Tricks of A Truly Lazy SAS Programmer

Chris Toppe, Ph.D.
Independent Sector

I'm not sure where I picked up these tricks, mostly from SUGI and the regional user group meetings, I guess, over the last 10 years or so. Up till then, I used the manuals, which is a pretty inefficient way to learn. Then, staring with the first SESUG, I started attending these types of user events and my ability to do what I wanted really skyrocketed. Here are a collection of little short cuts and tricks that make my life easier, the kind of things that I need to do over and over again, and the kind of things I need to do to get what I want. I use them all the time. Maybe there's something here that'll work for you, too. I've tried to make then short and sweet in the examples so you can cut and paste if you find something you want to try.

Creating Variables

The first trick is for creating new variables. I show two kinds: binomials and ranges. The test data is just that, a small dataset that let's you see what's happening.

```
DATA TEMP;
INPUT @1 ID 1. @3GENDER 1. @5
INCOME 5.;
CARDS;
1 1 15000
1 2 36000
2 1 47500
2 2 22000
3 1 51000
3 2 77000
;
RUN;
```

The first variable I create is a classification variable that assigns people (observations) into an income groups of $25,000. To do this, I divide income by 25,000, take the integer, and add one. For example, the first observation has an income of $15,000. When I divide that by 25,000, I get a number less than zero, so the integer is 0. I add 1 to that and get the value of the income class, INCCLS, as 1. Piece of cake.

The second variable I create is a binomial. Note the simplicity of the code: one line. This line of code evaluates the value of the gender variable. If the value is 1, the new, binomial variable, MALE, is set to 1. All other values of gender result in MALE being set to 0.

The results of the Proc Print® show that INCCLS and MALE were both created as needed.

```
DATA TEMP;
SET TEMP;
INCCLS=(INT(INCOME/25000))+1;
MALE = (GENDER=1);
RUN;

ODS RTF FILE='C:\SESUG_1.DOC';
TITLE "CREATING NEW VARIABLES
THE EASY WAY";
PROC PRINT; RUN;
ODS RTF CLOSE;
```

ID	GENDER	INCOME	INCCLS	MALE
1	1	15000	1	1
1	2	36000	2	0
2	1	47500	2	1
2	2	22000	1	0
3	1	51000	3	1
3	2	77000	4	0

Getting Character Variables to Align

When I create character variables, I want them to line-up in some logical order, not alphabetical order. I do this by leaving leading blanks in the variable values. Then when the procedure displays the values in order, the order is determined by the number of blanks. Note in the code that there are no blanks in front of the value

"missing", so it will always be last. The Proc Print shows that this worked.

```
DATA TEMP; SET TEMP;

ALPHAINC="MISSING
";
     IF INCCLS = 1 THEN
ALPHAINC="     LESS THAN
$25,000";
ELSE IF INCCLS = 2 THEN
ALPHAINC="    $25,000 TO
$49,999";
ELSE IF INCCLS = 3 THEN
ALPHAINC="   $50,000 TO $74,999
";
ELSE IF INCCLS = 4 THEN
ALPHAINC="  $75,000 TO $99,000
";
                    ELSE
ALPHAINC=" $100,000 AND UP
";
RUN;

ODS RTF FILE='C:\SESUG_2.DOC';
PROC FREQ DATA=TEMP;
TABLES ALPHAINC;
RUN;
ODS RTF CLOSE;
```

ALPHAINC	Freq	Pct	Cum Freq	Cum Pct
LESS THAN $25,000	2	33.33	2	33.33
$25,000 TO $49,999	2	33.33	4	66.67
$50,000 TO $74,999	1	16.67	5	83.33
$75,000 TO $99,000	1	16.67	6	100.00

Getting Totals the Easy Way

You've got to learn Proc SQL®. There are some things you can do in SQL that are just painful in base SAS®. This paper will not teach you SQL, but, again, you can copy this little program to your program window and just change the variable and dataset names to fit your needs.

Find in the code where it says "SUM(INCOME) AS TOTAL ". This tells Proc SQL to sum the variable income and name the result TOTAL (and gives it the dollar9. format). Next, I create another variable: (100*INCOME/CALCULATED TOTAL) and names it PCT (with a 5.2 format). Notice what happens. The first pass of the data accumulates TOTAL. This is then remerged back onto the dataset. Then PCT is calculated. Defining PCT as being created with a CALCULATED variable tells SQL not to try to create PCT before the value of TOTAL is created. The PROC PRINT shows the values in their formatted form.

```
PROC SQL;
CREATE TABLE TEMP AS
  SELECT *, SUM(INCOME) AS TOTAL
FORMAT DOLLAR9.,
  (100*INCOME/CALCULATED TOTAL)
AS TOTAL FORMAT=5.2 AS PCT
  FROM TEMP;
  QUIT;
ODS RTF FILE='C:\SESUG_3.DOC';
PROC PRINT; RUN;
ODS RTF CLOSE;
```

ID	GENDER	INCOME	INCCLS	MALE	ALPHAINC	TOTAL	PCT
1	1	15000	1	1	LESS THAN $25,000	$248,500	6.04
1	2	36000	2	0	$25,000 TO $49,999	$248,500	14.49
2	1	47500	2	1	$25,000 TO $49,999	$248,500	19.11
2	2	22000	1	0	LESS THAN $25,000	$248,500	8.85

ID	GENDER	INCOME	INCCLS	MALE	ALPHAINC	TOTAL	PCT
3	1	51000	3	1	$50,000 TO $74,999	$248,500	20.52
3	2	77000	4	0	$75,000 TO $99,000	$248,500	30.99

Subtotals, too

SQL goes even further, letting you do something that is even more difficult in base SAS: getting sub-totals and group percentages. Note the only changes to the above program (in which the grand total and overall percentages were created) are the names of the new variables and the addition of one statement: "GROUP BY ID;". Now SQL created sub-totals and group percentages for each ID group. Proc Print shows that this worked.

```
PROC SQL;
CREATE TABLE TEMP AS
  SELECT *, SUM(INCOME) AS
SUBTOTAL FORMAT DOLLAR9.,
  (100*INCOME/CALCULATED
SUBTOTAL) AS SUBPCT FORMAT 5.2,
   FROM TEMP
  GROUP BY ID;
  TITLE "GETTING SUB-TOTALS AND
PERCENTS OF SUB-TOTALS";

QUIT;
ODS RTF FILE='C:\SESUG_4.DOC';
PROC PRINT; RUN;
ODS RTF CLOSE;
```

ID	GENDER	INCOME	INCCLS	MALE	ALPHAINC	TOTAL	PCT	SUB-TOTAL	SUB-PCT	PCT-TOT
1	2	36000	2	0	$25,000 TO $49,999	$248,500	14.49	$51,000	70.59	20.52
1	1	15000	1	1	LESS THAN $25,000	$248,500	6.04	$51,000	29.41	20.52
2	1	47500	2	1	$25,000 TO $49,999	$248,500	19.11	$69,500	68.35	27.97
2	2	22000	1	0	LESS THAN $25,000	$248,500	8.85	$69,500	31.65	27.97
3	2	77000	4	0	$75,000 TO $99,000	$248,500	30.99	$128,000	60.16	51.51
3	1	51000	3	1	$50,000 TO $74,999	$248,500	20.52	$128,000	39.84	51.51

Array Basics

I teach in the graduate school of Georgetown University supervising original public policy research. I wish my students, otherwise very smart people, knew how to work with arrays. Some work with datasets that have hundreds of variables and then do something, like changing a "-9" to a "." in hundreds of lines of hard code (If VAR1=-9 then VAR1=.; …IF VAR200=-9 then VAR200=.;). Arrays let you do the same things to a list if variables.

To demonstrate this, I create a little 3-observation dataset that has 3 variables, X1-X3. I then use arrays to create three sets of new variables. Note that the array name is not the variable name. That is, the array ETHEL creates the variables Y1-Y3. The

array processing code refers to the array names but the program creates the named variables. In my code, FRED sets up the initial array with the existing variables X1-X3. Ethel creates Y1-Y3 as FRED*10, so when Fred is X1, ETHEL is Y1, so Y1=X1*10. Likewise, LUCY is ETHYL*10 (and is therefore FRED*100). Unlike SQL, you can refer to a created variable as it is created. Note the array RICKY. It uses the binomial code used in the first example in this paper. When the remainder (MOD function) of dividing FRED by 2 is 0, RICKY (variables A1-A3) have a value of 1. That is, when FRED is an even number (or 0), RICKY = 1.

Proc Print shows the values of the variables. Again, note that the variable names are not the array names.

```
DATA TEMP2;
INPUT @1 (X1-X3) (3*2.);
CARDS;
1 2 3
0 1 2
2 3 4
;
RUN;
DATA TEMP2; SET TEMP2;
ARRAY FRED X1-X3;
ARRAY ETHEL Y1-Y3;
ARRAY LUCY Z1-Z3;
ARRAY RICKY A1-A3;
DO OVER FRED;
  ETHEL=FRED*10;
  LUCY=ETHEL*10;
  RICKY=(MOD(FRED,2)=0);
END;
RUN;
ODS RTF FILE='C:\SESUG_5.DOC';
PROC PRINT;
TITLE "ARRAY BASICS";
RUN;
ODS RTF CLOSE;
```

Obs	X1	X2	X3	Y1	Y2	Y3	Z1	Z2	Z3	A1	A2	A3
1	1	2	3	10	20	30	100	200	300	0	1	0
2	0	1	2	0	10	20	0	100	200	1	0	1
3	2	3	4	20	30	40	200	300	400	1	0	1

Random Number Functions

There are just times when I want to test something and don't want to use real data. For example, if there is to much variability in my data to make is useful for a test. To get around this, I use the random number functions. In this example, I use the RANUNI® function which is a random number between 0 and 1 with a uniform distribution. That is, any value between 0 and 1 is equally as likely as any other number. I use the binomial shortcut to randomly assign the values of MALE and COLLEGE to each of the five observations I'm creating. For MALE, there is a 50% chance that any one observation will get a value of 1. For COLLEGE, there is a 75% chance that COLLEGE will be set to 1. I then use these values to create a value for the variable INCOME. Proc Print shows the results. Note that 50% of the observations are not MALE (which would have been impossible with an odd number of obsertations anyway) and that only 40% of the observations got a value of 1 for COLLEGE. There is nothing wrong here. It's just the luck of the draw, so to speak. If we'd done this with a larger number of observations, the percentages would approach 50% for MALE and 75% for COLLEGE. You can also look at the INCOME variable and see how the value of INCOME was influenced by the values of MALE and COLLEGE.

```
DATA TEMP3;
DO I = 1 TO 5;

  MALE=(RANUNI(0) LT .5);
  COLLEGE=(RANUNI(0) LT .75);

INCOME=(100000+(10000*COLLEGE)+(
5000*MALE));
```

```
  OUTPUT;
END;
RUN;

ODS RTF FILE='C:\SESUG_6.DOC';
PROC PRINT; RUN;
ODS RTF CLOSE;
```

I	MALE	COLLEGE	INCOME
1	1	0	105000
2	0	0	100000
3	1	0	105000
4	1	1	115000
5	1	1	115000

TAKING A RANDOM SAMPLE

Another useful way to use a random number function is making a random sample of your data. If you are working with a large dataset, you should consider writing and testing your code on a small sample of your data. It'll be faster and easier to understand. With this code, I take a random 10% sample. There is no output, just the code.

```
DATA TEMP; SET SASDATA.OLD;
IF RANUNI(0) LE .1 THEN OUTPUT;
/*A 10% RANDOM SAMPLE*/
RUN;
```

Proc Means Output

You've just got to get used to working with Proc Means® or Proc Summary®. The procedures are identical except that in Proc Means the default is to create output while Proc Summary does not. I use Proc Means with the NOPRINT option to suppress the output as my interest is in creating an output dataset that has my results.

I use this methodology more than any other single SAS procedure. It is the bast way I know to see how many records I have for my sub-groups and to visually (not statistically) examine how they differ.

Using the same 5-observation dataset just created, I tell the procedure to analyze INCOME (VAR INCOME;) by the combinations of MALE and COLLEGE (CLASS MALE COLLEGE;). I instruct the procedure to create output (OUTPUT) that consists of an output dataset named STATS (OUT=STATS) built using 3 statistics (N= MEAN= SUM=). The AUTONAME option tells SAS to name the variables in the output dataset as a combination of the variable name (INCOME) and the statistic name (N, MEAN, SUM). Therefore, in my output dataset I'll have a variable named, for example, INCOME_MEAN.

The important thing, however, is to learn now to read to output, which is shown from Proc Print.

Look first at the MALE column. The values are ".", '1', and '2'. In this case, "." does not mean missing, it means "ignored". The same us true of the values in the COLLEGE column. Therefore, the first row of the output says, "when you ignore the value MALE and you ignore the value of COLLEGE, the mean income (INCOME_MEAN) is $10,800. The second line shows that the mean income is $10,333 then COLLEGE = 0 and the values of MALE are ignored, while the third row shows an average of $11,500 when COLLEGE=1 and the values of MALE are ignored. I now have the number of records with non-missing values (INCOME_N), the average income (IMCOME_MEAN) and the total (INCOME_SUM) for all possible combinations of my classifying variables.

```
PROC MEANS DATA=TEMP3 NOPRINT;
VAR INCOME;
CLASS MALE COLLEGE;
OUTPUT OUT=STATS N= MEAN= SUM= /
AUTONAME;
RUN;

ODS RTF FILE='C:\SESUG_7.DOC';
PROC PRINT; RUN;
ODS RTF CLOSE;
```

MALE	COLLEGE	_TYPE_	_FREQ_	INCOME_N	INCOME_Mean	INCOME_Sum
.	.	0	5	5	108000.00	540000
.	0	1	3	3	103333.33	310000
.	1	1	2	2	115000.00	230000
0	.	2	1	1	100000.00	100000
1	.	2	4	4	110000.00	440000
0	0	3	1	1	100000.00	100000
1	0	3	2	2	105000.00	210000
1	1	3	2	2	115000.00	230000

Putting it All Together

Now let's see how this all falls together. First, I'll create a yes-no format with blanks in the format so that the output always stays in the right order: yes, no, missing. As before, notice the blanks in the format values.

```
PROC FORMAT;
VALUE YNFMT
      1 = "  YES"
      0 = " NO"
  OTHER = "MISSING";

RUN;
```

Then some data, this time 5,000 observations with some missing data. I get about half of the observations to be male, then change the value of MALE to missing for about 10% of the observations. About 70% of the observations are assigned a 1 for college, with about 10% given a missing value (but a different 10%). Note that the calls to RANNUI have different forms, but they both do the same thing: 10% are assigned a missing value for MALE and 10% for COLLEGE. Again, not the same 10% are given a missing value for MALE and COLLEGE, but 10% overall. INCOME is created using the RANNOR function, a random distribution function (the bell-shaped curve). If the person went to college (COLLEGE = 1) then their income is increased by 10%. If they are female (MALE = 0) they get another 10% boost in their incomes.

```
DATA ALOT;
DO I = 1 TO 5000;
MALE=(RANUNI(0) LE .5);
IF RANUNI(0) LE .1 THEN MALE=.;
COLLEGE=(RANUNI(0) LE .7);
IF RANUNI(0) GE .9 THEN
COLLEGE=.;
INCOME =
100000+(RANNOR(0)*10000);
IF COLLEGE=1 THEN
INCOME=INCOME*1.1;
IF MALE=0 THEN
INCOME=INCOME*1.1;
OUTPUT;
END;
RUN;
```

Using Proc Tabulate®

Now we're ready do look at how we can use the formats to improve the look and readability of a Proc Tabulate run. First, the basics. The Proc Tabulate will examine INCOME within the values of MALE and COLLEGE. Notice how Proc Tabulate arranges the table.

```
ODS RTF FILE='C:\SESUG_8.DOC';
PROC TABULATE DATA=ALOT;
CLASS MALE COLLEGE;
VAR INCOME;
TABLES (MALE ALL),(COLLEGE
ALL)*INCOME*(N*F=COMMA5.
MEAN*F=DOLLAR9.);
RUN;
ODS RTF CLOSE;
```

	COLLEGE					
	0		1		All	
	INCOME		INCOME		INCOME	
	N	Mean	N	Mean	N	Mean
MALE						
0	637	$109,790	1,494	$121,016	2,131	$117,661
1	598	$100,019	1,381	$110,142	1,979	$107,083
All	1,235	$105,059	2,875	$115,793	4,110	$112,567

The first thing to notice is that the missing values have not been used. This may or may not be what you want. Second, notice the values of MALE and COLLEGE are data values (0 - 1), not Yes-No. We can fix both of these.

To get missing values into the table, I add the MISSING option to the call to Proc Tabulate. I also use the YNFMT to format the MALE and COLLEGE variables. The new table is closer to what I want.

```
ODS RTF FILE='C:\SESUG_9.DOC';
PROC TABULATE DATA=ALOT MISSING;
CLASS MALE COLLEGE;
VAR INCOME;
TABLES (MALE ALL),(COLLEGE
ALL)*INCOME*(N*F=COMMA5.
MEAN*F=DOLLAR9.);
FORMAT MALE COLLEGE YNFMT.;
RUN;
ODS RTF CLOSE;
```

	COLLEGE							
	MISSING		NO		YES		All	
	INCOME		INCOME		INCOME		INCOME	
	N	Mean	N	Mean	N	Mean	N	Mean
MALE								
MISSING	42	$102,949	120	$100,259	304	$109,633	466	$106,617
NO	225	$110,420	637	$109,790	1,494	$121,016	2,356	$116,969
YES	199	$100,560	598	$100,019	1,381	$110,142	2,178	$106,487
All	466	$105,536	1,355	$104,634	3,179	$115,204	5,000	$111,438

However, the rows and column values still appear in the wrong order. What I want is to have Yes first, No second and Missing last. To do this all I have to do is add an ORDER=FORMATTED statement to the call to Proc Tabulate, keeping the missing option.

```
ODS RTF FILE='C:\SESUG_10.DOC';
PROC TABULATE DATA=ALOT
ORDER=FORMATTED MISSING;
CLASS MALE COLLEGE;
VAR INCOME;
TABLES (MALE ALL),(COLLEGE
ALL)*INCOME*(N*F=COMMA5.
MEAN*F=DOLLAR9.);
FORMAT MALE COLLEGE YNFMT.;
TITLE "WITH 'ORDER=FORMATTED'
OPTION";
RUN;
ODS RTF CLOSE;
```

	COLLEGE							
	YES		NO		MISSING		All	
	INCOME		INCOME		INCOME		INCOME	
	N	Mean	N	Mean	N	Mean	N	Mean
MALE								
YES	1,381	$110,142	598	$100,019	199	$100,560	2,178	$106,487
NO	1,494	$121,016	637	$109,790	225	$110,420	2,356	$116,969
MISSING	304	$109,633	120	$100,259	42	$102,949	466	$106,617
All	3,179	$115,204	1,355	$104,634	466	$105,536	5,000	$111,438

There it is. The values of MALE and COLLEGE are in the desired order. If I run this program over again with a different set of variables, they will still be in this order. That means I can learn to read this table once and it will always be read the same way.

ODS

Did you notice I was using the Output Delivery SYSTEM® for all the procedures to create the output for this paper? It really is as simple as it looks if you're willing to take the SAS defaults and do a little editing in Word. The call to ODS is straight forward:

```
ODS RTF FILE='C:\SESUG_10.DOC';
```

I open a Rich Text File (RTF) named with a "DOC" extension so it can be read right into Microsoft® Word®. After I run the procedure, I close ODS:

```
ODS RTF CLOSE;
```

All I have to do after that is open the document in Word and copy the table into the right place in my text. For this paper, because of the column widths, I did have to edit come of the tables, but that's pretty easy to do. Not only does ODS make your output look better, it also eliminates the time and errors in creating a table in Word from scratch.

A Little Macro

Now that we've done all this work, wouldn't it be nice if we could automate this process of creating formatted Word output for any combination of variables. I accomplish that using just a little of the SAS Macro® language.

First, I call the macro with a name and the input variables. This tells SAS to expect two variable names when I call the macro TWOVARS.

```
%MACRO TWOVARS(VAR1= , VAR2=);
```

I can now wrote my code using macro variables in the few places they appear. First, I'll use them in the ODS file name:

```
ODS RFT
FILE=C:\&VAR1._&VAR2._TAB.DOC
```

When I run the macro TWOVARS, the macro variable names &VAR1 and &VAR2 will be replaced by the variable names used in the call. For example:

```
MACRO TWOVARS(VAR1=MALE,
VAR2=COLLEGE)
```

Resolves to the following ODS file name upon execution:

```
ODS RFT
FILE=C:\MALE_COLLEGE_TAB.DOC
```

Next, I'll put my macro names into the Proc Tabulate code:

```
PROC TABULATE DATA=ALOT
ORDER=FORMATTED MISSING;
CLASS &VAR1 &VAR2;
VAR INCOME;
TABLES (&VAR1 ALL),(&VAR2
ALL)*INCOME*(N*F=COMMA5.
MEAN*F=DOLLAR9.);
FORMAT &VAR1 &VAR2 YNFMT.;
RUN;
ODS RTF CLOSE;
%MEND TWOVARS;
```

With that done, I first have to compile the Marco, a step that creates no output. That is, I execute the job from the first Macro statement %MARCO TWOVARS… to the %MEND TWOVARS statement. Once compiled, all I have to do to run Macro is call it:

MACRO TWOVARS(VAR1=MALE, VAR2=COLLEGE)

Some things about Macro that may not be obvious:

- There is no ";" after the call to the macro.

  MACRO TWOVARS(VAR1=MALE, VAR2=COLLEGE)

- When you use a macro variable in a name such as the file name in the ODS statement, you have to put a dot after the macro variable name so SAS knows where it ends. The dot is part of the macro name and therefore is not kept after it resolves.
- ODS RFT FILE=C:\&VAR1._&VAR2._TAB.DOC

  ☝ ☝

  resloves to:

  ODS RFT FILE=C:\MALE_COLLEGE_TAB.DO

And that's it for this time. I hope you found something useful here and that you see the value in attending meetings like this one. There are all kinds of tricks and shortcuts, so make an effort to learn a few. If you program like I do -- all the time -- then being a little lazy is a good thing.

About the Author
Chris Toppe is Director of Philanthropic Studies at Independent Sector. In this role he manages the collection, analysis, and reporting of data on the charitable activities of Americans. His focus is on uncovering facts and findings that are useful to practitioners and policy makers. Chris also teaches in the graduate school of Georgetown University where he supervises original research for graduate students enrolled in the Georgetown Public Policy Institute. Chris has been using SAS since the 1980s and has presented papers at over two dozen local, regional, and SUGI conferences. Contact Information: Chris Toppe, Ph.D. Director, Philanthropic Studies, Independent Sector, 1200 18th Street, NW, Suite 200, Washington, DC 20036 202.467.6115 (office) 202.467.6101 (fax) chris@IndependentSector.org.