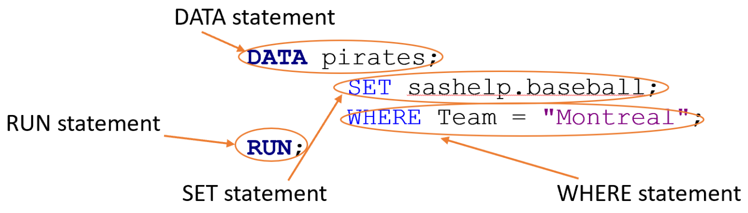
Contents

Statements generally start with a keyword and end with a semicolon.



```
DATA pirates;
    SET sashelp.baseball;
    WHERE Team = "Montreal";
RUN;
```

Example use of PROC SORT to reorder observations (rows) in a dataset.

Specify the dataset to sort
(Two-level name, references library and dataset)

Output dataset (don't overwrite)
One-level name used → output to work library

PROC SORT statement
to start PROC step

```
PROC SORT DATA = sashelp.stocks OUT = sortedStocks;
          BY High;
    RUN;
```

BY statement – specify the variable in
sashelp.stocks to sort on

End PROC step with RUN statement

Example use of PROC PRINT

PROC PRINT statement
to start PROC step

```
PROC PRINT DATA = sortedStocks;
RUN;
```

Specify the dataset to print
(One-level name, references dataset in work library)

End PROC step with RUN statement

Creating a library with a LIBNAME statement

Stand-alone LIBNAME statement
for creating a library (among other things)

```
LIBNAME NCSU '/home/u58009206/myLib';
```

Define library name

Path to folder –
u#### will differ
depending on user

Understanding code from PROC IMPORT

Stand-alone
statement to
reference a file
location

Internal name to reference

Path to file

```
FILENAME REFFILE '/home/u424592/my_shared_file_links/u424592/01-
ProgrammingInSASReadingData/forestfires.xlsx';

PROC IMPORT DATAFILE=REFFILE
            DBMS=XLSX
            OUT=WORK.IMPORT1;
            GETNAMES=YES;
RUN;
```

Reference file path (could use full path here)

PROC IMPORT statement
to start PROC step

Specify the 'database management system'
(here type of file)

Two-level name where imported data should be saved

Get variables names from the file?

End PROC step with RUN statement

Another PROC IMPORT example

Call PROC IMPORT statement to start the PROC step that will create a SAS data file (.sas7bdat)

Specify path to file on shared folder – first set of u#### will differ by user

```
PROC IMPORT DATAFILE='/home/u58009206/my_shared_file_links/u424592/01-
ProgrammingInSASReadingData/neuralgia.csv'
            DBMS=CSV
            OUT=NCSU.neuralgia;
            GETNAMES=YES;
RUN;
```

Let SAS know the file type is comma delimited.

Create a dataset called neuralgia in the NCSU library

Indicate the first row contains variable names

End PROC step with a RUN statement

Reading in data from a URL with FILENAME and PROC IMPORT

Reference name for use with PROC IMPORT

Specify path to file via URL

```
FILENAME fromWeb URL 'https://www4.stat.ncsu.edu/~online/datasets/neuralgia.csv';

PROC IMPORT DATAFILE=fromWeb
        DBMS=CSV
        OUT=work.neuralgia2;
        GETNAMES=YES;
RUN;
```

Use FILENAME reference

Indicating no column names in the raw data file with PROC IMPORT

```
FILENAME umpData URL 'https://www4.stat.ncsu.edu/~online/datasets/umps2012.txt';

PROC IMPORT DATAFILE=umpData
        DBMS=DLM
        OUT=NCSU.umps;
        DELIMITER='>';
        GETNAMES=NO;
RUN;
```

Specify there is no row with column names

DATA step syntax

DATA statement starts the DATA step

Specify the new data set you are creating with a one- or two-level name (can overwrite the dataset we copy)

SET statement copies an existing SAS dataset specified with a one- or two-level name (options available we'll discuss later)

```
DATA library.new_data_set;
       SET library.data_to_modify;
       <Many statements can go here>;
RUN;
```

Many useful statements can then be done to modify data, create new data, add labels, etc.

End DATA step with a RUN statement

Using the RENAME statement in a DATA step

Overwriting current data set by using same name in both places

```
DATA NCSU.umps;
     SET NCSU.umps;
     RENAME VAR1  =  Year
            VAR2  =  Month
            VAR3  =  Day
            VAR4  =  Home
            VAR5  =  Away
            VAR6  =  HPUmpire;
RUN;
```

Default variable names are VAR1, …, VAR 6. Rename each appropriately

RENAME statement allows for renaming of variable names in your dataset

Specifying a particular sheet when reading an excel file with PROC IMPORT

```
PROC IMPORT DATAFILE='/home/u58009206/my_shared_file_links/u424592/01-
ProgrammingInSASReadingData/censusEd.xlsx'
        DBMS=xlsx
        OUT=NCSU.census;
        GETNAMES=YES;
        SHEET="EDU01B";
RUN;
```

Instream data (data written in the program)

DATA statement starts the DATA step. Data being written to NCSU library as a dataset called heights

INPUT statement defines the variables to be read. The $ after a variable indicates it is a CHAR variable

```
DATA NCSU.heights;
      INPUT Person $ height;
      DATALINES;
      Justin 66
      Dave 68
      Jane 61
      ;
RUN;
```

Data to be read in. Each row is an observation and data values are separated by a space

DATALINES statement tells SAS that lines of data are about to appear below

RUN statement ends the DATA step

Semicolon indicates the raw data is finished

Reading excel data with a LIBNAME engine

LIBNAME statement creates a library

Path to excel file – again the first u#### will differ based on the user

```
LIBNAME census xlsx '/home/u58009206/my_shared_file_links/u424592/01-
ProgrammingInSASReadingData/censusEd.xlsx';
```

Name of new library

Specifies you are reading in an xlsx file

Contents

Using the RENAME statement in a DATA step

Overwriting current data set by using same name in both places

```
DATA NCSU.umps;
     SET NCSU.umps;
     RENAME VAR1 = Year
            VAR2 = Month
            VAR3 = Day
            VAR4 = Home
            VAR5 = Away
            VAR6 = HPUmpire;
RUN;
```

Default variable names are VAR1, ..., VAR 6. Rename each appropriately

RENAME statement allows for renaming of variable names in your dataset

Using the RENAME data set option in a DATA step

Create new data set that is a copy of the old one

Default variable names are VAR1, ..., VAR 6. Rename each appropriately

```
DATA NCSU.umps3;
       SET NCSU.umps (RENAME=(VAR1 = Year VAR2 = Month
                      VAR3 = Day VAR4 = Home
                      VAR5 = Away VAR6 = HPUmpire));
RUN;
```

RENAME **option** on the SET statement allows for renaming of variable names in your dataset

Using a LABEL statement in a DATA step

LABEL statement in a DATA step creates **permanent labels** to be associated with these three variables

```
DATA NCSU.umps2;
       SET NCSU.umps2;
       LABEL Home = 'Home Team for Game'
             Away = 'Away Team for Game'
             HPUmpire = 'Home Plate Umpire';
RUN;
```

Displaying LABELs in a PROC PRINT step

Adding **LABEL option** on PROC PRINT statement prints the dataset with labels!

```
PROC PRINT DATA = NCSU.umps2 LABEL;
RUN;
```

Using a DROP statement in a DATA step (KEEP is similar)

List variables to remove in the DROP statement

```
DATA NCSU.fandangoDrop;
       SET NCSU.fandango;
       DROP Votes Rating;
RUN;
```

Using a DROP data set option in a DATA step (KEEP is similar)

List variables to remove in the DROP option

```
DATA NCSU.fandangoDrop;
      SET NCSU.fandango (DROP = Votes Rating);
RUN;
```

Creating new variables in a DATA step

```
DATA NCSU.fandangoNew;
      SET NCSU.fandango;
      avg = MEAN(Rating,Stars);
      firstWord = SCAN(film, 1);
      secondWord = SCAN(film, 2);
RUN;
```

New variable names

Function of other variables in dataset (return 1st and 2nd word of title, respectively)

IF THEN ELSE syntax

```
IF condition THEN action;

IF condition THEN action;
ELSE action;

IF condition THEN action;
ELSE IF condition THEN action;
ELSE action;
```

IF THEN ELSE example

```
IF (stars > 4.4) AND (rating > 4.4) THEN Status = "watch";
ELSE IF (stars > 4.4 AND rating LE 4.4) OR (stars LE 4.4 AND rating > 4.4) THEN
Status = "maybe";
ELSE Status = "no";
```

PROC STDIZE syntax

```
PROC STDIZE DATA = OUT =  METHOD =  OPREFIX SPREFIX=;
       BY variable(s);
       VAR variable;
RUN;
```

Option used to keep the original variables and prefix the newly created variables

Start the PROC STDIZE step

- Specify the dataset with DATA =
- Optional output dataset with OUT =
- Type of standardization with METHOD =
  - Usually MEAN or STD

End the step with a RUN statement.

VAR specifies the variables that should be standardized.
If not used, all variables are standardized.

Optional use of BY (data should be sorted).
Produces standardization for each setting of the BY variable.

PROC SORT syntax

```
PROC SORT DATA = libref.dataset OUT = libref.out_data;
          BY VAR1 DESCENDING VAR2 ...;
RUN;
```

Example of subsetting data with a WHERE statement

```
PROC PRINT DATA = sashelp.baseball;
          WHERE (Team = "Cleveland") AND (CrAtBat < 1000);
RUN;
```

Could use & instead
Note: Parentheses not needed

Use of IN in a WHERE statement

```
DATA subset;
       SET sashelp.baseball;
       WHERE Team IN ("Cleveland", "Atlanta", "Boston");
RUN;
```

Syntax is to provide a list of
values to compare with
(parentheses need)

Example using IF to include on rows that meet a condition

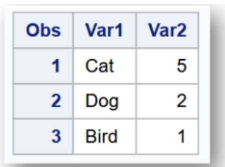
```
DATA newbaseball;
       SET sashelp.baseball;
       IF (Team = "Cleveland") OR (Team = "Atlanta");
RUN;
```

Example using IF THEN DELETE to remove rows

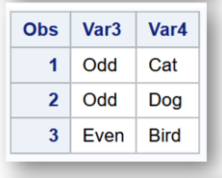
```
DATA newbaseball;
       SET sashelp.baseball;
       IF (Team = "Cleveland") THEN DELETE;
RUN;
```

Doing a one-to-one MERGE in a DATA step

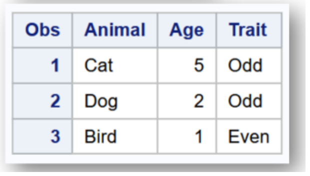
```
DATA NCSU.first;
        INPUT Var1 $ Var2;
        DATALINES;
        Cat 5
        Dog 2
        Bird 1
        ;
RUN;

DATA NCSU.second;
        INPUT Var3 $ Var4 $;
        DATALINES;
        Odd Cat
        Odd Dog
        Even Bird
        ;
RUN;

DATA NCSU.merged;
        MERGE NCSU.first  (RENAME=(Var1=Animal Var2=Age))
              NCSU.second  (RENAME=(Var3=Trait) DROP = Var4);
RUN;
```



Obs	Var1	Var2
1	Cat	5
2	Dog	2
3	Bird	1



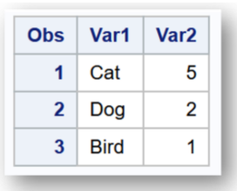
Obs	Var3	Var4
1	Odd	Cat
2	Odd	Dog
3	Even	Bird



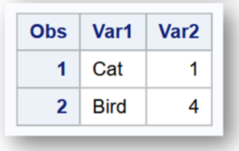
Obs	Animal	Age	Trait
1	Cat	5	Odd
2	Dog	2	Odd
3	Bird	1	Even

Concatenating two data sets

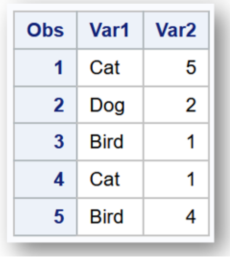
```
DATA NCSU.top;
        SET NCSU.first;
RUN;

DATA NCSU.bottom;
        INPUT Var1 $ Var2;
        DATALINES;
        Cat 1
        Bird 4
        ;
RUN;

DATA NCSU.concat;
        SET NCSU.top NCSU.bottom;
RUN;
```



Obs	Var1	Var2
1	Cat	5
2	Dog	2
3	Bird	1



Obs	Var1	Var2
1	Cat	1
2	Bird	4



Obs	Var1	Var2
1	Cat	5
2	Dog	2
3	Bird	1
4	Cat	1
5	Bird	4

Contents

Basic PROC FREQ syntax

Start the PROC FREQ statement, specify the dataset with DATA =

Optional use of BY (data should be sorted)

Recall WHERE statements can be used in most PROCs to subset the data

```
PROC FREQ DATA =  ;
    WHERE variables ;
    BY variables ;
    TABLES requests </ options> ;
RUN;
```

Common option to output the created table to a dataset

- TABLES var1 var2/OUT = lib.data;

TABLE or TABLES statement asks for contingency tables:

- TABLES var1 var2
- TABLES var1*var2 var3

Basic PROC UNIVARIATE syntax & example

```
PROC UNIVARIATE <options>;
    BY variables;
    CLASS variable-1 <(v-options)> <variable-2 <(v-options)>;
    HISTOGRAM <variables> </ options>;
    VAR variables;
RUN;
```

```
PROC UNIVARIATE DATA = CO2data;
      VAR uptake;
RUN;
```

Basic PROC MEANS syntax & example

```
PROC MEANS <options><statistic-keyword(s)>;
    BY <DESCENDING> variable-1 ...;
    CLASS variable(s) </ options>;
    VAR variable(s) </ WEIGHT=weight-variable>;
RUN;
```

```
PROC MEANS DATA = CO2data MEAN VAR STDDEV MIN Q1 MEDIAN Q3 MAX MAXDEC = 2;
        VAR uptake;
RUN;
```

Basic PROC CORR syntax & example

```
PROC CORR <options>;
   BY variables;
   VAR variables;
RUN;
```

```
PROC CORR DATA = NCSU.titanic COV;
        VAR age fare pclass survived;
RUN;
```

Basic PROC SGPLOT syntax & examples

```
PROC SGPLOT <options>;
   DENSITY response-variable </options>;
   DOT category-variable </options> ;
   HBAR category-variable </options>;
   HBOX analysis-variable </options>;
   HISTOGRAM response-variable </options>;
   .... (so many!)
RUN;
```

```
PROC SGPLOT DATA = NCSU.titanic;
       VBAR survived/GROUP = sex
                  GROUPDISPLAY = cluster;
RUN;
```

```
PROC SGPLOT DATA = CO2data;
       VBOX uptake/CATEGORY = Treatment;
       SCATTER X = Treatment
               Y = uptake/JITTER;
RUN;
```

Basic PROC SGPLANEL syntax & example

```
PROC SGPANEL <options>;
   PANELBY variable(s) </options>;
   Most all of the same plots via the same statements!
RUN;
```

```
PROC SGPANEL DATA = NCSU.titanic;
       PANELBY embarked;
       VBAR survived/GROUP = sex
                  GROUPDISPLAY = cluster;
RUN;
```

```
PROC SGPANEL DATA = CO2data;
       PANELBY Type;
       VBOX uptake/CATEGORY = Treatment;
       SCATTER X = Treatment
               Y = uptake/JITTER;
RUN;
```

Contents

Basic PROC CORR syntax

```
PROC CORR <options>;
     BY variables;
     VAR variables;
RUN;
```

PROC CORR statement begins the PROC step

- DATA = option
- PLOTS=MATRIX(HIST)
- SPEARMAN
- FISHER

Can find correlations for each level of a categorical variable

Variables to find the correlations between

Basic PROC GLM syntax for fitting a regression model

```
PROC GLM <options>;
     ~~CLASS variable;~~
     MODEL y = x </ options>;
     BY variables;
RUN;
QUIT;
```

PROC GLM statement begins the PROC step

- DATA = option
- ALPHA = option
- PLOTS = all

MODEL statement specifies the response

- Use response = numeric predictor
- Options
  - CLPARM
  - CLM
  - CLI

Can fit your model for each level of a categorical variable

Include a QUIT statement so SAS knows you are done modeling