STK 210: Practical 1

1 The SAS System

SAS (Statistical Analysis System) is an integrated system of software solutions that enables you to perform statistical and mathematical analysis. There are two components to SAS programs namely:

1. the DATA step and
2. the PROC step.

The data step reads data into a SAS dataset, the PROC step analyses the SAS dataset. DATA steps begin with the word DATA and PROC steps begin with the word PROC. The `run;` command signals to SAS that the previous commands can be executed.

The SAS programs used in this notes are available on ClickUP in the **Practical work folder - Practical 1**.

1.1 The DATA step

SAS organises data into a rectangular form or table that is called a SAS data set. The following table shows the **raw data**. The data describes participants in a 16-week weight program at a health and fitness club. The data for each participant includes:

- an identification number (*numerical*),
- name (*character* variable),
- team name (*character* variables), and
- weight (in U.S. pounds) the beginning and end of the program (*numerical* variables).

IdNumber	Name	Team	StartWeight	EndWeight
1023	David	red	189	165
1049	Amelia	yellow	145	124
1219	Alan	red	210	192
1246	Ravi	yellow	194	177
1078	Ashley	red	127	118

In a SAS data set, each row represents information about an individual entity and is called an **observation**. Each column represents the same type of information and is called a **variable**. Each separate piece of information is a **data value**.

The following SAS program creates a SAS data set named WEIGHT_CLUB from the health club data:

SAS Program:

```
data weight_club;
input IdNumber Name $ Team $ StartWeight EndWeight;
Loss=StartWeight-EndWeight;
datalines;
1023 David  red    189 165
1049 Amelia yellow 145 124
1219 Alan   red    210 192
1246 Ravi   yellow 194 177
1078 Ashley red    127 118
;
```

- The DATA statement tells SAS to begin building a SAS data set named WEIGHT_CLUB.
- The INPUT statement identifies the fields to be read from the input data and names the SAS variables to be created from them. Character variables are identified by using a $ sign.
- The third statement is an assignment statement. It calculates the weight each person lost and assigns the result to a new variable, LOSS.
- The data lines follow the DATALINES statement. This approach to processing raw data is useful when you have only a few lines of data.
- The semicolon signals the end of the raw data, and is a step boundary. It tells SAS that the preceding statements are ready for execution.

Note:

- In this example we use a so-called *list input* where each field in the raw data is separated by at least one space.
- When you use a *one-level* name for the data set like we have done here the SAS data set WEIGHT_CLUB is *temporary*; i.e. it exists only for the duration of your current session.
- When we use `libname.sasdata` a permanent data file is stored, but will not be covered in this course.
- SAS places this data set in a SAS data library referred to as WORK. All files that SAS stores in the WORK library are temporary and deleted at the end of a session.

1.1.1 Some Concepts and Rules

- SAS variable names must be 32 characters or less, constructed of letters, digits and the underscore character.
- SAS is not case sensitive, except inside of quoted strings.
- Missing values are handled consistently in SAS, and are represented by a period `.` for numerical variables and a blank for character variables.
- Each statement in SAS must end in a semicolon `;`.

1.2 The PROC step

SAS can simplify programming for you with its library of built-in programs known as SAS procedures. SAS procedures use data values from SAS data sets to produce preprogrammed reports, requiring minimal effort from you.

1.2.1 PROC PRINT

This procedure, known as the PRINT procedure, displays the variables in a simple, organized form. The data in the SAS data set `WEIGHT_CLUB` is being printed here.

SAS Program:

```
proc print data=weight_club;
title 'Health Club Data';
run;
```

Since no VAR statement is used here, all the variables are printed. The following output shows the results:

SAS Output:

```
Health Club Data
```

Obs	Id Number	Name	Team	Start Weight	End Weight	Loss
1	1023	David	red	189	165	24
2	1049	Amelia	yellow	145	124	21
3	1219	Alan	red	210	192	18
4	1246	Ravi	yellow	194	177	17
5	1078	Ashley	red	127	118	9

1.2.2 In General

The syntax of the PROC step is as follows:

```
PROC procedure <DATA=SAS-data-set>;
  < Statements of procedure that is invoked. Each followed by semi-colon ; >
run;
```

- A PROC step tells SAS to invoke a particular SAS procedure to process the SAS data set that is specified in the DATA= option.
- If you omit the DATA= option, then the procedure processes the most recently created SAS data set in the program.
- The RUN statement signals the end of the PROC step and can be ignored if this is followed by another DATA or PROC step.

1.3 The SAS Windowing Environment

We are going to use the SAS windowing environment. When you invoke SAS, the following windows appear.

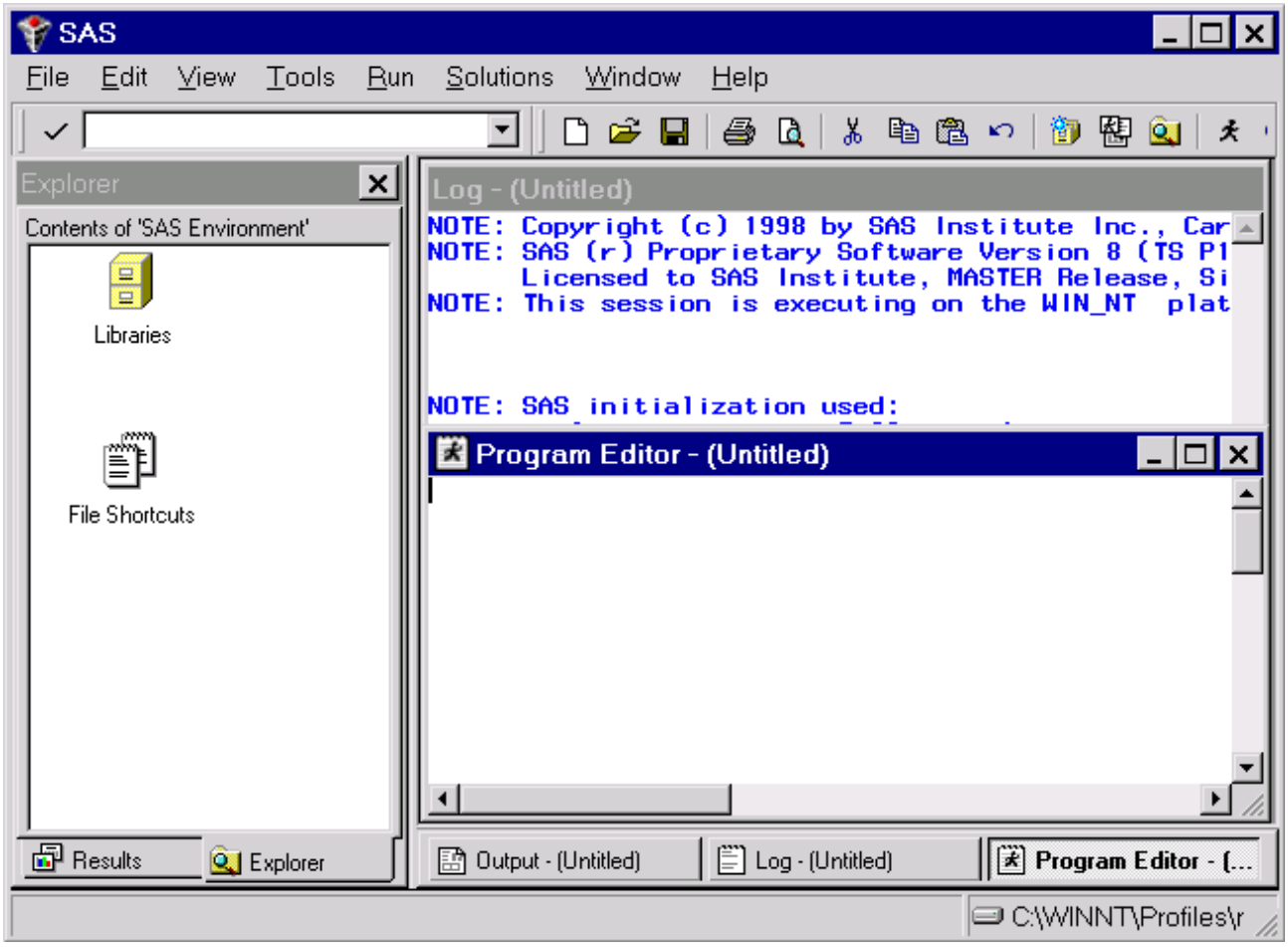

Figure 1.1: SAS Windowing Environment

The window at the top right is the Log window which contains the SAS log for the session. The window at the bottom right is the Program Editor window. This window provides an editor in which you edit your SAS programs. When you fill the Program Editor window, scroll down to continue typing the program.

When you finish editing the program, submit it to SAS by making use of the  run button in the Application Toolbar and view the output. (If SAS does not create output, check the SAS log for error messages.) The following displays show the first and second pages of the Output window.

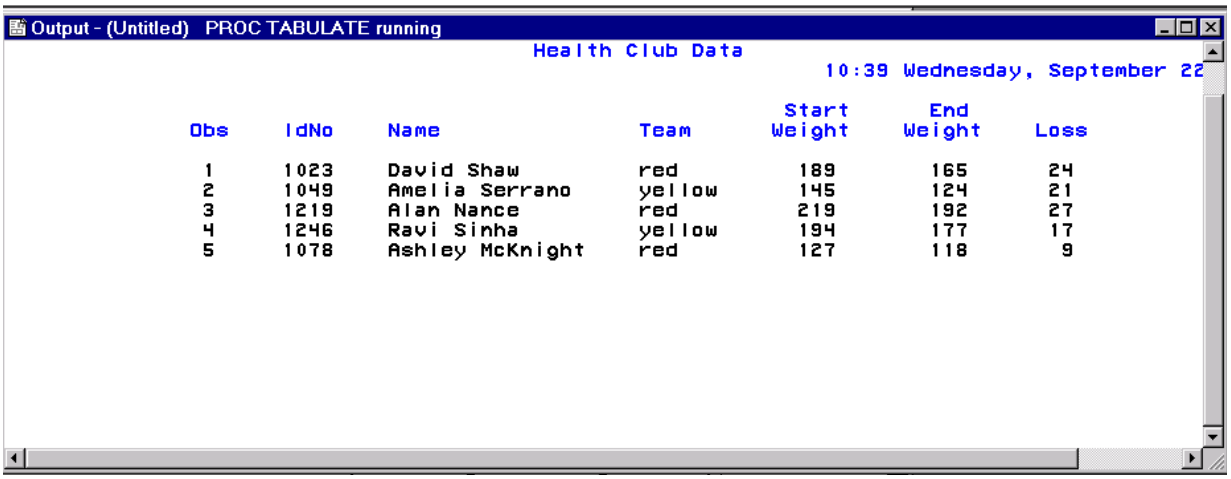

Figure 1.2: The Output in the Output Window

After you finish viewing the output, you can return to the Program Editor window to begin creating a new program. By default, the output from all submissions remains in the Output window, and all statements that you submit remain in memory until the end of your session. You can view the output at any time, and you can recall previously submitted statements for editing and resubmitting. You can also clear a window of its contents.

In the window at the left of Figure 1.1 you can find SAS explorer where all the SAS data sets are stored. You can also make use of the  SAS explorer button in the Application Window.

1.4 Creating new variables in a SAS data set

To create a SAS data set, you can read data from different locations i.e.

- raw data, following a DATALINES statement and
- SAS data set, may be *temporary* or *permanent* in the SAS library

We are now going to use a SAS data set to create new variables by making use of the SET statement.

1.4.1 Example 1

Say we want to convert the weights in the SAS data set WEIGHT_CLUB from pounds to kilogram.

Note: 1 pound = 0.453592 kg

SAS Program:

```
data weight_kg; set weight_club;
  weight_s=startweight*0.453592;
  weight_e=endweight*0.453592;
  weight_l=loss*0.453592;
run;

proc print data=weight_kg;
  var  idnumber name weight_s weight_e weight_l;
  title 'Health Club Data';
run;
```

SAS Output:

Health Club Data

Obs	Id Number	Name	weight_s	weight_e	weight_l
1	1023	David	85.7289	74.8427	10.8862
2	1049	Amelia	65.7708	56.2454	9.5254
3	1219	Alan	95.2543	87.0897	8.1647
4	1246	Ravi	87.9968	80.2858	7.7111
5	1078	Ashley	57.6062	53.5239	4.0823

- In this example we used the SAS data set WEIGHT_CLUB to create a new SAS data set WEIGHT_KG, where we have added three new variables namely

  WEIGHT_S, WEIGHT_E and WEIGHT_L

  These three variables are now all in kilograms (kg).

- You can view the SAS data set WEIGHT_KG in SAS Explorer in the WORK library where all the temporary SAS data sets are stored.

1.4.2 Example 2

In the second example we would like to classify all the members in the data set who lost at least 10% of their starting weight as being "successful".

SAS Program:

```
data weight; set weight_kg;
  weight_p=weight_l/weight_s*100;
  if weight_p<10 then  success=0;
  if weight_p>=10 then success=1;
  if weight_p<10 then  pass='N';
  if weight_p>=10 then pass='Y';
  weight_s=round(weight_s,0.1);
  weight_e=round(weight_e,0.1);
  weight_l=round(weight_l,0.01);
  weight_p=round(weight_p,0.001);
run;

proc print data=weight;
  var  idnumber name weight_s weight_e weight_l weight_p success pass;
  title 'Health Club Data';
run;
```

SAS Output:

```
Health Club Data

      Id
Obs Number  Name  weight_s weight_e weight_l weight_p success pass
 1   1023  David    85.7     74.8     10.89   12.698     1     Y
 2   1049  Amelia   65.8     56.2      9.53   14.483     1     Y
 3   1219  Alan     95.3     87.1      8.16    8.571     0     N
 4   1246  Ravi     88.0     80.3      7.71    8.763     0     N
 5   1078  Ashley   57.6     53.5      4.08    7.087     0     N
```

- The IF statement is used to create:
  - the numerical variable SUCCESS and
  - the character variable PASS

  to indicate whether the percentage weight loss is at least 10%.
- The ROUND function is used to round the values to the appropriate decimal.

Note:

- You can once again view your SAS data WEIGHT in the WORK library in SAS Explorer.
- You can also discard the variables STARTWEIGHT, ENDWEIGHT and LOSS by making use of the drop statement in the DATA step:

```
drop startweight endweight loss;
```

1.5 Operators in SAS

1.5.1 Arithmetic Operators

One way to perform calculations on numeric variables is to write an assignment statement using arithmetic operators. Arithmetic operators indicate addition, subtraction, multiplication, division, and exponentiation (raising to a power). The following table shows operators that you can use in arithmetic expressions.

Table 1.1 Operators in Arithmetic Expressions

Symbol	Operation	Example
+	addition	x = y + z;
-	subtraction	x = y - z;
*	multiplication	x = y * z;
/	division	x = y / z;
**	exponentiation	x = y ** z;

1.5.2 Logical Operators

Often in a program you need to know if variables are equal to each other, or if they are greater than or less than each other. To compare two numeric variables, you can write an IF-THEN/ELSE statement using logical operators. The following table lists some of the logical operators you can use for variable comparisons.

Table 1.2 Logical Operators

Symbol	Logical Operation
=	equal
^=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to

2 Exercise

The data for this exercise is placed on ClickUP and is also available on the share drive in the Informatorium

`S:\STK210\PRAC1\FITNESS2.SAS`

The data contains each participant's identification number, name, team name, and weight (in U.S. pounds) at the beginning and end of a 16-week weight program.

1. Create the SAS data set `CLUB` with the following variables:

   - ID
   - NAME
   - TEAM
   - WEIGHT1
   - WEIGHT2

   Remember:

   - Use the `$` sign to specify the character variables.
   - You can view the SAS data set `CLUB` by
     - viewing the data set in SAS Explorer, in the WORK library or
     - by making use of PROC PRINT

2. Create the SAS data set `CLUB1` with the SET statement and do the following:

   - Convert the variables WEIGHT1 and WEIGHT2 which is measured in pounds to kilogram:

     $$\boxed{1 \text{ pound} = 0.453592kg}$$

     E.g. you can use the statement

     ```
     WEIGHT1=WEIGHT1*0.453592;
     ```

     in the DATA step.

   - Round the weight at the beginning and end of a 16-week weight program (that is now converted to kilograms) to the nearest first decimal by making use of the ROUND function.
   - Create the variable weight LOSS by making use of the statement

     ```
     LOSS=WEIGHT1-WEIGHT2;
     ```

     in the DATA step.

   - You can view your data again by making use of PROC PRINT or by viewing the SAS data set `CLUB1` in the WORK library of SAS Explorer.

3. Use PROC UNIVARIATE to obtain descriptive statistics for the variable LOSS.

   Example:

   ```
   proc univariate data=club1;
     var loss;
   run;
   ```

   (a) Give the average weight loss.

   (b) Give the standard deviation and interpret the value.

   (c) Give the minimum and maximum weight loss of the participant.

   (d) Give the ninetieth percentile and interpret this value.

   (e) Give the value of the interquartile range and interpret this value.

4. Use PROC MEANS to obtain descriptive statistics for the variables WEIGHT1, WEIGHT2 and LOSS.

   Example:

   ```
   proc means data=club1;
     var weight1 weight2 loss;
   run;
   ```

   (a) Give the mean values for the variables WEIGHT1, WEGHT2 and LOSS;

5. Use PROC FREQ to obtain descriptive statistics for the variable TEAM.

   Example:

   ```
   proc freq data=club1;
     tables team;
   run;
   ```

   (a) Give the number of participants in the red team.

   (b) Give the percentage of participants in the green team.

6. Use PROC MEANS with a CLASS statement to obtain the mean weight loss for each the four teams.

   Example:

   ```
   proc means data=club1;
   class team;
   var loss;
   run;
   ```

(a) Identify the team with the most weight loss on average. Name the team with their average weight loss.

(b) Identify the team with the least weight loss on average. Name the team with their average weight loss.

7. Create the SAS data set `CLUB2` from the SAS data set `CLUB1` with the variable LOSS_GRP, where the weight loss is classified into 5 groups namely:

LOSS	LOSS_GRP
≤ 5	1
$(5-6]$	2
$(6-7]$	3
$(7-8]$	4
>8	5

NB: SAS will classify missing values in the first category if we use the statement

```
if loss<=5 then lossgrp=1;
```

It is therefore necessary to delete observations with missing values by making use a DELETE statement. See Example below.

Example:

```
data club2; set club1;
  if loss=.  then delete;
  if loss<=5   then lossgrp=1;
  if 5<loss<=6 then lossgrp=2;
  ⋮ etc.
run;
```

Note: You can view the data with PROC PRINT or with SAS Explorer.

(a) Use PROC FREQ to create a frequency distribution for the weight loss, with the categories indicated in the table. Give the cumulative percentage distribution as well.

(b) Use PROC MEANS with a CLASS statement to obtain descriptive statistics for each of the 5 categories of the variable LOSS.

i. Give the mean weight loss for the participants who lost more than 8kg in the program.

ii. Identify the category of LOSS_GRP with the largest variation. Explain your answer.