# STAT 475: DATA ANALYSIS WITH SAS

CLASS NOTES

SUNG E. KIM
DEPT OF MATH AND STATISTICS
CALIFORNIA STATE UNIVERSITY-LONG BEACH

<sup>\*</sup> THE NOTES ARE WRITTEN ONLY FOR THE PURPOSE OF TEACHING THE CLASS AND SHOULD NOT BE REFERRED TO OTHER PURPOSE

## NOTE #1: DATA STEP: BASIC

```
Syntax Basic:
```

```
DATA yourdataname;
< statements >
RUN;
```

#### Note:

- a. SAS is not case sensitive
- b. A data name must be 32 characters or less in length and must start with a letter or .
- c. All statements must end with a semicolon (;).

Getting Data into SAS: There are basically three ways of getting data into SAS;

- a. Typing data directly in the editor widow in DATA step (see Example 1). You can also enter data using the Viewtable window (Tools/Table Editor).
- b. Call in an external ASCII data file (text file) using *infile* statement in DATA step. Use PROC IMPORT for data format other than ASCII. (e.g. Excel, Lotus, etc.)
- c. Using the Import Wizard under File/Import Data.

```
/*** Example1 (Typing data directly) ***/
/* put all the comments you want in between these brackets.
  Comments are shown in green in the Editor window */
* you can also put comments between star (*) and semicolon(;) ;
DATA mydata1;
                    *The name of data is mydatal;
                 *and has 2 variables city and F.;
INPUT city $ F;
                    *city is character and F is numeric variable;
                    *New variables can be defined after INPUT;
C=(F-32)*5/9;
                    *statement;
Format C F4.1;
                    *Format is used to specify the variable format;
label C="Celsius" F="Fahrenheit";
DATALINES;
                     *Now, entering the data;
LA 85
SD 78
Orange 81
Anaheim 88
Riverside 94
LV 102
                      *the end of data;
PROC PRINT label DATA=mydata1; *print the data with labels;
/*** Example2 (INFILE) ***
Save the six lines of data in Example1 as an ASCII file (text file).
Assuming the file is named 'E:\testdata1.dat'.
*/
     DATA mydata2;
      INFILE 'E:\testdata1.dat';
```

```
INPUT city $ F;
       C=(F-32)*5/9;
       Format C F4.1;
       label C="Celsius" F="Fahrenheit";
    RUN;
    PROC PRINT label DATA= mydata2;
    RUN;
/* You can also do */
    Filename test1 'E:\testdata1.dat';
    DATA mydata2;
     INFILE test1;
     INPUT city $ F;
       C=(F-32)*5/9;
       Format C F4.1;
       label C="Celsius" F="Fahrenheit";
    RUN;
    PROC PRINT label DATA= mydata2;
/*** Example3 (PROC IMPORT) ***
 Assuming the data above is saved as an excel file named E:\testdata1.xls */
 PROC IMPORT OUT=mydata3 REPLACE
    DATAFILE='E:\testdata1.xls';
    SHEET="sheet1";
    GETNAMES=NO;
 RUN;
```

## Making a DATA file from SAS data

You can generate an output data or report using FILE and PUT statement.

```
/*** Example4 ***/
    DATA mydata3; set mydata2;
    Today=Today();
    FILE 'E:\outdata1.dat' PRINT;
    PUT @1 "Daily Maximum Temperature Report " Today MMDDYY8. //
        @2 "CITY" @15 "Celsius" @20 "Fahrenheit"/
        @2 city @20 F @25 C;
    RUN;

/* A better one. Header is used */

    DATA mydata2; set mydata1; today=today();
    FILE 'E:\outdata1.dat' PRINT notitle HEADER=head;
    PUT @2 city @20 C @32 F; return;
    head:
    PUT @1 "Daily Maximum Temperature Report " Today MMDDYY8. //
        @2 "CITY" @15 "Celsius" @30 "Fahrenheit"/
```

```
@1 40*'=';
return;
RUN;
```

## Working with Library

SAS Data Name consists of two levels: libref.dataname. For example, mylib.mydata1; mylib is the libref and mydata1 is the data name in the folder where you defined the library mylib. To have your data permanent you have to make a library in a specified folder and use the two-level data name in a DATA step. Note that libref name can not exceed 8 characters in length. You can name the data without the library reference (libref). Then SAS will use the default libref name WORK. This library is temporary means that once you close a SAS session the data will disappear. There are two ways of making a library.

1. Before a DATA step, type

```
LIBNAME mylib 'E:\SASCLASS';
```

Now, the library mylib is generated. Use the two-level data name in the DATA step.

```
DATA mylib.mydata1;
     < statements>
RUN;
```

2. You can also define a library using the New Library Window in SAS Explorer (right click in the Active Library window then New). Once it is done, use the two-level name in the DATA step.

NOTE: Without making a library, you can also make a permanent SAS data by direct referencing in DATA step. For example,

```
/******* In-class exercise *************************/
/*
NAME DOB HT WT SEX WAGE EDU
Susie Frieman 07/11/81 42 41 F 34000 HS
Charlie Smith 10/26/54 65 160 M 125000 C
Calvin Lee 01/10/81 69 140 M 87525 C
Lucy Chen
                 11/23/79 58 97 F 48900
                 07/30/65 72 182 M 38500 HS
Bob Lead
                 01/07/68 58 110 F 79000 HS
Sue Halfcott
                 03/06/48 60 125 M 154500 C
Hans Richman
   1. Copy and paste the data above in notepad and save as prac1.dat in the
       directory E:\SASCLASS (E is a CDR drive. Make sure your drive name. You
       should make the directory SASCLASS in your CD first.)
   2. copy and paste the following code to a new SAS editor window (File->
       new program). */
   OPTIONS LINESIZE=75 PAGESIZE=54 NODATE PAGENO=1;
   DM "output; clear; log; clear"; *this will clear output and log window;
   LIBNAME SASPRAC 'E:\SASCLASS';
   DATA SASPRAC.ONE;
    INFILE 'E:\SASCLASS\prac1.dat' FIRSTOBS=2 MISSOVER;
    LENGTH NAME $ 16;
    INPUT @1 NAME $16. @17 DOB MMDDYY8. Height Weight @33 Gender $ @36 Salary @43 Edu $;
               *Make sure the column numbers and modify the numbers accordingly;
    LABEL DOB='DATE OF BIRTH';
    LABEL Edu='Education';
   PROC PRINT; *if DATA= is missing, SAS uses the most recent data;
    RUN;
   PROC FORMAT;
    VALUE $edulevel 'HS'= 'High School or below'
                   'C' = 'College or above';
                   'F' = 'Female'
    VALUE $Sex
                   'M' = 'Male' ;
    VALUE incomelevel 0 -< 50000 = 'Low'
                     50000 -< 100000 = 'Mid'
                     100000 - HIGH = 'High';
   PROC Print label data=SASPRAC.ONE;
    FORMAT DOB WORDDATE18. Gender $Sex. Salary DOLLAR11.2 Salary incomelevel. Edu $edulevel.;
     TITLE 'SAS Data Step Class Practice';
   PROC Contents data=SASPRAC.ONE; run;
   DATA NULL; SET SASPRAC.ONE;
    Newsalary = salary * 1.05;
    Newmonth = Newsalary/12;
    FILE 'C:\teaching\SAS\pract1 out.dat' PRINT NOTITLE;
    PUT @3 "Salary raise notice for " Name /
        @3 55*'=' /
        05 "Your Annual Salary for next year will be" Newsalary Dollar11.2 / 05 "Your new monthly wage of " Newmonth Dollar11.2 /
        @5 "will appear on your March paycheck." /
        @3 55*'='
        //;
    PUT _PAGE_;
```

/\* 3. Run the code. Detail discussion will be given during class\*/ Some options in INFILE statement

```
INFILE 'E:\foldername\dataname.dat' <options>;
INPUT var1 var2 ...;
```

**MISSOVER:** To tell SAS there are missing data at end of the data line (see the inclass-prac). Without this option SAS will go to the next line and keep reading the variable's value. That means SAS will not read the missing data as missing.

TRUNCOVER: Similar to MISSOVER but can be used with column or formatted input.

- **DLM** = 'delimiter': In case data values are separated by a delimiter other than spaces. For example, DLM = ',' tells that data values are separated by comma.
- **DSD**: DSD can be used if character values are placed in quotes (example, "*Lastname*, *firstname*"). Many cases data come with this format (for example, CVS data from Excel). DSD option treats two consecutive commas as a missing value. (see example below)
- **FIRSTOBS** = n: Some data come with header and/or column names. You can have SAS to start reading the data at nth line.
- **OBS** = n: In case you want to read only a part of data. With this option, SAS will read n data lines. This option can be used with FIRSTOBS option.

```
/*** Example5 (Delimited Data) ***/
/* Assume this data saved as 'E:\SASCLASS\example5.dat'
MS PROGRAM CANDIDATE RECORD
DATE 12/23/08
STATISTICS PROGRAM
"Cathy, Wood", Senior, Comp SCI, 3.48, 1270
"Mike, Michaski", Junior, SCI, 2.96,
"Henry, Wood", Senior, COMP ENG, 3.04, 1380
"Kim, Andy", Senior, Math, 3.25, 1250
"Mueller, Kay", Junior, APP STAT, 3.65, 1320
*/
LIBNAME SASPRAC 'E:\SASCLASS';
DATA SASPRAC.student;
INFILE 'E:\SASCLASS\example5.dat' FIRSTOBS=4 DLM=',' DSD MISSOVER;
LENGTH name $ 16 Major $ 10;
INPUT Name $ Grade $ Major $ GPA GRE;
RUN;
PROC PRINT DATA = SASPRAC.student; RUN;
/* Run the code without DSD option and see what happens */
/***********************
/* Data step above can be written as below *********/
DATA SASPRAC.student;
INFILE 'E:\SASCLASS \example5.dat' FIRSTOBS=4 DLM=',' DSD MISSOVER;
INPUT Name: $16. Grade $ Major $ GPA GRE; RUN;
```

**NOTE:** INFILE options can be used with DATALINES statement. For example, you can still use DLM option with DATALINES. See example below.

```
/*** Example5 (revisit) ***/
DATA SASPRAC.student;
INFILE datalines FIRSTOBS=4 DLM=',' DSD MISSOVER;
LENGTH name $ 16 Major $10;
INPUT Name $ Grade $ Major $ GPA GRE;
DATALINES;
MS PROGRAM CANDIDATE RECORD
DATE 12/23/08
STATISTICS PROGRAM
"Cathy, Wood", Senior, Comp SCI, 3.48, 1270
"Mike, Michaski", Junior, SCI, 2.96,
"Henry, Wood", Senior, COMP ENG, 3.04, 1380
"Kim, Andy", Senior, Math, 3.25, 1250
"Mueller, Kay", Junior, APP STAT, 3.65, 1320
RUN;
PROC PRINT DATA= SASPRAC.student; RUN;
```

## **SAS FUNCTIONS** (Excerpted from SAS Online Documentation)

#### Arithmetic Functions

returns absolute value ABS (argument)

DIM<n>(array-name) returns the number of elements in a one-

> dimensional array or the number of elements in a specified dimension of a multidimensional array. n specifies the dimension, in a multidimensional array, for which you want to know the the number

of elements.

DIM(array-name, bound-n) returns the number of elements in a one-

> dimensional array or the number of elements in the specified dimension of a multidimensional array

bound-n specifies the dimension in a

multidimensional array, for which you want to know

the number of elements.

HBOUND<n>(array-name) returns the upper bound of an array HBOUND(array-name, bound-n) returns the upper bound of an array LBOUND<n>(array-name) returns the lower bound of an array LBOUND(array-name, bound-n) returns the lower bound of an array

MAX(argument, argument, ...) returns the largest value of the numeric arguments

MIN(argument, argument, ...) returns the smallest value of the numeric

arguments

MOD(argument-1, argument-2) returns the remainder

SIGN(argument) returns the sign of a value or 0

SQRT (argument) returns the square root

## Character Functions

returns one character in the ASCII or EBCDIC BYTE(n)

collating sequence where nis an integer

representing a specific ASCII or EBCDIC character

COLLATE(start-position<, end-position>) | (start-position<,, length>)

returns an ASCII or EBCDIC collating sequence

character string

COMPBL (source) removes multiple blanks between words in a

character string

COMPRESS(source<, characters-to-remove>) removes specific characters from

a character string

DEQUOTE (argument) removes quotation marks from a character value

searches the source for the character string INDEX(source, excerpt)

specified by the excerpt

INDEXC(source, excerpt-1<, ... excerpt-n>) searches the source for any

character present in the excerpt

INDEXW(source, excerpt) searches the source for a specified pattern as a

word

left-aligns a SAS character string LEFT (argument) LENGTH(argument) returns the length of an argument

converts all letters in an argument to lowercase LOWCASE (argument) QUOTE(argument) adds double quotation marks to a character value RANK(x) returns the position of a character in the ASCII

or EBCDIC collating sequence

REPEAT (argument, n) repeats a character expression REVERSE (argument) reverses a character expression right-aligns a character expression SCAN(argument, n<, delimiters>) returns a given word from a character

expression

SOUNDEX (argument) encodes a string to facilitate searching SUBSTR(argument, position<, n>) = characters-to-replace replaces character value contents var=SUBSTR(argument, position<, n>) extracts a substring from an argument. (var is any valid SAS variable name.) TRANSLATE(source, to-1, from-1<,...to-n, from-n>) replaces specific characters in a character expression TRANWRD (source, target, replacement) replaces or removes all occurrences of a word in a character string removes trailing blanks from character expression TRIM(argument) and returns one blank if the expression is missing TRIMN(argument) removes trailing blanks from character expressions and returns a null string if the expression is missing UPCASE(argument) converts all letters in an argument to uppercase VERIFY(source, excerpt-1<,...excerpt-n) returns the position of the first

character unique to an expression

#### Date and Time Functions

DATDIF(sdate, edate, basis) returns the number of days between two dates returns the current date as a SAS date value DATE() DATEJUL(julian-date) converts a Julian date to a SAS date value DATEPART(datetime) extracts the date from a SAS datetime value returns the current date and time of day DATETIME() DAY (date) returns the day of the month from a SAS date value DHMS (date, hour, minute, second) returns a SAS datetime value from date, hour, minute, and second HMS (hour, minute, second) returns a SAS time value from hour, minute, and second HOUR(<time | datetime>) returns the hour from a SAS time or datetime value INTCK('interval', from, to) returns the number of time intervals in a given time span INTNX('interval',start-from,increment<,'alignment'>) advances a date, time, or datetime value by a given interval, and returns a date, time, or datetime value JULDATE (date) returns the Julian date from a SAS date value returns a SAS date value from month, day, and year MDY (month, day, year) values MINUTE(time | datetime) returns the minute from a SAS time or datetime value MONTH (date) returns the month from a SAS date value returns the quarter of the year from a SAS date QTR(date) value SECOND(time | datetime) returns the second from a SAS time or datetime value returns the current time of day TIME() extracts a time value from a SAS datetime value returns the current date as a SAS date value TIMEPART (datetime) TODAY() returns the day of the week from a SAS date value returns the year from a SAS date value WEEKDAY (date) YEAR (date) YRDIF(sdate,edate,basis) returns the difference in years between two dates YYQ(year, quarter) returns a SAS date value from the year and quarter

## Mathematical Functions

AIRY(x) returns the value of the AIRY function
DAIRY(x) returns the derivative of the AIRY function

DIGAMMA (argument) returns the value of the DIGAMMA function returns the value of the (normal) error function ERF(argument) returns the value of the (normal) error function ERFC(argument) EXP(argument) returns the value of the exponential function GAMMA(argument) returns the value of the GAMMA function IBESSEL(nu,x,kode) returns the value of the modified bessel function returns the value of the bessel function JBESSEL (nu, x) LGAMMA (argument) returns the natural logarithm of the GAMMA function LOG(argument) returns the natural (base e) logarithm returns the logarithm to the base 2 LOG2 (argument) LOG10 (argument) returns the logarithm to the base 10 TRIGAMMA(argument) returns the value of the TRIGAMMA function

#### Noncentrality Functions

CNONCT(x,df,prob) returns the noncentrality parameter from a chi-squared distribution

FNONCT(x,ndf,ddf,prob) returns the value of the noncentrality parameter of an F distribution

TNONCT(x,df,prob) returns the value of the noncentrality parameter from the student's t distribution

## Probability and Density Functions

CDF('dist', quantile, parm-1,..., parm-k) computes cumulative distribution functions LOGPDF|LOGPMF('dist',quantile,parm-1,...,parm-k)computes the logarithm of a probability density (mass) function. The two functions are identical. LOGSDF('dist',quantile,parm-1,...,parm-k) computes the logarithm of a survival function PDF|PMF('dist',quantile,parm-1,...,parm-k)computes probability density (mass) functions POISSON (m, n) returns the probability from a POISSON distribution PROBBETA(x,a,b) returns the probability from a beta distribution PROBBNML(p,n,m) returns the probability from a binomial distribution PROBCHI(x,df<,nc>) returns the probability from a chi-squared distribution PROBF(x,ndf,ddf<,nc>) returns the probability from an F distribution PROBGAM(x,a)returns the probability from a gamma distribution PROBHYPR(N, K, n, x <, r >) returns the probability from a hypergeometric distribution probabilities and critical values (quantiles) from PROBMC. various distributions for multiple comparisons of the means of several groups. returns the probability from a negative binomial PROBNEGB (p, n, m) distribution PROBBNRM(x, y, r) standardized bivariate normal distribution PROBNORM(x) returns the probability from the standard normal distribution

#### Quantile Functions

PROBT(x,df<,nc>)

BETAINV(p,a,b)	returns a	quantile	from the	beta distribution
CINV(p,df<,nc>)	returns a	quantile	from the	chi-squared distribution
<pre>FINV(p,ndf,ddf&lt;,nc&gt;)</pre>	returns a	quantile	from the	F distribution
GAMINV(p,a)	returns a	quantile	from the	gamma distribution
PROBIT(p)	returns a	quantile	from the	standard normal distribution

SDF('dist', quantile, parm-1,..., parm-k) computes a survival function

returns the probability from a Student's t distribution

TINV(p,df<,nc>) returns a quantile from the t distribution

## Sample Statistics Functions

```
 \hbox{CSS (argument, argument, ...)} \qquad \hbox{returns the corrected sum of squares} \\ \hbox{CV (argument, argument, ...)} \qquad \hbox{returns the coefficient of variation} 
KURTOSIS (argument, argument, ...) returns the kurtosis (or 4th moment)
MAX(argument, argument, ...) returns the largest value MIN(argument, argument, ...) returns the smallest value MEAN(argument, argument, ...) returns the arithmetic mean (average)
MISSING (numeric-expression | character-expression) returns a numeric
                                           result that indicates whether the argument
                                           contains a missing value
N(argument, argument, ....)
                                         returns the number of nonmissing values
NMISS(argument, argument, ...) returns the number of missing values
ORDINAL(count, argument, argument, ...) returns the largest value of a part of a
                                           list
RANGE (argument, argument, ...) returns the range of values
{\tt SKEWNESS} \ ({\tt argument, argument, ...}) \ {\tt returns} \ {\tt the} \ {\tt skewness}
STD(argument, argument, ...) returns the standard deviation
STDERR(argument, argument, ...) returns the standard error of the mean
SUM(argument, argument,...) returns the sum
USS(argument, argument,...) returns the uncorrected sum of squares
VAR(argument, argument,...) returns the variance
```

#### State and ZIP Code Functions

```
FIPNAME (expression) converts FIPS codes to uppercase state names
FIPNAMEL(expression) converts FIPS codes to mixed case state names
FIPSTATE (expression) converts FIPS codes to two-character postal codes
STFIPS (postal-code) converts state postal codes to FIPS state codes
STNAME (postal-code) converts state postal codes to uppercase state names
STNAMEL (postal-code) converts state postal codes to mixed case state names
ZIPFIPS (zip-code) converts ZIP codes to FIPS state codes
ZIPNAME (zip-code) converts ZIP codes to uppercase state names
ZIPSTATE (zip-code) converts ZIP codes to mixed case state names
ZIPSTATE (zip-code) converts ZIP codes to state postal codes
```

# Trigonometric and Hyperbolic Functions

```
ARCOS(argument) returns the arccosine
ARSIN(argument) returns the arcsine
ATAN(argument) returns the arctangent
COS(argument) returns the cosine
COSH(argument) returns the hyperbolic cosine
SIN(argument) returns the sine
SINH(argument) returns the hyperbolic sine
TAN(argument) returns the tangent
TANH(argument) returns the hyperbolic tangent
```

#### Truncation Functions

```
CEIL(argument)

returns the smallest integer that is greater than or equal to the argument

returns the largest integer that is less than or equal to the argument

FUZZ(argument)

returns the nearest integer if the argument is within 1E-12

INT(argument)

returns the integer value

ROUND(argument, round-off-unit)

rounds to the nearest round-off unit
```

TRUNC (number, length) truncates a numeric value to a specified length

## Function used with INPUT statement

**varname \$** A dollar sign (\$) following a variable name means that the variable has character values. If character values have length of over 8, you can specify with the LENGTH statement. If you don't specify the length and values exceed 8 characters, SAS will truncate the value at the 8th character.

**varname \$k.** SAS will read the variable values up to *k* (greater than 8) characters. The length can also be specified using LENGTH statement before the INPUT statement.

@n Most basic data format is free format input that data is separated by at least one space between values. If the data is column formatted you should use the column controller @. For example, INPUT @5 score; tells SAS to move the column pointer to the 5<sup>th</sup> column and read *score*.

+**m** This moves the column point m spaces to the right.

@@ If there are multiple observations in a line you must add @@ (double trailing sign) at the end of the INPUT statement so that SAS continue to read the next observation. Without this, once SAS finishes reading an observation, the pointer will move to the next line. For example,

```
DATA prac;
INPUT Hts Wts @@;
DATALINES;
158 47 178 79 180 98 163 51;
PROC PRINT; RUN;
```

varname:

The colon (:) following the variable name tells SAS to read the next variable when the delimiter (comma this case) is encountered. Without this, SAS will read characters past the delimiter. Try the code without the colon and see. This function can be used with DLM option in INFILE statement.

**varname n-m** This tell SAS the variable values are in column n to m. For example,

INPUT name \$1-16;

#n If an observation consists of more than one line, this row pointer can be used (see example6).

**varname \$ &** This indicates the character variable with a blank. SAS will read the next variable with double or more spaces (see example7).

```
/* Example6 (Column arranged Data) */
   /* Assume this data saved as 'E:\SASCLASS\example6.dat'
                        12/14/87 F
                                         125 7th St
   Cathy Heywood
                                                          Long Beach
                        3.48 1270
   Senior Comp SCI
                        10/03/89 M
  Mike Michaski
                                         25 Harbor Dr
                                                          Anaheim
   Junior SCI
                        2.96
                       01/25/86 M
  Henry Wood
                                         16 Euclid Dr
                                                         Fullerton
   Senior COMP ENG 3.04 1380
Kim Andy 04/17/91 F
                                         568 E st
                                                          Orange
                       3.25 1250
   Senior Math
                       11/23/82 F
  Mueller Kay
                                         5847 E2nd St
                                                         Irvine
   Junior APP STAT 3.65 1320
   DATA SASPRAC.example6 (DROP = Add City DOB c Birthyear);
      INFILE 'E:\SASCLASS\example6.dat' MISSOVER TRUNCOVER;
      INPUT #1 Name $15. @17 DOB MMDDYY8. @17 DOB c $ @26 Gender $1. @28
              Add $12. @41 City $11.
            #2 Grade $ @8 Major $8. @17 GPA GRE;
      Format DOB date9.;
      Age=INT(YRDIF(DOB, TODAY(), 'Actual'));
      Address=TRIM(Add) || ', ' || City;
      Length Name g $17.;
      Name_g=LEFT(Name) || '/' || Gender;
      Birthyear=SCAN(DOB c,3); *Using SCAN, Birthyear is a character var;
      YOB=INPUT (Birthyear, f2.) +1900; *INPUT coverts a character var to numeric;
                                    *YOB is now numeric var so that you can;
                                    *do numerical calculations;
   /* You can also do YOB=INPUT(SCAN(DOB c,3))+1900
                       or YOB=INPUT(SUBSTR(DOB c,7,2))+1900 */
      LABEL Name g='Name/Gender' DOB='Date of Birth' YOB='Year of Birth';
   RUN;
   PROC PRINT LABEL DATA= SASPRAC.example6; RUN;
   DATA SASPRAC.example6_1; set SASPRAC.example6;
      if Grade = 'Senior' & Age > = 20; RUN;
   PROC PRINT DATA= SASPRAC.example6 1; RUN;
   /* Example7 (example $&)*/
   DATA SASPRAC.example6;
      INFILE 'C:\teaching\SAS\example6.dat' MISSOVER TRUNCOVER;
      Length Name $ 15 Add $ 12 City $ 11;
      INPUT #1 Name $ & @17 DOB MMDDYY8. @26 Gender $1. @28 Add $ &
            @41 City $ &
            #2 Grade $ @8 Major $8. @17 GPA GRE;
      Format DOB worddate12.; RUN;
   PROC PRINT DATA= SASPRAC.example6; RUN;
```

# NOTE #2: DATA MANIPULATIONS I

# 2.1 Subsetting Data

<u>IF</u>: You can use IF statement to subset your data. For example, for a grade roster data if you want to choose senior students data only, then after you define *Class* variable in INPUT statement use

```
IF Class = 'senior';
```

Note that the equal sign (=) can be replaced with EQ. That is, IF Class EQ 'senior';

More than one conditions can be chosen by using AND/OR/IN. For example,

```
IF Class = 'senior' and Gender = 'Female';
IF Class = 'senior' or Class = 'junior';
IF Class in ('senior', 'junior');
```

The second and the third statements will return the same result. "&" can replace "and" and "|" can replace "or". For subsetting using a numeric variable, the following mnemonic or symbolic expression can be used:

```
Equal to: EQ =
Not equal to: NE ~=
Less than: LT <
Greater than GT >
Less than or equal: LE <=
Greater than or equal: GE >=
```

For example,

```
IF 70 <= Average < 90;
IF Average GE 70 & Average LT 90;</pre>
```

<u>IF-OUTPUT:</u> You can create multiple sub-datasets using IF-THEN-OUTPUT statement. For example,

```
IF Gender = 'M' THEN OUTPUT MALES;
ELSE IF Gender = 'F' THEN OUTPUT FEMALES;
ELSE Missing(Gender) THEN OUTPUT MISSINGDATA;
```

This will generate three SAS data sets; males, females, and missing data.

WHERE: Subsetting also can be done using WHERE statement. Simply replace IF with WHERE.

```
WHERE Class = 'senior' and Gender = 'Female';
WHERE Class = 'senior' or Class = 'junior';
WHERE Class in ('senior', 'junior');
```

Note that WHERE statement can be used in SAS PROC while IF can only used in DATA step. Also note that WHERE has a larger choice of operators.

# 2.2 CONDITIONAL PROCESS

<u>IF and ELSE IF</u>: IF-THEN statement can be used to apply a statement to some part or subset of observations.

```
IF conditions THEN statements;
or
   IF conditions THEN DO;
        Statements;
END:
```

Multiple conditions can be combined with AND (&) or OR (|). To apply separate statements for more than one groups of observations ELSE IF statement can be used:

```
IF conditions THEN statements;
ELSE IF conditions THEN statements;
ELSE IF conditions THEN statements;
...
For example,

IF missing(Score) THEN Grade = .;
ELSE IF Score <= 50 THEN Grade = "D";
ELSE IF Score <= 70 THEN Grade = "C";
ELSE IF Score <= 90 THEN Grade = "B";
ELSE IF Score > 90 THEN Grade = "A";
```

IF-THEN statements can be used with INPUT statement.

```
IF conditions THEN INPUT variable list;
ELSE IF conditions THEN INPUT variable list;
```

<u>SELECT-WHEN:</u> Series of IF and ELSE IF statements can be simplified using SELECT-WHEN statement.

```
SELECT (varname);
  WHEN (conditions or values) statement;
...
  Otherwise statement;
END;
```

Don't forget END; at the end. More than one values in when statement can be separated by a comma. For example, WHEN (1, 2, 3). Above example can be replaced with

```
SELECT (Score);
WHEN (missing(Score))Grade = .;
WHEN (Score LE 50) Grade = "D";
WHEN (Score LE 70) Grade = "C";
WHEN (Score LE 90) Grade = "B";
OTHEWISE Grade = "A";
END;
```

## 2.3 LOOPING

<u>DO loop:</u> Iterative loop for sequential calculations or input. For data having each observation line represents values from a group or a variable it is useful to use Do loop with INPUT statement. For example, consider the data

```
87 74 89 70 74
69 68 77 80 54
98 96 97 84 82
```

Here, scores in each line are test scores from five students for three different subjects (for example, first line is English, second is Math, and the third is History score). Then we can write

```
DATA score;

Do Subj = 'English', 'Math', 'History';

DO Student = 1 to 5;

INPUT Score @; * @ is a pointer holder;

OUTPUT; *to write out an observation to the output data;

END; * See example2_3 for more about OUTPUT;

END;

DATALINES;

87 74 89 70 74

69 68 77 80 54

98 96 97 84 82

; RUN;
```

This example used both character values (English, Math, History) and numeric values (1,...,5) for DO loop indices. More examples will follow.

Other form of DO loop: DO WHILE, DO UNTIL

ARRAY: In case you have many variables to transform, you may simply your program using array. After you define the variables in INPUT statement you can write an array statement as follows (see example below): ARRAY arrayname (n) \$ variable list

```
/* Example2 1 */
/* DATA: 'E\SASCLASS\example2 1.dat'
STAT101 Elementary Statistic
Fall 2010 Grade Roster
C Gonzalez J 82 60 78
T Chung
             Sr 92 58 96
         Sr 32 00
F 52 69 77
I Irwin
            S 98 92 89
C Reid
            s 60 48 70
P Lee
B Washington J 47 63
K Piazza F 87 72 90
M Crafton F 100 74100
A Shulz S 47 61
S Olson
C Chen
            F 74 77 75
            S 90 73 68
            F 100 98
H Wood
LIBNAME SASPRAC 'E:\SASCLASS';
PROC FORMAT;
VALUE $class 'F'= 'Freshman' 'S'= 'Sophomore'
              'J'= 'Junior' 'Sr'='Seniour';
RUN;
DATA SASPRAC.roster;
INFILE 'E\SASCLASS\example2 1.dat' FIRSTOBS=3 MISSOVER;
 INPUT NAME $2-14 Class $15-16 Class. (Score1 Score2 Score3)(3.);
 ARRAY Test (3) Scorel-Score3;
 DO i = 1 to 3;
     If missing(Test(i)) then Test(i) =0;
 END;
DROP i;
Average = ROUND (MEAN (Score1, Score2, Score3), 1);
If Average GE 90 then Grade = "A";
  ELSE if Average >= 80 then Grade = "B";
  ELSE if Average >= 70 then Grade = "C";
  ELSE if Average >= 50 then Grade = "D";
   ELSE if Average < 50 then Grade = "F";
SELECT (Grade);
  WHEN ("A") GP = 4;
  WHEN ("B") GP = 3;
  WHEN ("C") GP = 2;
  WHEN ("D") GP = 1;
  otherwise GP = 0;
END;
LABEL Score1='Midterm I' Score2='Midterm II' Score3='Final' GP='Grade Point';
RUN;
PROC Print LABEL NOOBS U data=SASPRAC.roster;
    * NOOBS will remove Obs # and U will remove title, date, and page #;
  FORMAT Class $class.;
 TITLE "STAT101 Final Grade Roster";
RUN;
/* Example2 2 */
```

```
LIBNAME SASPRAC 'J:\SASCLASS';
DATA SASPRAC.sales (DROP = type p);
INPUT @1 type $ @;
  IF type="0" THEN DO;
     INPUT @3 Store $10.;
       Delete;
       END;
  RETAIN Store;
  ELSE IF type ne "@" THEN
     INPUT p $1-3 d 4-6 unit 7-11 @12 price Dollar7.2;
SELECT (p);
  WHEN ("011") prod = "CDR50";
  WHEN ("012") prod = "DVDR-";
  WHEN ("014") prod = "DVDDL";
  WHEN ("017") prod = "CDR100";
  WHEN ("020") prod = "USB2G";
  WHEN ("021") prod = "USB8G";
  OTHERWISE;
END;
sales=unit*price/d;
FORMAT price Dollar7.2 sales Dollar8.2;
LABEL prod="Product Name" d="Survey Duration" unit="Unit sold" price="Unit
price" sales="Daily Sales";
DATALINES;
@ Kenwood
0110300023601200
0120600065203650
0140300102504190
0170600150702160
0201200056203650
0210900023410100
@ Westside
0110300017801290
0120600025603470
0140300087204090
0170600180701960
0201200114803290
0210900040209900
@ SouthHill
0110300030700900
0120600037403750
0140300087404290
0170600099802045
0201200078403880
0210900041509990
RUN;
PROC PRINT LABEL U NOOBS DATA= SASPRAC.sales;
RUN;
/* Example2_3 (OUTPUT statement) */
DATA e2 3;
INFILE DATALINES MISSOVER;
INPUT (Name subj room) ($) @; OUTPUT;
```

Use the principal of \$5000.

## **NOTE #3: DATA MANIPULATIONS II**

## 3.1 COMBINING DATA SETS

<u>SET data1 data2 ...</u>: SET statement is used to stack several data sets; that is combining observations (rows). For example,

```
DATA one;
INPUT name $ score gender $;
DATALINES;
Smith 78 F
Chen 58 F
Rod 69 M
;
DATA two;
INPUT name $ score;
DATALINES;
Park 72
Taylor 81
Lee 57
;
DATA one_two; SET one two;
RUN;
PROC sort data=one_two; by name; RUN;
PROC PRINT; RUN;
```

You may also want to sort each data first then combine them using

```
SET one two; by name;
```

SET data1; SET data2: Combine data side-by-side. If a variable is common for both data then SAS will use the values from the last data set. For a practice, use the above data and replace the SET statement with SET one; SET two;

<u>Conditional SET:</u> In the previous example, suppose that we want to calculate the difference between each score and overall mean. Then, first we can make a new data having the mean score then combine with the original data. Details will be discussed in class. Note that we use PROC MEANS which will be covered more later.

```
/* Example */
PROC means DATA=one_two;
Var score;
Output out=onetwo_out(KEEP=meanscore) Mean = meanscore;
run;
PROC PRINT; RUN;

DATA new;
   SET one_two;
   IF _N_=1 then SET onetwo_out;
   Diff = score - meanscore;
RUN;

PROC PRINT; RUN;
```

<u>One-to One MERGE</u>: Merge statement is used to combine variables (column) from multiple data sets. Let's consider two data sets below. One is the student roster and another is grade data.

```
DATA roster;
INPUT name $ major $ class $;
DATALINES;
Smith CS Senior
Chen STAT Junior
Rod MATH Sophomore
Park MATH Junior
Taylor SC Sophomore
Lee STAT Senior
DATA grade;
INPUT student $ score grade $;
DATALINES;
Smith 65 C
Chen 78 B
Park 95 A
Taylor 87 B
Lee 69 C
PROC sort data=roster; by name; run;
PROC sort data=grade; by student; run;
DATA list;
 MERGE roster grade (rename=(student=name)); by name;
PROC PRINT; RUN;
```

It is important to note that before merging two data sets by name variable data must be sorted by the variable. In case two data sets use difference names of the variable you want to use to merge them, you can use RENAME= option in MERGE statement.

Note also that student name Rod is in the roster data but not in grade data and you may want to keep those records which are in both data sets. You can use IN= option to do this. For example,

```
DATA list;
  MERGE roster(IN=Inroster) grade(IN=Ingrade); by name;
  IF Inroster=1 and Ingrade=1;
RUN;
PROC PRINT; RUN;
```

Here both Inroster and Ingrade variables (you name them) are logical variables which return 1 if an observation is in the data. For example, Inroster=1 and Ingrade=1 means the observation is in both roster and grade data. Inroster=1 and Ingrade=0 means that the observation is in roster data but not in grade data (the case for the student named Ron). The data list in the example above will involved roster and grade information for five students excluding Ron.

One-to-n Merge: Shorter record will repeat. See below.

```
DATA one;
INPUT ID X;
DATALINES;
1 32
3 54
4 76
5 34
```

```
DATA two;
INPUT ID A $;
DATALINES;

1 CD
1 DVD
2 USB
3 DVD
3 HD
3 CDR
5 CD
5 DVD;
DATA combine; MERGE one two; BY ID; RUN;
PROC PRINT; RUN;
```

Note that reversing the order in MERGE statement will not affect the result. Result will be discussed in class

<u>n-to-n Merge:</u> No problem. They will match line by line assuming data are sorted by the variable used in BY statement.

n-to-m Merge: You may not want to do this. See below.

```
DATA one;
INPUT ID X;
DATALINES;
1 32
1 40
1 99
3 54
4 76
5 50
5 34
DATA two;
INPUT ID A $;
DATALINES;
1 CD
1 DVD
2 USB
3 DVD
3 HD
3 CDR
5 CD
5 DVD
DATA combine; merge one two; by ID; run;
PROC print; run;
```

```
/* Example 3 1 */
 data stock;
    input Model $ unit Price;
    format Price dollar8.2;
 datalines;
 CRX050 1254 69.50
 KTX012 965 99.99
 DVR010 365 169.85
 SAM055 62 129.95
 LGC052 124 144.59
 PHS199 785 81.99
 proc sort; by model; run;
 proc print; run;
 DATA purc;
 INFILE DATALINES MISSOVER;
 INPUT (CustID Model) ($) Quantity @; OUTPUT;
 INPUT Model $ Quantity @; OUTPUT;
 INPUT Model $ Quantity ; OUTPUT;
 datalines;
 1001 CRX050 150 SAM055 25
 1002 CRX050 200 LGC052 40 PHS199 30
 1003 SAM055 50
 1004 KTX012 200 LGC052 90 CRX050 30
 1005 CRX050 100
 DATA purc; set purc;
 if missing (Model) then delete; run;
 proc sort; by Model; run;
 proc print; run;
 Data byModel (drop=Quantity CustID); set purc; by model;
 If first.model then unit out=0;
 unit out+Quantity;
 if last.model; run;
 Data comb;
 merge stock byModel; by model;
 if missing(unit out) then unit out=0;
 Unit left=unit-unit out;
 if Unit left <=0 then file print;</pre>
   put "Please note that the model " model " is out of order"; output;
 run;
 proc print; run;
```

/\* IN-CLASS \*/

/\* Reconsider the sales data in Example 2 2

(a) Create separate data set for each store

```
/* For example,
   DATA D1 D2 D3;
   IF store = 'KENWOOD' THEN OUTPUT D1;
   ELSE IF store='WESTSIDE' THEN OUTPUT D2;
   ELSE IF store='SOUTHHILL' THEN OUTPUT D3;
*/
```

- (b) For each data calculate the overall average sales a day, identify the most sold item per day and its proportion to overall daily sales
- (c) Combine the data in (b) and also identify the most sold item all three stores combined. Your data should look something like

STORE	Daily_sales	Most_Sold/proportion
KENWOOD	\$xxxxxxxx	XXXXX / .xxxx
WESTSIDE	\$xxxxxxxx	XXXXX / .xxxx
SOUTHHILL	\$xxxxxxxx	XXXXX / .xxxx
TOTAL	\$xxxxxxxx	XXXXX / .xxxx

## NOTE #4: ARRAYS

Many cases we have to deal with a group of many variables for data manipulation, summary, and analysis. SAS array can be used wisely to have your tasks more efficient and simple. It can be better explained by examples.

```
/* Example 1-1 */
/* replacing missing score with observation min */
DATA score1;
input ID $ test1-test5;
datalines;
001 52 63 98 54 25
002 54 . 89 74 45
003 63 65 87 89 74
004 36 25 45 . 39
005 96 98 97 89 74
006 85 . 74 65 .
DATA array1; set score1;
Array array_test{5} test1-test5;
 Do i = 1 to 5;
   if missing(array test{i}) then array test{i}=smallest(1,of test1-test5);
 end:
Drop i;
average=mean(of test1-test5);
proc print; run;
/* Example 1-2 */
/* replacing missing score with variable min */
DATA score1;
input ID $ test1-test5;
datalines;
001 52 63 98 54 25
002 54 . 89 74 45
003 63 65 87 89 74
004 36 25 45 . 39
005 96 98 97 89 74
006 85 . 74 65 .
PROC TRANSPOSE data=score1 out=score2; run;
DATA score_tr; set score2;
Array array_tr{6} col1-col6;
Do i = 1 to 6;
   if missing(array tr{i}) then array tr{i}=min(of col1-col6);
 end;
Drop i;
PROC TRANSPOSE data=score tr out=score1 (drop= NAME );
run;
DATA score3; set score1;
average=mean(of test1-test5);
PROC PRINT; RUN;
```

```
/* Example 2 */
 DATA score2;
  input name $ (grade1-grade3) ($) unit1-unit3;
  datalines;
 Mike A B C 4 5 4
 Ruth C D B 3 3 4
 Kate B B A 5 3 4
 Scott B A A 4 4 5
 DATA array2; set score2;
  Array array grade{3} grade1-grade3;
  Array array unit{3} unit1-unit3;
  Array array_GP{3};
 Do i = 1 to 3;
  if array grade{i}='A' then array GP{i}=4*array unit{i};
  if array_grade{i}='B' then array_GP{i}=3*array_unit{i};
  if array_grade{i}='C' then array_GP{i}=2*array_unit{i};
  if array grade{i}='D' then array GP{i}=1*array unit{i};
  end;
  Drop i;
  total GP=sum(of array GP1-array GP3); total unit=sum(of unit1-unit3);
  GPA=total GP/total unit;
  format GPA f3.1;
 proc print; run;
 /* Example 3 */
data array3;
  input code $ dept $ Q1 Q2 status $;
DATALINES;
AAA QA 2.3 5.9 Green
AAB SD 5.2 4.7 NA
XXX RT 6.2 4.1 Blue
ACC NA 1.2 2.3 NA
AAD XXX 1.0 5.3 Green
proc print; run;
data array3_1;
  set array3;
   array char array{*} $ CHARACTER; * SAS will calculate the size of the;
                                        * array. character ( numeric ) is
                                        * used to use all character (numeric);
   array num_array{*} NUMERIC ;
                                        * variables in data;
     do i = 1 to dim(char_array);
                                       \star dim() will return the number of;
                                       * column in the array;
      if char array{i} in ('NA' 'XXX') then
      call missing(char array{i});
     char array{i} = lowcase(char array{i});
    end;
     do i = 1 to dim(num_array);
      if num array\{i\} LE 3.0 then
       call missing(num array{i});
   end;
   drop i;
run;
proc print; run;
 /* CLASS PRACTICE 1*/
 /* Using SAS array convert the following Fahrenheit degrees to Celsius.*/
```

```
/* Example 4 */
^{\prime\prime} Example of using multidimensional array to read 2-dim table vaule ^{\star\prime}
/* This example will read percent points of t distribution at
/* df=1 to 10 and prob points .90, .95, .975, .99
DATA tval;
input df p;
Datalines;
1 .90
3 .99
9.95
10 .975
Data t_table;
array ttable{10,4} temporary;
if N_= 1 then
  Do \overline{d}f=1 to 10;
    Do prob=1 to 4;
        input ttable{df,prob} @;
    END;
  END;
 SET tval;
 IF p=.90 then prob=1;
 ELSE IF p=.95 then prob=2;
 ELSE IF p=.975 then prob=3;
 ELSE IF p=.99 then prob=4;
 t_value=ttable{df,prob};
 DATALINES;
 3.078 6.314 12.706 31.821
 1.886 2.920 4.303 6.965
 1.638 2.353 3.182 4.541
 1.533 2.132 2.776 3.747
 1.476 2.015 2.571 3.365
 1.440 1.943 2.447 3.143
 1.415 1.895 2.365 2.998
 1.397 1.860 2.306 2.896
 1.383 1.833 2.262 2.821
1.372 1.812 2.228 2.764
 proc print data=t_table NOOBS;
 var df p t_value;
 run;
/* CLASS PRACTICE 2 */
To enter a college your SAT scores must be at least 670, 650, 610 for Reading,
writing, and math, respectively, or total score of 2050. Consider the following
score list.
001 580 590 680
002 780 710 600
003 680 650 600
```

/\* 45 85 47 96 102 58 78 65 87 58 \*/

```
004 690 680 710
005 780 800 760
006 520 780 780
007 630 680 720
008 780 710 640
009 580 780 710
010 670 650 620
```

Use SAS array to count the number of tests passed and totals scores. Then, determine whether each student is accepted or rejected.

## NOTE #5: DISPLAYING DATA

## PROC PRINT;

```
/* Example 5-1 */
PROC FORMAT;
VALUE $Class 'F'= 'Freshman' 'S'= 'Sophomore'
'J'= 'Junior' 'Sr'='Seniour';
RUN;
DATA roster;
INFILE datalines FIRSTOBS=3 MISSOVER;
 INPUT NAME $2-14 Class $15-16 (Score1 Score2 Score3)(3.);
 ARRAY Test (3) Score1-Score3;
 DO i = 1 \text{ to } 3;
     If missing(Test(i)) then Test(i) =0;
 END;
DROP i;
Avg=round (mean (of Score1-Score3), 4.1);
Datalines;
STAT101 Elementary Statistic
Fall 2010 Grade Roster
C Gonzalez J 82 60 78
          Sr 92 58 96
T Chung
I Irwin
           F 52 69 77
C Reid S 98 92 89
P Lee S 60 48 70
             S 60 48 70
P Lee
B Washington J 47 63
K Piazza F 87 72 90
M Crafton F 100 74100
A Shulz S 47 61
S Olson F 74 77 75
C Chen S 90 73 68
H Wood F 100 98
H Wood
Title1 " CLASS ROSTER STAT101";
Title2 "========;
Title4 "Prof. S KIM";
footnote "footnote appears at the bottom of output page";
PROC sort DATA=roster; by Class; run;
Options nodate pageno=1 linesize=64 pagesize=60;
PROC PRINT DATA=roster LABEL heading=horizontal N="No of Students = ";
by Class;
Where Avg GE 70 & Class in ('S' 'F');
id;
var NAME Class Avg;
sum Avg;
format Class $Class.;
label NAME = "Student Name"
       Avg = "Average Score";
run;
/* Example 5-2*/
/* Consider the store data in Example 2-2 */
```

```
/* Assuming the data is saved as SASPRAC.sales */
Proc sort DATA= SASPRAC.sales; By Store; RUN;

Title;
Footnote;
Option nodate pageno=1;

PROC PRINT LABEL DATA= SASPRAC.sales (firstobs=1 obs=3) DOUBLE N="Total Number of items: ";
id Store;
by Store;
RUN;
```

<u>PROC REPORT</u>; More controls over Proc Print. Usages of variables are DISPLAY (default for character), Analysis (default for numeric), ORDER, GROUP, ACROSS, or COMPUTE. Each usage is illustrated in examples below.

```
/* Example 5-3 */
Option pageno=1 nodate ;
PROC FORMAT;
VALUE $Class 'F'= 'Freshman' 'S'= 'Sophomore'
 'J'= 'Junior' 'Sr'='Seniour';
RUN;
DATA roster;
INFILE datalines FIRSTOBS=3 MISSOVER;
 INPUT NAME $2-14 Class $15-16 (Score1 Score2 Score3)(3.);
  LastName=scan(Name, 2, '');
  ARRAY Test (3) Score1-Score3;
  DO i = 1 \text{ to } 3;
     If missing(Test(i)) then Test(i) =0;
  END;
  DROP i;
 Avg=round (mean (of Score1-Score3), 4.1);
Datalines;
STAT101 Elementary Statistic
Fall 2010 Grade Roster
 C Gonzalez J 82 60 78
T Chung Sr 92 58 96 I Irwin F 52 69 77
 C Reid S 98 92 89
 P Lee
             S 60 48 70
 B Washington J 47 63
 K Piazza F 87 72 90
 M Crafton F 100 74100
A Shulz S 47 61
S Olson F 74 77 75
C Chen S 90 73 68
H Wood F 100 98
proc print; run;
Title; footnote;
Title3 " CLASS ROSTER STAT101";
Title4 "=======;;
Title5 "Prof. S KIM";
PROC REPORT DATA=roster nowd headline ls=84 ps=80;
 column Class Name LastName Avg Grade;
 define Class / order width=8 Format = $Class.;
```

```
define Name
                 / width=15 spacing = 10;
define LastName / order noprint;
define Avg / display "Average/Score" width=7;
                               *need display to be used in COMPUTE;
define Grade
                  / computed center width=7;
COMPUTE Grade / character length=7;
  if Avg gt 90 then Grade = 'A';
 Else if Avg gt 70 then Grade = 'B';
 Else if Avg gt 50 then Grade = 'C';
 Else if not missing(Avg) then Grade = 'D';
ENDCOMP;
run;
PROC REPORT DATA=roster nowd headline 1s=84 ps=80;
column Class Avg ;
define Class / group Format = $Class.;
define Avg / analysis mean "Class Average"; *sum is default;
RUN;
```

## NOTE:

- 1. COLUMN statement is used as VAR statement in PROC PRINT.
- 2. To subset the data in a report, WHERE statement can be used with PROC REPORT the same way we used in PROC PRINT (see example 1).
- 3. Both ORDER and GROUP order rows and suppress printing repetitions of the variables. The differences are that GROUP collapses observation with the same values in the variable and gives you summary report, while ORDER provides the list of all observations.
- 4. ANALYSIS is used to calculate a statistic. The default statistic is SUM. Above example used mean for the summary statistic. Other available statistics are: CSS (correlated sum of squares), USS (uncorrelated SS), CV (coefficient of variation), MAX, MEAN, MIN, N, NMISS (number of observation with missing values), and RANGE.
- 5. Computed variables are that you define in PROC REPORT and are not in the data set. You have to include the names of the computed variable in the COLUMN statement.
- 6. You have to use DISPLAY for a numerical variable to be used in COMPUTE. ACROSS functions similarly to GROUP, but it creates the cross variable horizontally while GROUP put the variable vertically. The following example illustrates the use of ACROSS variables.

```
/* Example 5 4 */
/* This example is taken from the book "Learning SAS by Example" */
/* by Ron Cody */
data blood;
   infile 'J:\SASCLASS\blood.txt' truncover;
   length Gender $ 6 BloodType $ 2 AgeGroup $ 5;
   input Subject
         Gender
         BloodType
         AgeGroup
         WBC
         RBC.
         Chol;
   label Gender = "Gender"
         BloodType = "Blood Type"
         AgeGroup = "Age Group"
         Chol = "Cholesterol";
run;
title "Averate Blood Counts by Age Group";
proc report data=blood nowd headline;
```

- /\* In-Class practice (from Cody's book)
- 1. Using the blood data, produce a report showing the mean cholesterol for each combination of Gender and blood type.
- 2. Using the same data, produce a table of average WBC and RBC counts for each combination of blood type and gender.  $\star/$

## NOTE #6: DESCRIPTIVE AND UNIVARIATE STATISTICS I

## PROC MEANS;

# Mostly used Statistics in PROC MEANS:

```
Lower and Upper 95% confidence interval for mean
LCLM/UCLM 95% Lower/Upper Confidence Limit for mean
KURT
            Kurtosis
MAX
            Maximum
MEAN
           Average
MEDIAN
           Median
            Minimum
            Number of observations without missing
            Number of observations with missings
NMISS
PROBT
            Probability of a greater absolute value for t-value
P95
            95th percentile (also available P1, P5, P10, P25, P50, P75, P90,
            P99)
Q1 / Q3
            25th / 75th percentile
RANGE
            Range
STD
            Standard deviation
SUM
            Sum
            t-test for Ho: mean = 0
VAR
            Variance
/* Example 5-1 */
DATA Ex5 1;
INPUT Class $ Gender $ Score;
DATALINES;
A Male 96
A Male 87
A Male 89
A Female 98
A Female 82
B Male 65
B Male 85
B Female 63
B Female 93
B Female 77
C Male 62
C Male 94
C Male 80
C Female 99
PROC MEANS DATA = ex5 1;
BY Class; RUN; * To use BY statement data must be sorted by the variable;
PROC SORT DATA = ex5 1 OUT=ex5 1sort;
by Gender; run;
PROC MEANS DATA = ex5_1sort MEAN N;
BY Gender; RUN;
```

```
PROC MEANS DATA = ex5 1 chartype; * this option will give you binary TYPE;
Class Class Gender; *Class statement doesn't require sorted data;
OUTPUT OUT = ex5 lout
 N = count
 Mean = meanscore;
Proc Print DATA=ex5 lout; run;
Proc Print DATA=ex5 lout (DROP = FREQ );
Where TYPE EQ '11'; *note TYPE is a character variable;
/* Example 5-2 */
DATA GNP; SET SASHELP.GNP;
Year =year(date);
quarter = qtr(date);
PROC MEANS DATA = GNP chartype;
VAR GNP CONSUMP INVEST EXPORTS GOVT;
CLASS quarter;
OUTPUT OUT = gnp out
      /* (drop = _:) will remove all variable beginning with an underscore */
  N (quarter) = count
  MEAN (GNP CONSUMP) =
  STD (GNP CONSUMP) =
  MAX (INVEST EXPORTS) =
  LCLM (GNP) = UCLM (GNP) = / autoname;
PROC PRINT data = gnp out heading=horizontal; RUN;
DATA gnp CI; SET gnp out (KEEP=Quarter GNP Mean GNP StdDev count);
 DO i =1 to 5;
  IF N = i then
   DO;
     LL = GNP_Mean - TINV (.975, count-1) * GNP_StdDev/SQRT(count);
     UL = GNP Mean + TINV (.975, count-1) * GNP StdDev/SQRT(count);
 END;
 DROP i;
RUN;
Proc Print DATA = gnp CI heading=horizontal; run;
/* Example 5-3 */
/* This example simulate 100 random samples of each size 30 from N(MU, STD) and
calculate 95% CI for mean for each sample */
DATA Rand Norm;
Count=100; N=30; MU=5; STD=2; seed=0;
 DO I=1 TO Count;
      DO K=1 TO N;
      X=MU+STD*RANNOR(seed); OUTPUT;
      END;
 END;
RUN;
PROC PRINT; RUN;
PROC MEANS NOPRINT MEAN; BY I; VAR X;
OUTPUT OUT=mean out MEAN=Mean STD= SD N= N;
RUN;
```

```
DATA CI; SET mean_out; MU=5;
LOWER=Mean-TINV(.975, N-1)*SD/SQRT(N);
UPPER=Mean+TINV(.975, N-1)*SD/SQRT(N);
IN=0; IF LOWER<MU<UPPER THEN IN=1;

PROC MEANS MEAN SUM;
VAR Mean SD LOWER UPPER IN;
RUN;

/* IN-CLASS

Generate 100 random samples from Poisson distribution with a mean of your choice and calculate 90% CI (a) using normal approximation (2) using 5% and 95% tiles. Count the number of intervals which contain the true mean and compare.

*/
```

## **Functions for Random Samples:**

```
Distribution

Binomial (n,p)

Exponential (lambda)

Beta (alpha, beta)

Normal (mu, sigma)

Poisson (mean)

Uniform (b, a+b)

RANBIN (seed, n,p)

ranexp (seed) /lambda

beta*rangam (seed, alpha);

mu+sigma*rannor (seed);

RANPOI (seed, mean)

a*ranuni (seed) +b
```

<u>PROC FREQ</u>; The procedure provides tables (one, two, and three ways) of counting frequencies of both character and numeric variables.

```
/* Example */
/* Consider the data in Example 5-1 */
Proc Format;
value grade 0 -< 70 = 'C to D'
             70 - < 90 = 'B to C'
             90 - HIGH = 'A to B';
RUN;
PROC FREQ DATA=Ex5 1 order=formatted;
                *also available order=data, order=freq;
Format Score grade.;
                *format is used to convert numeric to character category;
 Tables Class / nocum nopercent;
                *One-way table. nocum removes cumulative statistics columns;
                *nopercent will not give percent;
Tables Gender*Score / Chisq;
                *Chisq gives test for independence between gender and score;
Table Score * (Class Gender) / nocol norow fisher;
                    *two two-way tables: Score*Class, Score*Gender;
                    *nocol norow will remove conditional prob and ;
                    *fisher also gives Fisher's Exact test for independence;
RUN;
```

<u>PROC TABULATE</u>: This procedure can be used to generate tabular reports which involve descriptive statistics.

# NOTE #6-1: DESCRIPTIVE AND UNIVARIATE STATISTICS I

# **GRAPHICS**

```
The outputs and options will be discussed in details in class.
/* Example 1*/
/* Revisit the Sales data in Example 2-2 */
OPTIONS LINESIZE=75 PAGESIZE=54 NODATE PAGENO=1;
DM "output; clear; log; clear";
GOPTIONS reset=all goutmode=replace
                        Vsize=6 Hsize=6 Horigin=1.2
                        htitle=2.0 ftitle=simplex
                        htext=1.0 ftext=simplex;
PROC GCHART DATA=sales;
vbar prod / sumvar =unit;
vbar d/ discrete;
vbar store / sumvar = unit
            type = sum
             subgroup = prod;
pie prod / sumvar = sales
          type = sum;
pie store / sumvar = sales
            type = sum
            subgroup = prod;
run; quit;
/* Example 2 */
/* Data is SASHELP.AIR */
/* Scatter plot with a smooth line superimposed*/
Title "Example Scatterplot of AIR data";
Title2 h=1.0 "with Interpolation line superimposed";
Symbol value = dot interpol = sms50 line=1 width=.5;
  *number following sms (sorted smooth) indicates smootheness of curve;
  * choose from 0 to 99 (most smooth). Default is 0;
  * try several numbers;
PROC GPLOT DATA = SASHELP.AIR ;
plot air*DATE;
run;
```

# NOTE #7: DESCRIPTIVE AND UNIVARIATE STATISTICS II

<u>PROC UNIVARIATE</u>; The procedure provides more extensive list of statistics and is one of the most useful procedures.

The default output of this procedure is very comprehensive. We will discuss a sample output in class. Following is list of statements that can be used with the procedure. We will discuss the statements via following examples.

#### List of statements:

```
BY variables ;
 CLASS variables ;
 FREQ variables ;
 HISTOGRAM < variables > < / options > ;
 ID variables ;
 PROBPLOT < variables > < / options > ;
 QQPLOT < variables > < / options > ;
 VAR variables ;
 WEIGHT variable;
/* Example 7-1 */
OPTIONS LINESIZE=75 PAGESIZE=54 NODATE PAGENO=1;
DM "output; clear; log; clear";
GOPTIONS reset=all goutmode=replace
                         Vsize=6 Hsize=6 Horigin=1.2
                          htitle=1.0 ftitle=simplex
                          htext=1.0 ftext=simplex;
DATA GNP; SET SASHELP.GNP;
Year =year(date);
quarter = qtr(date);
RUN;
PROC SORT; by quarter; RUN;
PROC UNIVARIATE DATA = GNP NORMAL;
var GNP;
by quarter;
QQPLOT GNP;
HISTOGRAM GNP /Midpoints = 600 to 5400 by 600 NORMAL CFILL = ltgray;
INSET MEAN = 'Mean' (8.1) STD = 'StdDev' (8.1) / POSITION = NE;
RUN;
```

All statistics provided in PROC MEANS also available with PROC UNIVARIATE. Beside those statistics, PROC UNIVARIATE computes much more comprehensive list of statistics (see sample output).

In OUTPUT statement we can make use of the statistics used with PROC MEANS. Using PROC UNIVARIATE we can output more percentiles than those automatically calculated with the procedure. See the example below.

```
/*Example 7-2 */
DATA Rand_Norm;
   Count=100; N=30; MU=5; STD=2; seed=0;
   DO I=1 TO Count;
   DO K=1 TO N;
       X=MU+STD*RANNOR(seed); OUTPUT;
   END;
END;
RUN;

Proc Univariate data = Rand_Norm;
   var X;
   by I;
   output out=norm_out pctlpre=P_ pctlpts=2.5 to 10 by 2.5 95 to 100 by 2.5;
   run;
   proc print; run;
```

# Making output data set from portion of SAS output

We have noticed that the SAS output from PROC UNIVARIATE is very comprehensive and you may not need all of them. Using ODS (Output Delivery System) you can choose parts of SAS output to be printed or to be saved as an output data that can be used for further analysis. To have parts to be selected we first need to know the parts names. For this, you can first run the following, for example,

```
/*Example 7-3 */
ODS TRACE ON;
PROC UNIVARIATE DATA = SASHELP.GNP NORMAL;
var GNP INVEST;
RUN;
ODS TRACE OFF;
```

In the LOG window you will see the following list. Note that the following is a part of the list.

```
Output Added:

Name: Moments
Label: Moments
Template: base.univariate.Moments
Path: Univariate.GNP.Moments

Output Added:

Name: BasicMeasures
Label: Basic Measures of Location and Variability
Template: base.univariate.Measures
Path: Univariate.GNP.BasicMeasures

Path: Univariate.GNP.BasicMeasures
```

Now, you can print the moment part of the SAS output using ODS LISTING statement in the procedure.

```
/*Example 7-4 */
ODS LISTING EXCLUDE ALL; *this will turn off all output;
```

```
PROC UNIVARIATE DATA = SASHELP.GNP NORMAL;
  var GNP INVEST;
  ODS LISTING SELECT Moments;
RUN;
ODS LISTING; *this will turn back on all output;
```

Note that you will see two moment statistics tables; one for GNP and another for INVEST. This is because both peaces have the same output name, "Moments". To have the output for GNP only you can use the path name. For example, you can replace the ODS LISING statements as

```
ODS LISTING SELECT Univariate.GNP.Moments;
```

Next step is to create a SAS data from parts of SAS output. For this we can use ODS OUPUT statement.

```
/*Example 7-5 */
PROC UNIVARIATE DATA = SASHELP.GNP NORMAL;
var GNP INVEST;
ODS OUTPUT Univariate.GNP.Quantiles = GNP_Q;
ODS OUTPUT Univariate.INVEST.Quantiles = INV_Q;
RUN;
DATA QUANT; SET GNP_Q INV_Q;
PROC PRINT DATA=QUANT NOOBS;
WHERE Quantile in ('5%','95%'); RUN;
```

Though mostly useful with a comprehensive procedure like UNIVARIATE, ODS can be used with any statistical procedures. For example,

```
/*Example 7-6 */
      DATA GNP; SET SASHELP.GNP;
      Year =year(date);
      quarter = qtr(date); RUN;
      PROC TTEST DATA = GNP; where quarter in (1,2);
      class quarter;
      var GNP ;
      ODS OUTPUT ttests=GNP t; RUN;
      DATA NULL; SET GNP t;
      IF variances = 'Equal';
      FILE PRINT;
      PUT @10 "Two sample T-Test " / @10 "Comparing mean GNP for first and second
      quarter" /
      @10 60*'-' / ;
      IF Probt < .05 Then
         PUT @10 "Since P-value = " Probt " less than .05, we reject Ho: equal mean
      at .05 level";
      ELSE IF Probt >= .05 Then
        PUT @10 "Since P-value = " Probt " greater than .05, " / @10 "we DO NOT
      reject Ho: equal mean at .05 level"; RUN;
/* INCLASS PRACTICE 1*/
```

1. Simulate a random sample of size 1000 from Uniform(0,1). Write a SAS code to provide the test for normality. Also provide histogram and QQ plot. Based on the result of the normality test provide either 95% t-confidence interval or 95% percentile interval. You should use the ODS and your output should look something like:

Variable : XXXX

Sampling Distribution: Uniform (0,1)

Normality Test (Shapiro-Wilk): Rejected/accepted with P-value xxx

Sample mean: xxx Sample std: xxx

95% confidence interval: (xxx,xxx)

NOTE: Due to the lack of normality the CI is based on 2.5 and 97.5 percentiles. (this note is not necessary for normal sample)

2. Repeat with a random sample of size 1000 from Poisson distribution with mean of your choice.

#### /\* INCLASS PRACTICE 2\*/

We study the approximate normality of the sum of continuous uniform random variables.

- (1) Simulate a random sample,  $Y_1$ , of size 1000 from Unif(0,1)
- (2) Simulate another r.s., U, from Unif(0,1) and let  $Y_2=Y_1+U$
- (3) Simulate another r.s., U, from Unif(0,1) and let  $Y_3=Y_2+U$
- (4) Continue until you have  $Y_1, Y_2, ..., Y_{10}$ . Your data should consist of these 10 variables.
- (5) Using the Univariate procedure, provide histogram for each variable. Also provide test results for normality for each variable. You output should look something like:

Variable	P-value(S-W)	Normality
Y1	XXX	YES/NO
Y2	XXX	YES/NO

(6) Depending on the result from normality test, provide either 95% tconfidence interval or 95% percentile interval for mean for each variable. Add the information to the result in (5), for example,

Variable	P-value(S-W)	Normality	Sample mean	CI
Y1	XXX	YES/NO	XXX	(xxx, xxx)
Y2	XXX	YES/NO	XXX	(xxx, xxx)

NOTE that due to the lack of normality for variable xx, xx, xx, and xx, the CI's are based on 2.5 and 97.5 percentile.

### NOTE #8: SAS/IML

SAS/IML (Interactive Matrix Language) is flexible and powerful programming language that operates on one or two dimensional data matrix. The software comes with lots of built-in operators and call routines that can be used for some complex calculations in matrix. We will discuss some basic features of IML trough examples.

There are two ways of generating matrix: defining a matrix in Proc IML statement or reading a SAS data set into a matrix. For example,

```
PROC IML;

M1 = {11 12, 21 22, 31 32};

M2 = M1`;

PRINT M1 M2;

QUIT;
```

Note that the RUN statement is not used to execute IML statements. IML statements are executed immediately when submitted. Use the QUIT statement to terminate the IML procedure.

This will create a 3×2 matrix M1 and its transpose M2.

We can also create a matrix from a SAS data. For example,

```
DATA D1;
input X $ Y Z;
Datalines;
A 14 23
B 12 20
C 10 18
D 8 15;

PROC IML;
USE D1;
READ ALL VAR{Y Z} INTO M1;
PRINT M1;
QUIT;
```

This will read the SAS data D1 and generate the matrix M1 using the two numeric variables. Note that you can't mix numeric and character variables in a matrix.

You can create SAS dataset from matrices generated from PROC IML by adding the following statements:

```
CREATE dataname FROM matrixname;
APPEND FROM matrixname;
```

Following is a list of functions and operators used in PROC IML.

```
+, -, #, / Element-by-element addition, subtraction, multiplication, and division

** Matrix multiplication

** Matrix power

## Element by element power

sqrt(A) Element-by-element square root

nrow(A), ncol(A) number of rows, number of columns in A

det(A), inv(A), trace(A) determinant, inverse, trace of A
```

```
eigval(A), eigvec(A)
                       a vector of eigenvalues and eigenvectors of A
diag(A)
                       make a diagonal matrix of A
                       make a column vector out of the diagonal element of A
vecdiag(A)
                       extract (m,n) element of A
A[m,n]
A[m]
                       extract mth row of A (all columns)
A[,n]
                       extract nth column of A (all rows)
A[+], A[+,], A[,+]
                       overall sum, row vector of column sums, and column vector of row sums
A[:,], A[,:]
                       row vector of column means and column vector of row means
A[<>,], A[><,]
                       row vector of the maximum (minimum) of each of columns
A[,<>], A[,><]
                       column vector of the maximum (minimum) of each of rows
A[<:>,], A[>:<,]
                       row vector of index of the maximum (minimum) of each of columns
                       column vector of index of the maximum (minimum) of each of rows
A[,<:>], A[,>:<]
                       identity matrix of size n
I(n)
                       create m by n matrix with matrix entry A
repeat(A, m, n)
shape(A, m, n)
                       create m by n matrix with elements in A
                       create m by n matrix with element x
J(m,n,x)
A \parallel B
                       put the two matrices horizontally
A // B
                       put the two matrices vertically
t(A)
                       A transpose
/* Example 8-1 */
PROC IML;
   M = \{ 958, 617, 293 \};
   ColMin=M[><,]; ColMax= M[<>,];
   RowMin=M[,><]; RowMax= M[,<>];
   E=eigval(M); V=eigvec(M);
  sum=M[+,]; mean=M[:,]; ss=M[##];
  ss1= (M#M)[+];
  diag=diag(M); dvec=vecdiag(M); *diag function also make a diagonal matrix;
                                       * from a vector;
  X=repeat(M, 2,2);
                       XX=shape(M, 2,2);
  PRINT M ColMin ColMax RowMin RowMax;
  PRINT E V;
  PRINT sum mean ss ss1;
  PRINT diag dvec;
  PRINT X XX;
  CREAT D1 FROM M;
  APPEND FROM M;
OUIT;
PROC PRINT DATA = D1; RUN;
/* Example 8-2 */
/* OLS estimation for a multiple regression*/
PROC IMI:
USE SASHELP.GNP;
READ point (1:100) VAR{GNP CONSUMP INVEST} INTO M;
N=NROW(M); r=NCOL(M);
Y=M[,1]; X=SHAPE(1,N,1)||M[,2:3];
 BETA=INV(t(X)*X)*t(X)*Y;
```

```
SIGMA = SQRT(t(Y-X*BETA)*(Y-X*BETA)/(N-r));
 SE=SIGMA*SQRT(VECDIAG(INV(t(X)*X)));
 TVALUE=BETA/SE;
 PVALUE=(1-PROBT(ABS(TVALUE),N-r))*2;
Table=BETA||SE||TVALUE||PVALUE;
Title "Parameter Estimates";
PRINT Table(|colname={ESTIMATE SE TVALUE PVALUE}
              rowname={INTERCPT CONSUMP INVEST} FORMAT=8.2|);
CREATE D1 FROM Table;
APPEND FROM Table;
QUIT;
/* Example 8-3 */
/* Solving nonlinear Equation */
/* one dimension */
goptions
colors=(black );
axis1 style=0 major=none minor=none value=none label=none;
axis2 style=0 major=none minor=none value=none label=none;
PROC IML;
  Create Newton1 var{X F};
  X=30; F=10;
  Do i=1 to 20 until (abs(F) < .00001);
     F=X*X-cos(X);
     DF=2*X+sin(X);
    X=X-(F/DF); print X F;
 end:
append;
quit;
data forplot; set Newton1;
  do c=0 to 2 by .01;
   y=c*c-cos(c); output;
  end:
run;
proc gplot;
  plot y*c F*X F*X/overlay
  haxis= axis1 vaxis= axis2 href = 0 vref = 0;
 axis1 order = 0 to 2 by .2;
 axis2 order = -1 to 2 by .1;
 SYMBOL1 V=o H=.5 CV=BLACK;
 SYMBOL2 V=X H=3 CV=RED;
run;
run; quit;
/* Example 8-4 */
/* Solving nonlinear Equation System */
/* Two dimension */
PROC IML;
create final var{x y};
do a = -2 to 2 by 4;
  do b = -1 to 1 by 2;
    theta = (a) // (b); F = \{1, 1\};
    do jj = 1 to 20 until (abs(SUM(F)) < .00001);
      x = theta[1,];
      y = theta[2,];
      F = (x##2 + (4* y##2) - 4)// (y - x##2 + (.4*x) + 1.96);
```

```
DF = ((2*x) | (8 *y)) //((-2*x + .4) | | 1);
      theta = theta - (INV(DF)*F);
      theta2 = theta`;
    end;
    print theta2;
   append;
   end;
end;
quit;
data graph;
  set final;
  do c = -2 to 2 by .001;
   z = sqrt((4 - c**2)/4);
   z1 = -sqrt((4 - c**2)/4);
   z2 = c**2 - .4*c - 1.96;
    output;
  end ;
proc gplot;
  plot z*c z1*c z2*c y*x/ overlay
  haxis= axis1 vaxis= axis2 href = 0 vref = 0;
  axis1 order = -3 to 3 by 1;
  axis2 order = -3 to 3 by 1;
  SYMBOL1 V=o H=.5 CV=BLACK;
  SYMBOL2 V=o H=.5 CV=BLACK;
 SYMBOL3 V=o H=.5 CV=BLACK;
 SYMBOL4 V=X H=.5 CV=RED;
run;
/* IN-CLASS PRACTICE 1
Using IML solve the equation
   F(x,y) = \begin{pmatrix} y - x^2 + \cos(x) \\ y - 4x^2 + 2x + 2 \end{pmatrix} = 0.
Plot the function.
/* Example 8-5 */
/* Here we use IML to find the maxima and/or minima of a polynomial function */
PROC IML;
  Create Newton2 var{X F};
  Do a = -5 to 5 by 10;
     X=a; F=10;
     Do i=1 to 20 until (abs(F) < .00001);
      F=6*X##2 - 6*X - 12;
       DF=12*X - 6;
       X=X-(F/DF); print X F;
     end;
   append;
  end;
quit;
data forplot; set Newton2;
  do c=-5 to 5 by .01;
       y1=2*c**3-3*c**2-12*c+12; output;
       y2=6*c**2-6*c-12; output;
       y3 = 12*c-6;
  end;
```

```
run;
proc gplot;
 plot y1*c y2*c y3*c F*X/overlay
  haxis= axis1 vaxis= axis2 href = 0 vref = 0;
  axis1 order = -5 to 5 by 1;
  axis2 order = -30 to 30 by 10;
 SYMBOL1 V=o H=.5 CV=BLACK;
 SYMBOL2 V=o H=.5 CV=RED;
 SYMBOL3 V=o H=.5 CV=BLUE;
 SYMBOL4 V=X H=3 CV=BLACK;
run; quit;
/* In-class PRACTICE 2 */
/* Use IML to find the maxima and minima of the function
      F(x) = x^4 - 2x^2
Graph the function and the first and the second derivatives.
* /
/* Example 8-6*/
/st This example use IML to find Maximum Likelihood Estimates for the Normal st/
/* distribution using the Newton method */
/* Detailed discussion will be given in class */
DM "OUTPUT; CLEAR; LOG; CLEAR";
%LET MU=1;
%LET STD=2;
%LET N=1000;
DATA NORM1;
DO I=1 TO &N;
X=&MU+&STD*RANNOR(0); OUTPUT;
END;
DATA NORM1; SET NORM1;
KEEP X ;
TITLE " MLE FOR NORMAL DISTRIBUTION ";
PROC IML;
USE NORM1; READ ALL INTO X; X=X`; LLD1= { 1, 1 };
NN=NCOL(X);
THETA={ 0 , .1 };
DO JJ=1 TO 20 UNTIL ( abs(SUM(LLD1)) < .0001 );
 MU=THETA[1,]; S2=THETA[2,];
 ONE=SHAPE(1,NN,1);
 LLD1 = (((X-MU)*ONE)/S2))/((-NN/(2*S2))+((X-MU)*(X-MU)^/(2*S2*S2)));
  LLD2 = (-NN/S2 | | -((X-MU)*ONE) / (S2*S2)) //
        (-((X-MU)*ONE)/(S2*S2) | | (NN/(2*S2*S2))-((X-MU)*(X-MU)^/(S2**3)));
  THETA = THETA - INV(LLD2)*LLD1;
PRINT JJ THETA;
XBAR=X[,+]/NN;
SIGMA2=((X-XBAR)*(X-XBAR)`) /NN;
PRINT XBAR SIGMA2; RUN; QUIT;
```

### NOTE #9: MACRO

SAS Macros are powerful tool for your SAS program being more flexible and efficient. This chapter discusses the basic aspect of the SAS Macro via examples.

```
/* Example 9-1 */
OPTIONS MPRINT MACROGEN SYMBOLGEN SPOOL;

%MACRO CIF(P=,r=,n=);
data AA;
   Do I=1 to 20;
        Aval=&P*(1+&r/&n)**(&n*I); output;
end;
run;

title "20 year balance with initial depoist of $ &P ";
title2 "at &n month compounding rate of &r";

Proc print label noobs;
format Aval dollar12.2;
label I = 'year' Aval = 'Balance';
run;

%MEND;
```

Note that the Macro starts with %Macro name(input arguments); and ends with %Mend;. To invoke the macro, you may add the following at the end of the macro.

```
%CIF(P=3000, r=0.05, n=12);
```

However, you may want to save the macro program as a file in a folder and call whenever you need it. To do that, assuming the macro program is saved as E:\mymacro\cif.sas in a new program editor

```
%INCLUDE 'E:\mymacro\cif.sas';
     %CIF(P=3000, r=0.05, n=12);
/* Example 9-2 */
OPTIONS MPRINT MACROGEN SYMBOLGEN;
Title;
%MACRO SIMUL(par=, n=, var=, out=);
* This macro simulate random sample from Normal distribution
* Macro variable:
   par : mean sd, for example par = 5 3
    n : sample size
    var : variable name for the random sample
    out: the output data containing the random sample
****************
data &out;
%LET mu=%SCAN(&par,1,' ');
%LET sigma=%SCAN(&par,2,' ');
```

```
Do I=1 to &n;
   &var =&mu+ rannor(0) *σ output;
run;
PROC univariate normal;
Histogram &var;
run;
%mend;
/* the macro is saved as 'E:\mymacro\simul.sas' */
%INCLUDE 'E:\mymacro\simul.sas';
%SIMUL(par=5 2 , n=100, var=X, out=sim);
/* Example 9-3 */
%MACRO TRIMMEAN(IN=,OUT=, var=, percent=, TM=);
/**************
* This Macro to calculate p% trimmed-mean
* MACRO Variable
     IN: Input dataset name
     OUT: Output dataset containing Trimmed Mean
    percent: Percent of data to be trimmed
    TM: Calculated Trimmed Mean in output data
*******************************
%LET p1=%EVAL(&percent/2);
%LET p2=%EVAL(100-&p1);
PROC UNIVARIATE DATA=&IN NOPRINT; VAR &var;
OUTPUT OUT=ONE PCTLPTS=&p1 &p2 PCTLPRE=P;
RUN;
DATA TWO; SET ∈ IF N =1 THEN SET ONE;
IF P &p1<&var<P &p2 THEN Y=&var;</pre>
PROC MEANS NOPRINT DATA=TWO; VAR Y;
OUTPUT OUT=THREE MEAN=&TM;
DATA &OUT; SET THREE;
VARIABLE="&var";
DROP _TYPE_ _FREQ_;
RUN;
%MEND TRIMMEAN;
%INCLUDE 'E:\mymacro\simul.sas';
%SIMUL(par=5 2 , n=100, var=X, out=sim);
%TRIMMEAN (IN=sim, OUT=out, var=X, percent=10,TM=TRMean);
PROC PRINT DATA=out;
RUN;
```

Macro Functions (extracted from SAS Macro Programming Made Easy, SAS Press)

Macro character functions

**%INDEX(source, string)** returns the position in *source* of the first character of *string*.

**%LENGTH(string|text expression)** returns the length of string or the length of the results of the resolution of text expression.

**%SCAN(argument, n <,delimiters>)** returns the *nth* word in *argument* where the words in *argument* are separated by *delimiters*.

**%SUBSTR(argument, position<, length>)** extracts a substring of *length* characters from *argument* starting at *position*.

**%UPCASE**(string|text expression) converts character string or text expression to uppercase.

#### Macro Evaluation functions

 ${\tt %EVAL}$  (arithmetic expression | logical expression) evaluates expressions using integer arithmetic.

**%SYSEVALF(arithmetic expression|logical expression <,conversion-type>)** evaluates expressions using floating point arithmetic

**%SYSFUNC**(function(argument(s))<,format>) executes SAS language function or user-written function and returns the results to the macro facility.

#### DATA step interface tools

 ${\tt SYMGET}(argument)$  SAS language function that obtains the value of a macro variable specified as argument and returns this as a character value during DATA step execution.

**SYMGETN**(argument) SAS language function that obtains the value of a macro variable specified as argument and returns this as a numeric value.

**CALL SYMPUT (macro-variable, value)**; SAS language routine that assigns value produced in a DATA step to a macro-variable. This routine does not trim leading and trailing blanks.

CALL SYMPUTX (macro-variable, value <, symboltable>); SAS language routine that assigns value produced in a DATA step to a macro-variable. This routine removes both leading and trailing blanks. Optionally, this routine can direct the macro processor to store the macro variable in a specific symbol table.

**CALL EXECUTE** (argument); SAS language routine that executes the resolved value of argument. Arguments that resolve to a macro facility reference execute immediately. Any SAS language statements resulting from the resolution are executed at the end of the step.

**RESOLVE** (argument) SAS language function that resolves argument during DATA step execution where argument is a text expression. Text expressions include macro variables and macro program calls.

```
/* some example */
OPTION MACROGEN SYMBOLGEN;
```

title "Report for %sysfunc(date(),monname.) %sysfunc(date(),year.)";
Data mprac;

```
%LET date = Today is &sysdate &sysday;
 %LET time= The time is &systime;
 CALL SYMPUT('Todaym', SYMGET('date') || "-" || SYMGET('time')); *Todaym is
a Macro variable;
 Todays=SYMGET('Todaym'); *Todays is a SAS variable;
 %put &todaym; *print the macro variable in LOG widnow;
run;
%LET pr=%STR(PROC PRINT; RUN;);
≺
LET a = 123456789;
LET b = SUBSTR(&a, 2, 3);
%LET c=%EVAL(&a-&b);
%LET d=%SYSEVALF(&a/&b);
%put &a &b &c &d;
/* Example 9-4 */
/* More examples */
%macro sales (IN=, group=, var= );
OPTIONS LINESIZE=75 PAGESIZE=54 NODATE PAGENO=1;
DM "output; clear; log; clear";
ODS RTF File="J:\STA475\ex9 4.rtf"; *this will make a pdf output file;
ODS Listing Close;
*** some summary stat ***;
PROC TABULATE DATA = ∈
CLASS &group;
var &var;
table (&group ALL), &var*(sum mean)*f=10.2;
Keylabel ALL = 'Overall';
PROC GCHART DATA = ∈
Title "Daily Sales Total";
PIE &group / Coutline= black Percent = outside
                  SUMVAR=&var;
**** to make indices for the class variable ****;
PROC SORT data=∈ by &group;
DATA Data1; set ∈ by &group;
ind+first.&group;
run;
**** to calculate the number of levels of the class variable ***;
PROC MEANS MAX DATA=DATA1 NOPRINT;
VAR ind;
OUTPUT out = out1 MAX=max;
DATA _NILL_; SET out1;
CALL SYMPUT ('no_ind',max);
**** this will generate separate data sets for each store;
DATA %DO i=1 %TO &no ind;
         store&i
         %END;
 SET DATA1;
```

```
%DO i=1 %TO &no ind;
   IF ind=&i then output store&i;
  %END:
run;
%DO i=1 %TO &no ind;
PROC MEANS DATA=store&i;
         *PROC MEANS DATA=DATA1 (WHERE= ( ind = &i) ); *this will do the same;
 VAR &var; RUN;
%END;
ODS Listing;
ODS RTF Close;
%mend sales;
DATA sales (DROP = type p);
INPUT @1 type $ @;
IF type="0" THEN DO;
INPUT @3 Store $10.;
Delete;
END;
RETAIN Store;
ELSE IF type ne "@" THEN
INPUT p $1-3 d 4-6 unit 7-11 @12 price Dollar7.2;
SELECT (p);
WHEN ("011") prod = "CDR50";
WHEN ("012") prod = "DVDR-";
WHEN ("014") prod = "DVDDL";
WHEN ("017") prod = "CDR100";
WHEN ("020") prod = "USB2G";
WHEN ("021") prod = "USB8G";
OTHERWISE;
END;
sales=unit*price/d;
FORMAT price Dollar7.2 sales Dollar8.2;
LABEL prod="Product Name" d="Survey Duration" unit="Unit sold"
price="Unit price" sales="Daily Sales";
DATALINES;
@ Kenwood
0110300023601200
0120600065203650
0140300102504190
0170600150702160
0201200056203650
0210900023410100
@ Westside
0110300017801290
0120600025603470
0140300087204090
0170600180701960
0201200114803290
0210900040209900
@ SouthHill
0110300030700900
0120600037403750
0140300087404290
0170600099802045
0201200078403880
0210900041509990
RUN;
PROC PRINT LABEL U NOOBS DATA= sales;
%sales (IN=sales, group=Store, var=sales);
```

cat&i = ( %scan(&names, &i, %str()) < &check <= %scan(&names,</pre>

if &check ne . then do; %do i=2 %to &n;

%end;

&i+1, %str()));

end; keep &categor &dep &ind;

label cat&i="%upcase(&check) group &i";

%let categor=&categor cat&i;
%let list=&list "cat&i",;

```
%* Run a model with the indicator variables in place of the
  variable to be checked for linearity;
proc REG data= temp2 ;
  model &dep = &categor &ind;
  title "Checking %upcase(&check) for linearity";
  title2 "Dep Var = %upcase(&dep)";
  title3 "Covariates: %upcase(&ind)";
%endmacro:
%mend:
*the following is a sample application of the macro;
DATA GNP4; SET SASHELP.GNP;
%CHKLIN(DATA=GNP4,CHECK=INVEST,DEP=GNP,IND=EXPORTS GOVT);
*notice I did not specify N= so N=5;
RUN;
*now I will do N=7;
%CHKLIN(DATA=GNP4,CHECK=INVEST,DEP=GNP,IND=EXPORTS GOVT,N=7);
RUN;
/* IN-CLASS 9-1
Redo Assignment 1 problem 2 using SAS Macro. Define the initial salary, annual
interest rate, compounding term, raise rate, contribution percentages as input
arguments.
/* IN-CLASS 9-2
Write a SAS MACRO program to divide a continuous variable into K equally spaced
categories. Your MACRO should
i) read in a data set, a variable, and the number of categories
ii) compute the high and low values of the variable, and the width, for example, if
    k=4 and high=60 and low=20 then width=(60-20)/4=10
iii) compute a variable whose value is the category the observations belongs in
iv) print the number of observations in each category
    make a nice histogram of the resulting counts (with nice title, etc)
SHOW how to apply the MACRO to a random sample of size 200 from a Poisson distribution
with mean 30. Make six equally spaced categories.
```

# NOTE #10: SQL (STRUCTURED QUERY LANGUAGE)

PROC SQL (available in SAS 6.06 or later versions) provides a versatile tool for data manipulations. This procedure can not replace SAS DATA step, but some complex tasks in Data step can be handled easily using SQL. SAS SQL uses the terminologies similar to other SQL language. Some basic terminology is given below.

Data step	Proc SQL
Dataset	Table
Variable	Column
Observation	Row
Merge	Join
Append	Union

The syntax of the procedure is

```
PROC SQL;
CREATE TABLE tablename AS
SELECT var,var,... /* use * to select all variables */
FROM sasdata, sasdata, ...
WHERE expression
ORDER BY var,var,...;
QUIT;
```

EXAMPLE WILL COME SOON.....