An Introduction to SAS-Lecture 2

Konstantinos Fokianos
University of Cyprus

Reading data arranged in columns

SAS provides two methods of reading data values that are uniformly aligned in columns: column input and formatted input. Both provide the ability to read data from fixed locations in the input record and both expect to find data in those locations.

```
DATA COLINPUT;
   INPUT ID 1 HEIGHT 2-3 WEIGHT 4-6 GENDER $ 7 AGE 8-9;
DATALINES;
168144M23
278202M34
362 99F37
461101F45
;

PROC PRINT;
   TITLE 'Example 4';
RUN;
```

Reading data arranged in columns

- In this example, we do not need to leave any space between data values.
- The column specifications in the `INPUT` statement provide instruction as to where to find the data values.
- Notice that the value `99` (WEIGHT variable) is placed in columns `5-6` and not `4-5`.
- However SAS will read the right numbers because the columns `4-6` have been specified for this variable.

Readable Programs

```
DATA COLINPUT;
   INPUT ID       1
      HEIGHT    2-3
      WEIGHT    4-6
      GENDER  $   7
      AGE       8-9;
DATALINES;
168144M23
278202M34
362 99F37
461101F45
;

PROC PRINT DATA=COLINPUT;
   TITLE 'Example 4';
RUN;
```

Reading Selected Variable from your Data

When, you read data in columns, you have the freedome to skip any columns you wish and read the variables of interest.

```
DATA COLINPUT;
   INPUT ID       1
      AGE       8-9;
DATALINES;
168144M23
278202M34
362 99F37
461101F45
;

PROC PRINT DATA=COLINPUT;
   TITLE 'Example 5';
RUN;
```

Reading Values in Different Order

When using column or formatted input you can read data fields in any order you want. For instance

```
DATA COLINPUT;
   INPUT  AGE      8-9
          ID       1
          WEIGHT   4-6
          HEIGHT   2-3
          GENDER $ 7;
DATALINES;
168144M23
278202M34
362 99F37
461101F45
;

PROC PRINT;
   TITLE 'Example 6';
RUN;
```

Reading Column Data that Require Informats

Instead of using starting and ending column numbers to describe the location of the data, you can use the

- the starting column number
- the length of the data value
- and a SAS informat.

Variable Name	Starting Column	Length	Format	Description
ID	1	3	Numeric	Subject ID
GENDER	4	1	Character	M=Male, F=Female
AGE	9	2	Numeric	Age of Subject
HEIGHT	11	2	Numeric	Height in Inches
DOB	13	6	MMDDYY6	Date of Birth

Reading Column Data that Require Informats

```
DATA POINTER;
   INPUT @1  ID       3.
         @5  GENDER  $1.
         @7  AGE      2.
         @10 HEIGHT   2.
         @13 DOB      MMDDYY6.;
FORMAT DOB MMDDYY8.;
DATALINES;
101 M 26 68 012366
102 M 32 78 031460
103 F 45 62 112647
104 F 22 66 080170
;

PROC PRINT;
   TITLE 'Example 7';
RUN;
```

Reading Column Data that Require Informats

- The `@` character is called column pointer: it indicates the starting column for an action. When appear before a variable name in an INPUT statement, it informs SAS to go to a ceratin column. Then, you can use an informat to tell the program how to read the data.
- There are many types of SAS informats. Numeric variables use an informat of the form *w.d* where *w* is the width of the field (number of columns) and *d* is the number of places to the right of the decimal point in the value. When *d* is omitted, it is assumed to be zero.
- There is a variety of data infromats. Here we use `MMDDYY6.` to tell SAS to read from 6 columns: the first two being the month, the next two the day of the month and the last two the year. If the values contain special characters (slash or dash), then use `MMDDYY8.`.
- **All SAS informats contain a period (.). DO NOT OMIT IT!!!!**

Using Informat Lists and Relative Pointer Controls

- Read a data set which contains ID, answers to 10 questions, Height and Age.
- Here is a better way

```
DATA SHORTWAY;
   INPUT ID 1-3
      @4 (Q1-Q5)(1.)
      @9 (Q6-Q10 HEIGHT AGE)(2.);
DATALINES;
101113241O161415156823
102143321212141316722 1
103233421414121210662 8
104155321616131412662 2
;

PROC PRINT;
   TITLE 'Example 8.1';
RUN;
```

Using Informat Lists and Relative Pointer Controls

```
DATA LONGWAY;
   INPUT ID      1-3
         Q1      4
         Q2      5
         Q3      6
         Q4      7
         Q5      8
         Q6      9-10
         Q7      11-12
         Q8      13-14
         Q9      15-16
         Q10     17-18
         HEIGHT 19-20
         AGE     21-22;
DATALINES;
1011132410161415156823
1021433212121413167221
1032334214141212106628
1041553216161314126622
;

PROC PRINT;
   TITLE 'Example 8';
RUN;
```

Using Informat Lists and Relative Pointer Controls

- After reading the value ID, five values are read for variables Q1, Q2, Q3, Q4 and Q5. They are all read with 1. informat. Then, we initiate a new list which has Q6-Q10, Height and Age with a 2. informat.
- An alternate coding for the previous example is the following

```
INPUT @1 (ID Q1-Q10 HEIGHT AGE)(3. 5*1 7*2);
```

SAS Log

Read the SAS Log!!!!
That is the place where you can find information about whether the program has run right or not. Please check the `LOST CARD` statement in the SAS Log (if it occurs).

```
DATA ERRORS;
   INPUT X 1-2
         Y 4-5;
DATALINES;
11 23
23 NA
NA 47
55 66
;
```

SAS Log

```
1
2    DATA ERRORS;
3       INPUT X 1-2
4             Y 4-5;
5    DATALINES;

NOTE: Invalid data for Y in line 7 4-5.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6-
7          23 NA
X=23 Y=. _ERROR_=1 _N_=2
NOTE: Invalid data for X in line 8 1-2.
8          NA 47
X=. Y=47 _ERROR_=1 _N_=3
NOTE: The data set WORK.ERRORS has 4 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.31 seconds
      cpu time            0.03 seconds


10   ;
```

SAS Log

When the system encounters an invalid value for a variable, it does numerous things:

- It tells you about its discovery
- It assigns a missing value

However you

- can either place a single `?` following a variable name to tell SAS to suppress this type of error message while continuing to print the offending line data to the log, or
- you can save more paper by placing `??` to suppress all the errors and offending lines.

SAS Log

```
DATA ERRORS;
       INPUT X ?? 1-2
             Y ?? 4-5;
    DATALINES;
    11 23
    23 NA
    NA 47
    55 66
    ;


 11    DATA ERRORS;
12           INPUT X ?? 1-2
13                 Y ?? 4-5;
14        DATALINES;

NOTE: The data set WORK.ERRORS has 4 observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.00 seconds


19        ;
```

Reading Data from External Files

Most of the times your data will be stored in an external file. The only changes that you need to do in your SAS program is to include an `INFILE` statement that identifies the location of the file with the data values:

```
DATA USCRIME;
       INFILE "REGRESSION.DAT";
 INPUT R AGE S ED EX0 EX1 LF M N NW U1 U2 W X;

 PROC PRINT DATA=USCRIME;
 RUN;
```

Reading Data from External Files

The other method to tell SAS where to look for your data is to create a fileref (file reference) by means of a `FILENAME` statement and then to refer to this fileref in the `INFILE` statement. For the previous example

```
FILENAME  TEST "regression.data";

DATA EXTERNAL;
     INFILE TEST;
     INPUT R AGE S ED EX0 EX1 LF M N NW U1 U2 W X;

 PROC PRINT DATA=USCRIME;
 RUN;
```

Reading Data from External Files

Reading the first 20 records of data

```
INFILE TEST  OBS=20;
```

Skipping the first 20 observations

```
INFILE TEST  OBS=21;
```

Reading a subset of data

```
INFILE TEST  FIRSTOBS=21 OBS=26;
```

Using IF-THEN ELSE Statements

Suppose that you have the variable `SCORE` given by

```
55, 65,74,76, 88,92,94,96, 98
```

and you want to create a new variable called `GRADE` such that it is 0, when the SCORE is less than 65, 1 when the score in greater of equal than 65 and less than 70, 2 when the score is greater or equal to 70 and less than 80, 3 when the score is greater or equal than 80 and less than 90 and 4 otherwise.

Using IF-THEN ELSE Statements

Here is the program that allows you to do that:

```
DATA GRADES;
     INPUT SCORE;
 DATALINES;
 55
 65
 74
 76
 88
 92
 94
 96
 98
 ;
DATA RECODE;
   SET GRADES;
   IF   0 LE SCORE LT 65 THEN GRADE=0;
   ELSE IF 65 LE SCORE LT 70 THEN GRADE=1;
   ELSE IF 70 LE SCORE LT 80 THEN GRADE=2;
   ELSE IF 80 LE SCORE LT 90 THEN GRADE=3;
   ELSE IF       SCORE GE 90 THEN GRADE=4;
PROC PRINT;
   TITLE 'Example 11';
RUN;
```

Using IF-THEN ELSE Statements

Style Issues: You can use instead

```
IF value_1 LE SCORE AND SCORE LT value_2 THEN GRADE=assigned value;
IF score GE value_1 AND SCORE LT value_2 THEN GRADE=assigned value;
```

Efficiency: The code could have been written without the `ELSE` statement but it would have taken longer to run.
Never Use

```
IF SCORE LT 65 THEN GRADE=0;  ****WRONG PROGRAMMING!!!!!
```

The problem is related to the existence of possible missing values.

Using SELECT Statements

An alternative to `IF_THEN/ELSE` coding is to use the select statement.

```
DATA RECODE;
   SET GRADES;
 SELECT;
      WHEN (0 LE SCORE LT 65)  GRADE = 0;
      WHEN (65 LE SCORE LT 70) GRADE = 1;
      WHEN (70 LE SCORE LT 80) GRADE = 2;
      WHEN (80 LE SCORE LT 90) GRADE = 3;
      WHEN (SCORE GE 90)       GRADE = 4;
   END;
PROC PRINT;
   TITLE 'Example 12';
RUN;
```

Using FORMATS to recode a variable

In the following you use a `FORMAT` statement in a `PROC` step to achieve the same grouping.

```
PROC FORMAT;
   VALUE SCOREFMT   0 - 64   = 'Fail'
                   65 - 70   = 'Low Pass'
                   70 - 80   = 'Pass'
                   80 - 90   = 'High Pass'
                   90 - HIGH = 'Honors';
RUN;

PROC FREQ DATA=GRADES;
   TITLE 'Example 13';
   TABLES SCORE;
   FORMAT SCORE SCOREFMT.;
RUN;
```

Using FORMATS to recode a variable

- The `PROC FORMAT` defines a new varaible such that any variable falling in the left hand side of equal sign is assigned the format of the right hand side.
- Use the `FORMAT` statement in the frequency distribution procedure to assign the format `SCOREFMT` to `SCORE`.

Using the PUT statement

Creating a new variable by means of others can be done with the `PUT` statement.

```
DATA NEW;
   SET GRADES;
   CATEGORY = PUT (SCORE,SCOREFMT.);
RUN;

PROC PRINT;
   TITLE 'Example 13a';
RUN;
```