Paper B03-2009

Handy Procedures to Expand Your Analytics Skill Set

Mary MacDougall, National City, now part of PNC, Cleveland, OH

ABSTRACT

A good handyman can accomplish a lot with a general-purpose tool like a hammer or screw driver, but for some projects, it's critical to have a special-purpose tool like a pipe wrench in your tool box. PROC REPORT is great for everyday reporting tasks, but a senior analyst should also be familiar with procedures that are effective for specific data preparation and analysis tasks. This paper gives an introduction to my favorite Base SAS and SAS/Stat procedures: UNIVARIATE, RANK, TRANSPOSE, FORMAT, and SURVEYSELECT, with examples from the field of direct marketing. So the next time you need to Decile, Flatten, Bin, Sample, or Check for Outliers, you will know which Proc to start with.

INTRODUCTION

Sometimes changes in data sources or business requirements make it necessary to cobble together a temporary solution. And when a new, urgent requirement comes up in the middle of a project, it is tempting to dive in and start developing a custom program using a DATA step, a PROC SQL query or macro language. However, if you take a minute to search the documentation, you often find that the good people at SAS have anticipated your needs and built a function or procedure that can be adapted to solve the problem.

In fact, every general analyst should be familiar with some special-purpose procedures. You may not use these analytic tools every day, but if you know their names and basic functionality, you will be glad you have them in your toolbox when the need arises.

CASE STUDY

In the following scenarios, you are an analyst in the marketing department of a retailer that has recently acquired a wholesale business. Some preliminary reports summarizing the acquired company's historical sales data were distributed to senior management yesterday. And today, your inbox is full of e-mails:

1. **Sales manager**: The average sales per customer metric seems too high. Why?
2. **Campaign manager:** For the acquired company's last campaign, what was the response rate for the contact group compared to the holdout group?
3. **Marketing Research manager:** Pull a random sample of customers for a focus group…
4. **Call Center manager:** We want to make welcome calls to our best customers. Pull the top two deciles by dollar sales . . .
5. **Regional manager:** Run a report for my region showing unit sales metrics by percentile.

Before you open a new SAS session and start writing DATA step code to answer these business questions, let me introduce you to a few of my favorite procedures.

PROC UNIVARIATE: CHECK FOR OUTLIERS

Sales manager: *The average sales per customer seems too high. Why?*

When a business partner asks for a specific metric, it is important to understand the business question that prompted the request. For example, when someone asks for an average, they may not want the statistical mean of a specific sample. They may really be asking, "What is a typical value in this case?" If some records contain extreme values, these outliers may cause the mean to be much higher or lower than the median. In that case, it might be more helpful to report the median value. Or you could choose to exclude outliers, or cap the extreme values so they have less influence on the mean. Another approach would be to include the outliers in the analysis, but make sure your audience is informed about the extreme value's influence on the mean.

Before you can advise which approach to use, you need to understand how the values in your data are distributed. This would be a good time to pull the UNIVARIATE procedure out of your SAS/Stat toolbox.

Why is the average sales per customer metric higher than expected? To answer this question, you need to understand the distribution of sales data. PROC UNIVARIATE can tell you if the distribution is symmetric, with an equal number of observations having values above and below the mean. Or, as is common when dealing with subject areas like transaction counts, sales volumes, or other behavioral data, your values may be skewed to the right with long tails.

If you want to check for extreme values, PROC UNIVARIATE provides a quick snapshot of the highest and lowest values. PROC UNIVARIATE produces many tables and statistics, but for this example we will start by using Output Delivery System (ODS) to select only the ExtremeObs table of the UNIVARIATE output. A full listing of UNIVARIATE Procedure ODS table names is available in the Base SAS Procedures Guide.

The variable we are interested in is Revenue from the CustSales data set. We use the NEXTROBS option to request the top 10 and bottom 10 observations. In addition to the value of Revenue and the observation number for each record, we want to see the fields CustID, Segment, and SalesRep.

```
ods select ExtremeObs;
proc univariate data = CustSales nextrobs = 10;
  var Revenue;
  id CustID Segment SalesRep;
run;
```

The output lists the customer records with the lowest and highest revenue.

The UNIVARIATE Procedure
Variable: Revenue

Extreme Observations

Lowest

			Sales	
Value	CustID	Segment	Rep	Obs
0.0001	8632	RESIDENTIAL	PATEL	367600
0.0001	4434	RESIDENTIAL	ANDREW	367593
0.0001	9182	RESIDENTIAL	PATEL	367556
0.0001	8672	RESIDENTIAL	PATEL	367469
0.0001	5385	RESIDENTIAL	PATEL	367452
0.0001	6160	RESIDENTIAL	PATEL	367383
0.0001	5557	RESIDENTIAL	PATEL	367282
0.0001	6172	RESIDENTIAL	PATEL	367177
0.0001	5707	RESIDENTIAL	PATEL	367155
0.0001	2840	RESIDENTIAL	ANDREW	367121

Extreme Observations

Highest

			Sales	
Value	CustID	Segment	Rep	Obs
743777	6086	INDUSTRIAL	DONALD	26971
749540	1016	INDUSTRIAL	DONALD	9784
759564	2815	INDUSTRIAL	DONALD	48096
763003	0994	INDUSTRIAL	DONALD	117003
863751	6192	INDUSTRIAL	DONALD	33274
901870	9011	INDUSTRIAL	DONALD	10175
1207751	1004	INDUSTRIAL	DONALD	132401
1262649	4696	INDUSTRIAL	DONALD	57407
1558638	5705	INDUSTRIAL	DONALD	40819
1940855	0443	INDUSTRIAL	DONALD	9726

Figure 1. UNIVARIATE procedure: ExtremeObs output

A look at the extreme values suggests your data may have some outliers skewing the distribution to the right. You also notice that customers with the highest sales amounts are in the “Industrial” segment, so you investigate further by sorting the data and rerunning the procedure with a BY statement, requesting only the mean and median statistics.

```
proc sort data=CustSales;
  by Segment;
run;
```

```
proc univariate data = CustSales noprint;
  by Segment;
  var Revenue;
  output out= Stats n=Customers mean=MeanRevenue median=MedianRevenue;
run;
```

The output data set shows that the Industrial segment has much higher average and median revenue than the Residential segment.

```
Stats
                              Mean       Median
  Segment      Customers     Revenue    Revenue

INDUSTRIAL          7641    10149.38    1406.95
RESIDENTIAL       360093      358.53      47.07
```

Figure 2. UNIVARIATE procedure: Selected Statistics with BY Group output

A picture is worth a thousand words, so why not look at a graphical comparison of the distributions? With a simple change to your ODS SELECT statement and the addition of the PLOT option, you can request side-by-side box plots for your BY groups:

```
ods select SSplots;
proc univariate data = CustSales plot;
  by Segment;
  var Revenue;
run;
```

The procedure creates this line printer graphic.

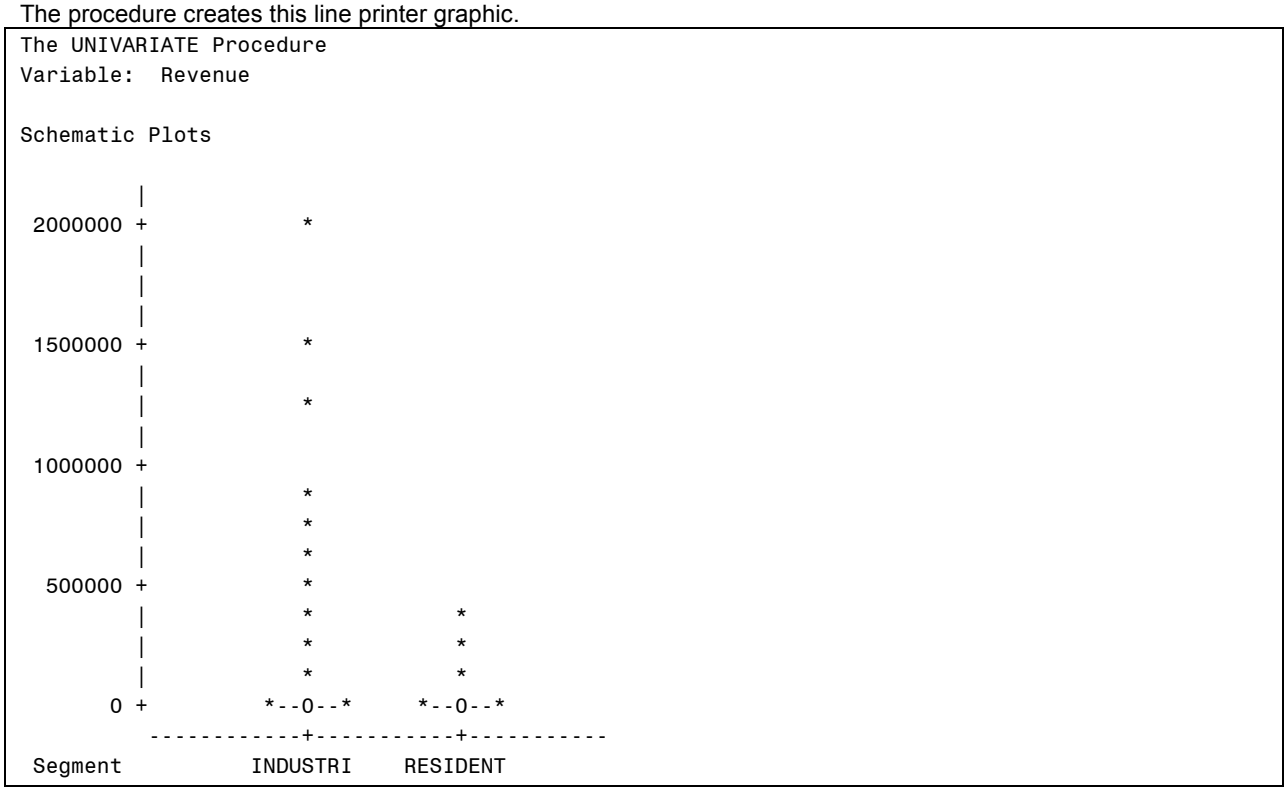

Figure 3. UNIVARIATE procedure: SSplot output

After seeing the side by side plots, the Sales manager agrees that, because of the recent acquisition of several large Industrial customers, the average of the entire customer population is no longer a useful metric, and sales averages for the Residential and Industrial customer segments should now be reported separately.

PROC TRANSPOSE: FLATTEN IT

Campaign manager: *For the acquired company's last campaign, what was the response rate for the contact group compared to the holdout group?*

It is often necessary to prepare the source data to suit the analytic method you have chosen. To flatten a detail data set, you can use PROC SUMMARY or PROC SQL with a GROUP BY clause to aggregate numeric values. But what if you do not want values to be aggregated; you just want to transpose records and columns, turning a data set with many records and a few fields into a set with fewer records and many fields? You could write complicated DATA step logic to create new fields, but why not use PROC TRANSPOSE to produce the flattened data set structure you need?

In this example, we are analyzing the results of a direct mail campaign. The response data is summarized by Propensity, Month and Group. Propensity is the customer's relative likelihood to respond, according to a propensity-to-buy model. Group identifies whether the customer received direct mail contacts was assigned to the group that was held out as a control. Month is number of months since the customer was first contacted.

Response

Propensity	Month	Group	CustCount	CumResp	CummRespPct
HIGH	1	CONTACT	707	11	0.01556
HIGH	1	HOLDOUT	4038	23	0.00570
HIGH	2	CONTACT	707	17	0.02405
HIGH	2	HOLDOUT	4038	53	0.01313
HIGH	3	CONTACT	707	24	0.03395
HIGH	3	HOLDOUT	4038	85	0.02105
MED	1	CONTACT	1639	53	0.03234
MED	1	HOLDOUT	11486	143	0.01245
MED	2	CONTACT	1639	75	0.04576
MED	2	HOLDOUT	11486	357	0.03108
MED	3	CONTACT	1639	113	0.06894
MED	3	HOLDOUT	11486	651	0.05668

Figure 4. Response data before TRANSPOSE procedure

The Response data set contains the data we need, but to complete the analysis, we want to flatten the set see the Contact and Holdout groups' response metrics, side by side for each month.

In this example, we transpose CummRespPct, which contains the response metric. To group the transposed data by Propensity and Month, we put those fields in the BY statement. Note that the source data set was already in the correct order so no additional sorting was needed. The ID statement specifies that Group contains the values to use to name the new fields when CummRespPct is transposed. The TRANSPOSE procedure automatically creates the _NAME_ field to show which field was transposed. In this simple example, we do not need that field, so we use a data set option to drop it from the output data set.

```
proc transpose data = response out = bymonth (drop=_name_);
  by Propensity Month;
  id Group;
  var CummRespPct ;
run;
```

ByMonth

Propensity	Month	CONTACT	HOLDOUT
HIGH	1	0.01556	0.00570
HIGH	2	0.02405	0.01313
HIGH	3	0.03395	0.02105
MED	1	0.03234	0.01245
MED	2	0.04576	0.03108
MED	3	0.06894	0.05668

Figure 5. Response data after TRANSPOSE procedure

The output dataset now has two new fields called Contact and Holdout that contain the cumulative response rates by Propensity and Month.

PROC SURVEYSELECT: SAMPLE IT

Marketing Research manager: *Pull 12 online customers for a focus group.*

If you need to select a random sample from a data set, you could build a DATA step solution using the RANUNI random-number function. But there is no need to reinvent the statistical wheel. If you have SAS/Stat, the SURVEYSELECT procedure provides an easy way to do simple random sampling and more complex sampling tasks, like stratified sampling or weighted sampling.

In this example, the SalesHist data set contains a list of customers and data about their purchasing behavior. We use a WHERE clause to select only those customers who have an account through the company's online store. The option METHOD=SRS requests a simple random sample. The request is for 12 names, so we set N=12.

```
proc surveyselect data=sales_hist (where=(online_cust = 1)) method=srs  n=12
       out = Focus_list_simple;
run;
```

The output data set contains a list of twelve customers, selected randomly from the set of online customers.

Focus_list_simple

Obs	Cust ID	Segment	Total Revenue	Website Visits
1	2440	RESIDENTIAL	9.06	5
2	0176	RESIDENTIAL	15.37	3
3	2535	RESIDENTIAL	2.49	6
4	9364	RESIDENTIAL	1070.01	0
5	5393	RESIDENTIAL	479.13	4
6	4901	RESIDENTIAL	44.35	8
7	7470	RESIDENTIAL	0.07	16
8	1685	RESIDENTIAL	2.10	2
9	8330	RESIDENTIAL	98.10	11
10	2480	RESIDENTIAL	0.45	0
11	6390	RESIDENTIAL	113.93	5
12	4977	RESIDENTIAL	26.60	32

Figure 6. SURVEYSELECT with METHOD=SRS output

When you present the list, the Research manager thanks you for your quick response and asks if it would be possible to, "Weight the sample so frequent website users are more likely to be selected." With PROC SURVEYSELECT, you can answer this new request with only a few code changes.

The option METHOD=PPS_SYS requests that units, in this case customers, are selected by systematic random sampling with probability proportional to size, and the SIZE statement specifies that the probability will be weighted by the value of WebsiteVisits. So, as requested, frequent website users will be more likely to be included in the focus group.

```
proc surveyselect data=sales_hist (where=(online_cust = 1)) method=pps_sys  n=12
       out = Focus_list_weighted;
       size WebsiteVisits;
run;
```

The result is a new sample of twelve customers. Since we used a sampling method with replacement, the output includes ExpectedHits, which indicates the expected number of selections. SamplingWeight is the inverse of ExpectedHits. The example output shows that customers with more website visits had a higher number of expected hits.

Focus_list_weighted

Obs	Cust ID	Website Visits	Number Hits	Expected Hits	Sampling Weight
1	8126	192	1	.000064847	15420.84
2	3374	321	1	.000108417	9223.68
3	9565	504	1	.000170224	5874.61
4	7441	410	1	.000138476	7221.47
5	5075	223	1	.000075317	13277.14
6	2695	190	1	.000064172	15583.17
7	2554	304	1	.000102675	9739.48
8	8012	430	1	.000145231	6885.59
9	1371	71	1	.000023980	41701.43
10	4901	56	1	.000018914	52871.46
11	5667	68	1	.000022967	43541.20
12	1226	21	1	.000007093	140990.56

Figure 7. SURVEYSELECT with METHOD=PPS_SYS output

In addition to simple random sampling and sampling with proportional probability, PROC SURVEYSELECT offers many other methods, including METHOD=SYS for stratified random samples.

PROC RANK: DECILE IT

Call Center manager: *We want to make welcome calls to our best customers. Pull the top two deciles by dollar sales . . .*

To answer this request, you could build a custom solution that involves sorting and counting records in a DATA step. But instead, when you need percentiles, think of using "PROC RANK with Groups = 100". For deciles, think "PROC RANK with Groups = 10"; for quartiles, think "PROC RANK with Groups = 4", and so on.

The code to answer the campaign manager's request for a list of customers in the top two deciles (20%) by sales revenue would look like this:

```
proc rank data = CustSales groups = 10 out = deciles;
  var Revenue;
  ranks RevDecile;
run;
```

PROC RANK will output a data set called Deciles that will list all customers. The customers with the lowest Revenue value will have RevDecile set to 0 and those with highest Revenue will have RevDecile = 9. If you prefer to have the values ranked from largest to smallest, add the DESCENDING option to the RANK statement.

To select only the top two deciles, you could add a where clause to filter which records will be loaded to the output data set.

```
proc rank data = CustSales groups = 10 out = deciles (where = (RevDecile >= 8));
  var Revenue;
  ranks RevDecile;
run;
```

Deciles

Segment	Revenue	Period	Prod Line	Cust ID	Sales Rep	Rev Decile
INDUSTRIAL	12048.01	YTD200908	CUSTOM	8180	DONALD	8
INDUSTRIAL	29567.69	YTD200908	CUSTOM	4847	DONALD	9
INDUSTRIAL	12826.83	YTD200908	CUSTOM	3604	DONALD	8
INDUSTRIAL	74870.10	YTD200908	CUSTOM	9431	DONALD	9
INDUSTRIAL	570396.18	YTD200908	CUSTOM	7842	DONALD	9
INDUSTRIAL	12791.00	YTD200908	CUSTOM	6525	DONALD	8
INDUSTRIAL	13572.88	YTD200908	CUSTOM	8702	DONALD	8
{	{	{	{	{	{	{

Figure 8. RANK procedure: Top 2 deciles output

Regional manager: *Run a report for my region showing total, average, minimum and maximum unit sales, for each percentile grouping.*

You can also use PROC RANK with PROC SUMMARY to produce the summary report requested by the Regional manager. This time we need percentile ranks based on Unit sales, so we set GROUPS = 100, and the variable to be ranked is Units.

```
proc rank data = CustSales groups = 100 out = Percentiles;
  where Region = 'NORTH';
  var Units;
  ranks UnitSalesPctl;
run;
```

The result of this procedure is a list of customers with the RANKS variable UnitSalesPctl having values from 0 to 99. This RANKS variable will be the CLASS variable in the PROC SUMMARY step. In the OUTPUT statement, we ask for the statistics Sum, Mean, Min and Max.

```
proc summary data=percentiles nway;
  where Region = 'NORTH';
  class UnitSalesPctl;
  var units;
  output out=Percentile_Summary (drop=_type_ rename=(_freq_=Customers))
         sum=Units mean=Average min=Minimum max=Maximum ;
run;
```

Percentile_Summary

Unit Sales Pctl	Customers	Units	Average	Minimum	Maximum
0	1325	16,545	12	1	22
1	1325	42,713	32	22	43
2	1326	73,760	56	43	69
3	1325	113,351	86	70	100
4	1326	154,835	117	100	135
{	{	{	{	{	{
95	1326	453,584,591	342,070	318,814	368,951
96	1325	531,482,477	401,119	368,954	438,036
97	1326	642,524,233	484,558	438,095	534,203
98	1325	838,202,232	632,605	534,532	759,970
99	1325	1,592,194,657	1,201,656	760,355	12,521,934

Figure 9. SUMMARY procedure using percentile values as class variable

The resulting data set contains the requested metrics for customers by percentile grouping.

PROC FORMAT: BIN IT

Soon after presenting the Percentile Summary report, you receive a follow-up request from the Regional manager:

"Provide a summary report showing the same metrics, but instead of percentiles, group customers according to our new tier scheme ..."

To answer this request for custom groupings, you could write a DATA step with IF-THEN-ELSE statements to create a new field, and then use that as the CLASS variable in your PROC SUMMARY. But you can save some time and typing if you use a custom SAS format to group the data into categories or bins while summarizing the metrics.

The FORMAT procedure is commonly used to create custom SAS formats that improve the appearance of output, and custom formats are also an efficient way to bin values. By using a FORMAT statement to associate a variable with a custom format, you can group data in different ways without having to create a new data set.

This example uses the Percentiles data set created by PROC RANK in the preceding example. PROC FORMAT creates a custom format called *Tiers*. that groups percentile values from 0 to 99 into the customer tiers requested by the Regional manager.

```
proc format;
  value tiers
    0 -   1 = '1. Platinum'
    2 -   9 = '2. Gold    '
   10 -  24 = '3. Silver  '
   25 -  49 = '4. Bronze  '
   50 -  99 = '5. Green   ' ;
run;

proc summary data=percentiles nway order=formatted;
  class  UnitSalesPctl;
  var units;
  output out=TierSummary (drop=_type_ rename=(_freq_=Customers))
         sum=Units mean=Average min=Minimum max=Maximum ;
  format UnitSalesPctl tiers.;
run;
```

With only a few changes to the PROC SUMMARY code from the previous example, we have a data set with metrics summarized by tier. UnitSalesPctl is still the CLASS variable. The main change is that the FORMAT statement now specifies that UnitSalesPctl will be formatted with the *Tiers*. format. We also added an ORDER=FORMATTED option to the SUMMARY statement so the output will be sorted according to their formatted values of the CLASS variable.

Tier_Summary

Unit Sales Pctl	Customers	Units	Average	Minimum	Maximum
1. Platinum	2650	59,257	22	1	43
2. Gold	10608	2,053,511	194	43	400
3. Silver	19876	24,124,049	1,214	400	2,437
4. Bronze	33136	232,711,321	7,023	2,438	14,432
5. Green	66270	9,049,256,924	136,551	14,433	12,521,934

Figure 10. SUMMARY procedure using custom-formatted class variable

Using a custom format has the advantage of being a modular approach that is easily documented and maintained. If your company adopts this tier scheme as a standard way to categorize customers, the PROC FORMAT step could be moved into a separate program that creates a permanent format in your company's custom format library. Then if the tier groupings are ever modified, for example to add a "Diamond" tier, the format can be updated without having to modify the report code.

CONCLUSION

Someday computers will be able to provide answers to our most complex business questions at the click of a mouse. Until then, it will be the analyst's job to interpret the question, decide on an analytic approach and translate that approach into steps a computer can process. While there are many ways to write custom solutions using SAS, an experienced analyst should be familiar with special-purpose procedures that can reduce the need for custom coding, testing and documentation. This paper introduced five procedures that are handy when you need to Flatten, Sample, Decile, Bin, or Check for Outliers. Explore the online documentation and discussion forums to find more examples, and continue to expand your own analytics skill set.

REFERENCES

Derby, Nathaniel. 2009. "A Little Stats Won't Hurt You." Proceedings of SAS Global Forum 2009. Cary, NC: SAS Institute Inc.

SAS Institute. 2008. "Base SAS 9.1.3 Procedures Guide." SAS OnlineDoc® 9.1.3, Cary, NC: SAS Institute Inc. http://support.sas.com/onlinedoc/913/.

Svolba, Gerhard. 2006. *Data Preparation for Analytics Using SAS®*. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

The author would like to thank her colleague Mary Kay Curtis, for introducing her to the technique of using custom formats to bin values.

CONTACT INFORMATION

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

Mary MacDougall
National City, now part of PNC
32000 Mill Creek Blvd.
Highland Hills, OH 44122
216.488.7608
mary.macdougall@gmail.com