Dated: 17/03/2018

Statistical Analysis System: Class 8

::

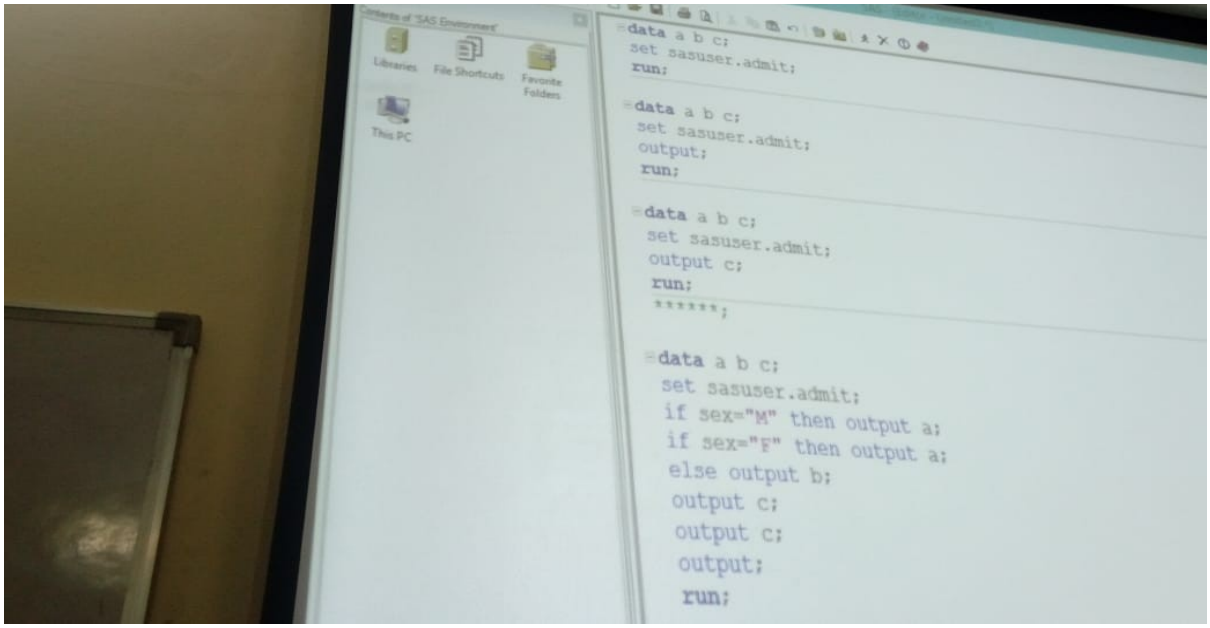
Some Keywords:

1. **Nodupkey:** Is used to remove duplicates from a dataset used with **ProcSort.**
   Nodupkey will remove those observations that have duplicate BY values (variables specified in BY).
   While using Nodupkey , always specify primary key variable in BY Statement.
2. **_ALL_:** If duplicates are to be removed based on all the variables in a dataset then instead of giving all the variables with BY statement we can use **"By _ALL_"** with Nodupkey.
3. **Dupout:** This is used to capture / store data which is dropped by Nodupkey.
4. **Nodup:** This keyword removes those dulplicates from the dataset which has similar values for all the variables in it.
   "BY statement" with Nodup is just to imply sorting.
   Nodup = Noduprec
   Nodupkey works as Nodup for, when " By _All_" staement is used with Nodupkey.
   Nodup works as Nodupkey

Implicit and Explicit Output:

Implicit Output: Where Output is not explicitly mentioned as a statement in the datastep.

Explicit Output: Where Output is explicitly mentioned as a statement in the datastep**.**



Example 1:

Data a b c;

Set sasuser.admit;

Run;

Explained: 3 datasets (a, b, c) are created where data is copied from S.A, here since output is not mentioned as a statement in the datastep therefore this is an **Implicit output**.

Example 2:

Data a b c;

Set sasuser.admit;

Output;

Run;

Explained: 3 datasets (a, b, c) are created where data is copied from S.A, here output is mentioned as a statement in the datastep therefore this is an **Explicit output**.

Example 3:

Data a b c;

Set sasuser.admit;

Output c;

Run;

Explained: 3 datasets (a, b, c) are created. But only dataset C will have the values copied from S.A and dataset a, b will have null values since here output is mentioned as a statement only with dataset C in the datastep.

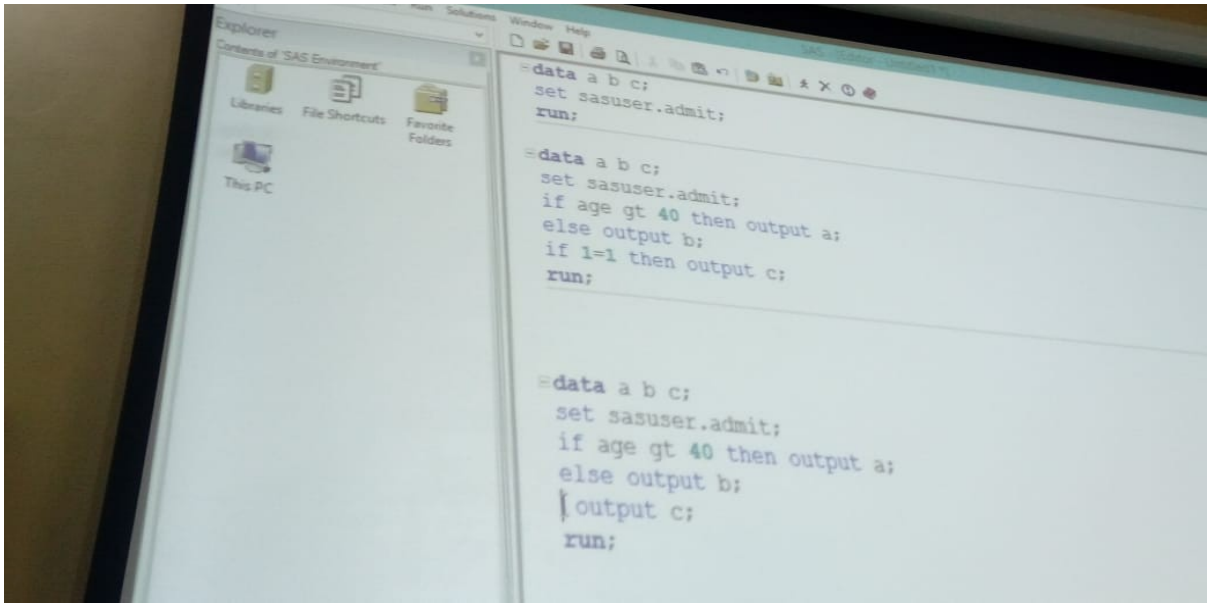
Note: If we declare an output statement for a single dataset in a single datastep, then we neccesarily have to declare the output for the rest of the datasets as well otherwise they will be created be as empty datasets.

Example 4:

```
Data a b c;                                  // Statement 1
Set sasuser.admit;                           // Statement 2
If sex = “M” then output a;                  // Statement 3
If sex = “F” then output a;                  // Statement 4
Else output b;                               // Statement 5
Output c;                                    // Statement 6
Output c;                                    // Statement 7
Output;                                      // Statement 8
Run;
```

Explained: Original S.A has 21 Observations (10 male, 11 female) and 9 Variables, now:

- Statement 3, for dataset “a”, has 10 Obs and 9 Var.
- Statement 4, adds 11 Obs (female), so dataset “a” now has 21 Obs & 9 Var.
- Statement 5, works with in continuation to the previous If-statement and therefore dataset “b” has has 10 Obs and 9 Var.
- Statement 6, copies S.A into “c” therefore dataset “c” has 21 Obs & 9 Var.
- Statement 7, copies / adds S.A into “c” again therefore dataset “c” now has in total 42 Obs & 9 Var.
- Statement 8, adds S.A dataset in the already created datsets (i.e into a, b, c). So now
  Dataset “a” has has 42 Obs & 9 Var.
  Dataset “b” has has 31 Obs & 9 Var.
  Dataset “c” has has 63 Obs & 9 Var.



Example 5:

Data a b c;

set sasuser.admit;

if age gt 40 then output a;

else output b;

if 1=1 then output c;

run;

Example 6:

Data a b c;

set sasuser.admit;

if age gt 40 then output a;

else output b;

output c;

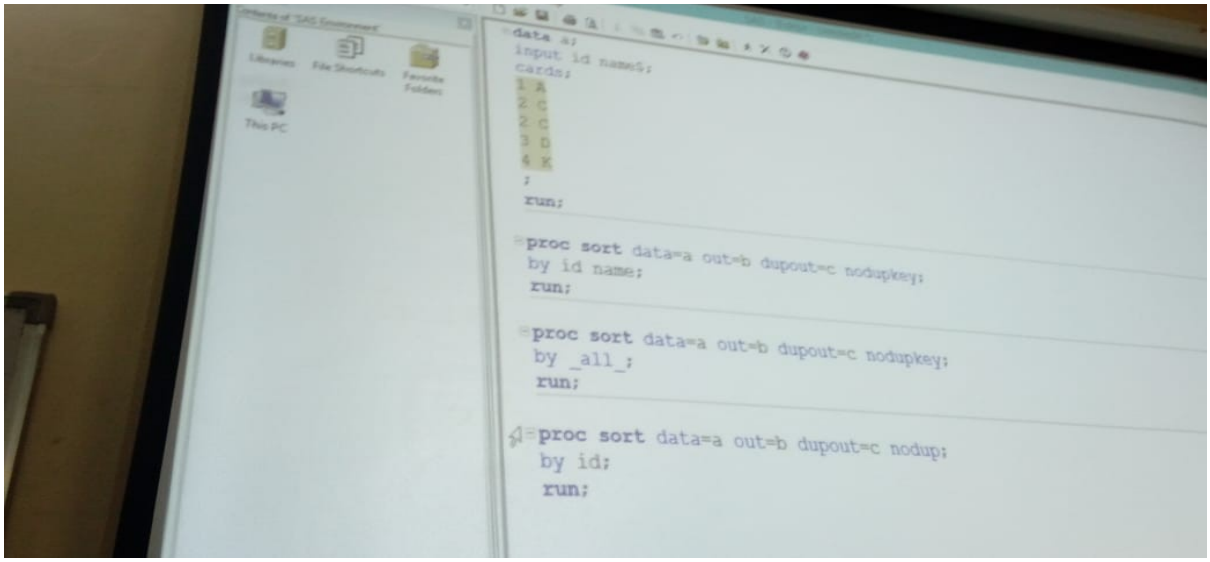
run;

Explained: Example 5 and Example 6 gives the same result, as explained below:

- Dataset “a” will have all the values where age > 40.

- Dataset “b” will have all the values where age <= 40.
- Dataset “c” will have all the values from sasuser.admit.

Note: if 1=1, is a condition (i.e always true) to get the entire dataset copied from dataset “admit” to dataset “c”.



Example:

Data a;

Input id name$;

Cards;

1 A

2 C

2 C

3 D

4 K

;

Run;

- Proc sort data = a out = b dupout = c nodupkey;

  by id name;

run;

Explained: dataset “b” is created where nodupkey will remove duplicate of the key (id, name) given in the by statement and dataset “c” will have that deleted duplicate observations.

- Proc sort data = a out = b dupout = c nodupkey;

  by _all_;

  run;

  Explained: dataset “b” is created where nodupkey will remove duplicate of all the variables given in the by statement (by _all_) and dataset “c” will have that deleted duplicate observations.

- Proc sort data = a out = b dupout = c nodupkey;

  by id;

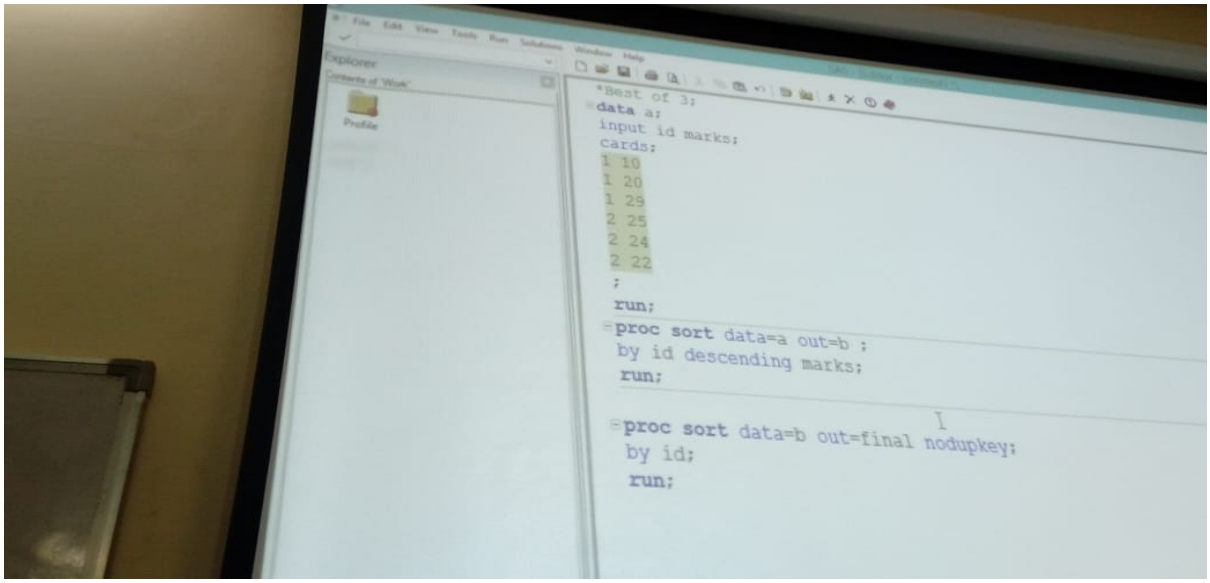
  run;

  Explained: dataset “b” is created where nodupkey will remove duplicate of the key (id) given in the by statement and dataset “c” will have that deleted duplicate observations.

  All the above 3 Proc Sort codes will have the below output in dataset “a”:

ID	Name
1	A
2	C
3	D
4	K

Best Of Three:



Example:

Data a;

Input id marks;

Cards;

1 10

1 20

1 29

2 25

2 24

2 22

;

Run;

Proc sort data = a out = b;

By id descending marks;

Run;

Proc sort data = b out = final nodupkey;

By id;

Run;

Explained: In this we are first sorting dataset “a” by (id, marks in descending order) into a new dataset “b”. Then we are removing duplicates from dataset “b” using **By id**, into dataset “final”.

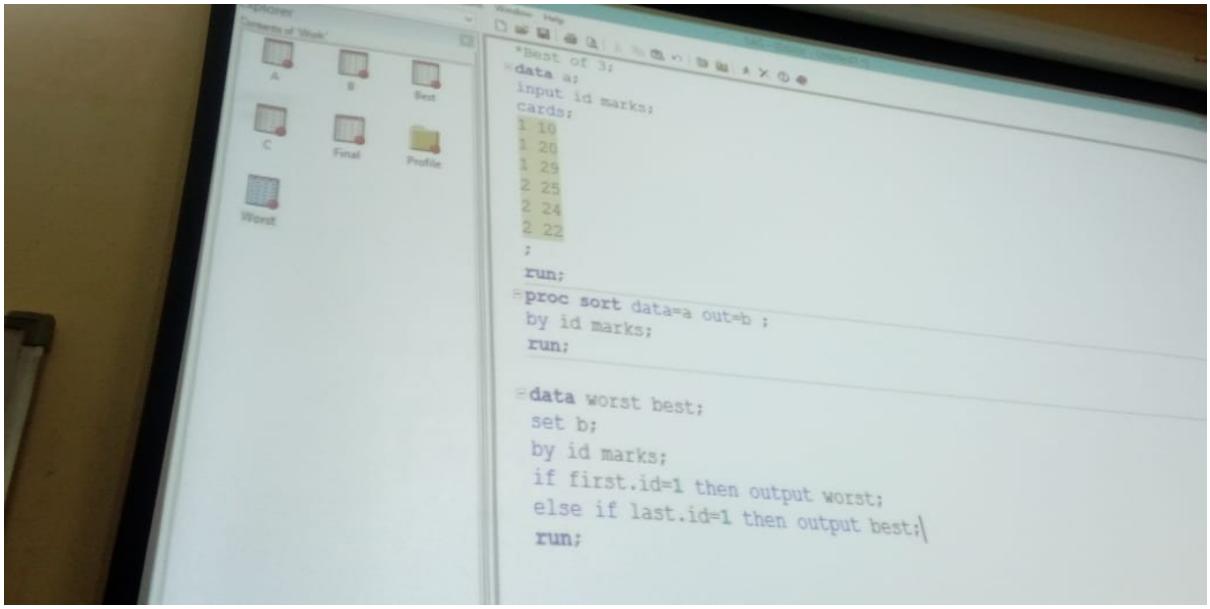
Output (results into best performance):

Id	Marks
1	29
2	25

First. And Last.

- When we use By keyword with Set, it creates 2 automatic boolean variables at the backend which are named as First. And Last.
- These can have values only as 1 or 0.
- Here the data has to be sorted first necessarily.

Example 1:



In the example above (see in pic) data needs to be sorted first:

Proc sort data = a out = b;

By id marks; \\ data sorted by id, marks both in ascending order.

```
Run;
Data worst best;
Set b;
By id marks;
If first.id=1 then output worst;        \\ gives the worst / lowest performance
Else if last.id=1 then output best;     \\ gives the best / highest performance
Run
```

Explained:

Id	Marks	First.id	Last.id
1	10	1	0
1	20	0	0
1	29	0	1
2	22	1	0
2	24	0	0
2	25	0	1

Output:

Best performance (Last.id = 1)

Id	Marks
1	29
2	25

Worst performance (First.id = 1)

Id	Marks
1	10
2	22

To see the First. & Last. Boolean variables created at backend, refer below code:

```
Data check;
Set b;
```

By id marks;

X=first.id;

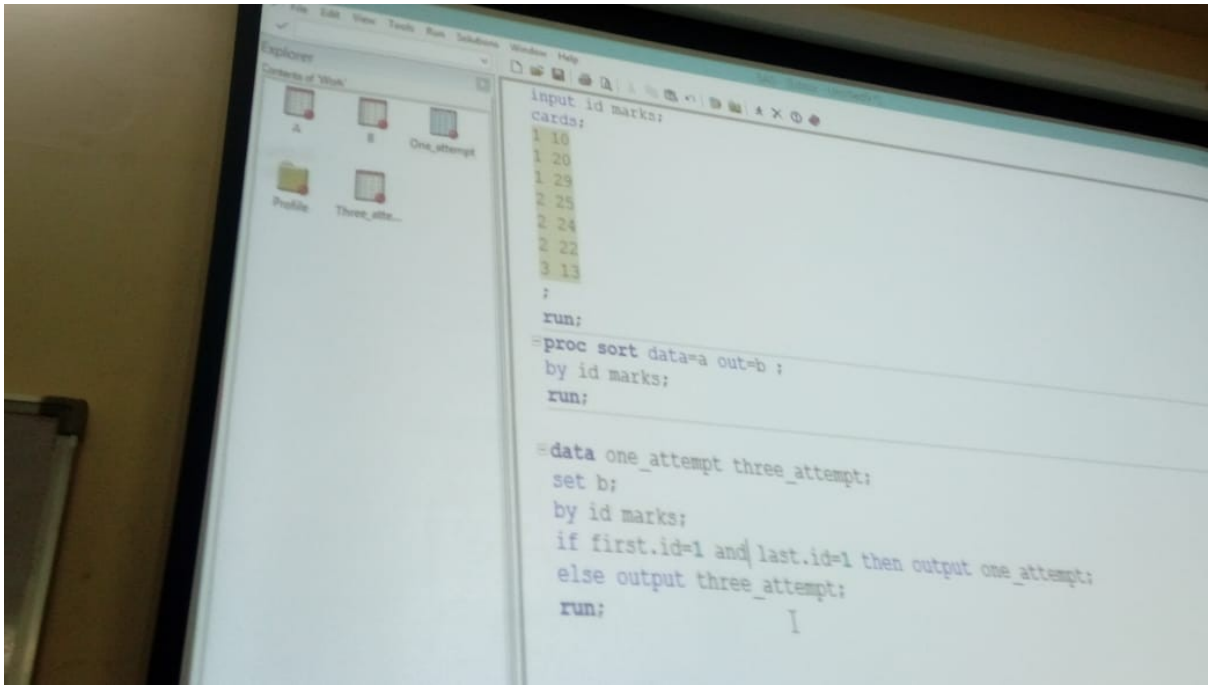
Y=last.id;

Run;

Explained:

Id	Marks	X	Y
1	10	1	0
1	20	0	0
1	29	0	1
2	22	1	0
2	24	0	0
2	25	0	1

Example 2:



Id	Marks	First.id	Last.id
1	10	1	0
1	20	0	0
1	29	0	1
2	22	1	0
2	24	0	0
2	25	0	1
3	13	1	1

Output:

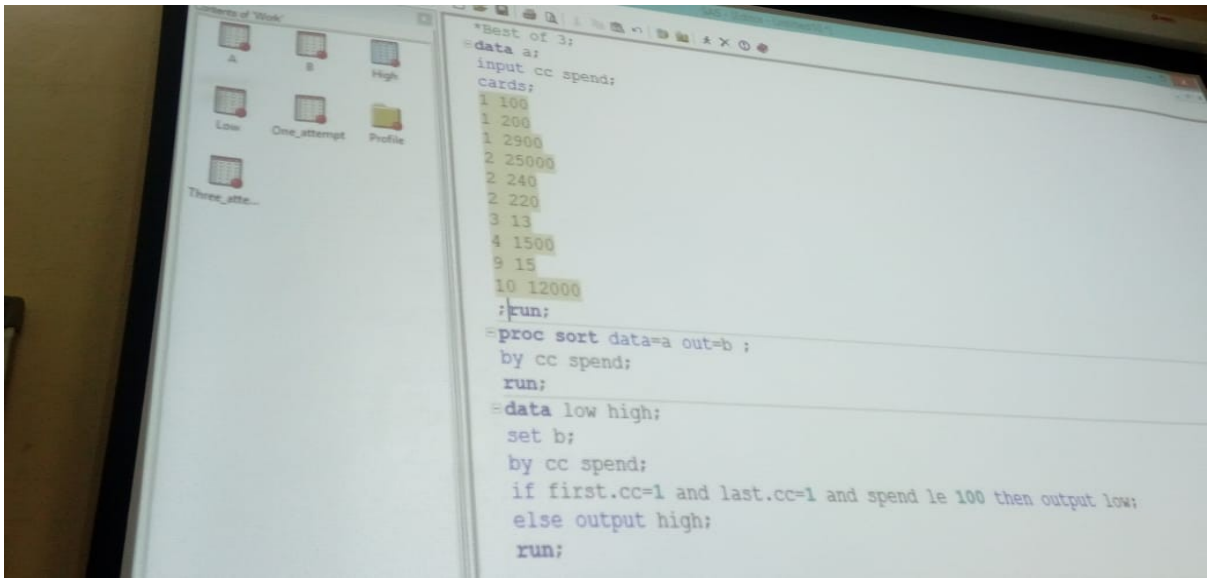
One_attempt (First.id=1 AND Last.id = 1)

Id	Marks
3	13

Three_attempt

Id	Marks
1	10
1	20
1	29
2	22
2	24
2	25

Example 3:



Output:

CC	Spend	First.cc	Last.cc
1	100	1	0
1	200	0	0
1	2900	0	1
2	220	1	0
2	240	0	0
2	25000	0	1
3	13	1	1
4	1500	1	1

9	15	1	1
10	12000	1	1

Low_spender

CC	Spend
3	13
9	15

High_spender

CC	Spend
1	100
1	200
1	2900
2	220
2	240
2	25000
4	1500
10	12000