PharmaSUG2016 - Paper BB-06

Superior Highlighting:

Identifying New or Changed Data in Excel Output using SAS®

Kim Truett, KCT Data, Inc., Alpharetta, GA

ABSTRACT

Non-SAS®-users often review data in Excel. As these outputs grow larger, the ability to identify new or changed data is invaluable to the reviewer. This is easily identified in SAS using procedures such as PROC COMPARE. However, it is not obvious how to automate representation of this in Excel. Just prior to PharmaSUG 2013, I was approaching a solution to get SAS to automate this but could not figure out the final steps. So I took this puzzler to Coders Corner. During the hour that followed, John King, Gary Moore, and I devised a solution. This paper is a presentation of that brainstorming session.

This paper will show how a SAS program can be used to automatically identify new or changed data using rtf codes that change the color of the cell background to indicate new or changed data in an Excel output.

INTRODUCTION

Excel output is a very accessible way to provide data to non-SAS users and is often provided on a regular (weekly or monthly) basis. As these files grow larger, it becomes very useful to be able to highlight any data that is new or changed, in order to expedite the clinical or medical review. Because SAS outputs to Excel, and we can export formats as well, this paper explores one way to do that. The basic method is to attach flags to the rows and data points to indicate what is new or changed, and then to use a series of CALL DEFINES to actually export these flags into Excel formats using the REPORT procedure.

BODY

For this paper, there are four types of data that are going to be highlighted:

- new data
- changed data
- values over a threshold
- change from baseline over a threshold

Step 1: Presumably, you start with a pre-existing data set. For the purpose of this paper, sample code is provided at the end to create the data set used, so the reader may generate data to play with the code. The creation of that data set is not discussed here, as it is very straightforward.

Step 2: Calculate change from baseline. Even when just a simple listing of data is requested, this is an easy, but very useful calculation that you can provide in the listing (adding extra value to your services). Again, the code for this is presented below, but is very simple, and is not discussed within this paper.

Step 3: Produce a FINAL data set with only the needed variables, and your calculated change from baseline. This data set should also be saved to a permanent SAS data set, with a date in the file name for later retrieval. This system for naming the data sets lets you compare today's data to data from last week, last month, or even last year's annual report. These previous versions, plus the newly generated current version, should be saved in a specific folder.

```
%let prevdate = 20160401;
%let datesent = 20160501;

data crf_prev.final_IOP_&datesent.;
    set final;
    run;
```

The MACRO variable *prevdate* contains the date of the previously saved FINAL data set, which will be used when the program compares the old FINAL data set to the current FINAL dataset. The MACRO variable *datesent* contains the date of the current FINAL data set, which is used to create the permanent copy of the FINAL data set, which will be used in the future.

View of FINAL data set:

subjid	visit-num	fo-cid	iop_b	Vst2	Vst3	Vst4	Vst5	CFB_2	CFB_3	CFB_4	CFB_5
101	1	OD	17	18	19	20	21	1	2	3	4
101	1	OS	10	9	8	7	6	-1	-2	-3	-4
102	1	OD	18	19	20	21	22	1	2	3	4
102	1	OS	11	10	9	8	7	-1	-2	-3	-4
103	1	OD	19	20	21	22	23	1	2	3	4
103	1	OS	12	11	10	9	8	-1	-2	-3	-4
104	1	OD	20	21	22	23	24	1	2	3	4
104	1	OS	13	12	11	10	9	-1	-2	-3	-4
105	1	OD	21	22	23	24	25	1	2	3	4
105	1	OS	14	13	12	11	10	-1	-2	-3	-4

Step 4: Flag any entirely new rows. To do this, merge the previous version of the data back into the FINAL data set, and, using the IN data set option on the DATA step, set a flag ("newrow") for any rows that are present in the FINAL data set, but not in the previous data set..

```
data newrowflag;
    merge final (in=ina) crf_prev.final_IOP_&prevdate. (in=inb keep = subjid focid);
    by subjid focid;
    * set a flag for any new row;
    if ina and not inb then newrow='y';
    run;
```

The data now looks like this, with the newrow variable added which flags the two new rows added to the data.

subjid	visit-num	fo-cid	iop_b	Vst2	Vst3	Vst4	Vst5	CFB_2	CFB_3	CFB_4	newrow
101	1	OD	17	18	19	20	21	1	2	3	
101	1	OS	10	9	8	7	6	-1	-2	-3	
102	1	OD	18	19	20	21	22	1	2	3	
102	1	OS	11	10	9	8	7	-1	-2	-3	
103	1	OD	19	20	21	22	23	1	2	3	
103	1	OS	12	11	10	9	8	-1	-2	-3	
104	1	OD	20	21	22	23	24	1	2	3	
104	1	OS	13	12	11	10	9	-1	-2	-3	
105	1	OD	21	22	23	24	25	1	2	3	y
105	1	OS	14	13	12	11	10	-1	-2	-3	y

Step 5: Flag any data that has changed. Use the COMPARE procedure to compare the current FINAL data set to the previous data set. Output the results of the PROC COMPARE to a data set and then transpose that data set.

The options used on the proc compare are:

outdif: Write an observation that contains the differences for each pair of matching observations
outnoequal: Suppress the writing of observations when all values are equal
noprint: Suppress all printed output (so output is only to the OUT= data set).

```
proc compare base=crf_prev.final_IOP_&prevdate.
    compare=final out=check1
    outdif outnoequal noprint;
    id subjid focid;
    run;
```

The proc COMPARE output file looks like this: The value is set to 'E' when the values are equal; set to missing when the

value is new; and shows the actual value when there is a difference between the two data sets being compared.

Type of Observation	Observation Number	subjid	focid	visitnum	iop_b	Vst2	Vst3	Vst4	Vst5	CFB_2	CFB_3	CFB_4	CFB_5
DIF	1	101	OD	E	E	E	E	E	.	E	E	E	.
DIF	2	101	OS	E	E	E	E	E	.	E	E	E	.
DIF	3	102	OD	E	E	E	E	E	.	E	E	E	.
DIF	4	102	OS	E	E	E	E	E	.	E	E	E	.
DIF	5	103	OD	E	E	E	10	E	.	E	9	E	.
DIF	6	103	OS	E	E	E	E	E	.	E	E	E	.
DIF	7	104	OD	E	E	E	E	E	.	E	E	E	.
DIF	8	104	OS	E	E	E	E	E	.	E	E	E	.

The next step is to read the proc compare data set back into SAS and transpose it, so that the rows become variables that can be merged back into the data.

```
proc transpose data=check1 out=check2(where=(upcase(_name_) ne 'subjid'));
    by subjid focid;
    var _all_;
    run;
```

Proc transpose output file (just one subject, one eye shown here, but contains subsequent rows for all subjects and eyes:

subjid	focid	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	COL1
101	OD	_TYPE_	Type of Observation	DIF
101	OD	_OBS_	Observation Number	1
101	OD	subjid		101
101	OD	focid		OD
101	OD	visitnum		E
101	OD	iop_b		E
101	OD	Vst2		E
101	OD	Vst3		E
101	OD	Vst4		E
101	OD	Vst5		.
101	OD	CFB_2		E
101	OD	CFB_3		E
101	OD	CFB_4		E
101	OD	CFB_5		.

Using the COL1 flags from PROC COMPARE, we now create one flag variable for each data set variable by appending the variable name to the word “FLAG”.

```
data check3;
    set check2;
    * setting flags;
    if compress(col1,'.') eq ' ' then flag=1;
    else if compress(col1,'E') eq ' ' then flag=0;
    else flag=2;
    length nameid $32;
    * creating one flag for each variable;
    nameid = cats('FLAG_',_name_);
```

```
    run;
```

The resulting data is the transposed again to create observations, one per subjid focid with the new nameid (FLAG_varname) as the variable, and the contents of flag as the value.

```
proc transpose data=check3 out=check4;
    by subjid focid;
    id nameid;
    var flag;
    run;
```

That data set looks like this.

subjid	fo-cid	NAME OF FOR-MER VARI-ABLE	FLAG_ _TYPE_	FLAG_ _OBS_	FLAG_subjid	FLAG_fo-cid	FLAG_vis-itnum	FLAG_iop_b	FLAG_Vst2
101	OD	flag	1	1	1	1	0	0	0
101	OS	flag	1	1	1	1	0	0	0
102	OD	flag	1	1	1	1	0	0	0
102	OS	flag	1	1	1	1	0	0	0
103	OD	flag	1	1	1	1	0	0	0
103	OS	flag	1	1	1	1	0	0	0
104	OD	flag	1	1	1	1	0	0	0
104	OS	flag	1	1	1	1	0	0	0

FLAG_Vst3	FLAG_Vst4	FLAG_Vst5	FLAG_CFB_2	FLAG_CFB_3	FLAG_CFB_4	FLAG_CFB_5
0	0	1	0	0	0	1
0	0	1	0	0	0	1
0	0	1	0	0	0	1
0	0	1	0	0	0	1
2	0	1	0	2	0	1
0	0	1	0	0	0	1
0	0	1	0	0	0	1
0	0	1	0	0	0	1

Now those flags get merged back into the original data set.

```
data check5;
   merge newrowflag (in=ina) check4;
   by subjid focid;
   if ina;
run;
```

The resulting data set contains the data plus a flag (newrow) for any new rows, and a flag for each variable that is set to 2, if the value is changed; set to 1, if the value is new; and set to 0, if the value is unchanged.

Step 6: Set up the Excel output. To prepare to output the data in Excel, the FORMAT procedure is used to define color formats are used to highlight significant data values. When these formats are applied in the DEFINE statements of the PROC REPORT procedure, the reported values that are less than or equal to 8 will have LightYellow cell backgrounds, and reported values that are greater than or equal to 22 will have LightRed cell backgrounds. Change from baseline values that are less than or equal to -4 will have SpringGreen cell backgrounds, and change from baseline values that are greater than or equal to 3 will have LightGreen cell backgrounds.

```
Proc format;
Value sigbase
  low-<8   = 'LightYellow'
  >22-high = 'LightRed';
Value sigchg
  low-<-4 = 'SpringGreen'
  >3-high  = '#90EE90'; *Hex code for LightGreen to show that names or hex code is ok;
run;
```

After this, the general format of the Excel output is specified.

```
options center orientation = landscape nomlogic nosymbolgen nomprint;

ods listing close;
ods tagsets.ExcelXP path= "C:\Excel Output" file= "IOP Changes_&sysdate..xls" style=
XLsansPrinter;

ods tagsets.ExcelXP options(sheet_name= "IOP"
        autofit_height = "yes" autofilter="all"
        width_fudge = "0.7" ROW_REPEAT='1'
        orientation='landscape' COLUMN_REPEAT='1-2'
        frozen_headers= "1");
```

Step 7: COMPUTE blocks are used in PROC REPORT to format the rows.

```
Proc report data=check5 nowd;
    column newrow subjid flag_: focid iop_b Vst2 Vst3 Vst4 Vst5 CFB_2 CFB_3 CFB_4 CFB_5;
    define subjid / display style(column)=[just=center cellwidth=1.1in] 'Subject';
    define newrow / noprint;
    define flag_: / noprint display;
    define focid / display style(column)=[just=center cellwidth=1.1in] 'Eye';
    define iop_b / display style(column)=[just=center cellwidth=1.1in] 'Baseline IOP';
    define Vst2 / display style={just=center cellwidth=1.1in background=sigbase.};
    define CFB_2  / display style(column)=[just=center cellwidth=1.1in background=sigchg.];
    Etc., for Vst3 – Vst5 and CFB_3 - CFB_5
```

Newrow and the flags are on the COLUMN statement, and are defined with the NOPRINT option, so these will not actually print in the output. Rather they are used in the COMPUTE blocks to set the formatting of the rows.

The COMPUTE block uses CALL DEFINE to set the background color to LightBlue in the style statement, if newrow='y';

```
    compute newrow;
    /* this sets background color for any new rows */
        if NEWROW = 'y' then do;
            call define(_row_, "style", "style=[background=LightBlue]");
        END;
    endcomp;
```

The flag variables for each data point are also on the COLUMN statement and have a DEFINE statement with NOPRINT. These flags are processed one at a time to change the background color to LightBlue if the value is new or a slightly darker blue, SteelBlue, if the value is changed from the previous data.

```
    compute iop_b;
    /* this sets background color for any new data 1 = new 2 = changed */
        if FLAG_iop_b = 1 then
            call define(_col_, "style", "style=[background=LightBlue]");
        if FLAG_iop_b = 2 then
            call define(_col_, "style", "style=[background=Steelblue]");
    endcomp;
```

This is repeated for every variable:

```
       compute Vst2;
       /* this sets background color for any new data 1 = new 2 = changed*/
             if FLAG_Vst2 = 1 then
                   call define(_col_, "style", "style=[background=LightBlue]");
             if FLAG_Vst2 = 2 then
```

```
                call define(_col_, "style", "style=[background=Steelblue]");
    endcomp;
```

Note that the COMPUTE blocks are run for the variable of interest, not the flag variable, since that is the variable for which we want to change the background color. Also, within the CALL DEFINE, the individual column is highlighted (_COL_) not the entire row (_ROW_).

When the data is outputted to Excel without these flags, it looks like this:

A	B	C	D	E	F	G	H	I	J	K
Subjec	Eye	Baseline IOF	Vst2	Vst3	Vst4	Vst5	CFB_2	CFB_3	CFB_4	CFB_5
101	OD	17	18	19	20	21	1	2	3	4
101	OS	10	9	8	7	6	-1	-2	-3	-4
102	OD	18	19	20	21	22	1	2	3	4
102	OS	11	10	9	8	7	-1	-2	-3	-4
103	OD	19	20	21	22	23	1	2	3	4
103	OS	12	11	10	9	8	-1	-2	-3	-4
104	OD	20	21	22	23	24	1	2	3	4
104	OS	13	12	11	10	9	-1	-2	-3	-4
105	OD	21	22	23	24	25	1	2	3	4
105	OS	14	13	12	11	10	-1	-2	-3	-4

But when we add the flags to highlight new data, changed data and outliers, the reviewer now sees this – clearly showing new data, changed data, as well as highlighting outliers (lightblue is new data; steelblue (darker blue) is changed data; red is high abnormal, and yellow is low abnormal).

A	B	C	D	E	F	G	H	I	J	K
Subjec	Eye	Baseline IOF	Vst2	Vst3	Vst4	Vst5	CFB_2	CFB_3	CFB_4	CFB_5
101	OD	17	18	19	20	21	1	2	3	4
101	OS	10	9	8	7	6	-1	-2	-3	-4
102	OD	18	19	20	21	22	1	2	3	4
102	OS	11	10	9	8	7	-1	-2	-3	-4
103	OD	19	20	21	22	23	1	2	3	4
103	OS	12	11	10	9	8	-1	-2	-3	-4
104	OD	20	21	22	23	24	1	2	3	4
104	OS	13	12	11	10	9	-1	-2	-3	-4
105	OD	21	22	23	24	25	1	2	3	4
105	OS	14	13	12	11	10	-1	-2	-3	-4

Bonus highlighting

The compute blocks can also be used to highlight change from baseline using a compute block. This code contains the original code for setting the background color for new data, but adds a computation and a second set of call defines to highlight percent change from baseline.

```
  compute CFB_2;
      call missing(pcfb);
      if cmiss(iop_b, CFB_2) = 0 then do;
            pcfb = CFB_2 / baseline;
      end;

      /* this sets background color for any new data */
      if FLAG_CFB_2 = 1 then
                  call define(_col_, "style", "style=[background=lightblue]");

      /* this sets background color for any change over a specific threshhold */
      else if . < pcfb <= -0.15 then do;
                  call define(_col_, "style", "style=[background=pink]");
      end;
```

```
      else do;
            call define(_col_, "style", "style=[background=lightyellow]");
      end;
  endcomp;
```

The end result adds one more possible set of highlights. While at first glance, this seems kaleidoscopic, by selecting the flags from among these, that best serve your audience, review of data can be simplified.

A	B	C	D	E	F	G	H	I	J	K
Subjec	Eye	Baseline IOP	Vst2	Vst3	Vst4	Vst5	CFB_2	CFB_3	CFB_4	CFB_5
101	OD	17	18	19	20	21	1	2	3	4
101	OS	10	9	8	7	6	-1	-2	-3	-4
102	OD	18	19	20	21	22	1	2	3	4
102	OS	11	10	9	8	7	-1	-2	-3	-4
103	OD	19	20	21	22	23	1	2	3	4
103	OS	12	11	10	9	8	-1	-2	-3	-4
104	OD	20	21	22	23	24	1	2	3	4
104	OS	13	12	11	10	9	-1	-2	-3	-4
105	OD	21	22	23	24	25	1	2	3	4
105	OS	14	13	12	11	10	-1	-2	-3	-4

Conclusion

Using the call define statements to append rtf formatting codes to Excel output simplifies the task of reviewing data by showing what data is new, what data is changed, and any data that is unexpected.

This could be changed to highlight AEs or medication of interest, as well as specific subjects – once appropriate flags are calculated and attached to the data, the use of this method to attach the rtf codes will work for any type of flagged data.

Generating the data used in this paper

The following code was used to generate the data used in this paper:

```
data IOP0;
do subjid = 101 to 105;
  do visitnum = 1 to 5;
       focid = 'OD';
       IOP = subjid - 85 + visitnum;
       output;
       focid = 'OS';
       IOP = subjid - 90 - visitnum;
       output;
  end;
end;
run;
proc sort; by subjid visitnum focid;
run;
```

This code generates the current data.

One way to calculate and attach change from baseline to the data.

```
proc transpose data=iop0 (where=(visitnum ^= 1)) out=iop0_t prefix=Vst;
  by subjid focid;
  id visitnum;
  var iop;
run;

data all;
 merge iop0 (where=(visitnum = 1) rename=(iop=iop_b))
            iop0_t ;
by subjid focid;
```

```
run;

data final;
 set all;
 array vst(4) Vst2 Vst3 Vst4 Vst5;
 array cfb(4) CFB_2  CFB_3  CFB_4 CFB_5;
 do i = 1 to 4;
    cfb(i) = vst(i) - iop_b;
 end;
 drop i _NAME_;
 run;
```

To generate the previous version of the data, the last two observations were deleted, and some of the data values were changed, then that data set was saved as a data set named final_IOP_20160401.

References

Contact Information

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

Kim Truett
11585 Jones 11877 Douglas Rd, Ste 102-146
Alpharetta, GA 30022
770.372.0989
Email: KCTData03@kctdm.com