SESUG 2021 Paper 054

Identifying Data Inconsistencies with SAS®

Imelda C. Go, Cognia, Inc.

ABSTRACT

This paper goes over a quality control example on how to identify data inconsistencies when the expectation is data consistency. PROC MEANS is used to diagnose data inconsistency and to generate an output data set. This data set is then presented to the reader using PROC TABULATE, which helps present the inconsistencies in a way different from viewing a data set. For example, a test item appears on different test forms. The metadata for the same item are expected to be identical across test forms where the item appears. How can you easily confirm that the item metadata are identical across all test forms and how can you identify inconsistencies in a report that facilitates the identification and correction of the inconsistencies? With macro techniques, we can further automate this process to include output via ODS Excel for archiving and sharing our findings.

INTRODUCTION

The method uses PROC MEANS predominantly to identify the inconsistencies. An example from the assessment industry will be used to illustrate the technique. Let us assume the following:

- We have 3 mathematics test forms (A-C) with 3 items each (2 operational (OP) test items and 1 field (FT) test).
- The 1st and 3rd items are both OP items, while the 2nd item is an FT item.
- The expectation is that the same item (based on ItemID), wherever it appears among the forms, has the same AnswerKey, ItemFunction, ItemType, and Position values.

Subject	FormID	ItemID	AnswerKey	ItemFunction	ItemType	Position
math	A	11	A	OP	MC	1
math	A	33	Yellow	FT	CR	2
math	A	22	C	OP	MC	3
math	B	11	A	OP	CR	1
math	B	44	Red	FT	CR	2
math	B	22	C	OP	MC	3
math	C	11	A	OP	CR	1
math	C	55	Blue	FT	CR	3
math	C	22	B	OP	MC	2

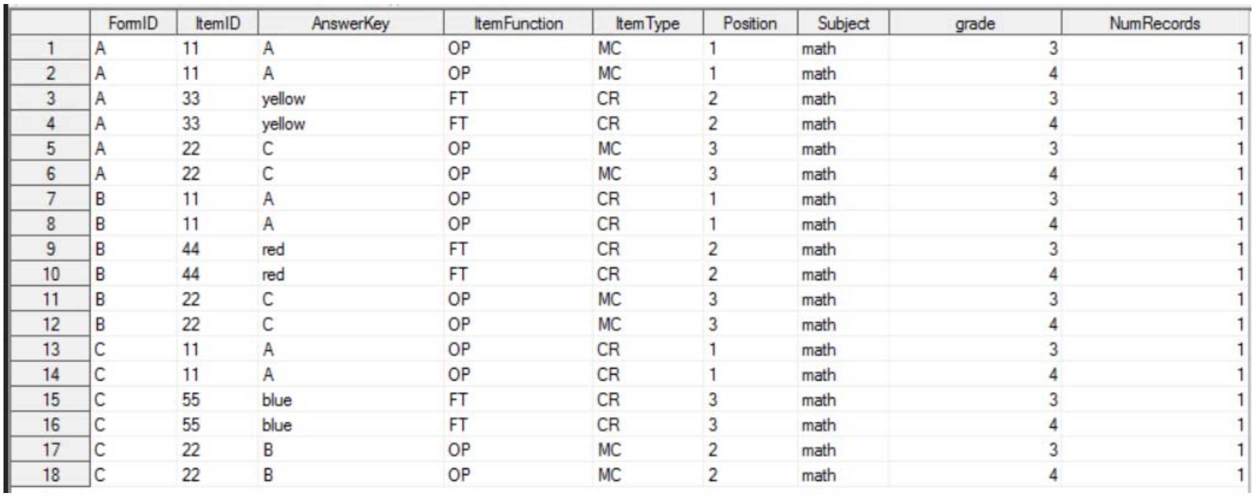
Given the above data, we see the following issues:

- For ItemID=22, the AnswerKey is C on two forms and is B on form C.
- For ItemID=11, the ItemType value is inconsistent. It is MC on one form and CR on the other two.
- ItemID=11 and ItemID=22 appear as the 1st and 3rd items on forms A and B but not on form C.

For such a short test, we can examine the data visually with ease. We want a solution, which tells us exactly where the inconsistencies are quickly and with less effort.

For the sake of providing an example, let us also expand the scenario to using these items for a 3rd and 4th grade test. (The 3rd and 4th grade test have the exact same items.) Also assume that there are no missing values for each variable. The sample code was prepared with the intent to illustrate the technique. In actual practice, certain parts of it can be rewritten more succinctly (e.g., PROC SQL coding) or be adjusted (e.g., missing values are present).

We will process the 3rd and 4th grade test data set using PROC MEANS. Without a VAR statement, PROC MEANS will attempt to use all numeric variables on the data set. To make things simpler, add a numeric variable called `NumRecords` with a value of 1 to the data set.



	FormID	ItemID	AnswerKey	ItemFunction	ItemType	Position	Subject	grade	NumRecords
1	A	11	A	OP	MC	1	math	3	1
2	A	11	A	OP	MC	1	math	4	1
3	A	33	yellow	FT	CR	2	math	3	1
4	A	33	yellow	FT	CR	2	math	4	1
5	A	22	C	OP	MC	3	math	3	1
6	A	22	C	OP	MC	3	math	4	1
7	B	11	A	OP	CR	1	math	3	1
8	B	11	A	OP	CR	1	math	4	1
9	B	44	red	FT	CR	2	math	3	1
10	B	44	red	FT	CR	2	math	4	1
11	B	22	C	OP	MC	3	math	3	1
12	B	22	C	OP	MC	3	math	4	1
13	C	11	A	OP	CR	1	math	3	1
14	C	11	A	OP	CR	1	math	4	1
15	C	55	blue	FT	CR	3	math	3	1
16	C	55	blue	FT	CR	3	math	4	1
17	C	22	B	OP	MC	2	math	3	1
18	C	22	B	OP	MC	2	math	4	1

We will use a combination of PROC MEANS, PROC TABULATE, and ODS Excel to create output, regarding data inconsistencies, and send the output to an Excel file with multiple worksheets.

The only input required from the user is shown below in the six %LET statements. Beyond that, the code is data-driven/generalized and will function based on the values in the %LET statements.

`%let InputData=Sample;`
`%let DataVars=AnswerKey ItemFunction ItemType Position;`
`%let IDVar=ItemID;`
`%let FormVar=FormID;`
`%let GroupVars=Subject &IDVar.;`
`%let OutPath=C:\Users\imelda.go\Desktop;`
	`&InputData` specifies the data set.
	`&DataVars` is a list of variables that need to be checked for consistency.
	`&IDVar` is the item ID.
	`&FormVar` is the test form ID.
	`&GroupVars` lists the groups of items where you want to check inconsistencies for. It is at minimum the `&IDvar`.
	By specifying `Subject` prior to `&IDVar`, you will be checking for inconsistencies within each value of the `subject` variable. Leave `&IDVar` in the last position in the list since `&GroupVars` will be used in a CLASS statement where the order of variables listed affects the _type_ variable generated by PROC MEANS.

`%let GroupVarsAsterisk = %sysfunc(translate(&GroupVars,'*',' '));`
`%let GroupVarsComma = %sysfunc(translate(&GroupVars,',',' '));`
`%let NDataVars=%sysfunc(countw(&DataVars));`
`%let NGroupVars=%sysfunc(countw(&GroupVars));`
	`&GroupVarsAsterisk` is the `&GroupVars` value with an asterisk (`*`) as the delimiter, which will be used in TYPES statement.
	`&GroupVarsComma` is the `&GroupVars` value with a comma (,) as the delimiter, which will be used in PROC SQL.
	`&NDataVars` is the number of variables listed in `&DataVars`, which will be used for automation (DO loop statement).
	`&NGroupVars` is the number of variables in listed in `&GroupVars`, which will be used to calculate the value of _type_ that we need to identify records of interest.

```
data TypeValue;
retain type 0;
if &NGroupVars>0 then
do i=1 to &NGroupVars;
type=type+ 2**(i-1);
end;
output;
call symputx('type',type);
```

Calculate the value of _type_ that corresponds to the record with the total number of times a combination of values appear for variables listed in `&GroupVars`.

In this example `&GroupVar=subject ItemID` so we are counting the number of records per `subject` and `itemID` combination. In this case the value of interest is _type_=3. It is 3 because we have 2 variables in `&GroupVars`. Since these are the last two variables listed in the CLASS statement, the corresponding _type_ value will be $2^1+2^0=2+1=3$.

```
proc means data=&InputData noprint n
chartype;
id &FormVar;
class &DataVars &GroupVars/missing;
types &GroupVarsasterisk.
&GroupVarsasterisk.*(&DataVars);
var NumRecords;
output out=CountCombos (where=(
_stat_="N" ));
```

PROC MEANS allows us to count various things. We can limit what we count by using the TYPES statement.

The ID statement adds the `&FormVar` variable to the output data set.

The order in which the variables are listed in the CLASS statement affect the values of the `_type_` variable. Due to the need for predictability, put `&ItemGroupVars` at the very end of the list of variables in the CLASS statement. You need to know the right `_type_` value to use in the next step.

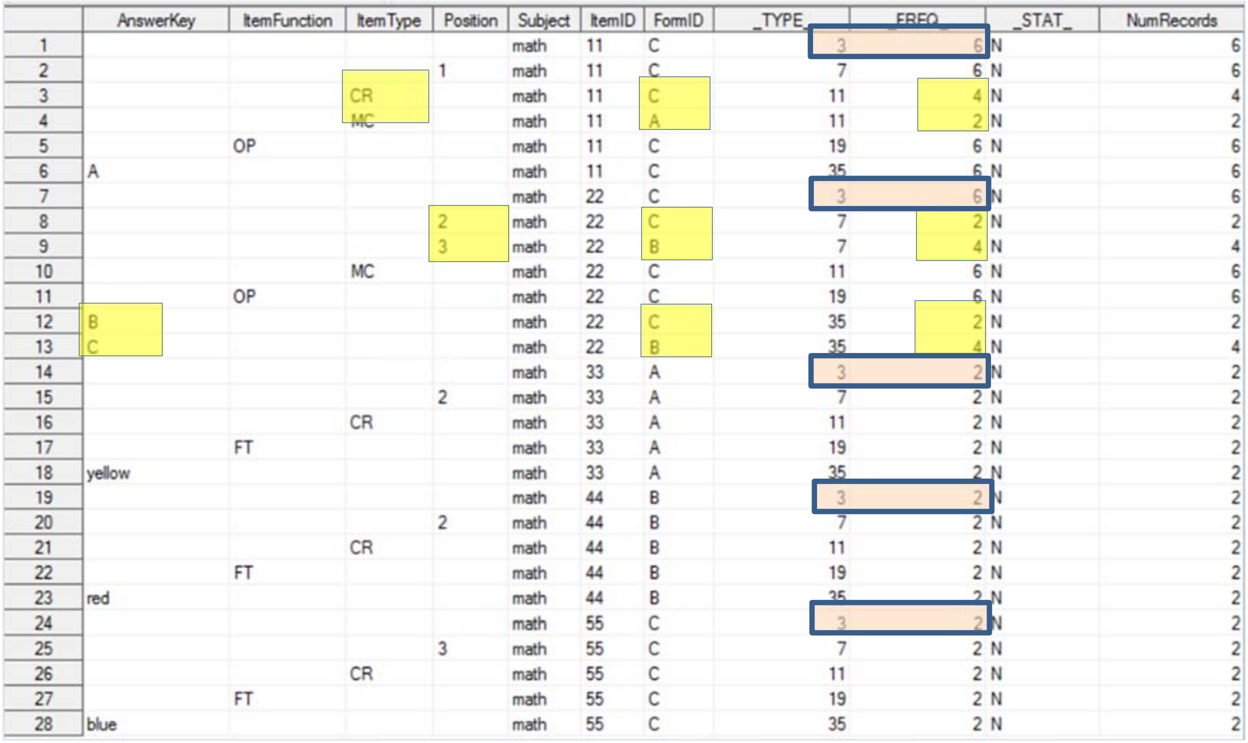
The data were sorted for instructional purposes to illustrate below the rationale for the technique.

The sorted data set is shown below.

- Whenever `_type_=3`, `NumRecords` is equal to the total number of records that appear for each unique combination of `subject` and `ItemID`. In general, the data set tells us the number of unique combinations listed in the CLASS statement but specifically the combinations of interest specified in the TYPES statement. The _type_ value tells us exactly which combination of variables is involved for each row in the data set. The specific values for each combination of interest are in the data set.

- The variables in the CLASS statement are `AnswerKey, ItemFunction, ItemType, Position, Subject, ItemID`. The sample data are such that `NumRecords` is always 6 if the data are all consistent. Wherever we see `NumRecords` not equal to 6, we see inconsistencies in the values as shown. Where there are inconsistencies, note that the _type_ values occur more than once among variable combinations. If the _type_ value occurs only once among the combinations of those variables, then the data were consistent. We can also see which test forms these inconsistencies occur because we used the ID statement.

-



	AnswerKey	ItemFunction	ItemType	Position	Subject	ItemID	FormID	_TYPE_	_FREQ_	_STAT_	NumRecords
1					math	11	C	3	6	N	6
2				1	math	11	C	7	6	N	6
3			CR		math	11	C	11	4	N	4
4			MC		math	11	A	11	2	N	2
5		OP			math	11	C	19	6	N	6
6	A				math	11	C	35	6	N	6
7					math	22	C	3	6	N	6
8				2	math	22	C	7	2	N	2
9				3	math	22	B	7	4	N	4
10			MC		math	22	C	11	6	N	6
11		OP			math	22	C	19	6	N	6
12	B				math	22	C	35	2	N	2
13	C				math	22	B	35	4	N	4
14					math	33	A	3	2	N	2
15				2	math	33	A	7	2	N	2
16			CR		math	33	A	11	2	N	2
17		FT			math	33	A	19	2	N	2
18	yellow				math	33	A	35	2	N	2
19					math	44	B	3	2	N	2
20				2	math	44	B	7	2	N	2
21			CR		math	44	B	11	2	N	2
22		FT			math	44	B	19	2	N	2
23	red				math	44	B	35	2	N	2
24					math	55	C	3	2	N	2
25				3	math	55	C	7	2	N	2
26			CR		math	55	C	11	2	N	2
27		FT			math	55	C	19	2	N	2
28	blue				math	55	C	35	2	N	2

The value of _type_ depends on the order in which the variables in the CLASS statement are listed. The decimal value of _type_ appears as the default and using the PROC MEANS CHARTYPE option will produce a character string representative of binary notation, which is equivalent to the decimal value of _type_. For this reason, the `&GroupVars` was placed at the end of the CLASS statement to increase predictability of the value and provide automation opportunities.

AnswerKey	**ItemFunction**	**ItemType**	**Position**	**Subject**	**ItemID**	**Decimal _type_ without chartypes PROC MEANS option**	**Binary _type_ with chartypes PROC MEANS option**
0	0	0	0	1	1	3	000011
0	0	0	1	1	1	7	000111
0	0	1	0	1	1	11	001011
0	1	0	0	1	1	19	010011
1	0	0	0	1	1	35	100011

```
%macro filter;
proc sort data=CountCombos; by &GroupVars
_type_;

data CheckThese ;
length Flag $30.;
retain TotalRecords . ;
set CountCombos;
by &GroupVars _type_;
if first.&IDVar and _type_=&type then
do; TotalRecords=NumRecords; delete; end;
```

This macro will create a `flag` variable that will specify the variable with inconsistency.

When `_type_=3`, then `NumRecords` is the `TotalRecords` for each `subject` and `ItemID` combination.

When _type_ is not 3, then it is the record that counts the unique combination of values for the corresponding variables.

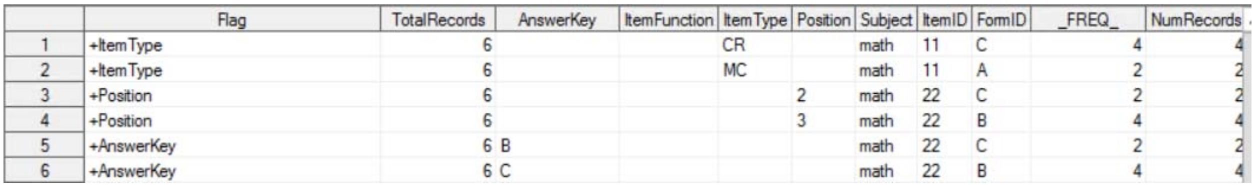
```
if NumRecords ne TotalRecords then do;
  %do i = 1 %to &NDataVars;
   %let column=%scan(&DataVars,&i);
    if &column ne "" then
Flag=strip(Flag)||"+&column";
   %end;
output; drop _stat_ _type_; end;
run;
%mend filter;

%filter;
```

Check if the record needs to be flagged. That is, check if `NumRecords` is not equal to `TotalRecords` for the corresponding `&ItemVar` variable. Only keep flagged records. If `Numrecords=TotalRecords`, that tells us that there was only one value for the column (i.e., data value was consistent for that combination of `subject` and `ItemID`).

If it is a flagged record, the `flag` value will contain the name of the variable with inconsistencies.

Dataset checkthese has the following contents.

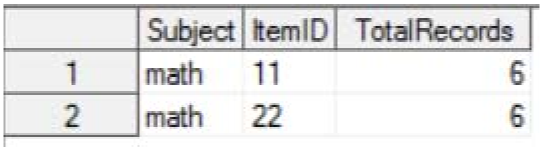


	Flag	TotalRecords	AnswerKey	ItemFunction	ItemType	Position	Subject	ItemID	FormID	_FREQ_	NumRecords
1	+ItemType	6			CR		math	11	C	4	4
2	+ItemType	6			MC		math	11	A	2	2
3	+Position	6				2	math	22	C	2	2
4	+Position	6				3	math	22	B	4	4
5	+AnswerKey	6	B				math	22	C	2	2
6	+AnswerKey	6	C				math	22	B	4	4

We will now take the checkthese data and do a few things to it to produce another way of looking at the data.

```
proc sql;
create table UniqueIDs
as select unique Subject, &IDVar
,TotalRecords from CheckThese
where NumRecords ne TotalRecords;
```

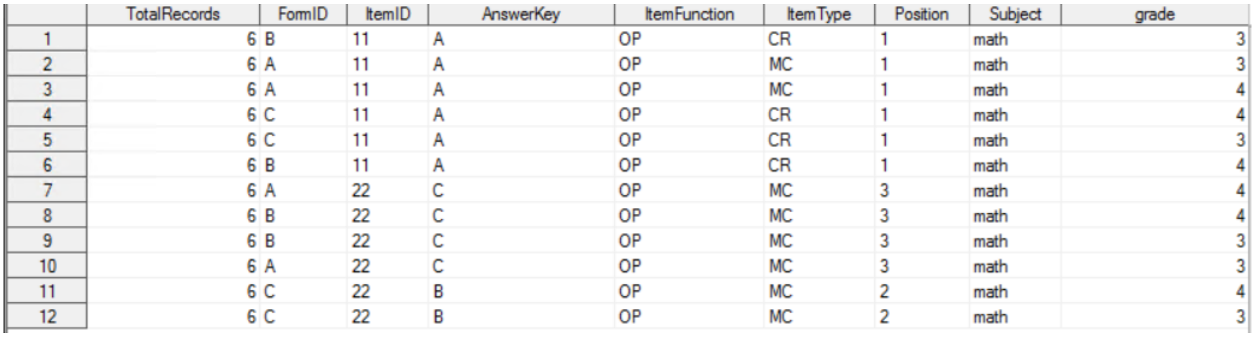
This is the list of items with flagged inconsistencies.



	Subject	ItemID	TotalRecords
1	math	11	6
2	math	22	6

```
proc sql;
create table PreFinal as
select a.TotalRecords, b.*
from UniqueIDs as a left join &InputData
(drop=NumRecords) as b
on a.&IDVar=b.&IDVar and
a.Subject=b.Subject;
```

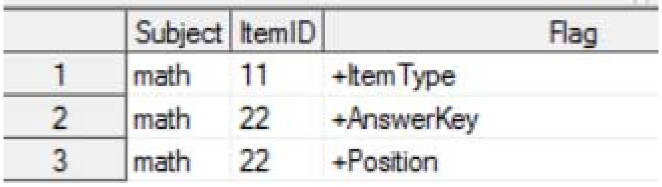
Add the item data to this list of unique IDs with inconsistencies.



	TotalRecords	FormID	ItemID	AnswerKey	ItemFunction	ItemType	Position	Subject	grade
1	6	B	11	A	OP	CR	1	math	3
2	6	A	11	A	OP	MC	1	math	3
3	6	A	11	A	OP	MC	1	math	4
4	6	C	11	A	OP	CR	1	math	4
5	6	C	11	A	OP	CR	1	math	3
6	6	B	11	A	OP	CR	1	math	4
7	6	A	22	C	OP	MC	3	math	4
8	6	B	22	C	OP	MC	3	math	4
9	6	B	22	C	OP	MC	3	math	3
10	6	A	22	C	OP	MC	3	math	3
11	6	C	22	B	OP	MC	2	math	4
12	6	C	22	B	OP	MC	2	math	3

```
proc sql;
create table UniqueFlags
as select unique Subject, ItemID , Flag
from CheckThese;
```

Get all the unique subject, ItemID, and flag combinations from checkthese dataset.

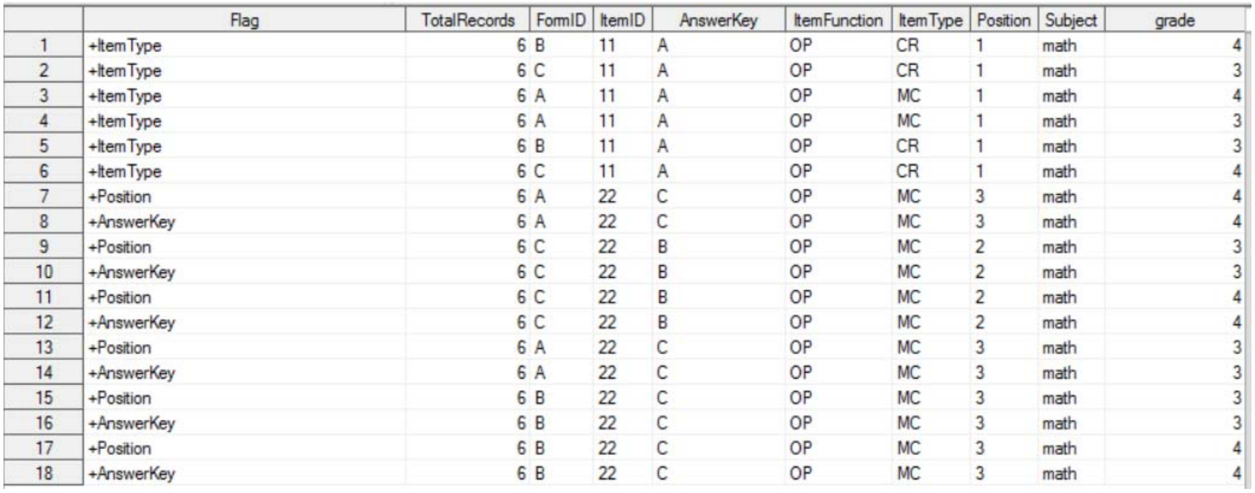


	Subject	ItemID	Flag
1	math	11	+ItemType
2	math	22	+AnswerKey
3	math	22	+Position

```
proc sql;
create table Final as
select a.Flag, b.*
from UniqueFlags as a LEFT JOIN PreFinal
as b
ON a.&IDVar=b.&IDVar and
a.Subject=b.Subject;
quit;
```

This is the data with all the original item data with the flagged records together with the flag values. The flag values tell us which variable is being flagged for each row.

In this data set, we can see all the item data together with the flag value that tells us what the inconsistency is. In the first six records, the flag indicates there is an inconsistency in ItemType values for itemID. In the We see that the ItemType is MC in FormID=A and is CR on the other forms.



	Flag	TotalRecords	FormID	ItemID	AnswerKey	ItemFunction	ItemType	Position	Subject	grade
1	+ItemType	6	B	11	A	OP	CR	1	math	4
2	+ItemType	6	C	11	A	OP	CR	1	math	3
3	+ItemType	6	A	11	A	OP	MC	1	math	4
4	+ItemType	6	A	11	A	OP	MC	1	math	3
5	+ItemType	6	B	11	A	OP	CR	1	math	3
6	+ItemType	6	C	11	A	OP	CR	1	math	4
7	+Position	6	A	22	C	OP	MC	3	math	4
8	+AnswerKey	6	A	22	C	OP	MC	3	math	4
9	+Position	6	C	22	B	OP	MC	2	math	3
10	+AnswerKey	6	C	22	B	OP	MC	2	math	3
11	+Position	6	C	22	B	OP	MC	2	math	4
12	+AnswerKey	6	C	22	B	OP	MC	2	math	4
13	+Position	6	A	22	C	OP	MC	3	math	3
14	+AnswerKey	6	A	22	C	OP	MC	3	math	3
15	+Position	6	B	22	C	OP	MC	3	math	3
16	+AnswerKey	6	B	22	C	OP	MC	3	math	3
17	+Position	6	B	22	C	OP	MC	3	math	4
18	+AnswerKey	6	B	22	C	OP	MC	3	math	4

```
%macro GenerateTables(column);
proc tabulate data=Final out=test
format=12.0;
class Flag &GroupVars &FormVar &DataVars
TotalRecords;
table
&GroupVarsAsterisk.*(&column)*TotalRecor
ds="# of &InputData Records with
ItemID", all='Number of Records with
Combination' FormID;
where find(Flag,"&column")>0;
title4 "&column: Item Inconsistencies
Across Forms";
run;

proc tabulate data=&InputData
format=12.0;
class &GroupVars &FormVar &DataVars ;
table  &GroupVarsasterisk.*(&column),
all FormID;
title4 "&column: Data
Consistency/Inconcistency Across Forms";
run;
%mend GenerateTables;
```

For each variable specified, up to two PROC TABULATE output tables are generated.

The first PROC TABULATE output shows us the inconsistencies only as they appear in the `final` data set. This uses a WHERE statement and therefore, may not produce any output if no records satisfy the condition.

The second one serves as a record for what is in the data and uses the original `&dataset`. Consistencies and/or inconsistencies will appear in the output.

```
%macro feedback(varlist);
  %do i = 1 %to &NDataVars;
   %let column=%scan(&varlist,&i);
    ods excel options
(sheet_name="&column");
   %GenerateTables(column=&column);
  %end;
%mend feedback;
```

The DO loop goes over each variable in `&ndatavars` and runs the `%generatetables` macro for each variable.

ODS Excel options specified the worksheet name.

Each DO loop iteration invokes the `%generatetables` macro for each variable in `&varlist`.

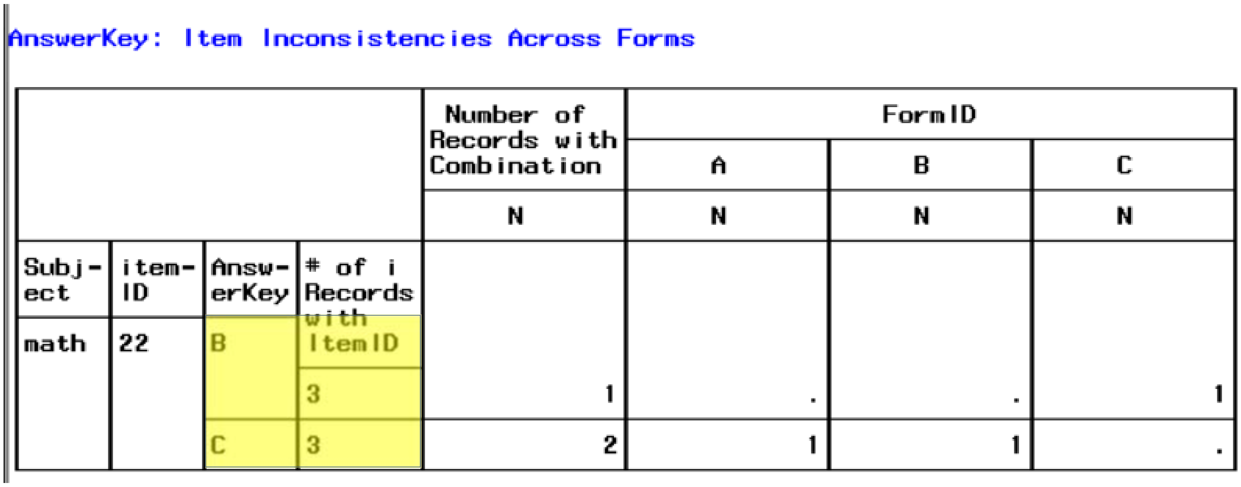
```
ods excel file="&OutPath.\&groupvars
feedback.xlsx" options
(sheet_interval='table'
embedded_titles='yes');
%feedback(varlist=&DataVars);
ods excel close;
```

We are going to send the output to an Excel file using ODS.

We invoke the `%feedback` macro by defaulting the `varlist` to `&DataVars`, which is all the variables we wanted to check the consistency of.

The macro variable `&varlist` has 4 variables in this example. There will be at most 4 x 2 of PROC TABULATE output tables. The first PROC TABULATE output table will appear as long as there were inconsistencies found for the particular variable. The second PROC TABULATE output will always appear since it uses the original data set will all values in it. The PROC TABULATE presentation of the information offers another way of looking at the data**.**

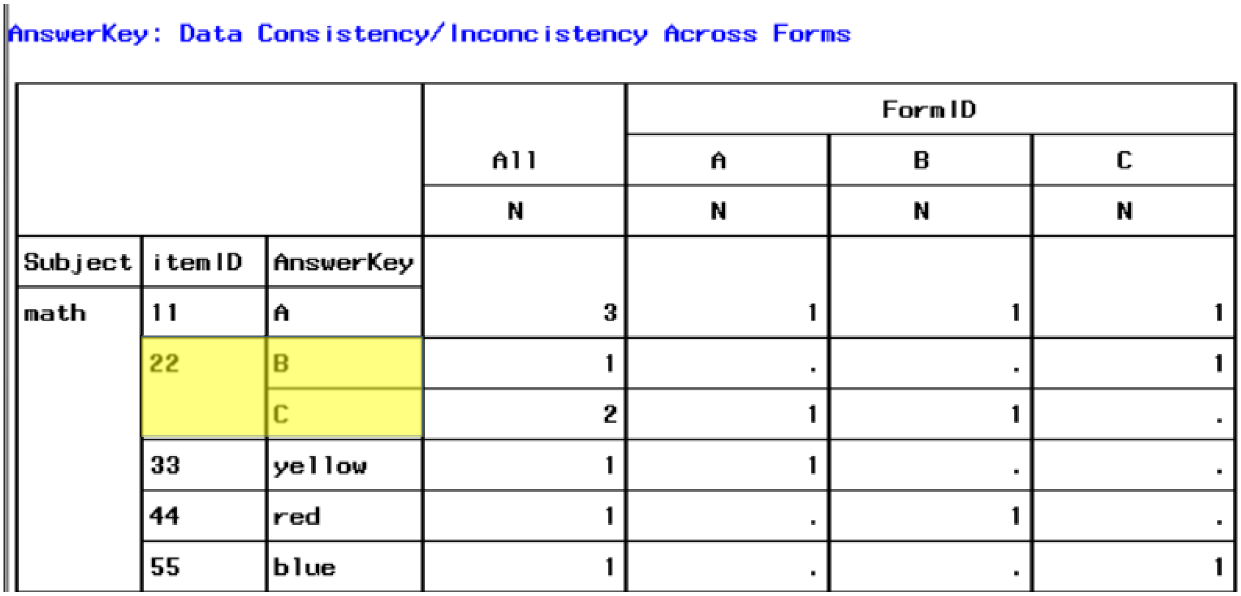
We can see easily below that the `itemID=22` row has two keys. The key of B appears in form C and the key of C appears in forms A and B.



AnswerKey: Item Inconsistencies Across Forms

				Number of Records with Combination	FormID		
					A	B	C
				N	N	N	N
Subj-ect	item-ID	Answ-erKey	# of i Records with ItemID				
math	22	B	3	1	.	.	1
		C	3	2	1	1	.

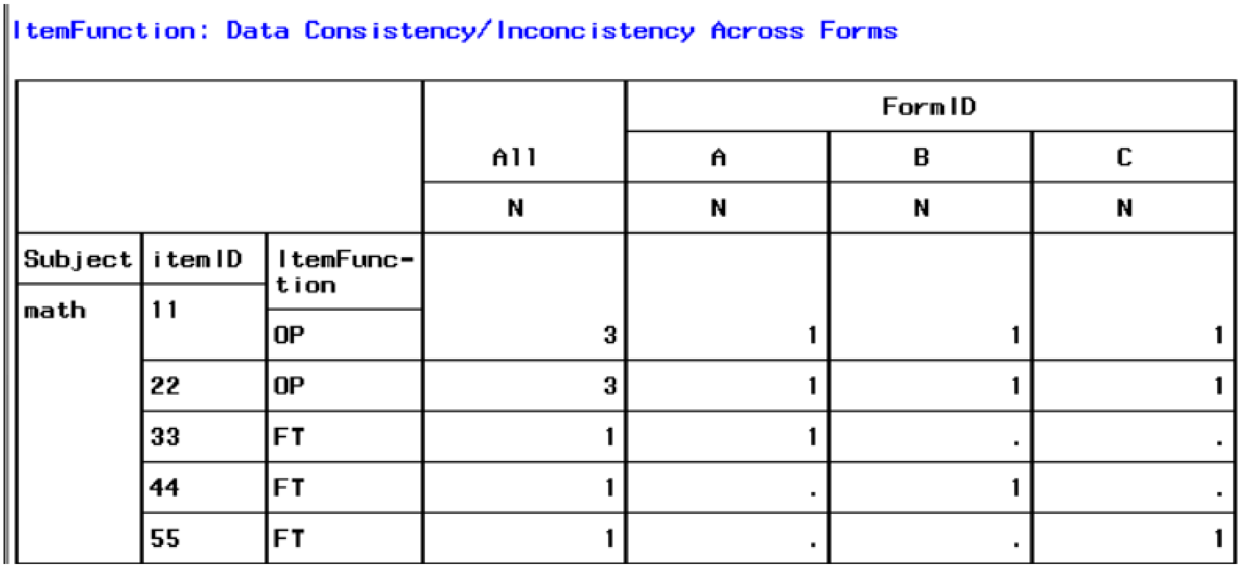
The following is similar to the above table but also contains the information for all `itemID` values.



AnswerKey: Data Consistency/Inconcistency Across Forms

			All	FormID		
				A	B	C
			N	N	N	N
Subject	itemID	AnswerKey				
math	11	A	3	1	1	1
	22	B	1	.	.	1
		C	2	1	1	.
	33	yellow	1	1	.	.
	44	red	1	.	1	.
	55	blue	1	.	.	1

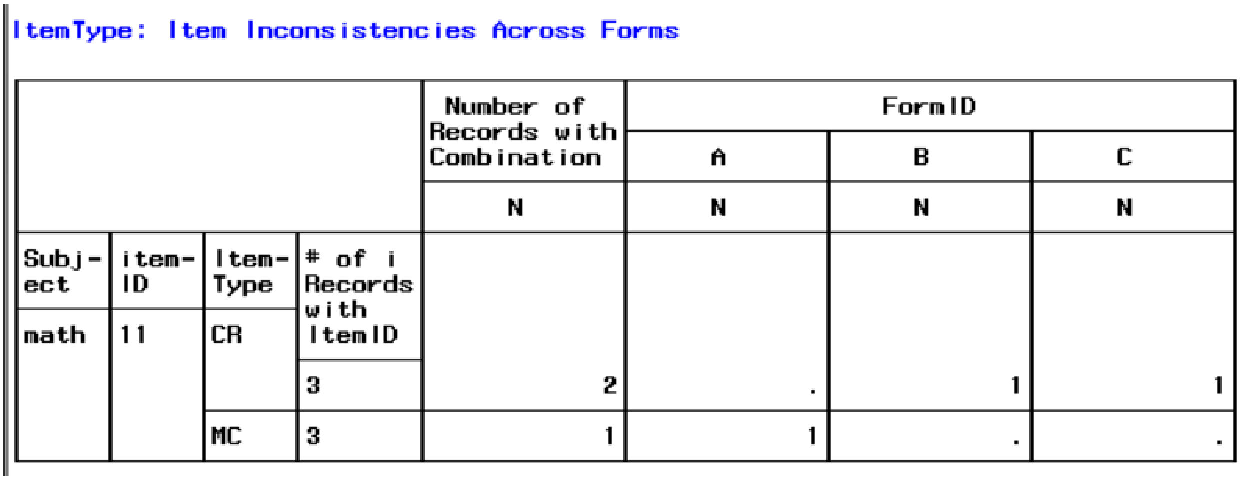
No inconsistencies were detected for `ItemFunction`. Only the second PROC TABULATE output was produced.



ItemFunction: Data Consistency/Inconcistency Across Forms

			All	FormID		
				A	B	C
			N	N	N	N
Subject	itemID	ItemFunction				
math	11	OP	3	1	1	1
	22	OP	3	1	1	1
	33	FT	1	1	.	.
	44	FT	1	.	1	.
	55	FT	1	.	.	1

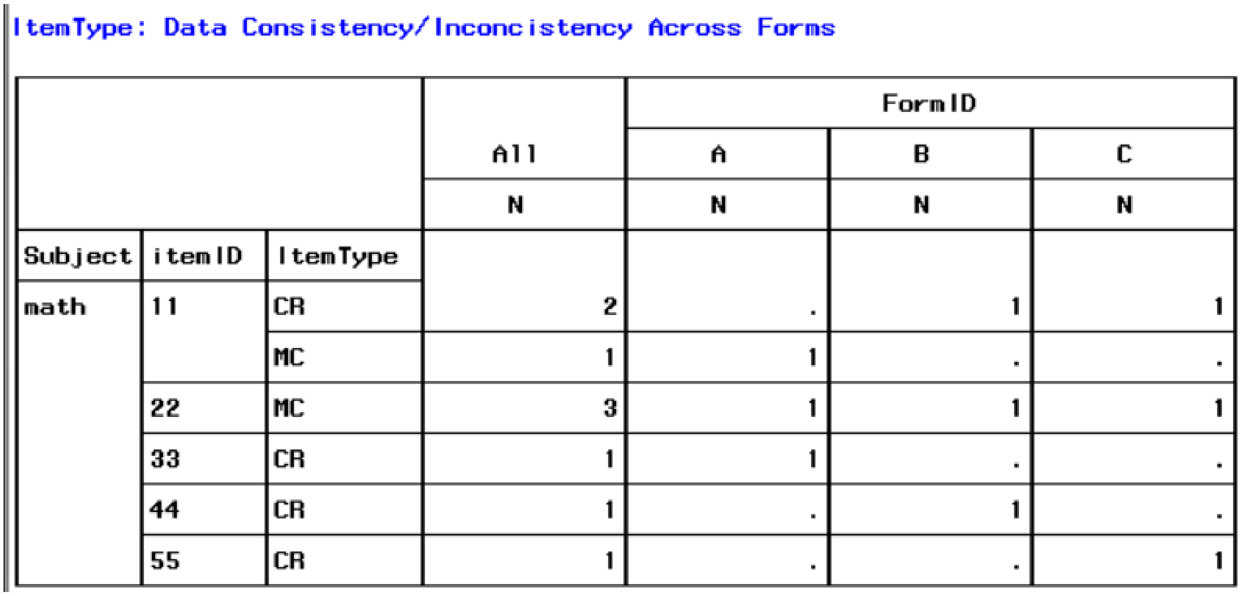
The following table shows that there was an inconsistency in `ItemType` when `ItemID=11`.



ItemType: Item Inconsistencies Across Forms

				Number of Records with Combination	FormID		
					A	B	C
				N	N	N	N
Subject	itemID	ItemType	# of i Records with ItemID				
math	11	CR	3	2	.	1	1
		MC	3	1	1	.	.

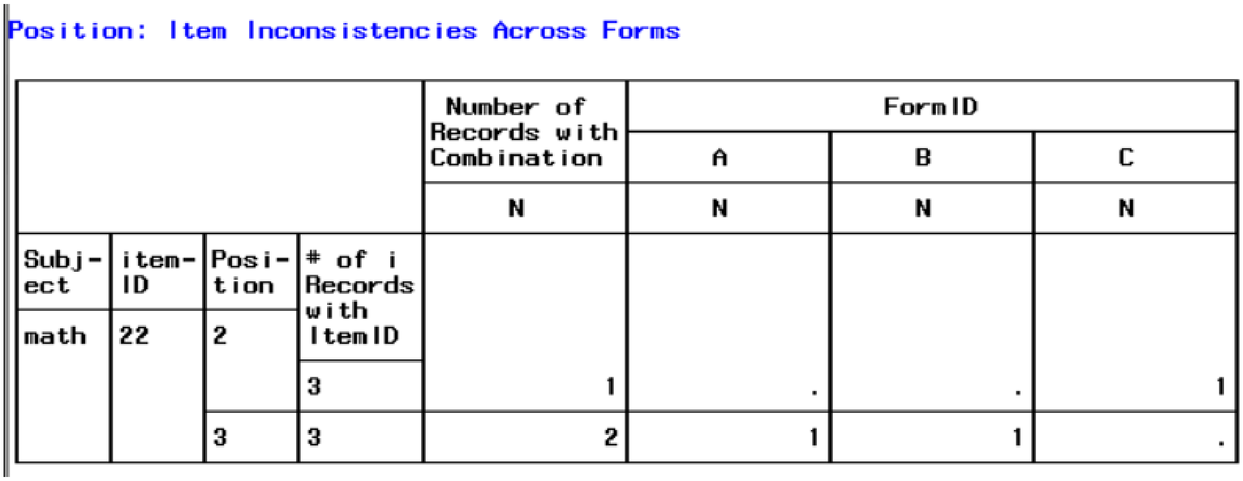
The following contains the information for `ItemType` consistency/inconsistency.



ItemType: Data Consistency/Inconcistency Across Forms

				FormID		
			All	A	B	C
			N	N	N	N
Subject	itemID	ItemType				
math	11	CR	2	.	1	1
		MC	1	1	.	.
	22	MC	3	1	1	1
	33	CR	1	1	.	.
	44	CR	1	.	1	.
	55	CR	1	.	.	1

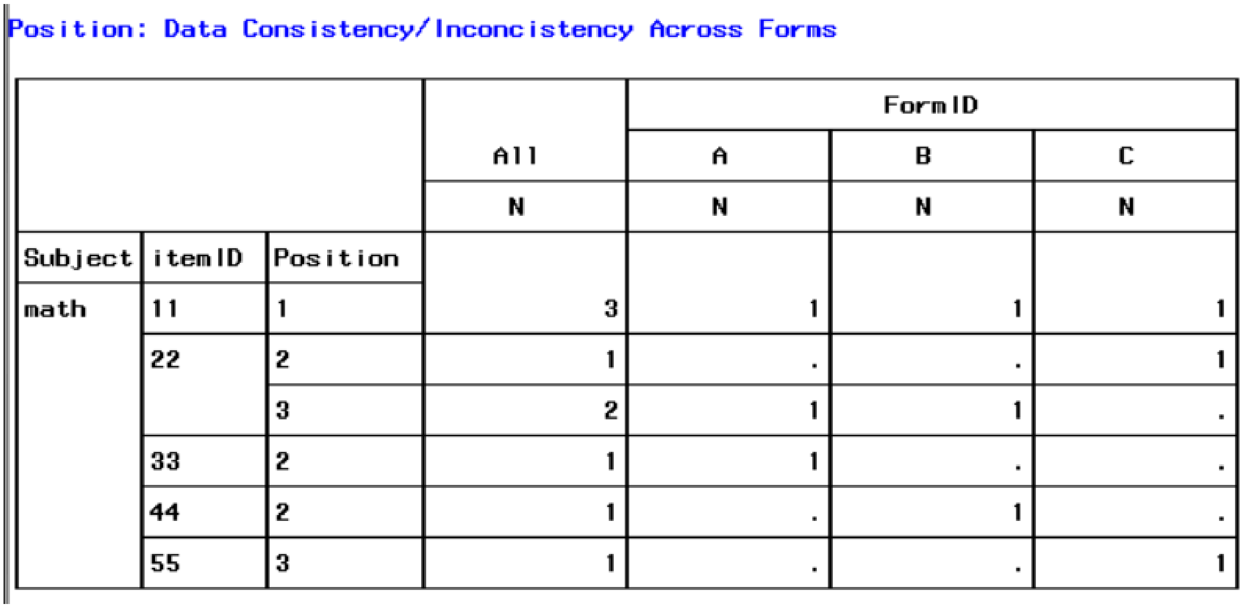
Position also has inconsistencies for `ItemID=22`.



Position: Item Inconsistencies Across Forms

				Number of Records with Combination	FormID		
					A	B	C
				N	N	N	N
Subject	itemID	Position	# of i Records with ItemID				
math	22	2	3	1	.	.	1
		3	3	2	1	1	.

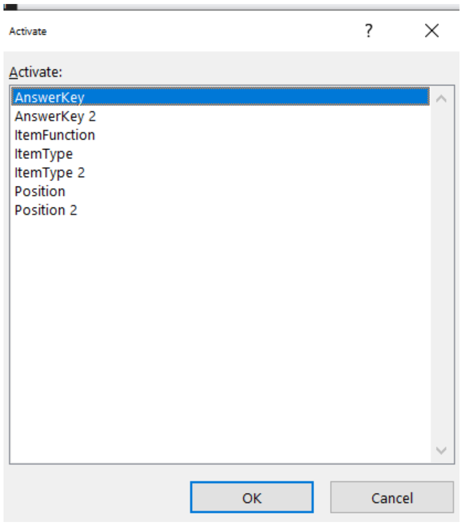
The following shows `Position` values for all items.



Position: Data Consistency/Inconcistency Across Forms

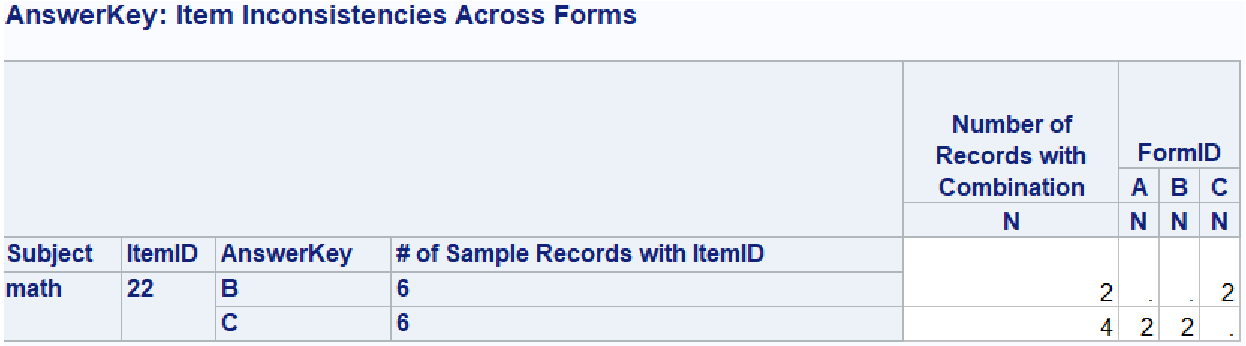
				FormID		
			All	A	B	C
			N	N	N	N
Subject	itemID	Position				
math	11	1	3	1	1	1
	22	2	1	.	.	1
		3	2	1	1	.
	33	2	1	1	.	.
	44	2	1	.	1	.
	55	3	1	.	.	1

All of these output tables were sent to an Excel file with several worksheets, one for each PROC TABULATE output table. The worksheets for produced are as follows.



AnswerKey
AnswerKey 2
ItemFunction
ItemType
ItemType 2
Position
Position 2

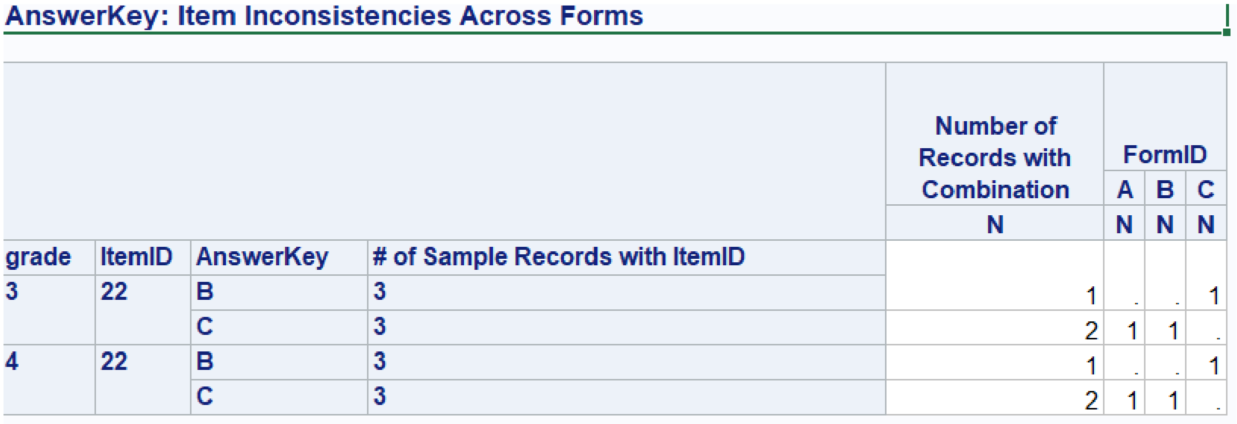
The `&GroupVar` value helps organize the tables according to relevant pools of items. In this example, `&GroupVar=Subject`. This means we want to look at all the duplicates within each subject value.



AnswerKey: Item Inconsistencies Across Forms

				Number of Records with Combination	FormID		
					A	B	C
				N	N	N	N
Subject	ItemID	AnswerKey	# of Sample Records with ItemID				
math	22	B	6	2	.	.	2
		C	6	4	2	2	.

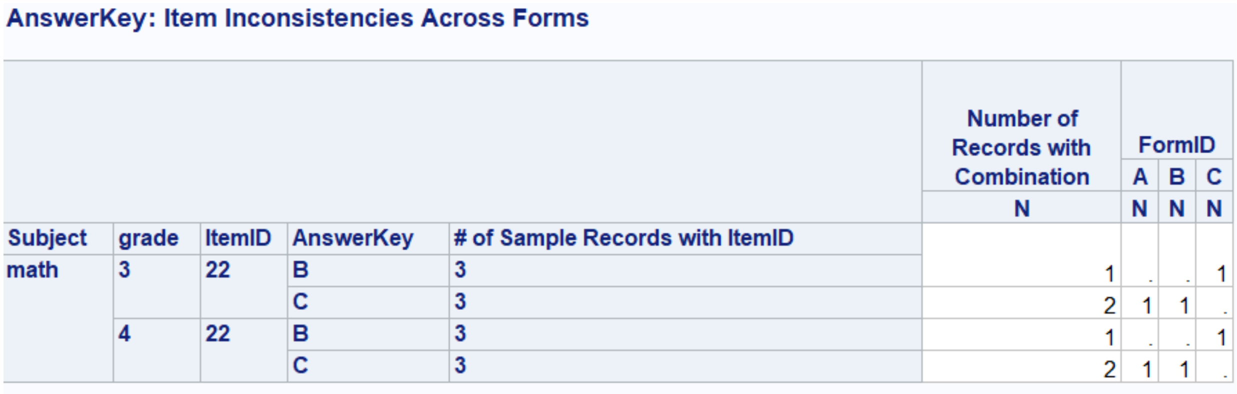
If we use `&GroupVar=grade`, the output will automatically adjust. Here we are looking at duplicates within each grade value.



AnswerKey: Item Inconsistencies Across Forms

				Number of Records with Combination	FormID		
					A	B	C
				N	N	N	N
grade	ItemID	AnswerKey	# of Sample Records with ItemID				
3	22	B	3	1	.	.	1
		C	3	2	1	1	.
4	22	B	3	1	.	.	1
		C	3	2	1	1	.

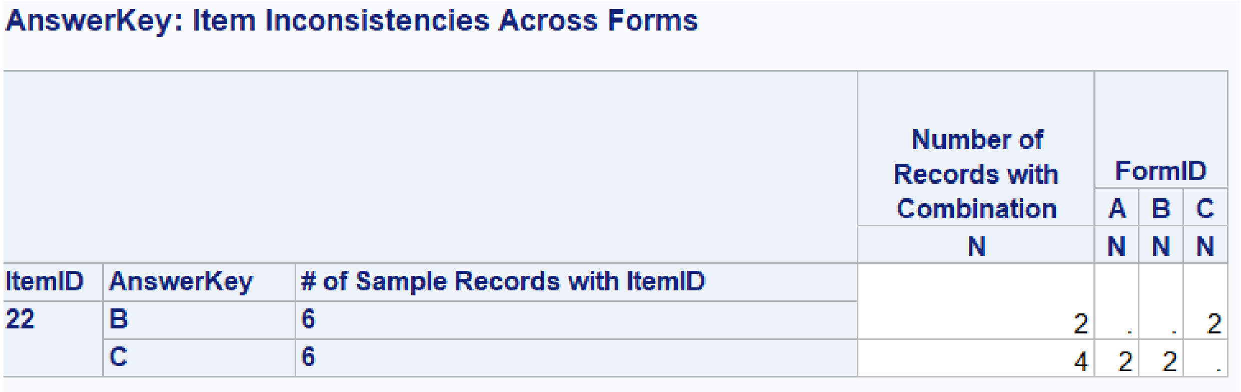
If we use `&GroupVar=subject grade`, we will be looking at duplicates within each combination of subject and grade.



AnswerKey: Item Inconsistencies Across Forms

					Number of Records with Combination	FormID		
						A	B	C
					N	N	N	N
Subject	grade	ItemID	AnswerKey	# of Sample Records with ItemID				
math	3	22	B	3	1	.	.	1
			C	3	2	1	1	.
	4	22	B	3	1	.	.	1
			C	3	2	1	1	.

When `&GroupVar=&ItemVar`, then we will be strictly looking at item inconsistencies by `ItemID` in this example.



AnswerKey: Item Inconsistencies Across Forms

			Number of Records with Combination	FormID		
				A	B	C
			N	N	N	N
ItemID	AnswerKey	# of Sample Records with ItemID				
22	B	6	2	.	.	2
	C	6	4	2	2	.

Here's the SAS coding described above in one continuous section.

```
%let InputData=Sample;
%let DataVars=AnswerKey ItemFunction ItemType Position;
%let IDVar=ItemID;
%let FormVar=FormID;
%let GroupVars=Subject &IDVar.;
%let OutPath=C:\Users\imelda.go\Desktop;
***********************************;
%let GroupVarsAsterisk = %sysfunc(translate(&GroupVars,'*',' '));
%let GroupVarsComma = %sysfunc(translate(&GroupVars,',',' '));
%let NDataVars=%sysfunc(countw(&DataVars));
%let NGroupVars=%sysfunc(countw(&GroupVars));

data TypeValue;
retain type 0;
if &NGroupVars>0 then
do i=1 to &NGroupVars;
type=type+ 2**(i-1);
end;
output;
call symputx('type',type);

proc means data=&InputData noprint n chartype;
class &DataVars &GroupVars/missing;
types &GroupVarsasterisk. &GroupVarsasterisk.*(&DataVars);
var NumRecords;
output out=CountCombos (where=( _stat_="N" ));

proc sort data=CountCombos; by &GroupVars _type_;
```

```
%macro filter;
proc sort data=CountCombos; by &GroupVars _type_;

data CheckThese ;
length Flag $30.;
retain TotalRecords . ;
set CountCombos;
by &GroupVars _type_;
if first.&IDVar and _type_=&type then
do; TotalRecords=NumRecords; delete; end;

if NumRecords ne TotalRecords then do;
  %do i = 1 %to &NDataVars;
   %let column=%scan(&DataVars,&i);
    if &column ne "" then  Flag=strip(Flag)||"+&column";
   %end;
output; drop _stat_ _type_; end;
run;
%mend filter;

%filter;

proc sql;
create table UniqueIDs
as select unique Subject, &IDVar ,TotalRecords from CheckThese
where NumRecords ne TotalRecords;

proc sql;
create table PreFinal as
select a.TotalRecords, b.*
from UniqueIDs as a left join &InputData (drop=NumRecords) as b
on a.&IDVar=b.&IDVar and a.Subject=b.Subject;

proc sql;
create table UniqueFlags
as select unique Subject, ItemID , Flag from CheckThese;

proc sql;
create table Final as
select a.Flag, b.*
from UniqueFlags as a LEFT JOIN PreFinal as b
ON a.&IDVar=b.&IDVar and a.Subject=b.Subject;
quit;

%macro GenerateTables(column);
proc tabulate data=Final out=test format=12.0;
class Flag &GroupVars &FormVar &DataVars TotalRecords;
table  &GroupVarsAsterisk.*(&column)*TotalRecords="# of &InputData Records
with ItemID", all='Number of Records with Combination' FormID;
where find(Flag,"&column")>0;
title4 "&column: Item Inconsistencies Across Forms";

proc tabulate data=&InputData format=12.0;
class &GroupVars &FormVar &DataVars ;
table  &GroupVarsasterisk.*(&column), all FormID;
title4 "&column: Data Consistency/Inconcistency Across Forms";
run;
%mend GenerateTables;
```

```
%macro feedback(varlist);
  %do i = 1 %to &NDataVars;
   %let column=%scan(&varlist,&i);
    ods excel options (sheet_name="&column");
   %GenerateTables(column=&column);
  %end;
%mend feedback;

ods excel file="&OutPath.\&groupvars feedback.xlsx" options
(sheet_interval='table' embedded_titles='yes');
%feedback(varlist=&DataVars);
ods excel close;
```

CONCLUSION

The SAS programming language offers different tools to identify data inconsistencies. The use of different tools together has the potential to result in greater automated and data-driven coding efficiency.

REFERENCES

Lafler, Kirk Paul (2017). “Removing Duplicates Using SAS®”, Proceedings of the 2017 SAS Global Forum (SGF) Conference.

CONTACT INFORMATION

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

Imelda C. Go, Ph.D.
Cognia, Inc.
imelda.go@cognia.org