PhUSE 2016

Paper IS04

Playing Detective: Hints and Tips for Independent Programming QC

Bethan Thomas, PRA Health Sciences, Swansea, United Kingdom

ABSTRACT

Every programming team strives to produce high quality datasets and TFLs. One method of validating programming in pursuit of achieving this high quality is to perform independent double programming. The role of the validation programmer is critical to ensuring that the final outputs accurately represent the data. However, at first, this role can seem daunting and the task may seem overwhelming. This paper will provide some hints and tips for validation programmers. These tips will include programming techniques, SAS® shortcuts and use of study documents. This paper will provide practical advice and simple steps that can be followed to understand and perform the process with both ease and enjoyment, as one takes on the task of becoming a Data Detective! There will be plenty of examples provided to highlight the techniques and offer practical applications.

INTRODUCTION

Independent double programming involves a lot more than simply writing another program with a 'proc compare' at the end. A good QC (Quality Control) program will also account for many conceivable scenarios to catch any curiosities in the data that could slip through the net; it will include defensive checks to identify undetected problems with the source data, it will exemplify good programming practices so that other team members can easily continue the work in the future and it will be created with the specifications of both the source documentation (e.g. CRF) and output documentation (e.g. SAP, shells, SDTM/ADaM Implementation guides). Therefore, the paper will begin by covering the initial development of a QC program. In the Detective analogy, this could be considered the testimony of the witness's version of events.

Furthermore, once the writing is complete, the QC programmer must begin the process of identifying discrepancies and determining whether the primary or validation program is correct – examining the evidence! For junior programmers progressing to performing QC for the first time, this can be a big jump. A jumbled 'proc compare' output can seem impossible to get to the bottom of, with very few clues to tell you where to start looking. However, by applying a few simple techniques, the discrepancies can be dealt with one at a time. The first hurdle may simply be ensuring that both production and validation outputs contain the correct number of observations. A great way of determining where the differences lie is calculating frequency counts and then selecting subsets of observations or variables to examine. It is then much easier to identify the exact point where the main and QC datasets diverge from each other. Another technique is to begin by looking at the variables that an output is ordered by, as differences in sorting may make it impossible to tell where else the datasets vary.

Once variances have been identified, it is vital to refer back to the documentation to decide which programmer (if either) is correct. Therefore this paper will discuss key items to reference and where to find the answers to questions that may arise. This documentation could be compared to the laws in a detective's case. Similarly to the process of developing a program, during validation it is extremely helpful to consider continually both the data and the documentation and to ensure that the two are reconciled to each other. It is important not to simply assume that the QC is always correct and the primary program requires the change. It is also possible that the dataset specifications can have more than one potential interpretation and it may be that clarification or change is required.

As well as considering the process of QC, this paper will also dig into the detective's toolkit and go through some handy hints for becoming more efficient at validation. This includes utilizing the different options available in a 'proc compare', developing user-defined SAS® Abbreviations/keyboard macros for frequently used pieces of code and using SAS®-defined keyboard shortcuts to quickly manoeuvre your way around the data.

Note, throughout this paper the terms {main, production, primary} and {QC, validation} will be used interchangeably.

WRITING YOUR QC PROGRAM

The top tip for writing a QC program would be to make no assumptions. Just as a detective cannot assume that the man in the balaclava was the one who robbed the bank, the QC programmer cannot assume that the main dataset is already completely correct. The QC program is like a safety net; of course we would endeavour that the sites always enter data accurately and logically, data management pick up on every anomaly, the SAP is always interpreted correctly in the specifications and the production programmer creates a program that is robust and exactly per the specification and relevant standards. However, the reality is that even in a largely automated environment such as clinical trials, there is still room for human error at every stage, so it is best not assume that everything is already covered when beginning writing a validation program. Therefore, the QC program should be extra robust and seek to handle many scenarios, whilst also avoiding duplication of work. For example, if you have a copy of the edit checks, it would be unnecessary to re-program these.

It is also extremely helpful to have the main important documents open or nearby as you program. In the case of an ADaM dataset, you may be creating a derived row which specifications state should be the average of three readings. You notice that this row is not being created when there are only two readings or more than three readings. In this case it would be necessary to refer to the SAP and shells to check whether the desired output is the average of all readings (but in most cases this is three) or the average of exactly three readings. Another example might be that the specifications provide a character mapping for ADT; since you have the ADaM Implementation Guide open you can confirm that variables whose names end in DT should be numeric dates and thus a change to specifications is required.

MAINTAINING PROGRAM INDEPENDENCE

The premise of independent double programming is that if two programmers autonomously create an identical output, then it can be considered accurate and correct. Therefore, in order to maintain integrity, it is vital that programmers do not look at each other's programs or suggest specific code. Instead, the programmers can discuss the matters descriptively and decide how to proceed. This will ensure that the evidence you build up as a Data Detective will remain unbiased. For example, if you know the ordering differs between primary and validation outputs, then it is better to check your own program against the specifications and advise the other programmer to do likewise, than to dictate that VARIABLEX, VARIABLEY, VARIABLEZ should be used. Also, if possible, a '%include' should be used to run the other program and check the log rather than opening the file itself.

DETECTING DATA DISCREPANCIES

Possibly the largest proportion of validation work, will be spent locating data discrepancies. At first sight, a proc compare output can be pretty daunting. There are lots of different causes of mismatches, some are easier to solve and others require a little more investigation. This section will provide some hints of how to interpret some common causes of discrepancies and find the culprits of the problems.

DIFFERING ORDER VARIABLES

Sometimes the differences can be resolved simply by checking the sort variables. In the example below, there are equal numbers of observations in both datasets but the majority of rows and variables are mismatching with no clear/consistent pattern. Therefore the most obvious item to check first is that the correct order variables have been used

```
The COMPARE Procedure
Comparison of PHUSE.ADLB with PHUSE.ADLB_KEYVARS
(Method=EXACT)

Data Set Summary

Dataset                      Created          Modified  NVar    NObs

PHUSE.ADLB          23AUG16:08:10:05  23AUG16:08:10:05    51    3804
PHUSE.ADLB_KEYVARS  23AUG16:08:10:05  23AUG16:08:10:05    51    3804
```

Variables with Unequal Values

Variable	Type	Len	Label	Ndif	MaxDif	MissDif
LBSEQ	NUM	8	Sequence Number	3744	392	
TRTP	CHAR	13	Planned Treatment	2117		
TRTPN	NUM	8	Planned Treatment (N)	2117	1.000	
LBREFID	CHAR	20	Specimen ID	2523		
LBTESTCD	CHAR	8	Lab Test or Examination Short Name	3715		
LBTEST	CHAR	40	Lab Test or Examination Name	3715		
LBCAT	CHAR	24	Category for Lab Test	2243		
LBSCAT	CHAR	30	Subcategory for Lab Test	3381		
LBSTAT	CHAR	8	Completion Status	142		
VISITNUM	NUM	8	Visit Number	3627	43.000	
VISIT	CHAR	11	Visit Name	3627		
LBDTC	CHAR	19	Date/Time of Specimen Collection	3650		

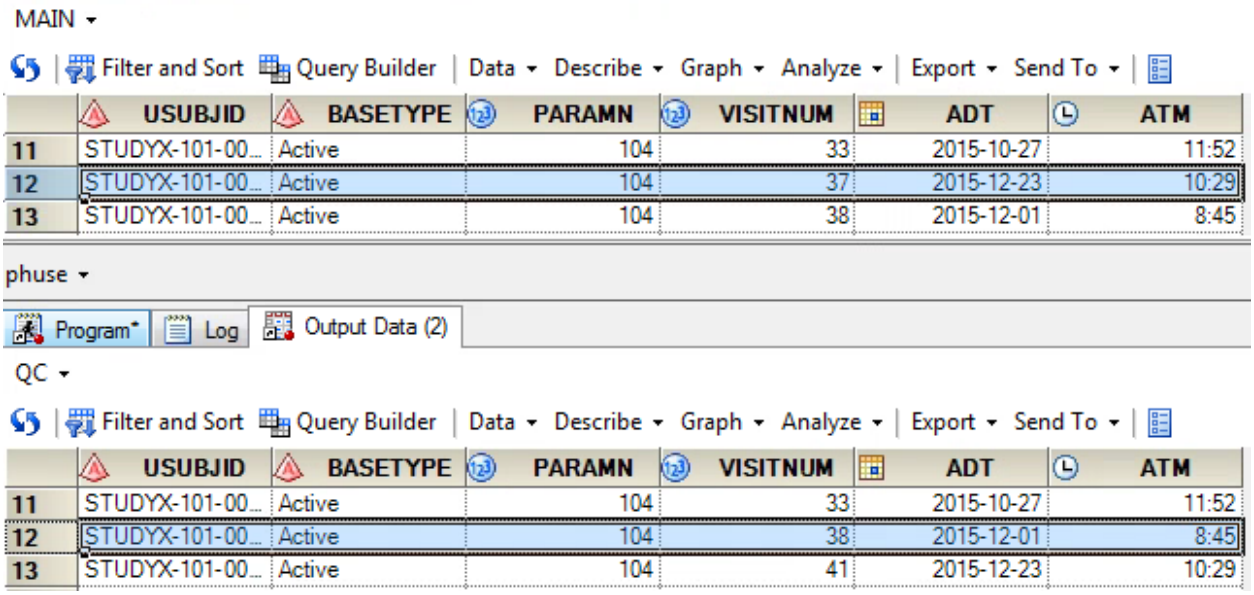
	Sequence Number	
Obs	Base LBSEQ	Compare LBSEQ
2	2.0000	1.0000
3	3.0000	2.0000
4	4.0000	3.0000
5	5.0000	4.0000
6	26.0000	5.0000
7	63.0000	2.0000
8	94.0000	3.0000
9	125.0000	4.0000
10	156.0000	5.0000
11	187.0000	26.0000
12	278.0000	63.0000
13	317.0000	94.0000
14	348.0000	125.0000

At other times there may be patterns or particular blocks of observations that repeatedly mismatch on multiple variables, whilst the rest of the data is equal. This may indicate that the sorting is very close, but perhaps the variables used are only similar, not the same, e.g. PARAM vs PARAMCD, the full text and codes are often in sync alphabetically but not always, perhaps all parameters from K onwards are differing, because when PARAMCD=K, PARAM=Potassium which comes much later alphabetically. Similarly, if both programs have a timing variable in the same place on the by statement, but one is using AVISITN, whereas the other is using ADT or VISITNUM, in the vast majority of cases, these should all be equal but where they differ, all other variables will be subsequently affected so you will see the same blocks of observations recurring in the compare output. You may also see patterns like below where different rows are flagged interchangeably, possibly hinting that the observations are equal but appearing in a different order. Perhaps unscheduled rows with AVISITN missing are appearing first if ordered by AVISITN, whereas when sorted by ADT they appear chronologically.

	Baseline Record Flag	
Obs	Base Value ABLFL	Compare Value ABLFL
126	Y	
131		Y
141	Y	
145		Y
155	Y	
159		Y
169	Y	
173		Y
183	Y	
188		Y
198	Y	
203		Y
213	Y	
218		Y

ORDER VARIABLES DIFFER

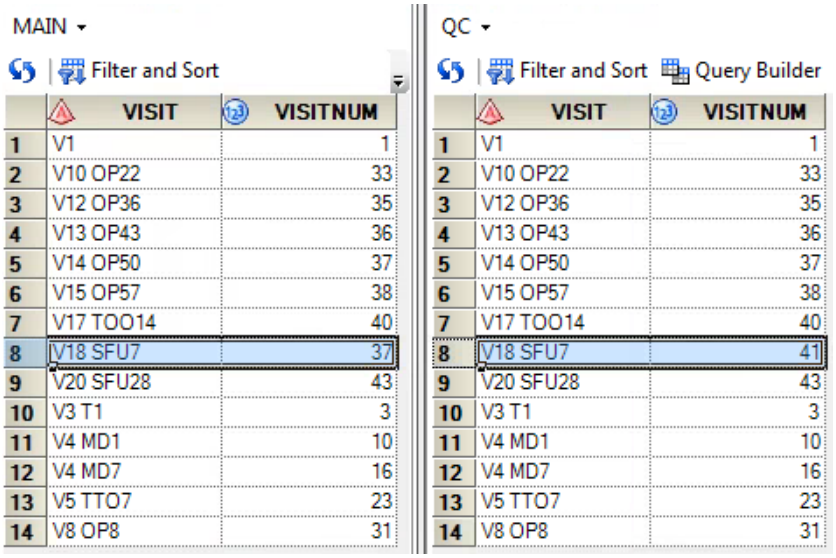
Sometimes it looks as though the datasets are sorted differently, but both programmers confirm that they are using the correct order variables. This may imply that the actual values of the order variables differ in main to QC. The output can look very similar to that described above with recurring patterns and blocks of mismatching observations. A technique that may be helpful in this case is to identify the first observation in a particular block and compare the values of the order variables only.



	USUBJID	BASETYPE	PARAMN	VISITNUM	ADT	ATM
11	STUDYX-101-00...	Active	104	33	2015-10-27	11:52
12	STUDYX-101-00...	Active	104	37	2015-12-23	10:29
13	STUDYX-101-00...	Active	104	38	2015-12-01	8:45

	USUBJID	BASETYPE	PARAMN	VISITNUM	ADT	ATM
11	STUDYX-101-00...	Active	104	33	2015-10-27	11:52
12	STUDYX-101-00...	Active	104	38	2015-12-01	8:45
13	STUDYX-101-00...	Active	104	41	2015-12-23	10:29

If the by variables are USUBJID BASETYPE PARAMN VISITNUM ADT ATM, then the first mismatch is VISITNUM, it equals 37 in Main, but in QC it is equal to 38. The values of ADT and ATM where VISITNUM=38 seem to be consistent in Main and QC, but their values where VISITNUM=37 in Main seem to match VISITNUM=41 in QC. Something strange seems to be happening here, therefore the next step would be to look at the VISITNUM mapping a little closer. One way of doing this is to compare all the distinct values. Since VISITNUM is a coded value, it is easier to look at the distinct values alongside VISIT, as follows:



	VISIT	VISITNUM
1	V1	1
2	V10 OP22	33
3	V12 OP36	35
4	V13 OP43	36
5	V14 OP50	37
6	V15 OP57	38
7	V17 TOO14	40
8	V18 SFU7	37
9	V20 SFU28	43
10	V3 T1	3
11	V4 MD1	10
12	V4 MD7	16
13	V5 TTO7	23
14	V8 OP8	31

	VISIT	VISITNUM
1	V1	1
2	V10 OP22	33
3	V12 OP36	35
4	V13 OP43	36
5	V14 OP50	37
6	V15 OP57	38
7	V17 TOO14	40
8	V18 SFU7	41
9	V20 SFU28	43
10	V3 T1	3
11	V4 MD1	10
12	V4 MD7	16
13	V5 TTO7	23
14	V8 OP8	31

We can now see that the values of VISITNUM differ when VISIT='V18 SFU7' so both programmers should refer back to the specification to check this mapping. Although, Data Detective's intuition might suggest that the numbering in QC is more logical and is a one-to-one mapping as required.

SOURCE DATA AND DOCUMENTATION

Our primary goal as programmers, and therefore as program validators, is not simply to get clean logs and tidy compare outputs. If production and validation outputs end up identically wrong then there is no added value in double programming. Therefore, it is extremely important, especially as a validation programmer, to be familiar with the source data structure and regularly refer to its associated documentation to ensure it is accurately reflected.

If we take a simple AVISIT mapping as an example in ADPE, the specification is simply, 'Populate for scheduled assessments.' You notice whilst comparing that there is a record which is identical in the primary and validation datasets except that QC has populated AVISIT with 'Visit 3', whereas the primary dataset has AVISIT set to null. Rather than just assuming the one with AVISIT populated must be correct, the validator's next step should be to reference the schedule of assessments found in the Protocol and/or SAP. The schedule of assessments may indicate that whilst Visit 3 is scheduled, it is not planned to perform a Physical Examination at this visit. It transpires that the validation programmer populated AVISIT in all cases unless the value of VISITNUM indicated an unscheduled visit (e.g. VISITNUM=4.01), whilst the primary programmer only populated this where a Physical Exam was specifically scheduled. At this point, it may be worth referencing the SAP to see if it provides more detail on how it classifies unscheduled visits and how they should be handled for analysis. This case illustrates the importance of referencing both the Protocol and the SAP when programming ADaM domains.

Here is another example from SDTM programming. The snapshots below come from Main and QC datasets for a Biospecimen Events (BE) domain.

MAIN

Filter and Sort | Query Builder | Data | Describe | Graph | Analyze | Export | Send To

	SUBJECT	BESEQ	BEREFID	BETERM	BECAT	BESCAT	BEPRESP	BEOCCUR	BESTDTC
1	25110	1	C1844019420	COLLECTED	COLLECTION	GENE EXPRESSION	Y	Y	2016-01-08
2	25110	2	C1844019660	COLLECTED	COLLECTION	GENE EXPRESSION	Y	Y	2016-02-05
3	25110	3	C1844019780	COLLECTED	COLLECTION	GENE EXPRESSION	Y	Y	2016-04-01
4	25110	4		COLLECTED	COLLECTION	GENETIC SAMPLES	Y	N	
5	25110	5	C1844019420	COLLECTED	COLLECTION	GENETIC SAMPLES	Y	Y	2016-01-08
6	25110	6	C1844019780	COLLECTED	COLLECTION	GENETIC SAMPLES	Y	Y	2016-04-01

Program

Program* | Log | Output Data (2)

QC

Filter and Sort | Query Builder | Data | Describe | Graph | Analyze | Export | Send To

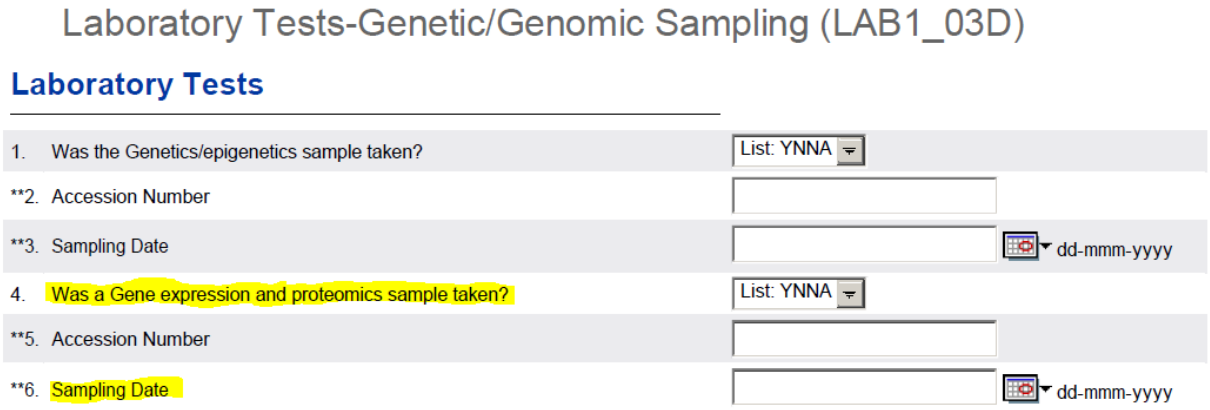
	SUBJECT	BESEQ	BEREFID	BETERM	BECAT	BESCAT	BEPRESP	BEOCCUR	BESTDTC
1	25110	1	C1844019660	COLLECTED	COLLECTION	GENE EXPRESSION	Y	Y	2016-02-05
2	25110	2	C1844019780	COLLECTED	COLLECTION	GENE EXPRESSION	Y	Y	2016-04-01
3	25110	3		COLLECTED	COLLECTION	GENETIC SAMPLES	Y	N	
4	25110	4	C1844019420	COLLECTED	COLLECTION	GENETIC SAMPLES	Y	Y	2016-01-08
5	25110	5	C1844019780	COLLECTED	COLLECTION	GENETIC SAMPLES	Y	Y	2016-04-01

You will notice that a record for Gene Expression on 8th January is in the main dataset but is not present in QC. The validator should use this as a trigger to check whether there should be a record or not for Gene Expression on that date for this subject. The first place to check would be the source data, here is a snapshot:

	Event Label	Form Name	Was the Genetics/epigenetics sample taken?	Accession Number	Sampling Date	Was a Gene expression and proteomics sample taken?	Accession Number	Sampling Date	If Yes: Date of Informed Consent
1	Baseline Visit 1 (Week 0)	LAB1_03E	Yes	C1844019420	08-Jan-2016		C1844019420		18-Nov-2015
2	Visit 4 (Week 4)	LAB1_03D	No			Yes	C1844019660	05-Feb-2016	
3	Visit 8 (Week 12)	LAB1_03D	Yes	C1844019780	01-Apr-2016	Yes	C1844019780	01-Apr-2016	

You can see that the genetic sample taken question and date are populated in the first record but the gene expression sample taken question and date are missing, although there is a corresponding accession number for gene expression. At this point it is difficult to tell whether a record should have been created or not, or whether perhaps a data query should be raised, therefore it may worth checking the CRF for any clues on how the data is to be entered and interpreted. The shrewder Data Detective may note that the form name for the Baseline record is different from the two subsequent records, therefore both forms should be located and inspected to see if they provides us with any further clues.

Form 1 (LAB1_03D)

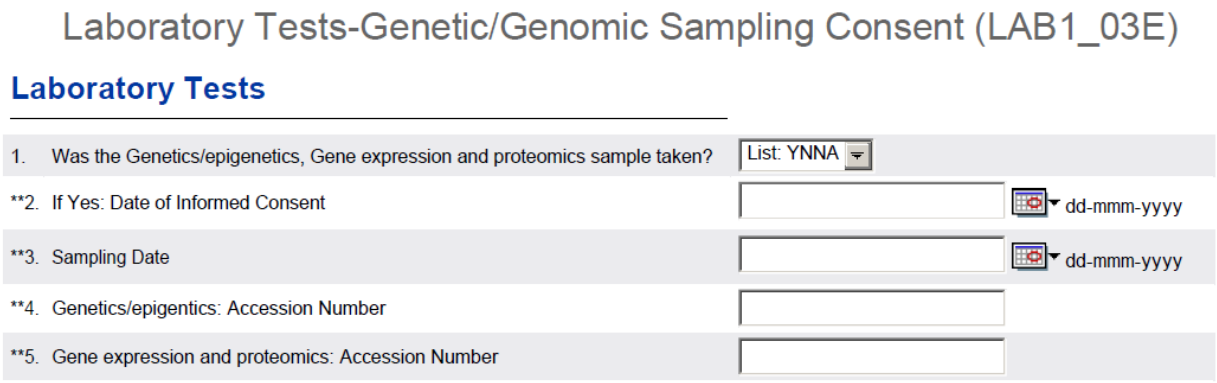


Laboratory Tests-Genetic/Genomic Sampling (LAB1_03D)

Laboratory Tests

1.	Was the Genetics/epigenetics sample taken?	List: YNNA
**2.	Accession Number	
**3.	Sampling Date	dd-mmm-yyyy
4.	Was a Gene expression and proteomics sample taken?	List: YNNA
**5.	Accession Number	
**6.	Sampling Date	dd-mmm-yyyy

Form 2 (LAB1_03E)



Laboratory Tests-Genetic/Genomic Sampling Consent (LAB1_03E)

Laboratory Tests

1.	Was the Genetics/epigenetics, Gene expression and proteomics sample taken?	List: YNNA
**2.	If Yes: Date of Informed Consent	dd-mmm-yyyy
**3.	Sampling Date	dd-mmm-yyyy
**4.	Genetics/epigentics: Accession Number	
**5.	Gene expression and proteomics: Accession Number	

The confusion now becomes clearer. Two very similar forms are used, but whereas in the first form, the gene expression collection has a separate prompt and date collected, on the second form, the same prompt and date are used for both genetic and gene expression samples. If we go back and look at our source data, we see that the relevant record comes from the latter form. Thus we can infer that the gene expression sample was collected since the accession number for it is populated, and the prompt and date for the genetic sample are present. The production SDTM dataset is correct, the QC programmer should go back and check their code to ensure all relevant data is captured in the BE domain.

DISCREPANCIES IN OBSERVATION COUNTS

Often the first hurdle when investigating dataset discrepancies is to determine the correct number of observations, as in the previous example. When first produced, production and validation datasets frequently contain differing numbers of records. In such cases, a simple proc compare output can look extremely confusing and provide very few clues about where to start checking. There are many ways one could begin; subsetting the data, using ID variables in the proc compare, subsetting the variables to compare, looking at the first differing observation etc. so it can be difficult to know where to start.

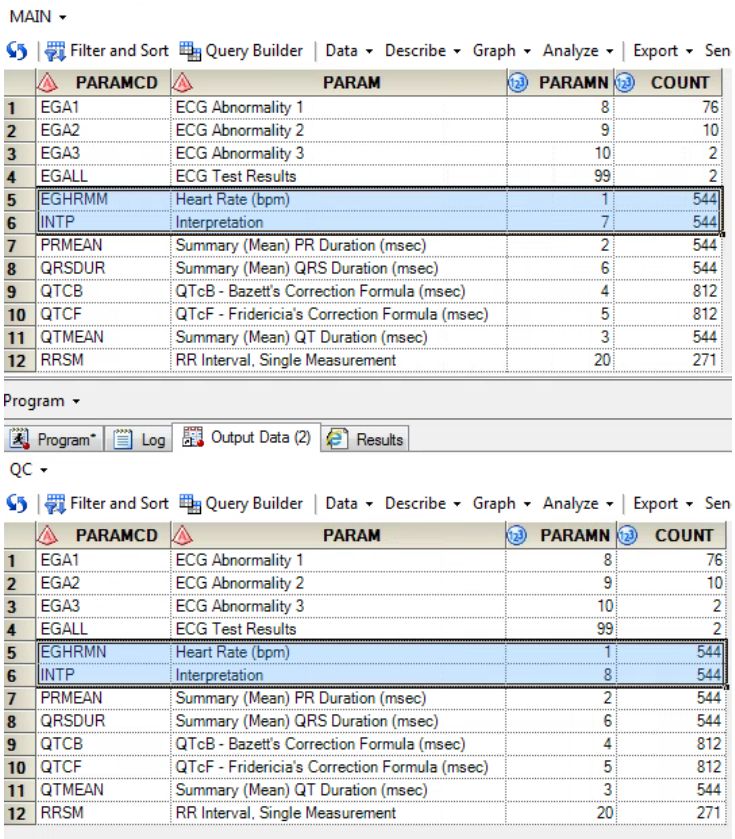
One simple, yet highly effective tool is the FREQ procedure. By taking the frequency counts for each test, visit, subject, category or other variable of interest, and then comparing main and QC using the same variable(s) in the ID statement, you can narrow down your search for the root of a problem very quickly. For example, the vital signs datasets have differing numbers of records. Firstly, you compare the frequencies by subject and you see that the QC dataset always has more records per subject, next you compare the frequencies by visit and find the same outcome, finally you compare frequencies by test and note that the main dataset does not contain any weight, height or BMI results. You check the source data and CRF and note that this information is recorded on a different form and consequently may have accidentally been omitted from the production dataset.

Sometimes the proc freq tool may not immediately provide you with your solution, but it can be a helpful pointer along the way. This might be the case if the frequency counts by subject do not match in some cases, but do in others. Since USUBJID is not particular to any domain, it is not immediately evident what the mismatching subjects have in common. It may be helpful to begin by subsetting each dataset to look at just one mismatching subject. If this gives you quite a small number of observations then you may be able to visually compare the primary and QC datasets, or you may want to take frequencies again by test or visit until you have a manageable number of observations to examine or until you have found the root cause of the issue.

If using this technique then it is worth noting that it works best with identifier or grouping qualifier variables such as subject, parameter, category etc. By definition, these variables naturally separate themselves into distinct groups with unifying characteristics. The technique is less effective if used with result qualifiers, certain timing variables such as date or relative day, or free text fields such as verbatim terms or comments. By definition, these variables can often have unlimited potential outcomes and observations where these are equal may not necessarily have any characteristics in common. In an ADaM context, ---FL variables can also vary in usefulness with this tool; it may be useful to check frequencies by SAFFL, but counting by ABLFL or ANL01FL may not be a quick route to the cause of discrepancies. If you have complex derivations in ADaM involving creating records, multiple baselines or derived parameters then the variables DTYPE, BASETYPE and PARAMTYP can be useful as well alongside grouping variables such as PARAM, AVISIT and USUBJID as it may be necessary to determine the cases where an item should or should not be created if the production and validation programs have derived different numbers of records to each other.

MAPPING OF CODED VARIABLES

If the primary and validation programs are mapping decodes differently, this can result in a messy looking proc compare output, especially if these variables are used for sorting. Once again, the proc freq can be utilized in order to easily identify where the discrepancies are. For example, there are lots of differences when you compare main and QC for ADEG. It may be useful to compare frequencies by including PARAMCD*PARAM*PARAMN in the table statement. This will alert you if one program is numbering the parameters differently or has a spelling mistake as below. In this case, the actual frequency counts may not be the point of interest but it is a quick way of identifying all the distinct values and combinations.



	PARAMCD	PARAM	PARAMN	COUNT
1	EGA1	ECG Abnormality 1	8	76
2	EGA2	ECG Abnormality 2	9	10
3	EGA3	ECG Abnormality 3	10	2
4	EGALL	ECG Test Results	99	2
5	EGHRMM	Heart Rate (bpm)	1	544
6	INTP	Interpretation	7	544
7	PRMEAN	Summary (Mean) PR Duration (msec)	2	544
8	QRSDUR	Summary (Mean) QRS Duration (msec)	6	544
9	QTCB	QTcB - Bazett's Correction Formula (msec)	4	812
10	QTCF	QTcF - Fridericia's Correction Formula (msec)	5	812
11	QTMEAN	Summary (Mean) QT Duration (msec)	3	544
12	RRSM	RR Interval, Single Measurement	20	271

	PARAMCD	PARAM	PARAMN	COUNT
1	EGA1	ECG Abnormality 1	8	76
2	EGA2	ECG Abnormality 2	9	10
3	EGA3	ECG Abnormality 3	10	2
4	EGALL	ECG Test Results	99	2
5	EGHRMN	Heart Rate (bpm)	1	544
6	INTP	Interpretation	8	544
7	PRMEAN	Summary (Mean) PR Duration (msec)	2	544
8	QRSDUR	Summary (Mean) QRS Duration (msec)	6	544
9	QTCB	QTcB - Bazett's Correction Formula (msec)	4	812
10	QTCF	QTcF - Fridericia's Correction Formula (msec)	5	812
11	QTMEAN	Summary (Mean) QT Duration (msec)	3	544
12	RRSM	RR Interval, Single Measurement	20	271

A QC PROGRAM AND A PROGRAM TO QC

Writing portions of SAS® code as described can be incredibly helpful in quickly validating datasets. This can enable you to subset data by observations or variables, calculate frequency counts etc. However, it is advisable to keep this code in a separate program to avoid accidentally making permanent changes to the program. I.e. if you have subset the data which is compared and then you run this PROC COMPARE whilst preparing a deliverable, you will not have validated the entire dataset but rather only a subset. Similarly, in the separate program you using to QC, it is important to ensure you create new datasets when you subset or calculate frequencies, rather than overwriting the current production and QC datasets.

Keeping these programs and datasets separate gives you the freedom to try a variety of techniques and even modify data while you narrow down issues and work to get validated datasets. For instance, the variable USUBJID is often longer than the number of characters displayed in the output for ID variables. Since you may not need to see the STUDYID part of the values for this purpose, you could create a shorter version to help you pin point subjects to investigate. Another example is below:

Obs	Gender/Age(years)/Race/Weight(kg) Base Value BLGARW	Compare Value BLGARW
1	M/27/WHIT/59.4	M/27/WHITE/59.4
3	M/27/WHIT/79.7	M/27/WHITE/79.7
4	F/36/WHIT/79.1	F/36/WHITE/79.2
5	M/42/WHIT/128.3	M/42/WHITE/128.3
6	F/22/WHIT/52	F/22/WHITE/52
7	F/45/WHIT/	F/45/WHITE/
8	F/40/WHIT/49	F/40/WHITE/49
9	M/25/WHIT/	M/24/WHITE
10	M/39/WHIT/	M/39/WHITE/
11	M/39/WHIT/60	M/39/WHITE/60

It is clear that the race part of this concatenated variable is mismatching for all white subjects. However, the highlighted differences may be less immediately clear. In order to ensure that the rest of the text is matching, the primary value could be temporarily modified in a work dataset and then re-compared to check the remaining part of the variable.

It is important to note that whilst it is helpful to create temporary programs and datasets in order to identify issues, it is the permanent programs and datasets that must be validated. Therefore, always ensure you have thoroughly checked the final permanent outputs before signing them off as QC complete.

THE LISTALL OPTION AND A WARNING REGARDING ID VARIABLES

The LISTALL option in the compare procedure is also useful because as well as comparing observations present in both datasets, it will also list observations or variables present in one dataset but not the other. Coupled with ID variables, this is very helpful. For instance if PARAMCD counts were being compared, the output might state that PARAMCD='SYSBP' is only found in the Main dataset, If the LISTALL option is used without ID variables, the output would simply state that the last observation is found in the Main dataset only, and not point to the specific parameter.

It is important to note that if one of the compared datasets has zero observations and an ID statement is used without the LISTALL option, then at first view the output can appear like a complete match, as follows.

```
The COMPARE Procedure
Comparison of WORK.MAIN with WORK.QC
(Method=EXACT)

Data Set Summary

Dataset             Created          Modified  NVar    NObs

WORK.MAIN  02SEP16:03:26:40  02SEP16:03:26:40    66    1189
WORK.QC    02SEP16:03:26:40  02SEP16:03:26:40    66    1190


Variables Summary

Number of Variables in Common: 66.
Number of ID Variables: 4.
```

```
Observation Summary

Observation      Base  Compare  ID

First Obs           1        1  STUDYID=EP0069 USUBJID=EP0069-101-00701 PARAMN=1 AVISITN=101
Last  Obs        1189     1190  STUDYID=EP0069 USUBJID=EP0069-602-00767 PARAMN=5 AVISITN=105

Number of Observations in Common: 1189.
Number of Observations in WORK.QC but not in WORK.MAIN: 1.
Total Number of Observations Read from WORK.MAIN: 1189.
Total Number of Observations Read from WORK.QC: 1190.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 1189.

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

In the screenshot above, the datasets might appear to match, since the output states 'No unequal values were found. All values compared are exactly equal.' However, under closer inspection, the observation counts are closely unequal. This could easily happen, particularly if this is one of a number of compare outputs in a batch run. However, if either LISTALL is selected or the ID statement is omitted, it becomes a lot more obvious that there are issues to look into.

SAS® SHORTCUT KEYS

The kind of code used in the program described above may get replicated and adjusted every time you perform QC. One way of easily reproducing the same skeleton of code is to create a SAS® keyboard macro or abbreviation. Depending on the version of SAS® used, the method of doing this may vary slightly; in SAS Enterprise guide the functionality can be found by going to the Program menu and into Editor Macros, in older versions of SAS® this can be found in the Tools menu and into Keyboard macros. There are multiple papers available with details on how to do this but the key is that this enables you to quickly generate frequently used strings of code simply by typing a word prompt or shortcut key. Here is an example of a piece of code which can aid the kind of QC techniques listed in this paper:

```
data main;
      set adam.;
      subject= scan(usubjid,2,'-')|| '-' ||scan(usubjid,3,'-');
*     if;
*     where;
*     keep;
*     drop;
```

```
run;

/*proc freq data=main noprint;*/
/*     table subject / out=main (drop=percent);*/
/*run;*/


data qc;
       set qadam.;
       subject= scan(usubjid,2,'-')|| '-' ||scan(usubjid,3,'-');
*      if;
*      where;
*      keep;
*      drop;
run;

/*proc freq data=qc noprint;*/
/*     table subject  / out=qc (drop=percent);*/
/*run;*/

proc compare base=main compare=qc listall;
/*     id subject ;*/
run;
```

The skeleton of this text comprises many of the tools described in this paper: reading in primary and validation datasets, subsetting observations or variables, creating helpful variables, taking frequency counts, use of ID variables and LISTALL option in proc compare, whilst remaining flexible enough that different parts can be easily commented in or out with little modification to the code. This code can be instantly generated if a keyboard macro is associated with it, thus saving the validator time and allowing more time to actually perform validation.

OTHER ASPECTS OF VALIDATION

Automated compares and log checks are obviously incredibly helpful in validating outputs, however it is important not to rely on them exclusively. There is great value in visually checking data to spot obvious anomalies and also ensure that on the rare occasion when main and QC programmers make identical mistakes, they are identified. Though automated processes have a great capacity to quickly and accurately check large amounts of data, they cannot replicate the holistic review of a programmer who understands the source, desired format, purpose, context and associated documents the way a human programmer can. Important items to watch out for include implausible values and dates, unexpected missing data and truncation of values.

Standards compliance validators such as Pinnacle 21 are also important both for picking up on more intrinsic matters and for educating the QC programmer in clues to look for in the data that might allude to a compliance issue.

There are many things for a Data Detective to remember and validation can be a long process, stretched over a period of time. It can be useful to create or reference some kind of ‘QC Checklist’ which covers all the tasks and items to be checked for each type of output (SDTM, ADaM, TFL etc.) to ensure thoroughness and consistency. This may a checklist which is relevant to all studies, or perhaps there might be study-specific points that could be added if necessary.

ACKNOWLEDGMENTS

Thanks to Sally Morgan and Helen Besley for interesting discussions and training on QC as a process and also for reviewing the abstract and paper.

Thanks also to Marion Thomas for proof-reading the paper and sharing ideas.

CONCLUSION

Independent double programming is an excellent method of ensuring high quality outputs are produced, which map data correctly per industry standards and study documentation. This process may seem like a puzzle or a mystery to be solved and therefore may be difficult at first, but with programming techniques as our magnifying glass and the analytical logic of great detectives, this can be a satisfying and effective process for every QC programmer. SAS®

has a great deal of functionality to support both the creation of an independent validation program but also for detecting and pinpointing the cause of discrepancies between main and QC datasets. The FREQ procedure is very powerful and highly versatile in reaching these conclusions alongside the COMPARE procedure. However, these tools are most effectively used when coupled with rigorous logic, analysis and understanding of the associated source, output and documentation. Referencing the CRF, SAP, protocol, specifications and implementation guides continually will ensure the final outputs are produced more quickly and to a higher quality. Increasing our efficiency and quality of programming will ultimately support the goal of increasing the efficiency and quality of clinical research.

CONTACT INFORMATION

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

Bethan Thomas
PRA Health Sciences
Llys Tawe
Kings Road
Swansea
SA1 8NE
Work Phone: +44 (0)1792 525710
Fax: +44 (0)1792525601
Email: thomasbethan@prahs.com