SAS Revision guide last updated February 26, 2023

Overview

SAS is a statistical software suite for data management, advanced analytics, multivariate analysis, business intelligence, criminal investigation, and predictive analysis. A typical SAS program is organised into blocks of code, called *steps*. Specifically,

1. The *data step*, where data creation and manipulation takes place. Example:

```
DATA want (OPTIONS);
        SET have (OPTIONS);
        ...
RUN;
```

2. And, the *proc* (or procedure) step, where data manipulation, statistical analysis and reporting takes place. Example:

```
PROC PRINT DATA = have;
        VAR variable;
RUN;
```

Statements which occur outside of the data step or proc step are called *global statements*. When these are executed, their effects continue until the end of the SAS session (an example is the LIBNAME statement, which defines a SAS library).

Data step related key words

1. `DATA` : Starts the datastep, by creating a *table* (`want` is the corresponding name)

   (a) `INPUT` : Used in creating a simple dataset, example usage: NB: Put $ after variable if it is a character

```
DATA want;
  INPUT student $ course $ mark;
  DATALINES;
Tom Physics 90
John French 85
Josh Biology 88
;
RUN;
```

   (b) `LABEL` : Assigning labels to variables
   Example usage: `DATA want; SET have; LABEL student = 'Student Name' course = 'Course name'; RUN;`

   (c) `MODIFY` : Used to quickly change records
   Example usage: `DATA want; MODIFY want(WHERE=(student='Tom')); mark=89; RUN;`

   (d) `IF-THEN-ELSE` : Used to update or create columns using an if this, then do condition (for a single statement),

```
DATA want; SET have;
IF age = 15 THEN group = 'minor' ;
ELSE IF age = 18 THEN group = 'adult'; RUN;
```

   (e) `IF-THEN-DO` : Used to update or create columns using an if this, then do condition (for multiple statements),

```
DATA want; SET have;
IF age = 15 THEN DO; group = 'minor'; educ = 'secondary'; END;
IF age = 18 THEN DO; group = 'adult'; educ = 'tertiary'; END; RUN;
```

   (f) `CALL SYMPUT()` : Produces a series of macro variables using values produced in a data step

```
DATA want;
  INPUT position $ player $;
  CALL SYMPUT(position, player);
 DATALINES;
 shortstp Ann
 pitcher Tom
 frstbase Bill
 ; RUN ;
%PUT &shortstp; >> 'Ann'
%PUT &pitcher; >> 'Tom'
```

(g) `DO macro-variable=start %TO stop <%BY increment>...` : Provides iterations/loops

```
DATA want;
  balance = 1000;
  DO i = 1 TO 4;
  balance + 100;
  END; RUN;
```

```
DATA want;
  DO _N_=1 BY 1 UNTIL (last.id);
  SET have;
   BY id;
  sum=SUM(sum,var);
  END; RUN;
```

(h) `FILENAME/INFILE` : Function to specify/read an external data file.
Usage: `FILENAME name1 '<directory>';` / `INFILE name1;`

(i) `MERGE` : Function to merge multiple tables into one (SQL full join equivalent)
Usage: `DATA want; MERGE table1 table2; BY id; RUN;`

(j) `SUM()/MEAN()/MIN()/MAX()` : Sum/mean, minimum, maximum

(k) `+/-/*/\/**` : Add/Subtract/Multiply/Divide/Power

(l) `RAND()` : Create a random value from a specified distribution.
Usage: `RAND('exponential')` : Exponential distribution, `RAND('uniform')` : Uniform distribution.

(m) `||` : Combine 2 character values.
Usage: `DATA want; text1 = 'He'; text2 = 'llo'; combinedtext = text1 || text2; RUN;`

(n) `SUBSTR(var,starting_position,number_of_characters)` : Extract partial text from a character value.
Usage: `DATA want; text1 = 'Good Day'; text2 = SUBSTR(text1, 6, 3); RUN;` text2: 'Day'

(o) `TRIM()/COMPRESS()` : To remove (trailing space)/(all spaces) from a character value
Usage: `TRIM(text1);` , `COMPRESS(text1);`

(p) `INDEX()` : Function to identify (first) position where specified text is found from a character value
Usage: `DATA want; text1 = "Good Day"; pos = INDEX(text1, "oo"); RUN;` pos: '2'

(q) `LOWCASE()/UPCASE/PROPCASE()` : Convert a string to (lower case)/(upper case)/(proper case)

(r) `ROUND()/CEIL()/FLOOR()` : Rounds to (nearest integer)/(towards ∞)/(towards $-\infty$)

(s) `LENGTH()` : Calculate the length of each string in a column

(t) `LENGTH` : Specifies the number of bytes `SAS` is to use for storing values (default is 8 bytes)

i. `$` : Specifies that the preceeding variables are character variables of type `CHAR`
ii. Discuss difference between bytes and number of characters

(u) `LOG()/EXP()/ABS()` : Logarithmic, exponential, absolute value

(v) `FORMAT` : Assign a format for a variable

```
DATA want;
  FORMAT W $char3. /* Character of length 3 */ %$
         Y 10.3    /* Number of 10 spaces for output, */
                   /* 1 space for the decimal, and 3 for digits right of decimal */
```

2. `SET` : Without any other keyword, will create the `want` table, as an exact copy of the `have` table
   (a) `DROP=/KEEP=` : Specifies which columns to drop/keep from the existing table.
   Example usage: `SET have(DROP = column1 column2 column3)`
      i. `drop = col1 -- coln` drops all columns from column1 to columnn inclusive.
      ii. `keep = col1 -- numeric - coln` keeps all numeric variables from col1 to coln (same w other datatypes)
   (b) `FIRSTOBS=` : Specifies the first observation that SAS processes
   Example usage: `SET have(FIRSTOBS = 5)`
   (c) `OBS=` : Specifies the last observation that SAS processes
   Example usage: `SET have(OBS = 1000)`
   (d) `RENAME=` : To change the name of one or more variables
   Example usage: `SET have(RENAME=(oldvar1=newvar1 oldvar2=newvar2))`
   (e) `WHERE=` : Specifies conditions to use when selecting observations
   Example usage: `SET have(WHERE = (age >= 15 AND gender = 'M'))`

Proc step related key words

1. `PROC PRINT` : Used to view tables

   Usage: `PROC PRINT DATA=have; RUN;`

2. `PROC SUMMARY` : Used for data summaries

```
PROC SUMMARY DATA=have NWAY MISSING;
        VAR Comp1 Comp2;
        CLASS type ; /* Grouping by type */
        OUTPUT OUT=want SUM=;
RUN;
```

   (a) Options (after 'have' and before ';'):
      i. `MISSING` : Treats missing values as a valid subgroup
      ii. `NWAY` : Calculates only highest level of interaction (highest _TYPE_ and _WAY_ values)
      iii. `ORDER` : Specify sort order of CLASS variables
      iv. `DESCENDING` : Arranges lowest summary levels first (default is ascending arrangement)
   (b) Optional statements (after first ';' and before 'OUTPUT OUT'):
      i. `VAR` : Variables with values (such as costs or counts)
      ii. `CLASS` : Rows to aggregate by (such as models or makes)
      iii. `BY` : ...
   (c) Options on the OUTPUT OUT:
      i. N, MIN, MAX, MEAN, STD, SUM
      ii. NMISS, PRT, VAR, RANGE, CSS, SKEWNESS, USS, CV, SUMWGT, KURTOSIS, STDERR, T
   (d) _TYPE_
      i. _TYPE_ = 0: Represents the entire data set - i.e. no distinction between classes
      ii. _TYPE_ = 1: Represents class1 (across all class1) - i.e. different rows for each val in class1
      iii. _TYPE_ = 2: Represents class2 (across all class1) - i.e. different rows for each val in class2
      iv. _TYPE_ = 3: Represents class1 within class2 (combination of both)
   (e) _WAY_

3. `PROC SQL` : Used to write SQL code

   Example usage: `PROC SQL; CREATE TABLE table1 AS SELECT ... ; QUIT;`

   Options: `PROC SQL NOPRINT` : No display output, useful when creating macro-vars i.e. `SELECT name INTO: list`

4. `PROC SORT` : To sort a table (`BY` is equivalent to SQL's `ORDER BY` , but descending/desc argument before variable)

   Usage: `PROC SORT DATA=have OUT=want; BY DESCENDING points rebounds; RUN;`

5. `PROC TRANPOSE` : To convert rows into columns in a table

```
PROC TRANPOSE DATA=have OUT=want(DROP=_NAME_);
        BY name NOTSORTED /* Here we have the row names */
        ID subject; /* Here we have the column names */
        VAR marks; /* Here we have values corresponding to a combination of (row, column) */
RUN;
```

	name	subject	marks
1	Samma	Maths	96
2	Sandy	English	76
3	Devesh	German	76
4	Rakesh	Maths	50
5	Priya	English	62
6	Kranti	Maths	92
7	William	German	87

(a) Table: have

	name	Maths	English	German
1	Samma	96	.	.
2	Sandy	.	76	.
3	Devesh	.	.	76
4	Rakesh	50	.	.
5	Priya	.	62	.
6	Kranti	92	.	.
7	William	.	.	87

(b) Table: want

Options before `OUT=` specification,

(a) `NAME=` : changes the name of the `_NAME_` variable, general name of the variables transposed

(b) `PREFIX=` : allows the prefix to the transposed values to be changed

Options after first line,

(a) `ID` : Include values of a variable as variable names in the output data set (column names)

(b) `BY` : The by variables themselves aren't transposed (row names)

(c) `VAR` : Actual data that needs to be transposed, (values corresponding to a combination of (row, column)

6. `PROC FREQ` : To compute frequency count and percentage of a variable

   `PROC FREQ DATA=want; TABLES name; RUN;` (computes the frequency of each name as a number and percentage)
   Options:

   (a) `TABLES name/NOCUM` : Does not return cumulative scores

   (b) `TABLES name * gender` : Computes frequency statistics of name for each gender, matrix form (and vice versa)

   (c) `TABLES name * gender / LIST` : Computes (b) but in list form

7. `PROC MEANS` : Essentially the same as `PROC SUMMARY`

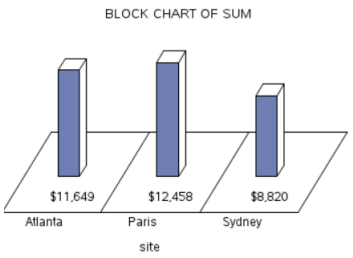
8. `PROC GCHART` : To create pie charts or histograms

```
PROC GCHART DATA=have;
TITLE "Total Sales";
FORMAT sales dollar8.;
BLOCK site / SUMVAR=sales;
RUN; QUIT;
```

(a) Code

	dept	site	quarter	sales
1	Parts	Sydney	1	7043.97
2	Parts	Atlanta	1	8225.26
3	Parts	Paris	1	5543.97
4	Tools	Sydney	4	1775.74
5	Tools	Atlanta	4	3424.19
6	Tools	Paris	4	6914.25

(b) Table: have



(c) Output

Options,

(a) `TITLE` : Title of the chart

(b) `FORMAT` : Formats values of the sales statistic

(c) `BLOCK` : Produces a block style chart

(d) `SUMVAR=` : Calculates the sum of `sales` for each variable `site`

9. `PROC REG/PROC GLM` : To perform (regression analysis)/(general linear modelling)

10. `PROC GPLOT` : To plot graphs (2-3 dimensional i.e. adding 'bubbles' for 3d for instance)

    Usage: `PROC GPLOT DATA=have; PLOT height*weight; RUN;` - run on data and see if we should make any quick changes

11. `PROC FORMAT` : To provide formats (provide value labels). These must then be assigned to each variable through a datastep i.e. `DATA want; SET have; FORMAT score score_code.; RUN;` , formats here end with a `.`

```
PROC FORMAT;
        VALUE score_code
        0.85 - HIGH = '7'
        0.75 -< 0.85 = '6'
        0.65 -< 0.75 = '5'
        0.5 -< 0.65 = '4'
        OTHER = '<4'
RUN;
```

    Options:

    (a) Labels for character labels must

        i. Start with a dollar sign i.e. `VALUE $genderlabel`

        ii. Code values on the left must be quoted i.e. `"M" = "Male";`

    (b) Single value, `1 = 'Strongly Disagree'`

    (c) Multiple values, `1,2,3 = 'Disagree'`

    (d) Many values, `1-3 = 'Disagree'`

12. `PROC REPORT` : To create a report from a data set

```
TITLE 'Player Statistics for Dallas Mavericks';
PROC REPORT DATA=have;
        WHERE team = 'Mavs';
        COLUMN conf team points;
        DEFINE conf / DISPLAY 'Conference' CENTER;
RUN;
```

    (a) `TITLE`: Creates a title for the report

    (b) `WHERE` : Filters dataset to only contain rows where team is 'Mavs'

    (c) `COLUMN` : Specifies which columns to display in the report in a certain order

    (d) `DISPLAY` : Specifies the title to use for the column called 'conf'

    (e) `CENTER` : Specifies the text to be centered in the column

13. `PROC UNIVARIATE` : To examine the distribution of data

    `PROC UNIVARIATE DATA=have; VAR points; BY team; RUN;` (stats for points variable, grouped by team variable)

14. `PROC CONTENTS` : To generate summary information about the contents of a dataset

    Usage: `PROC CONTENTS DATA=have ORDER=varnum; RUN;`

    (a) `ORDER=varnum` , lists variables in the order they appear in the dataset (default is alphabetical order)

15. `PROC DATASETS` : Efficient way to manage, manipulate and modify SAS datasets

    (a) View contents of a SAS library

    `PROC DATASETS LIB=work; RUN; QUIT;` : Outputs list of datasets with some features, found in 'work' library

    i. `MEMTYPE=data` : limits output to only show datasets

    ii. `; CONTENTS DATA=have;` : Specifies the dataset we would like to see attributes for

    (b) Combine SAS datasets

    i. General: `PROC DATASETS LIB=work; APPEND OUT=have DATA=add_to_have; RUN;`

    ii. Option: `DATA=add_to_have FORCE` : Ignores errors where columns between datasets don't exactly match

    (c) Copy, move and delete datasets

    i. Copy all datasets to new library: `PROC DATASETS; COPY IN=work OUT=new_library; RUN; QUIT;`

    ii. Copy specific datasets: `... COPY IN=work OUT=new_library; SELECT table1 table2; ...`

    iii. Move datasets bewteen libraries: `... COPY IN=work OUT=new_library MOVE; ...`

    iv. Delete specific datasets: `PROC DATASETS LIB=work; DELETE table1; RUN; QUIT;`

    v. Remove all datasets in a library: `PROC DATASETS LIB=work KILL; RUN; QUIT;`

    (d) Modifying dataset and variable attributes

```
PROC DATASETS LIB=work;
  MODIFY table1 (LABEL = 'Description of table1'); /* choosing table to modify */
  RENAME school = university; /* renaming a column */
  FORMAT height weight NUMBER7.2; /* reformatting 2 columns */
  LABEL name = 'Student Name'
        sex = 'Student Gender'; /* relabelling 2 columns */
  RUN;
  CONTENTS DATA=table1; /* to view changes after modification */
RUN;
QUIT;
```

    (e) Creating an index

    `PROC DATASETS LIB=work; MODIFY table1; INDEX CREATE weight; RUN; QUIT;` (unsure of its usefulness)

Global/other statements

1. `%LET` : To store a value or list.
   Defining: `%LET x=5;` , Use: Value of x is `&x.` , Lists: `%LET list = apple banana grape;`

2. `%MACRO ... %MEND;` : Provides systematic methods in writing code,

```
%MACRO TEST(input1,input2,output); PROC SQL;
        CREATE TABLE output AS
        SELECT a.*, b.* FROM input1 AS a LEFT JOIN input2 AS b
        ON a.id = b.id
;QUIT; %MEND;
```

   Then: `%test(table1,table2,want)` produces the table 'want' by left joining table1 and table2.

3. `LIBNAME` : Used to reference tables saved in physical directories
   Defining, `LIBNAME ref1 '<directory>';` , Referencing, `ref1.table1`

4. `%INCLUDE` : Call macros stored in other code (same effect as copying lines from other code to current program)
   `%INCLUDE "<directory>";`

5. `INTO` : Stores the value of one or more columns for later use in another PROC SQL query or SAS statement

   `PUT` : Writes (or prints) the results to the SAS log.

```
PROC SQL NOPRINT;
  SELECT AVG(height)
  INTO :var1
  FROM table1 ; QUIT;
  %PUT &var1; >> Result: 178.82
```

(a) `TRIMMED` : Removes leading and trailing blanks from values that are stored in a single macro variable

(b) `NOTRIM` : Doesn't trim leading and trailing blanks from the values before creating the macro variables

(c) `SELECT col1, col2 INTO :var1, :var2` : Creating multiple macro variables

(d) `-` : Specifying range without an upper bound