formats can avoid the necessity of creating new variables for performing grouped analyses, formats can also be used to create new variables within a DATA step. This can be accomplished by using the PUT function along with the user-defined format. Remember that all format values are character, whether or not the variable is numeric.

The PUT function can create a new variable based on the value of an already existing variable and a format. In the following example, a new variable, *playername*, is created using the format created in the previous example:

```
*report 4;
*Creating new fields from formats;
data addnames;
       set pap.battingorder;
       length name $15;
       name = put(number,playernames.);
run;
```

Permanent Formats

Just as with datasets, you have the option of saving your formats permanently. Formats are not stored in datasets but rather in catalogs. By defining the library where the catalog is located and using the LIBRARY= statement, the formats can be stored permanently:

```
libname ourlib 'j:/shared/yankees';
proc format library = ourlib;
       value run
              low - 10  = 'Mediocre'
              11 - 25   = 'Average'
              26 - 50   = 'Good'  ;
       value ab
              0 - 100    = 'Few'
              101 - 200  = 'So So'
              201 - 300  = 'Good'
              301 - 400  = 'Better'
              401 - high = 'Best'  ;
run;
```

These formats are now stored in a permanent catalog. If they are placed on a network drive, all analysts can access those catalogs by simply using the correct LIBNAME statement. Remember these formats are available for use in both DATA and PROC steps as long as they are defined before use.

Adding the option FMTSEARCH causes SAS to search format catalogs in the order listed, until the desired member is found. Here is a simple example using a permanent format. Note that the catalog is identified in the LIBNAME statement prior to use and the possible locations of the format to be used are listed in the FMTSEARCH option:

```
libname ourlib 'j:/shared/yankees';
options fmtsearch = (ourlib pap);
proc freq data = pap.battingavg;
       tables r * ab / nopercent norow nocol nocum;
       format r run. ab ab.;
       label r  = 'Runs'
             ab = 'At Bat';
run;
```

One More Advanced Example

When you are working with large datasets, efficiency is extremely important. You can use what you learned in this session to write more efficient SAS code.

In this example, the team has a file of 5,000 Yankees fans (FANS2006) that attended at least 75 games this past season. Each fan has a unique identifier attached to each file (*idnum*), and this dataset contains this variable. There is a large dataset (YANKEESFANS) that contains information on all 5 million+ fans from the past 10 seasons. This file contains the same unique identifier (*idnum*).

To prepare a report on these 5,000 fans, we could use a SAS DATA step MERGE, but the larger the number of observations the slower the process. Instead, we will use what we have learned to produce a format of those fan ids:

```
data fans;
       retain fmtname "myfans" label "in2006" ;
       set pap.fans2006;
       start = idnum;
       keep start label fmtname;
run;
proc format library = ourlib cntlin = fans;
run;

data report4;
       set pap.yankeesfans;
       where put(idnum,myfans.) = 'in2006';
run;
proc freq data = report4;
       tables zipcode gender;
       title 'Fan Report 4';
run;
```

This process subsets the data simply and efficiently. When you use this method, make sure you have a pretty good idea what to expect from the output.

A Few Additional Notes

If you receive a dataset where variables have been stored with user-defined formats but no format catalog, you can access the dataset by 'turning off' the format check:

```
options nofmterr;
```

If you receive a format catalog but no definitions for what is actually stored in the catalog, you can print out the contents by using the FMTLIB option of the procedure:

```
proc format fmtlib;
run;
```

When you are working with character data, it is very important to know everything about the data. If you are formatting and/or matching, using the function UPCASE or LOWCASE will make the manipulation one step easier. You can also used mixed case in formats:

```
proc format;
       value $address
              street, STREET, ST, st = 'Street';
run;
```

If you are formatting a character field that contains a single quote, make sure to enclose the entire value in double quotes:

```
TOB = "Thomas O'Brien"
```

Conclusion

Again, the FORMAT procedure can be extremely useful when dealing with data. Remember to cross check your values by first running a Proc FREQ or a similar procedure and then running the procedure again with formatted values.

Contact Information

For more information contact:

Marge Scerbo
National Study Center
701 W. Pratt, Room 554
Baltimore, MD 21201
Email: mscerbo@som.umaryland.edu