# Efficiency Programming with Macro Variable Arrays

Veronica Renauldo is an Associate Statistical Programmer at QST Consultations, LTD located in Allendale, MI. She is involved with data cleaning and data set/output production and quality control in numerous clinical trials, and works on both CDISC and non-CDISC pharmaceutical and health care device studies. She also assists in training new employees for the company. Veronica received a Bachelor's degree in Statistics with a minor in Mathematics followed by a Master's degree in Biostatistics in April of 2014 and 2016, respectively, from Grand Valley State University. She is a SAS certified Base SAS 9 programmer and has been using SAS for 7 years. Working in a fast paced contract research organization where efficiency and reproducibility are key in most aspects of her work, Veronica has developed strong macro processing skills that can be employed in a wide array of statistical inference and data set manipulation.

# Outline

- Data step Arrays
- Macro Variables
- Macro Functions
- Array of Macro variables
- Macro Variable Arrays
  - Definition
  - Ways to create a macro variable array using examples
  - Advanced Macro Variable Array Examples
- Questions

# Data Step Arrays

- Increases efficiency
- Applies logic to several variables of the same type
- Use looping structures
- The dimension of an array: explicitly defined or use of an asterisk

```
data one;
   array numz (10);
   array chrz (10) $200;
   do i = 1 to dim(numz);
      numz(i) = i;
      chrz(i) = 'Character Version '||strip(put(i, 2.));
   end;
run;
```

# Macro Variables

- Character stings used for symbolic substitutions within SAS code
  - Numeric macro variables are technically character
  - `%let popn = 160;`
- Maximum length = 65,534 characters.
- Calls prefixed with at least one & compile with a period
  - `&popn.`
- Have a constant value that is set in two different ways
  - Automatically by SAS
  - User defined

# Automatic Macro Variables

- Macro variables that are set when a SAS product is deployed.
- %PUT _ALL_; will show all macro variables (automatic and user defined) in the log
- SYSDATE = date at which the SAS produced was invoked in the DATE7 format

```
      %put _all_;
AUTOMATIC SYSDATE 03AUG18
AUTOMATIC SYSDATE9 03AUG2018
AUTOMATIC SYSDAY Friday
AUTOMATIC SYSDEVIC
AUTOMATIC SYSDMG 0
AUTOMATIC SYSDSN          _NULL_
AUTOMATIC SYSENCODING wlatin1
AUTOMATIC SYSENDIAN LITTLE
AUTOMATIC SYSENV FORE
AUTOMATIC SYSERR 0
AUTOMATIC SYSERRORTEXT
AUTOMATIC SYSFILRC 0
AUTOMATIC SYSINDEX 0
AUTOMATIC SYSINFO 0
AUTOMATIC SYSJOBID 10184
AUTOMATIC SYSLAST _NULL_
```

Type Name Value

# User Defined Macro Variables

- Created by the user
- Anything the user deems worthy of being substituted
- Examples
  - Footnotes that will be used for multiple outputs

```
%let fnote1 = Observed Data Only;
```

  - Population N value that will be used in several calculations

```
%let N = 68;
```

  - Treatment arm n values needed for calculations

```
%let N1 = 24;
%let N2 = 44;
```

# Macro Functions

# Macro Functions

- Like data step variable functions but on macros
- Different syntax than data step variable functions but most macro functions require the same format as data step variable functions
- Examples:
  - %UPCASE: up-cases all text within the specified macro variable
  - %LENGTH: returns the text of the specified macro variable
  - %LOWCASE: low-cases all text within specified macro variable

# Macro Functions: %BQUOTE

**%BQUOTE(*string*)**

- Quoting Function
- Allows a user to mask any operator or special character in a macro variable.
  - Special Characters: & % ' " ( ) + - * / < > = ¬ ^ ~ ; , blank
  - Operators: AND OR NOT EQ NE LE LT GE GT
- Can handle unmatched pairs of symbols
- Execution time function: Masking occurs during macro execution (mask resolved values of a macro variable)

Ex. %let title = %BQUOTE(Best Graphic Ever;);

# Macro Functions: %SCAN

**%SCAN(*argument, n, <delimiters>*)**

- Works the same as a DATA step SCAN function
- Searches the argument and returns the *nth* word.
  - A word is one or more characters separated by one or more delimiters.
- Default delimiters: blank . < ( + & ! $ * ) ; ^ - / , % |

# Macro Functions: COUNTW and %SYSFUNC

**COUNTW(*string, chars, modifiers)***

- Counts the number of words in a character string

**%SYSFUNC(function)**

- Executes SAS datastep functions or user written functions
- Almost all functions that can be used in a data step can be used on macro variables when using this macro function

***SAS Functions Not Available with %SYSFUNC and %QSYSFUNC***

| | | |
|---|---|---|
| All Variable Information Functions | ALLCOMB | ALLPERM |
| DIF | DIM | HBOUND |
| IORCMSG | INPUT | LAG |
| LBOUND | LEXCOMB | LEXCOMBI |
| LEXPERK | LEXPERM | MISSING |
| PUT | RESOLVE | SYMGET |

# Array of Macro Variables

# Array of Macro Variables

```
%let race1 = white;
%let race2 = asian;
%let race3 = other;

%macro print();
   %do i = 1 %to 3;
      %put &&race&i..;
   %end;
%mend;

%print();
```

```
MLOGIC(PRINT):  Beginning execution.
MLOGIC(PRINT):  %DO loop beginning; index variable I; start
      value is 1; stop value is 3; by value is 1.
MLOGIC(PRINT):  %PUT &&race&i..
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 1
SYMBOLGEN:  Macro variable RACE1 resolves to white
white
MLOGIC(PRINT):  %DO loop index variable I is now 2; loop will
      iterate again.
MLOGIC(PRINT):  %PUT &&race&i..
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 2
SYMBOLGEN:  Macro variable RACE2 resolves to asian
asian
MLOGIC(PRINT):  %DO loop index variable I is now 3; loop will
      iterate again.
MLOGIC(PRINT):  %PUT &&race&i..
SYMBOLGEN:  && resolves to &.
SYMBOLGEN:  Macro variable I resolves to 3
SYMBOLGEN:  Macro variable RACE3 resolves to other
other
MLOGIC(PRINT):  %DO loop index variable I is now 4; loop will
      not iterate again.
MLOGIC(PRINT):  Ending execution.
```

# Macro Variable Array

# Macro Variable Arrays (MVA)

- Singular macro variable that contains an entire array of elements to be processed
- Array elements can be data set names, variable names, variable values, etc.
- Elements are parsed out of the array to be used in the desired macro process

```
%let race = white asian other;
%let dats = one two three;
%let varz = weight height temp sysbp diabp pulse;
```

# What are Macro Variable Arrays Used For?

- Used for repetitive macro processing
- Maximizes the capabilities of macros
- Increases efficiency (reduces programmer errors)


Image from: https://marketingland.com/efficiency-arbitrage-two-strategies-performance-marking-178328

# Methods to create a MVA

1. Explicitly Defined
   a. %LET statement
   b. Macro Variable parameter within a Macro
2. Data step with CALL and SYMPUT functions
3. PROC SQL

# MVA Creation: %LET Statement

```
%let race = white asian other;


%macro print();
   %do i = 1 %to 3;
      %put %scan(&race., &i.);
   %end;
%mend;

%print();
```

# MVA Creation: %LET Statement

```
MLOGIC(PRINT):  Beginning execution.
MLOGIC(PRINT):  %DO loop beginning; index variable I; start
      value is 1; stop value is 3; by value is 1.
MLOGIC(PRINT):  %PUT %scan(&race., &i.)
SYMBOLGEN:  Macro variable RACE resolves to white asian other
SYMBOLGEN:  Macro variable I resolves to 1
white
MLOGIC(PRINT):  %DO loop index variable I is now 2; loop will
      iterate again.
MLOGIC(PRINT):  %PUT %scan(&race., &i.)
SYMBOLGEN:  Macro variable RACE resolves to white asian other
SYMBOLGEN:  Macro variable I resolves to 2
asian
MLOGIC(PRINT):  %DO loop index variable I is now 3; loop will
      iterate again.
MLOGIC(PRINT):  %PUT %scan(&race., &i.)
SYMBOLGEN:  Macro variable RACE resolves to white asian other
SYMBOLGEN:  Macro variable I resolves to 3
other
MLOGIC(PRINT):  %DO loop index variable I is now 4; loop will
      not iterate again.
MLOGIC(PRINT):  Ending execution.
```

# Array of Macro Variables vs Macro Variable Array

Array of Macro Variables

```
%let race1 = white;
%let race2 = asian;
%let race3 = other;

%macro print();
   %do i = 1 %to 3;
      %put &&race&i..;
   %end;
%mend;

%print();
```

%SCAN and other Macro Functions

One Macro Variable

Macro Variable

Consecutive Ampersands Resolve

Series of Ampersands Resolve

Macro Variable Array

```
%let race = white asian other;

%macro print();
   %do i = 1 %to 3;
      %put %scan(&race., &i.);
   %end;
%mend;

%print();
```

# MVA Creation: Macro Variable Parameter

- Macro will read in several permanent SAS data sets and outputs them to the work library for temporary use

Macro Variables

Macro Variable Array

```
%macro rin(lib, var, dat);
   %do i = 1 %to %sysfunc(countw(&dat.));
      proc sort data=&lib..%scan(&dat., &i.)
                out=r%scan(&dat., &i.);
         by &var.;
      run;
   %end;
%mend;
%rin(sashelp, height, class classfit fish gridded heart);
```

# MVA Creation: Macro Variable Parameter

```
MLOGIC(RIN):  Beginning execution.
MLOGIC(RIN):  Parameter LIB has value sashelp
MLOGIC(RIN):  Parameter VAR has value height
MLOGIC(RIN):  Parameter DAT has value class classfit fish
      gridded heart
MLOGIC(RIN):  %DO loop beginning; index variable I; start value
      is 1; stop value is 5; by value is 1.
MPRINT(RIN):   proc sort data=sashelp.class out= rclass;
MPRINT(RIN):   by height;
MPRINT(RIN):   run;

NOTE: There were 19 observations read from the data set
      SASHELP.CLASS.
NOTE: The data set WORK.RCLASS has 19 observations and 5
      variables.
NOTE: Compressing data set WORK.RCLASS increased size by 100.00
      percent.
      Compressed is 2 pages; un-compressed would require 1
      pages.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

# MVA Creation: Macro Variable Parameter

```
MLOGIC(RIN):  %DO loop index variable I is now 5; loop will
      iterate again.
MPRINT(RIN):   proc sort data=sashelp.heart out= rheart;
MPRINT(RIN):   by height;
MPRINT(RIN):   run;

NOTE: There were 5209 observations read from the data set
      SASHELP.HEART.
NOTE: The data set WORK.RHEART has 5209 observations and 17
      variables.
NOTE: Compressing data set WORK.RHEART decreased size by 31.19
      percent.
      Compressed is 75 pages; un-compressed would require 109
      pages.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds

MLOGIC(RIN):  %DO loop index variable I is now 6; loop will not
      iterate again.
MLOGIC(RIN):  Ending execution.
```

# MVA Creation: CALL SYMPUT

- A method of assigned a value to a macro variable within a DATA step
- Can be employed a regular DATA step or a DATA NULL step
- Objective of example: to obtain frequency data for origin and type for each manufacturer in the SASHELP.CARS dataset

# MVA Creation: CALL SYMPUT

Desired output: frequency data for origin and type for each manufacturer in the SASHELP.CARS dataset

```
data _null_;
   set sashelp.cars end = last;
   by make;
   length brand $5000;
   retain brand;
   if first.make then brand = strip(brand)||'*'||strip(make);
   if last then do;
      brand = strip(substr(brand, 2));
      call symput('brands', strip(brand));
   end;
run;
```

# MVA Creation: CALL SYMPUT

| Obs | Make | Model | Brand |
|---|---|---|---|
| 6 | Acura | 3.5 RL w/Navigation 4dr | *Acura |
| 8 | Audi | A4 1.8T 4dr | *Acura*Audi |
| 27 | BMW | X3 3.0i | *Acura*Audi*BMW |
| 427 | Volvo | V40 | *Acura*Audi*BMW*Buick*Cadillac*Chevrolet*Chrysler*Dodge*Ford*GMC*Honda*Hummer*Hyundai*Infiniti*Isuzu*Jaguar*Jeep*Kia*Land Rover*Lexus*Lincoln*MINI*Mazda*Mercedes-Benz*Mercury*Mitsubishi*Nissan*Oldsmobile*Pontiac*Porsche*Saab*Saturn*Scion*Subaru*Suzuki*Toyota*Volkswagen*Volvo |
| 428 | Volvo | XC70 | Acura*Audi*BMW*Buick*Cadillac*Chevrolet*Chrysler*Dodge*Ford*GMC*Honda*Hummer*Hyundai*Infiniti*Isuzu*Jaguar*Jeep*Kia*Land Rover*Lexus*Lincoln*MINI*Mazda*Mercedes-Benz*Mercury*Mitsubishi*Nissan*Oldsmobile*Pontiac*Porsche*Saab*Saturn*Scion*Subaru*Suzuki*Toyota*Volkswagen*Volvo |

# MVA Creation: CALL SYMPUT

Desired output: frequency data for origin and type for each manufacturer in the SASHELP.CARS dataset

```
69   %put &brands.;
Acura*Audi*BMW*Buick*Cadillac*Chevrolet*Chrysler*Dodge*Ford*GMC*
Honda*Hummer*Hyundai*Infiniti*Isuzu*Jaguar*Jeep*Kia*Land
Rover*Lexus*Lincoln*MINI*Mazda*Mercedes-Benz*Mercury*Mitsubishi*
Nissan*Oldsmobile*Pontiac*Porsche*Saab*Saturn*Scion*Subaru*Suzuk
i*Toyota*Volkswagen*Volvo
```

There are 38 manufacturers.

# MVA Creation: CALL SYMPUT

```
%macro frqz();
   %do i = 1 %to %sysfunc(countw(&brands., *));
      title "%scan(&brands., &i., *)";
      proc freq data=sashelp.cars;
         where make = "%scan(&brands., &i., *)";
         tables origin*type/list;
      run;
      title;
   %end;
%mend;

%frqz();
```

# MVA Creation: CALL SYMPUT

```
MLOGIC(FRQZ):  Beginning execution.
MLOGIC(FRQZ):  %DO loop beginning; index variable I; start
      value is 1; stop value is 38; by value is 1.
MPRINT(FRQZ):   title "Acura";
MPRINT(FRQZ):   proc freq data=sashelp.cars;
MPRINT(FRQZ):   where make = "Acura";
MPRINT(FRQZ):   tables origin*type/list;
MPRINT(FRQZ):   run;

NOTE: Writing HTML Body file: sashtml.htm
NOTE: There were 7 observations read from the data set
      SASHELP.CARS.
      WHERE make='Acura';
NOTE: PROCEDURE FREQ used (Total process time):
      real time           1.49 seconds
      cpu time            0.71 seconds
```

# MVA Creation: CALL SYMPUT

```
MLOGIC(FRQZ):  %DO loop index variable I is now 19; loop will
       iterate again.
MPRINT(FRQZ):   title "Land Rover";
MPRINT(FRQZ):   proc freq data=sashelp.cars;
MPRINT(FRQZ):   where make = "Land Rover";
MPRINT(FRQZ):   tables origin*type/list;
MPRINT(FRQZ):   run;

NOTE: There were 3 observations read from the data set
      SASHELP.CARS.
      WHERE make='Land Rover';
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.05 seconds
      cpu time            0.04 seconds
```

# MVA Creation: CALL SYMPUT

```
MLOGIC(FRQZ):  %DO loop index variable I is now 38; loop will
       iterate again.
MPRINT(FRQZ):    title "Volvo";
MPRINT(FRQZ):    proc freq data=sashelp.cars;
MPRINT(FRQZ):    where make = "Volvo";
MPRINT(FRQZ):    tables origin*type/list;
MPRINT(FRQZ):    run;

NOTE: There were 12 observations read from the data set
      SASHELP.CARS.
      WHERE make='Volvo';
NOTE: PROCEDURE FREQ used (Total process time):
      real time           0.06 seconds
      cpu time            0.04 seconds


MPRINT(FRQZ):    title;
MLOGIC(FRQZ):  %DO loop index variable I is now 39; loop will
      not iterate again.
MLOGIC(FRQZ):  Ending execution.
```

# MVA Creation: CALL SYMPUT

**Acura**

The FREQ Procedure

| Origin | Type | Frequency | Percent | Cumulative Frequency | Cumulative Percent |
|---|---|---|---|---|---|
| Asia | SUV | 1 | 14.29 | 1 | 14.29 |
| Asia | Sedan | 5 | 71.43 | 6 | 85.71 |
| Asia | Sports | 1 | 14.29 | 7 | 100.00 |

**Volvo**

The FREQ Procedure

| Origin | Type | Frequency | Percent | Cumulative Frequency | Cumulative Percent |
|---|---|---|---|---|---|
| Europe | SUV | 1 | 8.33 | 1 | 8.33 |
| Europe | Sedan | 9 | 75.00 | 10 | 83.33 |
| Europe | Wagon | 2 | 16.67 | 12 | 100.00 |

**Land Rover**

The FREQ Procedure

| Origin | Type | Frequency | Percent | Cumulative Frequency | Cumulative Percent |
|---|---|---|---|---|---|
| Europe | SUV | 3 | 100.00 | 3 | 100.00 |

# MVA Creation: PROC SQL

Desired output: one report of demographic information per student in SASHELP.CLASS

```
proc sql noprint;
    select distinct(name) into :names separated by ' '
    from sashelp.class;
    select count(distinct(name)) into :counts trimmed
    from sashelp.class;
quit;
```

```
99   %put counts = &counts.;
counts = 19
100  %put names = &names.;
names = Alfred Alice Barbara Carol Henry James Jane Janet
Jeffrey John Joyce Judy Louise Mary Philip Robert Ronald Thomas
William
```

# MVA Creation: PROC SQL

```
%macro reports();
   %do i = 1 %to &counts.;
      title "%scan(&names., &i.)";
      proc report data=sashelp.class nowd;
         where name = "%scan(&names, &i.)";
         columns sex age height weight;
         define sex / 'Sex';
         define age / 'Age';
         define height / 'Height';
         define weight / 'Weight';
      run;
      title;
   %end;
%mend;
```

# MVA Creation: PROC SQL

```
103  %reports();
NOTE: Writing HTML Body file: sashtml3.htm

NOTE: There were 1 observations read from the data set SASHELP.CLASS.
      WHERE name='Alfred';
NOTE: PROCEDURE REPORT used (Total process time):
      real time           1.26 seconds
      cpu time            0.56 seconds
...

NOTE: There were 1 observations read from the data set SASHELP.CLASS.
      WHERE name='William';
NOTE: PROCEDURE REPORT used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds
```

# MVA Creation: PROC SQL

**Alfred**

| Sex | Age | Height | Weight |
|---|---|---|---|
| M | 14 | 69 | 112.5 |

**William**

| Sex | Age | Height | Weight |
|---|---|---|---|
| M | 15 | 66.5 | 112 |

# Advanced Examples

# Multiple Macro Variable Arrays Example

- 4 correlations to be produced using SASHELP.CARS
  - Invoice amount vs MPG highway
  - Invoice amount vs number of cylinders
  - MSRP vs MPG highway
  - MSRP vs number of cylinders
- 2 macro variable arrays will produce 4 outputs

# Multiple Macro Variable Arrays Example

Macro Variable

Macro Variable Arrays

```
%macro corr(dat, var1, var2);
  %do i = 1 %to %sysfunc(countw(&var1.));
    %do k = 1 %to %sysfunc(countw(&var2.));
      proc corr data=sashelp.&dat. outp=c_%scan(&var1., &i.)_%scan(&var2., &k.);
        var %scan(&var1., &i.) %scan(&var2., &k.);
      run;

      proc print data=c_%scan(&var1., &i.)_%scan(&var2., &k.) noobs label;
        where lowcase(_name_) =  "%scan(&var1., &i.)";
        var %scan(&var2., &k.);
        label %scan(&var2., &k.) = "Corr %scan(&var1., &i.)-%scan(&var2., &k.)";
      run;
    %end;
  %end;
%mend;
```

# Multiple Macro Variable Arrays Example

```
%macro corr(dat, var1, var2);
  %do i = 1 %to %sysfunc(countw(&var1.));
    %do k = 1 %to %sysfunc(countw(&var2.));
      proc corr data=sashelp.&dat. outp=c_%scan(&var1., &i.)_%scan(&var2., &k.);
        var %scan(&var1., &i.) %scan(&var2., &k.);
      run;

      proc print data=c_%scan(&var1., &i.)_%scan(&var2., &k.) noobs label;
        where lowcase(_name_) =  "%scan(&var1., &i.)";
        var %scan(&var2., &k.);
        label %scan(&var2., &k.) = "Corr %scan(&var1., &i.)-%scan(&var2., &k.)";
      run;
    %end;
  %end;
%mend;

%corr(cars, invoice msrp, mpg_highway cylinders);
```

# Multiple Macro Variable Arrays Example

```
130  %corr(cars, invoice msrp, mpg_highway cylinders);
MLOGIC(CORR):  Beginning execution.
MLOGIC(CORR):  Parameter DAT has value cars
MLOGIC(CORR):  Parameter VAR1 has value invoice msrp
MLOGIC(CORR):  Parameter VAR2 has value mpg_highway cylinders
MLOGIC(CORR):  %DO loop beginning; index variable I; start value is 1; stop value is 2;
      by value is 1.
MLOGIC(CORR):  %DO loop beginning; index variable K; start value is 1; stop value is 2;
      by value is 1.
MPRINT(CORR):   proc corr data=sashelp.cars noprint outp=c_invoice_mpg_highway;
MPRINT(CORR):   var invoice mpg_highway;
MPRINT(CORR):   run;

NOTE: Writing HTML Body file: sashtml1.htm
NOTE: The data set WORK.C_INVOICE_MPG_HIGHWAY has 5 observations and 4 variables.
NOTE: Compressing data set WORK.C_INVOICE_MPG_HIGHWAY increased size by 100.00 percent.
      Compressed is 2 pages; un-compressed would require 1 pages.
NOTE: PROCEDURE CORR used (Total process time):
      real time           1.26 seconds
      cpu time            0.73 seconds
```

# Multiple Macro Variable Arrays Example

```
MPRINT(CORR):   proc print data=c_invoice_mpg_highway noobs label;
MPRINT(CORR):   where lowcase(_name_) = "invoice";
MPRINT(CORR):   var mpg_highway;
MPRINT(CORR):   label mpg_highway = "Corr invoice and mpg_highway";
MPRINT(CORR):   run;

NOTE: There were 1 observations read from the data set WORK.C_INVOICE_MPG_HIGHWAY.
      WHERE LOWCASE(_name_)='invoice';
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.02 seconds
      cpu time            0.03 seconds
```

# Multiple Macro Variable Arrays Example

| Corr invoice and mpg_highway |
| --- |
| -0.43459 |

| Corr invoice and cylinders |
| --- |
| 0.64523 |

| Corr msrp and mpg_highway |
| --- |
| -0.43962 |

| Corr msrp and cylinders |
| --- |
| 0.64974 |

# %BQUOTE Function with MVAs Example

- Desired output: Mean summaries of Mother's age and pregnancy weight gain from Infant birth weight (bweight) SASHELP data set with the title of each summary containing the label of the variable used

Dataset | Variable(s) | Label(s)

```
%macro means(dat, var, lab);
   %do i = 1 %to %sysfunc(countw(&var.));
      title "%scan(&lab., &i., ~)";
      proc means data=sashelp.&dat.;
         var %scan(&var., &i.);
      run;
      title;
   %end;
%mend;
```

# %BQUOTE Function with MVAs Example

```
%macro means(dat, var, lab);
   %do i = 1 %to %sysfunc(countw(&var.));
      title "%scan(&lab., &i., ~)";
      proc means data=sashelp.&dat.;
         var %scan(&var., &i.);
      run;
      title;
   %end;
%mend;

%means(bweight, mom_age m_wtgain, %bquote(Mother's
Age)~%bquote(Mother's Pregnancy Weight Gain));
```

# %BQUOTE Function with MVAs Example

```
146  %means(bweight, mom_age m_wtgain, %bquote(Mother's Age)~%bquote(Mother's Pregnancy
146! Weight Gain));
MLOGIC(MEANS):  Beginning execution.
MLOGIC(MEANS):  Parameter DAT has value bweight
MLOGIC(MEANS):  Parameter VAR has value mom_age m_wtgain
MLOGIC(MEANS):  Parameter LAB has value ┘Mother◄s Age□~┘Mother◄s Pregnancy Weight Gain□
MLOGIC(MEANS):  %DO loop beginning; index variable I; start value is 1; stop value is 2;
      by value is 1.

NOTE: There were 50000 observations read from the data set SASHELP.BWEIGHT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.04 seconds
      cpu time            0.06 seconds


MLOGIC(MEANS):  %DO loop index variable I is now 2; loop will iterate again.

NOTE: There were 50000 observations read from the data set SASHELP.BWEIGHT.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds


MLOGIC(MEANS):  %DO loop index variable I is now 3; loop will not iterate again.
MLOGIC(MEANS):  Ending execution.
```

# %BQUOTE Function with MVAs Example

**Mother's Age**

The MEANS Procedure

| Analysis Variable : mom_age | | | | |
|---|---|---|---|---|
| N | Mean | Std Dev | Minimum | Maximum |
| 50000 | 0.4161400 | 5.7284539 | -9.0000000 | 18.0000000 |

**Mother's Pregnancy Weight Gain**

The MEANS Procedure

| Analysis Variable : m_wtgain | | | | |
|---|---|---|---|---|
| N | Mean | Std Dev | Minimum | Maximum |
| 50000 | 0.7092200 | 12.8761168 | -30.0000000 | 68.0000000 |

# Conclusion

- Macros optimize repetitive tasks on data sets, variables, or variable values
- Macro variable arrays take this a step further
  - Combines array processing with macro language
  - Increase robustness of macros
  - Can be automated
  - Reduce programmer error
- Ways to create a macro variable array
  - Explicitly defined
    - %LET Statement
    - Macro parameter
  - CALL SYMPUT
  - PROC SQL

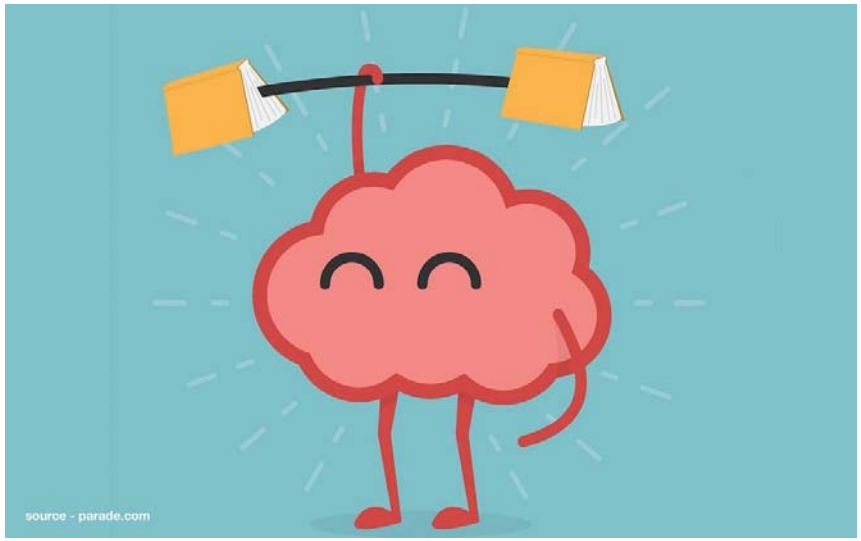


Image from: https://psychologycompass.com/blog/tactics-to-stop-worrying/

# Any questions?


Name: Veronica Renauldo

Organization: QST Consultations, LTD

Work Phone: (616) 892-3723

E-Mail: vrenauldo@qstconsultations.com