Paper 10460 - 2016

Missing Values, They Are NOT Nothing

Justin Jia, TransUnion Canada, Burlington, Ontario, Canada

Amanda Lin, CIBC, Toronto, Ontario, Canada

ABSTRACT

When analyzing data with SAS®, we often encounter missing or null values in data. Missing values can arise from the availability, collectibility, or other issues with the data. They represent the imperfect nature of real data. Under most circumstances, we need to clean, filter, separate, impute, or investigate the missing values in data. These processes can take up a lot of time. For these reasons, missing values are usually unwelcome and need to be avoided in data analysis; however, there are two sides to every coin, and if we can think outside the box, we can exploit the negative features of missing values for positive uses. Sometimes, we can create and use missing values to achieve our particular goals in data manipulation and analysis. These approaches can make data analyses convenient and improve work efficiency for SAS programming. This kind of creative and critical thinking is the most valuable quality for data analysts. This paper uses real-world examples to demonstrate the creative uses of missing values in data analysis and SAS programming, and discusses the advantages and disadvantages of these methods and approaches. The illustrated methods and advanced programming skills can be used in a wide variety of data analysis and business analytics fields.

INTRODUCTION

In data analysis and business analytics, it is not unusual that we run into missing or null values for categorical or numeric variables. Missing values in data may result from the availability of source data or other issues during data collection and manipulation. They usually represent an imperfect feature of real data. Under most circumstances, they are of no use and do not make a contribution to data analyses. Even worse, data analysts need to invest time and effort to deal with them. Depending on the nature of analysis work and the characteristics of missing values, we handle them in different ways. For example, for categorical variables, we can filter out missing values or separate them as an additional category in data reporting. For continuous numeric variables, we impute the missing values in predictors with mean, median or mode values in predictive modeling, provided that the percentage of missing values is small. If the data contains a high percentage of missing values, it is difficult to use the data, and requires further investigation. Please note that in customer data analyses and business analytics, handling missing values not only requires time, attention and techniques, but also needs keen business insights. For these reasons, missing values are generally unwelcome and they need to be avoided in data analyses.

However, as there are two sides to every coin, everything has advantages and disadvantages. If we can think outside of the box, we can exploit the negative attributes for positive uses. Despite their negative aspects in data analysis, missing values are not absolutely useless. On some occasions, they can act in the opposite way. For example, in real data analyses, we can make use of missing values to flag, roll up, reshape and score data. Utilization of missing values not only enables us to realize our particular goals in data analysis, but it also brings convenience and conciseness in SAS coding and improves processing efficiency. This kind of creative and critical thinking is the most valuable quality for data analysts.

Therefore, missing values are not absolutely of no use in data analysis and business analytics if we have in-depth understanding of them and utilize them appropriately. This paper will present and illustrate how to create and utilize missing values in data analysis with SAS programming, which can make SAS coding concise and improve work efficiency as well. It will cover below aspects:

1) Flag data with missing values for better efficiency.
2) Summarize and roll up data.
3) Summarize and present zero rows in data.
4) Reshape data.

5) Weigh and score data in statistical modeling.
6) Special missing values.

We will use real world examples to show some creative ideas for using missing values in data analysis and SAS programming, and discuss the advantages and disadvantages of these methods. The illustrated methods and advanced programming skills can have important applications in a wide variety of analytic work fields. Please note that the advanced programming skills and analytical methods are not only restricted to SAS programming, they can be extended to data analysis using other analytical tools as well.

I. FLAG DATA WITH MISSING VALUES FOR BETTER EFFICIENCY

In this paper, we use a fake banking data set as our input data. As shown in Table 1, this data set has 2125 observations and 7 variables. The data is at account level, Customer_Key contains duplicates because a customer may have multiple financial products within or outside a financial institution.

Table 1. Partial printout of the input Banking data set.

Customer_Key	Province	Age	Tenure	FI	Prod_Type	Balance
1001	ON	87	17	4. Other	ISA	$118,414.38
1001	ON	87	15.9	2. Bank B	TFSA	$20,853.07
1002	ON	83	13.6	3. Bank C	ISA	$3.63
1003	ON	44	16	1. Bank A	ISA	$4.78
1004	ON	58	10.5	3. Bank C	ISA	$4.83
1005	BC	50	15.4	2. Bank B	ISA	$0.15
1005	BC	50	15	3. Bank C	TFSA	$68.40
1006	ON	72	14.9	3. Bank C	ISA	$0.01
1007	ON	48	14.1	3. Bank C	TFSA	$5,280.27
1008	BC	52	15.1	2. Bank B	ISA	$96,149.36
1009	ON	67	12.9	3. Bank C	ISA	$185,752.15
1009	ON	67	12.9	3. Bank C	TFSA	$20,869.02
1010	ON	63	15.9	3. Bank C	ISA	$23.63
1010	ON	63	14.9	4. Other	RSP_DEP	$6.47

Customer_Key: numeric, N8, unique identifier variable. There are 997 unique customer keys in the input data.

Province: character variable, $2, Canadian provinces.

Age: numeric, N8, customer age.

Tenure: numeric, N8, how long a customer with a bank.

FI: character variable, $10, the name of a financial institution.

Prod_Type: character variable, $10, the product type of a banking account. ISA = Savings Account, RSP_DEP = RSP Deposit, TFSA = TFSA Account, MTG=Mortgage Account, MF=Mutual Fund Account, CHQ=Chequing Account, GIC = GIC Account.

Balance: numeric with Dollar12.2 format, the balance of a banking account.

In financial industry, it is frequently needed to analyze and report customer data in terms of the product categories that each customer owns and the total product penetration etc. The number of unique product categories for each customer is usually called Product Use Count (PUC). Therefore we need to create numerous flags or indicators based on the conditions of one or more variables to profile customers. Below code gives an example of creating binary flags with a DATA step and then using PROC SQL to roll up the flagged data to customer level. Table 2 shows the partial print out of the created Profile data set.

```
***************************** I. Flag uses. ******************************;
data Flag;
set Banking;

if Prod_Type="CHQ"          then CHQ=1;
if Prod_Type="ISA"          then SAV=1;
if Prod_Type="GIC"          then GIC=1;
if Prod_Type="MF"           then MF=1;
if Prod_Type="MTG"          then MTG=1;
if Prod_Type="RSP_DEP"      then RSP=1;
if Prod_Type="TFSA"         then TFSA=1;


/*if Prod_Type="CHQ"        then CHQ=1;       else CHQ=0;*/
/*if Prod_Type="ISA"        then SAV=1;       else SAV=0;*/
/*if Prod_Type="GIC"        then GIC=1;       else GIC=0;*/
/*if Prod_Type="MF"         then MF=1;        else MF=0;*/
/*if Prod_Type="MTG"        then MTG=1;       else MTG=0;*/
/*if Prod_Type="RSP_DEP"    then RSP=1;       else RSP=0;*/
/*if Prod_Type="TFSA"       then TFSA=1;      else TFSA=0;*/
run;

proc sql;
create table Profile as
select Customer_Key,
       max(CHQ)             as CHQ,
       max(SAV)             as SAV,
       max(GIC)             as GIC,
       max(MF)              as MF,
       max(MTG)             as MTG,
       max(RSP)             as RSP,
       max(TFSA)            as TFSA,

       sum(calculated CHQ, calculated SAV,calculated GIC, calculated MF,
       calculated MTG, calculated RSP, calculated TFSA)   as PUC,
       calculated PUC/7     as Penetration format=percentn9.2

from Flag
group by Customer_Key;
quit;
```

Table 2. Partial printout of the created Profile data set.

Customer_Key	CHQ	SAV	GIC	MF	MTG	RSP	TFSA	PUC	Penetration
1001		1					1	2	28.57%
1002		1						1	14.29%
1003		1						1	14.29%
1004		1						1	14.29%
1005		1					1	2	28.57%
1006		1						1	14.29%
1007							1	1	14.29%
1008		1						1	14.29%
1009		1					1	2	28.57%
1010		1				1		2	28.57%

As shown above, to perform our task, we can create binary flags or indicators either with 1/missing or 1/0 to indicate True and False outcomes. However, flagging with 1 or missing has some advantages over using 1/0 values: 1) more concise SAS coding: flagging with 1 or 0 requires the ELSE clauses, which are mandatory for FALSE conditions. However, they are unnecessary when using 1 or missing values as indicators, because missing is the default value for a false condition in SAS. 2) If input data is huge with millions of records, flagging with 1 and missing values can improve the rollup efficiency as well, since SAS will ignore missing values by default. Therefore, using missing values in this case can improve both coding and processing efficiencies.

II. SUMMARIZE AND ROLL UP DATA.

In real work, data analysts often need to roll up data from a granular level to an aggregate level. When data contains multiple hierarchies and duplicates, summarizing data becomes bothersome and time consuming. For example, in the financial industry, a financial customer may have multiple banking accounts, while multiple customers may share a joint account. If we want to roll up the data from account level to customer level, we must count the distinct customers and summarize all the account balances. We need to take extreme cautions to avoid double counting of data.

In our case, the input data is at account level, therefore the Customer_Key is not unique because a financial customer may have multiple banking accounts. For business reporting, we need calculate and report the unique clients, total balances (also called total funds managed in financial industry), average balance per account and average funds managed per client for each province and tenure segment. To perform this task, the first step is to summarize the data from account level to customer level. Attributed to duplicates of Customer_Key, a lot of summarizing procedures such as PROC SUMMARY, PROC MEANS, PROC REPORT, PROC TABULATE fail to work since they cannot count the distinct customers. Under this circumstance, one solution is to use PROC SQL via its COUNT(DISTINC) function[1]. Below code shows how to use PROC SQL to rollup the data and calculate the required variables, then utilize PROC TABULATE to generate the desired tabular summary report. As shown in Table 3, the average funds managed per client is different from the average account balance due to double counting issues.

```
******************** II. Summarize and roll up data. ********************;
proc format fmtlib;
value Tenure
low-<5="1. < 5 years"
5-10="2. 5-10 years"
10<-high="3. > 10 years";

picture tabpct  (round )
low-<0='000000.00'(prefix='-$' multiplier=1)
0-high='000000.00'(prefix='$' multiplier=1);
run;

proc sql;
create table Rollup as
select Province,
    put(Tenure, Tenure.)                as  Ten_Seg,
    count(distinct Customer_Key)        as  Client,
    sum(Balance)                        as  Funds       format=dollar12.0,
    mean(Balance)                       as  Avg_Balance format=dollar12.0,
   calculated Funds/Calculated Client as Funds_Per_Client format=dollar12.0

from Banking
group by Province, Ten_Seg;
quit;
```

```
proc tabulate data=Rollup;
class Province Ten_Seg;
var Client Avg_Balance Funds Funds_Per_Client;
table Province=" ", Ten_Seg=" "*(
      Client="Client"*sum="#"*f=comma12.0
      Funds="Funds Managed"*sum="$"*f=dollar12.0
      Avg_Balance="Avg Balance"*sum="$"*f=dollar12.0
      Funds_Per_Client="Funds Managed Per Client"*sum="$"*f=dollar12.0)
      / misstext=" " row=float;
run;
```

Table 3. Output of the created summary report.

	1. < 5 years				2. 5-10 years				3. > 10 years			
	Client	Funds Managed	Avg Balance	Funds Managed Per Client	Client	Funds Managed	Avg Balance	Funds Managed Per Client	Client	Funds Managed	Avg Balance	Funds Managed Per Client
	#	$	$	$	#	$	$	$	#	$	$	$
AB	4	$2,812	$562	$703	23	$123,743	$2,946	$5,380	24	$918,178	$13,117	$38,257
BC	8	$86,733	$7,228	$10,842	46	$1,016,475	$12,396	$22,097	73	$4,036,000	$22,422	$55,288
MT					8	$294,905	$17,347	$36,863	6	$154,970	$11,069	$25,828
NS					4	$66,384	$6,035	$16,596	5	$134,198	$12,200	$26,840
ON	67	$497,086	$5,072	$7,419	318	$5,361,473	$10,232	$16,860	323	$16,692,149	$20,761	$51,678
PE					1	$531	$531	$531	4	$32,127	$3,570	$8,032
QC	3	$24,017	$8,006	$8,006	50	$471,002	$5,675	$9,420	54	$1,559,130	$9,806	$28,873

Though PROC SQL is able to perform our task, on some occasions, we would prefer to use other summarizing tools due to their distinctive features. For example, PROC REPORT or PROC TABULATE are preferred because they are convenient for creating enhanced external tabular reports (in Excel, HTML, PDF or RTF formats) along with SAS ODS facilities[2]. PROC MEANS or PROC SUMMARY may be preferred because they require less code and can group and analyze data for all the combinations of class variables in one step. Therefore they are better choices than PROC SQL. But, how can we avoid double counting of data if we choose to use these procedures?

A creative solution to the double counting problem is to create and use missing values. As shown below, we first sort the data by Province, Ten_Seg and Customer_Key, then in the following DATA step, we create a new numeric counter variable Client with a value of 1 if first.CustomerID=1, otherwise we set it to missing. Thus, we can apply PROC TABULATE on the detailed data without pre-rollup to create the desired summary report. The generated summary report is exactly the same as Table 3. Similarly, we can also use PROC MEANS to summarize the data. Therefore, we can count distinct IDs in this way through the use of missing values, because by default SAS will NOT process missing values in an analysis variable.

```
********************Utilize missing values to roll up. ******************;
data Seg ;
set Banking;
Ten_Seg=put(Tenure, Tenure.);
run;

proc sort data=Seg;
by Province Ten_Seg Customer_Key;
run;

data Tag;
set Seg;
by Province Ten_Seg Customer_Key;
if first.Customer_Key then Client=1;
else Client=. ;
run;
```

```
proc tabulate data=Tag ;
class Province Ten_Seg;
var Client Balance;
table Province=" ", Ten_Seg=" "*(
      Client="Client"*sum="#"*f=comma12.0
      Balance="Funds Managed"*sum="$"*f=dollar12.0
      Balance="Avg Balance"*mean="$"*f=dollar12.0
      Balance="Funds Managed Per Client"*PCTSUM<Client>="$"*f=tabpct.)
      /misstext=" " row=float;
run;
```

Another application example is that, in financial industry, we often need to analyze and report data (e.g., number of clients, total funds managed, average account balance, average funds managed per client, etc.) both inside and outside a financial institution. To accomplish this task, we can also make use of missing values to conveniently compute both on bank and off bank values. Below code illustrates the programming skills.

As shown below, suppose that we want to calculate the financial data inside and outside Bank A, we first create a binary indicator ON. Then we can use either PROC SQL or DATA step to do it. In the SQL method, we use the CASE WHEN clauses to perform conditional executions. For any false condition, we set the sum-up value to missing. SAS will ignore these missing values during summarization. It is important to note that, for SUM function, we can use either missing or zero values for the false conditions. They produce the same results. However, for MEAN function, we must use missing values instead of zero values. If zero values are used, the calculated mean values will be wrong because SAS will exclude missing values from analysis, but will include zero values into calculations. This is a distinct difference between them. Therefore, we can see that missing values are better than zero values to use. Table 4 illustrates the summary results of using SQL method.

```
******************** Compute On-, Off- Bank Balances. ********************;
data On_Off;
set Banking;
if FI="1. Bank A" then ON=1;
else ON=0;
run;


************************ Methond A: SQL Method. ******************;

proc sql;
create table Report as
select Province,
count(distinct case when ON=1 then Customer_Key else . end) as On_Client,
sum( case when ON =1 then Balance else . end)       as On_Funds,
mean(case when ON =1 then Balance else . end)       as Avg_On_Balance,
calculated On_Funds/calculated On_Client            as On_Funds_Per_Client,


count(distinct case when ON^=1 then Customer_Key else . end) as Off_Client,
sum( case when ON ^=1 then Balance else . end)      as Off_Funds,
mean(case when ON ^=1 then Balance else . end)      as Avg_Off_Balance,
calculated Off_Funds/calculated Off_Client    as Off_Funds_Per_Client,

count(distinct Customer_Key)                        as Client,
sum(Balance)                                        as Funds,
mean(Balance)                                       as Avg_Balance,
calculated Funds/calculated Client                  as Funds_Per_Client
```

```
from On_Off
group by Province;
quit;
```

Table 4. Summary Report of Funds Managed On and Off Bank A.

	On FI Clients	On FI Funds	Avg On FI Balance	On FI Funds Per Client	Off FI Clients	Off FI Funds	Avg Off FI Balance	Off FI Funds Per Client	All Clients	Total Funds Managed	Avg Account Balance	Total Funds Managed Per Client
	#	$	$	$	#	$	$	$	#	$	$	$
AB	22	$225,216	$8,662	$10,237	49	$819,517	$9,006	$16,725	51	$1,044,733	$8,929	$20,485
BC	49	$922,226	$14,410	$18,821	110	$4,216,982	$20,081	$38,336	123	$5,139,209	$18,756	$41,782
MT	4	$52,365	$10,473	$13,091	13	$397,509	$15,289	$30,578	14	$449,874	$14,512	$32,134
NS	2	$129,007	$25,801	$64,504	9	$71,575	$4,210	$7,953	9	$200,582	$9,117	$22,287
ON	264	$4,752,767	$15,282	$18,003	621	$17,797,941	$15,962	$28,660	691	$22,550,708	$15,814	$32,635
PE	4	$27,644	$5,529	$6,911	4	$5,014	$1,003	$1,253	5	$32,658	$3,266	$6,532
QC	43	$644,543	$11,936	$14,989	94	$1,409,607	$7,380	$14,996	104	$2,054,150	$8,384	$19,751

Similarly, we can also use DATA step to achieve our goal. As illustrated below, the input data is sorted and tagged based on the conditions of the ON indicator and first.Customer_Key. Then we use PROC TABULATE to summarize the data and generate the desired summary report. Table 4 demonstrates the created summary report, which is identical to the results computed by using SQL method. Again, we must use missing values rather than zero values for the ELSE clauses because we need to utilize the MEAN function.

```
********************* Method B: Data Step method. ***********************;
proc sort data=On_Off out=Sorted;
by Province Customer_Key;
run;

data Tag_Total;
set Sorted;
by Province Customer_Key;
if first.Customer_Key then Client=1;
else Client=.;
run;

proc sort data=Tag_Total out=Tag_Sorted;
by Province descending ON Customer_Key;
run;

data Tag_On_Off;
set Tag_Sorted;
by Province descending ON Customer_Key;

if ON=1 then do;
On_Balance=Balance;
Off_Balance= . ;
if first.Customer_Key then On_Client=1;
else On_Client=.;
end;

else do;
On_Balance= . ;
Off_Balance=Balance;
if first.Customer_Key then Off_Client=1;
else Off_Client=.;
end;
run;
```

```
proc tabulate data=Tag_On_Off;
class Province ON;
var  On_Client Off_Client On_Balance Off_Balance Client Balance;

table Province=" ", (On_Client="On FI Clients"*sum="#"*f=comma12.0
On_Balance="On FI Funds"*sum="$"*f=dollar12.0
On_Balance="Avg On FI Balance"*mean="$"*f=dollar12.0
On_Balance="On FI Funds Per Client"*PCTSUM<On_Client>="$"*f=tabpct.

Off_Client="Off FI Clients"*sum="#"*f=comma12.0
Off_Balance="Off FI Funds"*sum="$"*f=dollar12.0
Off_Balance="Avg Off FI Balance"*mean="$"*f=dollar12.0
Off_Balance="Off FI Funds Per Client"*PCTSUM<Off_Client>="$"*f=tabpct.

Client="All Clients"*sum="#"*f=comma12.0
Balance="Total Funds Managed"*sum="$"*f=dollar12.0
Balance="Avg Account Balance"*mean="$"*f=dollar12.0
Balance="Total Funds Managed Per
Client"*PCTSUM<Client>="$"*f=tabpct.)/misstext=" " row=float;
run;
```

III. SUMMARIZE AND PRESENT ZERO ROWS IN DATA.

When creating analytic reports with SAS, sometimes we need to summarize and present what is not in the data as well as what is in the data[3]. For example, when creating summary tables in the pharmaceutical or clinical trial industry, attributed to the collectability or other issues of data, some reporting tables like physical examinations, demographic characteristics, and some efficacy tables usually need to be partially made up in some way in the real practice world. For a complete reporting, SAS programmers thus need to summarize and present zero rows in input data. To accomplish this task, we can take advantage of missing values again.

For example, in our case, our data only contains six Canadian provinces. However, as everyone knows, Canada has 13 provinces and territories in total. For a complete reporting, we need include and show the seven remaining provinces and territories as well. For these zero rows in data, we can leave them as blank rows in a report. We can realize this goal through several different methods.

Prior to the task, we need to create a dimension data set called Complete which contains all the 13 Canadian provinces and territories. Then we can create a custom format named $Prov using this dimension data set.

The first approach is to use PROC MEANS to compute all the descriptive statistics of Balance and output the results to the Stat_A data set. Then we use the dimension data set Complete to left join the Stat_A table, consequently, all the statistics for the absent provinces will have missing values. Table 5 shows the complete summary report produced by method A, in which all the missing provinces and territories are included and presented as blank rows.

```
****************Summarize and present zero rows in data. ****************;
****************Create a dimension data set and custom format. *********;
data Complete;
input Province : $2. Region : & $20.  @@;
cards;
AB   1. Western Region  BC  1. Western Region  SK   1. Western Region
NS   2. Ocean Region    NB  2. Ocean Region    NL   2. Ocean Region
PE   2. Ocean Region    MT  3. Central Region  QC   3. Central Region
ON   3. Central Region  NU  4. Territories     NT   4. Territories
YK   4. Territories
;
run;
```

```
proc sort data=Complete;
by Province;  run;

data Complete;
set Complete;
Start=Province;
FMTName="$Prov";
Label=Province;
run;

proc format cntlin=Complete fmtlib;
run;

******************** Method A: Proc Means without Formats.***************;
proc means data=Banking noprint;
class Province;
var Balance;
output out=Stat_A N=N Nmiss=Nmiss Min=Min Max=Max Mean=Mean STD=STD
Median=Median;
run;

proc sql;
create table Method_A(drop=Prov) as
select a.Province,  b.*

from Complete a left join Stat_A(rename=(Province=Prov)) b
on a.Province=b.Prov;
quit;

proc print data=Method_A(drop=_Type_ _Freq_) noobs;
format Min Max Mean STD Median dollar12.0;
run;
```

Table 5. Summary Report Created By Method A

Province	N	Nmiss	Min	Max	Mean	STD	Median
AB	117	0	$0	$255,201	$8,929	$30,057	$424
BC	274	0	$0	$394,098	$18,756	$44,197	$2,917
MT	31	0	$0	$211,234	$14,512	$40,096	$541
NB							
NL							
NS	22	0	$0	$89,839	$9,117	$19,205	$1,974
NT							
NU							
ON	1,426	0	($277)	$552,492	$15,814	$45,476	$1,273
PE	10	0	$0	$19,462	$3,266	$6,070	$894
QC	245	0	$0	$196,123	$8,384	$21,265	$659
SK							
YK							

A second approach is to present zero rows through the use of the created custom format. As shown in Method B, we can use PROC MEANS with the COMPLETETYPES and PRELOADFMT options to summarize the data. The COMPLETETYPES option will instruct the procedure to create all possible combinations of the values of the classification variables, even if that combination does not exist in the input data set. This option must be used with the PRELOADFMT option for CLASS statement, which specifies that all formats are preloaded for the class variables. These two options are a must to present

the zero rows for a complete reporting. Table 6 shows the summary results, which are similar to those in Table 5 except that zero values rather than blanks are used for the N and NMISS columns.

In the same way, we can choose PROC TABULATE with PRELOADFMT and PRINTMISS options, or, PROC REPORT with COMPLETEROWS and PRELOADFMT options to accomplish our task. These two procedures are able to write out the results directly to external ODS destinations, and produce reports identical to Table 6.

```
************** Method B: Proc Means with Preloaded Formats. *************;

proc means data=Banking noprint completetypes;
format Province $Prov.;
class Province/missing preloadfmt;
var Balance;
output out=Stat_B(where=(_Type_ ^=0)) N=N Nmiss=Nmiss Min=Min Max=Max
Mean=Mean STD=STD Median=Median;
run;

proc print data=Stat_B(drop=_Type_ _Freq_) noobs;
format Min Max Mean STD Median dollar12.0;
run;

************* Method C: Proc Tabulate with Preloaded Formats. ***********;

proc tabulate data=Banking  out=Stat_C ;
format Province $Prov.;
class Province /missing preloadfmt;
var   Balance;
table Province=" ",
      Balance=" "*(N="N"*f=comma12.0
     Nmiss="Nmiss"*f=comma12.0
      Min="Min"*f=dollar12.0
      Max="Max"*f=dollar12.0
      Mean="Mean"*f=dollar12.0
      STD="STD"*f=dollar12.0
     Median="Median"*f=dollar12.0)/printmiss misstext=" " row=float;
run;

************* Method D: Proc Report with Preloaded Formats. *************;

proc report data=Banking out=Stat_D  completerows;
format Province $Prov.;
column Province Balance=N  Balance=Nmiss  Balance=Min
       Balance=Max Balance=Mean Balance=STD Balance=Median;

define Province/group "Province" missing preloadfmt;
define N/N            "N"         f=comma12.0;
define Nmiss/Nmiss    "Nmiss"     f=comma12.0;
define Min/min        "Min"       f=dollar12.0;
define Max/max        "Max"       f=dollar12.0;
define Mean/Mean      "Mean"      f=dollar12.0;
define STD/STD        "STD"       f=dollar12.0;
define Median/Median  "Median"    f=dollar12.0;
run;
```

Table 6. Summary Reports Created By Methods B, C and D.

Province	N	Nmiss	Min	Max	Mean	STD	Median
AB	117	0	$0	$255,201	$8,929	$30,057	$424
BC	274	0	$0	$394,098	$18,756	$44,197	$2,917
MT	31	0	$0	$211,234	$14,512	$40,096	$541
NB	0	0					
NL	0	0					
NS	22	0	$0	$89,839	$9,117	$19,205	$1,974
NT	0	0					
NU	0	0					
ON	1,426	0	($277)	$552,492	$15,814	$45,476	$1,273
PE	10	0	$0	$19,462	$3,266	$6,070	$894
QC	245	0	$0	$196,123	$8,384	$21,265	$659
SK	0	0					
YK	0	0					

IV. RESHAPE DATA

In many data analysis fields, data analysts frequently need to reshape data, to transpose data from long to wide, or to rotate data from wide to long. To transpose data in SAS, we can use either PROC TRANSPOSE or a DATA step array method to do it[4]. For example, if we want to transpose the Balance from vertical to horizontal, we can use both PROC TRANSPOSE and DATA step methods, which are shown below. In the DATA step method, we set all the array elements to missing at the start of each client by using the CALL MISSING routine. Please note that it is a necessity to set them to missing values, otherwise we cannot transform the data properly.

```
***************************** Reshape data. *****************************;
******************Method A: Proc Transpose Method. ******************;
proc sort data=Banking  out=Sorted;
by Customer_Key Prod_Type;
run;

proc transpose data=Sorted  out=Trans_A(drop=_Label_ _Name_);
by Customer_Key;
var Balance;
ID  Prod_Type;
run;

data Trans_A;
retain Customer_Key CHQ GIC ISA MF MTG RSP_DEP TFSA;
set Trans_A;
run;

******************Method B: DATA Step Array Method. ******************;
data Trans_B;
set Sorted;
by Customer_Key Prod_Type;

if Prod_Type="CHQ"                then I=1;
else if Prod_Type="GIC"           then I=2;
else if Prod_Type="ISA"           then I=3;
else if Prod_Type="MF"            then I=4;
else if Prod_Type="MTG"           then I=5;
else if Prod_Type="RSP_DEP"       then I=6;
else if Prod_Type="TFSA"          then I=7;
```

```
array Product(7) CHQ GIC ISA MF MTG RSP_DEP TFSA;
retain CHQ GIC ISA MF MTG RSP_DEP TFSA;

if first.Customer_Key then do;
call missing(of Product(*));
end;

Product(I)=Balance;

if last.Customer_Key then output;

keep Customer_Key CHQ GIC ISA MF MTG RSP_DEP TFSA;
format CHQ GIC ISA MF MTG RSP_DEP TFSA dollar12.2;
run;
```

The two methods presented above produce the same results, Trans_A and Trans_B are identical. Table 9 shows the partial printout of the two created data sets. Compared with PROC TRANSPOSE, an array method is much more flexible and versatile; therefore it is widely used in SAS data manipulation. Especially, when we have special needs and PROC TRANSPOSE fails to work, an array method is often the solution due to its flexibility.

Table 7. Partial printout of the created Trans_A data set.

Customer_Key	CHQ	GIC	ISA	MF	MTG	RSP_DEP	TFSA
1001			$118,414.38				$20,853.07
1002			$3.63				
1003			$4.78				
1004			$4.83				
1005			$0.15				$68.40
1006			$0.01				
1007							$5,280.27
1008			$96,149.36				
1009			$185,752.15				$20,869.02
1010			$23.63			$6.47	

V. WEIGH AND SCORE DATA IN STATISTICAL MODELING

In statistical modeling (regression, decision tree, neural network etc), it is a general practice to split the model development sample into training and validation data sets[5]. The training dataset is used to build the predictive model, while the intact validation set is for a cross validation purpose. This cross validation approach can identify and avoid over fitting of the model. Therefore, we need to use the parameter estimates obtained on the training dataset to score both the training and validation partitions.

To achieve this goal, usually we need to split a model development sample into Train and Validation data sets, then build model on Train data set and score the whole development sample by using the PROC SCORE procedure or PROC LOGISTIC/PROC GLM procedures along with a SCORE statement. As shown in below example, the Sample data set is derived from the Neuralgia data set at SAS Support website[6]. We first randomly select 70% of its records as Train dataset, and we use PROC LOGISTIC to construct the model and use SCORE statement to score the whole sample data[7]. Please note, we cannot use WHERE statement with PROC LOGISTIC to subset data because it will impact on both the input data for the PROC LOGISTIC procedure and on the scoring data for the SCORE statement.

However, an easy and simple alternative method is to use a WEIGHT statement to build the model and score the data. Method B illustrates this neat trick, which does not require splitting of data. In the given DATA step, we create a weight variable called Split_WT. For the Training observations in Sample data set, the weight variable is assigned a value of 1 (or equal to the sampling weight if the sample was

prepared by oversampling). For other observations, the weight variable is set to missing. Along with the use of a WEIGHT statement, then we can use the whole Sample data set as input data for the PROC LOGISTIC procedure. In performing statistical calculations, by default, SAS will exclude the observations with missing weights from the analysis. In this way, we can easily achieve our goal: fit model with training data only, but score the whole input data set with the fitted model. The scored data is output to the Scored_B dataset via OUTPUT statement, the predicted probability of each observation is exactly the same as that in Scored_A.

```
****************************** Score data. ****************************;
****************** Method A: Split sample. ****************************;
data Sample;
set  Neuralgia;
if ranuni(35678) <0.7 then Train=1;
else Train=0;
run;

data Train;
set Sample;
where Train=1;
run;

proc logistic data=Train outdesign=Design outmodel=OutModel outest=Effects;
class Treatment Sex;
model Pain(EVENT='Yes')= Treatment Sex Age Duration/STB details ;
score data=Sample  out=Scored_A   FITSTAT ;
run;


*************** Method B: Use weight without splitting sample.************;
data Sample;
set  Sample;
if Train=1 then Split_WT=1;
else Split_WT=. ;

/*if Train=1 then Split_WT=Sample_WT;*/
/*else Split_WT=. ;*/
run;

proc logistic data=Sample descending ;
weight Split_WT;
class Treatment Sex;
model Pain(EVENT='Yes')= Treatment Sex Age Duration /STB details;
output  out=Scored_B   predicted=Prob;
run;
```

VI. SPECIAL MISSING VALUES

Special missing values are a type of numeric missing values that enable us to represent different categories of missing data[8, 9]. In data analysis, sometimes we need to use special missing values to analyze and distinguish different types of missing data. Under these circumstances, the different categories of missing data have their own business meanings and provide useful information for our analytics. If we ignore them or combine them into one category, it will result in a loss of information or analytic bias.

There are totally 27 special missing values in SAS by using a period and the letters A-Z or an underscore, namely, ._, .A, .B, .C … .X, .Y, .Z. The sorting order of these special values is: ._ < .[standard missing] < .A < .B < …. <.Z, and they are all treated as FALSE in Boolean logic. We can use any of the 26 letters of the alphabet (uppercase or lowercase, not case sensitive) or the underscore (_), but only one single

letter. Please note that special missing values are only available for numeric variables. These special values are not only processed correctly as missing in DATA step and procedure calculations, but also distinguish among different types of missing data.

For example, in clinical trials, data censoring is a common problem. Data censoring refers to the missing data that occurs when participating patients fail to complete the study and drop out without further measurements. Handling these missing data is usually both challenging and complicated. There are different reasons for data censoring, such as death, adverse reactions, unpleasant study procedures, lack of improvement, early recovery, and other factors related or unrelated to trial procedure and treatments. These different categories of missing data are informative in nature, analysis of them may provide useful information for the clinical study and helps to reduce study bias[10]. If we ignore these missing values or treat them the same, we may lose important information for our study. Instead, we must record and characterize them with different indicators for further data analysis. In this circumstance, we must use special missing values to represent and analyze them.

As shown below, we use a Survey data set to illustrate the use of special missing values. In this survey, Age and Income are private and sensitive information. They may have missing values for different reasons, such as:

1) Invalid or out-of-range data. For example, zero, negative or extreme values (>200) for age.
2) Skip patterns.
3) The survey subject refuses to provide.
4) The interviewer forgot to ask.
5) Any other reasons.

These different categories of missing values have their own meanings, analyzing them helps to understand the validness of the study and reduce the bias of analytic results. We cannot ignore or treat them as all the same.

To represent special missing values in SAS, we must use the MISSING statement to read in data from raw text files or from CARDS and DATALINES. It will assign characters in input data to represent special missing values for numeric data. Please note that the MISSING statement is a global statement, it can appear anywhere in a SAS program. Table 8 shows the print out of the Survey data set.

```
********************** Special Missing Values.**************************;

data Survey;
infile datalines missover;
input Survey_ID  Age Income  @@;
format Income dollar12.0;
missing F I N R  ;
datalines;
1001 I  35000         1002 62 R         1003 36 42000     1004 F  R
1005 47 76300         1006 29 N         1007 R  48200     1008 55 63000
1009 43 F             1010 R  58600     1011 .  39200     1012 25 .
;
run;


proc format;
value Age
.="Skip pattern"            .I="Invalid"      .F="Forgot to ask"
.R="Refuse to provide"      low-<20="< 20"    20-40="20-40"
41-60="41-60"               61-High="> 60";
```

```
value Income
.="Skip pattern"            .F="Forgot to ask"      .N="Not applicable"
.R="Refuse to provide"      low-<40000="< $40K"     40000-60000="$40-60K"
60000<-high="> $60K";
run;

proc freq data=Survey;
format Age Age. Income Income. ;
tables Age Income /missing;
run;

proc means data=Survey N Nmiss Min Max Mean STD maxdec=1;
var Age Income;
run;
```

Table 8. Partial printout of the created Survey data set.

Survey_ID	Age	Income
1001	I	$35,000
1002	62	R
1003	36	$42,000
1004	F	R
1005	47	$76,300
1006	29	N
1007	R	$48,200
1008	55	$63,000
1009	43	F
1010	R	$58,600
1011		$39,200
1012	25	

As shown above, the missing data is represented by special missing values I, F R. These special missing values allow us to distinguish among different categories of missing data and perform calculations on them. If we apply PROC FREG (with MISSING option) and PROC MEANS on it along with the created custom formats, we will get below results.

Table 9. Frequency distribution of Age by using PROC FREQ.

Age	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Forgot to ask	1	8.33	1	8.33
Invalid	1	8.33	2	16.67
Refuse to provide	2	16.67	4	33.33
Skip pattern	1	8.33	5	41.67
20-40	3	25	8	66.67
41-60	3	25	11	91.67
> 60	1	8.33	12	100

Table 10. Output of PROC MEANS.

Variable	N	N Miss	Minimum	Maximum	Mean	Std Dev
Age	7	5	25	62	42.4	13.5
Income	7	5	35000	76300	51757.1	14843.2

It is important to note that the use of special missing values has distinctive advantages over numeric missing codes such as Forgot to ask= -9, Invalid= -8, Refuse to provide = -7 etc. These numeric missing codes will trigger below problems and result in incorrect analytic results.

1) SAS will NOT treat them as missing when performing calculations or analysis, instead, SAS will include them in numeric calculations.
2) They may be included accidentally in format ranges.

Therefore we must use special missing values as a solution.

Below code presents some tips of using special missing values. For example, if we want to use special missing values as filters to subset the data, the below DATA step gives a variety of choices. The MISSING function can detect all kinds of missing values, including both standard and special missing values. We can use the IF…THEN clause to test special missing values and create an indicator called Special. Please note, when we perform calculations and create new numeric variables, all the special missing values will be propagated as standard missing values. If we want to retain these special values, we need to use the second IF…THEN clause to create the new variable Net_Income. In this way, the special missing values will be retained in the derived Net_Income variable, otherwise they will be changed to standard missing values.

```
****************** Tips of using special missing values. ****************;

data Tips;
set  Survey;

/*where Age=.R;*/
/*where Age in (.I, .E);*/
/*where Age in (., .I, .E, .R);*/
/*where missing(Age);*/
/*where missing(Age) and Income=.R;*/
/*where missing(Age) and Age ^=.;*/
/*if    Age=.R;*/
/*where Age > ._;*/

if missing(Age) and Age ^=.       then  Special =1;
else if missing(Age)  and Age  =. then  Special =0;

if not missing(Income) then Net_Income=Income-9800;
else Net_Income=Income;
run;
```

A real work example of using special missing values is in statistical model vetting. As everyone knows, credit risk scores usually have a range from 300 to 900. We often need to perform risk score validation through decile analysis[11]. However, real input data usually contains invalid values which represent the No Hit or No Score reasons. For decile analysis and other score validations, we must exclude these No Hit or No Score records from analysis. However, for a complete reporting, we need to include and present them in separate tables. To accomplish this task, missing values provide a good solution again.

As shown below, the input Sample data (derived from the SAS course data Develop[5]) has a character variable Score (model score) and a numeric variable Bad (customer performance indicator). Score has values of No Hit, No Score or a number ranging from 300 to 900. In the beginning DATA step, we first create a new numeric score variable named Score_Num based on the Score variable. No Hit and No Score categories are represented by special missing values of .H and .S respectively. If we apply PROC FREQ along with the custom Score format on this variable, we will get the overall frequency distribution of model scores shown in Table 11.

```
**Work Example: Use of Special Missing Values in Model Score Validation.**;

data Prep;
set Sample;

if Score="No Hit"           then Score_Num=.H;
else if Score="No Score"    then Score_Num=.S;
else Score_Num=input(Score, 3.);
run;

proc format;
value Score
.H="No Hit"           .S="No Score"           0<-high="Scorable";

value Bad
0="Good"              1="Bad";

value Rank
.H="No Hit"           .S="No Score";
run;

proc freq data=Prep nlevels;
format Score_Num  Score. ;
tables Score_Num /missing;
run;
```

Table 11. Frequency distribution of model scores by using PROC FREQ.

Score_Num	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No Hit	551	6.66	551	6.66
No Score	80	0.97	631	7.63
Scorable	7642	92.37	8273	100

Then PROC RANK is used to rank the valid model scores and divide them into 10 score buckets. With the use of special missing values, PROC RANK will automatically exclude No Hit and No Score records from binning and set them as separate categories, because it does NOT rank missing values by default. Table 12 demonstrates the frequency distributions of the 10 score buckets along with the No Hit/ No Score categories obtained by using PROC FREQ procedure. If we do a two-way cross tabulation on Rank and Bad, but without the MISSING option for TABLES statement, it will produce a decile analysis table Table 13. We can see that No Hit and No Score records are excluded from the decile analysis table, and the highlighted numbers are the bad rate for each decile. It is worthy to note that, in this case, we cannot use numeric missing codes such as negative or zero values to represent the special missing categories, otherwise SAS will include them into the ranking calculations and create problems.

```
******************** Decile Analysis on Model Score. ********************;
proc rank data=Prep out=Ranked groups=10 ;
var  Score_Num;
ranks Rank;
run;

proc freq data=Ranked ;
format Rank Rank.;
tables Rank /missing;
tables Rank*Bad/nocol;
run;
```

Table 12. Frequency distribution of model score ranks.

Rank for Variable Score_Num				
Rank	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No Hit	551	6.66	551	6.66
No Score	80	0.97	631	7.63
0	759	9.17	1390	16.8
1	766	9.26	2156	26.06
2	775	9.37	2931	35.43
3	761	9.2	3692	44.63
4	769	9.3	4461	53.92
5	734	8.87	5195	62.79
6	766	9.26	5961	72.05
7	780	9.43	6741	81.48
8	764	9.23	7505	90.72
9	768	9.28	8273	100

Table 13. Decile Analysis of Model Score Performance.

Table of Rank by Bad				
		Bad	Good	Total
Rank(Rank for Variable Score_Num)				
0	Frequency	30	729	759
	Percent	0.39	9.54	9.93
	Row Pct	**3.95**	96.05	
1	Frequency	14	752	766
	Percent	0.18	9.84	10.02
	Row Pct	**1.83**	98.17	
2	Frequency	8	767	775
	Percent	0.1	10.04	10.14
	Row Pct	**1.03**	98.97	
3	Frequency	4	757	761
	Percent	0.05	9.91	9.96
	Row Pct	**0.53**	99.47	
4	Frequency	2	767	769
	Percent	0.03	10.04	10.06
	Row Pct	**0.26**	99.74	
5	Frequency	4	730	734
	Percent	0.05	9.55	9.6
	Row Pct	**0.54**	99.46	
6	Frequency	2	764	766
	Percent	0.03	10	10.02
	Row Pct	**0.26**	99.74	
7	Frequency	0	780	780
	Percent	0	10.21	10.21
	Row Pct	**0**	100	
8	Frequency	4	760	764
	Percent	0.05	9.95	10
	Row Pct	**0.52**	99.48	
9	Frequency	3	765	768
	Percent	0.04	10.01	10.05
	Row Pct	**0.39**	99.61	
Total	Frequency	71	7571	7642
	Percent	**0.93**	99.07	100
Frequency Missing = 631				

CONCLUSION

As demonstrated and discussed in this paper, despite their negative attributes in nature, missing values are NOT absolutely useless in data analysis and business analytics. If we have in-depth understanding of them and utilize them appropriately, they can bring significant benefits to our data manipulation and data analysis with SAS programming. The illustrated creative methods and advanced programming skills can be used in a wide variety of data analysis and business analytics fields.

REFERENCES

[1] SAS Support Website, http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a002473699.htm

[2] SAS Support Website, http://support.sas.com/documentation/cdl/en/proc/61895/HTML/default/viewer.htm#a000146759.htm

[3] Stacey D. Phillips, Gary Klein, "Oh No, a Zero Row: 5 Ways to Summarize Absolutely Nothing", Paper CC22, Proceedings of the Pharmaceutical SAS User Group Conference, Phoenix, AZ, May 2005.

[4] SAS Support Website, http://support.sas.com/documentation/cdl/en/proc/65145/HTML/default/viewer.htm#p1r2tjnp8ewe3sn1acnpnrs3xbad.htm

[5] "Predictive Modeling Using Logistic Regression", SAS Institute, ISBN-13: 978-9993159780, 2003.

[6] SAS Support Website, https://support.sas.com/documentation/cdl/en/statug/63347/HTML/default/viewer.htm#statug_logistic_sect060.htm

[7] SAS Support Website, https://support.sas.com/documentation/cdl/en/statug/63347/HTML/default/viewer.htm#logistic_toc.htm

[8] SAS Support Website, http://support.sas.com/documentation/cdl/en/lrcon/62955/HTML/default/viewer.htm#a000992455.htm

[9] Quentin McMullen , "How to Represent Missing Data: Special Missing Values vs. 999999999", NESUG Proceedings, PS8009.

[10] W.J. Shih, "Problems in dealing with missing data and informative censoring in clinical trials", Curr Control Trials Cardiovasc Med. 2002; 3(1): 4.

[11] Naeem Siddiqi, "Credit Risk Scorecards: Developing and Implementing Intelligent Credit Scoring", ISBN: 978-0-471-75451-0, SAS Institute, 2005.

ACKNOWLEDGEMENTS

Special thanks to Ethane Miller (University of California, Berkeley) for his valuable mentoring and suggestions during reviewing and revising this paper. We are very grateful to him for the help and support. Thank you so much.

CONTACT INFORMATION

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

Justin Jia	TransUnion Canada, Burlington, Ontario, Canada
Amanda Lin	CIBC, Toronto, Ontario, Canada