Paper 122-2013

Don't Let a Bad Date Ruin Your Day: Dealing with Invalid Dates in SAS

Lucheng Shao, University of California, Irvine

ABSTRACT

As we have to step out of fairy tale land and into reality when we grow up, we can't always expect the input dates to be good. This paper shows you what SAS does when it runs into input dates that are normally good but have now gone bad, and how those problems can be addressed by code. It is intended for readers who are familiar with SAS BASE but not with bad dates.

INTRODUCTION

What is a 'bad date'?
There is no strict definition of a bad date, however, for this paper we will consider any date that will lead to errors in SAS a bad date. A non-exhausted list of examples of bad dates could be:

1. Invalid dates
• Character in numeric fields (eg. 02/Sep/2011 instead of 02/09/2011)
• A date that can never exist (eg. 31/11/2006)
• A date that looks is invalid but may have an agreed upon valid meaning (eg. 99/99/9999 which might mean 'No End Date')

2. Unexpected input dates
• Dates where the data format can vary
eg. 03/04/2001
05/08/2011
11/09/05
08/14/1998

In this paper, we'll guide you through examples of different types of bad dates, what the log and output will look like, and how to solve those problems by code. We'll further discuss how to clear up the log when we are expecting bad dates during INPUT, and how to deal with and take advantage of invalid dates that have a meaning.

WHEN THE LOG TELLS YOU THAT YOUR INPUT DATES ARE GETTING SICK

As clinical manifestations (eg. headache, running nose, sore throat, etc.) may suggest that influenza viruses have invaded your cells, the log warns you when SAS encounters invalid input dates. As diagnoses, a warning NOTE will be triggered in the log; the offending variable will be set to missing, and the automatic error variable (_ERROR_) will be set to 1.

Example1 is the sample log where SAS reads in a date variable from INPUT1.DAT where there are two types of invalid dates: 31/11/2006 and 99/99/9999. In this example, the former one appeared twice. As you can see, the log will report errors repeatedly, even if it runs into exactly the same invalid date.

```
    466  filename indata 'C:\Documents and Settings\Lucheng
Shao\Desktop\INPUT1.DATt';
    467
    468  data invalid;
    469  infile indata;
    470  input row 2. EndDate ddmmyy10.;
    471  run;

    NOTE: The infile INDATA is:
          File Name=C:\Documents and Settings\Lucheng
Shao\Desktop\INPUT1.DAT,
          RECFM=V,LRECL=256

    NOTE: Invalid data for EndDate in line 3 3-12.
    RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6---
-+----7----+----8----+-
    3         3 31/11/2006 12
    row=3 EndDate=. _ERROR_=1 _N_=3
    NOTE: Invalid data for EndDate in line 4 3-12.
    4         4 99/99/9999 12
    row=4 EndDate=. _ERROR_=1 _N_=4
    NOTE: Invalid data for EndDate in line 5 3-12.
    5         5 31/11/2006 12
    row=5 EndDate=. _ERROR_=1 _N_=5
    NOTE: 5 records were read from the infile INDATA.
          The minimum record length was 12.
          The maximum record length was 12.
    NOTE: The data set WORK.INVALID has 5 observations and 2 variables.
    NOTE: DATA statement used (Total process time):
          real time           0.03 seconds
          cpu time            0.00 seconds
```

Example 1: Repeated invalid data notes

BE CAREFUL! INVALID DATES MAY NOT BE THE ONLY PATHOLOGIC FACTOR FOR THE DISEASE

A headache could be one pathological characteristic of a common cold, but could also be a complication of brain cancer. Medical misdiagnosis can result in unnecessary and serious consequences to patients. The log file is always one of the best places to check out to see if our program is 'healthy'. As you can imagine, if the log is full of warning notes reporting invalid dates, you will be less likely to detect other 'pathologic factors'.
For example, invalid dates will be set to missing by SAS as default, while other reasons could also result in missing values of a date variable.

In Example 2, INPUT2.DAT is a larger data file with similar date values as INPUT1.DAT; however it has a special entry in row 13 – 'not available'. Both invalid input dates (31/11/2006 and 99/99/9999) and 'not available' will be converted to missing during the input process, so we won't be able to distinct the censored case from the rest of invalid dates. Suppose our goal is to find those censored cases. Our first approach to accomplish this may be to scan the log file. For a small dataset this may be possible, but for a large dataset we will easily get lost in the forest of warning messages reporting invalid input dates.

```
    514  filename indata 'C:\Documents and Settings\Lucheng
Shao\Desktop\INPUT2.DAT';
    515
    516  data invalid;
```

```
    517  infile indata;
    518  input row 2. EndDate ddmmyy10.;
    519  run;

    NOTE: The infile INDATA is:
          File Name=C:\Documents and Settings\Lucheng Shao\Desktop\INPUT2.DAT,
          RECFM=V,LRECL=256

    NOTE: Invalid data for EndDate in line 3 3-12.
    RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----
7----+----8----+-
    3         3 31/11/2006 12
    row=3 EndDate=. _ERROR_=1 _N_=3
    NOTE: Invalid data for EndDate in line 4 3-12.
    4         4 99/99/9999 12
    row=4 EndDate=. _ERROR_=1 _N_=4
    NOTE: Invalid data for EndDate in line 5 3-12.
    5         5 31/11/2006 12
    row=5 EndDate=. _ERROR_=1 _N_=5
    NOTE: Invalid data for EndDate in line 8 3-12.
    8         8 31/11/2006 12
    row=8 EndDate=. _ERROR_=1 _N_=8
    NOTE: Invalid data for EndDate in line 9 3-12.
    9         9 99/99/9999 12
    row=9 EndDate=. _ERROR_=1 _N_=9
    NOTE: Invalid data for EndDate in line 10 3-12.
    10        10 31/11/2006 13
    row=10 EndDate=. _ERROR_=1 _N_=10
    NOTE: Invalid data for EndDate in line 11 3-12.
    11        11 31/08/2010 13
    row=11 EndDate=. _ERROR_=1 _N_=11
    NOTE: Invalid data for EndDate in line 12 3-12.
    12        12 01/11/2011 13
    row=12 EndDate=. _ERROR_=1 _N_=12
    NOTE: Invalid data for EndDate in line 13 3-12.
    13        13 Not available 16
    row=13 EndDate=. _ERROR_=1 _N_=13
    NOTE: Invalid data for EndDate in line 14 3-12.
    14        14 99/99/9999 13
    row=14 EndDate=. _ERROR_=1 _N_=14
    NOTE: Invalid data for EndDate in line 15 3-12.
    15        15 31/11/2006 13
    row=15 EndDate=. _ERROR_=1 _N_=15
    NOTE: 15 records were read from the infile INDATA.
          The minimum record length was 12.
          The maximum record length was 16.
    NOTE: The data set WORK.INVALID has 15 observations and 2 variables.
    NOTE: DATA statement used (Total process time):
          real time           0.03 seconds
          cpu time            0.01 seconds
```

Example 2: Multiple Invalid data notes

Sometimes the date variable itself may not even be your target variable (a variable which will be used in later data manipulation), but it may still cause problems. A take home message: When you notice that the log file is getting too many warning messages reporting invalid dates, it would be best to slow down. Mistakes may already have happened. Never assume missing values of a date variable are resulted from a single source. Clear up the log as much as you can to see the problem more clearly. The rest of the paper will show you how to prune the log file as needed.

SUIT THE REMEDY TO THE CASE

(1) THE DATE VARIABLE IS NOT A TARGET VARIABLE – MAKE THE LOG FILE AS CLEAR AS POSSIBLE

Sometimes we may not care why the date has a missing value and we will keep it in the dataset anyway. If that is the case, informat modifier ? or ?? can help us clear up the log file, so that we can see other problems more easily.

Both ? and ?? will suppress the NOTE: Invalid Data. The difference is that ?? will also suppress the automatic error variable being set to 1, while ? will not. Thus, ?? clears up the log more effectively. In both cases, the offending variable will be set to missing. Let's look at some sample log where ? or ?? is applied.

```
    548  filename indata 'C:\Documents and Settings\Lucheng
    Shao\Desktop\INPUT1.DAT';
    549
    550  data invalid;
    551  infile indata;
    552  input row 2. EndDate ? ddmmyy10.;
    553  run;

NOTE: The infile INDATA is:
      File Name=C:\Documents and Settings\Lucheng Shao\Desktop\INPUT1.DAT,
      RECFM=V,LRECL=256

RULE:     ----+----1----+----2----+----3----+----4----+----5----+----6----+----
7----+----8----+-
3         3 31/11/2006 12
row=3 EndDate=. _ERROR_=1 _N_=3
4         4 99/99/9999 12
row=4 EndDate=. _ERROR_=1 _N_=4
5         5 31/11/2006 12
row=5 EndDate=. _ERROR_=1 _N_=5
NOTE: 5 records were read from the infile INDATA.
      The minimum record length was 12.
      The maximum record length was 12.
NOTE: The data set WORK.INVALID has 5 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Example 3: The sample log where the informat modifier '?' is applied

Compared with Example 1, the sample log in Example 3 is much easier to visually inspect now as the message 'Note: Invalid data' is suppressed by applying the informat modifier '?'.

```
554  filename indata 'C:\Documents and Settings\Lucheng
Shao\Desktop\INPUT1.DAT';
555
556  data invalid;
557  infile indata;
558  input row 2. EndDate ?? ddmmyy10.;
559  run;

NOTE: The infile INDATA is:
      File Name=C:\Documents and Settings\Lucheng Shao\Desktop\INPUT1.DAT,
      RECFM=V,LRECL=256

NOTE: 5 records were read from the infile INDATA.
      The minimum record length was 12.
      The maximum record length was 12.
NOTE: The data set WORK.INVALID has 5 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Example 4: The sample log where the informat modifier '??' is applied

After substituting the informat modifier '?' with '??', the sample log in Example4 is entirely free of error messages. If there are other problems in your input they are easier to see now since you no longer have the clutter of the invalid date NOTEs.

Although the date variable might not be your target variable, we may still be interested in finding out what those invalid dates are. A good practice could be to write these values to the end of the log so they can be further analyzed, or sent back to the source for clarification. Example 5 will walk you through how to do this.

```
715  filename indata 'C:\Documents and Settings\Lucheng
Shao\Desktop\INPUT1.DAT';
716
717  data invalid;
718  infile indata;
719  input row 2. EndDate $10.;
720  cDate= input(EndDate, ?? ddmmyy10.);
721  format cDate ddmmyy10.;
722  if cDate=. then put row= EndDate=;
723  run;

NOTE: The infile INDATA is:
      File Name=C:\Documents and Settings\Lucheng Shao\Desktop\INPUT1.DAT,
      RECFM=V,LRECL=256

row=3 EndDate=31/11/2006
row=4 EndDate=99/99/9999
row=5 EndDate=31/11/2006
NOTE: 5 records were read from the infile INDATA.
      The minimum record length was 12.
      The maximum record length was 12.
NOTE: The data set WORK.INVALID has 5 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.00 seconds
```

Example 5: Invalid dates attached to the log

Line 719 reads the input into a character variable (EndDate). Line 720 uses the input() function to covert EndDate into a numeric variable (cDate). Any missing value of this numeric variable serves as an indicator for invalid dates. As we have learned from Example4, '??' suppresses any error messages. Line 722 outputs those invalid dates which have caused cDate to be missing to the end of the log.

(2) THE DATE VARIABLE IS A TARGET VARIABLE –HAVE TO DEAL WITH INVALID DATES

A common way to treat REAL invalid dates is to convert a date like '31/11/2006' into missing. You might ask: What is a REAL invalid date? A REAL invalid date is a date that can never exist. For example, we will never have 31st in November for whatever year.

On the other hand we can have a FAKE invalid date. An example of a FAKE invalid date could be a value such as 99/99/9999; in many instances a value like this represents 'No End Date'. We could apply informat modifier ? or ?? to FAKE invalid dates to suppress warning messages, as we do to REAL invalid dates. The cost of doing so, however, is that we will lose valuable information or the ability to use the implicit value. Consider such as example: If the application requires a visitDate which is between a startDate and an endDate, we need to have a valid value in endDate; if the value of 99/99/9999 represents 'No End Date', we could assign an arbitrary future date. As in Example 6, 12/31/2099 is chosen.

```
filename indata 'C:\Documents and Settings\Lucheng Shao\Desktop\INPUT1.DAT';

data invalid;
infile indata;
input row 2. EndDate $10.;
cDate= input(EndDate, ?? ddmmyy10.);
drop EndDate;
format cDate ddmmyy10.;
if cDate=.
then
  do;
    if EndDate='99/99/9999'
       then cDate='31Dec2099'd;
  end;
run;
```

Example 6: 'No End Date' is converted to an arbitrary future date using IF statement

Another approach to handle the problem of '99/99/9999' is to build an INFORMAT with PROC FORMAT. A benefit from doing so is that the created INFORMAT can be saved permanently and being applied to input dates in any other dataset even after your current SAS session is ended. In Example 7, a new informat called '$spe' is created by the INVALUE statement and saved in a library called 'project'. Applied with this informat, any date like '99/99/9999' among the input will be converted to '31/12/2099'.

```
libname project 'C:\Documents and Settings\Lucheng Shao\Desktop';
proc format library=project;
invalue $spe '99/99/9999'='31/12/2099'
             ;
run;

filename indata 'C:\Documents and Settings\Lucheng Shao\Desktop\INPUT1.DAT';
data invalid;
infile indata;
input row 2. EndDate $spe.;
drop EndDate;
cDate=input(EndDate, ddmmyy10.);
format cDate ddmmyy10.;
run;
```

Example 7: 'No End Date' is converted to an arbitrary future date using user-defined informat

WHAT IF THE PROBLEM IS CHARACTERS IN NUMERIC FIELDS?

Simply converting characters back into numeric form sounds reasonable. Be careful! There could be something tricky here. Let's see what could happen in Example 8. Suppose INPUT3.DAT is our input:

```
filename indata 'C:\Documents and Settings\Lucheng Shao\Desktop\INPUT3.DAT';

data invalid;
infile indata;
input row 2. EndDate $11.;
run;
```

Example 8: Characters in numeric fields

Let's take a look at the output file using PROC PRINT.

```
                        Obs    row       EndDate

                         1      1     02/Sep/2011
                         2      2     3 06/07/200
```

It turned out to be much different than what we've expected. The first observation was read in correctly, but anything after the second value of the variable 'row' was wrong. The reason is that last field on the input file is text and the file does not have fixed length, i.e. the spaces that would normally be used to pad out the length of the input line are not included. In this example, SAS was expecting to read in two variables, 'row' which is 2 characters long, and 'EndDate', which is 11 characters long. The error occurred when SAS run out of things to read on the second line, because '31/08/2010' is only 10 characters long rather than 11. Thus, SAS went to the next line to grab a string with 11 characters, which happened to be '3 06/07/200'. After that, it encountered the same problem when it was trying to read in the fourth line.

An option called PAD on the INFILE statement is valuable when such error occurs.

```
infile indata PAD;
```

Basically what PAD does is it adds blanks to pad out the length of the input line so that each line will reach its default length (256) in SAS. After adding this PAD option, the output of Example 8 looks correct. Then, we can simply use IF statement to convert characters back into numeric form.

```
filename indata 'C:\Documents and Settings\Lucheng Shao\Desktop\INPUT3.DAT';

data invalid;
infile indata PAD;
input row 2. EndDate $11.;
if EndDate='02/Sep/2011' then EndDate='02/09/2011';
run;
```

Example 8 Continued: Using PAD option

WHAT IF THE PROBLEM IS INPUT DATES WITH UNEXPECTED DATA FORMAT?

A common example could be sometimes the raw data has a four digit year and sometimes it has a 2 digit year (eg. INPUT4.DAT). One possible way to solve this kind of problem is to go back to the original dataset and make all the

input dates consistent. However, this becomes infeasible when the dataset becomes large. A more efficient way is described in Example 9.

```
filename indata 'C:\Documents and Settings\Lucheng Shao\Desktop\INPUT4.DAT';

data invalid;
infile indata length=x;
input row 2. @;
length cDate $10;
format EndDate ddmmyy10.;
drop cDate;
put x=;
if x=10
  then do;
      input cDate $8.;
         EndDate=input(cDate, ?? ddmmyy8.);
         if EndDate=.
         then do;
                if cDate='99/99/9999'
                  then EndDate='31Dec2050'd;
              end;
       end;
else if x=12
   then do;
      input cDate $10.;
         EndDate=input(cDate, ?? ddmmyy10.);
         if EndDate=.
         then do;
                if cDate='99/99/9999'
                  then EndDate='31Dec2050'd;
              end;
       end;
run;
```

Example 9: Input dates with unexpected format – take advantage of line length

Here's the trick. We assigned the input line length to a variable called 'x', which serves as an indicator. The trailing @ is a place holder. After reading in the 'row' variable, SAS will assign strings with difference length to the second variable 'cDate', depending on the value of the indicator variable 'x'.

Be cautious with those blanks that are added to the end of an input line. In Example 9, if we add one blank to the end of the first input line, it will result in a missing value for EndDate in the output. Interestingly, if two blanks are added to the end of the third input line, we will get correct EndDate output as the line length becomes 12 now. SAS will convert '05' to '2005' automatically.

CONCLUSION

This paper showed several types of invalid input dates and how to deal with them correspondingly. It also described a way to take advantage of a special case of invalid dates, which is 'No End Date'. There may be other approaches. However, what really matters is that we have to keep in mind that invalid input dates may occur from time to time. It will always be a good habit to be cautious with all input dates and check out the reasons for any missing value of a date variable among the output.

ACKNOWLEDGEMENTS

I would like to thank Peter Eberhardt for all his support and advice in editing this paper.

CONTACT INFORMATION

Lucheng Shao
Donald Bren School of Information
and Computer Sciences
University of California, Irvine
6210 Donald Bren Hall
Irvine, CA, 92617-3425
Email: luchengs@uci.edu

LISTINGS

INPUT1.DAT
1 31/08/2010
2 01/11/2011
3 31/11/2006
4 99/99/9999
5 31/11/2006

INPUT2.DAT
1 31/08/2010
2 01/11/2011
3 31/11/2006
4 99/99/9999
5 31/11/2006
6 31/08/2010
7 01/11/2011
8 31/11/2006
9 99/99/9999
10 31/11/2006
11 31/08/2010
12 01/11/2011
13 NA
14 99/99/9999
15 31/11/2006

INPUT3.DAT
1 02/Sep/2011
2 31/08/2010
3 06/07/2009
4 24/11/2007

INPUT4.DAT
1 03/04/2001
2 05/08/2011
3 11/09/05
4 14/08/1998
5 99/99/9999
6 31/11/2006