Paper 56-25

Avoiding Mayhem in the New Millennium: Working with Missing Data

JoAnn Matthews, Highmark Blue Cross/Blue Shield, Pittsburgh PA

ABSTRACT

Anyone who has ever worked with data is familiar with the adage "data are messy". And whether you have collected the data yourself, or inherit a dataset, there is usually missing data. This paper will discuss the issue of missing data and the techniques for identifying, processing, and manipulating missing data.

Topics to be discussed will include how SAS® processes data in the program data vector, the best methods to input data that may contain blank or missing values, and what to do when an invalid or missing data error message is encountered in the log. How missing variables are coded and how to differentiate missing numeric data with the MISSING statement in the data step will be explored. How SAS® sorts missing values will be examined. Techniques such as the difference between the MISSOVER option in the input statement and the MISSING= SAS® system option, the efficiency of the RETAIN statement and the use of the SUM function vs. mathematic calculations will be explored. Considerations when doing IF THEN/ELSE statements with missing data will also be discussed. Ways to identify missing data using PROC FREQ, PROC UNIVARIATE and PROC MEANS with the NMISS option will be compared. And lastly, avoiding propagation of missing values in calculations will be explained.

INTRODUCTION

Whether you are using a permanent SAS® dataset, a flat ascii file, or any other type of data, in most real life instances, there is usually missing data. Understanding how SAS® processes data through the Program Data Vector is helpful. This will shed light on why missing data can be a challenge. Knowing what kind of input statement to use when writing the input statement for data that contain an unknown amount of missing observations is an important consideration to ensure that your results are what you would expect. Using a MISSOVER statement in your list statement can be very valuable. What to do when you encounter the ever present "INVALID OR MISSING DATA ERROR MESSAGE" is another important issue. Knowing why this happens and when to be concerned about this message is very helpful.

Whether you are going to analyze your data or perform some mathematical calculations on the dataset you are working with, knowing how many of your observations are missing is important. When your data set is very small, simply eyeballing the data can answer any concerns you may have about missing data. But when you are working with huge datasets, i.e. several million observations, eyeballing the data is simply not possible. There are methods to identify how much of your data is missing. Subsetting your data with an IF statement or a WHERE statement, then issuing a PROC PRINT can be very useful if the dataset is not too large. If, however, your dataset if unwieldy or very large, then using PROC FREQ, PROC UNIVARIATE or PROC MEANS will help you to identify missing data. Knowing this information will be helpful to determine if you are can simply "do the math" of if you need to use math functions. Math functions have features to calculate averages, sums, even when there is a large number of missing observations.

MISSING DATA AND THE DATA STEP

How do we define missing data? If you look up missing in the SAS® Language Manual Glossary, you will read the following definition:

a missing value is defined as an incomplete data set. In Input use a period or blank as a placeholder for missing values of character variables, and use a period, a blank or a special missing character (assigned with the MISS = option) as a placeholder for numeric variables. The SAS system displays a blank to represent a missing value for a character variable and a period or a special character to represent a missing value for a numeric variable.

SAS® programs contain data steps and procedures or "procs" for short. A procedure is a software tool that is a collection of statements that perform a specific action, i.e. PROC SORT – sorts the data, PROC

PRINT – prints the data, or PROC CHART – charts the data, to name a few. A dataset reads in observations (rows) and variables (columns). Or another way to look at it, is a dataset consists of a group of statements in the SAS® Language that reads raw data or existing permanent SAS® datasets to create a file. A SAS® data set created in a data step is called a SAS® data file. A data file is a data set in which data values are physically stored. With the Data Step you read raw data or other forms of data and perform calculations and manipulations to analyze data or create reports.

HOW SAS PROCESSES DATA

When we run a program, SAS® creates temporary work space out on the Program Data Vector or "PDV". The PDV is a temporary area of memory, or storage area where the SAS® system builds a SAS® Data Set, one observation at a time. An observation is a row in a dataset that contains specific data values. In order to understand how missing values need to be handled, it is important to first understand how SAS® processes data. First, data must be in a form SAS® can recognize: SAS® Data sets contain

* Descriptor Information

* Data Values

The descriptor information describes the contents of a data set. Data Values are observations that have been collected and organized in a rectangular structure - ROWS (observations) and COLUMNS (variables).

Rows	Columns Var1	Var2	Var3
Obs 1	--	--	--
Obs 2	--	--	--
Obs 3	--	--	--

DATA STEP PROCESSING

The Data Step is used for

- Retrieval (getting data into data set)
- Editing (checking for errors and correcting data)
- Computing (making new variables)
- Printing
- Writing to Disk or Tape
- Producing new Data Sets (subsetting, Set, Merge Update)
- Data Manipulation (rearranging data for use in a Procedure)

A typical SAS® Data Step:

1.Compile – The SAS® system checks syntax and creates input

2. Program Data Vector creates temporary work space

3.Execution - Data Sep counts iterations, sets values to missing

SAS Reads data

Is there a record?
NO? If no data – Closes
Goes to next step

Is there a record?
YES? Reads Input Record
Executes Additional Statements
Writes OBS to Data Set
Returns to Data Statement

Compilation Phase

1. Submits Data Step
   – SAS® checks syntax
   – compiles i.e. translates into machine code

2. Creates
   - Input Buffer
   - Area of memory where raw data are read
   - Program Data Vector
   - Are of memory where SAS® builds data set

3)Data value read, from input buffer
   - Assigned to appropriate variables
   - creates Descriptor information
   Information SAS® creates and maintains about dataset

INPUT BUFFER
----+----1----+----2----+----3----+----4---+----5----+----

Program Data Vector -the variable set to missing – SAS® *creates automatic variables*

ID	Var1	Var2	Loss
001	.	.	.

SAS® *creates automatic variables _n__ = 1 and _error_ = 0 and decided what variables to drop (d) and what variables to keep (k)*

k d d d

1st data line read:

001 13 14 1
*_n_* = 1 *_ERROR_* = 0

k d d k

Keep = observations into dataset
01 1

back to next line of data

SAS® sets all observations to missing in Program Data Vector

data reset in PDV to missing

input buffer
---+---1---+---2---+---3---+---4---+---5---+---6

next line
002 . . .

ALL VARIABLES IN PDV are initially set to missing

When the data are set to missing depends on source of input data, i.e.whether from a raw data file or SAS® Data Set

From Raw Data:

When the dataset is from raw data, the SAS® System sets the value to missing with the exception of

1) Variables in RETAIN statements – we will discuss this further in the paper

2) Variables in SUM statements

3) Data elements in *temporary* array

4) Any variable created with options in the File or Infile statement

5) Automatic variables

Data from SAS Data Set:

When the data are from a SET, MERGE or UPDATE statement, the SAS® System sets values to missing only before the first iteration of the Data Step. Variables retain their value until new values become available

Variable values are also set to missing when the BY grouping changes. When the By grouping variable changes, the variables are set to missing and remain missing until the data set has no more observations to process.

MISSING DATA

If the data are incomplete due to missing values, the data still has a rectangular shape because the absent values are recorded as missing. Missing numeric variables are represented by a period "." and missing character variables are represented by a blank space. In the following dataset – VarB is a character variable , VarA and VarC are numeric. Notice that for alpha characters can be blank but missing numeric variables should be represented by a '.' – period.

		Var A	Var B	Var C	
OBS	1	1	Aa	2	2
	2	Bb	1		
	3	2		3	
	4	2	Cc	.	

THE MILLENNIUM WEIGHT LOSS STUDY

Let's create a Millennium data set for weight loss for a group of Disney characters and see how this missing data would be handled in the SAS system.

We have four participants in our weight loss dataset – Mickey Mouse, Daffy Duck, Quackie Jack Duck and Donald Trump Duck. We will assign identification numbers (ID) to each of them for study purposes. Mickey will be 001, Daffy will be 002, Quacky will be 003 and Donald will be 004. We will weigh them for baseline weight and then again in six weeks to access how they did on the weight loss program. Our data will look like the following:

NAME:	ID	Wt1	Wt2
Mickey Mouse	001	13	14
Daffy Duck	002	11	missing
Quacky Jack Duck	003	17	15
Donald Trump Duck	004	16	19

As you can see from the data we collected, no data were collected for Daffy Duck for Wt2. Donald Trump Duck, due to his excessive lifestyle, not only did not decrease his weight on our Millennium weight loss program, but rather had some serious indulgences and gained 3 ounces.

Before we can analyze the weight loss, we need to get the data into a format that SAS® can read. Therefore, we need to create a SAS® program using data step logic. The first thing we will need to do is write an input statement. The input statement tells SAS® what the name of our variables will be.

The simplest form of data input is a cards statement where we read the raw data in the data stream, following a CARDS statement – directly into the program within the data step.

Our data step will look like the following:

```
DATA DISNEY;
INPUT ID WT1 WT2;
LOSS = WT2 – WT1;
CARDS;
001     13      14
002     11      .
003     17      15
004     16      19
;
RUN;
```

Notice that we input Daffy Duck's missing weight by the use of a period. This is because Wt2 variable is a numeric and SAS expects missing numeric data to be represented by a period.

Let's observe how SAS® processes this data.

First SAS® sets the values to missing before the first iteration of the data step;

Id	wt1	wt2
.	.	.

SAS® reads the raw data into the input buffer

001	13	14

SAS® then decides which values it needs to keep to calculate loss, then calculates loss variable, deletes raw values 13 and 14 and retains ID and loss – in this case 001 and 1.

Automatic variables *_n_* and *_error_* are created and observations 001 and 01 are placed in the data set.
All variables are set to missing in the PDV, automatic variables are created and SAS® begins the data step processing again, this time capturing 002 . 11, retains id and loss. In this case, loss is missing because one of the values in the calculation was missing, 002 and missing are retained in the data set, all variables are reset to missing and SAS® goes to the next line of data. If there were not a notation for a missing value, SAS® would go to the next line to read this value. The results would not be what we expected if this were to happen.

INPUTTING WITH MISSING DATA

LIST INPUT

Deciding how to read your data into SAS® is important when there is missing data. To use LIST input, all missing data must be represented by a period. Using LIST input also requires that your data be separated by one blank or delimiter. Here is an example of LIST input.

```
DATA DISNEY;
INPUT NAME ID WT1 WT2;
```

Note that the rules to use LIST input include the following:

- Data must be separated by one blank or default delimiter
- MISSING values must be represented by periods
- Character must be 8 bytes with no embedded blanks
- Fields must be read in order
- Data must be standard numeric or character

If you use LIST input and you have missing variables that are not represented by a "period" – SAS® will go to the next line. For this example, if we were to leave 002's WT2 variable blank, SAS® would go to 003 for that value. Instead of reading 4 observations and 3 variables, we would not get an error message in the log, but we would only see 3 observations, and data that are not representative of our sample. The output would be wrong. If we are intimately

knowledgeable about our sample, this may not pose a problem. But if, more often that not, we are not intimately knowledgeable about our data, and especially if there are thousands, or even millions or records, our results would not be correct.

MISSOVER STATEMENT WITH LIST INPUT

By using the MISSOVER statement with your infile statement, you can prevent SAS® from going to a new line when it encounters a missing value in the current line.

```
DATA DISNEY;
INFILE CARDS MISSOVER;
INPUT var1 var2 …
```

MISSOVER in the infile statement tells SAS® to not go to the 2nd line when it encounters missing values.

REPRESENTING SPECIAL MISSING VALUES

Thus far, it has been demonstrated that you can represent missing values on data lines with either blanks, a single period, or if you prefer missing data can be identified with '9', '99', '9999'. There is another option for missing values; you can identify missing values as special missing values with a MISSING statement.

If , for example, in our sample millennium weight loss study Daffy Duck's weight was missing because he refused to step on the scale, we might want to represent his weight with as "R", for "refused", rather than a period. On the other hand, if his data was not collected because he was simply absent for the weigh-in, we might want to represent that missing weight with an "a". We can identify special characters that we would like SAS® to use for missing with a MISSING option statement in our data step by using the MISSING System Option.

This is how the code would look:

```
DATA DISNEY;
        MISSING R A;
        INPUT ID 3. WT1 2. WT2 2.;
        LOSS = WT2 – WT1;
        CARDS;
        001     13      14
        002     11      R
        003     17      15
        004     16      19
        ;
        RUN;
```

SAS® will interpret an "R" or an "A" as special missing values. You would not get an error message in the Log when SAS® encounters this value. Otherwise, without the missing option you would get an error message – because you are using a character value for a numeric variable. MISSING numeric data can be identified with upper case or lower case A-Z.

INVALIDDATA

In addition to blanks or period and special missing values, you may find that numeric values contain illegal characters such as letters or underscores not identified in a MISSING statement. You can use the INVALIDDATA option to force SAS® to assign a special missing value instead of the traditional ' . '. For example, you might prefer to see a dash or several dashes instead of the period for missing values. This INVALIDIDDATA statement is used on the options line before the data step that read the data lines.

```
OPTIONS INVALIDDATA =  '---';
DATA DISNEY;
        INPUT ID 3. WT1 2. WT2 2.;
        LOSS = WT2 – WT1;
        CARDS;
        001     13      14
        002     11
        003     17      15
        004     16      19
        ;
        RUN;
```

In this example, instead of setting the missing value to '.' – SAS® will assign three dashes ' --- ' for mising values. This option will not override blanks, periods, or special values specified with a MISSING statement.

COLUMN INPUT

Column input is a much safer method for large data sets. If you use COLUMN input, and identify the specific column locations, you do not need to worry about placeholders for missing values. In addition, your data can contain embedded blanks. Here is an example of COLUMN input:

```
DATA DISNEY;
INPUT NAME  $ 1-6 ID 7-9  WT1 10-11  WT2 12-13;
```

Note that the rules to use COLUMN input include the following:

- Data must be in the same field
- Data must be in standard numeric or character format
- Data can contain embedded blanks 1 – 200 characters long
- No placeholder for missing
- Input values can be read in any order
- SAS ignores leading or trailing blanks

FORMATTED INPUT

Formatted data combines informats with column input. FORMATTED INPUT takes the following syntax:

```
DATA DISNEY;
INPUT NAME $ 12. +3 WT1_2. +3 WT2 _3.;
```

- Data can contain embedded blanks 1 – 200 characters
- Missing Data are o.k.
- Pointer controls position
- Data can be non-standard format, i.e. packed decimals, can contain commas

NAMED FORMAT

Values are preceded by the name of the variable with NAMED format. An example of named format takes the following format:

```
DATA DISNEY;
INPUT NAME=$ ID= WT1= WT2 =;
```

FORMATTED INPUT - ADVANCED FEATURES

Formatted input allows you to read data values that contain nonstandard data. The general form of the formatted input contains a pointer control (@ sign followed by variable name an informat name, width delimiter (period) and decimal if numeric informats.

@ var $informat-namew.d

In the mainframe world where I live, we generally use the advanced column pointer @number before the variable name followed by the informat statement. We most often use $charW. informat for character variables because this input allows embedded blanks (nonstandard data). Our input statement for the Disney data would look like the following:

```
DATA DISNEY;
INPUT @ 1 NAME $CHAR10.
      @ 12 WT1 2.
      @ 14 WT2 2.
;
```

Note that the $char informat treats an embedded blank or any other non-standard data (zoned decimal, hexadecimal, packed decimal etc) as valid data. The $charW. informat does not trim leading blanks. This can be of significance when working with data that may contain leading blanks. If there is a chance that your data contains leading or embedded blanks, never use list input. By default list input tells SAS® to interpret any blanks in the data as a delimiter.

PAD OPTION

You can use the pad option when you read data using column input or formatted input, and the data lines containing character values are uneven. This option will pad the end of the data lines with blanks and prevent SAS® from going to the next line to read data.

This is an example of the pad option:

```
data demo;
 infile 'c:\sas\demo.dat' pad;
input id 1-3 name 5-16 no 18-20 street $ 22-
city $ 39-45 state 47-48
zip $ 50-55;
run;
```

SPECIAL MEANING FOR MISSING VARIABLES

SAS® can differentiate between classes of missing variables in numeric data by using the MISSING statement in the DATA STEP. For instance, you may want your data to be numeric for a variable, but there are instances where the value is missing due to some condition – i.e. the Disney character refused to answer, or used an "x" rather than a value in the questionnaire that he completed. You might want to differentiate these values in the data. You can do this easily in SAS® by using the MISSING Statement in the data step. To do this the syntax looks like the following:

```
DATA DISNEY;
MISSING R X;
INPUT ID VAR1 VAR2 VAR3;
```

```
CARDS;
01  2
02  3
03  R
04  2
05  X;
```

If you use the MISSING statement in your data step, when SAS® reads these character values for numeric data, you will not see an Invalid Numeric Data message in the log. This can be of value.

MISSING VALUES IN CONDITIONAL EXPRESSIONS**:**

When you are interested in identifying missing values you can use conditional expressions. Missing numeric values are represented by a single period enclosed in quotes:

```
IF DONALD = ' . ' ;
```

Missing character values are represented by a blank enclosed in quotes:

```
IF DONALD = " ";
```

Special missing numeric values can be followed by a decimal and letter or underscore and letter

```
IF DONALD = " .B " OR "_B";
```

Using missing values in *if then/else* expressions can be tricky to code. If there are missing values, you need to write your if then/else statements to capture the missing condition. Take for example, calculating a condition based on age. A good example of this might be conditional expressions where you want to identify scouting categories where ages are collected for scouts, but not for leaders. You would need to code to incorporate these missing values.

```
If age <10 and age > 12 then scout =
'brownie';
Else if age > 13 and age > 15 then scout =
'junior';
Else if age >16 and < 18 then
scout = 'girlscout';
Else if age = '.' then scout = 'leader';
```

SORTING WITH MISSING BY VALUES:

Understanding how SAS® sorts values will help in explaining why missing variables in a by statement are always represented first. The SAS® System sorts numeric data in a special order. SAS® handles missing values before other numeric values in a very precise order. Considering how SAS® will sort these missing values is important in order to evaluate how to represent your data.

The sort order for alpha/numeric variables is as follows:

SORT ORDER	SYMBOL	DESCRIPTION
Smallest	_	Underscore
	.	Missing or period
	A – Z	Alphabetical characters
	M,X,etc	Special Missing Values Assigned by Missing Statement in Data Step
	-n	Negative numbers
	0	Zero
Largest	+n	Positive numbers

Therefore if you sort by a variable that contains many missing values, those missings will be your first BY grouping. This can be disconcerting unless you understand the sort order that the SAS® System deploys.

The SAS® System also sorts character data in the same fashion. Therefore, missing values will appear before observations that contain character values.

DIFFERENTIATION OF MISSING VALUES

You can differentiate values, if different than you would expect, by assigning a period to a letter or an underscore to a letter as well. In an if then/else statement you can use a .D to assign a value that is different than what you would expect. In our Disney example if loss were not calculated, but reported, and the value of loss was not what was expected based on reported baseline and follow-up weight, then we might want to differentiate that value with a special character. We can use the period character value **".D**" for this value. The Syntax for this differentiation value is as follows:

```
If (wt2 – wt2) ne loss then loss = .d;
```

CHECKING FOR MISSING VALUES

You can check for missing values by using an expression:

```
IF LOSS = '.' THEN DO;
```

Similarly if you were checking for special and missing values, knowing how SAS® handles underscore and period as less in value than character based on the sort order, you could use the following expression:

```
IF LOSS <=.D THEN DO;.
```

PROPOGATION OF MISSING VALUES

When you use a missing value in a mathematical calculation, SAS® sets the result to missing. Therefore the next calculation if a result of the 1st calculation will also be missing and you will get a warning message in your log. However, SAS® will continue to work, propagating the erroneous mathematical results. This can be quite dangerous. This is called **propagation** of missing values. To avoid propagation use SUM functions. These functions ignore missing values. The SUM functions that can handle mathematical problems in spite of missing values include:

SUM FUNTIONS

MAX	Largest non-missing value
MIN	Smallest non-missing value
MEAN	mean of non-missing values
ROUND	rounds to nearest round off unit
SUM	sum of non-missing values

If you were to do a calculation in SAS with missing data your calculations might not be as you would expect. For example:

A + b	1	2	=	3
	1	4	=	5
	1	.	=	.

If you try to perform an illegal mathematical operation SAS prints a warning message in the log and assigns a missing value to the result. Also, if you try to do an arithmetic expression on a non-numeric variable, SAS automatically converts character values to numeric values. SAS will print an error message and set the result of the conversion to missing.

If however, you use the sum function, SAS® ignores missing values.

sum (a,b)	1	2	=	3
	1	4	=	5
	1	.	=	1

The SUM function can handle missing values. No error message will be printed in the log.

RETAIN STATEMENT

As stated previously the SAS® System sets all variables read in from a raw data file to missing in the Program Data Vector with the exception of variables in the RETAIN statement, variables in the SUM statement and variables in temporary arrays, or variables created with the options statement. The RETAIN statement and the SUM function work somewhat in concert. The RETAIN statement is very efficient because it causes a variable whose value is assigned by an INPUT or assignment statement to retain its value from the current iteration of the data step to the next. And similarly any variables used with the SUM function are also retained, and not set to missing.

Without the RETAIN statement, SAS® automatically sets variables to missing before the next iteration of the data step. This is a more efficient approach. Keep in mind that the retain statement can appear anywhere in the data step. Let's look at how this statement can be used in conjunction with calculations using one of the sum functions, remembering that we can use MIN, MAX, SUM and AVG with the sum function for calculations. Let's use our Disney weight loss data for this example:

NAME:	ID	Wt1	Wt2
Mickey Mouse	001	13	12
Daffy Duck	002	11	no value reported
Q Jack Duck	003	17	15
Donald T Duck	004	16	17

Let's retain a variable MINWT – which is the minimum weight of WT1 or WT2, and use a MIN function. For purpose of this example, let's say that it makes sense to keep track of the minimum weight for each subject, but we also want to calculate the total weight loss for our population.

```
DATA DISNEY;

        INPUT ID WT1 WT2;
        RETAIN MINWT;
        MINWT = MIN(MINWT, WT2);
        WEIGHTLS = WT2 – WT1;
        TOTLOSS + WEIGHTLS;
```

The output would look like this:

ID	WT1	WT2	WEIGHTLS
001	13	12	1
002	11	.	.
003	17	15	2
004	16	17	-1

Our output would become:

Wt1	wt2	min	Weightls	TOTLS
13	12	12	1	1
11	.	11	.	.
17	15	11	2	3
16	17	16	1	

In this case, the minimum function was used to keep the minimum value from the previous iteration of the data step. Since it appears in the RETAIN statement the variable TotLs adds the weightls cumulatively while retaining the minimum value for wt2.

ERROR MESSAGES IN THE LOG

One of the best ways to know how much of your data is missing is to know your data well. If it is a small dataset, browsing your dataset, or editing it can be helpful. More often than not, though, your dataset will be so large that eyeballing will not make sense. Using PROC CONTENTS will help you to determine how many observations there are. Checking the Log to be sure your number of observations were read in properly is also important.

The Log should be your best friend. Never attempt to do analysis until you are sure that you have successfully read in the correct number of observations. Check the Log. Seeing a message that SAS® went to a new line should prompt you that something is very wrong. While this message will not look like an error message – it truly is. "SAS WENT TO A NEW LINE WHEN INPUT STATEMENT REACHED PAST THE END OF A LINE" should send up red flags. Also an error message "Lost Card" is a warning that SAS® was looking for another line of data and did not find it. This is another error message doesn't look that problematic, but is!

"INVALID DATA FOR VAR1 IN LINE n" may or not be problematic. This error message may be quite acceptable if you are confident that this data is truly missing. More often that not, however, this error messages means that SAS® could not read the data from your raw data file.are serious. Invalid data might mean that while the input classified the variable as character, SAS® encountered numeric, or vice versa. Again, investigation will lead you to the solution, and knowing your data well is the best scenario.

IDENTIFYING MISSING DATA

Some techniques that are employed to identify missing data in your datasets are simply subsetting the data using an IF statement, or using statistical procedures such as PROC FREQ, PROC UNIVARIATE, or PROC MEANS with the NMISS option.

When you work with small datasets, identifying missing data may be a matter of eyeballing the printed data. When, however, you work with large data set, you can subset the data with an expression, and get a better view of the missing variables. By using a subsetting expression with a proc print, you can access the output and see how many observations contain missing values. To do this, you can use the following subsetting expression:

```
If weightls = '.';
PROC PRINT;
```

By printing the data with this expression, you can quickly know how many observations are missing simply by checking the observation number in the log. This is acceptable for relatively small datasets. But when working with datasets that contain millions of records, there are some more sophisticated techniques that give you an exact count of missing values.

One frequent technique that we use with claims data is PROC FREQ. Subsetting the variables into meaningful subcategories, perhaps by line of business or claim type code, and doing a two-way proc freq will give you a two by two table to determine not only how many values are missing but in which category they are missing.

An example would be:

```
Proc freq;
Tables lob*ctypecd;
```

In this example, you will see how many observations for each of the claim type categories, and at the end of the output, there will be a missing frequency statistic.

PROC UNIVARIATE is another option. This procedure gives you the most extensive summary of numeric data by checking the MISSING VALUES SECTION of the output. This section will tell you specific information about missing values: MISSING VALUE shows how the values are coded, COUNT will give you the number of missing values and % COUNT/NOBS give you the percent of missing data.

Remember that if you do not include a VAR statement, PROC UNIVARIATE will give you summary statistics on all numeric variables in your dataset. If you are only interested in specific numeric variables, you should use the VAR statement.

```
PROC UNIVARIATE DATA=DISNEY;
VAR WT1 WT2;
```

As in the case of the PROC FREQ, you can use a BY statement to obtain separate analysis for by groupings. Remember to always sort your data first by the by grouping variable if your data are not sorted this way.

Lastly you can use a PROC MEANS with a NMISS option to obtain missing information about numeric variables. The syntax for this option is:

```
PROC MEANS DATA =DISNEY NMISS;
By groupvar;
VAR WT1 WT2;
```

And similarly to PROC UNIVARIATE, unless you want to see information on all numeric variables, you should limit the variables by using a VAR statement.

CONCLUSION

Most data files will contain missing values. Working with missing data need not be problematic as long as you understand some general principals about how SAS reads this missing data, how the SAS system sorts the data, and how you can go about identifying missing data. Missing data certainly can be tricky to work with, but by following a few simple techniques you can be assured that like the Y2K bug, it will not cause mayhem.

REFERENCES

SAS® Institute, *Fundamentals of the SAS® System (Version 6),* Course Notes, 1992.

SAS® Institute, *SAS® Language and Procedures, (Version 6),* First Edition, 1989, Cary, NC

SAS® Institute, *SAS® Language: Reference*, Version 6, First Edition, 1990, Cary, NC

Lora D. Delwiche and Susan J. Slaughter*, The Little SAS® Book: A Primer,* 1995, SAS Institute Inc., Cary NC

CONTACT

JoAnn Matthews
Highmark Blue Cross/Blue Shield
120 Fifth Avenue
Pittsburgh PA 15222
Phone 412 –544-2397
e-mail joann.matthews@highmark.com