

## Easy Solutions To %LET You Reduce Repetitive Programming

Ekaterina Roudneva, University of California, Davis, CA

### ABSTRACT

Cleaning data and generating reports can sometimes involve using the same pieces of information or code over and over again. This can consist of running similar programs one at a time in a specific sequence or updating multiple macro variables or fields. Related programs might need to import the same exact data resulting in that data being read in multiple times. These approaches are redundant, prone to errors, and can be very time consuming. By using macro variables or recognizing similarities in the code and condensing them into fewer programs, it is possible to increase efficiency and reduce the need for copy and pasting. This paper will go over some techniques that help reduce repetitive tasks by recognizing parts of code that can be condensed. Topics include using %LET and PROC SQL SELECT INTO statements to create macro variables and use them to reference commonly used dates, folder paths, dataset names and variables. It will also discuss how to transform these macro variables to accommodate different formats such as alternate date styles. The last topic will examine using the %INCLUDE statement to run multiple programs from one main program, eliminating the need to run those programs one at a time.

### INTRODUCTION

It is easy to find yourself creating a program and then later having to modify it to update various reports or frequency outputs, or to clean additional data. This task can potentially involve manually modifying the date of the report each time you run it. It is also common to run similar pieces of code for different variables or datasets with only slight modifications. Many of these manual steps can be automated. This paper will go over some techniques to help reduce the need to modify and rewrite programs and condense them using macros and macro variables. These together are very powerful tools of SAS that can help to reduce human error and make programming faster and easier. Because macros are a form of metaprogramming, they can seem complicated to beginning programmers. This paper will introduce some key concepts of macros that can be used and easily applied even for users new to SAS macro programming. It is intended to help teach useful macro writing techniques as well as show tips and tricks on what to look for in your code to help you automate it.

### MACROS AND MACRO VARIABLES

SAS macros are a type of function that allows a SAS programmer to create data driven solutions within their SAS code. They can be used for to invoke specific code, or generate conditional statements, or make substitutions.

Macro variables are, in the context of this paper, user defined variables that are stored in SAS memory and that can be used within a program. They can have global or local scope and are character strings, though when used they can be converted to another type if they are in a format that SAS would normally recognize. For example, if you want to store a value of '5' in a variable it will be stored as a string but can be used as a number when calling the variable.

Macro variables exist outside of datasets and can be set to a specific value of text. These variables can then be referenced throughout the code and represent specific text that is generated by SAS when that macro variable is referenced. Global macro variables are macro variables that are defined outside of macros and that exist throughout the SAS session and can be referenced anywhere. Local macro variables are defined within a macro and then can only be used within the macro that they were defined in. This paper will cover global variables although all the material can also be applied to local macro variables.

## CREATING AND USING MACRO VARIABLES

### WHY USE MACRO VARIABLES?

Macro variables allow you to reduce redundant programming. Instead of having to update a piece of text in 10 different places, you can change it once and let SAS substitute it making updates or changes a lot easier.

### CREATING MACRO VARIABLES USING %LET STATEMENT

The first thing to think about when writing programs is setting them up so that they can be easily updated. One of the basic functions that can be used to accomplish this is creating commonly used macro variables with the %LET statement. We use %LET statements to assign a value to a macro variable. Quotation marks are not necessary in the assignment because SAS sees everything between the '=' and ';' in a %LET statement as the variable value.

```
%LET macro-variable-name = value;
```

Start your program by defining paths, dates, libraries and other variables that are likely to be used in more than one context. This way they can be referred to easily and can be updated or altered by changing the information at the top of the program instead of having to go through and manually change multiple parts of the program.

```
LIBNAME sasdata "C:\Users\eroudneva\Desktop\Project\SAS Data";  
%LET outdate=25oct18; *Output date;  
%LET outpath=C:\Users\eroudneva\Desktop\Project\Output; *Path for report  
output;  
%LET dataset=sashelp.countseries;
```

Once a macro variable is defined it can be referenced by adding a prefix '&' in front. You can use a %PUT statement to view the content of a macro variable. You can also use %PUT \_USER\_ to view all user-created macro variables.

```
%PUT &outdate;  
%PUT &outpath;  
%PUT _user_;
```

Output 1 shows the log output.

```
26  %PUT &outdate;  
25oct18  
27  %PUT &outpath;  
C:\Users\eroudneva\Desktop\Project\Output  
28  %PUT _user_;  
GLOBAL OUTDATE 25oct18  
GLOBAL OUTPATH C:\Users\eroudneva\Desktop\Project\Output  
GLOBAL SYSDBMSG  
GLOBAL SYSDBRC 0
```

**Output 1. Log Output From a %PUT Statement of User Defined Macro Variables**

If you want to add a text string to a macro variable, a period "." determines the end of the macro variable. You then follow it with the text string you wish to add.

```
%LET pet=Dog;  
Proc print data=&pet.food; Run;  
*This statement will print dataset called dogfood;
```

Use double quotation marks to resolve macro variables inside quotations. Macro variables inside single quotation marks will not be resolved.

```
%LET pet=Dog;
%PUT "&pet."; *Output = "Dog";
%PUT '&pet.'; *Output = '&pet.';
```

## AUTOMATIC MACRO VARIABLES

Some macro variables are predefined in SAS. For example, &SYSDATE resolves to the date of the current SAS session in DDMONYY format (Example: 26OCT19). Table 1 Shows some automatic macro variables.

Automatic Macro Variable	Description	Example
SYSDATE	Current SAS Session start date (DDMONYY format)	28OCT18
SYSDATE9	Current SAS session start date (DDMONYYYY format)	28OCT2018
SYSDAY	Current SAS session start day of the week	Tuesday
SYSTIME	Current SAS session start time	16:20

**Table 1. Some Examples of Automatic Macro Variables**

## USING AND MANIPULATING MACRO VARIABLES

Dataset COUNTSERIES from SASHELP library shown in Display 1 will be used to demonstrate how to use and manipulate macro variables.

	Date	Units
99	01MAR2012	4
100	01APR2012	0
101	01MAY2012	3
102	01JUN2012	2
103	01JUL2012	0
104	01AUG2012	9
105	01SEP2012	4
106	01OCT2012	5
107	01NOV2012	0
108	01DEC2012	2

**Display 1. View of Last 10 Observations in sashelp.countseries Dataset**

In a situation where you need to have a specific date at the top of a PROC FREQ procedure, you can output the report title with a defined macro variable. Output 2 shows PROC FREQ procedure output using a title with user defined *outdate* macro variable

```

%LET outdate=25oct18;
proc freq data= sashelp.countseries;
tables date*units/nocol norow nopercent;
format date year4.;
title "Report as of &outdate.";
run;

```

**Report as of 25oct18**

**The FREQ Procedure**

Table of Date by Units													
Date	Units											Total	
	0	1	2	3	4	5	6	7	8	9	10		11
2004	6	0	0	0	3	1	0	0	0	1	1	0	12
2005	3	0	1	2	1	3	0	1	0	1	0	0	12

**Output 2. PROC FREQ Output Using a Title with User Defined Date Macro Variable**

If you need the date in another format instead of manually changing it, you can change it via a %SYSFUNC statement. Since Macro variables are always stored as text, and dates in SAS are numeric, in order to convert from one date format to another you need to first convert to a numeric value and then to a different format. For example, to change date text value to a different format you need to first convert original format *date7.* to a numeric value using INPUTN and then change the format, in this case *worddate.* Output 3 shows the resulting title after changing the macro variable format.

```

%LET outdate_word=%sysfunc(INPUTN(&outdate., date7.), worddate.);
%PUT &outdate_word.;

PROC FREQ data= sashelp.countseries;
TABLES date*units/nocol norow nopercent;
format date year4.;
title "Report as of &outdate_word.";
run;

```

**Report as of October 25, 2018**

**The FREQ Procedure**

Table of Date by Units													
Date	Units											Total	
	0	1	2	3	4	5	6	7	8	9	10		11
2004	6	0	0	0	3	1	0	0	0	1	1	0	12
2005	3	0	1	2	1	3	0	1	0	1	0	0	12

**Output 3. PROC FREQ Output After Changing Macro Variable Date Format**

In addition to changing date and numeric formats, it is possible to use other functions to change macro variables. Table 2 lists some examples of functions for macro variables.

Macro Variable Functions	Description	Example	Result
%LOWCASE	Converts to low case	%LET pet=Cat; %PUT %LOWCASE(&pet);	cat
%UPCASE	Converts to upper case	%LET pet=Cat; %PUT UPCASE(&pet);	CAT
%CMPRES	Removes extra spaces	%LET pet=Black      Cat; %PUT %CMPRES(&pet)	Black Cat
%LENGTH	Returns length of macro variable	%LET pet=Cat; %PUT %Length(Cat);	3
%SUBSTR	Extract substring	%LET pet=Cat; %PUT %substr(&pet,1,2)	Ca
%SYSFUNC	Executes SAS functions.	%let datevar=19328; %PUT %sysfunc(year(&datevar.));	2012
	Use with INPUTN to convert one format to another format	%let datevar=28oct18; %PUT %sysfunc(INPUTN(&datevar., Date7.), worddate.);	October 28, 2018
	Use with PUTN to convert a numeric value to a formatted character value	%LET numdate=19328; %PUT %SYSFUNC(PUTN(&numdate.,mmdyy8.));	12/01/12

**Table 2 Examples of Useful Functions for Macro Variables**

## CREATING AND USING MACRO VARIABLES USING PROC SQL SELECT INTO STATEMENT

Sometimes you may need to create a macro variable from a specific dataset. For example, suppose we want the maximum date in COUNTSERIES dataset in the report title. One method involves using PROC SQL, the SELECT statement and the INTO clause to create a macro variable *maxdate* and store it as a text string.

```
proc sql noprint;
select max(date) into :maxdate
from sashelp.countseries;
quit;
%put &maxdate;
```

Using a %PUT statement allows us to see that macro variable *maxdate* was assigned a value of "19328". Dates in SAS are stored as numbers - the number of days from January 1, 1960. To show the date as a word, we need to convert it to a character value. Use %SYSFUNC and PUTN to convert from a numeric to a character format. Output 4 shows the log output after changing to a formatted value.

```
%let maxdate_word=%sysfunc(Putn(&maxdate., worddate.));
%put &maxdate_word.;
```

```
11558 %put &maxdate_word. ;
December 1, 2012
```

**Output 4. Log Output Showing Value of maxdate\_word Macro Variable**

Now we can use created macro variable *&maxdate\_word* in the report title to have the latest date in the title as shown in Output 5.

```
proc freq data= sashelp.countseries.;
tables date*units/nocol norow nopercnt;
format date year4.;
title "Report as of &maxdate_word.";
run;
```

**Report as of December 1, 2012**

The FREQ Procedure

Frequency	Table of Date by Units												
	Date	Units											Total
		0	1	2	3	4	5	6	7	8	9	10	
2004	6	0	0	0	3	1	0	0	0	1	1	0	12
2005	3	0	1	2	1	3	0	1	0	1	0	0	12
2006	1	0	1	1	2	3	2	0	1	0	1	0	12
2007	5	0	0	2	2	0	2	0	1	0	0	0	12
2008	4	0	1	1	0	3	1	1	1	0	0	0	12
2009	4	1	0	2	2	1	0	0	0	1	0	1	12
2010	2	1	1	1	0	2	3	0	2	0	0	0	12
2011	2	2	0	2	2	2	1	1	0	0	0	0	12
2012	4	0	3	1	2	1	0	0	0	1	0	0	12
<b>Total</b>	<b>31</b>	<b>4</b>	<b>7</b>	<b>12</b>	<b>14</b>	<b>16</b>	<b>9</b>	<b>3</b>	<b>5</b>	<b>4</b>	<b>2</b>	<b>1</b>	<b>108</b>

**Output 5. PROC FREQ Output That Includes Latest Date in Title**

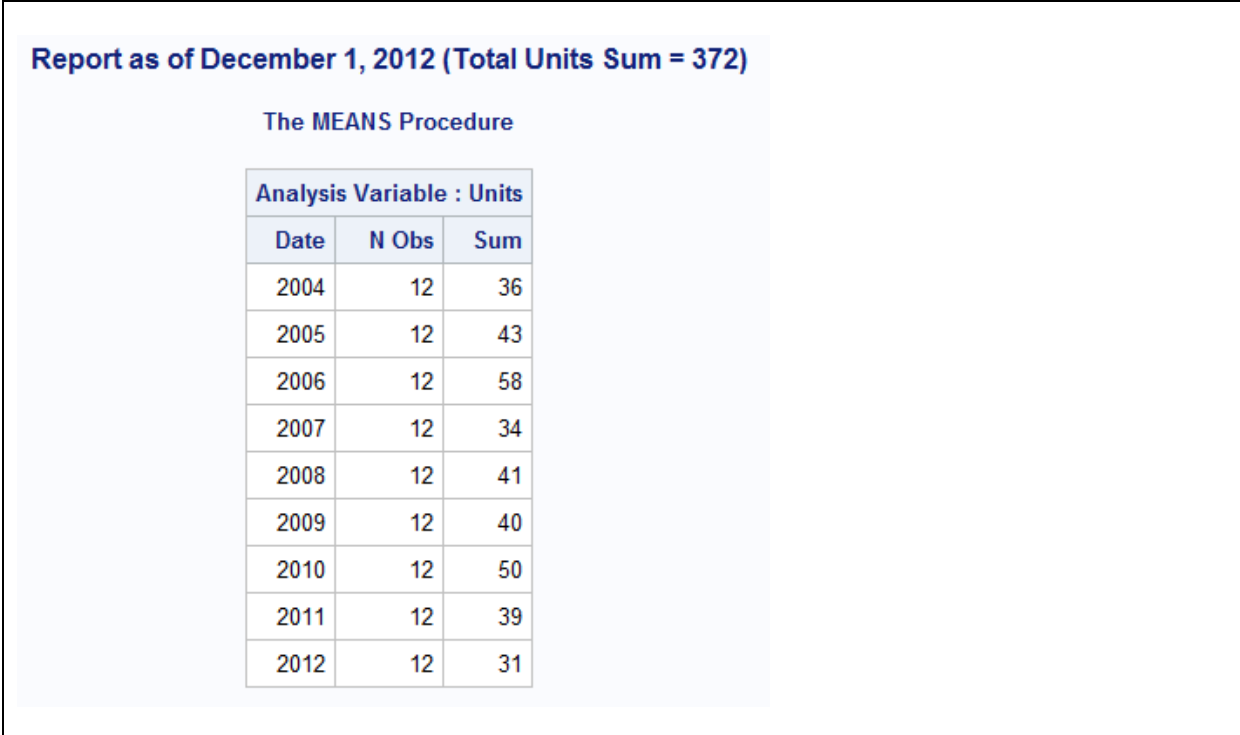
In addition to being able to get max or min values, you can also get sums and other counts using PROC SQL. We use a PROC SQL SELECT INTO Statement to get the total sum of units into a macro variable called *&unitssum*.

```
proc sql noprint;
select sum(units) into :unitssum
from sashelp.countseries;
quit;
%put &unitssum;
```

Now we can use this created macro variable in a PROC MEANS Statement.

```
proc means data=countseries sum maxdec=0;
var units;
class date;
format date year4.;
title "Report as of &maxdate_word. (Total Units Sum = &unitssum.)";
run;
```

Output 6 shows resulting table title



The screenshot displays the output of a PROC MEANS procedure. At the top, the title reads "Report as of December 1, 2012 (Total Units Sum = 372)". Below this, the text "The MEANS Procedure" is centered. A table titled "Analysis Variable : Units" follows, with columns for "Date", "N Obs", and "Sum". The data rows show values for each year from 2004 to 2012.

Analysis Variable : Units		
Date	N Obs	Sum
2004	12	36
2005	12	43
2006	12	58
2007	12	34
2008	12	41
2009	12	40
2010	12	50
2011	12	39
2012	12	31

Output 6. PROC FREQ Output That Includes Total Units Sum Value in Title

## Renaming variables using PROC SQL

A PROC SQL SELECT INTO statement can be used to create macro variables that contain lists of values. Take the following example where we want to rename all variables in a dataset to have the suffix `_new`. Renaming variables can be a very time consuming task for datasets with a lot of variables. PROC SQL SELECT INTO statement can be used in this case to easily and quickly rename all variables with minimal time.

Here is an example of variable names from the dataset SHOES in SASHELP library. We can get a list of variable names using `dictionary.columns` dataset which contains metadata information on all SAS datasets.

```
proc sql noprint;
select name INTO :original_vars separated by ' '
from dictionary.columns
where libname='SASHELP' and memname='SHOES'
;
quit;
%put &original_vars.;
```

The *original\_vars* macro variable now has the text string “Region Product Subsidiary Stores Sales Inventory”. To rename variables in the format of var = var\_new we use a CATS function.

```
proc sql noprint;
  select cats(name, '=', cats(name, "_new"))
         into :rename_list separated by ' '
  from dictionary.columns
  where libname='SASHELP' and memname='SHOES'
  ;
quit;
%put &rename_list.;
```

Macro variable *rename\_list* will have value of “Region=Region\_new Product=Product\_new Subsidiary=Subsidiary\_new Stores=Stores\_new Sales=Sales\_new Inventory=Inventory\_new Returns=Returns\_new”. Now these lists can be used in a DATA step to rename all the variables in the dataset *sashelp.shoes*.

```
data Shoes_renamed; set sashelp.shoes;
  rename &rename_list.;
run;
```

Checking PROC CONTENTS in Output 7 confirms that all variables were renamed to include the suffix *\_new*.

```
proc contents data=shoes_renamed; run;
```

Alphabetic List of Variables and Attributes						
#	Variable	Type	Len	Format	Informat	Label
6	Inventory_new	Num	8	DOLLAR12.	DOLLAR12.	Total Inventory
2	Product_new	Char	14			
1	Region_new	Char	25			
7	Returns_new	Num	8	DOLLAR12.	DOLLAR12.	Total Returns
5	Sales_new	Num	8	DOLLAR12.	DOLLAR12.	Total Sales
4	Stores_new	Num	8			Number of Stores
3	Subsidiary_new	Char	12			

**Output 7. PROC CONTENTS Output of Shoes Dataset with New Variable Names**



## USING MACROS AND %INCLUDE STATEMENT

### MACROS

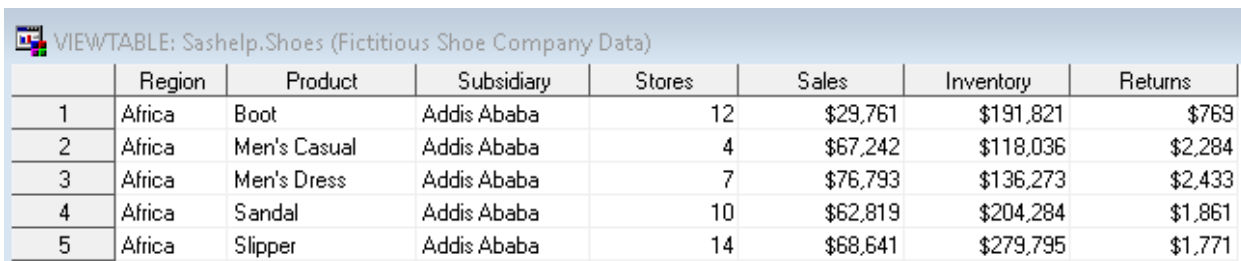
Macros are user defined functions that can be called at any point in your code after their definition. They can be very useful in avoiding having to copy and paste similar code or in writing powerful programs that automate a lot of tedious work in writing or updating code. Defining a macro requires a %MACRO statement in the beginning to start the macro and a %MEND statement to end the definition. To call a macro, put a % in front of the name of the macro.

```
%MACRO macro-name;  
    macro-text;  
%MEND;  
%macro-name
```

SAS Macros can contain functional logic. For example, they can include parameters, macro variables, do loops and condition statements.

```
%MACRO DoLoop (Count) ;  
    %do i=1 %to &count. ;  
        macro-text ;  
    %end ;  
%MEND ;  
%DoLoop (Count=9) ;
```

Suppose we want to generate a separate PDF report for sales by product for each region in the dataset SHOES in SASHELP library. A view of this dataset is shown in Display 2. Without using macros, we would need to create code ten times for the ten different regions. But by using macros we can get all ten reports from one piece of code. Additionally, if we want to make changes to the report we only have to do it once instead of ten times.



	Region	Product	Subsidiary	Stores	Sales	Inventory	Returns
1	Africa	Boot	Addis Ababa	12	\$29,761	\$191,821	\$769
2	Africa	Men's Casual	Addis Ababa	4	\$67,242	\$118,036	\$2,284
3	Africa	Men's Dress	Addis Ababa	7	\$76,793	\$136,273	\$2,433
4	Africa	Sandal	Addis Ababa	10	\$62,819	\$204,284	\$1,861
5	Africa	Slipper	Addis Ababa	14	\$68,641	\$279,795	\$1,771

**Display 2. View of Dataset sashelp.Shoes**

Output 8 shows a PROC FREQ output of the frequency of different regions in the dataset.

```
proc freq data=sashelp.shoes;  
    tables region;  
run;
```

Region	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Africa	56	14.18	56	14.18
Asia	14	3.54	70	17.72
Canada	37	9.37	107	27.09
Central America/Caribbean	32	8.10	139	35.19
Eastern Europe	31	7.85	170	43.04
Middle East	24	6.08	194	49.11
Pacific	45	11.39	239	60.51
South America	54	13.67	293	74.18
United States	40	10.13	333	84.30
Western Europe	62	15.70	395	100.00

### Output 8. PROC FREQ of Region Variable in Dataset sashelp.Shoes

We can create macros that help us eliminate the need for repetitive code by identifying and isolating the changing variable – in this case region and making it its own macro variable.

```
%Macro Region(Region);
    proc sgplot data=sashelp.shoes;
        vbar product / response=sales;
        where region="&Region.";
        title "&Region. Shoes Total Sales";
    run;
%Mend;
%Region(Region=Africa);
%Region(Region=Asia);
```

We can make the process even more automated by making the code dynamic. In this example we use PROC SQL SELECT INTO to create macro variables that contain all the possible regions. This way we can avoid having to type in the region name for all possibilities.

First we need to see how many different distinct regions are in the dataset. We can use PROC SQL SELECT INTO with a *distinct* count statement. SAS also creates a *sqlobs* macro variable that keeps track of the number of observations for the last run PROC SQL statement.

```
proc sql noprint;
select count(distinct region)
    into :RegionCount
    from shoes
;
quit;
%put Total number of regions:&RegionCount.;
```

Looking at the log in Output 9 shows us the value is 10, although with extra leading blanks.

```
98 %put Total number of regions:&RegionCount. ;
Total number of regions:      10
99
...      .      .
```

#### Output 9. Log Output of Regions Count

Use a compress statement to remove trailing and leading blanks from the macro variable. See Output 10.

```
%let RegionCount =%CMPRES(&RegionCount.);
%put Total number of regions:&RegionCount.;
```

```
106
107 %let RegionCount =%CMPRES(&RegionCount.);
108 %put Total number of regions:&RegionCount. ;
Total number of regions:10
```

#### Output 10. Log Output of Regions Count After Using %CMPRES Function to Remove Leading Spaces

Then we can create macro variables *Region1-Region10* using a PROC SQL SELECT INTO that contain all the possible region names. We can either specify an arbitrary high number (say 100) or use the previously created region count variable so that SAS will create macro variables Region1 through Region10.

```
proc sql noprint;
select distinct region
  into :Region1-:Region&RegionCount.
  from shoes
;
quit;
%put &region1.;
%put &region2.;
%put &region3.;
%put &region4.;
%put &region5.;
```

Or you can use %PUT \_user\_ to view all user generated macro variables;

```
%put _user_;
```

Here is Output 11 log for some of the region macro variables.

```

800 %put &region1.;
Africa
801 %put &region2.;
Asia
802 %put &region3.;
Canada
803 %put &region4.;
Central America/Caribbean
804 %put &region5.;
Eastern Europe
805 %put &region6.;
Middle East
806 %put &region7.;
Pacific

```

### Output 11. Log Output

Once the *Region* variables are created they can be used in macros along with a DO LOOP. Always check the log for errors as well as the output to make sure everything has run correctly. You may notice that *Region4* Central America/Caribbean has a slash in the name. In order to be able to generate a report for it, we must remove the spaces and slash from the region file name since file names cannot have slashes and should not have spaces to ensure compatibility.

```

%Macro RegionReports;
  %do i=1 %to &RegionCount.;
    *Keep only alphabetic characters for file region name;
    %let FileName=%SYSFUNC(compress(&&Region&i.,,ka));

    options nodate;
    ods _all_ close;
    ods pdf file="&outpath.\&FileName._Sales.pdf" startpage=no;
    proc sgplot data=sashelp.shoes;
      vbar product / response=sales;
      where region="&&Region&i.";
      title "&&Region&i. Shoes Total Sales";
    run;

    ods pdf close;
    ods html;
  %end;
%Mend;
%RegionReports;

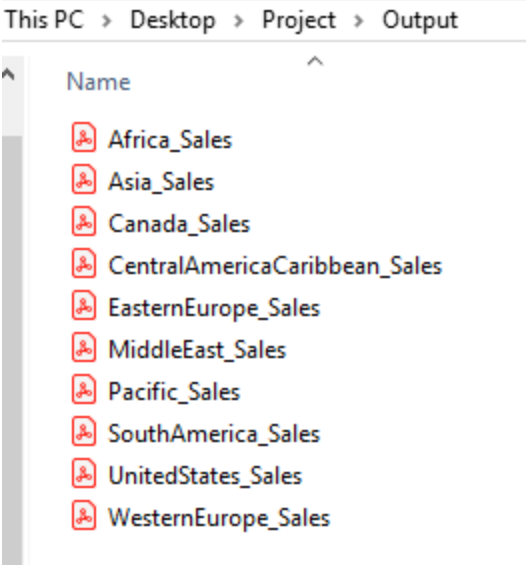
```

The following Table 3 shows how *&&Region&i.* macro variables resolve in the first few do loop steps.

Value of "i"	&&Region&i. Resolves to	Macro Variable Value
1	&Region1.	Africa
2	&Region2.	Asia
3	&Region3.	Canada

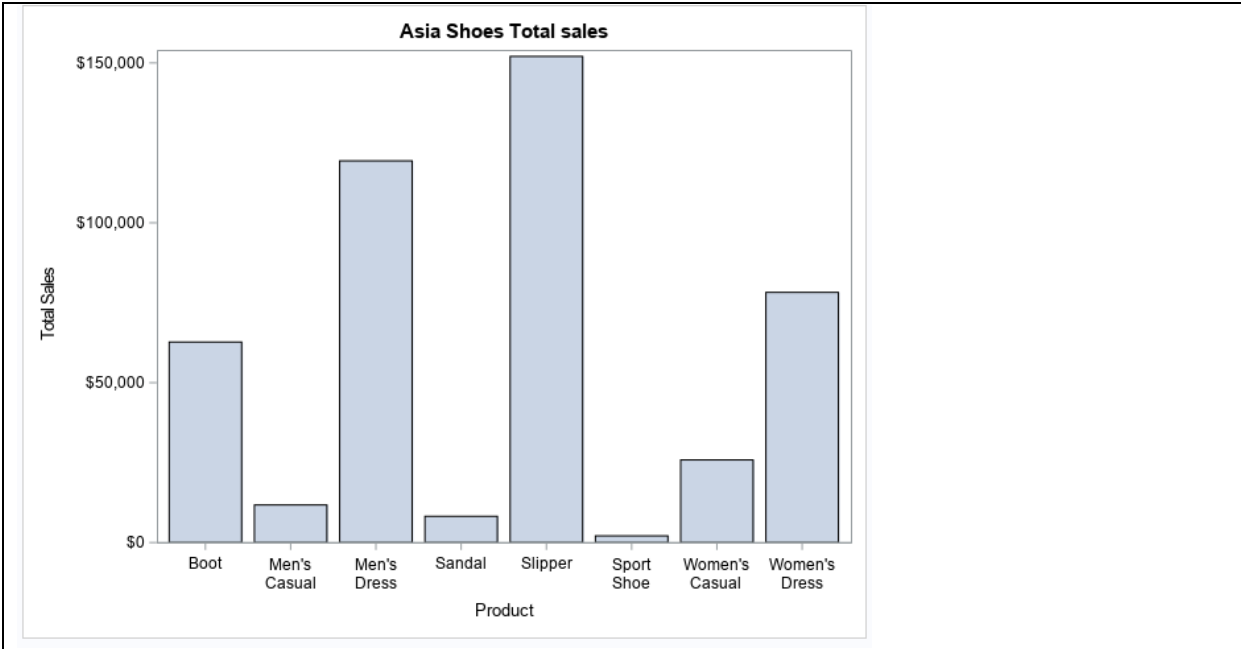
**Table 3. Value of *&&Region&i.* in first few do loop iterations**

Running the macro creates a total of 10 separate PDF reports. Display 3 shows the directory where the files were created.



**Display 3. Output For Macro PDF Report**

Output 12 shows an example of output for region “Asia”.



**Output 12. An Example of Report Output for Region “Asia”**

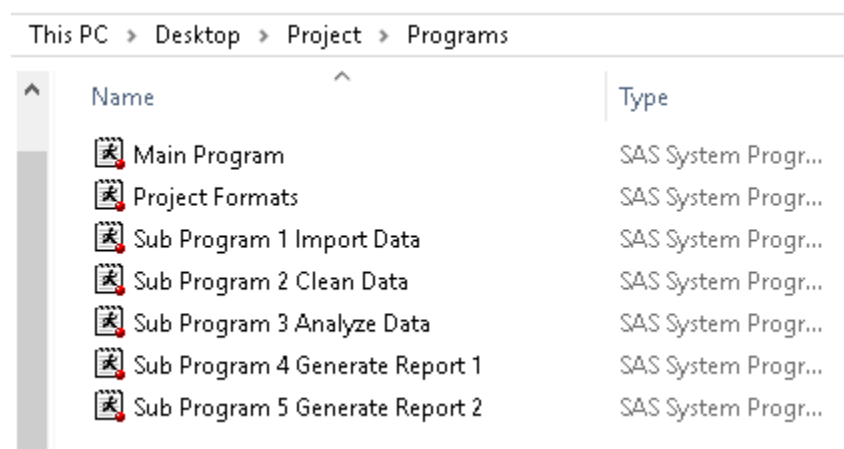
## %INCLUDE STATEMENT

An %INCLUDE Statement points to a file to execute. It can be used to run multiple SAS programs from one main program. It can also be used to include macro variables that are used across multiple programs or run a program that contains a commonly used macro or section of code.

One example of where this may be useful is when you need to run multiple SAS programs one at a time while also doing some outside manual manipulation. This process can be very time consuming. If you are not familiar with the process you may have to rely on comprehensive, step-by-step directions. If you miss a step, you may not know until later in the process, and then you will have to redo the process from the beginning.

Having a main program that points to multiple sub-programs to run in sequence can eliminate this time-consuming process. %INCLUDE can be used to run external SAS programs, or programs that create commonly used macros or formats.

Display 4 shows an example of program folder for a project.



Name	Type
Main Program	SAS System Progr...
Project Formats	SAS System Progr...
Sub Program 1 Import Data	SAS System Progr...
Sub Program 2 Clean Data	SAS System Progr...
Sub Program 3 Analyze Data	SAS System Progr...
Sub Program 4 Generate Report 1	SAS System Progr...
Sub Program 5 Generate Report 2	SAS System Progr...

**Display 4. Program Directory**

This project has a total of 6 programs that import raw data, clean data, run analysis, create formats and generate various reports. They have to be run in a specific order and sometimes may require additional steps. In order to remove the need to update all 6 programs to generate a report, we run all 6 programs from the main program instead of running them one by one.

Here is main program code example. First you establish library references, paths, and macro variables with dates and information referenced in multiple programs.

```
/******  
Program: Main Program  
Date created: 07/16/2019  
Last modified:  
Author: Katya Roudneva  
Notes:  
Runs  
Project Formats  
Sub Program 1. Import Data  
Sub Program 2. Clean Data  
Sub Program 3. Analyze Data  
Sub Program 4. Generate Report 1  
Sub Program 5. Generate Report 2  
*****/
```

```

/*Library References*/
libname sasdate "C:\Users\eroudneva\Desktop\Project\SAS Data";

/*Paths*/
%let path=C:\Users\eroudneva\Desktop\Project\Programs; /*programs path*/
%let outpath=C:\Users\eroudneva\Desktop\Project\Output; /*report out path*/

%let reportdate=October 26th, 2018;
%let data_date=20SEP19;

```

Then you use an %INCLUDE statement to run the different programs.

```

/*Step 1. Import Raw Data*/
%INCLUDE "&path.\Sub Program 1 Import Data.sas";

/*Step 2. Run Cleaning code - create clean dataset*/
%INCLUDE "&path.\Sub Program 2 Clean Data.sas";

/*Step 3. Create Formats Used in analytical dataset and in generating
reports*/
%INCLUDE "&path.\Project Formats.sas";

/*Step 4. Run Analysis code - create analytical dataset*/
%INCLUDE "&path.\Sub Program 3 Analyze Data.sas";

/*Step 5. Run Report 1 and/or report 2 and generate output PDF/Excel
files*/
%INCLUDE "&path.\Sub Program 4 Generate Report 1.sas";
%INCLUDE "&path.\Sub Program 5 Generate Report 2.sas";

```

Using an %INCLUDE statement suppresses log output details. To see all the log output as you would if you run each program directly use *options source2*.

```
options source2;
```

These processes can only be automated if your programs are data driven and can be run without any modifications to the code. In order to accomplish that, you may need to rework the individual programs so they are not dependent on redundant updating of dates, other variables or pieces of code. The key to this automation is to identify repetitive elements that can be brought out from the individual programs into one main program. This way the variables or code will only need to be updated once or can even be updated automatically. Writing programs in this way minimizes mistakes due to human error since things only need to be updated once. It also can substantially reduce the time spent updating.

## CONCLUSION

SAS programming often involves a lot of repetition, for example using functionally identical code in different ways. Using macros and macro variables can help make programming faster and easier both by organizing code and by allowing for an easy way to update and modify programs for future use. By noting similarities across programs, you can reduce both code redundancy and manual copy/pasting or updating. While macros can make programming easier not, all situations can be improved by them. In some situations, they may increase complexity more than they decrease workload. However, if you find yourself writing pieces of code or updating programs repeatedly it may be worth taking the time to write them. The methods talked about in this paper are just some examples of the numerous ways of making SAS programming more efficient. Hopefully this information will help you incorporate macros and macro variables into your code and help you create programs that are written more quickly, maintained and improved easier, and are generally better.

## CONTACT INFORMATION

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

Ekaterina (Katya) Roudneva  
University of California, Davis  
1616 Da Vinci Court  
Davis, CA 95616  
(530) 754-0815  
eroudneva@ucdavis.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.