
Yanmei Lu, Contractor for DataMeans & Russ Lavery, Contractor, Numeric Resources

ABSTRACT
This paper discusses how Proc Tabulate and Proc Summary can be used to help a programmer avoid errors. This paper might be especially useful for new programmers.

A programmer should know where “every observation goes”. Additionally, Good Programming Practice requires that a programmer “exercise” (check in some way) every combination of the clauses in every IF statement to see how observations are classified by the IF. Proc Tabulate is an excellent, fast and simple tool for doing that checking. Importantly, Proc Tabulate produces a pattern, in its output, that makes checking fast. Additionally, using Proc Tabulate allows a programmer to account for every observation in the data set, a “comforting” fact.

Many programming tasks require taking subsets of the data, based on logical rules. Programmers are often told to check the number of observations in the starting data set and not to lose, or create, observations without knowing why. With Proc Summary we can also suggest that a programmer also check the sums of important variables (Dollars, Scripts, units sold) in the source data set and never change the sum without knowing why.

SECTION 1) VALIDATING CODE WITH PROC TABULATE:
Complex if statements can be easily checked using Proc Tabulate by exploiting patterns in Proc Tabulate output. Learning some Proc Tabulate options make the job easier. First, use the ALL option on both the row and column dimensions in the table to show column, row and overall totals. Second, the missing option should be used so that missing values of the class variables show up in the table. Thirdly, use the standard tricks (noseps, PS=, LS= format= and RTS=) to make output fit on a page for easy reading. Finally, creating the table in a logical manner makes the pattern in the output easier to spot. As a warning, it is preferable to use this technique on a test data set or a final version of the data.

GOALS AND PROBLEMS IN DEBUGGING
A programmer should account for every observation and should “exercise” every path through a series of IF statements and rigorously check assignment formulas. The SAS method of propagating missing values makes this task difficult for complex assignment statements. This paper uses IF statements as examples for checking logic, but the technique is also useful for checking other assignments like formats.

Imagine a series of IF statements with four components and a result. (this example is not carried further in the paper)

If state=="CA" and sex=="F" and hobby=="Beach" and Group=="Beach Boys" then
Personality=="California Girl" ; Else ........

It is difficult to predict the number of obs. in any level of the variable Personality from looking at freqs of variables (state, hobby and group). Predicting counts of people in the different levels of Personality is especially complex where there are many missing values for state, sex, hobby and group. The predicting counts problem is difficult because observations can have missing values for more than one of the variables (state, sex, hobby or group). The simple freqs for state sex, hobby and group, shown below, do not show that there are obs. that have multiple missing/Unknown values.

<table>
<thead>
<tr>
<th>sex</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>100</td>
<td>0.307</td>
<td>100</td>
<td>0.30</td>
</tr>
<tr>
<td>M</td>
<td>200</td>
<td>0.615</td>
<td>300</td>
<td>0.92</td>
</tr>
<tr>
<td>Unk</td>
<td>25</td>
<td>0.076</td>
<td>325</td>
<td>1.00</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Frequency</th>
<th>Percent</th>
<th>Cumulative Frequency</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>272</td>
<td>0.836</td>
<td>272</td>
<td>0.83</td>
</tr>
<tr>
<td>DE</td>
<td>8</td>
<td>0.024</td>
<td>280</td>
<td>0.86</td>
</tr>
<tr>
<td>Unk</td>
<td>45</td>
<td>0.138</td>
<td>325</td>
<td>1.00</td>
</tr>
</tbody>
</table>
FREQs on individual variables do not show if obs. are mission on both state and sex. The number of obs. that “fail” the above IF statement might be different from the number of obs. with missing sex added to the number of obs. with missing state added to the number of obs. with missing group. A ‘multivariate’ tool is required for checking complex IF statements and assignments. Proc Tabulate is just that tool.

For simple IF statements, Proc Freq can also be used. However, Proc Freq is less powerful than Proc Tabulate and it is more less able to create output with an easily checked graphic pattern that can easily be checked.

Below is a program that reads data and creates variables using a fairly complex series of IF statements. It is not more less able to create output with an easily checked graphic pattern that can easily be checked.

Remember Proc Tabulate is a very powerful cross-tab producer. The Proc Tabulate statement that specifies the look/form of the table is:

Table list of variables to be put on the side of the table (separated by an ’) - then a comma - then a list of variables to be put across the across the top of the table.

The trick is: Put the variables on the left side of the = in your IF statement on the “side of the table” and the - then a comma - then a list of variables to be put across the across the top of the table.

The examples are shown below.

The example assigns values of an imaginary physical exam test to three levels (low normal or high) based on the resulting class on the “top of the table”.

The logic to the left causes the creation of unpleasantly complex nested if statements. A short sample of the code is shown below. The full code is in the appendix can be pasted into SAS if one wants to reproduce this example.

 Program: TABULATE_4_CC
Programmer: YL & RL Date 11/25/2005
Purpose:
Illustrate using proc tabulate to check if statement coding.
If statements are to be used to check assignment of flags to lab tests
Create a data set with multiple missing values to make coding difficult

Business/Medical Logic for cutpoints is shown below:

<table>
<thead>
<tr>
<th>sex</th>
<th>age</th>
<th>pre_condition</th>
<th>Test</th>
<th>of</th>
<th>Normal</th>
<th>of</th>
<th>Normal</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>7–12</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>25</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>13–21</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>16</td>
<td>22</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>21–50</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>19</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>51–70</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>23</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>7–12</td>
<td>active</td>
<td>Treadmill</td>
<td>20</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>13–21</td>
<td>active</td>
<td>Treadmill</td>
<td>11</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>21–50</td>
<td>active</td>
<td>Treadmill</td>
<td>14</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>51–70</td>
<td>active</td>
<td>Treadmill</td>
<td>18</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>7–12</td>
<td>athletic</td>
<td>Treadmill</td>
<td>15</td>
<td>18</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>13–21</td>
<td>athletic</td>
<td>Treadmill</td>
<td>6</td>
<td>12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>21–50</td>
<td>athletic</td>
<td>Treadmill</td>
<td>9</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M</td>
<td>51–70</td>
<td>athletic</td>
<td>Treadmill</td>
<td>13</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>7–12</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>27</td>
<td>30</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>13–21</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>18</td>
<td>24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>21–50</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>21</td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>51–70</td>
<td>sedentary</td>
<td>Treadmill</td>
<td>25</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>7–12</td>
<td>active</td>
<td>Treadmill</td>
<td>22</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>13–21</td>
<td>active</td>
<td>Treadmill</td>
<td>13</td>
<td>19</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>21–50</td>
<td>active</td>
<td>Treadmill</td>
<td>16</td>
<td>21</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>51–70</td>
<td>active</td>
<td>Treadmill</td>
<td>20</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>7–12</td>
<td>athletic</td>
<td>Treadmill</td>
<td>17</td>
<td>20</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>13–21</td>
<td>athletic</td>
<td>Treadmill</td>
<td>8</td>
<td>14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>21–50</td>
<td>athletic</td>
<td>Treadmill</td>
<td>11</td>
<td>16</td>
<td></td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>51–70</td>
<td>athletic</td>
<td>Treadmill</td>
<td>15</td>
<td>18</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The IF statements take up many lines of code and our task is to check both the data and our accuracy in coding the IF statements. The above code and three Proc Tabulates can be pasted into SAS and used for study.

**HINTS TO MAKE THE TECHNIQUE MORE EFFECTIVE**

Proc Tabulate is most useful when a properly designed test data set is created to test the IF statements and then Proc Tabulate is used to show how those observations were assigned. A properly constructed data set exercises all paths through the if statements and lets the programmer can know that the coded logic is 100% correct.

However, it is reasonably good practice to use Proc Tabulate to check how “real data” is classified if the data is in its final form (no more refreshing to be done), even if all paths through the IF statements are not exercised. If the data set is in the final state, this technique tests all the combinations of variables used in the IF statements that are actually in the data. This practice does not check that your code is 100% perfect, but it does check that your code has perfectly classified all the data that happens to be in the dataset. If the data set is “frozen”, this is a reasonably good check of coding quality.

It is dangerous to use this Proc Tabulate technique on data that will be refreshed at a future date, as often happens in a clinical trial. It would be poor practice to check an early version of a clinical trial dataset (say with 25% of the observations entered) and to consider the program logic “validated” because Proc Tabulate shows that all that 25% of the obs. were classified correctly. New data, with new combinations of variables, could bring new problems.

Below, in blue, find the log note for the creation of the example dataset followed by the output from the first Proc Tabulate. The log (pasted in in blue) shows there are 26 observations in the dataset and all 26 show up on the tabulate (see RED 26 in Lower Right Hand Corner of the table below). Since the table shows every observation in the dataset, if we understand this table, we understand the whole dataset. If we use a QC procedure that checks every obs. in the dataset, there will be no hidden surprises.

The use of continuous variables in the IF can make the table long, but there are some things to do to make the output easier to read and to make any pattern in the data stand out. It is usually not necessary to examine every line of the table. One can save much time by exploiting patterns in the table. Exploiting patterns will be illustrated below.

It is suggested that one use the all option on both the row and column dimension in the Proc Tabulate. This causes printing of total number of obs. in the table in the lower right hand corner (the red 26 below). The missing option should be used in the Proc Tabulate so that missing values of the class variables show up (see blue box below) in the table. Using these options assures that the number is the lower right hand corner of the table is the number of obs. in the data set. In this example, note that the last subject in the data set, Steve, has several missing values. An example of an obs. with multiple missing class values is seen in the solid green box. An obs. with a missing continuous value is shown in the dotted green box below.

Formatting the output so that all the output fits on one page, and structuring the table to emphasize the pattern in the logic, makes for easier reading. Use the standard Proc Tabulate tricks (noseps, PS=, LS=, format= and RTS=) to make output fit on a page.

The assigned Flag values have a numeric prefix to force a logical sorting (1-Low, 2_Norm…) order in the Proc Tabulate output. A logical sorting order makes the pattern in the output easier to spot (see red box to be discussed later). There are other ordering tricks.

**OUTPUT AND INTERPRETATION**

What jumps out from the table below is that data issues that have affected the IF statements. There is a biking test (green box), that should not be in the data at all. Treadmill is spelled two ways. Male is coded in upper and lower case and athletic is spelled two ways. Finally, there is a missing value for the variable named value. Tabulate lets a programmer check EVERY observation in the data set and determine why classifications were made as they were.

In well-conditioned data, patterns jump out of the table. This data set is small and of very poor quality so the pattern is not as strong as it usually is in real data. To see an example of the pattern, look at the solid red boxes. It is
suggested that this pattern be read as: For treadmill, athletic females with values from 7 to 10 are coded to “1_LOW”. All subjects with values between 7 and 10 can be considered as a group. If subjects with value of 7 and 10 are classified correctly, we likely do not need to check the values between 7 and 10. At 11 and at 20, the pattern shifts and we should pay attention to where the pattern shifts.

Even if there were a great many levels of the variable value, this table could be validated quickly by checking the first and last elements in a group.

The output can run over several pages and a speed trick is to check logic at 1) the start of a section (treadmill-M-8 years old), 2) the end of a section and 3) where the pattern changes inside a section. Two pattern changes are illustrated by the arrows slanting down and to the right. The “shift to the right” is a break in the pattern and an indication that a different IF statement took effect.

The patterns can be exploited to speed checking as follows: if M-Sedentary-18 is correct and M-Sedentary-8-20 is correct, we can assume rows between these two rows (here just M-Sedentary-8-19 and no big time savings) are correct. In real data, there can be many rows between the values you check. If the data is fairly clean and the table can be structured so that the pattern is easy to see, every observation in very large data sets can be checked quickly by checking the places where the pattern changes.

Additionally, observations with multiple missing values are easy to spot (see dotted red box). When the variable being created is the result of a formula a missing value in the formula usually results in a missing result. Missing values in SAS will propagate. Tabulate is especially good at identifying observations with multiple missing values.
NOTE: The data set WORK.IMAGINARY_LABS has 26 observations and 8 variables.

```
  real time    0.57 seconds  cpu time    0.07 seconds
  test  ,sex ,lifestyle ,age      ,value         ,      ,      ,      ,      ,      ,  N   ,  N   ,  N   ,  N   ,  N   ,  N   ,
  , F   , Athletic , 29 , 18 , 1 , . , . , . , . , . , . , . , 1 , 1 ,

  , F   , Athletic , 29 , 7 , 7 , . , . , . , . , . , . , . , 1 , 1 ,

  , F   , Active , 12 , 26 , 1 , . , . , 1 , 1 ,

  , F   , Active , 12 , 30 , 1 , . , . , 1 , 1 ,

  , M   , Active , 12 , 26 , 1 , . , . , 1 , 1 ,

  , M   , Active , 12 , 26 , 1 , . , . , 1 , 1 ,

  , M   , Treadmill , 10 , 26 , 1 , . , . , 1 , 1 ,

  , M   , Sedentary , 8 , 18 , 1 , . , 1 , 1 ,

  , M   , Sedentary , 8 , 18 , 2 , . , . , 1 , 1 ,

  , M   , Sedentary , 55 , 11 , 1 , . , . , 1 , 1 ,

  , M   , Active , 30 , 16 , 1 , . , . , 1 , 1 ,

  , M   , Active , 12 , 19 , 1 , . , . , 1 , 1 ,

  , M   , Treadmill , 10 , 26 , 1 , . , . , 1 , 1 ,

  , M   , Treadmill , 10 , 26 , 1 , . , . , 1 , 1 ,

0.57 seconds 0.07 seconds
```
---

**Treadmill**

<table>
<thead>
<tr>
<th>Sex</th>
<th>Lifestyle</th>
<th>Age</th>
<th>Value</th>
<th>Test</th>
<th>Sex</th>
<th>Lifestyle</th>
<th>Age</th>
<th>Value</th>
<th>Test</th>
<th>Sex</th>
<th>Lifestyle</th>
<th>Age</th>
<th>Value</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>Active</td>
<td>29</td>
<td>18</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>F</td>
<td>Athletic</td>
<td>29</td>
<td>7</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>F</td>
<td>Active</td>
<td>12</td>
<td>26</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Active</td>
<td>12</td>
<td>30</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Active</td>
<td>12</td>
<td>26</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Sedentary</td>
<td>8</td>
<td>18</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Sedentary</td>
<td>8</td>
<td>18</td>
<td>2</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Sedentary</td>
<td>55</td>
<td>11</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Active</td>
<td>30</td>
<td>16</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Active</td>
<td>12</td>
<td>19</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Treadmill</td>
<td>10</td>
<td>26</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
<tr>
<td>M</td>
<td>Treadmill</td>
<td>10</td>
<td>26</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>1</td>
<td>1</td>
<td>.</td>
<td>1</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>
Below, the table was changed to have Flag nested under Sex. It is not obvious in the SAS code, but the ranges for men and women have thresholds that differ by 2 units. A table that shows men and women side by side, lets that pattern show. It can be seen, in the red box below, that an active 29-year-old with a value of 15 will be classified into normal if male, and high if female. Again, the table should be constructed to make the pattern easy to see.
SECTION 2) VALIDATING CODE WITH PROC SUMMARY:

Almost every SAS® programmer was told at the beginning of her/his training to “never lose, or create, an observation without knowing why it happened.” With Proc Summary, this instruction can be expanded to (assuming that the programmer is working with a data set containing dollars) to “never change the total dollar value in the data set without knowing why”. If a programmer tracks every observation, and every dollar, from the raw data to the final data set, s/he can have a high level of comfort with the answer. This paper uses a simple project to illustrate the technique. Using Proc Summary with the types statement makes this simple to do.

The technique that is described in this paper is simple. It consists of running Proc Summary, with the “appropriate” types statement, on the raw dataset and again whenever the dataset is subset or merged. This paper will use the dataset sashelp.prdsales, and a data set we created, to illustrate this technique.

PROBLEMS STATEMENT/BUSINESS TASK
Imagine that the client wants a report on Canadian sales from the data set sashelp.prdsales. Specifically, the client wants a report on Canadian sales activity broken out by the person responsible for buying the type of goods (called the Buyer). A flowchart of the process is shown to right. We will check the totals in the file after each subset and merge. Most often we will use a summary to check the totals.

Start out with a Proc Summary on the raw dataset. A convenient trick is to paste the information from the listing and log into an XLS sheet. This allows the programmer to keep from having to remember long numbers. It also allows a programmer to do calculations easily.

Start out with a Proc Summary on the raw dataset. A convenient trick is to paste the information from the listing and log into an XLS sheet. This allows the programmer to keep from having to remember long numbers. It also allows a programmer to do calculations easily.

*initial summary on raw data;
Proc summary data=sashelp.prdsale
   missing print sum;
Class country;
Var actual;
Types () country;
The log says: NOTE: There were 1440 observations read from the data set SASHELP.PRDSALE. and the red Proc Summary output (pasted into Excel at right) shows that we have accounted for every one.

Canadian totals are 480 obs and 246,990 dollars. The programmer now knows s/he must keep track of 246,990 dollars in sales as the program progresses. Merge in names of “buyers” and the products that they buy and watch the totals.

DATA BUYERS;
INFILE DATALINES missover;
INPUT @1 Buyer_Name $CHAR8. @10 PRODUCT $CHAR10.;
DATALINES;
 /*NOTE: Intentionally there is no buyer for TABLE*/
BROWN   BED
CHUNG   CHAIR
DAVIS   DESK
SUNG     SOFA
; run;

proc sort data=buyers; by product; run;

Proc sort data=sashelp.prdsale out=prdsale; by product;
where country="CANADA"; run; /*Get the Canadian Sales*/

data post_merge;
merge prdsale(in=p) BUYERS(in=B);
/*Merge in the buyer information*/
by product;
prod_buy=P*10 + B*1; /*this variable can be used to monitor the merge*/ run;

*Use proc summary to check n and sum after the merge*/
Proc summary data=post_merge missing print sum;
Class Buyer_Name country product_buy; /*Check the total after a merge*/
var actual;
types () country*prod_buy*Buyer_Name; run;

After a subset, or merge, we submit another Proc Summary. The types statement lets a programmer get overall and detail dollar summaries in one Proc Summary. Types and Ways statements are powerful.

We pasted results into Excel as a convenience and reformatted the data into SAS font then did a text to columns conversion.

The red arrow points to the result for type () showing that the subset still contains every obs. and dollar for Canadian sales.

The bottom of the blue arrow points to the result of a summation done after pasting the listing into Excel. The blue arrow shows that we are accounting for all the dollars after we merged in buyer information. However, the red row, in the screenprint has a missing buyer and indicates that our data step did not “find” a match for one product. Pasting text from the log into Excel makes this QC easy.

Types () will match numbers in source
Types Var1*var2*etc will match numbers after sub-setting

IF WE ARE ONLY GOING TO REPORT ON KNOWN BUYERS, WE WILL PASS ON $200,290 DOLLARS AND 96*4=384 OBS.

Since we only want to report where we have buyer information. We summarize and then filter on the variable we used to monitor the merge. Proc Print, with the sum statement, can show us what the data set looks like and allow us to keep track of the total dollars.

Proc summary
data=post_merge;
class country Buyer_Name product year; output
ton sum_yr sum(actual)=;
ways 4;
where prod_buy=11;
run;

%macro skip;
Proc print data=sum_yr;
sum _numeric_ ;
run;
We need to transpose the file to get the report to look the way the client wants it. That is shown below.

```sas
proc transpose data=sum_yr
   out=TR_sum_yr(drop=_name_) prefix=Yr_
   var actual;
   id year;
by country Buyer_Name product;
run;
```

proc print data=TR_sum_yr;
   sum _numeric_;
run;

We pasted text from the listing into Excel and used Excel to check that the column totals still add to $200,290 (see red rectangle).

The report is now done and we have kept track of every dollar as it passed through the program. There can be high confidence in that the report is correct.

This might be a good trick to teach beginning programmers.

**CONCLUSION**

Good Programming Practice requires that a programmer keep track of every observations and totals of important variables as they write programs. Proc Tabulate and Proc Summary are both powerful QC tools. They allow programmers to keep track of observations and important totals as a program is being written.

**CONTACT INFORMATION**

Your comments and questions are valued and encouraged. You can contact the author at:

Yanmei Lu, Independent Contractor for Datameans  
Email: Yanmeiliu@hotmail.com  

Russell Lavery, Independent Contractor for Numeric Resources  
Email: russ.lavery@verizon.net  

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.
Appendix

data imaginary_labs ;
infile datalines firstobs=4 missover;
input @1 name $char6. @9 visit 1. @15 sex $1. @21 age 2. @30 lifestyle $char9. 
@45 test $char9. @60 value 2.0;
tflag="Initial";
if test="Treadmill" then
do;
tflag="looping"; /*Just to track execution*/
if sex="M" then
do;
   If lifestyle="Sedentary" then
   do;
      tflag="Sed";
      if age GE 7 and age LT 12 then
      do;
         if value LT 25 then tflag="1_Low ";
         else if value GT 28 then tflag="3_High";
         else tflag="2_Norm";
      end;
      Else if age GE 13 and age LT 21 then
      do;
         if value LT 16 then tflag="1_Low ";
         Else if value GT 22 then tflag="3_High";
         else tflag="2_Norm";
      end;
      ELSE if age GE 21 and age LT 50 then
      do;
         if value LT 19 then tflag="1_Low ";
         Else if value GT 24 then tflag="3_High";
         else tflag="2_Norm";
      end;
      Else if age GE 51 and age LE 70 Then
      do;
         if value LT 23 then tflag="1_Low ";
         else if value GT 26 then tflag="3_High";
         else tflag="2_Norm";
      end;
end; /*Male Sedentary*/
Else If lifestyle="Active" then
   do;
      tflag="Act"; /*Just to track execution*/
      if age GE 7 and age LE 12 then
      do;
         if value LT 20 then tflag="1_Low ";
         else if value GT 23 then tflag="3_High";
         else tflag="2_Norm";
      end;
      if age GE 13 and age LE 21 then
      do;
         if value LT 11 then tflag="1_Low ";
         else if value GT 17 then tflag="3_High";
         else tflag="2_Norm";
      end;
      if age GE 21 and age LE 50 then
      do;
         if value LT 14 then tflag="1_Low ";
         else if value GT 19 then tflag="3_High";
         else tflag="2_Norm";
      end;
      if age GE 51 and age LE 70 then
      do;
         if value LT 18 then tflag="1_Low ";
         else if value GT 21 then tflag="3_High";
         else tflag="2_Norm";
      end;
end; /*Male Active*/
Else If lifestyle="Athletic" then
do;
tflag="Ath"; /* Just to track execution */
if
  age GE 7 and age LE 12 then
    do;
      if
        value LT 15 then tflag="1_Low ";
      else if
        value GT 18 then tflag="3_High ";
      else tflag="2_Norm ";
    end;
  if
  age GE 13 and age LE 21 then
    do;
      if
        value LT 6 then tflag="1_Low ";
      else if
        value GT 12 then tflag="3_High ";
      else tflag="2_Norm ";
    end;
  if
  age GE 21 and age LE 50 then
    do;
      if
        value LT 9 then tflag="1_Low ";
      else if
        value GT 14 then tflag="3_High ";
      else tflag="2_Norm ";
    end;
  if
  age GE 51 and age LE 70 then
    do;
      if
        value LT 13 then tflag="1_Low ";
      else if
        value GT 16 then tflag="3_High ";
      else tflag="2_Norm ";
    end;
end; /* M athletic */
end; /* sex = M*/
ELSE if
sex="F" then
  do;
    If
    lifestyle="Sedentary" then
      do;
        tflag="Sed";
        if
          age GE 7 and age LT 12 then
            do;
              if
                value LT 27 then tflag="1_Low ";
              else if
                value GT 30 then tflag="3_High ";
              else tflag="2_Norm ";
            end;
          Else if
          age GE 13 and age LT 21 then
            do;
              if
                value LT 18 then tflag="1_Low ";
              else if
                value GT 24 then tflag="3_High ";
              else tflag="2_Norm ";
            end;
          ELSE if
          age GE 21 and age LT 50 then
            do;
              if
                value LT 21 then tflag="1_Low ";
              else if
                value GT 26 then tflag="3_High ";
              else tflag="2_Norm ";
            end;
          Else if
          age GE 51 and age LT 70 then
            do;
              if
                value LT 25 then tflag="1_Low ";
              else if
                value GT 28 then tflag="3_High ";
              else tflag="2_Norm ";
            end;
        end; /* F Sedentary */
      Else If
      lifestyle="Active" then
        do;
          tflag="Act"; /* Just to track execution */
          if
            age GE 7 and age LE 12 then
              do;
                if
                  value LT 22 then tflag="1_Low ";
                else if
                  value GT 25 then tflag="3_High ";
                else tflag="2_Norm ";
              end;
            if
            age GE 13 and age LE 21 then
              do;
                if
                  value LT 13 then tflag="1_Low ";
else if value GT 19 then tflag="3_High";
else tflag="2_Norm";
end;
if age GE 21 and age LE 50 then
  do;
    if value LT 16 then tflag="1_Low ";
  else if value GT 21 then tflag="3_High";
    else tflag="2_Norm";
  end;
if age GE 51 and age LE 70 then
  do;
    if value LT 20 then tflag="1_Low ";
else if value GT 23 then tflag="3_High";
    else tflag="2_Norm";
  end;
end; /*F Active*/
Else If lifestyle="Athletic" then
  do;
    tflag="Ath";/*Just to track execution*/
      if age GE 7 and age LE 12 then
        do;
          if value LT 17 then tflag="1_Low ";
        else if value GT 20 then tflag="3_High";
          else tflag="2_Norm";
        end;
      if age GE 13 and age LE 21 then
        do;
          if value LT 8 then tflag="1_Low ";
        else if value GT 14 then tflag="3_High";
          else tflag="2_Norm";
        end;
      if age GE 21 and age LE 50 then
        do;
          if value LT 11 then tflag="1_Low ";
        else if value GT 16 then tflag="3_High";
          else tflag="2_Norm";
        end;
      if age GE 51 and age LE 70 then
        do;
          if value LT 15 then tflag="1_Low ";
        else if value GT 18 then tflag="3_High";
          else tflag="2_Norm";
        end;
end; /*F athletic*/
end; /*sex = F*/
End ; /*test=treadmill*/
datallines;
  1 2 3 4 5 6 7 8
12345678901234567890123456789012345678901234567890123456789012345678901234567890
name visit sex age lifestyle test value
Illust 1 M 8 Sedentary Treadmill 18
Illust 2 M 8 Sedentary Treadmill 18
Illust 3 M 8 Sedentary Treadmill 19
Illust 4 M 8 Sedentary Treadmill 20
Illust 5 M 8 Sedentary Treadmill 26
Illust 6 M 8 Sedentary Treadmill 30
Illust 7 M 8 Sedentary Treadmill 33
Illust 8 M 8 Sedentary Treadmill 35
Russ 1 M 55 Sedentary Treadmill 28
Russ 2 M 55 Sedentary Treadmill 29
Russ 3 M 55 Sedentary Treadmill 11
Debb 1 F 29 Athletic Treadmill 09
Debb 2 F 29 Athletic Treadmill 12
Debb 3 F 29 Athletic Biking 18
Shu 1 F 29 Athletic Treadmill 10
Shu 2 F 29 Athletic Treadmill 11
<table>
<thead>
<tr>
<th>Name</th>
<th>Gender</th>
<th>Age</th>
<th>Type</th>
<th>Activity</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alex</td>
<td>M</td>
<td>12</td>
<td>Active</td>
<td>Treadmill</td>
<td>26</td>
</tr>
<tr>
<td>Alex</td>
<td>M</td>
<td>12</td>
<td>Active</td>
<td>treadmill</td>
<td>19</td>
</tr>
<tr>
<td>Pam</td>
<td>F</td>
<td>29</td>
<td>Athletic</td>
<td>Treadmill</td>
<td>7</td>
</tr>
<tr>
<td>Pam</td>
<td>F</td>
<td>29</td>
<td>Athletic</td>
<td>Treadmill</td>
<td>15</td>
</tr>
<tr>
<td>Pam</td>
<td>F</td>
<td>29</td>
<td>Athletic</td>
<td>Treadmill</td>
<td>20</td>
</tr>
<tr>
<td>Guy</td>
<td>M</td>
<td>29</td>
<td>Athletic</td>
<td>Treadmill</td>
<td>15</td>
</tr>
<tr>
<td>Steve</td>
<td>M</td>
<td>30</td>
<td>Active</td>
<td>Treadmill</td>
<td>16</td>
</tr>
<tr>
<td>Steve</td>
<td>M</td>
<td>30</td>
<td>Active</td>
<td>Treadmill</td>
<td>16</td>
</tr>
<tr>
<td>Steve</td>
<td>M</td>
<td>30</td>
<td>Active</td>
<td>Treadmill</td>
<td>16</td>
</tr>
<tr>
<td>Steve</td>
<td>M</td>
<td>30</td>
<td>Active</td>
<td>Treadmill</td>
<td></td>
</tr>
<tr>
<td>run;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>