1-proc sql;

describe table score;

2- proc sql;

select **mean(score1,score2,score3) as X format 4.1**, *here we are saying calculated the mean with a format of 4.1*

"Fixed String" as Y, *here we are putting a column with a fixed value with a variable name*

'01/01/2022' as Z label='hello', *here simply we are assigning a value with the variable Z but with a label . P.S: when the label value is* ***#*** don’t put any label

from score;

3- proc sql;

select mean(score1,score2) as X1 format 4.1,

mean(score2,score3) as X2 format 4.1,

mean(score3,score1) as X3 format 4.1,

(calculated X1 - Calculated X2 - Calculated X3) as X4 format 4.1,

name

from score;

the Calculated keywork mean that we need to use the variable that have been created in the query in the same query

4-

proc sql;

select round(mean(score1,score2,score3),.1) as X1 format 4.1,

case when Calculated X1 < 60 then 'F'

when Calculated X1 >= 60 and Calculated X1 < 70 then 'D'

when Calculated X1 >= 70 and Calculated X1 < 80 then 'C'

when Calculated X1 >= 80 and Calculated X1 < 90 then 'B'

when Calculated X1 >= 90 then 'A'

else 'Absente'

end as GRADE from score where Calculated GRADE ='A';

5 - coalesce(score1,0) or case when score1 = . then 0 else score1 end as score1,

Here if score1 is missing we are replacing it with 0

6- proc sql;

select name,gender,mean(score1,score2,score3)

from score

order by 3 asc; it means we are sorting by column 3 (the mean of scores)

In the order we can also include a variable that is not available in the select but it should be in the table

7-

proc sql;

select \* from score

where score1 **is missing** or score2 is missing or score3 is missing;

proc sql;

select \* from score

where score1 **is not missing** or score2 is missing or score3 is missing;

proc sql;

select \*

from score

where mean(score1,score2,score3) **between 70 and 100**;

proc sql;

select \*

from score

where **name like 'T%' or name like '\_\_\_a';-> here we mean the name starts by T or the name end by a but with 3 chars before it(3 underscores)**

proc sql;

select \*

from score

where name eqt 'Ja';-> it’s like seeing if the name have a prefix

8- proc sql;

select

count(\*), -> we count the number of rows

count( gender), -> we count the number of row with non missing variable gender

count(distinct gender), -> we count the number of row with distinct values of variable gender

sum(score1),

mean(score1) from score;

quit;

if we use aggregate function without mention any other variables that is not an aggregate , our date will be aggregate and since we don’t have group by it will in one line

9-

Groupby in SAS:

If sum(v1) then it’s an aggregate function,if sum(v1,v2,v3) it’s not an aggregate function anymore

An aggregate function operates only on one variable or a expression

So if we do a groupby with v1 and we use v2 outside of the groupby in the where statement sas will display all the rows and if we have aggregation function they will be aggregate as specified in the groupby statement (v1=x value=y,v1=x1 value=y1) and they will be assigned to rows with respect to the group that they belong by binding group of the row to the aggregation function calculated with respect to that group

Example:

Original data set:

X1 Y1 1

X2 Y2 3

X1 Y3 4

X2 Y4 10

Group by statement with select value outside of groupby and doesn’t belong to aggregation function

X1 Y1 1 5 (sum of line 1 and ligne 3)

X2 Y2 3 13(sum of line 2 and ligne 4)

X1 Y3 4 5(sum of line 1 and ligne 3)

X2 Y4 10 13(sum of line 2 and ligne 4)

10-

Both of them give the same results,

The first sql without the where will simply attached the second table to each row of the first; so basically we have length(table1)\*length(table2)

proc sql;

create table table3 as

select g.stu\_id label='Student ID',

score1 label = 'Score at the First Test',

score2 label = 'Score at the Second Test',

score3 label = 'Score at the Third Test',

gender label = 'Gender',

g.name label = 'Name'

from table1 as g, table2 as ng

where g.stu\_id=ng.stu\_id

order by gender,name;

proc sql;

select g.stu\_id label='Student ID',

score1 label = 'Score at the First Test',

score2 label = 'Score at the Second Test',

score3 label = 'Score at the Third Test',

gender label = 'Sex',

g.name label = 'Name'

from table1 as g inner join table2 as ng

on g.stu\_id = ng.stu\_id

order by stu\_id;

11-

**Here it’s a Cartesian product between tables, we are getting all possible combos between 2 tables**

proc sql;

select \*

from mylib.table1 as g

cross join mylib.table2 as ng

order by g.stu\_id,ng.stu\_id;

**Here we are putting 2 tables the one under the other, same concept as proc append**

proc sql;

select \*

from mylib.table1 as g

union join mylib.table2 as ng

order by g.stu\_id,ng.stu\_id;

**Here we are make something like a left join, but we are automatically detecting what are the column that matches**

proc sql;

select \*

from mylib.table1 as g

natural join mylib.table2 as ng

order by g.stu\_id,ng.stu\_id;

**proc sql feedback; is just to see details about how our query is executed**

12-proc sql;

create table mylib.table4 (stu\_id char(10), gender char(1), birthday num

informat=data9. format=date9.);

quit;

proc sql;

describe table mylib.table4;

13-proc sql outobs=10;

create table mylib.table5 as select

name format = $15. label='Name',

score1 format = mmddyy10. label='BirthDay'

from mylib.table1;

quit;

proc sql;

create table mylib.table6 like mylib.table5; -> creating a table with the same structure

quit;

proc sql;

create table mylib.table7 as select \* from mylib.table1 (drop=score3); -> dropping a specific column of the create table

quit;

14-proc sql;

insert into mylib.table6

set name='Nassim',score1=100000

set name = 'Vincent',score1=45;

select \* from mylib.table6;

quit;

proc sql;

insert into mylib.table6 -> hint we need to be careful about column position

values('Nassim Youssef',1000000)

values('Herculano',90000)

values('',.);

select \* from mylib.table6;

quit;

proc sql;

insert into mylib.table7(name,score1) -> P.S the mentioned cols her should match the one of the select

select \* from mylib.table6;

quit;

15 - proc sql;

update mylib.table7

set score1='01MAR1920'd

where name like '%N%';

quit;

or we can use

update mylib.table7

set score1 = score1 \*

case when score1 lt 60 then 1.1

when score1 between 60 and 69 then 1.2

when score1 between 70 and 79 then 1.3

else 1.9

end

16- proc sql;

delete from mylib.table7 where score1<100;

quit;

16-proc sql outobs=5; is to specify that we need to include 5 obs in our dataset

17- proc sql outobs=10;

alter table mylib.table7

add score4 num format=dollar4.0 label='Score 4'

modify score1 format=date9. label='Score 5 changed'

drop score5;

;quit;

proc sql;

drop table mylib.table8;

quit;

17- views are the same as table. To create view or drop then we use the same syntax use for the tables but instead of the keyword table we use view. Also we can use view instead of table in our select by simply putting the view name instead of the table. Select \* from table same as select \* from view\_name

18-

proc sql outobs=10;

select \*

from (select \* from mylib.table1 where score1 >60) as x

where mean(x.score1,x.score2,x.score3) >70

;quit;

19-

PROC PRINT DATA=MYLIB.table1;

TITLE "&SYSDATE9/&SYSTIME/&SYSDAY"; -> some macro variable available by SAS by default

RUN;

20- OPTIONS SYMBOLGEN; It will display the macro variable values

21- %put the value of &x &sysdate;

22- %let proc\_mean= %str(proc means data=mylib.table1;run;);

&proc\_mean;

The %str will let us put semicolons in our macro variable definition

Or

%let X=%str(;);

%let proc\_mean= proc means data=mylib.table1 &x run &x ;

23-

%LET X = mylib.table1;

TITLE "%UPCASE(&X) &X %UPCASE(testing)"; the upcase function will transform a string inside it to an uppercase value, mylib.table1 -> will be displayed as **MYLIB.TABLE1 and testing as TESTING**

24-

%scan and %substr act the same as the sas function version

25- %SYSFUNC(mdy(12,12,2020),mmddyy10.) or “%SYSFUNC(mdy(12,12,2020),mmddyy10.),”

SYSFUNC take a sas function as arg and the format of the output -> SYSFUNC( func(args), format)

ddmmyy10 is also a function

26-

%let s = score;

%let x = 1;

proc print data=mylib.table1;

var &s&x; -> same as var score1

run;

27-

%let v =score;

proc print data=mylib.table1;

var %str(&v)1 &v.1; here if we put &v1 it will be an error because v1 doesn’t exist

title “%str(&v)1 &v.1” -> same as “score1 score1”

title "&v1" -. Will fail

run; %str(&v)1 or &v.1 is the same score1, the same note when we operate inside “ “

P.S the same applies when we are dealing with libraries, because **&v.** is also considered a macro variable name so we need to put **&v..**

28- symput('V1', score1) -> it’s use to dynamically create a macro variable,

It can be used inside a dataset to transfer data between datasetp and proc,

data mylib.table3;

set mylib.table1;

if score1 = 88 then call symput('V1',sum(score1));

29-&& resolve to &

30- data mylib.table3;

set mylib.table1;

call symput('V' || left(stu\_id), score1); -> here we are creating multiple macro variables

run;

data table4;

set mylib.table3;

X=symget('V3'); -> here we are getting the macro variable by name

run;

31- proc sql;

select score1-score2,score3

into :xyz,:x -> here we are creating a macro variable that holds only the first line and the variable score1-score2 and x holds score3

from table4

;

%put \_user\_;

quit;

32-

proc sql;

select \*

into :xyz seperated by ' '

from table4

; since we have \* the :xyz will take only the variable in the table, and the separated by will simply append all values of the first variable into xyz

33- efficient way of detecting dups is throught grouping and comparing first. Last.

34 – in proc sort we can put out , so if we have nodup the out will have non duplicate

35-Options MPRINT; for printing the code, Options MLOGIC; for printing the logic in it

36 –

%macro test(data=, mean=); P.S near the equal sign we can put value by default

proc print data=&data;

run;

proc means data=&mean;

run;

%mend;

%test(mean=mylib.table1, data=mylib.table2);

Or

%macro test(data, mean);

proc print data=&data;

run;

proc means data=&mean;

run;

%mend;

%test(mylib.table1, mylib.table2); here position matters

37 –

%macro test(data=, mean=);

proc print data= %if &data ne %then &data; %else &mean; ;

run;

%mend; if else then do are used in the way we do with datastep

38- %macro test(data,start=, end=);

%do i=&start %to &end;

proc print data=&data;

title &i;

run;

%end;

%mend;

38-options MSGLEVEL=i; -> to pring info about index

data mylib.table3(index=(stu\_id/unique name)); -> normal index

set mylib.table1;

run;

data mylib.table3(index=(S\_ID=(stu\_id name))); -> S\_ID is an index as a combo of 2 varaible

set mylib.table1;

run;

proc datasets lib=mylib nolist;

modify table3;

index delete stu\_id name;

index create stu\_id/unique;

index create S\_ID=(stu\_id name);

quit;

proc sql;

create index stu\_id on mylib.table3;

drop index stu\_id from mylib.table3;

create index S\_ID on mylib.table3(stu\_id,name);

quit;

39 – union is to unified to query but when we find 2 duplicate row( exact values for all vars) we don’t put them

Union all -> we put everything.

Select \* form x (union or union all) select …

The same rules goes for the except but we get everything in the first table except the value in the second tables

The same rules goes for intersect and intersect all but we find what is in common

40 -**where exists** (select…) -> is a querry used to check if a subquery or a table have values

**Where not exists**

41-

In general we need to specify the libname.dataset.package

proc fcmp outlib=work.dataset.package; ->here we are defining a function,like substr sum mean

function day\_x(start,end);

return (end - start);

endsub;

quit;

option cmplib=(work.dataset.package); -> in order to find this function we need to include this

42-

proc fcmp outlib=work.dataset.y;

subroutine sub\_rt(in1, in2, out1, out2);

outargs out1, out2; -> from our function args we need to provide the one that will be as output

out1=in1-in2;

out2=in2 -in1;

endsub;

quit;

option cmplib=(work.dataset);

data x;

set mylib.table1;

A=.;

B=.;

call sub\_rt(score1, score2,A,B); -> here we are saying that the created column A and B will be evaluated thru this sub-routine

run;

43-

proc means data=mylib.table1 noprint;

var sale\_m1 sale\_m2 sale\_m3;

output out=summary mean=x1 x2 x3 sum=x4 x5 x6 max=x7 x8 x9;

run;

here we are simply summarizing the data set, the out= is to specify the output

the mean = x1 x2 x3 represent the means sale\_m1 sale\_m2 sale\_m3, the sum = x4 x5 x6 represent the sum of sale\_m1 sale\_m2 sale\_m3 as x4 x5 x6 columns

44 –

data mylib.table11;

set mylib.table1;

if \_N\_ = 1 then set summary; ->

run;

here since we put set inside an if all it’s variable will be retained and appended table1 variables so basically, we have table data1 with the sums in summary next to it

45-

proc import datafile='/home/u38430500/data/score\_data\_id\_partial\_score4\_extra.xlsx' dbms=xlsx

out=mylib.table1 replace;

run;

proc import datafile='/home/u38430500/data/score\_data\_id.xlsx' dbms=xlsx

out=mylib.table2(index=(SID = (stu\_id name) )) replace;

run;

data mylib.table3;

set mylib.table1;

set mylib.table2 key=SID;

if \_iorc\_ = 0 then blablabla -> means if we find match the \_iorc\_ will be zero

run;

\_error\_=0 mean suppress the error log

What this code do is trying to match the index SID (stu\_id name) with stu\_id and name of table1

46-

data mylib.table3;

if \_N\_=1 then

do;

declare hash ag (dataset : "table1"); -> defining the datasource

ag.defineKey("stu\_id"); ->defining what is the column to be taken as key

ag.definedata("score1");->defining what is the column to be taken as value

ag.defineDone();

call missing(score1);

end;

set table2;

X= ag.find();

run;

since the key is taken as stu\_id, the find will use the stu\_id of table2 as key to find the value in the hash

47- we van use format to transform data in in dataset to other value by simply using the put or the input

And the format, in this case the format will act as a value tranformer

48-

Array x [200:210] \_temporary\_ (‘value1’ ‘value2’) Can be use as lookup technique, here we create an array that is only accessible from index between 200 and 210, so if we need to map a value we simply put the code (200-210) and we get the associated value

49 –

proc format;

picture phone\_number (default=24 )

low-high = '999 999 9999' -> the nines here is to say that we have an number input that need to be drawn in this forms in this case 3digits space 3 digits space 4 digits

(prefix='+352 ');

run;

proc format;

picture bd\_formater (default=25)

low-high = '%d %B %Y' (datatype = date); **-> here %B is a directive (available in the sas docs) it’s use to extract a particular feature of the input, P.S the datatype here is to say that the number input is in the form of a date**

run;

proc print data=table3;

format phone phone\_number.;

run;

proc print data=table2;

format birthdate bd\_formater.;

run;

50-

options NOSTIMER; -> to disable CPU logging usage

Options FULLSTIMER; -> to enable having full statics

51-proc compare base=table3 compare=table4;

run;

52- the keyword length in data step doesn’t change the size of the input but forces the size of new variables

53-

data table3(compress=binary); -> is used mainly when we have specific pattern

set table2;

length default=4;

X=1234;

run;

data table4(compress=char pointobs=NO reuse=yes);-> or yes ins used to compress repeated data

set table2;

X=1234;

run;

PointObs : does not allow random access to the dataset

Reuse=yes mean sas will track free spaces and use it when new observation are added, (free space remains in it’s place when we delete observation , using compress and the reuse option we can use this free space

54-FILENAME score (

"/home/u38430500/data/score\_month1.txt",

"/home/u38430500/data/score\_month2.txt",

"/home/u38430500/data/score\_month3.txt",

"/home/u38430500/data/score\_month4.txt",

"/home/u38430500/data/score\_month5.txt");

data scoreall/view=scoreall; -> **view is simply not creating a physical, it’s like a stored query, a query that will run when do select xyz from view; it only have instruction on how to retrieve data**

infile score;

input name $ score month$;

~~run;~~

55- when we are setting a macro variable no need to specify that it’s a string

56- proc datasets library=work;

modify FR1;

index create stu\_id/unique;

index delete stu\_id;

ic create primary key(stu\_id class);

quit;

57-

proc sql;

create index stu\_id on FR;

drop index stu\_id from FR;

create unique index S\_ID on FR(stu\_id,class);

alter table FR add primary key (stu\_id,class);

quit;

58-

%macro t/parmbuff; -> means dynamic number of params

%put %sysfunc(countw(&syspbuff)); -> syspbuff is representing the vars passed by the macro function

%mend;

%t(1234,1234,1234,1234);

59- this is the basic way to a loop with macro

%local i;

%do i=0 %to 10;

%put &i;

%end;

60-

%symdel macorv1

61-

%local x; or %global x; -> should be define as null then we use %let x =… to give them value

62- this how we can increment inside macro

%local x;

%let x = 1;

%let x = %eval(&x+1);

63-

%do %until(&x = 10);

%let x = %eval(&x+1);

%put &x;

%end;

64-

%include '/home/u38430500/sasuser.v94/Program 1.sas' / source2; -> simply to include macros defined in other sas Files

65-

defineData(ALL:”yes”) -> to put all varaibles as data

x.add(key:”value”,data:value2) -> to manualy include dataset

if we didn’t specify a dataset into hash we add value manulay

hash.find(key:varname); find data with specific key

the find invocation will set variables define in the data of the hash to the dataset with respect to the key

if the key and data specify in the hash doesn’t excite as member of the final dataset we should call missing on them

66-

**IF \_IORC\_=%SYSRC** (*mnemonic*) **THEN** *executable\_statement***;**

Here simply **%SYSRC** (*mnemonic*) is making an operation for example mnemonic \_SOC mean the observation was found

67-

data mylib.empmaster; update mylib.empmaster mylib.empchanges; by empid; run;

68- %symdel to delete a macrovariable

69-

LAGi ->means getting the value of variable from current row – i

COUNT(str,word) -> to count how many time a word appear in the string

COUNTC(str,charslist) count how many time does any char from the list is in the str

COUNTW(str,delimiter) count the number of words sperated by a deliminiter by default is a space

The find functions is smillar to the count functions but it required a startindex as last arg

Findc is like countc however it will try to find a char based on the order in char list for example -> list = IM and value is mila then 2, because I is the first and M is the seconds the findc will return 2 and not 1 because it will try to search of a I before an M

Findw(str,word,delimiter,start)

P.S the modifier should always be the last arg