Dynamic and Reproducible Health Analytics Using SAS Macros

May 20, 2026

Overview

- Why SAS Macros
- SAS Macro Elements
  - Macro variables
  - Macro functions
  - Macro statements
  - Macros
- Examples
  - Missingness summary macro
  - Automated data profiling macro


THAT'S WHAT I DO
I WRITE MACROS
%macro
&var
%if SQL
%else &var
%mend
%let
then
%if
&var
&&
I CREATE MACROS
AND I KNOW THINGS


SAS MACROS
WE KNEAD EM
YOU NEED EM

When to Use SAS Macros

Dynamic output generation

```
/* Define a macro to export a SAS data to Excel */
%MACRO Z_export_to_excel(datain=, out_file=);
    proc export data=&datain.
        outfile="&out_file"
        dbms=xlsx replace;
        sheet="&datain";
    run;
%MEND Z_export_to_excel;
```

```
%LET startdt='01APR2024'd;
%LET enddt='31MAR2025'd;
PROC SQL;
    CREATE TABLE nacrs_fy24 AS
    SELECT *,
        DATEPART(reg_dt_tm) AS reg_dt FORMAT=DATE9.,
        DATEPART(disposition_dt_tm) AS disposition_dt
FORMAT=DATE9.
    FROM spedw.vw_nacrs
    WHERE &startdt ≤ DATEPART(reg_dt_tm) ≤ &enddt;
QUIT;
```

Data selection for certain time windows

```
%MACRO z_missing_summary(datain=);
%LOCAL n c;
/*--- Step 1: Count numeric and character ---*/
DATA _NULL_;
    IF 0 THEN SET &datain;
    ARRAY nn _NUMERIC_;
    ARRAY cc _CHARACTER_;
     ......
```

Automated validation checks across datasets

```
/* Set missing to 0 for a list of
variables */
%macro lowrisk0(varlist);
%let nvar=%sysfunc(countw(
    %do i=1 %to &nvar;
        %let vv=%scan(&varlist,
        if &vv=. then &vv=0;
    %end;
%mend lowrisk0;
```

Repetitive tasks

Batch Variable Selection and Transformation

```
/* Add suffix to variable names */
%MACRO rename_bysuffix(varlist, suffix);
%LET nvar=%sysfunc(countw(&varlist));
    %DO i=1 %TO &nvar;
        %LET var=%scan(&varlist, &i);
        RENAME &var=&var._&suffix;
    %END;
%MEND rename_bysuffix;
```

ADVANTAGES OF SAS MACROS

Introduction

SAS macros are powerful tools for automating repetitive tasks and generating dynamic code in SAS programming.

KEY ADVANTAGES

Code Reusability

- Write once, use many times across multiple programs.

Dynamic Code Generation

- Create flexible code that adapts to varying inputs.

Simplification of Complex Tasks

- Break down complicated processes into manageable.

Improved Efficiency

- Reduce coding time and minimize errors.

Example of Data Quality Validation

MANUAL APPROACH (NON-MACRO)

Validating SALES_JAN Table:

```
data SALES_JAN;
  set WORK.TEMP_JAN;
  if amount <> 0 then put 'ERROR: invalid amount in JAN';
  * ... more validation code ...
run;
```

Validating SALES_FEB Table:

```
dats SALES_FEB;
  set WORK.TEMP_FEB;
  if amount <= 0 then put 'ERROR: invalid amount in FEB';
  * ... repeat of validation code ...
run;
```

Repeat Code (prone to errors)

MACRO APPROACH (DYNAMIC)

```
%macro Validate(table_name=);
  data &data&_name.;
    set WORK.TEMP_&table_name.;
    %f amount <= 0 then put 'ERROR: invalid amount in &
    * ... all validation code here ...
  run;
%mend Validate;
```

Dynamic Execution:

```
%Validate(table_name=SALES_JAN);
%* ... can validate any number of tables ...
%* Update rule once to apply everywhere.
```

Update rule once here to fix everywhere

BEST PRACTICES

- Use meaningful names
- Include error handling
- Add comments for clarity

SAS macros streamline development, enhance flexibility, and boost productivity. Start leveraging SAS macros today!

LEARN MORE →

4 CORE ELEMENTS OF SAS MACROS

Powerful tools that extend and automate your SAS programs

1 MACRO VARIABLES

Store values that can be referenced throughout a macro program.

USER-DEFINED

Created by you to store and manage your own values.

```
%LET name = John;
```

AUTOMATIC

Created by SAS automatically to provide information.

```
&SYSDATE9
```

2 MACROS (MACRO DEFINITIONS)

Reusable, named blocks of SAS code that perform a specific task.

```
%MACRO mymacro(param);
    /* SAS code */
%MEND mymacro;
```

3 MACRO FUNCTIONS

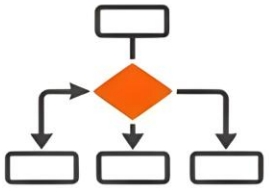
Perform operations and return values that can be used in macro expressions.

%EVAL(expression)	Evaluate arithmetic expressions.
%SCAN(string, n , d)	Extract the n^n word from a string.
%SUBSTR(string, start, len)	Extract a substring from a string.

Many more: %UPCASE, %LOWCASE, %LENGTH ...

4 MACRO STATEMENTS

Control the flow of macro execution and logic.



%IF condition %THEN %DO; /* code */ %END;	Execute code based on a condition.

%IF condition %THEN %DO;
  /* code */
%END; %ELSE %DO;
  /* code */
%END;
	Execute code if condition is true, otherwise, run alternate code.

%DO i = 1 %TO 10; /* code */ %END;	Iterate code a specific number of times.

WORK TOGETHER.

{x} Store values with variables → Build reusable macro code → Use functions to manipulate values → Control flow with statements

AUTOMATE. REUSE. ACHIEVE MORE.

A Macro Example

- Begin with a % and a macro keyword and end with semicolon (;)
- Let you write a block of code once and reuse it
- Works like a function with parameters

```
/* Define a macro to export a SAS data to Excel */
%MACRO Z_export_to_excel(datain=, out_file=);
    proc export data=&datain.
        outfile="&out_file"
        dbms=xlsx replace;
        sheet="&datain";
    run;
%MEND Z_export_to_excel;
```

```
/* Call the macro */
%LET outpath=/home/phsabc.ehcnet.ca/ruth.zhang/workspace/nd1233aa/RZhang/Dataset_profile;
%Z_export_to_excel(out_file = &outpath./Report/SASHELP_cars.xlsx, datain = SASHELP.CARS)
/* NOTE: The export data set has 428 observations and 15 variables. */
/* NOTE:
"/home/phsabc.ehcnet.ca/ruth.zhang/workspace/nd1233aa/RZhang/Dataset_profile/Report/SASHELP
_cars.xlsx" file was successfully created. */
```

Creating a Macro Variable List using PROC SQL

```
proc sql noprint;
  select distinct origin
    into :origin1-
    from sashelp.cars
     order by origin;
  %let numorigins = &sqlobs;
quit;
```

Get only unique values of ORIGIN.

The dash creates a series of sequential macro variables, one for each value returned by the query. No need for an upper bound.

Create one more macro variable so we know how many items are in our list.

```
11    %put _user_;
GLOBAL NUMORIGINS 3
GLOBAL ORIGIN1 Asia
GLOBAL ORIGIN2 Europe
GLOBAL ORIGIN3 USA
```

Example #1: Dynamic Report Creation

Macro Wizard Code – Part 1 of 2

```
%macro graph_stocks;

  proc sql noprint;
    select distinct stock
      into :stock1-
      from sashelp.stocks;
    %let numstocks = &sqlobs;
  quit;
```

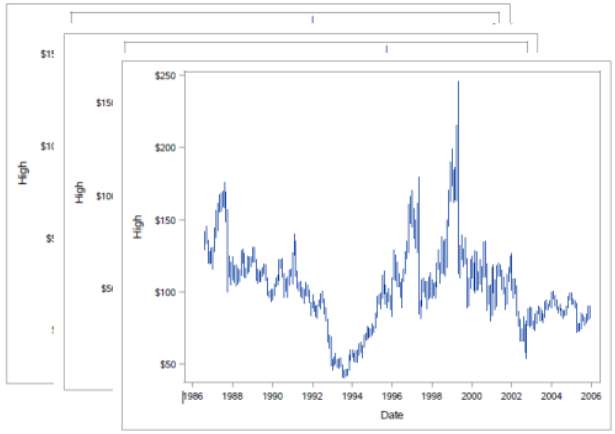
Use PROC SQL to place the unique values of STOCK in a macro variable list.

```
GRAPH_STOCKS NUMSTOCKS 3
GRAPH_STOCKS STOCK1 IBM
GRAPH_STOCKS STOCK2 Intel
GRAPH_STOCKS STOCK3 Microsoft
```


sas innovate

Example #1: Dynamic Report Creation

Output



PDF IBM.pdf PDF Intel.pdf PDF Microsoft.pdf

A separate PDF file for each unique value of STOCK

sas innovate

Example #1: Dynamic Report Creation

Macro Wizard Code – Part 2 of 2

```
  %do i = 1 %to &numstocks;
    ods pdf file="&&stock&i...pdf";
      proc sgplot data=sashelp.stocks;
        where stock = "&&stock&i";
        highlow x=date high=high low=low;
      run;
    ods pdf close;
  %end;
%mend graph_stocks;

%graph_stocks
```

Each iteration of the %DO loop generates the code to plot one stock.

```
&&stock&i...pdf
      ↓
&stock1..pdf
      ↓
IBM.pdf
```

Two-pass macro variable resolution

sas innovate

Obs	Stock	Date	Open	High	Low	Close	Volume	AdjClose
1	IBM	01DEC05	$89.15	$89.92	$81.56	$82.20	5,976,252	$81.37
2	IBM	01NOV05	$81.85	$89.94	$80.64	$88.90	5,556,471	$88.01
3	IBM	03OCT05	$80.22	$84.60	$78.70	$81.88	7,019,666	$80.86
4	IBM	01SEP05	$80.16	$82.11	$76.93	$80.22	5,772,280	$79.22
5	IBM	01AUG05	$83.00	$84.20	$79.87	$80.62	4,801,386	$79.62
6	IBM	01JUL05	$74.30	$85.11	$74.16	$83.46	8,056,590	$82.23
7	IBM	01JUN05	$75.57	$77.73	$73.45	$74.20	6,439,536	$73.10
8	IBM	02MAY05	$76.88	$78.11	$72.50	$75.55	6,896,904	$74.43
9	IBM	01APR05	$91.49	$91.76	$71.85	$76.38	10,709,200	$75.05
10	IBM	01MAR05	$92.64	$93.73	$89.09	$91.38	5,025,627	$89.79
11	IBM	01FEB05	$93.67	$94.97	$91.55	$92.58	4,455,657	$90.97
12	IBM	03JAN05	$98.97	$99.10	$91.44	$93.42	5,960,945	$91.62
13	IBM	01DEC04	$94.50	$99.00	$94.47	$98.58	5,043,800	$96.68

Automated Data Profiling SAS Macro

- Examine a dataset and generate the following in a multiple tab Excel file

Dataset overview	Number of observations, data field list, and number of duplications
Missingness for all variables	Number of non-missing rows, number and % of missing rows
Date and datetime variables	Date range Count over years
Numeric variables	Descriptive statistics Histogram
Character variables	Number of distinct levels Top categories Bar chart
Sample data for randomly selected 20 patients	If a patient has multiple rows, all rows will be selected

REFER TO: DATA_PROFILE_MACRO_DOCUMENT

%data_profile_macro

```
%LET outpath=/home/phsabc.ehcnet.ca/ruth.zhang/workspace/nd1233aa/RZhang/Dataset_profile;
PROC SQL;
        CREATE TABLE nacrs_from2023 AS
        SELECT *,
                DATEPART(reg_dt_tm) AS reg_dt FORMAT=DATE9.,
                DATEPART(disposition_dt_tm) AS disposition_dt FORMAT=DATE9.
        FROM spedw.vw_nacrs
         WHERE '01JAN2023'd<=DATEPART(reg_dt_tm);
QUIT;

%INC "&outpath./SASMacro/Z_Data_profile_macro_v2.sas";
%data_profile_macro(datain = nacrs_from2023,
 key = patient_master_key,
 eventdt = reg_dt,
 outpath = &outpath./Report/);
```

REFER TO: DATAPROFILE_NACRS_FROM2023_20260504.xlsx

DATAPROFILE_NACRS_FROM2023_20260504.xlsx

Data profiling for: NACRS_from2023.
Total unique rows: 5850915.
Removed duplicate rows (duplicate by all variables): 0.
Total unique patients: 2448560.
Total unique patient + reg_dt: 5739468.

NAME	TYPE	VARNUM	LABEL	FORMAT	INFORMAT
patient_master_key	1	1	patient_master_key		
nacrs_id	1	2	nacrs_id		
file_year	2	3	file_year	$	$
batch_period	1	4	batch_period		
ed_visit_indicator	1	5	ed_visit_indicator		
hospital_prov	2	6	hospital_prov	$	$
hospital	2	7	hospital	$	$
hospital_from	2	8	hospital_from	$	$
hospital_to	2	9	hospital_to	$	$
visit_disposition	2	10	visit_disposition	$	$
reg_dt_tm	1	11	reg_dt_tm	DATETIME	DATETIME
triage_dt_tm	1	12	triage_dt_tm	DATETIME	DATETIME
assess_dt_tm	1	13	assess_dt_tm	DATETIME	DATETIME
left_er_dt_tm	1	14	left_er_dt_tm	DATETIME	DATETIME
disposition_dt_tm	1	15	disposition_dt_tm	DATETIME	DATETIME
triage_level	2	16	triage_level	$	$
admit_ambulance	2	17	admit_ambulance	$	$

Summary | Missing | Date | Numerical | Categorical | Top10_Levels | SampleData | +

Missing summary

Variable	Label	Type	N_Non_Missing	N_Missing	Pct_Missing	N_Total
patient_master_key	patient_master_key	N	5,850,915	0	0	5,850,915
nacrs_id	nacrs_id	N	5,850,915	0	0	5,850,915
batch_period	batch_period	N	5,850,915	0	0	5,850,915
ed_visit_indicator	ed_visit_indicator	N	5,846,613	4,302	0.07	5,850,915
reg_dt_tm	reg_dt_tm	N	5,850,915	0	0	5,850,915
triage_dt_tm	triage_dt_tm	N	5,847,479	3,436	0.06	5,850,915
assess_dt_tm	assess_dt_tm	N	5,378,905	472,010	8.07	5,850,915
left_er_dt_tm	left_er_dt_tm	N	4,880,015	970,900	16.59	5,850,915
disposition_dt_tm	disposition_dt_tm	N	5,849,999	916	0.02	5,850,915
age_year	age_year	N	5,849,055	1,860	0.03	5,850,915
age_month	age_month	N	5,850,915	0	0	5,850,915
lha	lha	N	5,850,915	0	0	5,850,915
cdu_in_dt_tm	cdu_in_dt_tm	N	58,087	5,792,828	99.01	5,850,915
cdu_out_dt_tm	cdu_out_dt_tm	N	58,087	5,792,828	99.01	5,850,915
chsa_2018_id	chsa_2018_id	N	0	5,850,915	100	5,850,915
chsa_2022_id	chsa_2022_id	N	0	5,850,915	100	5,850,915
ed_inv_tech_cnt_1	ed_inv_tech_cnt_1	N	0	5,850,915	100	5,850,915

Missing Summary SAS Macro

- Summarize missingness for all variables in a SAS dataset
- Works even if there was no character or numeric variables
- Report variable names, labels, types, counts of missing and non-missing values, and percentage missing
- Temporary arrays used ensure efficient processing

```
%MACRO z_missing_summary(datain=);
  %LOCAL n c;
  /*--- Step 1: Count numeric and character variables (metadata only) ---*/
  DATA _NULL_;
    IF 0 THEN SET &datain;
    ARRAY nn _NUMERIC_;
    ARRAY cc _CHARACTER_;
    /* L. specifies that the macro variable is stored in the most local symbol table that exists */
    CALL SYMPUTX('n', DIM(nn), 'L');
    CALL SYMPUTX('c', DIM(cc), 'L');
  RUN;
  %PUT === &datain has &n numeric variables;
  %PUT === &datain has &c character variables;
```

```
/*--- Step 2: One-pass accumulation; guard against 0-dimension arrays ---*/
DATA _missingsumm(KEEP=varname varlab vartype nonmissing nmissing misspct ntotal);
  SET &datain NOBS=nobs END=end;
  /* Conditionally define the TEMP arrays only when sizes > 0. */
  %IF (&n > 0) %THEN %DO;
     ARRAY nn _NUMERIC_;
    ARRAY nf(&n) _TEMPORARY_;
   /* Accumulate non-missing counts row-by-row */
    DO OVER nn;
      nf(_I_) + NOT MISSING(nn);
    END;
  %END;
  %IF (&c > 0) %THEN %DO;
   ARRAY cc $ _CHARACTER_;
    ARRAY cf(&c) _TEMPORARY_;
    DO OVER cc;
      cf(_I_) + NOT MISSING(cc);
    END;
  %END;
```

```
  /* Output results only once, at end of file */
  IF end THEN DO;
    LENGTH varname varlab $100;
    ntotal = nobs;

      %IF (&n > 0) %THEN %DO;
        vartype = 'N';
        DO OVER nn;
          varname = VNAME(nn);
          varlab  = VLABEL(nn);
          nonmissing = nf(_I_);
          nmissing  = nobs - nonmissing;
          misspct   = ROUND(100 * nmissing / nobs, 0.01);
          OUTPUT;
        END;
      %END;
      %IF (&c > 0) %THEN %DO;
        vartype = 'C';
        DO OVER cc;
          varname = VNAME(cc);
          varlab  = VLABEL(cc);
          nonmissing = cf(_I_);
          nmissing  = nobs - nonmissing;
          misspct   = ROUND(100 * nmissing / nobs, 0.01);
          OUTPUT;
        END;
      %END;
  END; /* end=1 */
RUN;
```

```
  /*--- Step 3: Report ---*/
  PROC REPORT DATA=_missingsumm NOWD;
    DEFINE varname / LEFT "Variable";
    DEFINE varlab / LEFT "Label";
    DEFINE vartype / CENTER "Type";
    DEFINE nonmissing / CENTER "N_Non_Missing";
    DEFINE nmissing / CENTER "N_Missing";
    DEFINE misspct / CENTER "Pct_Missing";
    DEFINE ntotal / CENTER "N_Total";
    FORMAT ntotal nonmissing nmissing COMMA12.;
  RUN;

  /*--- Step 4: Cleanup ---*/
  PROC DATASETS LIB=work NOLIST;    DELETE _missingsumm;  RUN;
  %PUT NOTE: [z_missing_summary] Macro completed successfully.;
%MEND z_missing_summary;
```

Macro Variables Scopes

%Global

- Use global variables when a value must be accessed by multiple nested macros or throughout the entire script.
- Variables defined using **%global** are stored in the Global Symbol Table.
- If a variable already exists, **%global** keeps its current value. If not, it creates a new one with a null value.

%Local

- Best practice: use local variables within macros to prevent overwriting global variables of the same name, such as using i for a loop.
- Variables defined using **%local** are stored in a local symbol table created specifically for that macro.
- When the macro stops running, the variable and its value disappear.

if-then-else and %if-%then-%else

sas

```sas
%macro check_data(report_type);
    data results;
        set mydata;
        /* 'if' evaluates for every row in the dataset */
        if score > 90 then grade = 'A';
        else grade = 'B';
    run;

    /* '%if' evaluates once to decide if we should print the report */
    %if &report_type = PRINT %then %do;
        proc print data=results;
        run;
    %end;
%mend check_data;
```

Feature	`if-then-else`	`%if-%then-%else`
Execution Time	During DATA step execution.	During macro compilation/resolution.
Evaluates	Dataset variables (row by row).	Macro variables or text constants.
Context	Only valid inside a DATA step.	Valid only inside a `%macro` definition.
Purpose	Controls data processing logic.	Conditionally generates SAS code.

if-then-else and %if-%then-%else

When you DO use `%if` in a Data Step

You use `%if` when the code structure itself needs to change based on a macro parameter or condition. SAS Communities

For example, you might want to conditionally apply a formula or conditionally include an entirely different variable:

sas

```
data combined_data;
    set input_data;

    /* %IF controls if the line of code is generated at compile time */
    %if &APPLY_TAX = YES %then %do;
        final_price = base_price * 1.05;
    %end;
    %else %do;
        final_price = base_price;
    %end;

    /* Standard IF controls row-level logic at execution time */
    if status = 'Active' then active_flag = 1;
    else active_flag = 0;
run;
```