Chapter 7: Macros in SAS

- Macros provide for more flexible programming in SAS.
- Macros make SAS more “object-oriented,” like R.
- In some ways macros serve similar purposes as functions in R.
  1. Can change a piece of a program and have that change be reflected throughout the program.
  2. Can write repeatable code that can be used in many places efficiently.
  3. Can allow programs to be dependent on data values that may be unknown at the time the program is written.

- Macro statements consist of special code that SAS interprets and translates to standard SAS code before executing the program (this is done internally).
- *Macros* are *subprograms* that are placed in key positions in the main SAS program simply by specifying the name of the macro in a special way.
- *Macro variables* are similar to SAS variables, but do not belong to a data set. They are typically substituted into the program in key places.

- Macros are specified with a % prefix:

```
%macroname;
```

- Macro variables are specified with a & prefix:

```
&mymacrovar
```

- If a macro variable is defined in a macro, it is *local* — it can only be used in that macro.
- If a macro variable is defined in "open code" (outside any macros), it is *global* and may be used anywhere.

Defining Simple Macro Variables

- The simplest way to assign a value to a Macro variable is using a %LET statement.

```
%LET macrovarname = value;
%LET mypower = 3;
%LET powerword = cube;
Y = X**&mypower;
TITLE "Taking the &powerword of each X value";
```

(Note: Double quotes are needed, because SAS will not find macros inside single quotes.)

- Note that the macro variables are referenced with an ampersand symbol. When SAS sees `&macrovarname` it will replace this with the value that was assigned to `macrovarname` in the `%LET` statement.
- This way, if the value needs to be changed, we can change it once (in the `%LET` statement) rather than everywhere the variable is used.
- To make a macro variable *global*, simply define it (with a `%LET` statement) *outside of* a macro.

Macros as a repeatable set of SAS statements

- One important use of macros is packaging a piece of SAS code to be placed in various places in the SAS program.

  A macro has the syntax:

  ```
  %MACRO macroname;
  ```

  . . . Various

  SAS statements . . .

  ```
  %MEND macroname;
  ```

- This set of SAS statements that lies between the `%MACRO` and `%MEND` lines can be placed anywhere in a program by "invoking" the macro:

  ```
  %macroname
  ```

- This serves as a substitute for rewriting all those statements multiple times.

Macros with Parameters

- Macros that, when invoked, simply repeat the same statements can be helpful but not too flexible.
- Using parameters can add flexibility to macros.
- Parameters are macro variables whose values are set each time the macro is invoked.
- Parameters of macros are similar to arguments of R functions.

```
%MACRO macroname (param1 = _____, param2 = _____);
```

. . . Various

SAS statements . . .

```
%MEND macroname;
```

- When invoking the macro, we also provide the values of the parameters.

```
%macroname(param1 = 7.5, param2 = January)
```

- Within the macro code, the parameter name is referred to with an ampersand:

```
Month = &param2;
```

Conditional Logic in Macros

- Macro equivalent of conditional statements exists: `%IF, %THEN, %ELSE, %DO, %END.`
- These are used in similar fashion as non-macro equivalents, except:
  1. They can only be used within a macro.
  2. They can contain entire `DATA` steps or `PROC` steps, unlike regular `IF-THEN` statements which must appear within a `DATA` step.
- Automatic macro variables such as

  `&SYSDATE` ← gives current date in character form

  `&SYSDAY` ← gives current day of week

  can be useful (with conditional logic) to run a SAS program that should do different tasks depending on the date or on the day of the week.

Using CALL SYMPUT to Assign a Value to a Macro Variable

The `CALL SYMPUT` statement assigns a value to a macro variable, typically on the basis of data read in a previous `DATA` step.

```
CALL SYMPUT("macrovarname", value);
```

1. Read in variables (say, `var1, var2,` etc.) in a `DATA` step.
2. In a second `DATA` step, use `CALL SYMPUT`:

   Example:
   ```
   IF var1 > var2 THEN
   CALL SYMPUT("comparison", "greater");
   ELSE CALL SYMPUT("comparison", "less");
   ```
   or
   ```
   IF var1 > var2 THEN
   CALL SYMPUT("comparison", var3);
   ```
3. Invoke the macro variable (`&comparison`) in a separate step (cannot invoke in same `DATA` step).

Looping in SAS: `DO WHILE` and `DO UNTIL`

- These loops execute a statement or set of statements while a logical condition is met (or until it is not met).
- Example syntax:

```
DO WHILE (logical condition in parentheses);
Some SAS statements;
END;
```

- `DO WHILE` checks the condition at the *beginning* of each iteration of the loop.
- `DO UNTIL` checks the condition at the *end* of each iteration of the loop (so it always runs at least one iteration).