An Array of Conditions: Using Arrays to Condense Repetitive Code
Amanda Browning, Educational Testing Service, Princeton, NJ

ABSTRACT

Arrays are often seen as intimidating and are sometimes avoided by SAS® programmers, possibly because they are conceptually difficult compared to other techniques. As a result, programmers may be missing out on a strategy that can save time and increase accuracy. Arrays can make coding more efficient and reduce the possibility of error in programming by condensing repetitive coding. The purpose of this paper is to demonstrate how arrays can simplify complex conditional coding problems. The topics covered in this paper are suitable for beginner and intermediate programmers.

INTRODUCTION

Often, there are multiple ways to solve programming problems in SAS. Sometimes simple lines of code are just as adequate as more complicated techniques, such as arrays. From time to time, however, we get data that is more complex and requires extremely repetitive coding. While it is possible to handle it with simple statements, this strategy is tedious and thus, we make ourselves more vulnerable to error.

The example we will examine throughout this paper consists of a fictional company's sales data for one year. The format of the data will require us to use conditional logic to reorganize and make sense of the data. The solution will be presented using both simple conditional code and arrays to illustrate the benefits of using arrays when repetitive coding is required.

DATA

We have a text file with one year of sales data for a company. Figure 1 below illustrates the file's layout. Each line of data begins with an account number, followed by a month, that month’s Sales, the Cost of goods sold and finally, the Revenue from those sales. Shown in this figure are the first three entries for six accounts. We need to analyze the data by account number, but we also need to analyze the data by month. The data we are given, however, are only in variable order (Month, Sales, Cost & Revenue). We know in which columns each variable will occur because we know the length and position of each variable, but when we read the data with SAS, we cannot be certain that the first month we read for each account will be January (01), or the second will be February (02), etc. This may be because there are not sales for each month for each account, or the company may have gained or lost an account mid-year. As we can see highlighted in Figure 1, most of the values in the first Month variable (columns 8-9) represent accounts corresponding to January (01), but the third and sixth accounts do not.

Figure 1

<table>
<thead>
<tr>
<th>Variable Column</th>
<th>Acct_No</th>
<th>Month</th>
<th>Sales</th>
<th>Cost</th>
<th>Revenue</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101572</td>
<td>01</td>
<td>400</td>
<td>125</td>
<td>275</td>
</tr>
<tr>
<td>2</td>
<td>102552</td>
<td>01</td>
<td>700</td>
<td>250</td>
<td>450</td>
</tr>
<tr>
<td>3</td>
<td>160421</td>
<td>02</td>
<td>875</td>
<td>300</td>
<td>575</td>
</tr>
<tr>
<td>4</td>
<td>11298</td>
<td>01</td>
<td>550</td>
<td>250</td>
<td>300</td>
</tr>
<tr>
<td>5</td>
<td>127226</td>
<td>01</td>
<td>775</td>
<td>250</td>
<td>525</td>
</tr>
<tr>
<td>6</td>
<td>107328</td>
<td>10</td>
<td>200</td>
<td>025</td>
<td>175</td>
</tr>
</tbody>
</table>

The text file is read in using the following SAS code.

DATA Sales_Data;
 infile "C:\Documents and Settings\abrowning\Desktop\NESUG\salesdata.txt" lrecl = 140 missover pad;
 input
   @1 Acct_No $CHAR6.
   @8 (Month1 - Month12) (2. + 9)
   @10 (Sales1 - Sales12) (3. + 8)
   @13 (Cost1 - Cost12) (3. + 8)
   @16 (Revenue1 - Revenue12) (3. + 8);
Again, even though we are reading in 12 months of data, we are simply reading the variables in the order they appear in the text file. Each account may not have the corresponding monthly data. This is the case in our example: if we look at the third observation in Figure 2 below, the entry under Month1 is for February (02). There were no sales for this account in January. Similarly, the 6th account only contains fourth quarter sales; therefore, the Month1 entry corresponds to sales in October (10).

**Figure 2**

<table>
<thead>
<tr>
<th>Acct_No</th>
<th>Month1</th>
<th>Month2</th>
<th>Month3</th>
<th>Sales1</th>
<th>Sales2</th>
<th>Sales3</th>
<th>Cost1</th>
<th>Cost2</th>
<th>Cost3</th>
<th>Revenue1</th>
<th>Revenue2</th>
<th>Revenue3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>400</td>
<td>500</td>
<td>600</td>
<td>125</td>
<td>200</td>
<td>250</td>
<td>275</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>02</td>
<td>03</td>
<td>03</td>
<td>700</td>
<td>600</td>
<td>550</td>
<td>250</td>
<td>200</td>
<td>200</td>
<td>450</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>875</td>
<td>450</td>
<td>450</td>
<td>300</td>
<td>125</td>
<td>100</td>
<td>575</td>
<td>325</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>02</td>
<td>03</td>
<td>03</td>
<td>550</td>
<td>550</td>
<td>900</td>
<td>250</td>
<td>200</td>
<td>375</td>
<td>300</td>
<td>350</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>02</td>
<td>03</td>
<td>03</td>
<td>775</td>
<td>400</td>
<td>200</td>
<td>250</td>
<td>125</td>
<td>025</td>
<td>525</td>
<td>275</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>200</td>
<td>450</td>
<td>775</td>
<td>25</td>
<td>100</td>
<td>175</td>
<td>350</td>
<td>525</td>
</tr>
</tbody>
</table>

In order to be able to conduct analyses by month, we need to organize the data so that Month1 actually corresponds only to values of 01, Month2 only lists values of 02, and so on. Figure 3 shows Month1, Month2 and Month3 in the order we desire. This is accomplished by creating new variables (MonthN, SalesN, CostN and RevN) and assigning monthly data to these new variables.

**Figure 3**

<table>
<thead>
<tr>
<th>Acct_No</th>
<th>MonthN1</th>
<th>SalesN1</th>
<th>CostN1</th>
<th>RevN1</th>
<th>MonthN2</th>
<th>SalesN2</th>
<th>CostN2</th>
<th>RevN2</th>
<th>MonthN3</th>
<th>SalesN3</th>
<th>CostN3</th>
<th>RevN3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01</td>
<td>400</td>
<td>125</td>
<td>275</td>
<td>02</td>
<td>500</td>
<td>200</td>
<td>300</td>
<td>03</td>
<td>600</td>
<td>250</td>
<td>350</td>
</tr>
<tr>
<td>2</td>
<td>01</td>
<td>700</td>
<td>250</td>
<td>450</td>
<td>02</td>
<td>600</td>
<td>200</td>
<td>400</td>
<td>03</td>
<td>550</td>
<td>200</td>
<td>350</td>
</tr>
<tr>
<td>3</td>
<td>02</td>
<td>875</td>
<td>300</td>
<td>575</td>
<td>02</td>
<td>550</td>
<td>200</td>
<td>350</td>
<td>03</td>
<td>900</td>
<td>375</td>
<td>525</td>
</tr>
<tr>
<td>4</td>
<td>02</td>
<td>400</td>
<td>125</td>
<td>275</td>
<td>03</td>
<td>200</td>
<td>025</td>
<td>175</td>
<td>03</td>
<td>200</td>
<td>025</td>
<td>175</td>
</tr>
<tr>
<td>5</td>
<td>01</td>
<td>775</td>
<td>250</td>
<td>525</td>
<td>02</td>
<td>400</td>
<td>125</td>
<td>275</td>
<td>03</td>
<td>200</td>
<td>025</td>
<td>175</td>
</tr>
<tr>
<td>6</td>
<td>01</td>
<td>775</td>
<td>250</td>
<td>525</td>
<td>02</td>
<td>400</td>
<td>125</td>
<td>275</td>
<td>03</td>
<td>200</td>
<td>025</td>
<td>175</td>
</tr>
</tbody>
</table>

**SIMPLE CONDITIONAL CODE**

As with most programming in SAS, there are multiple ways to attack this problem. This can be done with simple conditional statements. Sometimes, simple code is just as effective as more complex coding techniques. We will first examine the coding solution to this data issue using simple conditional statements and later using arrays. The example will illustrate that there are cases where the decision to utilize arrays over simple statements is clear.

The following code will create new variables for Month1. For each account, SAS checks to see if the Month1 variable we created when we read in the data has a value of 1. If it is does, SAS creates the MonthN1, SalesN1, CostN1 and RevenueN1 variables and assigns to them the values from Month1, Sales1, Cost1 and Revenue1 respectively. Regardless of whether the first if statement is true, SAS continues checking each statement. Next, SAS checks to see if the Month1 variable has a value of 2. If Month1 has a value of 2, SAS creates the MonthN2, SalesN2, CostN2 and RevN2 variables and assigns to them the values from Month1, Sales1, Cost1 and Revenue1 respectively. SAS continues to check Month1 for values up through 12.

**DATA**  
Sales_Data_Long;  
set Sales_Data;  
  if Month1=1 then do;  
    MonthN1=1;SalesN1=Sales1;CostN1=Cost1;RevN1=Revenue1;end;  
  if Month1=2 then do;  
    MonthN2=2;SalesN2=Sales1;CostN2=Cost1;RevN2=Revenue1;end;  
  if Month1=3 then do;  
    MonthN3=3;SalesN3=Sales1;CostN3=Cost1;RevN3=Revenue1;end;  
  if Month1=4 then do;  
    MonthN4=4;SalesN4=Sales1;CostN4=Cost1;RevN4=Revenue1;end;  
  if Month1=5 then do;
MonthN5=5;SalesN5=Sales1;CostN5=Cost1;RevN5=Revenue1;end;
if Month1=6 then do;
  MonthN6=6;SalesN6=Sales1;CostN6=Cost1;RevN6=Revenue1;end;
if Month1=7 then do;
  MonthN7=7;SalesN7=Sales1;CostN7=Cost1;RevN7=Revenue1;end;
if Month1=8 then do;
  MonthN8=8;SalesN8=Sales1;CostN8=Cost1;RevN8=Revenue1;end;
if Month1=9 then do;
  MonthN9=9;SalesN9=Sales1;CostN9=Cost1;RevN9=Revenue1;end;
if Month1=10 then do;
  MonthN10=10;SalesN10=Sales1;CostN10=Cost1;RevN10=Revenue1;end;
if Month1=11 then do;
  MonthN11=11;SalesN11=Sales1;CostN11=Cost1;RevN11=Revenue1;end;
if Month1=12 then do;
  MonthN12=12;SalesN12=Sales1;CostN12=Cost1;RevN12=Revenue1;
RUN;

Below are the first three observations of data for the first month. Figure 4 shows the data as it was read in and Figure 5 displays the data assigned to our newly created variables. For the first two accounts, the data from Month1, Sales1, Cost1, and Rev1 are copied to Month_N1, SalesN1, CostN1 and RevN1. For the third account, the Month1, Sales1, Cost1, and Revenue1 variables are copied to MonthN2, SalesN2, CostN2, and RevN2. This process continues through Month12 and the data is copied to the new variables that correspond to the actual month.

<table>
<thead>
<tr>
<th>Acct_No</th>
<th>Month1</th>
<th>Sales1</th>
<th>Cost1</th>
<th>Revenue1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101572</td>
<td>01</td>
<td>400</td>
<td>125</td>
</tr>
<tr>
<td>2</td>
<td>102552</td>
<td>01</td>
<td>700</td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>160421</td>
<td>02</td>
<td>875</td>
<td>300</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Account_No</th>
<th>MonthN1</th>
<th>SalesN1</th>
<th>CostN1</th>
<th>RevN1</th>
<th>MonthN2</th>
<th>SalesN2</th>
<th>CostN2</th>
<th>RevN2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101572</td>
<td>01</td>
<td>400</td>
<td>125</td>
<td>02</td>
<td>500</td>
<td>200</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>102552</td>
<td>01</td>
<td>700</td>
<td>250</td>
<td>02</td>
<td>600</td>
<td>200</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>160421</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>02</td>
<td>875</td>
<td>300</td>
<td>575</td>
</tr>
</tbody>
</table>

As we can see, the simple conditional coding successfully organizes the data by month. It is also very cumbersome code. This programming technique is time consuming and exposes us to error. It is very easy to make a mistake and use the wrong variable(s) somewhere in this code. The following is an example of the coding required to read just the first, second and twelfth months of the data. We can imagine the amount of coding necessary to read all twelve months of data.

DATA Sales_Data_Long;
set Sales_Data;
  if Month1=1 then do;
    MonthN1=1;SalesN1=Sales1;CostN1=Cost1;RevN1=Revenue1;end;
  if Month1=2 then do;
    MonthN2=2;SalesN2=Sales1;CostN2=Cost1;RevN2=Revenue1;end;
  if Month1=3 then do;
    MonthN3=3;SalesN3=Sales1;CostN3=Cost1;RevN3=Revenue1;end;
  if Month1=4 then do;
    MonthN4=4;SalesN4=Sales1;CostN4=Cost1;RevN4=Revenue1;end;
  if Month1=5 then do;
    MonthN5=5;SalesN5=Sales1;CostN5=Cost1;RevN5=Revenue1;end;
  if Month1=6 then do;
    MonthN6=6;SalesN6=Sales1;CostN6=Cost1;RevN6=Revenue1;end;
  if Month1=7 then do;
    MonthN7=7;SalesN7=Sales1;CostN7=Cost1;RevN7=Revenue1;end;
  if Month1=8 then do;

3
MonthN8=8;SalesN8=Sales1;CostN8=Cost1;RevN8=Revenue1;end;
if Month1=9 then do;
    MonthN9=9;SalesN9=Sales1;CostN9=Cost1;RevN9=Revenue1;end;
if Month1=10 then do;
    MonthN10=10;SalesN10=Sales1;CostN10=Cost1;RevN10=Revenue1;end;
if Month1=11 then do;
    MonthN11=11;SalesN11=Sales1;CostN11=Cost1;RevN11=Revenue1;end;
if Month1=12 then do;
    MonthN12=12;SalesN12=Sales1;CostN12=Cost1;RevN12=Revenue1;end;
if Month2=1 then do;
    MonthN1=1;SalesN1=Sales2;CostN1=Cost2;RevN1=Revenue2;end;
if Month2=2 then do;
    MonthN2=2;SalesN2=Sales2;CostN2=Cost2;RevN2=Revenue2;end;
if Month2=3 then do;
    MonthN3=3;SalesN3=Sales2;CostN3=Cost2;RevN3=Revenue2;end;
if Month2=4 then do;
    MonthN4=4;SalesN4=Sales2;CostN4=Cost2;RevN4=Revenue2;end;
if Month2=5 then do;
    MonthN5=5;SalesN5=Sales2;CostN5=Cost2;RevN5=Revenue2;end;
if Month2=6 then do;
    MonthN6=6;SalesN6=Sales2;CostN6=Cost2;RevN6=Revenue2;end;
if Month2=7 then do;
    MonthN7=7;SalesN7=Sales2;CostN7=Cost2;RevN7=Revenue2;end;
if Month2=8 then do;
    MonthN8=8;SalesN8=Sales2;CostN8=Cost2;RevN8=Revenue2;end;
if Month2=9 then do;
    MonthN9=9;SalesN9=Sales2;CostN9=Cost2;RevN9=Revenue2;end;
if Month2=10 then do;
    MonthN10=10;SalesN10=Sales2;CostN10=Cost2;RevN10=Revenue2;end;
if Month2=11 then do;
    MonthN11=11;SalesN11=Sales2;CostN11=Cost2;RevN11=Revenue2;end;
if Month2=12 then do;
    MonthN12=12;SalesN12=Sales2;CostN12=Cost2;RevN12=Revenue2;end;
    .
    .
    if Month12=1 then do;
        MonthN1=1;SalesN1=Sales12;CostN1=Cost12;RevN1=Revenue12;end;
if Month12=2 then do;
    MonthN2=2;SalesN2=Sales12;CostN2=Cost12;RevN2=Revenue12;end;
if Month12=3 then do;
    MonthN3=3;SalesN3=Sales12;CostN3=Cost12;RevN3=Revenue12;end;
if Month12=4 then do;
    MonthN4=4;SalesN4=Sales12;CostN4=Cost12;RevN4=Revenue12;end;
if Month12=5 then do;
    MonthN5=5;SalesN5=Sales12;CostN5=Cost12;RevN5=Revenue12;end;
if Month12=6 then do;
    MonthN6=6;SalesN6=Sales12;CostN6=Cost12;RevN6=Revenue12;end;
if Month12=7 then do;
    MonthN7=7;SalesN7=Sales12;CostN7=Cost12;RevN7=Revenue12;end;
if Month12=8 then do;
    MonthN8=8;SalesN8=Sales12;CostN8=Cost12;RevN8=Revenue12;end;
if Month12=9 then do;
    MonthN9=9;SalesN9=Sales12;CostN9=Cost12;RevN9=Revenue12;end;
if Month12=10 then do;
    MonthN10=10;SalesN10=Sales12;CostN10=Cost12;RevN10=Revenue12;end;
if Month12=11 then do;
    MonthN11=11;SalesN11=Sales12;CostN11=Cost12;RevN11=Revenue12;end;
if Month12=12 then do;
    MonthN12=12;SalesN12=Sales12;CostN12=Cost12;RevN12=Revenue12;end;
RUN;

Using conditional statements to handle this programming problem is redundant and prime for error. It works, but there is a better way. Using an array allows us to do the exact same work with much less code.
Arrays are sometimes avoided because they are a little more complicated than simpler lines of code. They are a way to streamline repetitive coding and consequently reduce the possibility of error in your program.

```sas
DATA Sales_Data;
infile "C:\Documents and Settings\abrowning\Desktop\NESUG\salesdata.txt" lrecl = 140 missover pad;
input
   @1   Acct_No   $CHAR6.
   @8   (Month1 - Month12)   (2. + 9)
   @10  (Sales1 - Sales12)   (3. + 8)
   @13  (Cost1 - Cost12)     (3. + 8)
   @16  (Revenue1 - Revenue12) (3. + 8);
Run;
```

As you can see, we read the data in just as we did previously. Remember, we read in Month1 through Month12, but the data is not necessarily in that order. We need to sift through the data and associate it with its true month’s data. In order to do this, we need to set up input and output arrays. We need a set of arrays to hold the values of the variables we read from the text file (input arrays). We also need a set of arrays to handle the new variables we want to create (output arrays).

The input array will contain the data we read in from the text file. Our array statements are simply assigning these variables to the arrays we specify. As you can see below, the Month array holds Month1 through Month12, the Sales array holds Sales1 through Sales12, the Cost array holds Cost1 through Cost12 and the Revenue array holds Revenue1 through Revenue12.

```sas
DATA Sales_Data_Array;
   set Sales_Data;
   array Month[12] Month1 - Month12;  *input array for Month;
   array Sales[12] Sales1 - Sales12;  *input array for Sales;
   array Cost[12] Cost1 - Cost12;   *input array for Cost;
   array Revenue[12] Revenue1 - Revenue12; *input array for Revenue;
   array MonthN[12] MonthN1-MonthN12;     *output array for Month;
   array SalesN[12] SalesN1 - SalesN12;  *output array for Sales;
   array CostN[12] CostN1 – CostN12;   *output array for Cost;
   array RevenueN[12] RevenueN1 - RevenueN12; *output array for Revenue;
```

As you can see, we read the data in just as we did previously. Remember, we read in Month1 through Month12, but the data is not necessarily in that order. We need to sift through the data and associate it with its true month’s data. In order to do this, we need to set up input and output arrays. We need a set of arrays to hold the values of the variables we read from the text file (input arrays). We also need a set of arrays to handle the new variables we want to create (output arrays).

```sas
do i = 1 to 12;
do j = 1 to 12;
   if Month[i]=j then do;
      MonthN[j]=j;
      SalesN[j]=Sales[i];
      CostN[j]=Cost[i];
      RevN[j]=Revenue[i];
   end;
end;
RUN;
```
Figure 6 shows the first 3 months of input data. We have the same issue we had before. We need to move the data in order to be able to analyze monthly sales.

<table>
<thead>
<tr>
<th>Acct_No</th>
<th>Month1</th>
<th>Month2</th>
<th>Month3</th>
<th>Sales1</th>
<th>Sales2</th>
<th>Sales3</th>
<th>Cost1</th>
<th>Cost2</th>
<th>Cost3</th>
<th>Revenue1</th>
<th>Revenue2</th>
<th>Revenue3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101572</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>400</td>
<td>500</td>
<td>600</td>
<td>125</td>
<td>200</td>
<td>250</td>
<td>275</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>102552</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>700</td>
<td>600</td>
<td>550</td>
<td>250</td>
<td>200</td>
<td>200</td>
<td>450</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>160421</td>
<td>02</td>
<td>03</td>
<td>04</td>
<td>875</td>
<td>450</td>
<td>450</td>
<td>300</td>
<td>125</td>
<td>100</td>
<td>575</td>
<td>325</td>
</tr>
<tr>
<td>4</td>
<td>111298</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>550</td>
<td>550</td>
<td>900</td>
<td>250</td>
<td>200</td>
<td>375</td>
<td>300</td>
<td>525</td>
</tr>
<tr>
<td>5</td>
<td>127226</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>775</td>
<td>400</td>
<td>200</td>
<td>125</td>
<td>025</td>
<td>525</td>
<td>275</td>
<td>175</td>
</tr>
<tr>
<td>6</td>
<td>107328</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>200</td>
<td>450</td>
<td>775</td>
<td>25</td>
<td>100</td>
<td>250</td>
<td>175</td>
<td>350</td>
</tr>
</tbody>
</table>

The output array variables are those new variables we create to hold the data when it is reorganized by month. In our example, each array has 12 elements and each is assigned the 12 new variables for Month, Sales, Cost and Revenue. These array statements simply initialize the new variables we want created. The array statement creates the specified variables, but all values are blank (see Figure 7) until we assign them in the conditional part of our coding.

```plaintext
array MonthN[12] MonthN1-MonthN12;
array SalesN[12] SalesN1 - SalesN12;
array CostN[12] CostN1 - CostN12;
```

Finally, we have the conditional statements we need to move the monthly data to its appropriate location. We have two DO loops to process the data. The first, outer loop (do i = 1 to 12;), corresponds to the input variables and the second, inner loop (do j = 1 to 12;), corresponds to the new variables we are creating.

```plaintext
do i = 1 to 12;
  do j = 1 to 12;
    if Month[i]=j then do;
      MonthN[j]=j;
      SalesN[j]=Sales[i];
      CostN[j]=Cost[i];
      RevN[j]=Revenue[i];
    end;
  end;
end;
```

This array does exactly what the simple statements do. It starts with the outer loop (i=1) with Month1 and checks its value against the inner loop values of j (from 1 to 12). If the conditional month statement Month1=1 is true, it assigns Month1 and its associated sales data to MonthN1, SalesN1, CostN1 and RevN1. When the program has checked Month1’s value against all values of j (1 through 12), it then moves returns to the outer loop and i increases by 1. This continues until the program has checked all 12 months of the input data.

```plaintext
if Month1=1 then do;
  MonthN1=1;
  SalesN1=Sale1;
```
CostN1=Cost1;
RevN1=Revenue1;
end;

if Month1=2 then do;
  MonthN2=1;
  SalesN2=Sale1;
  CostN2=Cost1;
  RevN2=Revenue1;
end;

if Month12=12 then do;
  MonthN12=12;
  SalesN12=Sale12;
  CostN12=Cost12;
  RevN12=Revenue12;
end;

The result is exactly what we want. Figure 8 shows our Month, Sales, Cost and Revenue data has been moved to the appropriate new variables. Under Month1, we now only see data entries that actually have January (01) as the Month. The financial data corresponding to January (01) has also been moved to the appropriate new variables: SalesN1, CostN1 and RevN1.

<table>
<thead>
<tr>
<th>Acct No</th>
<th>MonthN1</th>
<th>MonthN2</th>
<th>MonthN3</th>
<th>SalesN1</th>
<th>SalesN2</th>
<th>SalesN3</th>
<th>CostN1</th>
<th>CostN2</th>
<th>CostN3</th>
<th>RevN1</th>
<th>RevN2</th>
<th>RevN3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>101572</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>400</td>
<td>500</td>
<td>600</td>
<td>125</td>
<td>200</td>
<td>250</td>
<td>275</td>
<td>300</td>
</tr>
<tr>
<td>2</td>
<td>102552</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>700</td>
<td>600</td>
<td>550</td>
<td>250</td>
<td>200</td>
<td>200</td>
<td>450</td>
<td>400</td>
</tr>
<tr>
<td>3</td>
<td>160421</td>
<td>.</td>
<td>02</td>
<td>03</td>
<td>.</td>
<td>875</td>
<td>450</td>
<td>.</td>
<td>300</td>
<td>100</td>
<td>.</td>
<td>575</td>
</tr>
<tr>
<td>4</td>
<td>111298</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>550</td>
<td>550</td>
<td>900</td>
<td>250</td>
<td>200</td>
<td>375</td>
<td>300</td>
<td>350</td>
</tr>
<tr>
<td>5</td>
<td>127226</td>
<td>01</td>
<td>02</td>
<td>03</td>
<td>775</td>
<td>400</td>
<td>200</td>
<td>250</td>
<td>125</td>
<td>025</td>
<td>525</td>
<td>275</td>
</tr>
<tr>
<td>6</td>
<td>107328</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
<td>.</td>
</tr>
</tbody>
</table>

CONCLUSION

When we compare simple statements and array coding for our problem, it is clear arrays save time and improve the quality of a program when extremely repetitive code is involved. The array code handles all 12 months of data in just 21 lines. One month of coding using the simple conditional statements take 12 lines, for a total of 144 lines for all 12 months. Imagine if we had 2 or 3 years of data. There would be no revisions necessary to our array code, but we would need to add double or triple the coding to our simple code. Obviously, there are instances where arrays are an extremely valuable tool for SAS programmers.

ACKNOWLEDGMENTS

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 registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:
Amanda C. Browning
Educational Testing Service
Rosedale Road MS 11-P
Princeton, NJ 08541
(609) 734-1868
abrowning@ets.org