Paper 118-2022

Automating Reports Using Macros and Macro Variables

Ekaterina Roudneva, University of California, Davis, CA

ABSTRACT

Report generation can involve manually updating code when you get new information. This paper will go over some techniques that can help you automate your reports and remove the need to manually change code when a dataset is updated or when working with different data. Topics include using PROC SQL to create macro variables and data driven programs. This will be applied to a real-world example that automates the creation of a data dictionary and a frequency report of all variables for specified datasets. In the example, macro variables in combination with %DO loops and macro arrays are used to create dynamic code that links variables with defined formats and outputs a report of all frequencies for every single variable. Using macros and %IF %THEN branching logic can further make the code more flexible and validate the input. This paper is intended for an audience that has basic knowledge of the SAS macro language.

INTRODUCTION

While sometimes reports are only generated once, other times the same report needs to be applied to many different datasets or run multiple times as new information gets added. These changes can create the need to modify programs that generate the report each time it is run and can involve tasks such as manually modifying the date of the report each time you run it, hard coding different values or adding additional categories and variables. When it is something that only needs to be changed once or twice it is not a big deal to do manually but if it is a situation where the same thing needs to be repeated hundreds or more times with varying input, the task can become very time consuming. Data driven programs can provide flexibility, and avoid hard-coded values, and automate the need to make manual changes as well as anticipate future changes or additions in the data. This paper will introduce some ways of automating steps in reporting and making programs data driven through the use of macros and macro variables without having to rely on manual changes when something changes. It is intended to help teach useful macro writing and validation techniques as well as show tips and tricks on what to look for in your code to help you automate it.

WHAT IS AUTOMATION?

Automation can seem daunting at first but when broken down automation is just a series of steps that allow a task to be done with minimal manual input. It can mean anything from a report that used to need frequent updates that can now be run with a single press of a button, to a report that gets automatically emailed every week with the latest data. Automation can be achieved by using the following principles:

- No manual changes
- Ability to handle multiple inputs
- Validation
- Output depends on input

CREATING AND USING MACRO VARIABLES

OVERVIEW

This paper explains the process of creating a macro that generates a report of the data and a corresponding data dictionary for a specified dataset. It is based on a real-life example of the following situation:

- There are hundreds of different forms with different datasets that have hundreds to thousands of variables

- New versions of the forms are frequently added, variables may get added or removed
- Reports and data dictionaries need to be created for the data that has been collected and data entered

Creating an individual program that creates a report and data dictionary for each dataset is very time consuming and requires a lot of manual checking to figure out which variables have been added or removed. Each form also requires a completely different program since the variables are different. To simplify the process, we can create a macro that generates a report of the data and a corresponding data dictionary with variable names, formats, and frequent values in the data for a specified dataset.

Dataset SNACKS from SASHELP library shown in Display 1 will be used to demonstrate the process of using macro variables to automate a generated report.

VIEWTABLE: Sashelp.Snacks (Daily snack food sales)

	QtySold	Price	Advertised	Holiday	Date	Product
1	0	1.99	0	0	01JAN2002	Baked potato chips
2	0	1.99	0	0	02JAN2002	Baked potato chips
3	0	1.99	0	0	03JAN2002	Baked potato chips
4	0	1.99	0	0	04JAN2002	Baked potato chips
5	0	1.99	0	0	05JAN2002	Baked potato chips
6	0	1.99	0	0	06JAN2002	Baked potato chips
7	0	1.99	0	0	07JAN2002	Baked potato chips
8	0	1.99	0	0	08JAN2002	Baked potato chips
9	0	1.99	0	0	09JAN2002	Baked potato chips
10	0	1.99	0	0	10JAN2002	Baked potato chips

Display 1. View of First 10 Observations in sashelp.snacks Dataset

REDUCE MANUAL INPUT

To make a program more dynamic and data driven first identify where automation is possible and where hard coded values can be substituted by macro variables. This can be done by starting with a small part of a program that works for one specific case. Once the code works with one case, additional parameters can be added, and the code can be updated to work for additional situations. Dates, paths and filenames can be created as macro variables and then used throughout the program. Once enough steps are automated a report that before required a lot of changes to be run multiple times can be run with a click of a button and be applicable to multiple datasets.

Below is an example of macro variables defined at the top of the program:

```
%LET outpath=C:\Users\eroudneva\Desktop\Output;
%LET outdate=01aug22;
```

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 instead of writing it out in the title.

MANIPULATING MACRO VARIABLES

If you need the date in another format, instead of manually changing it, you can change an existing macro variable using 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, then output the numeric value to another format, in this case *worddate*. Output 1 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.snacks;
tables date*holiday/nocol norow nopercent;
format date year4.;
title "Report as of &outdate_word.";
run;
```

Report as of August 1, 2022

Frequency	Table of Date by Holiday		
Date(Date of sale)	Holiday(Holiday (1=yes))		
	0	1	Total
2002	8750	4025	12775
2003	8820	3955	12775
2004	7945	2275	10220
Total	25515	10255	35770

Output 1. 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 1 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.,mmddyy8.));	12/01/12

Table 1. Examples of Useful Functions for Macro Variables

SYSFUNC can be used to create a macro variable of todays date by reformatting the automatic SAS macro variable SYSDATE:

```
%let todaydate_word=%sysfunc(INPUTN(&sysdate., Date7.), worddate.);
%put &todaydate_word.;
```

CREATING MACRO VARIABLES USING PROC SQL SELECT INTO STATEMENT

Sometimes you may need to create a macro variable or multiple macro variables from a specific dataset. For example, suppose we want the minimum and maximum date in SNACKS dataset in the report title. One method involves using PROC SQL, the SELECT statement and the INTO clause to create macro variables *mindate* and *maxdate* and store them as text strings. Dates in SAS are stored as numbers - the number of days from January 1, 1960. To show the date as a mm/dd/yy format, we need to convert it to a character value first. Use %SYSFUNC and PUTN to convert from a numeric to a character format.

```
proc sql noprint;
select min(date), max(date) into :mindate,:maxdate
from sashelp.snacks;
quit;

%let mindate_mmddyy=%sysfunc(Putn(&mindate.,mmddyy8.));
%let maxdate_mmddyy=%sysfunc(Putn(&maxdate.,mmddyy8.));
%put &mindate_mmddyy. &maxdate_mmddyy.;
```

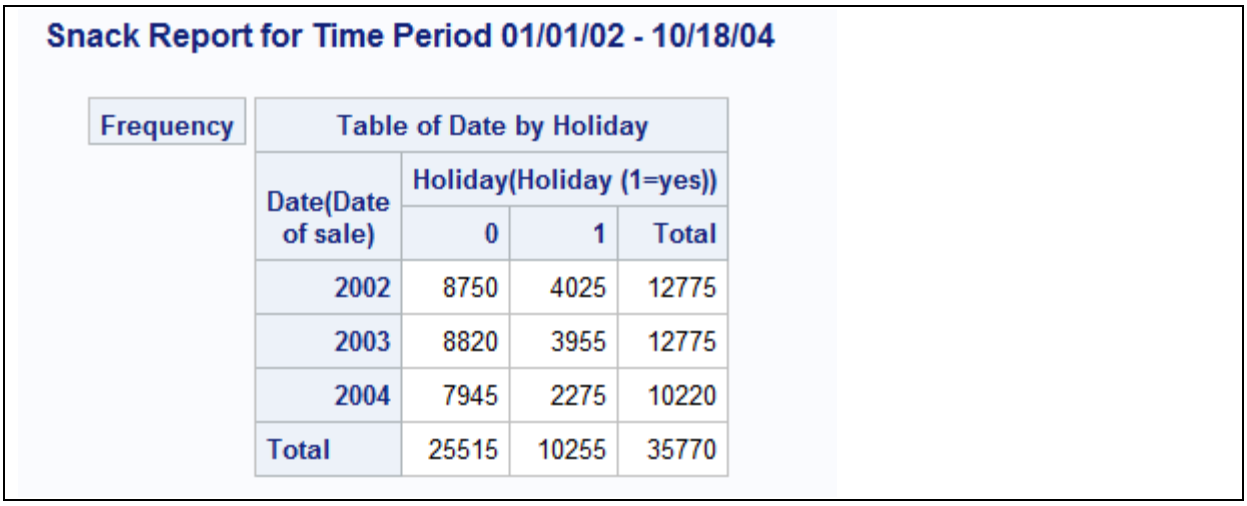
Output 2 shows the log output after changing the dates to a formatted value.

```
2525  %let mindate_mmddyy=%sysfunc(Putn(&mindate.,mmddyy8.));
2526  %let maxdate_mmddyy=%sysfunc(Putn(&maxdate.,mmddyy8.));
2527  %put &mindate_mmddyy. &maxdate_mmddyy.;
01/01/02 10/18/04
```

Output 2. Log Output Showing Value of mindate_mmddyy and maxdate_mmddyy Macro Variable

Now we can use created macro variables *&mindate_word* and *&maxdate_word* in the report title to have the date range of the data in the title as shown in Output 3.

```
proc freq data= sashelp.snacks;
tables date*holiday/nocol norow nopercent;
format date year4.;
title "Snack Report for Time Period &mindate_mmddyy. - &maxdate_mmddyy.";
run;
title;
```



Snack Report for Time Period 01/01/02 - 10/18/04

Frequency	Table of Date by Holiday		
Date(Date of sale)	Holiday(Holiday (1=yes))		
	0	1	Total
2002	8750	4025	12775
2003	8820	3955	12775
2004	7945	2275	10220
Total	25515	10255	35770

Output 3. PROC FREQ Output That Includes Date Range in Title

CREATE INPUT MACRO VARIABLES

To start a program, define the input macro variables that will be changing in the report. In this example we want the input variables to be the dataset name, the report title, and the report file name. These will eventually become macro parameters.

```
%let dataset=sashelp.snacks;
%let ReportTitle=Snacks;
%let ReportFileName=snacks_report;
```

Since the report should be driven by variables in each dataset the first step will be to run a PROC CONTENTS and create a dataset with a list of variables in the dataset.

```
/*Get dataset contents*/
proc contents varnum data=&dataset.
out=contents (keep=name type length varnum label format
rename=(name=variablename)) noprint;
run;
```

In some cases you may want to use an external source of formats for the report. Having all the format links in one place can make it easier to make updates since they will only need to be changed in one place if there is an update. For this example, a format list dataset that includes variables and their associated formats is created so that these formats can be used in the report. Another way to store these would be in a separate SAS dataset or in an Excel file.

```
data format_list;
input VariableName :$35. FormatName :$30.;
datalines;
Advertised yesno
Holiday yesno;
run;


proc format;
value yesno
1="Yes"
0="No";
run;
```

Format list dataset can be added as another macro variable:

```
%let formatdt=format_list;
```

In the program the contents dataset can be merged with the format_list dataset and the format variable can be replaced with the new defined format.

```
proc SQL;
Create table contents as
select a.*, b.formatname
from contents a left join &formatdt. as b
on upcase(a.variablename)=upcase(b.variablename);
quit;

data contents; set contents;
if formatname^="" then format=formatname;
run;
```

CREATING MACRO VARIABLE LISTS USING PROC SQL

A PROC SQL SELECT INTO statement can be used to create macro variables that contain lists of values. There are two ways to create macro variable lists using PROC SQL.

Method 1 – Create a list in one macro variable

The first way is to put the list into a single macro variable. The separator is defined in the code.

- The disadvantage of this method is that macro variables have a character limit of 65,534 characters. If there is a large amount of input data the list may run of out of space and the data will be cut off.

```
proc sql noprint;
  select variablename INTO :varnamelist separated by ' '
  from contents;
quit;
%put &varnamelist.;
```

Output 4 shows the value of the macro variable *varnamelist* that includes all the variable names in a dataset separated by a space.

```
2623      %put &varnamelist.;
Advertised Date Holiday Price Product QtySold
```

Output 4. Log Output Showing Value of varnamelist Macro Variable

This macro variable can be used with a scan do loop in the following way:

```
%macro scanloop;
    %do i=1 %to %sysfunc(countw(&varnamelist.,%str( )));
       %let ThisVar=%scan(&varnamelist.,&i,%str( ));
       %put &ThisVar.;

       proc freq data=&dataset.;
       tables &ThisVar./ maxlevels=10;;
       run;

    %end;
%mend;
%scanloop;
```

Method 2 – Create a list in multiple macro variables

The other way to create macro variables using PROC SQL is to create lists in multiple macro variables. First create a count of variables in the dataset and put it in a macro variable.

```
*Count number of variables;
proc sql noprint;
select count(variablename) into : numvars trimmed
from contents;
quit;
%put number of vars: &numvars.;
```

The log in Output 5 shows that the number of variables in dataset SNACK is 6

```
2562        %put number of vars: &numvars.;
number of vars: 6
```

Output 5. Log Output Showing Value of numvars Macro Variable

The lists can now be created in macro variables *varname1 to varnameX.* We can either specify an arbitrarily high number (say 1000) or use the created count *numvars* macro variable so that SAS will create macro variables varname1 through varname6.

```
Proc sql noprint;
select variablename
    into :varname1-:varname&numvars.
    from contents
;
quit;
%put &varname1.;
%put &varname2.;
%put &varname3.;
```

In our example, we do not just want a macro variable of the variable name, but macro variables of all the associated information including variable number in the dataset, variable name, variable type, variable format and variable label so that this information can be used in later code. To order the variables, add an ORDER BY statement.

```
Proc sql noprint;
    select varnum,variablename,type,format,label
    into :VarNum1-:VarNum&numvars.,
         :VarName1-:VarName&numvars.,
         :VarType1-:VarType&numvars.,
         :VarFmt1-:VarFmt&numvars.,
         :VarLabel1-:VarLabel&numvars.
    from contents
    order by varnum
;
quit;
%put &VarNum1. &varname1. &vartype1. &varfmt1. &varlabel1.;
%put &VarNum2. &varname2. &vartype2. &varfmt2. &varlabel2.;
%put &VarNum3. &varname3. &vartype3. &varfmt3. &varlabel3.;
```

To check the macro variable values you can either use a %PUT statement to check the first few, or you can use %PUT _user_ to view all user generated macro variables. Output 6 shows the log output of the %PUT statements:

```
383       %put &VarNum1. &varname1. &vartype1. &varfmt1. &varlabel1.;
1 QtySold 1  Quantity sold
384       %put &VarNum2. &varname2. &vartype2. &varfmt2. &varlabel2.;
2 Price 1  Retail price of product
385       %put &VarNum3. &varname3. &vartype3. &varfmt3. &varlabel3.;
3 Advertised 1 yesno Advertised (1=yes)
```

Output 6. Log Output Showing Values of created Macro Variables

Once the macro variables are created they can be used in a scan do loop.

```
%Macro ScanLoop;
%do i=1 %to &numvars.;
       %let ThisVar=&&varname&i.;
       %put &ThisVar.;

       proc freq data=&dataset.;
       tables &ThisVar./ maxlevels=10;;
       run;

%end;
%Mend;
%ScanLoop;
```

The following Table 2 shows how *&&VarName&i.* macro variables resolve in the first few do loop steps.

Value of "i"	&&VarName&i. Resolves to	Macro Variable Value
1	&VarName1.	QtySold
2	&VarName 2.	Price
3	&VarName 3.	Advertised

Table 2. Value of *&&VarName&i.* in first few do loop iterations

CREATING AN OUTPUT PDF FILE USING %IF %THEN LOGIC

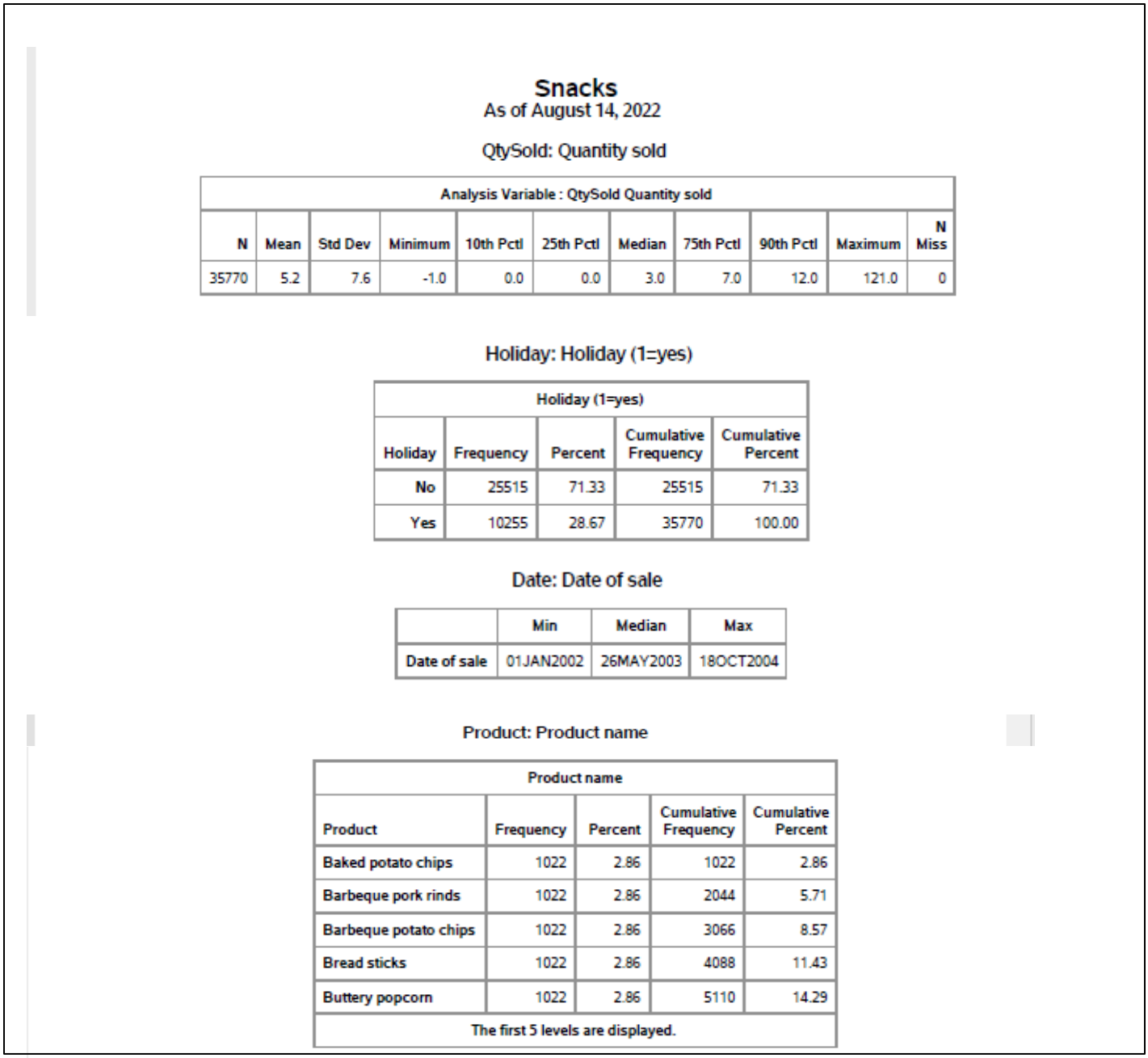
Running a PROC FREQ on all variables is not always very useful since not all variables are categorical. Since there are multiple macro variables, the type of variable and the format can be used with a series of %IF %THEN statements to generate different types of output depending on the type of variable. We can also add additional branching logic that uses the number of distinct values in a variable to further determine what type of output gets generated. Output 7 shows the code used to create a PDF report file of all variables in a specified dataset:

```
options nodate;
ODS NOPROCTITLE;
ods _all_ close;
ods pdf file="&outpath.\&ReportFileName..pdf" startpage=no;
ods escapechar= "^";

ods pdf text= "^S={just=center font_weight=bold font_size=14pt}^n&ReportTitle.";
ods pdf text= "^S={just=center font_weight=bold font_size=10pt}As of
&todaydate_word.";

%macro loop;
%do i=1 %to &numvars.;
/*Get number of unique values for each variable - put into macro variable*/
       proc sql noprint;
       select count(distinct &&VarName&i.) into :UnqVal trimmed
```

```
	from &dataset.;
	quit;
	%put number of unique values for variable &&VarName&i.: &UnqVal.;

ods pdf text= "^S={just=center font_weight=bold font_size=10pt}^n&&VarName&i.:
&&VarLabel&i..";
/*If date format, show min median max only*/
	%if &&VarFmt&i..=DATE %then %do;
		proc tabulate data=&dataset.;
		  var &&VarName&i..;
		  tables &&VarName&i..,(min median max)*f=date9.;
		run;
	%end;

/*If more than 5 unique values and its a numeric type variable - Output means
statistics */
	%else %if &UnqVal.>5 and &&VarType&i..=1 %then %do;
		proc means data=&dataset. N mean std min P10 P25 median P75 P90
		max nmiss maxdec=1;
		var &&VarName&i..;
		run;
	%end;

/*If less than 5 unique values or a character type variable - Output frequencies*/
	%else %do;
		proc freq data=&dataset. order=freq;
		/*If less than 5 unique values - show all*/
		%if &UnqVal.<=5 %then %do;
			tables &&VarName&i.;
		%end;
		/*Otherwise show top 5 values*/
		%else %do;
			tables &&VarName&i. / maxlevels=5;
		%end;
		/*Apply format if there is a format for the variable*/
		%if &&varfmt&i..^=  %then %do;
			format &&VarName&i. &&VarFmt&i...;
		%end;
		run;
	%end;
%end;
%mend;
%loop;

ods pdf close;
ods html;
```

Output 7. Code used to create a PDF Report File of All Variables in a Specified Dataset

Output 8 shows an example report for the above code with four different types of variables – numeric variables with more than 5 unique values, categorical variables with less than 5 unique values, categorical variables with more than 5 unique values and variables with format date.



Snacks
As of August 14, 2022

QtySold: Quantity sold

Analysis Variable : QtySold Quantity sold										
N	Mean	Std Dev	Minimum	10th Pctl	25th Pctl	Median	75th Pctl	90th Pctl	Maximum	N Miss
35770	5.2	7.6	-1.0	0.0	0.0	3.0	7.0	12.0	121.0	0

Holiday: Holiday (1=yes)

Holiday (1=yes)				
Holiday	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	25515	71.33	25515	71.33
Yes	10255	28.67	35770	100.00

Date: Date of sale

	Min	Median	Max
Date of sale	01JAN2002	26MAY2003	18OCT2004

Product: Product name

Product name				
Product	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Baked potato chips	1022	2.86	1022	2.86
Barbeque pork rinds	1022	2.86	2044	5.71
Barbeque potato chips	1022	2.86	3066	8.57
Bread sticks	1022	2.86	4088	11.43
Buttery popcorn	1022	2.86	5110	14.29
The first 5 levels are displayed.				

Output 8. An Example PDF Output Showing a Report With Each Type of Variable

MACROS

CREATING MACROS

Defined macro variables that are used in a program can be turned into a macro. In the example used throughout this paper the input macro variables are as follows:

```
%let dataset=sashelp.snacks;
%let ReportTitle=Snacks;
%let ReportFileName=snacks_report;
%let formatdt=format_list;
```

These macro variables can used in a macro as macro parameters:

```
%macro CreateReport(dataset,outpath,reporttitle,reportfilename,formatdt);
      Program code
%mend;

%CreateReport(dataset=sashelp.snacks,
outpath=&outpath.,
reporttitle=Snacks,
reportfilename=Snacks_report,
formatdt=format_list);
```

Additional macro variables can be incorporated into the title, in this case to include the time range:

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

%let mindate_mmddyy=%sysfunc(Putn(&mindate.,mmddyy8.));
%let maxdate_mmddyy=%sysfunc(Putn(&maxdate.,mmddyy8.));

%CreateReport(dataset=sashelp.snacks,outpath=&outpath.,
reporttitle=%str(Snacks Report for Time Period &mindate_mmddyy. -
&maxdate_mmddyy.),
reportfilename=snacks_report
,formatdt=format_list);
```

The resulting report as shown in Output 9

Snacks Report for Time Period 01/01/02 - 10/18/04
As of August 15, 2022

QtySold: Quantity sold

Analysis Variable : QtySold Quantity sold										
N	Mean	Std Dev	Minimum	10th Pctl	25th Pctl	Median	75th Pctl	90th Pctl	Maximum	N Miss
35770	5.2	7.6	-1.0	0.0	0.0	3.0	7.0	12.0	121.0	0

Output 9. PDF Output of a Report that Uses mindate_mmddyy and maxdate_mmddyy Macro Variables in the Title

ADD MACRO PARAMETERS AND BRANCHING LOGIC

To output a corresponding data dictionary for the variables in the dataset additional macro parameters can be added and default values can be set with an equal's sign after each macro argument. For the example, excelfilename= has been added to the macro parameters as having a default value of nothing. %IF %THEN statements can be included in the macro that only output an excel data dictionary if an excel file name is defined, and only merge the formats file with proc contents dataset if formatdt macro variable has a value.

```
%macro CreateReport(dataset=, outpath=, reporttitle=, reportfilename=,
excelfilename=, formatdt=);

    Code that creates proc contents dataset
    %if &FormatDt. ne %then %do;
```

```
      Merge format dataset with proc contents dataset
    %end;
    Program code
    Output PDF Report

    %if &ExcelFileName. ne %then %do;
      Output excel data dictionary
    %end;

%mend;

%CreateReport(dataset=sashelp.snacks,
outpath=&outpath.,
reporttitle=Snacks Report,
reportfilename=snacks_report,
excelfilename=snacks_datadict,
formatdt=format_list);
```

The macro can be run with fewer arguments to only output a PDF report and not use an existing formats dataset:

```
%CreateReport(dataset=sashelp.snacks,outpath=&outpath.,
reporttitle=%str(Snacks Report,
reportfilename=snacks_report);
```

Create a dataset with frequencies for all variables

For the example, I can output a minimal data dictionary using the proc contents dataset. Another option is to use the created macro variable lists to run frequencies on all the variables. This data can be combined to create a master dataset of all the possible values and their frequencies. Output 10 shows code used to create a dataset that has all the possible frequencies for a specified dataset.

```
/*Create an empty dataset*/
data out_vars_all;
length var $32.;
var="";
run;


%Macro GetVarValues;
%do i=1 %to &numvars.;

%put  Getting frequencies for variable &&VarName&i.;

*Get the values;
proc freq data=&dataset. noprint;
tables &&VarName&i.. /out=out_freq (rename=(&&VarName&i..=value);
run;

*For each variable;
data out_freq_var; set out_freq (rename=(value=value_tmp));
       *Create character type value;
       length value valuefmt varLabel $200 varfmt $40.;
       value=cats(value_tmp);
       /*create a formatted value var if there is a format*/
      %if  &&varfmt&i..^= %then %do; valuefmt=put(value_tmp,&&varfmt&i...);
      %end;
       /*otherwise use unformatted value*/
       %else %do; valuefmt=cats(value_tmp); %end;
       /*Create a numeric value*/
       %if  &&VarType&i..=1 %then %do; value_num=value_tmp; %end;
```

```
       drop value_tmp;

       length var $32.;
       var="&&VarName&i.."; vartype="&&VarType&i..";
       varfmt="&&VarFmt&i.."; varLabel="&&VarLabel&i..";
       varOrder=&&VarNum&i..; valueN+1;
run;

proc SQL; Create table out_freq_var2 as
select a.*,
     sum(count) as ValueTotalN,
     count(distinct value) as ValueUnqN,
     sum(case when missing(value) then count else . end) as ValueMissN,
     sum(case when not missing(value) then count else . end) as ValueNonMissN
     /*if a numeric variable - create mean statistics*/
      %if &&VarType&i..=1 %then %do;
          , mean(value_num) as ValueMean,
          min(value_num) as ValueMin,
          max(value_num) as ValueMax,
         std(value_num) as ValueStd
      %end;
from out_freq_var a;
Quit;
/*Merge the datasets together*/
data out_vars_all; set out_vars_all out_freq_var2;
if var="" then delete;
run;

%end;
%Mend;
%GetVarValues;
```

Output 10. Code That Creates a Dataset That Has All the Possible Frequencies for a Specified Dataset.

The resulting dataset can be modified as shown in Output 11 to only keep values if there are less than 5 unique values and then merge all the information for each variable on one row.

```
*only keep if less than 5 unique values;
proc sort data=out_vars_all; by var; run;
data out_vars_all_fin; set out_vars_all; by var;
if ValueUnqN>5 then do;
       if not first.var then delete;
       count=.; percent=.; value=""; valuefmt=""; value_num=.;
end;
percent_num=percent/100;
valuemean=round(valuemean,.1);
valuestd=round(valuestd,.1);

if ValueMissN=. then ValueMissN=0;
if ValueNonMissN=. then ValueNonMissN=0;
ValueMissN_pct=ValueMissN/ValueTotalN;
ValueNonMissN_pct=ValueNonMissN/ValueTotalN;

format percent_num ValueMissN_pct ValueNonMissN_pct percent.;
run;


*get all info on one row;
proc sort data=out_vars_all_fin; by var valueN; run;
data report_byvar; set out_vars_all_fin; by var valueN;
```

```
length DataFormatNs DataFormatNs_row DataFormatPctsNonMiss DataFormatPctsNonMiss_row
$400. DataFormatValuesfmt_row DataFormatValuesfmt DataValsWithFreqs
DataValsWithFreqs_row $1000.;

if first.var then do;
DataFormatValuesFmt=""; DataFormatNs=""; DataFormatPctsNonMiss="";
DataValsWithFreqs="";
end;

DataFormatValuesfmt_row=strip(valuefmt);
DataFormatNs_row=strip(put(count,8.));
DataFormatPctsNonMiss_row=strip(put(round(percent,0.1),8.));
if count^=. then do;
if percent=. then DataValsWithFreqs_row=catx("
",DataFormatValuesfmt_row,cats("(N=",DataFormatNs_row,")"));
else DataValsWithFreqs_row=catx("
",DataFormatValuesfmt_row,cats("(N=",DataFormatNs_row,",NonMissPct=",DataFormatPctsNon
Miss_row,"%)"));
end;
retain DataValsWithFreqs;
if value^='' then do;
      DataValsWithFreqs=catx("|",DataValsWithFreqs,DataValsWithFreqs_row);
end;

      if last.var;
run;
```

Output 11. Code to Only Keep Frequency Values If There Are Less Than 5 Unique Values and Then Merge All the Information for Each Variable On One Row.

Data can be further merged with format information, then formatted as shown in Output 12 to create a final dataset that contains frequency information for each variable in a specified dataset.

```
*Output formats;
proc format cntlout=out_formats; run;
data out_Formats_byfmt(keep = fmtname values);
      set out_Formats;
      by fmtname;
      length values $4000. valfmt $2000 labeln $1000.;
      if first.fmtname then do; values = ""; end;

      labeln = strip(label);
      retain values "" code "" valfmt "";
      values = catx(" | ",values, cats(start, ' = ', labeln));
      if last.fmtname then output;
run;


*merge with report;
proc SQL;
Create table report_byvar2 as
select a.*, b.values as varfmt_values
   from report_byvar a left join out_Formats_byfmt as b
       on upcase(a.varfmt)=upcase(b.fmtname);
quit;

data report_byvar_fin; set report_byvar2;
length vartypeSp $10;
if vartype=2 then vartypeSp="Character";
```

```
else if vartype=1 then vartypeSp="Numeric";
run;
```

Output 12. Code To Merge the Data with Format Information and Create A Final Dataset That Contains Frequency Information For Each Variable in a Specified Dataset.

Output an excel data dictionary

The resulting dataset can be output into an excel file:

```
%if ExcelFileName^=NONE %then %do;
OPTIONS MISSING=" ";
option nodate number;
ods _all_ close;
ods excel file="&outpath.\&excelfilename..xlsx" options(
sheet_name="Sheet1" embedded_titles='no' flow="tables") ;

    proc report data=report_byvar_fin;
    run;

ods excel close;
ods html;
%end;
```

Output 13 shows the excel output with the data dictionary for the dataset specified in the macro

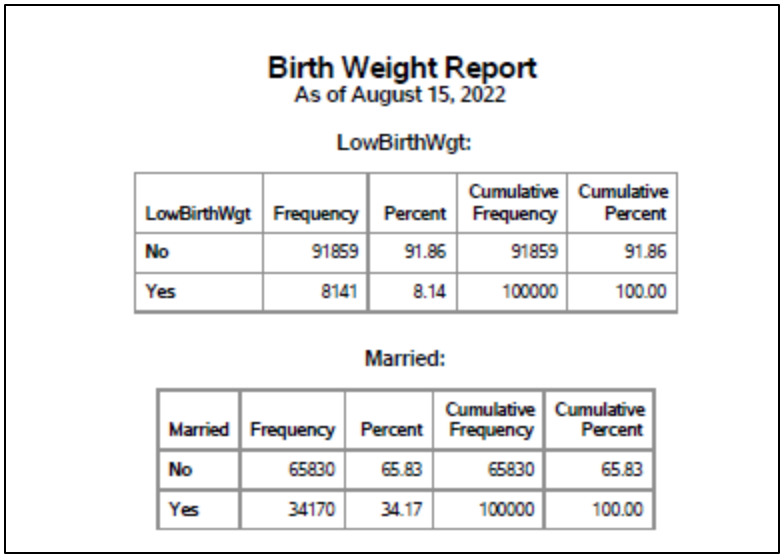
	B	C	D	E	F	G	H	I	J	K	
	var	vartypeSp	varlabel	aluetotal	ValueUnqN	varfmt	varfmt_values	DataValsWithFreqs	alueMiss	ueMissN	ue
1	QtySold	Numeric	Quantity sold	35770	133				0	0%	
2	Price	Numeric	Retail price of prc	35770	6				0	0%	
3	Advertised	Numeric	Advertised (1=ye	35770	2	yesno	0=No | 1=Yes	No (N=34793,NonMissPct=97%)|Yes	0	0%	
4	Holiday	Numeric	Holiday (1=yes)	35770	2	yesno	0=No | 1=Yes	No (N=25515,NonMissPct=71%)|Yes	0	0%	
5	Date	Numeric	Date of sale	35770	1022	DATE			0	0%	
6	Product	Character	Product name	35770	35				0	0%	

Output 13. An Excel File Data Dictionary Output for SNACKS Dataset in SASHELP library.

Testing macro with other inputs

Once the macro works with one set of inputs, it can be tested with other inputs. In the example below, the macro is used with another dataset BIRTHWGT in SASHELP library to output a PDF report shown in Output 14

```
%CreateReport(dataset=sashelp.birthwgt,outpath=&outpath.,reporttitle=Birth
Weight Report,reportfilename=birthwgt_report,
excelfilename=birthwgt_datadict);
```



Birth Weight Report
As of August 15, 2022

LowBirthWgt:

LowBirthWgt	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	91859	91.86	91859	91.86
Yes	8141	8.14	100000	100.00

Married:

Married	Frequency	Percent	Cumulative Frequency	Cumulative Percent
No	65830	65.83	65830	65.83
Yes	34170	34.17	100000	100.00

Output 14. A PDF Frequency Output for BIRTHWGT Dataset in SASHELP library

Output 15 shows the data dictionary output for BIRTHWGT in SASHELP library.

	B	C	D	E	F	G	H	I	J	K
	var	vartypeSp	varlabel	valuetotalN	ValueUnqI	varfmt	varfmt_vali	DataValsWithFreqs	ValueMissN	ValueMissN_
	LowBirthWgt	Character		100000	2			No (N=91859,NonMissPct=92%)|Yes (N=8141,NonMissPct=8%)	0	0%
	Married	Character		100000	2			No (N=65830,NonMissPct=66%)|Yes (N=34170,NonMissPct=34%)	0	0%
	AgeGroup	Numeric		100000	3			1 (N=10245,NonMissPct=10%)|2 (N=75633,NonMissPct=76%)|3 (N=14122,[	0	0%
	Race	Character		100000	5			Asian (N=5224,NonMissPct=5%)|Black (N=14133,NonMissPct=14%)|Hispar	0	0%
	Drinking	Character		100000	2			No (N=80914,NonMissPct=86%)|Yes (N=13474,NonMissPct=14%)	5612	6%
	Death	Character		100000	2			No (N=99411,NonMissPct=99%)|Yes (N=589,NonMissPct=1%)	0	0%
	Smoking	Character		100000	2			No (N=73043,NonMissPct=77%)|Yes (N=21345,NonMissPct=23%)	5612	6%
	SomeCollege	Character		100000	2			No (N=48339,NonMissPct=52%)|Yes (N=44953,NonMissPct=48%)	6708	7%

Output 15. An Excel File Data Dictionary Output for BIRTHWGT Dataset in SASHELP library.

It is also possible to modify the output into whatever format is needed and add additional parameters and logic. Below Output 16 shows an example of another type of output that was created from the data using macro variables:

VarN	Variable	Label	Type	Code	Value	N	With Miss %	Non Miss %
1	QtySold	Quantity sold	Numeric		Mean=5.18 StdDev=7.56 Min=-1 Max=121			
					Total	35770	100%	100%
2	Price	Retail price of product	Numeric		Mean=2.1 StdDev=0.78 Min=0.99 Max=3.49			
					Total	35770	100%	100%
3	Advertised	Advertised (1=yes)	Categorical		0=No	34793	97%	97%
					1=Yes	977	3%	3%
					Total	35770	100%	100%

Output 16. A Word Frequency File Output for SNACKS Dataset in SASHELP library

VALIDATION

A good programming practice is to add validation checks to macro programs to check the input. As an example, you can add %IF %THEN statement in the beginning of the program to check if the input dataset exists. %RETURN will stop the macro from running any further if the statement is met. An addition of a %PUT statement will add a message to the log as shown in Output 17.

```
/*Check that dataset exists*/
%if %sysfunc(exist(&dataset.))=0 %then %do;
    %put ERROR: &dataset. dataset does not exist.;
    %return;
%end;
```

```
773  %CreateReport(dataset=shoesxx,outpath=&outpath.,reporttitle=Shoes Report,filename=shoes_report);
ERROR: shoesxx dataset does not exist.
```

Output 17 Log Output Showing an Error

Adding ERROR:, WARNING: or NOTE: to the beginning of the %PUT statement will highlight the text in the log. By default, ERROR messages are shown in red, WARNING messages are green and NOTE messages are blue. Text that doesn't start with those is shown in black. Output 18 shows the formatting of different text messages.

```
%put ERROR: Text;
%put WARNING: Text;
%put NOTE: Text;
%put Text;
```

```
ERROR: Text
WARNING: Text
NOTE: Text
Text
```

Output 18 Log Output Showing Formatting of Different Types of Text Messages

DEBUGGING

When working with macros it is not uncommon for there to be issues and for the code to not work as expected. You can use the SAS system options such as MLOGIC, MPRINT, and SYMBOLGEN to print out additional information about macro variables in the log. These options can be enabled with the following code:

```
OPTIONS symbolgen mprint mlogic;
```

To disable the options put a "no" in front of each option:

```
OPTIONS nosymbolgen nomprint nomlogic;
```

Table 3 has the definition from the SAS documentation and an example of each option

Enable Option	Disable Option	Definition	Example Log Output
SYMBOLGEN	NOSYMBOLGEN	Tells you what each macro variable resolves to by writing messages to the SAS log.	SYMBOLGEN: && resolves to &. SYMBOLGEN: Macro variable I resolves to 1 SYMBOLGEN: Macro variable VARTYPE1 resolves to 2

MLOGIC	NOMLOGIC	Traces the flow of execution of your macro, including the resolution of parameters, the scope of variables (global or local), the conditions of macro expressions being evaluated, the number of loop iterations, and the beginning and end of each macro execution	MLOGIC(GETVARVALUES): %IF condition &&varfmt&i..^= is FALSE MLOGIC(GETVARVALUES): %IF condition &&VarType&i..=1 is FALSE
MPRINT	NOMPRINT	Displays the text generated by macro execution. Each SAS statement begins a new line. Each line of MPRINT output is identified with the prefix MPRINT(macro-name):, to identify the macro that generates the statement.	MPRINT(GETVARVALUES): var="Region"; MPRINT(GETVARVALUES): vartype="2"; MPRINT(GETVARVALUES): varfmt=""; MPRINT(GETVARVALUES): varLabel=""; MPRINT(GETVARVALUES): varOrder=1; MPRINT(GETVARVALUES): valueN+1; MPRINT(GETVARVALUES): run;

Table 3. Some Examples of SAS system options

Other tips to debug programs are as following

- Use %PUT statements to keep track of which steps have been run
- Only run a small part of the macro at a time
- Change macro functions to global macro variables and run macro step by step

CONCLUSION

By automating reports as much as possible and making the code be data driven and not rely on human input, it is possible to reduce the amount of time it takes to do tasks, have fewer errors, and be more consistent. Macros and macro variables can allow for flexibility when running reports for different datasets or when the data is changing. It is not always possible to fully automate a report, but even partial automation can help make code more efficient and remove the need to manually change code. 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 reports more automated. Hopefully this information will help you incorporate macros and macro variables into your code and help you create programs and reports 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