Demand for Analysis-Ready Data Sets. An Introduction to Banking and Credit Card Analytics

Research papers and training manuals often use data that are too clean to reflect reality. Projects may run behind schedule solely due to overwhelming data clean up. Managers sometimes request for assistance with business problems that are poorly defined from an analytical perspective. As a result and in addition to analytical challenges, it is common practice that data specialists must re-submit upgraded jobs that sometimes run for hours.

While offering original SAS® solutions to selected aspects of credit-related business problems, this case-driven paper provides a **comprehensive and flexible data and attribute management framework** for analytics departments, particularly in the banking and credit card industries. The case involves the creation of an **analysis-ready data set** for **score validation**, **joint odds analysis**, and **performance trend analysis** as part of risk and marketing analysis of credit card portfolios. Assuming advanced Base SAS® programmers to be the readers, the paper suggests flexible ways to automate the coding process with simple macro routines while covering validation requirements. Interested readers may download the latest version of the paper and related files from http://www.visualstat.com/bikila.

1. Database Types and Data Set Levels

Analysts normally start a project by accessing multiple databases, relational or not. Generally of considerable size and broad in scope, **source files** constitute the first data set level. They range from text files on magnetic tapes, to tables from known database management systems such as SQL Server® or Oracle®, through to the ideal – SAS® data sets.

The programmer typically *samples* one source file according to project specifications, and merges it with other source files to incorporate additional variables of interest. Next, the process may require further *sub-setting* through exclusions: a Portfolio Marketing Manager will not include in a promotional offer, customers with credit bureau scores beneath an arbitrary cut. At this stage, additional source-table joins may bring in class and analysis variables of choice, and the analyst-programmer eventually creates new variables, thus building the final **analysis-ready data set**.

Instead of multiple, memory eating and process intensive subsets, there should be only one analysis-ready data set per project or major part of a project. Well-designed Boolean variables enable to tag observations appropriately for easy sub-setting and grouping in higher-level data sets. Coupled with effective SAS® programming, careful variable selection and management ensure that the analysis-ready data set is seldom modified after completion for the sole purpose of meeting upgraded requests for analysis from internal or external clients.

Created in list form from analysis-ready data sets, **results data sets** hold for one or more analysis variables, as large a number of summary statistics as one can foresee the need for. These summary data sets group statistics by the values of one or more class variables. Results data sets serve as inputs for **reports data sets** and **actual reports**. While results data sets have readily available statistics for a wide range of reporting needs, reports data sets meet formatting requirements of specific reports to be deployed in a variety of systems. Using the analysis ready data set developed herein, [2] Bikila bi Gwet (2003) expands on results data sets, reports data sets, and actual reports.

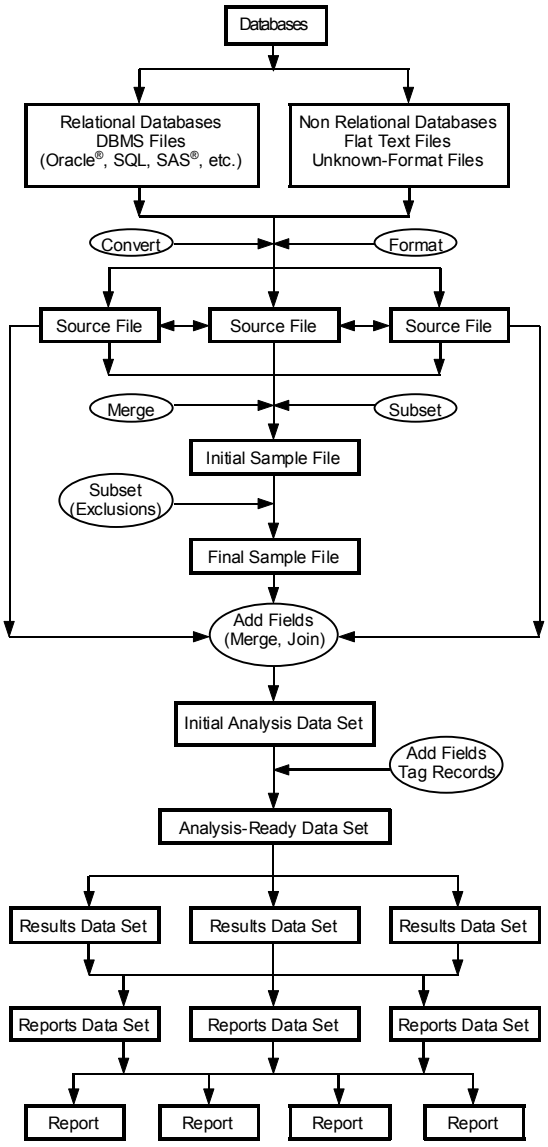

Fig. 1.1. Database Types & Data Set Levels

2. Assessment and Choice of Score Files

A fictive bank (the company) owns a fast-growing credit card portfolio that stores credit card customer data in two SQL Server® tables converted into SAS® data sets: *vstCustomerProfile* (≈ 300 columns & 55,000 rows) is a snapshot of cardholder data at month end and is overwritten every month; *vstCustomerTimeSeries* (≈ 200 columns & 750,000 rows) holds historical data since January 2002; and *vstSysPrinAgentInfo* is a small parameter table with bank system and principal agent information used to populate other source tables. It matches individual portfolios and agent information with the other tables through variable *VstSysPrinAgent*. Tables *vstCustomerProfile* and *vstCustomerTimeSeries* link through variable *CustomerID*.

Every quarter, the company sends a CD with personal information on its credit cardholders to a credit bureau to get fresh credit bureau scores. The warehouse team loads new credit bureau data to *vstCustomerTimeSeries* during the last month of each quarter (originally in *datetime22.3* format, variable *MonthEnd* = 31Mar2002, 30Jun2002, etc., for variable *DateLastCBScore* = a day in February 2002, May 2002, etc.) The warehouse manager informs the analyst that a large concentration of *DateLastCBScore*'s as of 15Feb2002, 15May2002, etc., indicates that the new scores have loaded.

```
title; options nodate nonumber ps=32700 formdlim=' ';
filename mgttxt  "C:\bbg\!Bikila Papers\SESUG03\Data Management Section";
libname  mgt     "C:\bbg\!Bikila Papers\SESUG03\Data Management Section";
libname  source  "C:\bbg\!Bikila Papers\dbSource";

  /*Routines 2.1. Convert & Reformat Date Values. Index Source Data Sets*/
options msglevel=i;
proc datasets lib=source nolist;
   modify vstSysPrinAgentInfo;
      index create VstSysPrinAgent;
   run;
quit;

data source.vstCustomerProfile
    (index=(CustomerID VstSysPrinAgent));
   set source.vstCustomerProfile
      (rename=(MonthEnd=iMonthEnd));
   MonthEnd=datepart(iMonthEnd);
   format MonthEnd date9.;
run;

data source.vstCustomerTimeSeries
    (index=(CustomerIDMonthEnd=(CustomerID MonthEnd) CustomerID MonthEnd
            MonthEndCustomerID=(MonthEnd CustomerID) VstSysPrinAgent));
   set source.vstCustomerTimeSeries
      (rename=(MonthEnd=iMonthEnd DateLastCBScore=iDateLastCBScore));
   MonthEnd=datepart(iMonthEnd); DateLastCBScore=datepart(iDateLastCBScore);
   format MonthEnd DateLastCBScore date9.;
run;

  /*Routine 2.2. Locate Concentration of Score Updates*/
proc freq data=source.vstCustomerTimeSeries;
   where (MonthEnd='31mar2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=02) or
         (MonthEnd='30jun2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=05) or
         (MonthEnd='30sep2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=08) or
         (MonthEnd='31dec2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=11) or
         (MonthEnd='31mar2003'd & year(DateLastCBScore)=2003 & month(DateLastCBScore)=02);
   tables MonthEnd*DateLastCBScore / noprint out=mgt.iScoreDates;
run;
```

[1] Bikila bi Gwet (2003) suggests additional code that enhances data set *iScoreDates* and saves it as *ScoreDates*, which Routine 2.3 displays (download *Demand Out.xls* from http://www.visualstat.com/bikila for details.) Further coding generates data set *ScoreFileTimeSpan* (Output 2.2.) An analysis-ready data set based on *ScoreFile28Feb2002* will have enough history for 12-month performance analysis and 12-month forward score migration analysis. On the other hand, *ScoreFile28Feb2003* is the basis for 12-month backward score migration analysis. Behavior scores and credit bureau scores from the three intermediary score files are necessary only for score migration analyses. This paper limits itself to the creation of the first analysis-ready data set, *ScoreSeries28Feb2002* for performance analysis only. Check its extensions to score migration analysis in [1] Bikila bi Gwet (2003.)

```
  /*Routine 2.3. Summary Report on Score Dates*/
proc report data=mgt.ScoreDates nowd headline nocenter;
   columns MonthEnd DateLastCBScore Frequency;
   define MonthEnd / group order=internal 'Month/End';
   define DateLastCBScore / group 'Date Last/CB Score'
                            width=9 format=monyy7.;
   define Frequency / 'Count';
   rbreak after / summarize ol;
   title1 'Output 2.1. Data Set mgt.ScoreDates';
   title2 '            Count of Score File Records';
run;
title;
```

Output 2.1. Data Set mgt.ScoreDates
Count of Score File Records

Month End	Date Last CB Score	Count
31MAR2002	FEB2002	38,140
30JUN2002	MAY2002	40,919
30SEP2002	AUG2002	41,163
31DEC2002	NOV2002	44,652
31MAR2003	FEB2003	47,235
		212,109

```
proc print data=mgt.ScoreFileTimeSpan(drop=FileNum FileRevNum Fwd2MigrMonth) noobs;
   title 'Output 2.2. Project Definitions. Data Set mgt.ScoreFileTimeSpan';
run; title;
```

Output 2.2. Project Definitions. Data Set mgt.ScoreFileTimeSpan

Bwd Migr Month	Fwd Migr Month	Perf Month	MonthEnd	ScoreDate	ScoreFile	Records
12	0	13	31Mar2002	28Feb2002	ScoreFile28Feb2002	38,140
9	3	10	30Jun2002	31May2002	ScoreFile31May2002	40,919
6	6	7	30Sep2002	31Aug2002	ScoreFile31Aug2002	41,163
3	9	4	31Dec2002	30Nov2002	ScoreFile30Nov2002	44,652
0	12	1	31Mar2003	28Feb2003	ScoreFile28Feb2003	47,235

3. Variables' Selection and Management

3.1. Source Data Sets' Variables

From source data sets, SQL dictionary tables help create SAS® files with selected attributes as data, which the analyst (Jane Doe) saves in spreadsheets *Demand Contents.xls*, downloadable from http://www.visualstat.com/bikila. To help manage variables, she adds the following fields to each spreadsheet: *TargetFile, Status, Purpose, AnalysisLevel*. If a variable from the first source file has more than one target file, then Jane inserts an additional row in the spreadsheet.

```
  /*Routine 3.1. Data Sets with Data and Variable Attributes as Data*/
proc sql;
   create table mgt.vstCustomerTimeSeriesVars as
   select varnum as VarNum, name as Variable, 'vstCustomerTimeSeries' as SourceFile,
          upcase(type) as Type, format as Format
      from dictionary.columns
         where libname='SOURCE' & memname='VSTCUSTOMERTIMESERIES';
   create table mgt.vstCustomerProfileVars as
   select varnum as VarNum, name as Variable, 'vstCustomerProfile' as SourceFile,
          upcase(type) as Type, format as Format
      from dictionary.columns
         where libname='SOURCE' & memname='VSTCUSTOMERPROFILE';
   create table mgt.vstSysPrinAgentInfoVars as
   select varnum as VarNum, name as Variable, 'vstSysPrinAgentInfo' as SourceFile,
          upcase(type) as Type, format as Format
      from dictionary.columns
         where libname='SOURCE' & memname='VSTSYSPRINAGENTINFO';
quit;
```

The analyst uses these spreadsheets to upgrade the process of building **analysis-ready score series**. There is a number of ways one can create SAS® data sets from these spreadsheets. Creating comma-delimited (or other) text files and reading them with the DATA step is one option that this section presents. [1] Bikila bi Gwet (2003) walks the reader through the process of establishing an open database connection (ODBC) between SAS® software and these spreadsheets. This process requires licensing SAS/ACCESS® to ODBC, and possibly re-installing your office suite with the custom-install option checked.

```
  /*Routines 3.2. Read Comma-Delimited Text Files*/
data mgt.vstCustomerTimeSeriesVars;
   length VarNum 8 Variable SourceFile $ 32 Type $ 4 Format $ 15
          TargetFile Status Purpose $ 32 AnalysisLevel 8;
   infile mgttxt('vstCustomerTimeSeriesVars.csv') dsd missover firstobs=3;
   input VarNum Variable $ SourceFile $ Type $ Format $ TargetFile $ Status $ Purpose $ AnalysisLevel;
data mgt.vstCustomerProfileVars;
   length VarNum 8 Variable SourceFile $ 32 Type $ 4 Format $ 15
          TargetFile Status Purpose $ 32 AnalysisLevel 8;
   infile mgttxt('vstCustomerProfileVars.csv') dsd missover firstobs=3;
   input VarNum Variable $ SourceFile $ Type $ Format $ TargetFile $ Status $ Purpose $ AnalysisLevel;
data mgt.vstSysPrinAgentInfoVars;
   length VarNum 8 Variable SourceFile $ 32 Type $ 4 Format $ 15
          TargetFile Status Purpose $ 32 AnalysisLevel 8;
   infile mgttxt('vstSysPrinAgentInfoVars.csv') dsd missover firstobs=3;
   input VarNum Variable $ SourceFile $ Type $ Format $ TargetFile $ Status $ Purpose $ AnalysisLevel;
data mgt.vstCalculatedVars;
   length VarNum 8 Variable SourceFile $ 32 Type $ 4 Format $ 15 TargetFile
          Status Purpose $ 32 AnalysisLevel 8 Coding $ 65 LabelExpression $ 60;
   infile mgttxt('vstCalculatedVars.csv') dsd missover firstobs=3;
   input VarNum Variable $ SourceFile $ Type $ Format $
         TargetFile $ Status $ Purpose $ AnalysisLevel $;
   if Variable='ScoreDate' then Coding='Assigned';
   else select(Variable);
      when('BalanceChargedOff') Coding='BalanceChargedOff=Balance-BalanceExclWriteOff;';
      when('BalanceChargedOffCredit')
            Coding='BalanceChargedOffCredit=BalanceExclFraud-BalanceExclWriteOff;';
      when('BalanceChargedOffFraud') Coding='BalanceChargedOffFraud=Balance-BalanceExclFraud;';
      when('LossLessRecoveries') Coding='LossLessRecoveries=BalanceChargedOff-Recoveries;';
      when('LossFraudLessRecoveries')
            Coding='LossFraudLessRecoveries=BalanceChargedOffFraud-Recoveries;';
      when('LossCreditLessRecoveries')
            Coding='LossCreditLessRecoveries=BalanceChargedOffCredit-Recoveries;';
      otherwise;
   end; /*else select*/
   LabelExpression=trim(Variable)||"='"||trim(Variable)||"'";
run;

options nocenter ls=256;
proc print data=mgt.vstCustomerTimeSeriesVars noobs; title "mgt.vstCustomerTimeSeriesVars"; run;
proc print data=mgt.vstCustomerProfileVars noobs; title "mgt.vstCustomerProfileVars"; run;
proc print data=mgt.vstSysPrinAgentInfoVars noobs; title "mgt.vstSysPrinAgentVars"; run;
proc print data=mgt.vstCalculatedVars noobs; title "mgt.vstCalculatedVars"; run;
title; options center ls=96;
```

3.2. Variables' Subsets

In a foreseeable future, the analyst anticipates using a considerable, yet limited portion of the hundreds of variables present in source tables in a specific manner at any given step of the project.

```
  /*Routines 3.3. Retained Variables & Subsets*/
data mgt.RetainedVariables(drop=VarNum);
   length VarNum 8 Variable SourceFile $ 32
          Type $ 4 Format $ 15 TargetFile
          Status Purpose $ 32 AnalysisLevel 8
          Coding $ 65 LabelExpression $ 60;
   set mgt.vstCustomerTimeSeriesVars
       mgt.vstCustomerProfileVars
       mgt.vstSysPrinAgentInfoVars
       mgt.vstCalculatedVars;
   if TargetFile^=' ';
proc sort data=mgt.RetainedVariables;
   by Variable;
data mgt.InitialScoreFileVariables;
   set mgt.RetainedVariables;
   where SourceFile="vstCustomerTimeSeries" &
         TargetFile in ("Matching","Score File");
data mgt.RvstSysPrinAgentInfoVariables;
   set mgt.RetainedVariables;
   where SourceFile="vstSysPrinAgentInfo";
data mgt.RvstCustomerProfileVariables;
   set mgt.RetainedVariables;
   where SourceFile="vstCustomerProfile";
data mgt.RetainedScoreFileVariables;
   set mgt.RetainedVariables;
   where TargetFile="Score File" &
         Status="Permanent";
run;
```

The following data set holds primarily time-sensitive variables that come from *vstCustomerTimeSeries* to populate the score file, thus forming a score series. While the data set KEEP= option must bring in all these variables when using the MERGE statement in a DATA step, it should leave out matching variables (*where Purpose='Matching'*) when joining data sets with the SQL procedure (see section on the creation of score series.) However, the score file must keep untouched, scores and other variables that do not change, or are arbitrarily kept constant over the performance period. After all, it is these scores that are available now, and that managers use to make business decisions that affect future performance. The only variables common to data sets *Time_Score Series Variables* and *Retained Score File Variables* are *CustomerID* and *MonthEnd* – the two matching fields required to merge these data sets. See [1] Bikila bi Gwet (2003) for more on this.

```
  /*Retained Variables & Subsets (Continued)*/
data mgt.Time_ScoreSeriesVariables;
   set mgt.RetainedVariables;
   where SourceFile="vstCustomerTimeSeries" &
         TargetFile="Score Series" &
         Status="Permanent";
data mgt.ScoreSeriesComputedVariables;
   set mgt.RetainedVariables;
   where SourceFile="Calculated" &
         TargetFile="Score Series";
data mgt.RetainedAnalysisVariables;
   set mgt.RetainedVariables;
   where index(upcase(Purpose),"ANALYSIS")>0;
proc sort data=mgt.RetainedAnalysisVariables;
   by AnalysisLevel Variable;
run;
```

4. Creation of Score Files

4.1. Useful Macro Routines

Given an input data set *&INPUTDS* and a variable name *&VARNAME*, Routine 4.1 concatenates the values of *&VARNAME*, separating them by delimiter *&DLM*. The resulting macro variable name is *&VARNAME.Str* by default. If the field name was *Variable*, the concatenated macro variable name would be *VariableStr*. Routine 4.2 merges each data set from list *&INDSNAMES* with transaction data set *&TRANSACTIONDS*, using macro routine *%MAINROUTINE*, which the programmer may change every time. The routine picks the name of the resulting data set from the same rank in list *&OUTDSNAMES*. No processing when a data set name is missing! Delimiter *&NAMEDLM* separates the data set names in each list.

```
  /*Routine 4.1. Text Builder*/
%macro concatestr(inputds,varname, dlm=%str( ),mactxtname=);
   %local mtxtn;
   %if &mactxtname^= %then %let mtxtn=&mactxtname; %else %let mtxtn=&varname.Str;
   %global &mtxtn;
   data _null_;
      set &inputds;
      if _n_=1 then call symput("&mtxtn",trim(&varname));
               else call symput("&mtxtn",symget("&mtxtn")||"&dlm"||trim(&varname));
   run;
%mend concatestr;
```

```
  /*Routine 4.2. AddVars Macro Routine*/
%macro addvars(indsnames,outdsnames,namedlm=%str( ),txtdlm=%str( ),
               inlib=mgt,outlib=mgt,VarsDataSet=mgt.RvstSysPrinAgentInfoVariables,
               varname=Variable,TransactionDS=source.vstSysPrinAgentInfo);
   %local jds indsn inds outdsn outds;  %global dss;
   /*Get variable names  & concatenate*/
   %concatestr(&VarsDataSet,&varname,dlm=%str(&txtdlm));
   /*Process score files*/
   %let jds=1; %let dss=0;
   %let indsn=%scan(&indsnames,&jds,&namedlm); %let inds=&inlib..&indsn;
   %let outdsn=%scan(&outdsnames,&jds,&namedlm); %let outds=&outlib..&outdsn;
   options msglevel=i;
   %do %while("&indsn"^="" & "&outdsn"^="");
      %put PROCESSING DATA SET &inds.. PLEASE WAIT!;
      %mainroutine
      %let jds=%eval(&jds+1); %let dss=%eval(&dss+1);
      %let indsn=%scan(&indsnames,&jds,&namedlm); %let inds=&inlib..&indsn;
      %let outdsn=%scan(&outdsnames,&jds,&namedlm); %let outds=&outlib..&outdsn;
   %end; /*%do %while*/
%mend addvars;
```

4.2. Initial Score Files

The number of observations in each score file resulting from Routine 4.3 equals that established in data set *mgt.ScoreDates* (Output 2.2.) Within the current variable management framework, SAS® macro facilty automatically supplies variable names to the KEEP= data set option as shown in the next section. Hence, pasted from a printout of data set *InitialScoreFileVariables*, variable names in the KEEP= option of Routine 4.3 could have been supplied by a macro variable.

```
  /*Routine 4.3. Initial Score Files. Variables' Source: Data Set InitialScoreFileVariables*/
data mgt.iScoreFile28Feb2002 mgt.iScoreFile31May2002
     mgt.iScoreFile31Aug2002 mgt.iScoreFile30Nov2002 mgt.iScoreFile28Feb2003;
   set source.vstCustomerTimeSeries
      (keep=AccountPurchasedTag AnnualProfit AttritionScore Balance BankruptcyScore
            BehaviorScore BehaviorScoreCustom CBScore CBScoreKey CustomerID DateLastCBScore
            ExternalStatus ExternalStatusReas MonthEnd OpenDate OpenDateOriginal
            Owned RPMScore RiskScore RiskScorePredictor TagWriteOffValThisME VstSysPrinAgent);
   format ScoreDate date9.;
   label ScoreDate="Score Date";
   select;
      when(MonthEnd='31mar2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=02) do;
         ScoreDate="28Feb2002"d; output mgt.iScoreFile28Feb2002;
      end;
      when(MonthEnd='30jun2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=05) do;
         ScoreDate="31May2002"d; output mgt.iScoreFile31May2002;
      end;
      when(MonthEnd='30sep2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=08) do;
         ScoreDate="31Aug2002"d; output mgt.iScoreFile31Aug2002;
      end;
      when(MonthEnd='31dec2002'd & year(DateLastCBScore)=2002 & month(DateLastCBScore)=11) do;
         ScoreDate="30Nov2002"d; output mgt.iScoreFile30Nov2002;
      end;
      when(MonthEnd='31mar2003'd & year(DateLastCBScore)=2003 & month(DateLastCBScore)=02) do;
         ScoreDate="28Feb2003"d; output mgt.iScoreFile28Feb2003;
      end;
      otherwise;
   end;
run;
```

4.3. Adding Variables from Other Source Data Sets

Routine 4.4 brings agent bank variables from data sets *vst Sys Prin Agent Info* into the score files. Retained *vst Sys Prin Agent Info* fields from data set *Rvst Sys Prin Agent Info Variables* constitute the source of variable names that feed the data set KEEP= option. In this option, include matching key *VstSysPrinAgent* when using DATA step MERGE statements, do not when using SQL joins.

```
  /*Routine 4.4. Add vstSysPrinAgentInfo Variables*/
%macro mainroutine;
   proc sort data=&inds; by VstSysPrinAgent; run;
   data &outds;
      merge &inds(in=inscorefile)
            &TransactionDS(keep=&VariableStr);
      by VstSysPrinAgent;
      if inscorefile;
   run;
%mend mainroutine;

  /*Run the macro*/
%addvars(iScoreFile28Feb2002 iScoreFile31May2002 iScoreFile31Aug2002 iScoreFile30Nov2002
         iScoreFile28Feb2003,ScoreFile28Feb2002 ScoreFile31May2002 ScoreFile31Aug2002
         ScoreFile30Nov2002 ScoreFile28Feb2003,inlib=mgt,outlib=mgt,
         VarsDataSet=mgt.RvstSysPrinAgentInfoVariables);
```

Routine 4.5 adds *vstCustomerProfile* variables to the score files. Data set *RvstCustomerProfileVariables* holds column names the SQL procedure needs. However, one must drop the matching key *CustomerID* before running the routine.

```
data work.TempRvstCustomerProfileVariables;
   set mgt.RvstCustomerProfileVariables;
   where Variable^='CustomerID';
run;
```

```
proc print noobs;
   var Variable Type Format SourceFile
       TargetFile Status Purpose;
   title "Data Set work.TempRvstCustomerProfileVariables";
run; title;
```

Data Set work.TempRvstCustomerProfileVariables

Variable	Type	Format	SourceFile	TargetFile	Status	Purpose
AccountConvertedTag	CHAR	$1.	vstCustomerProfile	Score File	Permanent	Subsetting
UserDefinedTag07	CHAR	$1.	vstCustomerProfile	Score File	Permanent	Subsetting

```
  /*Routine 4.5. Add vstCustomerProfile Variables*/
%macro mainroutine;
   proc sql;
      create table &outds as
         select a.*, &VariableStr
            from &inds a left join &TransactionDS b
               on a.CustomerID=b.CustomerID;
   quit;
%mend mainroutine;

  /*Run the macro*/
%addvars(ScoreFile28Feb2002 ScoreFile31May2002 ScoreFile31Aug2002 ScoreFile30Nov2002
         ScoreFile28Feb2003,&indsnames,namedlm=%str( ),txtdlm=%str(,),inlib=mgt,outlib=mgt,
         VarsDataSet=work.TempRvstCustomerProfileVariables,TransactionDS=source.vstCustomerProfile);
```

4.4. Score File Subsets by Exclusion

To be part of the analysis file, the credit card customer must come from a portfolio the company owns (*Owned='Y'*), should not be a valuation-related charge off (*TagWriteOffValThisME=0*), a corporate, business or test account, and should not be a fraud charge off [*not (ExternalStatus='Z' & ExternalStatusReas=88*).] In addition, the customer must have valid credit bureau and behavior scores (see DATA step of Routine 4.6.)

```
  /*Routine 4.6. Subset Score Files*/
%macro subsetsf(indsnames,outdsnames,dlm=%str( ),inlib=mgt,outlib=mgt);
   %local jds indsn inds outdsn outds;  %global dss;
   /*Process score files*/
   %let jds=1; %let dss=0;
   %let indsn=%scan(&indsnames,&jds,&dlm); %let inds=&inlib..&indsn;
   %let outdsn=%scan(&outdsnames,&jds,&dlm); %let outds=&outlib..&outdsn;
   options msglevel=i;
   %do %while("&indsn"^="" & "&outdsn"^="");
      %put PROCESSING DATA SET &inds.. PLEASE WAIT!;
      data &outds;
         set &inds;
         where upcase(Owned)='Y' &  not TagWriteOffValThisME &
               upcase(UserDefinedTag07) not in ('C','T') &
               upcase(RetailBusiness)^="B" & upcase(AgentGroupCode)^='TST' &
               not (upcase(ExternalStatus)='Z' & ExternalStatusReas=88) &
               400<=BehaviorScore<=999 & 250<=CBScore<=900;
      run;
      %let jds=%eval(&jds+1); %let dss=%eval(&dss+1);
      %let indsn=%scan(&indsnames,&jds,&dlm); %let inds=&inlib..&indsn;
      %let outdsn=%scan(&outdsnames,&jds,&dlm); %let outds=&outlib..&outdsn;
   %end; /*%do %while*/
%mend subsetsf;

  /*Run the macro*/
%subsetsf(ScoreFile28Feb2002 ScoreFile31May2002 ScoreFile31Aug2002 ScoreFile30Nov2002
          ScoreFile28Feb2003,&indsnames,dlm=%str( ),inlib=mgt,outlib=mgt);
```

5. Creation of Score Series

Initially generated from *ScoreFile28Feb2002*, *ScoreSeries28Feb2002* will be the analysis-ready data set for two projects: **1) Analysis of 12-Month Portfolio Performance by Score**, and **2) 12-Month Forward Score Migration Analysis**. On the other hand, *ScoreFile28Feb2003* is the starting point in creating analysis-ready data set *ScoreSeries28Feb2003* for a third project: **3) 12-Month Backward Score Migration Analysis** (recall Output 2.2.) In conjunction with *ScoreFile28Feb2002* and *ScoreFile28Feb2003,* the remaining three files provide data for 3-month, 6-month, and 9-month score migrations. This paper addresses issues that relate to the first project. Interested readers will find further discussions on the three projects in [1] Bikila bi Gwet (2003) and ensuing seminars.

5.1. Initial Score Series 28Feb2002

Merging *ScoreFile28Feb2002,* a cross-section, with *vstCustomerTimeSeries* yields only matching records for each value of *MonthEnd*. The outcome is the same for the DATA step MERGE with the IN= option and SQL LEFT JOIN (see [1] Bikila bi Gwet (2003) for an illustration with sample data.) However, performance analysis demands that we stick to all and only customers originally in the score file. Allowing inflows of new accounts or outflows of inactives or closed accounts would surely distort the relative importance of bad (or good) customers in the odds analysis. One solution is to first create a bogus score series through self concatenation of *ScoreFile28Feb2002* as many times as there are *MonthEnd*'s in the performance period, increasing variable *MonthEnd* by one month each time. Then matching the resulting *Score Series 28Feb2002* with *vstCustomerTimeSeries* by *MonthEnd* **and** *CustomerID* solves the problem.

Although Routine 4.6 removed them from score files, valuation charge offs may surface later in time series (see Routine 5.4.)

```
  /*Routine 5.1. Get last MonthEnd*/
data _null_;
   set source.vstCustomerTimeSeries nobs=totobs;
   lastobs=totobs;
   set source.vstCustomerTimeSeries point=lastobs;
   call symput('LastMonthEnd',trim(left(MonthEnd)));
   call symput('LastMonthEndChar',put(MonthEnd,date9.));
   stop;
run;
%put LastMonthEnd=&LastMonthEnd &LastMonthEndChar;
LastMonthEnd=15825 30APR2003
  /*Initial ScoreSeries. Step 1 of 2*/
%concatestr(mgt.RetainedScoreFileVariables,Variable);
%put VariableStr=&VariableStr;
```

```
  /*Routine 5.2. Bogus Score Series*/
data mgt.ScoreSeries28Feb2002;
   length MonthEnd ScoreDate 8;
   set mgt.ScoreFile28Feb2002(keep=&VariableStr);
   drop jdt endate totdates;
   retain MonthEnd totdates;
   endate=&LastMonthEnd;
   if _n_=1 then
      totdates=intck('month',MonthEnd,endate)+1;
      else;
   format MonthEnd date9.;
   do jdt=1 to totdates;
      if jdt>1 then
         MonthEnd=intnx('month',MonthEnd,1,'e');
         else;
      output;
   end;
run;
  /*Initial ScoreSeries. Step 2 of 2*/
proc sort data=mgt.ScoreSeries28Feb2002;
   by MonthEnd CustomerID;
run;
%concatestr(mgt.Time_ScoreSeriesVariables,
            Variable,mactxtname=TSVariableStr);
%put TSVariableStr=&TSVariableStr;
%concatestr(mgt.ScoreSeriesComputedVariables,
            Coding);
%put CodingStr=&CodingStr;
%concatestr(mgt.ScoreSeriesComputedVariables,
            LabelExpression);
%put LabelExpressionStr=&LabelExpressionStr;
```

```
  /*Routine 5.3. Actual Score Series*/
options msglevel=i;
data work.ScoreSeries28Feb2002
     work.ChargedOffVal28Feb2002
    (keep=MonthEnd CustomerID TagWriteOffValThisME);
   merge mgt.ScoreSeries28Feb2002
         (in=inscoreseries)
          source.vstCustomerTimeSeries
         (keep=&TSVariableStr);
   by MonthEnd CustomerID;
   if inscoreseries;
   label &LabelExpressionStr;
   &CodingStr
   format BalanceChargedOff BalanceChargedOffCredit
          BalanceChargedOffFraud LossLessRecoveries
          LossFraudLessRecoveries
          LossCreditLessRecoveries dollar25.2;
   if TagWriteOffValThisME then
      output work.ChargedOffVal28Feb2002; else
      output work.ScoreSeries28Feb2002;
run;
  /*Routine 5.4. Remove all valuation charge offs
    if ChargedOffVal28Feb2002 has observations*/
proc sql;
   create table mgt.ScoreSeries28Feb2002 as
   select *
      from work.ScoreSeries28Feb2002
         where CustomerID not in
            (select CustomerID
               from work.ChargedOffVal28Feb2002);
quit;
```

5.2. Final Score Series 28Feb2002 – The Analysis-Ready Data Set

The last step in the process of creating analysis-ready *ScoreSeries28Feb2002* consists of assigning each account a series of tags that help identify customers as good, bad or indeterminate. The tags the routine creates are flexible enough to allow for multiple goodness and badness definitions for an account. In addition to a flow chart, [1] Bikila bi Gwet (2003) suggests an exhaustive SQL-based validation process that ensures adequate delivery by Routine 5.5.

```
  /*Routine 5.5. Account Performance Tags*/
options msglevel=i;
data mgt.ScoreSeries28Feb2002;
   set mgt.ScoreSeries28Feb2002;
   by CustomerID MonthEnd;
   label FirstOpen="Open 28Feb2002 (1 or 0)" FirstFraud="Fraud Account on 28Feb2002 (1 or 0)"
         Current="Current 28Feb2002 (1 or 0)" _05Days="05 Days 28Feb2002 (1 or 0)"
         _05DaysEver="05 Days Ever 28Feb2002 (1 or 0)"
         _05DaysWorseEver="05 Days or Worse Ever 28Feb2002 (1 or 0)"
         _30Days="30 Days 28Feb2002 (1 or 0)" _30DaysEver="30 Days Ever 28Feb2002 (1 or 0)"
         _30DaysWorseEver="30 Days or Worse Ever 28Feb2002 (1 or 0)"
         _60Days="60 Days 28Feb2002 (1 or 0)" _60DaysEver="60 Days Ever 28Feb2002 (1 or 0)"
         _60DaysWorseEver="60 Days or Worse Ever 28Feb2002 (1 or 0)"
         _90Days="90 Days 28Feb2002 (1 or 0)" _90DaysEver="90 Days Ever 28Feb2002 (1 or 0)"
         _90DaysWorseEver="90 Days or Worse Ever 28Feb2002 (1 or 0)"
         ChargedOff="Charged Off 28Feb2002 (1 or 0)"
         ChargedOffCredit="Charged Off Credit 28Feb2002 (1 or 0)"
         ChargedOffFraud="Charged Off Fraud 28Feb2002 (1 or 0)";
```

```
   retain FirstOpen FirstFraud _05DaysEver _30DaysEver _60DaysEver _90DaysEver
          _05DaysWorseEver _30DaysWorseEver _60DaysWorseEver _90DaysWorseEver;
   Current=0; _05Days=0; _30Days=0; _60Days=0; _90Days=0;
   ChargedOff=0; ChargedOffCredit=0; ChargedOffFraud=0;
   if first.CustomerID then do;
      _05DaysEver=0; _30DaysEver=0; _60DaysEver=0; _90DaysEver=0;
      _05DaysWorseEver=0; _30DaysWorseEver=0; _60DaysWorseEver=0; _90DaysWorseEver=0;
      if MonthEnd<="31Mar2002"d & ExternalStatus=' ' then FirstOpen=1; else FirstOpen=0;
      if MonthEnd<="31Mar2002"d &
         ExternalStatus='Z' & ExternalStatusReas=88 then FirstFraud=1; else FirstFraud=0;
   end; else;
   if upcase(ExternalStatus)='Z' then do;
      ChargedOff=1;
      if ExternalStatusReas=88 then ChargedOffFraud=1;
      else do;
         ChargedOffCredit=1; _90DaysWorseEver=1; _60DaysWorseEver=1;
        _30DaysWorseEver=1; _05DaysWorseEver=1;
      end;
   end; else
   select(DelinquencyCycles);
      when(.);
      when(0) Current=1;
      when(1) do; _05Days=1; _05DaysEver=1; _05DaysWorseEver=1; end;
      when(2) do; _30Days=1; _30DaysEver=1; _30DaysWorseEver=1; _05DaysWorseEver=1; end;
      when(3) do;
         _60Days=1; _60DaysEver=1; _60DaysWorseEver=1; _30DaysWorseEver=1; _05DaysWorseEver=1;
      end;
      when(4) do;
         _90Days=1; _90DaysEver=1; _90DaysWorseEver=1;
         _60DaysWorseEver=1; _30DaysWorseEver=1; _05DaysWorseEver=1;
      end;
      otherwise do;
         if DelinquencyCycles>=5 & DelinquencyCycles=int(DelinquencyCycles) then do;
            _90DaysWorseEver=1; _60DaysWorseEver=1; _30DaysWorseEver=1; _05DaysWorseEver=1;
         end; else;
      end; /*otherwise*/
   end; /*select*/
run;
```

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. *Credit Card Portfolio Performance Assessment. A Case of Business Intelligence. Copyright © 2003 by Bikila bi Gwet* (to be published).

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

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

[5] SAS Institute Inc., *SAS® Procedures Guide, Version 8,* Cary, NC: SAS Institute Inc., 1999, 1729 pp.