SESUG Paper 135-2022

Names From Template (NFT)

Dr. Kannan Deivasigamani, Centene – TSS COA

Abstract

SAS programmers at times have these datasets with plenty of variables that are similar or cloned from another dataset but contain additional variables due to some intermediate computations. At the end of the day, we are tasked with retaining the original variables and drop any additional ones. We end up copying a list of variables manually or typing them in the keep statement. What if there is a way to automatically have the list available and the copy to be made using the template dataset that has the variables we need. Well, you've got it. This macro does that exactly that. It keeps the variables from the template dataset and makes a new copy or overwrites the existing one depending on your requirement. The macro accepts 3 parameters as input from the user. The first parameter is the input dataset that has all the variables that needs to be cleaned up. The second parameter is the variable-template which is a SAS dataset that has the variables of interest. The final parameter is the name of the output dataset that is the cleaned-up version. This could either be the original name or a new name depending on the user's need. No more typing the variable names from another dataset or copying manually from elsewhere. This macro is a solution to the challenge. This increases productivity and makes a programmer more powerful.

INTRODUCTION

The objective of the author for this paper was to develop a tool that can be called easily and that can be used as a template to allow other datasets to filter through by column names. Typically, when someone needs to keep a specific set of variables, it is common to see that developers either copy a list of variables from a dataset and include it either in their select statements or use a data-step to have the variables listed as part of a "KEEP" statement. The variable names are manually copied and pasted into the appropriate code steps. This repetitive task is the germinal spark for the idea behind this paper. The author devised a tool to accomplish the same without having to code for it. The goal while designing the tool was to provide the required input dataset that contains the variable names that is to be used as the template. The other parameters fed into the tool are, the input dataset with multiple variables and lastly the output dataset name that contains the filtered output. The entire structure needs to be easy to use and manage for the long haul. With this in mind, a macro with 3 parameters has been coded and tested for implementation.

MACRO DETAILS

Main macro to apply template

Name: ***nft***

Parameters: indsn, template, outdsn (all positional parameters)

Indsn = input dataset

Template = dataset that contains variables to be used as a template

Outdsn = final output dataset with application of the template/filter

Invocation: % nft (indsn, template, outdsn)

Bonus macro for custom print

Name: ***p***

Parameters: D, T, N= (D & T as positional parameters and N= is a keyword parameter)

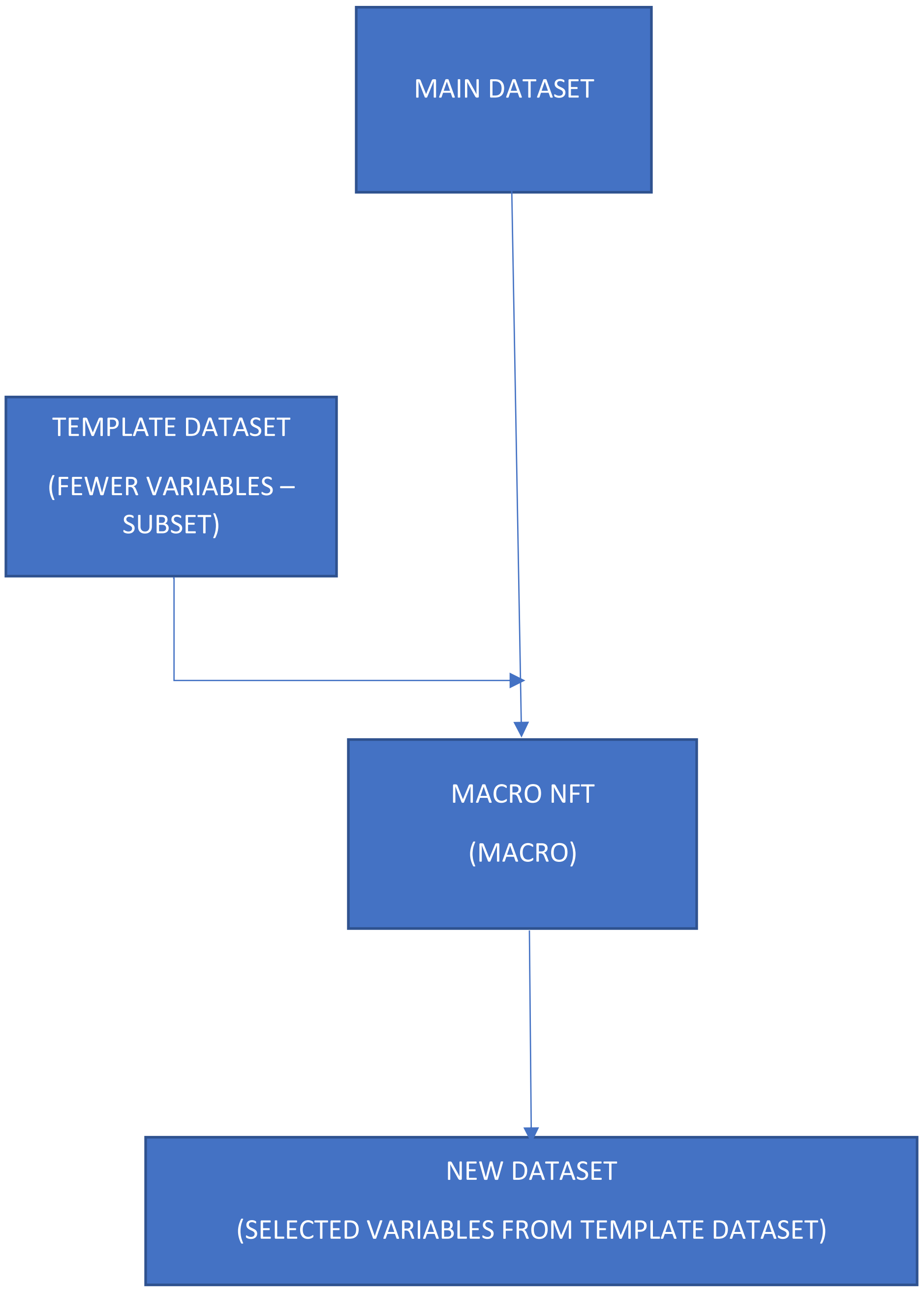
D = Dataset to be printed

T = Title to show in the Output

N= Number of observations to print

Invocation: p (d, T, N=)

PICTORIAL REPRESENTATION OF THE PROCESS


MAIN DATASET
TEMPLATE DATASET
(FEWER VARIABLES – SUBSET)
MACRO NFT
(MACRO)
NEW DATASET
(SELECTED VARIABLES FROM TEMPLATE DATASET)

MACRO OPERATION

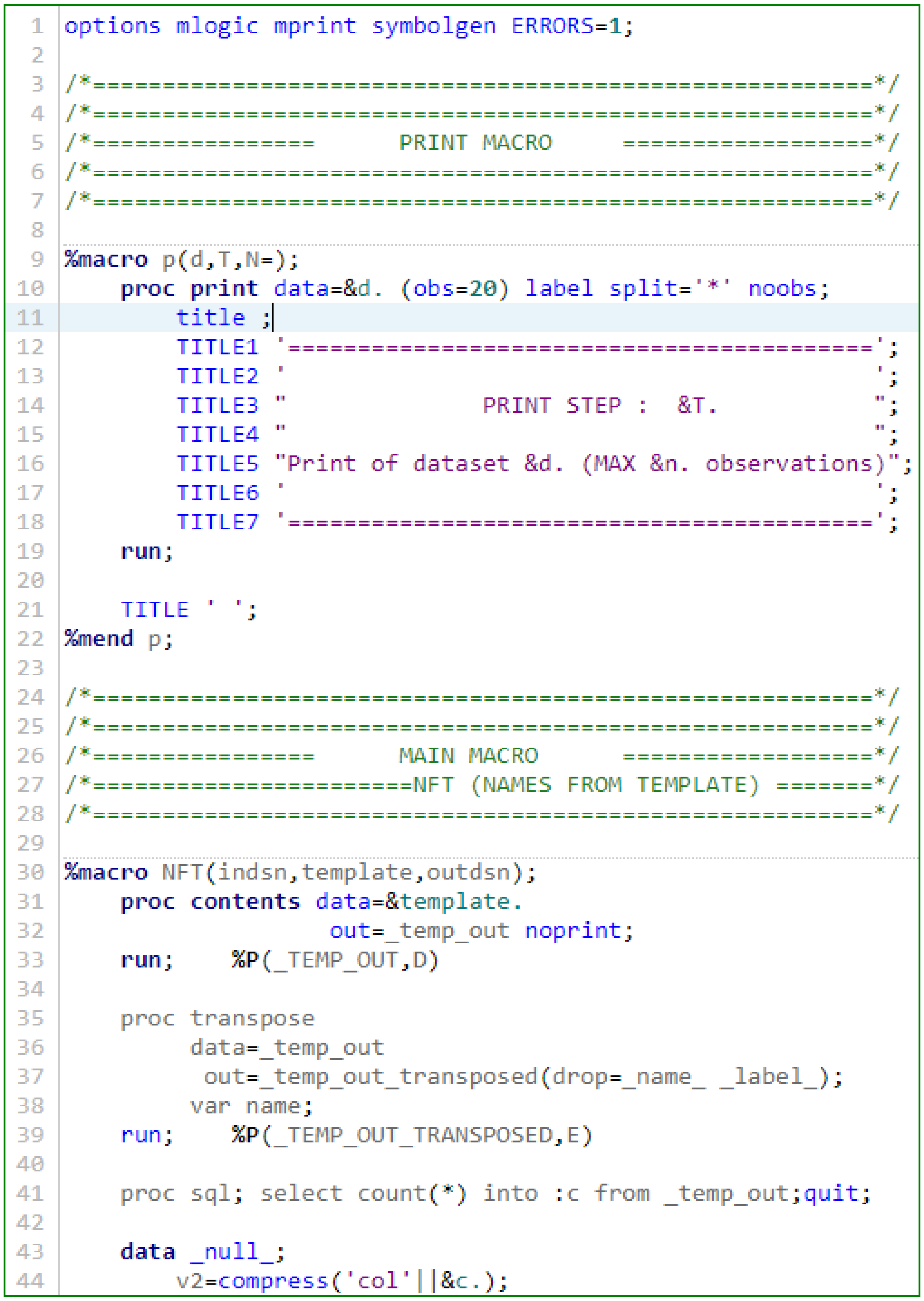
The macro code shows 3 parameters that are positional and are required for operations. The first parameter is the master dataset from which the original data is brought in. The value expected is a name of a SAS dataset that can be either temporary or permanent, given there are proper access rights. The 2nd parameter is the name of the template dataset which also needs to be a SAS dataset; either temporary or permanent. This dataset need not contain any data. The final parameter passed while calling this macro is the name of the output dataset that should have the results of the macro with the template applied.

The first step in the macro reads the input "template" dataset and gathers all the variable names and transposes all the names to have them in one row and multiple columns as applicable. The number of variables is counted into a macro variable "c" via the SQL statement. This counter variable will be used to define the array in the step below. The variables are looped through to create a macro variable "var_list" to hold all var names from the template dataset in one row delimited by spaces. The final step is to assign this new macro variable to the "keep" statement to limit the output dataset to the variable/column names from the template dataset.

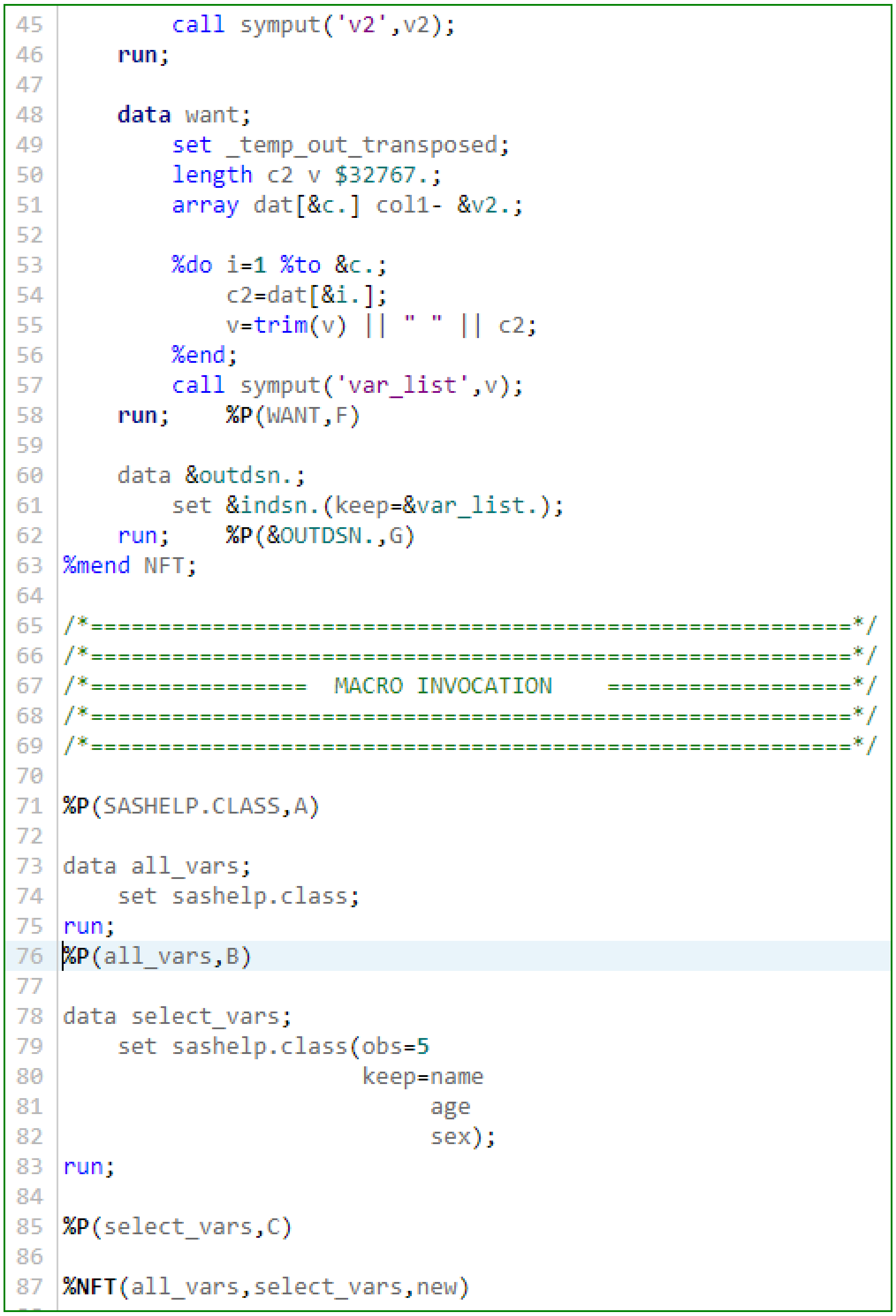
While the macro offers flexibility in terms of invocation, care should be taken to ensure that the template is a subset of the larger dataset and there are no new variable names introduced in the template as the code will not work with any mismatches. Care also needs to be taken to ensure that the input and template datasets are SAS datasets and are not flat files or delimited files in which case, they need to be converted to SAS datasets first before passing them as parameters while invoking this macro.

The code has 3 logical parts. The first is the custom print macro that takes in 3 parameters that is intuitive designed to simply save time and space by reusing the code to custom print different datasets with different titles and limit the observations if necessary as it is evident from the macro definition and invocation code "p (d, T, N=)". The second is the main macro that is the heart of the operation. This segment reads the template and applies it to the input dataset to generate the output. The last part is where the test is demonstrated. The mock datasets are prepared and the macro is invoked accompanied by the corresponding prints. Only the input and final output datasets are shown and the rest are masked. However, if the code is run as-is, the additional prints will show in the output.

SOURCE CODE

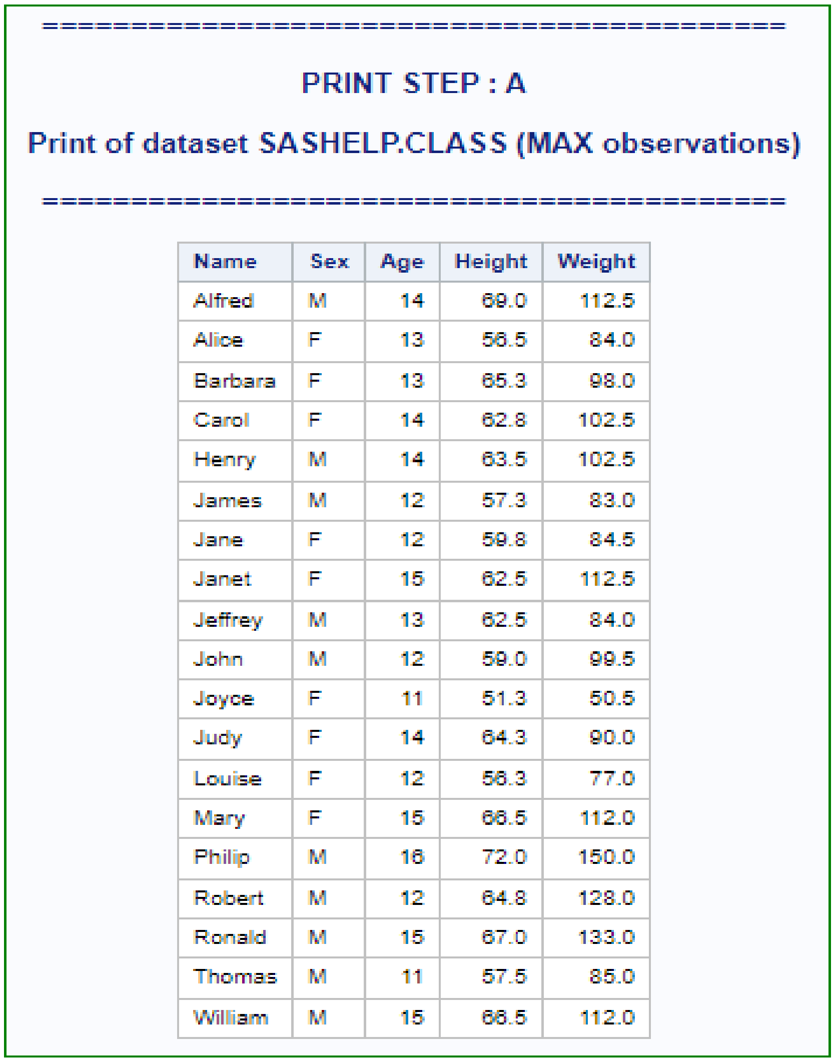


```
options mlogic mprint symbolgen ERRORS=1;

/*========================================================*/
/*========================================================*/
/*================      PRINT MACRO     ==================*/
/*========================================================*/
/*========================================================*/

%macro p(d,T,N=);
    proc print data=&d. (obs=20) label split='*' noobs;
        title ;
        TITLE1 '==========================================';
        TITLE2 '                                          ';
        TITLE3 "              PRINT STEP :  &T.           ";
        TITLE4 "                                          ";
        TITLE5 "Print of dataset &d. (MAX &n. observations)";
        TITLE6 '                                          ';
        TITLE7 '==========================================';
    run;

    TITLE ' ';
%mend p;

/*========================================================*/
/*========================================================*/
/*================      MAIN MACRO      ==================*/
/*=======================NFT (NAMES FROM TEMPLATE) =======*/
/*========================================================*/

%macro NFT(indsn,template,outdsn);
    proc contents data=&template.
                   out=_temp_out noprint;
    run;    %P(_TEMP_OUT,D)

    proc transpose
         data=_temp_out
          out=_temp_out_transposed(drop=_name_ _label_);
         var name;
    run;    %P(_TEMP_OUT_TRANSPOSED,E)

    proc sql; select count(*) into :c from _temp_out;quit;

    data _null_;
        v2=compress('col'||&c.);
```



```
        call symput('v2',v2);
    run;

    data want;
        set _temp_out_transposed;
        length c2 v $32767.;
        array dat[&c.] col1- &v2.;

        %do i=1 %to &c.;
            c2=dat[&i.];
            v=trim(v) || " " || c2;
        %end;
        call symput('var_list',v);
    run;    %P(WANT,F)

    data &outdsn.;
        set &indsn.(keep=&var_list.);
    run;    %P(&OUTDSN.,G)
%mend NFT;

/*========================================================*/
/*========================================================*/
/*================  MACRO INVOCATION    =================*/
/*========================================================*/
/*========================================================*/

%P(SASHELP.CLASS,A)

data all_vars;
    set sashelp.class;
run;
%P(all_vars,B)

data select_vars;
    set sashelp.class(obs=5
                      keep=name
                           age
                           sex);
run;

%P(select_vars,C)

%NFT(all_vars,select_vars,new)
```

RESULTS OF MACRO-EXECUTION



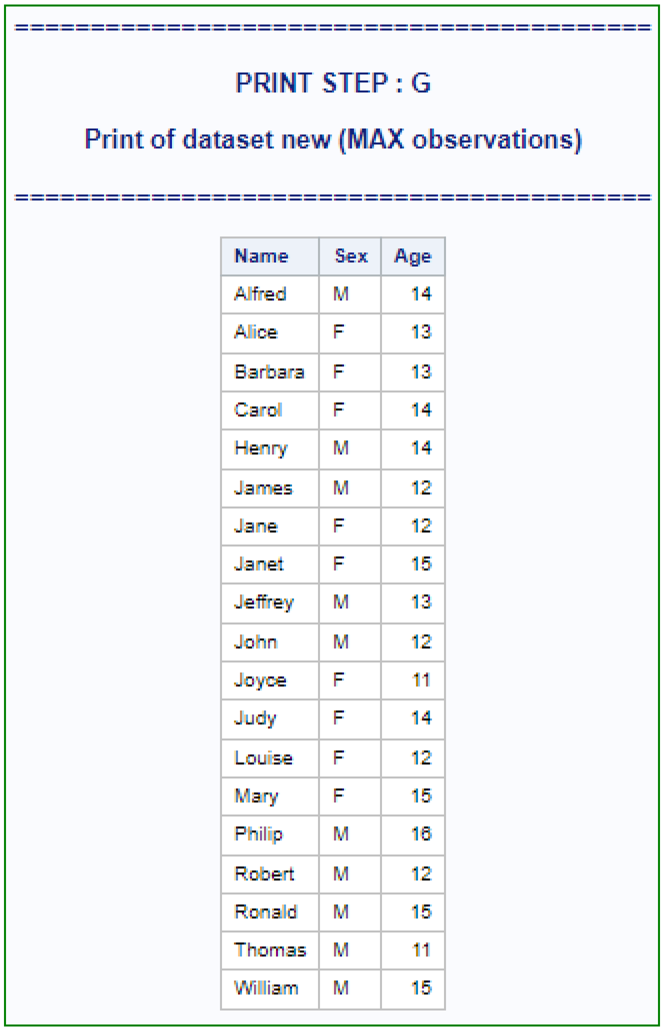
==

PRINT STEP : A

Print of dataset SASHELP.CLASS (MAX observations)

==

Name	Sex	Age	Height	Weight
Alfred	M	14	69.0	112.5
Alice	F	13	56.5	84.0
Barbara	F	13	65.3	98.0
Carol	F	14	62.8	102.5
Henry	M	14	63.5	102.5
James	M	12	57.3	83.0
Jane	F	12	59.8	84.5
Janet	F	15	62.5	112.5
Jeffrey	M	13	62.5	84.0
John	M	12	59.0	99.5
Joyce	F	11	51.3	50.5
Judy	F	14	64.3	90.0
Louise	F	12	56.3	77.0
Mary	F	15	66.5	112.0
Philip	M	16	72.0	150.0
Robert	M	12	64.8	128.0
Ronald	M	15	67.0	133.0
Thomas	M	11	57.5	85.0
William	M	15	66.5	112.0



===

PRINT STEP : G

Print of dataset new (MAX observations)

===

Name	Sex	Age
Alfred	M	14
Alice	F	13
Barbara	F	13
Carol	F	14
Henry	M	14
James	M	12
Jane	F	12
Janet	F	15
Jeffrey	M	13
John	M	12
Joyce	F	11
Judy	F	14
Louise	F	12
Mary	F	15
Philip	M	16
Robert	M	12
Ronald	M	15
Thomas	M	11
William	M	15

Above shown Step A is the input dataset and Step G is the final output dataset. All intermediate prints are suppressed for brevity.

LIMITATIONS

- The macro expects SAS datasets as inputs for first and second parameters. It is not capable of handling flat files at the current state. If flat files are to be used, they first need to be converted to SAS datasets.
- The width of the variable list is currently limited to the max size of $32767. Care needs to be ensured to custom process if it exceeds this limit.

CONCLUSION

The code was created to help programmers to save time by simple invocation of this macro by passing the appropriate parameters as a way of “Power Programming” by cutting down the development time and getting results quicker with smaller code. Also, modularizing this may help with easier management and standardization of code if used by multiple users as long as the macro is in a SAS “autocall” or a %include code to be made available to a group of users or across an organization.

RECOMMENDED READING

- Base SAS® Procedures Guide
- SAS® Macro Language: Reference Guide®
- SAS® Macro Programming

CONTACT INFORMATION

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

Dr. Kannan Deivasigamani

TextDrK@gmail.com

https://www.linkedin.com/in/kannandeivasigamani