SAS Macros

Luke & Daiva

2015-03-11

Cheatsheet: Using SAS macros

SAS has a powerful feature know as the macro language. If you have repetitive code, or a particular analysis that is fairly complex, macros are there to make your life easier! Below are some basic things to remember and to know for using macros. Luke also has a brief intro/tutorial on writing your own macros at his blog. Luke also has developed a personal macro library on GitHub that is fairly well documented, so you can look them over if you want. Maybe you will find something that suits your own analysis!

SAS macro commands:

```
%macro name (arg1, arg2=);
```

> This is the command format you would use to start a macro. An example is shown in the "Example" section at the bottom.

- `%macro` part tells SAS that the upcoming code is a macro.
- `name` is the name you would give your macro, for example `means` or `corr` or `regression` and so on.
- `arg1` and `arg2` are known as arguments. They are used to include other variables within the macro. A better explanation is below in the example section.
- `arg1` is known as a positional argument (or parameter) because it has no `=` sign after it. A positional argument means that what ever variable is first supplied to the macro takes on the value of `arg1`. For example, for a macro such as `%macro means(vars, where=);`, when you call the macro `%means(Height, where = Wgt < 100);` the 'Height' variable takes on the value of `arg1` because it is first.
- `arg2=` is known as a keyword argument (or parameter) because of the `=` sign. Thus, in order to use this argument, you need to specifically call it. For example, `%means(Height, where = Wgt < 100);` the `where` argument needs to be called directly, while the `vars` argument is replaced by 'Height' because it is a positional argument.

```
%mend name;
```

This ends the macro definition (`mend` = macro end). So to end the `%macro means();` example, you use `%mend means;`. See the example below.

```
%let variable = something;
```

This is known as a macro variable. The `%let` statement is kind of like telling SAS to create a jar. You name this jar as 'variable' and inside the jar you place 'something'. This can be very useful when you have a long list of variables that you repeated use. For example, `%let jar = BMI Wgt Hgt Age;`. 'jar' now contains these 4 variables, which can be called using `&jar` (see below).

```
&variable
```

This is also known as a macro variable. However, unlike the `%let`, here you are not creating a macro variable, but rather telling SAS use the contents of the macro variable. Continuing with from the example directly above, `&jar` is replaced with 'BMI Wgt Hgt Age' before SAS processes the `proc` or `data` command.

```
%if ... %then ...;
```

This is known as a conditional. This is a fairly advanced component of macros, but is really where using macros really starts to shine.

```
%do i = 1 %to something;
```

This is known as a 'do loop'. Like the `%if ... %then ...;` above, this is an advanced but *extremely* powerful feature of macros that lets you do some very impressive things!

Example

We want to create a macro for calculating means, than running it on some some data. This is real code that can be run, so try it out on your own!

```
%macro means(vars, where=, class=, data=);
    proc means data=&data;
        var &vars;
        where &where;
        class &class;
        run;
    %mend means;
```

```
%let length = Length1 Length2 Length3;
%let others = Weight Height Width;

%means(&length, where = Weight < 200,
    class = Species, data = sashelp.fish);

%means(&others, class = Species,
    data = sashelp.fish);
```

Lets break this down:

```
%macro means(vars, where=, class=, data=);
```

This creates a macro called `means` that has 4 arguments, 1 of which is positional (`vars`).

```
proc means data=&data;
    var &vars;
    where &where;
    class &class;
run;
```

This is the meat of the macro. Using the ampersand `&`, we can place the arguments at various places throughout the macro. When SAS runs this code, `&data` will be replaced by what ever you put into it, and so on.

```
%mend means;
```

This tells SAS that your own custom macro is finished.

```
%let length = Length1 Length2 Length3;
%let others = Weight Height Width;
```

These two commands are macro variables. Basically, we are creating two 'jars' here, named 'length' and 'others'. Each 'jar' contains 3 variables.

```
%means(&length, where = Weight < 200,
    class = Species, data = sashelp.fish);
```

This is where we actually invoke the macro `means` that we created. Because `vars` was a positional argument, we don't have to call it directly (ie: `vars = &length`). Just putting `&length` in the first position tells SAS what the variable is. Because the other 3 arguments were keyword arguments, they have to be explicitly called (eg: `where =`).

This is a *very* basic example. They can get fairly complex, but *very* powerful as you add more components to the macro. Anytime you have repetitive or complex code, create a macro and recycle your code. This saves an incredible amount of time and headache!