SAS

23 settembre 2024

Indice

1 Introduction **5**
1.1 First concepts/definitions . . . 5
1.1.1 Data IO . . . 6
1.1.2 Data Import . . . 6
1.1.2.1 Text files . . . 6
1.1.2.2 Excel files . . . 7
1.2 Dataset structure . . . 8
1.3 Data reporting . . . 9
1.3.1 `proc print` . . . 9
1.3.2 `proc sort` . . . 10
1.3.3 Grouped reporting (sort + print by) . . . 11
1.3.4 Report enhancing: title, footnotes . . . 11
1.3.5 Reporting exercises . . . 12

2 Data management **15**
2.1 Data manipulation . . . 15
2.1.1 Selecting variables: `drop`, `keep` . . . 15
2.1.2 Constructing expression: operators . . . 15
2.1.3 Selecting observation: `IF`, `where` . . . 16
2.1.4 Creating variables . . . 17
2.1.5 Using SAS functions . . . 18
2.1.6 `by` with `set`, `.first` and `.last` . . . 21
2.2 Informat and format . . . 24
2.2.1 Informat . . . 24
2.2.2 Format . . . 26
2.2.3 Converting numerics and chars . . . 27
2.2.3.1 From character to numeric: `input` . . . 27
2.2.3.2 From numeric to character: `put` . . . 28
2.2.4 User defined format . . . 29
2.3 Transpose . . . 34
2.4 Concatenate . . . 37
2.5 Merge . . . 38
2.5.1 Merging with missing id variables . . . 38

3 Programming **43**
3.1 Arrays . . . 43
3.2 Macro . . . 45
3.2.1 Macro variables . . . 45

3.2.2 Macro programs 46

4 Reporting 51

4.1 proc means 51
4.2 proc freq 52
4.3 ods language 57
4.3.1 Output to sas dataset 57
4.3.2 Output to RTF or PDF 58
4.4 proc report for reports 59

Capitolo 1

Introduction

1.1 First concepts/definitions

Definition 1.1.1 (Language and intepreter features)**.** Some truth:

- sas language is case *insensitive*,
- comment as classic C-style

```
/* This is a comment */
```

- There are three main windows
  - *Editor*: where we program. `F3` is used to run code (highlight chunks if selective run is needed)
  - *Log*: to check for warning/error in execution
  - *Output*: to look at risultati.

Definition 1.1.2 (SAS program, steps and instructions)**.** Furthermore:

- a SAS program is a sequence of *steps*.
- there are two kinds of steps: `data` steps (creates SAS data sets, import sas file) and `proc` steps (import data from foreign format, do analysis, generate reports/visualization)
- a step is a set of *instruction*/*statement*, all ended by `;`
- an instruction can be *splitted* across lines if too long; viceversa different instructions can be written in the *same line*, but they have to be separated by `;`
- steps begin with either a `data` *statement* (for `data` steps) or a `proc` statement (for `proc` steps).
- steps ends either when another step begin, or when `run` or `quit` instruction are found.

1.1.1 Data IO

Definition 1.1.3 (Libraries)**.** A collection of SAS dataset that are referenced/stored as a unit. Can be thought as directory somewhere on the filesystem.

Important remark 1 (Default libraries). Two of them are:

- `work`: default temporary library used to store/access SAS datasets for the duration of the interactive session (`work` is mapped under the OS temporary directory).
- `sashelp`: system library with sample data (permanent)

Definition 1.1.4 (libref)**.** Name used to refer to a SAS library (eg `work`, `sashelp`)

Important remark 2 (dataset naming/referencing). All data sets have a two level name composed as `libref.datasetname`.
Dataset archived under `work` can be accessed directly for convenience: eg `testdata` instead of `work.testdata`

Definition 1.1.5 (Libraries/libnames definition)**.** To save data permanently on disk we need to define a library/libname before. `libname` statement specify the already existing directory where to save data.

```
libname istat "/home/l/dati/istat";
```

This association is effective until changed/cancelled or until the session ends.

Important remark 3. Once the libref is active we can use it to read/write from/to the directory, specifying dataset name after libref in a data step

Example 1.1.1 (Saving a file)**.** This will write `rcfl.sas7bdat` file under `/home/l/dati/istat` (libname defined before) using the instructions specified afterward in the data step (not reported here)

```
data istat.rcfl;
...
```

Example 1.1.2 (Working in memory)**.** This will read `data.example`, add `heightCM` computed variable and save it under `work.a` temporary dataset:

```
data a; /* set work.a as output dataset*/
  set data.example;  /* use data.example as input dataset*/
  heightCM = heightM * 100; /* add this variable*/
run; /* ends the data step */
```

1.1.2 Data Import

1.1.2.1 Text files

Important remark 4. Main statement to import data text files:

- `infile`: specifies the path of the file;

- `input`: specifies file structure aka variable names, the number of variables and their format (character or numeric). For character variables the name is followed by `$`

The syntax used varies according to kind of file to be imported.

Definition 1.1.6 (Free format data)**.** Variables/columns are separated by delimiting symbols (as blank space or a semicolon).

Example 1.1.3 (Free format import)**.** If separed by space like this:

```
Luigi Rossi M 160 55
Mario Bianchi M 180 75
Alice Verdi F 160 50
Sara Rossi F 168 60
```

this is how to read it:

```
data a;
  infile "free_format.txt"; /* path of the file*/
  input nome $ cognome $ sesso $ altezza peso; /* variables spec*/
run;
```

Definition 1.1.7 (Fixed column data)**.** Variables have a fixed structure and they can be distinguished by their position in the row.

Example 1.1.4 (Fixed column import)**.** Same data with a fixed structure:

```
Luigi   Rossi   M 160 55
Mario   Bianchi M 180 75
Alice   Verdi   F 160 50
Sara    Rossi   F 168 60
```

are to be imported like this:

```
data b;
  infile "column_format.txt"; /* path */
  /* below variables spec with position (columns)*/
  input nome $ 1-8 cognome $ 9-16 sesso $17 altezza 19-21 peso 23-24;
run;
```

1.1.2.2 Excel files

Important remark 5. With xls files the `proc import` is used.

Example 1.1.5 (Excel file import)**.** To import an excel file:

```
/*
The first statement specify output dataset (exceldata), infile path and
type of file to be imported; for .xlsx file change to "xlsx".
replace is used to overwrite an existing dataset.
*/
proc import out=exceldata
            datafile="/home/l/excel_file.xls"
            dbms=xls
            replace; /* <- first statement ends here*/
  sheet=foglio1;    /* excel sheet selection*/
  GETNAMES=yes;     /* get variables names from the first row */
run;
```

1.2 Dataset structure

Important remark 6. A SAS data set contains a *descriptor portion* (at beginning) and a *data portion* (afterwards).

Definition 1.2.1 (Descriptor portion)**.** Contains this metadata:

- general properties (eg dataset, number of observations)
- variable properties (such as name, type, and length)

Definition 1.2.2 (Data portion)**.** Data portion of a SAS dataset contains the data values, characters or numerics.

Important remark 7. We use `proc contents` for descriptor portion and `proc print` for data portion.

Definition 1.2.3 (`proc contents`)**.** Visualizes the content and the tructure of a dataset (eg name of the datset, path to the file, number of observation, number of variables, variable name/type/format/label, other dataset metadata).

Example 1.2.1. The syntax is like:

```
proc contents data=work.testdb;
run;
```

Important remark 8 (Dataset and variables names). Some rules:

- 1 to 32 characters long; composed of letters, underscores (_), or numbers;
- must *start* with a letter or underscore.
- not case sensitive (eg `Asd` and `asd` are the same)

Definition 1.2.4 (Variable label)**.** It is:

- a text string up to 256 (special) characters that can be assigned to the variable in order to clarify its content
- set in a data step with the label statement;

```
LABEL variable1="description of variable1";
```

- visualized in column heading of an open dataset. To visualize the real name, double click on the column heading;
- during programming the user must still refer to the variable name.

Definition 1.2.5 (Variable types)**.** There are two types:

- **character variables**
  - contain any value (letters, numerals, special characters, and blanks)
  - up to 32.767 characters in length
  - 1 byte per character.
- **numeric variables**:

  - Store numeric values using floating point or binary representation
  - can store 16 or 17 significant digits.
  - 8 bytes of storage by default

Remark 1. A date is memorized as integer, number of days since 1-1-1960.

Definition 1.2.6 (Missing values)**.** Are represented by:

- a blank `" "` (space between virgolette) for character values
- a period (`.`) for numeric values

1.3 Data reporting

1.3.1 `proc print`

Definition 1.3.1 (`proc print`)**.** Used to display the data portion of a data set.

```
proc print data=datasets;
run;
```

Remark 2. Some useful options follow

Definition 1.3.2 (`var` statement)**.** Select which variables to print

```
proc print data=fev;
  var id baseline week_6;
run;
```

Definition 1.3.3 (`where` statement)**.** select rows according to criteria

```
proc print data=orion.sales;
  var last_name first_name salary;
  where salary < 25500;
run;
```

Definition 1.3.4 (`sum` statement)**.** add report totals for requested numeric variables

```
proc print data=orion.sales;
  var last_name first_name salary;
  sum salary;
run;
```

Definition 1.3.5 (`noobs` option)**.** Used to suppress the `Obs` column (id/progressive observation number)

```
proc print data = orion.sales noobs;
  var last_name first_name salary;
  where salary < 25500;
run;
```

Definition 1.3.6 (`ID` statement)**.** specifies the variable/s to print at the beginning of each row as id (instead of an observation number).

```
proc print data = orion.customer_dim;
  where customer_age = 21;
  id customer_id;
  var customer_name customer_gender
      customer_country customer_group
      customer_age_group customer_type;
run;
```

Definition 1.3.7 (`label` option)**.** print labels instead of varnames

```
proc print data=fev label;
run;
```

1.3.2 `proc sort`

Definition 1.3.8 (`proc sort`)**.** Used to sort a dataset, syntax is

```
proc SORT data=input-SAS-data-set <OUT=output-SAS-data-set>;
  by <DESCENDING> variable<s>;
RUN;
```

where

- `by` statement specifies the sort variables and, optionally, the sort order (ascending by default)
- if `out` option is used (optional) a new dataset is created, otherwise the dataset will be overwritten.
- does not generate printed output

Important remark 9 (multiple sorting variables)*.* Some `by` statement examples

```
by x y; /* ascending for x, and for y within x */
by descending x y;  /* descending for x, and ascending for y within x*/
by x descending y; /* ascending for x, and descending for y within x*/
by descending x descending y; /* descending for x, and for y within x */
```

Example 1.3.1. Sort by ascending salary in a new dataset

```
proc sort data=orion.sales out=work.sales;
  by salary;
run;
```

Example 1.3.2. Sort sales employees grouped by Country, in descending Salary order within country

```
proc sort data=orion.sales out=work.sales;
  by country descending salary;
run;
```

1.3.3 Grouped reporting (sort + print by)

Definition 1.3.9 (`proc print by`: grouped reporting)**.** `by` statement in a `proc print` specifies the variable/s used to form groups/splits in the report. However observations in the data set *must* be in order by the `by` variable/s

Example 1.3.3. display selected variables, the salary subtotal for each country, and the salary grand total

```
proc sort data=orion.sales out=work.sales;
  by country descending salary;
run;

proc print data=work.sales noobs;
  by country;
  sum Salary;
  var first_name last_name gender salary;
run;
```

1.3.4 Report enhancing: title, footnotes

Important remark 10. To enhance reporting we can set a main title for the report and use footnotes (aside of variable labels, already presented).

Definition 1.3.10 (`title` statement)**.** The global TITLE statement specifies title lines for SAS output at the top of the page (default title is `The SAS System`):

- The syntax is

  ```
  TITLEn 'text';
  ```

  with n from 1 to 10 (an unnumbered title is equivalent to `title1`)
- titles remain in effect until they are changed/canceled, or SAS session ends
- The null title statement cancels all title

  ```
  TITLE;
  ```

Definition 1.3.11 (`footnote` statement)**.** The syntax is

```
FOOTNOTEn 'text';
```

and work the same way as title but appear at the bottom of the page

Important remark 11 (Changing titles/footnotes). To change a title/footnote, submit a TITLE/FOOTNOTE statement with the same number but different text. That:

- replaces a previous title with the same number
- cancels all titles with higher numbers

Example 1.3.4. Eg

```
/* If after*/
title1 'ABC Company';
title2 'Sales Division';
title3 'Salary Report';
/* we say */
title1 'Salary Report';
```

The latter statement changes title 1 and cancels titles 2 and 3

Example 1.3.5. enhanced reporting incoming

```
title1 'Orion Star Sales Staff';
title2 'Salary Report';
footnote1 'Confidential';
proc print data=orion.sales;
  var employee_id last_name salary;
  run;
title;
footnote;
```

1.3.5 Reporting exercises

Example 1.3.6 (Esercizi 1/2 lezione 2)**.** Si ha:

```
/* create a libref called data where dataset are stored*/
libname data "/home/l/data";

/* Run a proc contents */
proc contents data=data.clinic_stats;
run;

/* print the clinic_stats dataset */
proc print data=data.clinic_stats;
run;

/*print the clinic_stats dataset displaying labels*/
proc print data=data.clinic_stats label;
run;

/*keep only the variables: clinic, month num_surg*/
proc print data=data.clinic_stats;
  var clinic month num_surg;
run;

/* Compute the sum of surgeries */
proc print data=data.clinic_stats;
  var clinic  month num_surg;
  sum num_surg;
run;

/* Extra: Compute the sum of surgeries in 2009 */
```

```
proc print data=data.clinic_stats;
  where year(monyy)=2009;
  var clinic month num_surg;
  sum num_surg;
run;

/* Print by clinic groups and descending surgeries*/
proc sort data=data.clinic_stats out=t1;
  by clinic descending num_surg;
run;
proc print data=t1;
  by clinic;
  sum num_surg;
run;

/* Same but add titles and footnotes */
title1 "Hospitals CENTRAL and EAST";
title2 "Number of surgeries";
footnote1 "SAS lab 2022";
proc print data=t1;
  by clinic;
  sum num_surg;
run;
```

Example 1.3.7 (Esercizio 3 lezione 2)**.** Si ha:

```
/* POINT 1*/
libname data "/home/l/data";

/* Run a proc print to make a report of the dataset, displaying
labels for the columns. Add the title «Costumer details» to the
report.*/
title "Customer details";
proc print data= data.customer_dim label;
run;

/* Select only observation having age greater then 30
years. Delete the Obs column. Check that the dataset contains 55
observations.  */
proc print data= data.customer_dim label noobs;
  where Customer_Age>30;
run;

/* Add a statement to use the variable Costumer_ID instead of the
Obs column as ID column. For this report, delete the title
``Costumer details'' */
title;
proc print data= data.customer_dim label noobs;
  where Customer_Age>30;
  id customer_id;
run;
```

Example 1.3.8 (Esercizio 4 lezione 2)**.** Si ha:

```
/* set libname if not already done */
/* libname data "...\Data"; */

/* Order the dataset data.employee_payroll by employee_gender and,
within gender, sort by salary in decreasing order. store the
sorted observations in a temporary dataset named sort_sal
*/
proc sort data=data.Employee_payroll out=sort_sal;
  by Employee_Gender descending Salary;
run;

/* Run a proc print of a subset of sort_sal. Select only the
observations of emplyees with a salary greater than
65.000. Group the report by Employee_Gender and include a total
and a subtotal for Salary. Keep only Employee_ID, Salary and
Marital_Status.*/
proc print data=sort_sal;
  by Employee_Gender;
  sum Salary;
  where Salary > 65000;
  var Employee_ID Salary Marital_Status;
run;
```

Example 1.3.9 (Esercizio 5 lezione 2)**.** Si ha:

```
/* set the library*/
/* libname data "...\Data"; */

/* Create a report with a subgroup of data.employee_addresses
dataset as the one shown below. Prior to the report, sort the
dataset by State, City, Employee_Name. The report displays
observations sorted by State.
*/
proc sort data=data.employee_addresses out=work.address;
  where country="US";
  by state city employee_name;
run;
title "US Employee by State";
proc print data=work.address noobs label;
  var employee_id employee_name city postal_code;
  by state;
run;
```

Capitolo 2

Data management

2.1 Data manipulation

Important remark 12. Data manipulation is done inside a `data` step; the template for this is the following code (that currently copies input dataset to output, without changing):

```
data output.dataset;
  set input.dataset;
run;
```

2.1.1 Selecting variables: `drop`, `keep`

Definition 2.1.1 (`drop` statement)**.** In a `data` step removes one or more variables from the input dataset:

```
data fev_selection;
  set fev;
  drop name week_6;
run;
```

Definition 2.1.2 (`keep` statement)**.** In a `data` step, all the variables written after `keep` are retained in the dataset, while the other discarded:

```
data fev_selection;
  set fev;
  keep id baseline;
run;
```

2.1.2 Constructing expression: operators

Important remark 13. Expression are a building block for selecting observation, creating new variables etc, so will be largely used in the following.

Definition 2.1.3 (Expression)**.** A composition of variable names, operators and constant (number or `"strings"`).

Definition 2.1.4 (Operators)**.** Regarding operators:

- **arithmetic operator**: `+`, `-`, `*` `/` and `**` (exponentiation)
- **logical operator**: `and`, `or`, not done by `^`
- **comparison operator** are reported below
  - `=` or `eq`, equal to
  - `^=` o `ne`, different from
  - `>` o `gt`, greater than
  - `<` `lt`, less than
  - `>=` `ge` greater or equal
  - `<=` `le` less or equal
  - `in` equal to an element of the following list, specified as `(constant1, constant2)` or `(constant1 constant2)`
  - `not in` not present in the following list, specified as for `in`

Important remark 14. Peculiar to SAS are the syntax for exponentiation, negation, equal (similar to assignment), `in` and `not in`.

2.1.3 Selecting observation: `IF`, `where`

Definition 2.1.5 (`if/where` statements)**.** Inside a `data` step, both `if` and `where` statements are used to select observation to keep

```
data fev_high;
  set fev;
  if expression; /* same as where expression*/
run;
```

Example 2.1.1 (Basic `if`)**.** Create the new dataset fev1high containing all observations of the `fev` dataset with a `baseline` value > 3

```
data fev_high;
  set fev;
  if baseline > 3;   /* same obtained with where instead of if */
run;
```

Example 2.1.2 (Missingness handling)**.** From `fev` select only records with non missing observations for `week_6`

```
data fev_notmiss;
  set fev;
  if week_6 ne .;
run;
```

Example 2.1.3 (Selecting if present in a list: `in`)**.** It's done using `if` with `in`

```
data fev_name;
  set fev;
  if name in ("Murray, W" "LaMance, K");
run;
```

Definition 2.1.6 (Differences between `if` and `where`)**.** We have:

1. `if` can be used only in a `data` step, while `where` can be used both in the `data` and `proc` step.
2. `where` statement in a `proc` step make only the observations that meet the specified condition to be taken into consideration by the procedure, eg

   ```
   /* shows only the observations with FEV1 at
   week 6 that are not missing */
   proc print data=fev;
      where week_6 ne .;
   run;
   ```

3. `if` statement with `then` keyword can be used for other than selecting rows (while `where` is used only to filter).

Important remark 15. My take home message: use `if` with `data` steps and `where` for `proc` steps.

Definition 2.1.7 (`if-then`)**.** `if-then` statement specifies to execute a certain action iff a condition is verified. Syntax is:

```
if condition then action;
```

Some action are `output`, `delete`.

Definition 2.1.8 (`output` action)**.** `output` is used to redirect observation to the new dataset (save them); it's the default action and could be omitted

```
data fev_high;
  set fev;
  if baseline > 3 then output;
run;
```

It will be especially useful when we want to redirect some observations to a dataset and some other to another (using `if` to handle it all).

Definition 2.1.9 (`delete` action)**.** If we specify `delete` as action, SAS will delete the observation from the new dataset.

```
data fev_high;
  set fev;
  if baseline > 3 then delete;
run;
```

Important remark 16. Therefore simple `output`/`delete` statements are the row equivalent for variables `keep` and `drop`.

2.1.4 Creating variables

Definition 2.1.10 (Variable creation)**.** Inside a `data` step one can create a variable

- using the syntax

```
varname = expression;
```

- using `if-then-else` structure in a conditional way (see example below)

Computation involving *missing values* will result in missing

Example 2.1.4 (Simple variables)**.** We have

```
data fev_new;
  set fev;
  site = "Bologna";              /* a common constant for every observation*/
  mean = (baseline + week_6)/2;  /* an expression depending on other data*/
run;
```

Example 2.1.5 (Conditional dummy with `if-then`)**.** We have

```
data fev_flag2;
  set fev;
  if week_6 > 3 then high=1;
  else if week_6 ne . then high=0; /* se questa riga mancava
                                      i rimanenti erano tutti missing */
run;
```

Example 2.1.6 (Creating *two* dataset in a single step)**.** Check it out

```
data under40 over40; /* <- look */
  set sashelp.cars;
  keep make model msrp cost_group;  /* actually keep is done at the end of the st
  if msrp < 20000 then do;
    cost_group = 1;
    output under40;
  end;
  else if msrp < 40000 then do;
    cost_group = 2;
    output under40;
  end;
  else do;
    cost_group = 3;
    output over40;
  end;
run;
```

2.1.5 Using SAS functions

Definition 2.1.11 (SAS Functions)**.** Handy routines used to operate on variables/values in expression:

- the syntax is

  ```
  function(arg1, arg2, ...)
  ```

- can be divided in groups according to the input they handle: numeric values only, character values only, both numeric and chars.

Definition 2.1.12 (Numeric functions)**.** A list:

- `mean(arg1, arg2, ...)`: mean of the non missing values (arguments can be numeric variable or constants). Eg `mean(3,.)=3`
- `sum(arg1, arg2,...)`: sum of the non missing args
- `max(arg1, arg2,...)`: max of the non missing args
- `min(arg1, arg2,...)`: min of the non missing args
- `log(arg)`: natural logarithm
- `exp(arg)`: the exponentia
- `sqrt(arg)`: the square root.
- `abs(arg)`: absolute value
- `int(arg)`: integer part (eg `int(2.8)=2`)
- `round(arg, rounding unit)`: returns the value rounded to the nearest multiple of the rounding units, eg

  ```
  round(1.736, 1) -> 2
  round(1.736, .1) -> 1.7
  round(1.736, .01) -> 1.74
  round(1.736, .001) -> 1.736
  ```

Definition 2.1.13 (Character functions)**.** another list:

- `upcase(arg)`: converts all the argument letters in upper case (eg `upcase("Bologna")` returns ‘‘BOLOGNA’’)
- `cats(arg1, arg2, ...)`: concatenate the arguments in a unique text string. The arguments can also be numeric values.
- `catx(separator, arg1,arg2,...)`: concatenate the arguments in a unique text string and it inserts a delimiter/separator specified as first argument of the function. Arguments to concatenate can also be numeric values.
- `substr(arg, starting position, length)`: extracts from the argument a substring that has a length specified, starting from the starting position of the original string. Eg `substr("Bologna", 4, 4)` returns `"ogna"`

Definition 2.1.14 (Both char and numeric function)**.** A final list:

- `missing(arg)`: returns value 1 if the argument is missing, 0 otherwise.
- `cmiss(arg1, arg2,...)`: returns the count of missing arguments.

Important remark 17 (Other functions)*.* SAS has several other function, look here.

Example 2.1.7 (`mean`)**.** Mean calculation by using an expression or the `mean` function:

```
data fev_means;
  set fev;
  mean1 = (baseline+week_6)/2;
  mean2 = mean(baseline,week_6);
run;
```

Example 2.1.8 (Functions in `if` statement)**.** Use of functions with an `if` statement:

```
data fev_highmean;
  set fev;
  if mean(baseline,week_6) > 3;
run;
```

Example 2.1.9 (`cats` and `catx` usage)**.** Eg.

```
data fev_cat;
  set fev;
  subject1 = cats(id, name);
  subject2 = catx(" ", id, name);
  drop baseline week_6;
run;
```

Example 2.1.10 (`missing` and `cmiss` use)**.** An example:

```
data fev_missing;
  set fev;
  missing_w6 = missing(week_6);
  nmissing = cmiss(id, name, baseline, week_6);
run;
```

Example 2.1.11 (Combining function)**.** An example:

```
data fev_combo;
  set fev;
  subject = upcase(substr(name, 1, 3));
  mean = round(mean(baseline, week_6), .1);
run;
```

Example 2.1.12 (Exercise 1 lection 3)**.** Starting from the `trial` dataset, create the `trialnew` dataset that contains the following variables:

- `id` (subject identifier), obtained by concatenating the `site` and the patient number `ptno` (e.g. 14);
- `mean`, `min`, `max` (i.e. mean, maximum and minimum between each obs numeric measurements)
- `high`, a flag variable that has value `y` if the mean is > 30, else `n`

By using `proc print`, visualize only `id` of the patient with `flag` variable equal to `y`.

```
data trial_new;
  set trial;
  id = cats(site, ptno);
  mean = mean(v1_score, v2_score, v3_score);
  min = min(v1_score, v2_score, v3_score);
  max = max(v1_score, v2_score, v3_score);
  if mean > 30 then flag="y";
  else flag = "n";
  keep id mean min max flag;
run;

proc print data=trial_new;
  where flag="y";
  var id;
run;
```

Example 2.1.13 (Exercise 2 lesson 3)**.** Starting from `medhis` data set, which is saved in the `data` folder, create a dataset named `t1` that contains only pathologies occurred after 2005.

```
data t1;
  set data.medhis;
  if substr(stdt,1,4) >= '2005' and stdt ne 'nd' then output;
run;

proc print;
run;
```

2.1.6 `by` with `set`, `.first` and `.last`

Important remark 18. Useful for duplicate finding.

Important remark 19 (Usage of `by` with set)*.* When creating a new dataset, starting from an *ordered* original dataset using `set`, `by varname` statement creates internally two variables:

- `first.varname` is a dummy equal to 1 if the current observation is the first show of a value for that `varname`, 0 otherwise
- `last.varname` is equal to 1 if the current observation is the last show of a value for `varname`, 0 otherwise

Example 2.1.14 (Exercise 3 lesson 3)**.** Starting from the dataset `data.medhis`, that contains one or more records for the same subject, keep for each subject the first previous illness (variable `hisdis`) in alphabetic order (using a combination of `first.variable` and output statement)

```
proc sort data=data.medhis out=med1;
  by pt hisdis;
run;

data med2;
  set med1;
  by pt;
  if first.pt then output;
run;
```

Example 2.1.15 (Exercise 4 lesson 3)**.** Starting from the dataset `data.header`, that is ordered by patient and the visit number, keep for each subject only the record about the last visit, unless this visit is the only one (in this case do not keep the record).

```
proc sort data=data.header out=vis1;
  by pt visit;
run;

data vis2;
  set vis1;
  by pt visit;
  if last.pt and not first.pt then output;
run;
```

Example 2.1.16 (Exercise 5 lesson 3)**.** The steps are:

1. create a `libref` named data that refers to the folder `lession3/data`, where the dataset for the exercise is saved
2. `au_salesforce` dataset contains data on sales employees in Australia. Orion Stars Enterprise wants to create a user ID, by using the following rules:
   - taking the first letter from the employee name (`first_name`)
   - taking the first 3 letters from the employee name (`last_name`)
   - concatenating the texts obtained from the 2 previous steps, separating them by an underscore
   - putting all the letters in uppercase

   Do the previous steps in 2 different ways: first step by step, then in one single statement, by combining different functions
3. print the observations maintaining only the variables `First_name`, `Last_name` and the new variable which contains the user ID.

```
/* 1 */
libname data "...\Lection 3\Data";

/* 2 */
data work.id;
  set data.au_salesforce;
  a = substr(first_name, 1, 1);
  b = substr(last_name, 1, 3);
  id_low = catx("_", a, b);
  id = upcase(id_low);
run;
```

```
data work.id02;
  set data.au_salesforce;
  id = upcase(catx("_", substr(first_name,1,1), substr(last_name,1,3)));
run;

/* 3 */
proc print data=work.id;
  var first_name last_name id;
run;
```

Example 2.1.17 (Exercise 6 lesson 3)**.** The steps are:

1. do the first bullet point of the previous exercise if not already done.
2. write a `data` step for creating a new dataset `work.region`, using `data.supplier` as input dataset, doing the following manipulations:
   - if `country` variable equals `ca` or `US` then the new variable named `region` will be equal to `north america`
   - If Country variable equals any other value then the new variable named `region` will be equal to `other`
   - include the following variables in the new dataset: `supplier_name`, `country` and `region`
3. produce a report that shows only records from `north america`
4. could we alternatively use the `if` statement in the previous `proc print`?

```
/* 2 */
data work.region;
  set data.supplier;
  if country in ('ca' 'us') then region='north america';
  else region='other';
  keep supplier_name country region;
run;

/* 3 */
proc print data=work.region noobs;
  where region="north america";
run;
```

Example 2.1.18 (Exercise 7 lesson 3)**.** The steps are:

1. do the first bullet point of the exercise 5 if not already done.
2. Write a `data` step for creating a new dataset `work.bigdonations`, using `data.employee_donations` as input dataset, doing the following manipulations:
   - create the variable `total`, which contains the sum of the donations of each quarter (for each subject, `qtr1` contains the donation given during the first quarter, `qtr2` contains the donation given during the second quarter, and so on)

- create the variable `numqtrs`, which contains the number of quarters in which the employee did a donation (i.e. the number of non-missing `qrt1-qrt4`)
- do not include `recipients` and `paid_by` variables
- select only the observations that verify the following 2 conditions:
  - `total` should be higher than 50
  - `numqtrs` should not be 4

3. produce a report of the bigdonations dataset that shows only variables `employee_id` and `total`, and setting the title to "Donations".

```
/* 2 */
data work.bigdonations;
  set data.employee_donations;
  total= sum(qtr1, qtr2, qtr3, qtr4);
  numqtrs = 4 - cmiss(qtr1, qtr2, qtr3, qtr4);
  drop recipients paid_by;
  if total > 50 and numqtrs ^= 4;
run;

/* 3 */
title "Donations";
proc print data=work.bigdonations noobs;
  var employee_id total;
run;
```

2.2 Informat and format

Important remark 20. Some truth:

- `informats` are instructions for *reading* data into a SAS
- `formats` are instructions for *visualization/reporting* data in datasets/reports
- both `informat/format` use a *common syntax*; we'll see the most commonly used of them

2.2.1 Informat

Definition 2.2.1 (informat)**.** Instruction that specifies to SAS the *input format* of data and is used to correctly read data values into a variable.

Example 2.2.1. To remove the dollar sign and commas before storing the numeric value 1000000 in a variable, SAS reads this value with the DOLLAR10.

```
input value    -> informat   -> STORED Value
$1,000,000     -> DOLLAR10.  -> 1000000
```

Definition 2.2.2 (Informat syntax)**.** The following applies:

```
$<informat>w.         /* characters*/
<informat><w>.<d>     /* Numeric */
```

where:

- `$` indicates a character informat
- `informat` refers to the sometimes optional informat name
- `w` indicates the number of columns to read
- `.` is mandatory
- `d` is the number of decimals digits (optional)

Example 2.2.2 (Numeric informats examples)**.** We have:

```
input  informat  stored
123    3.0       123
123    3.        123
123    3.1       12.3
```

When input data contains comma, SAS ignore the `d` specified

```
input  informat  stored
1.23   4.1       1.23
1.23   3.1       1.2     <- w must contain the digit in count!
```

Example 2.2.3 (Other examples)**.** `input        informat  stored`

```
123,456.78  comma10.  123456.78
$123,456.78 dollar11. 123456.78
```

Important remark 21 (Dates functioning)*.* SAS stores dates as days passed from 1 January 1960; to read them we use have several informat

Example 2.2.4. Date 9 October 2009 for corresponds to the value 18179 (number of days since 1-1-1960); to read it we have

```
input        informat
09/10/09     ddmmyy8.
09/10/2009   ddmmyy10.
09OCT09      date7.
09OCT2009    date9.
```

Important remark 22 (`informat` and `datalines`)*.* Datalines and informat can be used to create dataset on the fly

Example 2.2.5. We want to create the dataset `h` with `height` (in m) and collection `date which` (a date). If data is

```
140 05AUG04
164 03MAR08
173 12SEP09
```

we report it between datalines statement and a `;` line, as in

```
  data h;
    input @1 height 3.2 @5 date date7.;
    datalines;
    140 05AUG04
    164 03MAR08
    173 12SEP09
    ;
 run;
```

2.2.2 Format

Important remark 23 (Difference between `format` and `informat`). There is an important difference between `format` and `informat` in this example: `informat` intervenes when the information are stored in the dataset; on the other hand, `format` is applied only to the data appearance (it doesn't modify the actual value in the dataset).

Remark 3. `format`/`informat` useful especially for dates.

Definition 2.2.3 (`format`). Instruction that specifies to SAS to change the appearance of a variable (without changing the stored value); the syntax is the same as `informat`

```
$<format><w>. /* character variables*/
<format>w.<d> /* numeric variables*/
```

with:

- `$` indicates a character format
- format refers to the sometimes optional format name
- `w` indicates the length of the format (also special
- `.` is mandatory
- `d` is the number of digits/decimals

Example 2.2.6. For the numeric variables :

- `w.d` use `w` spaces rounding at `d` decimals
- `w.` uses `w` spaces rounding at nearest integer

```
value    format  visualized
145.254    7.3     145.254      /* 7 spazi (6digit + comma di input), 3 digit visuali
145.254    8.3     145.254      /* qui molti spazi concessi inutilizzati  nel dato di
145.254    8.2     145.25       /* qui molti spazi  concessi inutilizzati nel dato di
145.254    6.2     145.25
145.254    5.1     145.3

145.254    3.      145           /*<- w. */
```

For the char variables `w` spaces are shown

```
value       format  visualized
Gargamella  $20.    Gargamella
Gargamella  $8.     Gargamel
Gargamella  $4.     Garg
```

For dates, eg 9 October 2009 for SAS corresponds to the value 18179; there are some alternatives

```
value   format     visualized
18179   ddmmyy8.   09/10/09
18179   ddmmyy10.  09/10/2009
18179   date7.     09OCT09
18179   date9.     09OCT2009
```

Definition 2.2.4 (`format` statement)**.** To apply formats we do it in a format statement, which can be set in a proc print step (for reportint) or in a data (for dataset visualization)

```
/* in a proc print step*/
proc print data=h;
  format height 3.1 date date7.;
run;

/* in a data step*/
data h_formatted;
  set h;
  format height 3.1 date date7.;
RUN;
```

Important remark 24. `h` and `h_formatted` will contain the same data but visualized differently.

Important remark 25 (Recycling formats). If you want to give the same format to different variables, you can use specify all the variable first and then the common format. Eg

```
format date_a date_b date7.         /* <- this is compact and ok*/
format date_a date7. date_b date7.  /* <- this is ok but verbose*/
```

2.2.3 Converting numerics and chars

2.2.3.1 From character to numeric: `input`

Definition 2.2.5 (`input` function)**.** Returns a numeric value of the parsed character variable using the following syntax inside a data step

```
input(source, informat.)
```

where:

- `source`: specifies a character constant, variable, or expression
- `informat`: is the SAS informat that you want to apply to the source

Example 2.2.7 (`input` function)**.** Eg:

```
data s_format;
  set s;
  price = input(pricec, 4.);
run;
```

Example 2.2.8 (Exercise 1 lesson 4)**.** `purchase` SAS dataset contains some information about a company customers; in particular, the variables `purchase` and `purchase2` are the expenses that each customer made for the first 2 purchases.
Starting from `purchase`, create a new dataset which contains the total expenses for each customer.

```
libname data '...\Data';

data purchase_tot;
  set data.purchase;
  tot_purchase = input(purchase, dollar7.) + input(purchase2, dollar7.);
run;
```

2.2.3.2 From numeric to character: `put`

Definition 2.2.6 (`put` function)**.** The syntax is

`put (source, format.)`

where:

- `source`: constant, variable (character or numeric), or expression to reformat.
- `format`: is the SAS format to apply (the argument must be follwed by a period and optional width and decimal specifications)

Example 2.2.9 (`put` function)**.** Eg:

```
data s_format;
  set s;
  datec = put(miadata, date9.); /* l'output è un char tipo 01JAN2002*/
run;
```

Important remark 26. If the `put` function returns a character value to a variable that has not yet been assigned length, the variable length is determined by the width of the format. So, in the previous example w was equal to 9.
The same goes for `input` function.

Example 2.2.10 (Exercise 2 lesson 4)**.** Starting from `purchase`, create a new dataset which contains a new customer id that is the concatenation of the original `customer_id` and the customer date of birth.

```
data purchase_id;
  set data.purchase;
  customer_id_new = catx("-", customer_id, put(birth_date, date7.));
run;
```

2.2.4 User defined format

Important remark 27. One:

1. creates a user-defined formats using the `proc format` procedure
2. applies it using a `format` statement in a report

Let's see the two steps.

Definition 2.2.7 (Step 1: proc format/value statement)**.** Used to define a custom format, the syntax is:

```
proc format;
  value format-name range1 = 'label '
                    range2 = 'label '
                    ...;
run;
```

Regarding the

- `format-name`:
  - up to 32 chars
  - begin with a dollar sign followed by a letter or underscore (for character formats) or to begin with a letter or underscore (numerics)
  - cannot end with a number, have a system/SAS format name
  - cannot include a period in the value statement.
- each `range` can be a single value, a range of values or a list of values
- `labels` up to 32767 chars and enclosed in quotation marks

Example 2.2.11 (Format definition)**.** For char formats

```
proc format;
  value $ctryfmt 'AU'='Australia'
                 'US'='United States'
                  other='Miscoded'; /* matches all remaining values*/
run;
```

For numeric formats:

```
proc format;
  value placement 1 = 'First'
                  2 = 'Second'
                  3 = 'Third';
run;
```

Numeric format with *range* specification:

```
proc format;
  value tiers    low-<50000  ='Tier 1'    /* < near 50000 excludes it */
               50000-<100000 ='Tier 2'    /* 100000 excluded */
              100000-high    ='Tier 3';
run;
```

some points:

- `low` does not includes missing variables for numeric variables (yes for characters ones)
- less than symbol excludes the *near* number, eg `50000<-100000` excludes 50000 but includes 100000

Finally ranges can be specified using lists, ranges discrete values and keywords:

```
proc format;
  value mnthfmt 1,2,3='Qtr 1'
                  4-6='Qtr 2'
                  7-9='Qtr 3'
                10-12='Qtr 4'
                    .='missing'
                other='unknown';
run;
```

Important remark 28 (Multiple value statements). Several `value` statements can be included in a single `proc format` step, eg

```
proc format;
  value $ctryfmt  'AU'='Australia'
                  'US'='United States'
                other ='Miscoded';
  value tiers    low-<50000  ='Tier 1'
               50000-<100000='Tier 2'
              100000-high   ='Tier 3';
run;
```

Definition 2.2.8 (Step2: using formats with `format` statement in `proc print`)**.** Once defined we can use the formats for prettyprinting as usual

```
proc print data = db;
  var var1 var2 var3;
  format var1 format-name1 var2 var3 format-name2;
run;
```

Example 2.2.12. we have

```
proc print data=orion.sales;
  var Employee_ID Job_Title Salary Country Birth_Date Hire_Date;
  format Salary dollar10.
         Birth_Date Hire_Date monyy7.
         Country $ctryfmt.;
run;
```

Example 2.2.13 (Exercise 3 lesson 4)**.** Starting from the Demog dataset, create a new dataset where:

- With an input command and the associated informat, create a new variable "data" with the birth dates. Format the variable as `xx/xx/xxxx`.
- Format weight and height with one decimal

- Format sex=1 for male and sex=2 for female

```
proc format;
  value $sex '1'='Male' '2'='Female';
run;

data t5;
  set data.demog;
  datadinascita=input(dob, yymmdd8.);
  format datadinascita date9. wt ht 5.1 sex $sex.;
run;
```

Example 2.2.14 (Exercise 4 lesson 4)**.** Starting from the Excel sheet datesas.xls,

1. import the file creating a temporary file named T1
2. create a dataset named T2 that contains the variable `data_sas` with the dates in visdt and put them in SAS format.
3. starting from the dataset T2, print the data in order to visualize `data_sas` formatted as `xx/xx/xxxx`
4. starting from the dataset T2, create a temporary dataset T3 where the variable `data_sas` is formatted as `01JAN2016`
5. starting from the dataset T2, print the data with the variable Visit formatted as "Visit x".

```
/* step 1-2*/
proc import out=t1
  datafile='...\datesas.xls'
  dbms=excel replace;
  range='datesas$';
  getnames=yes;
run;

data t2;
  set t1;
  data_sas=input(visdt,yymmdd8.);
run;

proc print; /* print without formatting */
run;


/* point 3 */
proc print data=t2;
  format data_sas ddmmyy10.;
run;

/* point 4 */
```

```
data t3;
  set t2;
  format data_sas date9.;
run;

/* point 5*/
proc format;
value visit 1='Visit 1'
            2='Visit 2'
            3='Visit 3'
            4='Visit 4'
            5='Visit 5'
            6='Visit 6';
run;

proc print data=T2;
  format visit visit.;
run;

data t4;
  set t3;
  visit_ = cat("visit ",visit);
run;
```

Example 2.2.15 (Exercise 5 lesson 4).

1. Create a libref named data that points to the folder where this dataset is stored.
2. With a data step, create work.hire from data.sales applying the following manipulation: create the variable `Hire_age`, that contains the age of the employee when he/she was hired (tip: use the variables that contain the birth date and the hiring date: divide by 365.25 to obtain the value in years)
3. Run a proc print that:
   - Selects only the US employees;
   - contains `Employee_ID First_name Last_name Job_title Birth_Date Hire_Date Hire_age`;
   - Formats `Birth_Date` as `ddmmyy10.`, `Hire_date` as `date9.` and displays `Hire_age` as an integer

   Add an appropriate title

```
/*POINT 1*/
libname data "...\Data";

/*POINT 2*/
data work.hire;
  set data.sales;
  hire_age = (Hire_Date-Birth_Date)/365.25;
run;
```

```
/*POINT 3*/
title 'Hire age of sales employees';

proc print data=hire noobs;
  where country='US';
  var Employee_ID First_name Last_name Job_title Birth_Date
      Hire_Date Hire_age;
  format Birth_Date ddmmyy10. Hire_date ddmmyy10. Hire_age 2. ;
run;
title;
```

Example 2.2.16 (Exercise 6)**.** 1. complete point 1 of the previous exercise if not done yet (otherwise do not repeat it)

2. Create the character format `$GENDER` so that values are visualised as follows: F=Female, M=Male, other stuff = NA

3. Create the numeric format `SALRANGE` so that the values of salary are displayed as follows

```
Less then 100000 (excluded)      --> Less then $100,000
Greater than or equal to 100000 --> Greater than or equal to $100,000
Missing                          --> Missing Salary
Any other value                  --> Non valid Salary
```

4. How could we optimize the code joining points 2 and 3?

5. Use the created formats on variables Gender and Salary stored in data.nonsales.

```
/*POINT 2*/
proc format;
  value $gender 'F'='Female' 'M'='Male' other='NA';
run;

/* POINT 3*/
proc format;
  value salrange low -< 100000 = 'Less then $100,000 '
                 100000- high = 'Greater than or equal to $100,000'
                            . = 'Missing Salary'
                        other = 'Non valid Salary';
run;

/* POINT 4 */
proc format;
  value $gender 'F'='Female' 'M'='Male' other='NA';
  value salrange low -< 100000 = 'Less then $100,000 '
                 100000- high = 'Greater than or equal to $100,000'
                            . = 'Missing Salary'
                        other = 'Non valid Salary';
```

```
run;

/* POINT 5 */
proc print data=data.nonsales;
  format gender $gender. salary salrange.;
run;
```

Example 2.2.17 (Exercise 7 lesson 4)**.** 1. Complete point 1. of the previous exercise if not done yet (otherwise do not repeat it).

2. Create the dataset `work.US_num` with a data step using `data.us_newhire` as input, with the following manipulations:
   - Create the numeric variable `Birth_Date` from the variable `Birthday` and apply to it the `date9` informat
   - Extract the last 4 numbers from the character variable `ID`, storing them in a new character variable
   - Create the numeric variable `ID_n`, that contains the 4 saved numbers from the previous point in a numeric format.
3. run a proc print keeping only the variables `ID_n` `Birth_Date` and `Telephone`. Apply the format `ddmmyy10` to the variable `Birth_Date`.

```
/*point 2*/
data US_num;
  set data.us_newhire;
  Birth_Date =input(Birthday,date9.);
  last=substr(id,12,4);
  ID_n=input(last,4.);
run;

/*point 3*/
proc print data=US_num;
  var id_n Birth_Date Telephone;
  format Birth_Date ddmmyy10.;
run;
```

2.3 Transpose

Definition 2.3.1 (proc TRANSPOSE)**.** Transposing from long to wide or viceversa is done with `proc TRANSPOSE`, which syntax is

```
proc transpose data=<datasetName> out=<datasetName>;
  by <variablesList>;
  var < variablesList >;
  id <variable>;
run;
```

where

- `data=input-data-set`: names the SAS data set to transpose (most recently created SAS data set if missing).

- `OUT=output-data-set`: name the output data set.
- `VAR varlist` lists variables to transpose. If no variable is specified, by default it transposes all numeric variables in the data set.
- `by <variables>`: specifies the variables that identify the rows, whose values will not be transposed. The variables in by statement should be *ordered with a proc sort.*
- `ID <variables>`: specifies a variable whose values name the transposed variables. There could be only one ID variable and it shouldn't contain duplicates.

Furthermore we can specify these statements:

- `PREFIX=prefix`: a prefix to use in constructing names for transposed variables in the output data set. For example, if `PREFIX=VAR`, variables will be named `VAR1`, `VAR2`, . . . , `VARn`
- `LABEL=label`: specifies name for the variable in the output data set that contains the label of the variable that is being transposed to create the current observation. The default value is `_LABEL_`
- `NAME=name` specifies the name for the variable in the output data set that contains the name of the variable being transposed to create the current observation. The default value is `_NAME_`

Important remark 29. Before using proc transpose you should order the original dataset for the `by`-variable(s).

Example 2.3.1 (Exercise 1 lesson 5)**.** Assume that you want to calculate the difference between one visit and next one in the dataset Header. Since you have the visit dates in columns you can't use the difference function. You should create a column for each visit that contains the dates for the visit.
Using the dataset of our previous lesson, transpose the dataset HEADER, that contains the visit dates for each subject. In the output dataset you should obtain only one record per subject, with as many variables as the number of registered visits.

```
libname data '.../Lection5/Data';

data t1;
  set data.header;
  visitdt=input(visdt,yymmdd8.);
  format visitdt date9.;
run;

proc sort data=t1 out=t2;
  by pt visitdt;
run;

proc transpose data=t2 out=t3;
  by pt;
  var visitdt;
run;

data t4;
  set t3;
  diff1=col2-col1;
  diff2=col3-col2;
  diff3=col4-col3;
  diff4=col5-col4;
  diff5=col6-col5;
  diff6=col7-col6;
  diff7=col8-col7;
run;
```

Example 2.3.2 (Exercise 2 lesson 5)**.** Repeat Exercise 1, this time use the prefix option to construct names for transposed variables in the output data set. The prefix of the new variables should be as follows: `Date_Of_VisitN`

```
/* Exercise 2 */
proc transpose data=t2 out=t4 prefix=Date_Of_Visit;
  by pt;
  var visitdt;
run;
```

Example 2.3.3. Transpose the dataset t5 (that contains the visit dates for the first 3 patients) and modify the data so that patient 3 has the visit 4 missing. In the output dataset you should obtain only one record per subject, with as many variables as the number of registered visits, where the visit dates are in the column corresponding to the value of the variable Visit.

```
/* Exercise 3 */
/* Selectinon of the 3 first patients*/
data t5;
  set t1;
  where pt in ('011001' '011002' '011003');
run;

/* I want the visit 4 of patient 3 to be missing*/
data t6;
  set t5;
  if pt='011003' and visit=4 then visitdt=.;
run;

proc sort data=t6 out=t7;
  by pt visitdt;
run;

/* transpose without ID */
proc transpose data=t7 out=t8;
  by pt;
  var visitdt;
```

```
run;

/* transpose with ID */
proc transpose data=t7 out=t9;
  by pt;
  var visitdt;
  id visit;
run;
```

2.4 Concatenate

Important remark 30. rbind equivalent: path is different if the two dataset have the same structure or not

Definition 2.4.1 (Concatenate with similar dataset)**.** we use several dataset in the `set` statement of a data step as in

```
data output;
   set input1 input2 ...;
run;
```

where `set` reads observations from each data set in the order in which they are listed

Important remark 31. If variable of the input dataset have different names (even partial) SAS will be conservative and will create separate variables for the non matching ones with missing values, unless the `rename` option is used in set statement.

Example 2.4.1. if `input1` = First, Gender, Country and `input2` = First Gender Region (but both Country and Region have a State) then SAS will create `output` = First, Gender, Country, Region unless we code something like

```
data empsall2;
  set empscn empsjp(rename=(Region=Country));
run;
```

Definition 2.4.2 (Using `RENAME` option)**.** The syntax is

```
SAS-data-set (RENAME = (old-name-1=new-name-1
                        old-name-2=new-name-2
                        ...
                        old-name-n=new-name-n))
```

where

- The `RENAME` option must be specified in parenthesis immediately after the appropriate SAS data set name
- The name change affects the output data set only, i.e. it has no effect on the input data set

2.5 Merge

Important remark 32. We have several situation tackled in what follows

1. `1 to 1`: a single observation in one data set is linked to exactly one observation in another data set based on the values of one or more selected variables.
2. `1 to many`: a single observation in one data set is linked to more than one observation in another data set based on the values of one or more selected variables.
3. `non matches`: at least one observation in one data set is linked to no observation in another data set based on the values of one or more selected variables.

Definition 2.5.1 (`merge`/`by` statements in `data` step)**.** We need `merge` statement (to specify datsets), used with `by` statement (to specify matching variables, to merge datasets. The format is

```
merge SAS-data-set1 SAS-data-set2 ...;
by <DESCENDING> variable(s);
```

with the following requirements:

- two or more datasets are needed
- variables in by must be available in all datasets
- data sets must be *sorted* (using proc sort) by the variables listed in the `by` statement

Example 2.5.1. Despite several 1-1, 1-m situation the syntax is common:

```
data empsauh;
  merge empsau phoneh;
  by EmpID;
run;
```

2.5.1 Merging with missing id variables

Important remark 33. If there are missing in the key in one or both datasets, all will be kept, eg merging the two dataset by Empid below

```
First Gender  EmpID      EmpID  Phone
Togar      M 121150     121150 +61(2)5555-1795
Kylie      F 121151     121152 +61(2)5555-1667
Birin      M 121152     121153 +61(2)5555-1348
```

will give both matches and non matches

```
First Gender EmpID Phone
Togar     M 121150 +61(2)5555-1795
Kylie     F 121151
Birin     M 121152 +61(2)5555-1667
            121153 +61(2)5555-1348
```

Important remark 34. We can handle what records are kept by using `IN` option and if to select

Definition 2.5.2 (`IN`)**.** The `IN=` option creates a variable that indicates whether the data set contributed building the current observation

```
merge SAS-data-set (IN=tmpvar) ...
```

tmpvar is a temporary numeric variable valued 1 if the data set did contribute to the current observation or 0 otherwise. We can then use if for selecting.

Example 2.5.2. Merging and selecting employees that have company phones:

```
data empsauc;
  merge empsau(in=Emps) phonec(in=Cell);
  by EmpID;
  if Emps=1 and Cell=1;
run;
```

Select employees that do not have company phones:

```
data empsauc;
  merge empsau(in=Emps) phonec(in=Cell);
  by EmpID;
  if Emps=1 and Cell=0;
run;
```

Select company phones associated with an invalid employee ID:

```
data empsauc;
  merge empsau(in=Emps)  phonec(in=Cell);
  by EmpID;
  if Emps=0 and Cell=1;
run;
```

Select all non-matches

```
data empsauc;
  merge empsau(in=Emps) phonec(in=Cell);
  by EmpID;
  if Emps=0 or Cell=0;
run;
```

Example 2.5.3 (Exercise 4 lesson 5)**.**

1. Merge One-to-One: merge the dataset demog with the result of the proc transpose of the dataset header
2. Merge One-to-Many: merge the dataset demog with the dataset header

```
/* point 1*/
proc sort data=data.demog;
  by pt;
run;

data all;
  merge data.demog t4;
  by pt;
run;

/* point 2 */
data all2;
  merge data.demog t2;
  by pt;
run;
```

Example 2.5.4 (Exercise 5 lesson 5)**.**

1. Create a libref named data linked to the folder where the dataset is contained (contained in `lesson_5/data`).

2. Run a proc contents for `data.sales` and `data.nonsales`. Compare the two datasets, which variables have different names?

3. Add a data step to concatenate the two datasets creating the new dataset `work.allemployees`. Change the names of the different variables and keep only `Employee_ID`, `First_name`, `Last_name`, `Job_title` and `Salary`.

```
/*Exercise 5*/
/*point 1*/
libname data '...\Lection V\Data';

/*point 2*/
proc contents data=data.sales;
run;
proc contents data=data.nonsales;
run;
/*The name of the variables in the two datasets differ in First and First_name fo
and Last and Last_name for the surname*/

/*point 3*/
data work.allemployees;
  set data.sales data.nonsales (rename=(First=First_name last=last_name));
  keep Employee_ID First_name Last_name job_title Salary;
run;
```

Example 2.5.5 (Exercise 6 lesson 5)**.**

1. Do point 1 of the previous exercise, if not done yet.

2. The dataset `data.product_list` contains the list of products sold by Orion. The dataset `data.supplier` contains the list of Orion suppliers. Two reports need to be created, the first one with the list of products to which it corresponds no supplier and the second with the list of products to which it corresponds a supplier. To do so:

   - Sort the dataset `data.product_list` by `Supplier_ID`. Call the output dataset `work.product`
   - Merge `work.product` and `data.supplier` by `Supplier_ID` and use an if statement to select observations for the final dataset `work.prodsup01` created with the only contribution of the dataset `work.product`

- Do the same as in the previous point, this time make sure that the dataset `work.prodsup02` contains only the observations that derive from both datasets

```
/*Exercise 6*/
/*point 1*/
*libname data '...\Lection V\Data';

/*point 2*/
proc sort data=data.product_list out=work.product;
  by supplier_id;
run;

data prodsup01;
  merge work.product (in=P) data.supplier (in=S);
  by supplier_ID;
  if P=1 and S=0;
run;

data prodsup02;
  merge work.product (in=P) data.supplier (in=S);
  by supplier_ID;
  if P=1 and S=1;
run;
```

Example 2.5.6 (Exercise 7 lesson 5). 1. Do point 1 of the previous exercise, if not done yet.

2. The dataset `order_summary` is structured like this: Each order per client is an observation. We want to count the total amount of orders per client in a year as follows (immagine sulle slides):

   - Transpose `order_summary`, so that the new dataset `cust_orders` contains one obs per client (use prefix="Month")
   - Run a `data` step to delete the `_NAME_` column and create the `Total` variable as the sum of orders amount per client for the year. Name the new dataset as `cust_orders_total`

```
/*Exercise 7*/
/*point 1*/
*libname data '...\Lection V\Data';

/*point 2*/
proc transpose data=data.order_summary out=cust_orders prefix=Month;
  by Customer_ID;
  id order_month;
  var sale_amt;
run;

data cust_orders_tot;
```

```
  set cust_orders;
  drop _NAME_;
  /* following is same as sum(month1,month2, ... ,month12); but less verbose*/
  Total=sum(of month1-month12);
run;
```

Capitolo 3

Programming

3.1 Arrays

Definition 3.1.1 (Array)**.** In a data step, it's a temporary way (end with the steps) to identify similar variables (all numeric or all characters) to be processed in the same way.
An array can refer to existing variables or not; in the latter case it will define/create new variables if used.

Important remark 35 (`array` statement syntax)*.* Inside a data step:

```
ARRAY array-name {subscript} <$> <array-elements>;
```

where

- `array-name` specifies the name of the array
- `subscript` describes the number of elements (can be enclosed by braces, tonde quadre graffe). If an asterisk is used SAS automatically counts the number of elements that will be part of the array
- `$` indicates character elements
- `array-elements` names of the elements that compose the array

Example 3.1.1 (Replacing `999` with `.`)**.** If array didn't exist:

```
data array_no;
  set dati.array_ds;
  if AGE = 999 then AGE=.;
  if HEIGHT = 999 then HEIGHT=.;
  if WEIGHT = 999 then WEIGHT=.;
  if VAR1 = 999 then VAR1=.;
  if VAR2 = 999 then VAR2=.;
  if VAR3 = 999 then VAR3=.;
  if VAR4 = 999 then VAR4=.;
run;
```

with array

```
data array_yes;
  set dati.array_ds;
  array a[7] AGE HEIGHT WEIGHT VAR1 VAR2 VAR3 VAR4; /* define an array with  the
  do i=1 to 7; /* for all the index */
    if a[i] = 999 then a[i] = .;     /* eg a[i] refereces to age */
  end; /* closes the instruction*/
run;
```

Same example with list of variables, `* dim`:

```
  array a[*] AGE HEIGHT WEIGHT VAR1-VAR4;  /* autocount the elements*/
  do i=1 to dim(a);                        /* dim in a do gives # of elements in
    if a[i] = 999 then a[i] = .;
  end; /* closes the instruction*/
```

Example 3.1.2 (Creating new vars with array)**.** A copy of the data replacing 999 with missing values

```
data array_yes;
  set array;
  array a[*] AGE HEIGHT WEIGHT VAR1-VAR4; /* array based on existing vars*/
  array b[*] NEW_AGE NEW_HEIGHT NEW_WEIGHT NEW_VAR1-NEW_VAR4; /* array based on
  do i=1 to dim(a);
    if a[i] = 999 then b[i]=.;
    else b[i]=a[i];
  end;
  run;
```

Example 3.1.3 (LOCF)**.** To implement the algorithm:

- If a patient doesn't have the baseline value (If VAR1 is missing), he/she is excluded from the analysis.
- If a patient doesn't even have at least one post-baseline value, he/she is excluded from the analysis.
- If an observation is missing, the one from the previous period is "carried forward"

```
data LOCF (drop=AGE HEIGHT WEIGHT i);
set array_yes;
array a[*] VAR1-VAR4;
array b[*] LOCF_VAR1-LOCF_VAR4;  /* new group of variable used*/

/* line below means baseline available and at least 1 non missing value post */
if cmiss(VAR2, VAR3, VAR4)<3 and a[1] ne . then do;
  b[1]=a[1];                                /* keep the same baseline value in
  do i=2 to dim(a);
    if a[i] ne . then b[i]=a[i];            /* if the value isn't missing it's
    else b[i]=b[i-1];                       /* if it's missing take the previo
  end;
end;
run;
```

3.2 Macro

3.2.1 Macro variables

Definition 3.2.1 (Macro variables)**.** Are a way to substitute a string (with both characters and numbers) in several places and can refer to simplify referring to the same variable, dataset or string

Important remark 36 (Macro definition and usage)*.* To define we use `%let` statement (outside other steps) as follows:

```
% let name=value;
```

In order to recall a macro variable the symbol ampersand & has to precede the macro variable's name:

```
&macro-variable-name
```

When the program is executed, SAS resolves the reference and it substitutes the macro variable with the assigned value.

Example 3.2.1 (Examples of macro definition)**.**

```
%let Age=38;
%let BMI=72/1.65*1.65;
%let Status=Normal;       /* should be suitable for a variable*/
%let Status1="Normal";    /* double quotes are kept*/
%let Status2=   Normal   ;    /* all spaces are kept*/
%let Status3=Normal, High;
```

Example 3.2.2 (Full example of definition/usage)**.** We want to print all subjects contained in the HEART dataset with "Normal" blood pressure and add an appropriate title. In this case, we have to repeat the string "Normal" two times.

```
    proc print data=sashelp.heart;
      where BP_Status="Normal";
      title "Subjects with Normal BP Status";
    run;
```

To have more flexible code

```
    %let status=Normal;
    proc print data=sashelp.heart;
      where BP_Status= "&status" ; /* use double quotes here*/
      title "Subjects with &status BP Status";
    run;
```

In this way we avoid repeating ourself and changing to "High" the reports (instead of normal) is quicker/safer (modify just one point)

Important remark 37 (Printing set macro)*.* Use `%put` statement as follows

TODO: non chiarissimo le virgolette e il punto a termine vedere il delwiche

```
%put &macro-variable;
```

To print special stuff we have:

```
%put _ALL_;         /* list value of all macro variables*/
%put _AUTOMATIC_;   /* "       "   "   automatic variables */
%put _USER_;        /* "       "   "   user variables*/
```

Example 3.2.3. eg

```
%put &age;
%put &status;
%put &subject;
```

3.2.2 Macro programs

Important remark 38. Macro programs are useful when we need to repeat the same DATA or PROC step for different variables. Can be taught as C functions but should be a simple code replacement, a-la C preprocessor

Important remark 39. The benefit in using macros is that code lines are reduced and changes to the code are facilitated: it will be sufficient to update the changes only inside the macro

Remark 4. In order to use a macro program 3 step are needed

1. define the macro program
2. compile it
3. call it

Definition 3.2.2 (Macro definition)**.** To define a macro without arguments/parameters

```
%MACRO macro-name;     /*starts with the statement %MACRO ... */
code                   /* here data step, proc steps, sas statements, macro variab
%MEND <macro-name>;  /* ..ends with the statement %MEND */
```

For a macro with parameters

```
%MACRO macro-name(keyword = default, ..., keyword = default);
code
%MEND <macro-name>;
```

SAS automatically creates a macro variable for each parameter: `keyword` will be the name of the macro variable and `default` the value that SAS assignes if none is specified during the macro call; if value is missing, it will be null.

Example 3.2.4 (Example without arguments)**.** Macro programs can also be defined without input variables. We can use them to call a code that is always equal to itself.
For example, if at the beginning of each program we want to delete all datasets from the work library to start programming in a clean space, we can define the PROC DATASET inside a macro, and then call it at the beginning of each program

```
%macro cleanwork;
proc datasets library=work kill nolist memtype=DATA;
run; quit();
% mend cleanwork;
```

Example 3.2.5 (Example with arguments)**.** A first example

```
%macro sort(group_var=, ds_sorted=);

proc sort data=data.employees
  out=&ds_sorted;
  by &group_var;
run;

proc print data= &ds_sorted;
  var Employee_ID &group_var;
run;

%mend sort;
```

Definition 3.2.3 (Macro compiling)**.** When the macro program's code defined in the previous step is executed, SAS saves the macro (such as `%sort`), that can be used whenever during the SAS session.

Definition 3.2.4 (Macro call)**.** In order to recall a macro program the syntax is the following

```
%macro-name;  /*without parameters */
%macro-name(keyword = value, ..., keyword = value); /* with parameters*/
```

Example 3.2.6 (Macro call)**.** Using the macro from the previous examples

```
%cleanwork;
%sort(group_var=Name,      ds_sorted=employees_1);  /* sorting producing different output d
%sort(group_var=Job_Title, ds_sorted=employees_2);  /* .. based on different .. */
%sort(group_var=Salary,    ds_sorted=employees_3);  /* .. sorting variables */
```

Example 3.2.7 (Exercise 1 lesson 6)**.**
1. Create a libref named Data that points to the folder `lesson_VI/data`
2. Copy all dataset from the Data library to the work library with the most efficient method

```
/*exercise 1*/
/*point 1*/
libname data '...\Lesson_6\Data';

/*point 2*/
%macro copy_data (dataset=);
        data work.&dataset.;
        set data.&dataset.;
        run;
%mend copy_data ;
```

```
%copy_data (dataset=demog);
%copy_data (dataset=diapef);
%copy_data (dataset=diapefh);
%copy_data (dataset=distat);
%copy_data (dataset=header);
%copy_data (dataset=medhis);
%copy_data (dataset=smoke);
%copy_data (dataset=trial);
```

Example 3.2.8 (Exercise 2 lesson 6)**.**

1. Create a libref named Data that points to the folder `lesson_VI/data`.
2. The dataset `data.orders_midyear` contains the monthly orders (in dollars) per client of mid year. The sales manager would like to apply a 5% discount on next year prices. He wants to know the impact of his choice supposing that the same quantity is sold. Create a data step in which:
   - An array called `month_s` of dimension 6 is created. It will contain the discounted values per month.
   - An array called mon containing original values of `month1-month6`.
   - Computes a DO-loop to find the discounted values.
   - Values of month1-month6 are summed and the result is saved in Tot;
   - Values of `month_s1-month_s6` are summed and results are saved in `Tot_s`;
   - The difference between values found in the previous two points is computed, and the result is saved in the variable Difference
3. Print the dataset created including only variables `Tot`, `Tot_s` and `Difference`, applying the format `dollar10.2`.

```
/*point 1*/
*libname data '...\Lesson_6\Data';


/* point 2 */
data discount_sales;
        set data.orders_midyear;
        array Month_s[6];
        array mon[*] month1-month6;
        drop i;
        do i=1 to 6;
                month_s[i]=mon[i]*0.95;
        end;
        Tot=sum(of month1-month6);
        Tot_s=sum(of month_s1-month_s6);
        Difference=tot-tot_s;
run;
```

```
/* point 3 */
proc print data=discount_sales noobs;
        var tot tot_s difference;
        format tot tot_s difference dollar10.2;
run;
```

Example 3.2.9 (Exercise 3 lesson 6)**.**

1. do previous point 1if not already done
2. Write a program that prints the information contained in the dataset `data.employee_payroll` about the employees that earn at least a minimun designated salary. Use a macro variable named minsal that contains the minimum salary and assign to it the initial value of 60000 (using the keyword `%let`).
3. Add also
   - The appropriate formats for dates and the format `dollar8.` for the `Salary` variable.
   - a title statement that uses the macro variable minsal in the title "Employees that earn 60000 or more"
4. modify the program so that to include only the employees that earn more than 10000 in the report.
5. modify the SAS code adding a macro program that uses the code written previously. Use this program to create different reports making the minimum salary vary.

```
/*point 1*/
*libname data '...\Lesson_6\Data';

/* point 2 */
%let minsal=60000;
title "Employees that earn &minsal or more";
proc print data=data.employee_payroll;
   where Salary >= &minsal;
   format Birth_Date Employee_Hire_Date
          Employee_Term_Date date9. Salary dollar8.;
run;
title;

/* point 3 */
%let minsal=10000;
title "Employees that earn &minsal or more";
proc print data=data.employee_payroll;
   where Salary >= &minsal;
   format Birth_Date Employee_Hire_Date
          Employee_Term_Date date9. Salary dollar8.;
run;
title;
```

```
/* point 4*/
%macro sal_rep(sal_min=);
title "Employees that earn &minsal or more";
proc print data=data.employee_payroll;
   where Salary >= &sal_min;
   format Birth_Date Employee_Hire_Date
          Employee_Term_Date date9. Salary dollar8.;
run;
title;
%mend;

%sal_rep(sal_min=20000);
%sal_rep(sal_min=30000);
%sal_rep(sal_min=70000);
```

Capitolo 4

Reporting

4.1 proc means

Definition 4.1.1 (`PROC MEANS`)**.** For *numeric variables*, it produces descriptive analysis (n, mean, STD, median, min, max, etc...) and univariate statistic tests (confidence intervals and t-tests) and has the following syntax

```
proc means data=input_dataset <options*> <noprint>;
  by variable(s);
  class variables;
  var variables;
  output out=output_dataset <statistics specifications>;
run;
```

where

- `by` calculates the statistics separately for each BY group (ex: age classes). Data *must be pre-sorted* for the by variables;
- `class` defines the variables that identify the analysis subgroups. The CLASS statement *does not require* the data to be sorted;
- `var` defines the variables that we want to analyze
- `output out=` creates an output dataset that contains the statistics specified in optional `statistics specifications`. usable functions are specified in table 4.1. If we dont require specific statistics as in the following we will have the default ones: `n`, `mean`, `sd`, `min`, `max`.

Example 4.1.1 (Basic example)**.** Here the results will be saved in the STAT dataset and they will also be printed in the output window.

```
proc means data=demog;     /* input dataset*/
  var wt;                   /* variable selection*/
  output out=stat;  /* output dataset; no statistics specification so default one*/
run;
```

To avoid printing results in the output window use the `noprint` option as in

keyword	meaning
`n`	Number of non-missing observartions
`mean`	mean
`min`	min
`max`	max
`sum`	sum of obs
`stderr`	standard error
`var`	var
`median`	median
`q1` or `p25`	first quartile
`q3` or `p75`	third quartile
`p5`, `p20`, `p80`	other quantiles
`T`	t-test where $H_0 : \mu = 0$
`probT`	p-value of t-test
`LCLM`	lower limit 95 confidence interval (ALPHA option to change 95)
`UCLM`	Upper limit 95 confidence interval (ALPHA option to change 95)

Tabella 4.1: proc means statistics specification

```
proc means data=demog noprint;
...
```

Example 4.1.2 (A more complex example with BY or CLASS)**.** Given the VS dataset, which contains data on the diastolic blood pressure at 4 different visits and the treatment assigned to each patient: we want to produce some descriptive statistics of these variables by treatment.

```
proc means data = vs noprint;
  BY tmt;                       /* CLASS tmt if we want an all-groups row added*/
  var diabp1-diabp4;
  output out = bydata   n = num1-num4
                     mean = m1-m4
                      std = sd1-sd4
                     lclm = inf1-inf4
                     uclm = sup1-sup4;
run;
```

`BY` if we want only treatment and control rows of statistics; replace BY with `CLASS` if we need to add a row of statistics for all the dataset.

4.2 proc freq

Definition 4.2.1 (`PROC FREQ`)**.** For *categorical variables*, `proc freq` produces descriptive analyses (frequency tables) and statistical tests (on relationship). It has the syntax

```
proc freq data=input_dataset <options> <noprint>;
  by variable(s);
  tables var1*var2 ... / <options>;
  output out= output_dataset;
run;
```

where

- `by` calculates the frequencies separately for each BY group
- `tables` defines the variables that we want in the frequency table
- `output out=` creates an output dataset that contains the statistics specified

Example 4.2.1 (One way table)**.** univariate of sex freqs (absolute, percent, cumulative)

```
proc freq data=demog;
  table sex;
run;
```

Example 4.2.2 (Two way table)**.** Bivariate with freq, overall perc, row perc, col perc.

```
proc freq data=demog;
  table sex*tmt;
run;
```

same stuff but in long format (eg an R `expand.grid`), with the `list` option

```
proc freq data=demog;
  table sex*tmt / list;
run;
```

The `norow`, `nocol`, and `nopercent` options deletes respectively row, columns and overall percentages. Eg with `norow`

```
proc freq data=demog;
  table sex*tmt / norow;
run;
```

If used together, the NOROW NOCOL and NOPERCENT options produce an output with absolute frequencies only:

```
proc freq data=demog;
  table sex*tmt / norow nocol nopercent;
run;
```

Example 4.2.3 (Missing handling)**.** The `MISSING` and `MISSPRINT` options consider in the frequency table the information regarding missing data:

- with `MISSPRINT` the missing data are shown, but the percentages and the totals are computed using only non missing data
- Using `MISSING`, totals and percentages are computed using also missing data

```
proc freq data=demog;
  table sex*tmt / missing; /* otherwise missprint here*/
run;
```

Example 4.2.4 (out and outpct)**.** OUT= option is useful when we want to save the frequency table into a SAS dataset. The OUTPCT option adds the row and column percentages to the output dataset.

```
proc freq data=demog;
  table sex*tmt / out=frequency outpct ;
run;
```

Example 4.2.5 (chisq)**.** in the following code chi square test will be added and both `frequency` and `test` dataset will be created. such a spaghetti code

```
proc freq data=demog;
  table sex*tmt / missing out=frequency outpct chisq;
  output out=test chisq;
run;
```

Example 4.2.6 (Exercise 1 lesson 7)**.** Starting from the data Patients.xls calculate the mean of the difference between the first and the last visit for SBP, of the patients treated with the drug A and the ones who are treated with the drug B. Calculate the following statistics: n mean, std, min, max by group without all/globals row

```
libname data '...\Lesson_VII\Data';

/*Exercise 1*/
PROC IMPORT OUT=t1
            DATAFILE= "...\Lesson_VII\Data\patients.xls"
            DBMS=XLS REPLACE;
     SHEET=Foglio1;
     GETNAMES=YES;
RUN;

proc sort data=t1 out=t2;
  by id;
run;

data t3;
 set t2;
 by id;
 where sbp ne .;
 if first.id or last.id then output;
run;
proc transpose data=t3 out=t4;
by id tmt;
var sbp ;
run;

/* Alternative method
data t5;
 merge t4(in=si) t1(where=(visita=1));
 by id;
 if si=1;
```

```
run; */

data t5;
set t4;
diff= col2-col1;
run;

proc means data=t5;
 class tmt;
 var diff;
 output out=t6 mean=m T=test probT=pvalue;
run;
```

Example 4.2.7 (Exercise 2 lesson 7)**.** Starting from the data Patients.xls calculate the frequency of the patients with SBP $<$ 140, between 140 and 159, between 160 and 179 and $>=$180 of patients treated with the drug A and patients treated with the drug B for each visit.

```
data pt_freq;
        set t1;
        if sbp=. then sbp_cat=.;
        else if sbp < 140 then sbp_cat=1;
        else if 140 =< sbp < 160 then sbp_cat=2;
        else if 160=< sbp < 180 then sbp_cat=3;
        else sbp_cat=4;
run;

proc sort data=pt_freq;
  by visita;
run;

proc freq data=pt_freq;
  tables sbp_cat*tmt;
  by visita;
run;
```

Example 4.2.8 (Exercise 3 lesson 7)**.**

1. Create a libref named data that points to the folder where the exercise is stored (path: `lesson_7/data`).
2. From the dataset data.employees compute the minimum, maximum and medium salary:
   (a) Considering all employees in the company
   (b) Dividing by job role
   (c) Dividing by gender and job role
3. We want to rank the most paid job roles: do again point b. and save it in a dataset named Salary.

```
/*Exercise 3*/
/*point a*/
libname data "...\Lesson_VII\Data";
```

```
/*point b*/
/*1)*/
proc means data=data.employees min max mean;
        var salary;
run;

/*2)*/
proc means data=data.employees min max mean;
        var salary;
        class job_title ;
run;

/*3)*/
proc means data=data.employees min max mean;
        var salary;
        class gender job_title ;
run;

/*point c*/
proc means data=data.employees min max mean noprint;
        var salary;
        class job_title ;
        output out=salary         min=minimum_salary
                                                max=maximum_salary
                                                mean=medium_salary;
run;

proc sort data=salary;
        by descending medium_salary;
run;
```

Example 4.2.9 (Exercise 4 lesson 7)**.** 1. Do point 1 of Exercise 1 if not done yet.

2. From the dataset data.employees, count employees for each job role. Use a procedure that allow to visualize this frequency distribution.

3. Make a report to display the roles of employees in descending order of frequency. In order to do this, save the output of the previous point into the work.jobr dataset, run a proc sort.

4. Create a two-way table to display how many women and men are there for each job role (without percentages).

```
/*Exercise 4*/
/*Point a*/
*libname data "...\Lesson_VII\Data";

/*Point b*/
proc freq data=data.employees;
```

```
        tables job_title  ;
run;

/*Point c*/
proc freq data=data.employees noprint;
        tables job_title / out=jobr ;
run;

proc sort data=jobr;
        by descending PERCENT;
run;

/*Point d*/
proc freq data=data.employees ;
        tables job_title*gender /nopercent norow nocol ;
run;
```

4.3 ods language

Important remark 40. It allows to save the results of a procedure in a SAS dataset (sas7bdat or in work) or in other common format (RTF, DOC, PDF, HTML).

4.3.1 Output to sas dataset

Important remark 41 (To a sas dataset). In order to move the procedure's results to a SAS dataset, you should follow two steps

1. Identify which output objects are generated when the procedure of interest is run: we use `ODS TRACE` statement before the procedure of interest, so SAS shows in the LOG window the keyword that identifies the output object produced, together with the path and other useful information.
2. Save the results contained in the output object in a SAS dataset: we use the name highlighted by ods trace to create a sas dataset that contains the same results

Example 4.3.1. With the following code SAS writes in the LOG window the keyword (e.g. `Name: CrossTabFreqs`) that identifies the output object produced

```
ods trace on / listing;
proc freq data=data.fitness_new;
  tables gym_code*gender;
run;
ods trace off;
```

afger that we save the results contained in the output object in a SAS dataset

```
ods output CrossTabFreqs=frq;
/* below the same code as above*/
proc freq data=data.fitness_new;
```

```
  tables gym_code*gender;
run;
```

SAS creates a dataset named `frq` that contains all these information

Example 4.3.2 (multiple outputs handling)**.** A SAS procedure usually produces more than one output object. EG the proc contents produces 3 objects: `Attributes` (general properties of the dataset), `EngineHost` (information about the engin/host) and `Variables` (properties of the variables).
With the following code we create 3 datasets (`attr`, `enh`, `var`) with that content

```
ods output Attributes=attr EngineHost=enh Variables=var;
proc contents data=data.fitness_new;
run;
```

Example 4.3.3 (avoiding results in output window)**.** If you don't want SAS to print the procedure's results in the output window, add ODS LISTING CLOSE before recalling the first ODS and ODS LISTING at the end (in this way the output window is reactivated)

```
ods listing close;   /* Closes the output window */
ods output CrossTabFreqs=frq;
proc freq data=data.fitness_new;
table gym_code*gender / out=freq outpct;
run;
ods listing;  /* reopens the output window*/
```

Tip: do not use NOPRINT option with ODS, otherwise it will not be possible to produce the dataset requested with the ODS.

4.3.2 Output to RTF or PDF

```
ods rtf file=‘‘D:\corsoSAS\freq.rtf";  /* or ods pdf file=‘‘D:\corsoSAS\freq.pdf"; */
proc freq data=data.fitness_new ;
  table gym_code*gender;
run;
ods rtf close;
```

Important remark 42. However the default report are verbose and usually, the ODS RTF or PDF or HTML are used with printing procedures, such as PROC PRINT or PROC REPORT following REPORT

1. A dataset SAS is created from the procedure of interest
2. A report via PROC PRINT or PROC REPORT is created

Example 4.3.4.

```
/* step 1*/
proc freq data=data.fitness_new;
  table gym_code*gender / out=freq;
run;

/* step 2*/
ods html close;
ods rtf file="D:\SAScourse\freq1.rtf";

title "Characteristics of athletes";
footnote;
options nodate nonumber;

proc print data=freq noobs label;
  var gym_code gender percent count;
  label gym_code='Gym Code' gender = "Athletes' gender"
        frequency='Absolute frequencies' percent='(%)';
run;

ods rtf close;
ods html;
```

4.4 proc report for reports

Important remark 43. PROC report combines together characteristics of the proc means, proc freq and proc print, allowing to

- produce reports (with a wider range of options compared to PROC PRINT); it's our focus here
- calculate frequencies, means and other descriptive statistics

Important remark 44 (proc report syntax)*.* we have

```
proc report data=dataset <options> ;
  column var1 var2 var3;
  define var1 \ varoptions;
  define var2 \ varoptions;
  define var3 \ varoptions;
run;
```

with:

- `column` declares the variables that have to be included in the report, in the order in which they are written
- `define` assigns information regarding the variables format
- as `options` we can have
  - `nocenter` prints the report aligning it to the left instead of putting it in the center
  - `split=` defines the character use for wrapping the text (default /)
  - `headline` draws a line under the column titles
  - `headskip` puts a blank line under the title
  - and others ...
- as `varoptions` we can have

  - `display`: shows all values for the specified variable
  - `order`: specifies the column used to sort the report

Example 4.4.1 (Exercise 1 lesson 8)**.**

1. Create a libref named "data" that points to the datasets stored in the folder `Lesson_VIII`.
2. Run a proc report of `data.employees` dataset to display variables: `Country`, `City`, `Employee_ID`, `Name`, `Job_title`, `Department`, `Gender` and `Salary`. The first rows of the output are shown below (see slides

```
/*point 1*/
libname data '...\Lesson_VIII\Data';

/*Point 2 */
ods pdf file="...\Lesson_VIII\Orders_july\orders.pdf";

proc report data=data.employees split='£';
        column country city Employee_ID Name ;
        define country / order 'Country' STYLE(column)=[cellwidth=1.5 CM;
    define city / order 'City';
        define Employee_ID / display 'ID';
        define Name / display 'Surname, Name';

        compute before _page_ / style={just=left} ;
                line "Employees' information";
        endcomp;

        compute after _page_ / style={just=left};
                line "";
                line "Name and Surname of the employees";
        endcomp;
run;
ods pdf close;
```

Example 4.4.2 (Exercise 2 lesson 8)**.**

1. Define a libref named data that points to the folder where the dataset for the exercise is stored
2. Create a folder named `Orders_July` in your desktop. Print the dataset `data.mnth7_2011` in one Word file and one PDF file, and save them in `Orders_July`. Print the title: "July 2011 orders".
3. Save in `Orders_July` two more files (both Word and PDF) that contain the output of two procedures operating on the dataset data.emplyees:
   - A proc freq that computes the number of employees for each department. Add also the title: «Number of employees per departement»
   - A proc print that displays name, city and gender of employees divided by department. Add the title: «Employees grouped by department»

```
/*Point 1*/
*libname data "..\Lesson_VIII\Data";
```

```
/*Point 2 */
ods html close;
ods pdf file="...\Lesson_VIII\Orders_july\orders.pdf";
ods rtf file="...\Lesson_VIII\Orders_july\orders.rtf";
title 'Orders July 2011';
proc print data=data.mnth7_2011;
run;
title;
ods pdf close;
ods rtf close;
ods html;

/*Point 3*/
ods html close;
ods pdf file="...\Lesson_VIII\Orders_july\employees.pdf";
ods rtf file="...\Lesson_VIII\Orders_july\employees.rtf";

title 'Number of employees per department';
proc freq data=data.employees;
        table Department;
run;

proc sort data= data.employees out=emp_sort;
        by department;
run;

title 'Employees grouped by department';
proc print data=emp_sort;
        by department;
        var Name City Gender;
run;

title;

ods pdf close;
ods rtf close;
ods html;
```

Example 4.4.3 (Exercise 3 lesson 8)**.**

1. Do point 1 of the previous exercise if not done yet (otherwise do not repeat it)
2. Using ODS Trace, try to understand which datasets can be created starting from the following proc freq:

   ```
   proc freq data=data.employees;
   table department;
   run;
   ```

3. Using the information at point 2, save the output of that procedure in a dataset named frq

```
/*Point 1*/
*libname data "...\Lesson_VIII\Data";

/*Point 2 */
ods trace on /listing;
proc freq data=data.employees;
        table Department;
run;
ods trace off;


ods noresults;
ods html close;
ods output OneWayFreqs=frq;
proc freq data=data.employees;
table department;
run;
ods html;
```

Example 4.4.4 (Exercise 4 lesson 8)**.** 1. Do point 1 of the previous exercise if not done yet (otherwise do not repeat it)

2. Create a Word file that contains the report created in exercise 1 and modify it as follows:

   - Delete the frame from all cells and add a line under the columns' titles.
   - For the columns' headers, define a blue background, Calibri font, bold characters and width=9
   - For the columns, choose the lightblue background, Calibri font, bold characters and width=8.
   - Change the columns style by fixing a width and justifying right or centered, as you wish
   - Choose Calibri font for the title, bold characters and size=12.
   - For the notes, define Calibri font, italic style and size=8.

```
/*Point 1*/
*libname data "...\Lesson_VIII\Data";

/*Point 2 */
ods html close;
ods rtf file="\\ARNAS01\Cros_NT\Cristina_Poletti\UNIBO 2019\ECON\Lezione_VIII\Orders_
title;footnote;
options nodate nonumber;

proc report data=dati.employees split='£'
        style(report)=[rules=groups frame=void]
        style(header)=[background=blue font_face=calibri font_weight=bold  font_size=
        style(column)=[background=lightblue font_face=calibri font_size=8pt ];
        column country city Employee_ID Name ;
        define country / order width= 5 'Country'  STYLE(column)=[cellwidth=1.5 CM just=left  ]
    define city / order 'City' STYLE(column)=[cellwidth=1.5 CM just=left ];
        define Employee_ID / display 'ID' STYLE(column)=[cellwidth=1.5 CM just=center ];
        define Name / display 'Surname, Name' STYLE(column)=[cellwidth=2.5 CM just=left ];

        compute before _page_ /style={font_face=calibri font_size=12pt font_weight=bold};
                line "Employees'information";
        endcomp;

        compute after _page_ / style={font_face=calibri font_size=8pt font_style=italic};
                line "";
                line "Name and Surname of the employees";
        endcomp;
run;


ods rtf close;
ods html;
```