Pivoting Data. An Alternative to ACROSS Variables of the REPORT Procedure

Analysis often demands that prospects be grouped according to the values of a class variable. Business and sampling considerations dictate the number and size of these segments, which sometimes change within the same project. The creation of **reports data sets** as opposed to, or in addition to **actual reports**, increases flexibility. Because in most business settings, portions of projects run several times with minor adjustments, flexibility creates opportunities for timesaving and easy report sharing between systems.

A second segmentation usually improves the quality of business intelligence. At the first summary level, data analysis systems produce segmented **results data sets** in list form. That is the case of SAS® file, *SampleResult30Nov2001*. In a time series, an analyst had tagged credit cardholders as good or bad according to their delinquency status. Created from that **analysis-ready data set**, the results file contains good/bad odds based on the number of records, balance, and annual profit by behavior score and credit bureau (FICO®) score. Dividing the number of good by the number of bad customers gives good/bad odds based on the number of records.
For details, download *SampleResult30Nov2001* from http://www.visualstat.com/bikila.

```
        Output 0.1. Data Set rpt.SampleResult30Nov2001
         Statistics Based on Number of Records (Obs=8)

            Behavior CB
      Month Score    Score    Records          Records Records
        End Break    Break        Odd  Records    Good     Bad
__________________________________________________________

31MAY2002 Low-559  450-499          0       36       6      30
31MAY2002 Low-559  500-549          0       37       7      30
31MAY2002 Low-559  550-599          0       23       1      22
31MAY2002 Low-559  600-649          0        4       1       3
31MAY2002 Low-559  650-699          .        4       .       4
31MAY2002 Low-559  750-799          .        1       .       1
31MAY2002 560-589  Low-449          1        9       3       6
31MAY2002 560-589  450-499          1      102      37      65
```

BehaviorScoreBreak and *CBScoreBreak* are the main class variables, followed by summary statistics for twelve analysis variables, including *RecordsOdd, Records, RecordsGood, RecordsBad*, and not shown here, *SumAnnualProfitOdd, SumAnnualProfit, SumAnnualProfitGood, and SumAnnualProfitBad*. ACROSS variables of the REPORT procedure effectively pivot results data sets (Output 1.1.) One problem with resulting output data sets are awkward column names like _c4_, difficult to associate with the class-variable values they represent. Renaming and labeling these columns in real-life is tedious.

This paper offers an alternate pivot routine that automatically creates meaningful column names and labels. After replacing invalid symbols such as dashes (-) with valid characters such as underscores (_), it associates each column name with the name of the corresponding analysis variable and the value of the pivoted class variable. It controls the length of resulting variable names.

1. Pivoted Summary Statistics: The REPORT Procedure

The SORT procedure with the NODUPKEY option returns distinct values of *CBScoreBreak* and *BehaviorScoreBreak* (not shown), which feed the FORMAT procedure. The latter creates user-defined formats that establish links between score ranges and the corresponding numeric ranks as shown in the next DATA step.

```
title;
options nodate nonumber ps=32700 formdlim=' ';
filename rpttxt  "C:\bbg\!Bikila Papers\SESUG03\Data Presentation Section";
libname  rpt     "C:\bbg\!Bikila Papers\SESUG03\Data Presentation Section";
libname  library "C:\bbg\!Bikila Papers\SESUG03\Data Presentation Section";
```

```
  /*Unique Values of Across
    Variable CBScoreBreak*/
proc sort data=rpt.SampleResult30Nov2001
           out=rpt.ParmCBScoreBreak
              (keep=CBScoreBreak) nodupkey;
   by CBScoreBreak;
run;
```

```
  /*Rename rpt.SampleResult30Nov2001
    before creating new variables*/
proc datasets lib=rpt nolist;
   change SampleResult30Nov2001=iSampleResult30Nov2001;
   run;
   contents data=iSampleResult30Nov2001;
   run;
quit;
```

```
  /*Routine 1.1. Create BehaviorScoreRank & CBScoreRank*/
data rpt.SampleResult30Nov2001;
   length PerformanceMonth MonthEnd
          BehaviorScoreRank 8 BehaviorScoreBreak $ 10
          CBScoreRank 8 CBScoreBreak $ 10;
   set rpt.iSampleResult30Nov2001;
   label BehaviorScoreRank="Behavior Score Rank"
         CBScoreRank="CB Score Rank";
   BehaviorScoreRank=input(put(BehaviorScoreBreak,$bsrkfmt.),?? 2.);
   CBScoreRank=input(put(CBScoreBreak,$cbrkfmt.),?? 2.);
run;
```

With format **$cbrkfmt.**, the PUT function converts *CBScoreBreak* into serial numbers 4, 5, …, 13 of type character. As such, their default sort order is 10, 11, 12, 13, 4, …, 9. The INPUT function converts these characters to numeric values and assigns them to *CBScoreRank*. Should the PUT function generate missing values (blanks), the double-question mark (??) ensures that automatic variable _ERROR_ is never set to 1, no error messages appear in the log, and the DATA step never stops. The same applies to *BehaviorScoreRank*.

Using ACROSS variable *CBScoreRank*, the REPORT procedure successfully pivots data in the input data set to produce a joint odds report based on the number of records. [1] Bikila bi Gwet (2003) shows the detailed coding.

In the output data set, the column names the REPORT procedure generates present challenges, especially in the presence of dozens of variables and many similar but different reports, particularly during process automation through macro programming.

```
              Output 1.1. Data Set rpt.RecordsOddProcReport

 Behavior Behavior
    Score Score
     Rank Break    _C3_ _C4_ _C5_ _C6_ _C7_ _C8_ _C9_ _C10_ _C11_ _C12_
        3 Low-559     .    0    0    0    0    .    .     .     .     .
        4 560-589     0    0    0    0    0    0    1     1     .     .
        5 590-619     1    1    1    2    2    2    2     1     1     .
        6 620-649     0    1    2    3    4    8    9    12    14     .
        7 650-679     2    2    5    6    9   14   34    55    97     .
        8 680-709     4    4    6   10   15   29   82   203   136     .
        9 710-739     2   11    7   15   33   59  224   492   723     .
       10 740-769     .   11   26   67   95  147  649  1366   692     .
       11 770-799     .    .    .    .    .  422  788  1635  3286     .
```

```
proc format lib=library;
   value $bsrkfmt
        'Low-559'='3'
        '560-589'='4'
        '590-619'='5'
        '620-649'='6'
        '650-679'='7'
        '680-709'='8'
        '710-739'='9'
        '740-769'='10'
        '770-799'='11';
   value $cbrkfmt
        'Low-449'='4'
        '450-499'='5'
        '500-549'='6'
        '550-599'='7'
        '600-649'='8'
        '650-699'='9'
        '700-749'='10'
        '750-799'='11'
        '800-849'='12'
        '850-899'='13';
   value rkbsfmt
         3='Low-559'
         4='560-589'
         5='590-619'
         6='620-649'
         7='650-679'
         8='680-709'
         9='710-739'
        10='740-769'
        11='770-799';
   value rkcbfmt
         4='Low-449'
         5='450-499'
         6='500-549'
         7='550-599'
         8='600-649'
         9='650-699'
        10='700-749'
        11='750-799'
        12='800-849'
        13='850-899';
run;
```

2. Design of the PIVOT Routine

2.1. Data Transpose and Data Pivot

Consider the following summary *Results* data set. Because of the large number of analysis variables (not shown here), the analyst-programmer decides to create *Reports* data sets by rotating *Results*. Hence, she can display 12 months worth of data on an 8½ x11 landscape-laid out sheet.

In addition, the analyst wants to produce a cross section of high and low balances by behavior score and FICO® score.

```
  /*Routine 2.1*/
data rpt.Results;
   input BehaviorScore:$7. FICODim FICOScore:$7.
         Date:date. HighBal:comma. LowBal:comma.;
   format Date date9. HighBal LowBal dollar7.;
   cards;
650-679 1 700-749 31Jan2000 $1,934   $501
650-679 2 750-799 28Feb2000 $9,763 $1,983
650-679 3 800-849 31Mar2000 $7,327 $5,671
680-709 1 700-749 30Apr2000 $3,573 $2,231
680-709 2 750-799 31May2000 $5,032 $4,185
680-709 3 800-849 30Jun2000 $2,484 $1,339
;
proc report data=rpt.Results nowd;
   columns BehaviorScore FICODim FICOScore
           Date HighBal LowBal;
   define BehaviorScore / width=8
         'Behavior/Score' spacing=1;
   define FICODim / width=4
         'FICO/Dim' spacing=1;
   define FICOScore / 'FICO/Score' spacing=1;
   define HighBal / 'High/Bal' spacing=1;
   define LowBal / 'Low/Bal' spacing=1;
   title 'Output 2.1. Data Set rpt.Results';
run;
title;
```

```
         Output 2.1. Data Set rpt.Results

 Behavior FICO FICO                  High     Low
 Score     Dim Score         Date     Bal     Bal
 650-679     1 700-749  31JAN2000  $1,934    $501
 650-679     2 750-799  28FEB2000  $9,763  $1,983
 650-679     3 800-849  31MAR2000  $7,327  $5,671
 680-709     1 700-749  30APR2000  $3,573  $2,231
 680-709     2 750-799  31MAY2000  $5,032  $4,185
 680-709     3 800-849  30JUN2000  $2,484  $1,339
```

```
  /*Routine 2.2. Data Transpose*/
data rpt.Report1(keep=Variable
                      Date31Jan2000 Date28Feb2000
                      Date31Mar2000 Date30Apr2000
                      Date31May2000 Date30Jun2000);
   set rpt.Results end=last;
   length Variable $ 8;
   format Date31Jan2000 Date28Feb2000
          Date31Mar2000 Date30Apr2000
          Date31May2000 Date30Jun2000
          dollar10.;
   label Date31Jan2000="31Jan2000"
         Date28Feb2000="28Feb2000"
         Date31Mar2000="31Mar2000"
         Date30Apr2000="30Apr2000"
         Date31May2000="31May2000"
         Date30Jun2000="30Jun2000";
   /*Original variables*/
   array oldv{2} HighBal LowBal;
   /*New variables*/
   array newv{6} Date31Jan2000 Date28Feb2000
                 Date31Mar2000 Date30Apr2000
                 Date31May2000 Date30Jun2000;
   /*Look-up array*/
   array allv{2,6} _temporary_;
   retain allv;
   /*Look up all analysis data values*/
   do jold=1 to 2;
      allv{jold,_n_}=oldv{jold};
   end; /*do jold*/
   /*Process after reading last observation*/
   if last then do jold=1 to 2;
      select(jold); /*Analysis variables*/
         when(1) Variable='HighBal';
         when(2) Variable='LowBal';
         otherwise;
      end; /*select*/
      /*Assign looked-up data to new variables*/
      do jnew=1 to 6;
         newv{jnew}=allv{jold,jnew};
      end; /*do jnew*/
      output;
   end; /*do jold*/
run;
proc print noobs label;
   title 'Output 2.2. Data Set rpt.Report1';
run; title;
```

```
                                Output 2.2. Data Set rpt.Report1

  Variable     31Jan2000     28Feb2000     31Mar2000     30Apr2000     31May2000     30Jun2000

  HighBal         $1,934        $9,763        $7,327        $3,573        $5,032        $2,484
  LowBal            $501        $1,983        $5,671        $2,231        $4,185        $1,339
```

Whereas data in *Report1* result from a **data transpose**, a **pivot routine** creates data set *Report2*. [1] Bikila bi Gwet (2003) further discusses the transpose and pivot routines on this sample data set. For now, let's point out that apart from its mechanics, the pivot routine presents additional challenges that minuscule sample data sets misrepresent. On one hand, array definitions make use of variable names that are *FICOScore*-value specific and well ordered by these values. On the other hand, the routine creates appropriate labels for new variables. In other words, to ensure flexibility and allow for automation, **the pivot routine must establish a link between data values and variable attributes**. The remainder of this paper develops and applies these techniques to the *SampleResult30Nov2001* data set.

```
                Output 2.3. Data Set rpt.Report2
               High and Low Balances by FICO Score

 Behavior High Bal High Bal High Bal  Low Bal  Low Bal  Low Bal
 Score     700-749  750-799  800-849  700-749  750-799  800-849
 650-679    $1,934   $9,763   $7,327     $501   $1,983   $5,671
 680-709    $3,573   $5,032   $2,484   $2,231   $4,185   $1,339
```

2.2. Straight SAS PIVOT

An earlier SORT procedure saved distinct values of *CBScoreBreak* in data set *ParmCBScoreBreak*. This data set will help create a parameter table (*MotherSonMatrix*) holding new variable names, labels and other data and variable attributes. Variable *AcrossDimension* is a true serial rank that begins with 1 irrespective of the first value of *AcrossRank* inherited from user-defined format **$cbrkfmt.** This property proves useful later in the process for if the first value of *AcrossRank* is not 1, this variable cannot replace *AcrossDimension* in the pivot process.

```
   /*Routine 2.3. Across Rank & Mother Length*/
data work.ParmCBScoreBreak;
   set rpt.ParmCBScoreBreak end=last;
   AcrossRank=
      input(put(CBScoreBreak,$cbrkfmt.),?? 2.);
   retain MotherLength 0;
   MotherLength=
      max(length(CBScoreBreak),MotherLength);
run;
```

```
  /*Routine 2.4. Across Dimension*/
proc sort data=work.ParmCBScoreBreak;
   by AcrossRank;
data rpt.ParmCBScoreBreak;
   set work.ParmCBScoreBreak;
   AcrossDimension+1;
run;
```

```
  /*Routine 2.5. Get variable AcrossDimension*/
proc sort data=rpt.SampleResult30Nov2001;
   by CBScoreRank;
data rpt.SampleResult30Nov2001;
   length PerformanceMonth MonthEnd
          BehaviorScoreRank 8
          BehaviorScoreBreak $ 10 AcrossDimension
          CBScoreRank 8 CBScoreBreak $ 10;
   label AcrossDimension='Across Dimension';
   merge rpt.SampleResult30Nov2001(in=inresult)
         rpt.ParmCBScoreBreak
        (keep=AcrossDimension AcrossRank
         rename=(AcrossRank=CBScoreRank));
   by CBScoreRank;
   if inresult;
proc sort data=rpt.SampleResult30Nov2001;
   by MonthEnd BehaviorScoreRank CBScoreRank;
run;
```

Routine 2.6 creates data set in Output 2.4, which displays the same information as the REPORT procedure (Output 1.1.)

```
                 Output 2.4. Data Set rpt.RecordsOdd30Nov2001
                        PerformanceMonth=12 (30Nov2002)

Behavior Records Records Records Records Records Records Records Records Records Records
 Score     Odd     Odd     Odd     Odd     Odd     Odd     Odd     Odd     Odd     Odd
 Break   Low-449 450-499 500-549 550-599 600-649 650-699 700-749 750-799 800-849 850-899

Low-559     .         0       0       0       0       .       .       .       .       .
560-589     0         0       0       0       0       0       1       1       .       .
590-619     1         1       1       2       2       2       2       1       1       .
620-649     0         1       2       3       4       8       9      12      14       .
650-679     2         2       5       6       9      14      34      55      97       .
680-709     4         4       6      10      15      29      82     203     136       .
710-739     2        11       7      15      33      59     224     492     723       .
740-769     .        11      26      67      95     147     649    1366     692       .
770-799     .         .       .       .       .     422     788    1635    3286       .
```

```
  /*Routine 2.6. Straight SAS Pivot: Report Data Set for Joint Odds Based on Records*/
data rpt.RecordsOdd30Nov2001
    (keep=PerformanceMonth MonthEnd BehaviorScoreRank BehaviorScoreBreak
          RecordsOddLow_449 RecordsOdd450_499 RecordsOdd500_549 RecordsOdd550_599
          RecordsOdd600_649 RecordsOdd650_699 RecordsOdd700_749 RecordsOdd750_799
          RecordsOdd800_849 RecordsOdd850_899 RecordsGoodLow_449 RecordsGood450_499
          RecordsGood500_549 RecordsGood550_599 RecordsGood600_649 RecordsGood650_699
          RecordsGood700_749 RecordsGood750_799 RecordsGood800_849 RecordsGood850_899
          RecordsBadLow_449 RecordsBad450_499 RecordsBad500_549 RecordsBad550_599
          RecordsBad600_649 RecordsBad650_699 RecordsBad700_749 RecordsBad750_799
          RecordsBad800_849 RecordsBad850_899);
   length PerformanceMonth MonthEnd BehaviorScoreRank 8 BehaviorScoreBreak $ 10;
   label RecordsOddLow_449='Records Odd Low-449' RecordsOdd450_499='Records Odd 450-499'
         RecordsOdd500_549='Records Odd 500-549' RecordsOdd550_599='Records Odd 550-599'
         RecordsOdd600_649='Records Odd 600-649' RecordsOdd650_699='Records Odd 650-699'
         RecordsOdd700_749='Records Odd 700-749' RecordsOdd750_799='Records Odd 750-799'
         RecordsOdd800_849='Records Odd 800-849' RecordsOdd850_899='Records Odd 850-899'
         RecordsGoodLow_449='Records Good Low-449' RecordsGood450_499='Records Good 450-499'
         RecordsGood500_549='Records Good 500-549' RecordsGood550_599='Records Good 550-599'
         RecordsGood600_649='Records Good 600-649' RecordsGood650_699='Records Good 650-699'
         RecordsGood700_749='Records Good 700-749' RecordsGood750_799='Records Good 750-799'
         RecordsGood800_849='Records Good 800-849' RecordsGood850_899='Records Good 850-899'
         RecordsBadLow_449='Records Bad Low-449' RecordsBad450_499='Records Bad 450-499'
         RecordsBad500_549='Records Bad 500-549' RecordsBad550_599='Records Bad 550-599'
         RecordsBad600_649='Records Bad 600-649' RecordsBad650_699='Records Bad 650-699'
         RecordsBad700_749='Records Bad 700-749' RecordsBad750_799='Records Bad 750-799'
         RecordsBad800_849='Records Bad 800-849' RecordsBad850_899='Records Bad 850-899';
   set rpt.SampleResult30Nov2001 end=last;
   format RecordsGoodLow_449 RecordsGood450_499 RecordsGood500_549 RecordsGood550_599
          RecordsGood600_649 RecordsGood650_699 RecordsGood700_749 RecordsGood750_799
          RecordsGood800_849 RecordsGood850_899 RecordsBadLow_449 RecordsBad450_499
          RecordsBad500_549 RecordsBad550_599 RecordsBad600_649 RecordsBad650_699
          RecordsBad700_749 RecordsBad750_799 RecordsBad800_849 RecordsBad850_899 comma9.;
   by MonthEnd BehaviorScoreRank CBScoreRank;
   array roddar{*} RecordsOddLow_449 RecordsOdd450_499 RecordsOdd500_549
                   RecordsOdd550_599 RecordsOdd600_649 RecordsOdd650_699
                   RecordsOdd700_749 RecordsOdd750_799 RecordsOdd800_849 RecordsOdd850_899;
   array rgoodar{*} RecordsGoodLow_449 RecordsGood450_499 RecordsGood500_549
                    RecordsGood550_599 RecordsGood600_649 RecordsGood650_699
                    RecordsGood700_749 RecordsGood750_799 RecordsGood800_849 RecordsGood850_899;
   array rbadar{*} RecordsBadLow_449 RecordsBad450_499 RecordsBad500_549
                   RecordsBad550_599 RecordsBad600_649 RecordsBad650_699
                   RecordsBad700_749 RecordsBad750_799 RecordsBad800_849 RecordsBad850_899;
   retain roddar rgoodar rbadar;
   drop jvr;
   if first.BehaviorScoreRank then do jvr=1 to dim(roddar);
         roddar{jvr}=.; rgoodar{jvr}=.; rbadar{jvr}=.;
   end; else;
   do jvr=1 to dim(roddar);
      if jvr=AcrossDimension then do;
         roddar{jvr}=RecordsOdd; rgoodar{jvr}=RecordsGood; rbadar{jvr}=RecordsBad;
      end; else;
   end;
   if last.BehaviorScoreRank then output; else;
run;
```

3. Automation of the PIVOT Routine

Instead of creating a complete macro system, this section suggests tools the reader may adapt to build one.

3.1. Variable Names and Corresponding Labels

Consider the below variable names read in-stream into SAS data set *Split0*. The next DATA step splits these names to create labels in title case while producing split statistics. The reader could expand this routine to include variable names containing special symbols like dashes, underscores, and so on, to be used in the split criteria. Additional split criteria and coding may handle mono case variable names, though rendered less relevant by the 32 characters allowed by releases 7-9 of SAS® Software.

The %LET statement assigns the text of the SPLIT routine to global macro variable *SplitRoutine*, which the next two DATA steps call as *&SplitRoutine*.

```
  /*Routine 3.1*/
data rpt.Split0;
   length AnalysisVariable $ 21;
   input AnalysisVariable $;
   cards;
SumActualBalance
CBScore
BehaviorScore
BehaviorScoreACS
CBScoreRank
BehaviorScoreRank
CBScoreBreak
BehaviorScoreBreak
BehaviorScoreACSBreak
BScore
CBScore
BCBResult
;
```

Both the DATALINES statement and the CARDS statement achieve the same goal. Use either one.

```
  /*Routine 3.2. Analysis Label: The SPLIT Routine*/
%let SplitRoutine=%str(
   upltr='ABCDEFGHIJKLMNOPQRSTUVWXYZ';
   loltr='abcdefghijklmnopqrstuvwxyz';
   array ltr{21} $ 1 _temporary_;
   VariableSplits=0;
   do jj=1 to VariableLength;
      ltr{jj}=substr(AnalysisVariable,jj,1);
      select;
         when(jj=1) AnalysisLabel=ltr{jj};
         when(jj=2) do;
            if index(loltr,ltr{jj-1})>0 &
               index(upltr,ltr{jj})>0 then do;
               VariableSplits+1;
               AnalysisLabel=trim(AnalysisLabel)||' '||ltr{jj};
            end; else AnalysisLabel=trim(AnalysisLabel)||ltr{jj};
         end;
         otherwise do; /*jj>=3*/
            if index(loltr,ltr{jj-1})>0 &
               index(upltr,ltr{jj})>0 then do;
               VariableSplits+1;
               AnalysisLabel=trim(AnalysisLabel)||' '||ltr{jj};
            end;
            else do;
               AnalysisLabel=trim(AnalysisLabel)||ltr{jj};
               if index(upltr,ltr{jj-2})>0 &
                  index(upltr,ltr{jj-1})>0 &
                  index(loltr,ltr{jj})>0
               then do;
                  lblen=length(AnalysisLabel); VariableSplits+1;
                  AnalysisLabel=substr(AnalysisLabel,1,lblen-2)||
                                 ' '||substr(AnalysisLabel,lblen-1);
               end; else;
            end; /*else*/
         end; /*otherwise*/
      end; /*select*/
   end; /*do jj*/
   VariablePieces=VariableSplits+1;
   VariableLabelLength=length(AnalysisLabel);
                     );
```

```
data rpt.Split;
   set rpt.Split0;
   VariableLength=length(AnalysisVariable);
   length AnalysisLabel $ 25;
   drop jj upltr loltr lblen;
   &SplitRoutine
run;
```

```
proc print data=rpt.Split noobs;
   var AnalysisVariable AnalysisLabel;
   title "Output 3.1. Data Set rpt.Split (Partial)";
run;
title;
```

3.2. Mother-Son Matrix

Routine 3.4 reads a 10-observation, 4-variable data set, *ParmCBScoreBreak*, to create a 90-observation, 18-variable data set, *MotherSonMatrix*. For each observation read in, a DO loop creates several variables and assigns them a series of 9 data values. These values correspond to the 9 analysis variables. *MotherSonMatrix* makes is possible to easily manage variables and labels that the pivot routine needs.

The process of incorporating this DATA step into a macro program is relatively simple if the routine can avoid hard coding WHEN statements of the SELECT group. Three analysis variables require three WHEN statements, not nine. Macro Routine 3.3 offers a solution by building text for the DATA step.

```
   Output 3.1. Data Set rpt.Split (Partial)

AnalysisVariable         AnalysisLabel

SumActualBalance         Sum Actual Balance
CBScore                  CB Score
BehaviorScore            Behavior Score
BehaviorScoreACS         Behavior Score ACS
CBScoreRank              CB Score Rank
BehaviorScoreRank        Behavior Score Rank
CBScoreBreak             CB Score Break
BehaviorScoreBreak       Behavior Score Break
BehaviorScoreACSBreak    Behavior Score ACS Break
BScore                   B Score
CBScore                  CB Score
BCBResult                BCB Result
```

```
  /*Routine 3.3. Build text for WHEN statements*/
%macro selectxt(varlist,fmtlist,dlm=%str(~));
   %global whenstmt vars; %local jvar var fmt fmtass;
   %let jvar=1; %let vars=0; %let whenstmt=;
   %let var=%scan(&varlist,&jvar,&dlm); %let fmt=%scan(&fmtlist,&jvar,&dlm);
   %do %while("&var"^="");
      %if "&fmt"^="" %then %let fmtass=%str(AnalysisFormat="&fmt..";);
                     %else %let fmtass=;
      %let whenstmt=%str(&whenstmt when(&jvar) do; AnalysisVariable="&var"; &fmtass end;);
      %let jvar=%eval(&jvar+1); %let vars=%eval(&vars+1);
      %let var=%scan(&varlist,&jvar,&dlm); %let fmt=%scan(&fmtlist,&jvar,&dlm);
   %end;
%mend selectxt;
```

Supply tilde-delimited lists of analysis variables and their formats to positional macro variables *VARLIST* and *FMTLIST*. Because Odd-related variables carry no formats, the delimiter supplied by keyword macro variable *DLM* must not be a blank space. If routines 3.3 – 3.5 are not part of the same macro program, globalize macro variables *WHENSTMT* and *VARS*. Within the conditional DO WHILE loop, Routine 3.3 assigns variable names and their formats to local macro variables *VAR* and *FMT*, creates WHEN statements, and concatenates them into macro variable *WHENSTMT*. Macro variable *VARS* saves the total number of variables for possible future use.

What if the number of variables is too large to list in *VARLIST*? [2] Bikila bi Gwet (2003) suggests a combination of SQL dictionary tables and techniques of Routines 3.7/3.9.

```
  /*Test run Routine 3.3*/
%selectxt(RecordsOdd~RecordsGood~RecordsBad~
          SumActualBalanceOdd~SumActualBalanceGood~
          SumActualBalanceBad~SumAnnualProfitOdd~
          SumAnnualProfitGood~SumAnnualProfitBad,
          %str( ~comma9~comma9~ ~
               dollar12~dollar12~ ~dollar12~dollar12));
%put whenstmt=&whenstmt;
%selectxt(RecordsOdd~RecordsGood~RecordsBad,
          %str( ~comma9~comma9));
%put whenstmt=&whenstmt;
```

Note: Generalized routines can easily build any text for DATA steps. See [1] Bikila bi Gwet (2003) for details.

Routines 3.4 – 3.5 create the data set displayed in Output 3.2.

```
                   Output 3.2. Data Set rpt.MotherSonMatrix (Partial)

Mother
 Label   Son Name                     Label Expression

Low-449  RecordsOddLow_449            RecordsOddLow_449='Records Odd Low-449'
Low-449  RecordsGoodLow_449           RecordsGoodLow_449='Records Good Low-449'
Low-449  RecordsBadLow_449            RecordsBadLow_449='Records Bad Low-449'
```

```
  /*Routine 3.4. Mother-Son Matrix (Parameter Data Set)*/
data rpt.MotherSonMatrix;
   set rpt.ParmCBScoreBreak;
   drop jj upltr loltr lblen;
   /*Mother Name & Label*/
   MotherLabel=CBScoreBreak;
   MotherName=tranwrd(MotherLabel,"-","_"); /*Insert in a DO loop if many invalid symbols*/
   length AnalysisRank 8 AnalysisVariable $ 21 AnalysisLabel $ 37
          AnalysisFormat $ 10 SonName $ 28; /*21+7=28; 37=32+5*/
   label AnalysisVariable="Analysis Variable" AnalysisFormat="Analysis Format"
         AnalysisLabel="Analysis Label" SonName="Son Name"
         SonNameLength="Son Name Length" SonLabel="Son Label" SonLabelLength="Son Label Length";
   /*Son Names & Labels by Analysis Variable*/
   do AnalysisRank=1 to 9;
      select(AnalysisRank);
         when(1) do; AnalysisVariable="RecordsOdd"; end;
         when(2) do; AnalysisVariable="RecordsGood"; AnalysisFormat="comma9."; end;
         when(3) do; AnalysisVariable="RecordsBad"; AnalysisFormat="comma9."; end;
         when(4) do; AnalysisVariable="SumActualBalanceOdd"; end;
         when(5) do; AnalysisVariable="SumActualBalanceGood"; AnalysisFormat="dollar12."; end;
         when(6) do; AnalysisVariable="SumActualBalanceBad"; AnalysisFormat="dollar12."; end;
         when(7) do; AnalysisVariable="SumAnnualProfitOdd"; end;
         when(8) do; AnalysisVariable="SumAnnualProfitGood"; AnalysisFormat="dollar12."; end;
         when(9) do; AnalysisVariable="SumAnnualProfitBad"; AnalysisFormat="dollar12."; end;
         otherwise;
      end;
      VariableLength=length(AnalysisVariable);
      /*Analysis Label: The SPLIT Routine*/
      &SplitRoutine
      /*SonName & SonLabel*/
      SonName=trim(AnalysisVariable)||trim(MotherName);
      if length(SonName)>32 then /*Control SonName's length*/
      do jj=1 to 32 until(length(SonName)<32); SonName=substr(SonName,2); end; else;
      if substr(SonName,1,1)^='_' & index(upltr,upcase(substr(SonName,1,1)))=0
         then SonName='_'||trim(SonName); else;
      SonNameLength=length(SonName); SonLabel=trim(AnalysisLabel)||' '||trim(MotherLabel);
      SonLabelLength=length(SonLabel);
      output;
   end; /*do AnalysisRank*/
run;
```

```
  /*Routine 3.5. Create Variable LabelExpression*/
  /*Routine 3.5.1. Length for LabelExpression*/
data _null_;
   set rpt.MotherSonMatrix end=last;
   retain MaxLabelLength;
   MaxLabelLength=
      max(MaxLabelLength,
          sum(SonNameLength,SonLabelLength));
   /*Add 3 for the equal sign & 2 quotation marks*/
   if last then do;
      MaxLabelLength=MaxLabelLength+3;
      call symput('MaxLabelLength',
                  trim(left(MaxLabelLength)));
   end; else;
run;
```

```
  /*Routine 3.5.2. Create LabelExpression*/
data rpt.MotherSonMatrix;
   length CBScoreBreak $ 10 AcrossRank 8
          MotherLength 8 AcrossDimension 8
          MotherLabel $ 10 MotherName $ 10
          AnalysisRank 8 AnalysisVariable $ 21
          AnalysisLabel $ 37 AnalysisFormat $ 10
          SonName $ 28 SonNameLength 8
          SonLabel $ 48 SonLabelLength 8
          LabelExpression $ &MaxLabelLength;
   set rpt.MotherSonMatrix;
   label LabelExpression="Label Expression";
   LabelExpression=trim(SonName)||
                   "='"||trim(SonLabel)||"'";
run;
```

3.3. Building Text for the Straight SAS Pivot Routine

Combined with SAS macro facility, the Mother-Son Matrix enables to build text for the KEEP= option, the LABEL statement, the FORMAT statement, and the array definitions in Routine 2.7. Doing so will cover the remaining elements required for full automation of the pivot routine.

```
  /*Routine 3.6. Selected Attributes*/
data rpt.SelectedAttributes
    (drop=AnalysisVariable);
   set rpt.MotherSonMatrix
      (keep=AnalysisVariable SonName
            AnalysisFormat LabelExpression);
   where AnalysisVariable in
       ("RecordsOdd","RecordsGood","RecordsBad");
proc report data=rpt.SelectedAttributes nowd;
   column SonName AnalysisFormat LabelExpression;
   define SonName / 'Son Name' width=17 spacing=1;
   define AnalysisFormat /
         'Analysis/Format' width=8 spacing=1;
   define LabelExpression /
         'Label/Expression' width=41 spacing=1;
   title "Output 3.3. Data Set rpt.SelectedAttributes";
run; title;
```

Using the macro facility, Routine 3.7 concatenates the values of relevant variables in the *Selected Attributes* data set. Hence, the KEEP= option in Routine 2.7 becomes

```
(keep=PerformanceMonth
      MonthEnd BehaviorScoreRank
      BehaviorScoreBreak &SonNameStr);
```

The LABEL statement becomes

```
  label &LabelExpressionStr;
```

Routine 3.8 creates subsets of the *Selected Attributes* data set, and Routine 3.9 puts the concatenation DATA step of Routine 3.7 into a macro program for reuse. Macro program *CONCATESTR* creates text for the FORMAT statement and array definitions in pivot Routine 2.7.

Note that limited space of this paper dictated the creation of the *Selected Attributes* data set. Otherwise, routines 3.7 or 3.8 would have used the entire Mother-Son Matrix, which in itself, is a reduced version of reality.

```
            Output 3.3. Data Set rpt.SelectedAttributes


                  Analysis Label
Son Name          Format   Expression
RecordsOddLow_449          RecordsOddLow_449='Records Odd Low-449'
RecordsGoodLow_44 comma9.  RecordsGoodLow_449='Records Good Low-449'
RecordsBadLow_449 comma9.  RecordsBadLow_449='Records Bad Low-449'
                               • • •
RecordsOdd850_899          RecordsOdd850_899='Records Odd 850-899'
RecordsGood850_89 comma9.  RecordsGood850_899='Records Good 850-899'
RecordsBad850_899 comma9.  RecordsBad850_899='Records Bad 850-899'
```

```
  /*Routine 3.7. Concatenate Values of Variables SonName & LabelExpression*/
data _null_;
   set rpt.SelectedAttributes;
   if _n_=1 then do;
      call symput("SonNameStr",trim(SonName));
      call symput("LabelExpressionStr",trim(LabelExpression));
   end;
   else do;
      call symput("SonNameStr",symget("SonNameStr")||' '||trim(SonName));
      call symput("LabelExpressionStr",symget("LabelExpressionStr")||' '||trim(LabelExpression));
   end;
run;
%put SonNameStr=&SonNameStr; %put LabelExpressionStr=&LabelExpressionStr;
```

Partial Log

```
1193  %put LabelExpressionStr=&LabelExpressionStr;
LabelExpressionStr=RecordsOddLow_449='Records Odd Low-449' RecordsGoodLow_449='Records Good
Low-449' RecordsBadLow_449='Records Bad Low-449' RecordsOdd450_499='Records Odd 450-499'
RecordsGood450_499='Records Good 450-499' RecordsBad450_499='Records Bad 450-499'
```

```
  /*Routine 3.8. Parameter Data Sets: Building text for FORMAT statement and array definitions*/
data rpt.GoodBadVariables rpt.OddVariables rpt.GoodVariables rpt.BadVariables;
   set rpt.SelectedAttributes(keep=SonName AnalysisFormat);
   if index(SonName,'Good')>0|index(SonName,'Bad')>0
                              then output rpt.GoodBadVariables; else;
   if index(SonName,'Odd')>0  then output rpt.OddVariables;     else;
   if index(SonName,'Good')>0 then output rpt.GoodVariables;    else;
   if index(SonName,'Bad')>0  then output rpt.BadVariables;     else;
run;

  /*Routine 3.9. Text Concatenation Macro Program*/
%macro concatestr(inputds,varname,dlm=%str( ));
   %global &varname.Str;
   data _null_;
      set &inputds;
      if _n_=1 then call symput("&varname.Str",trim(&varname)); else
                    call symput("&varname.Str",symget("&varname.Str")||"&dlm"||trim(&varname));
   run;
%mend concatestr;
```

Note: If *&VARNAME* is a numeric variable (right-justified), make **left(&varname)** the argument of the TRIM function.

```
%concatestr(rpt.GoodBadVariables,SonName);
%put SonNameStr=&SonNameStr;
%concatestr(rpt.OddVariables,SonName);
%put SonNameStr=&SonNameStr;
%concatestr(rpt.GoodVariables,SonName);
%put SonNameStr=&SonNameStr;
%concatestr(rpt.BadVariables,SonName);
%put SonNameStr=&SonNameStr;
```

Note that ***%CONCATESTR*** could have produced the earlier concatenation, *LabelExpressionStr* as

```
%concatestr(rpt.SelectedAttributes,
            LabelExpression);
%put LabelExpressionStr=&LabelExpressionStr;
```

Partial Log

```
1215  %put SonNameStr=&SonNameStr;
SonNameStr=RecordsGoodLow_449 RecordsBadLow_449
RecordsGood450_499 RecordsBad450_499
RecordsGood500_549 RecordsBad500_549 RecordsGood550_599
RecordsBad550_599 RecordsGood600_649
RecordsBad600_649 RecordsGood650_699 RecordsBad650_699
RecordsGood700_749 RecordsBad700_749
RecordsGood750_799 RecordsBad750_799 RecordsGood800_849
RecordsBad800_849 RecordsGood850_899
RecordsBad850_899
1217  %put SonNameStr=&SonNameStr;
SonNameStr=RecordsOddLow_449 RecordsOdd450_499
RecordsOdd500_549 RecordsOdd550_599
RecordsOdd600_649 RecordsOdd650_699 RecordsOdd700_749
RecordsOdd750_799 RecordsOdd800_849
RecordsOdd850_899
```

Bibliography

[1] Bikila bi Gwet. *SAS® and VisualStat® Solutions to Banking and Credit Card Field Cases. Copyright © 2003 by Bikila bi Gwet* (to be published).

[2] Bikila bi Gwet. *Demand for Analysis-Ready Data Sets. An Introduction to Banking and Credit Card Analytics. Copyright © 2003 by Bikila bi Gwet* (to be published).

[3] *SAS® Programming II: Manipulating Data with the DATA Step Course Notes.* Copyright © 2002 by SAS Institute Inc., Cary, NC 27513, USA. ~ pp. 7-27, 7-46 – 7-49..

[4] SAS Institute Inc., *SAS® Procedures Guide, Version 8,* Cary, NC: SAS Institute Inc., 1999, 1729 pp. Pages 859 – 881, 884 – 886, 889, 890, 893 – 898, 908 – 916, 918 – 922, 946 – 988.

[5] SAS Institute Inc., *SAS Language Reference: Concepts*, Cary, NC: SAS Institute Inc., 1999. 554 pages.

[6] SAS Institute Inc., *SAS® Language Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999 (Dictionary.)