Paper 93-2010

How I Learned to Type 1,388,571 wpm Or Creating Extensive Shift Tables

Kelley Weston; Quintiles; Overland Park, KS USA

Abstract

At times in the pharmaceutical industry we need to be able to create some code called a shift table, which in essence is just a tool that can show the progression of a subject's test. If there are very few possible values, and only the first and last visits to consider, then this may be a trivial task. However, if there are several visits, and / or many possible values, then this becomes something that takes some planning.

Introduction

As an example, if a patient has a given test at only the first and last visits, and there are only 2 possible results of the test, say "A" and "B", then the shift table might be represented as "AB", meaning that this patient had a result of "A" at the first visit, and "B" at the last visit. Furthermore, we might want to spell out the meanings of these values in another variable. In this case, it might be represented as "Value A to Value B". This results in only 4 (2 * 2) possible test result combinations: AA, AB, BA, BB.

However, what if instead of only 2 visits, there were more - let's say 4; and if instead of only 2 values, there were more - let's say 7. Now instead of only 4 possible combinations, there are now 7 * 7 * 7 * 7 = 2401 combinations. How do we now create all of the possible combinations, and how do we create the second variable that spells out all of the possible meanings of those combinations?

We will look at the following:

1. The goal – what specifically we need to create, and in what form
2. The problems – what is involved in attempting to accomplish the goal
3. The solution
4. The method – how the solution is implemented

The goal

The usual method of creating the second shift table variable – the one with the meanings spelled out – is to simply apply a format. This means that we need to have a format with values that look something like the following:

"AAAA" = "Value A to Value A to Value A to Value A"
…
"GGGG = "Value G to Value G to Value G to Value G"

In this example, with 4 positions, and 7 possible values, there will be 2401 values, including missing values. For example, if we are looking at the first visit, we would just want to print "Value A", not "Value A to missing to missing to missing".

The Problems

The problems include the following:

- There are far too many values to type into a program
- Typing the values manually would involve a lot of time
- Ensuring that all of the values are accounted for
- Ensuring that there are no duplicates
- Ensuring that all values and meanings are entered correctly

The Solution

Let SAS® create the combinations and meanings for the combinations.

The Method

The method can be outlined as follows:

1. Create a data set with the individual values and meanings. This will consist of only a very few observations. In this example, we have 7 values.
2. Let SAS create all the combinations and put them into a data set. Cross the data set with itself 3 times to create the 4 positions. This will use the Cartesian product.
3. Post-process the data set to account for baseline and missing values.
4. Create the data sets from which we will create the format.
5. Write out the formats.
6. Use the formats.

Step 1 – Create the data set

We can create one variable that will be in the variable that holds the values, and a second variable that holds the meanings of the first variable:

```
DATA work.values;
  Length x $1 y $10;
  x = "A"; y = "Value A"; OUTPUT;
  x = "B"; y = "Value B"; OUTPUT;
  x = "C"; y = "Value C"; OUTPUT;
  x = "D"; y = "Value D"; OUTPUT;
  x = "E"; y = "Value E"; OUTPUT;
  x = "M"; y = "Missing"; OUTPUT;
  x = "-"; y = "-";       OUTPUT;
RUN;
```

This represents 5 possible measurements, plus a missing measurement, and finally a placeholder for a measurement that has not yet occurred.

Step 2 – Create the combinations

We now need to create all of the possible combinations of the values, in order to create the 4 positions. Since we will be creating a Cartesian product, the easiest way to do this is with PROC SQL.

```
PROC SORT DATA = work.values;
  BY x;
Run;

PROC SQL;
  * cross the data set with itself;
  CREATE TABLE work.values2 AS
    SELECT (a.x !! b.x)           AS x,
           (a.y !! " to " !! b.y) AS y
    FROM work.values AS a,
         work.values AS b;

  * cross the above data set with itself;
  * define the lengths of each variable;
  CREATE TABLE work.values3 AS
    Select (a.x !! b.x)                 AS x
             LENGTH = 200,
           COMPBL(a.y !! " to " !! b.y) AS y
             LENGTH = 200
    FROM work.values2 AS a,
         Work.values2 AS b;
QUIT;
```

NOTE: you will receive NOTES in the log that you are performing a Cartesian product, and that it cannot be optimized. This is just a note, and it expected, and can be otherwise ignored.

Step 3 – Post-process the data set

Since the Cartesian product will form all combinations, all of the values of the definitions will have 4 places. However, some of the definitions will represent the baseline (first visit) values, in which case the first variable (in this case "x") will end with 3 dashes. We wish the definition to specify that it is at baseline. This will be the first condition to check.

The last thing to do is to eliminate any multiple blanks.

```
DATA work.values3 (DROP = to_);
  SET work.values3;

  IF x EQ "----"
     THEN y = "At baseline";
```

```
      IF SUBSTR(x, 2) EQ "---" AND
         X NE "----"
         THEN y = SCAN(y, 1) !! " at baseline";

      Y = COMPBL(y);
    RUN;
```

You could certainly do any other post-processing that would be appropriate for your situation. This will give you some ideas for the possibilities.

Step 4 – Create the format data set

In creating the format data set, we need to define a few variables with specific names, since the format data set will require them.

- The "start" variable represents the starting value – in this case, this is the variable "x". This will be on the left side of the equal sign in the value statement.
- The "label" variable represents the meaning of the starting value. This will be on the right side of the equal sign in the value statement.
- The "fmtname" variable will become the format name.
- The "type" variable represents the type of the format – "c" for character, or "n" for numeric.
- The "end" variable

```
DATA work.ctrl;
      LENGTH x y $200;
      SET work.values;
      RETAIN fmtname 'cat' type 'c';
      RENAME x = start
             y = label;
      end = x;
RUN;

DATA work.ctrl2;
      LENGTH x y $200;
      SET work.values3;
      RETAIN fmtname 'cat_all' type 'c';
      RENAME x = start
             y = label;
      end = x;
RUN;
```

Step 5 – Write out the formats

Since we have the values in one data set, and the meanings in another data set, we can actually write out a format from each data set, giving us maximum flexibility in the use of them. Both data sets will be written out in a similar manner.

The first format to be created is from the values data set, containing just the unique values. In this instance in contains only 7 observations.

```
     PROC FORMAT LIBRARY = library
                 CNTLIN  = work.ctrl;
     RUN;
```

The second format is from the final data set, containing the Cartesian product values, that in this instance contains 2401 observations.

```
     PROC FORMAT LIBRARY = library
                 CNTLIN  = work.ctrl2;
     RUN;
```

Step 6 – Using the formats

To use the formats to create new variables, you can use something like the following:

Assuming that variable "x1" contains only a single character that is one of the unique values, use something like this:

```
     Y = PUT(x1, $cat.);
```

If variable "x2" contains a 4-character string, where each character is one of the unique values, use something like this:

```
     Z = PUT(x2, $cat_all.);
```

Conclusion / Summary

What initially looked like an insurmountable task – entering many values, ensuring accuracy, no omissions, etc. – actually turned out to be a relatively easy job with a little planning.

Oh yeah – you may be wondering about the title. In calculating how many characters were created in the dataset, and the time it took to run the program, it turned out that it was the equivalent of typing over a million words per minute, if I were to type it all in manually.

References

SAS Online Documentation

About the Author

Kelley has over 18 years experience with SAS programming. Kelley has been a programmer with a major telecommunications company and a SAS instructor teaching SAS classes nationwide, as well as teaching hands-on workshops at regional SAS conferences and PharmaSUG. His programming experience also includes clinical trials with a major pharmaceutical company and an international CRO. He is an active member of KCASUG (Kansas City Area SAS Users Group), including stints as Webmaster (2006 & 2007) and Chairman (since 2008).

Contact Information

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

Kelley Weston
c/o Quintiles
6700 W. 115th St.
Overland Park, KS 66211
Kw.mwsug@gmail.com

Trademarks

Appendix

Here is a sample of a shift table produced by this method (there were 2401 combinations produced in this format):

```
start    end      label

----     ----     At baseline
---A     ---A     - to - to - to Value A
---B     ---B     - to - to - to Value B
---C     ---C     - to - to - to Value C
---D     ---D     - to - to - to Value D
---E     ---E     - to - to - to Value E
---M     ---M     - to - to - to Missing
--A-     --A-     - to - to Value A to -
--AA     --AA     - to - to Value A to Value A
...
-AB-     -AB-     - to Value A to Value B to -
-ABA     -ABA     - to Value A to Value B to Value A
-ABB     -ABB     - to Value A to Value B to Value B
-ABC     -ABC     - to Value A to Value B to Value C
-ABD     -ABD     - to Value A to Value B to Value D
-ABE     -ABE     - to Value A to Value B to Value E
-ABM     -ABM     - to Value A to Value B to Missing
...
```

A---	A---	Value at baseline
A--A	A--A	Value A to - to - to Value A
A--B	A--B	Value A to - to - to Value B
A--C	A--C	Value A to - to - to Value C
A--D	A--D	Value A to - to - to Value D
A--E	A--E	Value A to - to - to Value E
A--M	A--M	Value A to - to - to Missing
A-A-	A-A-	Value A to - to Value A to -
A-AA	A-AA	Value A to - to Value A to Value A
...		
ABA-	ABA-	Value A to Value B to Value A to -
ABAA	ABAA	Value A to Value B to Value A to Value A
ABAB	ABAB	Value A to Value B to Value A to Value B
ABAC	ABAC	Value A to Value B to Value A to Value C
ABAD	ABAD	Value A to Value B to Value A to Value D
ABAE	ABAE	Value A to Value B to Value A to Value E
ABAM	ABAM	Value A to Value B to Value A to Missing
...		
MMMA	MMMA	Missing to Missing to Missing to Value A
MMMB	MMMB	Missing to Missing to Missing to Value B
MMMC	MMMC	Missing to Missing to Missing to Value C
MMMD	MMMD	Missing to Missing to Missing to Value D
MMME	MMME	Missing to Missing to Missing to Value E
MMMM	MMMM	Missing to Missing to Missing to Missing