Super Simple Macros to Make a Statistician’s Life Easier
Peter L. Flom, Peter Flom Consulting, LLC

ABSTRACT
Macros can be a very complex topic, but some very simple macros can make life easier for a data analyst or statistician. I give a very basic introduction to macros from the perspective of a data analyst, and present some macros I have found useful. I include only certain types of macros, deliberately choosing the options I find easiest to understand and use. Again, this is a paper intended for statisticians and data analysts, not programmers. I am following the KISS principle: Keep It Simple, Statistician!

Keywords: Macros Simple Statistics.

INTRODUCTION
I am a data analyst/statistician, not a programmer. I’ve been using SAS® for over 20 years. Mostly I use SAS STAT. I’ve only started realizing how useful macros can be in the last few years. This isn’t because existing material on macros is bad - it isn’t. But it’s mostly written by programmers and for programmers and mostly to do things that have to do with data management rather than statistics. These macros can be very complex. But I just use simple macros. They make my programs shorter, easier to read, and easier to modify.

WHAT IS A MACRO VARIABLE?
A macro variable is a SAS tool to replace text. It must be defined before it can be used. Macro variable names must be between 1 and 32 characters in length, must start with a letter or underscore (_) and can contain letters, numbers or underscores after the first character. There are several ways to define macro variables, but the simplest (and the only one I will use in this paper) is with the %let statement. this has the following syntax:

```sas
%let macrovar = text;
```

For example, if we wanted to create a macrovariable called IV (for independent variable, presumably) and set it equal to ‘sex’ we would use:

```sas
%let IV = sex;
```

We can then use the macro variable just as we would the ordinary text, but prefaced with an ampersand (&) e.g.

```sas
proc glm data = ds;
   model depvar = &IV;
run;
```

If you want to use a macro variable as quoted text, you should use double quotes; single quotes will make the macro variable show up as the name of the macro variable, rather than its code. For example:

```sas
title 'IV = &IV';
proc glm data = ds;
   model depvar = &IV;
run;
```

will have a title ‘IV = &IV’, which is probably not what you want. The correct version is

```sas
title "IV = &IV";
```

See the example, below, as well.

When you want to combine macro variables with other text or other macros (e.g. as prefixes or suffixes), things can get complex, but we will not do that in this paper.
WHAT IS A MACRO?

A macro is a section of code that looks like this:

```
%macro macroname
   macro text
%mend macroname
```

(technically, the macroname is not required on the %mend statement, but it is highly recommended as it makes your code easier to read).

Per Carpneter(2004) macro text can include ‘

- Constant text
- Macro variables
- Macro program statements
- Macro expressions
- Macro functions

',

However, we will deal only with the first two in this paper.

We looked at macro variable above, and ‘constant text’ can include (again per Carpenter)

- SAS data set names
- SAS variable names
- SAS statements
- SAS steps (like DATA and PROC steps)
- Complete and partial SAS programs
- Any combination of the above

A macro is invoked with %macroname

Macros can also include parameters, and these can get complex, as well. In this paper, we will only use what are known as keyword parameters. Keyword parameters can have default values, but we will not use these. Parameters are introduced in parentheses after the macroname, e.g.

```
%macroname(par1=);
```

This will be clearer in the example.

WHAT MAKES A GOOD MACRO?

What makes a good macro? This question will have different answers for different people. For me, a good macro is

1. Easy to write
2. Quick to write - often only a few minutes
3. Easy to understand
4. Easy to debug - sometimes, they work the first time!
5. Disposable. I don’t have complex macros that work in lots of situations, I have simple macros that work quickly in one situation.
EXAMPLE: MANY REGRESSIONS

THE PROBLEM
Suppose you need to run a large number of bivariate regressions. Some of the independent variables are categorical, some are not. You could write a lot of PROC GLM statements, each along these lines:

```
proc glm data = mydata;
  model dv = iv1;
run;
```

but some will need a CLASS statement:

```
proc glm data = mydata;
  class iv2;
  model dv = iv2;
run;
```

So, if you had 30 such regressions, you would have at least 90 lines of code. But then your boss, colleague, co-author or editor says you need to control for three variables in all those regressions; you could cut and paste those 90 lines, then do some sort of replace to make each one something like:

```
proc glm data = mydata;
  model dv = iv1 cv1 cv2 cv3;
run;
```

but now you have 180 lines of code. Then someone decides they need some graphical output. Then another person wants to look at some of these only without one of the control variables. Oy.

A MACRO VARIABLE SOLUTION
Using a macro variable approach, we could do something like this:

```
%let iv = iv1;
%let iv = iv1 |
title "IV = &iv"; /*Note the double quotes*/
proc glm data = mydata;
  model dv = &iv;
run;
```

but this saves us relatively little in this case. We would have to type in a %let statement for each IV. We would then save the repetition of the PROC GLM statement, but the result is a bit awkward, since the %let statements could not all be adjacent to the RUN statement.

Also, this does not deal with the need for CLASS statements for categorical IVs. Nor does this solution help much with the modifications that may be required to the program.

One problem with this program that is easy to fix is that the output will also be less than clear, because there is no title.

```
%let iv = iv1;
  title "IV = &iv"; /*Note the double quotes*/
proc glm data = mydata;
  model dv = &iv;
run;
```

THE MACRO SOLUTION
Here is one macro solution. We will start with two macros, one for continuous IVs, one for categorical ones.

```
%macro regcont(IV=) /*This starts the macro */
  title "IV = &IV"; /*Note the double quotes*/
proc glm data = mydata;
  model dv = &iv;
%mend regcont /*This ends the macro */
```
and

%macro regcat(IV=) /*This starts the macro */
  title "IV = &IV";
  class &IV;
  proc glm data = mydata;
    model dv = &iv cv1 cv2 cv3;
%mend regcat /*This ends the macro */

To use these, we would run %macrocont(iv1); %macrocat(iv2); and so on. Now the macro is one place, and the calls to the macro are in another. If we wish to modify the macro, we can create new ones and run them. For example, if we want to add control variables to the continuous macro case we could do this:

%macro regcontCV(IV=)
  title "IV = &IV, with control variables"; /*change the title to add clarity*/
  proc glm data = mydata;
    model dv = &iv cv1 cv2 cv3 ;
%mend regcontCV

and make parallel changes to the categorical version, then run

%macrocontCV(iv1); %macrocatCV(iv2);

and so on.

A more complex macro approach would have options in one macro for various things; it is possible to write a macro that would take care of continuous vs. categorical variables, and control variables being included or not, and other things, all in one macro. You could also have a parameter for a dataset and a dependent variable. I prefer not to do this. It makes the macro harder to write, harder to read, much harder to debug, and, no matter how many options you add, you will never have all the options you want. At the extreme, you would wind up creating something almost as complex to use as the PROC you wanted to implement. But there is always a tradeoff between complexity and versatility, and you can modify my approach. For instance, if you have to run identical regressions on many different datasets (perhaps a new dataset each day or week) then it would make sense to include a parameter for dataset. Here is an example from PROC MIXED, where I had to do very similar analysis for 5 different IVs and two DVs:

%macro mixfinal(dv=, iv=);
  ods select LRT SolutionF Tests3;
  title 'Random intercepts and slopes';
  title2 'AR(1) covariances';
  title3 "&dv and &IV analysis";
  proc mixed data = schooler.long ;
    class pid &IV waveclass ;
    model &dv = &IV months &IV*months/solution;
    id pid wave;
    random intercept months/subject = pid type = un ;
    repeated waveclass/type = ar(1) subject = pid;
  run;
%mend mixfinal;

Which is called by, e.g., %mixcheck(anxdep, ITT);

SUMMARY

Even very simple macros can make your programs shorter, easier to read and easier to modify.

BIBLIOGRAPHY

ACKNOWLEDGMENTS

I would like to thank Ron Fehd for providing help with \LaTeX.

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 names and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION

Peter L. Flom
Peter Flom Consulting, LLC
5 Penn Plaza
Room 2342
New York, NY 10001
Phone: (917) 488-7176
peterflomconsulting@mindspring.com
Personal webpage: http://www.statisticalanalysisconsulting.com/