Paper HW05-2016

Intermediate SAS® Macro Programming

Chuck Kincaid, Experis Business Analytics

ABSTRACT

The SAS Macro Language powerfully enhances a programmer's capabilities by providing an advanced level of flexibility and robustness to otherwise static programs. Programmers can use the Macro language to create dynamic programs that can be re-used in new situations, driven by the data, dependent upon the operating environment, and made to handle repetition without extensive coding. This Hands-On Workshop will explore some of the more advanced capabilities with the SAS Macro Language that may not be familiar to those just starting in the language. Some of the features explored include understanding variable scope, interacting with data steps, evaluating expressions, quoting variables, macro looping, macro debugging, and creating data driven programs. Basic experience with macro programming is assumed.

INTRODUCTION

Please look for a fuller paper in the future.

VARIABLE SCOPING

Here is the code and comments for Solution Set #1:

```
/*
    Solution Set #1

    Variable Scoping
        Global and Local variables
        Parameters
        Resolution
        Indirect macro resolution

    Execution vs. Compile
    Call Symput
    Call execute

      If you run into trouble, try this sequence...  *******

      "; '; %mend;
*/

/*************************************************************************
    Exercise 1.0 Simple macro
*************************************************************************/

%let myteam=Florida;
%macro printmystats1();
    proc print data=work.baseball;
        where (team = "&myteam");
    run;
%mend printmystats1;
%printmystats1();

/*************************************************************************
    Exercise 1.1
    Global and Automatic variables
```

```
    Seeing the variables
*************************************************************************/

%put    Exercise 1.1;
%put  Global and Automatic variables;
%put    Seeing the variables;
%put _all_;  * prints all macro variables;

/*************************************************************************
    Exercise 1.1b
    Compare SASHELP.VMACRO to %put _all_
*************************************************************************/

title1 'Exercise 1.1b';
title2 'Compare SASHELP.VMACRO to % put _all_';
proc print data=sashelp.vmacro;     *the vmacro table holds all the macro
variable definitions;
run;

/*************************************************************************
    Exercise 1.2
    local variables

      No global variable called "otherteam"
    %let within %macro call
    %let assigns to local variable
*************************************************************************/


%macro printmystats2();
    %let otherteam=Kansas St; * otherteam is local because it is created in
                              the macro and there is no global variable;
    proc print data=work.baseball;
        where ((team = "&myteam") or (team = "&otherteam"));
    run;

    %put Exercise 1.2;
    %put local variables;
    %put No global variable called "otherteam";
    %put % let within % macro call;
    %put % let assigns to local variable;
      %put ;
    %put *********_global_*************;
    %put _GLOBAL_;      * otherteam does not show up here;
    %put ********_local_**************;
    %put _local_; * otherteam shows up here because it is local;
    %put ********_user_**************;
    %put _USER_;  * otherteam shows up here because it was created by the
user;

%mend printmystats2;
%printmystats2();

/* Outside of the macro, there are no local variables   */
%put ***** outside of the macro *****;
%put _user_;
```

```
/*************************************************************************
    Exercise 1.3
    local variables
    Same thing with variable as parameter
*************************************************************************/

%macro printmystats3(otherteam);    * parameters are local variables;
    proc print data=work.baseball;
        where ((team = "&myteam") or (team = "&otherteam"));
    run;

    %put **** inside macro **********;
    %put _USER_;
    %put ******* myteam otherteam *******;
    %put &myteam &otherteam;  * another way to see specific variables;
%mend printmystats3;

%printmystats3(Kansas St);

/* Outside of the macro, there are no local variables   */
%put ****outside of the macro****;
%put _USER_;

/*************************************************************************
    Exercise 1.4
    What if there is a global variable called "otherteam"?
    First, explore with local variable as parameter
*************************************************************************/

%let otherteam = St Marys;    * defining the variable in open code makes it
                                global;
%printmystats3(Kansas St);    * The "otherteam" variable is being created in
                                the macro as a parameter.;
                              * It stays local and does not change the global
                                value;

%put ***** outside of the macro *****;
%put _user_;

/*************************************************************************
    Exercise 1.5
    What if there is a global variable called "otherteam"?
    Second, explore with local variable in % let statement
*************************************************************************/

%macro printmystats4();
    %let otherteam=Kansas St; * "local" variable is not local. It changes the
                                global variable;
    proc print data=work.baseball;
        where ((team = "&myteam") or (team = "&otherteam"));
    run;

      %put ****inside of the macro****;
    %put _USER_;

%mend printmystats4;
```

```
%printmystats4();

%put ***** outside of the macro *****;
%put _user_;            * the global version has changed;

%let otherteam=;  * reset the global variable to blank;

/*************************************************************************
Exercise 1.6
    What if there is a global variable called "otherteam"
    Forcing a variable to be local
            a) eliminates possible conflicts
            b) saves space, since local variables do not exist after macro
execution
*************************************************************************/

%macro printmystats5();

    %local otherteam;   * there is now a local macro variable and a global
                            macro variable;
    %let otherteam=Kansas St;

    proc print data=work.baseball;
        where ((team = "&myteam") or (team = "&otherteam"));
    run;

    %put ****inside of the macro****;
    %put _USER_;

%mend printmystats5;

%printmystats5();

%put ***** outside of the macro *****;
%put _user_;

/*************************************************************************
    Exercise 1.7a
    What if we wanted to make a variable global
      Either
            a) define it outside the macro as we have done above
            b) define it as global inside the macro

      First, let's see why we need to do this.
*************************************************************************/

%macro varlist();
      %let myvars = BB H HBP;

      %put ***** in varlist *****;
    %put _USER_;

%mend;

%macro printmystats6();

    %local otherteam;
```

```
    %let otherteam=Kansas St;
      %varlist(); * the variable myvars is created and only exists as a local
                    variable within the macro varlist;

    proc print data=work.baseball;
        where ((team = "&myteam") or (team = "&otherteam"));
            var &myvars;      * An error message will be generated - myvars
                              cannot be resolved since it no longer exists;
    run;

    %put ***** in printmystats6 *****;
    %put _USER_;

%mend printmystats6;

%printmystats6();

%put ***** outside of the macro *****;
%put _user_;

/*************************************************************************
    Exercise 1.7b
    What if we wanted to make a variable global

      First option is to define it outside the macro
*************************************************************************/

%let myvars = ;   * now the variable exists in the global table;

%macro varlist();
      %let myvars = BB H HBP; * we are changing the global version of the
                                variable;
    %put ***** in varlist *****;
    %put _USER_;
%mend;

%printmystats6();       * the variable now exists and gets its value from the
global table;

%put ***** outside of the macro *****;
%put _user_;

/*************************************************************************
    Exercise 1.7c
    What if we wanted to make a variable global

      Second option is to define it as global inside the macro
      First, delete the global macro variable myvar.
*************************************************************************/

%symdel myvars ;
%put _user_;

%macro varlist();
      %global myvars;   * put the variable in the global symbol table;
      %let myvars = BB H HBP;
```

```
      %put ***** in varlist *****;
    %put _USER_;
%mend;

%macro printmystats7();
      %local otherteam;
    %let otherteam=Kansas St;
      %varlist();

    proc print data=work.baseball;
        where ((team = "&myteam") or (team = "&otherteam"));
            var &myvars;
    run;

    %put ***** in printmystats7 *****;
    %put _USER_;
%mend printmystats7;

%printmystats7();

%put ***** outside of the macro *****;
%put _user_;
```

MACRO OPTIONS

```
/*
    Solution Set #2

    Macro options
    Other debugging techniques

*/

%macro varlist();
      %global myvars;
      %let myvars = BB H HBP;
%mend;

%macro printmystats8();
      %local otherteam;
    %let otherteam=Kansas St;
      %varlist();

    proc print data=work.baseball;
        where ((team = "&myteam") or (team = "&otherteam"));
            var &myvars;
    run;
%mend printmystats8;

/*************************************************************************
      Exercise 2.1
      Macro options

      Another way to see the values of macro variables
*************************************************************************/
```

```
options symbolgen ;     * displays the results of resolving macro variable
references;

%printmystats8();

/*************************************************************************
      Exercise 2.2
      Macro options

      We can also see how the macro executes
*************************************************************************/

options mlogic nosymbolgen;   * causes the macro processor to trace its
                              execution and to write the trace information to
                              the SAS log;
                              * nosymbolgen turn off the option;

%printmystats8();

/*************************************************************************
      Exercise 2.3
      Macro options

      We can save the SAS code that the macro creates
*************************************************************************/

options mprint nomlogic;      * displays the SAS statements that are
                              generated by macro execution;
                              * nomlogic turn off the option;
%printmystats8();

/*************************************************************************
      Exercise 2.4
      Macro options

      We can the SAS code that the macro creates
      and we can save the SAS code into a file to be used later
*************************************************************************/

options mprint mfile;   * mfile routes output produced by the MPRINT option
                        to an external file;

filename mprint "&homedir\results\printmystats8.sas";

%printmystats8();

/*************************************************************************
      Exercise 2.5
      Macro options

      The MPRINT / MFILE options keep writing to the file.
      We can turn the option off.
*************************************************************************/

options mprint nomfile;

%printmystats8();
```

```
/*************************************************************************
      Exercise 2.6

      Can redefine macros by submitting new definitions
      Can also delete macro definitions
*************************************************************************/

%sysmacdelete printmystats8;

%printmystats8();

/*************************************************************************
      Exercise 2.7

      Use PROC OPTIONS to see the macro options and their values
*************************************************************************/

proc options group=macro;
run;

/*
      See the macro options and their values with a bit more detail
*/

proc options group=macro value;
run;

/*************************************************************************
      Exercise 2.8

      Reset the values of the macro options
      Can explicitly change values in OPTION statement

      Can also change the values via macro function calls;
      SYSFUNC execute SAS functions or user-written functions
      GETOPTION is a SAS function that returns the value of a SAS system or
graphics option.
*************************************************************************/

/*    Check the value of MVARSIZE before we change it.
      MVARSIZE specifies the maximum size for macro variable values that are
stored in memory.
      KEYWORD shows the name of the variable
      DEFAULTVALUE shows the ... default value
*/
      %put %sysfunc(getoption(mvarsize));

      %put %sysfunc(getoption(mvarsize,keyword));

      %put %sysfunc(getoption(mvarsize,keyword,defaultvalue));

/*    Get the current value for mvarsize                                  */
      %let orig_size=%sysfunc(getoption(mvarsize,keyword));
      %put &orig_size;

/* Change the MVARSIZE value and check the change.            */
```

```
      options mvarsize=2000;
      %put %sysfunc(getoption(mvarsize,keyword));

/* Change MVARSIZE back to the original value and check it.     */
   options &orig_size; run;
   %put %sysfunc(getoption(mvarsize,keyword));
```

EVALUATING EXPRESSIONS

```
/*
    Solution Set #3

    Evaluating expressions
*/

/*************************************************************************
      Exercise 3.1
      macro processor is text substituion
*************************************************************************/

%macro bestteams(stat,cutoff);
      title1 "Teams with &stat over &cutoff";
      proc print data=baseball(where=(&stat > &cutoff));
      run;
%mend bestteams;

%bestteams(r,38);       * this expression, 38, gets resolved correctly;

%bestteams(r,18+20);    * this expression, 18 + 20, does get resolved by the
                        where option;
                        * not by the macro processor;

/*************************************************************************
      Exercise 3.2
      Evaluating expressions

      Some macro language elements do evaluate arithmetic and logical
expressions
*************************************************************************/


%macro bestteams2(cutoff=60);
      %if &cutoff >= 60 %then %do;
            title1 "Teams with more than &cutoff runs";
            proc print data=baseball(where=(r > &cutoff));
            run;
      %end;
%mend bestteams2;

%bestteams2();          * the %if evaluates the inequality
                        expression "60 >= 60";

%bestteams2(cutoff=30+40);    * the %if evaluates the integer
                              arithmetic inequality expression "30+40 >= 60";

%bestteams2(cutoff=31.0 + 36);      * the %if only evaluates integer
                                    arithmetic without any characters, ;
```

```
                                    * not even decimal points ;

/*************************************************************************
      Exercise 3.3
      Evaluating expressions

      We can explicitly evaluate expressions with % EVAL

      Use %EVAL to cause & cutoff to be evaluated in the title.
      Do we need %EVAL in the % IF statement?
      Do we need %EVAL in the where option?
*************************************************************************/

%macro bestteams3(cutoff=60);
   %if &cutoff >= 60 %then %do;     /* Do not need %EVAL here since the
                                    %IF does the integer evaluation */
title1 "Teams with more than %eval(&cutoff) runs";
proc print data=baseball(where=(r > &cutoff));  /* Do not need %EVAL here
                                                   since the where option
                                                   evaluates expressions */
                                                   /* when the proc is run */
run;
      %end;
%mend bestteams3;

%bestteams3(cutoff=30+40);    * the title should show 70 instead of 30+40;

/*    Can the %EVAL be used to fix the error caused by the following
parameters? */

%bestteams3(cutoff=31.0 + 36);      * No. the %EVAL only evaluates integer
                                    arithmetic ;

/*************************************************************************
      Exercise 3.4
      Evaluating floating point arithmetic expressions

      When we want to evaluate floating point arithmetic we can use %SYSEVALF
      Where do we need it?
*************************************************************************/

%macro bestteams4(cutoff=60);
/* need %SYSEVALF here since the %IF does integer arithmetic evaluation */
   %if %sysevalf(&cutoff) >= 60 %then %do;

/* need %SYSEVALF here since there is no evaluation */
      title1 "Teams with more than %sysevalf(&cutoff) runs";

/* do not need %SYSEVALF here since the WHERE option does the evaluation */
      proc print data=baseball(where=(r > &cutoff));
      run;
   %end;
%mend bestteams4;

%bestteams4(cutoff=30.9 + 40.9);
title1;
```

```
/*************************************************************************
      Exercise 3.5
      Evaluating floating point arithmetic expressions

      Batting Average is the # of Hits divided by the # of times at bat.
      BA = H / AB

      Compile and execute this macro % EVAL and % SYSEVALF
      What do you get in each situation?
      Why?
*************************************************************************/

%macro ba(h,ab);
      %global ba;
      %let ba = (&h / &ab);
      %put &ba;

%mend ba;

* without % EVAL or % SYSEVALF all three of these are simply text
substitutions. No evaluation is done.;

%ba(33,106);      * will evaluate the expression with % EVAL, but returns the
                  integer portion;
                  * will return the floating point value with % SYSEVALF;

%ba(,106);  * Returns error with % EVAL and with % SYSEVALF;

%ba(.,106); * returns error with % EVAL and misssing value with % SYSEVALF;

/*************************************************************************
      Exercise 3.6
      Evaluating floating point arithmetic inequality expressions

      Compile this macro and execute with the parameters below
      What happens in each?
      Why?

      How can you make it work?
*************************************************************************/

%macro topteam(team1,obp1,team2,obp2);
      %global top;
      %if (&obp1 > &obp2) %then %let top = &team1;
      %else %let top=&team2;
%mend topteam;

/*
Both of these comparisons are done as characters because of the decimal
point.
      0.38 < 0.387 because 0=0, .=., 8=8, (blank) < 7
      .38 < 0.35 because . < 0
*/

* Southern California comes out on top;
%topteam(Bradley,0.38,Southern California,0.387);
%put &top;
```

```
%topteam(Bradley,.38,Oral Roberts,0.35);  * Oral Roberts comes out on top;
%put &top;

/*
      This macro executes the way we'd want.
*/

%macro topteam(team1,obp1,team2,obp2);
      %global top;
      %if %sysevalf(&obp1 > &obp2) %then %let top = &team1;
      %else %let top=&team2;
%mend topteam;

%topteam(Bradley,0.38,Southern California,0.387);
%put &top;

%topteam(Bradley,.38,Oral Roberts,0.35);
%put &top;
```

THE MACRO / DATA STEP BARRIER

```
/*
    Solution Set #4

   CALL SYMPUT and SYMGET statements cross the macro / data step barrier
   Introduce the four times in the execution of a SAS program

   Macro compile time
   Macro execution time
   SAS compile time
   SAS execution time
*/

/*************************************************************************
      Exercise 4.1
      Creating macro variables within data step whose values are data-driven

      Keywords:         CALL SYMPUT
*************************************************************************/

data hab;
      merge h ab;
      by quantile;
      if quantile = '75% Q3' then do;
            call symput('H',h);
            call symput('AB',ab);
      end;
run;

/*************************************************************************
      Another form for printing macro variables to the log

      Structure:  &=<variable>
*************************************************************************/

%put &=h &=ab;
```

```
proc print data=baseball(where=((H > &H) or (AB > &AB)));   * Now get the
output;
run;

/*************************************************************************
      Exercise 4.2
      Creating macro variables within data step programmatically

      compress(quantile,'%','s')  removes all the percent signs and all the
spaces ("s")
      compress(quantile)                  removes all the blank spaces
*************************************************************************/

data _null_;
      merge h ab;
      by quantile;
      ba = h / ab;
* We can build the macro variable names from the data;
      call symput('BA'||compress(quantile,'%','s'),compress(ba));
      call symput('BAL'||compress(quantile,'%','s'),compress(quantile));
run;

%put _user_;




proc sgplot data=baseball;
      histogram avg;
      refline &ba0min &ba1 &ba5 &ba10 &ba25q1 &ba50median &ba75q3 &ba90 &ba95
&ba99 &ba100max / axis=x
            label=("&bal0min" "&bal1" "&bal5" "&bal10" "&bal25q1"
"&bal50median" "&bal75q3" "&bal90" "&bal95"
                        "&bal99" "&bal100max");
run;


/*************************************************************************
      Exercise 4.3
      Resolving macro variables within data step

      Use SYMGET to resolve macro variables according to the data.

      SLG_ = Slugging Percentage which is a popular measure of the power of a
      hitter. It is calculated as total bases divided by at bats: where AB is
      the number of at-bats for a given player, and 1B, 2B, 3B, and HR are
      the number of singles, doubles, triples, and home runs, respectively.
*************************************************************************/

/*    For each year, get the highest slugging percentage    */
proc sort data=baseball;
      by year descending slg_ ;
run;
```

```
/*    Create a macro variable with this max for each year   */
data _null_;
      set baseball;
      by year ;
      if first.year then call symput("slg"||compress(year),slg_);
run;

/*************************************************************************
      Pull the macro variables back into the data set depending on the year


      Keywords:         SYMGET
*************************************************************************/
data baseball3;
      set baseball;
      retain max_year;
      by year;
      if first.year then max_year = symget("slg"||compress(year));
      index = slg_ / max_year;
run;

/*
Create paneled histograms of the indices to compare from year to year
*/


proc sgpanel data=baseball3;
      panelby year;
      histogram index;
run;

/*************************************************************************
      Exercise 4.4
      Resolving macro variables within data step

      Use SYMGET to resolve macro variables whose values change in the data
      Step what will team1, team2, team3, and team4 resolve to?
*************************************************************************/

options mprint;

%let team = Kansas St;

%macro whosonfirst();

      data _null_;
            team1 = "&team";        * variable is resolved at compile time;
            team2 = symget('team'); * variable is resolved at execution time;
            call symput('team','Florida');   * variable is given a value at
                                               execution time;
            team3 = "&team";        * variable is resolved at compile time;
            team4 = symget('team'); * variable is resolved at execution time;
            put (team1-team4) (=);
      run;

%mend whosonfirst;
```

```
%whosonfirst();

* as if we executed the following code;

data _null_;
      team1 = "Kansas St";
      team2 = "Kansas St";

      team3 = "Kansas St";
      team4 = "Florida";
      put (team1-team4) (=);
run;
```

QUOTING

```
/*
    Solution Set #5

    Quoting
    Iterative logic
    Explain quoting options if haven't already
*/

/*************************************************************************
      Exercise 5.1
      Quoting allows special characters to be used

      Try these statements with the various quoting functions. What happens
      with each? Check both the log and the output. Use macro functions to
      help understand. Which are best?
      STR, NRSTR, BQUOTE, NRBQUOTE, SUPERQ
*************************************************************************/

/*
the STR function masks at macro compilation. It does not mask the % or the &
*/
title1 "STR => %str(&bal50median)";
proc print data=baseball(obs=1);
run;

/*
the NRSTR function masks at macro compilation. It does mask the % or the &
that is why the &bal50median never gets resolved
*/
title1 "NRSTR => %nrstr(&bal50median)";
proc print data=baseball(obs=1);
run;

/*
the BQUOTE function masks at macro execution
However, it does not mask the % or the &
*/
title1 "BQUOTE => %bquote(&bal50median)";
proc print data=baseball(obs=1);
run;
```

```
/*    the NRBQUOTE function masks at macro execution. It does mask the % or
the &
      However, it gives a warning if there is an unresolvable macro variable
      reference or macro invocation                                    */
title1 "NRBQUOTE => %nrbquote(&bal50median)";
proc print data=baseball(obs=1);
run;

/*
the SUPERQ function masks all items that might require macro quoting at macro
execution. Note that its argument is the name of the macro variable without
the &
*/
title1 "SUPERQ => %superq(bal50median)";
proc print data=baseball(obs=1);
run;


/*************************************************************************
      Exercise 5.2
      Quoting allows special characters to be used

      Use the %STR function to let the semicolon be part of the
      macro variable at compilation time
*************************************************************************/

* have to quote the semicolon so that it is interpreted as part of the
variables value;
%let graphic = %str(histogram avg;) ;

%macro histogram(statistic,label);

      title1 "&label";
      proc sgplot data=baseball;
            &graphic
            refline &statistic / axis=x lineattrs=(thickness=5px);
      run;

%mend histogram;

%histogram(&ba50median,Median Value);

/*************************************************************************
      Exercise 5.3
      Quoting allows special characters to be used

      What quoting function gives us what we want without errors or warnings
*************************************************************************/

%macro histogram2(statistic,label);
      title1 "&label";
      proc sgplot data=baseball;
            &graphic
            refline &statistic / axis=x lineattrs=(thickness=5px);
      run;

%mend histogram2;
```

```
%histogram2(&ba50median,%superq(bal50median));

/*    See that again, but now in slow motion          */
option mprint symbolgen mlogic;

%histogram2(&ba50median,%superq(bal50median));

/*************************************************************************
      Exercise 5.4
      Quoting allows special characters to be used

      We could have avoided all of that by not compressing the quantile
      label. We can show all of the quantiles.
*************************************************************************/

data _null_;
      merge h ab;
      by quantile;
      ba = h / ab;
      call symputx('BAx'||compress(quantile,'%','s'),ba);
      call symput('BALx'||compress(quantile,'%','s'),quantile);
run;

%put _user_;

proc sgplot data=baseball;
      histogram avg;
      refine &bax0min &bax1 &bax5 &bax10 &bax25q1 &bax50median &bax75q3
&bax90 &bax95 &bax99 &bax100max / axis=x
            label=("&balx0min" "&balx1" "&balx5" "&balx10" "&balx25q1"
"&balx50median" "&balx75q3" "&balx90" "&balx95"
                        "&balx99" "&balx100max");
run;

/*************************************************************************
      Exercise 5.5
      Quoting allows special characters to be used

      Suppose we don't want to type all of those variable names! We can use a
      %DO loop to create the macro variables
*************************************************************************/

%macro test_theory();

      data habx;
            merge h ab;
            by quantile;
            ba = h / ab;
      run;

      proc sort data=habx;
            by h;
      run;

      data _null_;
            set habx;
```

```
            call symputx('BAx'||compress(_N_),ba);
            call symput('BALx'||compress(_N_),quantile);
      run;

      proc sgplot data=baseball;
            histogram avg;
            refline
                  %do i = 1 %to 11;
                        &&BAx&i
                  %end;
                        / axis=x
                  label=(
                        %do i = 1 %to 11;
                              "%trim(&&BALx&i)"
                        %end;
                  );
      run;

%mend test_theory;

%test_theory();

/*************************************************************************
      Exercise 5.6
      Quoting allows special characters to be used

      Maybe we don't know how many variables we have. We could pass that in
      as a parameter or we can find out from the data set
*************************************************************************/

%macro test_theory();

      data habx;
            merge h ab;
            by quantile;
            ba = h / ab;
      run;

      proc sort data=habx;
            by h;
      run;

      data _null_;
            set habx;
            call symputx('BAx'||compress(_N_),ba);
            call symput('BALx'||compress(_N_),quantile);
      run;

      /* this section might be it's own macro   */
      data _null_;
            set habx nobs=count;
            call symput('num', left(put(count,8.)));
            stop;
      run;

      proc sgplot data=baseball;
            histogram avg;
```

```
            refline
                  %do i = 1 %to &num;
                        &&BAx&i
                  %end;
                        / axis=x
                  label=(
                        %do i = 1 %to &num;
                              "%trim(&&BALx&i)"
                        %end;
                  );
      run;

%mend test_theory;

%test_theory();
```

REFERENCES

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Chuck Kincaid
Experis Business Analytics
269-553-5140
chuck.kincaid@experis.com
www.experis.us/analytics
Twitter: @experisanalytic
LinkedIn: Experis Business Intelligence and Analytics