PharmaSUG China 2016 - 78

What's the BIG Deal with Missing Data?

Peter Eberhardt, Fernwood Consulting Group Inc.

Mina Chen, Roche Product Development in Asia Pacific, Shanghai, China

ABSTRACT

In clinical trials, a major problem in the data analysis is missing values caused by unrecorded results of measurements at some planned visits, or patients dropping out of the study before completion. Missing values can have a surprising impact on the way that data are dealt with, which may result in biased treatment comparisons as well as impact the overall statistical results of the study.

Generally, missing values can be represented by SAS® in a number of ways, there are various functions, options and techniques associated with missing values, and procedures will have specific ways of handling them. However, not all SAS programmers are aware of the System options, DATA step functions, and DATA step routines that specifically deal with missing values.

This paper will include the basics of how to detect missing values, and how to effectively make use of various functions and tools within SAS to utilize missing values.

INTRODUCTION

In clinical trials, as with all analytic projects, a major problem in the analysis is the presence of missing values; that is, analytic variables for which we have no values. There can be many reasons we have missing values, and knowing the reason the data are missing can help us deal with them. Missing values can have a surprising impact on the way that data are dealt with, which may result in biased treatment comparisons as well as impact the overall statistical results of the study. In this paper we will focus on analysis variables and variables used in calculations to create analysis variables, dates being a common example of the latter.

WHAT IS MISSING

Most of us think of missing values as a variable that has no value. Although this is effectively true, that is there is no analytical value (no number upon which we can perform operations), the variable does have a value; by default this value for a numeric is represented by a period (.); for a character variable, missing is represented by all spaces. As we will see this is important to remember. In addition, missing values are a SAS construct, only available in SAS (although other systems may also have something similar).

Missing vs NULL

In data base systems (RDBMS) we encounter NULLs, a NULL means the data do not exist. This is different than a SAS missing, which does have a value. When numeric values are passed to/from an RDBMS, missing are converted to NULLs and NULLs are converted to the default missing (.). Effectively we can treat numeric missing/NULL the same way in SAS DATA step programs and PROCs since they are implicitly converted. When a character missing (all spaces) is passed to an RDBMS, the value is not converted to NULL, rather it simply becomes a variable with all spaces. On the other hand, when a character variable from an RDBMS which is NULL or all spaces is converted to SAS, it is interpreted as missing by SAS.

MISSING VALUES IN CLINICAL TRIAL

Missing data is a potential source of bias when analyzing clinical trials. Interpretation of the results of a trial is always problematic when the proportion of missing values is substantial. There are many possible reasons for missing data (e.g. patient refusal to continue in the study, patient withdrawals due to treatment failure, treatment success or adverse events, patients moving, lab assessments not done), only some of which are related to study treatment. Different degrees of data incompleteness can occur, i.e. measurements may be available only at baseline, or measurements may be missing at baseline, or may be missing for one, several or all follow-up assessments. Even if a patient completes the study, some data may remain simply unreported or uncollected.

Generally we use a period (.) as missing for numeric data, and blank for character data.

The situations with missing date are:

1. Missing efficacy endpoints

The result of efficacy parameter can be numeric or character. For example:

The example dataset contains patient id, visit information, assessment time as well as efficacy results in both numeric and character format. 'Y' (corresponding numeric value is 1) means response and 'N' (numeric value is 0) means non-response.

Generally, we use the LOCF (Last observation carried forward) method to impute missing endpoints. For each individual, missing values are replaced by the last observed value of that variable.

USUBJID	VISIT	VISITN	ADTM	AVALC	AVALN
1001	BASELINE	0	27SEP2013	N	0
1001	WEEK 1	1	04OCT2013	N	0
1001	WEEK 2	2	11OCT2013	N	0
1001	WEEK 4	3	25OCT2013	N	0
1001	WEEK 8	4	22NOV2013	N	0
1001	WEEK 12	5	20DEC2013	N	0
1001	WEEK 18	6	31JAN2013	N	0
1001	WEEK 24	7	14MAR2014	Y	1
1001	WEEK 30	8	25APR2014	Y	1
1001	WEEK 36	9	06JUN2014	Y	1
1001	WEEK 42	10	18JUL2014	Y	1
1001	WEEK 46	11	29AUG2014		.
1001	FOLLOW-UP WEEK 4	12	26SEP2014		.
1001	FOLLOW-UP WEEK 12	13	21NOV2014	Y	1
1001	FOLLOW-UP WEEK 24	14	12FEB2015	N	0

2. Missing date (partial date)

Dates are a critical part of the data collected in clinical trials. However, partial dates are almost inevitable during the data collection. In clinical trials, partial dates are most common in variables where the date is historical information (e.g. concomitant medication, medical history). This is a general problem because it is possible that the subject does not recall a wholly accurate start date for a medication they have been taking for a number of years. Generally it is preferable to leave the date as it is if the date is merely to be reported. However, if the date is to be used to calculate the duration of an adverse event, then a partial date will prevent the duration being calculated. In this situation, the first or last day of the month, or first or last month of the year, or the first or last study contact day, or the first or last date of dosing may be used to impute partial dates.

Here's an example of AE data. In this dataset, the day part of the start date of adverse event "PYREXIA" is missing, and the month part and day part of adverse event "WEIGHT DECREASED" are missing. We usually use '01' to impute missing day part and '01' for missing month part.

USUBJID	AEDECOD	AESTDTC	AEENDTC
1001	VOMITING	2013-11-22	2014-11-21
1001	PYREXIA	2013-12	2013-12-20
1001	WEIGHT DECREASED	2013	2014-09-26
1001	HEADACHE	2014-03-21	2014-03-21
1001	PYREXIA	2014-03-21	2014-03-21

3. Not conducted laboratory or vital signs assessments

When conducting the laboratory or vital signs assessments, the results are leave as missing if the assessments are not conducted. In this case we usually delete these missing values from analysis.

Handling Missing Values in the DATA step

In a SAS DATA we will often see a construct like:

```
DATA showMissing;
      set hasMissingValue;
```

```
      if varA = .
            then put 'A is missing';
            else put varA=;
run;
```

In this example we are simply displaying values to the log; normally the logic in the IF/THEN/ELSE sections would be more complex. There is an inherent problem with this example; varA can take on 28 different values and still be missing. This example on checked for one of the possible missing values. So, what are we missing?

What are we missing?

It is common to use the statement:

```
      if varA = .
```

in your DATA step code. Although it is common, it is not a best practice. A best practice would be:

```
      if missing(varA)
```

In total, there are 28 different missing values listed from smallest to largest:

- ._ (dot underscore)
- . (dot)
- .A - .Z (dot A to dot Z case insensitive)

The statement

```
if missing(varA)
```

would catch all the possible missing values, whereas

```
      if varA = .
```

would not catch **._** or **.A** - **.Z**, that is, it could lead to incorrect processing of data while still executing correctly.

Why so many missing?

Even though we do not have a value we can use for analysis, we may have some information about why the value is missing, for example there is no visit record because the visit was not required, or perhaps the patient cancelled. By coding all missing values as the default **dot**, we lose information. If we know the patient cancelled, we could code the value as .C, if the visit was not required we could code it as .N. When we perform analysis upon the data these values are omitted since these are missing values. Later we may want to see how often patients cancelled visits, or subset those who did not require a visit. To do so, we simply filter on the specific missing value:

```
      if varA = .C
```

or

```
      where varA = .N
```

By using these missing values we have not compromised our analysis, yet at the same time we have increased the information in our data.

Reading in missing values

If you have a text data file or EXCEL worksheet that has your raw data with these missing values coded, then you may have to create INFORMATs to read the data. By utilizing INFORMATs we can have very flexible ways of

reading in data that represent missing values. Let's look at one of the more challenging and important variable types, SAS dates.

SAS dates with missing values

If we start with the simplest approach of reading SAS dates from a text file, we could just ensure all the reasons for missing are assigned one of the special missing values, for example .N for not required and .C for cancelled. A simple DATA step to illustrate this is as follows:

```
data datesWithMissing;
input studyDt yymmdd10.;
format studyDt yymmdd10.;
datalines;
2016/08/07
2016/08/06
.c
.n
;;;;
run;
```

When SAS reads numeric data, in this case a date variable, it will automatically recognize the missing values and assign them appropriately. Although this works well, it seems likely that there could be issues with data entry not keying in the dot, that is enter **C** not **,C.** if this happened, the value of studyDt would revert to the default missing value (.). To work around this you would need to create a new INFORMAT using PROC fcmp. Expaining PROC fcmp is beyond the scope of this paper; see Eberhardt 2009, Eberhardt 2010, for more on the procedure. For now, here is an example; since PROC fcmp uses data step syntax, the program should be clear.

```
options cmplib=work.missval;
proc fcmp outlib=work.missval.miss1;;
function cnvMissDt (inp $) ;
    length c1 $100;
    c1 = inp;
    /*
       first see if a date,
       if not a date and not special missing
       then will be the default
    */
    d1 = input(c1,  ANYDTDTE10.);
    if d1 = . /* if not a date, value is defaut missing */
    then
      do;
        if length(c1) = 1  /* was it one char only */
        then
          do;
            c2=rank(upcase(c1));
            if (65 <= c2 <= 90) or (c2 = 95)
            then /* if in A - Z, convert to .A - .Z
              do;
                b = rank(upcase(c1)); put b=;
                c1= cat('.', c1);
                d1= input(c1,  2.);
              end;
          end;
```

```
        if length(c1) = 2
        then  /* try converting
          do;
            d1= input(c1,  2.);
          end;
      end;
    return(d1);
endsub;
quit;

/* use the function in a format */
proc format;
 invalue miss (default=10)
 other = [cnvMissDT()]
 ;
 run;
 quit;

data datesWithMissing ;
input d1 miss10.;
datalines;
2016-01-01
2016-02-30
.a
.e
a
;;
run;
```

Some useful functions for missing values

MISSING()

We already saw an example of this, and the reason why we would want to use it. The other important part of the missing() function is the fact it works on both numeric and character variables:

```
data _null_;
   a = '1'; b = 1;
   put 'test 1:';
   if missing(a) then put '     a is missing';
   if missing(b) then put '     b is missing';
   a = ''; b = .z;
   put 'test 2:';
   if missing(a) then put '     a is missing';
   if missing(b) then put '     b is missing';
   put 'done';
run;
```

and the log:

```
test 1:
test 2:
     a is missing
     b is missing
done
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Although the missing() function only tests on variable at a time, we can use the cats() function in conjunction with the missing() function to test a set of character variables:

```
data _null_;
   a = '1'; b = '2';
   put 'test 1:';
   if missing( cats(a , b )) then put "     both are missing";
   a = ' '; b = ' ';
   put 'test 2:';
   if missing( cats(a , b )) then put "     both are missing";
   put 'done';
run;
```

and the log:

```
test 1:
test 2:
     both are missing
done
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

CALL MISSING()

The call missing() routineis useful to set a number of variables to missing in one call. One common approach to setting a number of variables to missing would look like:

```
data missingValues;
     length a b c d e  8.;
     length v w x y z $4.;
     array nums(*)    a--e;
     array chars(*) $ v--z;
     do i = 1 to dim(nums);
        nums(i) = .;
     end;
     do i = 1 to dim(chars);
        chars(i) = ' ';
     end;
run;
```

Here we put the variable we want into an array, then iterate over the array and set the individual variables to missing. This can be simplified to:

```
data missingValues;
     length a b c d e  8.;
     length v w x y z $4.;
     array nums(*)    a--e;
     array chars(*) $ v--z;
     call missing(of nums(*), of chars(*));
run;
```

Here we still make use of arrays (a very useful construct), but now in one call, set all the numeric and all the character variables to missing;

NMISS() and CMISS()

When you have several variables, you might want to know how many have missing values; to do this use the nmiss() (for numeric) and cmiss() (for character) functions:

```
data missingValues;
     length a b c d e  8.;
     length v w x y z $4.;
     array nums(*)    a--e;
     array chars(*) $ v--z;
     call missing(of nums(*), of chars(*));
     nn = nmiss(of nums(*));  put 'number of missing nums:  ' nn 3.;
     nc = cmiss(of chars(*)); put 'number of missing chars: ' nc 3.;
run;
```

and the log:

```
number of missing nums:    5
number of missing chars:   5
NOTE: The data set WORK.MISSINGVALUES has 1 observations and 12
variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds
```

If you want to know the total number of missing values, numeric and character, you can use either the nmiss() or the cmiss() function. In general it is better to use the cmiss() function to avoid the data conversion message in the log.

Using nmiss():

```
data missingValues;
     length a b c d e  8.;
     length v w x y z $4.;
     array nums(*)    a--e;
     array chars(*) $ v--z;
     call missing(of nums(*), of chars(*));
     nt = nmiss(of nums(*),of chars(*)); put 'total number of
missing: ' nt 3.;
run;
```

and the log:

```
NOTE: Character values have been converted to numeric values at the
places given by: (Line):(Column).  743:31
total number of missing:  10
NOTE: The data set WORK.MISSINGVALUES has 1 observations and 11
variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Using cmiss():

```
data missingValues;
     length a b c d e  8.;
     length v w x y z $4.;
     array nums(*)    a--e;
     array chars(*) $ v--z;
     call missing(of nums(*), of chars(*));
     nt = cmiss(of nums(*),of chars(*)); put 'total number of
missing: ' nt 3.;
run;
```

and the log:

```
total number of missing:  10
NOTE: The data set WORK.MISSINGVALUES has 1 observations and 11
variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

N()

The n() function is the opposite of the nmiss() function – it returns the number of non-missing values.

CONCLUSION

Properly handling missing values is an important part of every clinical trial. Understanding the ways we can assign and handle missing values can both simplify and enhance our analysis. By understanding and using the rich variety of missing values available to us we can add value to our data.

References

Eberhardt, Peter. 2009. "A Cup of Coffee and Proc FCMP: I Cannot Function Without Them." *Proceedings of the SAS Global Forum 2009 Conference.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings09/147-2009.pdf.

Eberhardt, Peter. 2010. "Functioning at an Advanced Level: PROC FCMP and PROC PROTO" *Proceedings of the SAS Global Forum 2010 Conference.* Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings10/024-2010.pdf

Langston, Rick 2012 “Using the New Features in PROC FORMAT” *Proceedings of the SAS Global Forum 2012 Conference.* Cary, NC: SAS Institute Inc. Available at https://support.sas.com/resources/papers/proceedings12/245-2012.pdf

Contact Information

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

Name: Peter Eberhardt
Enterprise: Fernwood Consulting Group Inc.
City, Prov: Toronto, ON, Canada
E-mail: peter@fermwood.ca
Twitter: rkinRobin

Name: Mina Chen
Enterprise: Roche (China) Holding Ltd.
Address: 11 Building, floor 4, 1100 Long Dong Ave. Shanghai
City, Prov: Shanghai, China
E-mail: mina.chen@roche.com