Using PROC RANK and PROC UNIVARIATE to Rank or Decile Variables
Jonas V. Bilenas, JP Morgan Chase Bank, Wilmington, DE

ABSTRACT
In direct marketing applications prospects are often ranked by scores that predict response, risk and/or profitability. Justification is to evaluate lifts in performance by soliciting the most profitable individuals as defined by score ranking. Many programmers write complicated code using PROC SORT, MACRO calls, and DATA STEPS to rank or decile these scores. These programs often don’t handle score ties which can result when 2 or more prospects with the same score end up in 2 or more ranks. We will show how using PROC RANK will provide a quick and simple way to rank or decile individuals that will handle ties with the PROC RANK TIES option. We will look at the syntax of PROC RANK with a number of examples and applications. Since PROC RANK does not have a WEIGHT statement to handle weighted samples we will show how to handle weights using PROC UNIVARIATE and PROC FORMAT to perform a weighted ranking.

INTRODUCTION
How many times are you asked to group one or more numeric variables into quartiles, deciles, or other number of groups? Are you ever asked to generate a report where you are asked to cross 2 numeric variables after binning into groups? PROC RANK can help you easily create these groupings with a few lines of code. In this paper we will review the syntax of PROC RANK, indicate why you would want to use the procedure, show a number of examples in using PROC RANK, and provide an alternative to PROC RANK when dealing with sampled data.

WHAT IS PROC RANK?
PROC RANK computes the RANKS from one or more numeric variables across observations in a SAS® data set and creates a new data set that captures these rankings. PROC RANK does not produce any printed output but has many options to specify the order of ranks, handling ties in variable values, and can generate variable bins or groupings based on the specification of the GROUPS option. Some of the applications of PROC RANK are listed here:

- Nonparametric Statistics. Many nonparametric procedures are simply the parametric procedures using ranks of variables as opposed to using the raw, unranked variable values.
- Generating deciles, quartiles, percentiles or other groups from numeric variables. The GROUPS option is used here to specify the binning. Deciles are created by specifying GROUPS=10, quartiles can be generated by GROUPS=4, and percentiles are created with setting GROUPS=100.
- Exploratory scatter plots when the Y-axis is binary. When both X and Y variables are continuous a scatter plot with a smoothed fit (i.e.; PROC LOESS) can be informative. When the Y variable is binary (taking on values of 0 and 1) the graph is difficult to interpret. Generating bins of the X variable and plotting means of the Y variable (or log of odds of Y) as a function of means of binned X variable can be more informative.

SIMPLE EXAMPLE USING PROC RANK
Let us create some simple data to see how PROC RANK works. The following code will generate a data set with one variable (SPEND):

```sas
data test;
  input spend @@;
datalines;
  122345 235 12 5677 214 432 121 1567;
run;
```
We can generate some simple PROC RANK code to get a feel for the data output from the procedure. Note that the OUT= option in line 1 specifies the name of the output data set from PROC RANK. If this option is not included, the RANK procedure will generate an incremental data set with a prefix of DATA (example; DATA2).

```latex
proc rank data=test out=r_test;
  var spend;
run;

proc print data=r_test;
run;
```

Recall that PROC RANK does not produce any printed output so we included the PROC PRINT to see the resulting data created by PROC RANK (R_TEST). Output from the code is shown in Figure 1. Notice that the SPEND variable now contains the ranks for non-missing values of the original variable SPEND from the TEST data. If you wish to retain the original values of the SPEND variable you can modify the code to include a RANKS statement. RANKS, in line 1 below, names the variable you wish to store the value of ranks in. In this code we name a variable called R_SPEND which captures the RANKS of the SPEND variable specified in the VAR statement. Results from PROC PRINT are shown in Figure 2.

```latex
proc rank data=test out=r_test;
  var spend;
  ranks r_spend;
run;
```

By default, RANKS are in ascending order. Sometimes you may want to reverse the order. This can be done with the DESCENDING option in PROC RANK. Using the TEST data, we modify the PROC RANK code and show the results of the PROC PRINT in Figure 3.

```latex
proc rank data=test out=r_test descending;
  var spend;
  ranks r_spend;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>spend</th>
<th>r_spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>122345</td>
<td>8</td>
</tr>
<tr>
<td>2</td>
<td>235</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>5677</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>214</td>
<td>3</td>
</tr>
<tr>
<td>6</td>
<td>432</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>121</td>
<td>2</td>
</tr>
<tr>
<td>8</td>
<td>1567</td>
<td>6</td>
</tr>
</tbody>
</table>

By default, RANKS are in ascending order. Sometimes you may want to reverse the order. This can be done with the DESCENDING option in PROC RANK. Using the TEST data, we modify the PROC RANK code and show the results of the PROC PRINT in Figure 3.

<table>
<thead>
<tr>
<th>Obs</th>
<th>spend</th>
<th>r_spend</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>122345</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>235</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>12</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>5677</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>214</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>432</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>121</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1567</td>
<td>3</td>
</tr>
</tbody>
</table>
WHAT ABOUT TIES?

If there are tied values in the variable that is being ranked, the RANK procedure has a number of options to handle ties. The default option is MEANS where the mean rank is returned for tied values. Let's modify the TEST data to include some ties and look at the resulting output. Code is shown here with output in Figure 4.

```sas
data test;
  input spend @@;
datalines;
122345 235 12 5677 214 432 121 12 1567 235 ;
run;

proc rank data=test out=r_test descending;
  var spend;
  ranks r_spend;
run;

proc print data=r_test;
run;
```

Other options for ties include TIES=LOW, TIES=HIGH, and TIES=DENSE (added in SAS 9.2). With TIES=LOW, the tied values are assigned to the lower rank. With TIES=HIGH, the tied values are assigned to the higher rank. For the DENSE option, the ranks are consecutive integers that begin with 1 end with the number of unique values of the VAR variable. Code is shown for TIES=LOW. Output for LOW is shown in figure 5. Results with HIGH is shown if figure 6 and DENSE is illustrated in figure 7.

```sas
proc rank data=test out=r_test descending ties=low;
  var spend;
  ranks r_spend;
run;
```

<table>
<thead>
<tr>
<th>Fig 5: TIES=LOW</th>
<th>Fig 6: TIES=HIGH</th>
<th>Fig 7: TIES=DENSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>spend</td>
<td>r_spend</td>
<td>spend</td>
</tr>
<tr>
<td>122345</td>
<td>1</td>
<td>122345</td>
</tr>
<tr>
<td>235</td>
<td>5</td>
<td>235</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>5677</td>
<td>2</td>
<td>5677</td>
</tr>
<tr>
<td>214</td>
<td>7</td>
<td>214</td>
</tr>
<tr>
<td>432</td>
<td>4</td>
<td>432</td>
</tr>
<tr>
<td>121</td>
<td>8</td>
<td>121</td>
</tr>
<tr>
<td>12</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>1567</td>
<td>3</td>
<td>1567</td>
</tr>
<tr>
<td>235</td>
<td>5</td>
<td>235</td>
</tr>
</tbody>
</table>
GENERATING DECILES OR OTHER GROUPINGS WITH PROC RANK

Let us look at the task of generating deciles of numeric variables using PROC RANK. Let us begin by creating a simulated data set to work with. Code:

```sas
data test2;
  do i = 1 to 10000;
    score = round(650+rannor(243)*20,1);
    event = (score+rannor(12)*25>700);
    output;
  end;
run;
```

For the data (TEST2) generated from the above code we have simulated 10,000 records. SCORE can be thought of as an integer credit score predicting, for this example, response to a direct mail offer for credit. The variable EVENT is the actual binary event; for example, response to the offer (1=yes, 0=no).

We wish to look at the performance of the SCORE in predicting the EVENT. We can decile the score from high score to low and look at the performance of the mean EVENT at each decile. To handle this task, we can use the RANK procedure and use the GROUPS= option. The GROUPS= options assigns group values for the non-missing values of the variable specified in the VAR statement. If a RANKS statement is specified, the group values are stored in the variable named in the RANKS statement. Group values range from 0 to the number specified minus 1. For example, if we wish deciles we can specify GROUPS=10 and the resulting group values range from 0 to 9.

PROC RANK will never have tied values of the VAR variable in more than one group. Group assignment for tied values are determined by the TIES= option. As a result, we may end up with fewer groups than specified by the GROUPS option. If ties occur at a group split rule, the number of observations in each group may not be equal across groups. These are all good safety features in PROC RANK. You don’t want tied values ending up in more than one group.

The following code will run the decile splits for SCORE in descending order and generate a report on results by decile. Output of the report is shown in figure 8.

```sas
proc rank data=test2          out=ranky    ties=low
    descending           groups=10;
    var score;
    ranks r_score;
run;

proc means data=ranky noprint;
    class r_score;
    var score event;
    output out=report
        n=n
        min(score)=score_min max(score)=score_max
        mean(event)=event_mean;
run;
```

Applications Big & Small
NESUG 2009
Some comments about the code and output:

- In Line 1 of the code we added the GROUPS=10 option.
- In line 2 we did not specify the PROC MEANS option NWAY. As a result we have an overall result in the output from MEAN with a R_SCORE of missing.
- Note that the overall mean of response (EVENT) is 5.86%. The top scoring decile is at 28.74%
- In PROC MEANS we asked for the minimum and maximum scores to be captured in the REPORT data (line 3). This can be useful for setting score cut-offs. Also note in the report that there are no ties in scores.

### SCATTERPLOTS GENERATED FROM DECILED DATA

Output from PROC RANK when using GROUPS can be useful in scatter plots when the Y variable is binary. For example, if we look at code that plots the EVENT as a function of SCORE we notice that it is hard to tell from the graph if the X variable (SCORE) rank orders the variable we are trying to predict (EVENT: response) or the functional form of event as a function of the X variable (SCORE). Code and output:

```plaintext
filename gout ".";
goptions reset=all
device=cgmOF97L
gsfname=gout display
;
axis1 label=(angle=90 "Event");
symbol1 i=none v=plus;

proc gplot data=test2;
  plot event*score
    /vaxis=axis1
grid;
run;quit;
```

**Figure 9. Output from GPLOT code**
PROC RANK can assist us to get a more informative plot of the relationship between EVENT and SCORE. For each score decile we can plot the mean of the event (mean response) or typically, for binary data, we can plot the log of odds of the event if we will be using logistic regression to specify the functional relationship. Code is shown below and GPLOT output is shown in figure 10.

```sas
proc means data=ranky noprint nway;
   class r_score;
   var score event;
   output out=plotit mean=;
run;

data plotit2;
   set plotit;
   odds=event/(1-event);
run;

axis2 label = (angle=90 "ODDS")
   logbase=e logstyle=expand;

symbol1 i=join v=plus;

proc gplot data=plotit2;
   plot odds*score
   /vaxis=axis2
   grid;
run;quit;
```

Figure 10. Output from GPLOT code using output from PROC RANK.
HOW TO HANDLE WEIGHTED DATA

PROC RANK does not have a WEIGHT statement. We often sample different populations at different rates and require results to be weighted back by a variable that indicates the weight used in sampling. Let's modify the TEST2 data and generate samples from the data. Since the events (EVENT=1) are rare, let's sample respondents at 100% and only sample the non-events (EVENT=0) at 2%. Code to generate the sampled data:

data test2;
  do i = 1 to 10000;
    score = round(650+rannor(243)*20,1);
    event = (score+rannor(12)*25>700);
    noa=1;
    neg_score = score*-1;
    r_score = neg_score;
    output;
  end;
run;

proc surveyselect data=test2 (where=(event=1)) out=events seed=62309 method=srs rate=1 stats;
run;

proc surveyselect data=test2 (where=(event=0)) out=event0 seed=62309 method=srs rate=.02 stats;
run;

proc append base=events data=event0 force;
run;

In lines 1 and 2 we add variables that we will use in subsequent TABULATE code to weight up to population results. In line 2 we multiply the score by -1 since we want to create a descending effect. If you want to have a ascending effect just have R_SCORE=SCORE and drop the NEG_SCORE assignment. In lines 3 and 4 we run PROC SURVEYSELECT to generate the samples from TEST2 data. The STATS option will generate the weight variable (SAMPLINGWEIGHT) for observations in the final data set. PROC APPEND in line 5 combines the 2 data sampled data into a single output data called EVENTS. More information on using PROC SURVEYSELECT can be found in Cassell (2007, 2008).

We can use PROC UNIVARIATE to generate weighted percentiles and output to a data set. Let’s see how PROC UNIVARIATE can generate the decile values from the sampled EVENTS data:

proc univariate data=events noprint;
  var neg_score;
  output out=p pctlpre=P_ pctlpts=10 to 100 by 10;
  weight SamplingWeight;
run;

proc transpose data=p out=pt;
run;

proc sort data=pt
  nodupkey force noequals;
  by COL1;
run;

In lines 8 and 9 we add variables that we will use in subsequent TABULATE code to weight up to population results. In line 9 we multiply the score by -1 since we want to create a descending effect. If you want to have a ascending effect just have R_SCORE=SCORE and drop the NEG_SCORE assignment. In lines 10 and 11 we run PROC SURVEYSELECT to generate the samples from TEST2 data. The STATS option will generate the weight variable (SAMPLINGWEIGHT) for observations in the final data set. PROC APPEND in line 12 combines the 2 data sampled data into a single output data called EVENTS. More information on using PROC SURVEYSELECT can be found in Cassell (2007, 2008).

We can use PROC UNIVARIATE to generate weighted percentiles and output to a data set. Let’s see how PROC UNIVARIATE can generate the decile values from the sampled EVENTS data:

proc univariate data=events noprint;
  var neg_score;
  output out=p pctlpre=P_ pctlpts=10 to 100 by 10;
  weight SamplingWeight;
run;

proc transpose data=p out=pt;
run;

proc sort data=pt
  nodupkey force noequals;
  by COL1;
run;

Fig 11.
NAME_     COL1
P_10       -676
P_20       -665
P_30       -660
P_40       -655
P_50       -650
P_60       -645
P_70       -640
P_80       -632
P_90       -623
P_100      -588
In the PROC UNIVARIATE code above we generate the OUTPUT data set P than has percentile specification for
deciles. We also include the WEIGHT statement to weight the results by the weight variable generated by SURV-
VEYSELECT.

We TRANSPOSE the resulting output from UNIVARIATE and remove duplicate decile values in the PROC
SORT to handle any ties. This will mimic the TIES=DENSE specification in PROC RANK. Figure 11 shows us a
PROC PRINT from the PT data.

We can take the results from the PT data and use it to generate a user defined FORMAT to use when we report
the population results. Code is listed here:

```sas
data cntlin;
  set pt end=eof;
  length HLO SEXCL EEXCL $1 LABEL $2;
  retain fmtname "score" type 'N' end;

  nrec+1;
  if nrec=1 then do;
    HLO='L'; SEXCL='N'; EEXCL='Y'; start=.; end=COL1;
    label=put(nrec-1,z2.); output;
  end;
  else if not eof then do;
    HLO=' '; SEXCL='N'; EEXCL='Y'; start=end; end=COL1;
    label=put(nrec-1,z2.); output;
  end;
  else if eof then do;
    HLO='H'; SEXCL='N'; EEXCL='N'; start=end; end=.;
    label=put(nrec-1,z2.); output;
  end;
run;

proc format cntlin=cntlin;
run;
```

We are setting up a user defined FORMAT from the PT data. Labels will take on the row number we read from
PT calculated in line 3 and formatted as a 2 digit leading zero value. I use 2 digits in the event we want to modify
the code for more 10 groups. To prevent a multi label FORMAT, we specify that ranges (START to END) be
less than the decile cut-off (COL1) (3) except for the last record (EOF=1) where we set a HIGH keyword value
for the end condition (5). START values are the retained END values from the previous record except for the
first record where we set up a LOW keyword value. FMTLIB prints out the generated SCORE FORMAT as
shown on the next page.
We can now use the SCORE format in a report. Code is provided using PROC TABULATE. Similar results can be obtained using PROC MEANS with PROC PRINT. Note that the analysis variable is SCORE and the CLASS variable is R_SCORE. These can be the same values but for this example, R_SCORE is the negative of SCORE to get the descending effect. Also note that we apply the user defined FORMAT SCORE we created in code above in the FORMAT statement in PROC TABULATE. Code and output:

```plaintext
proc format;
  picture pct (round) low-high = '009.99%';
run;

proc tabulate data=events noseps formchar='           ';
  class r_score;
  format r_score score.;
  var score event noa;
  weight SamplingWeight;
  keylabel sum=' ' pctsum=' ' mean=' ';
  table r_score='Ranked Score' all
    ,
    noa='Weighted N'*f=comma8.
    noa='%'*pctsum<r_score all>*f=pct.
    score*(min max)*f=3.
    event='event rate'*mean*f=percent8.2
    /rts=10 row=flloat misstext=' ';
  title UNIVARIATE;
run;
```
### UNIVARIATE

<table>
<thead>
<tr>
<th>Score</th>
<th>N</th>
<th>%</th>
<th>Min</th>
<th>Max</th>
<th>Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>966</td>
<td>9.66%</td>
<td>677</td>
<td>719</td>
<td>27.84%</td>
</tr>
<tr>
<td>01</td>
<td>899</td>
<td>8.99%</td>
<td>666</td>
<td>676</td>
<td>16.90%</td>
</tr>
<tr>
<td>02</td>
<td>945</td>
<td>9.45%</td>
<td>661</td>
<td>665</td>
<td>5.08%</td>
</tr>
<tr>
<td>03</td>
<td>897</td>
<td>8.97%</td>
<td>656</td>
<td>660</td>
<td>5.58%</td>
</tr>
<tr>
<td>04</td>
<td>1,272</td>
<td>12.72%</td>
<td>651</td>
<td>655</td>
<td>2.12%</td>
</tr>
<tr>
<td>05</td>
<td>764</td>
<td>7.64%</td>
<td>646</td>
<td>650</td>
<td>2.22%</td>
</tr>
<tr>
<td>06</td>
<td>1,204</td>
<td>12.04%</td>
<td>641</td>
<td>645</td>
<td>0.75%</td>
</tr>
<tr>
<td>07</td>
<td>957</td>
<td>9.57%</td>
<td>633</td>
<td>640</td>
<td>1.15%</td>
</tr>
<tr>
<td>08</td>
<td>998</td>
<td>9.98%</td>
<td>624</td>
<td>632</td>
<td>0.20%</td>
</tr>
<tr>
<td>09</td>
<td>1,097</td>
<td>10.97%</td>
<td>588</td>
<td>623</td>
<td>0.09%</td>
</tr>
<tr>
<td>All</td>
<td>10,000</td>
<td>100.00%</td>
<td>588</td>
<td>719</td>
<td>5.86%</td>
</tr>
</tbody>
</table>

### OTHER CONSIDERATIONS AND OPTIONS WHEN USING PROC RANK

There are a few other options and considerations when using PROC RANK. Some of these will be mentioned in this section with additional examples.

### SPECIFYING MULTIPLE VARIABLES AND ROUNING.

Examples we have touched on have RANKED only 1 variable. We can specify any number of variables to rank in a single step with or without multiple RANKS specification. However, variables and the ranks must be processed in memory. You may need MEMSIZE and/or REALSIZE options to get PROC RANK to run with large number of variables.

With respect to rounding, I received this next example from Mike Zdeb who illustrated that as a result of how SAS stores numeric data, it maybe wise to round the variables to the expected precision so that you don’t get unexpected results. This example also illustrates multiple variables ranked in a single PROC RANK. Code:

```sas
data test;
  input fn1 fn2 @@;
  diffa = fn2 - fn1;
  diffb = round(diffa, 0.01);
datalines;
0.72 0.68 0.74 0.64 0.63 0.64 0.64 0.69 0.7 0.54 0.56 0.5 0.83 0.78 0.56 0.49
0.65 0.77 0.9 0.87 0.7 0.74 0.85 0.76 0.55 0.56 0.68 0.64 0.64 0.61 0.61 0.66
0.82 0.88 0.66 0.63 0.66 0.71 0.66 0.61 0.59 0.65 0.78 0.63 0.61 0.81 0.82 0.67
0.65 0.76 0.71 0.71 0.74 0.74 0.6 0.85 0.49 0.48 0.54 0.38 0.69 0.67 0.73 0.83
0.67 0.63 0.6 0.54 0.47 0.54 0.42 0.46 0.69 0.64 0.65 0.62 0.57 0.71 0.67 0.79
0.81 0.72;
run;
```
proc rank data=test out=ranks ties=low;
  var diffa diffb;
  ranks r_diffa r_diffb;
run;

proc print data=ranks;
run;

proc print data=ranks;
  format diffa diffb 21.18;
run;

Selected output from the first PROC PRINT shows that DIFFA and DIFFB are identical but the ranks are different:

<table>
<thead>
<tr>
<th>Obs</th>
<th>fn1</th>
<th>fn2</th>
<th>diffa</th>
<th>diffb</th>
<th>r_diffa</th>
<th>r_diffb</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.72</td>
<td>0.68</td>
<td>-0.04</td>
<td>-0.04</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>0.74</td>
<td>0.64</td>
<td>-0.10</td>
<td>-0.10</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>0.63</td>
<td>0.64</td>
<td>0.01</td>
<td>0.01</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>0.64</td>
<td>0.69</td>
<td>0.05</td>
<td>0.05</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>5</td>
<td>0.70</td>
<td>0.54</td>
<td>-0.16</td>
<td>-0.16</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0.56</td>
<td>0.50</td>
<td>-0.06</td>
<td>-0.06</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>0.83</td>
<td>0.78</td>
<td>-0.05</td>
<td>-0.05</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>

Looking at the second PROC PRINT with more significant digits displayed shows that DIFFA and DIFFB are different, resulting in different ranks.

<table>
<thead>
<tr>
<th>diffa</th>
<th>diffb</th>
<th>r_diffa</th>
<th>r_diffb</th>
</tr>
</thead>
<tbody>
<tr>
<td>-0.039999999999999900</td>
<td>-0.040000000000000000</td>
<td>16</td>
<td>14</td>
</tr>
<tr>
<td>-0.100000000000000000</td>
<td>-0.100000000000000000</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>0.010000000000000000</td>
<td>0.010000000000000000</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>0.049999999999999900</td>
<td>0.050000000000000000</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>-0.159999999999999900</td>
<td>-0.160000000000000000</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>-0.060000000000000000</td>
<td>-0.060000000000000000</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>-0.049999999999999900</td>
<td>-0.050000000000000000</td>
<td>12</td>
<td>11</td>
</tr>
</tbody>
</table>
BY VARIABLES AND P OPTIONS

With PROC RANK, we can return the percentiles (0-100) in the RANKS variable using the P option. The F option is similar but returns fraction (0-1). An application of these options can be used to calculate the 2 sample Kolmogorov-Smirnov (KS) statistic. The 2 sample KS statistic is a non-parametric test to see if 2 samples have the same distribution. It is defined as:

$$KS = \max_j \left| F_1(x_j) - F_2(x_j) \right|$$

Where KS is the maximum absolute cumulative percentile difference for variable x for 2 samples. PROC NPAR1WAY can calculate this statistic but sometimes people like to calculate the KS on a decile level as opposed to the individual score levels. To compare the results, let's look at the 2 sample KS for the TEST2 data from PROC NPAR1WAY and then compare with some calculations in PROC RANK.

The output you want to look at from PROC NPAR1WAY is D-Statistic. Here is the code (SAS 9.2):

```sas
proc npar1way data=test2 D;
class event;
var score;
run;
```

Selected output from NPAR1WAY:

<table>
<thead>
<tr>
<th>Kolmogorov-Smirnov Two-Sample Test (Asymptotic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D = max</td>
</tr>
<tr>
<td>Pr &gt; D</td>
</tr>
</tbody>
</table>

Note that this is a 2 tail test of significance and that NPAR1WAY also prints out lower and upper tail tests.

We can use PROC RANK to get similar results on a decile level. However note that the result we get from NPAR1WAY is much easier to code up. If we sort the TEST2 data by EVENT, we can get P (percentiles) for each value of EVENT in one PROC RANK using a BY statement. Notice the P option in PROC RANK:

```sas
proc rank data=test2 out=events
ties=low descending p;
by event;
var score;
ranks F_score;
run;
```

We can then decile the SCORE variable in data EVENTS using GROUPS= options:

```sas
proc rank data=events
out=test3
groups=10
ties=low
descending;
var score;
ranks D_score;
run;
```
Data TEST3 from this latest PROC RANK has these variables we can use to calculate the KS values at each decile:

- **SCORE**: Raw score.
- **EVENT**: 1,0
- **F_SCORE**: Cumulative percentile of score by EVENT groups.
- **D_SCORE**: Deciles (0-9) of SCORE.

Recall that PROC RANK has no output, so we now need to calculate some results off of TEST3.

```sas
proc means data=test3 noprint nway;
class D_score;
var F_score;
output out=event max(F_score)=F_event
    n=n_event;
    where event;
run;

proc means data=test3 noprint nway;
class D_score;
var F_score;
output out=nonevent max(F_score)=F_nonevent
    n=n_nonevent;
    where event=0;
run;

data ks;
    merge event nonevent;
    by D_score;
    KS=abs(F_event - F_nonevent);
    n=n_event + n_nonevent;
run;
```

Now we can print KS results at each decile:

```sas
proc format;
    picture pct (round)
        low-high = '009.99%';
run;

proc print data=ks noobs label;
    var D_score n F_event F_nonevent KS;
    format n comma6. F_event
        F_nonevent KS pct. ;
    label D_score='Decile'
        F_event='Cumulative Event'
        F_nonevent='Cumulative Non Event';
run;
```

<table>
<thead>
<tr>
<th>Decile</th>
<th>n</th>
<th>Cumulative Event</th>
<th>Cumulative Non Event</th>
<th>KS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1,009</td>
<td>46.08%</td>
<td>6.75%</td>
<td>39.33%</td>
</tr>
<tr>
<td>1</td>
<td>1,111</td>
<td>69.62%</td>
<td>16.84%</td>
<td>52.79%</td>
</tr>
<tr>
<td>2</td>
<td>982</td>
<td>80.20%</td>
<td>25.98%</td>
<td>54.22%</td>
</tr>
<tr>
<td>3</td>
<td>1,094</td>
<td>89.25%</td>
<td>36.73%</td>
<td>52.52%</td>
</tr>
<tr>
<td>4</td>
<td>998</td>
<td>94.20%</td>
<td>47.20%</td>
<td>47.00%</td>
</tr>
<tr>
<td>5</td>
<td>810</td>
<td>96.25%</td>
<td>55.53%</td>
<td>40.71%</td>
</tr>
<tr>
<td>6</td>
<td>1,061</td>
<td>97.78%</td>
<td>67.31%</td>
<td>30.47%</td>
</tr>
<tr>
<td>7</td>
<td>975</td>
<td>99.49%</td>
<td>77.70%</td>
<td>21.78%</td>
</tr>
<tr>
<td>8</td>
<td>969</td>
<td>99.83%</td>
<td>88.64%</td>
<td>11.18%</td>
</tr>
<tr>
<td>9</td>
<td>991</td>
<td>100.00%</td>
<td>100.00%</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

Fig 12

Applications Big & Small

NESUG 2009
Figure 12 has the output. Notice that the resulting overall KS is the maximum absolute difference printed in the last column. Result here, on a decile level, is KS=54.22%. The KS (D statistic) from NPAR1WAY was 0.5463 (or 54.63%). Comparable but NPAR1WAY will typically be higher than the decile PROC RANK method in that the NPAR1WAY statistic was calculated on a SCORE value as opposed to a decile (D_SCORE) value and maximum separation may not always occur at the decile level.

**ORDER (DEFAULT OR DESCENDING) AND TIES MAY GIVE YOU DIFFERENT RESULTS**

With PROC RANK results may be different dependent on order or TIES specifications. The number of GROUPS you get and the frequency by group may change depending on order and TIES specifications. Let’s look at a simple example to illustrate. Code generates some simulated data where we only have 2 values of the variable we wish to rank into 2 groups. The ORDERS data has 100 rows with 40 at 0 and 60 at 1.

```plaintext
data orders;
  do i = 1 to 100;
    x = (i>40);
    output;
  end;
run;
proc rank data=orders out=ranked
ties=low    groups=2;
  var x;
  ranks r_x;
run;
proc freq data=ranked;
table r_x;
run;
```

Output shows that there is only 1 group. At first this may not make sense but makes sense when you think how you coded PROC RANK. We specified GROUP=2 with TIES=LOW. The 50th percentile of X in the data is 1 and goes into the first group, R_X of 0. All other values are 1 so all get dumped into group 0.

```plaintext
Rank for Variable

<table>
<thead>
<tr>
<th>r_x</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
<td>100.00</td>
<td>100</td>
<td>100.00</td>
</tr>
</tbody>
</table>
```

Results would be different dependent on how we specify the order and TIES. If we specify DESCENDING option the results are:

```plaintext
Rank for Variable x

<table>
<thead>
<tr>
<th>r_x</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>60</td>
<td>60.00</td>
<td>60</td>
<td>60.00</td>
</tr>
<tr>
<td>1</td>
<td>40</td>
<td>40.00</td>
<td>100</td>
<td>100.00</td>
</tr>
</tbody>
</table>
```
If we use defaults for ties and order results are:

<table>
<thead>
<tr>
<th>Rank for Variable x</th>
<th>Cumulative</th>
<th>Cumulative</th>
</tr>
</thead>
<tbody>
<tr>
<td>r_x</td>
<td>Frequency</td>
<td>Percent</td>
</tr>
<tr>
<td>0</td>
<td>40</td>
<td>40.00</td>
</tr>
<tr>
<td>1</td>
<td>60</td>
<td>60.00</td>
</tr>
</tbody>
</table>

Of course, if we specified GROUPS=10, all runs should get 2 groups. It is important to know your data and the question I would have is why group rank a variable that only has 2 values?

**CONCLUSIONS**

We have looked at using PROC RANK to rank and group numeric data. The procedure is very simple to use and often easier to use than using sorts, macros, and data steps to get the same results. PROC RANK does not produce any listed output so you will need subsequent procedures or data steps to process the data for analysis and/or reporting. There are a few PROC RANK options that generate normal or exponential scores from the data, but those options are left to the reader to research. We have also looked at how PROC UNIVARIATE and PROC FORMAT can be used when we need to get deciles or other grouping with sampled data.

**REFERENCES:**

ACKNOWLEDGMENTS
Thanks to Yan Jiang and Haiyan Weng for asking me how to handle weights in ranking.
Thanks as well to WenSui Liu and Mike Zdeb for providing me with new and interesting ways of doing things in SAS.
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
This work is an independent effort and does not necessarily represent the practices followed at JP Morgan Chase Bank.

CONTACT INFORMATION
In case a reader wants to get in touch with you, please put your contact information at the end of the paper.)
Your comments and questions are valued and encouraged. Contact the author at:

Jonas V. Bilenas
JP Morgan Chase Bank
Wilmington, DE 19801
Email: Jonas.Bilenas@chase.com
jonas@jonasbilenas.com

********************************************