Paper TT-02-2015

# You Can’t Spell ‘Assume’ without S.A.S.

Mike Tangedal, Capella University

## Abstract

The world of SAS programming is fraught with assumptions.  One major assumption is to immediately apply fancy reporting SAS procedures directly to source data.  However, preventative programming logic applied during the data step also might fall prey to assumptions due to the whims of the Program Data Vector.  This paper serves to highlight the most frequent assumptions on the road from source data to reportable results.  Included is the saga of date formatting, default value overriding, importing and exporting files, table joins, variable length and format, as well as assumptions made from what is not shown in the data.  Through proactive SAS programming, assumptions can be eliminated resulting in the most credible reported results.

## Introduction

A long time ago in a university programming lab far, far away I sat in front of a monitor lit with green text and learned SAS.  Like so many before me, I was caught in the neophyte world of semicolon placement and self-entered data sets.  The procedure was named ‘ANOVA’ and although my education on rudimentary experimental design was satisfactory, the printed output on the green bar paper was not matching expected results.  So in the tradition of all aspiring apprentices, I queried upon the wisdom of the lab tech.  Upon hearing the passion of my logic and witnessing the error-free SAS log, she knowingly turned to the list of numbers under my ‘cards;’ statement.  “There it is”, she stated in a dry manner I found a bit too matter of fact.  “You typed in an ‘O’ – that should be a zero.”  “Well then”, I stammered, “How come SAS doesn’t know I meant zero and just fix that?”  “That’s your job” she said in an even more dry tone and walked away with the air of someone nearing daily tolerance of geeks.

The assumption an O/0 is a 0/O is in no way exclusive to SAS.  However - especially in the realm of data management, SAS programming is as fraught with assumption as any area of technical expertise.  This is through no fault of the SAS software system itself.  As data sets have progressed from kilobytes to gigabytes to terabytes, SAS software simply must make assumptions in order to continue to process data as quickly as possible.  But we all know what happens when we assume.

## Looks Can be So Deceiving

Let’s start at the beginning with the classic ‘PROC ANOVA’.  SAS was an incredibly powerful tool in the development of experimental design but that power came with great responsibility.  Allowing the programmer to perform all manner of transformations on experimental data also allowed for all manner of assumptions.  Here is an example of a One-Way Anova test that is fraught with peril.

```
data OneWay;
      input date result @@;
      cards;
   010182 12.0 020182 13.0 030182 12.5 040182 9.0 050182 14.3
   010182 10.0 020182 10.0 030182 10.0 040182 8.0 050182 8.0
   010182 14.0 020182 15.0 030182 16.0 040182 17.0 050182 18.0
   010182 14.0 020182 15.0 030182 16.0 040182 17.0 050182 18.0
   010182 11.0 020182 10.0 030182 9.0 040182 8.0 050182 7.O
;
      run;
proc anova;
      class date;
      model date=result;
      run;
```

A study of the code reveals no obvious coding or logic errors.  The mistakes here are all based on assumptions and the mistakes are indeed plentiful.  The most obvious can be found (and then easily corrected) from viewing the SAS Log.

```
NOTE: Invalid data for result in line 812 57-59.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8----
812           010182 11.0 020182 10.0 030182 9.0 040182 8.0 050182 7.0
date=50182 result=. _ERROR_=1 _N_=25
NOTE: SAS went to a new line when INPUT statement reached past the end of a line.
NOTE: The data set WORK.ONEWAY has 25 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Why is the value of 'result' missing for the last value?  Why did capital 'o' look so much like zero on those old green screen monitors?  Oh well, this is an easily fixed if not highly frustrating error.

Unfortunately this is just the beginning of the frustration with this code as the Anova procedure fails to run due to an error noting the variable 'date' is presumed to be continuous.  Alas, the variable 'date' does not contain SAS date values although from reading perception the variable most certainly should.  In fact a printout of the data set 'OneWay' reveals all the values of 'date' to be missing their leading zeroes.  Something here is seriously amiss.

## A SAS Date Is as a SAS Date Does

Of the multitude of SAS-related problems I have helped to solve, the most common conundrum is the notion between the appearance and value of SAS dates.  As SAS programmers, we are all aware that a SAS date is simply an integer representing the number of days since January 1, 1960.  However, since (most of us anyway) learned to interpret numbers written in traditional date formats as dates before we learned to program them into SAS, overcoming the natural presumption to interpret such numbers automatically as dates can be quite challenging.

For example, '02/04/06' is easily translated to February 4, 2006.  Then again, it just might mean April 2, 2006 or even April 2, 1906.  Our job as diligent SAS programmers is to provide the proper informat as well as format statements to negate these assumptions.  Input file data appearing as '020406' could also represent this date.  Thankfully SAS has all manner of informat and format options to read and display date values in any manner of options.  Why spend time converting text strings and integers to SAS data values when they can be read into SAS correctly from the input file?

SAS date manipulation can be as frustrating as SAS data interpretation in that the SAS functions dealing with time data do not conform directly to natural inclination.  Presiding over this problem is the notion of the difference between a SAS datetime value and a simple date value.  Without explicit direction, SAS stores date values in the most logical manner available.  Rarely is the occasion when the SAS decision engine should be fully trusted over that of the programmer.  Datetime fields are a prime example.  Get in the habit of running frequent 'proc contents' or other such simple data exploratory options on newly available data sets to determine if fields are stored as datetime fields or date fields.  Datetime fields can easily be converted to date fields using the datepart() function.

Take a simple example of date manipulation such as jumping ahead a week for a date variable or calculating how many weeks from one date to the next.  The 'intnx' and 'intck' are just as invaluable as dangerous resources for SAS date manipulation.  For example, here is some simple code to create a new field a week from a source date as well as calculate the difference in weeks between two dates.

```
data test;
       format notnextmon date9.;
```

```
      now = '04feb06'd;
      later = '05feb06'd;
      notnextmon=intnx('month',now,1);
      notweekdiff=intck('week',now,later);
      put notnextmon= notweekdiff=;
      run;
```

Here are the results from the SAS log: `notnextmon=01MAR2006 notweekdiff=1`
A month from now is not the start of next month and the next day is simply not a week from now. Alas, the 'intnx' function always returns the beginning of a time period and the 'intck' function sets the start of each new week as Sunday. Here's the SAS code to correctly note a month from now and the difference in weeks.

```
data test;
      format nextmon date9.;
      now = '04feb06'd;
      later = '05feb06'd;
      nextmon=intnx('month',now,1)+day(now)-1;
      weekdiff=round(intck('day',now,later)/7);
      put nextmon= weekdiff=;
      run;
```

Here are the results from the SAS log: `nextmon=04MAR2006 weekdiff=0`
The key to the solution is to understand the presumptions of the 'intnx' and 'intck' functions and to reduce the calculations to actual number of days instead of weeks and months that have arbitrary start dates. Next month is simple the start of next month plus the number of days minus one added to the current day of the month. The number of weeks is simply the difference in days divided by seven and then rounded to the nearest integer. The key to getting around SAS date assumptions is to be proactive in date interpretation and manipulation.

## Behind the Scenes at the PDV

Data set interaction is such a common and frequent practice in SAS programming that maintenance and documentation of all variables being added and altered can become cumbersome. Through the use of the Program Data Vector, SAS does its best to maintain all variables. However, the assumptions made by the PDV can cause unexpected results and errors. Maintaining control of data set interaction through simple statements such as 'keep=' and 'in=' can thwart the supposed best intentions by the PDV.

Discussions on merging data sets in all the popular SAS programming books rightfully focus on methodology and importance of finding a common key between data sets. Just as in dealing with SAS dates, the natural inclination to correctly match the data sets on the key can override the danger that is the common variables that may exist between the data sets. Take these two simple examples of data sets.

```
data master (keep=key name state);
      length name $10 state $4;
      do key = 1 to 10;
            name = cats('Mike',put(key,2.));
            state= cats('MN',put(key,2.));
            output;
            end;
      run;
data lookup;
      key=2; city='Minneapolis'; state='MN'; output;
      key=4; city='Detroit'; state='MI'; output;
      key=6; city='Chicago'; state='IL'; output;
      key=8; city='Kansas City, MO'; output;
      key=22; city='Bombay'; state='India'; output;
      run;
```

The first is a representative of a master data set with a unique key and some other clearly defined variables.  The lookup data set has the matching key variable as well as some other information that can be matched to the master.

First thing first – what is the glaring assumption from the lookup data set?  What are the default attributes of the variables if not explicitly stated through ‘length’ and ‘format’ statements?  What is the length of the text variable ‘city’?   Well, it is defined by its first assignment, so the length is 11.  Obviously this can be quite dangerous if any of the city names that follow are longer than 11 characters.  The same goes for the variable ‘state’ and even ‘key’ for that matter.  The point is this – taking the extra time to clearly define all variables in all data sets to be matched with a master data set will prevent the assumptions made by the PDV.

```
data master_lu;
       merge master (in=good) lookup;
       by key;
       if good;
       run;
```

Examination of the resulting data set (via a simple proc print) shows no immediate reason for concern.

```
Obs     name     state    key    city

  1     Mike1     MN1       1
  2     Mike2     MN        2    Minneapolis
  3     Mike3     MN3       3
  4     Mike4     MI        4    Detroit
  5     Mike5     MN5       5
  6     Mike6     IL        6    Chicago
  7     Mike7     MN7       7
  8     Mike8     IL        8    Kansas City
  9     Mike9     MN9       9
 10     Mike10    MN10     10
```

The results show the same ten observations from the Master table.  This is due to the use of the ‘in=’ parameter on the merge statement above.  This is a left join with a lookup table.  Joining two tables without regard to the type of join (left, right, full) is never a good idea. The key and the name field from the Master table are unchanged, a City variable has been added, and State was altered.  Was the intention to alter the State field from the Master table with the same field from the lookup table?  This wasn’t explicitly noted in the code so SAS pulled over all fields from the lookup table.  This clearly has potential for being a dangerous assumption.

Although examination of the City field above doesn’t appear to have any issues, what is not known is what is not stated.  The attributes of the new field on the Master table were derived from the attributes on the lookup table.  Observation 8 above has the value of ‘Kansas City’ for the City field.  This is the first 11 characters of the original field from the lookup table.  This is a convenient break but negative consequences could happen for other observations.  Also, what consideration should be made of those records from the Master table that did not match the lookup table?  Should a default value be assigned to the City field?  The PDV assumes any field from the Master table that did not match the lookup table should be assigned a blank value.

## Art Vandelay

As important as responsibility of control over the assumptions of the PDV when manipulating data sets, often this control is restricted by convenience.  Such is the case with using the powerful tools of Proc Import and Proc Export.  This is not to say the assumptions that lie within utilizing such powerful tools cannot be overcome.  However, before assuming the role of a professional importer/exporter, assumptions of Proc Import and Proc Export should be understood.

Within any professional organization a sizeable proportion prefers data to be organized in spreadsheet form.  For most, this means Microsoft Excel.  Thankfully SAS now works quite well with Excel although assumptions are made in the Proc Import/Export engine

1) To avoid confusion and translation error, keep all SAS dates in an easily recognizable format such as 'mmddyy8' or 'date9' format when importing from a spreadsheet or exporting to a spreadsheet.
2) Use the 'sheet' option on Proc Import if at all possible to read the correct tab within the spreadsheet.
3) Unless the entire spreadsheet tab is to be read and converted to a data set within Proc Import, used the 'range' option to specify which cells are to be read
4) If the text size within a particular column is large enough to warrant concern when using Proc Import, use the 'textsize' option
5) Use the 'scantext' option within Proc Import if text fields of various sizes may cause Proc Import to presume incorrectly.

I rarely use the 'scantext' option when reading Excel files since the SAS engine at best is making a good guess whereas I can create a completely accurate definition with just a bit of work.  Here is a paper that best describes this simple process: http://www2.sas.com/proceedings/sugi30/038-30.pdf

Reconfiguring the data step code created by Proc Import is not always the easiest solution.  There are ways to maximize the likelihood the information in the source spreadsheet will be reflected completely within the SAS data set

1) Format all date fields to mmddyy format and reformat any other fields that might look like dates so that SAS doesn't make the conversion by mistake.
2) If the source spreadsheet has blank rows, sort the data so that these rows do not appear at the top of the spreadsheet.  SAS creates field definitions based on the contents of the first fields read, so give SAS the best chance to create an accurate definition.
3) Remove header or peripheral information from the source spreadsheet that is not integral to SAS defining the data within.  The resulting spreadsheet should have the first row be only the names of each column and then all the data below.
4) Change the names of each column to valid SAS variable names.  Don't presume SAS will convert unmatched quotes and special symbols in column name headings to understandable text.
5) Remove any column/row formatting done for appearance only.  Most likely changes in appearance only will not affect the Proc Import engine, but the chance for inaccuracy still exists

## The Prettier the Report, the Less Attention Paid to the Numbers Within

The most recent updates to SAS software rightfully focus on the enhanced reporting capabilities since that is where the enticement to the core customer is based.  A SAS programmer of moderate experience can now progress from source data on almost any platform to impressive graphical reports with drill-down capacity in no time.  Alas, just with the old adage about people who assume, there is also a wise axiom regarding overreaching.  To paraphrase: just because SAS software can make a slick report from source data does not mean that SAS software **should** make a slick report from source data.

For this example, I will be using the data set available from the SAS help directory: class.  This data set consists of the name, age, sex, height, and weight of 19 students.  Here is a simple example of Proc Report to summarize this data, followed by the resulting report exported via simple ODS.

```
proc format;
      value agecat 0-12='preteen' 13-19='teen' 20-high='adult';
      run;
proc report data=sashelp.class nowd;
```

```
      column sex, age, (n height weight);
      define sex / across;
      define age /across format=agecat.;
      define n / format=comma6.;
      define height/analysis mean format=6.1;
      define weight/analysis mean format=6.1;
      run;
```

| Sex | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | | | | | | M | | | | | |
| Age | | | | | | Age | | | | | |
| preteen | | | teen | | | preteen | | | teen | | |
| n | Height | Weight | n | Height | Weight | n | Height | Weight | n | Height | Weight |
| 3 | 55.8 | 70.7 | 6 | 63.0 | 99.8 | 4 | 59.7 | 98.9 | 6 | 66.8 | 115.7 |

The report above is not pretty, but compact enough to convey a fair summary of the data set in minimal space. Note that although formatting was applied in the 'define' statements above, detailed labels and specific cell formatting was not done. Again, the best way to remove assumptions from SAS is to explicitly define variables with as much information as possible when provided with that opportunity. All in all, this is a nice little report free of most assumption.

Now I will dirty up the data a bit by clearing one value of each variable and then creating the same report again to see how SAS deals with missing values in Proc Report.

| Sex | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| F | | | | | | M | | | | | |
| Age | | | | | | Age | | | | | |
| preteen | | | teen | | | preteen | | | teen | | |
| n | Height | Weight | n | Height | Weight | n | Height | Weight | n | Height | Weight |
| 3 | 55.8 | 70.7 | 4 | 64.4 | 104.3 | 4 | 59.7 | 98.9 | 6 | 66.8 | 118.3 |

Well, the report is the same except that a few students got dropped. The only indication of this is from examination of the previous report as the SAS log matches that of the previous execution. According to the specifications from the code in the current version of Proc Report, everything is going perfectly.

Thankfully Proc Report has the means to deal with missing values and missing categories. Here is a modified version of the above Proc Report code with added options to deal with all sorts of missing scenarios.

```
proc report data=class nowd completerows;
      column sex, age, (n height weight);
      define sex / across missing;
      define age /across format=agecat. missing preloadfmt exclusive;
      define n / format=comma6.;
      define height/analysis mean format=6.1 nozero;
      define weight/analysis mean format=6.1 nozero;
      run;
```

| Sex | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | | F | | | | | | |
| Age | | | | | Age | | | | | | |
| adult | preteen | teen | | | adult | preteen | | | teen | | |
| N | n | n | Height | Weight | n | n | Height | Weight | n | Height | Weight |
| 0 | 0 | 1 | 56.5 | 84.0 | 0 | 3 | 55.8 | 70.7 | 4 | 64.4 | 104.3 |

The complete report from the altered Proc Report code isn't shown above, but enough is to illustrate the changes.

1) A whole new section of the report is created for records with a missing value for 'Sex'. Obviously this could be labeled as such to avoid confusion
2) The variable 'Age' is now categorized into three sections as determined by the 'completerows' option in the Proc Report statement and the 'preloadfmt exclusive' options for the define statement for 'Age'. Why isn't the missing value for Age also listed here? Because it is not part of the format definition. Note the classification of 'adult' is present in the report even though none of the data meet that criteria. Sometimes a report needs to list all possible classifications even though the data might not have all those classifications present.
3) The 'nozero' options for the define statements for 'height' and 'weight' will prevent printing of these values if all data for that cell contain missing or zero values. Note this is just the case for adults and preteens in the first section of the report so height and weight are not listed here.

## Conclusion

The Proc Report section above highlights a conundrum amongst SAS programmers provided the means to highlight assumptions made by SAS software. Who is to say which assumptions are noteworthy? In the report above segregated by Sex and Age, missing values for either variable are obviously noteworthy in that they compromise the whole of the report. But what of a missing height or weight value? If the source data had millions of records and just a few with missing height and weight data, proactive programming to exclude or segregate this data would not enhance the report. Most likely such reporting would distract from the intended focus.

The bottom line is that sometimes SAS assumptions are beneficial and sometimes they are detrimental. The implications all depend upon the scenario. Our job as SAS programmers is to possess an innate understanding of the base data far greater than is assumed by SAS in order to be proactive and thwart destructive assumptions while allowing SAS to assume away in less problematic areas.

## Recommended Reading

Most of the essential SAS wisdom I've learned has come from Art Carpenter. All of the ideas on Proc Report stem from this publication.

Carpenter, Art. 2007. *Carpenter's Complete Guide to the SAS® Report Procedure.* Cary; NC: SAS Institute Inc.

## Contact Information

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

Mike Tangedal
3116 41st Ave S
Minneapolis MN 55406
Phone: 612-747-3797
E-mail: Michael.tangedal@capella.edu