

Paper SBC-127

## Identifying and listing outliers without using PROC Format option Other=Invalid

Ruben Chiflikyan, RTI International, Research Triangle Park, NC  
Mila Chiflikyan, RTI International, Research Triangle Park, NC  
Donna Medeiros, RTI International, Research Triangle Park, NC

### ABSTRACT

The identification of outliers in formatted datasets is a critical data quality issue, particular in the data review process. Having a long history, it culminated by Ronald Fehd establishing a set of powerful macros for performing range and logic checks on datasets and the writing of file exceptions to edit and use for updates. As a necessary condition for this approach, the existence of a properly written data dictionary containing the other=INVALID option for all format values is required. In practice, however, often one has to work with the formats that do not have the required structure, and, for various reasons, it is difficult to transform them in the form required. As a supplement to the established approach, we have created a macro that handles situations where the presence of the other=INVALID option is not obligatory in character, numeric regular and multi-label value formats. As an output of this macro, we obtain a table with the identifier(s), variable name, variable format, and an unformatted value, which can be used for future analysis.

**KEYWORDS:** data cleaning, formats, value labels, outliers, macros, macro variables, arrays.

### INTRODUCTION

It well known that the creation of high quality of data, is one of the highest priorities in data management. Many approaches and techniques have been developed and successfully implemented to handle various aspects of this multifaceted complex problem (see, e.g., [1] and references therein). A constituent part of creation of the well-defined data dictionary, the identification of outliers in the formatted datasets is a critical data quality issue, especially in the data review process. A consistent approach for performing range and logic checks on datasets and the writing of file exceptions to edit and use for updates was implemented successfully in a series of macros created by Ronald Fehd [2, 3]. As a necessary condition for this approach, the existence of a properly written data dictionary containing the other=INVALID option for all format values is required. In practice, however, often one has to work with the formats that do not have the required structure. As a supplement to one of the established approaches above, we have created a program that handles the situations, where the presence of the other=INVALID option is not obligatory in character, numeric regular and multi-label value formats. As an output of this macro, we obtain a table with the identifier(s), variable name, variable format, and an unformatted value, which can be used for future analysis.

Say for instance, we have a dataset, where user-defined formats are applied to some of the variables. Ideally, one expects that all these variables should have formatted values, or value labels. However, it does not happen in cases when the formats are not well defined or contain some 'unexpected' values. So, before taking care of all these inconsistencies, at first, one needs to identify all those fields with such inconsistencies. We restrict our analysis only by creating of an output file with all problematic cases, and further steps to work with this data will be not considered here.

### PROGRAM DESCRIPTION

The following program demonstrates the mechanics using car example data.

The formats below were created for applying to certain variables in the dataset to be defined below. We need to check whether all variables labels are affected by these formats. All unformatted variables values – outliers - will identified and outputted in a single SAS® dataset for possible analytical manipulations.

For illustration purposes, 2 regular numeric, 1 multilabel numeric, and 2 character formats will be used here.

```
proc format;
value mpgf
-1      = 'dont know'
```

```

. = 'missing'
10 - 15 = 'poor mpg'
15 - 30 = 'average mpg '
30 - 40 = 'excellent mpg'
Other = 'OTHER VALUES'
;
value speedf (multilabel)
60 < - 80 = 'low speed'
80 < - 100 = 'medium speed 1'
100 < - 120 = 'medium speed 2'
80 < - 120 = 'medium speed'
120 < - high = 'high speed'
;
value yearsf
-1 = 'dont know'
. = 'missing'
1 - 5 = 'group 1'
6 - 10 = 'group 2'
11 - 15 = 'group 3'
;
value $ modelf
'model_a', 'model_b' = 'MODEL AB'
'model_c' = 'MODEL C'
' ' = 'Missing'
;
value $ makef
'make_a' = 'MAKE A'
'make_b', 'make_c' = 'MAKE BC'
; run;

```

The mpgf., speedf., yearsf., \$modelf., and \$makef. formats defined above will be applied accordingly to the mpg, speed, years, model, and make variables defined in the zero dataset defined below. In Fig., one can see the dataset with the following unformatted variables: id \$1., id2 \$1., mpg 4., speed 4., years 4., model \$7., and make \$6.. Later, in Fig.2, you see the same table, where the formats mpgf., speedf., \$modelf., \$makef., and yearsf. are applied to the mpg, speed, model, make, and years variables, accordingly. (By comparing these 2 tables, one can see which data fields could not be formatted.)

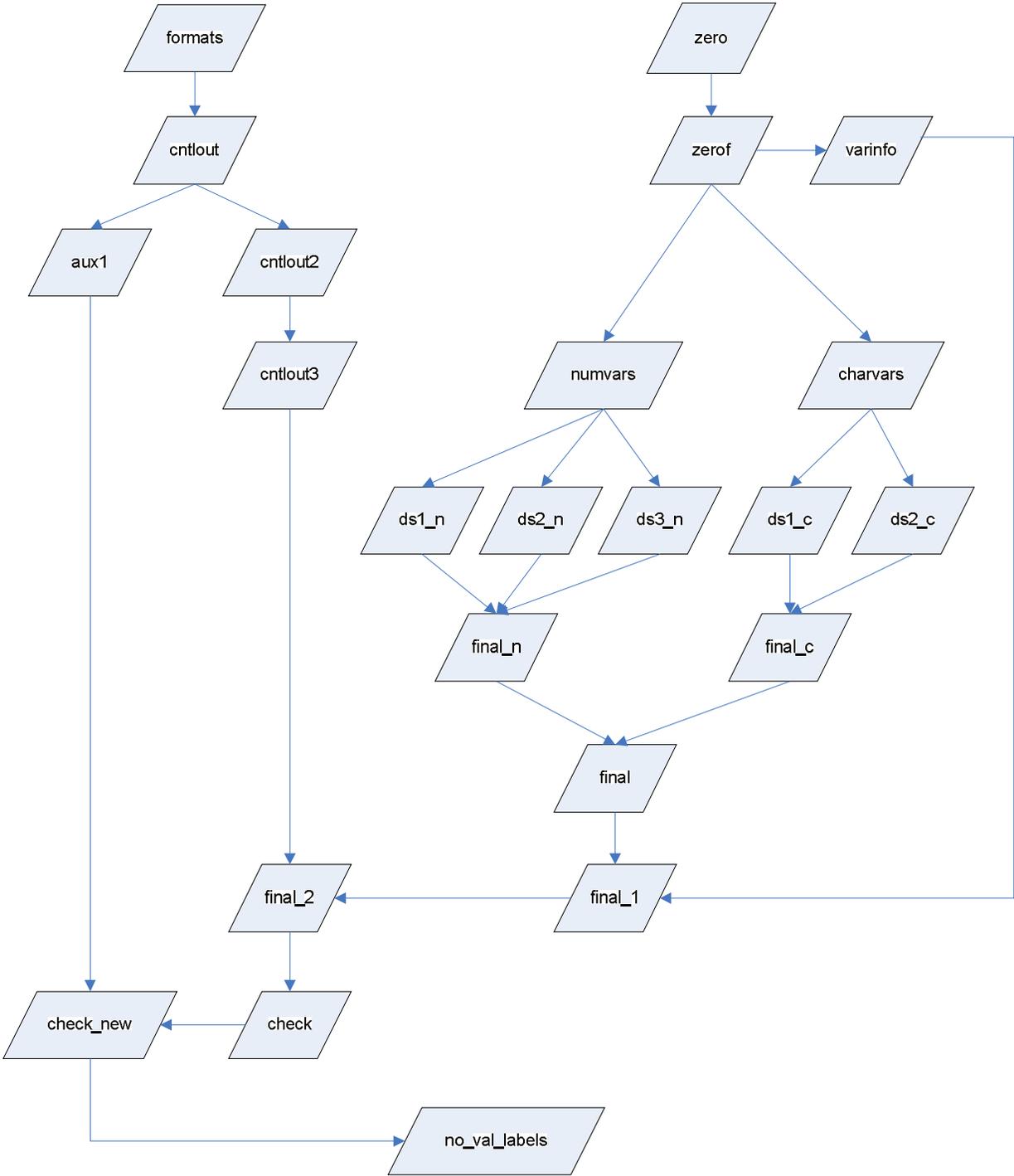
This is the outlook of the dataset with unformatted variable values:

Obs	id	id2	mpg	speed	years	model	make
1	1	1	-1	120	1	model_a	make_a
2	2	1	.	110	2	model_b	make_c
3	3	2	19	80	5	model_a	make_c
4	4	3	32	-70	10	model_c	make_b
5	6	3	23	130	12	model_d	
6	6	4	13	150	6	model_b	make_a
7	6	5	19	70	16	model_a	make_d
8	7	1	5	170	.	model_c	make_c

**Figure 1.** The table with unformatted variable values.

Tracing the details of this program, a flowchart of the main processes is presented on Fig.2. The parallelograms denote SAS data sets, and the arrows show the order of their creation in the program. All the necessary specific details can be found the paper text below.

**The flow chart of the main processes**



**Figure 2.** The flow chart of the main processes.

```

* assigning formats and labels to the variables in the zero dataset;
data zerof;
set zero;
label id='First Identificator' id2='Second Identificator' mpg='Miles per
Gallon'
speed='Miles per Hour' model='Model of the car' make ='Make of the car'
years='Years in service';
format mpg mpgf. speed speedf. model $modelf. make $makef. years yearsf. ;
run;

```

Assigning formats to the variables in Fig 3.

Obs	id	id2	mpg	speed	years	model	make
1	1	1	dont know	medium speed	group 1	MODEL AB	MAKE A
2	2	1	missing	medium speed	group 1	MODEL AB	MAKE BC
3	3	2	average mpg	low speed	group 1	MODEL AB	MAKE BC
4	4	3	excellent mpg	-70	group 2	MODEL C	MAKE BC
5	6	3	average mpg	high speed	group 3	model_d	
6	6	4	poor mpg	high speed	group 2	MODEL AB	MAKE A
7	6	5	average mpg	low speed	16	MODEL AB	make_d
8	7	1	OTHER VALUES	high speed	missing	MODEL C	MAKE BC

**Figure 3.** The same table with formatted variable values.

When comparing visually the tables in Fig1 and Fig.3, one can see that there are problematic fields - fields with unformatted values - for speed variable in obs #4, for years variable in obs #7, and for model and make variables in obs #5. The appropriate corrections can be easily performed for small datasets, however, in datasets with many hundred variables and hundred thousand of observations manual approach would be quite time-consuming. The goal of approach is creation of the program that will identify all problematic fields and output them

Firstly, we need to identify all case identifiers. In this specific case we have only two, namely: id and id2;

```
%let ids1=%upcase(id id2);
```

Further, using OPEN function, we create varnam, varlbl, varfrmt, and vartyp variables, whose values are variable names, labels, formats, and types, accordingly.

```

data varinfo;
length varnam varlbl varfrmt $20 vartyp $1;
drop dsid i rc;
dsid=open("zerof","i");
num=attrn(dsid,"nvars");
do i=1 to num; *num is the number of vars;
varnam=upcase(varname(dsid,i));
varlbl=upcase(varlabel(dsid,i));
varfrmt=upcase(varformat(dsid,i));
vartyp=upcase(vartype(dsid,i));
output;
end;
rc=close(dsid);
drop num;
run;

```

Before going further, we need be sure that the identifiers defined in \$ids1 macro variable are in the dataset under investigation. All necessary testing is quite straightforward and is done in %test1 macro below.

\* let us check whether all values of 'varnam' variable in ids dataset are among the values of the same variable in varinfo dataset;

```

%macro test1(ids=&ids1);
* defines how many words are in the macro string;
%let ids1=%sysfunc(compbl(&ids));
%let lng1=%sysfunc(length(&ids1));
%let ids2=%sysfunc(compress(&ids));
%let lng2=%sysfunc(length(&ids2));
%let numidss=%eval(&lng1-&lng2+1);

* creating dataset with one variable named varnam with all identifiers names;
data ids;
length varnam $20; %do I=1 %to &numidss; varnam="%scan(&ids,&I)"; output; %end;
run;

proc sql;
create table comm as
select ids.varnam from ids, varinfo
where ids.varnam=varinfo.varnam;

select count(*) into :numrows
from comm;
quit;

data _null_;
if &numrows < &numidss
then do;
put "The list of IDs is not correct!";
abort;
end;
else put " The list of IDs is correct";
run;
%mend test1;
%test1(ids=&ids1);

```

In the SQL statement below we create the following macro variables &onlynums and &onlychars – the number of all numeric and character variables, accordingly, &allnums and &allchars – the list all of numeric variables and character variables, accordingly.

```

proc sql noprint;
select count(*), varnam
into :onlynums, :allnums
separated by " "
from (select varnam, vartyp
from varinfo(where=(vartyp='N'))
where varnam not in (select varnam from ids)
)
;
select count(*), varnam
into :onlychars, :allchars
separated by " "
from (select varnam, vartyp
from varinfo(where=(vartyp='C'))
where varnam not in (select varnam from ids)
)
;
quit;

```

Here &allnums=MPG SPEED YEARS, &onlynums=3, &allchars=MODEL MAKE, and &onlychars=2.

To proceed further, from zerof original dataset, we create numvars and charvars datasets, which contain besides identifiers only numeric and character variables, accordingly.

```

data numvars(keep=&ids1 &allnums) charvars(keep=&ids1 &ids1 &allchars);
set zero;
format _all_;
run;

```

The goal of the next %charnms(p=, only=, all=, ds=) macro is creation of the new varnam variable, whose values will be variable names. Firstly, from the numvars dataset we create ds1\_n, ds2\_n, and ds3\_n datasets. For example, ds1\_n dataset contains 4 variables: 2 identifiers, mpg variable and varnam = "mpg" for all observations. For creation of varnam variable VNAME function was used. The ds2\_n, and ds3\_n datasets were created similarly. Note the number of the datasets (in this case =3) is dynamically determined by the number of numeric variables (&onlynums) in the zero dataset. After it, all 3 datasets are set together in the newly created final\_n dataset. In the similar fashion, the ds1\_c and ds2\_c dataset are created, and set together to the final\_c dataset. The first 12 lines of final\_n dataset are presented in Fig4.

Obs	id	id2	varval	varnam
1	1	1	-1	MPG
2	1	1	120	SPEED
3	1	1	1	YEARS
4	2	1	.	MPG
5	2	1	110	SPEED
6	2	1	2	YEARS
7	3	2	19	MPG
8	3	2	80	SPEED
9	3	2	5	YEARS
10	4	3	32	MPG
11	4	3	-70	SPEED
12	4	3	10	YEARS

**Figure 4.** The subset of the final\_n dataset with variable names as values of varnam.

The usage of keyword parameters in %charnms(p=, only=, all=, ds=) macro is as follows: p accepts 'n' or 'c' values, identifying the type of variables under consideration; only is equal to the number of variables contained in &onlynums and &onlychars macro variables; all is a string with variable names presented by &allnums and &allchars; ds is the dataset name( numvars or charvars) that is to be transformed.

%charnms(p=n, only=&onlynums, all=&allnums, ds=numvars);

```

%macro charnms(p=, only=, all=, ds=);
%*local I J K L d;
data %do K=1 %to &only;
    %let d=%scan(&all,&K,' ');
    ds&K._&p(keep=&ids1 &d z&K rename=(&d=varval_&p z&K =varnam))
%end;

length varnam $20 varval_&p $20 %do J=1 %to &only; z&J %end; $20;
varnam=' ';
varval_&p=' ';
set &ds;

* representing variable names as values of z&I variables;
%do I=1 %to &only;
    %let nm=%scan(&all,&I,' ');

```

```

        z&I=vname(&nm);
%end;
run;

%do L=1 %to &only;
    proc sort data=ds&L._&p; by &ids1; run;
%end;

data final_&p(keep=varval &ids1 varnam);
length varval $20;
set

%do M=1 %to &only;
    ds&M._&p
%end;
;
by &ids1;
varval=varval_&p;
varval=trim(left(varval));
varnam=upcase(varnam);
run;

%mend charnms;
%charnms(p=n, only=&onlynums, all=&allnums, ds=numvars);
%charnms(p=c, only=&onlychars, all=&allchars, ds=charvars);

```

After creating of final\_n and final\_c dataset with the same variable names, they are combined together.

```

data final; *combining;
set final_n final_c;
run;

proc sort data=final; * for future merge;
by varnam;
run;

proc sort data=varinfo; * for future merge;
by varnam;
run;

```

In the next datastep, variable label, format, and type information contained in varinfo dataset is added.

```

data final_1;
merge final(in=in1) varinfo(in=in2);
by varnam;
if in1 and in2;
varfrmt=upcase(scan(varfrmt,1,'. '));
if substr(varfrmt,1,1)='$' then varfrmt=substr(varfrmt,2);
run;

```

Now, we are creating the cntlout dataset from the formats above.

```

proc format library=work.formats
    cntlout=cntlout(keep=fmtname start end label);
run;

```

Here we create cntlout2 dataset containing start ^='''OTHER''' cases and aux1 dataset with start ^='''OTHER''''. To include 'LOW' and 'HIGH' values of the end variable into comparison scheme, we formally replace them with very big numbers.

```

data cntlout2(keep=varfrmt start end label) aux1(keep=varfrmt);
length varfrmt $20;
set cntlout;
start=trim(left(start));
end=trim(left(end));
varfrmt=trim(left(upcase(fmtname)));

```

```

if end='HIGH' then end='1000000';
if start='LOW' then start='0';
if start ^= "***OTHER**" then output cntlout2;
else output aux1;
run;

proc sort data=aux1; * preparing for future merge;
by varfrmt;
run;

proc sort data=final_1; * for future merge;
by varfrmt;
run;

```

As one can see from the cntlout dataset, the number of elements in each group in fmrtname variable is defined by the structure of PROC FORMAT above. The PROC SQL below defines the maximum number of elements among all both numeric and character variables in the dataset.

```

proc sql noprint;
select max(max_grp) into :gmax
from (select varfrmt, count(varfrmt) as max_grp
from cntlout2
group by varfrmt);
quit;

```

Here is the cntlout2 dataset derived from the formats defined above.

varfrmt	start	end	label
MAKEF	make_a	make_a	MAKE A
MAKEF	make_b	make_b	MAKE BC
MAKEF	make_c	make_c	MAKE BC
MODEL			Missing
MODEL	model_a	model_a	MODEL AB
MODEL	model_b	model_b	MODEL AB
MODEL	model_c	model_c	MODEL C
MPGF	-1	-1	dont know
MPGF	.	.	missing
MPGF	10	15	poor mpg
MPGF	15	30	average mpg
MPGF	30	40	excellent mpg

**Figure 5.** The subset of cntlout2 dataset derived from formats.

```

proc sort data=cntlout2(keep=varfrmt start end label);*sorting before
transformation;
by varfrmt start;
run;

```

Here, we restructure cntlout2 dataset by creating a single observation from multiple observations with the same value of varnam variable.

```

data cntlout5;

```

```

length lb1-lb%trim(%left(&gmax)) $20;
set cntlout4;
by varfrmt;
array strt[*]$ st1-st%trim(%left(&gmax));
array ends[*]$ en1-en%trim(%left(&gmax));
array labels[*] $ lb1-lb%trim(%left(&gmax));
retain J st1-st%trim(%left(&gmax))
        en1-en%trim(%left(&gmax))
        lb1-lb%trim(%left(&gmax));
if first.varfrmt then do; J=0;
do I=1 to &gmax; strt(I)=.; ends(I)=.; labels(I)=.; end; end;
J=J+1; strt(J)=start; ends(J)=end; labels(J)=label;
if last.varfrmt;
drop I J start end label; label varfrmt=' ';
run;

```

The restructured table where the values of varfrmt variable are unique can be seen in Fig.6.

varfrmt	lb1	lb2	lb3	lb4	lb5	st1	st2
MAKEF	MAKE A	MAKE BC	MAKE BC	.	.	make_a	make_b
MODEL F	Missing	MODEL AB	MODEL AB	MODEL C	.		model_a
MPGF	dont know	missing	poor mpg	average mpg	excellent mpg	-1	.
SPEEDF	medium speed 2	high speed	low speed	medium speed 1	medium speed	100	120
YEARSF	dont know	missing	group 1	group 3	group 2	-1	.

st3	st4	st5	en1	en2	en3	en4	en5
make_c	.	.	make_a	make_b	make_c	.	.
model_b	model_c	.		model_a	model_b	model_c	.
10	15	30	-1	.	15	30	40
60	80	80	120	1000000	80	100	120
1	11	6	-1	.	5	15	10

**Figure 6.** The restructured cntlout3 table with unique varfrmt values.

Finally, we need to merge by varfrmt variable the final\_1 dataset, which contains varnam, varval, id, id1, varlbl, varfrmt, and vartyp variables, with the cntlout3 dataset, which contains varfrmt, lb1-lb5, st1-st5, and en1-en5 variables. It is done to find inconsistencies between varval variable values and start and variables.

```

* for future merge;
proc sort data=cntlout3;
by varfrmt;
run;

* adding all segments to each variable value;
data final_2;
merge final_1(in=in1) cntlout3(in=in2);
by varfrmt;
if in1 and in2;
run;

```

In the dataset below, flag=1 is created for those observations, where there is at least one value of varval that is between the corresponding values of start and end variables. The value flag=0 is assigned for the rest of cases.

```

data check(keep=&ids1 varnam varval varfrmt flag);
set final_2;
array strt[*] $ st1-st%trim(%left(&gmax));
array ends[*] $ en1-en%trim(%left(&gmax));
flag=0;
if vartyp='N' then
do;
  do I=1 to &gmax;
    if input(strt(I),4.) <= input(varval,4.) <= input(ends(I),4.) then
      do;
        flag=1;
        leave; *current loop ends;
      end;
  end;
end;
if vartyp='C' then
do;
  do I=1 to &gmax;
    if strt(I) <= varval <= ends(I) then
      do;
        flag=1;
        leave; *current loop ends;
      end;
  end;
end;
run;

```

Below you can the subset of the check dataset with all cases with flag=0.

Obs	id	id2	varnam	varfrmt	varval	flag
5	6	3	MAKE	MAKEF		0
7	6	5	MAKE	MAKEF	make_d	0
13	6	3	MODEL	MODEL F	model_d	0
24	7	1	MPG	MPGF	5	0
28	4	3	SPEED	SPEED F	-70	0
39	6	5	YEARS	YEARS F	16	0

**Figure 7.** All cases with missing variable labels (stage 1).

By comparing Fig.7 table with final\_2 taking into account PROC FORMAT procedure above, we see that flag=0 is assigned 'incorrectly' to the observation #24. According to the mpgf. format, the value of varval=5 falls into the category other='OTHER VALUES'. To take care of situations with this type of formats, we use aux1 table, which contains the names of formats with other='OTHER VALUES' option and make the corresponding corrections in the next 2 data steps. It is easy to check that now the value of flag variable in the newly created check\_new dataset equals 0 for observation #24.

```

proc sort data=check;
by varfrmt;
run;

* recoding flag=0 to flag=1 if the format had 'other' option;
data check_new;
merge check(in=in1) aux1(in=in2);
by varfrmt;
if in1 and in2 and flag=0 then flag=1;

```

```
run;
```

This is the final dataset with all cases, for which data labels were not found.

```
data no_val_labels;  
set check_new;  
if flag=0;  
keep &ids1 varnam varval varfmt;  
label varnam='Variable Name'  
varval='Variable Value';  
run;
```

The no\_val\_labels dataset has the following form identifying all cases with problematic varval values:

Obs	id	id2	varnam	varfmt	varval
1	6	3	MAKE	MAKEF	
2	6	5	MAKE	MAKEF	make_d
3	6	3	MODEL	MODEL F	model_d
4	4	3	SPEED	SPEED F	-70
5	6	5	YEARS	YEARS F	16

**Figure 8.** The final table with all data fields not having value labels.

## CONCLUSION

We presented quite straightforward method for catching data fields that do not have corresponding value labels. This approach can be considered as complementary to one of the techniques developed earlier by Ronald Fehd. As opposed to his approach we do not require the presence of PROC FORMAT option=INVALID to be present in formats used for formatting. Taking into account that in reality one can never expect that that PROC FORMAT procedure will contain INVALID option *all* formats, our approach can be used, at least, as the first check. The final dataset contains all problematic data fields that for some reasons are without labels. The output variables are identifier(s), variable and format names, and problematic variable value. The subsequent actions will depend on project specifications.

## REFERENCES

1. Cody, Ron, (1999) Cody's Data Cleaning Technique Using SAS Software, Cary, NC: SAS Institute Inc, 1999, 226 p.
2. Fehd, Ronald, INVALID: a Data Review Macro Using PROC FORMAT Option OTHER=INVALID to identify and List Outliers, Proceedings of PharmaSUG 2004, May 3-26, 2004, San Diego, California.
3. Fehd, Ronald, A Beginner's Tour of a Project using SAS Macros Led by SAS-L's Macro Maven, Proceedings of the 26<sup>th</sup> Annual SAS Users Group International Conference, April 22-25, 2001, Long Beach, California.

## CONTACT INFORMATION

Your comments and questions are valued and encouraged. The electronic copy of this paper is available. Please, contact the authors at:

Ruben Chiflikyan  
Research Triangle Institute, International  
3040 Cornwallis Road, PO Box 12194  
Research Triangle Park, NC, 27709-2194  
Work Phone: (919)541-6064  
Fax: (919)541-6178  
E-mail: rchiflikyan@rti.org  
Web: <http://www.rti.org>

Mila Chiflikyan  
Research Triangle Institute, International  
3040 Cornwallis Road, PO Box 12194  
Research Triangle Park, NC, 27709-2194  
Work Phone: (919)316-3360  
Fax: (919)541-6178  
E-mail: [milachif@rti.org](mailto:milachif@rti.org)  
Web: <http://www.rti.org>

Donna Medeiros  
Research Triangle Institute, International  
3040 Cornwallis Road, PO Box 12194  
Research Triangle Park, NC, 27709-2194  
Work Phone: (919)6000-21064  
Fax: (919)541-6178  
E-mail: [djm@rti.org](mailto:djm@rti.org)  
Web: <http://www.rti.org>

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.  
Other brand and product names are trademarks of their respective companies.