Introduction to SAS

Why use statistical packages

- Built-in functions
- Data manipulation
- Updated often to include new applications
- Different packages complete certain tasks more easily than others
- Packages we will introduce
  - SAS
  - R (S-plus)

SAS

- Easy to input and output data sets
- Preferred for data manipulation
- “proc” used to complete analyses with built-in functions
- Macros used to build your own functions

Outline

- SAS Structure
- Efficient SAS Code for Large Files
- SAS Macro Facility

Common errors

- Missing semicolon
- Misspelling
- Unmatched quotes/comments
- Mixed proc and data statement
- Using wrong options

SAS Structure

- Data Step: input, create, manipulate or output data
  - Always start with a data line
  - Ex. data one;
- Procedure Step: complete an operation on data
  - Always start with a proc line
  - Ex. proc contents;

Statements for Reading Data

- data statement names the data set you are making
- Can use any of the following commands to input data
  - infile Identifies an external raw data file to read with an INPUT statement
  - input Lists variable names in the input file
  - cards Indicates internal data
  - set Reads a SAS data set

Example

```
data temp;
infile ‘g:\shared\BIO271summer\baby.csv' delimiter=','
    dsd;
input id headcir length bwt gestwks mage mnocig
    mheight mppwt fage fedyrs fnocig fheig;
run;
proc print data = temp (obs=10);
run;
```

Delimiter Option

- blank space (default)
- DELIMITER= option specifies that the INPUT statement use a character other than a blank as a delimiter for data values that are read with list input

Delimiter Example

Sometimes you want to input the data yourself

Try the following data step:

```
data nums;
infile datalines dsd delimiter='&';
input X Y Z;
datalines;
1&2&3
4&5&6
7&8&9 ;
```

Notice that there are no semicolons until the end of the datalines

DSD option

- Change how SAS treats delimiters when list input is used and sets the default delimiter to a comma. When you specify DSD, SAS treats two consecutive delimiters as a missing value and removes quotation marks from character values.
- Use the DSD option and list input to read a character value that contains a delimiter within a quoted string. The INPUT statement treats the delimiter as a valid character and removes the quotation marks from the character string before the value is stored. Use the tilde (~) format modifier to retain the quotation marks.

Example: Reading Delimited Data

SAS data step:

```
data scores;
  infile datalines delimiter=',';
  input test1 test2 test3;
  datalines;
91,87,95
97,,92
,1,1
;
```

Output:

Obs	test1	test2	test3
1	91	87	95
2	97	92	1

Example: Correction

SAS data step

```
data scores;
  infile datalines delimiter=',' dsd;
  input test1 test2 test3;
  datalines;
91,87,95
97,,92
,1,1
;
```

Output:

Obs	test1	test2	test3
1	91	87	95
2	97	.	92
3	.	1	1

Modified List Input

Read data that are separated by commas and that may contain commas as part of a character value:

```
data scores;
infile datalines dsd;
input Name : $9. Score Team : $25. Div $;
datalines;
Joseph,76,"Red Racers, Washington",AAA
Mitchel,82,"Blue Bunnies, Richmond",AAA
Sue Ellen,74,"Green Gazelles, Atlanta",AA
;
```

Modified List Input

Output:

Obs	Name	Score	Team	Div
1	Joseph	76	Red Racers, Washington	AAA
2	Mitchel	82	Blue Bunnies, Richmond	AAA
3	Sue Ellen	74	Green Gazelles, Atlanta	AA

Dynamic Data Exchange (DDE)

- Dynamic Data Exchange (DDE) is a method of dynamically exchanging information between Windows applications. DDE uses a client/server relationship to enable a client application to request information from a server application. In Version 8, the SAS System is always the client. In this role, the SAS System requests data from server applications, sends data to server applications, or sends commands to server applications.
- You can use DDE with the DATA step, the SAS macro facility, SAS/AF applications, or any other portion of the SAS System that requests and generates data. DDE has many potential uses, one of which is to acquire data from a Windows spreadsheet or database application.

Dynamic Data Exchange (DDE)

- NOTAB is used only in the context of Dynamic Data Exchange (DDE). This option enables you to use nontab character delimiters between variables.

DDE Example

Statements for Outputting Data

- file: Specifies the current output file for PUT statements
- put: Writes lines to the SAS log, to the SAS procedure output file, or to an external file that is specified in the most recent FILE statement.

Example:

```
data _null_;
set new;
file 'c:\out.csv' delimiter=',' dsd;
put seqno no ;
run;
```

Comparisons

- The INFILE statement specifies the **input file** for any INPUT statements in the DATA step. The FILE statement specifies the **output file** for any PUT statements in the DATA step.
- Both the FILE and INFILE statements allow you to use options that provide SAS with additional information about the external file being used.
- An INFILE statement usually identifies data from an external file. A DATALINES statement indicates that data follow in the job stream. You can use the INFILE statement with the file specification DATALINES to take advantage of certain data-reading options that effect how the INPUT statement reads in-stream data.

Read Dates with Formatted Input

```
DATA Dates;
INPUT @1   A date11.
      @13 B ddmmyy6.
      @20 C mmddyy10.
      @31 D yymmdd8.;

FORMAT A B C D mmddyy10.; cards;
13/APR/1999 130499 04-13-1999 99 04 13
01/JAN/1960 010160 01-01-1960 60 01 01;
RUN;
```

Obs	A	B	C	D	duration
1	04/13/1999	04/13/1999	04/13/1999	04/13/1999	10694
2	01/01/1960	01/01/1960	01/01/1960	01/01/1960	-3653

Procedures To Import/Outport Data

- IMPORT: reads data from an external data source and writes it to a SAS data set.
- CPORT: writes SAS data sets, SAS catalogs, or SAS data libraries to sequential file formats (transport files).
- CIMPORT: imports a transport file that was created (exported) by the CPORT procedure. It restores the transport file to its original form as a SAS catalog, SAS data set, or SAS data library.

PROC IMPORT

- Syntax:

```
PROC IMPORT
 DATAFILE="filename" | TABLE="tablename"
 OUT=SAS-data-set
 <DBMS=identifier><REPLACE>;
```

PORC IMPORT

Space.txt:

MAKE MPG WEIGHT PRICE
AMC 22 2930 4099
AMC 17 3350 4749
AMC 22 2640 3799
Buick 20 3250 4816
Buick 15 4080 7827

```
proc import datafile="space.txt" out=mydata
   dbms=dlm replace;
getnames=yes;
datarow=4;
run;
```

Common DBMS Specifications

Identifier	Input Data Source	Extension
ACCESS	Microsoft Access Database	.MDB
DBF	dBASE file	.DBF
EXCEL	EXCEL file	.XLS
DLM	delimited file (default delimiter is a blank)	.*
CSV	comma-separated file	.CSV
TAB	tab-delimited file	.TXT

SAS Programming Efficiency

- **CPU time**
- **I/O time**
- **Memory**
- **Data storage**
- **Programming time**

Use ELSE statement to reduce CPU time

```
IF agegrp=3 THEN DO;...END;
ELSE IF agegrp=2 THEN DO;...END;
ELSE IF agegrp=1 THEN DO;...END;
```

Subset a SAS Dataset

```
DATA div1; SET adults;
    IF division=1; RUN;
DATA div2; SET adults;
    IF division=2; RUN;

DATA div1 div2;
SET adults;
        IF division=1 THEN OUTPUT div1;
ELSE IF division=2 THEN OUTPUT div2;
```

MODIFY is Better Than SET

```
DATA salary;
SET salary;
  wages=wagesy*0.1;

DATA salary;
MODIFY salary;
 wages=wages*0.1;
```

Save Space by DROP or KEEP

```
DATA new;
SET  old (KEEP=a b c);
RUN;

DATA new;
SET old (DROP=a);
RUN;
```

Save Space by Deleting Data Sets

```
DATA three;
MERGE one two;
BY type;
RUN;

PROC DATASETS;
DELETE one two;
RUN;
```

Save Space by Compress

```
DATA new (COMPRESS=YES);
SET  old;

PROC SORT DATA=a OUT=b (COMPRESS=YES);

PROC SUMMARY;
VAR score;
OUTPUT OUT=SUM1 (COMPRESS=YES) SUM=;
```

Read Only What You Need

```
DATA large:
   INFILE myDATA;
   INPUT @15 type $2. @ ;
   INPUT @1 X $1. @2 Y $5. ;
```

```
DATA large:
INFILE myDATA;
INPUT @15 type $2. @ ;
IF type in ('10','11','12') THEN
   INPUT @1 X $1. @2 Y $5.;
```

PROC FORMAT Is Better Than IF-THEN

```
DATA new;
SET  old;
        IF   0 LE age LE 10 THEN agegroup=0;
ELSE IF 10 LE age LE 20 THEN agegroup=10;
ELSE IF 20 LE age LE 30 THEN agegroup=20;
ELSE IF 30 LE age LE 40 THEN agegroup=30;
RUN;

PROC FORMAT;
VALUE age  0-09=0 10-19=10 20-29=20 30-39=30;
RUN;

DATA new;
SET  old;
agegroup=PUT(age,age.);
RUN;
```

Shorten Expressions with Functions

```
array c{10} cost1-cost10;
tot=0;
do I=1 to 10;
if c{i} ne . then do;
tot+c{i};
end;
end;
```

```
tot=sum(of cost1-cost10);
```

IF-THEN Better Than AND

IF status1=1 and status2=9 THEN OUTPUT;

IF status1=1 THEN
IF status2=9 THEN OUTPUT;

Use SAS Functions Whenever Possible

```
DATA new; SET  old;
  meanxyz = (x+y+z)/3;
RUN;
```

```
DATA new; SET   old;
  meanxyz = mean(x, y, z);
RUN;
```

Use RETAIN to Initialize Constants

```
DATA new; SET old;
a = 5; b = 13;
(programming statements); RUN;
```

```
DATA new; SET old;
retain a 5 b 13;
(programming statements);
RUN;
```

Efficient Sort

```
PROC SORT;
BY vara varb varc vard vare;
RUN;

DATA new; SET old;
sortvar=vara||varb||varc||vard||vare;
RUN;
PROC SORT;
BY sortvar;
RUN;
```

Use Arrays and Macros

Using arrays and macros can save you the time of having to repeatedly type groups of statements.

- Example: Convert Missing Values to 0

```
data one; input chr $ a b c; cards;
    x  2   .   9
     y   .   3   .
     z  8   .   . ;
data two; set one; drop i;
  array x(*) _numeric_;
    do i= 1 to dim(x);
  if x(i) = . then x(i)=0;
end; run;
```

When w has many missing values.

```
DATA new;
SET old;
wyzsum = 26 + y + z + w;
RUN;
```

```
DATA new;
SET old;
  IF x > . THEN wyzsum = 26 + y + z + w;
RUN;
```

Put Loops With the Fewest Iterations Outermost

```
DATA new;
SET old;
  DO i = 1 TO 100;
    DO j = 1 TO 10;
(programming
    statements);
END;
END;
RUN;
```

```
DATA new;
SET old;
     DO i = 1 TO 10;
      DO j = 1 TO 100;
(programming
    statements);
END;
END;
RUN;
```

IN Better Than OR

```
IF status=1 OR status=5 THEN
   newstat="single";
ELSE newstat="not single";

IF status IN (1,5) THEN newstat="single";
ELSE newstat="not single";
```

SAS Macro

What can we do with Macro?

- **Avoid repetitious SAS code**
- **Create generalizable and flexible SAS code**
- **Pass information from one part of a SAS job to another**
- **Conditionally execute data steps and PROCs**
- **Dynamically create code at execution time**

SAS Macro Facility

- **SAS macro variable**
- **SAS Macro**
- **Autocall Macro Facility**
- **Stored Compiled Macro Facility**

SAS Macro Delimiters

Two delimiters will trigger the macro processor in a SAS program.

- **&macro-name**

  This refers to a macro variable. The current value of the variable will replace &macro-name;

- **%macro-name**

  This refers to a macro, which consists of one or more complete SAS statements, or even whole data or proc steps.

SAS Macro Variables

- **SAS Macro variables can be defined and used anywhere in a SAS program, except in data lines. They are independent of a SAS dataset.**
- **Macro variables contain a single character value that remains constant until it is explicitly changed.**

SAS Macro Variables

%LET: assign text to a macro variable;

%LET macrovar = value

1. **Macrovar is the name of a global macro variable;**
2. **Value is macro variable value, which is a character string without quotation or macro expression.**

%PUT:

%put _all_, %put _user_

&macrovar:

SAS Macro Variables

- **SAS-supplied Macro Variables:**

  %put &SYSDAY; Tuesday

  %put &SYSDATE; 30SEP03

  %put &SYSTIME; 11:02

  %put &SYSVER; 8.2

- **%put _all_ shows SAS-supplied automatic and user-defined macro variables.**

SAS Macro Variables

Combine Macro Variables with Text

```
%LET first = John;
%LET last = Smith;
%put &first.&last; (combine)
%put &first. &last; (blank separate)
%put Mr. &first. &last; (prefix)
%put &first. &last. HSPH; (suffix)
```

output:

```
JohnSmith
John Smith
Mr. John Smith
John Smith HSPH
```

Create SAS Macro

- **Definition:**

  %MACRO macro-name (parm1, parm2,…parmk);

  Macro definition (&parm1,&parm2,…&parmk)

  %MEND macro-name;

- Application:

  %macro-name(values of parm1, parm2,…,parmk);

SAS Macro Example

Import Excel to SAS Datasets by a Macro

```
%macro excelsas(in=, out=);
proc import out=work.&out
        datafile="c:\&in"
       dbms=excel2000 replace;
      getnames=yes; run;
%mend excelsas;

% excelsas(class1, score1)
% excelsas(class2, score2)
```

SAS System Options

- System options are global instructions that affect the entire SAS session and control the way SAS performs operations. SAS system options differ from SAS data set options and statement options in that once you invoke a system option, it remains in effect for all subsequent **data** and **proc** steps in a SAS job, unless you specify them.
- In order to view which options are available and in effect for your SAS session, use **proc options**.

  PROC OPTIONS; RUN;

SAS system options

- NOCAPS Translate quoted strings and titles to upper case?
- CENTER Center SAS output?
- DATE Date printed in title?
- ERRORS=20 Maximum number of observations with error messages
- FIRSTOBS=1 First observation of each data set to be processed
- FMTERR Treat missing format or informat as an error?
- LABEL Allow procedures to use variable labels?
- LINESIZE=96 Line size for printed output
- MISSING=. Character printed to represent numeric missing values
- NUMBER Print page number on each page of SAS output?
- OBS=MAX Number of last observation to be processed
- PAGENO=1 Resets the current page number on the print file
- PAGESIZE=54 Number of lines printed per page of output
- YEARCUTOFF=1900 Cutoff year for DATE7. informat

Log, output and procedure options

- **center** controls whether SAS procedure output is centered. By default, output is centered. To specify not centered, use **nocenter**.
- **date** prints the date and time to the log and output window. By default, the date and time is printed. To suppress the printing of the date, use **nodate**.
- **label** allows SAS procedures to use labels with variables. By default, labels are permitted. To suppress the printing of labels, use **nolabel**.
- **notes** controls whether notes are printed to the SAS log. By default, notes are printed. To suppress the printing of notes, use **nonotes**.
- **number** controls whether page numbers are printed. By default, page numbers are printed. To suppress the printing of page numbers, use **nonumber**.
- **linesize=** specifies the line size (printer line width) for the SAS log and the SAS procedure output file used by the **data** step and procedures.
- **pagesize=** specifies # of lines that can be printed per page of SAS output.
- **missing=** specifies the character to be printed for missing numeric values.
- **formchar=** specifies the the list of graphics characters that define table boundaries.

Example:

OPTIONS NOCENTER NODATE NONOTES LINESIZE=80 MISSING=. ;

SAS data set control options

SAS data set control options specify how SAS data sets are input, processed, and output.

- **firstobs=** causes SAS to begin reading at a specified observation in a data set. The default is **firstobs=1**.
- **obs=** specifies the last observation from a data set or the last record from a raw data file that SAS is to read. To return to using all observations in a data set use **obs=all**
- **replace** specifies whether permanently stored SAS data sets are to be replaced. By default, the SAS system will over-write existing SAS data sets if the SAS data set is re-specified in a **data** step. To suppress this option, use **noreplace**.

Example:

- OPTIONS OBS=100 NOREPLACE;

Error handling options

Error handling options specify how the SAS System reports on and recovers from error conditions.

- **errors=** controls the maximum number of observations for which complete error messages are printed. The default maximum number of complete error messages is **errors=20**
- **fmterr** controls whether the SAS System generates an error message when the system cannot find a format to associate with a variable. SAS will generate an ERROR message for every unknown format it encounters and will terminate the SAS job without running any following **data** and **proc** steps. To read a SAS system data set without requiring a SAS format library, use **nofmterr.**

Example:

```
OPTIONS ERRORS=100 NOFMTERR;
```

Using where statement

where statement allows us to run procedures on a subset records.

Examples:

```
PROC PRINT DATA=auto;
WHERE (rep78 >= 3);
VAR make rep78;
RUN;

PROC PRINT DATA=auto;
WHERE (rep78 <= 2) and (rep78 ^= .) ;
VAR make price rep78 ;
RUN;
```

Missing Values

As a general rule, SAS procedures that perform computations handle missing data by omitting the missing values.

Summary of how missing values are handled in SAS procedures

- **proc means**
  For each variable, the number of non-missing values are used
- **proc freq**
  By default, missing values are excluded and percentages are based on the number of non-missing values. If you use the **missing** option on the **tables** statement, the percentages are based on the total number of observations (non-missing and missing) and the percentage of missing values are reported in the table.

Summary of how missing values are handled in SAS procedures

- **proc corr**
  By default, correlations are computed based on the number of pairs with non-missing data (**pairwise deletion of missing data)**. The **nomiss** option can be used to request that correlations be computed only for observations that have non-missing data for all variables on the **var** statement (**listwise deletion of missing data**).
- **proc reg**
  If any of the variables on the **model** or **var** statement are missing, they are excluded from the analysis (i.e., **listwise deletion of missing data)**

Summary of how missing values are handled in SAS procedures

- **proc glm**
  If you have an analysis with just one variable on the left side of the model statement (just one outcome or dependent variable), observations are eliminated if any of the variables on the model statement are missing. Likewise, if you are performing a **repeated measures ANOVA** or a **MANOVA**, then observations are eliminated if any of the variables in the model statement are missing. For other situations, see the SAS/STAT manual about **proc glm**.

Missing values in assignment statements

- As a general rule, computations involving missing values yield missing values.

  2 + 2 yields 4
  2 + . yields .

- **mean**(of var1-varn):average the data for the non-missing values in a list of variables.

  avg = mean(of var1-var10)

  N(of var1-varn): determine the number of non-missing values in a list of variables

  n = N(var1, var2, var3)

Missing values in logical statements

- SAS treats a missing value as the smallest possible value (e.g., negative infinity) in logical statements.

```
DATA times6;
SET times ;
if (var1 <= 1.5) then varc1 = 0; else varc1 = 1 ;
RUN ;
```

Output:

```
Obs id var1 varc1
1   1   1.5  0
2   2     .  0
3   3   2.1  1
```

Subsetting Data

Subsetting variables using **keep** or **drop** statements

Example:

```
DATA auto2;
SET auto;
KEEP make mpg price;
RUN;

DATA auto3;
SET auto;
DROP rep78 hdroom trunk weight length turn displ gratio foreign;
RUN;
```

Subsetting Data

Subsetting observations using **if** statements

Example:

```
 DATA auto4;
SET auto;
IF rep78 ^= . ;
RUN;

 DATA auto5;
SET auto;
IF rep78 > 3 THEN DELETE ;
RUN;
```

Labeling variables

Variable label: Use the **label** statement in the data step to assign labels to the variables. You could also assign labels to variables in proc steps, but then the labels only exist for that step. When labels are assigned in the data step they are available for all procedures that use that data set.

Example:

```
DATA auto2;
SET auto;
LABEL rep78 ="1978 Repair Record" mpg ="Miles Per Gallon" foreign="Where Car
 Was Made";
RUN;
PROC CONTENTS DATA=auto2;
RUN;
```

Labeling variable values

Labeling values is a two step process. First, you must create the label formats with **proc format** using a **value** statement. Next, you attach the label format to the variable with a **format** statement. This **format** statement can be used in either **proc** or **data** steps.

Example:

```
*first create the label formats forgnf and makef;
PROC FORMAT;
VALUE forgnf 0="domestic" 1="foreign" ;
VALUE $makef "AMC" ="American Motors" "Buick" ="Buick (GM)" "Cad." ="Cadallac (GM)"
    "Chev." ="Cheverolet (GM)" "Datsun" ="Datsun (Nissan)";
RUN;
*now we link them to the variables foreign and make;
PROC FREQ DATA=auto2;
FORMAT foreign forgnf. make $makef.;
TABLES foreign make; RUN;
```

Sort data

Use **proc sort** to sort this data file.

Examples:

```
PROC SORT DATA=auto ; BY foreign ; RUN ;

PROC SORT DATA=auto OUT=auto2 ;
BY foreign ; RUN ;

PROC SORT DATA=auto OUT=auto3;
BY descending foreign ; RUN ;

PROC SORT DATA=auto OUT=auto2 noduplicates;
BY foreign ; RUN ;
```

Making and using permanent SAS data files

- Use a **libname** statement.

```
libname diss 'c:\dissertation\';
data diss.salary;
input sal1996-sal2000 ;
cards;
14000 16500 18000 22000 29000
;
run;
```

- specify the name of the data file by directly specifying the path name of the file

```
data 'c:\dissertation\salarylong';
input Salary1996-Salary2000 ;
cards;
14000 16500 18000 22000 29000
;
run;
```

Merge data files

One-to-one merge: there are three steps to match merge two data files dads and faminc on the same variable famid.

1. Use **proc sort** to sort **dads** on **famid** and save that file (we will call it **dads2**)

   PROC SORT DATA=dads OUT=dads2; BY famid; RUN;

- Use **proc sort** to sort **faminc** on **famid** and save that file (we will call it **faminc2**)

  PROC SORT DATA=faminc OUT=faminc2; BY famid; RUN;

- merge the **dads2** and **faminc2** files based on **famid**

  DATA dadfam ; MERGE dads2 faminc2; BY famid; RUN:

Merge data files

One-to-many merge: there are three steps to match merge two data files dads and kids on the same variable famid.

1. Use **proc sort** to sort **dads** on **famid** and save that file (we will call it **dads2**)

   PROC SORT DATA=dads OUT=dads2; BY famid; RUN;

- Use **proc sort** to sort **kids** on **famid** and save that file (we will call it **kid2**)

  PROC SORT DATA=kids OUT=kids2; BY famid; RUN;

- merge the **dads2** and **faminc2** files based on **famid**

  DATA dadkid; MERGE dads2 kids2; BY famid; RUN:

Merge data files: mismatch

- Mismatching records in one-to-one merge: use the **in** option to create a 0/1 variable

```
DATA merge121;
MERGE dads(IN=fromdadx) faminc(IN=fromfamx);
BY famid;
fromdad = fromdadx;
fromfam = fromfamx;
RUN;
```

- Variables with the same name, but different information: rename variables

```
DATA merge121;
MERGE faminc(RENAME=(inc96=faminc96 inc97=faminc97 inc98=faminc98))
    dads(RENAME=(inc98=dadinc98));
BY famid;
RUN;
```

Concatenating data files in SAS

- Use **set** to stack data files

```
DATA dadmom;   SET dads moms; RUN;
```

```
DATA momdad;
SET dads(RENAME=(dadinc=inc)) moms(RENAME=(mominc=inc));
RUN;
```

- Two data files with different lengths for variables of the same name

```
DATA momdad;
LENGTH name $ 4;
SET dads moms;
RUN;
```

Concatenating data files in SAS

```
DATA dads; SET dads; full=fulltime; DROP fulltime;RUN;

DATA moms; SET moms;
IF fulltime="Y" THEN full=1; IF fulltime="N" THEN full=0;
DROP fulltime;RUN;

DATA momdad; SET dads moms;RUN;
```

SAS Macro

What can we do with Macro?

- **Avoid repetitious SAS code**
- **Create generalizable and flexible SAS code**
- **Pass information from one part of a SAS job to another**
- **Conditionally execute data steps and PROCs**
- **Dynamically create code at execution time**

SAS Macro Facility

- **SAS macro variable**
- **SAS Macro**
- **Autocall Macro Facility**
- **Stored Compiled Macro Facility**

SAS Macro Delimiters

Two delimiters will trigger the macro processor in a SAS program.

- **&macro-name**

  This refers to a macro variable. The current value of the variable will replace &macro-name;

- **%macro-name**

  This refers to a macro, which consists of one or more complete SAS statements, or even whole data or proc steps.

SAS Macro Variables

- **SAS Macro variables can be defined and used anywhere in a SAS program, except in data lines. They are independent of a SAS dataset.**
- **Macro variables contain a single character value that remains constant until it is explicitly changed.**
- **To record the SAS macro use**
- **options macro;**

SAS Macro Variables

%LET: assign text to a macro variable;

%LET macrovar = value

1. **Macrovar is the name of a global macro variable;**
2. **Value is macro variable value, which is a character string without quotation or macro expression.**

%PUT:

%put _all_, %put _user_

&macrovar:

SAS Macro Variables

- **SAS-supplied Macro Variables:**

  %put &SYSDAY; Tuesday

  %put &SYSDATE; 30SEP03

  %put &SYSTIME; 11:02

  %put &SYSVER; 8.2

- **%put _all_ shows SAS-supplied automatic and user-defined macro variables.**

SAS Macro Variables

Combine Macro Variables with Text

```
%LET first = John;
%LET last = Smith;
%put &first.&last; (combine)
%put &first. &last; (blank separate)
%put Mr. &first. &last; (prefix)
%put &first. &last. HSPH; (suffix)
```

output:

```
JohnSmith
John Smith
Mr. John Smith
John Smith HSPH
```

Create SAS Macro

- **Definition:**

  %MACRO macro-name (parm1, parm2,…parmk);

  Macro definition (&parm1,&parm2,…&parmk)

  %MEND macro-name;

- Application:

  %macro-name(values of parm1, parm2,…,parmk);

SAS Macro Example

Import Excel to SAS Datasets by a Macro

```
%macro excelsas(in,out);
proc import out=work.&out
        datafile="c:\&in"
       dbms=excel2000 replace;
      getnames=yes; run;
%mend excelsas;

% excelsas(class1, score1)
% excelsas(class2, score2)
```

SAS Macro Example

Use proc means by a Macro

```
%macro auto(var1, var2);
proc sort data=auto;
by &var2;
run;

proc means data=auto;
var  &var1;
by &var2;
run;
%mend auto;

%auto(price, rep78) ;
%auto(price, foreign);
```

Inclass practice

Use the auto data to do the following

- check missing values for each variable
- create a new variable model (first part of make)
- get means/frequencies for each variable by model
- create 5 data files with 1-5 repairs using macro