STAT 425 and STAT 625 Statistical Software

Lecture 6

More on Getting your Data to SAS And Working with your Data

GETTING YOUR DATA INTO SAS USING

PROC IMPORT

Reading Delimited Files with the Import Procedure

- A different way for SAS to read raw data is to use an **IMPORT** procedure, either in a Windows or UNIX environments.
- The general form of the procedure is as follows:

Proc Import Datafile='filename' Out=dat_set;

where the *filename* is file you want to read and *data-set* is the name of the SAS data set you want to create.

- If the file does not have the proper extension or it's of type DLM, then you must use the DBMS=option in the Proc Import statement.
- Also, use the REPLACE option if you already have a SAS data set with the name specified in the *out=option*, when you want to overwrite it.

Proc Import Datafile='filename' Out=dat_set

DBMS=identifier REPLACE;

Example:

```
title 'Reading Delimited files with Import procedure';
proc import datafile='J:\CLASSES\STAT46\list.csv' DBMS= CSV out=exo REPLACE;
run;

proc print data=exo;
```

- SAS will determine the file type by the extension of the file as follows:

Type of File	Extension	DBMS identifier
Comma-Delimited	.csv	CSV
Tab- Delimited	.txt	TAB
Delimiters other than comma or tabs		DLM

Optional Statements:

Some files need a few more instructions to be read correctly:

DATAROWS = n	Sart reading data in row n. Default is 1
DELINITER = ‘delimiter-character’	Delimiter for DLM. Default is space.
GETNAMES=NO	Do not get variable names from the first line of input file. Default is YES.
GUESSINGROWS=n	Use n rows to determine variable types. Default is 20.

Reading and writing Excel Spreadsheets

- One way to convert a spreadsheet into a SAS dataset is to convert it first into a comma separated values (CSV) file and then to read it using an infile statement.
- A second way is to convert it automatically if you have SAS/ACCESS interface to PC files, or SAS Entreprise Guide or SAS Studio.

Reading Excel Files with the IMPORT procedure

- With SAS/ACCESS Interface to PC Files, you can use the IMPORT procedure in the Windows or UNIX environments:

Proc Import datafile='filename' OUT=data-set DBMS=identifier REPLACE;

There are several DBMS identifiers that could be used, the most commonly used are: EXCEL, XLS, XLSX

Exemple:

```
title 'Import Excel spreadsheet';

proc import Datafile='J:\CLASSES\STAT46\BonusGift.xlsx' DBMS= xlsx out=Bonus ;
run;

proc print data=Bonus;
run;
```

Optional Statements:

- If there's more than one sheet in the file , then you can specify the sheet to read: *Sheet = "sheet-name";*
- If you want ot read specific cells in the sheet, you can specify a range . It could e a named range(if defined) or it could be specified by the upper-left and Lower –Right cells as follows:

Range = "sheet-name$UL:LR";

- By default the Import procedure takes the variable names from the first row of the spreadsheet, if you don't want this, SAS will name them V1, V2,….: *GETNAMES = no;*
- If in your spreadsheet you have a column containing both numeric and character values, then by default, the numbers will be converted to missing values, if you don't want this,then use:

MIXED = YES;

Working with Your DATA using

SAS Functions

- We have used simple expressions, using arithmetic operators to create new variables.
- SAS has numerous ways to transform variables and create new ones, and this includes:

Character string matching **Date and Time** **Descriptive Statistics** **Distance** **Financial**	**Macro** **Mathematical** **Probability** **Random Number** **State and Zip code** **Variable information**

- Functions perform a calculation on, or a transformation of, the arguments given in parenthesis following the function name. It's general form is as follows:

function-name (argument, argument, ….)

- All functions must have parenthesis even if they don't require any arguments

- Arguments are separated by commas, and can be:
  - ❖ variable names,
  - ❖constant values such as numbers or characters enclosed in quotation marks,
  - ❖Expressions.

Example:

In this example, we use the Mean function (not Proc Means): it calculates the mean of the non missing arguments.

```
libname Lecture6 'J:\CLASSES\STAT46';

DATA Lecture6.compet;
     informat J1 1.1
              J2 1.1
              J3 1.1
              J4 1.1
              J5 1.1
              J6 1.1
              J7 1.1;
infile 'J:\CLASSES\STAT46\divingLec6.csv' dsd;
input ID Dive J1 J2 J3 J4 J5 J6 J7;

* Calculate the average score for the first 3 Judges for each diver;

Score= mean(J1,J2, J3);

run;
title 'Mean Score';
proc print data=Lecture6.compet;
     var ID J1 J2 J3 Score;
     Format Score 3.1;
run;
```

Mean Score

Obs	ID	J1	J2	J3	Score
1	1	8.5	8.5	8.5	8.5
2	2	0.8	0.9	0.8	0.8
3	3	0.8	8.5	0.8	3.4
4	4	8.5	0.9	8.5	6.0
5	5	0.8	0.8	7.5	3.0

Performing Conditional Processing

In this part we will look into tools that allow programs ‘to make decisions’ based on data values.

Using IF – THEN Statements

IF condition Then action

The *condition* is an expression comparing one thing to another and the *action* is what SAS should do when the expression is true.

Example:

```
* Conditional Processing;
if Score lt 3 then Cat = 'D';
if Score ge 3 and Score lt 6 then Cat = 'C';
if Score ge 6 and Score lt 9 then Cat = 'B';
if Score ge 9  then Cat = 'A';
run;

title 'Listing of If-Then Conditioning';

proc print data=Lecture6.compet noobs;
run;
```

Listing of If-Then Conditioning

J1	J2	J3	J4	J5	J6	J7	ID	Dive	Score	Cat
8.5	8.5	8.5	8.5	0.9	8.5	8.5	1	1	8.50000	B
0.8	0.9	0.8	8.5	0.8	8.5	0.9	2	2	0.83333	D
0.8	8.5	0.8	0.8	8.5	8.5	8.5	3	3	3.36667	C
8.5	0.9	8.5	0.8	0.8	8.5	8.5	4	4	5.96667	C
0.8	0.8	7.5	0.8	0.8	8.5	0.8	5	5	3.03333	C

The terms of the statements are separated by a comparison operator which may be either symbolic operator or mnemonic operator. The decision to use one or another depends on programmer preferences and their availabilities on the key board. Some basic ones are:

Symbolic	Mnemonic	Meaning
=	EQ	equals
^= , or ~=,	NE	not equal
>	GT	greater than
<	LT	less than
>=	GE	greater or equal
<=	LE	Less or equal
	IN	Equal to one in a list

When a program has multiple IF – THEN statements, it can be improved by using ELSE IF after the first IF-THEN statement. Using this ELSE IF logic, when any of the IF statement is TRUE, all the following statements are not evaluated. This saves on processing time.

Example:

```
* Conditional Processing: IF-Else IF;

if Score lt 3 then Cate = '4';
else if Score ge 3 and Score lt 6 then Cate = '3';
else if Score ge 6 and Score lt 9 then Cate = '2';
else if Score ge 9  then Cate = '1';
```

Subsetting: The IF statement

To create a subset, from a raw data file or from an existing SAS data set, you can use:

IF expression ;

As you can notice there's no THEN in this statement. If the condition is TRUE, the program continues to the next statement, if the condition is FALSE, control returns to the top of the DATA step.

Example:

```
* Subsetting with IF;
If J1 ge 8;
run;
title 'Subsetting with IF';

proc print data=Lecture6.compet noobs;
     var ID J1 J2 J3 ;
run;
```

Subsetting with IF

ID	J1	J2	J3
1	8.5	8.5	8.5
4	8.5	0.9	8.5
8	8.5	8.5	0.8
9	9.5	0.9	0.9
10	8.5	8.5	0.8
14	8.5	8.5	8.5

Note that in the output table, there's only the values of J1 that are greater or equal to 8.5.

Using a SELECT statement for Logical Tests

A **SELECT** statement provides an alternative to a series of **IF** and **ELSE IF** statements:

Select (select-expression);
when (when-expression) ;
when (when-expression);
Otherwise (otherwise-expression);
End;

In these statements, the select-expression is compared to each of the *when-expressions*.

- If the comparison is true, the statement following the when-expression is executed and control skips to the end of the SELECT group.
- If the comparison is false, then the comparison is carried to the next when-expression.
- If no comparison is true, the expression following OTHERWISE statement is executed.
- Note that the otherwise-expression can be a null statement.
- An OTHERWISE statement must be there for the program to terminate

- Example:

```
* Using a SELECT statement;
Select ;
When (Score lt 3)   Cat = 'D';
When (Score lt 6)   Cat = 'C';
When ( Score lt 9)   Cat = 'B';
Otherwise Cat = 'A';
end;
run;
title 'Using a Select Statement';
proc print data=Lecture6.compet;
     var ID J1 J2 J3 Score Cat;
     Format Score 3.1;
run;
```

Using a Select Statement

Obs	ID	J1	J2	J3	Score	Cat
1	1	8.5	8.5	8.5	8.5	B
2	2	0.8	0.9	0.8	0.8	D
3	3	0.8	8.5	0.8	3.4	C
4	4	8.5	0.9	8.5	6.0	C
5	5	0.8	0.8	7.5	3.0	C
6	6	5.5	5.5	0.5	3.8	C
7	7	0.8	0.8	0.8	0.8	D
8	8	8.5	8.5	0.8	5.9	C

When you want to to test several conditions, you can use the **IN** operator, as follows:

IF X IN ('A', 'B', 'C') THEN… or IF X IN ('A' 'B' 'C') THEN…

The conditions in the parenthesis can be separated by commas or spaces.

The WHERE Statement

- Another way to subset your data, if you are reading data from a SAS data set, is to use a **WHERE** statement instead of AN IF statement.
- The advantages:
  - ❖there is a larger choice of operators that could be used.
  - ❖The WHERE statement could be used in a SAS procedure to subset the data being processed, which cannot be used with an IF statement.

Using Boolean Logic: AND, OR and NOT

It is possible to specify multiple conditions using the keywords: AND and OR:

IF condition AND condition THEN action

It is possible to use multiple Boolean operators and also it's possible to use parenthesis to give the operator precedence:

IF X AND (Y OR Z) THEN action

In this case, the OR operation is performed before the AND operation.

If instead you want to perform the AND operation first, you may use the parenthesis as follows:

IF (X AND Y) OR Z THEN action

The NOT operator has the highest precedence, it's performed before AND or OR:

IF (X AND NOT Y) OR Z THEN … is equivalent to*: IF (X AND (NOT Y)) OR Z THEN…*

CAUTION when using OR:

IF X=3 OR 4 THEN ….

In this statement, there is **X=3** on one side of OR and **4** on the other. In SAS, any value other than **0** or **missing** is true. So **4** is evaluated as TRUE and the statement **X=4** or **4** is always true.

- Like the comparison operators, the AND and OR symbols may be symbolic or mnemonic:

Symbolic	Mnemonic	Meaning
&	AND	All comparisons must be true.
| , !	OR	Only one comparison must be true

Example:

```
title 'Using a Where, IN and Boolean';

proc print data=Lecture6.compet;
    Where (Score ge 8 and Cat in ('A', 'B')) or (Score lt 7 and Cat in ('C','D'));
    var ID J1 J2 J3 Score Cat;
    Format Score 3.1;
run;
```

Using a Where, IN and Boolean

Obs	ID	J1	J2	J3	Score	Cat
1	1	8.5	8.5	8.5	8.5	B
2	2	0.8	0.9	0.8	0.8	D
3	3	0.8	8.5	0.8	3.4	C
4	4	8.5	0.9	8.5	6.0	C
5	5	0.8	0.8	7.5	3.0	C
6	6	5.5	5.5	0.5	3.8	C
7	7	0.8	0.8	0.8	0.8	D
8	8	8.5	8.5	0.8	5.9	C

Working with SAS Dates

- We have seen already some features about how SAS works with Dates.
- A SAS date is a numeric value equal to the number of days since January 1st,1960. In the table below there's some dates and their respective date values:

Date	SAS Date Value	Date	SAS Date Value
January 1, 1960	**0**	January 1, 1961	366
January 1, 1959	-365	January 1, 2020	21915

SAS has special tools to work with dates:

INFORMATS: to read variables that are dates, you need to use formatted style input. SAS has a variety of informats for reading dates in many different forms. All of thses informats convert the data into a number equal to the number of days since January 1st,1960. EXEMPLE:

Input BirthDate DATE10. ;

Setting the default century for input: When SAS sees a date such as: 10/09/18 it can't decide which year it is: 2018? 1918? 2118? 1818?

So, the system option: UEARCUTOFF= specifies the first year of a hundred-year span for SAS to use:

Options YEARCUTOFF = 1950;

- This statement tells SAS that this is a statement to interpret the data in two digit as occurring between 1950 and 2049.

Dates in SAS expressions:

- Once a variable has been read by SAS and has been properly informatted, then it can be read as any numeric variable and can be used in arithmetic expressions.

Example:

```
Age_ATHIRE= (Hire_Date - Birth_Date)/365.25;
```

In this example we use a difference in between the date variables Birth_Date and Hire_Date and then we divide by 365.25 to account for the leap year.

- Then Age_ATHIRE is formatted to display the age without decimals, as is shown in the table below:

```
proc print data=Lecture6.saleforcedata noobs;
      var ID Birth_Date Hire_Date Age_ATHIRE ;
format   Birth_Date ddmmyy10.
         Hire_Date ddmmyy8.
         Age_ATHIRE 2.0;

run;
```

ID	Birth_date	Hire_Date	Age_ATHIRE
120102	11/08/1973	01/06/93	20
120103	22/01/1953	01/01/78	25
120121	02/08/1948	01/01/78	29
120122	27/07/1958	01/07/82	24
120123	28/09/1968	01/10/89	21

Functions:

It’s possible to perform a number of handy operations with SAS date functions.

Example: In the following example, we use date functions to calculate the age at hire, again:

```
AAH=INT(YRDIF(Birth_Date, Hire_Date));
```

We used the function INT: to return an integer and YRDIF to calculate the difference between the two dates.

- As you can see, there's no special formatting for AAH because it will return just an integer that is the age at hire, as you can see in the table.

```
proc print data=Lecture6.saleforcedata noobs;
      var ID Birth_Date Hire_Date AAH ;
format   Birth_Date ddmmyy10.
         Hire_Date ddmmyy8.;
run;
```

ID	Birth_date	Hire_Date	AAH
120102	11/08/1973	01/06/93	19
120103	22/01/1953	01/01/78	24
120121	02/08/1948	01/01/78	29
120122	27/07/1958	01/07/82	23

Formats:

If you print a SAS date value, SAS will by default print the actual number of days since Jan,1st,1960. This number is not very meaningful to mostly everybody. For this SAS has a variety of formats to transform it in a more meaningful disply.

Example:

In the following example we have some banking data in which the date values have been read by SAS.

You will notice in the “date” variable the difference between the display of date in the first table, when the formatting of the data was not specified in the proc print step and the second table, when the formatting of the data was specified:

```
Data bankingdata;
     input ID date mmddyy10. gender $ credit;
datalines;
001 10/21/1955 M    1145
002 11/18/2001 F   18722
003 05/07/1944 M  123.45
;
run;

title 'Without date Format';
proc print data=bankingdata  noobs;
run;

title 'With date Format';
proc print data=bankingdata  noobs;
format date ddmmyy8.;
run;
```

Without date Format

ID	date	gender	credit
1	-1533	M	1145.00
2	15297	F	18722.00
3	-5717	M	123.45

With date Format

ID	date	gender	credit
1	21/10/55	M	1145.00
2	18/11/01	F	18722.00
3	07/05/44	M	123.45