SAS I & II - Synthesis

Lionel Panneel - Master in Data Science, Statistics orientation

5 mai 2021

Table des matières

1 SAS I : Essentials

1.1 Essentials

1.2 Accessing data

All columns must have a name, a type (numeric, character) and a length

1.2.1 General report

```
PROC CONTENTS DATA = dataset;
RUN;
```

1.2.2 Accessing data through libraries

Create library :
```
LIBNAME libref engine "path";
```
engine = type of data

Use library :
libref.table_name

Delete library reference :
```
LIBNAME libref CLEAR;
```

Forces table and columns names to follow SAS naming conventions :
```
OPTIONS VALIDVARNAME = V7;
```

1.2.3 Importing data into SAS

```
PROC IMPORT DATAFILE : "path/file_name.extension" DBMS = identifier
    OUT = output_table <REPLACE>;
    <GUESSINGROWS = n|max >
    <SHEET = sheet_name >
RUN;
```

REPLACE : replaces the table if already exists
GUESSINGROWS : specifies the number of rows used to determine columns type and length (default = 20)
SHEET : For Excel imported table

1.3 Exploring and validating data

1.3.1 Exploring data

Print the n first rows :
```
PROC PRINT DATA = input_data <(OBS = n)>;
RUN;
```

OBS : used to specify the last observation to be read

Give standard statistics (Variable, label, N, Mean, Std, min, max) :
```
PROC MEANS DATA = input_table ;
    VAR col_name(s);
RUN;
```

Give more precise statistics of specified columns :
```
PROC UNIVARIATE DATA = input_table ;
    VAR col_name(s) ;
```

```
RUN;
```

Gives contigency table

```
PROC FREQ DATA = input_table ;
    TABLES col_names ;
RUN;
```

1.3.2 Filtering rows

```
PROC procedure_name;
    WHERE expression ;
RUN;
```

expression is defined as : column operator value
where operator is :
— = / EQ
— ^ / NE
— > / GT
— < / / LT
— >= / GE
— <= / LE

Specifying SAS dates :
— "ddmmmyyyy"d
date
— "hh :mm< :ss>"t
time
— "DDMMMYYYY hh :mm :ss"dt
datetime

More on expressions :
— WHERE expression1 AND expression2
— WHERE expression1 OR expression2
— WHERE col_name <NOT> IN (val1 val2 ... valn);
— WHERE col_name = ANY(val1 val2 ... valn);
— WHERE col IS <NOT> MISSING;
— WHERE col BETWEEN val1 AND val2 ;
val1 and val2 are included
— WHERE col LIKE "value";
% : any number of character
_ : a single value

1.3.3 Formatting columns

```
PROC PRINT DATA = input_table;
    FORMAT col(s) format ;
RUN;
```

format = <$> format_name <w>.<d>
where $ indicates a character format, <w> gives the total width of the formatted value, d indicates the number of decimals for numeric format

1.3.4 Sortind data and removing duplicates

Sorting data :

```
PROC SORT DATA = input_table <OUT = output_table >;
    BY <DESCENDING> col(s) ;
RUN;
```

By default, ASCENDING

Identifying and removing duplicate rows :

```
PROC SORT DATA = input_table <OUT = output_table>;
    NODUPRECS <DUPOUT = output_table >;
    BY _ALL
RUN;
```

BY_ALL ensures that the duplicate rows are adjacent

Identifying and removing duplicate key values

```
PROC SORT DATA = input_table <OUT = output_table>
    NODUPKEY <DUPOUT = output_table >;
    BY <DESCENDING> col(s) ;
RUN;
```

NODUPKEY keeps only the first occurence of each unique value of the BY variable

1.4 Preparing data

1.4.1 Reading and filtering data

Create a data table

```
DATA output_table ;
    SET input_table;
RUN;
```

Filtering rows in the DATA step :

```
DATA output_table;
    SET input_table;
    WHERE expression ;
RUN;
```

Subsetting columns in the DATA step :

```
DATA output_table;
    SET input_table;
    KEEP col1 col2 ... ;
    DROP col1 col2 ...;
RUN;
```

Formatting columns in the DATA step :

```
DATA output_table;
   SET input_table;
   FORMAT col format;
RUN;
```

1.4.2 Computing new columns

```
DATA output_table;
    SET input_table;
    new_col = expression ;
RUN;
```

Use the functions :

```
DATA output_table;
    SET input_table;
    new_col = function(arguments);
```

RUN ;

More on numeric functions
— SUM(num1, num2...)
— MEAN(num1, num2...)
— MEDIAN(num1, num2...)
— RANGE(num1, num2...)
— MIN(num1, num2...)
— MAX(num1, num2...)
— N(num1, num2...)
— NMISS(num1, num2...)
All those functions ignore the missing values

More on character functions :
— UPCASE(char)
— LOWCASE(char)
— PROPCASE(char, <delimiters>)
By default, the delimiters are blank, forward, slash, hyphen, open parenthesis, period and tab
— CATS(char1 , char2 ...)
Concatenates character string and removes leading and trailing blanks from each argument
— SUBSTR(char, position , <length>)
Returns a substring from a character string. By default, 'length' = 1

More on date functions
— MONTH(SAS_date)
— YEAR(SAS_date)
— DAY(SAS_date)
— WEEKDAY(SAS_date)
— QTR(SAS_date)
Returns a number from 1 to 4 representing the quarter
— TODAY ()
— MDY(month , day, year)
Returns a SAS date value from month day and year values
— YRDIF(startdate, enddate, 'AGE')
Calculates a precise difference in years between two dates

1.4.3 conditional processing

```
DATA output_table ;
    SET input_table ;
    IF expression THEN statement ;
    <ELSE IF expression2 THEN statement2 ;>
    ELSE statement ;
RUN ;
```

Attention : the first mention of a column in the DATA step defines the name, type and ***length***

Defining a specific length :

```
DATA output_table ;
    SET input_table ;
    LENGTH col $ n ;
    IF expression THEN statement ;
RUN ;
```

Explicitly defines the length of the column 'col'

Processing multiple statements :

```
IF expression THEN DO ;
```

```
    <executable statements>
END;
<ELSE IF expression2 THEN DO;
    <executable statements2>
END; >
ELSE DO;
    <executable statements>
END;
```

1.5 Analyzing and reporting on data

1.5.1 Enhancing reports with titles, footnotes and labels

Defining titles :
TITLE<n> "title_text";

Defining footnotes :
FOOTNOTE<n> "footnote_text";

Clearing title :
TITLE;

Clearing footnote :
FOOTNOTE;

turns off procedures titles :
ODS NOPROCTITLE;

Disables titles like "PROC MEANS", etc...

Applying temporary labels to columns

```
PROC ...;
    LABEL col = "label";
RUN;
```

Note : if the labels are defined in the DATA step, they become permanent, but not in the PROC step

1.5.2 Creating frequency reports

```
PROC FREQ DATA = input_table <options>;
    TABLES col * col </options> ;
RUN;
```

** : creates a crossed table*

PROC FREQ options :
— NOPRINT

does not print the table

Tables options :
— NOROW
— NOCOL
— NOPERCENT
— CROSSLIST
— LIST
— OUT = output_table

1.5.3 Creating summary statistics reports

```
PROC MEANS DATA = input_table <stat-list> <options>;
    VAR col(s);
    CLASS col(s) ;
    WAYS n ;
RUN;
```

CLASS specifies columns to group the data before calculating statistics
WAYS specifies the number of ways to make unique combinations of class columns

Creating output summary table

```
PROC ...;
    OUTPUT OUT = output_table <statistic = col>;
RUN;
```

1.6 Exporting results

1.6.1 Exporting data

```
PROC EXPORT DATA = input_table OUTFILE = "output_file"
    DBMS = identifier> <REPLACE>
RUN;
```

1.6.2 Exporting reports

To a CSV file :

```
ODS CSVALL FILE = " filename.csv ";
    SAS code that produces the output
ODS CSVALL CLOSE;
```

To an Excel file :

```
ODS EXCEL FILE = " filename.xlsx " STYLE = style
    OPTIONS(SHEET_NAME = 'label');
    SAS code that produces the output
ODS EXCEL CLOSE;
```

To an PowerPoint file :

```
ODS POWERPOINT FILE = " filename.pptx " STYLE = style ;
    SAS code that produces the output
ODS POWERPOINT CLOSE;
```

To a rtf file :

```
ODS RTF FILE = " filename.rtf " STARTPAGE= NO;
    SAS code that produces the output
ODS RTF CLOSE;
```

To a pdf file :

```
ODS PDF FILE = " filename.pdf " STYLE = style STARTPAGE=NO
    PDFTOC=n; ODS PROCLABEL "label";
    SAS code that produces the output
ODS PDF CLOSE;
```

2 SAS I :data manipulation techniques

2.1 Controlling DATA step processing

2.1.1 Understanding DATA step processing

```
DATA output_table;
    SET input_table;
    PUTLOG "message";
    PUTLOG_ALL_
RUN;
```

PUTLOG " message" : allows to put a text string to the log
PUTLOG_ALL_ : write all columns and values in the PDV to the log
PUTLOG column = col(s) : write the selected column(s) and values in the PDV to the log

2.1.2 Directing DATA step output

Create explicit output

```
DATA output_table;
    SET input_table;
    OUTPUT;
RUN;
```

Sending output to multiple tables

```
DATA output_table1 output_table2;
    SET input_table;
    IF expression THEN OUTPUT output_table1;
    ELSE OUTPUT output_table2;
RUN;
```

Controlling column output

```
DATA output_table1 (DROP = col(s)) output_table2 (KEEP= col(s));
    SET input_table;
RUN;
```

Using DROP and KEEP statements does not write those columns in the PDV. Note that "_ :" selects every columns

2.2 Summarizing data

2.2.1 Creating an accumulating column

```
DATA output_table1 output_table2;
    SET input_table;
    RETAIN col <initial_value> ;
    col = col + 1;
RUN;
```

RETAIN retains the value each time that the PDV reinitialize and assigns an initial value. This is equivalent to :

```
DATA output_table1 output_table2;
    SET input_table;
    col + 1;
RUN;
```

2.2.2 Processing data in groups

Only *if the data is sorted*

```
DATA output_table;
    SET sorted_input_table;
    BY <DESCENDING> col(s) ;
RUN;
```

Creates automatically a First.by_col and Last.by_col in the PDV (can be used in a WHERE statement for instance). If there are multiple by_cols, a first. and last. variables are created for each column

Subsetting rows in execution
— With a WHERE statement
— With a IF statement

```
DATA output_table;
 SET sorted_input_table;
 IF expression;
RUN;
```

SAS continues processing for this row only if the IF statement is true

2.3 Manipulating data with functions

2.3.1 Understanding SAS functions and CALL routines

Specifying column lists
— col_1-col_n : specifies all columns from col_1 to col_n inclusive
— col1 – coln : specifies all columns from col1 to coln inclusive
— x-NUMERIC-a : specifies only numeric columns from col1 to coln inclusive
— x-CHARACTER-a : specifies only character columns from col1 to coln inclusive
— x : : specifies all columns that start by "x"
— _ALL_ : specifies all columns
— _NUMERIC_ : specifies all numeric columns
— _CHARACTER_ : specifies all character columns

The CALL routines

```
DATA output_table;
    SET sorted_input_table;
    CALL function (OF cols_selection);
RUN;
```

CALL routines alter the values or perform system actions

2.3.2 Using numeric and date fuctions

See above for basic functions.

More numeric/date functions :
— RAND('distribution', parameter1, ..., parameterk)
Gives a random number
— LARGEST(k , val1<, val2...>)
Gives the k largest values from a set of values
— SMALLEST(k , val1<, val2...>)
— ROUND(number <, rounding_unit>)
— CEIL(number)
Returns the smallest integer that is greater than or equal to the number

— FLOOR(number)
Returns the largest integer that is less than or equal to the number
— INT(number)
Returns the integer value
— DATEPART(datetime_value)
Returns the date (DDMMMYYYY) from a datetime (HH :SS DDMMMYYYY)
— TIMEPART(datetime_value)
Returns the TIME (HH :SS) from a datetime (HH :SS DDMMMYYYY)
— INTCK('interval', start_date, end_date <, 'method'>)
Counts the number of date or time intervals between two events. Possible intervals includes WEEK, MONTH, YEAR, WEEKDAY or HOUR. Possible methods are : discrete (each interval has a fixex boundary : it counts the number of times the boundary is passed) or continuous(it counts the number of times the entire date/time period is passed)
— INTNX('interval', start, increment <, 'alignment'>)
Shifts a date by a certain interval. Alignment can be : 'BEGINNING', 'MIDDLE', 'END' or 'SAME'

2.3.3 Using character functions

See above for basic functions

More character functions :

— COMPBL(string)
*returns a character value with all **mutliple** blanks in the string converted to **single** blanks*
— COMPRESS(string <, characters>)
returns a character value with specified characters removed from the string
— STRIP(string)
returns a character value with leading and trailing blanks removed from the string
— SCAN(string, n <, 'delimiters '>)
*Retrieve the word to extract from a string at the rank n, defined by a specific delimiter. By default, the delimiters are blank ! % $ & ( ) * + - , . / ; < | ^*
— FIND(string, substring <, 'modifiers'>)
Returns a number that represents the first character position where substring is found in string. The modifiers are 'I' = case Insensitive and 'T' = Trim leading and trailing blanks from string and substring
— FINDC(string, substring <, 'modifiers'>)
Returns a number that represents the first character position where character is found in string.
— FINDW(string, substring <, 'modifiers'>)
Returns a number that represents the first word position where substring is found in string.
— LENGTH(string)
Returns the length of a non-blank character string, excluding trailing blanks. Returns 1 for a completely blank string
— ANYDIGIT(string)
Returns the first position at which any digit is found in the string
— ANYALPHA(string)
Returns the first position at which any alpha character is found in the string
— ANYPUNCT(string)
Returns the first position at which any punctuation character is found in the string
— LENGTHC(string)
Returns the length of a string, including trailing blanks
— LENGTHN(string)
Returns the length of a string, excluding trailing blanks
— ANYLOWER(string)
Returns the position of the first lowercase character
— ANYUPPER(string)
Returns the position of the first uppercase character
— ANYSPACE(string)

Returns the position of the first whitespace character

— TRANWRD(source, target, replacement)
Replace, in a specific column (source) a string to find (target) by another string (replacement)

— TRANSLATE(source, to_1, from_1)
Searches a string for any character in a list of characters

— TRANSTRN(source_expression, target_expression, replacement_expression)
Replace or removes all occurrences of a substring in a character string

— CAT(string_1, ..., string_n)
Concatenates strings together, does not remove leading or trailing blanks

— CATS(string_1, ..., string_n)
Concatenates strings together, removes leading or trailing blanks

— CATX('delimiter', string_1, ..., string_n)
Concatenates strings together, remove leading or trailing blanks and insert a delimiter between each string

2.3.4 Using special functions to convert column type

— INPUT(source, informat)
Converts character values to numeric values, using a specific informat

— PUT(source, format)
Converts numeric or character values, using a specific format

What is informat ?
The informat specifies how the character value looks so that it can be converted to a numeric value.

— RENAME=(old_col_name=new_col_name)
Is used as an option of a table

2.4 Creating custom formats

2.4.1 Creating and using custom format

```
PROC FORMAT;
    VALUE format_name value_or_range1 = 'formatted_value1'
        value_or_range2 = 'formatted_value2'
        ...;
RUN;
```

The format name uses $ if this is a format applied on a character variable. When creating the format, we don't use the period (.)

Define ranges :
val1 operator val2
Use "val1 -< val2" to exclude the ending value.
Use "val1 - val2" to define a range inclusive.
Use "val1 <- val2" to exclude the starting value.
Use "low" or "high" to define the lowest/highest value.
Use "other" for all other value that is not yet defined

2.4.2 Creating custom formats from tables

```
PROC FORMAT CNTLIN = input_table FMTLIB;
    SELECT format_names ;
RUN;
```

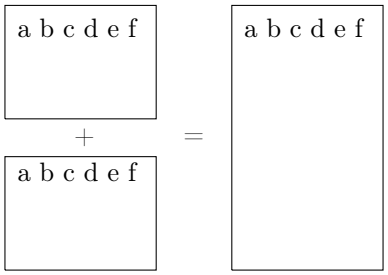
CNTLIN option specify a table for building a format. FMTLIB creates a report containing information about the custom formats. The input_table must have at least these columns :

FmtName	Start	End

Creating permanent formats
PROC FORMAT LIBRARY = lib<.table> ;
By default, the formats are stored in the WORK library.

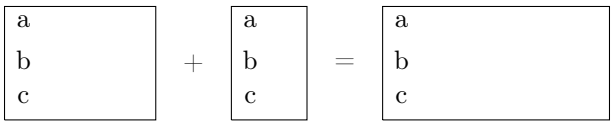
2.5 Combining tables

2.5.1 Concatenating tables




```
DATA output_table ;
    SET input_table1 input_table2 ;
RUN;
```

2.5.2 Merging tables




```
DATA output_table ;
    MERGE input_table1 input_table2 ;
    BY in_common_column(s) ;
RUN;
```

*The input tables **must** be sorted*

2.5.3 Identifying matching and nonmatching rows

Merging tables with nonmatching rows :

```
DATA output_table;
    MERGE input_table1 (IN = variable)
        input_table2 (IN = variable )... ;
    BY in_common_column(s) ;
RUN;
```

The IN= data set option can be used to identify matching and nonmatching rows. Writes 0 in the PDV if the BY value is not in the corresponding input table and 1 otherwise. It can then be used with a IF statement (not a WHERE because it is only written in the PDV)

2.6 Processing repetitive code

2.6.1 Using iterative DO loops

```
DO index_column = start TO stop <BY increment> ;
    ... repetitive code ...
END;
```

The defaults increment is 1. Note that the explicit OUTPUT can be usefull

2.6.2 Using conditional DO loops

```
DO UNTIL ( expression ) ;
    ... repetitive code ...
END ;
```

Executes repetitively until a condition is true. Checks the condition ***at the bottom*** *of the loop. Hence it* ***always executes at least once****.*

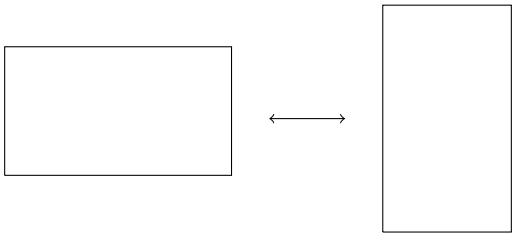
```
DO WHILE( expression ) ;
    ... repetitive code ...
END ;
```

Executes repetitively while a condition is true. Checks the condition ***at the top*** *of the loop*

Combining iterative and conditional DO loops :

```
DO index_column = start TO stop <BY increment> UNTIL | WHILE ( expression ) ;
```

2.7 Restructuring tables



2.7.1 Restructuring data with the DATA step

2.7.2 Restructuring data with the TRANSPOSE procedure

```
PROC TRANSPOSE DATA = input_table <OUT = output_table> <PREFIX = column> <NAME
=column> ;
    <ID col ;>
    <VAR col(s) ;>
RUN ;
```

PREFIX : provides a prefix for each value of the ID column in the output table. NAME option names the column that identifies the source column containing the transposed values. VAR lists the columns to be transposed

Transposing values within groups :

```
PROC TRANSPOSE DATA = input_table <OUT = output_table> ;
    <ID col ;>
    <VAR col(s) ;>
    <BY col(s) ;>
RUN ;
```

Each unique combination of BY values creates one row in the output table

3 SAS SQL 1 : Essentials

3.1 Essentials

3.1.1 Introduction to the SQL procedure

```
PROC SQL <options> ;
    SELECT col1, ...,coln
    FROM input_table
```

<WHERE clause>
<GROUP BY clause>
<HAVING clause>
<ORDER BY clause>
QUIT;

Viewing columns attributes :
PROC SQL;
DESCRIBE TABLE table ;
QUIT;

Limit the number of observations :
— FROM input_table (OBS = n);
OBS specifies the last observation that SAS processes in a data set
— FROM input_table (INOBS = n);
INOBS limits rows from each source table
— FROM input_table (OUTOBS = n);
OUTOBS restricts the number of rows that is returned
Controlling display
PROC SQL NUMBER; *adds a new column "Row" representing the row number as first column*

3.2 PROC SQL fundamentals

3.2.1 Generating simple reports

Filtering using the WHERE clause :
PROC SQL <options>;
SELECT col1, ...,coln
FROM input_table
WHERE expression <AND | OR expression >;
QUIT;

Expression must be like : "col operator comparator", with operator being mnemonic symbols (see above).

Other operators :
— WHERE col <NOT> IN (val1, ..., valn);
— WHERE col IS <NOT> NULL;
— WHERE col <NOT> BETWEEN val1AND val2 ;
Can be used on both character and numeric variables
— WHERE col <NOT> LIKE "value";

Sorting the output using the ORDER BY clause
ORDER BY col <DESC> <, ..., coln>;
Sorts the missing values before the nonmissing values. Can also be sorted by column index. In this case, it is relative to the position of the columns in the SELECT clause

Enhancing reports :
— TITLE<n> "title_text";
— FOOTNOTE<n> "footnote_text";
— SELECT col_name < <LABEL = >"label"> <FORMAT = formatw.d>;
Adding LABEL = is not ANSI conventional (but proposed by SAS).

Creating a new column :
expression AS alias

Calculating a new column with any operator :
SELECT col1, ..., coln,
CASE

```
        WHEN cond1 operator comparator THEN "new_val"
        ...
        ELSE "last_new_val"
    END AS new_col
FROM input_table;
```

Calculating a new column only with equality :

```
SELECT col1, ..., coln,
    CASE var
        WHEN "cond1" THEN "new_value"
        ...
        ELSE "last_new_value"
    END AS  new_col
FROM input_table;
```

Subsetting on newly calculated column

```
PROC SQL;
    SELECT col1, ..., coln, function(parameters) AS alias
    FROM input_table
    WHERE CALCULATED alias operator comparator;
QUIT;
```

3.2.2 Summarizing and grouping data

Eliminate duplicate rows :

```
SELECT DISTINCT col1 <, ..., coln>
```

Summary fonctions :

```
SELECT summary_function(col1 <,..., coln> ) AS alias
```

Summary functions are AVG, COUNT, MIN, MAX, SUM, NMISS, STD, VAR. The three last one are specific to SAS, not SQL. Note that "" specifies all rows.*

Grouping data and filter grouped data :

```
PROC SQL;
    SELECT col1, ..., coln, summary_function(parameters) AS alias
    FROM input_table
    WHERE expression;
    GROUP BY col;
    HAVING expression;
QUIT;
```

3.2.3 Creating and managing tables

```
PROC SQL;
    CREATE TABLE table AS
        query;
QUIT;
```

Copy the structure of an existing table :

```
CREATE TABLE table
    LIKE existing_table;
```

Create table by defining columns :

```
CREATE TABLE table (
    col1 type(length) <FORMAT = format>,
    ...);
```

Inserting rows with a query :

```
PROC SQL;
    INSERT INTO table <(col(s))>
        query;
QUIT;
```

Inserting rows with the VALUES clause :

```
PROC SQL;
    INSERT INTO table <(col(s))>
        VALUES( val1,... ,valn);
QUIT;
```

Inserting rows with the SET clause :

```
PROC SQL;
    INSERT INTO table
        SET( col1= val1
        col2 =val2;
QUIT;
```

Dropping table :

```
PROC SQL;
    DROP TABLE table;
QUIT;
```

Alter table :

```
    ALTER TABLE table;
```

adds/drop columns, changes column attributes in an existing table

Update table :

```
    UPDATE table;
```

Modifies a column's value in existing rows

Delete rows :

```
    DELETE table;
```

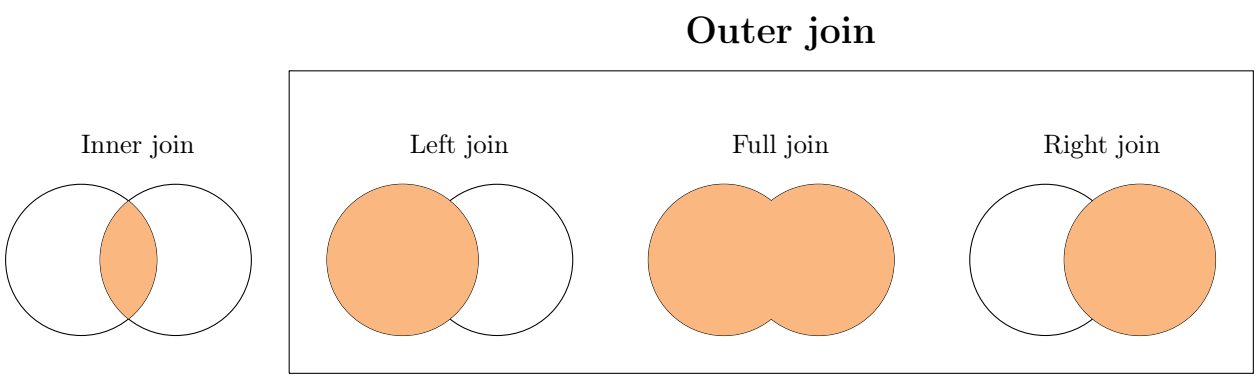
Removes row(s) from a table that is specified in the FROM clause

3.2.4 Using DICTIONARY tables

DICTIONARY.tables
DICTIONARY.columns
DICTIONARY.libnames

3.3 SQL joins

3.3.1 Introduction to SQL joins



Default join - Cartesian product :

```
PROC SQL;
    SELECT col1, ..., coln
    FROM table1, table2;
QUIT;
```

3.3.2 Inner joins

```
PROC SQL;
    SELECT col1, ..., coln
    FROM table1INNER JOIN table2
    ON table1.col = table2.col;
QUIT;
```

Alternative inner join :

```
SELECT col1, ..., coln
FROM table1, table2
WHERE table1.col = table2.col;
```

Using table aliases

```
SELECT t1.col, ..., t2.col
FROM table1AS t1, table2 AS t2
WHERE t1.colx = t2.colx;
```

Matching rows with a Natural join

```
FROM table1 AS t1 NATURAL JOIN table2 AS t2;
```

Natural join assumes that we want to base the join on all pairs of common columns

Feedback option

PROC SQL FEEDBACK
Expands a SELECT statement to the Log

Attention : PROC SQL treats missing values as matches for join. To avoid this type of join :

```
ON t1.col = t2.col AND t1.col IS NOT NULL;
```

3.3.3 Outer joins

```
PROC SQL;
    SELECT col1, ..., coln
    FROM table1 LEFT | RIGHT | FULL JOIN table2
    ON table1.col = table2.col;
QUIT;
```

COALESCE function :

```
SELECT COALESCE (t1.colID, t2.colID) AS colID
```

The COALESCE function returns the value of the first nonmissing argument

3.3.4 Complex joins

Reflexive join :

```
FROM table1 AS t1 JOIN_type table1 AS t2
```

Usefull for joining one table by itself

3.4 Subqueries

3.4.1 Subquery in WHERE and HAVING clauses

```
PROC SQL;
    SELECT col1, ..., coln
    FROM table
    WHERE col operator (
        SELECT .. );
QUIT;
```

```
PROC SQL;
    SELECT col1, ..., coln
    FROM table
    GROUP BY col
    HAVING col operator (
        SELECT .. );
QUIT;
```

3.4.2 In-line views (Query in the FROM clause)

```
FROM (
    SELECT col FROM ...)
```
ORDER BY cannot be used in an in-line view

CREATE VIEW statement :

```
PROC SQL;
CREATE VIEW libname.view AS
SELECT ...;
QUIT;
```

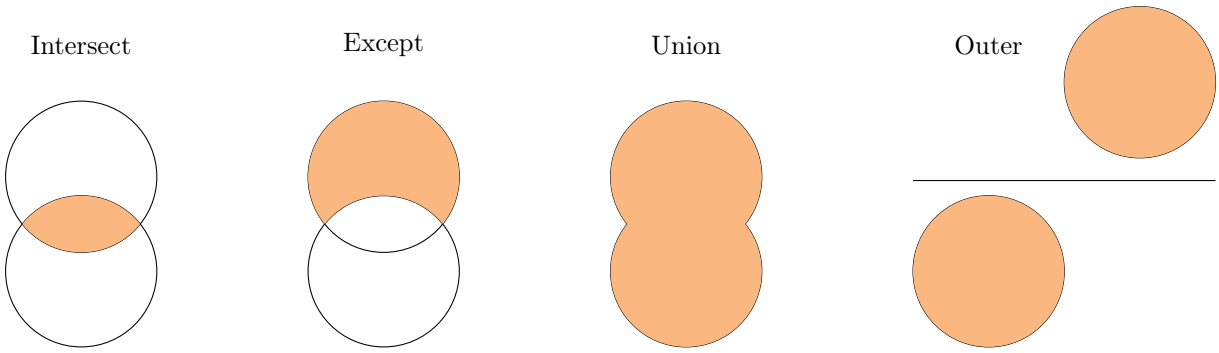
The query is stored as a permanent view in the library but does not create a new database. It has the advantage that it will retrieve the most current data as it will execute the query. Needs to be in the same library as the contributive database

3.4.3 Subquery in the SELECT clause

```
SELECT col, (SELECT ...;)
```
Can return a single value

3.5 Set operators

3.5.1 Introduction to set operators



Columns are aligned by position

3.5.2 INTERSECT, EXCEPT and UNION

```
PROC SQL;
    SELECT query
INTERCEPT | EXCEPT | UNION <ALL> <CORR>
    SELECT query ;
QUIT;
```

ALL displays all the rows, even in duplicate. CORR looks for a correspondance in the name of the variables

3.5.3 OUTER UNION

```
PROC SQL;
    SELECT query
OUTER UNION <CORR>
    SELECT query ;
QUIT;
```

4 SAS II : Macro language 1 : Essentials

4.1 The SAS macro facility

4.1.1 Program flow and tokenization

Word scanner :
pulls text from the input stack. Transforms it into tokens. Sends tokens for processing. The blanks separates the tokens and a token ends when another start.

Types of tokens :

— Name
— Number
— Special
*Special token have special meaning for SAS. Exemple : * / + - ; ( ) . & %*
— Literal
literal token is a string of character, bounded by single/double quotes
When a % sign is encountered, the tokens are put in the Macro Pocessor and when a ";" is encountered, the Macro Processor executes and the Macro is put in the Global Symbol Table.

When a & sign is encountered, the Macro Processor searches in the Global Symbol Table to the associated macro value and write it in the Word Scanner.
Text enclosed by double quotes is tokenized by the Word Scanner and can then trigger the Macro Processor. This is not the case with single quotes.

4.2 Creating and using macro variables

Create a macro :
%LET macro = value ;

Important : Everything written after the "=" symbol is encoded as text by default, except if there are special tokens. It removes leading and trailing blanks.

Write a macro in the log :
%PUT &macro;
%PUT &=macro;

Attention, without blanks between the "&" sign, the "=" sign and the macro name

Automatic macro variables :
— SYSDATE9
Date the SAS session was initiated
— SYSDAY
Day of the week the SAS session was initiated
— SYSERR
Returns 0 if the step runs without error. Returns a number otherwise

— SYSTIME
Time the SAS session was initiated
— SYSHOSTNAME
Name of the machine where the SAS session is executing
— SYSLAST
Name of the last data set created
— SYSUSERID
User ID under which the SAS session is executing
— SYSSCP
Operating system abbreviated name
— SYSSCPL
Operating system detail
— SYSVLONG
SAS version and maintenance release

OPTIONS SYMBOLGEN;
Provides information in the log when a macro variable references resolves

Delimiting macro variable reference :
If we want to change a little bit the value of the macro. For instance, put a "s" at the end of the value.
&macro.s

To write a real dot after a macro value :
&macro. .

Write a simple quote around a macro :
%TSLIT(&macro)

4.3 Storing and processing text

4.3.1 Macro statement and functions

Deleting macro variables :
%SYMDEL macro ;

Macro functions :
— %UPCASE(&macro)
— %QUPCASE(&macro)
Same as UPCASE but encloses the result as quoted text.
— %SUBSTR(&macro, position,<length>)
— %QSUBSTR(&macro, position,<length>) *Same as SUBSTR but encloses the result as quoted text.*
— %SCAN(&macro, n <, charlist <, modifiers> >)
Search for a word that is specified by its position in a string
— %QSCAN(&macro, n <, charlist <, modifiers> >)
Same as SCAN but encloses the result as quoted text.
— %EVAL(math_expression)
executes the mathematical expression and truncates fractional results
— %SYSEVALF(math_expression <, conversion_type>)
executes the mathematical expression and returns float by default. Conversion_ type can be CEIL, FLOOR, INTEGER, BOOLEAN
— %SYSFUNC(data_step_function (argument(s)) <, format>)
Executes a data step function in a macro.
— %QSYSFUNC(data_step_function (argument(s)) <, format>)
Same as SYSFUNC but encloses the result as quoted text.
Note that nested macro functions are allowed

4.3.2 Working with special characters

Inserting special character as text :
— %STR(string)
Treat each special character in the string as plain text, except % and &.
— %NRSTR(string)
Treat each special character in the string as plain text, including % and &.
— *Use the % before a special character to make it clear that it is a plain text.*

— %SUPERQ(macro)
Don't need the & before the macro name. Masks all special characters and mnemonic operators
— %BQUOTE(text)
Masks special characters and mnemonic operators at execution, except & and %.

4.3.3 Using SQL to create macro variables

```
PROC SQL ;
    SELECT item1 <, ..., itemn>
    INTO : macro1 <, ..., : macron
    FROM ...
QUIT ;
```
Assigns the first item to macro1, etc. No need of the & symbol before the macro name. Also write in the Global Symbol Table SQLOBS defining the number of observation given by the SQL procedure

Don't print the SQL report :
PROC SQL NOPRINT ;

Write items in multiple macros, having the same prefix :
INTO :macroX- :macroY

Write items in multiple macros, having the same prefix but not knowing the exact number of macros :
INTO :macroX-

Creates only one macro but separates the values by a delimiter :
INTO :macro SEPARATED BY 'delimiter'

Remove the leading and trailing blanks in a INTO : procedure :
INTO : macro TRIMMED

4.3.4 Using DATA step to create macro variables

```
DATA _NULL_;
    CALL SYMPUTX('macro_variable', 'value'<, scope>);
RUN;
```

If the single quotes are not used, then every different values of the macro_variable are used and related with the corresponding values in the Global Symbol Table. The _NULL_ statement permitallows to use DATA step without creating an output table

4.3.5 Looking table and indirect referencing

Indirect referencing of macro variables :
&&first_part_macro&second_part_macro
Permits to create a macro name by the concatenation of multiple macros. The && signs will translate in a single & symbol, and the other single & signs will be transformed in their macro values.

4.4 Working with macro programs

4.4.1 Defining and calling a macro

Write a macro program :

```
%MACRO macro_program_name ;
    program;
%MEND <macro_name>;
```

Macro options :

```
<OPTIONS <MCOMPILENOTE=ALL> <MPRINT><MLOGIC>;>
%MACRO macro_program_name;
    program;
%MEND <macro_name>;
<OPTIONS MCOMPILENOTE=NONE;>
```

When MCOMPILENOTE is set to ALL, a note is written in the log upon successful macro compilation. Set it on NONE to restore default behaviour. The MPRINT option prints the executed code in the log. The MLOGIC is used for nested macros. It tests if the parameters for the nested macro are correctly validated and reports which ones are validated.

Execute a macro call/macro program :
%macro_program_name
Not &macro. Without semicolon, like a macro function. When encountering a % symbol, the Macro Processor searches in the work.sasmacr table and writes the registered code in the Input Stack. It is also possible to use macros (&macro) inside the macro program

Macro parameters :

```
%MACRO macro_program_name (par1, par2);
    program;
```

%MEND <macro_name>;

Using a macro with parameters :
%MACRO_program_name (par1, par2)

Write a macro in the Global Symbol Table (without assigning a value) :
%GLOBALmacro_name;
This permits to change the scope of macro written in a macro program for instance

Write a macro in the Global Symbol Table (and assigning a value to it) :
%GLOBAL / READONLYmacro_name = value;
Macros created with READONLY cannot be modified or deleted during the SAS session

Local Symbol Table : Created when a macro with parameters is called or when a local macro variable is created. It is deleted when the macro stops execution. When a macro already exists in the GST, SAS will rewrite its value, except if we specify that it has to be written in the LST. Can only be written inside a macro program.
%LOCAL var1 var2;

Display specific macro values :
— %PUT _ALL_;
— %PUT _AUTOMATIC_;
— %PUT _USER_;
Writes in the log the macro values edited by the user in the scope the _USER_ is located
— %PUT _LOCAL_;
Writes in the log the macro values edited by the user in the scope the _LOCAL_ is located

4.4.2 Conditional processing

Conditional processing of text :
%IF expression %THEN action1;
%ELSE action2;

Conditional multiple processing of text :
%IF expression %THEN %DO;
statement1;statement2;
%END;
%ELSE %DO;
statement3;statement4;
%END;

Write specific messages in the log :
— ERROR : error_text;
Written in red
— NOTE : note_text;
Written in blue
— WARNING : warning_text ;
Written in green

Conditional compound in macros :
Multiple conditions can be defined in the %IF — %THEN %DO statement. For that, mnemonics and operators can be used, as with other SAS steps.

Specifications for the IN mnemonic (# operator) :
%MACRO macro(par1) / MINOPERATOR;
%IF expression IN val1 val2 valn

```
    %THEN %PUT action1;
%ELSE action2;
```

No need of parenthesis around the list of values, neither commas. IN default delimiter is a space but can be specified with the / MINDELIMITER = 'delimiter' option

Specifications for the NOT mnemonic :
%IF NOT (expression) %THEN %DO

Specifications for multiple inequalities :
%IF val1 operator val2 AND val2 operator val3 %THEN %DO

4.4.3 Iterative processing

```
%DO index = start action1%TO stop < %BY increment>;
    code ;
%END;
```

```
%DO %WHILE ( condition < );
    code ;
%END;
```

Executed at the top of the WHILE loop

```
%DO %UNTIL( condition < );
    code ;
%END;
```

Executed at the bottom of the UNTIL loop

4.5 Advanced macro techniques

4.5.1 Building your own macro functions

Add a description to the macro program :
%MACRO macro_name \DES 'description';

Open the catalog with all macros :

```
PROC CATALOG CATALOG = work.sasmacr;
CONTENTS OUT = MACROLIST;
RUN;
PROC PRINT;
RUN;
```

To store a macro in a sustainable way : Store it in the Autocall Macro Facility. This is a folder in which each document corresponds to a specific defined macro that shares the same name as the name of the document. It can be only one macro in each document.

```
OPTIONS SASAUTOS = ("path/autocall", SASAUTOS);
PROC OPTIONS OPTION = SASAUTOS;
RUN;
```

Permits to combine our file (in the defined path location) and the already defined SASAUTOS.

4.5.2 User defined macro functions

Find the location of your own macro functions :

```
OPTIONS MAUTOCOMPLOC;
PROC OPTIONS OPTION = SASAUTOS;
```

4.5.3 User-defined macro utilities

A utility macro is a **generalized** SAS program that performs a usefull task

4.5.4 Data-driven macro calls

DOSUBL(text_SAS_code);
Executes immediately the text_SAS_code