Paper 182-2019

Much Ado About Nothing: dealing with blank values in your data

Ron Walker, Loughlin Consulting

ABSTRACT

Taken literally, the title of Shakespeare's play "Much Ado About Nothing" implies that a great fuss ("much ado") is made about something that is insignificant ("nothing"). For example, Claudio and Hero fall in love and are to be married. However, a circumstance arises where Claudio mistakenly believes that Hero's love for him isn't true. While Claudio's mistaken belief is but a "trifle" it becomes the focal point of the entire play.

If not identified properly, "nothing" has a way of becoming "something," and potentially requiring a great deal of attention. This is true both of life and of data.

INTRODUCTION

This has potential to be a HUGE topic covering almost all areas of SAS Programming – far bigger than can be covered in this brief presentation. For that reason, this paper will focus on the following areas:

- What causes unexpected blank values?
- When are blank values a problem?
- Ways to determine if your data has blank values.
- Tools and techniques for identifying and preventing blanks.
- Removing blanks from a string of text.

In this paper, we will review techniques for identifying the causes of unexpected blank values in data as well as some tips and tricks for how to effectively deal with them.

WHAT CAUSES UNEXPECTED BLANK VALUES?

Typically unexpected blanks result from problems with data capture or data retention. Occasionally, SAS will create unexpected blank values by converting raw data that doesn't conform to expectations. This is a feature, not a bug.

"Every variable and observation in a SAS data set must have a value. If a data value is unknown for a particular observation, a missing value is recorded in the SAS data set."

SAS® Certification Prep Guide. Base Programming for SAS9®, Third Edition, p. 16.

WHEN ARE BLANK VALUES A PROBLEM?

Blank values are not a problem if the following two conditions exist:

1. All Stakeholders are aware of the possibility of blank values,
2. All Stakeholders agree on what "blank" actually means.

Example 1: The dataset in Table 1 includes a blank (missing) value for the variable "Checkout," indicating that guest "Holly Dayin" has not checked out yet. No problem.

Table 1

FNAME	LNAME	CHECKIN	CHECKOUT
Mary	Ott	Aug 30, 2019	Sep 2, 2019
Carey	Oakey	Aug 31, 2019	Sep 1, 2019
Lon	Moore	Sep 1, 2019	Sep 3, 2019
Rita	Booke	Sep 1, 2019	Sep 4, 2019
Mo	Telsiks	Sep 2, 2019	Sep 4, 2019
Lynn	Guini	Sep 2, 2019	Sep 5, 2019
Holly	Dayin	Sep 3, 2019	.
Moe	Skeeto	Sep 4, 2019	Sep 5, 2019
Pete	Zaria	.	Sep 5, 2019

However, the data set also includes a blank for the variable “Checkin” for guest Pete Zaria, who has a “Checkout” date but no corresponding “Checkin” date. Blank value = problem.

Identifying and fixing the data input issue which caused the blank value for “Checkin” will be discussed later in this paper.

EXAMPLE 2: Blank values can also be a problem if we don’t expect them in our data and therefore we don’t take them into account in our programming.

This program reads a simple data set and attempts to create accumulating variable “day_sum” from the values of the variable “RoomRate.” However, a missing “RoomRate” value for guest “Moe Skeeto” causes the variable value to default to missing, as indicated in the resulting dataset (Table 2), and the “day_sum” variable stops accumulating at $639.

```
data RoomRate;
   input   FNAME : $10. LNAME : $15. RoomRate : 5.;
   Format RoomRate day_sum dollar5.;
   Retain day_sum 0;
   day_sum = day_sum + RoomRate;
datalines;
Mary Ott 125
Carey Oakey 93
Lon Moore 88
Rita Booke 75
Mo Telsiks 70
Lynn Guini 86
Holly Dayin 201
Moe Skeeto .
Pete Zaria 100
;
```

Table 2

FNAME	LNAME	RoomRate	day_sum
Mary	Ott	$125	$125
Carey	Oakey	$93	$218
Lon	Moore	$88	$306
Rita	Booke	$75	$381
Mo	Telsiks	$70	$451
Lynn	Guini	$86	$537
Holly	Dayin	$102	$639
Moe	Skeeto	.	.
Pete	Zaria	$100	.

SOLUTION: A Sum Statement (day_sum + RoomRate) is considered a best practice for accumulating variables specifically because it skips over missing values. Using a Sum Statement, our "day_sum" variable properly accumulates to the correct amount.

```
data RoomRate;
   input   FNAME : $10.  LNAME : $15.  RoomRate : 5.;
   Format RoomRate day_sum dollar5.;
   day_sum + RoomRate;
datalines;
Mary Ott 125
Carey Oakey 93
Lon Moore 88
Rita Booke 75
Mo Telsiks 70
Lynn Guini 86
Holly Dayin 201
Moe Skeeto .
Pete Zaria 100
;
```

Table 3

FNAME	LNAME	RoomRate	day_sum
Mary	Ott	$125	$125
Carey	Oakey	$93	$218
Lon	Moore	$88	$306
Rita	Booke	$75	$381
Mo	Telsiks	$70	$451
Lynn	Guini	$86	$537
Holly	Dayin	$102	$639
Moe	Skeeto	.	$639
Pete	Zaria	$100	$739

EXAMPLE3: Finally, blank values can be a problem if they reveal unanticipated problems in our programming logic.

The following program reads in a dataset which includes 5 distinct values for the variable “gender” (F, X, M, f, m), then invokes a data constraint allowing only two values (M, F) as valid, setting remaining variable values (X, f, m) to blank.

```
data missing_gender;
   input   FNAME : $10.  LNAME : $15.  GENDER : $1.;
   if gender not in ('M','F') then gender =" ";
datalines;
Mary Ott F
Carey Oakey X
Lon Moore M
Rita Booke f
Mo Telsiks M
Lynn Guini F
Holly Dayin F
Moe Skeeto m
Pete Zaria M
;
```

Table 4

FNAME	LNAME	GENDER
Mary	Ott	F
Carey	Oakey	
Lon	Moore	M
Rita	Booke	
Mo	Telsiks	M
Lynn	Guini	F
Holly	Dayin	F
Moe	Skeeto	
Pete	Zaria	M

In this case, no message will appear in the log indicating a problem with the program because the program is functioning as intended. Fixing the data input issue which caused the blank values for “Gender” will be discussed later in this paper.

WAYS TO DETERMINE IF YOUR DATA HAS BLANK VALUES

PROC PRINT

The simplest way to identify blank or missing values is to use the WHERE clause in the PRINT procedure with operators such as IS MISSING or IS NULL to print observations with missing values for the variable of interest.

Example: in the Table 4 dataset with 3 missing values for “gender,” the following program will print only those 3 observations, as indicated in Table 5.

```
proc print data=missing_gender noobs;
   where gender IS MISSING;
run;
```

Table 5

FNAME	LNAME	GENDER
Carey	Oakey	
Rita	Booke	
Moe	Skeeto	

The WHERE clause can also be written as follows:

- Where gender is NULL;
- Where gender is " ";

For numeric variables the WHERE clause can be written as follows:

- Where NumVar is .;

I prefer the IS MISSING operator as it works regardless of variable type, and it makes programs more readable, particularly to non-programmers.

PROC FREQ

Another useful tool for quickly finding blank values, in the FREQUENCY procedure the default is to list blank (or missing) values as a footnote, and not included them in calculations. The MISSING option changes this default behavior.

Example: the following program, not including the MISSING option, produces the report in Table 6. Note Frequency Missing = 3 indicated as a footnote (Frequency Missing =3), and note that missing values are not included as part of percentage calculations.

```
proc freq data=missing_gender;
  tables gender;
run;
```

Table 6

GENDER	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
F	3	50.00	3	50.00
M	3	50.00	6	100.00
Frequency Missing = 3				

Adding the MISSING option includes these values as part of the Frequency calculation (Table 7).

```
proc freq data=missing_gender;
  tables gender / MISSING;
run;
```

Table 7

GENDER	FREQUENCY	PERCENT	CUMULATIVE FREQUENCY	CUMULATIVE PERCENT
	3	33.33	3	33.33
F	3	33.33	6	66.67
M	3	33.33	9	100.00

PROC MEANS

Programs often begin with a MEANS procedure to capture summary statistics for use in later calculations. If missing numeric values aren't expected, the result can be summary statistics on only a partial data set, with no clear visibility to whether observations have been left out of the calculations.

Example: in Table 8 we revisit the scenario with a missing value for the variable RoomRate. Submitting a MEANS procedure will result in summary statistics for only 8 of the 9 variable values, as indicated in Table 9.

Table 8

FNAME	LNAME	RoomRate
Mary	Ott	$125
Carey	Oakey	$93
Lon	Moore	$88
Rita	Booke	$75
Mo	Telsiks	$70
Lynn	Guini	$86
Holly	Dayin	$102
Moe	Skeeto	.
Pete	Zaria	$100

```
proc means data=roomrate;
   var roomrate;
run;
```

Table 9

Analysis Variable: RoomRate				
N	Mean	Std Dev	Minimum	Maximum
8	92.4	17.2	70.0	125.0

No log message appears to indicate that statistics only reflect observations with non-missing values. The programmer needs to be aware of the number of observations in the dataset, and visually cross-reference that number with the number of observations used to create summary statistics.

Of course, if we only ever dealt with small datasets such as the above, we'd simply look at the above result and know that 9 observations were read but only 8 produced statistics. The problem is that the eyeball method doesn't scale to thousands (or even hundreds) of observations.

One easy remedy is to first run a MEANS procedure with the N and NMISS options, creating the report in Table 10.

```
proc means data=roomrate N NMISS;
   var roomrate;
run;
```

Table 10

Analysis Variable: RoomRate	
N	N Miss
8	1

Defensive programming methods such as this can save time and trouble later by quickly identifying problems in input datasets. In this case, the only fix is to either accept summary statistics on a partial data set, or fix the raw data to include a value for RoomRate. Assigning an appropriate rate (Table 11) yields the correct result (Tables 12-13).

Table 11

FNAME	LNAME	RoomRate
Mary	Ott	$125
Carey	Oakey	$93
Lon	Moore	$88
Rita	Booke	$75
Mo	Telsiks	$70
Lynn	Guini	$86
Holly	Dayin	$102
Moe	Skeeto	.
Pete	Zaria	$100

```
proc means data=roomrate N NMISS;
   var roomrate;
run;
```

```
proc means data=roomrate;
   var roomrate;
run;
```

Table 12

Analysis Variable: RoomRate	
N	N Miss
9	0

Table 13

Analysis Variable: RoomRate				
N	Mean	Std Dev	Minimum	Maximum
9	109.8	54.9	70.0	250.0

TOOLS AND TECHNIQUES FOR IDENTIFYING AND PREVENTING BLANKS

The techniques included above are good at finding blank values that have managed to sneak into our datasets. There are additional techniques that can help us identify missing or blank values as they're being added to data.

READ THE LOG

Simply reading the log is the first line of defense against all sorts of programming challenges. As a best practice, the log should probably be viewed prior to viewing output.

Revisiting the earlier example with the missing “Checkin” date for customer Pete Zaria (Table 14), the following program indicates that the reason is an invalid date value of 41SEP2019 (September 41st).

```
data checkout;
  input   @1 FNAME : $8. @9 LNAME : $8. @20 CHECKIN : date9.
            @30 CHECKOUT : date9.;
  If checkout > '5SEP2019'd then checkout=.;
  Format checkin checkout worddate12.;
datalines;
Mary    Ott         30AUG2019  2SEP2019
Carey   Oakey       31AUG2019  1SEP2019
Lon     Moore       1SEP2019   3SEP2019
Rita    Booke       1SEP2019   4SEP2019
Mo      Telsiks     2SEP2019   4SEP2019
Lynn    Guini       2SEP2019   5SEP2019
Holly   Dayin       3SEP2019   6SEP2019
Moe     Skeeto      4SEP2019   5SEP2019
Pete    Zaria       41SEP2019  5SEP2019
;
```

Table 14

FNAME	LNAME	CHECKIN	CHECKOUT
Mary	Ott	Aug 30, 2019	Sep 2, 2019
Carey	Oakey	Aug 31, 2019	Sep 1, 2019
Lon	Moore	Sep 1, 2019	Sep 3, 2019
Rita	Booke	Sep 1, 2019	Sep 4, 2019
Mo	Telsiks	Sep 2, 2019	Sep 4, 2019
Lynn	Guini	Sep 2, 2019	Sep 5, 2019
Holly	Dayin	Sep 3, 2019	.
Moe	Skeeto	Sep 4, 2019	Sep 5, 2019
Pete	Zaria	.	Sep 5, 2019

As SAS reads in data, “invalid” values are set to missing and a note is added to the log. This repeats for each invalid value up to the predetermined error limit. The note indicates which observation and even which specific value that caused the error.

```
1    data checkout2;
2        input   @1 Fname : $8.
3                @9 Lname : $8.
4                @20 Checkin: date9.
5                @30 Checkout: date9.;
6                if checkout > '5SEP2019'd then checkout=.;
7                format checkin checkout worddate12.;
8    datalines;

NOTE: Invalid data for Checkin in line 17 21-29.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----
6----+----7----+----8----+---
17         Pete    Zaria       41SEP2019 5SEP2019
Fname=Pete Lname=Zaria Checkin=. Checkout=Sep 5, 2019 _ERROR_=1 _N_=9
NOTE: The data set WORK.CHECKOUT2 has 9 observations and 4 variables.
NOTE: DATA statement used (Total process time):
      real time           0.10 seconds
      cpu time            0.07 seconds
```

Display 1. Log With Error Due To Invalid Date Value

While reading the log is essential, it only prints errors to the predetermined limit (typically 20). The error limit can be changed using the ERRORS= system option. A bigger challenge is that the log is not easily portable as a report to send to those responsible for providing raw data to advise them of which specific values are problematic.

READ NUMERIC DATA TWICE

Continuing with the missing date example, reading date values first with a numeric informat but then reading it again with a character informat allows us to capture the problematic value for later reporting.

This program creates the additional variables “Char_checkin” and “Char_checkout” using the $CHAR10. Informat. Note that the $n. Informat left justifies on input and as such ignores leading blanks, while the $CHARn. Informat includes leading blanks, making it a better choice for debugging.

```
data checkdates;
  input   @1 FNAME : $8. @9 LNAME : $8.
            @20 CHECKIN : date9.   @20 CHAR_CHECKIN : $CHAR10.
            @30 CHECKOUT : date9.  @30 CHAR_CHECKOUT : $CHAR10.;
  If checkout > ‘5SEP2019’d then checkout=.;
  Format checkin checkout worddate12.;
datalines;
Mary    Ott         30AUG2019  2SEP2019
Carey   Oakey       31AUG2019  1SEP2019
Lon     Moore       1SEP2019   3SEP2019
Rita    Booke       1SEP2019   4SEP2019
Mo      Telsiks     2SEP2019   4SEP2019
Lynn    Guini       2SEP2019   5SEP2019
Holly   Dayin       3SEP2019   6SEP2019
Moe     Skeeto      4SEP2019   5SEP2019
Pete    Zaria       41SEP2019  5SEP2019
;
```

The resulting dataset contains our blank “Checkin” value for Pete Zaria, but also displays the incorrect value (September 41st) from the raw data in “Char_checkin.” Unlike the log, this result can easily be exported to a report for additional review.

Table 15

FNAME	LNAME	CHECKIN	CHAR_CHECKIN	CHECKOUT	CHAR_CHECKOUT
Mary	Ott	Aug 30, 2019	30AUG2019	Sep 2, 2019	2SEP2019
Carey	Oakey	Aug 31, 2019	31AUG2019	Sep 1, 2019	1SEP2019
Lon	Moore	Sep 1, 2019	1SEP2019	Sep 3, 2019	3SEP2019
Rita	Booke	Sep 1, 2019	1SEP2019	Sep 4, 2019	4SEP2019
Mo	Telsiks	Sep 2, 2019	2SEP2019	Sep 4, 2019	4SEP2019
Lynn	Guini	Sep 2, 2019	2SEP2019	Sep 5, 2019	5SEP2019
Holly	Dayin	Sep 3, 2019	3SEP2019	.	6SEP2019
Moe	Skeeto	Sep 4, 2019	4SEP2019	Sep 5, 2019	5SEP2019
Pete	Zaria	.	41SEP2019	Sep 5, 2019	5SEP2019

The resulting dataset contains our blank “Checkin” value for Pete Zaria, but also displays the incorrect value (September 41st) from the raw data in “Char_checkin.” Unlike the log, this result can easily be exported to a report for additional review, such as in the following program and resulting dataset (Table 15). This can be particularly useful if the programmer is not responsible for data capture.

```
Title 'Invalid CHECKIN Date – Please review':
   Proc print data=checkdates noobs;
   where checkin IS MISSING;
   var FNAME LNAME CHECKIN CHAR_CHECKIN;
run;
title;
```

Table 16

Invalid CHECKIN Date – Please review			
FNAME	LNAME	CHECKIN	CHAR_CHECKIN
Pete	Zaria	.	41SEP2019

DATA CLEANING

Data cleaning and preparation can comprise 60% - 90% of the time spent in reporting or data modeling, a huge topic (see Ron Cody’s excellent book on the subject of Data Cleaning) which can only be touched on briefly here.

In the earlier example with missing values for gender (Table 16), we saw a data constraint that converted non-conforming input data to missing.

```
data missing_gender;
  input   FNAME : $10.  LNAME : $15.  GENDER : $1.;
  if gender not in ('M','F') then gender =" ";
datalines;
Mary Ott F
Carey Oakey X
```

```
Lon Moore M
Rita Booke f
Mo Telsiks M
Lynn Guini F
Holly Dayin F
Moe Skeeto m
Pete Zaria M
;
```

Table 17

FNAME	LNAME	GENDER
Mary	Ott	F
Carey	Oakey	
Lon	Moore	M
Rita	Booke	
Mo	Telsiks	M
Lynn	Guini	F
Holly	Dayin	F
Moe	Skeeto	
Pete	Zaria	M

One way to prevent blanks in this scenario is to convert all raw data values of gender to the required upper case using the UPCASE function, and doing so prior to invoking the data constraint.

Another is to be vigilant about business or societal changes which can impact data constraints. As of Summer 2019 at least 10 states allow “X” gender markers on driver’s licenses and State ID cards, typically for transgender, non-binary or intersex residents. Valid changes to input data can wreak havoc on previously stable code.

Adding the UPCASE function and updating the data constraint solves our missing gender issues.

```
data missing_gender;
   input   FNAME : $10.  LNAME : $15.  GENDER : $1.;
   gender = UPCASE (gender);
   if gender not in ('M','F','X') then gender =" ";
datalines;
Mary Ott F
Carey Oakey X
Lon Moore M
Rita Booke f
Mo Telsiks M
Lynn Guini F
Holly Dayin F
Moe Skeeto m
Pete Zaria M
;
```

Table 18

FNAME	LNAME	GENDER
Mary	Ott	F
Carey	Oakey	X
Lon	Moore	M
Rita	Booke	F
Mo	Telsiks	M
Lynn	Guini	F
Holly	Dayin	F
Moe	Skeeto	M
Pete	Zaria	M

REMOVING BLANKS FROM A STRING OF TEXT

One persistent source of unexpected blanks in data comes from strings of text with leading, trailing or even embedded blanks. There are several ways to remove unwanted blanks from text strings. A few of them are discussed here.

THE COMPBL, COMPRESS, TRIMN AND STRIP FUNCTIONS

These four functions are particularly useful in removing unwanted blanks from strings of text.

The COMPBL or “Compress Blank” function converts multiple blanks into a single blank in a text string.

The COMPRESS function removes blanks from a text string.

The TRIMN function removes leading blanks from a text string.

The STRIP function removes both leading and trailing blanks from a text string.

The following program reads in text strings with embedded blanks, and the resulting report (Table 18) shows the variation in lengths of the newly created variables.

REMOVING EMBEDDED BLANKS WITH THE COMPBL AND COMPRESS FUNCTIONS

```
data SpaceData;
   input SpaceText $20.;
   length=lengthn(SpaceText);
datalines;
NoSpaceData
One Space Data;
Two  Space  Data;
Three   Space   Data
;
```

Table 19

SpaceText	length
NoSpaceData	11
One Space Data	14
Two Space Data	16
Three Space Data	20

To demonstrate how the COMPBL and COMPRESS functions work, this program creates new variables "COMPB" and "COMPR" using the COMPBL and COMPRESS functions. Note in the resulting dataset (Table 19) that variable length is reduced appropriately as multiple blanks are converted to one (COMPBL) or no (COMPRESS) blanks.

```
data SpaceData2;
   set SpaceData;
   input SpaceText $20.;
   COMPB = COMPBL (SpaceText);
   COMPR = COMPRESS (SpaceText);
   L_ST = lengthn (SpaceText);
   L_COMPB = lengthn(compb);
   L_COMPR = lengthn(compr);
run;
```

Table 20

SpaceText	L_ST	COMPB	L_COMPB	COMPR	L_COMPR
NoSpaceData	11	NoSpaceData	11	NoSpaceData	11
One Space Data	14	One Space Data	14	OneSpaceData	12
Two Space Data	16	Two Space Data	14	TwoSpaceData	12
Three Space Data	20	Three Space Data	16	ThreeSpaceData	14

The COMPBL function reduced blanks in the "ThreeSpaceData" value by one, so instead of three space between each word, the program results in two blank spaces. COMPRESS, on the other hand, removes all embedded blanks.

REMOVING LEADING AND TRAILING BLANKS WITH THE TRIMN AND STRIP FUNCTIONS

The functionality is TRIMN and STRIP are easier to explain than to display. This example borrows a technique used by Ron Cody in his book "Learning SAS by Example, A Programmer's Guide" which I highly recommend. His book "Cody's Data Cleaning Techniques Using SAS" is also an invaluable resource and should be part of every programmer's toolkit.

The following program creates two new variables, both with leading and trailing blanks, as indicated in the "Combine" variable. It then creates three variables (Name 1-3) to demonstrate simple concatenation, the TRIMN and the STRIP functions, respectively.

```
data _null_;
   Length Combine $30.;
   First='   Ron   ';
   Last='   Walker   ';
   Combine=':' || First || Last || ':';
   Name1=First || Last;
   Name2=trimn(First) || " " || trimn(Last);
   Name3=strip(First) || " " || strip(Last);
   put Combine= /
      Name1=   /
      Name2=   /
      Name3=   /;
run;
```

```
191  data _null_;
192    Length Combine $30.;
193    First='   Ron   ';
194    Last='   Walker   ';
195    Combine=':' || First || Last || ':';
196    Name1=First || Last;
197    Name2=trimn(First) || " " || trimn(Last);
198    Name3=strip(First) || " " || strip(Last);
199    put Combine= /
200        Name1=   /
201        Name2=   /
202        Name3=   /;
203  run;

Combine=:   Ron      Walker   :
Name1=Ron      Walker
Name2=Ron    Walker
Name3=Ron Walker
```

Display 2. Log Demonstrating Removal of Leading and Trailing Blanks

The “Combine” variable confirms the presence of both leading and trailing blanks using colons as placeholders.

The “Name1” variable demonstrates that concatenation removes leading but not embedded or trailing blanks. Notice 6 embedded blanks (3 trailing from First, 3 leading from Last).

The “Name2” variable demonstrates that TRIMN removes leading blanks from both First and Last, but note that 3 trailing blanks from First have not been removed.

Finally, “Name3” confirms that STRIP effectively removed both leading and trailing blanks from the First and Last variables.

CONCLUSION

Blank values in data are often overlooked but can greatly impact program performance and report or modeling accuracy. I hope the above has provided a few useful tools, tips, tricks and techniques to confront the several challenges caused by the unexpected appearance of blank values in data. Thanks.

REFERENCES

Cody, Ron. 2007. *Learning SAS® by Example, A Programmer’s Guide*

Cody, Ron. 2017. *Cody’s Data Cleaning Techniques Using SAS®, Third Edition*

CONTACT INFORMATION

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

Ron Walker
Loughlin Consulting
310-710-2148
ron.walker.email@gmail.com
www.linkedin.com/in/Ron-Walker-LA