Paper DM-07

Swimming with Sharks: Using Formats with Summary Data

Tom Bugg, Wells Fargo Home Mortgage, Des Moines, IA

ABSTRACT

Creating summary tables for later use can be done efficiently using formats to bucket continuous variables, without the added time and disk space associated with an extra step of creating grouping, or “bucket” variables.

However, if we create datasets from these procedures for later use, we must use extreme caution. Even though there may be risk associated with creating and using summary data in this manner – it can be very worthwhile.

INTRODUCTION

Although handling summarized data for further processing can sometimes be compared with “swimming with sharks”, I’ve heard that some people get a kick out of it. A number of methods can be employed to avoid the pitfalls and take advantage of the advantages of using very small data sets for further processing. By becoming aware of the potential pitfalls, we can build an appropriate “shark cage” for protection.

GROUPING RECORDS BY “BUCKETS”

We often group loan or customer records for comparison purposes based on buckets of continuous variables such as score.

A number of methods are available for grouping, or creating “buckets”. Two possible ways include:

- We can Create a “Bucket” variable on the source data set, and then run all summary data queries based on this variable
- We can use the Format Procedure to create *virtual buckets*, and then create summary tables using the original continuous variable. Summary tables can be created using several different procedures, including (but not limited to):
    - The FREQ Procedure
    - The MEANS or SUMMARY Procedure
    - The TABULATE Procedure

For this paper, we’ll concentrate mostly on using PROC FREQ.

CREATING A BUCKET VARIABLE IN DETAIL DATA

A bucket variable can be created with simple if-then logic:

```
data swim_with_sharks_data;
     set swim_with_sharks_data;
     length score_bucket $15;
     if score = . then score_bucket = "Missing or Zero";
     else if score < 300 then score_bucket = "Missing or Zero";
     else if 300 le score < 620 then score_bucket = "LT 620";
     else if 620 le score < 660 then score_bucket = "620 - 659";
     else if 660 le score < 680 then score_bucket = "660 - 679";
     else if 680 le score < 700 then score_bucket = "680 - 699";
     else if 700 le score < 740 then score_bucket = "700 - 739";
     else if 740 le score < 780 then score_bucket = "740 - 779";
     else if score > 780 then score_bucket = "780+";
run;
```

Alternatively, the same bucket variable can be created with a format and a “put” statement:

```
 proc format;
     value score_b
        .       =      "Missing or Zero"
        low-0   =      "Missing or Zero"
        1-299   =      "Missing or Zero"
        300 - 619  =   "LT 620"
        620 - 659  =   "620 - 659"
        660 - 679  =   "660 - 679"
        680 - 699  =   '680 - 699'
        700 - 739  =   "700 - 739"
        740 - 779  =   "740 - 779"
        780-high   =   "780+";

data swim_with_sharks_data;
        set bugg.swim_with_sharks_data;
        score_bucket = put(score,score_b.);
run;
```

This second method can be especially useful if you have permanent formats set up to standardize reporting for your area

Both of these methods work. Why is using a format preferable when creating a bucket variable?

- Consistency – many times we have permanent formats that can be used, and we’ll be consistent within and across programs
- Simplicity – When typing out new code, the “put” statement is much shorter, and will lead to fewer coding errors/debugging time
- Speed – I ran the two methods repeatedly during heavy utilization times, and although the CPU time is fairly close between the two methods, using the format was consistently faster in total time

Example of Results from If-Then Method

```
NOTE: There were 500000 observations read from the data set SWIM_WITH_SHARKS_DATA.
NOTE: The data set SWIM_WITH_SHARKS_DATA has 500000 observations and 350 variables.
NOTE: Compressing data set SWIM_WITH_SHARKS_DATA decreased size by 49.91 percent.
     Compressed is 5112 pages; un-compressed would require 10205 pages.
NOTE: DATA statement used (Total process time):
     real time          59.00 seconds
     cpu time           23.74 seconds
```

Example of Results from Put(format) Method

```
NOTE: There were 500000 observations read from the data set SWIM_WITH_SHARKS_DATA.
NOTE: The data set SWIM_WITH_SHARKS_DATA  has 500000 observations and 350 variables.
NOTE: Compressing data set SWIM_WITH_SHARKS_DATA decreased size by 49.91 percent.
     Compressed is 5112 pages; un-compressed would require 10205 pages.
NOTE: DATA statement used (Total process time):
     real time          37.00 seconds
     cpu time           23.74 seconds
```

(note: when run on a weekend, there was no discernable difference in speed)

Once bucket variables are created, summary tables can be created using these new values with confidence that the resulting tables will be usable as is. A summary created with Proc Freq from either of these methods gives us the results below:

```
            Table of score_bucket by bus_group

score_bucket      bus_group
Frequency       |GROUPA  |GROUPB  |GROUPC  |GROUPD  |GROUPE  |  Total
620 - 659       |  26434 |  18238 |   4997 |    411 |       0 |  50080
660 - 679       |  16356 |  13660 |   3371 |    266 |       0 |  33653
680 - 699       |  17682 |  17732 |   4233 |    272 |       0 |  39919
700 - 739       |  36082 |  43227 |   9636 |    456 |       2 |  89403
740 - 779       |  45588 |  64350 |  12998 |    386 |       0 | 123322
780+            |  42293 |  73504 |  12060 |    205 |       1 | 128063
LT 620          |  11727 |  12523 |   3314 |     74 |       0 |  27638
Missing or Zero |   3148 |   4701 |     73 |      0 |       0 |   7922
Total             199310   247935    50682     2070        3   500000
```

USING A BUCKET VARIABLE

The following Proc Freq code creates the preceding output, and also saves a summary dataset for further use:

```
proc freq data  = swim_with_sharks_data;
    table score_bucket*bus_group/nocum norow nocol nopercent missing out=ifthen;
run;
```

Here's what we get if we use a simple proc print to see what the resulting data looks like (partial output):

```
                       bus_
score_bucket           group     COUNT    PERCENT

620 - 659              GROUPA    26434     5.2868
620 - 659              GROUPB    18238     3.6476
620 - 659              GROUPC     4997     0.9994
620 - 659              GROUPD      411     0.0822
660 - 679              GROUPA    16356     3.2712
660 - 679              GROUPB    13660     2.7320
660 - 679              GROUPC     3371     0.6742
660 - 679              GROUPD      266     0.0532
680 - 699              GROUPA    17682     3.5364
680 - 699              GROUPB    17732     3.5464
680 - 699              GROUPC     4233     0.8466
680 - 699              GROUPD      272     0.0544
700 - 739              GROUPA    36082     7.2164
700 - 739              GROUPB    43227     8.6454
700 - 739              GROUPC     9636     1.9272
700 - 739              GROUPD      456     0.0912
700 - 739              GROUPE        2     0.0004
```

Data in this summarized form can be useful in a number of tasks. We'll discuss these uses later, but for now let's talk about other ways of getting the same information.

BUCKETING WITHOUT CREATING A NEW VARIABLE – *VIRTUAL BUCKET*

What if we don't want to create a new variable? It's not always a good idea to create a new variable on a large "source" data set. Considerations include:

- Memory constraints
- Disk space constraints
- Time constraints

We can use formats directly with our summary code to give us the same results without the added step of creating a bucket variable. In essence, we're creating a *virtual bucket*:

```
proc freq data  = swim_with_sharks_data;
    table score*bus_line/nocum norow nocol nopercent missing out=fmt_method;
    format score score_b.;
run;
```

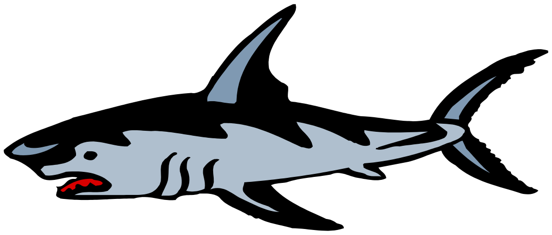
The output from the procedure (*other than the order*) looks identical to that of the earlier Proc Freq, without the need of creating a new variable:

```
                  Table of score by bus_group
score             bus_group

Frequency        |GROUPA  |GROUPB  |GROUPC  |GROUPD  |GROUPE  |  Total
Missing or Zero  |   3148 |   4701 |     73 |      0 |      0 |   7922
LT 620           |  11727 |  12523 |   3314 |     74 |      0 |  27638
620 - 659        |  26434 |  18238 |   4997 |    411 |      0 |  50080
660 - 679        |  16356 |  13660 |   3371 |    266 |      0 |  33653
680 - 699        |  17682 |  17732 |   4233 |    272 |      0 |  39919
700 - 739        |  36082 |  43227 |   9636 |    456 |      2 |  89403
740 - 779        |  45588 |  64350 |  12998 |    386 |      0 | 123322
780+             |  42293 |  73504 |  12060 |    205 |      1 | 128063
Total              199310   247935    50682     2070        3   500000
```

The resulting dataset appears to be identical as well (again, the order is different – *I wonder why?*), so it may *appear* that we are free to use the data in the same way.

```
                      bus_
score                 group     COUNT    PERCENT
Missing or Zero       GROUPA     3148     0.6296
Missing or Zero       GROUPB     4701     0.9402
Missing or Zero       GROUPC       73     0.0146
LT 620                GROUPA    11727     2.3454
LT 620                GROUPB    12523     2.5046
LT 620                GROUPC     3314     0.6628
LT 620                GROUPD       74     0.0148
620 - 639             GROUPA    26434     5.2868
620 - 639             GROUPB    18238     3.6476
620 - 639             GROUPC     4997     0.9994
620 - 639             GROUPD      411     0.0822
660 - 679             GROUPA    16356     3.2712
660 - 679             GROUPB    13660     2.7320
660 - 679             GROUPC     3371     0.6742
660 - 679             GROUPD      266     0.0532
680 - 699             GROUPA    17682     3.5364
680 - 699             GROUPB    17732     3.5464
680 - 699             GROUPC     4233     0.8466
```



BUCKETING WITHOUT CREATING A NEW VARIABLE – *USING OUTPUT DATA*

One thing I do frequently with summary data is to put it into the shape I'd like for output, export, or for use as a lookup table, etc. So let's do that with the table created with a "bucket" variable:

```
proc sort data = put;
  by score_bucket bus_group;

proc transpose data = put out=put_tr (drop=_name_ _label_);
  by score_bucket;
  var count;
  id bus_group;
run;
```

A printout of the resulting data set (put_tr) looks like this (what could be called a SAS pivot table):

```
  Transposed Data Based on "Bucket" Variable Summary

score_bucket        GROUPA    GROUPB    GROUPC    GROUPD    GROUPE

620 - 659           26,434    18,238     4,997       411         .
660 - 679           16,356    13,660     3,371       266         .
680 - 699           17,682    17,732     4,233       272         .
700 - 739           36,082    43,227     9,636       456         2
740 - 779           45,588    64,350    12,998       386         .
780+                42,293    73,504    12,060       205         1
LT 620              11,727    12,523     3,314        74         .
Missing or Zero      3,148     4,701        73         .         .
```

Now let's try to do the same thing with the table created using the Virtual Bucket method:

```
proc sort data = fmt_method;
   by score bus_group;

proc transpose data = fmt_method out= fmt_method_tr (drop=_name_ _label_);
   by score;
   var count;
   id bus_group;
run;
```

```
           Transposed Data Based on Put(Format) Summary
score                GROUPA     GROUPB     GROUPC     GROUPD     GROUPE

Missing or Zero       3,148      4,701         72          .          .
LT 620               11,661     12,442      3,364         78          .
=620 - 659           26,434     18,354      4,969        406          .
660 - 679            16,422     13,652      3,347        265          .
680 - 699            17,604     17,645      4,225        281          .
700 - 739            36,295     43,093      9,694        439          1
740 - 779            45,509     64,577     12,985        407          1
780+                 42,237     73,471     12,026        194          1
```

In this case, the data appears to come through intact, but is everything as it seems? Are there any hidden dangers?

Note: Depending on the data, you may get incomplete results and errors when attempting this step!!

Let's try to isolate buckets by looking at records in the "620-659" category:

Bucket Variable

```
data query_600_659_bucket;
   set ifthen;
   where score_bucket = '620 - 659';
Run;
```

```
600-659 Bucket from "Bucket" Variable Summary
score_       bus_
bucket       group      COUNT     PERCENT
620 - 659    GROUPA    26,434      5.2868
620 - 659    GROUPB    18,354      3.6708
620 - 659    GROUPC     4,969      0.9938
620 - 659    GROUPD       406      0.0812
```

Virtual Bucket

```
data query_600_659_bucket;
   set fmt_method;
   where score = '620 - 659';
Run;
```

(partial log output)

```
185  data query_600_659_bucket;
186      set fmt_method;
187      where score = '620 - 659';
ERROR: Where clause operator requires compatible variables.
188  run;
NOTE: The SAS System stopped processing this step because of
errors.
```

Aha! – Let's see why our variables look the same, but don't work that way. Running a Contents Procedure on the two summary datasets gives us the answer:

Partial Proc Contents output for IFTHEN summary table (Bucket Variable):

```
              Alphabetic List of Variables and Attributes
 #     Variable         Type     Len     Format      Label
 3     COUNT           Num       8     Frequency Count
 4     PERCENT         Num       8     Percent of Total Frequency
 2     bus_group       Char      6
 1     score_bucket    Char     15
```

Partial Proc Contents output for FMT_METHOD summary table (Virtual Bucket):

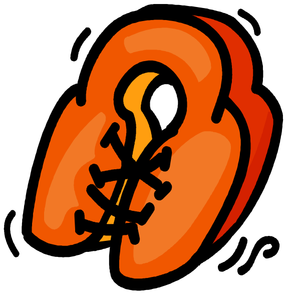
```
             Alphabetic List of Variables and Attributes
#     Variable         Type     Len     Format      Label
3     COUNT          Num       8                Frequency Count
4     PERCENT        Num       8                Percent of Total Frequency
2     bus_group      Char      6
1     score          Num       8     score_B.
```

The bucket variable was created as a character variable. The bucket created using the format method (virtual bucket) did not in fact create a new variable, but what we are seeing is the formatted version of a numeric variable.

BUCKETING WITHOUT CREATING A NEW VARIABLE – RESOLVING ISSUES

How do we make the Virtual Bucket version of the data usable for further data manipulation? Exactly the same way we would have created a bucket variable in the source data - using a put statement:

```
data fmt_method;
   set fmt_method;
   score_bucket = put(score,score_b.);
   drop score;
run;
```



We can now manipulate the data without fear of formatting issues:

```
proc sort data = fmt_method;
   by score_bucket bus_line;

proc transpose data = fmt_method out=fmt_method_tr (drop=_name_ _label_);
   by score_bucket;
   var count;
   id bus_line;
run;
```

```
                Transposed Data Based on Put(Format) Summary
                  Using "After-Summary" Bucket Description

score_bucket         GROUPA     GROUPB     GROUPC     GROUPD     GROUPE

620 - 659            26434      18354       4969       406          .
660 - 679            16422      13652       3347       265          .
680 - 699            17604      17645       4225       281          .
700 - 739            36295      43093       9694       439          1
740 - 779            45509      64577      12985       407          1
780+                 42237      73471      12026       194          1
LT 620               11661      12442       3364        78          .
Missing or Zero       3148       4701         72         .          .
```

BUCKETING WITHOUT CREATING A NEW VARIABLE – WHY???

OK – I know what you're thinking – I just took several steps just to accomplish the same thing I could have done by creating a bucket variable in the first place. Why the extra programming effort? As long as the results are the same, why the bother?

Remember the reasons we didn't want to create an extra variable?

- Memory constraints
- Disk space constraints
- Time constraints

MEMORY AND DISK SPACE CONSTRAINTS:

Adding a single 15-character variable increased the size of my dataset by about 4.9MB.

- The sample used for this paper was 500k records, with 349 variables (adding one to make 350)
- Imagine the impact on the original data, with 22.4 Million records - ≈213MB for adding a single 15-character variable.
- We summarize many different ways on many different continuous variables. *If we create additional variables for each of these, space could be quickly eaten up.*

TIME CONSTRAINTS: Let's look at the total time taken for the steps we've discussed:

	Using Bucket Variable	
stepname	**realtime**	**cputime**
FORMAT	00:01.0	00:00.0
DATA	00:52.0	00:27.3
FREQ	00:07.0	00:05.9
SORT	00:00.0	00:00.0
TRANSPOSE	00:00.0	00:00.0
PRINT	00:00.0	00:00.0
Total Time	**01:00.0**	**00:33.2**

	Using Virtual Bucket	
stepname	**realtime**	**cputime**
FORMAT	00:01.0	00:00.0
FREQ	00:08.0	00:05.8
DATA	00:00.0	00:00.1
SORT	00:00.0	00:00.1
TRANSPOSE	00:00.0	00:00.0
PRINT	00:00.0	00:00.0
Total Time	**00:09.0**	**00:06.0**

Even though we performed the same steps in both methods, we see a huge difference in time required.

- The data step creating a bucket variable was performed on the entire source dataset in the first example.
- The data step creating a bucket variable was performed only on the summary file in the second example.
- Especially in cases where we're building a repeatable or automated process, a little extra effort is more than worth it in terms of *time.*

EXAMPLE USING PROC SUMMARY/MEANS

Proc Freq works great for creating summaries involving counts. But if we want to sum dollar amounts, get average ratios, etc., we must use something else. Do we face the same issues? Let's look at an example using the Summary Procedure:

```
proc summary data = swim_with_sharks_data sum nway;
   class bus_group score;
   var amount;
   output out=means_summ sum=;
   format score score_b. amount dollar21.;
run;

title 'Proc Summary Output Data';
proc print data = means_summ noobs;
   var bus_group score amount;
run;
```

```
                               Proc Summary Output Data
          bus_
         group     fico                            amount

         GROUPA    Missing or Zero             $2,096,210
         GROUPA    LT 620                  $1,842,525,243
         GROUPA    620 - 659               $4,669,617,796
         GROUPA    660 - 679               $3,134,782,381
         GROUPA    680 - 699               $3,610,553,964
         GROUPA    700 - 739               $7,946,741,496
         GROUPA    740 - 779              $10,629,403,672
         GROUPA    780+                    $9,616,408,106
         GROUPB    Missing or Zero            $80,722,378
```

(partial output)

Do we have the same issue with the score field? We can tell quickly by re-printing the resulting data set using a standard format for the "score" variable:

```
proc print data = means_summ noobs;
   var bus_group score amount;
   format score 6.2;
run;
```

```
         Proc Summary Output Data
             Re-Format score                                  (partial output)
bus_
group      score                 amount
GROUPA     90.00             $2,096,210
GROUPA    300.00         $1,842,525,243
GROUPA    620.00         $4,669,617,796
GROUPA    660.00         $3,134,782,381
GROUPA    680.00         $3,610,553,964
GROUPA    700.00         $7,946,741,496
GROUPA    740.00        $10,629,403,672
GROUPA    780.00         $9,616,408,106
GROUPB     90.00            $80,722,378
GROUPB    300.00         $1,902,718,248
GROUPB    620.00         $3,247,428,975
GROUPB    660.00         $2,787,033,011
GROUPB    680.00         $3,858,068,485
GROUPB    700.00        $10,335,150,352
```

We can see that the SCORE variable is indeed numeric, and will indeed need to be treated with care.

CONCLUSION

Creating summary tables for later use can be done efficiently using formats to bucket continuous variables.

Once the summary table has been created, *great care must be taken* when handling the resulting values for further manipulation. When the data is understood, the issues are easily overcome.

Especially when building processes that will be repeated and automated, hours of processing time and vast amounts of memory and disk space can be saved by using this method.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Tom Bugg
Enterprise: Wells Fargo Home Mortgage
Address: MAC 2401-06A
1 Home Campus
City, State ZIP: Des Moines, IA 50328
Work Phone: (515)213-4309
Fax:
E-mail: thomas.b.bugg@wellsfargo.com