greatlearning
Learning for Life

Managing Data in SAS

Neelesh Singh

greatlearning
Learning for Life

Module Content

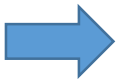
1. Modifying data with SET Statement
2. Sorting data
3. Managing Variables (Drop, keep and rename)
4. Subsetting data
   - Where/if, If then else, Looping, Logical Operators
   - SAS Arrays
5. Sorting Data
6. Retain
7. Transposing Data
8. Using Proc SQL

Creating variable with the SET statement

SET Statement is easiest way to create a variable

Obs	principal	rate	time
1	1000	0.01	2
2	3000	0.02	4

Expression:
Amount = Principal * (1+rate) ^{time}



Obs	principal	rate	time	Amount
1	1000	0.01	2	1020.10
2	3000	0.02	4	3247.30

```
Data Assets;
   input principal rate time ;
   cards;
   1000 0.01 2
   3000 0.02 4
   ;
RUN;
```

```
Data Interest;
set Assets;
Interest=principal * ((1+rate)**time);
RUN;
```

Stacking data sets using the SET statement

- The Number of observations in the new data set will equal to the sum of the number of observations in the old data sets
- The order of observations is determined by the order of the list of old data sets
- If one of the data set has a variables not contained in the other data sets, then observations from the other data sets will have missing values for that variable.

```
DATA Data_New;
        SET Data1 Data2 Data3;
RUN;
```

Stacking data sets using the SET statement

```
DATA East;
  INPUT Entrance $ PassNumber PartySize Age;
  cards;
  E 43 3 27
  E 44 3 24
  E 45 3  2
  ;
RUN;
DATA West;
  INPUT Entrance $ PassNumber PartySize Age Lot;
  Cards;
  W 21 5 41 1
  W 87 4 33 3
  W 65 2 67 1
  W 66 2 7 1
  ;
RUN;
DATA both;
  SET East West;
RUN;
```

Sorting data using BY Statement

Stacking dataset may unsort the data. Using BY statement to sort the final data by the desired variable

```
Data Data_sorted;
        SET DATA_Unsorted;
        BY VARIABLE;
RUN;
```

Managing variables

greatlearning
Learning for Life

- Data set option help to modify the current datasets; DataSet options are put in between parentheses directly following the data set name. Here are some commonly used data set options:

Data set options	Functions
Keep =variable-list	Which variables to keep
Drop=variable-list	Which variables to drop
Rename=(oldvar=newvar)	Rename oldvar to newvar
Firstobs=n	To start reading at observation n
Obs=n	To stop reading at observation n
IN=new-var-name	Create a temporary variable for tracking whether that data set contributed to the current observation

greatlearning
Learning for Life

Managing variables (Keep, Drop and Rename)

```
Data new;
   input a b c @@@;
   datalines;
   1 2 3 4 5 6 7 8 9
  ;
run;

data new1;
   set new (keep=a drop = b
            rename= (c=d) );
run;
```

Managing Observations

- **To create a new smaller dataset with a certain number of rows from an old bigger one.**
  - Example : To create a dataset NEW having only the first 5 rows of the dataset OLD.

```
data NEW;
                set OLD (OBS = 5);          < Data Manipulation Statement(s)>;
run;
```

```
data NEW;
                set OLD (FIRSTOBS = 5);     < Data Manipulation Statement(s)>;
run;
```

```
data NEW;
                set OLD (FIRSTOBS = 5 OBS = 8);          < Data Manipulation Statement(s)>;
run;
```

Sub setting data (using WHERE statement)

- What does the Where statement do ?
  - Selects observations from SAS datasets that meet a particular condition (arithmetic or logical)
- Operands used in Where Expressions
  - Operands include
    - constants
      - E.g. where score > 50
    - time & date values
      - where date >= “01JAN2003”d
    - values of variables that are obtained from the SAS datasets
      - where state = “Karnataka”
    - values created within the WHERE expression itself

Obs	Year	Product	Sales
1	2017	Cement	758
2	2017	Timber	254
3	2017	Paper	232
4	2017	Steel	456
5	2018	Cement	421
6	2018	Timber	546
7	2018	Paper	123
8	2018	Steel	715
9	2019	Cement	564
10	2019	Timber	586
11	2019	Paper	825
12	2019	Steel	625

Obs	Year	Product	Sales
1	2018	Cement	421
2	2018	Timber	546
3	2018	Paper	123
4	2018	Steel	715

SAS Code –

```
Data data1;
    set testdata;
    where Year =2018 ;
run;
```

Operators

greatlearning
Learning for Life

- Operators used in Where Expressions
  - Arithmetic
    - Multiplication (*), Division (/), Addition (+), Subtraction (-)
  - Comparison
    - Equal to (= or EQ), Not Equal to (^= or NE), Greater than (> or GT),
    - Less than (< or LT), Greater than or equal to (>= or GE),
    - Less than or equal to (<= or LE)
    - IN – equal to one of a list
  - Logical (Boolean)
    - AND, OR, NOT
  - Where Expression only
    - BETWEEN-AND -- an inclusive range
    - CONTAINS (or ?) -- a character string
    - IS NULL or IS MISSING -- missing values
    - LIKE -- match patterns

Sub setting data (using WHERE statement)

- Using multiple conditions in WHERE statement

```
Data data2;
set testdata;
where Year >2017 AND (Sales between 400 and 700) AND Product = 'Timber' ;
run;
```

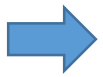
Obs	Year	Product	Sales
1	2018	Timber	546
2	2019	Timber	586

Sub setting data (using IF statement)

- If Statement Continues processing only those observations that meet the condition

```
Data data1;
set testdata;
IF Year =2018 ;
run;
```

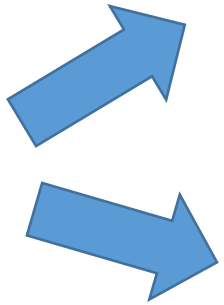
Obs	Year	Product	Sales
1	2017	Cement	758
2	2017	Timber	254
3	2017	Paper	232
4	2017	Steel	456
5	2018	Cement	421
6	2018	Timber	546
7	2018	Paper	123
8	2018	Steel	715
9	2019	Cement	564
10	2019	Timber	586
11	2019	Paper	825
12	2019	Steel	625



Obs	Year	Product	Sales
1	2018	Cement	421
2	2018	Timber	546
3	2018	Paper	123
4	2018	Steel	715

- Also IF statement can be used to create multiple datasets using the IF & the OUTPUT conditions.

Obs	Year	Product	Sales
1	2017	Cement	758
2	2017	Timber	254
3	2017	Paper	232
4	2017	Steel	456
5	2018	Cement	421
6	2018	Timber	546
7	2018	Paper	123
8	2018	Steel	715
9	2019	Cement	564
10	2019	Timber	586
11	2019	Paper	825
12	2019	Steel	625



Obs	Year	Product	Sales
1	2017	Cement	758
2	2017	Timber	254
3	2017	Paper	232
4	2017	Steel	456

Obs	Year	Product	Sales
1	2018	Cement	421
2	2018	Timber	546
3	2018	Paper	123
4	2018	Steel	715

```
Data data2017 data2018;
set testdata;
IF Year = 2017 THEN output data2017 ;
ELSE IF Year = 2018 THEN output data2018;
run;
```

Using IF and WHERE

- Differences between the WHERE & IF statement
  - The subsetting IF & WHERE statements are not equivalent. The two work differently & produce different output datasets in some cases. The most important differences are –
    - The subsetting IF is less efficient since it reads all the observations in the input data set before applying the condition specified in the statement
    - When the subsetting IF statement is used in the MERGE statement, SAS selects observations after the current observations are combined
      - When the WHERE statement is used with the MERGE statement, SAS applies the selection criteria to each input dataset before combining the current observations
    - The IF statement can select observations from an existing SAS data set or from raw data read with an INPUT statement
      - WHERE statement can only select observations from existing SAS data sets, but unlike IF, it can be used with SAS procedures
    - The WHERE statement cannot be executed conditionally, but the IF statement can

More on IF THEN ELSE

Example1 (Independent Statements with Single Condition)

```
data Example1;
    set TRANSACTIONS;
      IF TranAmt < 100 THEN Tag = 'Small Ticket'; ELSE Tag = 'Big Ticket';
Run;
```

Example2 (Independent Statements with Complex Condition)

```
data Example2;
    set TRANSACTIONS;
      IF TranAmt < 100 and TranCat = '001' THEN Tag = 'Small Sale';
      ELSE Tag = 'Big Sale';
run;
```

Example3 (Continuous Statements forming a Condition Loop)

```
data Example3;
    set TRANSACTIONS;
        IF TranCat = '001' THEN Type = 'Sale';
      ELSE IF TranCat = '002' THEN Type = 'Returns';
            run;
```

More on IF THEN ELSE

greatlearning
Learning for Life

Example4 (Nested Statements for Complex Condition)

```
data Example4;
    set TRANSACTIONS;
      IF TranAmt > 0 THEN
                    IF TranCat = '001' THEN Tag = 'Sale';
                ELSE IF TranCat = '002' THEN Tag = 'Returns';
                ELSE Tag = 'Other'
      ELSE
                Tag = 'Invalid Amount';
run;
```

Example5 (Block Statements)

```
data Example5;
    set TRANSACTIONS;
      IF TranCat = '001' and TranAmt > 1000 THEN
      DO;
                Type = 'Sale';
                Tag = 'Big Ticket';
      END;
run;
```

SELECT-WHEN-OTHERWISE

greatlearning
Learning for Life

Using SELECT - WHEN - OTHERWISE Clause
Used to direct the flow depending on the value of a variable :

Example1 (Simple SELECT Statement)

```
data Example1;
    set TRANSACTIONS;
      SELECT (TranCat);
                WHEN ('001')      Type = 'Sales';
                WHEN ('002')      Type = 'Returns';
                OTHERWISE         Type = 'Other';
      END;
run;
```

Example2 (SELECT Statement with DO Blocks)

```
data Example2;
    set TRANSACTIONS;
      SELECT (TranCat);
                WHEN ('001') DO; Type = 'Sales';   Amt = TranAmt; END;
                WHEN ('002') Do; Type = 'Returns'; Amt = TranAmt; END;
                OTHERWISE    DO; Type = 'Other';   Amt = TranAmt;
END;
      END;
run;
```

Using DO LOOPS-1

greatlearning
Learning for Life

- DO Statement
  - Designates a group of statements to be executed as a unit
- Syntax
  - **DO;**
    …. more SAS statements ….
    END;

- ITERATIVE DO Statement
  - Executes statements between DO and END repetitively based on the value of an index variable
- Syntax
  - **DO** *index-variable = specification-1 <, … specification-n>;*
    …. more SAS statements ….
    END;
  - index-variable
    - names a variable whose value governs execution of the DO loop
    - unless one specifies to drop it, the index variable is included in the dataset that is being created
  - specification
    - denotes an expression or a series of expressions in this form
      start <TO stop> <BY increment>

Using DO LOOPS

greatlearning
Learning for Life

- Syntax....
  - start
    - specifies the initial value of the index variable
  - TO stop
    - optionally specifies the ending value of the index variable
      - *Stop* must be a number or an expression that yields a number
  - BY increment
    - optionally specifies a positive or negative number (or an expression that yields a number) to control the incrementing index-variable

- Example :
  - SAS Code --

```
DATA T1;
   num = 0;
   DO  i = 1 TO 5;
      num = num + 1;
      output;
   END;
RUN;
```

NUM	I
1	1
2	2
3	3
4	4
5	5

T1

Using DO LOOPS

greatlearning
Learning for Life

Using DO iterative blocks

The **DO** blocks can also be used for iterative programming :

Example1 (Iterative DO blocks)

```
data Example1;
    set TRANSACTIONS;
            DO <Condition>;
                    SAS Data Step Statement(s);
            END;
run;
```

Examples of condition statements that could be used in DO iterative loops

- DO month='JAN','FEB','MAR';
- DO count=2,3,5,7,11,13,17;
- DO i=var1, var2, var3;
- DO i=1 to 10;
- DO i=1 to k-1, k+1 to n;
- DO i=n to 1 **BY** -1;

Using DO (WHILE and UNTIL)

- DO UNTIL Statement
  - Executes statements in a DO loop repetitively until a condition is true
- Syntax
  - **DO UNTIL** *(expression);*
    …. more SAS statements ….
    END;
  - expression
    - any SAS expression, enclosed in parentheses
    - at least one SAS expression must be specified
- Details
  - The expression is evaluated at the bottom of the loop after the statements in the DO loop have been executed
    - If the expression is true, the DO loop does not iterate again
    - The DO loop always iterate at least once

- DO WHILE Statement
  - Executes statements repetitively while a condition is true
- Syntax
  - **DO WHILE** *(expression);*
    …. more SAS statements ….
    END;
  - expression
    - any SAS expression, enclosed in parentheses
    - at least one SAS expression must be specified
- Details
  - The expression is evaluated at the top of the loop before the statements in the DO loop are executed
  - If the expression is true, the DO loop iterates
  - If the expression is false the first time it is evaluated, the DO loop does not iterate even once

Using DO (WHILE and UNTIL)

- ✓ Example :
  - ✓ SAS Code --

```
DATA T1;
   n = 0;
   DO WHILE(n < 8);
      n + 1;
      output;
   END;
  RUN;
```

The loop will be iterated 8 times, i.e., for n = 0, 1, 2, … 7

N
1
2
3
4
5
6
7
8

T1

Using DO (WHILE and UNTIL)

- ✓ Example :
  - ✓ SAS Code --

```
DATA T1;
   n = 0;
   DO WHILE(n < 8);
      n + 1;
      output;
   END;
  RUN;
```

The loop will be iterated 8 times, i.e., for n = 0, 1, 2, … 7

N
1
2
3
4
5
6
7
8

T1

Missing Value Imputation

Without Using Arrays

```
data test;
input a1 a2 a3;
cards;
12 . 34
. 34 23
2 2 .
;
data test1;
set test;
if a1=. then a1=0;
if a2=. then a2=0;
if a3=. then a3=0;
run;
```

Using Arrays

```
data test2(drop=i);
set test;
array t{3} a1 a2 a3;
do i=1 to 3;
if t{i}=. then t{i}=0;
end;
run;
```

Basic steps to be followed while writing a SAS Array

- **Declare the name of the array with the number of elements: t{3} in the example above**
- **Declare the elements (variables) in the array : a1, a2 and a3 in the example above**
- **Do statement indicating the elements for which the array processing needs to be done: do I= 1 to 3 in the example above**
- **Actual Data processing that needs to be done**
- **Close the do loop with an end statement**

Missing Value Imputation

	a1	a2	a3
1	12	.	34
2	.	34	23
3	2	2	.

	a1	a2	a3
1	12	0	34
2	0	34	23
3	2	2	0

```
data test2(drop=i);
set test;
array t{3} a1 a2 a3;
do i=1 to 3 by 2;
if t{i}=. then t{i}=0;
end;
run;
```

Modifying the Do statement to execute the array for alternative elements only

```
data test2(drop=i);
set test;
array t{3} a1 a2 a3;
do i=2,3;
if t{i}=. then t{i}=0;
end;
run;
```

Modifying the Do statement to execute the array for selected elements only

Modifying the Do statement to execute the array for alternative elements only

	a1	a2	a3
1	12	.	34
2	0	34	23
3	2	2	0

Modifying the Do statement to execute the array for selected elements only

VIEWTABLE: Work.Test2

	a1	a2	a3
1	12	0	34
2	.	34	23
3	2	2	0

```
data test2(drop=i);
set test;
array t{3} a1 - a3;
do i=1 to 3;
if t{i}=. then t{i}=0;
end;
run;
```

Declaring elements in an array using a – when the elements follow a particular sequence

```
data test2(drop=i);
set test;
array t{*} a1 - a3;
do i=1 to 3;
if t{i}=. then t{i}=0;
end;
run;
```

Using the * instead of declaring the exact number of dimensions of the array

Declaring elements in an array using a – when the elements follow a particular sequence

	a1	a2	a3
1	12	0	34
2	0	34	23
3	2	2	0

Using the * instead of declaring the exact number of dimensions of the array

VIEWTABLE: Work.Test2

	a1	a2	a3
1	12	0	34
2	0	34	23
3	2	2	0

The DIM function returns the no of elements in an array

```
data test2(drop=i);
set test;
array t{*} a1 - a3;
do i=1 to dim(t);
if t{i}=. then t{i}=0;
end;
run;
```

- In the example above, the DIM function returns a value of 3 and the array processing happens for elements a1, a2 and a3
- This function is useful when dealing with large number of variables and the exact number of such variables is not known

The DIM function returns the no of elements in an array

	a1	a2	a3
1	12	0	34
2	0	34	23
3	2	2	0

- In the example above, the DIM function returns a value of 3 and the array processing happens for elements a1, a2 and a3
- This function is useful when dealing with large number of variables and the exact number of such variables is not known

```
data test;
input a1 a2 a3 b1 b2 b3;
cards;
12 . 34 12 65 23
. 34 23 7 . 19
2 2 . 122 . .
;
data test1(drop=i );
set test;
array t1{3} a1-a3;
array t2{3} b1-b3;
array t3{3} z1-z3;
do i=1 to 3;
if t1{i}=. then t1{i}=0;
if t2{i}=. then t2{i}=999;
t3{i}=sum(t1{i},t2{i});
end;
```

Missing Value Treatment as well as simultaneous creation of new variables using the array statement

Missing Value Treatment as well as simultaneous creation of new variables using the array statement

	a1	a2	a3	b1	b2	b3
1	12	.	34	12	65	23
2	.	34	23	7	.	19
3	2	2	.	122	.	.

	a1	a2	a3	b1	b2	b3	z1	z2	z3
1	12	0	34	12	65	23	24	65	57
2	0	34	23	7	999	19	7	1033	42
3	2	2	0	122	999	999	124	1001	999

Leave Statement

```
data test;
input a1 a2 a3 ;
cards;
12 34 .
. 44 55
23 . 33
;
data test1(drop=i);
set test;
array ss{*} a1-a3 b1-b3;
do i=1 to 3 ;
if ss{i} eq . then leave;
ss{i+3}=2*ss{i};
end;
run;
```

The Leave statement stops processing all further obs in the row once the condition is satisfied

Stop Statement

```
data test;
input a1 a2 a3 ;
cards;
12 12 23
. 44 55
23 . 33
;

data test1(drop=i);
set test;
array ss{*} a1-a3 b1-b3;
do i=1 to 3 ;
if ss{i} eq . then stop;
ss{i+3}=2*ss{i};
end;
run;
```

The Stop statement stops processing all further obs once the condition is satisfied

Leave Statement

	a1	a2	a3
1	12	34	.
2	.	44	55
3	23	.	33

	a1	a2	a3	b1	b2	b3
1	12	34	.	24	68	.
2	.	44	55	.	.	.
3	23	.	33	46	.	.

The Leave statement stops processing all further obs in the row once the condition is satisfied

Stop Statement

	a1	a2	a3
1	12	12	23
2	.	44	55
3	23	.	33

	a1	a2	a3	b1	b2	b3
1	12	12	23	24	24	46

The Stop statement stops processing

all further obs once the condition is satisfied

```
data test;
input a1 $ a2 $ a3 $;
cards;
12 . 34
. 34 23
2 2 .
;
data test2(drop=i);
set test;
array t{3}$ a1 a2 a3;
do i=1 to 3;
if t{i}=" " then t{i}='XX';
end;
run;
```

Include a $ sign after declaring the element. Rest of the array processing is similar to that in case of numeric variables

Note: *You cannot process both a numeric variable as well as a character variable in the same array. All elements in an array need to be either wholly numeric or wholly character*

Include a $ sign after declaring the element. Rest of the array processing is similar to that in case of numeric variables

VIEWTABLE: Work.Test

	a1	a2	a3
1	12		34
2		34	23
3	2	2	

	a1	a2	a3
1	12	.X	34
2	.X	34	23
3	2	2	.X

Note: You cannot process both a numeric variable as well as a character variable in the same array.

All elements in an array need to be either wholly numeric or wholly character

Numeric

```
data test;
input a1 a2 $ a3;
cards;
12 . 34
. 34 23
2 2 .
;
data test2(drop=i);
set test;
array t{*} _numeric_;
do i=1 to dim(t);
if t{i}=. then t{i}=0;
end;
run;
```

The _Numeric_ option picks up all numeric variables from the data set and processes them

Character

```
data test;
input a1 $ a2 a3 $;
cards;
12 . 34
. 34 23
2 2 .
;
data test2(drop=i);
set test;
array t{*}$ _character_;
do i=1 to dim(t);
if t{i}=" " then t{i}='XX';
end;
run;
```

The _Character_ option picks up all character variables from the data set and processes them

greatlearning
Learning for Life

Numeric

VIEWTABLE: Work.Test

	a1	a2	a3
1	12		34
2	.	34	23
3	2	2	.

VIEWTABLE: Work.Test2

	a1	a2	a3
1	12		34
2	0	34	23
3	2	2	0

The _Numeric_ option picks up all numeric variables from the data set and processes *them*

Character

	a1	a2	a3
1	12	.	34
2		34	23
3	2	2	

	a1	a2	a3
1	12	.	34
2	⋈	34	23
3	2	2	⋈

The _Character_ option picks up all character variables from the data set and processes them

```
data test;
informat a1 date9.;
informat a2 date9.;
informat a3 date9.;
format a1 date9.;
format a2 date9.;
format a3 date9.;
input a1 a2 a3;
cards;
01Jan2000 01Feb2003 23Dec2002
;
data test1(drop=i);
set test;
array dt{3} a1-a3;
array intvl{3} x1-x3;
do i=1 to 3;
intvl{i}=intck('month',dt{i},'31Dec2004'd);
end;
run;
```

Since Date variables are in effect numeric variables, therefore the treatment in case of Array processing is exactly same as that of numeric variables

	a1	a2	a3
1	01JAN2000	01FEB2003	23DEC2002

	a1	a2	a3	x1	x2	x3
1	01JAN2000	01FEB2003	23DEC2002	59	22	24

Since Date variables are in effect numeric variables, therefore the treatment in case of Array processing is exactly same as that of numeric variables

```
data test;
input a1 - a3;
cards;
99 77 88
76 90 65
102 12 34;
run;

data test1(drop=i);
set test;
array tt{3} _temporary_ (90,80,70);
array tt1{3} a1-a3;
array tt2{3} ind1-ind3;
do i=1 to 3;
if tt1{i} >= tt{i} then tt2{i}=tt1{i};else tt2{i}=0;
end;
run;
```

The _temporary_ options allows assigning values to elements in an array in order to compare with values of other variables

VIEWTABLE: Work.Test

	a1	a2	a3
1	99	77	88
2	76	90	65
3	102	12	34

The _temporary_ options allows assigning values to elements in an array in order to compare with values of other variables

Arrays can be referenced with a lower bound and upper bound

```
data test;
input a1 a2 a3;
cards;
12 . 34
. 34 23
2 2 .
;
data test2(drop=i);
set test;
array t{11:13} a1 a2 a3;
do i=11 to 13;
if t{i}=. then t{i}=0;
end;
run;
```

In the example above 11 is the lower bound and 13 the upper bound

t(11) refers to var a1 and t(13) refers to the var a3

Note: By default the lower bound is 1

Arrays can be referenced with a lower bound and upper bound

VIEWTABLE: Work.Test

	a1	a2	a3
1	12	.	34
2	.	34	23
3	2	2	.

	a1	a2	a3
1	12	0	34
2	0	34	23
3	2	2	0

In the example above 11 is the lower bound and 13 the upper bound

t(11) refers to var a1 and t(13) refers to the var a3

Note: By default the lower bound is 1

Arrays can be referenced with a lower bound and upper bound

```
data test;
input a1 a2 a3;
cards;
12 . 34
. 34 23
2 2 .
;
data test2(drop=i);
set test;
array t{11:13} a1 a2 a3;
do i=lbound(t) to hbound(t);
if t{i}=. then t{i}=0;
end;
run;
```

***Note** : In this case if we had used the Dim Function instead of the Hbound function, then it would have returned a value of 3 and the array would not have executed. Therefore, in this case, we have to use the Function Hbound instead of Dim*

Arrays can be referenced with a lower bound and upper bound

VIEWTABLE: Work.Test

	a1	a2	a3
1	12	.	34
2	.	34	23
3	2	2	.

	a1	a2	a3
1	12	0	34
2	0	34	23
3	2	2	0

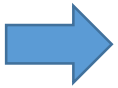
***Note** : In this case if we had used the Dim Function instead of the Hbound function, then it would have returned a value of 3 and the array would not have executed. Therefore, in this case, we have to use the Function Hbound instead of Dim*

SORTING DATA – PROC SORT

greatlearning
Learning for Life

PROC SORT Orders SAS dataset observations by the values of one or more character or numeric variables

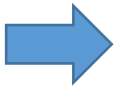
Obs	Year	Product	Sales
1	2017	Cement	758
2	2017	Timber	254
3	2017	Paper	232
4	2017	Steel	456



Obs	Year	Product	Sales
1	2017	Paper	232
2	2017	Timber	254
3	2017	Steel	456
4	2017	Cement	758

```
proc sort data =test  out=test1;
by a ;
run;
```

Default option is ascending



Obs	Year	Product	Sales
1	2017	Cement	758
2	2017	Steel	456
3	2017	Timber	254
4	2017	Paper	232

```
proc sort data =test  out=test1;
by descending a ;
run;
```

If the OUT = option is not mentioned, then SAS overwrites the existing dataset

SORTING DATA – PROC SORT

greatlearning
Learning for Life

- ✓Sort Procedure Syntax – Options….
  - ✓ <other options>….
    - ✓ NODUPKEY
      - ✓ checks for & eliminates observations with duplicate BY values
      - ✓ if this option is specified, PROC SORT compares all BY values for each observation to those for the previous observation that is written to the output dataset
      - ✓ if an exact match is found, then the observation is not written to the output dataset
    - ✓ NODUPRECS (ALIAS : NODUP)
      - ✓ checks for & eliminates duplicate observations
      - ✓ if this option is specified, PROC SORT compares all variable values for each observation to those for the previous observation that is written to the output dataset
      - ✓ if an exact match is found, then the observation is not written to the output dataset
    - ✓ DUPOUT = SAS Dataset
      - ✓ specifies the output dataset to which duplicate observations are written

SORTING DATA – PROC SORT

greatlearning
Learning for Life

Obs	Year	Product	Sales
1	2017	Cement	758
2	2017	Timber	254
3	2017	Paper	232
4	2017	Steel	456
5	2017	Cosmetic	456
6	2017	Cosmetic	456

Obs	Year	Product	Sales
1	2017	Paper	232
2	2017	Timber	254
3	2017	Steel	456
4	2017	Cement	758

Using NoDupkey

```
Proc sort data = Testdata out=test1 nodupkey;
by  Sales ;
run;
```

Obs	Year	Product	Sales
1	2017	Paper	232
2	2017	Timber	254
3	2017	Steel	456
4	2017	Cosmetic	456
5	2017	Cement	758

Using NoDuprecs

```
Proc sort data = Testdata out=test1 noduprecs;
by  Sales ;
run;
```

SAS log will have a note containing the no. of duplicate key observations deleted

Retain

The Retain flag is used to prevent a variable from being reset to missing at the top of the data step. This makes it a vital tool for passing information from one observation to another.
The Retain flag is automatically set to yes for all variables that come from the input data set. For new variables, you can set it using the retain statement.
retain x; (will set the Retain flag to yes for the variable x).
You can also set an initial value: **retain x 5;**

```
Data out1(keep=Name count3);
set Grades;
a=a+1;
run;
```

Name	a
Alexander Smith	.
John Simon	.
Patricia jones	.
Jack Benedict	.
Rene Porter	.

Here the variable 'a' is not retained and nor the variable is initialized. As a result we are adding 1 to a missing value every time, so the result is always missing.

```
Data out2(keep=Name count3);
set Grades;
retain a 0;
a=a+1;
run;
```

Name	a
Alexander Smith	1
John Simon	2
Patricia jones	3
Jack Benedict	4
Rene Porter	5

Here the variable 'a' is retained and properly initialized, so in this case we add one to 'a' for each observation , so it ends up containing the observation number.

TRANSPOSING DATA in SAS

greatlearning
Learning for Life

- PROC TRANSPOSE helps to reshape data in SAS. This means changing the data from vertical to horizontal or vice versa. This can be done using data step, arrays and loops but it would be a long code.

Obs	Year	Quarter	Sales
1	2017	Q1	758
2	2017	Q2	254
3	2017	Q3	232
4	2017	Q4	456

Obs	_NAME_	sales_Q1	sales_Q2	sales_Q3	sales_Q4
1	Sales	758	254	232	456

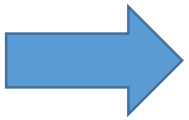
```
PROC TRANSPOSE
DATA = TestData OUT= TransTable PREFIX=sales_;
ID Quarter;
RUN;
```

TRANSPOSING DATA in SAS

greatlearning
Learning for Life

Example 2:

Obs	Year	Quarter	Sales	Profits
1	2017	Q1	758	364
2	2017	Q2	254	122
3	2017	Q3	232	111
4	2017	Q4	456	219
5	2018	Q1	421	202
6	2018	Q2	546	262
7	2018	Q3	123	59
8	2018	Q4	715	343
9	2019	Q1	564	271
10	2019	Q2	586	281



Obs	Year	sales_Q1	sales_Q2	sales_Q3	sales_Q4
1	2017	758	254	232	456
2	2018	421	546	123	715
3	2019	564	586	.	.

To Drop the variable names

```
PROC TRANSPOSE
DATA = TestData OUT= TransTable (DROP = _NAME_) PREFIX=sales_;
ID Quarter;
VAR Sales;
BY Year;
RUN;
```

Using Proc SQL

greatlearning
Learning for Life

We can perform most of the data operations in Proc SQL. Data Set is SAS is known as Tables.
Syntax:

```
PROC SQL;
 SELECT column(s)
 FROM table(s) | view(s)
 WHERE expression
 GROUP BY column(s)
 HAVING expression
 ORDER BY column(s);
QUIT;
```

Selecting subset or creating view from a SAS dataset:

```
#Selecting all variables
proc sql;
 select * from sashelp.class
 ;
quit;
```

```
#Selecting specific variables
proc sql;
 select Name,Sex, Weight
 from sashelp.class;
Quit;
```

Using Proc SQL

greatlearning
Learning for Life

Subsetting a tables based on number of observation

```
#Limiting the number of rows;
proc sql outobs=5;
 select Name,Sex, Weight
 from sashelp.class;
Quit;
```

 # Renaming a variable ;

```
options nolabel;
proc sql;
 select Name,Sex as Gender
 from sashelp.class;
Quit;
```

 # Creating a new table ;

```
proc sql;
 create table class_new as
  select * from sashelp.class
 ;
quit;
```

 # Creating a new variable ;

```
proc SQL;
select Name, Age, Height, Weight,
(weight)/(height*height) as BMI
from sashelp.class;
Quit;
```

Using Proc SQL

greatlearning
Learning for Life

 # Creating a new variable using calculated variable;

```
proc SQL;
select Name, Age, Height, Weight, (weight)/(height*height) as
BMI, Calculated BMI*703 as BMI_new
from sashelp.class;
Quit;
```

#Labelling and formatting;

```
options label;
proc sql;
 select Name, weight FORMAT= 8.2
, Sex Label ="Gender"
 from sashelp.class;
Quit;
```

#Removing duplicate rows;

```
proc sql;
 create table class_new as
  select * from sashelp.class
  ;
quit;

proc sql;
 create table class_new1 as
  select Name, Sex, Age from class_new;
  Insert into class_new1
  values ("Alice", "F", 13)
  ;
quit;

proc sql;
select DISTINCT Name, Sex, Age
from class_new1;
quit;
```

Using Proc SQL

#sorting;

```
proc sql;
 select *
 from sashelp.class
 Order by Weight;
 Quit;
```

 # Subsetting

```
proc sql;
  select * from sashelp.class where sex="M"
  ;
quit;

proc sql;
  select * from sashelp.class where height between 55 and 60
  ;
quit;
```

#Deleting rows;

```
proc sql;
   delete
      from class_new
      where Weight>60
;
quit;
```

#Nested Queries:

```
proc sql;
   delete
      from class_new
      where Weight > ( select Weight from
sashelp.class where height between 55 and 60)
;
quit;
```

greatlearning
Learning for Life

THANK YOU