Paper 1070-2021

Using the %SCAN Function

Jonas V. Bilenas

ABSTRACT

In this presentation, we will look at the MACRO function %SCAN. It is similar to the SCAN function in the DATA step, but is useful in macro code. We will look at two examples that use the %SCAN function. The first example will show you how to set up a macro that forces all data set variables to be capitalized. The second example will show how to split SGSCATTER plots into multiple plots.

INTRODUCTION

The %SCAN function pulls out parts of a macro variable that are delimited by special characters and uses those parts in a data step or in new macro variables. The %SCAN function comes in handy when you are running macro code that involves many variables, processed one at time. You could run that macro many times for each variable that you specify as a macro variable. But why not let the %SCAN function and some %DO loops handle the solution.

DETAILS OF THE %SCAN FUNCTION

The %SCAN function processes a macro variable that has a number of elements that you want to extract, one at a time. Each element needs to be extracted by specifying the delimiter character. Most of the time a blank delimiter is used but if an element has spaces, a comma is generally used. The default delimiters you can specify are generally dependent on the platform you using. Using SAS 9.4/Viya 3.5 SAS, Support Documentation provides these default delimiters:

- For ASCII computer characters:
  - blank ! $ % & ( ) * + , - . / ; < ^¦
- For EBCDIC:
  - blank ! $ % & ( ) * + , - . / ; < ¬ | ¢¦

The %SCAN function has 2-4 arguments

- First argument is the macro variable that you will parse.
- Second argument is the part number. 1=first part, 2= second, etc.
- The 3rd argument defines the delimiter that separates the parts of the macro variable. The examples used in this presentation use the blank as the delimiter.
- The 4th argument is optional. It is a **modifier** that specifies special codes to handle more specific needs. There are many specified in the SAS Support Documentation. There is a special modifier that would pull from right to left as opposed form left to right.

EXAMPLE 1: USING %SCAN TO FORCE ALL VARIABLES TO BE CAPITALIZED

A few years ago, I received an email from a colleague asking if there was an automatic way or a MACRO that can capitalize all SAS Variable names without having to specify each variable in the program code with a RENAME statement. When you run a PROC CONTENTS the output will sort the output based on uppercases letter before lowercase letters. Let's see a test case application. First, create the test data and then run a PROC CONTENTS:

```
DATA test;
  AAA2 = 5; bbb4 = 6; CC4 = 7; ddd_2 = 8; xYz44 = 'BB';
  OUTPUT;
RUN;

PROC CONTENTS DATA = test;
RUN;
```

Display 1 shows a piece of the PROC CONTENTS. Notice that variable **CC4** shows up before **bbb4**. If you have many variables in the data set it may be difficult to find a specific variable you are looking for unless all variable names are listed with capital letters.

Alphabetic List of Variables and Attributes

#	Variable	Type	Len
1	AAA2	Num	8
3	CC4	Num	8
2	bbb4	Num	8
4	ddd_2	Num	8
5	xYz44	Char	2

Display 1. Results of the PROC CONTENTS on the Test Code above

THE %SCAN AND MACRO CODE CORRECTS THE ISSUE

```
%MACRO UPCASEV(LIB, DS);
PROC CONTENTS DATA=&LIB..&DS. OUT=CNTS NOPRINT;                    /* 1*/
RUN;

PROC SQL NOPRINT;
  SELECT NAME INTO: VARS SEPARATED BY ' '                          /* 2*/
           FROM CNTS;
QUIT;
%PUT VARS=&VARS.;                                                  /* 3*/

%LET I = 1;                                                        /* 4*/
PROC DATASETS LIB=&LIB. NOLIST; MODIFY &DS.;
  %DO %UNTIL(%SCAN(&VARS,&I.,%STR( ) ) = %STR( ) );                /* 5*/
    %LET VAR=%SCAN(&VARS,&I.,%STR( ) );                            /* 6*/
    %LET REN=%UPCASE(&VAR.);                                       /* 7*/
    %IF &VAR. NE &REN %THEN %DO;                                   /* 8*/
      RENAME &VAR. = &REN.;                                        /* 9*/
    %END;
    %LET I = %EVAL(&I. + 1);                                       /*10*/
  %END; /* END FIRST LOOP */
RUN;QUIT;

PROC CONTENTS DATA=&DS.;
RUN;
%MEND UPCASEV;
```

Let’s look at some of the logic in the code. We will refer to each of the numbered comments in the code above.

1. The NOPRINT option in PROC CONTENTS does not generate the PROC CONTENTS. The OUT=CNTS creates a data set that contains metadata for the data selected by the DATA= statement.
2. The SQL creates a macro variable, called VARS with all variable names separated by a space.
3. If you include the line labeled /* 3*/, you can see in the log, the macro variable created. For this example: **VARS=AAA2 CC4 bbb4 ddd_2 xYz44**
4. We will be doing some macro looping with the %LET, %DO, and %END statements. We start with the &I set to 1.
5. Line 5 includes the %SCAN function with 3 arguments. The %DO %UNTIL specifies when to stop the %DO %UNTIL loops. The 3 arguments we specify in lines 5 and 6 are:
    a. which macro variable will be pulled out from the macro variable &VARS
    b. the part number (starting with **&I=1**)
    c. until a blank space is identified with **`%STR( )`**. Note that a blank space, many blank spaces and/or new lines will satisfy as %STR( ).
6. This line will create the macro variable called VAR which extracted from VARS with the %SCAN function again.
7. We create a new macro variable called REN as containing all uppercase letters fron the macro variable VAR.
8. If &VAR not equal to &REN then %DO
9. In PROC DATASETS rename, RENAME &VAR as &REN. See Raithel. Michael A. (2010) for more information on PROC DATASETS.
10. In line 10 we increment &I by 1. Make sure you do this somewhere in your code so that the %DO %UNTIL in line /* 5*/ will stop when all macro parts are extracted. Otherwise your code will keep running and running.

Run the macro code as shown:

```
%UPCASEV(LIB=WORK, DS=TEST);
```

Display 2 shows the results of the project.

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	AAA2	Num	8
2	BBB4	Num	8
3	CC4	Num	8
4	DDD_2	Num	8
5	XYZ44	Char	2

Display 2. All variable names are now all uppercase.

EXAMPLE 2: USING %SCAN AND SGSCATTER TO SPECIFY HOW MANY PLOTS ARE SHOWN SHOULD BE GENERATED FOR EACH CALL OF SGSCATTER

For this example, we will use the **SASHELP.BASEBALL** data that is available on most SAS platforms. Let's say we want to build a multiple linear regression model that would predict the Dependent Variable (DV), **CRATBAT** (Career Times at Bat). The data contains 17 Independet Variables (IVs). You want to do some exploratory work looking at the relationship between each IV and the DV to check out if the variables make sense on a bivariate level (DV and each IV_i) and also check for nonlinearity that would suggest transformations. We can try to use PROC SGSCATTER but if we include all 17 variables in a single PLOT statement the output will not look that good since all 17 plots would be included in a single output. Here is the program we tried and the resulting output:

```
%LET LIB=SASHELP; %LET DS=BASEBALL;
%LET DV=CRATBAT;  /* CAREER TIMES AT BAT */

PROC CONTENTS DATA=&LIB..&DS. NOPRINT OUT=CNTS; RUN;

PROC SQL NOPRINT;
  SELECT NAME
    INTO: VARS SEPARATED BY ' '
      FROM CNTS
        WHERE TYPE=1 AND NAME NOT EQ "&DV."
  ;
QUIT;
PROC SGSCATTER DATA=&LIB..&DS.;
     PLOT  &DV.*(&VARS.)
     / MARKERATTRS=(SIZE=2 COLOR=BLACK) GRID
      LOESS=(SMOOTH=0.5)
      ROWS=2 COLUMNS =3;
RUN;
```

Results from the code is shown in Figure 1. Hard to see what is going on. We need to break out the many variables we will SCAN and generate a number of PROC SGSCATTER calls. The problem can probably be resolved using PROC TEMPLATE but can using the %SCAN function and a number of %DO loops solve the issue?

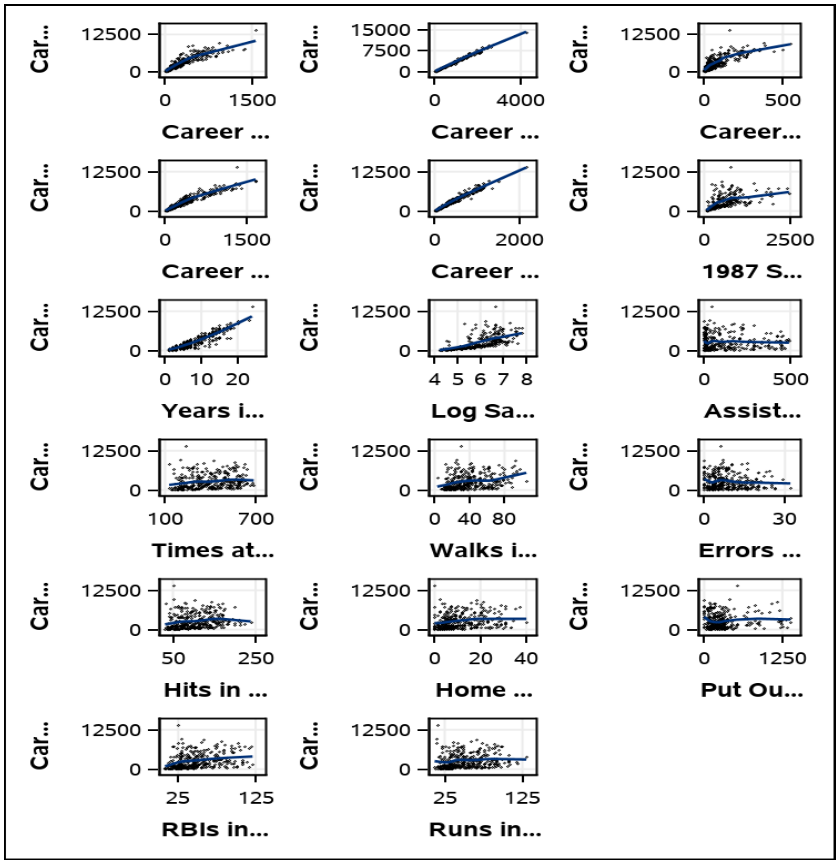


Figure 1. Results the code above using PROC SGSCATTER.

We have a modified code that will generate 5 SGSCATTER plots by using the %SCAN function along with %DO loops. The output of the graphs can be saved using various ODS (Output Delivery System) selections. The code will have a number of macro variables. The first part of the code will generate the macro variable we will %SCAN. The first section will include 5 macro variables:

1. **NPLOTS**: How many plots to include in each SGSCATTER output.
2. **ROWS**: How many rows for each output.
3. **LIB**: LIBNAME.
4. **DS**: Data Set Name.
5. **DV**: The Dependent Variable that you are interested in predicting.

```
%LET NPLOTS=4; %LET ROWS=2;
%LET LIB=SASHELP; %LET DS=BASEBALL;
%LET DV=CRATBAT;
PROC CONTENTS DATA=&LIB..&DS. NOPRINT OUT=CNTS; RUN;

PROC SQL NOPRINT;
  SELECT NAME
    INTO: VARS SEPARATED BY ' '
      FROM CNTS
        WHERE TYPE=1 AND NAME NOT EQ "&DV."
  ;
QUIT;
```

Notice that in the SQL code that we have 2 constraints to create the macro variable, &VARS:

1. TYPE=1: We want only numeric variables selected.
2. AND NAME NOT EQ "&DV.": The &DV in this example is a single variable, CRATBAT, we would like to predict. We will generate bivariate plots with the Y-axis being CRATBAT and each individual numeric independent variable on the X-axis. Having the Y-axis = X-axis is ok but we don't need it since we know CRATBAT=CRATBAT.

The next code will create a user MACRO called **PLOTIT** that, for this example, generate 5 SGSCATTER plots based on the **NPLOTS** macro variable. Rounding up to integer (17/5) = 5. Code can be a bit tricky. If your code is running too long it may indicate that you are not incrementing the second argument of the %SCAN function correctly and may need to abort the SAS code.

Note that the LOESS option in the code is a local weighted regression smoothed function to detect non-linearity in (OLS) ordinary least squares models. See Cleveland, W. S., Devlin, S. J., and Grosse, E. (1988) and Cleveland, W. S. and Grosse, E. (1991).

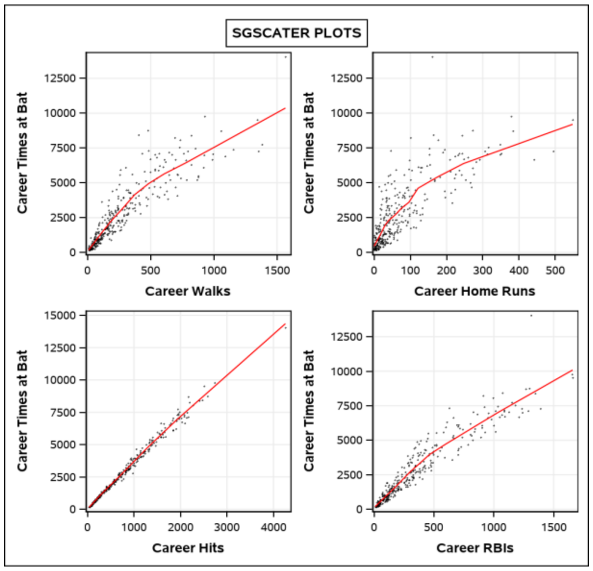
Here is the MACRO code, MACRO call:

```
%MACRO PLOTIT;
%LET I = 1; /* INITIALIZE &I. */

%DO %UNTIL(%SCAN(&VARS,&I.,%STR( ) ) = %STR( ) );
  %LET VAR=%SCAN(&VARS,&I.,%STR( ) );
  %DO PLT_STREAM= 1 %TO &NPLOTS;
    %LET PLT_&PLT_STREAM. = %SCAN(&VARS,&I.,%STR( ) );
    %LET I = %EVAL(&I. + 1);
  %END;

PROC SGSCATTER DATA=&LIB..&DS.;
  PLOT  &DV. * (
  %DO PLT_STREAM= 1 %TO &NPLOTS;
        &&PLT_&PLT_STREAM.
  %END; )
      /  MARKERATTRS=(SIZE=2 COLOR=BLACK) GRID
         LOESS=(SMOOTH=0.5
         LINEATTRS=(COLOR=RED THICKNESS=.5))
         ROWS=&ROWS.
         ;
         TITLE BOLD BOX=1 "SGSCATER PLOTS";
 RUN;
%END;
%MEND; /* END MACRO */
%PLOTIT;
```

The results using the above 2 codes will be shown in Figures 2-4. Of course, you can put the first code as part of the **PLOTIT** macro. Include all 5 macro variable names and include the PROC CONTENTS and PROC SQL before the first %LET I statement.



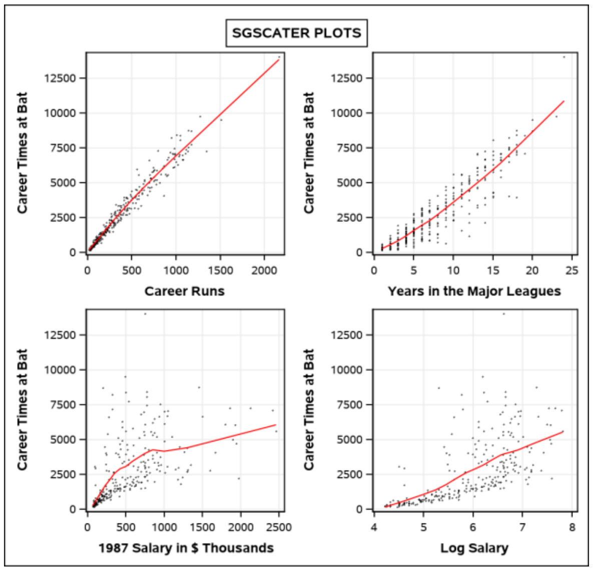
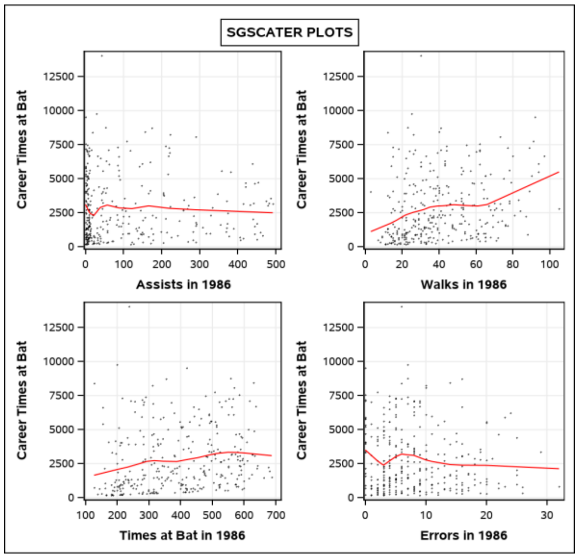


Figure 2. 2 SGSCATTER PLOTS of 4 PLOTS



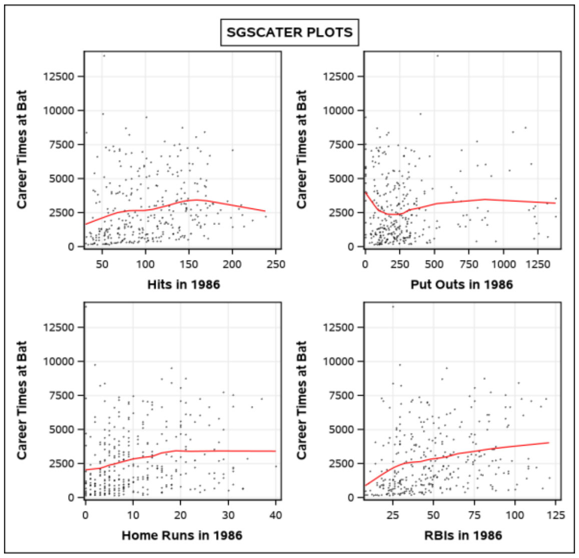


Figure 3. Additional PLOTS

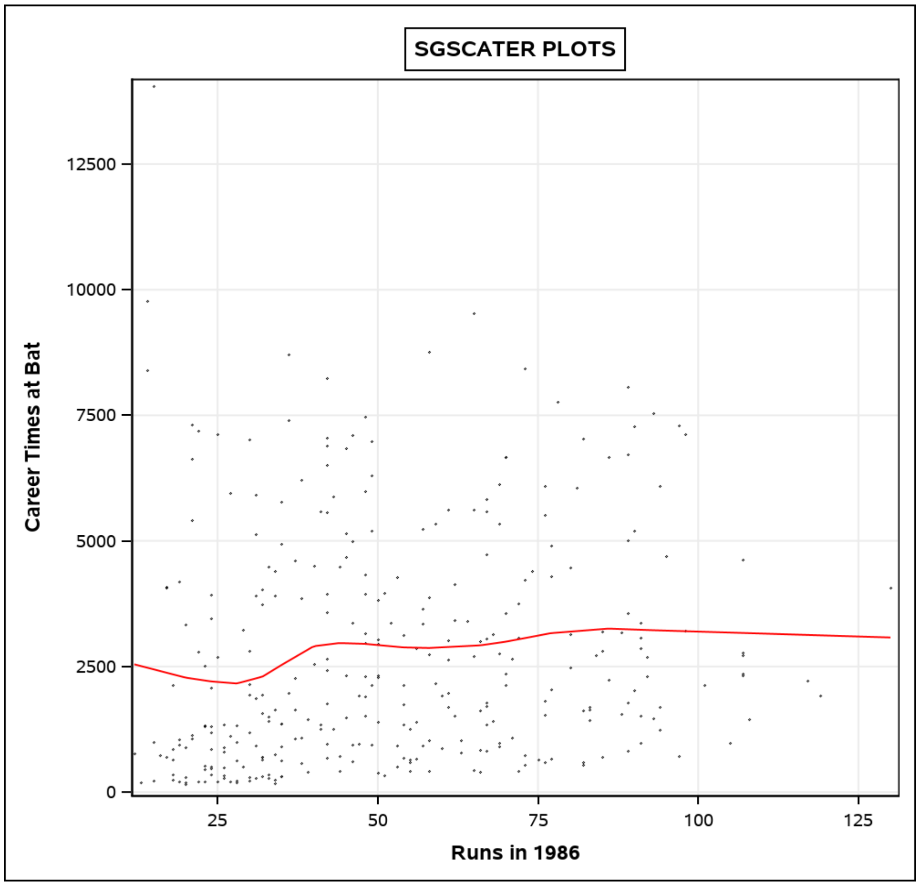


Figure 4. Final Plot

Figure 6. The final output of all explorations of 17 Independent Variables predictive influence on the Dependent Variable.

CONCLUSION

With SAS, there are many ways to tackle and solve the problem you are dealing with. Some solutions may not be that efficient but at least you get the problem solved.

REFERENCES

Cleveland, W. S., Devlin, S. J., and Grosse, E. (1988), "Regression by Local Fitting," Journal of Econometrics, 37, 87–114.

Cleveland, W. S. and Grosse, E. (1991), "Computational Methods for Local Regression," Statistics and Computing, 1, 47–62.

Raithel. Michael A. (2010). PROC DATASETS; The Swiss Army Knife of SAS® Procedures. SAS Global Forum 2010. https://support.sas.com/resources/papers/proceedings10/138-2010.pdf

ACKNOWLEDGMENTS

Thanks to all the SAS users, SAS statisticians, SAS gurus, and colleagues whom I have learned the "Power of SAS" over the many years I have been using SAS.

RECOMMENDED READING

Carpenter, Arthur L. (2005). Storing and Using a List of Values in a Macro Variable. SAS USERS GROUP INTERNATIONAL (SUGI) 30. https://support.sas.com/resources/papers/proceedings/proceedings/sugi30/028-30.pdf

Carpenter, Arthur L. (2016). Carpenter's Complete Guide to the SAS® Macro Language, Third Edition, SAS Institute.

Carpenter, Arthur L. (2017). Building Intelligent Macros: Using Metadata Functions with the SAS® Macro Language. https://support.sas.com/resources/papers/proceedings17/0835-2017.pdf

Delwiche, Lora D. and Slaughter, Susan J. (2012). Graphing Made Easy with SG Procedures. SAS Global Forum 2012. https://support.sas.com/resources/papers/proceedings12/259-2012.pdf

Lafler, Kirk Paul (2019). PROC SQL: Beyond the Basic Using SAS, Third Edition, SAS Institute

CONTACT INFORMATION <HEADING 1>

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

Jonas V. Bilenas
jonas@jonasbilenas.com
https://jonasbilenascom.wpcomstaging.com/

The contents of this paper are the work of the author and do not necessarily represent the opinions, recommendations, or practices of any company that I have worked for or are currently working for.

There is no warranty on the code illustrated in this presentation.