Paper BB-168

Don't Forget About Small Data

Lisa Eckler, Lisa Eckler Consulting Inc., Toronto, ON

ABSTRACT

Beginning in the world of data analytics and eventually flowing into mainstream media, we are seeing a lot about Big Data and how it can influence our work and our lives. Through examples, this paper will explore how Small Data -- which is everything Big Data is not -- can and should influence our programming efforts. The ease with which we can read and manipulate data from different formats into usable tables in SAS® makes using data to manage data very simple and supports healthy and efficient practices. This paper will explore how using small or summarized data can help to organize and track program development, simplify coding and optimize code.

INTRODUCTION

Before we look at how and why to use Small Data, let's define what we mean by "Small Data" and consider where to look for it.

There are many different definitions of Big Data with varying criteria, but most agree that Big Data is

- high in volume
- rapidly growing
- usually in need of advanced analytics to wrangle into meaningful condition.

By contrast, Small Data is typically

- compact
- low in volume (either naturally or as a summary of higher-volume detailed data)
- highly structured
- relatively static
- quick and easy to interpret and make use of.

These properties make Small Data intense and highly concentrated, so a little of it can go a long way. It's really "energy efficient" because it is so convenient to work with, saves our personal time and energy and doesn't get consumed in the process. Small Data is truly the antithesis of Big Data. It is everything Big Data is not, which is not to say that everything that is not Big Data is Small Data! In between lies much of the data we work with on a regular basis.

After discussing the strengths of Small Data, where to look for it and how to leverage it, we'll look at some example situations and code to highlight the thought process. Some familiarity with Base SAS procedures is assumed. There is nothing unique about the code included or discussed in this paper. There are some excellent papers written by members of the SAS user community that detail how to use the procedures and techniques touched on here, so we won't focus on how to use them but rather on why to use them. What is novel here is the approach to using these various techniques and available data as tools for our development process rather than as a permanent part of the program. You may already be using some of these techniques to simplify your work. Some are widely employed; others less common. Here, we

encourage programmers to look for more opportunities to make use of Small Data techniques to make development work easier.

DISCUSSION

Depending on what data is available, how much or how little of it there is and how complex the program or process we're building is, we can use Small Data to help

- plan and outline programs
- generate code while minimizing opportunities to introduce errors
- optimize code by treating the most likely values first
- select efficient and effective test cases
- validate programs
- supply definitions to document our results
- control programs or processes.

This list may seem fantastic because it touches almost every aspect of program design, development and testing, but the techniques described in this paper *can* assist with these activities and we might have very easy access to this assistance.

Small Data techniques are not meant to be part of the end product of our work, although there may be good reason to build some summarization of data or review of metadata into our ongoing processes as a reality check. More often, though, these techniques involve using "throw-away" code to make our coding efforts quicker, less tedious and less error prone, in addition to making the final program easier to read and understand and more efficient to run. The techniques applied will be invisible or barely visible once development is completed. It may be counter-intuitive to suggest that writing code that won't be in the end program is a good use of our time, but despite the code being mostly "throw-away" for the final program we're writing, it isn't really single-use. A virtual toolbox full of these tools, especially using macro variables, can be used again and again for other development work.

Where to find Small Data

At the beginning of a development effort, it is useful to pause and consider what things we already have and what additional things we desire at the end of the effort. Depending on the project or task at hand, various forms of Small Data probably already exist, readily available to make our work easier. Sometimes, this data will be recognizable, such as input to our program. Other times it may be camouflaged as rules, lists, summaries, or output from an existing application, which are not things we think of as data in the conventional sense.

Do we have some or all of:

- a structured list of requirements (data elements for report or results table)?
- mappings for values?
- a set of dates or parameters for iterations of a process or runs of a job?

- input data in a form that can easily be summarized?
- existing output or results (perhaps from some non-SAS tool) which produce the desired result?

If the answer is yes to any of the above, we have Small Data.

In order to create a report or a file, we would likely be given some sort of requirements document. ("Document" is a loose term here because it may actually be an Excel workbook, which is even more convenient to treat as data.) Requirements come in many forms but often involve lists of fields in tabular form -- which can be read into SAS tables or easily converted to something that can be brought into SAS tables. We generally think of a technical requirements document as what we want, but instead let's look at it in terms of what we have: it's a list of variable (or column or metric or fact) names, descriptions and possibly formatting preferences, data sources or rules for how to derive or calculate some of the variables. If the requirements document contains some but not all of these things, consider whether it would be worth building on it to create an enhanced requirements document which is the place to accumulate all of this information.

How can Small Data help us?

Small Data techniques are particularly useful for designing and building things like reports or data files, which are data-intensive, but can certainly be applied elsewhere to speed up development and improve or control programs. We can use summarized data, not just in the conventional way, where it supplies facts, but to supply or support structure.

How we use the data depends or what data we have available for the task at hand, as discussed above. It might be used to

- Define data formats -- when some sort of data mapping or categorization is needed
- Generate code or a shell for a DATA step
- Control when programs or components get run
- Validate (against sums, data types, variable names, labels)
- Keep track of definitions.

In very simple terms, working with Small Data involves using the data that is already within our grasp and simple DATA steps or procedures like PROC FORMAT, PROC CONTENTS, PROC COMPARE and PROC TRANSPOSE to manipulate that data. Small data can be helpful regardless of the application; it may even help us to harness big data.

We can make use of the Small Data quickly and easily, often with just a few lines of code. As always, we have many ways of accomplishing the same task in SAS, so selection of method for each situation depends on the size and complexity of the task, the anticipated need for future maintenance, and sometimes on personal preference. The procedure that you use often and are most familiar with - or the one the person who will maintain the code is most comfortable with – is usually a good choice.

Small Data isn't a "one technique fits all" approach. It's a flexible way of considering how to tackle designing and coding a program. Here are some examples to illustrate how Small Data can help in various aspects of our work and how you might begin to think about using it:

Example 1: Mapping values

Sample tables referred to in the examples can be found in Appendix 1.

Consider Table 1 as an example of a simple report we need to create according to the requirements shown in Table 2, with detailed input containing location (city or town and province or state) and values for X, Y and Z. We would also need to know how the zone is mapped and what reps have been assigned to which location. So, we have simple input data, some rules based on categorization of the input data and some rules not based on categorization.

Thinking Small

→ To derive Zone, PROC FORMAT might be quickest to set up and will be efficient to process.

→ To assign Rep, if we are dealing with only a small number of distinct combinations of city and state then a simple "IF <…> THEN <…> ELSE <…>" or "CASE" construct in a SAS step would handle the mapping to zone nicely.

→ When we get into larger numbers of locations that need to be mapped, though, we might look for an easier, less tedious way to code and/or a more efficient way to process the data. For those who are very database-oriented in their thinking, joining tables to pick up mapped values may be the natural choice, but with SAS you don't need to be limited to selecting values from data tables and there may be a more efficient choice. For example, small data sets which are essentially mapping tables can be turned into stored formats to simplify coding (alleviating lots of "IF <…> THEN <…>" code) OR static small data sets can be used to generate code. This would be a good time to look at a summary of the input data, and in particular at how many distinct combinations of city and state occur in the input and what the relative frequency of each combination is. Depending on the volume of data, the relative frequency of keys and the expected volatility of logic used for assignment, this might lead us to sequencing a series of "IF <…>" statements, generating a stored format from a data set or even generating the code for a series of "IF <…>" statements. Preparing to generate code involves writing code which writes other code to save time in writing code for our ultimate program. It may sound confusing but there are situations where this makes sense!

> Fun fact: How many cities and towns are there within the US and Canada? That depends on the definitions of city and town, which vary by jurisdiction, but based loosely on counting "incorporated places" there are approximately 20,000 distinct cities, towns and villages in the US and Canada combined.
>
> So, be prepared for the possibility that our very small example set may grow to 20,000 records or values to be mapped.

→ Since we'll have a summary of the keys in the input data handy, it could be used again later, when it's time to test our code. A random sample of data is helpful for testing code but a carefully selected sample which represents all the combinations of values is more helpful. Ensuring that all combinations of keys which occur in real input data are represented in our test cases means that all of the conditions will be exercised during testing with a minimal set.

→ Another approach to consider here is sorting the keys by state (or whatever is logical for the task), exporting the SAS data set to Excel, manually editing the Excel worksheet to enter the rep code, then re-importing into a SAS data set… and then maybe dynamically generating a stored format.

Some SAS code to support our Small Thinking

Note that code shown in this paper is NOT a program but rather some not-necessarily contiguous steps that may be combined into a program.

```
**------------------------------------------------------------------------**;
** Simple format:                                                       **;
** For a small number of values, this approach is fine.  For a larger   **;
** list of values, use the CNTLIN= option on PROC FORMAT to build a format**;
** directly from a data set containing summarized keys.                 **;
**------------------------------------------------------------------------**;

proc format;
    value $prov
        "BC","CA","OR","WA" = 1
        "ON","TX" = 2
        "GA","MD","ME","NS" = 3
        other = 999
;
run;

**------------------------------------------------------------------------**;
** Summarize to find all the unique combinations of City and State and  **;
** count the number of occurences of each, then sort by descending      **;
** frequency so that the most common combination of keys is first and   **;
** the least common is last.                                            **;
**------------------------------------------------------------------------**;

proc summary data = RAW_DATA nway missing sum;
    class city state_or_province;
    var x y z;
    output out = SUMMED_KEYS(drop=_type_) sum=;
run;

proc sort data = SUMMED_KEYS;
    by descending _freq_;
run;

**------------------------------------------------------------------------**;
** Generate a shell for "IF... THEN.." statements based on the data.  For **;
** efficiency, use the summarized key data sorted into descending order **;
** sequence so that the most common condition is fulfilled first.       **;
**------------------------------------------------------------------------**;

data _null_;
    set SUMMED_KEYS;
    string = "city='"|| trim(city) || "' and state_or_province = '" ||
                trim(state_or_province)||"'";
         if _n_ = 1 then put "     if " string @60 " then rep = ?;";
    else                 put "else if " string @60 " then rep = ?;";
run;
```

Example 2: Conversion task

Now, consider Table 1 as an example of a table which has been produced by some non-SAS tool which must now be replaced by a SAS program because we've reached the limits of the volume of data or complexity of additional requirements the other tool can handle. Perhaps there is too much data to continue handling in Excel or there's a need to turn preparation of the table into a hands-off, production process.

Thinking Small

- → Here, we benefit from already knowing what the correct answer is and we just need to figure out how to get from the raw data to the known answer using SAS. We may have the original requirements document, or not. Sometimes we need to work backwards from the required result to the raw data to infer the calculations or derivations.

- → Import the existing (original) results into a SAS table. Use PROC CONTENTS or Dictionary Tables to look at column names, labels, lengths and data types. These are the structural requirements for the table created by our new program.

- → Write the new SAS program. Some degree of guessing may be necessary if requirements are lacking. Run the new program to get new results.

- → Summarize numeric variables in the data set based on the original results. Do the same for the new results. Compare summaries of old and new results to determine whether the values in the summed variables match. For a small number of values, the comparisons can be a quick visual scan; for a large number of values this can be done programmatically by joining the summarized tables and flagging differences – or by using PROC COMPARE. (That means applying Small Data techniques to our Small Data.) In any case, beware of values stored as numbers in SAS that are stored differently in other databases, such as dates, and of differences in precision amongst different tools. Summing dates and comparing across tools or databases will probably not be meaningful so consider special treatment for these.

- → Get frequency counts for character columns that are categorical from the original results. Categorical (or "discrete") character values are those that take on a reasonably limited number of distinct values in our data. Things like gender, country code, province or state would be categorical, while things like names and street addresses are not because they could take on too many distinct values to give a concise summary. Do the same for the new results. Compare the frequency counts between original and new results. Again, this can be a visual comparison or can be done programmatically, depending on the volume.

- → The two steps above may seem like a lot of work but once written, they can be repeated for every iteration of changes to the SAS program with almost no effort – and if guesswork was involved in inferring requirements then there may be several iterations of the new program to check. Also, if the table we're building has hundreds or thousands of columns, there will very little extra work to validate all of them at once this way. This is especially helpful to make sure fixes to the program-in-progress don't inadvertently cause other problems. It also lets us easily keep track of how many columns are matching – which is a preliminary measure of our success – and how many don't match.

→ Investigate any differences in column names, formats or values between the original and new summaries.

→ Confirming that the sums of numeric values and the frequencies for categorical values match between the original results and the new results is *not* a substitute for validating individual details but it is a great first step which can save time by helping to ensure that program is complete before we begin a more intensive validation of details. Doing the confirmation described above first can save a lot of effort in reviewing results from PROC COMPARE, for example.

Some SAS code to support our Small Thinking

```
**------------------------------------------------------------------------**;
** Sum each of the numeric columns in the original data set.              **;
**------------------------------------------------------------------------**;

proc summary data = REPORT_DATA nway missing sum;
    var _numeric_;
    output out = SUMMED_OLD(drop=_type_) sum=;
run;

**------------------------------------------------------------------------**;
** Sum each of the numeric columns in the new data set.                   **;
**------------------------------------------------------------------------**;

proc summary data = NEW_RESULTS nway missing sum;
    var _numeric_;
    output out = SUMMED_NEW(drop=_type_) sum=;
run;

**------------------------------------------------------------------------**;
** Transpose the original and new summaries so that each column is        **;
** represented by a row in the transposed data.                           **;
**------------------------------------------------------------------------**;

proc transpose data = SUMMED_OLD out = T_SUMMED_OLD;
run;

proc transpose data = SUMMED_NEW out = T_SUMMED_NEW;
run;

**------------------------------------------------------------------------**;
** Combine the two sets of transposed data and compare the sums of each   **;
** numeric column.  Flag the non-matching sums for further investigation. **;
**------------------------------------------------------------------------**;

proc sql;
    select A.*,
           B._name_ as name2,
           B._label_ as label2,
           B.col1 as col2,
           case
                when (A.col1 = B.col1) then " "
                else "*"
                end as flag
    from T_SUMMED_OLD as A full join
         T_SUMMED_NEW as B
    on   A._name_ = B._name_ and
         A._label_ = B._label_
    order by A._name_, B._name_;
quit;
```

```
**------------------------------------------------------------------------**;
** Use Dictionary Tables to find the column name, data type, label and    **;
** length of each column in the original and new results tables.          **;
**------------------------------------------------------------------------**;

proc sql noprint;
    create table OLD_CONTENTS as
    select name, type, label, length
    from SASHELP.VCOLUMN
    where upper(libname)="WORK" and upper(memname)="REPORT_DATA"
    ;
    create table NEW_CONTENTS as
    select name, type, label, length
    from SASHELP.VCOLUMN
    where upper(libname)="WORK" and upper(memname)="NEW_RESULTS"
    ;
quit;

**------------------------------------------------------------------------**;
** Compare the attributes of columns in the original and new results      **;
** results tables.  Refer to Appendix 2 for an example of the output.     **;
** Investigate if there are any differences in values found in the        **;
** comparison OR there are different numbers of columns found in these    **;
** two tables.                                                            **;
**------------------------------------------------------------------------**;

proc compare base = OLD_CONTENTS compare = NEW_CONTENTS;
run;
```

Example 3: Treating requirements as metadata

Now, consider a need to create a reporting table formatted like Table 1 but much wider, based on a set of requirements formatted like Table 2, except much longer. The list of required data may contain hundreds of data elements that have similar names or descriptions with subtle variations. It would be a challenge to keep track of which was which. Sometimes, collecting data and performing calculations is straightforward but the sheer volume of data elements and calculations poses the greatest challenge. One way to tackle that challenge is to treat requirements as metadata. Often, requirements for a report or data file arrive in the form of a list of data elements, either in an Excel spreadsheet or some other tabular form. If we're lucky, the table includes things like the source of the data, a descriptor or short name and a written definition or long name. SAS lends itself to reading or importing that list into a data set where the data can be counted, sorted, used to assign labels and eventually compared to what's in the data sets that our new program or process produces.

Thinking Small

→ Once the requirements are imported into a SAS data set, they can be used as documentation to accompany the end product (as shown in the screen capture in Appendix 3) and/or used to compare against metadata from the new table (from Dictionary Tables or PROC CONTENTS) to ensure completeness and proper naming of columns.

→ It's easy to confirm, programmatically, that variable lengths meet specifications if those are included in the requirements data.

→ Column labels can be generated from descriptions in the requirements without the need to re-key them.

→ Consider using a structured naming convention for columns and including those column names in the requirements table, perhaps even generating the column name based on something in the requirements. This can make it easier to use SAS functions on column names as well as to keep track of the columns. If the end result is a report, consider adding a sequencing variable to the metadata to make re-sequencing or hiding columns simple in the future.

Some SAS code to support our Small Thinking

```
**------------------------------------------------------------------------**;
** Use ODS tagsets to format a report workbook which includes a tab       **;
** for Definitions which is populated from the data set created from the  **;
** Requirements document.                                                 **;
**------------------------------------------------------------------------**;

ods tagsets.excelxp file="C:\CONFERENCE_TEST.xls" style=analysis
     options(sheet_name="Report" embedded_titles='yes' autofit_height="yes" );

title1 j=left 'Example 3:  Sample Report';
proc report data = work.REPORT_DATA(obs=max) nowindows;
     columns city state_or_prov zone rep var1 var2 var3 var4;
     define var4 / f=percent8.2 ;
quit;

ods tagsets.excelxp options(sheet_name="Definitions" embedded_titles='yes' );
title1 j=left 'Include Report Requirements as a tab in the report workbook';
proc report data = REQUIREMENTS_DATA nowindows;
quit;

title1;
ods tagsets.excelxp close;
run;
```

Example 4: Treating control data (or schedules) as data

What if we need to run several SAS programs (or other programs) on a schedule? Instead of maintaining a list of which program needs to be run when and remembering to check it regularly and update it as programs are run, we could turn the list into a data set and use that to trigger submission of the appropriate programs when needed.

Thinking Small

→ Create a table of run dates for the year in Excel.

→ Import the Excel worksheet into a SAS data set.

→ On some regular basis, run a program to read the SAS data set and extract rows where the planned run date is less than or equal to the current date and the status is blank.

→ For each row that is extracted, submit the appropriate program. This can be done in an automated way based on the operating environment.

- → Have a step in each program or in an overall job submission program which updates the completion status in the SAS data set when the program finishes successfully.

- → Consider exporting the updated SAS data set to Excel when all submitted programs have ended, or keep in a SAS data set for subsequent days but generate a report on programs which should have run but don't have a successful completion status.

- → If we wanted to take this a step further toward full automation of SAS program submission, parameters could be accumulated in a table and a program scheduled for some designated time to check the table for unprocessed requests, trigger program runs using data from the table as parameters, then update the table to indicate completion. This forms a simple one-stop interface for supplying parameters, listing requests processed and outstanding requests.

CONCLUSION

Through examples, we've seen how using Small Data techniques to treat key values, summaries, requirements or control parameters as data and apply simple SAS steps to this data can make writing programs to manage data quicker, more efficient and less prone to error. None of these Small Data techniques is going to change our world the way some claim that Big Data might, but they can make our work easier with quicker results and better quality. Good things come in small packages. Don't forget to take advantage of your Small Data.

RECOMMENDED READING

Lafler, Kirk Paul. "Exploring the SAS® Metadata DICTIONARY Tables and SASHELP Views". This material was presented as a Table Talk at SAS Global Forum 2015 but not published as part of the Proceedings.
Kirk Lafler also has a number of other papers on topics related to working with metadata that may be of interest.

Williams, Christianna. "SAS® Formats Top Ten", Proceedings of SAS Global Forum 2015. Available at http://support.sas.com/resources/papers/proceedings15/3155-2015.pdf

Bilenas, Jonas V. "The Power of the PROC FORMAT", Proceedings of SAS Global Forum 2014. Available at http://support.sas.com/resources/papers/proceedings14/1504-2014.pdf

Rosenbloom, Mary F. O., Carpenter, Art. "Are You a Control Freak? Control Your Programs – Don't Let Them Control You!", Proceedings of SAS Global Forum 2015. Available at
http://support.sas.com/resources/papers/proceedings15/2220-2015.pdf
This is an advanced paper on data-driven control.

CONTACT INFORMATION

Your questions about the contents of this paper or comments are welcome.

Lisa Eckler
lisa.eckler@sympatico.ca

APPENDIX 1: Sample Data

City	State or Prov	Zone	Rep	var1	var2	var3	var4	…
Baltimore	MD	3	A	82035	43197	41585	49.18%	
Dallas	TX	2	A	10756	42004	23247	14.15%	
Halifax	NS	3	A	87546	79923	32157	43.86%	
Portland	ME	3	A	92667	5956	7212	87.56%	
Portland	OR	1	B	99456	34382	61027	51.04%	
San Antonio	TX	2	B	98677	69339	42664	46.84%	
San Diego	CA	1	B	43833	63439	33088	31.23%	
San Francisco	CA	1	B	38147	47188	91319	21.59%	
Savannah	GA	3	C	60390	26825	45381	45.54%	
Seattle	WA	1	C	80700	9553	65601	51.78%	
Toronto	ON	2	C	98949	30394	94949	44.12%	
Vancouver	BC	1	C	62006	90095	40204	32.24%	
…								

Table 1. Sample data table or report

Name	Description
City	Name of the city or town being measured
State or Prov	2 character abbreviation for the state or province the city or town is in
Zone	Numeric code for the zone the city or town is in. Zone is a function of the combination of city and state or province, so each city x state or province combination maps to one zone.
Rep	1 character code for the representative for each location being measured. There is no derivation here, it is an assigned value.
var1	Total number of occurrences of X at this location.
var2	Total number of occurrences of Y at this location.
var3	Total number of occurrences of Z at this location.
var4	var4 is calculated as var1 divided by the sum of var1 + var2 + var3
…	…

Table 2. Sample Requirements

Record	City	State or Province	X	Y	Z	…
1	Toronto	ON	61	92	34	
2	Dallas	TX	63	9	64	
3	Savannah	GA	7	57	79	
4	Halifax	NS	82	93	74	
5	Toronto	ON	21	97	36	
6	Toronto	ON	58	32	71	
7	Halifax	NS	43	40	26	
8	Savannah	GA	83	57	22	
…						

Table 3. Sample Raw Data

program name	program title	planned run date	completion status
my_program123	Monday Report	05Jan2015	
my_program123	Monday Report	12Jan2015	
my_program123	Monday Report	19Jan2015	
my_program123	Monday Report	26Jan2015	
my_program123	Monday Report	02Feb2015	
my_program123	Monday Report	09Feb2015	
…			
my_program124	Monthly Report	05Jan2015	
my_program124	Monthly Report	05Jan2015	
my_program124	Monthly Report	05Feb2015	
my_program124	Monthly Report	05Mar2015	
my_program124	Monthly Report	05Apr2015	
my_program124	Monthly Report	05May2015	
…			
my_program125	Quarterly Report	15Mar2015	
my_program125	Quarterly Report	15Jun2015	
my_program125	Quarterly Report	15Sep2015	
my_program125	Quarterly Report	15Dec2015	

Table 4. Sample Control Data

APPENDIX 2: Sample Output

The COMPARE Procedure
Comparison of WORK.OLD_CONTENTS with WORK.NEW_CONTENTS
(Method=EXACT)

Data Set Summary

Dataset	Created	Modified	NVar	NObs
WORK.OLD_CONTENTS	03JUL15:12:23:26	03JUL15:12:23:26	4	9
WORK.NEW_CONTENTS	03JUL15:12:23:27	03JUL15:12:23:27	4	8

Variables Summary

Number of Variables in Common: 4.

Observation Summary

Observation	Base	Compare
First Obs	1	1
Last Match	8	8
Last Obs	9	.

Number of Observations in Common: 8.
Number of Observations in WORK.OLD_CONTENTS but not in WORK.NEW_CONTENTS: 1.
Total Number of Observations Read from WORK.OLD_CONTENTS: 9.
Total Number of Observations Read from WORK.NEW_CONTENTS: 8.

Number of Observations with Some Compared Variables Unequal: 0.
Number of Observations with All Compared Variables Equal: 8.

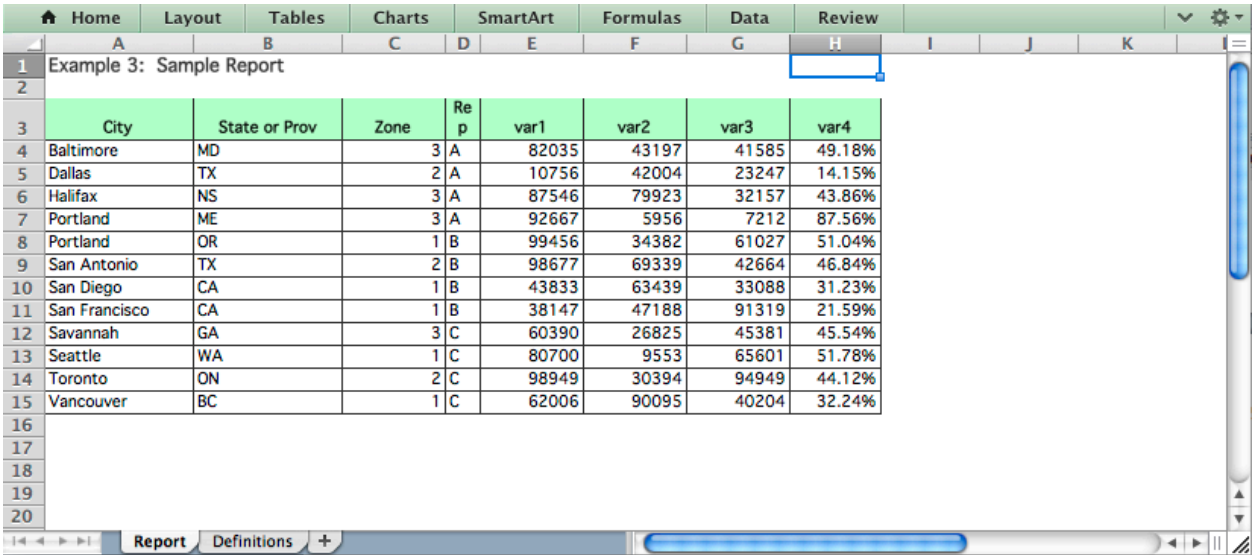
NOTE: No unequal values were found. All values compared are exactly equal.

Table 5. Partial output from PROC COMPARE in Example 2. The red and green has been added to highlight what is important here. All the variables in common match but there is one variable found in the old table and not in the new table.

NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	COL1	NAME OF FORMER VARIABLE	LABEL OF FORMER VARIABLE	col2	Flag
Zone	Zone	23	Zone	Zone	23	
FREQ		12	_FREQ_		12	
I		78			.	*
var1	var1	855162	var1	var1	855162	
var2	var2	542295	var2	var2	541095	*
var3	var3	578434	var3	var3	579634	*
var4	var4	5.1913	var4	var4	5.1913	

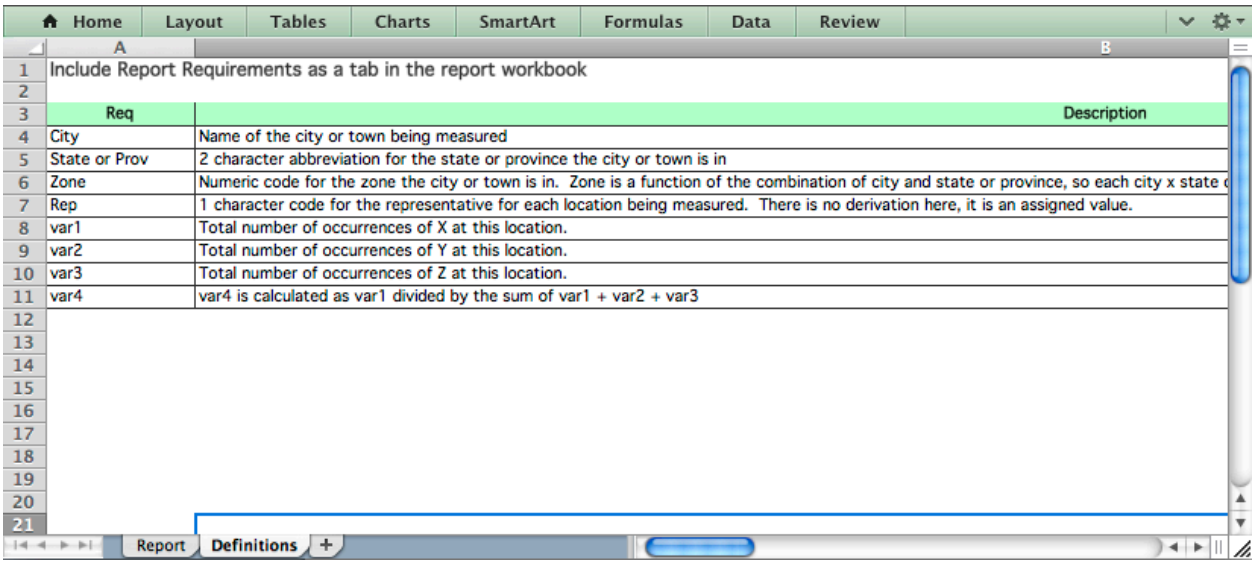
Table 6. Results from comparison in Example 2. The red has been added to highlight: "i" is an extraneous variable from the old program but two values differ and will need to be investigated further.

APPENDIX 3: Screen captures from Example 3 workbook



Example 3: Sample Report

City	State or Prov	Zone	Rep	var1	var2	var3	var4
Baltimore	MD	3	A	82035	43197	41585	49.18%
Dallas	TX	2	A	10756	42004	23247	14.15%
Halifax	NS	3	A	87546	79923	32157	43.86%
Portland	ME	3	A	92667	5956	7212	87.56%
Portland	OR	1	B	99456	34382	61027	51.04%
San Antonio	TX	2	B	98677	69339	42664	46.84%
San Diego	CA	1	B	43833	63439	33088	31.23%
San Francisco	CA	1	B	38147	47188	91319	21.59%
Savannah	GA	3	C	60390	26825	45381	45.54%
Seattle	WA	1	C	80700	9553	65601	51.78%
Toronto	ON	2	C	98949	30394	94949	44.12%
Vancouver	BC	1	C	62006	90095	40204	32.24%



Include Report Requirements as a tab in the report workbook

Req	Description
City	Name of the city or town being measured
State or Prov	2 character abbreviation for the state or province the city or town is in
Zone	Numeric code for the zone the city or town is in. Zone is a function of the combination of city and state or province, so each city x state
Rep	1 character code for the representative for each location being measured. There is no derivation here, it is an assigned value.
var1	Total number of occurrences of X at this location.
var2	Total number of occurrences of Y at this location.
var3	Total number of occurrences of Z at this location.
var4	var4 is calculated as var1 divided by the sum of var1 + var2 + var3