Paper PP21

Validation Tools Assemble - Proc Compare No Longer Enough to Save the Day On Its Own!

Eoin Byrne, Plus-Project Partnership, Leeds, UK

Peter Crumey, Plus-Project Partnership, Newcastle upon Tyne, UK

ABSTRACT

Statistical Analysis of any form relies heavily on the quality of the data, and this is paramount in clinical trials to ensure patient safety. This then puts a greater emphasis on the validation, as to ensure that all interested parties can trust the analysis. When a complete match happens and we see “NOTE: No unequal values were found. All values compared are exactly equal” it is easy to say that that is enough, but this is not always the case when using PROC COMPARE in SAS ® as not all complete matches are created equal.

```
Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 26.

NOTE: No unequal values were found. All values compared are exactly equal.
```

INTRODUCTION

A concern that can occur is during the production of datasets and their complex derivation of results. These derivations may be vastly different yet yield the same results for the wrong reason. While the data is seen as clean, the data’s integrity is in question and may still go unchecked due to a clean compare.

This paper intends to demonstrate the additional methods we can implement alongside the use of PROC COMPARE, to ensure data quality, integrity, and accuracy, as well as show the shortcomings of solely relying on this essential tool.

BENEFITS OF PROC COMPARE

PROC COMPARE is an essential tool for validation and of double programmed work, or to check the changes that have occurred between updated datasets. The code is universally understood and produces vast amounts of information that allow you to check the difference between thousands of records extremely quickly. A compare can be produced with minimal coding and a straightforward compare often used to check for new data within a dataset would need not much more than

```
proc compare base=base_data comp=new_data; run;
```

ID OPTION

Additionally, more options can be added to our compare to make it sophisticated and fit for purpose surrounding the data issues that we have. With the addition of the *ID* variables option, this allows for much easier debugging of the dataset and allows us to specify by which variables we want to compare the dataset. It is important to also remember to use the *LISTALL* option as using ID vars without this can lead to one of the biggest PROC COMPARE failures, i.e. thinking all is fine when actually ID values completely mismatch so nothing was actually compared!

We can see that when we add the ID variables, each ID variable is listed for any mismatches. This also allows us to ensure that when looking into debugging for these mismatches, we see exactly which result and not just an

observation number, which ensures the correct records are being compared, and tells you the ID variables rather than the obs number (which may be different between the two datasets).

```
proc compare base    = prod_advs
             compare = qc_advs listall;
             id studyid usubjid paramcd avisit ;
run;
```

```
Observation      Base  Compare  ID


First Obs          1        1  STUDYID=212895 USUBJID=206713.001101 PARAMCD=VS0006 AVISIT=Baseline
Last  Obs        168      168  STUDYID=212895 USUBJID=213744.000881 PARAMCD=VS0105 AVISIT=Day 1
```

WITH AND VAR STATEMENTS

We can then take this further by using *WITH* and *VAR* statements. These statements allow us to compare the base dataset with variables that have different names in the comparative data set.

We need to specify the variables in the base dataset manipulating the *VAR* statement and specify the matching variables using the *WITH* statement. If the *WITH* statement list is shorter than the *VAR* statement list, then PROC COMPARE assumes that the extra variables in the *VAR* statement have the same names across both datasets. A variable name can appear any number of times in the *VAR* statement or the *WITH* statement. By selecting *VAR* and *WITH* statement lists, you can compare the variables in any permutation.

Below is an example of how to use the *WITH* and *VAR*. We randomly create 200 results between 0 and 5 for two variables. We can then compare these variables, with differing names, and see how many match.

```
data base_data;
  do i = 1 to 200;
    predict=round(rand("uniform")*5,1);
    output;
  end;
run;

data act_data;
  set base_data;
  rename predict=actual;
run;
```

```
proc compare base=base_data comp=act_data;
  var predict;
  with actual;
run;
```

```
Number of Observations in Common: 200.
Total Number of Observations Read from WORK.BASE_DATA: 200.
Total Number of Observations Read from WORK.ACT_DATA: 200.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 200.

NOTE: No unequal values were found. All values compared are exactly equal.
```

Above we can see the compared results and we can see that all of our results are matching. This sort of compare works really well for the comparing between an actual and a predicted result.

DRAWBACKS

Simple PROC COMPAREs can often still give the sought after "NOTE: No unequal values were found. All values compared are exactly equal". Obvious issues, such as missing variables, missing observations, conflicting variable types, and mismatching ID Variables can all still produce the line that can be carelessly interpreted to mean that the data is matching. But even a more thorough programmer may have all his PROC COMPARE functionality right and all the results may be matching but the data isn't necessarily being produced the right way.

Derived results are the most common example of where results can be the same, but the derivation can give vastly different results depending on the inputted data. Complex derivations often yield the same results as simple derivations depending on the numbers that are being inputted.

During initial programming there is often the time to find these mismatches if they do occur, however during pre-programming there is often minimal amounts of data. When large amounts of data is available, these deviations in derivation occur. Unfortunately, during this time, work is usually expected much quicker as the initial programming has already been performed.

There are a number of solutions that can be implemented to check whether or not our newly derived results match our specs. One solution is programmatic check, while the other is more of a physical code review. Both have significant benefits when worked in conjunction with PROC COMPARE and we will look into both below.

PROGRAMMATIC REVIEW

We want to try and ensure that our derivations are rigorously tested early on when developing our ADaMs, when we have the most time available to us. One programmatic method that could be applied at early stages of our ADaM programming is to run both the production and the QC Code over a vast dataset of generated results (often referred to as UAT data) that is then saved into a reference area that both sides can access. Then independently, both derivations can be run through this UAT dataset and then both can be output for compare.

RANDOM GENERATOR

Using the same methodology as used in the example of *WITH* and *VAR* statements previously, we can generate a range of values to fall between our expected results. We need to ensure that when we have produced our random generator we save the outputted data set otherwise we will keep producing new results every time.

```
data Randgen;
    do x= 1 to 1000;
    a = round((rand("uniform")*100),1);      /*0-100 whole numbers*/
    b = round((rand("uniform")*100),1);      /*0-100 whole numbers*/
    c = round((rand("uniform")*100),0.1);    /*0-100 1 decimal place*/
    d = round((rand("uniform")*100),0.1);    /*0-100 1 decimal place*/
    e = round((rand("uniform")*10),1);       /*0-10 whole numbers*/
    f = round((rand("uniform")*10),1);       /*0-10 whole numbers*/
    g = round((rand("uniform")*10),0.1);     /*0-10 1 decimal place*/
    h = round((rand("uniform")*10),0.1);     /*0-10 1 decimal place*/
    i = round((rand("uniform")*4),0.1)+1;    /*1-5 1 decimal place*/
    j = round((rand("uniform")*4),0.1)+1;    /*1-5 1 decimal place*/
    output;
    end;
    drop x;
run;
```

An example of a potentially complex derivation that we could use is the CKD_EPI Equation for estimating Glomerular Filtration rate. The equation is :

$$GFR = 141 \times \min(S_{cr}/\kappa, 1)^{\alpha} \times \max(S_{cr}/\kappa, 1)^{-1.209} \times 0.993^{Age} \times 1.018\ [\text{if female}] \times 1.159\ [\text{if black}]$$

where:

S_{cr} *is serum creatinine in mg/dL,*

κ *is 0.7 for females and 0.9 for males,*

α *is -0.329 for females and -0.411 for males,*

min indicates the minimum of S_{cr}/κ *or 1, and*

max indicates the maximum of S_{cr}/κ *or 1.*

ADJUSTING THE METHOD

So with minor adjustments to our random number generated set above we can assure that we have all the correct ranges we are looking for in terms of input data to ensure both the production and QC codes are matching and producing reliable results.

First we have our randomly generated set of results, which will allow us to then apply this to our derivation. After having checked we are happy with our derivation, we can then use some of the PROC COMPARE techniques earlier to see.

```
data randgen_egfr;
  do unique_id = 1 to 1000;
    scr    =  round((rand("uniform")*4.75),0.01)+0.25;  /*Greater than average range of serum creatinine in mg/dL*/
    asexn  =  round(rand("uniform"),1)+1;               /*numeric Version of sex: Female=1, Male=2*/
    aracen =  round(rand("uniform"),1)+1;               /*numeric Version of race: African American=1, White/Other=2*/
    age    =  round((rand("uniform")*67),1)+18;         /*CKD_EPI requires a minimum age of 18 range to 85*/
    output;
  end;
run;
```

```
data egfr_deriv_prod ;
  set randgen_egfr;
  if      asexn=1 and aracen=1 then egfr=141*(min((scr/0.7),1)**-0.329)*max((scr/0.7),1)**-1.209*(0.993**age)*1.018*1.159;
  else if asexn=1 and aracen=2 then egfr=141*(min((scr/0.9),1)**-0.329)*max((scr/0.7),1)**-1.209*(0.993**age)*1.018;
  else if asexn=2 and aracen=1 then egfr=141*(min((scr/0.7),1)**-0.411)*max((scr/0.9),1)**-1.209*(0.993**age)*1.159;
  else if asexn=2 and aracen=2 then egfr=141*(min((scr/0.9),1)**-0.411)*max((scr/0.9),1)**-1.209*(0.993**age);

  if aracen=1 then arace='African American';
  else if aracen=2 then arace='White/Other';
  else put "WARN" "ING: code check randgen derivations, result falls outside of required parameters" aracen=;

  if asexn=1 then asex='Female';
  else if asexn=2 then asex='Male';
  else put "WARN" "ING: code check randgen derivations, result falls outside of required parameters" asexn=;

run;
```

```
data egfr_deriv_qc ;
  set randgen_egfr;
  if      asexn=1 and aracen=1 then egfr=141*(min((scr/0.7),1)**-0.329)*max((scr/0.7),1)**-1.209*(0.993**age)*1.018*1.159;
  else if asexn=1 and aracen=2 then egfr=141*(min((scr/0.7),1)**-0.329)*max((scr/0.7),1)**-1.209*(0.993**age)*1.018;
  else if asexn=2 and aracen=1 then egfr=141*(min((scr/0.9),1)**-0.411)*max((scr/0.9),1)**-1.209*(0.993**age)*1.159;
  else if asexn=2 and aracen=2 then egfr=141*(min((scr/0.9),1)**-0.411)*max((scr/0.9),1)**-1.209*(0.993**age);

  if aracen=1 then arace='African American';
  else if aracen=2 then arace='White/Other';
  else put "WARN" "ING: code check randgen derivations, result falls outside of required parameters" aracen=;

  if asexn=1 then asex='Female';
  else if asexn=2 then asex='Male';
  else put "WARN" "ING: code check randgen derivations, result falls outside of required parameters" asexn=;

run;
```

```
proc compare base = egfr_deriv_prod
             compare = egfr_deriv_qc listall;
             id unique_id arace asex;
run;
```

We ran our proc compare with the ID line added to ensure that there was an easy way to find which results were causing the biggest issues, if any. In this example, we have 54 cases of mismatching data between the QC and Prod derivations.

With the IDs added it is easy to spot the pattern.

```
Observation      Base   Compare   ID

First Obs           1         1  unique_id=1 racec=African American sexc=Male
First Unequal      15        15  unique_id=15 racec=White/Other sexc=Female
Last  Unequal     999       999  unique_id=999 racec=African American sexc=Male
Last  Obs        1000      1000  unique_id=1000 racec=White/Other sexc=Female

      Number of Observations in Common: 1000.
      Total Number of Observations Read from WORK.EGFR_DERIV_PROD: 1000.
      Total Number of Observations Read from WORK.EGFR_DERIV_QC: 1000.

      Number of Observations with Some Compared Variables Unequal: 54.
      Number of Observations with All Compared Variables Equal: 946.
```

unique_id	racec	sexc	Base GFR	Compare GFR	Diff.	% Diff
15	White/Other	Female	115.8799	106.6840	-9.1958	-7.9357
17	African American	Male	136.9045	151.8016	14.8970	10.8813
75	White/Other	Female	127.0994	117.0133	-10.0862	-7.9357
105	White/Other	Female	137.0778	126.1998	-10.8780	-7.9357
119	African American	Male	131.4405	137.2573	5.8168	4.4254
124	White/Other	Female	92.8517	85.4833	-7.3684	-7.9357
172	White/Other	Female	94.2839	93.2381	-1.0458	-1.1092
177	White/Other	Female	132.2985	121.7998	-10.4988	-7.9357
186	African American	Male	141.3099	156.6863	15.3764	10.8813

There is a difference in derivation for African American Men and for Women of any other race. Having looked into this and discussing with our prod programmer, we have worked out that they have used the wrong kappa value in their minimum part of the equation for these two instances.

This shows that with minimal data these issues could be missed early on in programming, and adding this sort of review may catch major issues before they become problems. However this does add additional steps and processes to what can be an already very labor-intensive task, so should really only be looked at being applied sparingly if time dictates so, and not on unnecessary calculations.

CODE REVIEW

Another way to ensure that the derivations are being done as per the specs is to apply an independent code review. This would involve assigning a separate programmer who hasn't worked on either the production or validation of the program to review the code of the production program and check that all derivations are being done as per the spec.

```
/*************************************************************************/

data adds;
 set adamdata.adds;
  where SCRNFL='Y' and parcat1 eq 'SCREEN' and paramcd in ("SFRES" "SFRESSP");
run;

proc sort;
  by siteid usubjid subjid dsstdtc;
run;

proc transpose data=adds out=adds_t (drop = _label_ _name_ );
 id paramcd;
 var avalc;
 by siteid usubjid subjid dsstdtc;
run;

data qc_dddata (keep = siteid cbpval1 usubjid subjid  dsstdtc st_col1);
length cbpval1  $108  st_col1 $200 ;
 set adds_t;
 cbpval1 = 'Site Id.: '||strip(siteid);
 st_col1 = strip(SFRES)||'/~'||strip(SFRESSP);
run;
```

Listing X

Listing of Reasons for [Screen Failure / Screen and Run-In Failure]

Site Id.: 111111

Unique Subject Id.	Subject Id.	Date of Screen/Run-In Failure	Type of Failure	Reason Term(s) for Screen//Run-In Failure/Specify Text
GSK123456.001055	000001	2012-04-07	Screen	DID NOT MEED INCLUSION/EXCLUSION CRITERIA
GSK123456.001067	001067	2012-04-05	Screen	PHYSICIAN DECISION: CHANGED MIND DID NOT MEED INCLUSION/EXCLUSION CRITERIA
GSK123456.002009	000002	2012-04-05	Run-In	ADVERSE EVENT
GSK123456.000365	000365	2012-04-05	Run-In	PROTOCOL DEVIATION: SUBJECT DID NOT STOP PROHIBITED MEDICATIONS
GSK123456.001069	001069	2012-04-05	Screen	DID NOT MEED INCLUSION/EXCLUSION
	001083	2012-04-05	Screen	WITHDRAWAL BY SUBJECT: CHANGED MIND

/Directory/program.sas 01JAN2004 12:01

Similar to programmatic review, a disadvantage of this is that it will require extra resource as now there are three independent study team members working on every program. So this will have to be something to be decided early on in the study timelines, rather than introducing it halfway through.

CONCLUSION

While PROC COMPARE is a strong time-saving procedure with many options that can be included in the code, it is always important not to forget the power of a human programmer's mind in alignment with the advantages of automated procedures. No matter how many datasets or outputs need to be done by a certain time or how much pressure there feels like there is, always take the time to think about the limitations of PROC COMPARE and ensure that the best possible job is being done in validating. There is ultimately nothing more important than the integrity of the results and this can easily be achieved by anyone. Take the time to assess how much time there is and how complex the derivations are. This is normally known as a risk-based approach – the higher the risk, the more safeguards are put in place, the lower the risk (not complex derivation, or not key endpoint), the less things need to be put in place. From this, it can be determined which of the approaches mentioned in this paper is best moving forward, whether it is none, one or both of the solutions discussed. Ultimately, every study is different and has different demands.

CONTACT INFORMATION

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

Eoin Byrne
Plus-Project Partnership
1200 Century Way Thorpe Park Business Park,
Ground and 1st Floor, Colton,
120,
Leeds,
LS15 8ZA

Email: eoin.byrne@plus-project.co.uk
LinkedIn: Eoin Byrne | LinkedIn
Web: https://www.plus-project.co.uk/

Peter Crumey
Plus-Project Partnership
Suite L3CA, No1, Booths Park,
Chelford Rd,
Knutsford,
Cheshire,
WA16 8GS

Email: peter.crumey@plus-project.co.uk
LinkedIn: https://www.linkedin.com/in/peter-crumey-431432117/
Web: https://www.plus-project.co.uk/