Paper 155-2025

Making a Readable PROC COMPARE Report in Excel

Jane Eslinger, Eslinger Enterprises, LLC

ABSTRACT

When using PROC COMPARE to examine two data sets for differences, the default output is a verbose and segmented report optimized for the ODS Listing destination. When sent to Excel, the report—with one variable holding all the information—becomes hard to comprehend. This paper demonstrates how to capture and manipulate key ODS OUTPUT data sets generated by PROC COMPARE. It then shows how to use those data sets with PROC REPORT to create a cleaner, more readable report in Excel perfect for review, documentation, or delivery.

INTRODUCTION

PROC COMPARE output was designed for the text-only ODS Listing destination. The spacing, and therefore readability, is based on a monospace font with a limited line size. Unfortunately, this style doesn't auto-adjust well when exported to Excel. Long strings get truncated with everything crammed into one column, and the result is difficult to parse.

The alternative to the printed report is to use the ODS OUTPUT statement to create data sets containing the results of PROC COMPARE. While it does not reformat the data into a readable report, it does provide data sets with a structure that can be used to build a report. The documentation for PROC COMPARE states:

> These tables do not contain any additional information, and they do not provide a different format for the information.

The text from the printed report is all stored in one variable called “batch” in each of the ODS OUTPUT data sets that PROC COMPARE creates. Each section below contains a figure showing the value of batch.

This paper explores strategies for modifying the ODS OUTPUT data sets generated by PROC COMPARE. The goal is to divide the batch variable into multiple variables that each get their own column in Excel. The paper focuses on sending the results to the ODS destination for Excel for two main reasons: First, that Excel is used by almost everyone who uses SAS, and second, Excel is not a paging destination; with the ability to scroll it, the data is still readable no matter how many variables there are in the data sets.

Please note that PROC COMPARE presents results in seven segments. This paper will focus on five:

- Data Set Summary
- Variables Summary
- Observation Summary
- Values Comparison Summary
- Value Comparison Results

The Table of Summary Statistics is not included in the paper. For it to be present in the PROC COMPARE results, the STATS, ALLSTATS, or PRINTALL option must be used. Since a list of summary statistics is not a common practice, that segment has been excluded from the paper.

The seventh segment is called the Comparison Results for Observations and is only generated when the TRANPOSE option is included in the PROC COMPARE statement. Again, since that is not common practice, that segment has been excluded from the paper.

Only a portion of the code required for manipulating the corresponding ODS OUTPUT data set for the segments of interest are presented in the body of this paper. The full code is in the Appendix.

DATA SETS FOR COMPARISON

The paper uses the data from the SASHELP.SPRINGS data set. The data set is available in SAS OnDemand for Academics but might not be available in other SAS installations. The code was tested with real-world data sets as well. The concepts and code covered in the paper are applicable to any data sets that might need to be compared through PROC COMPARE. Therefore, it is not necessary to have the SASHELP.SPRINGS data set to try out the code.

A work version of the data was created with changes made so that PROC COMPARE captures multiple types of differences.

```
data prod_springs;
    set sashelp.springs;
run;

data val_springs1;
    set sashelp.springs;
    if celsius > 120 then celsius = 120;
    name = tranwrd(name,'Icy', 'Iyc');
    format farenheit 8.1;
    label area = 'Geographical Area';
run;
```

Prod_springs is used as the base data set. Val_springs1 is used as the comparison data set.

PROC COMPARE CODE

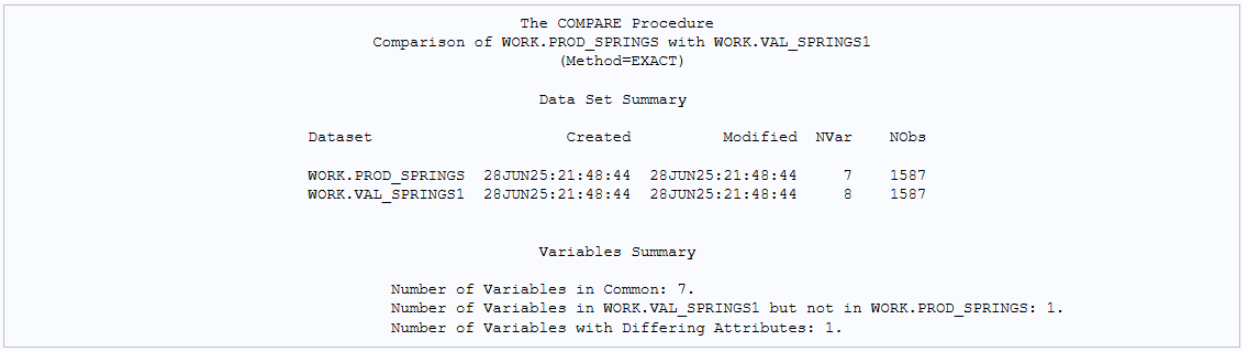
The PROC COMPARE uses an ID statement when ID variables are specified in the appropriate macro parameter. The LISTVAR option is used to create a list of variables found in only one data set. Alternatively, the LISTALL option could be used to create a list of both variables and observations found in one data set. See the documentation for an explanation of each option.

The ODS OUTPUT statement is what drives the code for the rest of the paper. The data sets hold the results of PROC COMPARE.

```
ods exclude all;
ods output
    CompareDatasets            = __data0
    CompareVariables(nowarn)   = __vars0
    CompareSummary             = __summ0
    CompareDifferences(nowarn) = __diff0;
proc compare base=Base compare=Compare listvar criterion=&criterion;
    %if &keys ne %then %do;
          id &keys;
    %end;
run;
ods select all;
```

DATA SET SUMMARY = COMPAREDATASETS DATA SET

The Data Set Summary report provides high-level metadata: dataset names, creation dates, observation counts, and (optionally) labels. It also provides counts on the number of observations that the input data sets have in common. By default, this report segment is the first of the overall report, as seen in Figure 1.



```
                                The COMPARE Procedure
                        Comparison of WORK.PROD_SPRINGS with WORK.VAL_SPRINGS1
                                        (Method=EXACT)

                                      Data Set Summary

 Dataset                      Created          Modified  NVar    NObs

 WORK.PROD_SPRINGS  28JUN25:21:48:44  28JUN25:21:48:44     7    1587
 WORK.VAL_SPRINGS1  28JUN25:21:48:44  28JUN25:21:48:44     8    1587


                                    Variables Summary

          Number of Variables in Common: 7.
          Number of Variables in WORK.VAL_SPRINGS1 but not in WORK.PROD_SPRINGS: 1.
          Number of Variables with Differing Attributes: 1.
```

Figure 1. Observation Summary

The information in this report is captured in the ODS OUTPUT data set called CompareDatasets. Each observation in the report is an observation in the data set, Figure 2. The variable type is created by PROC COMPARE.

type	batch
h	The COMPARE Procedure
h	Comparison of WORK.PROD_SPRINGS with WORK.VAL_SPRINGS1
h	(Method=EXACT)
h	
h	Data Set Summary
h	
h	Dataset Created Modified NVar NObs
d	
d	WORK.PROD_SPRINGS 28JUN25:21:51:13 28JUN25:21:51:13 7 1587
d	WORK.VAL_SPRINGS1 28JUN25:21:51:13 28JUN25:21:51:13 8 1587
d	
d	
h	Variables Summary
h	
d	Number of Variables in Common: 7.
d	Number of Variables in WORK.VAL_SPRINGS1 but not in WORK.PROD_SPRINGS: 1.
d	Number of Variables with Differing Attributes: 1.

Figure 2. CompareDatasets Data

For the first three observations, it makes sense that the procedure information is in one variable. The same is true for the content under the Variables Summary header. However, when sending to a non-Listing destination, it will be easier to read the Data Set Summary section if it is divided into multiple variables.

The number of variables needed for the Data Summary section is based on the input data sets. From one comparison to the next the number of pieces might change. The figures show five pieces: Dataset,

Created, Modified, NVar, and NObs. Neither of the two data sets has a data set label. If they did, label would be the sixth piece of information.

Both the header observation and the two data observations of this section can be parsed with the SCAN function, with a space as the delimiter, except the label. A label might contain spaces and should not be truncated at the first space. I recommend counting the number of pieces in the observation with type=h. If there is a label, then determine the location the text starts at and use SUBSTR instead of SCAN.

```
proc sql noprint;
    select countw(batch) into: data_num trimmed
    from __data0
    where index(batch,'Dataset');
quit;
… code …
array info{*} var1-var&data_num;
if type = 'h' and indexw(batch,'Dataset') then do;
*find starting place of label if it is present;
    %if &data_num = 6 %then %do;
          retain place;
          place = indexc(batch,'L'); ①
    %end;
end;

*separate data into their own variables;
if section = 2 ② then do i=1 to &data_num;
    if i < 6 then info{i} = scan(batch,i,' ');
    else info{i} = substr(batch,place); ③
end;
```

① The values in the header observation are always the same. If the label exists, it is the only value with a capital L.

② Within each segment, section numbers are assigned. Only the second section of this data requires dividing data into variables.

③ Use SUBSTR with the starting location as the second argument to parse the data set label.

The results of the code above produce Figure 3, with each variable in its own Excel column. With the cell alignment, this output looks more in line with the Listing output.

2	Data Set Summary					
3	**Dataset**	**Created**	**Modified**	**NVar**	**NObs**	
4	WORK.BASE	03JUL25:15:19:36	03JUL25:15:19:36	7	1587	
5	WORK.COMPARE	03JUL25:15:19:36	03JUL25:15:19:36	7	1587	
6						
7	Variables Summary					
8	Number of Variables in Common: 7.					
9	Number of Variables with Differing Attributes: 2.					
10						

Figure 3. CompareDatasets Results in Excel

VARIABLES SUMMARY = COMPAREVARIABLES DATA SET

The Variables Summary displays which variables and their metadata differ between the input data sets:

1. A list of variables that are in one of the data sets and not the other when the LISTVAR option is specified. (LISTALL also provides this information.)
2. Common variables with different types.
3. A list of variables that are in both data sets but have differing attributes.

If all variables and attributes match—type, length, label, format, and informat—neither the printed result, nor is the OUTPUT OUT data set created. To avoid warnings in the log if the data set is not created, use the NOWARN option in the OUTPUT OUT statement.

```
ods output CompareVariables(nowarn) = __vars0;
```

Figure 4 shows the printed report. The area's lack of label in WORK.PROD_SPRINGS is noticeable.

Listing of Common Variables with Differing Attributes

Variable	Dataset	Type	Length	Label
Area	WORK.PROD_SPRINGS	Char	14	
	WORK.VAL_SPRINGS1	Char	14	Geographical Area

Figure 4. Variable Attribute Summary

Figure 5 shows the ComareVariable data set. It contains the same information, though it looks strange due to the difference in font.

type	batch
d	
d	
h	Listing of Common Variables with Differing Attributes
d	
h	Variable Dataset Type Length Format Label
h	
d	Area WORK.BASE Char 14
d	WORK.COMPARE Char 14 Geographical Area
d	Farenheit WORK.BASE Num 8
d	WORK.COMPARE Num 8 8.1

Figure 5. CompareVariables Data

There are multiple aspects of the CompareVariables data set that make it hard to manipulate to get everything right. First, the number of columns in the Differing Attributes section can change, depending on the data. For example, if no informats or labels are present, those columns are omitted. Or, if there are no informats, then that column is not included, with the same thing for the label.

The second hurdle is that the system option PAGESIZE determines how many data observations are grouped together. If there are more than will fit on a "page", the header observations are repeated between "pages". For example, if more variables have a difference and pagesize is set to something small, there are multiple groups as shown in Figure 6.

h	Listing of Common Variables with Differing Attributes
d	
h	Variable Dataset Type Length Format Label
h	
d	Longitude WORK.BASE Num 8
d	WORK.COMPARE Num 8 COMMA10.3
h	The COMPARE Procedure
h	Comparison of WORK.BASE with WORK.COMPARE
h	(Method=EXACT)
h	
h	Listing of Common Variables with Differing Attributes
d	
h	Variable Dataset Type Length Format Label
h	
d	Name WORK.BASE Char 34
d	WORK.COMPARE Char 34 Name of Hot Spring

Figure 6. Multiple Groups of Differing Attributes

The third hurdle is that the number of rows displayed together is affected by the amount of data.

The strategy for this ODS OUTPUT data set is to:

1. Count the number of sections
2. Count how many pieces are in each section
3. Determine the starting position of each piece within each section

Starting position is key to manipulating the other ODS OUTPUT data sets discussed in this paper as well.

It might not seem like it when viewing the data set, but each data piece starts at the same location as the header piece. The word “Area” starts at the same position as the word “Variable” and “WORK.BASE” starts as the same position as the word “Dataset.” Note that, “WORK.COMPARE” is lined up with “Dataset” as well. (It is easier to see in Figure 4.)

```
data __vars0a;
    set __vars0;
    where &where1; ①
    retain section 0;
    if index(batch, 'Listing of') then do; ②
          section + 1;
          call symputx('var_sect_lbl' ||strip(put(section,8.)),
            strip(batch)); ③
          delete;
    end;
run;

data _null_;
    set __vars0a;
    if type = 'h' and ^missing(batch); ④
    n+1;
    call symputx('var'||strip(put(n,8.)) ||'_num',countw(batch)); ⑤
```

```
    call symputx('var_sect',n); ⑥
run;
```

① The PROC COMPARE procedure title is removed from all ODS OUTPUT data sets to reduce repetitiveness.

② The values in the header observations all start with the words “Listing of.” Each different “Listing of” value indicates a new section. These results happen to have only one section.

③ Each section has its own label macro variable.

④ Only the number of pieces in the header observations (type=’h’) need to be counted.

⑤ Each section has its own macro variable containing the count of pieces (Variable, Dataset, Type, Length, etc.).

⑥ Create a macro variable with the total number of sections.

The resulting macro variables have these values.

```
VAR_SECT_LBL1=Listing of Common Variables with Differing Attributes
VAR1_NUM=6
VAR_SECT=1
```

After assigning macro variable values, loop through the sections determining the starting positions for each. The length of the variable names, data set names, and labels make the starting positions shift to fit within the line size (system option LINESIZE) from one comparison to the next.

The ANYFIRST function searches a character string for a character that is valid as the first character in a SAS variable name under VALIDVARNAME=V7 and returns the first position at which that character is found. Its second argument, start, specifies the position at which the search should start. In the code, the strategy is to add the length of the header word to its starting position so that ANYFIRST does not keep finding only the first piece.

```
data __positions_vars&i;
    set __vars0a;
    where type = 'h' and ^missing(batch) and section = &i; ①

          array v1{*} vnum1-vnum&&var&i._num;
          do i=1 to &&var&i._num;
                if i=1 then v1{i} = anyfirst(batch); ②
                else v1{i} = anyfirst(batch,v1{i-1}+len1); ③
                len1 = length(scan(batch,i)); ④
          end;

    keep section vnum:;
run;
```

① Only look at one section at a time.

② For the first piece, look for the first character.

③ For all other pieces, start the search after the starting position of the previous piece plus its length.

④ Assign the length of the current piece to the variable called len1.

Remember, if a label is present, it could contain spaces. The SUBSTR function arguments include string, position, and length. For all non-label pieces, all three arguments will be used. For the label, only the string and starting position are needed.

```
%do j = 1 %to &var_sect;
    %do jj = 1 %to &&var&j._num;
          if section = &j and type = 'd' then do;
                if &jj < &&var&j._num then info{&jj} =
                 substr(batch,place{&jj},place{&jj+1}-place{&jj});
                else if &jj = &&var&j._num then info{&jj} =
                 substr(batch,place{&jj});
          end;
    %end;
    if section = &j then head2 = "&&var_sect_lbl&j.";

    %if &j > 1 %then %do;
          %let jjj = %eval(&j-1);
          if section = &j and bold='Y' and
           head2 = "&&var_sect_lbl&jjj" then delete; ①
    %end;
%end;
```

① If the section has multiple groupings, delete all but the first header.

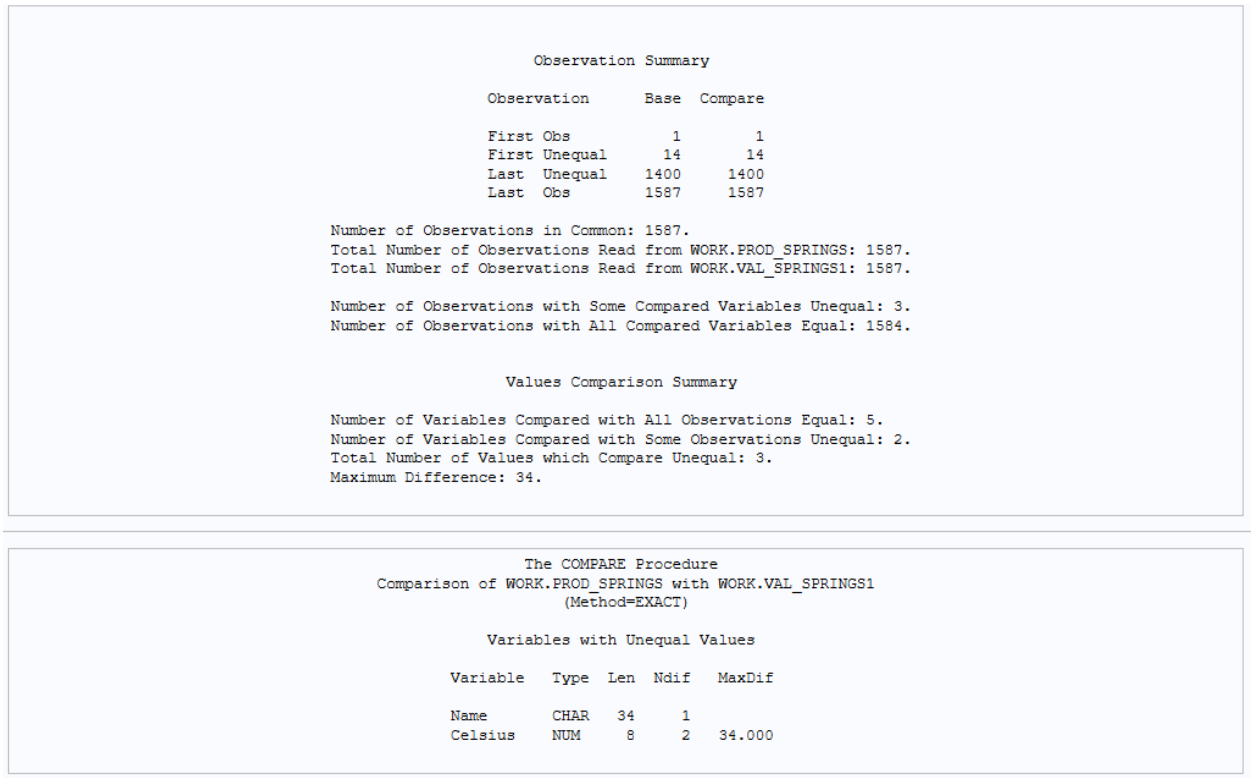
The results of the code above produce Figure 7.

11	Listing of Common Variables with Differing Attributes					
12	**Variable**	**Dataset**	**Type**	**Length**	**Format**	**Label**
13	Area	WORK.BASE	Char	14		
14		WORK.COMPARE	Char	14		Geographical Area
15	Farenheit	WORK.BASE	Num	8		
16		WORK.COMPARE	Num	8	8.1	
17						

Figure 7. CompareVariables Results in Excel

OBSERVATION SUMMARY = COMPARESUMMARY DATA SET

PROC COMPARE documentation describes the Observation Summary and Variables Comparison Summary as two separate reports, as seen in Figure 8. The Variables Comparison Summary may also include a section for Variables with Unequal Values, if applicable. However, as shown in Figure 9, all this information is captured in the CompareSummary output.



Observation Summary

Observation	Base	Compare
First Obs	1	1
First Unequal	14	14
Last Unequal	1400	1400
Last Obs	1587	1587

Number of Observations in Common: 1587.
Total Number of Observations Read from WORK.PROD_SPRINGS: 1587.
Total Number of Observations Read from WORK.VAL_SPRINGS1: 1587.

Number of Observations with Some Compared Variables Unequal: 3.
Number of Observations with All Compared Variables Equal: 1584.

Values Comparison Summary

Number of Variables Compared with All Observations Equal: 5.
Number of Variables Compared with Some Observations Unequal: 2.
Total Number of Values which Compare Unequal: 3.
Maximum Difference: 34.

The COMPARE Procedure
Comparison of WORK.PROD_SPRINGS with WORK.VAL_SPRINGS1
(Method=EXACT)

Variables with Unequal Values

Variable	Type	Len	Ndif	MaxDif
Name	CHAR	34	1	
Celsius	NUM	8	2	34.000

Figure 8. Observation Summary

type	batch
h	Observation Summary
h	
h	Observation Base Compare
d	
d	First Obs 1 1
d	First Unequal 14 14
d	Last Unequal 1400 1400
d	Last Obs 1587 1587
d	
d	Number of Observations in Common: 1587.
d	Total Number of Observations Read from WORK.BASE: 1587.
d	Total Number of Observations Read from WORK.COMPARE: 1587.
d	
d	Number of Observations with Some Compared Variables Unequal: 3.
d	Number of Observations with All Compared Variables Equal: 1584.
d	
d	
h	Values Comparison Summary
h	
d	Number of Variables Compared with All Observations Equal: 5.
d	Number of Variables Compared with Some Observations Unequal: 2.
d	Total Number of Values which Compare Unequal: 3.
d	Maximum Difference Criterion Value: 0.24818.
d	
h	The COMPARE Procedure
h	Comparison of WORK.BASE with WORK.COMPARE
h	(Method=RELATIVE(0.0000222), Criterion=1.0E-09)
h	
h	Variables with Unequal Values
h	
h	Variable Type Len Ndif MaxCrit
d	
d	Name CHAR 34 1
d	Celsius NUM 8 2 0.248

Figure 9. CompareSummary Data

The CompareSummary data set, shown in Figure 9, shows three visible sections. For the purposes of this paper, it will be broken into four sections. The Observation summary section will be divided into two: one

with the information about the first and last observations and one containing the five observations of text about the Number of Observations.

1. Observation summary – first and last observation numbers. First Unequal and Last Unequal will not be shown when all observations match.
2. Observation summary – five rows about the number of observations.
3. Values Comparison Summary.
4. Variables with Unequal Values (if present).

The information in the Number of Observations section and the Values Comparison sections will be assigned to a single variable, as they are just text. For the other sections, the starting position must be determined. There can be up to eight pieces in the Variables with Unequal Values section.

The Observation Summary section requires extra coding before the starting positions can be found. When the comparison includes an ID statement, the values of the ID variables are included after the Compare observation number. When those values are longer than the line size they wrap to the next observation. The code must first detect and account for wrapping before parsing can proceed. Figure 10 provides an example of what the data set might look like if the ID values wrap.

h		Observation	Base	Compare	ID
d					
d		First Obs	1	1	idvar1=this idvar has some text in it
d					idvar2=this idvar always has some text in it idvar3=will it wrap?
d		First Unequal	14	14	idvar1=this idvar has some text in it
d					idvar2=this idvar always has some text in it idvar3=will it wrap?

Figure 10. Example of an Observation Summary that wraps

```
data _null_;
    set __summ0a end=eof;

          retain firstn lastn ;
          if indexw(batch,'First Obs') then do;
                firstn = _n_;
                call symputx('firstobs',_n_);
          end;
          if indexw(batch,'First Unequal') then firstn = _n_ - firstn;
          if index(batch,'Last  Unequal') then lastn = _n_;
          if index(batch,'Last  Obs') then do;
                lastn = _n_ - lastn;
                call symputx('lastobs',_n_);
          end;

          if eof then do;
                if firstn > 1 or lastn > 1 then
                   call symputx('multiobservation','Y'); ①
                else call symputx('multiobservation','N');
          end;
run;
```

① The difference in observation number (_n_) is greater than one if the ID values wrapped.

If wrapping occurred, the observations are then consolidated into one observation.

```
data __multi;
    set __summ0a(rename=(batch=orig));
    length dummy batch $500;

    if _n_ < &firstobs then do; batch = orig; output; end;
    if &firstobs <= _n_ <= &lastobs + 1;

          dummy = lag(orig);
          if index(dummy,'First Obs') and ^index(orig,'First Unequal')
          then do; batch = catt(dummy, orig); output; end;
          if index(dummy,'First Unequal') and ^index(orig,'Last  Unequal')
          then do; batch = catt(dummy, orig); output; end;
          else if index(dummy,'Last  Unequal') and ^index(orig,'Last  Obs')
          then do; batch = catt(dummy, orig); output; end;
          else if index(dummy,'Last  Obs') and ^missing(orig)
          then do; batch = catt(dummy, orig); output; end;
    keep type batch;
run;
```

Determining the starting position for the CompareSummary data set is the same as for the CompareVariables data set, with one exception. The Variables with Unequal Values section displays the label when there is one. If the labels do not match between the two data sets, the label from the Compare data is labeled ‘Compare Label’. To avoid miscounting the number of pieces, it is best to change the space to something else, like an underscore.

```
batch = tranwrd(batch,'Compare Label','Compare_Label');
```

The code for splitting the batch variable into multiple variables is also the same as was shown for CompareVariables.

Figure 11 shows the results in Excel. Each of the sections created within the data set have their own heading. All sections are separated by a blank line.

18	Observation Summary					
19	**Observation**	**Base**	**Compare**			
20	First Obs	1	1			
21	First Unequal	14	14			
22	Last Unequal	1400	1400			
23	Last Obs	1587	1587			
24						
25	Observation Number Summary					
26	Number of Observations in Common: 1587.					
27	Total Number of Observations Read from WORK.BASE: 1587.					
28	Total Number of Observations Read from WORK.COMPARE: 1587.					
29	Number of Observations with Some Compared Variables Unequal: 3.					
30	Number of Observations with All Compared Variables Equal: 1584.					
31						
32	Values Comparison Summary					
33	Number of Variables Compared with All Observations Equal: 5.					
34	Number of Variables Compared with Some Observations Unequal: 2.					
35	Total Number of Values which Compare Unequal: 3.					
36	Maximum Difference Criterion Value: 0.24818.					
37						
38	Variables with Unequal Values					
39	**Variable**	**Type**	**Len**	**Ndif**	**MaxCrit**	
40	Name	CHAR	34	1		
41	Celsius	NUM	8	2	0.248	
42						

Figure 11. CompareSummary Results in Excel

VALUE COMPARISON = COMPAREDIFFERENCES DATA SET

The final segment is the Value Comparison. As with the previous segments, this one has its own set of challenges.

- The printed output and the CompareDifferences Output data set include observations filled with underscores (_).
- The vertical bars (|) used for alignment are also captured in the data set
- The header before the vertical bars for a comparison without ID variables is ‘Obs’, shown in Figure 12. A comparison with an ID statement has ID variable names before the vertical bars, shown in Figure 13.
- The pieces that are displayed differ between character variables and numeric variables.
- Some headers contain multiple words delimited by spaces and other default delimiters (“Base Value” and “% Diff”), which complicates parsing.
- The SCAN function does not consider a vertical bar as a word, so additional code is needed to capture it properly.

Figure 14 shows the same output captured in a data set.

Value Comparison Results for Variables

Obs	Base Value Name	Compare Value Name
88	Hot Springs Near Icy	Hot Springs Near Iyc

Obs	Base Celsius	Compare Celsius	Diff.	% Diff
14	154.0000	120.0000	-34.0000	-22.0779
1400	130.0000	120.0000	-10.0000	-7.6923

Figure 22. Value Comparison without ID Statement

Value Comparison Results for Variables

Type	Base Value Name	Compare Value Name
Hotspring	Hot Springs Near Icy	Hot Springs Near Iyc

Type	Base Celsius	Compare Celsius	Diff.	% Diff
Hotspring	154.0000	120.0000	-34.0000	-22.0779
Hotspring	130.0000	120.0000	-10.0000	-7.6923

Figure 33. Value Comparison with ID Statement

type	batch
d	
d	
h	Value Comparison Results for Variables
h	
d	_____
d	|| Base Value Compare Value
h	Obs || Name Name
d	_____ || _____+ _____+
d	||
d	88 || Hot Springs Near Icy Hot Springs Near Iyc
d	_____
d	
d	
d	_____
d	|| Base Compare
h	Obs || Celsius Celsius Diff. % Diff
d	_____ || _____ _____ _____ _____
d	||
d	14 || 154.0000 120.0000 -34.0000 -22.0779
d	1400 || 130.0000 120.0000 -10.0000 -7.6923
d	_____

Figure 44. CompareDifferences Data

As with the other segments, the first task is to count the number of sections. Notice in Figure 14 how the header observations are different based on the variable type. Each header observation that is different than the previous one needs to be treated as its own section.

```
data __diff0a;
    set __diff0;
    where &where1;
    if strip(batch) = 'Value Comparison Results for Variables' then delete;

    length dummy $500;
    dummy = lag(batch);

    retain section 0;
    if countc(batch,'_') > 20 and dummy='' then do; ①
          section + 1;
    end;
    drop dummy;
run;
```

① The type = ‘h’ observations cannot be used to count sections because the data observation above it would be left out. Instead, base the sections on the top observation of underscores.

As mentioned previously, counting the pieces in each section can be challenging. In the code below, those are mitigated by changing the batch value.

```
data _null_;
    set __diff0a;
    if type = 'h' and ^missing(batch);
          batch = tranwrd(batch,'% Diff','%_Diff');
          batch = tranwrd(batch,'Diff.','Diff_');
          batch = tranwrd(batch,'Base Value','Base_Value');
          batch = tranwrd(batch,'Compare Value','Compare_Value');

          n+1;
          *add 1 for ||;
          call symputx('dif'||strip(put(n,8.))||'_num',
            sum(countw(batch),1));
          call symputx('diff_sect',n);
run;
```

Determining the starting position for the CompareDifferences data set has its own set of challenges. Though the header observations were used in the previous segments to determine starting position, PROC COMPARE has inconsistent spacing on those observations when it mixes character and numeric. Therefore, the data observation with underscores and vertical bars is used to find the positions. Then, because SCAN ignores the vertical bar, its position must be determined separately. All positions must be reordered so that the vertical bar ends up in the appropriate variable.

```
data __positions_diff&i;
    set __diff0a;
    *using the underscore observations because proc compare
     has inconsistent spacing when it mixes character and
     numeric;
    where index(batch,'_') and index(batch,'||') and section = &i;①

          array v1{*} vnum1-vnum&&dif&i._num;
          do i=1 to &&dif&i._num;
                if i=1 then v1{i} = anyfirst(batch);
                else v1{i} = anyfirst(batch,v1{i-1}+len1);
                len1 = length(scan(batch,i));
                if i = &&dif&i._num then v1{i} = indexc(batch,'|');②
          end;

          call sortn(of v1{*});③

    keep section vnum:;
run;
```

① Choose the observation with consistent positioning.

② Find the starting position for the vertical bars.

③ Sort the values of the starting positions so they are in numerical order.

The code for splitting the batch variable into multiple variables is the same as the previous two segments. The results are shown in Figure 15.

43	Value Comparison Results for Variables					
44						
45		||	Base Value	Compare Value		
46	**Obs**	||	**Name**	**Name**		
47						
48	88	||	Hot Springs Near Icy	Hot Springs Near Iyc		
49						
50						
51		||	Base	Compare		
52	**Obs**	||	**Celsius**	**Celsius**	**Diff.**	**% Diff**
53						
54	14	||	154	120	-34	-22.0779
55	1400	||	130	120	-10	-7.6923
56						

Figure 55. CompareDifferences Results in Excel

Although the underscore rows can be removed, I chose to retain them to help the final Excel output resemble the printed PROC COMPARE results. With additional logic, these could be filtered out if preferred.

CONCLUSION

This paper demonstrated how to transform PROC COMPARE output, which is originally designed for monospace fonts, fixed line sizes, and printed reports, into a format that's easy to read and work with in Excel. Using the ODS OUTPUT tables from five key segments—Data Set Summary, Variables Summary, Observation Summary, Values Comparison Summary, and Value Comparison Results—the code systematically parsed and converted the content into structured variables. Although each segment required tailored logic due to formatting quirks, the final result is a well-organized Excel output, ideal for review, documentation, and delivery.

REFERENCES

SAS Institute Inc. 2025. COMPARE Procedure. SAS® Viya® Platform Programming Documentation. Cary, NC: SAS Institute Inc. Available at https://documentation.sas.com/doc/en/pgmsascdc/v_064/proc/n1nwxbchh5hpu1n1h28kmici2awd.htm.

SAS Institute Inc. 2025. SAS® Functions and CALL Routines: Reference. SAS® Viya® Platform Programming Documentation. Cary, NC: SAS Institute Inc. Available at https://documentation.sas.com/doc/en/pgmsascdc/v_064/lefunctionsref/titlepage.htm.

ACKNOWLEDGMENTS

The author is grateful to Josh Horstman for the challenge of making PROC COMPARE output pretty in Excel. I would also like to thank Inka Leprince and Richann Watson for reviewing the paper and Lauree Shepard for editing it.

CONTACT INFORMATION

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

Jane Eslinger
Eslinger Enterprises, LLC
eslingerent@gmail.com

APPENDIX

```
/************************************************************************
* Program: procompare_awesome
* Author: Jane Eslinger
* Date: 23Jun2025
*
* Purpose: To output Proc Compare results to Excel in a readable format
*
* Inputs: Base data set
*         Compare data set
*
* Output: &filepath.\&filename..xlsx with two tabs
*
* Revisions:
************************************************************************/

/*
REQUIRED Macro Parameters:
     basedr - libref for the base/production/main data set
     compdr - libref for the compare/validation/qc data set
     baseds - name of the base/production/main data set
     compds - name of the compare/validation/qc data set
OPTIONAL Macro Parameters:
     keys - variables to use in the ID statement in Proc Compare
     sortby - variables to sort the data sets by before Proc Compare
     filepath - the folder path for saving Excel
     filename - the name of the Excel file to create
     criterion - criterion for numeric variable comparison
     baselbl - label for the base/production/main data set in Differences tab
     complbl - label for the compare/validation/qc data set in Differences tab
*/

%macro pcompout(
     basedr = ,
     compdr = ,
     baseds = ,
     compds = ,
     keys = ,
     sortby = ,
     filepath = ,
     filename = ProcCompareOutput,
     criterion = 0.000000001,
     baselbl = Production,
     complbl = Validation,
);

************************************************************************
*             CHECK FOR REQUIRED MACRO PARAMETERS                      *
************************************************************************;

     %if %length(&basedr) = 0 %then %do;
          %put %sysfunc(compress(ERR OR:)) Parameter BASEDR Base libref not provided;
          %abort;
     %end;
     %if %length(&compdr) = 0 %then %do;
          %put %sysfunc(compress(ERR OR:)) Parameter COMPDR Comparison libref not provided;
          %abort;
     %end;
     %if %length(&baseds) = 0 %then %do;
          %put %sysfunc(compress(ERR OR:)) Parameter BASEDS Base data set not provided;
          %abort;
     %end;
     %if %length(&compds) = 0 %then %do;
          %put %sysfunc(compress(ERR OR:)) Parameter COMPDS Comparison data set not provided;
          %abort;
     %end;

************************************************************************
*                              SET UP                                  *
************************************************************************;

     filename temp "&filepath.\&filename..xlsx";
```

```
    data _null_;
           if fexist('temp') then rc = fdelete('temp');
    run;
    filename temp clear;

    %if &sortby ne %then %do;
           proc sort data=&basedr..&baseds out=base;
                  by &sortby;
           run;

           proc sort data=&compdr..&compds out=compare;
                  by &sortby;
           run;
    %end;
    %else %do;
           data base;
                  set &basedr..&baseds;
           run;

           data compare;
                  set &compdr..&compds;
           run;
    %end;

*************************************************************************
*                              SUMMARY TAB                              *
*************************************************************************;

    ods exclude all;
    ods output CompareDatasets    = __data0
           CompareVariables(nowarn)   = __vars0
           CompareSummary     = __summ0
           CompareDifferences(nowarn) = __diff0;
    proc compare base=Base compare=Compare listvar criterion=&criterion;
           %if &keys ne %then %do;
                  id &keys;
           %end;
    run;
    ods select all;

    %let where1 = %str(^index(batch,'Procedure') and ^index(batch,'Comparison of') and
^index(batch,'Method'));
    %let where2 = %str(index(batch,'Procedure') or index(batch,'Comparison') or
index(batch,'Method'));

    /***************** TITLE INFO *****************/
    data __head;
           set __data0 ;
           where &where2;
           length head1 $1000;

                  retain head1 ;
                  head1 = catx(' (*ESC*)n',head1,batch);

                  head1 = tranwrd(head1,'WORK.BASE',upcase("&basedr..&baseds"));
                  head1 = tranwrd(head1,'WORK.COMPARE',upcase("&compdr..&compds"));

           if _n_ = 3;
           keep head1;
    run;

    /***************** COMPAREDATASETS *****************/
    *# of pieces in __data0;
    proc sql noprint;
           select countw(batch) into: data_num trimmed
           from __data0
           where index(batch,'Dataset');
    quit;

    *create usuable/printable variables;
    data __data1;
```

```
	set __data0;
	length head2 var1-var&data_num $500;

		retain place 0;
		array info{*} var1-var&data_num;

		*change these headers to a data rows;
		if type = 'h' and indexw(batch,'Dataset') then do;
			type = 'd';
			bold = 'Y';
			*find starting place of label if it is present;
			%if &data_num = 6 %then %do;
				retain place;
				place = indexc(batch,'L');
			%end;
		end;

		*set up section numbers for proc report grouping variable;
		retain section 0;
		lag_type = lag(type);
		if _n_ <= 3 then section = 1;
		if _n_ > 3 and lag_type ^= type then section = section + 1;

		*separate data into their own variables;
		if section = 2 and ^missing(batch) then do i=1 to &data_num;
			if i < 6 then info{i} = scan(batch,i,' ');
			else info{i} = substr(batch,place);
		end;
		if section = 4 then var1 = strip(batch);

		*create grouping variable;
		retain head2;
		if compbl(strip(batch)) in ('Data Set Summary' 'Variables Summary')
		 then head2 = compbl(strip(batch));
		grp = 1;

	if section in (2 4);
	keep head2 var: section grp bold;
run;

/***************** COMPAREVARIABLES *****************/
%if %sysfunc(exist(__vars0)) %then %do; *if no attrib differnces data set not created;

	data __vars0a;
		set __vars0;
		where &where1;
		retain section 0;
		if index(batch, 'Listing of') then do;
			section + 1;
			call symputx('var_sect_lbl'||strip(put(section,8.)),strip(batch));
			delete;
		end;
	run;

	*determine how many sections there are and how many pieces are in each section;
	data _null_;
		set __vars0a;
		if type = 'h' and ^missing(batch);
			n+1;
			call symputx('var'||strip(put(n,8.))||'_num',countw(batch));
			call symputx('var_sect',n);
	run;

	%let maxvar = 0;
	%do i=1 %to &var_sect;
		*starting positions;
		data __positions_vars&i;
			set __vars0a;
			where type = 'h' and ^missing(batch) and section = &i;

				array v1{*} vnum1-vnum&&var&i._num;
```

```
                                do i=1 to &&var&i._num;
                                        if i=1 then v1{i} = anyfirst(batch);
                                        else v1{i} = anyfirst(batch,v1{i-1}+len1);
                                        len1 = length(scan(batch,i));
                                end;

                        keep section vnum:;
                run;

                %if &i=1 %then %do;
                        data __positions_vars;
                                set __positions_vars&i;
                        run;
                %end;
                %else %do;
                        data __positions_vars;
                                set __positions_vars __positions_vars&i;
                        run;
                %end;

                %let maxvar = %sysfunc(max(&maxvar, &&var&i._num));
        %end;

        *create usuable/printable variables;
        data __vars1;
                merge __vars0a __positions_vars;
                by section;
                if &where1;

                length head2 var1-var&maxvar $500;

                        array info{*} var1-var&maxvar;
                        array place{*} vnum1-vnum&maxvar;

                        if type = 'h' and index(batch,'Variable') then do;
                                type = 'd';
                                bold = 'Y';
                        end;

                        *separate data into their own variables;
                        %do j = 1 %to &var_sect;
                                %do jj = 1 %to &&var&j._num;
                                        if section = &j and type = 'd' then do;
                                                if &jj < &&var&j._num then info{&jj} =
                                                 substr(batch,place{&jj},place{&jj+1}-
place{&jj});
                                                else if &jj = &&var&j._num then info{&jj} =
                                                 substr(batch,place{&jj});
                                        end;
                                %end;
                                if section = &j then head2 = "&&var_sect_lbl&j.";

                                %if &j > 1 %then %do;
                                        %let jjj = %eval(&j-1);
                                        if section = &j and bold='Y' and
                                         head2 = "&&var_sect_lbl&jjj" then delete;
                                %end;
                        %end;


                        *create grouping variable;
                        grp = 2;

                keep head2 var: section grp bold;
        run;
    %end;
    %else %do;
        data __vars1;
                length head2 var1 $500;
                        head2 = 'Listing of Common Variables with Differing Attributes';
                        grp = 2;
```

```
                        var1 = 'No Differing Attributes';
        run;
        %let maxvar = 0;
%end;

/***************** COMPARESUMMARY *****************/
data __summ0a;
        set __summ0;
        where &where1;
run;

*determine if proc compare wrapped long first/last obs;
data _null_;
        set __summ0a end=eof;

                retain firstn lastn ;
                if indexw(batch,'First Obs') then do;
                        firstn = _n_;
                        call symputx('firstobs',_n_);
                end;
                if indexw(batch,'First Unequal') then firstn = _n_ - firstn;
                if index(batch,'Last  Unequal') then lastn = _n_;
                if index(batch,'Last  Obs') then do;
                        lastn = _n_ - lastn;
                        call symputx('lastobs',_n_);
                end;

                if eof then do;
                        if firstn > 1 or lastn > 1 then call symputx('multirow','Y');
                        else call symputx('multirow','N');
                end;
run;

%if &multirow = Y and &keys ^= %then %do;
        *change wrapped rows to one row;
        data __multi;
                set __summ0a(rename=(batch=orig));
                length dummy batch $500;

                if _n_ < &firstobs then do; batch = orig; output; end;
                if &firstobs <= _n_ <= &lastobs + 1;

                        dummy = lag(orig);
                        if index(dummy,'First Obs') and ^index(orig,'First Unequal') then
do; batch = catt(dummy, orig); output; end;
                        if index(dummy,'First Unequal') and ^index(orig,'Last  Unequal')
then  do; batch = catt(dummy, orig); output; end;
                        else if index(dummy,'Last  Unequal') and ^index(orig,'Last  Obs')
then do; batch = catt(dummy, orig); output; end;
                        else if index(dummy,'Last  Obs') and ^missing(orig) then do; batch
= catt(dummy, orig); output; end;
                keep type batch;
        run;

        data __other;
                set __summ0a;
                if _n_ > &lastobs + 1;
        run;

        *recombine;
        data __summ0b;
                set __multi __other;
        run;
%end;
%else %if &multirow = N or &keys = %then %do;
        data __summ0b;
                set __summ0a;
        run;
%end;

*find new last obs row;
```

```
data _null_;
       set __summ0b;
              if index(batch,'Last  Obs') then call symputx('lastobs',_n_);
run;

*artifically create non-missing header rows for Number Summary
 and Values Comparison Summary;
data __summ0c;
       set __summ0b;

              retain whr 0;
              if strip(batch) = 'Values Comparison Summary' then whr = _n_;
              if _n_ = whr + 1 then batch = 'a';

              output;
              if (_n_ = &lastobs + 1 and missing(batch)) or
                (_n_ = &lastobs + 2 and missing(batch)) then do;
                     type = 'h';
                     batch = 'Observation Number Summary';
                     output;
                     type = 'h';
                     batch = 'a';
                     output;
              end;
       drop whr;
run;

*determine how many sections there are;
data __summ0d;
       set __summ0c;
       where &where1;
              retain section 0;
              if type = 'h' and (index(batch, 'Summary') or index(batch,'Unequal')) then
do;
                     section + 1;
                     call symputx('sum_sect_lbl'||strip(put(section,8.)),strip(batch));
                     delete;
              end;
run;

*determine how many pieces are in each section;
data _null_;
       set __summ0d;
       if type = 'h' and ^missing(batch);

              batch = tranwrd(batch,'Compare Label','Compare_Label');
              call symputx('sum'||strip(put(section,8.))||'_num',countw(batch));
              call symputx('sum_sect',section);
run;

%let maxsum = 0;
%do i=1 %to &sum_sect;
       *starting positions;
       data __positions_summ&i;
              set __summ0d;
              where type = 'h' and ^missing(batch) and section = &i;

                     batch = tranwrd(batch,'Compare Label','Compare_Label');

                     array v1{*} vnum1-vnum&&sum&i._num;
                     len1 = length(scan(batch,1));
                     do i=1 to &&sum&i._num;
                            if i=1 then v1{i} = anyfirst(batch);
                            else v1{i} = anyfirst(batch,v1{i-1}+len1);
                            len1 = length(scan(batch,i));
                     end;

              keep section vnum:;
       run;

       %if &i=1 %then %do;
```

```
                data __positions_summ;
                        set __positions_summ&i;
                run;
        %end;
        %else %do;
                data __positions_summ;
                        set __positions_summ __positions_summ&i;
                run;
        %end;

        %let maxsum = %sysfunc(max(&maxsum, &&sum&i._num));
    %end;

    data __summ1;
        merge __summ0d(where=(batch^='a'))
              __positions_summ;
        by section;
        length head2 var1-var&maxsum $500;

                array info{*} var1-var&maxsum;
                array place{*} vnum1-vnum&maxsum;

                *change these headers to a data rows;
                if (indexw(batch,'Observation') and indexw(batch,'Base') and
index(batch,'Compare'))
                        or (indexw(batch,'Variable') and indexw(batch,'Type') and
index(batch,'Ndif'))
                        then do; type = 'd'; bold = 'Y'; end;

                *separate data into their own variables;
                %do j=1 %to &sum_sect;
                        if section = &j and type = 'd' then do i=1 to &&sum&j._num;
                                head2 = "&&sum_sect_lbl&j";
                                if i < &&sum&j._num then info{i} =
substr(batch,place{i},place{i+1}-place{i});
                                else if i = &&sum&j._num then info{i} =
substr(batch,place{i});
                        end;
                        %if &j > 3 %then %do;
                                %let jjj = %eval(&j-1);
                                if section > 4 and bold='Y' and
                                 head2 = "&&sum_sect_lbl&jjj" then delete;
                        %end;
                %end;

                *create grouping variable;
                grp = 3;

        keep head: var: section grp bold;
    run;

    /***************** COMPAREDIFFERENCES *****************/
    %if %sysfunc(exist(__diff0)) %then %do; *if no differnces data set not created;

        *determine how many sections there are;
        data __diff0a;
                set __diff0;
                where &where1;
                if strip(batch) = 'Value Comparison Results for Variables' then delete;

                length dummy $500;
                dummy = lag(batch);

                retain section 0;
                if countc(batch,'_') > 20 and dummy='' then do;
                        section + 1;
                end;
                drop dummy;
        run;

        *determine how many pieces are in each section;
```

```
data _null_;
	set __diff0a;
	if type = 'h' and ^missing(batch);
		batch = tranwrd(batch,'% Diff','%_Diff');
		batch = tranwrd(batch,'Diff.','Diff_');
		batch = tranwrd(batch,'Base Value','Base_Value');
		batch = tranwrd(batch,'Compare Value','Compare_Value');

		n+1;
		*add 1 for ||;
		call symputx('dif'||strip(put(n,8.))||'_num',sum(countw(batch),1));
		call symputx('diff_sect',n);
run;

%let maxdif = 0;
%do i=1 %to &diff_sect;
	*starting positions;
	data __positions_diff&i;
		set __diff0a;
		*using the underscore observations because proc compare
		 has inconsistent spacing when it mixes character and
		 numeric;
		where index(batch,'_') and index(batch,'||') and section = &i;

			array v1{*} vnum1-vnum&&dif&i._num;
			do i=1 to &&dif&i._num;
				if i=1 then v1{i} = anyfirst(batch);
				else v1{i} = anyfirst(batch,v1{i-1}+len1);
				len1 = length(scan(batch,i));
				if i = &&dif&i._num then v1{i} = indexc(batch,'|');
			end;

			call sortn(of v1{*});

		keep section vnum:;
	run;

	%if &i=1 %then %do;
		data __positions_diff;
			set __positions_diff&i;
		run;
	%end;
	%else %do;
		data __positions_diff;
			set __positions_diff __positions_diff&i;
		run;
	%end;

	%let maxdif = %sysfunc(max(&maxdif, &&dif&i._num));
%end;

*create usuable/printable variables;
data __diff1;
	merge __diff0a __positions_diff;
	by section;

	length head2 var1-var&maxdif $500;

		if countc(batch,'_') > 20 then type = 'x';

		*fixes due to inconsistent spacing by proc compare;
		batch = tranwrd(batch,'Compare Value',' Compare Value');

		array info{*} var1-var&maxdif;
		array place{*} vnum1-vnum&maxdif;

		*separate data into their own variables;
		%do j=1 %to &diff_sect;
			if section = &j then do i=1 to &&dif&j._num;
				if type = 'x' then info{i} = repeat('_',9);
```

```
                                        else if i < &&dif&j._num then info{i} =
substr(batch,place{i},place{i+1}-place{i});
                                        else if i = &&dif&j._num then info{i} =
substr(batch,place{i});
                                end;
                        %end;

                        *create grouping variable;
                        retain head2;
                        head2 = 'Value Comparison Results for Variables';
                        grp = 4;
                        if type = 'h' then bold = 'Y';

                if strip(batch) = '||' then delete;
                if _n_ > 3;
                keep head: var: section grp bold;
        run;
    %end;
    %else %do;
        data __diff1;
                length head2 var1 $500;
                        head2 = 'Value Comparison Results for Variables';
                        grp = 4;
                        var1 = 'No Differing Values';
        run;
        %let maxdif = 0;
    %end;
    %let maxall = %sysfunc(max(&maxvar, &maxsum, &data_num, &maxdif, 6));
    data tab_sum;
        set __data1 __vars1 __summ1 __diff1;
        if _n_ = 1 then set __head;
                array v{*} var1-var&maxall;
                do i=1 to dim(v);
                        v{i} = strip(v{i});
                end;

        if missing(var1) and missing(var2) then delete;
    run;


*************************************************************************
*                     SEND REPORT TO EXCEL                              *
*************************************************************************;
    ods excel file= "&filepath.\&filename..xlsx" options(sheet_name='Summary');

    *summary tab;
    proc report data=tab_sum noheader;
        column grp head1 head2 var1-var&maxall bold;
        define grp / order noprint;
        define head1 / order noprint;
        define head2 / order noprint order=data;
        define var1 / style(column)=[width=3.5in];
        define bold / noprint;

        compute before head1;
                if grp = 1 then num = 1000;
                else num = 0;
                line head1 $varying. num;
        endcomp;
        compute before head2;
                line head2 $1000.;
        endcomp;
        compute after head2;
                line ' ';
        endcomp;
        compute bold;
                if bold = 'Y' then call define(_row_,'style','style=[font_weight=bold]');
        endcomp;

    run;
```

```
ods excel close;

************************************************************************
*                           CLEAN UP                                   *
************************************************************************;
     /*proc datasets lib=work noprint;
            delete __:;
     quit;*/

%mend pcompout;
```