SAS宏

上海财经大学统计与管理学院

目 录

宏变量

SAS的宏语言是一个非常通用的和有用的工具。它经常被用于降低定期的SAS代码的量和它有利于把信息从一个程序传递到另一个程序。

此外，我们可以用它来编写“动态”和灵活的SAS程序。一般情况下，我们认为的宏语言是宏变量和宏程序的组成。在本次讨论中，我们将演示如何创建宏变量，以及如何编写基本的宏方案。

宏变量

在SAS中宏变量是一个字符串变量，允许您通过符号替换修改一个SAS程序中的文本。下面的示例演示如何创建和使用宏变量。首先，我们建立了一些系统选项以便有一个更简洁的输出格式。

```
options nodate nonumber nocenter formdlim="-";
data hsb2;
  input   id female race ses prog
          read write math science socst;
datalines;
 70  0  4  1  1  57  52  41  47  57
121  1  4  2  3  68  59  53  63  61
 86  0  4  3  1  44  33  54  58  31
```

（ch09_01.sas）

```
141  0  4  3  3  63  44  47  53  56
172  0  4  2  2  47  52  57  53  61
113  1  4  2  2  44  52  51  63  61
 50  0  3  2  1  50  59  42  53  61
 11  0  1  2  2  34  46  45  39  36
 84  0  4  2  1  63  57  54  51  63
 48  1  3  2  2  57  55  52  50  51
 75  1  4  2  3  60  46  51  53  61
 60  1  4  2  2  57  65  51  63  61
 95  0  4  3  2  73  60  71  61  71
104  0  4  3  2  54  63  57  55  46
 38  0  3  1  2  45  57  50  31  56
115  0  4  1  1  42  49  43  50  56
 76  0  4  3  2  47  52  51  50  56
195  0  4  2  1  57  57  60  56  52
;
run;
```

（ch09_01.sas）

SAS语言

假设我们想看看一些变量的均值，然后对相同的变量进行回归分析。

```
proc means data = hsb2;
   var write math female socst;
run;
proc reg data = hsb2;
   model read = write math female socst;
run;
quit;
```

（ch09_01.sas）

宏变量定义%LET语句

我们可以通过创建包含独立变量的所有的名字的宏变量简化该程序。宏变量可以使用%LET语句来创建。在语句中所有与宏变量或者宏程序有关的关键字以%开头；当我们引用宏变量时，它由&开头。当我们提交我们的程序，SAS首先会处理宏变量，用它们被定义成的字符串替换它们，然后处理该程序作为标准SAS程序。

```
%let indvars = write math female socst;
proc means data = hsb2;
   var &indvars;
run;
proc reg data = hsb2;
   model read = &indvars;
run;
quit;
```

（ch09_01.sas）

%LET语句应用例

产生n条来自1到100的整数随机数，n由用户指定。

```
%LET n=3;
DATA generate;
  do subj=1 to &n;
    x=int(100*ranuni(0)+1);
      output;
  end;
run;
title "Data Set with &n Random Numbers";
proc print data=generate noobs;
run;
```

结果

Data Set with 3 Random Numbers

subj	x
1	46
2	32
3	44

%PUT语句

我们可以用%PUT语句显示宏变量值在日志窗口中的文字。

```
%put my first macro variable indvars is &indvars;
```

日志窗口输出：

```
my first macro variable indvars is write math female socst
```

自动宏变量

SAS拥有许多系统定义的宏变量。这些宏变量在SAS启动时自动创建。因此，它们有时被称为自动宏变量。我们可以用%PUT语句再次显示这些系统定义的宏变量的值。

```
%put _automatic_;
```

部分输出结果

```
AUTOMATIC AFDSID 0
AUTOMATIC AFDSNAME
AUTOMATIC AFLIB
AUTOMATIC AFSTR1
AUTOMATIC AFSTR2
AUTOMATIC FSPBDV
AUTOMATIC SYSADDRBITS 64
AUTOMATIC SYSBUFFR
AUTOMATIC SYSCC 0
AUTOMATIC SYSCHARWIDTH 1
AUTOMATIC SYSCMD
AUTOMATIC SYSDATE 17DEC14
AUTOMATIC SYSDATE9 17DEC2014
AUTOMATIC SYSDAY Wednesday
```

自动宏变量

Title语句中宏变量引用

这些宏变量可以以与普通宏变量一样的方式被使用。例如，在下面的例子中，我们使用两个在标题语句中系统定义的宏变量。

```
title "today's date is &SYSDATE9 and today is &SYSDAY";
proc means data = hsb2;
   var &indvars;
run;
```

(ch09_01.sas)

输出结果:

today's date is 17DEC2014 and today is Wednesday

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
write	18	53.2222222	7.7273811	33.0000000	65.0000000
math	18	51.6666667	7.1373088	41.0000000	71.0000000
female	18	0.2777778	0.4608886	0	1.0000000
socst	18	55.3888889	9.6536423	31.0000000	71.0000000

自动宏变量

单引号与双引号

请注意，在标题语句中，我们使用的是标题双引号。通常情况下，我们可以使用单引号或双引号。当宏变量被嵌入在标题语句中，只有双引号将工作。下面的例子显示了一些对宏变量使用单引号的时候可能会出现的问题。

```
title 'The date is &SYSDATE9 and today is &SYSDAY';
proc means data = hsb2;
   var &indvars;
run;
```

(ch09_01.sas)

输出结果:

The date is &SYSDATE9 and today is &SYSDAY

The MEANS Procedure

Variable	N	Mean	Std Dev	Minimum	Maximum
write	18	53.2222222	7.7273811	33.0000000	65.0000000
math	18	51.6666667	7.1373088	41.0000000	71.0000000
female	18	0.2777778	0.4608886	0	1.0000000
socst	18	55.3888889	9.6536423	31.0000000	71.0000000

全局与局部宏变量

我们还可以显示由用户定义的宏变量。宏变量indvar，这是之前定义的，是一个用户定义的宏变量的一个例子。由于indvars是由宏程序之外定义的，所以它是默认的全局宏变量。全局宏变量可以在任何的SAS程序或数据的步骤而局部宏变量只能在它被定义的宏程序中使用。

```
%put _user_;
```

输出结果:

```
%put _user_;
GLOBAL INDVARS write math female socst
```

引用宏变量

你可以在程序中任意位置引用宏变量，包括这些特殊的地方：

宏变量引用与前置文本和（或）尾部文本相邻。

text*&variable*

***&variable*text**

text*&variable*text

相邻的宏变量引用：

&variable&variable

宏变量被引用时，SAS无法识别连接其后的文本，需要使用分隔符告知分隔符前的为宏变量部分。使用**句点.**作为宏变量分隔符。

宏变量分隔符应用例

例1

```
%LET prefix=abc;
data &prefix.123;
   x=3;
run;
```

例2

```
%LET libref=sashelp;
proc print data=&libref..class;
   title "Listing of CLASS";
run;
```

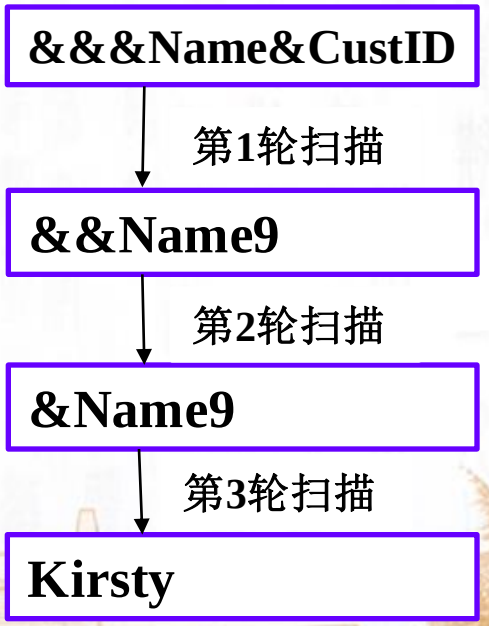
思考：上述宏变量引用其后为何有小数点，为何两例中的小数点数目不一样？如果去掉，是否会报错？为什么？

宏变量间接引用

间接引用宏变量通过多个“&”符号来实现。宏处理器对多个“&”处理规则是：从左到右扫描，若宏变量前仅含有一个“&”，则解析该宏变量；否则将相邻的两个“&”替换成一个“&”，重复上述扫描过程，直到解析过程完成。

例：

```
%LET CustID=9;
%LET Name9=Kirsty;
%PUT &&&Name&CustID;
```



图：对“&”符号的扫描

宏变量练习题

1.（宏变量的删除）通过%SYMDEL实现删除用户自定义宏变量，格式如下：

%SYMDEL 宏变量名1 宏变量名2;

（1）请尝试删除自定义的宏变量；

（2）（**附加题**）如何删除用户自定义的所有宏变量？

2.（**宏变量的显示**）通过%PUT语句在日志窗口显示所有宏变量，并查阅下列宏变量值是什么？

syslast,sysuserid,systime,sysdate9

3. 定义%LET office=Sydney;问下列语句哪些是正确的？

a. where City="&office"; b. where City="Syndey";

c. where City='&office'; d. where City=&office;

一、宏变量小结

1. 用 %LET语句定义宏变量；
2. 用%PUT语句显示宏变量值在日志窗口中的文字。
3. 系统定义的自动宏变量：%put _automatic_;
4. 用户定义的宏变量：%put _user_;
5. 用&解析宏变量
6. 单引号vs. 双引号;
7. 宏变量引用分隔符

宏函数

宏函数满足下列：

有与对应的DATA步字符函数相似的语法
产生相似的结果
处理宏变量和表达式
代表宏触发器
由宏处理器执行
字符常量参数不需要引号

宏字符串处理函数的参数可以是任意文本和（或）宏触发器：固定文本、宏变量引用、宏函数、宏调用。
固定文本参数不需要引号。

宏函数

常用的字符串处理函数：

%UPCASE	转换小写字母为大写字母。
%SUBSTR	从字符串中提取子串。
%SCAN	从字符串中提取一个词。
%INDEX	在字符串中搜索指定文本。

其他函数：

%EVAL	进行算术运算和逻辑运算。
%SYSFUNC	执行SAS函数。
%STR	引用特殊字符
%NRSTR	引用特殊字符，包括宏触发器。

字符串处理宏函数操作

%put &indvars;

结果：write math female socst

%let newind = %upcase(&indvars);

%put &newind;

结果：write math female socst

%let word2 = %scan(&indvars, 2);

%put &word2;

结果：math

%let subword = %substr(&indvars, 5, 3);

%put &subword;

结果：e m

```
%let sub=%substr("abcd",2,1);
%put &sub;
```

思考：sub等于？为什么？

%STR函数

%STR函数屏蔽（删除正常含义）这些特殊符号：+ - * / , < > = ; ' " LT EQ GT LE GE NE AND OR NOT blank

%STR函数的一般形式：

%STR(*argument*)

argument 可以是文本和宏触发器的任意组合。

%STR函数满足下列：

屏蔽符号，引用固定文本

使宏触发器能正常工作

在它的参数中保存前导空格和尾随空格

当在引号或括号前直接加一个百分比符号（%）时，在它的参数中屏蔽单个引号或括号

%STR函数

实例：在宏变量中储存SAS语句。

```
%let statement=%str(title "S&P 500";);
```

SAS日志窗口

```
3    %let statement=%str(title "S&P 500";);
WARNING: Apparent symbolic reference P not resolved.
```

%NRSTR函数

%NRSTR函数除了还可以屏蔽宏触发器以外，它与%STR函数的工作原理相同。

实例：使用%NRSTR阻止宏变量解析。

```
%let statement=%nrstr(title "S&P 500";);
```

宏运算函数

宏函数%EVAL能够进行算术运算和逻辑运算。举例：

```
%let k = 1;
%let tot = &k + 1;
%put &tot;
```

结果：1 + 1

```
%let tot = %eval(&k + 1);
%put &tot;
```

结果：2

注：The %eval 函数仅用于整数运算。这意味着，当任何部分表达的既不是一个整数，也不是逻辑语句时，我们会得到一个错误信息。

例：`%let tot = %eval(&k + 1.234);`

运行错误，此时采用%***sysevalf***宏函数

```
%let tot = %sysevalf(&k + 1.234);
%put &tot;
```

结果： 2.234

%SYSFUNC函数

%SYSFUNC宏函数执行SAS函数。

%SYSFUNC函数的一般形式：

%SYSFUNC(*SAS function(argument(s))* <,*format*>)

***SAS function(argument(s))*指SAS函数的名称和它对应的参数。**

第二个参数指由第一个参数返回的值的可选格式。

%SYSFUNC函数

实例：

```
%put syslast=&syslast;
%let dsn=%sysfunc(propcase(&syslast));
%put dsn=&dsn;
```

SAS日志窗口

```
156  %put syslast=&syslast;
syslast=WORK.ORDERS
157  %let dsn=%sysfunc(propcase(&syslast));
158  %put dsn=&dsn;
dsn=Work.Orders
```

PROPCASE函数将文本转换为适当格式。

%SYSFUNC函数

自动宏变量SYSDATE9和SYSTIME可以用于标题中。

```
title1 "&sysdate9";
title2 "&systime";
```

生成

```
07MAR2008
13:39
```

SYSDATE9和SYSTIME代表SAS会话开始的日期和时间。

%SYSFUNC函数

实例：以适当的格式生成包含当前日期和时间的标题。

```
title1 "%sysfunc(today(),weekdate.)";
title2 "%sysfunc(time(),timeAMPM8.)";
```

生成

```
Monday, March 10, 2008
        4:27 PM
```

%SYSFUNC函数

%SYSFUNC可以使用大多数SAS函数。下列情况是例外：

- 数组处理（DIM, HBOUND, LBOUND）
- 变量信息（VNAME, VLABEL, MISSING）
- 宏接口（RESOLVE, SYMGET）
- 数据转换（INPUT, PUT）
- 其他函数（IORCMSG, LAG, DIF）

INPUTC和INPUTN可以用来代替INPUT。PUTC和PUTN可以用来代替PUT。

二、宏函数小结

1. 字符串处理函数;
   - **%upcase;**
   - **%substr;**
   - **%scan;**
2. 运算函数;
   - **%eval;**
   - **%sysevalf;**

symput和symget函数

要实现在不同的数据步传递数据，需要借助宏变量作为桥梁，通过symput和symget来实现。您可以使用symput函数把数据步的信息传给宏变量，可以使用symget函数把宏变量的信息传给数据步。

Symput函数

CALL SYMPUT("*macro-variable-name*", *value*)

其中，参数1是我们正在创建的宏变量将存储数据步正在传递出的信息，参数2是字符串格式的值。请注意，新的宏变量是文本型，用匹配引号括起来。

Symget函数　　symget("*macro-variable-name*"）

参数可以是宏变量的名字，字符串变量或者字符表达式。

symput和symget函数

```
 proc means data = hsb2 n;
   var write;
   where write>=55;
   output out=w55 n=n;
run;
proc print data = w55;
run;
data _null_;
   set w55;
   call symput('n55', n);
run;
%put &n55 Observations have write >=55;
```

结果：9 Observations have write >=55

思考：为什么不用**%LET**语句？

（ch09_02.sas）

symput和symget函数

假设我们要创建的HSB2数据集是恒定的整个数据集，此变量的值是谁拥有的写作分数55以上的学生人数的新变量。我们已经存储在宏变量N55的数量因此这将是symget函数的参数。请注意，尽管N55是一个宏观变量，我们不使用之前N55的符号标志，相反，我们使用单引号。

```
data hsb2_55;
  set hsb2;
  w55 = symget('n55')+0;
run;
proc print data = hsb2_55;
  var write w55;
run;
```

（ch09_02.sas）

部分结果

Obs	write	w55
1	52	9
2	59	9
3	33	9
4	44	9
5	52	9
6	52	9
7	59	9
8	46	9
9	57	9
10	55	9

三、symput和symget函数小结

1. symput -- call symput('new_macro_variable', value_in_string_format)

2. symget -- symget('macro_variable')

SQL过程步创建宏变量

创建宏变量的另一种方式通过proc sql 。 SQL表示结构化查询语言，是一个标准化的数据库语言。 PROC SQL可以创建包含查询结果的值得SAS的观变量。在下面的例子中，我们创建名为W55宏变量，它包含写作得分高于或等于55的学生人数。

```
proc sql;
   select sum(write>=55) into :w55
   from hsb2;
quit;
%put w55 is &w55;
```

结果：w55 is 9

（ch09_03. sas）

SQL过程步创建宏变量

下面的例子说明如何对变量的SES的每一个水平创建组平均值，并将它们存储在write1，write2 和write3三个宏变量中。我们利用%put函数显示在日志文件中宏变量的值。

```
proc sql;
  select mean(write) into :write1 - :write3
  from hsb2
  group by ses;
quit;
%put write1 to write3 are &write1, &write2 and
&write3;
```

结果：write1 to write3 are 52.66667, 54.8 and 50.4

（ch09_03. sas）

四、SQL过程步创建宏变量小结

用带*select into*的*proc sql*语句创建宏变量。

SAS宏

到目前为止，我们已经取得了熟悉的宏观变量。现在，我们将利用这些知识来编写一些宏程序。宏程序总是以包括用户自定义的程序名称的*%macro*语句开头，以*%mend*语句结束。

当SAS是要编译SAS程序时，在到达编译器之前会把程序发送给词扫描仪。宏处理器将宏语法转换成稍后被变异的标准的SAS语法，然后编译。因此，宏语言作为一个动态的编辑SAS程序。

SAS宏

首先创建一些练习数据集，在接下来的数据步中，我们创建了四个数据文件: file1 - file4。

```
data file1 file2 file3 file4;
  input a @@;
  if _n_ <= 3 then output file1;
  if 3 < _n_<=  6 then output file2;
  if 6 < _n_ <= 9 then output file3;
  if 9 < _n_ <=12 then output file4;
cards;
1 2 3 4 5 6 7 8 9 10 11 12
;
run;
```

（ch09_04.sas）

SAS宏

在下面程序中，目标是要把若干数据集合并成一个数据集。假设我们有被命名为文件file2，file2等四个数据集。在一个标准的SAS程序，我们将不得不在SET语句中写出所有文件的名称。在宏程序中，我们将演示程序如何把文件名字写在set语句中。

通常，总是先写一个普通的SAS程序，测试它，然后把它变成一个程序。例如，下面的数据步骤将是我们用于串接四个文件的基本程序。

```
data all;
  set   file1     file2     file3     file4
  ;
run;
```

产生系列文件名的宏程序

我们怎么把这块SAS程序转换到SAS宏程序？我们需要开始以*%macro*语句开头，在*%macro*语句中我们指定宏的名称;然后我们写程序，最后我们用*%mend*语句结束宏程序。实际上，我们只需要大幅修改的只有SET语句。考虑所谓宏程序结合在下面的例子。为了自动创建文件名列表而不是一个个写出来，我们需要在*SET*语句中创建*DO*循环。

```
%macro combine;
  data all_1;
    set
    %do i = 1 %to 4;
      file&i
    %end;
    ;
  run;
%mend;
```

（ch09_04.sas）

产生系列文件名的宏程序

我们提交宏程序用与提交SAS程序一样的方式。该程序然后可以通过提交下面的代码来执行，这些代码包括带着宏程序的百分号。需要注意的是宏程序在语句中被调用，与所有标准的SAS程序不同的是，**不以分号结束**。此外，该宏不带任何参数。为了看看是怎么回事，我们打开一个SAS宏的系统选项--mprint。它会打印出由宏执行产生的SAS语句。

```
*executing the combine program;
options mprint;
%combine
```

结果：
```
MPRINT(COMBINE):   data all_1;
MPRINT(COMBINE):   set file1 file2 file3 file4 ;
MPRINT(COMBINE):   run;
```

带参数的宏

唯一的其他变化，不是执行4次DO循环，而是我们现在执行它用&num 。在代码的结尾，当我们终于执行了新版本的结合程序，我们指定我们要执行do循环三次，从而把文件1，文件2和文件3叠加起来。

```
%macro combine(num);
  data big;
    set
    %do i = 1 %to &num;
      file&i
    %end;
    ;
  run;
%mend; *executing the macro program;
%combine(3)
```

（ch09_04. sas）

结果：

```
MPRINT(COMBINE):  data big;
MPRINT(COMBINE):  set file1 file2 file3 ;
MPRINT(COMBINE):  run;
```

带参数的宏例

```
%macro gen(n,start,end);
  data generate;
    do subj=1 to &n;
       x=int(&End-&start+1)*ranuni(0)+&start);
       output;
     end;
run;
proc print data=generate noobs;
  title "Randomly Generated Data Set with &n Obs";
  title2 "Values are integers from &start to &end";
run;
%mend gen;

%gen(4,1,100)   *产生1到100的整数随机数4条;
```

重复若干次程序的宏

假设我们有许多二元因变量和两个自变量。我们的任务是，对每一个因变量关于两个自变量做逻辑回归分析。我们可以简单的为每个模型写一个*proc logistic*，但是这将是乏味和繁琐的。相反，我们选择编写一个通过自动循环所有的因变量并对每一个因变量关于所有自变量进行逻辑回归的宏程序。

重复若干次程序的宏

首先创建一个包括因变量*v1*到*v5*以及自变量ind1和ind2的数据集。

为了得到更好地了解我们如何编写宏程序，让我们先写一个标准的SAS程序拟合v1的逻辑模型。

```
proc logistic data = xxx descending;
  model v1 = ind1 ind2;
run;
```

```
data xxx;
   input v1-v5 ind1 ind2;
   cards;
1 0 1 1 0 34 23
0 0 1 0 1 22 32
1 1 1 0 0 12 10
0 1 0 1 1 56 90
0 1 0 1 1 26 80
1 1 0 0 0 46 45
0 0 0 1 1 57 53
1 1 0 0 0 22 77
0 1 0 1 1 44 45
1 1 0 0 0 41 72
;
run;
```

（ch09_05.sas）

重复若干次程序的宏

这个程序的哪一部分是必须要改变的？关键的变化是在模型语句中。下面的程序将演示改变它的一种方式。我们创建了一个遍历每个因变量的do循环，并且对每一个变量拟合一个逻辑模型。我们有一些参数，叫NUM，这将说明我们有多少个因变量将被使用。DO循环利用了自变量的命名规则的优势。

```
%macro mylogit(num);
    %do i = 1 %to &num;
        title "dependent variable is
v&i";
        proc logistic data=xxx des;
          model v&i = ind1 ind2;
        run;
   %end;
%mend;
*executing the macro using 5
dependent variable;
%mylogit(5)
```

（ch09_05.sas）

宏程序调试

在修改我们的宏程序之前，让我们暂停一秒钟。当我们写的SAS宏程序，SAS实际上会尽力帮助我们发现程序中的错误。两个SAS选项是特别有用的：mprint和mlogic。我们已经看到了mprint的选项如何帮助我们明白宏程序到通常SAS语句的翻译过程。

与其他SAS选项一起添加这两个选项。注意到，SAS在日志窗口中溢出了所有与宏程序和宏变量相关的相关信息。另一种调试方法是在宏程序里手动使用%PUT语句。例如，在下面例子中，%put是在循环语句后使用的。我们可以看到循环是否会正常停止这种方式。

宏程序调试

```
options mprint mlogic;
%macro mylogit(num);
  %do i = 1 %to &num;
    proc logistic data=xxx des;
      model v&i = ind1 ind2;
    run;
  %end;
  %put &i;
%mend;
*executing the macro using 5 dependent variable;
%mylogit(5)
```

（ch09_05.sas）

```
MLOGIC(MYLOGIT):  %DO loop index variable I is now 6; loop will not iterate again.
MLOGIC(MYLOGIT):  %PUT &i
6
MLOGIC(MYLOGIT):  Ending execution.
```

指定分析因变量

在目前版本的mylogit宏程序中存在一些限制；当因变量的名字是v1，v2等形式的时候，它只能对变量名的后缀循环。我们想修改mylogit宏使其能够处理任何取名的因变量（不管是否有规律），将因变量名列作为宏参数，然后宏将对名列提取每个因变量进行逻辑回归。使用宏函数%scan可以解决该问题。%scan将一次性扫描因变量的列表。因变量的名称将被存储在局部宏变量dep中，然后进行逻辑回归。在我们要求SAS来迭代该过程时while循环将工作，直到DEP中的变量用完，用判断是否缺失作为终止循环条件。我们为循环结构的每一个循环增加局部宏变量K，因为&K是在变量列表中的位置指示器。因此，对于第一次循环K＝1，并且scan函数把第一个变量保存在局部宏变量DEP中的因变量列表。然后SAS用列表中的第一个变量做自变量拟合一个逻辑模型，然后递增&K＝2，扫描到DEP中的第二个因变量，然后做逻辑回归。这一直持续到因变量列表已用尽，此时DEP将等于缺失值，循环结束。

指定分析因变量

```
%macro mylogit1(all_deps);
  %let k=1;
  %let dep = %scan(&all_deps, &k);
  %do %while("&dep" NE "");
    title "dependent variable is &dep";
    proc logistic data=xxx des;
      model &dep = ind1 ind2;
    run;
    %let k = %eval(&k + 1);
    %let dep = %scan(&all_deps, &k);
  %end;
%mend;
*run the program for the first three v's;
%mylogit1(v1 v2 v3)
```

（ch09_05.sas）

保存结果到数据集

接下来的推广，我们想实现的是能够从每个Logistic模型保存估计的数据集。所以，宏程序现在将对两个参数：all_dep--因变量列表，outset--一个包含所有的逻辑模型的估计信息的数据集。 mylogit1宏程序在proc logistic语句中使用outest选项创建一个包含所有拟合模型的参数估计的数据集。第一个拟合模型中的参数估计将会被存在叫_est1的数据集中，第二个拟合模型中的参数估计将会被存在叫_est2的数据集中，以此类推。如果我们为outest参数即存储参数估计结果指定一个文件名，则参数估计通过宏将结果保存到指定的文件名中。如果outest缺失，然后该程序使用proc datasets，删除所有包含参数估计的数据集。

保存结果到数据集

```
%macro mylogit1(all_deps, outest);
  %let k=1;
  %let dep = %scan(&all_deps, &k);
  %do %while("&dep" NE "");
    title "dependent variable is
&dep";
    proc logistic data=xxx des
outest=_est&k;
      model &dep = ind1 ind2;
    run;
    %let k = %eval(&k + 1);
    %let dep = %scan(&all_deps,&k);
  %end;
```

```
%if "&outest" NE "" %then
  %do;
    data &outest;
      set
      %do i = 1 %to &k - 1;
        _est&i
      %end;
      ;
    run;
    %let k = %eval(&k - 1);
```

（ch09_05.sas）

保存结果到数据集

```
proc datasets;
      delete _est1 - _est&k;
    run;
  %end;
  %else   %do;
     %put no dataset name was provided, files are not
combined;
  %end;
%mend;
%mylogit1(v1 v2 v3)
%mylogit1(v1 v2 v3, a)
proc print data = a;
  var intercept ind1 ind2;
run;
```

结果

Obs	Intercept	ind1	ind2
1	2.4570	-0.04282	-0.01709
2	0.3278	-0.09480	0.09078
3	33.3421	-0.50434	-0.40122

（ch09_05.sas）

优化的宏程序

继续优化得到mylogita的宏，其功能允许用户指定输入的数据文件，因变量的列表，自变量列表和保存逻辑回归的参数估计结果的文件。该宏带有两种类型的参数：位置参数和非位置参数。非位置参数后面跟着一个等号，并可能是一个缺省默认值。参数indvars是一个不具有默认值的例子，参数outest是一个有_out默认值的非位置参数的例子。参数INDATA和all_deps是位置参数。当我们要执行宏程序时，这些类型的参数调用时，其位置顺序的变化影响宏的正确运行，因此SAS设定位置参数必须出现在非位置参数前面，且位置参数按照正确的顺序被调用。

优化的宏程序

换句话说，在调用该宏时，输入数据集名参数必须是第一个参数，因变量列表是第二个参数。而自变量的列表和保存参数估计的数据集名两个参数位置不要求固定。我们可以改变这些参数的顺序，我们所要做的就是通过包括参数的名称和一个等号和值指定把数值给哪些参数，因此，在我们执行mylogita宏的第一个例子中，我们声明，自变量的列表应包括IND1和IND2（通过指定indvars= IND1 IND2），以及含有该参数估计的数据集的名称应该是myparms（由指定outest= myparms）。在第二个例子中，宏调用时，调整了两个参数位置，结果正常，因为我们使用了固定参数名调用方法。

优化的宏程序

```
%macro mylogita(indata, all_deps, indvars =, myout =_out );
 %let k=1;
 %let dep = %scan(&all_deps, &k);
 %do %while(&dep NE);
   title "The dependent variable is &dep";
   title2 "The independent variables are &indvars";
   proc logistic data=&indata des outest=est&k;
     model &dep = &indvars;
   run;
   %let k = %eval(&k + 1);
   %let dep = %scan(&all_deps, &k);
 %end;
```

续

```
data &myout;
    set
    %do i = 1 %to &k - 1;
      est&i
    %end;
    ;
  run;
%mend;
```

（ch09_05.sas）

优化的宏程序

```
*run the program;
%mylogita(xxx, v1 v2 v3, indvars = ind1 ind2, myout =
myparms)

title;
proc print data = myparms;
  var _name_ intercept ind1 ind2;
run;
```

（ch09_05.sas）

结果

Obs	_NAME_	Intercept	ind1	ind2
1	v1	2.4570	-0.04282	-0.01709
2	v2	0.3278	-0.09480	0.09078
3	v3	33.3421	-0.50434	-0.40122

优化的宏程序

```
* run the program again: unpositional arguments can be reordered;
%mylogita(hsb2,female, myout = myparm1, indvars = write math)
title;
proc print data = myparm1;
  var _name_ intercept write math;
run;
```

（ch09_05.sas）

结果

Obs	_NAME_	Intercept	write	math
1	female	-3.49607	0.068307	-0.022230

SAS宏总结

1. 用*%macro*和*%mend*定义SAS宏程序;
2. 用*%let*语句在宏程序中建立宏变量。
3. 利用宏函数，比如：*%scan*和*%eval*
4. 如何调用SAS宏程序（启动宏程序）
5. 如何调试SAS宏程序
6. 位置参数和非位置参数;