SAS

SAS (Statistical analysis system) is one of the most popular software for data analysis. It is widely used for various purposes such as data management, data mining, report writing, statistical analysis, business modeling, applications development and data warehousing.

Link 1---------------------------------------------------------------

Proc import

PROC IMPORT is the SAS procedure used to read data from excel into SAS

PROC IMPORT

DATAFILE="filename"

OUT=SAS-data-set

DBMS=identifier

REPLACE;

SHEET="Sheet-name";

GETNAMES=YES;

DATAROW=N;

RANGE="range-name";

RUN;

PROC IMPORT /\*procedure \*/

DATAFILE="filename" /\*location of file For example : DATAFILE = "C:\Desktop\age.xls"\*/

OUT=SAS-data-set /\*option tells SAS to create a dataset with any name of your choice\*/

DBMS=identifier /\*option tells SAS the type of file to read\*/

REPLACE; /\*REPLACE is used to overwrite the existing SAS dataset (If any) mentioned in the OUT= option\*/

SHEET="Sheet-name";/\*option is used to specify which sheet SAS would import\*/

GETNAMES=YES; /\*YES tells SAS to use the first row of data as variable names\*/

DATAROW=N; /\*option is used to specify starting row from where SAS would import the data\*/

RANGE="range-name"; /\*option is used to specify which range SAS would import\*/

RUN;

/\* OUT= option tells SAS to create a dataset with any name of your choice. By default, the imported dataset is saved on WORK library (temporary library)

Examples :

i. OUT = Age . In this statement, PROC IMPORT uses the WORK library. This implies OUT = Age is equivalent to OUT = Work.Age .

ii. OUT = Input.Age. In this statement, PROC IMPORT uses the Input library (Permanent library).\*/

/\*3. DBMS= option tells SAS the type of file to read.

Examples :

i. DBMS = XLS for Excel 97-2003 workbooks ii.DBMS = XLSX for Excel 2007 - 2013 workbooks

\*/

/\*5. SHEET= option is used to specify which sheet SAS would import.

Examples :

i. SHEET = "Sheet1" - To import data from worksheet named sheet1. ii. SHEET = "Goal" - To import data from worksheet named Goal\*/

/\*By default, PROC IMPORT uses GETNAMES= YES. If you type GETNAMES= NO, SAS would not read variable names from first row of the sheet

\*/

/\*For example : DATAROW =5 tells SAS to start reading data from row number 5.

Note :

i. When GETNAMES=YES, DATAROW must be greater than or equal to 2.

ii. When GETNAMES=NO, DATAROW must be greater than or equal to 1\*/

/\*i. RANGE="Sheet1$B2:D10"

This would tell SAS to import data from range B2:D10 from sheet1

ii. RANGE="Information"

This would tell SAS to import data from excel defined name range. In the example shown above, it is Information.\*/

**Importing an XLS (MS Excel 97-2003) format file into SAS**

PROC IMPORT DATAFILE= "C:\age.xls"

OUT= WORK.age

DBMS=XLS

REPLACE;

SHEET="Sheet1";

GETNAMES=YES;

RUN;

**Importing an XLSX (MS Excel 2007-2013) format file into SAS**

PROC IMPORT DATAFILE= "C:\age.xlsx"

OUT= WORK.age

DBMS=XLSX

REPLACE;

SHEET="Sheet1";

GETNAMES=YES;

RUN;

**Importing an excel file from specified row**

PROC IMPORT DATAFILE= "C:\Desktop\Excel\File1.xls"

OUT= INPUT

DBMS=XLS REPLACE;

SHEET="Sheet1";

GETNAMES=YES;

DATAROW=5;

RUN;

**DATAROW=5**tells SAS to start reading data from row number 5. In this case, variable (column) names would be pulled from **first row** but column values would be extracted from row 5

**Importing variable name from other than first row**

*PROC IMPORT DATAFILE= "E:\SAS Code Repository\Book1.xls"  
 DBMS=XLS  
 OUT= TEMP  REPLACE;****NAMEROW=2;******STARTROW=3;****GETNAMES=YES;  
RUN;*

**NAMEROW=2** tells SAS to extract variable names from second row and **STARTROW=3**is used to pull values from third row. **NAMEROW** only works with XLS but not with XLSX

**Importing only specified columns from excel file**

PROC IMPORT

OUT= WORK.want (keep=id x y z)

DATAFILE= "C:\Desktop\File1.xls"

DBMS=XLS REPLACE;

GETNAMES=YES;

RUN;

For example : You don't want to import three variables say A, B and C.

***PROC IMPORT OUT= WORK.want (DROP=A B C) DATAFILE= "C:\Desktop\File1.xls" DBMS=XLS REPLACE;     GETNAMES=YES;RUN;***

**Importing only rows that have non-missing data for all the variables**

PROC IMPORT OUT= WORK.want ( WHERE=(x NE . AND y NE . z NE .))

DATAFILE= "C:\Desktop\File1.xls"

DBMS=XLS REPLACE;

GETNAMES=YES;

RUN;

**Importing data from excel using specified range**

PROC IMPORT OUT= WORK.want

DATAFILE= "C:\Desktop\File1.xls"

DBMS=XLS REPLACE;

RANGE="Sheet1$B4:E100";

GETNAMES=YES;

RUN;

**Rename columns while Importing**

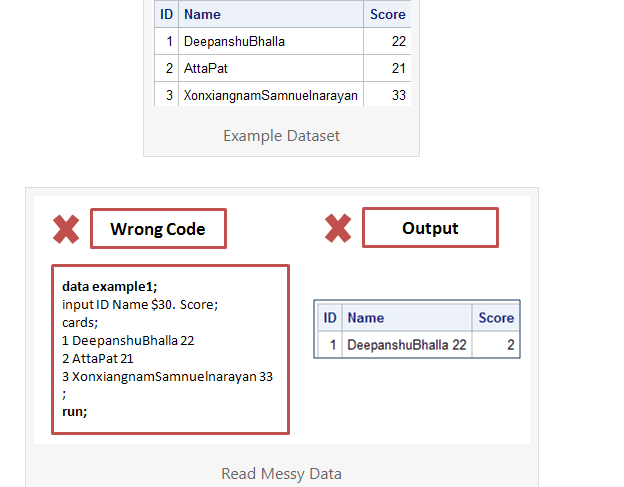
*PROC IMPORT DATAFILE= "E:\SAS Code Repository\Book1.xlsx"  
 DBMS=XLSX  
 OUT= TEMP (****RENAME=****(Score=TotalScore))  REPLACE;  
 GETNAMES=YES;  
RUN;*

**Importing an excel file from website into SAS**

*filename test temp;  
proc http  
 url="https://www2.census.gov/acs2005/GEORES.xls"  
 method="GET"  
 out=test;  
run;  
  
proc import file=test  
out=readin replace  
dbms=xls ;  
 NAMEROW=3;  
 STARTROW=4;  
run;*

Link 2---------------------------------------------------------------

**SAS : READ CHARACTER VARIABLE OF VARYING LENGTH**



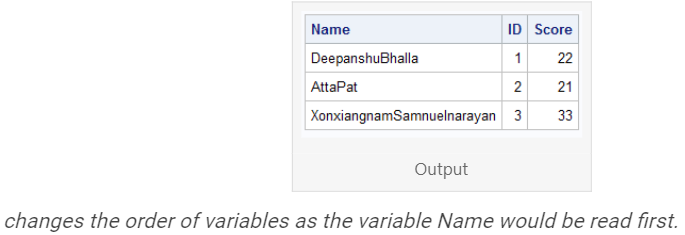
**Method I : Use COLON Modifier**

We can use colon modifier **:** to tell SAS to read variable "**Name"** until there is a space or other delimiter. The  $30. defines the variable as a character variable having max length 30.

*data example1;  
input ID Name****:****$30. Score;  
cards;  
1 DeepanshuBhalla 22  
2 AttaPat 21  
3 XonxiangnamSamnuelnarayan 33  
;  
proc print noobs;  
run;*

**Method II : Use LENGTH statement prior to INPUT Statement**

In the following program, we use a length statement prior to input statement to adjust varying length of a variable. In this case, the variable **Name**would be read first.**Use only $ instead of $30. after "Name" in INPUT statement.**

*data example2;  
length Name $30.;  
input ID Name $ Score;  
cards;  
1 DeepanshuBhalla 22  
2 AttaPat 21  
3 XonxiangnamSamnuelnarayan 33  
;  
proc print noobs;  
run;* 

**Method III : Use Ampersand (&) and Put Extra Space**

We can use ampersand (&) to tell SAS to read the variable until there are two or more spaces as a delimeter. This technique is very useful when the variable contains two or more words. For example, if we have observation like "Deepanshu Bhalla" rather than "DeepanshuBhalla".  
  
**Note : 2 spaces before 22, 21 and 33**

*data example1;  
input ID Name****&****$30. Score;  
cards;  
1 DeepanshuBhalla  22  
2 AttaPat  21  
3 XonxiangnamSamnuelnarayan  33  
;  
proc print noobs;  
run;*

**Example II : When a variable contains more than 1 word**  
  
In this case, we have a space between First Name and Last Name and we want to store both the first and last names in a single variable.



***In this case, the following methods do not work.***

1. *Colon modifier (:) does not work for a variable having multiple words*
2. *LENGTH Statement prior to INPUT Statement does not work here.*

***Use Ampersand (&) and add ADDITIONAL space works.***

*data example1;  
input ID Name & $30. Score;  
cards;  
1 Deepanshu Bhalla  22  
2 Atta Pat  21  
3 Xonxiangnam Samnuelnarayan  33  
;  
proc print noobs;  
run;*

***This trick works in reading data from external file.***

*data temp;  
infile "C:\Users\Deepanshu\Desktop\file1.txt";  
input ID Name****&****$30. Score;  
proc print noobs;  
run;*

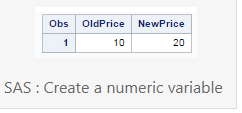
Link 3---------------------------------------------------------------

# CREATING OR MODIFYING A VARIABLE

*DATA Example1;  
OldPrice=10;  
RUN;*

**I. Creating a numeric variable**  
  
You create variables using the form: **variable = expression;**

*DATA Example1;  
SET Example1;  
NewPrice=2\*OldPrice;  
RUN;*



It always create new variable

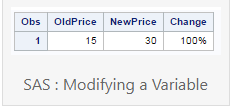
**II. Creating a character variable**  
  
In the same dataset Example1, let's create a character variable say **Type.**The character value for set is set **'Good'**.

*DATA Example1;  
SET Example1;****Type = 'Good'****;  
RUN;*

**III. Creating or Modifying a variable**

*DATA Readin;  
SET Example1;  
OldPrice=5 + OldPrice;  
NewPrice=OldPrice\*2;  
Change= ((NewPrice-OldPrice)/ OldPrice);  
Format Change Percent10.0;  
RUN;*

The **FORMAT statement** is used to display the change value in percentage format. In this case, we are creating a new dataset as well.



Link 4---------------------------------------------------------------

# DROPPING VARIABLES FROM A DATA SET IN SAS

* **DROP =** data set option
* **DROP**statement

# DATA outdata;

# INPUT age gender $ dept obs1 obs2 obs3;

# DATALINES;

# 1 F 3 17 6 24

# 1 M 1 19 25 7

# 3 M 4 24 10 20

# 3 F 2 19 23 8

# 2 F 1 14 23 12

# 2 M 5 1 23 9

# 3 M 1 8 21 7

# 1 F 1 7 7 14

# 3 F 2 2 1 22

# 1 M 5 20 5 2

# 3 M 4 21 8 18

# 1 M 4 7 9 25

# 2 F 5 10 17 20

# 3 F 4 21 25 7

# 3 F 3 9 9 5

# 3 M 3 7 21 25

# 2 F 1 1 22 13

# 2 F 5 20 22 5

# ;

# proc print;

# run;

# 

# Scenario : Create a new variable based on existing data and then drops the irrelevant variables

# By using the DROP statement, we can command SAS to drop variables only at completion of the DATA step.

# *data readin; set outdata; totalsum = sum(obs1,obs2,obs3); drop obs1 obs2 obs3; run;*

# *it will create another column in which sum will be executed*

# Consequence of using DROP = Option data readin; set outdata (drop = obs1 obs2 obs3); totalsum = sum(obs1,obs2,obs3); run;

# It will create emptynew vaibale because we drop the varibale

1. **DROP statement can be used anywhere in DATA steps whereas DROP = option must follow the SET statement.**

#### DROP statement

*data readin;  
set outdata;  
if gender = 'F';****drop age;*** *run;*

**OR**

*data readin;  
set outdata;****drop age;*** *if gender = 'F';  
run;*

#### DROP = option

*data readin;  
set outdata****(drop = age)****;  
if  gender = 'F';  
run;*

**III. Scenario : Dropping variables while printing**  
 **DROP statement** can be used in DATA steps only whereas **DROP = option** can be used in DATA steps and PROC steps (for printing)

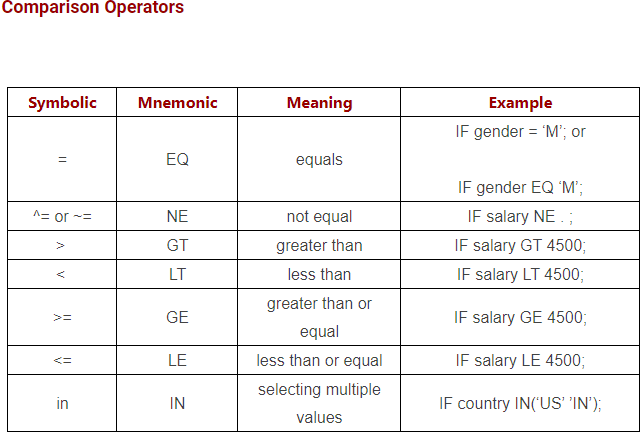
*proc print data = outdata****(drop = age)****;  
where gender = 'F';  
run;*

Link 5---------------------------------------------------------------

# IF-THEN-ELSE STATEMENTS

**Task 1 :**Suppose you are asked to exclude some of the observations in a SAS data set from an analysis that you are generating. For example, you want to exclude all IDs whose values are greater than 100.

To accomplish this task, we can use **IF, IF-THEN DELETE.**



1. **IF statement**  
     
   **IF** **(condition is true)** => It means subsetting a dataset.

Data readin;

Input ID Q1-Q3;

cards;

85 1 2 3

90 3 4 6

95 5 5 6

100 6 6 4

105 5 5 6

110 6 6 5

;

Data readin1;

Set readin;

IF ID LE 100; /\*<= will also work\*/

run;

1. **IF-THEN DELETE**  
     
   **IF** (condition is true) THEN (delete the selected observations);

Data readin1;

Set readin;

IF ID GT 100 THEN DELETE;

run;

**II. IF-THEN-ELSE Statement**  
 **Task 2:** Suppose you want to set a tag on all the IDs. The condition is :  
  
If value of ID is less than or equal to 100 set "Old" tag otherwise set "New" tag.

Data readin1;

Set readin;

IF ID LE 100 THEN TAG ="Old";

ELSE TAG ="New";

run;

**III. IF-THEN-ELSE IF Statement**  
 **Task 3:** Suppose you are asked to update the TAG column.

Data readin1;

Set readin;

IF ID LE 100 THEN TAG ="Old";

ELSE TAG ="New";

run;

/\*We can create another variable in if condition but we canot do in where condition\*/

Data readin1;

Set readin;

length TAG $20;

IF ID <= 85 THEN TAG ="Old";

IF id>85 and id<100 then TAG="medium";

if id >=100 then TAG="high";

run;

run;

proc print data=readin1;

run;

Data readin1;

Set readin;

length TAG $20; /\*it would also work if we don’t use this statement because it is last column I guess\*/

IF ID <= 85 THEN TAG ="Old";

else if id>85 and id<100 then TAG="medium";

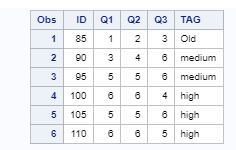
else if id >=100 then TAG="high";

run;

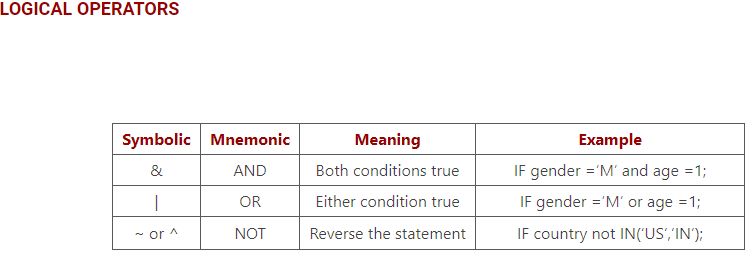
run;

proc print data=readin1;

run;



Same output



**Task 4:** Suppose you want to generate an analysis for Q1 including only responses that are valid (non-missing) and less than 3.

Data readin;

Input ID Q1-Q3;

cards;

85 1 2 3

90 . 4 6

95 2 5 6

100 6 6 4

105 . 5 6

110 6 6 5

;

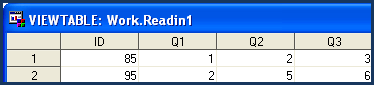
Data readin1;

Set readin;

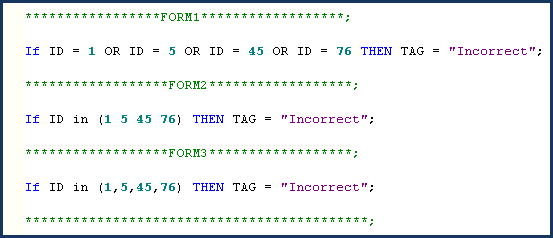
IF (Q1 LT 3) AND (Q1 NE .); /\* IF Q1 LT 3 AND Q1 NE .; this will also work \*/

run;

/\*it will also include missing value as well\*/ so it is not correct output yet

**IF (Q1 LT 3) AND (Q1 NE .) =>** Since missing values are smaller than any other value, we need to give SAS an additional command to separate out missing values.   


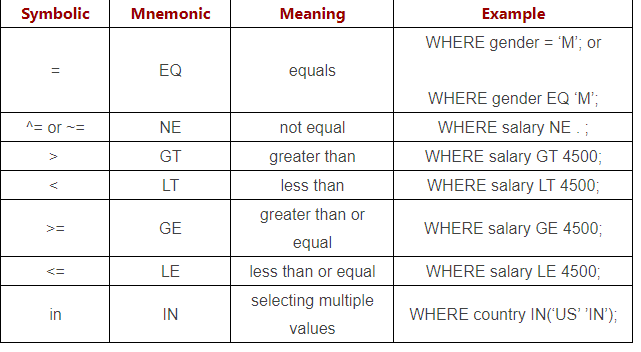
**Selecting Multiple Observations :**  
  
Suppose you want to set tag "Incorrect" to the specified IDs 1,5,45,76  
  
For this case, the logical statement would look like any one of the following statements. It can be written in three ways shown below.

[](https://1.bp.blogspot.com/-3kCRJZlCHIw/UkH7OUaXa4I/AAAAAAAABGY/DbiFZA65Cf8/s1600/form3.png)

**IN Operator**  
  
IN operator is used to select multiple values of a variable. It is an awesome alternative to OR operator.

Link 6---------------------------------------------------------------

# WHERE STATEMENT AND DATASET OPTIONS



**Task1 : Suppose you want to select only section A students. You know the variable Section contains information for students' sections.**

data readin;

input name $ Section $ Score;

cards;

Tom A 84

Raj A 80

Ram B 71

Atul A 77

Priya B 45

Sandy A 67

Sam A 57

David B 39

Wolf B 34

Rahul A 95

Sahul C 84

Lahul C 44

;

run;

data readin1;

set readin;

where Section EQ "A";

run;

**Task2 :** **Suppose you want to select section A and B students. You know the variable Section contains information for students' sections.**

data readin1;

set readin;

where Section IN ("A" "B");

run;

data readin1;

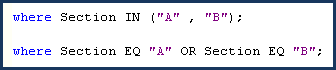
set readin;

where Section IN ("A" ,"B","C","F"); /\*No ‘F’ record but still it process output without error\*/

run;/\*comma will also work ("A" ,"B")\*/

proc print data=readin1;

run;

[](https://3.bp.blogspot.com/--5c1CJ6mvsM/UkPJfXfkb9I/AAAAAAAABG4/QXUzUBXlkw8/s1600/where2.png)

**BETWEEN-AND Operator : Between Two Numbers**  
  
**Task 3 :** **Suppose you want to select scores whose values are greater than or equal to 50 and less than or equal to 75.**

data readin1;

set readin;

where Score between 39 and 75; /\*39 and 75 are included\*/

run;

proc print data=readin1;

run;

**IS MISSING Operator : Selecting Missing Values**  
  
**Task 4 :** **Suppose you want to select only those observations in which students did not fill their section information.**  
The dataset is modified to include missing values in SECTION variable.

data readin1;

set readin;

where Section is missing;

run;

**IS NOT MISSING Operator : Selecting Non-Missing Values**  
 **Task 5 :** **Suppose you want to select only those observations in which students filled their section information.**

data readin1;

set readin;

where section is not missing;

run;

**The NOT operator can be used within WHERE statement in many ways :**  
  
**1. where section is missing and score is not missing;**

**2. where not (score in (34,44,84));**

**3. where not (Score between 50 and 75);**

**4. where NOT(Section EQ "A");**

**CONTAINS Operator : Searching specific character**

**Task 6 :** **Suppose you want to select only those observations in which students' name contain 'hul'.**

data readin1;

set readin;

where name contains 'om'; /TOM was name ,only one record will be printed/

run;

**Note : The CONTAINS operator is case sensitive.**

data readin1;

set readin;

where name contains 'OM';/\*no output because no letter in capital letter\*/

run;

**LIKE Operator : Pattern Matching**  
The LIKE operator selects observations by comparing the values of a character variable to a specified pattern. It is case sensitive.  
  
**Task7 :** **To select all students with a name that starts with the letter S.**

There are two special characters available for specifying a pattern:  
  
1. percent sign (%) - Wildcard Character  
  
2. underscore ( \_ ) - Fill in the blanks

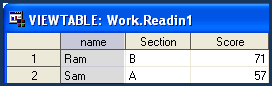
data readin1;

set readin;

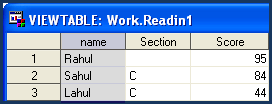
where name like 'S%';

run;

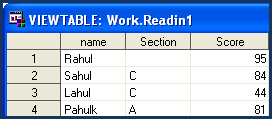
**Examples :**  
  
1. where name like '\_am';  
  
You can also write this statement like .....  
  
where name like '%am';

[](https://1.bp.blogspot.com/-bET7MNnEu2w/Ukb10yXWiiI/AAAAAAAABIc/kl8K7bk7TDo/s1600/where8.png)

2.  where name like '\_ahu\_';  
  
This would not select PAHULK from the variable NAME.

[](https://1.bp.blogspot.com/-ao2qibpIw1A/Ukb3xCLt30I/AAAAAAAABIo/kKFEXisgZ8Q/s1600/where9.png)

3. where name like '\_ahu\_\_';  
  
This would select PAHULK as double underscore (\_\_) is stated.

[](https://1.bp.blogspot.com/-FqHhyK420aQ/Ukb4MQweZAI/AAAAAAAABIw/00cElrXi2zE/s1600/where10.png)

**Sounds-like Operator : Selecting sound like characters**  
  
**Task8 : To select names that sound like 'Ram'.**

data readin;

input name $ Section $ Score;

cards;

Tom A 84

Raj A 80

Ram B 71

Atul . 77

Priya . 45

Sandy A 67

Sam A 57

David B 39

Wolf B 34

Rahul . 95

Sahul C 84

Lahul C 44

Pahulk A 81

Rama A 84

;

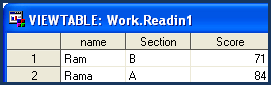
run;

data readin1;

set readin;

where name = \*'Ram';

run;

[](https://1.bp.blogspot.com/-L9uu9NvzhDc/Ukb7F57k0_I/AAAAAAAABI8/4ujYxFCcXMA/s1600/where11.png)

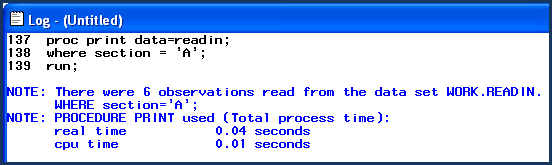
**WHERE = Data Set Option**  
  
1. In the example shown below, the WHERE= data set option is used to select only section A data.  
  
data readin1 (where = (section ='A'));  
set readin;  
run;  
  
2. The following example shows how to use WHERE= data set option in procedures   
  
proc print data=readin (where=(section='A'));  
run;

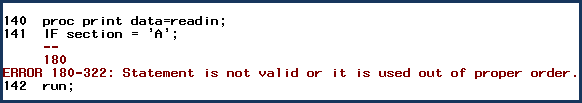
In this case, you can also use WHERE statement....  
  
proc print data=readin;  
where section='A';  
run;

Link 7---------------------------------------------------------------

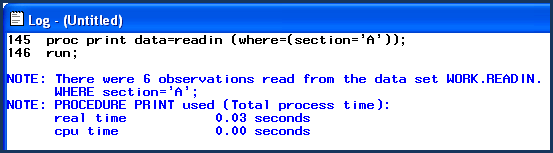
# WHERE VS. IF STATEMENTS

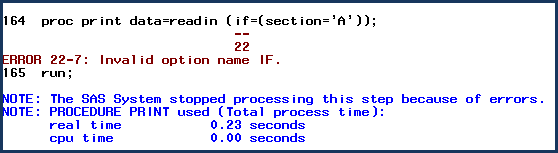
**WHERE condition wins against IF condition in the following cases :**  
  
1. The **WHERE** **statement**can be used in procedures to subset data while **IF** **statement** cannot be used in procedures.  
  
Look at the log of **WHERE and IF statements**shown below :

[](https://4.bp.blogspot.com/-ehRbR0j9mEY/Ukce4yOIcOI/AAAAAAAABJk/HuC-FgCDGlk/s1600/where12.png)

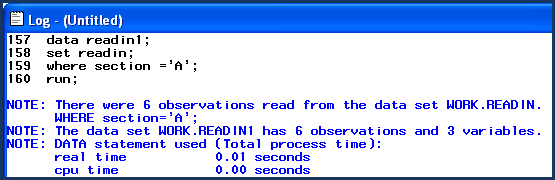
[](https://1.bp.blogspot.com/-I5Nsbs2_-Gk/UkcfuTH4ETI/AAAAAAAABJs/_2QuYvHgO5k/s1600/where13.png)

2. **WHERE** can be used as a data set option while **IF** cannot be used as a data set option.  
  
Look at the log of **WHERE and IF conditions**shown below :

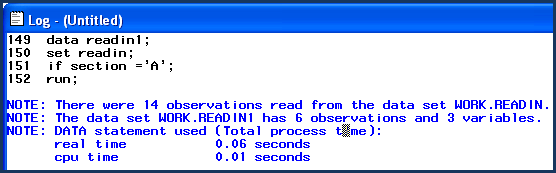
[](https://1.bp.blogspot.com/-r4-QTFwivUU/UkchJA-Hy4I/AAAAAAAABJ4/ZCCNkeljh4Q/s1600/where14.png)

[](https://4.bp.blogspot.com/-JX0voL1iLvw/UkdBYwm6yuI/AAAAAAAABKk/-NMnc37S90A/s1600/where17.png)

3. The **WHERE** **statement** is more efficient than **IF statement.** It tells SAS not to read all observations from the data set.  
  
Look at the LOG (shown below) after using **WHERE** **statement,** you only see a count of the number of observations that meet the criteria in the **WHERE statement**.  
  
Only 6 observations were read from the dataset READIN. In actual, the dataset READIN contains 14 observations.

[](https://2.bp.blogspot.com/-HpNFeW7cCPE/Ukcj-Ae-pfI/AAAAAAAABKU/sxa57wQJnQA/s1600/where16.png)

All 14 observations are read and the 6 that meet the **IF** criteria are placed in the new data set.

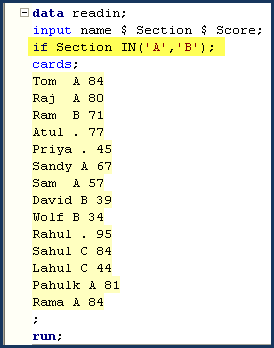
[](https://1.bp.blogspot.com/-zg--WkGzPrg/UkcjCHo7JGI/AAAAAAAABKM/kOgPb4LX4tI/s1600/where15.png)

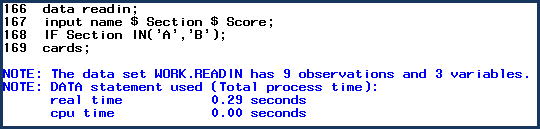
**NOTE : Both statements produced the same result.**

***The where clause****sends only those records that meet condition to PDV,****the IF statement****sends all the records to PDV and removes the records that do not meet condition before they get sent to the output buffer.*

1. The **WHERE** **statement** can be used to search for all similar character values that sound alike while **IF** **statement** cannot be used.  
     
   **For example,**you want to filter out all the names that sound alike 'Sam'.

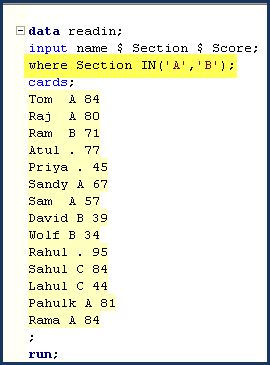
**IF condition wins against WHERE condition in the following cases :**  
  
1. When reading data using INPUT statement.  
  
**IF Statement**  
  
IF Statement can be used when specifying an INPUT statement.

[](https://2.bp.blogspot.com/-EkAqy9ddecU/UkdTiHWnXVI/AAAAAAAABK0/2ZpGpYw0xHU/s1600/where18.png)

[](https://2.bp.blogspot.com/-_SdD6BwP6zc/UkdUDGT_RFI/AAAAAAAABK8/82gPUgFm1_k/s1600/where19.png)

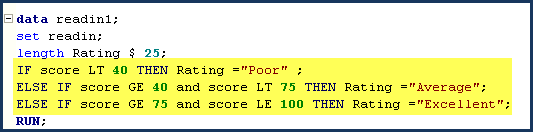
**WHERE Statement**

WHERE statement can not be used when specifying an input statement.

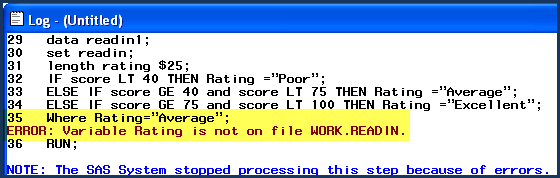
[](https://3.bp.blogspot.com/-k6blclC7Dgc/UkdUputtSpI/AAAAAAAABLE/Odv84PeVW6s/s1600/where20.png)

[](https://1.bp.blogspot.com/-sA9tLOUUvlQ/UkdVM_ctxSI/AAAAAAAABLM/O2XBDwP0g-0/s1600/where21.png)

2. When it is required to execute multiple conditional statements  
  
**Suppose, you have data for college students’ mathematics scores. You want to rate them on the basis of their scores.**  
  
**Conditions :**  
1. If a score is less than 40, create a new variable named “Rating” and give “Poor” rating to these students.  
2. If a score is greater than or equal to 40 but less than 75, give “Average” rating to these students.  
3. If a score is greater than or equal to 75 but less than or equal to 100, give “Excellent” rating to these students.  
  
This can be easily done using IF-THEN-ELSE IF statements. However, WHERE statement requires variables to exist in the data set.

[](https://4.bp.blogspot.com/-xpouauot4Zg/UkdXzKpDbUI/AAAAAAAABLY/83ID34Hyjak/s1600/where22.png)

3. When it is required to use newly created variables in data set.  
  
IF statement can be applied on a newly created variable whereas WHERE statement cannot be applied on a newly created variable. In the below example, IF statement doesn't require variables to exist in the READIN data set while WHERE statement requires variable to exist in the data set.

[](https://4.bp.blogspot.com/-0Tsk9BEPj-s/UkgnWk-Z5TI/AAAAAAAABMI/wZdi2nCjMsw/s1600/where23.png)

4. When to Use **\_N\_**, **FIRST.**, **LAST.** Variables  
  
**WHERE statement** cannot be applied on automatic variables such as **\_N\_, First., Last. Variables.**While **IF statement** can be applied on automatic variables.

|  |
| --- |
| <https://2.bp.blogspot.com/-ppR-PjzppNU/V1xXz6vdjPI/AAAAAAAAEk8/byNOZdPsqTMb_dAlLEkTqgtzWs4l1QfkACLcB/s1600/error.png> |
| Difference between IF and WHERE Conditions |

**Difference : WHERE and IF when merging data sets**

***WHERE statement****applies the subset condition before merging the data sets, Whereas,****IF statement****applies the subset condition after merging the data sets.*

**Create 2 Sample Datasets for Merging**

*data ex1;  
input ID Score;  
cards;  
1 25  
2 28  
3 35  
4 45  
;  
run;*

*data ex2;  
input ID Score;  
cards;  
1 95  
2 97  
;  
run;*

**Merge with WHERE Condition**

*data comb;  
merge ex1 ex2;  
by ID;  
where score <= 30;  
run;*

*It returns 2 observations.****WHERE condition applied before merging.****It applies separately on each of the 2 data sets before merging.*

**Merge with IF Condition**

*data comb;  
merge ex1 ex2;  
by ID;  
if score <= 30;  
run;*

*It returns 0 observation as****IF condition****applied after merging. Since there is no observation in which value of score is less than or equal to 30, it returns zero observation.*

Link 8---------------------------------------------------------------

**HOW DATA STEP AND PROC SQL WORKS**

[Deepanshu Bhalla](https://www.linkedin.com/in/deepanshubhalla) [16 Comments](https://www.listendata.com/2016/04/how-data-step-and-proc-sql-works.html#comment-form) [SAS](https://www.listendata.com/search/label/SAS), [SAS For Beginners](https://www.listendata.com/search/label/SAS%20For%20Beginners)

This tutorial explains the steps to process data in SAS.

**How Data Step Works**  
  
The processing of data is in 2 steps :  
  
**1. Compilation Phase**

**Step I : Syntax Checking**

*SAS scans each statement in the DATA step and check syntax errors, such as missing semicolons and invalid statements.*

**Step II : Creating Input Buffer**

***If you read in a raw data set such as txt or csv file,****the input buffer is created. The input buffer is used to hold raw data.****If you read in a SAS data set,****the input buffer will not be created.*

**Step III : Creating Program Data Vector (PDV)**

1. SAS creates a program data vector (memory on your system) containing the automatic variables \_N\_ and \_ERROR\_.

|  |
| --- |
| <https://3.bp.blogspot.com/-gjjhcKJEnG0/VwpjHKdI2TI/AAAAAAAAEIw/v5Zn0Q-1pIUR3_z5MVCSWUDHje461alcw/s1600/Auto%2BVar.png> |
| How PDV works |

***\_N\_ = 1****indicates the first observation is being processed, \_N\_ = 2 indicates the second observation is being processed, and so on.*

*The automatic variable****\_ERROR\_****with values of 1 or 0, if it is equal to 1 signals the data error of the currently-processed observation, such as reading the data with an incorrect data type.*

2. In addition to the two automatic variables, there is one space allocated for each of the variables in the input statement (reading data).

3. SAS also adds a position to the program data vector for any variables that are created in the DATA step. See the program below. The newly created **FinalScore**variable is derived from Score.

*Data temp2;  
set temp;****FinalScore = Score + 25;****Run;*

|  |
| --- |
| <https://2.bp.blogspot.com/-6rKnro1_Jng/VwpkgLfnptI/AAAAAAAAEI8/qkzGrPSHQMYunSEVHQNyECMsuqWAD1UTg/s1600/Auto%2BVar.png> |
| Newly Created Variable |

4. If the variables specified in the DROP statement, it will never be written to the output data set.

5. At the end of the compilation phase, SAS makes the descriptor portion of the SAS data set which includes the data set name, the number of observations, and the number, names, and attributes of variables.

**2. Execution Phase**

**Sequential Processing (Iterative)**

The DATA step executes once for each observation in the input data set. **Suppose dataset consists of 500 records, SAS will execute 500 times.**

1. At the beginning of the execution phase, SAS sets all of the data set variables in the program data vector to missing:

|  |
| --- |
| <https://1.bp.blogspot.com/-XfMgbjudUqc/VwpqXv9q8NI/AAAAAAAAEJM/B845S1R_qO4YvV23NZonhOkQbODRLfdpg/s1600/Auto%2BVar.png> |
| Execution Phase Step I |

*Variables that you specify in a****RETAIN****statement are not reset to missing values.*

2. The SET statement reads the first observation from the input data set and writes the values to the program data vector.

|  |
| --- |
| [https://4.bp.blogspot.com/-_AdOTEGbPuU/VwprMb2CPoI/AAAAAAAAEJU/1nTydhulCSsE4Ct1DaRmTFjP5d4Wn-XyA/s320/Auto%2BVar.png](https://4.bp.blogspot.com/-_AdOTEGbPuU/VwprMb2CPoI/AAAAAAAAEJU/1nTydhulCSsE4Ct1DaRmTFjP5d4Wn-XyA/s1600/Auto%2BVar.png) |
| Exceution Phase Step II |

3. Compute the first value for the derived variable, FinalScore.

|  |
| --- |
| [https://4.bp.blogspot.com/-0BS0keSIUgQ/VwprcdBD1xI/AAAAAAAAEJc/pIv7S2Nv1YssUt8FTXJNILLHwacUUTT9g/s320/Auto%2BVar.png](https://4.bp.blogspot.com/-0BS0keSIUgQ/VwprcdBD1xI/AAAAAAAAEJc/pIv7S2Nv1YssUt8FTXJNILLHwacUUTT9g/s1600/Auto%2BVar.png) |
| Execution Phase Step III |

4. At the end of the first iteration of the DATA step, the values in the program data vector are written to the **output data set temp2**as the first observation.

**Loop until all of the observations are read**

The value of the automatic variable \_N\_ is increased to 2. The values from the second observation are written to the program data vector. Repeat above Steps 2, 3 and 4. It continues until all of the observations are read.

**How PROC SQL Works**

***PROC SQL is a simultaneous process for all the observations.***

**Step I : Syntax Checking**

*SAS scans each statement in the SQL procedure and check syntax errors, such as missing semicolons and invalid statements.*

**Step II : SQL Optimizer**

*SQL optimizer scans the query inside the statement. The SQL Optimizer decides how the SQL query should be executed in order to minimize run time. The Optimizer examines submitted SQL code and characteristics of the SAS system and then creates efficient executable statements for the submitted query. The created code can be quite complicated and often involves the creating, sorting and merging of many temporary files as well as the trimming of variables and observation at times that will minimize run time.*

**Step III : Load into Data Engine**

*Any tables in the****FROM****statement are loaded into the data engine where they can then be accessed in memory.*

**Step IV : Code and Calculations are executed**

**Step V : Final Table is created in memory**

**Step VI : Final Table sent to the output table described in the CREATE TABLE statement.**

**Main Distinction : Data Step vs. Proc SQL**

*DATA step is a sequential process (one at a time for each observation), whereas the SQL procedure is a simultaneous process for all the observations.*

**Efficiency : DATA STEP vs. PROC SQL**

*The SQL procedure performed better with the smaller datasets (less than approx. 100 MB) whereas the data step excelled with the larger ones (more than approx. 100 MB).*

It is because the DATA step handles each record sequentially so it never uses a lot of memory, however, it takes time to process one at a time. So with a smaller dataset, the DATA step is going to take more time sending each record through.  
  
With the SQL procedure, everything is loaded up into memory at once. By doing this, the SQL procedure can process small datasets rather quickly since everything is available in memory. Conversely, when you move to larger datasets, your memory can get bogged down which then leads to the SQL procedure being a little bit slower compared to the DATA step which will never take up too much memory space.

*If you need to connect directly to a database and pull tables from there, then use PROC SQL.*

Link 9---------------------------------------------------------------

**PROC FREQ EXPLAINED WITH EXAMPLES**

This tutorial explains how to use PROC FREQ with various examples. The PROC FREQ is one of the most frequently used SAS procedures which helps to summarize categorical variable. It calculates count/frequency and cumulative frequency of categories of a categorical variable. In other words, it returns the number and percentage of cases falling in multiple categories of a categorical variable. It's not just restricted to counts. It also produces bar charts and tests for association between two categorical variables.

**Create a sample data set**  
  
The program below creates a sample SAS dataset which would be used in further examples to explain PROC FREQ.

*data example1;  
input x y $ z;  
cards;  
6 A 60  
6 A 70  
2 A 100  
2 B 10  
3 B 67  
2 C 81  
3 C 63  
5 C 55  
;  
run;*

***The created dataset looks like below -***

|  |  |  |
| --- | --- | --- |
| **X** | **Y** | **Z** |
| 6 | A | 60 |
| 6 | A | 70 |
| 2 | A | 100 |
| 2 | B | 10 |
| 3 | B | 67 |
| 2 | C | 81 |
| 3 | C | 63 |
| 5 | C | 55 |

**Example 1 : To check the distribution of a categorical variable (Character)**  
  
Suppose you want to see the frequency distribution of variable 'y'.

*proc freq data = example1;  
tables y;  
run;*

The **TABLES statements** tells SAS to return n-way frequency and crosstabulation tables and computes the statistics for these tables.

|  |
| --- |
| <https://3.bp.blogspot.com/-p8zUj-OZ72I/Vr9yOrjrj-I/AAAAAAAAEAE/6zSqIIfyEqg/s1600/PROC%2BFREQ.png> |
| Output : PROC FREQ |

It answers a question 'which category holds the maximum number of cases'. In this case, the category 'C' contains maximum number of values.  
 **Tip :**

*Categorical variables are of two types -****Nominal****and****Ordinal****. A nominal variable is a categorical variable in which categories do not have any order. For example, gender, city etc. An ordinal categorical variable has categories that can be ordered in a meaningful way. For example, rank, status (high/medium/low) etc.*

**Example 2 : To remove unwanted statistics in the table**

Suppose you do not want **cumulative frequency and cumulative percent** to be displayed in the table. The option **NOCUM**tells SAS to not to return cumulative scores.

*proc freq data = example1;  
tables y /****nocum****;  
run;*

|  |
| --- |
| <https://1.bp.blogspot.com/-qGF2G0-_h_Y/WN-FeQdWsSI/AAAAAAAAGFE/rysd_sVWCVoDyuthDxcJMnWQU8tNbwZbACLcB/s1600/proc%2Bfreq%2B1.png> |
| NOCUM Option |

 If you want only frequency, **not percent distribution and cumulative statistics**.

*proc freq data = example1;  
tables y /****nopercent nocum****;  
run;*

|  |
| --- |
| <https://4.bp.blogspot.com/-768Ej2VnOT0/WN-F-GB-xaI/AAAAAAAAGFM/DrdlNrmL-K4HrJMmKuUMEleEtkZj5qp1wCLcB/s1600/proc%2Bfreq%2B2.png> |
| NOPERCENT and NOCUM option |

**Example 3 : Cross Tabulation ( 2\*2 Table)**

Suppose you want to see the **distribution of variable 'y' by variable 'x'**.

*proc freq data = example1;  
tables y \* x;  
run;*

|  |
| --- |
| <https://1.bp.blogspot.com/-AUYwz1V4Lvc/WLnED1RoXLI/AAAAAAAAF7A/ApxL_kWBuKA8xakg1hhmsrcCMJOESTexQCLcB/s1600/proc%2Bfreq%2Bexample.png> |
| Proc Freq Output |

The output of the above SAS program is shown in the image above.  
  
  
**Example 4 : Show Table in List Form**

Suppose you do not want output to be shown in tabular form. Instead, you want final analysis to be displayed in list form (See the image below)

|  |
| --- |
| <https://2.bp.blogspot.com/-0yBBGl_c6bM/WLnE-vw5g0I/AAAAAAAAF7E/8NKfpMt0OhMp8n2aKYry64aorNoZxzrsACLcB/s1600/proc%2Bfreq%2Bexample2.png> |
| PROC FREQ List Form |

*proc freq data = example1;  
tables y \* x /****list****;  
run;*

The forward slash followed by **LIST**keyword produces the list styled table.

**Example 5 : Hide Unwanted Statistics in Cross Tabulation**

*proc freq data = example1;****tables y \* x / norow nocol nopercent;*** *run;*

The **NOROW**option hides row percentage in cross tabulation. Similarly, **NOCOL**option suppresses column percentage.

|  |
| --- |
| <https://2.bp.blogspot.com/-TAHoywCnmik/WN-HVGs1IUI/AAAAAAAAGFY/kO0TTu940yIUIbW-C3Hw_LljhKD3-kGnACLcB/s1600/proc%2Bfreq%2B3.png> |
| NOROW and NOCOL Options |

**Example 6 : Request Multiple Tables**  
  
Suppose you want to generate multiple crosstabs. To accomplish it, you can run the command below-

*proc freq data = example1;****tables y \* (x z) / norow nocol nopercent;*** *run;*

*The****tables y\*(x z)****statement is equivalent to****tables y\*x y\*z****statement. In this case, it returns two tables - y by x and y by z.*

**Example -**tables (a b)\*(c d); is equivalent to tables a\*c  b\*c  a\*d  b\*d;  
 **Example 7 : Number of Distinct Values**  
  
The **NLEVELS**option is used to count number of unique values in a variable.

*proc freq data = example1****nlevels****;  
tables y;  
run;*

In this case, it returns 3 for variable Y.

**Example 8 : Use WEIGHT Statement**

The **WEIGHT statement** is used when we already have the counts. It makes PROC FREQ use count data to produce frequency and crosstabulation tables.

*Data example2;  
input pre $ post $ count;  
cards;  
Yes Yes 30  
Yes No 10  
No Yes 40  
No No 20  
;  
run;  
proc freq data=example2;  
tables pre\*post;****weight count;****run;*

|  |
| --- |
| <https://2.bp.blogspot.com/-QRsIGZKrFRY/WN-Ma9sWU3I/AAAAAAAAGFo/ig83iH32qNYT0eEJ1ujxi4WUHEvmqVQxwCLcB/s1600/PROC%2BFREQ%2BWeight.png> |
| PROC FREQ Weight Statement |

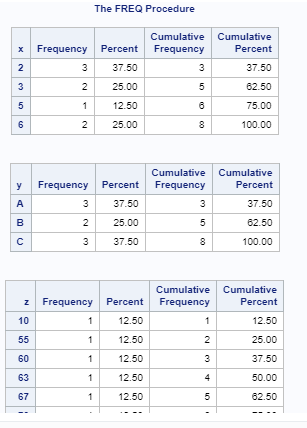
**Example 9 : Store result in a SAS dataset**  
  
Suppose you wish to save the result in a SAS dataset instead of printing it in result window.

*proc freq data = example1****noprint****;  
tables y \*x /****out = temp****;  
run;*

The **OUT option** is used to store result in a data file. **NOPRINT option**prevents SAS to print it in results window.

proc freq data=example1;

run;



That’s why we use table parameter in frequency table

Link 10---------------------------------------------------------------

**SAS TIP : SPECIFY A LIST OF VARIABLES**

[Deepanshu Bhalla](https://www.linkedin.com/in/deepanshubhalla) [17 Comments](https://www.listendata.com/2014/07/sas-tip-specify-list-of-variables.html#comment-form) [SAS](https://www.listendata.com/search/label/SAS)

Suppose you have a list of variables. You don't want to type the name of each variable to define them in a function or array. You are looking for a shortcut to accomplish this task.  
 **Create a dataset with a list of variables**

| **Q1** | **Q3** | **Q4** | **Q2** | **Q6** | **BU** | **Q5** |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 5 | sa | an | 3 |
| 2 | 4 | 3 | 6 | sm | sa | 4 |
| 6 | 5 | 3 | 8 | cb | na | 3 |

*data dummy;  
input q1 q3 q4 q2 q6$ bu$ q5;  
cards;  
1 2 3 5 sa an 3  
2 4 3 6 sm sa 4  
6 5 3 8 cb na 3  
;  
run;*

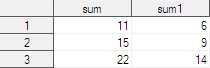
**How to specify a list of variables**

**A single dash (-)**is used to specify consecutively numbered variables. For example : q1-q4;

**A double dash (--)** is used to specify variables based on the order of the variables as they appear in the file, regardless of the name of the variables.

*data dummy1 (drop= q1--q5);  
set dummy;  
sum = sum(of q1-q4);  
sum1 = sum(of q1--q4);  
run;*

The output is shown in the image below -

[](https://1.bp.blogspot.com/-QwxGCZQVed0/U77ixc0QtqI/AAAAAAAADJA/xTwao66AJes/s1600/vars.png)

*In the above program, q1-q4 includes q1,q2,q3 and q4, whereas q1--q4 includes q1,q3 and q4 only as they appear the same way in file.*

*data dummy1 (drop= q1--q5);*

*set dummy;*

*sum = sum( q1-q4); # I didn’t use of eyword it output will be -2*

*sum1 = sum(of q1--q4);*

*run;*

*proc print data=dummy1;*

*run;*

**How to specify all NUMERIC variables**

*data dummy1 (drop= q1--q5);  
set dummy;  
sum = sum(of \_numeric\_);  
run;*

**How to use double dash in array**  
  
The following program subtracts one from values in variables q1,q3 and q4.

*data dummy1;  
set dummy;  
array vars q1--q4;  
do over vars;  
vars = vars - 1;  
end;  
run;*

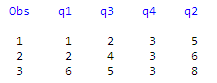
**How to use numeric variables in array**  
  
The following program subtracts one from values in numeric variables.

*data dummy1;  
set dummy;  
array vars \_numeric\_;  
do over vars;  
vars = vars - 1;  
end;  
run;*

**More Dash Symbol Usage**

1. Print all **NUMERIC** variables from q1 through q6.

*proc print;  
var q1-numeric-q6;  
run;*

[](https://4.bp.blogspot.com/-zn6dXqFOUwE/U77p4ry1V5I/AAAAAAAADJQ/x9AqSqO7S7Q/s1600/sas+tip.png)

2. Print all **CHARACTER** variables from q1 through q6.

*proc print;  
var q1-character-q6;  
run;*

[](https://1.bp.blogspot.com/-fhr2AXFnZHM/U77qi8aoIlI/AAAAAAAADJY/090xHlfVSkY/s1600/qqq6.png)

3. Print all **CHARACTER** variables.

*proc print;  
var \_character\_;  
run;*

Link 11---------------------------------------------------------------

**SAS : WILDCARD CHARACTER**

In this tutorial, you will learn how to use wildcard character in SAS.

**Example 1 : Keep all the variables start with 'X'**

*DATA READIN;  
INPUT ID X1 X\_T $;  
CARDS;  
2 3 01  
3 4 010  
4 5 022  
5 6 021  
6 7 032  
;  
RUN;*

*DATA READIN2;  
SET READIN****(KEEP = X:)****;  
RUN;*

The **COLON (:)**tells SAS to select all the variables starting with the character 'X'.

**Example 2 : Subset data using wildcard character**

*DATA READIN2;  
SET READIN;  
IF X\_T =: '01';  
RUN;*

In this case, the **COLON (:)**tells SAS to select all the cases starting with the character '01'.

**Example 3 : Use of WildCard in IN Operator**

*DATA READIN2;  
SET READIN;  
IF X\_T IN: ('01', '02');  
RUN;*

In this case, the **COLON (:)**tells SAS to select all the cases starting with the character '01' and '02'.

**Example 4 : Use of WildCard in GT LT (> <) Operators**

*DATA READIN2;  
SET READIN;  
IF X\_T >: '01';  
RUN;*

In this case, the **COLON (:)**tells SAS to select all the cases from character '01' up alphabetically.

**Example 5 : WildCard in Function**

*data example3;  
set temp2;  
total =sum(of height:);  
run;*

Link 12---------------------------------------------------------------

# CHARACTER FUNCTIONS

I have left function , I will work separately on this topic becsuae this is very usefull topic for interview

Link 13---------------------------------------------------------------

# DATE FORMATS AND INFORMATS

I have left function , I will work separately on this topic becsuae this is very usefull topic for interview

Link 14---------------------------------------------------------------

# DATE FUNCTIONS

I have left function , I will work separately on this topic becsuae this is very usefull topic for interview

Link 15---------------------------------------------------------------

# INTCK FUNCTION WITH EXAMPLES

I have left function , I will work separately on this topic becsuae this is very usefull topic for interview

Link 16---------------------------------------------------------------

# INTNX FUNCTION WITH EXAMPLES

I have left function , I will work separately on this topic becsuae this is very usefull topic for interview

Link 17---------------------------------------------------------------

# MISSING VALUES IN SAS

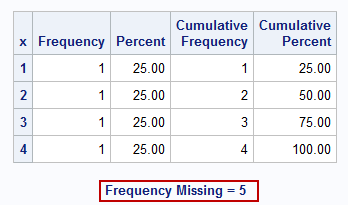
**Numeric Missing Values**  
SAS stores 28 missing values in a numeric variable.They are as follows :

1. dot-underscore  **. \_**
2. dot **.**
3. **.**A through **.**Z ( Not case sensitive)

**Sorting Order :**dot- underscore is the lowest valued missing value. After the dot-underscore, comes the dot, and then the dot-A. The dot-Z is the highest valued missing value.

***Run the following code and see how SAS treats them missing value***

*data temp;  
input x;  
cards;  
1  
2  
3  
.  
.A  
.X  
.Z  
.\_  
4  
;  
run;  
  
proc freq;  
table x;  
run;*

[](https://2.bp.blogspot.com/-R3sAuQcvXtE/VMfD-1vpiEI/AAAAAAAADdg/EPvemyqRpRI/s1600/missing.png)

**Check for missing numeric values**

The following code checks for dot missing value only. It does not check for other 27 special numeric missing values (.\_ , .A through .Z)

*If x =. then PUT "x is missing";*

The following code checks for all 28 numeric missing values (.  , .\_ , .A through .Z)

*If x <=.z then PUT "x is missing";*

The **MISSING** function accepts either a character or numeric variable as the argument and returns the value 1 if the argument contains a missing value or zero otherwise.

*If missing(x) then PUT "x is missing";*

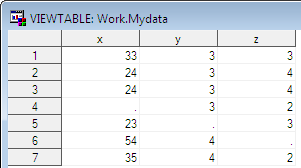
**Character missing values**

Character missing values are represented by a single blank enclosed in quotes **' '**.

*If y = ' ' then put "y is missing";  
If missing(y) then put "y is missing";*

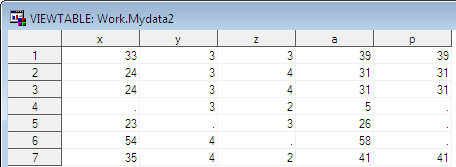
**Working with Missing Values**

Suppose we have a data set containing three variables - X, Y and Z. They all have some missing values. We wish to compute sum of all the variables.

[](https://3.bp.blogspot.com/-KX01gqDJraU/U4Gx39GHd5I/AAAAAAAACtg/Vy54QAjcXBI/s1600/dataset.png)

*data mydata2;  
set mydata;  
a=sum(x,y,z);  
p=x+y+z;  
run;*

**SUM**function returns the sum of non-missing arguments whereas **“+”** operator returns a missing value if any of the arguments are missing.

[](https://2.bp.blogspot.com/-jgqIv9nHz0g/U4GynHz1GPI/AAAAAAAACto/5NMSJIceMFQ/s1600/dataset2.png)

**Functions that handle MISSING data**

**NMISS :**The NMISS() function will return the number of missing values in the specified list of numeric variables. The NMISS() function will convert any character values to numeric before assessing if the argument value is missing.

*IF NMISS(x,y,z) = 0 then PUT " All variables have non-missing values";*

**CMISS :** The CMISS() function introduced in SAS 9.2 is similar to the NMISS() function that it counts the number arguments that are missing, but for both character and numeric variables without requiring character values to be converted to numeric.  
  
**N :** The N() function returns the number of non-missing values in a list of numeric variables.  
  
**CALL MISSING**  
  
SAS provides the statement CALL MISSING() to explicitly initialise or set a variable value to be missing.

*data \_null\_;****call missing( num\_1, num\_2, x );****num\_3 = x;  
put num\_1 = / num\_2 = / num\_3 = ;  
run;*

**Delete empty rows**

*options missing = ' ';  
data readin;  
set outdata;  
if missing(cats(of \_all\_)) then delete;  
run;*

**How missing values are handled in SAS procedures**  
  
**1.  PROC FREQ**  
  
By default, missing values are excluded and percentages are based on the number of non-missing values. If you use the **"/ MISSING"** option on the tables statement, the percentages are based on the total number of observations (non-missing and missing) and the percentage of missing values are reported in the table.

*PROC FREQ DATA= TEST;  
TABLES X /****MISSING****;  
RUN;*

**2.  PROC MEANS**  
  
It produces statistics on non-missing data only. The NMISS option is used to calculate number of missing values.

*Proc Means Data = test N NMISS;  
Var q1 - q5 ;  
Run;*

To see number of observations having a missing value for the classification variable, type **MISSING** option in PROC MEANS.

*Proc Means data = test N NMISS****MISSING****;  
Class Age ;  
Var q1 - q5;  
Run;*

**3. PROC CORR**  
  
By default, correlations are computed based on the number of pairs with non-missing data (pairwise deletion of missing data). The nomiss option can be used on the proc corr statement to request that correlations be computed only for observations that have non-missing data for all variables on the var statement (listwise deletion of missing data).  
  
**4. PROC REG**  
  
If any of the variables on the model or var statement are missing, they are excluded from the analysis (i.e., listwise deletion of missing data)  
  
**5. PROC LOGISTIC**  
  
If any of the variables on the model or var statement are missing, they are excluded from the analysis (i.e., listwise deletion of missing data)  
  
**6. PROC FACTOR**  
  
Missing values are deleted listwise, i.e., observations with missing values on any of the variables in the analysis are omitted from the analysis.

Link 18---------------------------------------------------------------

# CONVERT CHARACTER VARIABLE TO DATE

THIs tutorial explains multiple ways we can convert character variable to SAS date.

*Suppose you encounter a problem in which you need to convert character variable to SAS date format. It happens most of the times when we upload raw data file in TXT, EXCEL or CSV format to SAS. The problem with dates in character format is you cannot apply any calculations on them.*

**Create a Sample Data**

*data example;  
input dateofbirth $20.;  
cards;  
05/11/1980  
07/05/1990  
04/14/1981  
;  
run;*

**Convert Character Variable to SAS Date**

The **INPUT**function is used to convert character variable to numeric. With MMDDYY10. format, we assign format of the date.

*data out;  
set example;****dateofbirth2 = input(strip(dateofbirth),MMDDYY10.);*** *format dateofbirth2 MMDDYY10.;  
run;*

**Important Note :**Please make sure a new variable is created for conversion. If you use the same variable for conversion, the format of the variable would remain character.

|  |
| --- |
| <https://4.bp.blogspot.com/-YxMz7qSrQj0/V1R_OkOlZnI/AAAAAAAAEeY/xVg9KKhyJ4Q3C8iTlQQ7789b7poTbmEcgCLcB/s1600/sas%2Bdata.png> |
| Output : Convert Character Variable to Date |

**If Convert to Different Date Format**  
  
As you can see our original **dateofbirth**variable is in Month-Date-Year format but stored as a character. If we need to convert it to Date-Month-Year format and stored as in SAS date format.

*data out;  
set example;  
dateofbirth2 = input(strip(dateofbirth), MMDDYY10.);  
format dateofbirth2****DDMMYY10.****;  
run;*

Make sure you put the original date format in INPUT function and put the desired date format in **FORMAT statement**. If you put the different date format in INPUT function, it may lead to missing value. **For example,** 04/14/1981 cannot be converted to DDMMYY10. format directly as SAS reads 14 as month which is not possible.

Link 19---------------------------------------------------------------

# CONVERTING NUMBER FORMAT TO DATE FORMAT

Suppose you have a numeric variable that contains dates. You are asked to convert it to SAS date format. It seems to be a easy task but it sometimes becomes a daunting task when you don't know how SAS treats dates. The input data is shown below -

|  |
| --- |
| <https://4.bp.blogspot.com/-N_P6vX0x6ZI/V4vPizp6keI/AAAAAAAAE2w/KAs-WQ2Hv-A-C3DXn1uIBRWT2f10lzQzACLcB/s1600/date%2Bvalues.png> |
| Raw Date Values |

**Sample Data**

The following program is used to create a sample data.

*data temp;  
input date;  
cards;  
20160514  
19990505  
20131104  
20110724  
;  
run;*

**Solution**

*data temp2;  
set temp;  
newdate = input(put(date,8.),yymmdd8.);  
format newdate date10.;  
proc print noobs;  
run;*

|  |
| --- |
| <https://3.bp.blogspot.com/-QCnMBEAo8jM/V4vSErwiEYI/AAAAAAAAE28/anbsmRNTm6YpzXOXXTdMl6TXR-0px9vPgCLcB/s1600/convert%2Bformat%2Boutput.png> |
| Output |

**Explanation**

1. **PUT Function** is used to convert the numeric variable to character format.
2. **INPUT Function**is used to convert the character variable to sas date format
3. **FORMAT**Function is used to display the SAS date values in a particular SAS date format. If we would not use **format function,**SAS would display the date in SAS datevalues format. For example, 20588 is a sas datevalue and it is equivalent to '14MAY2016'.

Link 19---------------------------------------------------------------

# POWER OF PROC FORMAT

**Example 1 :**

Suppose you are asked to group **MSRP**variable based on the following conditions and check the number of observations falling in each groups

*Values greater than 40,000 should be labelled 'High'  
Values between 26,000 and 40,000 should be labelled 'Medium'  
Else 'Low';*

**Solution**

*proc format;  
value range  
40000-high='High'  
26000-< 40000='Medium'  
other ='Low';  
run;*

*proc freq data = sashelp.cars;  
table msrp;  
format msrp range.;  
run;*

**Example 2 :**

Same as with example 1. But you are asked to create a new variable called **TSRP**based on the following conditions applied in **MSRP**variable.

*Values greater than 40,000 should be labelled 'High'  
Values between 26,000 and 40,000 should be labelled 'Medium'  
Else 'Low'*

**Solution :**

*proc format;  
value range  
40000-high='High'  
26000-< 40000='Medium'  
other ='Low';  
run;*

*data temp;  
set sashelp.cars;****TSRP = put(msrp, range.);****run;*

**Example 3 : Subset Data**

**Method 1 :**

*data temp;  
set sashelp.cars;  
where put(msrp, range.) IN ('High' 'Medium');  
run;*

**Method 2 :**

*data temp (where = (tsrp IN ('High' 'Medium')));  
set sashelp.cars;  
tsrp = put(msrp, range.);  
run;*

**Method 3 :**

*proc sql;  
select \*,  
put(msrp, range.) as tsrp  
from sashelp.cars  
where calculated tsrp in ('High', 'Medium');  
quit;*

Link 20---------------------------------------------------------------

# DELETE EMPTY ROWS IN SAS

Suppose you want to delete empty rows from a dataset in SAS. It generally happens when we import data from external sources such as excel / csv files. It loads additional rows that are totally blank. Sometimes blank observations also affect the desired output so it's necessary to check missing cases and treat them.  
  
**Sample Dataset**  
The sample dataset looks like below. In the dataset, we have four variables - 1 character and 3 numeric. It would be used further in the example to demonstrate how to remove empty rows.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Score1** | **Score2** | **Score3** |
| Sam | 77 | 68 | 66 |
| Deepanshu | 50 |  | 89 |
|  |  |  |  |
| Shane | 55 | 78 | 89 |
| Roger | 50 | 97 | 86 |
|  |  |  |  |
| Priya | 88 | 68 | 93 |

**Create a SAS dataset**  
  
The following program creates a sample dataset in SAS.

*data outdata;  
length Name $12.;  
input Name $ Score1 Score2 Score3 ;  
infile datalines missover;  
datalines;  
Sam 77 68 66  
Deepanshu 50 . 89  
Shane 55 78 89  
Roger 50 97 86  
Priya 88 68 93  
;  
run;*

**Method I : Remove rows where all variables having missing / blank values**

*options missing = ' ';  
data readin;  
   set outdata;  
   if missing(cats(of \_all\_)) then delete;  
run;*

**Notes :**

*1. The****MISSING=****system option is used to display the missing values as a single space rather than as the default period (.)****options missing = ' ';*** *2. The CATS function concatenates the values. It also removes leading and trailing blanks.****cats(of \_all\_) - Concatenate all the variables****3.****missing(cats(of \_all\_))****-****Identifies all the rows in which missing values exist in all the variables.***

**Output**  
  
In the output dataset, we have 5 rows. One row is deleted from the original dataset.

|  |
| --- |
| <https://3.bp.blogspot.com/-4B8kn9XLVyo/Uw2QjwxTuYI/AAAAAAAAB0M/38b6zjqH3f4/s1600/sas+table.png> |
| SAS : Delete empty rows from a dataset |

**Method 2**

In this program, **NMISS**function checks the numeric of missing numeric values and **CMISS**checks the number of missing character values. In this code, we are telling SAS to delete records wherein both the character and numeric values are missing.

*data readin;  
set outdata;  
if nmiss( of \_numeric\_ ) and cmiss(of \_character\_) then delete ;  
run;*

**Method 3**

In the code below, we are using **COALESCE**and **COALESCEC**functions to return non-empty rows. These functions return first non-missing value. If all values are missing, it returns missing. Later we are checking whether these functions return missing or not.

*data readin;  
set outdata;  
if missing(coalescec(of \_character\_)) and missing(coalesce(of \_numeric\_)) then delete;  
run;*

**Example : Delete rows where any variable has missing values**

*data readin;  
set outdata;  
if nmiss(of \_numeric\_) OR  cmiss(of \_character\_) > 0 then delete;  
run;*

In this case, we are using OR operator to check if any of the variable has missing values. **It returns 4 observations.**Check out the output below -

|  |
| --- |
| <https://3.bp.blogspot.com/-6F_9q5TMdfo/WMBXut9cu_I/AAAAAAAAF84/TiOZb8ud3uUxWkrxHEXMXzjsHXNqaszVACLcB/s1600/Output_Delete_Empty_Rows.png> |
| Output |

Link 21---------------------------------------------------------------

# FIRST. AND LAST. VARIABLES

**FIRST.VARIABLE** assigns the value of 1 for the **first** observation in a BY group and the value of 0 for all other observations in the BY group.

**LAST.VARIABLE** assigns the value of 1 for the **last** observation in a BY group and the value of 0 for all other observations in the BY group.

***Note : Data set must be sorted BY group before applying FIRST. and LAST. Variables.***

**SAMPLE DATA SET**  
  
Suppose you have a dataset consisting 3 variables and 12 observations. The variables are ID, Name and Score. The variable ID is a grouping variable and it contains duplicates.

|  |  |  |
| --- | --- | --- |
| **ID** | **Name** | **Score** |
| 1 | David | 45 |
| 1 | David | 74 |
| 2 | Sam | 45 |
| 2 | Ram | 54 |
| 3 | Bane | 87 |
| 3 | Mary | 92 |
| 3 | Bane | 87 |
| 4 | Dane | 23 |
| 5 | Jenny | 87 |
| 5 | Ken | 87 |
| 6 | Simran | 63 |
| 8 | Priya | 72 |

**Create this data set in SAS**  
  
The program below creates the dataset in SAS. Copy the program below and paste it into SAS program editor and run/submit it.  
Use **PROC SORT**to sort the data set by ID. It is required to sort the data before using first. and last. variables.  
**Note : FIRST./LAST.** variables are **temporary variables**. That means they are not visible in the newly created data set. To make them visible, we need to create two new variables. In the program above, i have created First\_ID and Last\_ID variables.

|  |
| --- |
| <https://2.bp.blogspot.com/-TuV3T6VK5KM/UlMRyyK4AYI/AAAAAAAABPc/z-5QXKUCjRc/s1600/readin11-duplicate.png> |
| SAS : FIRST. vs. LAST. Variables |
|  |

**PROC SORT DATA = READIN;**

**BY ID;**

**RUN;**

**DATA READIN1;**

**SET READIN;**

**BY ID;**

**First\_ID= First.ID;**

**Last\_ID= Last.ID;**

**RUN;**

**How it works**  
  
FIRST.variable = 1 when an observation is the first observation in each group values of variable ID.  
FIRST.variable = 0 when an observation is not the first observation in each group values of variable ID.

LAST.variable = 1 when an observation is the last observation in each group values of variable ID.  
LAST.variable = 0 when an observation is not the last observation in each group values of variable ID.

*When****FIRST.variable = 1****and****LAST.VARIABLE = 1****, it means there is only a single value in the group. (See ID = 4 in the above data for reference)*

**Selecting First Observation within a Group**  
  
Suppose you need to select only the first observation among a group of observations. It is very easy to do it with IF statement. The IF statement subsets data when IF is not used in conjunction with THEN or ELSE statements.

*PROC SORT DATA = READIN;  
BY ID;  
RUN;*

*DATA READIN1;  
SET READIN;  
BY ID;****IF FIRST.ID****;  
PROC PRINT;  
RUN;*

|  |
| --- |
| <https://4.bp.blogspot.com/-ufWquvIwzOA/V1xksun4hoI/AAAAAAAAElc/jNIbBjKbcqY2hZdN7GocHSNd93XYL4pcwCLcB/s1600/error.png> |
| Output : First. Variable |

**Note :**It returns first observation among values of a group (total 7 observations).  
  
**Selecting Last Observation within a Group**  
  
Suppose you are asked to include only last observation from a group. Like the previous example, we can use last. variable to subset data.

*PROC SORT DATA = READIN;  
BY ID;  
RUN;*

*DATA READIN1;  
SET READIN;  
BY ID;****IF LAST.ID****;  
PROC PRINT;  
RUN;*

|  |
| --- |
| <https://3.bp.blogspot.com/-aQoAYo5kkEs/WF-cLHgRwyI/AAAAAAAAFuA/ZQDsUpTeypMu1xFY5KGHRTuby0tlViQXACLcB/s1600/Last%2BVariable%2BSAS.png> |
| SAS : Last. Variable |

**Q. Can we use WHERE instead of IF with First. and Last. Variables?**

*No. WHERE statement cannot be used with First. and Last. Variables. It is because WHERE statement requires variables already be created in the dataset before processing.*

**Generate Serial Number in a Group**  
  
Suppose you need to create serial numbers among a group of observations. See the snapshot below -

|  |
| --- |
| <https://4.bp.blogspot.com/-yYTKVZiNQKw/V1xr5qXT2bI/AAAAAAAAEls/a0ZSBQucE48-7dU1zoaUs0LNUSbS-jkLQCLcB/s1600/error.png> |
| Generate Serial Number in a Group |

*Data temp;  
set readin;  
by ID;****if first.id then N = 1****;****else N +1****;  
proc print;  
run;*

In the above program, we are setting N=1 when it is the first value of a group i.e. **ID**. Otherwise adding 1 to N. **The N+1 implies N = N + 1**in BY group processing**.**When there is a second observation in a group, N+1 adds 1 to N=1 so N becomes 2. It further increments by 1 when there is third observation in the group and so on.

Link 22---------------------------------------------------------------

# IDENTIFYING AND STORING UNIQUE AND DUPLICATE VALUES

data readin;

input ID Name $ Score;

cards;

1 David 45

1 David 74

2 Sam 45

2 Ram 54

3 Bane 87

3 Mary 92

3 Bane 87

4 Dane 23

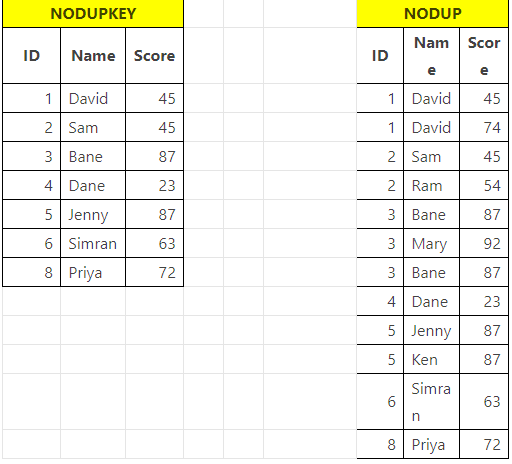
5 Jenny 87

5 Ken 87

6 Simran 63

8 Priya 72

;

Run; 

There are several ways to identify unique and duplicate values:  
  
**1. PROC SORT**  
  
In **PROC SORT**, there are two options by which we can remove duplicates.  
  
**1. NODUPKEY Option               2. NODUP Option**  
  
The **NODUPKEY** option removes duplicate observations where value of a variable listed in **BY** statement is repeated while **NODUP** option removes duplicate observations where values in **all the variables** are repeated (identical observations).

The difference between these two options are explained in detail below with SAS codes-

|  |  |
| --- | --- |
| **NODUPKEY**  **PROC SORT DATA** = readin NODUPKEY;  BY ID;  **RUN**; | **NODUP**  **PROC SORT DATA** = readin NODUP;   BY ID;  **RUN**; |

PROC SORT DATA = readin NODUP;

BY \_all\_; /\*will delete all duplicate value wheater it is in row 3 and row 5\*/

RUN;

Use the**DUPOUT=** option with NODUPKEY (or NODUP) to output duplicates to the specified SAS data set:

PROC SORT DATA = readin NODUPKEY;

DUPOUT= readin1;

BY ID;

RUN;

Link 23---------------------------------------------------------------

# DETAILED EXPLANATION OF PROC MEANS

**PROC MEANS** is one of the most common SAS procedure used for analyzing data. It is mainly used to calculate descriptive statistics such as mean, median, count, sum etc. It can also be used to calculate several other metrics such as percentiles, quartiles, standard deviation, variance and sample t-test.  
 **Uses of PROC MEANS**

1. Analyze numeric or continuous variables
2. Analyze numeric variables by group(s)
3. Identifies outlier or extreme values
4. Hypothesis Testing with Sample T-test

*Proc Means Data = test;  
Var q1 - q5;  
Run;*

|  |  |
| --- | --- |
| **Statistical Option** | **Description** |
| N | Number of observations |
| NMISS | Number of missing observations |
| MEAN | Arithmetic average |
| STD | Standard Deviation |
| MIN | Minimum |
| MAX | Maximum |
| SUM | Sum of observations |
| MEDIAN | 50th percentile |
| P1 | 1st percentile |
| P5 | 5th percentile |
| P10 | 10th percentile |
| P90 | 90th percentile |
| P95 | 95th percentile |
| P99 | 99th percentile |
| Q1 | First Quartile |
| Q3 | Third Quartile |

**Other Statistical Options**

|  |  |
| --- | --- |
| **Statistical Option** | **Description** |
| VAR | Variance |
| RANGE | Range |
| USS | Uncorr. sum of squares |
| CSS | Corr. sum of squares |
| STDERR | Standard Error |
| T | Student’s t value for testing Ho: md = 0 |
| PRT | P-value associated with t-test above |
| SUMWGT | Sum of the WEIGHT variable values |
| QRANGE | Quartile range |

**Limit Descriptive Statistics**

Suppose you want to see only two statistics - number of non-missing values and number of missing values.

*Proc Means Data = test****N NMISS****;  
Var q1 - q5 ;  
Run;*

**N** refers to number of non-missing values and **NMISS** implies number of missing values.

|  |
| --- |
| <https://4.bp.blogspot.com/-W32y-qzr_ks/WJDf-Wkjh9I/AAAAAAAAF2w/MX45Aoa0O8ETQeF-z16bKX1ZQ4JlHVFOACLcB/s1600/proc%2Bmeans1.png> |
| PROC MEANS - Missing |

*Proc Means data = test N NMISS****NOLABELS****;  
Var q1 - q5;  
Run;*

**Group the analysis**

Suppose you want to group or classify the analysis by Age. You can use the **CLASS** statement to accomplish this task. It is equivalent to **GROUP BY in SQL.**

*Proc Means data = test N NMISS NOLABELS;****Class****Age;  
Var q1 - q5;  
Run;*

You can use **NONOBS**option to delete N Obs column from the Proc Means table.

*Proc Means data = test N NMISS NOLABELS****NONOBS****;  
Class Age;  
Var q1 - q5;  
Run;*

**Change Sorting Order**

The **DESCENDING** option to the right of the slash in the first CLASS statement instructs PROC MEANS to analyze the data in **DESCENDING** order of the values of Age.

*Proc Means Data = test;  
Class Age / descending;  
Var q1 - q5 ;  
Run;*

Instead of displaying the results in “sort order” of the values of the Classification Variable (s) you specified in the CLASS Statement, order the results by frequency order using the **ORDER=FREQ** option in the CLASS Statement.

*Proc Means Data = test N;  
Class Age****/ Order = FREQ****;  
Var q1 - q5 ;  
Run;*

You can order the results by user defined format of  a variable specified in the CLASS statement using the **ORDER=FORMATTED** option in the CLASS Statement.

*Proc Means data = test N MEAN;  
Class Age / Order = formatted;  
Format Age Age.;  
Var q1 - q5;  
Run;*

**Grouping and Output in Separate Tables**  
  
Suppose you want to analyze variables Q1 - Q5 by variable AGE and want the output of each levels of AGE in **separate tables**. You can use **BY statement**to accomplish this task. See the example below-  
  
**Make sure** you sort the data before using BY statement.

*proc sort data= test;  
by age;  
run;*

*proc means data = test;  
by age;  
var q1 - q5 ;  
run;*

***Difference between CLASS and BY statement***  
  
The CLASS statement returns analysis for a grouping (classification) variable in a **single table**whereas BY statement returns the analysis for a grouping variable in **separate tables.**Another difference is CLASS statement does not require the classification variable to be pre-sorted whereas BY statement demands sorting.

|  |
| --- |
| <https://2.bp.blogspot.com/-cUQaIfxNXaU/WDHs6wzjUzI/AAAAAAAAFoQ/YjSCZCl6g_wnSUik3D-GUgFGkYrg1Ta5wCLcB/s1600/proc%2Bmeans%2Boutput.png> |
| PROC MEANS Output |

**Save output in a data set**  
  
You can use **NOPRINT** option to tell SAS not to print output in output window.

*Proc Means data = test****NOPRINT****;  
Class Age / Order = formatted;  
Format Age Age.;  
Var q1 - q5;****Output out = readin mean= median = /autoname;****Run;*

In the above code, readin is a data set in which output will be stored. The **MEAN= MEDIAN=** options tells SAS to generate mean and median in the output dataset. The **AUTONAME** Option automatically assigns unique variable names in the Output Data Set “holding” the statistics requested in the **OUTPUT** statement.

You can use **AUTOLABEL** option to automatically assigns unique label names in the Output Data Set “holding” the statistics requested in the **OUTPUT** statement.

*Proc Means Data = test noprint;  
Class Age ;  
Var q1 q2;  
Output out=F1 mean=  / autoname****autolabel****;  
Run;*

You can specify variables for which you want summary statistics to be saved in a output data set.

*Proc Means Data = test noprint;  
Class Age ;  
Var q1 q2;  
Output out=F1 mean(q1)= median(q2)= / autoname;  
Run;*

You can give custom names to variables stored in a output data set.

*Proc Means Data = test noprint;  
Class Age;  
Var q1 - q5 ;****Output out=F1 mean=\_mean1-\_mean5 median=\_median1-\_median5;****Run;*

**DROP = , KEEP = option**  
  
We can use DROP and KEEP options to remove or keep some specific variables.

*Proc Means Data = test noprint;  
Class Age;  
Var q1 - q5 ;  
Output out=F1****(drop = \_type\_ \_freq\_)****mean=\_mean1-\_mean5 median=\_median1-\_median5;  
Run;*

**WHERE Statement**  
  
The WHERE statement is used to filter or subset data. In the code below, we are filtering on variable Q1 and telling SAS to keep only those observations in which value of Q1 is greater than 1.

*Proc Means Data = test noprint;****Where Q1 > 1;****Class Age;  
Var q1 - q5 ;  
Output out=F1(drop= \_FREQ\_) mean= median= / autoname;  
Run;*

Like WHERE statement, we can use **WHERE= OPTION** to filter data. See the following program -

*Proc Means Data = test****(Where=( Q1 > 1))****noprint;  
Class Age;  
Var q1 - q5 ;  
Output out=F1(drop= \_FREQ\_) mean= median= / autoname;  
Run;*

Link 24---------------------------------------------------------------

# USE OF MULTIPLE SET STATEMENTS

DATA dat1;

INPUT id v1 v2;

CARDS;

1 10 100

2 15 150

3 20 200

;

DATA dat2;

INPUT id v3 v4;

CARDS;

1 1000 10000

2 1500 15000

3 2000 20000

4 800 30000

;

RUN;

DATA dat3;

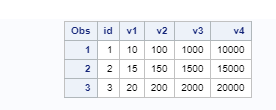
set dat1;

set dat2;

RUN;

proc print data=dat3;

run;



/\*the record which is not matching will drop\*/

Link 25---------------------------------------------------------------

**SAS MERGING TUTORIAL**

This tutorial is designed for beginners who want to get started with merging in SAS. It explains different types of joins with **MERGE**statement. Also it highlights some special topics related to merging.  
  
**Create 2 sample datasets**

*Data A;  
Input ID Name$ Height;  
cards;  
1 A 1  
3 B 2  
5 C 2  
7 D 2  
9 E 2  
;  
run;*

*Data B;  
Input ID Name$ Weight;  
cards;  
2 A 2  
4 B 3  
5 C 4  
7 D 5  
;  
run;*

**Important Steps when using MERGE Statement**  
  
**Step 1 : Both the data sets must be SORTED by the variable you want to use for merging**  
  
**Step 2 : The variable you want to use for merging must have same name in both the datasets**  
  
  
**Let's merge dataset A and B**  
  
First, Sort both the datasets with PROC SORT. See the code below -

*proc sort data = a;  
by id;  
run;  
proc sort data = b;  
by id;  
run;*

**Next Step :** Use **MERGE** statement to merge the datasets by the variable **ID**.

*data dummy;        
merge a (in=x) b(in=y);  
by id;  
a = x;  
b = y;  
run;*

**What is IN= in the above code?**  
  
The **IN= option** tells SAS to create a flag that has either the value 0 or 1. If the observation does not come from the dataset, then the flag returns 0. If the observation comes from the data set, then the flag returns 1. Since the IN= option creates **temporary variables,**we need to create permanent variables so that we can see the flag in the dataset. With this lines of code **"a = x; b = y;",**we tell SAS to create two variables named **a, b** and put the same values as stored in variables x and y. You can assign any name you want, not just a.b. See the Output shown in the image below -

|  |
| --- |
| <https://4.bp.blogspot.com/-Qp1TdpTEgFQ/V8XbOyDMY7I/AAAAAAAAFRQ/f0yDDtjqoDYr40wpSFlubJIitsU3MJs1wCLcB/s1600/merge1.png> |
| Output |

*In the above image, the****highlighted yellow rows****are the rows that are common in both the datasets. Hence, the values are 1 in variables A and B. The value 1 in variable A implies these rows come from dataset A and 0 implies these rows do not come from dataset A. The same logic holds for variable B. When variable B has 1, it means these rows come from dataset B.*

**If 'BY' Statement is NOT INCLUDED in MERGING?**  
  
The BY Statement tells SAS to **match records** based on the common variable you specify. Without the 'BY' statement, it does not perform matching of records. What would happen? Observations are combined based on their relative position in each data set. For example, observation one from the first data set combines with observation one of the second data set, the second observation from the first data set combines with the second observation from the second data set, and so on.

|  |
| --- |
| <https://1.bp.blogspot.com/-bfWWeYRaLcM/V8Xjz2MYf7I/AAAAAAAAFRg/JsrjClXjoV4pRHkj6qui9aAtbDxARd3bgCLcB/s1600/merge12.png> |
| MERGE without BY Statement |

*If there is a common variable in the two datasets, the value is overwritten by the value in the right dataset. Since ID and Name are the common variables, the values are overwritten by dataset B.*

*The number of observations in the combined dataset is equal to the number of observations in the dataset with largest number of observations. For example, dataset A has 5 observations and dataset B has 4 observations so final data would have 5 observations.*

**Types of JOINS**

**1. Inner Join (rows common to both tables)**

It returns rows common to both tables (data sets). In the final merged file, number of columns would be (Common columns in both the data sets + uncommon columns from data set A + uncommon columns from data set B).

|  |
| --- |
| <https://4.bp.blogspot.com/-WC-04UZ77C8/U7BS-ElgBEI/AAAAAAAADGY/jMFHX1LGC8E/s1600/Inner+Join.png> |
| INNER JOIN |

**SAS : INNER JOIN**

*proc sort data = a;  
by id;  
run;  
  
proc sort data = b;  
by id;  
run;  
  
Data dummy;  
Merge A (IN = X) B (IN=Y);  
by ID;  
If X and Y;  
run;*

**Note :** When using **IN= option**,SAS considers "If X and Y" equivalent to "If X=1 and Y=1".

|  |
| --- |
| https://images-blogger-opensocial.googleusercontent.com/gadgets/proxy?url=http%3A%2F%2F4.bp.blogspot.com%2F-P8lPmyHRN3E%2FU7BW8tRjV2I%2FAAAAAAAADGw%2FT5y6hCMqkDk%2Fs1600%2FInner%2BJoin.png&container=blogger&gadget=a&rewriteMime=image%2F* |
| Data Step Merge : INNER JOIN Example |

**Explanation**

*Since the above case is of****INNER JOIN,****Data Step Merge returns values 5 and 7 which are common in variable ID of both the datasets.*

**2. Left Join (Return all rows from the left table, and the matched rows from the right table)**

It returns all rows from the left table, and the matched rows from the right table.

|  |
| --- |
| <https://3.bp.blogspot.com/-rfuqYcitAvU/U7BeDPD5KAI/AAAAAAAADHY/Afn2a0-A_pw/s1600/left+join.png> |
| LEFT JOIN |

*proc sort data = a;  
by id;  
run;  
  
proc sort data = b;  
by id;  
run;  
  
Data dummy;          
Merge A (IN = X) B (IN=Y);  
by ID;  
If X ;  
run;*

**Note :**When you use **IN= option**, SAS considers "If X" equivalent to "If X=1". We can use either of the If statement.

|  |
| --- |
| https://images-blogger-opensocial.googleusercontent.com/gadgets/proxy?url=http%3A%2F%2F4.bp.blogspot.com%2F-ttiT0JCAgpI%2FU7BfvyMnxJI%2FAAAAAAAADHk%2FZkv8Sb2VaBk%2Fs1600%2Fleft%2Bjoin.png&container=blogger&gadget=a&rewriteMime=image%2F* |
| Data Step Merge : LEFT JOIN Example |

**Explanation**

*Since the above case is of****LEFT JOIN,****Data Step Merge returns all observations from dataset A with matching rows from dataset B.*

**3. Right Join (Return all rows from the right table, and the matched rows from the left table)**

It returns all rows from the right table, and the matched rows from the left table.

|  |
| --- |
| <https://3.bp.blogspot.com/-QZYC51i2SQY/U7BnNjQ6oxI/AAAAAAAADH0/_GBXz0MnDUI/s1600/right+join.png> |
| RIGHT JOIN |

*proc sort data = a;  
by id;  
run;  
  
proc sort data = b;  
by id;  
run;  
  
Data dummy;          
Merge A (IN = X) B (IN=Y);  
by ID;  
If Y ;  
run;*

**Explanation**

*Since the above case is of****RIGHT JOIN,****Data Step Merge returns all observations from dataset B with matching rows from dataset A.*

**4. Full Join (Return all rows from the left table and from the right table)**

It returns all rows from the left table and from the right table.

|  |
| --- |
| <https://1.bp.blogspot.com/-4kJL85l1BsI/U7BrRWlQkDI/AAAAAAAADIQ/LIJdrcXHHX8/s1600/Full+joinss.png> |
| FULL JOIN |

*proc sort data = a;  
by id;  
run;  
  
proc sort data = b;  
by id;  
run;  
  
Data dummy;          
Merge A B;  
by ID;  
run;*

**Note :**Since the **FULL JOIN** is the default type of JOIN in MERGE Statement, it does not require temporary variables with IN option.  
  
**Explanation**

*Since the above case is of****FULL JOIN,****Data Step Merge returns all observations from dataset A and B.*

**Different Length of BY variable**  
  
When we merge datasets with BY variable having different lengths, the length of the BY variable used during matching is determined by the left-hand side dataset in the merge. If length of dataset A is shorter than B, it may return zero records. **Solution**- Include bigger length of the common variable with **LENGTH**Statement before MERGE statement.

*data dummy;****length ID 8;*** *merge a b;  
by id;  
run;*

**Special Cases**

**1. If both the tables (data sets) have similar variable name (other than primary key), Data Step MERGE statement would take values of the common variable exist in the TABLE2 (Right table).**

**2. If primary key in both the tables (data sets) have duplicate values, Data Step MERGE statement would return a maximum number of values in both the tables. For example, Table 1 has 3 1's and Table 2 has 2 1's, Data Step Merge would return 3 1's. It is called 'One-to-Many Merge'.**

**See the special case shown in the image below -**

|  |
| --- |
| <https://1.bp.blogspot.com/-SMEUuSRexAw/V8W-acPb2gI/AAAAAAAAFQo/VeuTi0MH_okky8wqiRto1XHN1yJneRivwCLcB/s1600/merged.png> |
| Special Cases : Data Step Merge |

In this case, dataset A contains two 5s and dataset B contains three 5s. When we merged these two tables, it returns three 5s which is maximum number of 5s in both the dataset A and B.

**Another Important Point -**Did you notice the variable "name" exists in both the datasets A and B? In this example, the variable "name" is NOT a primary key to merge the tables. It is the variable "id" which is the primary key to merge these tables. When we merged the tables, DATA STEP MERGE takes values of variable "name" from dataset B.

**SAS Code for the above special case**

*data a;  
input id name$ height;  
cards;  
1 a 1  
3 b 2  
5 a 2  
5 b 3  
7 d 2  
9 e 2  
;  
run;  
data b;  
input id name$ weight;  
cards;  
2 a 2  
4 b 3  
5 d 4  
5 e 5  
5 f 6  
7 f 5  
;  
run;*

*data c;  
merge a (in=x) b(in=y);  
by id;  
if x;  
proc print;  
run;*

**Q. Do the "Special Cases" explained above hold true for all types of joins?**

Answer is **YES**. It holds true for all the types of joins.

**How to check whether you merged correctly?**

Link 26---------------------------------------------------------------

# MANY TO MANY MERGE

In SAS, many-to-many merges are handled very differently via Data Step MERGE and PROC SQL JOIN.

***Let's take an example -***

Suppose you have two data sets. You want to merge both the data sets but there are duplicate values in the common variable (ie. primary key) of any or both of the datasets.

|  |
| --- |
| <https://1.bp.blogspot.com/-YGXcmCChyGg/V1hKXeQPOjI/AAAAAAAAEgE/AVMgcFYXQIcC7IFgg_R2V6ioVY2bacbcQCLcB/s1600/optimized-fscs.png> |
| Many to Many Merging |

**Data Step Merge**

The DATA step Merge does not handle many-to-many matching very well. When we perform many to many merges. the result should be a cartesian (cross) product of matching observations. For example, if there are three records that match from one contributing data set to two records from the other, the resulting data set should have 3 × 2 = 6 records.

***Data Step MERGE does not create a cartesian product in case of a many-to-many relationship.******It will return number of records for a duplicate value equal to maximum number of the duplicate value in both the table.***

**SAS Code -**

*data dat1;*

*input ID Info;*

*cards ;*

*1 3123*

*1 1234*

*2 7482*

*2 8912*

*3 1284*

*;*

*run;*

*data dat2;*

*input ID Info2;*

*cards ;*

*1 4444*

*1 5555*

*1 8989*

*2 9099*

*2 8888*

*3 8989*

*;*

*run;*

*data combined;*

*merge dat1 dat2 ;*

*by ID;*

*run;*

|  |
| --- |
| <https://4.bp.blogspot.com/-deaCqrFZjsU/VoGFNlZNrEI/AAAAAAAAD4s/_A722ViYThQ/s1600/merge.png> |
| Output : Merge |

**Note :**In this example, we have 2 1s in dat1 and 3 1s in dat2. The maximum number of 1s in both the tables is 3. So it would return 3 1s in the merged dataset.  
  
**PROC SQL JOIN**  
 **PROC SQL JOIN creates all possible combinations of matching observations in case of a many-to-many relationship.**Cartesian product is a collection of all pairs of two given sets. For example, In ID variable, there are 2 1's in dat1 dataset and 3 1's in dat2 dataset, the cartesian product would be (3\*2 = 6 Observations) in the final result.

*proc sql noprint;  
create table combined2 as  
select \* from dat1 a  
join dat2 b  
on a.ID = b.ID;  
quit;*

***See the output shown in the image below -***

|  |
| --- |
| <https://3.bp.blogspot.com/-p3MBE4OteR4/V8X5DvKF6RI/AAAAAAAAFRw/si0JVEPjpu4xv7-kFFGzvxt9N9uVpnPhACLcB/s1600/many%2Bto%2Bmany.png> |
| SAS : Many to Many Merging |

Link 27---------------------------------------------------------------

**COMBINING AND AGGREGATING DATA**

This tutorial explains how to combine / append two data sets in SAS. In SAS, there are various method to append data sets. It can be done with data step method, PROC SQL as well as procedure called PROC APPEND to accomplish it. It is one of the most frequently data manipulation task in analytics work. For example, you have multiple human records files from various departments of your company and you are asked to join them so that there would be a single file containing information of all the departments.  
  
**1. Concatenate two data sets (Stack data set vertically) / Appending Data Sets**

-  Let's create two data sets - Data Set I and Data Set II

*Data Dataset1;  
Input Name $ Score;  
cards;  
David 30  
Ram 25  
Sam 74  
Sandy 36  
;  
run;  
  
Data Dataset2;  
Input Name $ Score;  
cards;  
Ken 36  
Obama 74  
Raj 30  
Shyam 25  
;  
run;*

- Append / Concatenate two data sets

*Data Stack;  
Set Dataset1 Dataset2;  
Run;*

**Output**

|  |
| --- |
| <https://4.bp.blogspot.com/-1aS6xJM8rSE/VJHFQSvLb9I/AAAAAAAADXw/ej7sMZVENZs/s1600/set1.png> |
| Append or Join Data |

***Note :****The stacked data set is not sorted because we have not used****BY statement****.*

**2. Interleaving SAS Data Sets (Sorted Stacked Data Set)**  
  
Interleaving combines individual sorted SAS data sets into one sorted data set. You interleave data sets using a SET statement and a BY statement in a DATA step.

***Make sure data sets are sorted before appending datasets. Datasets can be sorted with PROC SORT.***

*proc sort data = dataset1;  
by name;  
run;  
proc sort data = dataset2;  
by name;  
run;*

*Data Stack1;  
Set Dataset1 Dataset2;****By Name;*** *Run;*

**Output**

|  |
| --- |
| <https://4.bp.blogspot.com/-IstuSEv2veM/VJHHdzgjFnI/AAAAAAAADX8/SPj4iJaGhRM/s1600/set11.png> |
| Appending and Sorting Together |

**3. PROC SQL for concatenating two data sets**  
  
**OUTER UNION CORR**keyword is used in PROC SQL to concatenate two data sets. The **CORR**tells SAS to append data sets by **name (not by column position).**

*PROC SQL;  
CREATE TABLE stackk AS  
SELECT \*  
FROM Dataset1****OUTER UNION CORR*** *SELECT \*  
FROM Dataset2  
ORDER BY Name;  
QUIT;*

*The output of PROC SQL is same as the output of previous example.*  
 **4. PROC APPEND to concatenate data sets**  
  
In PROC APPEND, the data set specified in **BASE= option** refers to a data set in which other data set would be added or appended. In log, it writes **'Appending Dataset2 to Dataset1'**for our example. After running this code, the dataset1 contains 8 records ( 4 from the original 'dataset1' file and 4 from the dataset2)

*proc append base=dataset1 data=dataset2;  
run;*

 If you want to append data and store it to another dataset, you can run PROC APPEND twice to do it. In the first PROC APPEND, it would create a base table ALLDATA (as specfied in the code below).  ***If the dataset ALLDATA does not already exist, it would be automatically created by SAS.***

*proc append base=alldata data=dataset1;  
run;  
proc append base=alldata data=dataset2;  
run;*

**Is PROC APPEND faster?**  
  
PROC APPEND is faster than SET statement or PROC SQL UNION because it only reads in the data set being appended (i.e. the dataset identified by the syntax ‘DATA=’).  
  
**Application of PROC APPEND**

*PROC APPEND is most useful when you use it in a macro. For example, you create a macro in which there is a loop for calculation of some metrics of multiple variables. In each iteration, it returns a data set which needs to be appended with the output of subsequent iterations so that once loop completes all the iterations, we would have complete data set.*

**Important Point**  
  
Suppose you have two datasets having same variable names but the length of the common variable is different, It would throw a warning and it would not append datasets. To workaround this issue, we can use **FORCE option** to append the data sets.

*proc append base=dataset1 data=dataset2****force****;  
run;*

**5. Usage of Multiple Set Statement**  
  
Instead of specifying multiple data sets in a single SET statement, we can also multiple SET statements but the result /output would be totally different. See the output shown below -

*Data Stack3;  
Set Dataset1;  
Set Dataset2;  
Run;*

**Warning : It overwrites data.**

|  |
| --- |
| <https://4.bp.blogspot.com/-1hu9zN1GJMg/VJHJKI-zwwI/AAAAAAAADYI/ryXJp_SKWcM/s1600/set111.png> |
| Multiple SET Statements |

*The question arises****" why do we use multiple set statement if it overwrites data"****. The next topic describes the application of multiple set statement.*

**Application of Multiple Set Statement**

It combines data sets by adding column from the other dataset. Unlike INNER/LEFT Joins, it does not require any primary or unique key to join two datasets.  
  
*Lets create a new data set - Data Set 3*

*Data Dataset3;  
Input Section $;  
cards;  
A  
B  
C  
D  
;  
run;*

*Data Stack3;  
Set Dataset1;  
Set Dataset3;  
Run;*

|  |
| --- |
| <https://3.bp.blogspot.com/-F1EjSTbgGAw/WAW_dLvIT-I/AAAAAAAAFh0/vNJnBmkWvh4tX3aCYK_EW8_rNnJs0KJLgCLcB/s1600/Multiple%2BSET%2BStatement.png> |
| Multiple SET Statemet |

**Append Data : Many to Many Relationship**  
  
Multiple SET statement can also produce cartesian product of two tables. It can be done with POINT= option.

*Data Stack2;  
set Dataset1;  
do i=1 to num;  
set Dataset3 nobs=num point=i;  
if i=i then output;  
end;  
run;*

|  |
| --- |
| <https://2.bp.blogspot.com/-QDbQHTWAmK4/VJHLeTCLWHI/AAAAAAAADYU/qs806S1fVMo/s1600/set1111.png> |

Link 28---------------------------------------------------------------

# RETAIN STATEMENT IN SAS

**Create Sample Data**  
  
The following program creates a sample data for demonstration -

*data abcd;  
input x y;  
cards;  
1 25  
1 28  
1 27  
2 23  
2 35  
2 34  
3 25  
3 29  
;  
run;*

**Uses of RETAIN Statement**  
  
The **RETAIN**statement simply copies retaining values by telling the SAS not to reset the variables to missing at the beginning of each iteration of the DATA step. If you would not use retain statement then SAS would return missing at the beginning of each iteration.

*The retain statement keeps the value once assigned.*

**Generate Serial Number**  
  
Suppose you need to generate a serial number (or row index number) with data step.

*data aaa;  
set abcd;  
retain z 0;  
z = z + 1;  
run;*

**We can retain implicitly by using the +1 notation.**

*data aaa;  
set abcd;  
z + 1;  
run;*

**Output Dataset**

|  |
| --- |
| <https://2.bp.blogspot.com/-uvcckjp6sjM/Vi55vD4EtfI/AAAAAAAAD04/LKb9xtrQ2AU/s1600/Input%2BData.png> |
| Output Data Set |

**Cumulative Score**  
  
Suppose you need to calculate cumulative score. In financial data, we generally need to calculate cumulative score year to date.

*data aaa;  
set abcd;  
retain z 0;  
z = z + y;  
run;*

|  |
| --- |
| <https://1.bp.blogspot.com/-eFcIy4GiwCQ/Vi588JbHcOI/AAAAAAAAD1E/Ho5T7htj7S0/s1600/Input%2BData.png> |
| Output Data Set |

**Generate Serial Number by Group**  
  
Suppose you have a grouping variable say "region" and you need to generate a row index number by region.

*proc sort data = abcd;  
by x;  
run;  
  
data aaa;  
set abcd;  
retain z;  
if first.x then z = 1;  
else z = z + 1;  
by x;  
run;*

|  |
| --- |
| <https://3.bp.blogspot.com/-FkvsGhlAvVM/WGY5akoAoDI/AAAAAAAAFwU/gR0krb0RTREVpuJNpIo0CN9jXahaPK9UgCLcB/s1600/RETAIN%2BSAS.png> |
| SAS : Retain Statement |

**Cumulative Score by Group**  
  
Suppose you need to calculate cumulative sale by product categories.

*data aaa1;  
set aaa;  
retain z1;  
if first.x then z1 = y;  
else z1 = z1 + y;  
by x;  
run;*

|  |
| --- |
| <https://4.bp.blogspot.com/-CCkhjL15_C4/WGY6HrbAMOI/AAAAAAAAFwY/zwVcVy6rnv44q5ZtMD0-JyiU98nA44cWACLcB/s1600/Cumulative%2BScore%2BSAS.png> |
| SAS : Cumulative Score |

z1 constitutes cumulative values of variable y by grouping variable x.

**Number of Unique Observations**  
  
The number of unique rows by a group can easily be calculated with PROC FREQ and PROC MEANS. The following program explains how we can calculate number of observations in a categorical variable with Data Step.

*data aaa2;  
set abcd (drop = y);  
retain z;  
if first.x then z = 1;  
else z = z + 1;  
by x;  
if last.x then output;  
run;*

|  |
| --- |
| <https://1.bp.blogspot.com/-gBkWD8PGW2I/Vi6BaAEMUWI/AAAAAAAAD1c/cLfVwLv0zFQ/s1600/unique.png> |
| Unique Count |

***Suppose you have more than 1 grouping variable***

*data temp;  
input ID ID1 Score;  
cards;  
1 1 25  
1 1 26  
1 2 27  
1 2 29  
2 1 28  
2 1 29  
2 2 31  
;  
run;*

*data temp2;  
set temp;  
by ID ID1;  
if first.ID or first.ID1 then N = 1;  
else N+1;  
proc print;  
run;*

When you have more than 1 grouping variable, we can use multiple FIRST. statements with OR operator to generate serial numbers.

Link 29---------------------------------------------------------------

Proc transpose

PROC TRANSPOSE helps to reshape data in SAS. This tutorial explains the basic and intermediate applications of PROC TRANSPOSE with examples. It's a very powerful procedure when you need to change the shape of the data. For example, you have data in vertical (long) format and you are asked to change it to horizontal (wide) format. It can be done via data step as well but it would be a complex code which takes a lot of time to write and test it. To save programming time and maintaining the accuracy of the code, we should use TRANSPOSE procedure to restructure data.

|  |
| --- |
|  |
| Transpose Data with PROC TRANSPOSE |

**Example Data Set**

Let's create sample data which is used for explaining the TRANSPOSE procedure.  
Suppose you have data for students with their marks in respective subjects. In the data set, you have three variables 'Name', 'Subject' and 'Marks'. See the table below showing this data.

|  |  |  |
| --- | --- | --- |
| **Name** | **Subject** | **Marks** |
| Samma | Maths | 96 |
| Sandy | English | 76 |
| Devesh | German | 76 |
| Rakesh | Maths | 50 |
| Priya | English | 62 |
| Kranti | Maths | 92 |
| William | German | 87 |

**Create data set in SAS**

To see this data in SAS data set format, run the following code -

data transp;

input Name $ Subject $ Marks;

cards;

Samma Maths 96

Sandy English 76

Devesh German 76

Rakesh Maths 50

Priya English 62

Kranti Maths 92

William German 87

;

run;

It creates a data set named '**TRANSP**' which is stored in **WORK**library.

**Run Simplest Form of PROC TRANSPOSE**

*proc transpose data = transp out= outdata;  
run;*

The above code creates a data set called **outdata** which contains values of variable 'Marks' stored in horizontal (wide) format. In other words, it transposes only variable i.e. Marks (which is numeric). It is because **by default, PROC TRANSPOSE transposes all numeric variables in the data set.**

**Output Data Set**

The output of the data set looks like below -

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **\_NAME\_** | **COL1** | **COL2** | **COL3** | **COL4** | **COL5** | **COL6** | **COL7** |
| Marks | 96 | 76 | 76 | 50 | 62 | 92 | 87 |

**Options in PROC TRANSPOSE**

1. The **NAME=**option allows you to change the name of the **\_NAME\_**variable. It is the name of the variable that is transposed.  
2. The **PREFIX=**option allows you to change the prefix "**COL**". It is prefix to the transposed values.

*proc transpose data = transp name=VarName prefix=Student out= outdata;  
run;*

**Observe the above code with the previous section code -**There are two changes in the code above that are : specifying name 'VarName' to the variable Name. The other change is adding a prefix 'Student' to the transposed marks.

**Output Data Set**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **VarName** | **Student1** | **Student2** | **Student3** | **Student4** | **Student5** | **Student6** | **Student7** |
| Marks | 96 | 76 | 76 | 50 | 62 | 92 | 87 |

**Statements in PROC TRANSPOSE**

1. **ID -[Move to Column Name]** It allows you to include the values of a variable as variable names in the output data set. In other words, it tells SAS to give the variable names in the output file which were observations (rows) values in a variable in the input data set. If the variable in the ID statement is numeric, an underscore will be put by default at the beginning of the variable name. Instead of a default '\_', you can use **PREFIX= option** to give a specific prefix which can be any character value.For example, you want to add 'Height' as a prefix which would create variables like 'Height20' 'Height30'.  
  
2.**BY -**It allows you to transpose data within the combination of the BY variables. The BY variables themselves aren’t transposed. The variables need to be sorted before running PROC TRANSPOSE. You can sort the variables with PROC SORT.  
  
3.**VAR -[Transpose Column]** It lists the actual data that needs to be transposed. If you do not include a VAR statement, the procedure will transpose all numeric variables that are not included in a BY statement or a ID statement. If youwant to transpose a character variable, a VAR statement is required.

**Example 2 : Give name to transposed columns**

Suppose you want to have actual students' name instead of 'Student1 Student2 etc' in the variable names. You can use ID statement to accomplish this task. Check out the code below -

*proc transpose data = transp name=VarName out= outdata;  
id****name****;  
run;*

**Output Data Set**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **VarName** | **Samma** | **Sandy** | **Devesh** | **Rakesh** | **Priya** | **Kranti** | **William** |
| Marks | 96 | 76 | 76 | 50 | 62 | 92 | 87 |

In this case, the variable **'name'**is used for naming variables.

**Example 3 : Restructure Data**

Suppose you want to change the structure of data in the manner in which the row values of the variable 'Subjects' come at top i.e. heading / variable names and marks under the respective column in the output dataset.  
In this case, we need to sort the data as we are going to use BY processing in PROC TRANSPOSE.

proc sort data = transp;

by Name;

run;

proc transpose data = transp out= outdata;

by Name;

id Subject;

var Marks;

run;

In this example, we are specifying variable**Name** in the **BY** option which means we do not want to transpose this variable.. The variable **Marks** specified in the **VAR** option implies this variable is actually transposed and shape of the data format would be changed in the output data set.

|  |
| --- |
| <https://4.bp.blogspot.com/-IXi6ZwK3CHo/WA8KwinoslI/AAAAAAAAFlQ/NQ85Q_2jvg05pD0WwLTn3uTeoffNmL6cACLcB/s1600/Output%2Bproc%2Btranspose.png> |
| Output : PROC TRANSPOSE Tutorial |

If you look at the output above, everything looks perfect except the variable '\_NAME' which is not relevant. We can eliminate this variable with **DROP=**option.

*proc transpose data = transp out= outdata (****drop=\_name\_****);  
by Name;  
id Subject;  
var Marks;  
run;*

**Is SORTING required when i use BY statement?**  
  
Answer is **No**. The **NOTSORTED**option tells SAS that data is not sorted and it is not required to sort it. If you don't specify **NOTSORTED**option, you need to sort the variable that is listed in **BY**statement.

*proc transpose data = transp out= outdata (drop=\_name\_) ;  
by Name****NOTSORTED****;  
id Subject;  
var Marks;  
run;*

**Output Data Set**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Maths** | **English** | **German** |
| Samma | 96 | . | . |
| Sandy | . | 76 | . |
| Devesh | . | . | 76 |
| Rakesh | 50 | . | . |
| Priya | . | 62 | . |
| Kranti | 92 | . | . |
| William | . | . | 87 |

See the above output. You must have observed the names are not sorted in the output data set.

**How to use Two Variables in ID Statement**

We can use **DELIMITER= option**to separate values of two variables specified in the ID statements. In this example, we have used underscore ( \_ ) as a delimiter.

*proc transpose data = transp****delimiter=\_****name=VarName out= outdata;  
id name subject;  
run;*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **VarName** | **Samma\_Maths** | **Sandy\_English** | **Devesh\_German** | **Rakesh\_Maths** |
| Marks | 96 | 76 | 76 | 50 |

**How to label the Output Variables with PROC TRANSPOSE**

The ID statement tells SAS to provide variable names to the variables after the transpose. But if you want to label these variables, you can use **IDLABEL statement**which picks labels from a variable from the input file.

*proc transpose data=temp out=outdata prefix=height;  
by id;  
var scores;  
id height;****idlabel****heightl;  
run;*

**Practice Example**

Suppose you have monthly financial data. You need to convert long formatted data to wide format.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Months** | **Revenue** | **Balance** |
| 101 | 1 | 3 | 90 |
| 101 | 2 | 33 | 68 |
| 101 | 3 | 22 | 51 |
| 102 | 1 | 100 | 18 |
| 102 | 2 | 58 | 62 |
| 102 | 3 | 95 | 97 |

**SAS Code**

data example;

input ID Months Revenue Balance;

cards;

101 1 3 90

101 2 33 68

101 3 22 51

102 1 100 18

102 2 58 62

102 3 95 97

;

**Output**

The final output should like the following table.

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **balance\_1** | **balance\_2** | **balance\_3** |
| 101 | 90 | 68 | 51 |
| 102 | 18 | 62 | 97 |

**Solution :**

*proc transpose data=example out= Output1 (drop = \_NAME\_) prefix=balance\_;  
id months;  
var balance;  
by ID;  
run;*

In this case, the variable 'Month' specified in ID statement is a numeric variable. Hence, we have added prefix 'balance\_' to make it to the desired output.  
  
**If you want to see your output looks like the data shown in the image below -**

|  |
| --- |
| <https://3.bp.blogspot.com/-RSGAMWzwhRA/ViIU24uGbeI/AAAAAAAADyc/qHsF9BaNNXQ/s1600/trans1.png> |
| PROC TRANSPOSE : Reshape Data |

proc transpose data=example out=out1 name=variable prefix=x;

by id months;

run;

In this case, the information of the 'Revenue' and 'Balance' variables are stacked to one variable. And the variable 'x1' refers to the values corresponding to it.

Link 30---------------------------------------------------------------

# COMPLETE GUIDE TO PROC UNIVARIATE(LATER)

Link 31---------------------------------------------------------------

# PROC RANK

[Deepanshu Bhalla](https://www.linkedin.com/in/deepanshubhalla) [10 Comments](https://www.listendata.com/2016/01/proc-rank.html#comment-form) [SAS](https://www.listendata.com/search/label/SAS)

This tutorial explains how to calculate rank for one or more numeric variables with **PROC RANK**. In SAS, there are multiple ways to calculate rank overall or by a grouping variable. In data step, it can be done via retain statement. SAS made it easy to compute rank with **PROC RANK**.  
  
**Create Sample Data**

*data temp;  
input ID Gender $ Score;  
cards;  
1 M 33  
2 M 94  
3 M 66  
4 M 46  
5 F 92  
6 F 95  
7 F 18  
8 F 11  
;  
run;*

**Compute rank of numeric variable - "Score"**

*proc rank data= temp out = result;  
var Score;  
ranks ranking;  
run;*

**Notes :**

1. The **OUT** option is used to store output of the rank procedure.
2. The **VAR** option is used to specify numeric variable (s) for which you want to calculate rank
3. The **RANKS**option tells SAS to name the rank variable
4. By default, it calculates rank in ascending order.

|  |
| --- |
| <https://4.bp.blogspot.com/-F8pEgoqueBQ/Vpv2wAljOFI/AAAAAAAAD8E/Nes-2t33bSM/s1600/rank.png> |
| Output |

**Reverse order of ranking (Descending)**  
  
Suppose you need to assign the largest value of a variable as rank 1 and the last rank to the lowest value. The **descending**keyword tells SAS to sort the data in descending order and assign rank to the variable accordingly.

*proc rank data= temp****descending****out = result;  
var Score;  
ranks ranking;  
run;*

**Percentile Ranking (Quartile Rank)**  
  
Suppose you need to split the variable into **four**parts, you can use the **groups option** in PROC RANK. It means you are telling SAS to assign only 4 ranks to a variable.

*proc rank data= temp descending****groups = 4****out = result;  
var Score;  
ranks ranking;  
run;*

**Note :**

*GROUPS=4 for****quartile ranks****, and GROUPS=10 for****decile ranks****, GROUPS = 100 for****percentile ranks****.*

**Ranking within BY group (Gender)**  
  
Suppose you need to calculate rank by a grouping variable. To accomplish this task, you can use the **by statement**in proc rank. It is required to sort the data before using by statement.

*proc sort data = temp;  
by gender;  
run;  
proc rank data= temp descending out = result;  
var Score;  
ranks ranking;****by Gender;*** *run;*

**How to compute rank for same values**  
  
Let's create a sample dataset. See the variable score having same values **(33 appearing twice).**

*data temp2;  
input ID Gender $ Score;  
cards;  
1 M 33  
2 M 33  
3 M 66  
4 M 46  
;  
run;*

Specify option **TIES =**HIGH | LOW | MEAN | DENSE in PROC RANK.

*proc rank data= temp2****ties = dense****out = result;  
var Score;  
ranks rank\_dense;  
run;*

1. **LOW -**assigns the smallest of the corresponding ranks.
2. **HIGH -** assigns the largest of the corresponding ranks.
3. **MEAN -** assigns the mean of the corresponding ranks **(Default Option).**
4. **DENSE -**assigns the smallest of the corresponding rank and add +1 to the next rank (don't break sequence)

***See the comparison between these options in the image below -***

|  |
| --- |
| <https://4.bp.blogspot.com/-45QJEv0Jsqs/V9MLiiAJsdI/AAAAAAAAFWM/Do4UIOd89bwDMBpBCVWLCB8k7prh6IgLwCLcB/s1600/proc%2Brank%2Bmatch.png> |

Link 32---------------------------------------------------------------

# CALCULATING PERCENTILES WITH SAS(LATER)

Link 33---------------------------------------------------------------

# COALESCE FUNCTION

**COALESCE**  
  
The **COALESCE**function is used to select the **first non-missing value** in a list of variables. In other words, it returns the first non-blank value of each row.  
 ***Let's create a sample dataset in SAS to understand COALESCE function.***  
**Sample Data**

*data temp;  
input ID x1-x4;  
cards;  
1 . 89 85 .  
2 79 . 74 .  
3 80 82 86 85  
;  
run;*

**COALESCE : First Non-Missing Value**

*data want;  
set temp;  
first\_non\_miss =****coalesce(of x1-x4)****;  
run;*

If you look at the output shown in the image below, you would find **COALESCE**returns 89 in first observation which is the first non-missing value among x1= . , x2=89 , x3=85, x4 = .

|  |
| --- |
| <https://3.bp.blogspot.com/-oBRQ06VhyJg/Vu6Ej_pDe4I/AAAAAAAAEFA/pLKZKx_fvDQ_YP6u_MkdcDgZIEArxp_iA/s1600/sample%2Bdata2.png> |
| SAS : COALESCE Function |

**We can also use COALESCE function in PROC SQL.**

*proc sql;  
select \*,****coalesce(x1,x2,x3,x4)****as first\_non\_miss  
from temp;  
quit;*

**Last Non-Missing Value**  
  
Suppose you need to calculate last non-missing value instead of first non-missing value. Unfortunately, there is no such function which returns last non-missing value. To accomplish this task, we can **reverse** a list of variables and ask SAS to calculate first non-missing value. It would be equivalent to last non-missing value. Indirectly, we are asking SAS to read variables from right to left rather than left to right.

*data want;  
set temp;  
last\_non\_miss = coalesce(of x4-x1);  
run;*

**Note :**coalesce(of x4-x1) is equivalent to coalesce(x4, x3, x2, x1).

|  |
| --- |
| <https://3.bp.blogspot.com/-zR01mBpvVXU/WEP01HNkBHI/AAAAAAAAFqA/nk6u5ZDW6qgIUyD5ToMv1qc2dpMqRf2bACLcB/s1600/COALESCE%2BLast.png> |
| Last Non-Missing Value |

In this case, COALESCE returns 85 as it is a first non-missing value (read from right to left) among x4= . , x3= 85, x2=89, x1= . .

Link 34---------------------------------------------------------------

# FUZZY MATCHING WITH SAS(LATER)

Link 35---------------------------------------------------------------

# SAS ARRAYS AND DO LOOP MADE EASY(LATER)

Link 36---------------------------------------------------------------

# PROC TABULATE EXPLAINED(LATER)

Link 37---------------------------------------------------------------

# LENGTH OF NUMERIC VARIABLES(LATER)

Link 38---------------------------------------------------------------

# CHECK NUMBER OF OBSERVATIONS IN SAS DATASET

(LATER)

Link 39---------------------------------------------------------------

# SEND SAS OUTPUT TO EXCEL

*ods excel file="c:\test.xlsx"  
options(start\_at="B5“  
tab\_color="red"  
absolute\_row\_height="15"  
embedded\_titles="yes");  
  
ods text="Sales report for company X";  
proc print data=sashelp.orsales;  
title "Sample title showing new  
features";  
run;*

Link 40---------------------------------------------------------------

# VARIABLE NAME HAVING SPACES OR SPECIAL CHARACTERS

This article may be an eye-opener for you if you think a variable name cannot contain blanks or special characters except for the underscore in SAS. In this article, we would learn how we can read a variable whose name having spaces or special characters. Also, how to deal a variable name starts with a number. It also covers the same case with a dataset (table).  
  
**Why do we need to have spaces in a variable name?**

If you use teradata or any other database, you would encounter this problem very soon if you have not encountered it yet. Many times, database column contains blanks or special characters. To read them in SAS, we need to know how to read variables having spaces in their names.  
  
It is also required when we transpose our variables and the variable whose values name the transposed variables in the output data set contains special characters.  
  
  
**Let's create a sample data**

*data temp;  
input var1;  
cards;  
1  
2  
;  
run;*

**Rename the variable 'var1'  to 'variable one';**

***options validvarname=any;*** *data temp2;  
set temp;****rename var1 = 'variable one'n;*** *run;*

The**options validvarname=any;**tells SAS to allow you to have variable name begin with or contain spaces, special characters or numbers.  
  
**Additionally,** we need to put variable name having spaces in **quotes**followed by the letter **n**.

Q. If i don't use VALIDVARNAME=ANY option and use only 'variable one'n , how SAS would take it?  
  
Sol : SAS would return an error "variable name is not valid" as SAS by default cannot contain blanks or special characters.

|  |
| --- |
| <https://4.bp.blogspot.com/-5p5m2QVzD6M/V8h4PyaJ0BI/AAAAAAAAFSU/ANdNgTlXr1IX4cBl9vDt3P0w47GEZCoNQCLcB/s1600/spaces.png> |
| SAS : Variable Name having Spaces |

**Can variable name starts with a number?**

Yes, follow the code below -

***options validvarname=any;*** *data temp2;  
set temp;****rename var1 = '1variable'n;*** *run;*

**How about reading a dataset whose name having spaces?**

The option **VALIDMEMNAME= EXTEND**allows you to read or access dataset (table) whose name having spaces or special characters. In addition, we also need to put **name of variable in quotes** followed by the letter **n**.

*options****VALIDMEMNAME=EXTEND****;  
proc print data=****'price data'n****;  
run;*

Link 41---------------------------------------------------------------

**SPEED UP SAS CODE WITH INDEX(LATER)**