Data Engineering Applied in SAS: Processing, Organization and Analysis

Keita, Moussa

May 2016

Ingénierie des Données Appliquée sous SAS : Traitement, Organisation et Analyses

Par

Moussa Keita, PhD*

Mai 2016

(Version 1)

Résumé

La conduite de toute étude statistique, qu'elle soit descriptive, exploratoire, explicative ou prédictive passe d'abord par une phase de traitement, d'organisation et de structuration des données. Cependant de nos jours avec l'émergence du BIGDATA, la mise en œuvre pratique de ces opérations devient de plus en plus complexe en raison de la multiplicité et de la diversité des sources de données parfois non homogènes. En se basant sur le logiciel SAS, ce manuscrit propose une description des différentes étapes de l'ingénierie des données allant de la phase de collecte jusqu'à la phase d'analyse. Ce travail est organisé en quatre chapitres. Le premier chapitre est consacré à la description des opérations de structuration et de préparation des données sous SAS en utilisant les outils de la DATA STEP. Le deuxième chapitre est un prolongement du premier en introduisant l'utilisation du langage SQL dans la phase d'organisation des données. Quant au troisième chapitre, il vise à décrire l'usage des macro-variables et des macro-programmes dans la mise en œuvre des tâches complexes et répétitives. Et enfin le quatrième chapitre présente une description détaillée des différentes méthodologies d'analyse de données. Dans chacun de ces chapitres, toutes les notions abordées sont étayées par des exemples d'applications concrets permettant à l'utilisateur de choisir parmi l'ensemble des méthodes proposées celle qui répond le mieux à ses préoccupations. Toutefois, le manuscrit étant toujours en cours de progrès, nous restons ouverts à toutes les critiques et suggestions de nature à améliorer son contenu.

*Ecole d'Economie, Université d'Auvergne Clermont Ferrand 1 ;
Ecole Nationale Supérieure de Statistique et d'Economie Appliquée ENSEA-C.I

Contact info: Email : keitam09@ymail.com

Codes JEL: C8

Mots clés: Traitement de données, Programmation SAS, Analyses de données.

Table des matières

CHAPITRE I : LES OPERATIONS DE STRUCTURATION ET DE PREPARATION DES DONNEES

I.1. Lecture et enregistrement des données

Chaque jeu de données sous SAS se présente sous la forme d'une table à deux dimensions dans laquelle les colonnes représentent les variables et les lignes les observations. Les colonnes sont identifiées par des noms de variables. Ces noms variable doit commencer par une chaîne de caractère. Deux possibilités se présentent à l'utilisateur lors de la lecture des données : la saisie directe ou la lecture à partir d'un fichier externe. Dans le premier cas, on utilise la commande INPUT et dans le second on peut utiliser soit la commande INFILE soit la procédure PROC (IMPORT, DBF, etc..). L'objet de cette section est de donner quelques détails sur les méthodes de lecture de données sous SAS.

I.1.1. Lecture des données par saisie directe

I.1.1.1. Saisie

Lire les données par saisie directe c'est indiquer les valeurs des variables pour chaque ligne d'observation dans l'éditeur de programme. Cette saisie se fait avec l'instruction INPUT. Cette instruction est accompagnée par la clause DATALINES (ou CARDS) qui indique à SAS la fin de la déclaration de l'étape DATA et de début de la saisie sur la ligne suivante. Notez que les lignes de données ne se terminent pas par un point-virgule. Et la fin de la saisie ne termine pas non plus obligatoirement par un point-virgule. Le point-virgule à la fin de la saisie est simplement une habitude de bonne pratique. Mais il n'est pas nécessaire dans une étape INPUT.

L'exemple ci-dessous crée une table de données nommée MYDATA avec dix observations et sept variables. Les variables sont : ID, SEXE, AGE, REV, R1, R2 et R3 où REV désigne le revenu annuel en milliers, R1 à R3 sont des scores traduisant le degré d'appréciation de l'individu par rapport à trois produits particuliers.

```
DATA MYDATA;
  INPUT ID SEXE $ AGE REV R1 R2 R3 ;
  DATALINES;
 1  F  35 17  7 2 2
 17  M  50 14  5 5 3
 33  F  45  6  7 2 7
 49  M  24 14  7 5 7
 65  F  52  9  4 7 7
 81  M  44 11  7 7 7
 2   F  34 17  6 5 3
 18  M  40 14  7 5 2
 34  F  47  6  6 5 6
 50  M  35 17  5 7 5
 ;
```

Pour afficher les données dans la fenêtre des résultats, on peut utiliser une PROC PRINT comme suit :

```
PROC PRINT DATA=MYDATA; RUN;
```

Au lieu de la déclaration DATALINES on peut également utiliser l'instruction CARDS. Les deux sont équivalents.

Dans l'exemple ci-dessus, la déclaration de la variable SEXE variable dans l'instruction INPUT est suivie par un signe dollar ($) indiquant que celle-ci est une variable caractère. Sans instructions spécifiques, SAS suppose que toutes les variables sont numériques. Dans ce cas, si une valeur caractère est rencontrée pour un individu, alors SAS lui assigner une valeur manquante (.). Il est donc important de spécifier le type d'une variable lors de la saisie (numérique, caractère, date,…). Nous reviendrons sur la définition des types des variables un peu plus loin.

I.1.1.2. Définition de la longueur des champs lors de la saisie

Attention, lors de la saisie des données, car SAS considère que les valeurs des variables sont séparées par des espaces (blank). Mais il arrive qu'un enregistrement soit de longueur différente des autres enregistrements et qu'il existe un blank dans un enregistrement. Par exemple, en saisissant la ville New York, si aucune précaution n'est prise, SAS va séparer des deux mots en deux variables. Pour éviter une telle situation, on spécifie la longueur des variables (voir exemple ci-dessous).

```
DATA MYDATA;
 INPUT NOM $1-20  AGE VILLE $29-37 ETAT $5-15 ;
 DATALINES;
Oliver Schabenberger    33  Lansing    Michigan
John T. Smith        37     New York   New York
Barack Hussein Obama 54 Honolulu Hawaï
 ;
 RUN;
```

$ 1-20 A la suite de la variable nom, indique que nom est une variable de caractères dont les entrées se trouvent dans les colonnes 1-20 des datalines. Etant donné que des blank délimitent automatiquement les variables dans datalines, le fait que les deux âges ne sont pas alignés n'a aucun effet. Ils seront lus correctement sans pointer SAS vers une colonne spécifique de la ligne de données. La ville est alors lue de nouveau en pointant SAS vers des colonnes spécifiques.

I.1.1.3. Indicateur de fin d'enregistrement lors de la saisie : @@

Dans les exemples présentés ci-dessus, on constate que chaque enregistrement est situé sur une ligne spécifique. Cette manière d'écrire risque de prendre beaucoup d'espace lorsque le nombre d'observations est très élevé et que et le nombre de variables est très faible. Ainsi, il serait judicieux d'écrire plusieurs enregistrements sur une même ligne afin d'économiser de l'espace. Pour cela, on utilise l'identificateur d'enregistrement @@ dans la commande INPUT afin d'indiquer la fin d'un enregistrement et le début d'un autre (voir exemple ci-dessous).

```
DATA MYDATA;
  INPUT ID SEXE $ AGE REV R1 R2 R3 @@ ;
  DATALINES /* ou CARDS */;
1  F  35 17  7 2 2 17  M  50 14  5 5 3 33  F  45  6  7 2 7 49  M  24 14  7 5 7 65  F 52  9
4 7 7 81  M  44 11  7 7 7 2   F  34 17  6 5 3 18  M  40 14  7 5 2 34  F  47  6  6 5 6 50  M
35 17  5 7 5
;
```

Avec cette spécification, SAS passe en revue par chaque ligne de données afin de remplir les variables ID SEXE AGE REV R1 R2 R3 à leur tour. Sans le symbole @@, SAS passe à la ligne suivante, dès que la dernière variable a été remplie. Avec le symbole @@, SAS continue de remplir les variables jusqu'à ce qu'il atteigne la fin de la ligne.

I.1.2. Lecture des données à partir d'un fichier externe

Pour lire les données à partir d'un fichier externe, on peut utiliser soit la commande INFILE ou la procédure PROC IMPORT ou PROC DBF.

I.1.2.1. Utilisation de la commande INFILE

La commande INFILE est utilisée lorsque les données sont déjà saisies dans un fichier externe et qu'il faut importer sous SAS afin de créer une table. La commande INFILE est généralement associée à deux autres commandes FILENAME et INPUT. La commande FILENAME indique le nom du fichier à importer et la commande INPUT indique les noms et les types des variables. La structure générale des étapes de lecture est la suivante :

```
FILENAME  "Chemin +Nom de fichier données externe" ;
INFILE "Librairie. Nom table SAS" ;
INPUT "Liste des variables et leur type " ;
RUN ;
```

Les exemples ci-dessous illustrent l'utilisation de la commande INFILE.

I.1.2.2. Lecture d'un fichier de données avec séparateur simple (blank)

L'exemple ci-dessous importe les données à partir d'un fichier simple fichier texte dont les séparateurs sont des espaces (blank).

```
FILENAME MYFOLDER 'C:\données.txt';
DATA MYDATA;
  INFILE MYFOLDER;
  INPUT ID SEXE $ AGE REV R1 R2 R3;
RUN;
```

La ligne de commande FILENAME attribue un nom simple au fichier de données externe. Attention, le nom attribué ne doit pas être supérieur à 8 caractères. Ici le fichier de données est nommé MYFOLDER. Ensuite, on appelle ce fichier avec la commande INFILE. Les variables et leur type sont spécifiés dans la commande INPUT.

Notons aussi que la commande INFILE n'est pas nécessaire lors de la lecture des données. Elle sert juste à raccourcir le nom du fichier de données. Par exemple, on pouvait simplement écrire :

```
DATA MYDATA;
 INFILE 'C:\données.txt';
 INPUT ID SEXE $ AGE REV R1 R2 R3;
RUN;
```

I.1.2.3. Lecture d'un fichier de données avec autre séparateur

Très souvent, les enregistrements dans les fichiers de données externes sont séparées par des symboles spécifiques : virgules, point-virgule, tabulation, etc…). Dans chacun de ces cas, en utilisant la commande INFILE, il faut absolument indiquer le type de séparateur.

I.1.2.3.1. Lecture des fichiers de données séparées par une virgule ",“

```
FILENAME MYFOLDER 'C:\données.txt';
DATA MYDATA;
 INFILE MYFOLDER DELIMITER=',' DSD ;
 INPUT ID SEXE $ AGE REV R1 R2 R3;
RUN;
```

On peut abréger l'option DELIMITER par DLM (ils sont synonymes).
En indiquant le type de délimiteur, on associe généralement le mot clé DSD afin d'indiquer à SAS que quand il rencontre le symbole de séparateur entre griffes ou guillemets, il doit le traiter comme une donnée et non un séparateur. En effet il arrive que le symbole qui a été indiqué comme séparateur (par exemple ;) se retrouve dans les données. Si on n'indique pas cela à SAS, il va considérer tous les mots à part et les séparer comme des valeurs distinctes. L'option DSD permet d'éviter ce genre de situation. Il faut noter que lorsque l'option DSD est spécifiée et SAS rencontre deux délimiteurs consécutifs, il les traite comme une valeur manquante sur la variable correspondante.

I.1.2.3.2. Lecture des fichiers de données séparées par un point-virgule ";“

```
FILENAME MYFOLDER 'C:\données.txt';
DATA MYDATA;
 INFILE MYFOLDER DLM=';' DSD ;
 INPUT ID SEXE $ AGE REV R1 R2 R3;
RUN;
```

I.1.2.3.3. Lecture des fichiers de données avec séparateur "csv“

```
FILENAME MYFOLDER 'C:\données.csv';
DATA MYDATA;
 INFILE MYFOLDER DLM='3B'x dsd ;
 INPUT ID SEXE $ AGE REV R1 R2 R3;
RUN;
```

I.1.2.3.4. Lecture des fichiers de données avec séparateur "|"

```
FILENAME MYFOLDER 'C:\données.txt';
DATA MYDATA;
 INFILE MYFOLDER DLMSTRING='|' DSD ;
 INPUT ID SEXE $ AGE REV R1 R2 R3;

RUN;
```

I.1.2.3.5. Lecture des fichiers de données avec un séparateur défini par un caractère particulier

```
FILENAME MYFOLDER 'C:\données.csv';
DATA MYDATA;
 INFILE MYFOLDER DLMSTRING='~\^' DSD ;
 INPUT ID SEXE $ AGE REV R1 R2 R3;

RUN;
```

L'option DLMSTRING indique à SAS que les données sont séparées par les trois caractères groupés ~\^ .

La commande INFILE permet également d'indiquer à partir de quelle observation, il faut commencer à lire les données mais aussi le nombre d'observation total à importer. Pour cela, on utilise les options FIRSTS et OBS.

```
FILENAME MYFOLDER 'C:\données.txt';
DATA MYDATA;
 INFILE MYFOLDER DLM=';' DSD FIRSTOBS=10 OBS=500;
 INPUT ID SEXE $ AGE REV R1 R2 R3;
RUN;
```

L'option FIRSTOBS indique à SAS de commencer à lire les données à partir de la dixième ligne et qu'il faut importer 500 observations.

I.1.2.4. Utilisation de la commande PROC IMPORT

On peut utiliser la commande PROC IMPORT à la place de la commande INFILE pour lire les données sous SAS. Les commandes ci-dessous fournissent des exemples d'utilisation.

I.1.2.4.1. Importer des données à partir d'EXCEL

```
PROC IMPORT OUT= mydata DATAFILE= "C:\donnees.xlsx" DBMS=EXCEL REPLACE;
RANGE="Feuil1$"; GETNAMES=YES;  RUN;
```

L'option RANGE indique le nom de la feuille EXCEL qui contient les données et l'option GETNAMES= YES indique à SAS que les noms des variables sont situées à l'entête des données.

I.1.2.4.2. Importer des données à partir d'un fichier texte avec séparateur Tabulation

```
PROC IMPORT OUT=mydata DATAFILE="C:\donnees.txt" dbms=tab replace;
GETNAMES=YES; RUN;
```

I.1.2.4.3. Importer des données à partir d'un fichier CSV avec séparateur virgule

```
PROC IMPORT OUT=mydata DATAFILE="C:\donnees.csv" dbms=csv replace;
getnames=yes; run;
```

I.1.2.4.4. Importer des données à partir d'un fichier CSV avec séparateur point-virgule

```
PROC IMPORT OUT= mydata DATAFILE= "C:\donnees.csv" DBMS=DLM REPLACE;
DELIMITER='3B'x; GETNAMES=YES; DATAROW=2; RUN;
```

I.1.2.4.5. Utilisation de proc DBF

Lorsque le fichier de données externe se présente sous le format dbf (database format), on peut directement utiliser la commande PROC DBF comme suit

```
FILENAME fichier 'C:\données.dbf';
PROC DBF DB4=fichier OUT=MYDATA;
RUN;
```

I.1.3. Définition d'une librairie SAS et enregistrement de la table de données

Une librairie SAS est un dossier de destination (temporaire ou permanent) dans lequel sont enregistrés les objets SAS (table, et…).

Par défaut lorsqu'on crée une table SAS, elle est enregistrée automatiquement dans une librairie temporaire. La librairie temporaire de SAS s'appelle WORK. Elle est située dans un des sous-dossiers daté du jour situé dans le dossier temporaire "C:\Users\NOM\AppData\Local\Temp\SAS Temporary Files". Ce dossier est une librairie temporaire dont les contenues sont supprimés à la fin de la session SAS.

Pour enregistrer de façon permanente une table, il faut indiquer à SAS une librairie permanente c'est-à-dire un dossier de destination à sas autre. La définition d'une librairie se fait avec la commande LIBNAME :

```
LIBNAME MYLIB "C:\TUTORIAL SAS" ;
```

Cette commande une librairie nommée MYLIB correspondant au dossier " TUTORIAL SAS" situé sur le disque C. Autant que nécessaire, toute les tables créées dans la session SAS doivent être renvoyées vers cette librairie. Pour cela, on précède le nom de la table par MYLIB.. Par exemple MYLIB.MYDATA fait référence à la table MYDATA située dans la librairie MYLIB.

L'exemple ci-dessous permet de saisir les données dans une table nommée MYDATA et enregistrée dans la librairie MYLIB.

```
LIBNAME MYLIB "C:\TUTORIAL SAS" ;

   DATA MYLIB.MYDATA;
INPUT ID SEXE $ AGE REV R1 R2 R3 @@ ;
CARDS;
1  F  35 17  7 2 2 17  M  50 14  5 5 3 33  F  45  6  7 2 7 49  M  24 14  7 5 7 65  F 52  9
4 7 7 81  M  44 11  7 7 7 2  F  34 17  6 5 3 18  M  40 14  7 5 2 34  F  47  6  6 5 6 50  M
35 17  5 7 5
   ;
```

Affichons les données saisies

```
PROC PRINT DATA=MYLIB.MYDATA; RUN;
```

I.1.4. Exporter une table de données SAS vers des formats de données externes

Après avoir traité et organisé les données sous SAS, il arrive qu'on veuille exporter les données sous un autre format afin de le rendre lisible par d'autres logiciels. Par exemple, on peut vouloir exporter les données sous format TEXT ou EXCEL. Les commandes ci-dessous donnent quelques exemples d'exportation de table SAS.

I.1.4.1. Exporter les données vers un format txt avec séparateur tabulation

```
PROC EXPORT DATA= MYLIB.MYDATA
OUTFILE= "C:\TUTORIAL SAS\MYDATA1.TXT"
DBMS=TAB REPLACE;
RUN;
```

I.1.4.2. Exporter les données vers un format CSV avec séparateur " ,"

```
PROC EXPORT DATA= MYLIB.MYDATA
OUTFILE= "C:\ TUTORIAL SAS\MYDATA2.CSV"
DBMS= CSV REPLACE;
RUN;
```

I.1.4.3. Exporter les données vers un format EXCEL

```
PROC EXPORT DATA= MYLIB.MYDATA
OUTFILE= "C:\TUTORIAL SAS\MYDATA3.XLSX"
DBMS= EXCEL REPLACE;
RUN;
```

I.2. Opération sur variables

Après la lecture des données et leur enregistrement dans une ou plusieurs tables, de nombreuses opérations doivent être effectuées avant de passer à la phase de l'analyse de données. Les principales opérations sont le traitement et la mise en forme des données pré-existantes, la création des nouvelles variables à partir des variables existantes et l'organisation de l'ensemble des variables de la base données finale. Dans cette section, nous passons en revue un certain nombre d'opérations couramment rencontrées dans cette phase de management des données.

I.2.1. Création d'une nouvelle variable ou modification d'une variable existante

Considérons les données définies comme suit :

```
LIBNAME MYLIB "C:\TUTORIAL SAS" ;

   DATA MYLIB.MYDATA;
     INPUT ID SEXE $ AGE REV R1 R2 R3 @@ ;
     CARDS;
1  F  35 17  7 2 2 17  M  50 14  5 5 3 33  F  45  6  7 2 7 49  M  24 14  7 5 7 65  F 52  9
4 7 7 81  M  44 11  7 7 7 2   F  34 17  6 5 3 18  M  40 14  7 5 2 34  F  47  6  6 5 6 50  M
35 17  5 7 5
   ;
```

A partir de cette table, générons trois nouvelles variables : HOMME qui prend 1 lorsque le sexe=M (masculin) et 0 sinon ; AGE2 égal à l'âge au carré et MR égal à la moyenne de R1, R2 et R3. On a :

```
DATA MYLIB.MYDATA; SET MYLIB.MYDATA;
IF sexe="M" THEN homme=1;
ELSE homme=0;
age2=age**2;
REV=REV*1000 ;
MR=mean(R1, R2, R3) /* ou mean(of R1 R2 R3)*/;
 RUN;
```

Dans cette étape DATA, on crée deux nouvelle variables : homme, age2 et MR. On modifie les valeurs initiales de REV en les multipliant par 1000.

Lors de la création ou modification de variables, on utilise généralement les opérateurs arithmétiques ou de comparaisons tels que présentés dans le tableau ci-dessous :

Les opérateurs usuels de SAS

Opérateurs	Définition	Exemple
Opérateurs arithmétiques		
+	addition	x+y
-	soustraction	x-y
*	multiplication	x*y
/	division	x/5
**		x**3
Opérateurs de comparaison		
= (synonyme: EQ)	égal à	x=y
^= (synonyme: NE)	différent de	x^=y
¬= (synonyme: NE)	différent de	x¬=y
~= (synonyme: NE)	différent de	x~=y
> (synonyme: GT)	supérieur à	x>y
< (synonyme: LT)	inférieur à	x<y
>= (synonyme: GE)	supérieur ou égal à	x>=y
<= (synonyme: LE)	inférieur ou égal à	x<=y
><(synonyme: MIN)	Minimum	X=(y><z)
<>(synonyme: MAX)	maximum	X=(y<>z)
IN	est parmi	x in (3, 4, 5)
Opérateurs logiques		
&(synonyme: and)	ET	(a>b & c>d)
|(synonyme: OR)	OU	(a>b or c>d)
!(synonyme: OR)	OU	(a>b or c>d)
¦(synonyme: OR)	OU	(a>b or c>d)
¬(synonyme: NOT)	NON	not(a>b)
ˆ(synonyme: NOT)	NON	not(a>b)
~(synonyme: NOT)	NON	not(a>b)

En plus de ces opérateurs usuels, on rentre d'autres opérateurs mathématiques se présentant sous forme de fonctions. Ex : mean () ; min(), max(), median(), etc…(se référer aux "Functions and CALL Routines" de la base de connaissance de SAS.

I.2.2. Recoder une variable

Ici, on recode les jours de la semaine de 1 , 2 ,3, 4, 5, 6, 7 e, 0, 1 , 2 ,3 , 4 , 5 , 6. Ainsi, on a :

```
DATA BASETEST;
SET BASE_CREE;
SELECT (jour_sem);
    WHEN (1) jour_sem=0;
    WHEN (2) jour_sem=1;
    WHEN (3) jour_sem=2;
    WHEN (4) jour_sem= 3;
        WHEN (5) jour_sem= 4;
    WHEN (6) jour_sem= 5;
    WHEN (7) jour_sem= 6;
    OTHERWISE ; /* aucun changement n'est fait */
    END;

RUN;
```

I.2.3. Combinaison des clauses SELECT, WHEN et IF dans la création de variables

L'exemple ci-dessous illustre la combinaison de la commande SLECT, WHEN et IF lors de la création d'une nouvelle variable (catégorie de taille nommée TAILLEC).

```
DATA MYDATA;
FORMAT taillec $10. ;
SET MYDATA;
SELECT (sexe);
/* Nb: variable sexe 1 =masculin 2 =feminin*/
WHEN (1)
IF taille>190 THEN taillec='GRAND';
ELSE IF taille >170 THEN taillec='MOYEN';
ELSE taillec='PETIT';
WHEN (2)
IF taille>180 THEN taillec='GRANDE';
ELSE IF taille >160 THEN taillec='MOYENNE';
ELSE taillec='PETITE';
END;
KEEP taille taillec sexe;
RUN;
```

Ici, on effectue un traitement différencié dans la définition de la catégorie de taille selon qu'il s'agisse d'un homme ou d'une femme. Cela est possible à travers la clause SELECT et WHEN.

I.2.4. Sélection ou suppressions de variables

Pour sélectionner une liste de variable, on utilise la commande KEEP et pour supprimer une liste de variables, on utiliser la commande DROP.

Soit la table de données constituée des variables ID SEXE AGE REV R1 R2 R3. Sélectionnons les quatre variables ID SEXE AGE REV et enregistrons dans la table MYDATA1. On a :

```
DATA MYDATA1; SET MYDATA;
KEEP ID SEXE AGE REV ;
RUN;
```

On peut aussi procéder par suppression des autres variables. Ainsi, on a :

```
DATA MYDATA1; SET MYDATA;
DROP R1 R2 R3;
RUN;
```

Ces deux exemples montrent que pour sélectionner une liste de variables d'intérêt, on peut procéder de deux manières : soit garder directement les variables dont on a besoin en utilisant KEEP ou bien supprimer les trois restantes en utilisant DROP. Le choix entre les deux approches dépend du nombre de variables à sélectionner par rapport au nombre de variable à supprimer. Si le nombre de variables à sélectionner est inférieur au nombre de variable restantes, on utilise KEEP. Dans le cas contraire, il vaut mieux utiliser DROP pour raccourcir la ligne de commande.

NB : Dans les exemples ci-dessus, nous avons spécifiée les commandes KEEP ou DROP à l'intérieur de l'étape. On pouvait aussi les spécifier à l'extérieur de l'étape. Ainsi, on a par exemples :

```
DATA MYDATA1 (KEEP=ID SEXE AGE REV); SET MYDATA;
RUN;
```

Ou

```
DATA MYDATA1 (DROP=R1 R2 R3); SET MYDATA;
RUN;
```

NB : Il ne faut pas oublier qu'on peut combiner dans la même étape DATA la sélection des observations et la sélections de variables. Ex :

```
DATA MYLIB.MYDATA1 (DROP=R1 R2 R3); SET MYLIB.MYDATA;
WHERE AGE>=40  AND  SEXE="M";
RUN;
```

I.2.5. Renommer une variable

Pour renommer un variable, on utilise la commande RENAME en spécifiant l'ancien nom= nouveau nom. La commande RENAME peut être spécifiée à l'intérieur de l'étape DATA ou à l'extérieur. Les commandes ci-dessous illustrent les deux méthodes de spécification de ma commande RENAME. Le but ici est de renommer les variable R1, R2 et R3 comme Score1 , Score2 et Score3. On a :

```
DATA MYDATA1 ; SET MYDATA;
 RENAME R1=Score  R2=Score2 R3=Score3 ;
RUN;
```

Ou encore :

```
DATA MYDATA1 (RENAME =R1=Score  R2=Score2 R3=Score3)) ; SET MYDATA; RUN;
```

I.2.6. Ranger les variables selon un ordre défini

Pour ordonner les variables dans une table SAS, on utilise la commande RETAIN. Par exemple, on veut modifier l'ordre des variables de ID SEXE AGE REV R1 R2 R3 à ID AGE SEXE R1 R2 R3 REV, on utilise la commande suivante :

```
DATA MYDATA; RETAIN ID AGE SEXE R1 R2 R3 REV; SET MYDATA;
RUN;
```

I.2.7. Attribuer des libellés aux noms des variables (variables labels)

Définir le libellé d'une variable c'est attribuer un libellé permettant décrire cette variable. Par exemple la variable ID représente l'identifiant de l'individu. Alors, on peut fournir cette description dans la table en utilisant la commande LABEL. Exemple :

```
DATA MYDATA1; SET MYDATA;
LABEL ID ="IDENTIFIANT INDIVIDU"
REV="REVENU DE L'INDIVIDU"
R1="SCORE 1"
```

```
R2="SCORE 2"
R3="SCORE 3"; RUN;
```

I.2.8. Attribuer des libellés aux valeurs des variables (values labels)

I.2.8.1. Définition standard des values labels

Attribuer un label aux valeurs d'une variable c'est définir des valeurs permettant de donner une description détaillée sur les valeurs de cette variable. Par exemple dans la table la variable SEXE a été définie par "M" et "F" pour désigner les Hommes et les femmes respectivement. On peut alors définir des labels pour M et F afin de les rendre lisible. Pour cela, on suit deux étapes. D'abord, on définit les values labels en utilisant la commande PROC FORMAT. Ensuite on attribue ces values labels à la variable en question. Exemple :

```
PROC FORMAT ;
VALUE  $sexecode 'H'='HOMME'   'F'='FEMME';
RUN;
*attribution des labels;
DATA MYDATA1; SET MYDATA;
FORMAT SEXE $sexecode.  ;
RUN;
```

Le signe $ qui intervient dans la définition du label signifie que les labels values porte sur une variable caractère et non une variable numérique. Pour définir les labels values d'une variable numérique, on ignore le signe $. L'exemple ci-dessous définit trois values labels correspondant à trois types de variables (numérique, caractère et date) et les attribuent aux variables correspondant dans la table.

```
PROC FORMAT ;
/*value label d'une variable numérique*/
VALUE sectorcode 1 = 'Internet and Computers'        2  = 'Biotech and Healthcare' 3
= 'Communications and Electronics';
/*Value label d'une variable caractère*/
 VALUE  $citycode 'BR1'='Birmingham UK'
                  'BR2'='Plymouth UK'
                  'BR3'='York UK'
                  'US1'='Denver USA'
                  'US2'='Miami USA'
                  other='INCORRECT CODE';
/* Value label pour date */
```

```
 VALUE datecode  low-'31DEC1979'd=[worddate20.]  '01JAN1980'd-high= '** Non concerné**';
RUN;
/*attribution des labels*/
DATA testdata;   SET testdata ;  FORMAT secteur sectorcode. ville $citycode.  date datecode. ;RUN;
```

I.2.8.2. Définition de values labels pour une liste de valeurs

Définissons les attributs pour les jours de la semaine en y associant des values labels.

```
PROC FORMAT ;
/* Chaque jour de la semaine */
VALUE jour_semlab1 1= 'lundi' 2='Mardi' 3= 'mercredi' 4 ='jeudi' 5= 'vendredi'6 ='samedi' 7 ='dimanche' ;
/* Jours pairs et impairs */
VALUE jour_semlab2 1,3,5,7= 'jours impairs'  2,4,6 = 'jours pairs' ;
/* Première moitié et seconde moitié de la semaine */
VALUE jour_sem3lab3 1-4= 'première moitié'  5-7= 'Seconde moitié' ;
RUN;
```

Après avoir défini ces trois values labels, on peut les associer un a à un la variable jour(en fonction du type d'affichage qu'on souhaite faire)

I.2.9. Définition des attributs d'une variable : longueur, format et informat

I.2.9.1. Méthode standard de définition des attributs

La longueur (length) d'une variable correspond au nombre maximum de positions qu'une valeur de la valeur occupe sur la mémoire. Par exemple, pour la variable SEXE, le nombre de position est 1 car les différentes valeurs M et F ne contiennent qu'un seuil élément. Pour la variable âge, on peut choisir jusqu'à trois positions.

Le format d'une variable correspond aux labels values attribuées à cette variable (voir définition de labels values). Le format sert à mettre en forme l'affichage des valeurs d'une variable lors de l'exécution des commandes telles que PROC PRINT ou PROC FREQ.

Quant à l'Informat, il correspond aux labels values attribuées aux données et qui sont utilisés par le système interne de SAS.

D'une manière générale, pour définir les propriétés d'un ensemble de variables, on utilise la commande suivante :

```
DATA MYDATA; SET MYDATA;
LENGTH VAR1 12 VAR2 $15 VAR3 VAR4;
INFORMAT  VAR1 BEST10.  VAR3 DDMMYY10. VAR4 DOLLAR10.2;
FORMAT  VAR1  BEST10.  VAR3 DDMMYY10. ;
RUN;
```

Dans cette commande, on a défini quatre variables. La première (VAR1) est une variable numérique de longueur égale à 12. La deuxième (Var2) est une variable en caractères de longueur 15. La troisième et la quatrième (VAR3 et VAR4) sont par défaut des variables numériques puisque leur longueur n'a été définie.

Par la suite, on attribue des formats aux variables afin de mettre en forme leur affichage. Ainsi, pour la première variable, on lui attribue le format BEST10 (Voir la base de connaissance de SAS pour la liste des différents formats et informats dans SAS). On attribue à la variable VAR3 le format date DDMMYY10. Et pour la quatrième variable, on lui attribue le format spécial noté DOLLAR10.2 qui associe le signe $ à toutes les valeurs de la table. Pour ce qui concerne l'informat, il a été défini de la même manière que le format. Cependant, il peut arriver que le format et l'informat d'une variable ne soient pas de même nature.

NB : Il faut noter qu'on peut utiliser les values labels définies dans PROC format comme des formats ou des informats pour les variables. Exemple :

```
/* Définition des values labels */
PROC FORMAT ;
VALUE var1code 1 = 'Internet and Computers' 2  = 'Biotech and Healthcare' 3  =
'Communications and Electronics'; RUN;
/* Association des values labels */
DATA mydata; SET mydata;
LENGTH var1 12 var2 $15 var3 var4;
INFORMAT  var1 var1code. var2 $upcase9.  var3 ddmmyy10. var4 dollar10.2;
FORMAT  var1  var1code.  var2 $upcase9.  var3 ddmmyy10. ;
RUN;
```

Il existe également une autre option très utile de PROC FORMAT qui permet de définir les formats des variables mais aussi toute autre statistique produite au cours des analyses. Il s'agit de l'instruction PICTURE. Exemple :

```
PROC FORMAT ;
PICTURE MYPICTURE(ROUND)    LOW-<0 =' 999' (PREFIX='-')   1000-HIGH=' 99999'
(PREFIX='+')   .=' '    ;
RUN;
```

Dans cette définition, on crée un format qui s'appelle MYPICTURE qui indique que toute les valeurs inférieures à 0 doivent être affichées comme -999, que toutes les valeurs supérieures à 1000 doivent être affichées comme +999 et que toutes les valeurs avec le point (.) doivent être représentées par l'espace vide. Ce type de formatage est très utile dans la gestion des valeurs aberrantes ou mêmes sur des tableaux de résultats classiques. On peut invoquer ce format dans toutes les proc autorisant la mise ne forme des données. On peut aussi l'associer à une variable comme format. Elle a donc les mêmes propriétés que les formats définies par l'instruction VALUE.

I.2.9.2. Utilisation l'instruction ATTRIB

Au lieu d'utiliser séparément les commandes LENGTH, INFORMAT et FORMAT dans la data step, on peut directement utiliser la fonction ATTRIB afin de définir dans une ligne toutes les propriétés d'une variable. Exemple :

```
DATA MYDATA; SET MYDATA;
/* Une seule variable avec un seul attribut*/
ATTRIB COST LENGTH=4;
/* Une seule variable avec plusieurs attributs*/
ATTRIB SALEDAY INFORMAT=MMDDYY. FORMAT=WORDDATE.;
/* Plusieurs variables avec un même attribut*/
ATTRIB X Y LENGTH=$4 LABEL='TEST VARIABLE';
* Plusieurs variables avec plusieurs attributs*/
ATTRIB x LENGTH=$4 LABEL='TEST VARIABLE'
      y LENGTH=$2 LABEL='RESPONSE';
/* Une liste de variables avec un même attribut*/
ATTRIB MONTH1-MONTH12     LABEL='MONTHLY SALES';
RUN;
```

Il est également possible d'attribuer des formats à des variables à l'intérieur d'une étape proc en vue d'afficher les résultats. Exemple :

```
PROC PRINT DATA=MYDATA NOOBS LABEL;
LABEL SALARY='Salary in U.S. Dollars';
```

```
FORMAT SALARY USCURRENCY. SITE $CITY. ; /*où les contenus de USCURRENCY et
$CITY ont été préalablement définis avec proc format*/
RUN;
```

I.2.9.3. Supprimer le format d'une variable

Pour supprimer le format d'une variable, on utiliser la commande FORMAT suivi du nom de la variable. Exemple : Enlevons le format de la variable SEXE. On a :

```
DATA MYDATA; SET MYDATA;
FORMAT SEXE ;
RUN;
```

I.2.9.4. Autres opérations sur les formats des variables

I.2.9.4.1. Affichage des formats

Pour afficher la description d'un format (exemple SECTEURCODE) on fait :

```
PROC FORMAT FMTLIB LIBRARY=WORK ;
SELECT SECTEURCODE;
RUN;
```

On peut aussi afficher toutes les values labels en excluant certains :

```
PROC FORMAT FMTLIB LIBRARY=WORK ;
EXCLUDE SECTEURCODE;
RUN;
```

I.2.9.4.2. Modification ou suppression d'un format

Pour modifier un format, la procédure se passe en trois étapes:
-il faut d'abord extraire le format en utilisant l'option CNTLOUT du proc format pour stocker les informations dans une table spécifiée
-Ensuite faire proc SQL pour insérer un nouveau enregistrement à la table correspondant au nouvel ajout
-Et enfin importer cette nouvelle table sous forme de formats. Exemple:

```
/*Exportation du format */
PROC FORMAT  LIBRARY=WORK CNTLOUT= MYFORMAT ;
SELECT JOUR_SEMLAB;
RUN;
/*Ensuite on fait proc SQL pour insérer une ligne*/
PROC SQL ;
INSERT INTO myformat
SET fmtname='JOUR_SEMLAB' , start= '8', end='8', label= 'jour J' ; /*ajoutons un
8ième jour qui s'appelle Jour J (initialement il y a ligne dans la table correspondant à
7 codes*/
quit;
/*importons ce format*/
PROC FORMAT  LIBRARY=work CNTLIN= myformat ;
SELECT JOUR_SEMLAB;
RUN;
/*affichage */
PROC FORMAT  LIBRARY=work FMTLIB;
SELECT jour_semlab;
RUN;
```

I.2.10. Sélection automatique d'une liste de variables

Le plus souvent, on est amené à exécuter une certaine opération sur un ensemble de variables défini par une liste (par exemple, dans les clauses VAR, KEEP ou DROP, etc). Si les noms de variables ont un même préfixe ou un même suffixe numérique séquentiel, la sélection des variables devient relativement facile. Par contre lorsqu'il n'y a aucune distinction claire dans les noms de variables la sélection du groupe de variable devient plus compliquée. Toutefois SAS fournit plusieurs méthode de sélection automatique de la liste des variables avec quelques fonctions que nous allons discuter dans cette section.

En effet, on peut distinguer quatre principaux cas de sélection de liste de variables :

1-sélection d'un groupe de variables ayant le même préfixe avec un suffixe numérique séquentiel ;

2-sélection d'un groupe de variables ayant le même préfixe (mais suffixes différents) ;

3- sélection d'un groupe contigu de variables ;

4-sélection de variables selon leur type (numérique, caractère, etc…).

I.2.10.1. Sélection d'un groupe de variables ayant le même préfixe avec un suffixe numérique séquentiel

Pour sélectionner un groupe de variables ayant le même préfixe avec des suffixes formant une séquence numérotée, on utilise le symbole « **-** » entre la première et la dernière variable de la liste. Par exemple, si les variables sont VAR1, VAR2, VAR3, ..., VARn, alors on peut sélectionner ces variables en utilisant VAR1 - VARn.

```
DATA MYDATA(DROP= VAR1-VAR3); SET MYDATA;
RUN;
```

I.2.10.2. Sélection d'un groupe de variables ayant le même préfixe (avec ou non des suffixes différents)

Pour sélectionner une liste de variables ayant le même préfixe, on utilise le symbole « **:** » sur le suffixe qui indique le mot qui identifie les variables. Le symbole **:** indique la fin du suffixe. En fait, cette commande permet de sélectionner toutes les variables dont le nom commence par une lettre ou un mot bien précis. Par exemple, pour sélectionner toutes les variables dont le nom commence par VAR on fait VAR **:**

```
DATA MYDATA(DROP= VAR :); SET MYDATA;
RUN;
```

I.2.10.3. Sélection d'un groupe de variables contiguës

Si le groupe de variables n'est pas nécessairement une liste numérotée mais que les variables sont contiguës et successivement ordonnée dans la table de données, alors on peut sélectionner l'ensemble de la liste en indiquant le nom de la première et de la dernière variable séparée par un double tiret «**--**». Exemple: la liste des variables définie A11 Boo6 C14 D2 Z90 U19 R16 M8 A3 E77 peut être sélectionnée comme A11**--**E77.

```
DATA MYDATA(DROP= VAR1--VAR3 ); SET MYDATA;
RUN;
```

De telles méthodes de sélection des variables peuvent s'avérer utile dans plusieurs contextes tels que l'élaboration d'une boucle DO LOOP. Nous reviendrions sur ces aspects un peu plus tard.

I.2.10.4. Sélection de variables selon leur type (numérique, caractère, etc…).

On distingue trois principaux types de variables sous SAS : numérique quantitative, numérique-date et caractère. Lors d'une procédure SAS, il arrive qu'on veuille exécuter la ligne sur seulement un certain type de variables. Par exemple, la procédure PROC MEANS qui calcule les statistiques descriptives comme la moyenne, n'est valide que sur les variables numériques. Dès lors qu'on veut exécuter une telle procédure sur l'ensemble des variables numériques, on peut spécifier dans VAR le mot clé _numeric_. Pour le cas d'une PROC FREQ, on peut utiliser _character_ ou utiliser _all_ lorsqu'on veut faire les tableaux de fréquence sur toutes les variables de la table(voir exemples ci-dessous).

```
DATA MYDATA(DROP=_numeric_ ); SET MYDATA; RUN;
DATA MYDATA(DROP=_character_ ); SET MYLIB.MYDATA;RUN;
PROC MEANS DATA= MYDATA; VAR _numeric_; RUN;
PROC FREQ DATA= MYDATA; VAR _character_; RUN;
PROC FREQ DATA=MYDATA ; VAR _all_; RUN;
```

I.2.11. Convertir le type d'une variable

Il arrive souvent qu'une ou plusieurs variables dans la table ne se trouvent pas sous le format adéquat. Par exemple une variable numérique qui se présent sous le format caractère, une variable en chaînes de caractère (formé de chiffres) se présentant sous format numérique ou une date qui se présente sous format texte (caractères). Face à ces situations, il faut exécuter quelques opérations de traitement afin de convertir la variable en format adéquat. Cette section discute des méthodes de conversion en selon différents cas.

I.2.11.1. Convertir une variable caractère en format numérique

Pour convertir une variable en caractères (contenant uniquement des chiffres) en format numérique, on utilise la fonction INPUT. L'exemple ci-dessous est une illustration.

```
DATA MYDATA ; SET MYDATA ;
new_salaire =INPUT(salaire,10.3);
DROP salaire;
RENAME new_ salaire = salaire;
RUN ;
```

Cette commande convertit la variable salaire (qui se présente initialement sous le type caractère) en format numérique avec dix postions en valeur entière et 3 positions en valeurs décimales. La conversion se fait en trois étapes. D'abord, on crée une nouvelle variable nommée new_salaire qui contient les valeurs converties en numérique. Dans un second temps, on supprime l'ancienne variable salaire qui contient les valeurs sous format caractère. Et finalement dans un troisième temps, on renomme la variable new_salaire pour lui attribuer le nom initial salaire.

NB : Cette méthode n'est valable que lorsque la variable à convertir contient uniquement des valeurs en chiffres même si elles se présentent sous format caractère. Lorsque la variable contient des chaînes de caractères en lettre, les valeurs converties seront des données manquantes.

I.2.11.2. Convertir une variable numérique en format caractères

Pour convertir une variable numérique en format caractère (ne contenant que des chiffres), on utilise la fonction PUT. L'exemple ci-dessous est une illustration.

```
DATA MYDATA ; SET MYDATA ;
CHAR_IDENT = PUT(IDENT, 20.) ;
DROP IDENT ;
RENAME CHART_IDENT=IDENT ;
RUN ;
```

Cette commande convertit la variable IDENT (initialement numérique) en une variable caractère (de 20 position maximum) constituée uniquement de chiffres.
Attention dans l'utilisation de la fonction PUT, notamment lors de la spécification du nombre de positions. Voici ci-dessous quelques exemples sur l'effet de PUT.

Soit le chiffre 32000, en appliquant la fonction PUT telle que PUT(32000,5.) on trouve 32000 car la partie entière est 5 positions et la partie décimale est 0 positions.

Soit le chiffre 32000, en appliquant la fonction PUT telle que PUT(32000,5.2) on trouve 320.00 car même si la partie entière est 5 positions et la partie décimale est 2 positions (la partie décimale est prioritaire dans PUT).

Soit le chiffre 32,000 (écrit en format anglais avec virgule comme séparateur de milliers) en appliquant la fonction PUT telle que PUT(32,000, comma6.) on trouve 32000 car la partie décimale est 0 position. Le format de conversion comma6. est un format qui écrit les valeurs numériques en séparant tous les trois chiffres avec une virgule et qui sépare la fraction décimale avec un point.

I.2.11.3. Convertir en format date (numérique) une variable se présentant sous format date (en chaine de caractères)

On peut utiliser deux méthodes pour convertir une date en chaîne de caractères en date numérique : la méthode SUBSTR-MDY ou la méthode INPUT-PUT.

I.2.11.3.1. Méthode SUBSTR

Soit une date qui se présente sous la forme suivante "20160511" qui correspond en fait au 11 MAI 2016. Pour mettre cette date sous une forme numérique (reconnaissable comme date), on peut d'abord extraire chaque élément en tant que partie d'une date (en utilisant la fonction SUBSTR). Ensuite associer les trois éléments pour former une date (en utilisant la fonction MDY). Voir exemple ci-dessous.

```
/*Création d'une date au format caractère (supposition)*/
data MYDATA ; set MYDATA ;
DATECAR = "20160511";
Run;
/* Conversion de DATECAR en DATENUM*/
DATA MYDATA ; SET MYDATA;
an = substr(DATECAR,1,4) ;
mois = substr(DATECAR,5,2) ;
jour = substr(DATECAR,7,2) ;
DATENUM = mdy(mois,jour,an);
RUN;
/*formatage de la date*/
DATA MYDATA ; SET MYDATA;
ATTRIB DATENUM INFORMAT=mmddyy. FORMAT=mmddyy.;
RUN ;
```

I.2.11.3.2. Méthode INPUT-PUT

La méthode INPUT-PUT est une combinaison dans la même instruction de la fonction INPUT et de la fonction pour convertir directement la date du format caractère au format numérique. En prenant l'exemple de DATECAR dont la valeur est "20160511", on a les étapes suivantes :

```
DATA MYDATA ; set MYDATA ;
DATENUM = INPUT( PUT(DATECAR, 8.), Yymmdd10.);
PUT DATENUM = Date9.;
RUN;
/*formatage de la date*/
DATA MYDATA ; SET MYDATA;
ATTRIB DATENUM INFORMAT=mmddyy. FORMAT=mmddyy.; RUN ;
```

La fonction convertit d'abord "20160511" en format date caractères de 8 positions et en spécifiant le format date de cette valeur caractère, la fonction INPUT la convertit en date (numérique). Il faut signaler néanmoins que cette même si cette formulation des étapes est efficient en termes de nombre de lignes de commande, elle consomme beaucoup de temps.

REMARQUE

On peut rendre dynamique les fonctions INPUT et PUT lors de la conversion des variables en utilisant les fonctions : INPUTN, INPUTC, PUTN et PUTC (consulter la base de connaissance de SAS).

I.2.12. Exécuter une opération en boucle sur une liste de variables (boucle par colonnes)

Exécuter une boucle DO LOOP sur une liste de variables consiste à effectuer une même opération (définie par des instructions) sur chacune des variables de la liste. Par exemple, on souhaite diviser par 10 toutes les variables d'une liste. Ou encore on souhaite remplacer par un point (missing) toutes les variables d'une liste pour lesquelles la valeur renseignée est -999. Comme il s'agit d'une même opération pour toutes les variables, alors on peut utiliser une boucle qualifiée boucle par colonne (car il s'agit des variables).

Pour réaliser une boucle par colonnes, on utilise dans l'étape DATA l'instruction ARRAY (qui spécifie la liste des variables) et la boucle DO LOOP qui définit l'espace de la boucle ainsi que les opérations à effectuer. L'exemple ci-dessous est une application :

```
DATA MYDATA(drop=i);
SET MYDATA;
ARRAY myvar{5}      temperatue humidite cons_elect cons_eauFroid cons_eauChaud;
DO i=1 to 5 ;
IF myvar{i}=. THEN myvar{i}=0;
END;
RUN;
```

L'exemple ci-dessus est une boucle qui remplace les valeurs manquantes par 0 pour chacune des 5 variables de la liste myvar (temperatue humidite conso_elect conso_eauFroide conso_eauChaude).

Puisqu'il y a 5 variables dans la liste alors la dimension de la boucle sera égale à la dimension de myvar égale à 5. On ajoute l'instruction (drop=i) car la boucle crée automatique une variable nommée ici i qu'il va falloir supprimer après l'exécution de la boucle.

Il faut noter que lorsque la liste des variables est très longue et qu'on ait pas la possibilité de compter aisément le nombre de variables, on peut utiliser simplement dim(myvar) dans la boucle (voir exemple ci-dessous).

```
DATA MYDATA (drop=i);
SET MYDATA;
ARRAY myvar temperatue humidite conso_elect conso_eauFroide conso_eauChaude;
DO i=1 to dim(myvar) ;
IF myvar{i}=. THEN myvar{i}=0;
END;
RUN;
```

Par ailleurs, on peut faciliter l'écriture de la liste des variables en utilisant les règles de spécification des listes discuté dans la section précédente.

En effet, lorsque les variables ont le même préfixe avec des suffixes numériques, on peut adopter le listing avec simple tiret dans l'instruction ARRAY (voir exemple ci-dessous).

```
DATA MYDATA(drop=i);
SET MYDATA;
ARRAY mydummies dummy2005-dummy2010 ;
DO i=1 TO dim(mydummies); IF annee=2004+i THEN mydummies{i}=1 ;
IF annee^=2004+i THEN mydummies{i}=0 ; END; RUN;
```

Cette commande crée une dummy pour chaque année allant de 2005 à 2010 à partir de la variable annee de la table. Les variables binaires à créer ont toutes pour préfixe le mot dummy avec des suffixes allant de 2005 à 2010. Pour retrouver ces suffixes à l'intérieur de la boucle, on a fixé un compteur allant de 1 à dim(mydummies) qui est de 6.

Lorsque les variables qui forment le ARRAY ont le même préfixe (avec des suffixes différents ou pas), on peut construire la boucle comme suit :

```
DATA MYDATA(drop=i);
SET MYDATA;
ARRAY myvar conso : ;
DO i=1 TO dim(myvar);
myvar{i}= myvar{i}*100 ;
END; RUN;
```

Cette commande multiplie par 100 toutes les variables dont le nom commence par conso (par exemple conso_electri, conso_eauFroide, conso_eauChaude, etc…)

Lorsque les variables qui forment le ARRAY n'ont pas nécessairement le même préfixe, ni les mêmes suffixes mais qu'ils sont contiguës dans la table alors connaissant la première et la dernière variable de la liste on peut adopter le type de listing suivant :

```
DATA MYDATA(drop=i);
SET MYDATA;
ARRAY myvar temperatue--conso_eauChaude;
DO i=1 TO dim(myvar);
myvar{i}= myvar{i}*100 ;
END; RUN;
```

Lorsque le ARRAY est formée avec les variables de même type (par exemple numérique) alors on peut adopter le type de listing qui se présente comme suit :

```
DATA MYDATA(drop=i);
SET MYDATA;
ARRAY myvar _numeric_ ;
DO i=1 TO dim(myvar);
myvar{i}= myvar{i}*100 ;
END; RUN;
```

Ici, nous multiplions toutes les variables numériques de la table par 100.

I.2.13. Centrer et réduire les variables dans une table

Centrer et réduire les variables c'est soustraire les moyennes des valeurs et diviser par l'écart-type de façon à avoir une moyenne nulle et une variance égale à 1. Pour réaliser cette opération, on utilise la commande PROC STANDARD. L'exemple ci-dessous crée une nouvelle table nommée MYSTAND_DATA à partir de la table MYDATA en standardisant les variables.

```
PROC STANDARD DATA=MYDATA OUT=MYSTAND_DATA MEAN=0 STD=1;
VAR AGE REV R1 R2 R3;
RUN;
```

I.3. Opérations sur observations

I.3.1. Sélection ou suppression des observations selon une condition

Pour sélectionner les observations selon une condition on peut utiliser la commande IF combinée avec la commande DELETE ou la commande WHERE en spécifiant la condition.

Sélectionner les individus dont l'âge est inférieur à 40.

```
DATA MYLIB.MYDATA40; SET MYLIB.MYDATA;
IF age<40;
RUN;
```

Ou encore

```
DATA MYLIB.MYDATA40; SET MYLIB.MYDATA;
WHERE age<40;
RUN;
```

Au lieu de garder les individus de moins de 40 ans, on peut supprimer les individus de plus de 40 ans. On a alors :

```
DATA MYLIB.MYDATA40; SET MYLIB.MYDATA;
IF age>=30 THEN DELETE;
RUN;
```

Sélectionnons les hommes âgés de plus de 40 ans

```
DATA MYLIB.MYDATAH40; SET MYLIB.MYDATA;
IF AGE>=40  AND  SEXE="M";
RUN;
```

Ou encore

```
DATA MYLIB.MYDATAH40; SET MYLIB.MYDATA;
WHERE AGE>=40  AND  SEXE="M";
RUN;
```

I.3.2. Trier les observations selon les valeurs d'une ou de plusieurs variables

Pour trier les observations selon les valeurs d'une variable, on utiliser la commande PROC SORT. Par exemple, en utilisant la table MYDATA de la librairie MYLIB, trions les observations selon l'âge (du plus petit au plus grand). On a :

```
PROC SORT DATA=MYLIB.MYDATA; BY AGE; RUN;
```

Trions par rapport à l'âge (du plus grand au plus petit) :

```
PROC SORT DATA=MYLIB.MYDATA; BY DESCENDING AGE; RUN;
```

Il faut noter qu'on peut utiliser plusieurs variables afin d'effectuer le tri. Par exemple, trions les observations les valeurs croissantes de l'âge et du revenu. On a :

```
PROC SORT DATA=MYLIB.MYDATA; BY AGE REV; RUN;
```

Tri des observations par valeurs croissantes de l'âge et valeurs décroissante du revenu :

```
PROC SORT DATA=MYLIB.MYDATA; BY AGE DESCENDING REV; RUN;
```

I.3.3. Attribuer un rang aux observations par numérotation

I.3.3.1. Rang par rapport à tout l'échantillon

Pour attribuer un rang aux observations par numérotation dans SAS, on peut utiliser deux méthodes : soit utiliser la fonction _N_ ou bien utiliser la fonction RETAIN. Les deux commandes ci-dessous fournissent un exemple d'application pour chacune des deux méthodes :

```
/* Utilisation de la fonction _N_ */
DATA MYDATA;   SET MYDATA; numero1 = _N_;   RUN;
/* Utilisation de la fonction retain */
DATA MYDATA; SET MYDATA; RETAIN numero2 0;   numero2+1;  RUN;
```

NB : Pour que ces numérotations correspondent aux rangs des observations, il faut au préalable trier la table selon les valeurs croissantes ou décroissante par rapport à la variable de rang en utilisant PROC SORT. Par exemple pour attribuer des rangs aux clients par rapport à leur poids, on peut faire un tri croissant ou décroissant sur les montants des commandes passées.

Par ailleurs, lorsque l'on veut que la numérotation ne commence pas par 1 mais plutôt par un autre nombre, on peut utiliser la fonction retain pour généraliser la numérotation (voir exemple ci-dessous)

```
DATA MYDATA; SET MYDATA; RETAIN numero3 44;   numero3+1;  RUN;
```

Ici, la numérotation commence à partir de 45 et se poursuit jusqu'à la fin de la table.

I.3.3.2. Rang à l'intérieur d'un sous-groupe

La numérotation effectuée ci-dessous attribue un rang à l'observation par rapport à tous les autres observations de la table. Il arrive qu'on veuille numéroter les observations à l'intérieur d'un sous-groupe. Par rapport ranger les individus (identifiés par code_indiv) en fonction de leur âge à l'intérieur du ménage, ou ranger les employés (identifiés par code_emp) par ordre d'ancienneté à l'intérieur de l'entreprise, etc.. L'exemple ci-dessous range les individus à l'intérieur du ménage (NB : il faut d'abord faire PROC SORT afin de créer les rangs. Ex : âge).

```
PROC SORT DATA=MYDATA ;BY code_commune code_menage DESCENDING age ;
RUN ;
```

```
DATA MYDATA;
  SET MAYDATA;
  numero + 1;
  BY code_commune code_menage;
  IF  first.code_menage THEN numero = 1;
RUN;
```

Il faut remarquer ici que même si on cherche à attribuer des rangs aux individus (identifiés par code_indiv), la clause BY s'arrête au niveau de code_menage et que la clause IF porte également sur code_menage. Ainsi pour attribuer des rangs à l'intérieur des sous-groupes, la sélection se fait sur la clé d'identification du sous-groupe.

I.3.4. Générer un identifiant unique à partir d'un groupe de clés d'identification

Supposons qu'on dispose d'une table contenant des observations renseignées trois clés d'identification : la commune, le ménage et le numéro individu. Dans chaque commune, le numéro du ménage est codé à partir de 1 et dans chaque ménage, le

numéro de l'individu est codé à partir de 1. On souhaite attribuer à chaque individu un numéro unique capable de l'identifier parmi tous les individus au niveau national, alors, on procède par regroupement des valeurs commune-ménage-individu. Pour cela, on peut utiliser la commande suivante :

```
DATA mydata; SET mydata;
 BY codecommune codemenage  code_indiv;
 IF code_indiv^=lag(code_indiv) THEN ident+1;
  ELSE ident+0;
RUN;
```

I.3.5. Identifier les observations dupliquées par une variable indicatrice

Pour identifier les observations dupliquées par une variable indicatrice à partir d'une clé on peut utiliser la fonction FIRST. ou LAST. Les lignes de commande ci-dessous fournissent un exemple :

```
DATA MYDATA;
SET MYDATA;
BY ID;
IF first.ID THEN duplic=0;
IF duplicate=. THEN duplic=1;
RUN;
```

La variable ID représente ici la clé d'identification à partir de laquelle on effectue les valeurs dupliquées. Ici, on considère la première valeur observée (First.) comme la principale valeur (non dupliquée). Mais on pouvait aussi considérer la dernière valeur (Last.) comme la principale valeur. La variable duplique est une variable binaire qui prend 0 pour la principale observation et 1 pour les observations dupliquées.

Il faut noter que la commande ci-dessous recherche les valeurs dupliquées à partir d'une seule clé d'identification. Mais il arrive que la recherche des valeurs dupliquée porte sur l'association de plusieurs clés d'indentification. Par exemple : code commune- code ménage-code_indiv. Dans ce cas, la variable binaire de duplication de calcule comme suit :

```
DATA MYDATA;
SET MYDATA;
BY codecommune codemenage code_indiv;
IF first.code_indiv THEN duplic=0;
IF duplicate=. THEN duplic=1; RUN;
```

Il existe également une seconde méthode d'identification d'observations dupliquée qui consiste à ranger les observations par clé en attribuant des numéros à partir de 1. Dès lors l'observation qui porte le numéro 1 est la principale observation alors que les observations portant des numéros différents de 1 sont des observations dupliquées. Les commandes ci-dessous illustrent un exemple d'application de cette méthode.

```
DATA MYDATA (DROP=numero);
 SET MYDATA;
 numero + 1;
 BY codecommune codemen code_indiv;
  IF  first.code_indiv THEN numero = 1;
  IF numero=1 THEN duplic=0;
 ELSE duplic=1;
RUN;
```

I.3.6. Suppression directe des observations dupliquées selon les valeurs d'une ou de plusieurs variables

Pour supprimer les observations dupliquées par rapport à une condition bien définie, on peut utiliser soit utiliser l'étape DATA avec l'option first. (ou last.) ou la commande PROC SORT avec l'option NODUPKEY.

Par exemple, en utilisant la table MYDATA, supprimons les individus qui se répètent en utilisant la clé d'identification ID. On a :

```
DATA MYLIB.MYDATA;  SET MYLIB.MYDATA;
 BY ID;
IF first.ID; /*garde la première ligne dupliquée  */
 /*if last.ID; */ /*pour garder la dernière ligne dupliquée*/
 RUN;
```

On peut aussi utiliser la commande PROC SORT suivant :

```
PROC SORT DATA = MYLIB.MYDATA OUT = MYLIB.MYDATA NODUPKEY;  BY ID ;
RUN;
```

NB : La recherche de duplication peut s'effectuer en utilisant plusieurs variables en même temps. Exemple :

```
PROC SORT DATA = MYLIB.MYDATA OUT = MYLIB.MYDATA NODUPKEY;  BY ID AGE
SEX ;  RUN;
```

I.3.7. Calcul du nombre de répétitions d'une valeur sur une variable (comptage)

Le calcul du nombre de répétitions sur les valeurs d'une variable s'avère utile dans de nombreuses situations. Supposons qu'on ait organisé une enquête ménage sur plusieurs commune répartit sur plusieurs régions. Dans cette enquête, un questionnaire a été prévu pour renseigner les informations socio-démographiques sur tous les individus vivant dans le ménage. Dans cette enquête, des codes sont attribués à tous les régions. A l'intérieur de chaque région, un code est attribué à la commune et à l'intérieur de chaque commune un code est attribué aux ménages et à l'intérieur de chaque ménage, un code est attribué à chaque individu. On sait que dans cette situation, pour construire un identifiant unique aux individus au niveau national, il faut faire rassembler code région-code commune-code ménage-code individu.

Supposons maintenant qu'on veuille calculer la taille de chaque ménage présent dans cette enquête.

En effet, pour connaitre la taille du ménage, il faut compter le nombre de chaque code ménage dans chaque code commune. Il s'agit, en fait, de compter le nombre de répétitions de l'information relative au code ménage. L'exemple ci-dessous illustre la démarche de comptage du nombre de ménage par commune.

```
PROC SORT DATA=MYDATA  ;   BY code_region  code_commune code_menage ;
RUN ;
DATA MYDATA; SET MYDATA;
BY code_region  code_commune code_menage ;
RETAIN taille_men 0;
IF first.code_region or first.code_commune or first.code_menage  THEN taille_men=0;
taille_men=taille_men+1;
IF last.code_region or last.code_commune or last.code_menage ;
run;
```

I.3.8. Exécuter une opération en boucle sur les observations (boucle par lignes)

I.3.8.1. Boucle DO LOOP pour modifier une table existante

Exécuter une boucle DO LOOP sur les observations consiste à effectuer des opérations séquentielles (définie par des instructions) en se déplaçant de ligne en ligne. Dans sa structure la plus simple, une boucle sur les observations porte sur une seule colonne en créant ou en modifiant les valeurs de celle-ci en allant de ligne en ligne. L'exemple le plus courant d'une boucle sur observations est la numérotation des individus dans la table.

Cependant l'exécution d'une boucle DO LOOP sur une table existence peut produire quatre résultats différents selon la formulation de la boucle. Il serait important de comprendre les différentes formulations et les résultats associés afin de choisir celle qui correspond au besoin.

Par exemple prenons une table de données contenant initialement 50 observations correspondant chacune à un individu. Supposons qu'il existe déjà dans la table clé d'identification des individus nommée ID prenant les valeurs de 1 à 50. Nous souhaitons maintenant créer une nouvelle clé d'identification des individus nommée code prenant les valeurs de 1000 à 50000. On remarque alors que ce nouveau code est une multiplication par 1000 de l'ancien code (code=1000*ID). Nous voulons utiliser une boucle qui passe en revue tous les individus afin de les attribuer la valeur 1000*ID. Proposons pour cela quatre formulations de la boucle DO LOOP et déterminons par la suite celle (s) qui est (sont) efficace(s) pour résoudre le problème posé.

Première formulation

```
DATA MYDATA(DROP=i);
SET MYDATA;
DO i=1 TO 50 ;
CODE=1000*i;
/* ou ajouter PUT ; ou PUT CODE ; */
END;
RUN;
```

Cette première formulation retient simplement la dernière valeur du compteur pour remplir la colonne code. On aura donc pour tous les individus la même valeur égale à 50000. Il apparait donc clairement que cette formulation n'est pas adaptée au cas étudié.

Deuxième formulation

```
DATA MYDATA(DROP=i);
SET MYDATA;
DO i=1 TO 50 ;
IF ID=i THEN
CODE=1000*i;
/*  ou ajouter PUT ; ou PUT CODE ;*/
END;
RUN;
```

En exécutant cette deuxième, on voit que SAS attribue 1000 comme code à l'observation pour qui ID=1 ; 2000 à celle dont ID=2, …., 50000 à l'observation dont ID est égal à 50. Cela correspond bien au résultat recherché. Donc la seconde formulation est boucle efficace pour résoudre le problème posé.

Troisième formulation

```
DATA MYDATA(DROP=i);
SET MYDATA;
DO i=1 TO 50 ;
CODE=1000*i;
OUTPUT;
END;
RUN;
```

Quant à cette troisième formulation, elle démultiplie chaque observation de la table initiale (k-fois chaque observation où k est la valeur maximale du compteur, ici 50). Après avoir démultiplié k-fois chaque observation, SAS génère les valeurs de la variable calculée et attribue une valeur à chaque démultiplication de l'observation. La même opération est répétée pour toutes les observations initialement présentes dans la table. On voit alors que cette troisième formulation n'est pas efficace face au problème posé. Une telle formulation de la boucle peut surtout être utile dans les cas d'analyses combinatoires (dans lesquelles on démultiplie d'abord une valeur avant d'associer les autres valeurs pour former la combinaison).

Quatrième formulation

```
DATA MYDATA(DROP=i);
SET MYDATA;
DO i=1 TO 50 ;
IF ID=i THEN
CODE=1000*i;
OUTPUT;
END;
RUN;
/* Suppression des démultiplications de la table initiale*/
PROC SORT DATA = MYDATA OUT = MYDATA NODUPKEY;  BY CODE ; WHERE CODE ne .;  RUN;
```

La quatrième formulation de la boucle procède aussi par démultiplication des observations initiales comme dans la troisième formulation. Cependant la manière de distribuer les valeurs de la variable calculée est complètement différente. En effet, cette quatrième formulation marche comme suit. Dans un premier temps, elle démultiplie k-fois chaque observation de la table initiale. Dans un second temps, elle sélectionne les k démultiplications de la première observation et les attribue la première valeur de la variable calculée. Ensuite, elle sélectionne les (k-1) démultiplications de la seconde observation de la table initiale en les attribuant la seconde valeur de la variable calculée (NB : la première démultiplication de la seconde observation n'est pas sélectionnée, donc la variable calculée est manquante). Dans un troisième temps, les (k-2) démultiplications de la troisième observation initiale sont sélectionnées afin de les attribuer la troisième valeur calculée (ici aussi, les deux premières observations initiales non sélectionnées auront une valeur manquante pour la variable calculée). Ce processus se réitère jusqu'à la sélection de la démultiplication de la dernière observation initiale (il faut noter que pour la démultiplication de la dernière observation initiale seulement une seule ligne sera renseignée pour la variable calculée ; les k-1 autres démultiplications auront des valeurs manquantes pour la variable calculée). Au final cette formulation ne réalise que partiellement la tâche demandée car la table obtenue contient non seulement des duplications complètes d'informations associées à des valeurs manquantes. Pour compléter le travail, il faut alors ajouter une procédure de suppression des duplications tout en excluant les valeurs manquantes. D'où la nécessité d'ajouter une PROC SORT avec une instruction WHERE.

En conclusion, chaque formulation discutée ci-dessous permet d'aboutir à un résultat différent dont il faut apprécier son adéquation par rapport à l'objectif recherché.

Astuces

Avant d'exécuter les formulations 2 et 4, il faut penser d'abord à numéroter les observations de sorte à obtenir les mêmes valeurs que celle spécifiée dans le compteur de la boucle et ensuite utiliser la clause IF ID=i THEN. Par exemple, pour une boucle de type : DO i=**1** TO **50** , il faut numéroter les observations en utilisant la fonction _N_ comme suit :

```
DATA MYDATA;   SET MYDATA; ID = _N_;   RUN;
```

En revanche, pour une boucle de type DO i=**45** TO **95** , il faut numéroter les observations en utilisant la fonction RETAIN comme suit :

```
DATA MYDATA; SET MYDATA; RETAIN ID 44;   ID+1;  RUN;
```

I.3.8.2. Boucle DO LOOP pour créer une nouvelle table

I.3.8.2.1. Boucle définie par un ensemble de chiffres consécutifs

L'exemple ci-dessous crée une table nommée A égale au carré du compteur i :

```
DATA A;
DO i = 1 TO 5;
  y = i**2;
OUTPUT;
END;
RUN;
```

Pour éviter de démultiplier la table d'analyse initiale, on peut, par exemple, utiliser cette méthode pour générer une nouvelle variable à part et puis merger la table obtenue avec la table d'analyse (nous aborderons les méthodes de merge de tables dans la section suivante).

I.3.8.2.2. Boucle définie par une liste de valeurs

On peut aussi construire la boucle sur une liste de valeurs à la place (de i=1 to n). Exemple :

```
DO i = 1, 2, 3, 4, 5;
  y = i**2;
OUTPUT;
END;
RUN;
```

I.3.8.2.3. Une boucle définie par DO WHILE et DO UNTIL

Supposons qu'on veuille déterminer combien d'années il faudra pour accumuler 50000 sur un compte d'épargne rémunéré à 5% sachant qu'on dépose chaque année une somme 1200. Dans ce type de calcul, on peut se servir des DO WHILE ou DO UNTIL qui exécute un ensemble d'instructions tant qu'une condition est respectée (DO WHILE) ou tant qu'une condition n'est pas vérifiée (DO UNTIL). Il faut savoir que DO WHILE est la boucle réciproque à DO UNTIL. On peut donc les inter-changer à condition de modifier l'écriture de la condition. Les deux exemples ci-dessous illustrent l'utilisation de ces deux types de boucles pour calculer le nombre d'années minimum nécessaire pour accumuler les 50000.

Le programme suivant utilise une DO UNTIL pour effectuer le calcul pour nous:

```
/* DO WHILE*/
DATA mydata; SET mydata;
DO WHILE (montant_accumul<=50000) ;
montant_accumul+12000 ;
montant_accumul+ montant_accumul*0.05;
N_annee+1 ;
OUTPUT;
END;
RUN ;

/* DO UNTIL*/
DATA mydata; SET mydata;
DO UNTIL (montant_accumul>=50000) ;
montant_accumul+12000 ;
montant_accumul+ montant_accumul*0.05;
N_annee+1 ;
OUTPUT;
END; RUN ;
```

I.3.8.2.4. Boucle définie par une combinaison de DO LOOP avec DO WHILE and DO UNTIL

```
 /* avec while */
Data A;
y = 0;
DO i = 1 TO 5 BY 0.5 WHILE(y < 20);
   y = i**2;
   OUTPUT;
END;
RUN;

/* avec Until */
DATA A;
y = 0;
DO i = 1 To 5 by 0.5 UNTIL(y > 20);
   y = i**2;
OUTPUT;
END;
RUN;
```

I.3.8.2.5. Boucle définie par une combinaison de IF THEN avec DO LOOP

Soit la table A définie par :

```
DATA A;
DO i = 1 TO 10;
y = i**2;
z=i**3;
OUTPUT;
END;
RUN;
```

On peut définir une table B avec la combinaison suivante :

```
DATA B; SET A;
IF y>36 THEN
   DO;
      W1=y**2;
      PUT w1 ;
   END;
   ELSE w2=y**3;
   RUN ;
```

I.3.8.3. Calcul de la somme cumulée d'une variable

Dans cet exemple, nous allons calculer le montant cumulé du chiffre d'affaire du chiffre d'affaire une entreprise (ranger du plus petit au plus grand)

Pour calculer le cumul d'une variable par la méthode traditionnelle du RETAIN, on peut utiliser la formulation suivante:

```
/* Ranger les observations */
PROC SORT DATA= mydata; BY CA_Cmde; RUN;
/*Calcul du cumul*/
DATA mydata ; SET  mydata;
  IF FIRST.CA_Cmde THEN cumul=CA_Cmde;
  cumul+CA_Cmde;
RUN;
```

Et pour calculer le cumul à l'intérieur d'un sous-groupe défini par une clé d'identification, on ajoute l'instruction BY comme suit :

```
/* Ranger les observations */
PROC SORT DATA= mydata; BY  ID CA_Cmde; RUN;
/*Calcul du cumul*/
DATA mydata ; SET  mydata;
 BY ID /* ou ajouter NOTSORTED */;
  IF FIRST.ID THEN cumul_Id= CA_Cmde;
    cumul_Id + CA_Cmde;
RUN;
```

Notons aussi qu'on peut calculer le cumul en utilisant une boucle DO LOOP. Dans ce cas, on se servir de la fonction LAG() afin de récupérer chaque fois la valeur précédent du cumul et l'additionner à la valeur actuelle de la variable. Toutefois, cette méthode n'est facile à mettre en œuvre que dans le cadre d'une macro-commande (nous étudierons les macros au chapitre 3).

I.4. Opérations sur tables de données

I.4.1. Fusion de deux tables de données

On distingue trois principaux cas de fusions de tables de données : la fusion de deux tables de mêmes variables mais d'observations différentes, la fusion de deux tables de mêmes observations mais de variables différentes et la fusion de tables d'observations et de variables différentes. En considérant le dernier cas comme un cas particulier du fusions avec variables différentes, la fusion de tables SAS se réduit à deux principaux cas. Le but de cette section est de montrer les étapes à suivre pour réaliser chacun de ces deux types de fusions

I.4.1.1. Fusion de table de mêmes variables mais d'observations différentes (append)

Pour fusionner ces deux tables ayant les mêmes variables mais des observations différentes, on utilise simplement la commande SET suivie du nom des deux tables. Par exemple, supposons qu'on ait deux tables de données (PATIENT_TAB1 et PATIENT_TAB2) renseignées sur des patients provenant de deux centres hospitaliers différents. Les informations relevées sur les patients sont les mêmes dans les deux tables et aucun patient ne se répète dans les deux tables. Pour fusionner ces deux bases pour en faire une base unique, nous allons nous servir de la commande SET comme suit :

```
DATA PATIENT_DATA;
SET PATIENT_TAB1 PATIENT_TAB2 ;
RUN ;
```

Les deux tables sont simplement juxtaposées l'une sur l'autre pour construire une table finale nommée PATIENT_TAB.

I.4.1.2. Fusion de tables avec les mêmes observations mais de variables différentes (merge)

Conceptuellement, la fusion de bases portant sur les variables est l'une des procédures les plus délicates dans la gestion et la manipulation des données. Par exemple, pour fusionner deux bases dont les variables sont différentes, deux cas de figure sont couramment rencontrées qu'il faut d'abord comprendre.

D'abord, il faut toujours s'assurer qu'on ait dans la table de données au moins une variable servant de clé d'identification des individus (ou des observations). Par exemple pour une base de données provenant d'une enquête d'évaluation des performances scolaires des lycéens dans les communes d'un pays donné, on peut avoir jusqu'à trois variables d'identification (correspondant chacune à trois niveau d'information). D'abord le code de la commune qui identifie une commune particulière parmi les autres communes du pays. Ensuite le code du lycée qui identifie un lycée particulier parmi tous les lycées dans une commune donnée. Et enfin le code de l'élève qui permet d'identifier un élève particulier parmi les élèves d'un lycée donné. Ainsi, pour identifier un élève parmi tous les élève au niveau national, il faut aligner code commune, code lycée et code élève. Ce qui constitue la clé unique pour identifier un lycéen particulier dans la base.

L'élément capital qui ressort dans cette hiérarchisation des niveaux d'information, qui est d'ailleurs trivial, c'est que plusieurs élèves se retrouvent dans un lycée et plusieurs lycées peuvent se retrouver dans une commune. Même si cette information est évidente, elle a une implication fondamentale dans la perspective de fusion de table de données. Par exemple si l'on a deux bases de données : l'une contient uniquement les informations sur les lycéens (âge, sexe, etc..) et l'autre contient uniquement les informations sur les lycées (Nom, date de création, nombre d'enseignants, etc..). Pour fusionner ces deux bases, il faut garder juste à l'esprit que les informations sur un lycée particulier doivent être distribuées à tous les élèves appartenant à ce lycée. Pour cela la connaissance de l'identifiant du lycée (c'est-à-dire le code lycée) est indispensable. En l'absence d'identifiant lycée la fusion échoue et la base qu'on obtient sera complètement fausse.

Dès lors le premier élément qu'on doit regarder avant de fusionner des tables de données, c'est de s'assurer de l'existence d'une même clé d'identification dans chacune de vos bases. Et si ces identifiants existent mais sous des noms différents entre les bases, il faut harmoniser ces noms.

Après avoir vérifié ces préalables, on cherche à déterminer dans quel cas de figure on se trouve par rapport aux cas que nous allons maintenant présentés.

En effet, pour fusionner des deux bases dont les variables sont différentes, deux cas de figure se présentent : soit les informations portent sur la même unité d'analyse, soit l'information est hiérarchisée.

On dit que les informations portent sur la même unité d'analyse si l'on a deux bases portant sur les mêmes individus mais sur des variables différentes. Par exemple, pour revenir au cas des lycéens, si la première base contient les informations socio-démographiques sur le lycéen : âge, sexe, âge de ses parents, CSP de ses parents, etc. Et si la seconde base contient les informations sur les notes qu'il a obtenues lors d'un test d'évaluation (notes en Math, Langue, Histoire-Géo, etc..). Les informations contenues dans ces deux bases de données étant mesurées sur la même unité d'analyse (le lycéen), la procédure de fusion doit exploiter ce critère. Dans ce cas de figure, il est impératif que la variables captant le code de l'élève se retrouve dans les deux bases, car c'est cet identifiant qui permet de faire correspondre à chaque élève de la première base, le reste de ses informations provenant de la seconde base. Assurez-vous donc que votre variable d'identification est bien présente dans les deux bases à fusionner.

Le second cas de figure concerne la hiérarchisation des niveaux d'information. Il y a hiérarchisation des niveaux d'information lorsque les deux bases ne portent pas sur les mêmes unités d'analyse. Par exemple lorsque la première base contient les informations mesurées sur les élèves (âge, sexe, notes, etc..). Et que la seconde base contient les informations mesurées sur les lycées (date création, nombre de salles de classes, nombre d'enseignants, nombre moyen d'élève par classe, nombre moyen d'enseignants par élèves, etc). Ici on a deux niveaux d'informations hiérarchisées (élève et lycée). Nous utilisons le terme d'« informations hiérarchisées » pour rendre compte du fait que pour identifier un élève dans la base, il faut d'abord identifier son lycée. Ensuite dans ce lycée on identifie l'élève par le numéro qui lui est associé. Ainsi, comme plusieurs élèves peuvent se trouver dans un lycée, en fusionnant les deux niveaux d'informations, les informations concernant un lycée donné se répètent devant tous les élèves appartenant à ce lycée.

Tout comme pour le précédent cas de figure, il est impératif que la variable captant le code du lycée se retrouve dans les deux bases. Le code du lycée, étant présent dans les deux, il permet de faire correspondre à chaque élève de la première base, les informations concernant son lycée et qui proviennent de la seconde base. Donc, assurez-vous bien que votre variable d'identification du lycée est bien présente dans les deux bases à fusionner.

Que ça soit dans l'un ou cas, la commande pour faire la fusion de tables avec des variables différentes est MERGE. Cette commande doit impérativement être associée avec la commande BY dans le cas d'une fusion sur clé d'indentification. L'exemple ci-

dessous illustre l'application de la commande merge sur deux MYDATA1 et MYDATA2 pour former MYDATA3.

```
DATA MYDATA3;
MERGE MYDATA1 MYDATA2;
BY code_individu ;
RUN;
```

Ici les tables MYDATA1 et MYDATA2 ont été fusionnées en utilisant la clé d'identification commune code_individu. Cependant, lorsque les deux tables n'ont pas de clé d'identification commune et qu'il y a ait une correspondance directe entre les observations c'est-à-dire que les observations sont situées à la même position alors, on peut utiliser merge sans clause BY. Toutefois, ce choix doit effectuer avec une très grande précaution.

I.4.1.3. Fusion de tables avec observations et variables différentes (merge avec sélection des observations selon l'origine)

Il arrive souvent que les deux bases que nous souhaitons fusionner sur des observations contiennent des variables différentes. De même, il arrive de vouloir fusionner deux bases sur des variables mais dont certaines observations sont présentes dans l'une des bases ne sont pas présentes dans l'autre base. L'exemple le plus complexe de cette situation peut être le suivant :

Base A + Base B

id	Var1	Var2	Var3
1			
2			
3			
4			

id	Var4	Var5
1		
2		
3		
5		

Dans cet exemple, nous souhaitons fusionner la base A et la base B. L'individu n° 4 est absent dans la base B, tandis que l'individu n°5 est absent dans la base A.

Pour fusionner ces deux bases, nous combinons à la fois une fusion sur les observations et une fusion sur les variables. Néanmoins, il faut savoir que lorsque la fusion sur observations est réalisée en même temps que la fusion sur les variables, c'est la fusion sur les variables qui devient prioritaire. En d'autres termes, lorsqu'on veut fusionner la base A et la base B, on appliquera toujours une fusion sur les

variables (merge). La fusion sur les observations s'y adapte automatiquement. Toutefois, il faut noter que dans la base finale obtenue, les observations absentes dans l'une des bases prennent des valeurs manquantes sur les variables venant de la base où ces observations sont absentes. Par exemple, l'observation n°5 qui vient uniquement de la base B prend une valeur manquante sur les variables Var1, Var2 et Var3 dans la base finale. Tandis que l'observation n° 4 qui vient uniquement de la base A prend une valeur manquante sur les variables Var4 et Var5 dans la base finale. Schématiquement, cette situation se présente comme suit :

Base A + Base B = Base C

id	Var1	Var2	Var3
1			
2			
3			
4			

id	Var4	Var5
1		
2		
3		
5		

id	Var1	Var2	Var3	Var5	Var6
1					
2					
3					
4				.	.
5	.	.	.		

En pratique, pour fusionner la base A et la base B sous SAS, on utilise la commande MERGE en spécifiant dans l'option IN quelles observations on veut garder dans la table finale. Les lignes de commande ci-dessous fournissent quelques exemples d'application de merge avec sélection d'observations.

```
/* Observations se trouvant dans les deux tables + celles venant uniquement de la
table1 */
DATA table3;
MERGE table1 (IN=A)
table2(IN=B);
IF A ;
BY code_indiv ;
RUN;
```

```
/* Observations se trouvant dans les deux tables + celles venant uniquement de la
table 2 */
DATA table3;
MERGE table1 (IN=A)
table2(IN=B);
IF B ;
BY code_indiv ;
RUN;
```

```
/* Observation se retrouvant dans les deux tables à la fois*/
DATA table3;
MERGE table1 (IN=A)
table2(IN=B);
IF B and  A ;
BY code_indiv ;
RUN;

/* Observation venant uniquement de la table 1*/
data table3;
merge table1 (in=A)
table2(in=B);
if A and not B ;
BY code_indiv ;
RUN;

/* Observation venant uniquement de la table 2 */
data table3;
merge table1 (in=A)
table2(in=B);
if B and not A ;
BY code_indiv ;
RUN;

/** sans distinction sur l'origine des observations  */
data table3;
merge table1 (in=A)
table2(in=B);
BY code_indiv ;
run;
```

I.4.2. Reformatage d'une table de données

On distingue deux principaux types de reformatage de table de données. La première consiste à transposer la table de données de sorte que les individus se retrouvent en colonnes et les variables en lignes. Ce premier type de reformatage est une transposition des données. Le second type de reformatage consiste à éclater sur plusieurs colonnes les valeurs d'une variable initialement renseignées dans une seule colonne. Il peut aussi s'agir de regrouper dans une seule colonne les valeurs d'une variable éclatées sur plusieurs colonnes. Ce second type de reformatage est appelée RESHAPE. Lorsqu'il s'agit d'éclater sur plusieurs colonnes les valeurs d'une variable initialement renseignées dans une seule colonne, on parle de RESHAPE WIDE et

lorsqu'il s'agit de de regrouper dans une seule colonne les valeurs d'une variable éclatées sur plusieurs colonnes, on parle de RESHAPE LONG.

I.4.2.1. Transposition d'une table de données

La commande ci-dessous transpose la table MYDATA, en créant une nouvelle table nommée MYDATA_TRANSP.

```
PROC TRANSPOSE DATA=MYDATA
PREFIX= id
NAME = mesvar
OUT = mydata_transp ;
VAR SEXE AGE REV R1-R3 ;
ID id;
RUN;
```

La commande ID permet de numéroter les variables en colonne en utilisant les identifiants à partir de la variable id. La commande PREFIX ajoute le préfixe « id » aux numéros crées par la commande ID. La commande NAME permet d'attribuer un nom à la première colonne où apparaissent les noms des variables transposées.

Remarque :

La transposition peut s'avérer très utile dans certaines situations. Par exemple, on sait qu'il n'est pas facile de trouver une fonction SAS permettant de calculer sur une colonne la moyenne, le min, le max, etc… On peut alors préférer transposer les données et utiliser la fonction mean(), min() ou max() pour faire le calcul nécessaire en utilisant les colonnes formées par les individus. Ensuite, on peut récupérer ces valeurs comme étant les valeurs recherchées.

I.4.2.2. Reformatage long et large : Reshape WIDE et Reshape long

I.4.2.2.1. Du format long au format wide : reshape wide

Pour effectuer le reshape de long vers wide, considérons la table ci-dessous qui renseigne les revenus sur trois ménages (1,2 et 3) observés sur trois ans (1996, 1997 et 1998).

```
DATA MYLIB.longdata ;
  INPUT idmen annee revenu ;
CARDS ;
1 1996 40000
1 1997 40500
```

```
1 1998 41000
2 1996 45000
2 1997 45400
2 1998 45800
3 1996 75000
3 1997 76000
3 1998 77000
;
RUN ;
```

Ici les données se présentent sous format long puisque le revenu du ménage est renseigné dans la même colonne (sur les trois années).

On souhaite mettre les données sous format *wide* de sorte que le tableau se présente comme suit :

idmen	Annee1996	Annee1997	Annee1998
1	40000	40500	41000
2	45000	45400	45800
3	75000	76000	77000

Dans cette situation, puisque le numéro du ménage se répète plusieurs fois dans le format long, on ne peut pas utiliser PROC TRANSPOSE, il faut utiliser la transformation en reshape WIDE. Les lignes de commande ci-dessous décrivent les étapes de la transformation.

```
PROC SORT DATA=longdata  ;  BY idmen ; RUN ;
DATA widedata ;
 SET longdata ;
 BY idmen ;
 KEEP idmen revenu1996 -revenu1998 ;
 RETAIN revenu1996 -revenu1998 ;
 ARRAY liste(1996:1998) revenu1996 -revenu1998 ;
 IF first.idmen THEN
 DO;
  DO i = 1996 to 1998 ;
   liste( i ) = . ;
  END;
 END;
 liste( annee ) = revenu ;
 IF last.idmen THEN OUTPUT ;
RUN;
```

NB :

L'exemple ci-dessous n'est applicable qu'au cas d'une seule variable. Pour effectuer l'opération sur plusieurs variables, il faut mieux effectuer variable par variable et ensuite merger les tables finales obtenues. Lorsque le nombre de variable est élevé, on peut construire un programme macro en construisant une boucle (Nous verrons la construction des macros dans le chapitre 3).

I.4.2.2.2. Du format wide au format long : reshape long

Le reshape long consiste à reformater le tableau du format wide (où une seule variable est éclatée sur plusieurs colonnes) vers le format long (où la variable est renseignée sur une seule colonne). Dans l'exemple présenté ci-dessus, il s'agit de transformer la table suivante :

idmen	revenu1996	revenu 1997	revenu 1998
1	40000	40500	41000
2	45000	45400	45800
3	75000	76000	77000

de sorte à obtenir une table qui se présente sous la forme suivante

idmen	annee	revenu
1	1996	40000
1	1997	40500
1	1998	41000
2	1996	45000
2	1997	45400
2	1998	45800
3	1996	75000
3	1997	76000
3	1998	77000

Pour effectuer la transformation du format wide au format long, on peut se servir la commande PROC TRANSPOSE en y ajoutant quelques transformations supplémentaires. Les lignes de commandes ci-dessous transforment la table de revenu du format wide (widedata) au format long (longdata).

```
PROC TRANSPOSE data=widedata
PREFIX= idmen
NAME = anneevar
OUT =longdata ;
VAR revenu1996-revenu1998 ;
ID idmen;
BY idmen;
RUN;
/* Création des variables année et revenu */
DATA longdata(KEEP=idmen annee revenu) ; RETAIN idmen annee revenu; set
longdata;
/* Extraire les 4 chiffres de l'année de anneevar */
annee=substr(anneevar,7,4);
/* Conversion en numérique */
new_annee=input(annee,10.0);
DROP annee;
RENAME new_annee=annee;
/* regroupement des valeurs des revenus en une seule variable*/
revenu=max(of id1-id3)/* aller jusqu'au nombre maximal id*/;
RUN;
```

La table ainsi obtenu se présente sous format long.

NB :

L'exemple ci-dessous n'est applicable qu'au cas d'une seule variable. Pour effectuer l'opération sur plusieurs variables, il faut répéter l'opération sur chaque variable et ensuite merger les tables finales obtenues. On peut aussi penser à construire un programme macro pour réaliser une boucle (Voir chapitre 3 pour la construction des programmes macro).

I.5. Gestion des tables de données et des librairies

Plusieurs opérations entrent dans le cadre du management des tables de données. Il s'agit notamment de la description du contenu de la table (les variables et leurs attributs), de l'affichage et la modification des données, etc... Il existe à cet effet plusieurs commandes que nous allons passer en revue dans cette section.

I.5.1. Décrire le contenu d'une table

Pour décrire le contenu de la table, on utilise la commande PROC CONTENTS. L'exemple ci-dessous est une illustration.

```
PROC CONTENTS DATA= MYDATA; RUN ;
```

Il faut signaler qu'on peut ajouter OUT à cette commande afin de stocker dans une nouvelle la liste des variables contenues dans la table. Cela peut s'avérer utile dans certaines situations. Par exemple lorsqu'on veut récupérer la liste des variables et les récupérer dans une macro-variable pour d'autres utilisations. On reviendra sur ces aspects plus tard. Pour exporter la liste des variables, on fait :

```
PROC CONTENTS DATA= MYDATA OUT=LISTE_VARS(KEEP = NAME); RUN ;
```

I.5.2. Visualiser le contenu d'une table sous forme de fenêtre de données

Pour afficher le contenu d'une table sous forme de fenêtre de données, on utilise la commande PROC FSBROWSE (voir exemple ci-dessous).

```
PROC FSBROWSE DATA= MYDATA; RUN ;
```

Notons tout de même que lorsque l'on veut afficher les données dans la fenêtre de résultats, on utilise la commande PROC PRINT comme suit :

```
PROC PRINT DATA= MYDATA; RUN ;
```

Dans cette procédure, on peut afficher un nombre d'observations souhaitable en ajoutant les options FIRSTOBS et OBS. Dans l'exemple suivant, on affiche 50 observations à commencer par la première ligne de données (première observation dans la table).

```
PROC PRINT DATA= MYDATA(FIRSTOBS=1  OBS=50); RUN ;
```

I.5.3. Afficher le contenu d'une table pour modification

Pour afficher le contenu de la table pour modification, on utilise la commande PROC FSEDIT comme suit :

```
PROC FSEDIT DATA= MYDATA; RUN ;
```

I.5.4. Décrire les attributs d'une librairie

Pour afficher les attributs d'une librairie SAS, on utilise la commande PROC DATASETS. L'exemple ci-dessous affiche la liste des tables contenues dans la librairie MYLIB.

```
PROC DATASETS LIB=MYLIB; QUIT;
```

En ajoutant l'instruction CONTENTS avec le nom des tables, on obtient alors la description des contenues des tables telle que présenté dans PROC CONTENTS. (Voir exemple ci-dessous).

```
PROC DATASETS LIB=MYLIB;
CONTENTS DATA=MYDATA ;
 QUIT;
```

On pouvait également ajouter les options OUT et OUT2 pour stocker les informations sur la table MYDATA dans une nouvelle table telle qu'effectuée par l'option OUT de la commande PROC CONTENTS.

```
PROC DATASETS LIB=MYLIB;
CONTENTS DATA=MYDATA OUT=attr1 OUT2=attr2;
 QUIT;
```

La table attr1 contient la liste des variables et leurs caractéristiques et la table attr2 contient les informations sur les index et les contraintes d'intégrité.

Notons aussi qu'on peut afficher les contenues de toutes les tables en faisant DATA=_all_ comme ci-dessous.

```
PROC DATASETS LIB=MYLIB;
CONTENTS DATA=_all_ ;
 QUIT;
```

I.5.5. Supprimer une table dans la librairie

Pour supprimer une table dans la librairie, on utilise la commande PROC CONTENTS avec l'instruction DELETE telle que décrite ci-dessous.

```
PROC DATASETS LIB=MYLIB  MEMTYPE=DATA;
DELETE TAB1 TAB2;
RUN;
QUIT;
```

Cette procédure supprime les deux tables TAB1 et TAB2 dans la librairie MYLIB.

I.5.6. Sauver une table (et supprimer les autres) dans la librairie

Pour sauver une table (de la suppression) dans la librairie, on utilise la commande PROC CONTENTS avec l'instruction SAVE telle que décrite ci-dessous.

```
PROC DATASETS LIB=MYLIB  MEMTYPE=DATA;
SAVE TAB1 TAB2;
RUN;
QUIT;
```

Cette procédure sauvegarde les deux tables TAB1 et TAB2 en supprimant toutes les autres tables existantes dans la librairie MYLIB.

I.5.7. Vider le contenu d'une librairie

Pour vide le contenu d'une librairie, on utilise la commande PROC CONTENTS avec l'instruction KILL telle que décrite ci-dessous.

```
PROC DATASETS LIB=MYLIB KILL NOLIST  MEMTYPE=DATA; RUN; QUIT;
```

Cette commande supprime toutes les tables de données présentes dans la librairie MYLIB.

Pour supprimer tous les attributs d'une librairie et vider entièrement son contenu, on utilise le mot clé MEMTYPE=ALL. Ainsi, on a :

```
PROC DATASETS LIB=MYLIB KILL NOLIST  MEMTYPE=ALL; RUN; QUIT;
```

I.5.8. Quelques informations utiles dans une session SAS

Cette section présente quelques informations utiles lors de la phase du traitement et de l'organisation des données mais aussi en phase d'analyses.

I.5.8.1. Etape DATA sans nécessité de création de table

Lors d'une étape DATA où il n'est pas nécessaire de créer ou modifier une table de données, on peut utiliser la commande DATA _null_. En effet, supposons que l'on veuille simplement afficher le résultat d'un calcul obtenu par modification d'une table et que l'on n'ait pas besoin de créer une table en sortie. Alors, on utilise la variable automatique _null_ pour indiquer qu'aucune table ne sera créée. On peut simplement décider d'ajouter à l'intérieur de l'étape l'instruction PUT affichera dans la fenêtre log le résultat obtenu. L'exemple ci-dessous affiche dans le fichier Log les noms des individus présents dans les deux tables TAB1 et TAB2.

```
DATA _null_;
SET TAB1 (in=A) TAB2 (in=B);
IF A AND B THEN PUT nom;
RUN;
```

Comme cette liste est une information indicative, pour laquelle il n'est pas nécessaire que de créer une table contenant ces noms.

I.5.8.2. Utilisation d'une clause IF THEN ou d'une WHERE dans une étape DATA et dans une procédure PROC

Dans une étape DATA, on peut utiliser la clause IF THEN ou la clause WHERE pour traduire une condition. Par contre dans une procédure définie par PROC on n'utilise que la clause WHERE.

I.5.8.3. Structuration des commandes et des instructions

Dans SAS, chaque commande ou instruction commence par un mot-clé (PROC, DATA, VAR, DROP, etc…) et se termine par un point-virgule « ; ».

Beaucoup d'instructions offrent des options. Il faut alors séparer l'instruction de la liste des options qui s'y rattachent par un slash « / ».

Si on enchaîne plusieurs étapes PROC, un seul « run ; » à la fin de toutes suffit pour les compiler toutes correctement. Il faudrait alors que toutes les procédures soient sélectionnées dans la même exécution.

Après certaines procédures, on doit mettre un « quit ; ». C'est le cas avec des procédures qui font appel à des modules particuliers, telles les PROC GCHART et GPLOT ou la PROC SQL par exemple. Si l'on omet le QUIT, la table, ouverte par le module, n'est pas fermée, et cela peut générer des erreurs.

I.5.8.3.1. Les options de base d'une procédure

L'option DATA= : rares sont les PROC qui n'incluent pas dans leur syntaxe la spécification data= pour indique la table données. Toutefois, il faut noter que lorsque cette option est absente dans une proc, la procédure sélectionne automatique la dernière table créée.

L'option NOPRINT : une autre option répandue est l'option NOPRINT, qui demande à ce qu'aucune sortie ne soit imprimée dans l'output. Elle évite de surcharger cette dernière.

I.5.8.3.2. Les instructions de base

Il existe quelques instructions de base qui sont généralement incontournables dans les procédures SAS. Il s'agit notamment des instructions comme VAR, BY, CLASS, etc..

Une procédure PROC, l'instruction VAR permet de préciser sur quelles variables portent les opérations à exécuter. On la retrouve notamment dans les procédures MEANS, UNIVARIATE, CORR… Elle est donc surtout associée à des traitements de variables quantitatives. Ainsi, elle n'apparaît pas dans la PROC FREQ (qui utilise l'instruction TABLES), ni dans les procédures de modélisation qui utilisent l'instruction MODEL (Nous verrons plus en détails chacun de ces cas).

Lorsqu'une instruction BY *mavariable* ; est spécifiée dans une PROC, cela force la réalisation d'autant d'étapes PROC qu'il y a de modalités de *mavariable*.

Quant à l'instruction CLASS (qu'il ne faut pas confondre avec BY), elle permet de distinguer des sous-groupes à l'intérieur d'une même étape PROC. Par conséquent BY est prioritaire par rapport à CLASS.

L'instruction OUTPUT, qui permet de stocker certains des résultats dans une table.

La syntaxe est alors OUTPUT OUT=table_sortie mots-clés. Ainsi, on précise le nom de la table dans laquelle seront stockés les résultats des statistiques dont les mots-clés sont précisés (liste mots-clés fournit dans la description de la procédure). Notons qu'il existe de nombreuses options du type out= qui ont le même but mais sont adaptées à un cas précis.

On retrouvera aussi fréquemment deux instructions: l'instruction WHERE qui sert à sélectionner les observations répondant à une condition définie et l'instruction FORMAT qui sert à appliquer un format à certaines des variables sélectionnées.

I.5.8.4. Les valeurs manquantes dans SAS

Dans SAS, la valeur manquante d'une variable numérique est noté par missing ou par un point «**.**». Pour une variable en caractères la valeur manquante est matérialisée par un espace " ".

Pour ce qui concerne les variables numérique la valeur manquante est la plus petite valeur de nombres SAS (tandis que pour d'autres logiciels comme STATA par exemple, la valeur manquante représente +l'infini). Cette distinction peut s'averer très utile dans certaines situation où par exemple on spécifie une condition avec l'inégalité <. Par exemple, supposons qu'on veuille créer une variable binaire nommée jeune définie à partir l'âge des individus qui prend 1 lorsque l'âge est inférieur à 24 ans et 0 si l'âge est supérieur à 24. Si on ne prend pas la précaution d'exclure de ce calcul les individus qui ont un âge manquant, ils seront

systématiquement classés parmi les jeunes. La bonne démarche de création de la variable binaire est la suivante :

```
DATA MYDATA; SET MYLIB.MYDATA;
IF AGE<=24 and AGE^=. THEN JEUNE=1;
IF AGE>25 THEN JEUNE=0;
RUN;
```

I.5.8.5. Définition des options générales d'une session SAS

A l'entame d'une session est souvent utile de définir des options générales qui restent actifs durant toute la session. Ces options concernent à la fois la mise en page des sorties, la gestion des commentaires, des notes, des titres, l'affichage des dates.

Ces options sont généralement définies en début du programme. Voici-ci ci-dessous quelques-unes de ces options et la manière de les définir :

```
/*****************************/
OPTIONS
NUMBER /*NONUMBER*/
PAGENO=MIN
CENTER /* NOCENTER  */
LEFTMARGIN=1.5cm
RIGHTMARGIN=1.5cm
PAGESIZE=100
LINESIZE=64
LINESIZE= 88 PAGESIZE= 64  /** page en portrait **/
LINESIZE= 179 PAGESIZE= 65 /** page en paysage **/
ORIENTATION=PORTRAIT /*| LANDSCAPE | REVERSEPORTRAIT | REVERSELANDSCAPE  */
DATE /*NODATE */
DATESTYLE= DMY /* MDY  | YMD |   LOCALE */
DETAILS  /* NODETAILS */
ERRORS=0
FIRSTOBS=1
OBS= MAX
/*MISSING='NA'*/
LABEL /* NOLABEL */
GSTYLE /*| NOGSTYLE */
NOTES  /*NONOTES */
QUOTELENMAX /*NOQUOTELENMAX */
REPLACE /*NOREPLACE */
REUSE=YES  /* NO */
S=MAX
S2=MAX
SORTSIZE=MAX
;
/*****************************/
```

Les détails sur chacune de ces options peuvent être consultés dans la base de connaissance de SAS.

Il faut noter qu'en spécifiant ces options en début de programme, SAS initialise la session en fixant les options par défaut. Néanmoins, ces options par défaut peuvent à tout moment être modifiées par la suite à l'intérieur du programme ou à l'intérieur d'une PROC.

I.5.8.6. Exportation des résultats SAS vers un document de type Microsoft WORD (.rtf)

La commande ci-dessous permet d'exporter les résultats d'une PROC UNIVARIATE vers un fichier .rtf nommée Résultats.rtf

```
ODS RTF FILE= "C:\Résulats.rtf";
PROC UNIVARIATE data=mydata; VAR age revenu ; RUN;
ODS RTF CLOSE ;
```

La ligne ODS RTF FILE indique le nom du fichier qui doit accueillir les résultats et marque le début de l'exportation. Tandis que la ligne ODS RTF CLOSE marque la fin de fin de l'exportation des résultats.

Malheureusement, SAS n'autorise pas la suspension de l'exportation avec ODS RTF. Il faut spécifier toutes commandes à l'intérieur de la commande ODS. Mais lorsqu'on ne veut pas exporter tous les résultats du programme, et qu'on veut simplement exporter les résultats de plusieurs bouts de commandes non contiguës, il faut alors déclarer à chaque fois l'exportation et créer par conséquent autant de fichiers de résultats que de bouts de programmes.

I.5.8.7. Exportation des résultats SAS vers un document HTML

Pour exporter les résultats vers un document .HTML, on utiliser la commande ODS HTML telle qu'appliquée dans l'exemple ci-dessous.

```
ODS HTML BODY= "C:\Résulats.html";
PROC UNIVARIATE data=mydata; VAR age revenu ; RUN;
ODS LISTING CLOSE ;
```

Dans cet exemple, ODS HTML exporte les résultats de la PROC UNIVARIATE sur les variables âge et revenu de la table MYDATA. La sortie HTML est stockée ici sur le disque C.

CHAPITRE II : TRAITEMENT ET ORGANISATION DES DONNEES AVEC LE LANGAGE SQL

Le rôle premier d'un langage SQL est de faire des requêtes afin d'afficher les résultats. Une requête est un ensemble d'instructions permettant d'accéder à une information donnée à partir d'une ou de plusieurs tables. L'information obtenue peut être simplement affichée ou bien stockée sous forme d'une nouvelle table de données. Dans le premier cas, il s'agit d'une *requête-affichage* et dans le second cas, il s'agit d'un *requête-création*.

Le langage SQL de SAS défini par la procédure PROC SQL offre la possibilité de réaliser à la fois une requête-affichage, une requête-création mais aussi une requête modification. Cette dernière consiste à modifier les informations contenues dans une table sans nécessairement avoir besoin de les afficher, ni de créer une nouvelle table. A ce propos on peut signaler que le fait que la procédure PROC SQL permet à la fois de faire des requêtes-affichage, des requêtes-création et des requêtes modification, elle peut jouer le rôle d'une DATA STEP classique et mais celle d'une PROC ordinaire telle que PROC PRINT.

Dans ce chapitre, nous nous efforçons de présenter avec pédagogie toutes les étapes de mise en œuvre d'une requête SQL à travers des exemples concrets.

II.1. Définition d'un cadre de référence pour les requêtes SQL

Elaborer un cadre de référence pour une requête SQL, c'est identifier toutes les entités d'une base de données susceptibles d'intervenir dans la formulation de la requête. D'une manière générale, chaque entité est représentée par une table de données spécifique permettant d'enregistrer les informations sur cette entité.

Pour élaborer une requête, il faut d'abord bien comprendre la nature de l'information à exploiter mais aussi la ou les table(s) permettant de l''obtenir. Cela nécessite une bonne compréhension de la structure de toutes tables intervenant dans la formulation de la requête (niveau d'information, dimension de la table, type et format des variables, etc…)

Dans ce chapitre, pour étudier les requêtes-affichage et les requêtes-création, nous allons nous focaliser sur une base données dont la structure est décrite ci-après :

Considérons une société commerciale disposant d'une base de données constituée de trois tables: une première table contenant les informations sur les clients nommées CLIENTS, une seconde table contenant les informations sur les commandes passées par les clients nommée COMMANDES et une troisième table contenant les informations sur les produits qui constituent les commandes nommées PRODUITS. Ces trois tables sont définies par les informations suivantes :

CLIENTS { IDCLIENT(num), NOM ($30), PRENOM($30), ANNAISS(num), NUMTEL($15), ADRESSE($50), DEPARTEMENT($40), REGION($40) }

COMMANDES {NUMCMDE(num), IDCLIENT (num), DATECMDE(date), DATELIV(date)}

PRODUITS {NUMCMDE(num), CODEPROD(num), NOMPROD($50), QUANTITE(num), MONTANT_PROD (num) }

La table CLIENTS contient les informations suivantes : IDCLIENT qui représente l'identifiant du client (numérique) ; NOM et PRENOM qui sont des variables en chaînes de caractères de 30 positions maximum représentant respectivement le nom et le prénom du client ; ANNAISS est une variable numérique représentant l'année de naissance du client ; NUMTEL est une variable en caractères de 15 positions maximum représentant le numéro de téléphone du client ; ADRESSE est une variable en caractères de 50 positions représentant l'adresse géographique du client ; Il y a également DEPARTEMENT et REGION qui sont aussi des variables en caractères de 40 positions maximum chacune.

La table COMMANDES contient les informations suivantes : NUMCMDE qui représente le numéro de la commande (clé d'identification unique) représentant un numéro attribué à chaque commande passée par les clients IDCLIENT qui représente l'identifiant du client qui a passé la commande ; DATECMDE qui représente la date à laquelle la commande a été passée ; DATELIV qui représente la date de livraison. NOMPROD qui représente la dénomination du produit

La table PRODUITS recense toutes les informations sur les produits ayant fait objet de la commande. On dénote notamment, NUMCMDE qui représente le numéro de la commande dont le produit fait partie ; CODEPROD qui représente le code du produit ; QUANTITE et MONTANT_PROD qui représentent respectivement la quantité et le montant de la vente (achat) du produit dans la commande.

Pour mieux comprendre la structure de cette base de données, on s'intéresse d'abord à la nature des relations entre les trois tables. On distingue généralement quatre types de relations entre A et B :

- Une relation 1 à n où un élément de la table A peut correspondre n-éléments de la table B. C'est le cas par exemple entre la table CLIENTS et la table COMMANDES. En effet, comme un client peut passer plusieurs commandes alors il existe une relation 1 à n entre la table CLIENTS et la table COMMANDES. De même, il existe une relation 1 à n entre la table COMMANDES et la table PRODUITS car une commande peut contenir plusieurs produits. Ces types de relations sont généralement appelées relation « *père-fils* »

- Une relation n à 1 est l'équivalent d'une relation 1 à n en inversant l'ordre des tables. Il s'agit ici d'une relation « *père-fils* » inversée.

- Une relation 1 à 1 survient lorsqu'un élément de la table A ne peut correspondre qu'à un seul élément de la table B. On retrouve ce cas par exemple lorsqu'on dispose de deux tables sur les clients ; l'une contenant les informations sur le nom, prénom, adresse et numéro de téléphone et l'autre contenant les informations sur l'âge, le sexe, la catégorie socioprofessionnelle. Il existe donc une relation 1 à 1 entre ces deux tables car il n'existe qu'une correspondance unique. Il s'agit ici d'une relation dite « père-père » ou bien une relation « fils-fils » selon le niveau qu'occupe la table dans la hiérarchie de la base de données.

- Une relation n à n entre deux tables A et B est une relation dans laquelle il existe des correspondances multiples entre les éléments des deux tables. Dans ce type de relation, l'ordre de filiation « père-fils » n'est plus unique. Dans une relation n à n les pères peuvent avoir plusieurs fils et les fils peuvent avoir plusieurs pères.

La compréhension du type de relation entre les tables est vitale lors de l'exécution d'une requête SQL.

Dans l'exemple présenté ci-haut, il s'agit d'une relation 1 à n entre la table CLIENTS et la table COMMANDES mais aussi entre la table COMMANDES et la table PRODUITS. La relation entre la table CLIENTS et la table PRODUITS est une relation 1 à n indirect (c'est-à-dire une relation grand-père-fils) car la relation entre les deux tables passe d'abord par une table intermédiaire qui est la table COMMANDES. Dès lors toute requête mettant en relation les informations de la table CLIENTS avec la table

PRODUITS doit passer d'abord par la table COMANDES. Nous reviendrons en détails sur ce passage intermédiaire.

II.2. Les requêtes-affichage

Comme signalé précédemment, une requête-affichage est un ensemble d'instructions permettant d'extraire une information à partir d'une ou plusieurs tables et l'afficher sous forme de résultats.

La structure générale d'une requête affichage est la suivante :

```
PROC SQL ;
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
;
QUIT;
```

- L'instruction **SELECT** permet de sélectionner les informations à afficher.
- L'instruction **FROM** spécifie la ou les tables à partir desquelles les informations doivent être extraites. Lorsque l'instruction porte sur plusieurs tables, les noms de celles-ci doivent être séparés par des virgules.
- La clause **WHERE** spécifie la condition générale que les observations doivent vérifiées pour pouvoir figurer dans l'affichage.
- L'instruction **GROUP BY** permet d'exécuter la procédure et afficher les résultats par sous-groupes définis par les valeurs de la variable spécifiée dans l'instruction. Lorsque cette instruction est définie par plusieurs variables, celle-ci doivent être séparées par une virgule.
- L'instruction **HAVING** est une condition particulière que les observations sélectionnées dans chaque sous-groupe défini par GROUP BY doivent vérifier pour pouvoir figurer dans l'affichage finale. Il faut signaler ici que la clause WHERE est appliquée au premier niveau sur toute la table alors que l'instruction HAVING est appliquée au second niveau à l'intérieur de chaque sous-groupe défini par GROUP BY.
- L'instruction **ORDER BY** permet d'afficher les résultats selon un ordre défini par les valeurs de la variable spécifiée dans l'instruction. Lorsque cette

instruction est définie par plusieurs variables, celle-ci doivent également être séparées par une virgule.

Dans cette section, nous allons étudier deux types de requêtes-affichage : les requêtes-affichage sur une seule table et les requêtes-affichage sur plusieurs tables.

II.2.1. Requête-affichage à partir d'une seule table

II.2.1.1. Affichage d'une information déjà existante dans la table

Une requête affichage est une requête dans laquelle l'extraction d'information n'est pas accompagnée ni par la modification de la table initiale, ni par la création d'une nouvelle table. Il s'agit simplement d'afficher une information demandée. Par exemple, en considérant la base de données de la société commerciale décrite précédemment, on souhaite afficher le nom, prénom, le numéro de téléphone de tous les clients nés en 1990. Alors, sachant que l'information sur l'année de naissance est renseignée dans la table CLIENTS, la structure de requête est la suivante :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE ANNAISS=1990
;
QUIT;
```

La clause WHERE peut être définie avec plusieurs critères comme suit :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE ANNAISS>1990 AND ANNAISS<=2000
;
QUIT;
```

Cette commande affiche les informations sur les clients nés entre 1990 et 2000.

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE NOM 1990 AND ANNAISS<=2000
;
QUIT;
```

II.2.1.2. Affichage d'une information calculée à partir des variables pré-existantes dans la table

Pour afficher par requête une information calculée à partir des variables disponible dans la table, on spécifie sa formule de calcul dans la clause SELECT en indiquant son nom avec le mot clés AS. L'exemple ci-dessous calcule l'âge des clients nés à partir de 1990 en considérant 2016 comme l'année de réérence :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE
FROM CLIENTS
WHERE ANNAISS>=1990
;
QUIT;
```

Ici, l'âge est calculé en faisant la différence entre 2016 et l'année de naissance. La formule de calcul est 2016-ANNAISS.

II.2.1.3. Afficher les valeurs modifiées d'une variable sans création de nouvelle variable

Affichons par exemple les montants des produits achetés en milliers sans créer une valeur correspondante

```
PROC SQL ;
SELECT NUMCMDE, CODEPROD, NOMPROD, MONTANT_PROD /1000
FROM PRODUITS
;
QUIT;
```

Dans cette requête l'instruction AS est omis alors la variable existante est modifiée.

II.2.1.4. Ajout d'une colonne remplie d'un texte unique ou d'un nombre unique

On peut ajouter à l'affichage des colonnes remplies de la même valeur (soit du texte soit du nombre). L'exemple ci-dessous est une illustration :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE,
"Nous somme en 2016" as texte,
2016 as anneeactuelle
FROM CLIENTS
```

```
WHERE ANNAISS>=1990
;
QUIT;
```

Cette requête ajoute à l'affichage deux colonnes supplémentaires : texte qui est remplie de la phrase « Nous sommes en 2016 » et anneeactuelle remplie du chiffre 2016.

II.2.1.5. Faire disparaitre le nom d'une colonne lors de l'affichage

Par défaut, PROC SQL affiche les noms (ou les libellés) de toutes les variables spécifiées dans la clause SELECT. On peut faire disparaître le libellé d'une variable en ajoutant l'option LABEL= "#". Voir exemple ci-dessous :

```
PROC SQL ;
SELECT NOM LABEL='#', PRENOM LABEL='#', NUMTEL, 2016- ANNAISS AS AGE
FROM CLIENTS
WHERE ANNAISS>=1990
;
QUIT;
```

Cette requête indique que les labels des variables NOM et PRENOM ne doivent pas être affichés dans les résultats. Par contre les noms et prénoms des individus seront bien affichés.

II.2.1.6. Affichage d'une information calculée à partir d'une variable elle-même calculée des variables existantes dans la table

Pour une information calculée à partir des variables elles- mêmes calculées à partir des informations disponibles dans la table, on spécifie sa formule de calcul dans la clause SELECT en indiquant son nom et en y ajoutant les mots clés CALCULATED et AS. L'exemple ci-dessous, calcule l'âge à partir de l'âge en années obtenu en faisant la différence entre 2016 et l'année de naissance ANNAISS sur les clients nés après 1990.

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE, 12*(CALCULATED AGE)
AS AGEMOIS
FROM CLIENTS
WHERE ANNAISS>=1990
;
QUIT;
```

II.2.1.7. Utilisation des fonctions d'agrégation dans une PROC SQL sur table unique

La procédure proc SQL offre plusieurs fonctions permettant d'agréger les informations à un niveau hiérarchique donné en utilisant les fonctions de la statistique mathématique. Parmi ces fonctions on dénombre notamment la somme, la moyenne, la médiane, l'écart-type, la fréquence, etc…. Voici-ci-dessous les fonctions d'agrégations courantes :

- AVG, MEAN (moyenne ou moyenne des valeurs)
- COUNT, FREQ, N (nombre de valeurs non manquantes)
- CSS (somme des carrés corrigée)
- CV (coefficient de variation)
- MAX (Maximum)
- MIN (Minimum)
- NMISS (Nombre de valeurs manquantes)
- PRT (Probabilité d'une valeur absolue supérieure de t de Student)
- RANGE (Plage de valeurs)
- STD (écart-type)
- STDERR (Erreur standard de la moyenne)
- SUM (Somme des valeurs)
- SUMWGT (somme de la variable de pondération)
- T (valeur du t de Student pour tester l'hypothèse que la moyenne de la population est égale à zéro)
- USS (Somme des carrés non corrigée)
- VAR (variance)

L'exemple ci-dessous calcule l'âge moyen, l'âge minimum et maximum des clients :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE,
MEAN(CALCULATED AGE) AS AGE_MOYEN,
MIN(CALCULATED AGE) AS MAX_AGE,
MAX(CALCULATED AGE) AS MIN_AGE
FROM CLIENTS
;
QUIT;
```

Dans cette commande, nous utilisons le mot CALCULATED car la variable AGE doit d'abord être générée à partir de l'année de naissance.

Il faut aussi noter que la moyenne, le minimum et le maximum obtenus sont identiques pour toutes les observations de la table. Donc il y aura répétition de valeurs sur ces nouveaux indicateurs.

On peut vouloir calculer ces moyennes, min et max par sous-groupe (par exemple au niveau région). Pour cela on ajoute la clause GROUP BY comme suit :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE,
MEAN(CALCULATED AGE) AS AGE_MOYEN,
MIN(CALCULATED AGE) AS MAX_AGE,
MAX(CALCULATED AGE) AS MIN_AGE
FROM CLIENTS
GROUP BY REGION
;
QUIT;
```

Lorsqu'on veut trier les résultats affichés par région, on utilise ORDER BY comme suit :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE,
MEAN(CALCULATED AGE) AS AGE_MOYEN,
MIN(CALCULATED AGE) AS MAX_AGE,
MAX(CALCULATED AGE) AS MIN_AGE
FROM CLIENTS
GROUP BY REGION
ORDER BY REGION /* BY REGION DESC*/
;
QUIT;
```

ATTENTION

La commande GROUP BY considère la valeur manquante comme une catégorie à part entière lorsque la variable de groupement contient des valeurs manquantes. Il faut alors penser à exclure ces valeurs manquantes dans la clause WHERE.

Ajout de la condition particulière HAVING :

On peut ajouter la clause HAVING pour restreindre une seconde fois l'affichage des résultats. Mais cette fois, la restriction pour sur les sous-groupes et non sur l'échantillon total. Soit l'exemple ci-dessous :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE,
MEAN(CALCULATED AGE) AS AGE_MOYEN,
MIN(CALCULATED AGE) AS MAX_AGE,
MAX(CALCULATED AGE) AS MIN_AGE
FROM CLIENTS
WHERE ANNAISS>=1990
GROUP BY REGION
HAVING CALCULATED AGE>15
ORDER BY REGION DESC
;
QUIT;
```

Dans cette requête, on sélectionne tous les individus nés après 1990 mais on affiche par région toutes les observations pour lesquelles l'âge calculé est supérieur à 15 et on ordonne les résultats par ordre décroissant des régions.

II.2.1.8. Requête-affichage avec recodage des variables

Pour afficher les valeurs recodées d'une ou des plusieurs variables dans une PROC SQL, on utilise l'instruction CASE combinée avec l'instruction WHEN. Les valeurs recodées d'une variable peuvent être numériques ou en caractères. Les deux exemples ci-dessous calculent et affichent les valeurs recodées de la population des communes (codage numérique et codage en texte).

```
/* Codage numérique */
PROC SQL ;
SELECT   COMMUNE,   NBRE_HABITANTS,
CASE
WHEN NBRE_HABITANTS<= 5000 THEN 1
WHEN (NBRE_HABITANTS> 5000 AND NBRE_HABITANTS<= 10000) THEN 2
WHEN NBRE_HABITANTS> 10000 THEN 3
END AS CAT_HABITANT
FROM COMMUNE_DATA
WHERE NBRE_HABITANTS ^=.;
QUIT ;

/* Codage en texte */
PROC SQL ;
SELECT   COMMUNE,   NBRE_HABITANTS,
CASE
WHEN NBRE_HABITANTS<= 5000 THEN "Moins de 5000 hbts"
```

```
WHEN (NBRE_HABITANTS> 5000 AND NBRE_HABITANTS<= 10000) THEN "Entre
5000 et 10000 hbts"
WHEN NBRE_HABITANTS> 10000 THEN "Plus de 10000 hbts"
END AS CAT_HABITANT
FROM COMMUNE_DATA
WHERE NBRE_HABITANTS ^=.;
QUIT ;
```

Dans chacun des deux types de recodage, on calcule une nouvelle information nommée CAT_HABITANT en utilisant l'information initiale NBRE_HABITANTS. Notons que dans ce calcul, on exclut toutes les communes dont la population est manquante en utilisant la clause WHERE.

Notons aussi que les instructions CASE WHEN se comportent comme une formule classique de calcule de variable. Par conséquent, ils doivent figurer dans la clause SELECT séparée avec les autres éléments par une virgule. De plus le nom de la variable obtenue doit être déclarée à la suite en utilisant l'instruction AS. De par cette propriété, il peut exister plusieurs CASE WHEN dans une même requête (Voir exemple ci-dessous).

```
/* Codage numérique et textuel */
PROC SQL ;
SELECT  COMMUNE,  NBRE_HABITANTS,
CASE
WHEN NBRE_HABITANTS<= 5000 THEN 1
WHEN (NBRE_HABITANTS> 5000 AND NBRE_HABITANTS<= 10000) THEN 2
WHEN NBRE_HABITANTS> 10000 THEN 3
END AS CAT_HABITANT_NUM,
CASE
WHEN NBRE_HABITANTS<= 5000 THEN "Petite commune"
WHEN (NBRE_HABITANTS> 5000 AND NBRE_HABITANTS<= 10000) THEN "
Commune Moyenne "
WHEN NBRE_HABITANTS> 10000 THEN "Grande commune "
END AS CAT_HABITANT_TEXT
FROM COMMUNE_DATA
WHERE NBRE_HABITANTS ^=.;
QUIT ;
```

II.2.1.9. Sélection et affichage de toutes les informations d'une table dans une PROC SQL

Pour sélectionner toutes les informations d'une table dans une PROC SQL, on utilise le symbole étoile (*) dans la clause SELECT. Ce cas est illustré dans la requête suivante :

```
PROC SQL ;
SELECT *
FROM CLIENTS
WHERE ANNAISS>=1990
;
QUIT;
```

Cette requête sélectionne tous les clients nés après 1990 en affichant toutes les informations leurs concernant et qui figurent dans la table CLIENTS.

II.2.2. Requête-affichage à partir plusieurs tables

Pour effectuer une requête-affichage sur plusieurs tables dans une PROC SQL, on spécifie dans la clause FROM les noms de toutes les tables (séparés par des virgules) tout en veillant à adapter la structure de la clause WHERE.

II.2.2.1. Structure des requêtes simples sur plusieurs tables

Avant d'étudier plus en détails ce type de requête, introduisons d'abord un exemple pratique en reprenant l'exemple de la base de données de la société commerciale fournit dans la section 1 de ce chapitre. En effet, on veut afficher le nom, prénom de chaque client ainsi que la date de commande et la date de livraison de chaque commande qu'il a passé. Dans cette situation, le nom et le prénom se trouve dans la table CLIENTS alors que la date de commande et la date de livraison se trouve dans la table COMMANDES. Il faut alors utiliser simultanément les tables afin d'effectuer la requête. Pour cela, on utilise la formulation suivante :

```
PROC SQL;
SELECT  NOM, PRENOM, DATECMDE, DATELIV
FROM CLIENTS, COMMANDES
WHERE CLIENTS.IDCLIENT=COMMANDES.IDCLIENT
;
QUIT;
```

Cette structure se décrit comme suit. Dans l'instruction SELECT, on spécifie toutes les informations souhaitées (NOM, PRENOM, DATECMDE, DATELIV). Dans l'instruction FROM, on indique les tables dans lesquelles se trouvent ces informations. Dans l'instruction WHERE, on spécifie les conditions qui permettent de créer les correspondances entre les tables de la clause FROM. C'est cette instruction qui permet de mettre en connection les différentes tables sollicitées dans la requête. Très généralement cette condition est définie à partir des clés d'identification qui doit se retrouver dans les deux tables. Par exemple, on voit que la variable IDCLIENT se trouve à la fois dans la table CLIENTS mais aussi dans la table COMMANDES. Cela permet ainsi de chercher chaque client dans la table CLIENTS et lui associer toutes les commandes qu'il a passées. L'instruction WHERE est d'une importance capitale lors de la création des correspondances entre les enregistrements des tables et permet de garantir la réussite de la fusion des tables (nous reviendrons plus en détails sur la fusion des tables dans une PROC SQL).

Ici, on peut simplement noter que pour afficher les résultats d'une requête sur plusieurs tables, SAS procède en deux étapes :

- Dans une première étape, on fusionne les tables spécifiées dans la clause FROM pour former une table intermédiaire. Cette fusion est généralement assurée (pas obligatoirement) par une ou des clés d'indentification présentes dans chacune des table en utilisant la clause WHERE. Des conditions supplémentaires peuvent également être ajoutées à la clause WHERE mais qui ne portent pas sur les clés d'identification (ex : AND ANNAISS>1990).

- Dans un second temps, en se basant sur la table intermédiaire obtenue, SAS sélectionne les informations comme s'il s'agissait d'une requête sur table unique.

En analysant ces deux étapes, on peut conclure que la requête-affichage sur plusieurs tables est une combinaison de la fusion des tables et de la requête sur table unique. Dès lors, il apparait très important de comprendre la logique de fusions de tables dans une PROC SQL afin d'étudier plus en détail les requêtes affichage.

II.2.2.2. Fusions des tables avec PROC SQL (jointures de tables)

Dans le langage SQL, la fusion de tables est généralement qualifiée de jointure de tables. Nous avons déjà montré à la section 4 du chapitre 1 qu'il existe trois principaux types de fusions de tables :

- fusion de table de mêmes variables mais d'observations différentes (correspondant à *append*).
- Fusion de tables avec les mêmes observations mais de variables différentes (correspondant *merge*).
- fusion de tables avec observations et variables différentes (équivalent à *merge* avec sélection des observations selon l'origine.

Tous ces trois types de fusions peuvent être réalisés avec PROC SQL en utilisant simplement les instructions SELECT et FROM. Cependant la formulation sera différente selon qu'il s'agisse d'une union de tables (append) ou d'une jointure de tables (merge). Et quand il s'agit d'une jointure les résultats diffère considérablement selon que la clause WHERE est spécifiée non.

D'une manière générale, on distingue quatre grands cas dans la fusion de tables avec PROC SQL :

- Union de tables
- Jointure de tables avec clause WHERE et une restriction complète sur l'origine des observations
- Jointure de tables avec clause WHERE et une restriction partielle sur l'origine des observations
- Jointure de tables avec clause WHERE et sans restriction sur l'origine des observations
- Jointure de tables sans clause WHERE

II.2.2.2.1. Union de tables (append)

L'union de table concerne des tables qui ont les mêmes variables mais des observations différentes. Elle consiste en la juxtaposition de deux ou plusieurs tables. Elle correspond à l'instruction SET d'une DATA STEP (append). L'union de deux tables A et B s'obtient en indiquant dans la clause FROM l'instruction UNION JOINT. Elle se formule comme suit :

```
PROC SQL;
SELECT *
FROM A UNION JOINT B
;
QUIT;
```

II.2.2.2.2. Jointure de tables avec clause WHERE et restriction complète sur l'origine des observations

Une jointure de tables avec clause WHERE et une restriction complète sur l'origine des observations est une fusion dans laquelle la table finale obtenue ne contient que les observations qui répondent strictement aux conditions définie par les clés primaire dans la clause. Dès lors toutes les observations ne satisfaisant pas à ces conditions sont exclues. Par exemple, pour fusionner la table CLIENTS et la table COMMANDES, on peut utiliser la formulation de requête suivante :

```
PROC SQL;
SELECT *
FROM CLIENTS, COMMANDES
WHERE CLIENTS.IDCLIENT=COMMANDES.IDCLIENT
;
QUIT;
```

Dans cette jointure, tous les clients pour lesquels il n'y a pas de commandes correspondantes (c'est à dire tous les clients dont l'identifiant n'apparaît pas dans la table COMMANDES) ne sont pas sélectionnées. De mêmes toutes les commandes qui n'ont pas correspondant dans la table CLIENTS (c'est-à-dire toutes les commandes passées par des clients qui ne figurent plus dans la table CLIENTS pour des raisons diverses) ne sont pas aussi sélectionnées. Car pour chacun des cas la condition CLIENTS.IDCLIENT=COMMANDES.IDCLIENT n'est pas respectée.

Noter aussi qu'on peut joindre la table COMMANDES avec la table PRODUITS en utilisant une de tables avec une restriction complète sur l'origine des observations, Pour cela, on utilise la formulation suivante :

```
PROC SQL;
SELECT *
FROM COMMANDES, PRODUITS
WHERE COMMANDES. NUMCMDE = PRODUITS. NUMCMDE
;
QUIT;
```

On peut aussi utiliser ce type de formulation pour fusionner la table CLIENTS et la table PRODUITS. Toutefois, avant de réaliser cette fusion, il faut d'abord passer par la table COMMANDES car il n'y a pas de clé d'identification commune entre la table CLIENTS et la table PRODUITS. La connection doit être faite à partir de NUMCMDE qui se trouve dans la table COMMANDES et la table PRODUITS. Ensuite à partir de IDCLIENT qui se trouve dans la table COMMANDES et la table CLIENTS. La structure de la requête sera donc la suivante :

```
PROC SQL;
SELECT *
FROM CLIENTS, COMMANDES, PRODUITS
WHERE CLIENTS.IDCLIENT=COMMANDES.IDCLIENT AND
    COMMANDES. NUMCMDE = PRODUITS. NUMCMDE
;
QUIT;
```

En réalité cette requête joint les trois tables CLIENTS, COMMANDES et PRODUITS. Elle conserve toutes les informations provenant des trois tables. Pour éviter une telle surcharge on peut spécifier les noms des variables d'intérêt dans la clause SELECT.

Remarque :

Dans une jointure de tables avec clause WHERE et restriction complète sur l'origine des observations, on peut introduire dans la clause FROM la clause INNER JOIN tout en remplaçant la clause WHERE par ON. L'exemple ci-dessous reprend la jointure de la table CLIENTS avec la table COMMANDES en utilisant la clause INNER JOIN :

```
PROC SQL;
SELECT *
FROM CLIENTS INNER JOIN COMMANDES
ON CLIENTS.IDCLIENT=COMMANDES.IDCLIENT
;
QUIT;
```

La clause INNER JOIN indique qu'il s'agit d'une fusion où l'on ne considère que les observations présentes dans les deux tables c'est celle respectant strictement la condition. Ici, la condition n'est plus annoncée par le mot clé WHERE mais plutôt par le mot ON.

II.2.2.2.3. Jointure de tables avec clause WHERE et restriction partielle sur l'origine des observations

Pour mettre en œuvre ce type de requête, considérons deux table A et B dont les correspondances entre les enregistrements peuvent être établies en utilisant une clé d'identification nommée ID. Supposons également qu'il existe des observations présentes dans la table A mais qui ne sont pas présentes dans la table B. De même supposons qu'il existe des observations dans la table B mais qui n'existent pas dans la table A. Pour joindre ces deux tables, on peut utiliser la méthode de jointure avec la clause WHERE avec une restriction partielle sur l'origine des observations. La restriction est dite « partielle » car on souhaite garder dans la table finale non seulement les observations se trouvant dans les deux tables mais aussi celles provenant uniquement de l'une des deux tables.

Pour réaliser ce type de jointure, on utilise la requête OUTER JOINT. Par exemple, pour fusionner la table A et B en gardant dans la table finale les observations venant uniquement de la table A, on formule la requête comme suit :

```
PROC SQL;
SELECT *
FROM A LEFT JOIN B
ON A.ID= B.ID
;
QUIT;
```

Et pour joindre la table A et B en gardant dans la table finale les observations venant uniquement de la table B, on formule la requête comme suit :

```
PROC SQL;
SELECT *
FROM A RIGHT JOIN B
ON A.ID= B.ID
;
QUIT;
```

Il faut noter que la requête OUTER JOINT est une requête INNER JOIN à laquelle on ajoute les informations provenant d'une seule table sélection l'instruction LEFT ou RIGHT.

Dans une formulation INNER ou OUTER, la clause WHERE est remplacée par le mot clé ON.

II.2.2.2.4. Jointure de tables avec clause WHERE et sans restriction sur l'origine des observations

Dans une jointure de table avec clause WHERE mais sans restriction sur l'origine des observations, on garde dans la table finale toutes les observations provenant des deux tables mais aussi celles qui proviennent d'une table uniquement. Il s'agit ici d'une jointure INNER à laquelle on ajoute toutes les observations qui n'ont pas de correspondant dans l'autre table. En prenant l'exemple de deux tables A et B, cette requête se formule comme suit :

```
PROC SQL;
SELECT *
FROM A FULL JOIN B
ON A.ID= B.ID
;
QUIT;
```

II.2.2.2.5. Jointure de tables sans clause WHERE

La jointure sans clause WHERE entre deux tables A et B est une jointure dans laquelle chaque observation de la table A est mise en correspondance avec toutes les observations de la table B de sorte que dans la table finale chaque observation provenant de la table A se trouve démultipliée autant de fois qu'il y a d'observations dans la table B et chaque observation provenant de la table B se trouve démultipliée autant de fois qu'il y a d'observations dans la table A. Ce type de fusion est appelée jointure cartésienne où une correspondance de type Kronecker est établie entre les observations des deux tables.

Pour réaliser ce type de jointure, on utilise la formulation suivante :

```
PROC SQL;
SELECT *
FROM A, B
;
QUIT;
```

Cette requête sélectionne non seulement toutes les informations provenant de la table A et B mais aussi créer une correspondance cartésienne entre les observations.

On peut aussi adopter une spécification JOINT d'une jointure sans clause WHERE. Pour cela, il suffit l'instruction CROSS JOINT entre les noms des deux tables comme suit :

```
PROC SQL;
SELECT *
FROM A CROSS JOINT B
;
QUIT;
```

II.2.2.2. Requêtes-affichage sur plusieurs tables : calcul de nouvelles variables et utilisation des fonctions d'agrégation

Comme cela a été signalé un peu plus, une requête affichage sur plusieurs est une requête qui se réalise en deux étapes. Dans la première étape, SAS fait une jointure des tables spécifiées dans la clause FROM en tenant compte du type de jointure indiqué (INNER, OUTER LEFT, OUTER RIGHT, OUTER FULL, ou jointure cartésienne). Dans la seconde étape, la table obtenue après jointure est utilisée pour calculer l'information demandée. Toutes ces deux étapes sont réalisées en mode batch c'est-à-dire sans être visible en OUTPUT. Seule le résultat final est affiché. En se basant sur cette logique la requête-affichage de variables calculées sur plusieurs tables n'est que la juxtaposition d'une jointure de tables et d'une requête-affichage de variables calculées sur table unique. Les exemples ci-dessus illustrent l'application de ce type de requête.

- Afficher le nom, prénom et la durée moyenne (en jours) entre la date de commande et la date de livraison par client :

```
PROC SQL ;
SELECT IDCLIENT, NOM, PRENOM, MONTANT_PROD,
2016- ANNAISS AS AGE, DATELIV- DATECMDE AS DUREE,
MEAN(CALCULATED DUREE) AS DUREE_MOYENNE,
FROM CLIENTS, COMMANDES
WHERE CLIENTS.IDCLIENT =COMMANDES.IDCLIENT
GROUP BY IDCLIENT
;
QUIT;
```

- Afficher le nom, prénom, âge et montant total des commandes par client :

```
PROC SQL ;
SELECT IDCLIENT, NOM, PRENOM, MONTANT_PROD,
2016- ANNAISS AS AGE,
SUM(MONTANT_PROD) AS TOTAL_CMDE,
FROM CLIENTS, COMMANDES, PRODUITS
WHERE CLIENTS.IDCLIENT =COMMANDES.IDCLIENT AND
```

```
    COMMANDES. NUMCMDE = PRODUITS. NUMCMDE
GROUP BY IDCLIENT
;
QUIT;
```

II.3. Les requêtes-création de tables

Une requête création de table est une extension d'une requête affichage en ajoutant à la formulation de celle-ci l'instruction CREATE TABLE…AS. La structure générale d'une requête-création de table est la suivante :

```
PROC SQL ;
CREATE TABLE NOMTABLE AS
SELECT
FROM
WHERE
GROUP BY
HAVING
ORDER BY
;
QUIT;
```

On remarque alors que la seule différence de structure entre une requête affichage et une requête création est l'instruction CREATE TABLE… AS. Par conséquent toutes propriétés d'une requête affichage sont également valables pour une requête création. Les exemples ci-dessous sont des illustrations.

II.3.1. Requête-création à partir d'une seule

II.3.1.1. Requête sans calcul de nouvelles variables

Exemple :

```
PROC SQL;
CREATE TABLE TAB1 AS
SELECT  NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE ANNAISS>1990
;
QUIT;
```

II.3.1.2. Requête avec calcul de nouvelles variables

Exemple :

```
PROC SQL ;
CREATE TABLE TAB2 AS
SELECT NUMCMDE, CODEPROD, NOMPROD,
MONTANT_PROD /1000 AS MONTANT2
FROM PRODUITS
;
QUIT;
```

II.3.1.3. Requête avec utilisation des fonctions d'agrégation

Exemple :

```
PROC SQL ;
CREATE TABLE TAB_PRODUITS AS
SELECT NUMCMDE, CODEPROD, NOMPROD,
SUM(MONTANT_PROD) AS TAB_PRODUITS
FROM PRODUITS
GROUP BY CODEPROD
;
QUIT;
```

II.3.1.4. Requête avec modification des valeurs d'une variable sans création de nouvelle variable

Convertissons par exemple les montants des produits achetés en milliers sans créer une valeur correspondante

```
PROC SQL ;
CREATE TABLE TAB_PRODUITS AS
SELECT NUMCMDE, CODEPROD, NOMPROD, MONTANT_PROD /1000
FROM PRODUITS
;
QUIT;
```

Dans cette requête l'instruction AS est omise alors la variable existante est modifiée.

II.3.2. Requête-création à partir de plusieurs tables

II.3.2.1. Requête sans calcul de nouvelles variables

Exemple :

```
PROC SQL;
CREATE TABLE TAB2 AS
SELECT  NOM, PRENOM, DATECMDE, DATELIV
FROM CLIENTS, COMMANDES
WHERE CLIENTS.IDCLIENT=COMMANDES.IDCLIENT
;
QUIT;
```

II.3.2.2. Requête avec calcul de nouvelles variables

Exemple :

```
PROC SQL;
CREATE TABLE TAB3 AS
SELECT  NOM, PRENOM, DATECMDE, DATELIV,
DATECMDE-DATELIV AS DUREE_JOURS,
(CALCULATED DUREE_JOURS )/12 AS DUREE_MOIS
FROM CLIENTS, COMMANDES
WHERE CLIENTS.IDCLIENT=COMMANDES.IDCLIENT AND ANNAISS>1990
;
QUIT;
```

II.3.2.3. Requête avec utilisation des fonctions d'agrégation

Exemple : Créer une table nommée CMDE_CLIENTS contenant le nom, prénom, âge et montant total des commandes par client :

```
PROC SQL ;
CREATE TABLE CMDE_CLIENTS AS
SELECT IDCLIENT, NOM, PRENOM, MONTANT_PROD,
2016- ANNAISS AS AGE,
SUM(MONTANT_PROD) AS TOTAL_CMDE,
FROM CLIENTS, COMMANDES, PRODUITS
WHERE CLIENTS.IDCLIENT =COMMANDES.IDCLIENT AND
    COMMANDES. NUMCMDE = PRODUITS. NUMCMDE
GROUP BY IDCLIENT
;
QUIT;
```

II.3.2.4. Requête avec modification des valeurs d'une variable sans création de nouvelles variables

Exemple : Créer une table nommée CMDE_CLIENTS contenant le nom, prénom et montant des commandes des produits en milliers :

```
PROC SQL ;
CREATE TABLE CMDE_CLIENTS AS
SELECT IDCLIENT, NOM, PRENOM, MONTANT_PROD/1000,
FROM CLIENTS, COMMANDES, PRODUITS
WHERE CLIENTS.IDCLIENT =COMMANDES.IDCLIENT AND
    COMMANDES. NUMCMDE = PRODUITS. NUMCMDE
;
QUIT;
```

II.3.3. Requête-Création avec recodage des variables

Pour obtenir les valeurs recodées d'une ou des plusieurs variables dans une PROC SQL avec création de variable, il suffit d'ajouter à la requête l'instruction CASE combinée avec l'instruction WHEN. Ces valeurs recodées d'une variable peuvent être numériques ou en caractères. L'exemple ci-dessous calcule les valeurs recodées de la population des communes (codage numérique et codage en texte).

```
PROC SQL ;
CREATE TABLE COMMUNE_DATA_NEW AS
SELECT  COMMUNE,  NBRE_HABITANTS,
CASE
WHEN NBRE_HABITANTS<= 5000 THEN 1
WHEN (NBRE_HABITANTS> 5000 AND NBRE_HABITANTS<= 10000) THEN 2
WHEN NBRE_HABITANTS> 10000 THEN 3
END AS CAT_HABITANT_NUM,
CASE
WHEN NBRE_HABITANTS<= 5000 THEN "Petite commune"
WHEN (NBRE_HABITANTS> 5000 AND NBRE_HABITANTS<= 10000) THEN "
Commune Moyenne "
WHEN NBRE_HABITANTS> 10000 THEN "Grande commune "
END AS CAT_HABITANT_TEXT
FROM COMMUNE_DATA
WHERE NBRE_HABITANTS ^=.;
QUIT ;
```

Dans chacun des deux types de recodage, on calcule de nouvelles informations nommées CAT_HABITANT_NUM et CAT_HABITANT_TEXT en utilisant l'information

initiale NBRE_HABITANTS. Notons que dans ce calcul, on exclut toutes les communes dont la population est manquante en utilisant la clause WHERE.

Notons aussi que les instructions CASE WHEN se comportent comme une formule classique de calcul de variable. Par conséquent, ils doivent figurer dans la clause SELECT séparée avec les autres éléments par une virgule. De plus le nom de la variable obtenue doit être déclarée à la suite en utilisant l'instruction AS. De par cette propriété, il peut exister plusieurs CASE WHEN dans une même requête.

II.4. Requête-modification de tables

D'une manière générale, une requête-modification est une requête dans laquelle on modifie une table existante (sans avoir besoin de créer une nouvelle table) soit en ajoutant de nouvelles variables, soit en ajoutant de nouvelles lignes d'observations, ou simplement en modifiant les valeurs des variables existantes.

II.4.1. Modifier les variables dans une table existante

Pour pouvoir modifier les valeurs des variables dans une table avec une PROC SQL, on utilise l'instruction UPDATE en indiquant la formule de modification. Les exemples ci-dessous sont des illustrations.

Exemple 1 :

```
PROC SQL ;
 UPDATE country_tab
    SET
    superficie= superficie*1000000,
    population=population/1000000 ,
     ;
    QUIT;
```

Exemple 2 :

```
PROC SQL ;
 UPDATE country_tab
    SET
     population=population*1,05
         WHERE namel like '%B'
    ;
    QUIT;
```

II.4.2. Ajouter de nouvelles variables à une table existante

La commande UPDATE permet de modifier les variables existantes dans une table mais elle ne permet pas d'ajouter de nouvelles. Il faut alors utiliser la commande ALTER TABLE avec l'instruction ADD pour insérer dans une premier temps de nouvelles variables dans la table et dans un second temps utiliser UPDATE pour ajouter les formules de calcul. L'exemple ci-dessous est une illustration.

```
/* Déclaration de la variable*/
        PROC SQL;
     ALTER TABLE COUNTRY_TAB
    ADD density_pop NUM LABEL='Population Density' FORMAT=6.2;
        QUIT;
/* Formule de calcul*/
    PROC SQL;
     UPDATE country_tab
    SET density=population/area;
    QUIT;
```

II.4.3. Sélection ou suppression de variable avec création de nouvelle table

Pour sélectionner ou supprimer une ou plusieurs dans une proc SQL, on utilise les instructions KEEP ou DROP dans la clause FROM (Voir exemples ci-dessous).

```
/* SUPPRESSION */
 PROC SQL;
CREATE TABLE FACTURE2 AS
    SELECT *
     FROM FACTURE (DROP=NOM_CLIENT NUM_EMPLOYE) ;
 QUIT;

/* SELECTION */

 PROC SQL;
CREATE TABLE FACTURE2 AS
    SELECT *
     FROM FACTURE (KEEP=NUM_FACTURE  NUM_CLIENT  NOM_PRODUIT QUANTITE
PRIX_UNIT) ;
 QUIT;
```

II.4.4. Suppression de variable sans création de nouvelle table

Pour supprimer une ou plusieurs variables sans création de table, on utilise l'instruction ALTER TABLE en ajoutant la clause DROP (voir l'exemple ci-dessous).

```
PROC SQL;
  ALTER TABLE COUNTRY_TAB
    DROP DENSITY_POP ;
          QUIT;
```

Ici, on supprime la variable DENSITY_POP de la table COUNTRY_TAB.

II.4.5. Insérer de nouvelles lignes d'observations dans une table

Pour insérer des observations à une table on utilise la commande INSERT INTO.

On peut distinguer trois manières d'utiliser l'instruction INSERT INTO pour ajouter des observations à une table: soit on spécifie les valeurs en utilisant la clause SET, soit on utilise la clause VALUES ou bien on utilise les résultats provenant d'une requête.

II.4.5.1. Ajout d'observations en utilisant la clause SET

Exemple :

```
PROC SQL;
  INSERT INTO country_tab
    SET
       name='Bangladesh',
        capital='Dhaka',
        population=126391060
    SET
        name='Japan',
        capital='Tokyo',
        population=126352003;
   QUIT;
```

Cette commande ajoute deux observations à la table country_tab. Les variables renseignées sont : name, capital et population.

On constate ici qu'il faut spécifier de clause SET que d'observation à ajouter.

II.4.5.2. Ajout d'observations en utilisant la clause VALUES

Exemple :

```
PROC SQL;
  INSERT INTO country_tab2
    VALUES ('Pakistan', 'Islamabad', 123060000, ., ' ', .)
    VALUES ('Nigeria', 'Lagos', 99062000, ., ' ', .);
    QUIT;
```

Cette commande ajoute deux observations à la table country_tab2 dans les colonnes name, capital et population et en mettant des valeurs manquantes pour les autres variables qui sont à la 4ième, 5ième et 6ième position et qui sont respectivement numérique, caractère et numérique.

II.4.5.3. Ajout d'observations en utilisant une requête

Exemple :

```
PROC SQL;
  INSERT INTO country_tab3
  SELECT *
 FROM country_table
 WHERE population ge 130000000;
  QUIT;
```

Dans cette requête, on insère dans country_tab3 toutes les observations venant de country_table pour lesquelles la population est supérieure à 130000000.

Notons ici que puisque nous sélectionnons toutes les informations de la table country_table avec SELECT*, pour que cette requête fonctionne correctement, il faut que les deux tables soient de même structure (nom des variables, dimensions table, formats, etc). Lorsque les tables ne sont pas de même dimension, on peut spécifier la clause SELECT en indiquant la liste des variables de sorte à ajuster les informations à insérer à la table de base.

II.4.6. Faire la copie d'une table

II.4.6.1. Copier une table avec toutes les observations

Faire la copie d'une table, c'est simplement effectuer une requête-création en sélectionnant toutes les informations disponibles dans la table initiale. Exemple :

```
PROC SQL ;
CREATE TABLE COPIE_A AS
SELECT *
FROM A
;
QUIT;
```

II.4.6.2. Créer une table vide des observations à partir d'une table existante

Pour créer une table vide conforme à une autre table donnée, on utilise l'instruction LIKE comme suit :

```
PROC SQL;
  CREATE TABLE NEWTABLE
    LIKE OLDTABLE;

QUIT;
```

Où OLDTABLE est la table initiale et NEWTABLE est la nouvelle table (vide) mais dont les variables ont les mêmes propriétés que celle de la table initiale.

II.4.7. Suppression d'observations dans une PROC SQL

Pour supprimer des observations dans une table selon une condition, on utilise l'instruction DELETE. L'exemple ci-dessous est une illustration.

```
PROC SQL;
  DELETE
    FROM COUNTRY_TAB
    WHERE NAME LIKE 'R%';
        QUIT;
```

Ici, on supprime toutes les observations pour lesquelles la valeur de la colonne name commence par R.

Il faut noter que pour supprimer toutes les observations d'une table, on spécifie la requête sans clause WHERE. Ainsi on a :

```
PROC SQL;
  DELETE
    FROM COUNTRY_TAB
    QUIT;
```

Cette requête permet d'aboutir au même résultat qu'en utilisant CREATE TABLE et l'instruction LIKE.

II.4.8. Suppression d'une table dans une PROC SQL

Pour supprimer une table dans une requête SQL, on utilise simplement la commande DROP TABLE. L'exemple suivant est une illustration.

```
PROC SQL;
 DROP TABLE COUNTRY_TAB;
 QUIT;
```

II.5. Quelques fonctions utiles dans PROC SQL

II.5.1. Utilisation de l'opérateur LIKE pour la sélection sur des variables en caractères

II.5.1.1. Exemple 1 : Afficher les informations sur les clients dont le nom contient un mot donné

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE PRENOM LIKE "%Jean%"
;
QUIT;
```

Cette commande affiche les informations sur les clients dont le prénom contient Jean. Ici, on utilise l'opérateur logique LIKE pour sélectionner les observations. Les deux symboles indiquent « contient ». Il peut y avoir de l'espace avant et après l'écriture du mot jean dans le prénom. Cela ne pose pas de problème. En revanche la spécification en minuscule ou majuscule aura un effet sur les résultats. Pour pallier ce problème, il faut utiliser la fonction UPCASE ou LOWCASE pour convertir le prénom en majuscule ou en minuscule avant de formuler la clause WHERE. Exemple :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
```

```
FROM CLIENTS
WHERE UPCASE(PRENOM) LIKE "%JEAN%" /* Majuscule*/
/* WHERE LOWCASE(PRENOM) LIKE "%jean%" */ /* Minuscule*/
;
QUIT;
```

II.5.1.2. Exemple 2 : Afficher des clients dont le nom commence par un mot donné

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE UPCASE(PRENOM) LIKE "JEAN%" /* Majuscule*/
;
QUIT;
```

Afficher des clients dont le nom finit par un mot donné

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE UPCASE(PRENOM) LIKE "%JEAN" /* Majuscule*/
;
QUIT;
```

II.5.1.3. Exemple 3: Affichage prenant en compte la possibilité d'une différence d'orthographe

Supposons qu'on veuille afficher les informations sur tous les clients dont le prénom est Jean Baptiste. Mais il arrive que ce nom soit souvent mal orthographié lors de la saisie : par exemple écrire Jeans Baptiste au lieu de Jean Baptiste. Pour pouvoir neutraliser l'effet de cette faute d'orthographe lors de la sélection et l'affichage, on utilise le symbole _(underscore) à la place de la (ou les) lettre(s) susceptible(s) d'être mal orthographiée(s), donc susceptible de créer des différences entre les mots). Voir l'exemple ci-dessous :

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE UPCASE(PRENOM) LIKE "JEAN_ BAPTISTE"
;
QUIT;
```

Le symbole underscore cache la lettre spécifiée lors de la sélection. Il doit y avoir autant de symboles underscore que de lettres à afficher. Ces symboles doivent être placés à l'endroit où les lettres sont susceptibles d'être différents. Par exemple imaginons qu'on veuille repérer toutes les observations en rapport avec « Leukocyte esterase » mais on soupçonne une mauvaise saisie au niveau du k, c et y. On peut alors demander à ce que toutes les observations ressemblant au mot « Leukocyte esterase » en cachant ces trois lettres dans l'orthographe. On procède alors comme suit :

```
PROC SQL ;
SELECT ID, ECHANTILLON, PRELEVEMENT
FROM TESTS
WHERE UPCASE(PRELEVEMENT) LIKE "LEU_O__TE"
;
QUIT;
```

II.5.1.4. Exemple 4 : Affichage par une combinaison des symboles % et _

```
PROC SQL ;
SELECT NOM, PRENOM, NUMTEL
FROM CLIENTS
WHERE UPCASE(PRENOM) LIKE "%JEAN_ BAPTISTE%"
;
QUIT;
```

Cette commande affiche les clients dont le prénom contient Jean Baptiste avec possibilité de présence de faute d'orthographe en mettant s à Jean.

Attention :

L'opérateur LIKE s'applique que sur les variables caractères uniquement. De plus elle ne s'applique que dans une clause WHERE et non avec l'instruction IF THEN.

II.5.2. Sélectionner et afficher un nombre spécifique d'observations

D'abord on peut remarquer que les résultats affichés dans une PROC SQL sont équivalents aux résultats affichés par une PROC PRINT, car les deux proc affichent simplement la liste des observations sélectionnées. De la même manière que les résultats de l'affichage sont semblables, on peut aussi sélectionner un nombre

spécifique d'observations. Pour cela, on ajoute l'option OUTOBS qui est l'équivalent de l'option OBS dans une PROC PRINT. Voir l'exemple :

```
PROC SQL OUTOBS=100;
SELECT *
FROM CLIENTS
WHERE ANNAISS>=1990
;
QUIT;
```

II.5.3. Identification et suppression des observations dupliquées

Pour éviter l'affichage dupliqué des observations, on ajoute l'option DISTINCT à la clause SELECT comme suit :

```
PROC SQL;
SELECT DISTINCT REGION, COMMUNE, MENAGE
FROM INFOMEN
;
QUIT;
```

Cette requête affiche les observations non dupliquées sur les variables REGION-COMMUNE-MENAGE. La suppression d'observations peut s'avérer nécessaire dans plusieurs situations. Par exemple, supposons qu'on veuille afficher la moyenne, le minimum et le maximum de l'âge en utilisant une proc SQL, il devient nécessaire de spécifier l'option DISTINCT afin d'éviter la répétition des valeurs à l'affichage. Pour cela, on va utiliser la requête suivante :

```
PROC SQL ;
SELECT DISTINCT MEAN(CALCULATED AGE) AS AGE_MOYEN,
MIN(CALCULATED AGE) AS MAX_AGE,
MAX(CALCULATED AGE) AS MIN_AGE
FROM CLIENTS
WHERE ANNAISS>=1990
;
QUIT;
```

II.5.4. Utilisation des ALIAS dans une requête sur plusieurs tables

Lors d'une requête sur plusieurs tables, il arrive que la gestion des noms des tables soit rendue difficile par leur longueur. On peut alors décider d'attribuer des synonymes (ALIAS) aux tables afin raccourcir les commandes. La définition des alias des tables se font dans la spécification de la clause FROM. Par exemple supposons la requête suivante :

```
PROC SQL;
SELECT CLIENTS.NOM, CLIENTS.PRENOM, COMMANDES.DATECMDE,
COMMANDES.DATELIV, PRODUITS.NOMPROD, PRODUITS.MONTANT_PROD
FROM CLIENTS, COMMANDES, PRODUITS
WHERE CLIENTS.IDCLIENT=COMMANDES.IDCLIENT AND
    COMMANDES. NUMCMDE = PRODUITS. NUMCMDE
;
QUIT;
```

Cette requête indique de sélectionner le nom et prénom de la table CLIENTS, de sélectionner la date de commande et la date de livraison à partir de la table COMMANDES et de sélectionner le nom du produit et le montant des achats à partir de la table PRODUITS. Pour raccourcir la formulation de cette requête, on peut créer des alias aux tables tels que CLIENTS=A, COMMANDES=B et PRODUITS=C. Dès lors la requête peut être reformulée comme suit :

```
PROC SQL;
SELECT A.NOM, A.PRENOM, B.DATECMDE, B.DATELIV, C.NOMPROD,
C.MONTANT_PROD
FROM CLIENTS A, COMMANDES B, PRODUITS C
WHERE A.IDCLIENT=B.IDCLIENT AND
    B. NUMCMDE =C. NUMCMDE
;
QUIT;
```

Cette formulation permet non seulement de gagner quelques espaces mais quelques degrés de visibilité sur le code.

II.5.5. Description du contenu d'une table avec PROC SQL

Pour décrire le contenu d'une table avec PROC SQL, on utilise l'instruction DESCRIBE TABLE définie par la syntaxe suivante :

```
PROC SQL ;
DESCRIBE TABLE CLIENTS ;
QUIT ;
```

Cette commande donne la liste des variables et leurs attributs (value label, label, formats).

Il faut aussi noter que dans cet exemple, la table CLIENTS est supposée se trouver dans la librairie par défaut (WORK). Si ce n'est pas le cas, il faut spécifier le nom de la librairie devant le nom de la table comme dans une DATA STEP classique.

II.5.6. Tester la validité de la formulation d'une requête SQL

Proc SQL offre aussi la possibilité de vérifier la bonne formulation d'une requête SQL en utilisant l'instruction VALIDATE. Lorsque la formulation de la requête est correcte, SAS affiche le message suivant : " *PROC SQL statement has valid syntax*".

L'exemple ci-dessous vérifie la formulation de la requête suivante :

```
PROC SQL ;
VALIDATE
SELECT NOM, PRENOM, NUMTEL, 2016- ANNAISS AS AGE,
MEAN(CALCULATED AGE) AS AGE_MOYEN,
MIN(CALCULATED AGE) AS MAX_AGE,
MAX(CALCULATED AGE) AS MIN_AGE
FROM CLIENTS
WHERE ANNAISS>=1990
GROUP BY REGION
HAVING CALCULATED AGE>15
ORDER BY REGION DESC
;
QUIT;
```

Toutefois, il faut signaler l'instruction VALIDATE ne vérifie pas la validité du résultat de la requête mais simplement la formulation de la requête. Par exemple lorsque l'utilisateur se trompe pour sélectionner une variable à place d'une autre variable, SAS n'aucune possibilité de vérifier que la bonne information n'a pas été choisie. SAS vérifie simplement si la structure de la requête répond au standard dans une requête SQL. Par conséquent la validité des résultats d'une requête est de la responsabilité du seul utilisateur.

II.6. Sous-requêtes et requêtes imbriquées

Dans le traitement et l'organisation des grandes bases de données, il est très fréquent d'être confronté à problèmes nécessitant l'élaboration des sous requêtes et les requêtes imbriquées.

Une sous-requête est une requête spécifiée dans la clause WHERE d'une requête principale. Tandis que construire des requêtes imbriquées c'est utiliser les résultats d'une requête pour faire une autre requête. Les résultats de cette même requête sont utilisés pour réaliser une autre requête. Et ainsi de suite.

Les sous-requêtes et le requêtes imbriquées sont spécifiées généralement dans la clause WHERE ou la clause HAVING.

Pour introduire les notions de sous-requêtes et de requêtes, partons d'abord d'un cas simple.

Supposons un épicier-vendeur de produits exotiques disposant d'une base de données constituée de deux tables: la table **ACHATS_PRODUITS** qui fournit la description de tous les achats effectués par l'épicier auprès de ses fournisseurs (ex : numéro achat, code produit, nom produit, prix d'achat unitaire, quantité achetée, montant des achats, date achat) et la table **VENTES_PRODUITS** qui contient les informations sur les ventes au clients de chaque produit (numéro vente, code produit, nom produit, prix de vente unitaire, quantité vendue, montant des vente, date vente, etc.).

Intéressons-nous maintenant au produit « palme » et supposons que son prix-fournisseur (c'est à dire prix d'achat) est de 10 à l'unité. En tant que gestionnaire de base de données, cet épicier vous fait la requête suivante : « *Afficher les informations sur les ventes de tous les produits dont le prix de vente à l'unité est supérieur au prix fournisseur du palme* »

Dès lors, puisque vous connaissez à priori (de tête) le prix fournisseur du palme, alors vous écrirez la requête suivante :

```
PROC SQL ;
SELECT *
FROM VENTES_PRODUITS
WHERE PRIX_UNIT_VENTE>10 ;
QUIT ;
```

Dans cette requête, on sélectionne simplement toutes les informations sur toutes les ventes de la table VENTES_PRODUITS pour lesquelles les prix unitaires sont supérieurs à 10. Dans cette formulation, il n'y a pas besoin de sous-requêtes ni de requêtes imbriquées car le prix d'achat du palme est directement connu (10).

Par contre, lorsque le prix d'achat du palme n'était pas connu d'avance, il faut alors interroger la table ACHATS_PRODUITS pour connaitre le prix unitaire, ensuite utiliser cette valeur dans la clause WHERE de la requête principale.

II.6.1. Elaboration d'une sous-requête

En considérant l'exemple ci-dessous, pour pouvoir à la répondre à la requête de l'épicier « *Afficher les informations sur les ventes de tous les produits dont le prix de vente à l'unité est supérieur au prix fournisseur du palme* », nous allons élaborer une sous-requête dans la première étape.

Ainsi, lorsque le prix d'achat du palme n'est pas connu d'avance, on adopte la formulation suivante :

```
PROC SQL ;
SELECT *
FROM VENTES_PRODUITS
WHERE PRIX_UNIT_VENTE GT
(SELECT PRIX_UNIT_ACHAT
FROM ACHATS_PRODUITS
WHERE UPCASE(NOM_PRODUIT) LIKE "%PALME%" /* formulation moins risquée que
NOM_PRODUIT= "Palme", possibilité de différence d'orthographe */
;
QUIT ;
```

Ainsi à la différence de la première requête la valeur 10 est remplacée par une sous requête : SELECT PRIX_UNIT_ACHAT FROM ACHATS_PRODUITS WHERE UPCASE(NOM_PRODUIT) LIKE "%PALME%".

Remarque :

Utiliser une sous-requête c'est définir une requête pour traduire la condition spécifiée la clause WHERE de la principale requête.

Il faut noter qu'il est possible que la sous requête renvoie plusieurs valeurs au lieu d'une valeur unique. Par exemple s'il se trouve que la table contient plusieurs variétés

de palmes enregistrées avec des noms différents, alors la condition WHERE UPCASE(NOM_PRODUIT) LIKE "%PALME%" renverra plusieurs valeurs. Dès lors échouera. C'est pourquoi, dans une telle éventualité, on ajoute des opérateurs logiques spécifiques aux sous-ensembles dont certaines sont décrites ci-dessous :

ANY : qui signifie qu'au moins un des éléments du sous-ensemble de valeurs obtenu à partir d'une sous-requête doit satisfaire à une condition donnée : ex :

```
...
...
WHERE POPULATION > ANY (SELECT POPULATION FROM COUNTRIES)
```

ALL : qui indique que toute les valeurs obtenues à partir d'une sous-requête doit satisfaire à une condition donnée. Ex :

```
...
...
WHERE POPULATION > ALL (SELECT POPULATION FROM COUNTRIES)
```

BETWEEN-AND : qui indique les valeurs renvoyées par la sous-requête doivent être comprise entre deux valeurs particulières v1 et v2 de manière inclusive. Ex :

```
...
...
WHERE POPULATION 100000 AND 150000
```

EXISTS : qui indique qu'il existe une valeur dans le sous-ensemble renvoyé par la sous-requête. Ex :

```
...
...
WHERE EXISTS(SELECT *
FROM ACHATS_PRODUITS
WHERE UPCASE(NOM_PRODUIT) LIKE "%PALME%")
```

IN : qui indique que la valeur doit figurer dans la liste des éléments fournis par la sous-requête. Ex :

```
...
...
WHERE NAME_CONTIENENT IN ('Africa', 'Asia')
```

Ou

```
...
...
WHERE NOMBRE IN (1, 2 , 8 , 20)
```

IS NULL ou ***(IS MISSING)*** : qui indique la valeur est manquante. Ex :

```
...
...
WHERE POPULATION IS MISSING
```

```
...
...
WHERE POPULATION IS NULL
```

LIKE : qui indique que la valeur spécifiée doit ressembler à un mot clé (voir la section II.4.1 pour plus de détails sur la fonction like. Ex :

```
...
...
WHERE CONTINENT LIKE 'A%'
```

CONTAINS : qui indique que les valeurs des chaînes de caractère renvoyée doit contenir le mot en question. Ex :

```
...
WHERE CONTINENT CONTAINS 'America'
```

NB : Tous ces opérateurs peuvent précédés de l'opérateur NOT pour traduire la négative.

II.6.2. Structure des requêtes imbriquées

L'exemple ci-dessous illustre la structure générale des requêtes imbriquées :

```
PROC SQL;
    SELECT * FROM TABLE1
    WHERE COUNTRY IN
     (SELECT COUNTRY FROM TABLE2
        WHERE TABLE2.COUNTRY IN
         (SELECT NAME FROM TABLE3.COUNTRIES
           WHERE TABLE3.CONTINENT='AFRICA'));

        QUIT;
```

On constate que dans chaque clause WHERE, le critère est le résultat d'une requête, qui, elle-même, est définie de sorte que sa clause WHERE est définie à partir d'une requête.

Au final selon les règles de bonnes pratiques, il faut toujours penser à utiliser JOIN ou une sous-requête quand on travaille sur plusieurs tables à la fois. Il arrive d'ailleurs très souvent qu'on combine les deux dans une même requête. L'exemple ci-dessous en est une illustration.

```
      PROC SQL ;
    SELECT A.CITY , A.STATE,
       A.LATITUDE , A.LONGITUDE ,
       B.CITY , B.STATE,
       B.LATITUDE , B.LONGITUDE,
       SQRT(((B.LATITUDE-A.LATITUDE)**2) + ((B.LONGITUDE-A.LONGITUDE)**2)) AS
DIST
    FROM USCITYCOORDS A, USCITYCOORDS B
    WHERE A.CITY NE B.CITY AND CALCULATED DIST =
        (SELECT MIN(SQRT(((D.LATITUDE-C.LATITUDE)**2) +((D.LONGITUDE-
C.LONGITUDE)**2)))
           FROM MYTAB.USCITYCOORDS C, MYTAB.USCITYCOORDS D
           WHERE C.CITY = A.CITY AND .STATE = A.STATE AND  D.CITY NE C.CITY)
        ORDER BY A.CITY;
QUIT;
```

II.7. Quelques exemples de requêtes simples

Dans les exemples qui suivent, nous nous servons de quatre tables hypothétiques nommées respectivement : employe, client, facture et produit.

1- Calculez le montant total des achats par client

```
proc sql ;
 select  num_client, sum(quantite*prix_unit) as Total_achat
 from facture
 group by num_client
 order by num_client ;
 quit;
```

2- Calculez le montant total des achats par les clients dont le nom contient « o»

```
proc sql ;
select  sum(quantite*prix_unit) as Total_achatO
from facture
where  nom_client like '%o%' ; /*attention au majuscule */
quit;
```

3- Calculez le montant total des achats par produit

```
proc sql ;
select nom_produit,  sum(quantite*prix_unit) as Total_achatprod
from facture
group by nom_produit
order by Total_achatprod ;
quit;
```

4- Calculez le montant moyen des achats par produit

```
proc sql ;
select nom_produit, mean(quantite*prix_unit) as moyen_vente
from facture
group by nom_produit
order by moyen_vente ;
run;
quit;
```

5- a. Afficher les informations sur les employés ayant au moins 10 ans de service

```
proc sql ;
select *
from employe
where anne_emploi ge 10 ;
quit;
```

5-b. Afficher les informations sur les employés qui ne vivent pas à 'Mimisan', et qui ont plus de 10 ans de service

```
proc sql ;
select *
from employe
where ville_employe ne 'Mimisan' and anne_emploi ge 10  ;
quit;
```

5-c. Afficher les informations sur les employés dont le nom commence par la lettre S

```
proc sql ;
  select *
  from employe
  where nom_employe like 'S%';
  quit;
```

5-d. Afficher les informations sur les employés ayant un nom de six lettres qui finit par un e

```
  proc sql ;
  select *
  from employe
  where nom_employe like '____e'; /* on tape cinq fois le underscore en terminant par le e */
  quit;
```

5-e. Afficher les informations sur les employés ayant travaillé 1, 5 ou 10 ans

```
proc sql ;
  select *
  from employe
  where anne_emploi in (1 , 5 , 10 );
  quit;
```

5-f. Afficher les informations sur les employés dont le numéro est compris entre 301 et 401

```
  proc sql ;
  select *
  from employe
  where num_employe between 301 and 401;
  quit;
```

5-g. Afficher les informations sur les employés pour lesquels la variable num_supervisor présente une valeur manquante, i.e. les employés n'ayant pas de responsable au-dessus d'eux dans la hiérarchie de l'entreprise

```
proc sql ;
  select *
  from employe
  where num_supervisor =.; /* is missing, is null*/
  quit;
```

6- En utilisant les tables client et facture, visualiser les produits vendus à Mimisan

/* Pour cela, il faut d'abord joindre les deux tables avec la clé num_employe et la condition ville_employe='Mimisan'*/

```
proc sql;
select f.num_facture, f.nom_client , f.num_client, f.num_employe, f.nom_produit ,f.quantite, f.prix_unit
from facture f , employe e
where f.num_employe =e.num_employe and e.ville_employe='Mimisan' ;
quit;
```

*ou bien

```
proc sql;
select f.num_facture, f.nom_client , f.num_client, f.num_employe, f.nom_produit , f.quantite, f.prix_unit
from facture f inner join employe e
on f.num_employe =e.num_employe and e.ville_employe='Mimisan' ;
quit;
```

7) Afficher le nom et la fonction du responsable à coté du nom et de la fonction de chacun des employés *Les employés qui sont sans superviseurs doivent être gardés dans la base

```
proc sql;
select e1.num_employe,e1.nom_employe, e1.titre_employe,e1.num_supervisor, e2.num_employe, e2.nom_employe, e2.titre_employe
from employe e1 left join employe e2
on e1.num_supervisor=e2.num_employe and e1.num_supervisor^=. ;
quit;
```

8) Afficher les produits vendus par Samuel à des magasins de La Torche

/*Ici en fait il faut afficher les factures pour lesquels num_client ou nom_client se retrouvent dans la table client avec la ville_client='Latorche' et dont le nom d'employé est 'Samuel' (table employe)*/

```
proc sql;
select f.num_facture, f.nom_client , f.num_client, f.num_employe, f.nom_produit ,f.quantite, f.prix_unit,
c.ville_client, e.nom_employe
from facture f ,client c, employe e
where f.nom_client=c.nom_client
and f.num_client =c.num_client
and f.num_employe=e.num_employe
and e.nom_employe='Samuel'
```

```
and c.ville_client='LaTorche'
;
quit;
*reponse final
proc sql;
select distinct( f.nom_produit)
from facture f ,client c, employe e
where f.nom_client=c.nom_client
and f.num_client =c.num_client
and f.num_employe=e.num_employe
and e.nom_employe='Samuel'
and c.ville_client='LaTorche'
;
quit;
```

9- Afficher les produits proposés par le fournisseur mais non-vendus dans les magasins

*Ici utilise une sous-requête et la clause NOT IN en utilisant la table facture puis la table produit

```
proc sql;
select nom_produit
from produit
where nom_produit NOT IN (select distinct nom_produit
from facture) ;
quit ;
```

10- Créer une table contenant les noms, fonctions et anciennetés des employés ayant travaillé au moins six ans pour le fournisseur

```
proc sql ;
create table  tab1 as
select nom_employe, titre_employe, anne_emploi
from employe
where anne_emploi>=6 ;
quit;
```

11- Créez une table nouvprix (pour nouveaux prix), qui est une copie de la table produit

```
proc sql;
  create table nouvprix as
```

```
   select*
from produit;
```
Pour créer une base vide mais conforme à la table produit on fait;
```
proc sql;
  create table nouvprix
    like produit;
```

Créer un table en ne gardant pas certaines variables
```
          proc sql;
  create table nouvprix as
   select*
from produit (drop=cout_production); /* ou encore from produit (keep=nom_produit
cout_production) */
```

10- Applique une augmentation de 20% au prix catalogue des produits coûtant moins de 240 euros, et qui laisse inchangé le prix des produits
coûtant plus de 240 euros.
```
proc sql ;
 update produit
     set cout_catalog=cout_catalog*1.2
          where cout_catalog<240;
     quit;
```

CHAPITRE III : UTILISATION DES MACRO-VARIABLES ET DES MACROS-PROGRAMMES

III.1. Définitions et caractéristiques générales des macro-variables et des macro-programmes

Les macros-variables et les macros-programmes sont des outils de programmation SAS permettant d'automatiser certaines tâches répétitives constituées d'opérations de mêmes natures.

Une macro-variable est une variable temporaire permettant de stocker des valeurs (en chaînes de caractère) obtenues à partir d'une étape DATA ou PROC ou directement définies par l'utilisateur. Voici-ci-dessous deux exemples de situations simples où l'utilisation de macro-variable peut s'avérer utile :

Exemple 1 :

Supposons qu'on dispose d'une table contenant un échantillon d'individus renseignés sur des variables socioéconomiques : âge, sexe, éducation, revenu, etc… Supposons maintenant qu'on veuille calculer dans une étape DATA, une variable binaire à partir de la variable revenu. Cette variable binaire prend 1 lorsque le revenu est supérieur à la moyenne et 0 si le revenu est inférieur à la moyenne. Etant donné qu'on ne connait pas la moyenne du revenu dans l'échantillon, nous allons d'abord réaliser une PROC MEANS pour déterminer le revenu moyen et stocker cette valeur dans une macro-variable. Ensuite faire appel à cette macro-variable dans l'étape DATA pour calculer la variable binaire (la manière de stocker la moyenne dans une macro-variable et l'appel de cette variable dans l'étape DATA seront discutées un peu plus loin).

Exemple 2 :

On veut diviser toutes les variables quantitatives numériques d'une table par 1000 pour les convertir en milliers. Pour cela, nous avons besoin d'abord de la liste des variables numériques. Pour constituer cette liste sous forme d'une macro-variable, on a deux possibilités. Soit définir la macro-variable en lui assignant manuellement la liste des variables numériques de la table ou bien définir la macro-variable en lui assignant automatiquement la liste des variables numériques de la table en utilisant les possibilités offertes par l'étape DATA. Les méthodes d'assignation manuelle ou automatique des valeurs de la macro-variable seront présentées un peu plus loin.

Un macro-programme est un ensemble d'instructions permettant d'exécuter une ou plusieurs tâches définies sous forme de bouts de programmes. Un macro-programme est généralement constitué d'un ensemble de macro-commandes formées d'un ensemble de macro-variables, de macro-fonctions ainsi que des procédures classiques de SAS (DATA et PROC). Voici-ci-dessous exemple de situation simple dans laquelle l'utilisation des macro-programmes peut s'avérer utile.

Exemple 3 :

Supposons qu'on veuille exécuter les tâches décrites à l'exemple à 2 sur 6 tables de données différentes. C'est-à-dire qu'on veut diviser toutes les variables quantitatives numériques de chaque table par 1000 pour les convertir en milliers. Face à cette situation, on est tenté de réaliser les opérations table par table. Mais cette procédure paraît très long surtout lorsque le nombre de tables est très élevé. Elle est surtout inefficiente car elle occupe de l'espace et allonge inutilement la longueur du programme. Dès lors, on peut se servir d'un macro-programme. La structure générale de ce macro-programme sera la suivante. Dans un premier temps, on stocke les noms de toutes les tables dans une macro-variable. Et dans un second temps on construit une boucle DO LOOP dont les valeurs de l'indice sont définies en fonction des noms des tables stockés dans la macro-variable. A l'intérieur de cette boucle DO LOOP, on indique toutes opérations à réaliser afin d'exécuter les tâches demandées. Ces opérations peuvent être des procédures classiques de SAS (DATA ou PROC) mais aussi la définition de nouvelles macro-variables ou l'utilisation des macros fonctions.

Il faut noter que les **macro-fonctions** sont simplement des fonctions classiques de SAS utilisées à l'intérieur d'un macro-programme. Définir une macro-fonction c'est indiquer simplement devant le nom de la fonction le symbole %. Par exemple, pour la boucle DO LOOP précédemment évoquée, on écrira simplement %DO….. %TO ;……%END ; Dès lors, à la différence des procédures classiques de SAS, les mots clés et les fonctions spécifiée à l'intérieur d'un macro-programme sont précédés du signe %.

Synthèse et remarque

Il est important d'avoir à l'esprit les remarques suivantes lors de l'écriture d'un programme SAS.

Un programme SAS est généralement constitué d'un ou des plusieurs macro-programmes et ou des procédures libres de SAS (DATA ou PROC).

Un macro-programme est généralement constitué d'un ou des plusieurs procédures libres (DATA ou PROC) définies sur des macro-variables sur lesquelles portent une ou plusieurs macros-fonctions.

Une macro-fonction est une fonction SAS classique utilisée dans le cadre d'un macro-programme. Elle permet généralement de traiter une macro variable. Une macro-fonction se distingue d'une fonction classique SAS par le symbole % indiqué devant le nom de la fonction.

Une macro-variable est une variable généralement définie (directement ou indirectement) à partir des informations disponibles dans les tables de données (variables, observations, statistiques, etc…). Dans un programme SAS, on distingue deux types de macro-variables : les macro-variables ***GLOBAL*** et les macro-variables ***LOCAL***. Une macro-variable GLOBAL est une macro-variable qui peut être appelée par tous les macro-programmes disponibles à l'intérieur du programme SAS. Elle est donc indépendante de la structure des macro-programmes et peut fonctionner partout dans le programme principal (à l'exception de CARDS et DATALINES). La macro-variable GLOBAL doit être déclarée avant les macro-programmes qui lui font référence. Une macro-variable LOCAL est une macro-variable spécifique à un macro-programme donné. Elle est définie et utilisée à l'intérieur du seul macro-programme qui lui fait référence. Lorsque plusieurs macro-programmes font référence à une même macro-variable, il est souhaitable de définir cette macro-variable comme GLOBAL.

S'agissant de l'utilisation des macro-fonctions ou des fonctions classiques, prenons le cas d'une fonction SAS spécifiée à l'intérieur d'une étape DATA ou PROC elle-même spécifiée à l'intérieur d'un macro-programme. Cette fonction n'est pas précédée du symbole % car elle est d'abord considérée comme un élément de la procédure DATA et non du macro-programme. En revanche toutes les fonctions et les mots réservés spécifiés à l'extérieur de l'étape DATA doivent être précédés du symboles %.

Notons aussi qu'une macro-fonction peut être définie à l'extérieur d'un macro-programme à conditions qu'elle porte sur macro-variable définie comme GLOBAL.

III.2. Déclarer et définir une macro-variable

Il existe deux méthodes pour définir une macro-variable dans un programme SAS : une assignation manuelle ou une assignation automatique.

III.2.1. Définir une macro-variable par assignation manuelle

Pour définir manuellement une macro-variable, on utilise la syntaxe suivante %LET nomMacro= listeValeurMacro ;. Dans les deux exemples ci-dessous, on définit deux macro-variables nommées joursem1 et joursem2 listant les jours de la semaine :

```
%LET joursem1 = Lundi Mardi Mercredi Jeudi Vendredi Samedi Dimanche;
%LET joursem2 =1 2 3 4 5 6 7 ;
```

La macro-variable joursem1 contient uniquement du texte alors que la macro joursem2 contient les chiffres. Toutefois, il faut noter que par définition, une variable macro est une variable en caractères. Par conséquent les chiffres indiqués dans la macro joursem2 sont lus comme des caractères. Bien entendu, il y des macro-fonctions permettant à SAS de lire ces valeurs comme des chiffres. Par exemple la fonction %EVAL() ou %SYSEVALF(). Nous reviendrons en détail sur ces fonctions un peu plus tard.

Il faut aussi noter qu'une macro-variable définie sans argument est une macro dont le contenu est NULL c'est-à-dire vide. Exemple :

```
%LET MyMacroVar =;
```

Par ailleurs, une fois la macro-variable est définie, pour invoquer cette variable en cours du programme, on lui fait précéder par le signe &. Par exemple pour afficher les valeurs de la macro joursem1 et joursem2 dans le log, on fait :

```
%PUT &joursem1;
%PUT &joursem2;
```

Nous reviendrons plus en détails sur l'invocation des macro-variables dans les sections suivantes.

Remarque

Il faut signaler qu'on peut également définir une macro-variable en lui assignant les valeurs d'une ou de plusieurs autres macro-variables. Dans ce cas l'assignation se présente comme une addition des arguments. L'exemple ci-dessous est une illustration.

```
%LET prenom =Moussa;
%LET nom =Keita;
%LET NomComplet =&prenom &nom;
%PUT &NomComplet;
```

III.2.2. Définir une macro-variable par assignation automatique

Une assignation automatique de la valeur d'une macro-variable consiste à utiliser certaines fonctions spéciales de SAS pour extraire les valeurs d'une table (ou du système) pour les attribuer automatiquement à une macro-variable déclarée. Les exemples les plus courants d'assignations automatiques sont par exemple : compter le nombre d'observations d'une table ou sur une variable et stocker cette valeur dans une macro-variable ; ou bien compter le nombre de variables dans une table et extraire la listes des variables et la stocker dans une macro-variables.

Plusieurs fonctions SAS permettent d'extraire des informations d'une table ou du système lui-même pour les stocker sous forme de macro-variables. Nous allons passer en revue quelques-unes de ces fonctions.

III.2.2.1. Création automatique de macro-variables par la fonction CALL SYMPUT

La fonction CALL SYMPUT permet d'extraire une valeur dans une étape DATA et la stocker dans une macro variable. La syntaxe générale de l'utilisation d'une CALL SYMPUT est la suivante :

```
CALL SYMPUT("NomMacroVar", ValeurMacroVar)
```

Où NomMacroVar est le nom qu'on attribue à la macro-variable définie et ValeurMacroVar est le nom de la variable dans la table SAS dont valeur est assignée automatiquement à la macro-variable NomMacroVar. Cette valeur est extraire de la table en utilisant une étape DATA.

On peut distinguer plusieurs cas d'utilisation de CALL SYMPUT :

- CALL SYMPUT pour créer une seule macro-variable à partir d'une seule variable en utilisant une seule table
- CALL SYMPUT pour créer plusieurs macro-variables à partir d'une seule variable en utilisant une seule table
- CALL SYMPUT pour créer plusieurs macro-variables à partir de plusieurs variables en utilisant une seule table
- CALL SYMPUT pour créer une ou plusieurs macro-variables à partir d'une ou plusieurs variables en utilisant plusieurs tables

III.2.2.1.1. CALL SYMPUT pour créer une seule macro-variable à partir d'une seule variable en utilisant une seule table

Exemple

Supposons qu'on veuille récupérer la moyenne sur une variable nommée MYVAR à partir d'une table nommée MYTABLE et stocker cette valeur dans une macro-variable nommée myvalue. On procèdera en deux étapes :

Dans la première étape, on utilise PROC SUMMARY sur la variable MYVAR en utilisant la table MYTABLE avec l'option OUTPUT OUT pour créer une nouvelle table nommée STAT_DATA qui contient la moyenne dans une colonne nommée MYVAR_MEAN (notons aussi qu'on peut utiliser PROC MEANS). La syntaxe de cette création de table avec PROC SUMMARY est la suivante :

```
PROC SUMMARY DATA=MYTABLE MEAN NOPRINT;
VAR MYVAR ;
OUTPUT OUT= STAT_DATA MEAN=MYVAR_MEAN;
RUN;
```

Dans la seconde étape, on utilise la table MYSTAT pour extraire la valeur de MYVAR_MEAN et le stocker dans une macro-variable nommée myvalue. La syntaxe est la suivante :

```
DATA _NULL_;
SET STAT_DATA ;
CALL SYMPUT('myvalue', MYVAR_MEAN) ;
RUN;
```

Pour afficher la valeur obtenue, on fait :

```
%PUT &myvalue;
```

Notons que dans la deuxième étape, puisqu'il n'y a pas nécessité de créer une nouvelle table, alors, on utilise l'instruction DATA _NULL_

III.2.2.1.2. CALL SYMPUT pour créer plusieurs macro-variables à partir d'une seule variable en utilisant une seule table

Lorsqu'on veut créer plusieurs macro-variables à partir d'une seule variable dans une même étape DATA, on spécifie plusieurs CALL SYMPUT. Par exemple, supposons qu'on veuille stocker la moyenne, le minimum et le maximum de la variable MYVAR dans trois macro-variables différentes, on suit les deux étapes décrites ci-dessous :

```
/* Extraction de la moyenne, du min, et du max */
PROC SUMMARY DATA=MYTABLE MEAN MIN MAX NOPRINT ;
VAR MYVAR ;
OUTPUT OUT=STAT_DATA
MEAN=MEAN_MYVAR
MIN=MIN_MYVAR
MAX=MAX_MYVAR
;
RUN;
/* Récupération des valeurs stockées */
DATA _NULL_;
SET STAT_DATA ;
CALL SYMPUT('mean_ myvar', MEAN_ MYVAR) ;
CALL SYMPUT('min_ myvar', MIN_ MYVAR) ;
CALL SYMPUT('max_ myvar', MAX_ MYVAR) ;
RUN;
/* Affichage si besoin */
%PUT &mean_myvar;

%PUT &min_myvar;
%PUT &max_myvar;
```

III.2.2.1.3. CALL SYMPUT pour créer plusieurs macro-variables à partir de plusieurs variables en utilisant une seule table

Exemple : on veut récupérer et stocker la moyenne, le minimum et le maximum de deux variables MYVAR1 et MYVAR2 en assignant chaque valeur à une macro-variable spécifique. Pour cela, on suit les deux étapes décrites ci-dessous :

```
/* Extraction de la moyenne, du min, et du max */
PROC SUMMARY DATA=MYTABLE MEAN MIN MAX NOPRINT ;
VAR MYVAR1 MYVAR2 ;
OUTPUT OUT=STAT_DATA
MEAN=MEAN_MYVAR1 MEAN_MYVAR2
MIN=MIN_MYVAR1 MIN_MYVAR2
MAX=MAX_MYVAR1 MAX_MYVAR2
;
RUN;
```

```
/* Récupération des valeurs stockées */
DATA _NULL_;
SET STAT_DATA ;
/*Mean*/
CALL SYMPUT('mean_ myvar1', MEAN_ MYVAR1) ;
CALL SYMPUT('mean_ myvar2', MEAN_ MYVAR2) ;
/*Min*/
CALL SYMPUT('min_ myvar1', MIN_ MYVAR1) ;
CALL SYMPUT('min_ myvar2', MIN_ MYVAR2) ;
/*Max*/
CALL SYMPUT('max_ myvar1', MAX_ MYVAR1) ;
CALL SYMPUT('max_ myvar2', MAX_ MYVAR2) ;
RUN;
/* Affichage si besoin */
%PUT &mean_myvar1;
%PUT &mean_myvar2;
%PUT &min_myvar1;
%PUT &min_myvar2;
%PUT &max_myvar1;
%PUT &max_myvar2;
```

Cependant, lorsque le nombre de variables est très élevé, on peut raccourcir l'écriture des étapes d'extraction en élaborant un macro-programme pour faire une boucle %DO LOOP sur la liste des variables préalablement définie selon une macro-variable (nous reviendrons en détails sur l'élaboration des macro-programmes et la construction des boucles sur les éléments d'une liste).

III.2.2.1.4. CALL SYMPUT pour créer plusieurs macro-variables à partir de plusieurs variables en utilisant plusieurs tables

Pour effectuer un CALL SYMPUT sur plusieurs tables pour créer des variables macros en se basant sur plusieurs tables, il faut penser systématiquement à écrire un macro-programme construit sur une boucle %DO LOOP dont les compteurs sont définies à partir de la liste des tables et la liste des variables. Ces listes doivent être spécifiées d'abord sous forme de macro-variables (définie soit par assignation manuelle ou automatique).

III.2.2.1.5. Remarque importante sur l'utilisation de la fonction CALL SYMPUT

Il est important de noter que lorsque la fonction CALL SYMPUT est appliquée sur une variable dans une table donnée, elle extrait la dernière valeur de cette variable (dernière observation) et la stocke dans la macro-variable. Lorsqu'il s'agit d'une

variable numérique et que la dernière valeur soit manquante, alors la macro-variable crée contient un point comme valeur. Et lorsqu'il s'agit d'une variable en caractère et que la dernière valeur est manquante alors la macro-variable créer contient un espace comme valeur (c'est-à-dire vide).

Compte tenu de cette propriété de la fonction CALL SYMPUT, il est donc important de prendre toutes les précautions avant de l'appliquer à une table. D'abord, il faut s'assurer que la table utilisée contient une seule ligne d'observation. Lorsque la table contient plusieurs lignes d'observations, il faut s'assurer que la valeur à extraire se trouve sur la dernière ligne de la table. Dans tous autres cas, il faut utiliser des astuces techniques pour pouvoir mettre la ligne d'observation en question sur la dernière ligne. Par exemple, on peut numéroter les observations de sorte à attribuer le plus grand numéro à l'observation et ensuite utiliser PROC SORT pour faire un tri croissant des observations de sorte que la ligne d'intérêt se trouve à la dernière position. Ou encore, on peut carrément supprimer toutes les autres observations de sorte à ne garder que la seule ligne d'intérêt. Dans ce cas, si la table d'origine est importante, il faut veiller à créer une nouvelle table.

III.2.2.2. Création automatique de macro-variables par PROC SQL

Créer une macro-variable par SQL, c'est lui assigner automatiquement les résultats issues d'une requête PROC SQL. Cette assignation se fait en utilisant l'instruction INTO à la suite de la clause SELECT. Dans sa structure la plus simple, la syntaxe de la création de macro-variable par PROC SQL se présente comme suit :

```
PROC SQL;
  SELECT MYVAR
    INTO :myvalue
    FROM MYTABLE;
QUIT ;
%PUT &myvalue;
```

Dans cette requête, on sélectionne MYVAR à partir de la table MYTABLE et on stocke le résultat obtenu dans une macro appelée myvalue. Ensuite on affiche cette valeur en utilisant %PUT.

Il faut noter que malgré sa simplicité, la structure de commande présentée ci-dessus souffre de plusieurs limites. L'une des principales limites vient du fait que la requête exécutée peut renvoyer plusieurs lignes de résultats. Dès lors myvalue ne sera pas

correctement définie car elle ne considère que la première valeur de la liste. D'où la nécessité d'adapter cette structure initiale afin de la généraliser.

En effet, étant donné qu'une requête renvoie généralement un résultat à plusieurs lignes (par exemple requête demandant les noms des employés figurant dans une table nommée EMPLOYE, Cf. chapitre II), il existe alors différentes façons d'assigner les résultats de cette requête à des macros-variables. On peut distinguer trois principaux cas :

- Assigner la première ligne du résultat d'une requête à une macro-variable
- Assigner chaque ligne du résultat d'une requête à une macro variable spécifique
- Assigner toutes les lignes du résultat d'une requête à une seule macro-variable en les séparant par un délimiteur spécifiée.

Dans cette section, nous allons étudier chaque cas en donnant des exemples concrets.

III.2.2.2.1. Définir une macro-variable à partir de la première ligne de résultat d'une PROC SQL

Cas d'une SQL sur une seule variable

Contrairement à CALL SYMPUT qui extrait la macro-variable à partir de la dernière ligne d'une table dans une étape DATA, l'instruction INTO dans une PROC SQL considère par défaut la première ligne de résultats. Par conséquent, lorsqu'on veut créer une macro-variable uniquement à partir de la première ligne de résultats d'une requête SQL, la structure de code précédemment présentée reste valable. Pour rappel, on a :

```
PROC SQL;
  SELECT MYVAR
    INTO :myvalue
    FROM MYTABLE;
QUIT ;
%PUT &myvalue;
```

On peut rencontrer différents contextes où ce type de formulation peut être utilisé. Par exemple, supposons qu'on veuille récupérer le nom de l'employé le plus âgé dans une entreprise en considérant une table nommées EMPLOYE contenant le nom, le sexe, l'âge (en années), etc. Dans ce cas, on peut utiliser la formulation suivante :

```
PROC SQL;
```

```
  SELECT NOM
    INTO :myvalue
    FROM EMPLOYE ;
    ORDER BY AGE DESC ;
QUIT ;
%PUT &myvalue;
```

L'instruction DESC tri les employés par rapport à leur âge de sorte que le plus âgé se retrouve sur la première ligne. Dès lors, en appliquant l'instruction INTO la macro-variable sélectionne automatiquement le premier nom figurant sur la liste.

Cas d'une SQL sur plusieurs variables

Lorsque l'instruction SELECT contient plusieurs variables, on peut créer une macro-variables correspondant à chacune de ces variables en séparant les arguments de l'instruction INTO par des virgules. Par exemple, on récupérer le nom, l'âge, le sexe de l'employé le plus âgé, on utilise la formulation suivante :

```
PROC SQL;
  SELECT NOM, AGE, SEXE
    INTO :nom_val, :age_val, :sexe_val
    FROM EMPLOYE ;
    ORDER BY AGE DESC ;
QUIT ;
%PUT &nom_val;
%PUT &age_val;
%PUT &sexe_val;
```

Comme discuté dans le cas d'une seule variable, l'instruction ORDER BY AGE DESC sert uniquement à ordonner les employés du plus âgé au moins âgé de sorte que la macro-variable puisse correspondre à la première ligne de résultat de la requête SQL. Bien entendu, le tri peut être réalisé en considérant d'autres variables.

Remarque

Il faut noter que la création de macro-variable est aussi valable pour les variables calculées en cours du proc SQL (que ces variables soient calculée avec des formules simples ou en utilisant les fonctions d'agrégations). Exemple : récupérer l'âge en mois de l'employé le plus âgé et l'âge moyen des employé :

```
PROC SQL;
  SELECT
    (AGE)*12 AS AGE_MOIS,
    MEAN(AGE) AS AGE_MOYEN
    INTO :agemois_val, :agemoyen_val
    FROM EMPLOYE ;
    ORDER BY AGE DESC ;
QUIT ;
%PUT &agemois_val;
%PUT &agemoyen_val;
```

III.2.2.2.2. Définir une macro-variable contenant toutes les lignes de résultat d'une PROC SQL

Cas d'une SQL sur une seule variable

Nous avons montré que par défaut l'instruction INTO sélectionne la première valeur de la liste fournie par PROC SQL. Pour modifier ce comportement et créer une macro-variable contenant tous les éléments de la liste, on ajoute à la formulation de base l'instruction SEPARATED BY. Cette instruction permet alors d'indiquer un symbole permettant de délimiter chaque élément de la liste de valeurs. Dans l'exemple ci-dessous, on stocke le nom de tous les employés dans une macro-variable nommée nom_val.

```
PROC SQL;
  SELECT NOM
    INTO :nom_val SEPARATED BY ' '
    FROM EMPLOYE ;
    QUIT ;
%PUT &nom_val;
```

Cette requête stocke les noms de tous les employés dans une macro-variable nommée nom_val en séparant par l'espace (blank). Cela est fait en utilisant l'instruction SEPARATED BY ' '.

Remarque 1

Signalons que le choix du symbole séparateur est crucial, car il conditionne la dimension de la macro-variable créée. En effet, une macro-variable sans séparateur définie telle que SEPARATED BY '' ne contient potentiellement qu'un seul élément. Car il concatène tous les éléments sans aucune séparation. D'un autre côté, lorsqu'on utilise l'espace comme un séparateur, il est possible que la macro-variable contient

plus d'éléments que d'observations car les noms composés (ex : Jean Paul seront considérés comme deux éléments différents lors de l'exploitation de la macro-variable. C'est pourquoi, il est préférable un symbole de séparateur qui n'a aucune chance de figurer dans les données. Par exemple :

```
PROC SQL;
  SELECT NOM
    INTO :nom_val SEPARATED BY '£$ '
    FROM EMPLOYE ;
    QUIT ;
%PUT &nom_val;
```

Remarque 1

Il peut arriver que les valeurs stockées dans une macro-variable définie par assignation automatique par proc SQL soient dupliquées. Par exemple, en récupérant les noms des employés, il est possible que plusieurs employés aient le même nom. Comme, ils seront tous affichés par proc SQL par défaut, alors si cette duplication n'est utile pour la suite, il vaut supprimer les valeurs dupliquées. Pour cela, il faut utiliser l'instruction DISTINCT dans SELECT. Exemple :

```
PROC SQL;
  SELECT DISTINCT NOM
    INTO :nom_val SEPARATED BY '£$ '
    FROM EMPLOYE ;
    QUIT ;
%PUT &nom_val;
```

Attention toutefois à utiliser distinct lorsqu'il s'agit de plusieurs variables dans la clause SELECT (cas étudié ci-dessous).

Cas d'une SQL sur plusieurs variables

Lorsque la requête porte sur plusieurs variables, les macro-variables correspondantes doivent être spécifiées dans la clause INTO en les séparant par des virgules. L'exemple ci-dessous est une illustration :

```
PROC SQL;
  SELECT NOM, AGE
    INTO :nom_val SEPARATED BY '£$ ' , age_val SEPARATED BY '||'
    FROM EMPLOYE ;
    QUIT ;
%PUT &nom_val;
%PUT &age_val;
```

Cette requête stocke toute les valeurs obtenues sur nom dans une macro-variable nommée nom_val et toutes les valeurs obtenues sur l'âge dans une macro-variables age_val.

Il faut signaler qu'utiliser l'instruction **DISTINCT** ne permet d'éliminer les toutes les valeurs dupliquées sur NOM ou sur AGE car cette instruction porte sur les deux variables en même temps. Dès lors, la duplication sera supprimée que lorsqu'on identifie deux employés qui ont le même nom et le même âge.

Si l'on veut des macro-variables ne contenant aucune valeur dupliquée, il vaut mieux définir les macros-variables variable par variable. Et lorsque le nombre de variable est très élevée, on peut imaginer la construction d'un macro-programme en faisant une %DO LOOP sur la liste des variables elle-même définie sous forme de macro-variables (nous verrons la construction des macro-programmes un peu plus loin).

III.2.2.2.3. Définir plusieurs macro-variables, chacune correspondant à une ligne du résultat de la PROC SQL

Cas d'une SQL sur une seule variable

Nous avons montré que par défaut l'instruction INTO sélectionne la première valeur de la liste des valeurs fournie par PROC SQL. Toutes les valeurs restantes sont donc abandonnées. Pour modifier ce comportement et créer une macro-variable pour chaque élément de la liste, on adopte la formulation suivante (prenant l'exemple du nom des employés) :

```
PROC SQL;
  SELECT NOM
    INTO :nomval1- :nomvalN
    FROM EMPLOYE ;
    QUIT ;
%PUT &nomval1;
…
…
%PUT &nomvalN;
```

Dans cette formulation, les noms des employés sont individuellement stockées dans les macro-variables allant de nomval1 à nomvalN où N correspond au nombre d'observations de la table. Comme la table contenant 10 employés, on crée dix macro-variables telles que :

```
PROC SQL;
  SELECT NOM
    INTO :nomval1- :nomval10
    FROM EMPLOYE ;
    QUIT ;
%PUT &nomval1;
…
…
%PUT &nomval10;
```

Cette formulation indique qu'on connait à priori le nombre d'observation dans la table. En effet, comme on a 10 employés dans la table, on spécifie qu'on veut 10 macro-variables indexés de 1 à 10. A ce propos, il faut signaler que lorsque le nombre de macro-variables déclarée est supérieur au nombre d'observations obtenu dans PROC SQL, toutes les macro-variables indexées au-delà de ce nombre auront une valeur NULL c'est-à-dire vide. Par exemple, en spécifiant :

```
PROC SQL;
  SELECT NOM
    INTO :nomval1- :nomval13
    FROM EMPLOYE ;
    QUIT ;
```

Sachant que nous avons 10 employés dans la table EMPLOYE, alors les macro-variables nomval11 à nomval13 sont des macro-variables vides d'éléments (NULL).

En revanche, lorsque le nombre de macro-variables déclarée est inférieur au nombre d'observations obtenu dans PROC SQL, aucune macro-variable n'est créée pour les observations restantes. Par exemple, en spécifiant :

```
PROC SQL;
  SELECT NOM
    INTO :nomval1- :nomval6
    FROM EMPLOYE ;
    QUIT ;
```

Sachant que nous avons 10 employés dans la table EMPLOYE, alors aucune macro-variable ne sera créée pour les observations 7 à 10.

Compte tenu de cette propriété, on peut considérer la création de macro-variable dans cette comme une généralisation du premier cas étudié. En effet en spécifiant par exemple :

```
PROC SQL;
  SELECT NOM
    INTO :nomval1- :nomval1
    FROM EMPLOYE ;
    QUIT ;
```

On obtient le même résultat que lorsqu'on spécifie

```
PROC SQL;
  SELECT NOM
    INTO :nomval1
    FROM EMPLOYE ;
    QUIT ;
```

Car dans les deux cas, seule la première ligne du résultat du SQL est stockée sous forme de macro-variable.

Remarque : cas où le nombre d'observation n'est pas connu à priori

Dans les exemples présentés ci-dessus, il est supposé qu'on connait à priori le nombre d'observations dans la table. Par exemple, puisque nous savons qu'il y a 10 employés dans la table, nous extrayons les noms de ces 10 employés pour les stocker dans les macro-variables nomval1 à nomval10. Toutefois, lorsque l'on ne connait pas à priori le nombre d'observation dans la table (plus spécifiquement le nombre d'observation sur la variable considérée), il faut alors chercher avant tout à déterminer ce nombre afin de le spécifier dans la requête. Mais grâce aux fonctionnalités du PROC SQL ce travail est extrêmement simplifié. En effet, pour chaque PROC SQL exécutée, SAS crée automatiquement une macro-variable nommée **SQLOBS**. Cette macro-variable stocke le nombre d'observations associé à la PROC SQL dernièrement exécutée. Pour l'afficher il suffit de faire %PUT &SQLOBS ; Cette fonction sera donc utile pour déterminer la valeur maximale de la liste des macro-variables à créer. Mais son utilisation n'est pas directe. Effet, la création des macro-variables à partir de la valeur de SQLOBS se fait en trois étapes décrites comme suit :

- Dans une première étape, on fait une proc SQL (sans la clause INTO) afin que SAS crée de lui-même la macro-variable automatique SQLOBS.
- Dans un second temps, nous une nouvelle macro-variable en lui assignant " manuellement" la valeur de SQLOBS. Cette assignation se fait comme suit :

%LET i=&SQLOBS ;

- Et dans une troisième étape, on refait une PROC SQL avec la clause INTO en utilisant la valeur contenue dans i comme l'indice de la dernière macro variable à créer.

Ces trois étapes sont résumées dans l'exemple ci-dessous :

```
    /*Etape 1*/
PROC SQL;
  SELECT NOM
    FROM EMPLOYE ;
QUIT ;
    /*Etape 2*/
%LET i=&SQLOBS ;
    /*Etape 3*/
PROC SQL;
  SELECT NOM
   INTO :nomval1- :nomval&i
    FROM EMPLOYE ;
QUIT ;
```

Cas d'une SQL sur plusieurs variables

Lorsque la requête porte sur plusieurs variables, pour créer une macro-variable pour chaque élément de chaque colonne, on indique la liste des macro-variables à créer pour chaque en les séparant de celles des autres colonnes par des virgules. Dans les exemples ci-dessous, nous distinguons les cas selon que le nombre d'observations de la requête est connue ou pas.

En effet supposons qu'on veuille récupérer les noms et les âges des employés et stocker chaque valeur obtenue dans une macro-variable. Dès lors, si le nombre d'employés est à priori connu (ex : 10), on adopte la formulation suivante :

```
PROC SQL;
  SELECT NOM, AGE
    INTO :nomval1-nomval10, :ageval1- :ag_val10
    FROM EMPLOYE ;
    QUIT ;
```

En revanche, lorsque le nombre d'observations n'est pas connu à priori, on suit les trois étapes décrites ci-dessous :

```
    /*Etape 1*/
```

```
PROC SQL;
  SELECT NOM, AGE
    FROM EMPLOYE ;
QUIT ;
    /*Etape 2*/
%LET i=&SQLOBS ;
    /*Etape 3*/
PROC SQL;
  SELECT NOM, AGE
   INTO :nomval1- :nomval&i , :ageval1- :ageval&i
    FROM EMPLOYE ;
QUIT ;
```

Notons que dans toutes méthodes de création automatique de macro-variables discutées ci-dessus, lorsque le nombre de variables à traiter devient plus élevé, il faut préférer construire un macro-programme en élaborant une boucle %DO LOOP qui traite individuellement chaque variable et stocker les macro-variables sous des noms indexés avec les nom des variables initiales (nous verrons la construction de macro programme plus loin).

III.2.2.3. Les macro-variables prédéfinies dans le système SAS

Il existe plusieurs macro-variables prédéfinies dans le système SAS. Par exemple la macro-variable SYSDATE indique la date du jour paramétré sur le système. Tandis que la macro-variable SYSDAY indique le jour actuel en se basant sur la date indiquée par le système. Les deux exemples ci-dessous affichent dans le fichier log la date et jour actuels.

```
%PUT &SYSDATE;
%PUT &SYSDAY;
```

Pour afficher toutes les autres macro-variables automatiques du système SAS, on fait :

```
%PUT _AUTOMATIC_;
```

Ces macro-variables automatiques fournissent de nombreuses informations qui s'avérer utiles dans e nombreux contexte (consulter le fichier log).

Par ailleurs, on peut afficher les noms de toutes les macro-variables disponibles dans une session en faisant :

```
%PUT _ALL_;
```

Mais il y a aussi :

```
%PUT _GLOBAL_ ; /* Affiche toutes les macro-variables de type global*/
%PUT _LOCAL_ ; /* Affiche toutes les macro-variables de type local*/
%PUT _USER_ ; /* Affiche toutes les macro-variables définies par l'utilisateur*/
```

III.2.2.4. Stocker le nombre d'observations dans une macro-variable

On peut distinguer plusieurs méthodes d'extraction du nombre d'observation pour le stocker sous forme de macro-variables. Ici, nous passons en revue quelques une d'entre elles.

*III.2.2.4.1. Utilisation de CALL SYMPUT avec la fonction _N_*

Pour déterminer le nombre total d'observations dans une table, on peut utiliser CALL SYMPUT avec la fonction _N_ dans une étape DATA.

Exemple :

```
DATA _NULL_;
SET MYTABLE ;
CALL SYMPUT('nobs', _N_) ;
RUN;
```

Dans cet exemple, on utilise la table MYTABLE et récupère la valeur contenue dans la variable automatique _N_ et on stocke cette valeur dans une macro-variable nommée nobs.
Il faut noter qu'ici, _N_ est calculé sur toute la table, il ne tient donc pas compte du nombre de valeurs manquantes sur certaines variables. Par conséquent, lorsque les travaux portent sur les valeurs valides d'une variable donnée, il faut utiliser les méthodes d'extraction basées sur les variables (voir les autres cas ci-dessous).

III.2.2.4.2. Extraction de la macro-variable automatique SYSOBS

On peut aussi obtenir le nombre total d'observations d'une table en utilisant la macro-variable SYSOBS. La macro-variable SYSOBS se met automatiquement à jour après chaque étape DATA ou chaque PROC. Pour extraire sa valeur, il faut d'abord actualiser son contenu.

Exemple :

```
/* actualisation de la valeur de SYSOBS */
DATA MYDATA;
SET MYDATA ;
RUN;
/* actualisation de la valeur de SYSOBS */
%LET nobs=&SYSOBS ;
```

La valeur contenue dans SYSOBS est copiée dans une nouvelles macro-variable nommée nobs.

III.2.2.4.3. Utilisation de CALL SYMPUT pour extraire le nombre d'observations après une PROC

A la différence des deux précédentes méthodes, cette méthode permet de déterminer le nombre d'observations correctement renseignées sur une variable donnée. Par exemple, supposons qu'on veuille déterminer le nombre d'observations valides sur une variable nommée MYVAR dans une table nommée MYTABLE, on suit les étapes :

```
/* Extraction du nombre d'observations valides */
PROC SUMMARY DATA=MYTABLE N NOPRINT ; /* Ou PROC MEANS */
VAR MYVAR ;
OUTPUT OUT=STAT_DATA
N=Nobsvar
;
RUN;
/* Récupération de la valeur stockée */
DATA _NULL_;
SET STAT_DATA ;
CALL SYMPUT('nobs', Nobsvar) ;
RUN;
```

Cette extraction se fait en deux étapes. D'abord, on réalise une PROC SUMMARY sur le nombre d'observations valides de la variable et on exporte ce résultat dans une table nommée STAT_DATA (on pouvait aussi utiliser PROC MEANS lorsque la variable est numérique). Le nom de la variable qui contient le nombre d'observations dans la nouvelle table créée est Nobsvar. Dans la deuxième partie, on invoque le nom de cette table dans une étape DATA et on extrait la valeur de la moyenne en utilisant CALL SYMPUT. La valeur obtenue est alors stockée dans une macro-variable nommée nobs.

III.2.2.4.4. Extraction de la macro-variable automatique SQLOBS

On peut aussi obtenir le nombre d'observations valide sur une variable en utilisant la macro-variable SQLOBS. La macro-variable SQLOBS se met automatiquement à jour après chaque PROC SQL. Pour extraire sa valeur, il faut d'abord actualiser son contenu.

Exemple :

```
/* actualisation de la valeur de SQLOBS */
PROC SQL;
  SELECT AGE
    FROM EMPLOYE
    WHERE AGE ^=.
;
QUIT ;
/* actualisation de la valeur de SYSOBS */
%LET nobs=&SQLOBS ;
```

La valeur contenue dans SQLOBS est copiée dans une nouvelles macro-variable nommée nobs.

La condition WHERE permet d'exclure toutes les observations pour lesquelles l'âge est manquant, puisqu'il s'agit d'extraire le nombre d'observations valides sur une variable. Mais lorsqu'on veut obtenir le nombre total d'observations dans la table, on peut simplement enlever la clause WHERE ou bien adopter la formulation suivante :

```
/* actualisation de la valeur de SQLOBS */
PROC SQL;
  SELECT *
    FROM EMPLOYE      ;
QUIT ;
/* actualisation de la valeur de SYSOBS */
%LET nobs=&SQLOBS ;
```

III.2.2.5. Stocker la liste des variables dans une macro-variable

La méthode la plus naturelle pour extraire la liste des variables dans une table est d'abord de réaliser une PROC CONTENTS et stocker ces résultats dans une table. La seconde étape sera de faire une PROC SQL sur la colonne contenant le nom des variables nommée NAME. Dès lors, on peut utiliser la clause INTO pour stocker sous forme de macro-variables. A ce niveau, on peut faire deux choix, soit stocker tous les éléments de la liste dans une macro-variable spécifique ou grouper tous les éléments dans une seule liste.

III.2.2.5.1. Stocker le nom de chaque variable dans macro-variable spécifique

La première étape consiste à faire PROC CONTENTS pour exporter la liste des variables sous formes de table. On a :

```
PROC CONTENTS DATA=MYLIB.MYDATA NOPRINT OUT=VARDATA; RUN;
```

Dans la seconde étape, en supposant qu'on ne connait pas à priori le nombre de variable dans la table, on fait d'abord PROC SQL simple pour connaître SQLOBS. Ainsi, on a :

```
    /*Etape 1*/
PROC SQL;
  SELECT NAME
    FROM VARDATA ;
QUIT ;
    /*Etape 2*/
%LET i=&SQLOBS ;
    /*Etape 3*/
PROC SQL;
  SELECT NAME
   INTO :mcvar1- :mcvar&i
    FROM VARDATA ;
QUIT ;
```

Lorsqu'on veut sélectionner uniquement les variables numérique (date compris), on ajoute la condition WHERE TYPE=1 ; à toutes les lignes ci-dessus. Ainsi, on a :

```
    /*Etape 1*/
PROC SQL;
  SELECT NAME
    FROM VARDATA
WHERE TYPE=1 ;
QUIT ;
    /*Etape 2*/
%LET i=&SQLOBS ;
    /*Etape 3*/
PROC SQL;
  SELECT NAME
   INTO :mcvar1- :mcvar&i
    FROM VARDATA
WHERE TYPE=1 ;
QUIT ;
```

Et lorsqu'il s'agit des variables en caractères, on spécifie WHERE=2 ;

III.2.2.5.2. Stocker le nom de toutes les variables dans une seule macro-variable

Pour créer une macro-variable contenant la liste de toutes les variables de la table, on utilise une formulation SQL à laquelle on ajoute la clause INTO avec l'instruction SEPARATED BY qui indique le symbole qui doit être utilisé pour délimiter chaque élément de la liste. Ainsi après avoir exécuté PROC CONTENTS sur la table initiale pour stocker les informations dans une table, on peut utiliser la formulation suivante :

```
PROC SQL;
  SELECT NAME
    INTO :mcvar SEPARATED BY ' '
    FROM VARDATA ;
    QUIT ;
```

Et lorsqu'on veut simplement sélectionner les variables numériques, on ajoute une clause WHERE telle que :

```
PROC SQL;
  SELECT NAME
    INTO :mcvar SEPARATED BY ' '
    FROM VARDATA
   WHERE TYPE=1 ;
    QUIT ;
```

Et lorsqu'il s'agit des variables en caractères, on spécifie WHERE=2 ;

III.3. Invoquer une macro-variable dans un programme

Après la définition d'une macro-variable et l'assignation de sa valeur par la méthode manuelle ou automatique, la seconde étape de l'utilisation d'une macro-variable est son invocation dans le programme.

Invoquer une macro-variable c'est solliciter la valeur de cette macro-variable à une étape dans l'exécution du programme. La manière d'invoquer une macro-variable dans le programme répond à un certain nombre de principes qui dépendent de la manière dont la macro-variable est spécifiée. En effet, on peut distinguer trois principales manières d'invoquer une macro-variable dans un programme :

- L'invocation directe
- L'invocation indirecte avec indexation de premier niveau
- L'invocation indirecte avec indexation de second niveau

III.3.1. Invocation directe : utilisation du simple Ampersand (&)

L'invocation directe d'une macro-variable consiste à solliciter la valeur de la macro-variable directement dans le programme sans aucune transformation préalable de sa valeur. Dans cette forme d'invocation la macro-variable se comporte comme un argument principal du programme.

Pour comprendre l'idée de l'invocation directe d'une macro-variable, considérons la liste des gagnants à un jeu auxquels on a proposé un séjour touristique aux Etats-Unis. Dix (10) destinations touristiques sont proposées aux gagnants : Cary, New York, Chicago, Los Angeles, Austin, Boston, Orlando, Dallas, Knoxville et Asheville.

Les informations sur les gagnants (Nom, âge, Sexe, Destination, etc…) sont enregistrées dans une table nommée GAGNANTS. La colonne Destination de cette table indique la ville qui a été choisie par le gagnant pour son séjour touristique.

Maintenant on souhaite par exemple afficher (en utilisant un PROC PRINT ou un PROC SQL), tous les gagnants qui ont choisi New York comme destination.

On décide par exemple de stocker le nom de cette ville sous forme de macro tel que :

```
%LET ville= New York;
```

Dès lors, on peut invoquer "directement" cette macro-variable lors de l'exécution de la tâche demandée. Pour cela on peut faire :

```
PROC PRINT DATA= GAGNANTS;
WHERE DESTINATION="&ville";
RUN;
```

Dans cette commande, on invoque directement la macro-variable ville par &ville. Les doubles guillemets sont utilisés ici simplement parce que la variable DESTINATION est une variable en caractères. Cela n'a donc pas de rapport direct avec l'invocation directe (Quoique, l'invocation d'une macro-variable à l'intérieur des guillemets peut avoir des conséquences sur l'affichage. Nous reviendrons plus tard sur l'utilisation des guillemets lors de l'invocation d'une macro-variable en chaînes de caractères. D'une manière générale, il est conseillé d'utiliser des guillemets doubles quand la valeur renvoyée par une macro-variable doit être traitée comme une chaîne de caractères).

Lorsqu'il s'agit de variables numériques (ex : sélectionner tous les gagnants âgés de plus 35 ans), on peut faire :

```
%LET val= 35;

PROC PRINT DATA= GAGNANTS;
WHERE AGE> &val;
RUN;
```

Au final, l'invocation directe consiste à utiliser directement la valeur de la macro-variable comme un argument principal dans les tâches du programme.

L'invocation directe se fait en indiquant le nom de la macro précédée du simple Ampersand (&).

Lorsque la valeur renvoyée est traitée comme une chaîne de caractères (stricte), on utilise les doubles guillemets. Dans les autres cas, la spécification est libre.

III.3.2. Invocation indirecte de premier niveau: utilisation du simple Ampersand (&)

Une invocation indirecte de la macro-variable avec indexation de premier niveau survient lorsque la valeur renvoyée par la macro-variable doit d'abord être associée (soit comme préfixe ou comme suffixe) à un mot ou toute autre valeur pour déterminer l'argument principal du programme. En effet, dans l'invocation directe précédemment présentée, la macro-variable joue le rôle de l'argument principal du programme. Alors que dans l'invocation indirecte avec indexation de premier niveau,

la macro-variable est un argument secondaire qui doit être associé d'abord à un mot précis pour former, au final, l'argument principal du programme. Dans cette association, la macro-variable peut être soit un préfixe, un suffixe ou située en milieu de mot. Les exemples présentés ci-après décrivent chacun de ces cas.

III.3.2.1. Cas où la macro-variable est invoquée comme un préfixe

D'abord, pour comprendre l'idée d'invocation indirecte avec indexation de premier niveau, considérons l'étape DATA suivante dans laquelle on veut créer dix nouvelles variables dans la table MYDATA :

```
DATA MYDATA ;
SET MYDATA ;
DESTINATION_1="Cary" ;
DESTINATION_2="New York" ;
DESTINATION_3="Chicago" ;
DESTINATION_4="Los Angeles" ;
DESTINATION_5="Austin" ;
DESTINATION_6="Boston" ;
DESTINATION_7="Orlando" ;
DESTINATION_8="Dallas" ;
DESTINATION_9="Knoxville" ;
DESTINATION_10="Asheville" ;
;
RUN ;
```

Dans cette étape DATA, les dix variables à créer ont le même préfixe (DESTINATION_), on peut alors penser à utiliser une macro-variable en amont et l'invoquer de manière indirecte comme préfixe afin former les noms des variables à créer. Dans ce cas, on fera comme suit :

```
/*Création de la macro-variable */
%LET d= DESTINATION_;
/* Création des variables finales*/
DATA MYDATA ;
SET MYDATA ;
&d.1="Cary";
&d.2="New York";
&d.3="Chicago";
&d.4="Los Angeles";
&d.5="Austin";
&d.6="Boston";
```

```
&d.7="Orlando";
&d.8="Dallas";
&d.9="Knoxville";
&d.10="Asheville";
RUN ;
```

Dans cette étape DATA, le mot DESTINATION_ a été remplacée par son équivalent en macro-variable d.

Etant donné que le mot DESTINATION_ est un préfixe, pour invoquer la macro-variable d avec le reste du texte on met un point (.) qui signifie la fin de la macro-variable dans le texte.

III.3.2.2. Cas où la macro-variable est invoquée comme un suffixe

Lorsque la macro-variable est un suffixe dans le mot, il n'est pas nécessaire de mettre un point. Par exemple, considérons l'exemple suivant :

```
DATA MYDATA ;
SET MYDATA ;
max_value=10 ;
min_value=4 ;
mean_value=6.5 ;
RUN ;
```

Dans cet exemple où l'on crée trois variables nommées respectivement max_value, min_value et mean_value, le mot "value" est un suffixe qu'on peut traduire par une macro-variable. Pour cela, on peut faire comme suit :

```
%LET v= VALUE ;
DATA MYDATA ;
SET MYDATA ;
MAX_&v=10 ;
MIN_&v=4 ;
MEAN_&v=6.5 ;
RUN ;
```

Cet exemple montre que lorsque la macro-variable est un suffixe, on peut l'invoquer dans la formulation de l'argument principal du programme sans utiliser le point.

III.3.2.3. Cas où la macro-variable est invoquée en milieu de texte

Lorsque la macro-variable intervient en milieu de texte, son invocation doit être suivie par un point afin de la distinguer du reste du texte. Il n'est pas nécessaire de la faire précéder par un point car avec le symbole &, le texte qui suit est interprétée comme une macro-variable. Cette interprétation s'arrête lorsque le premier point est rencontré.

Dans l'exemple ci-dessous, on souhaite créer trois variables comme suit :

```
DATA MYDATA ;
SET MYDATA ;
PREMIERE_DESTINATION_EAU="Je ne quoi"
DEUXIEME_DESTINATION_AIR="On dira tout"
TROISIEME_DESTINATION_FEU="On ne comprend plus rien"
RUN ;
```

Le mot _DESTINATION_ se répète dans les noms des trois variables à créer mais, il n'est ni préfixe, ni suffixe. Dès lors pour stocker cette valeur dans une macro-variable afin de raccourcir l'écriture, on peut faire comme suit :

```
%LET d= _DESTINATION_ ;
DATA MYDATA ;
SET MYDATA ;
PREMIERE&d.EAU="Je ne quoi"
DEUXIEME&d.AIR="On dira tout"
TROISIEME&d.FEU="Personne ne comprend plus rien"
RUN ;
```

En conclusion, lorsqu'une macro-variable est invoquée à l'intérieur d'un texte, il doit être suivi par un point pour marquer la fin du texte associé au nom de la macro.

III.3.2.4. Association de plusieurs macro-variables dans une même invocation de premier niveau

Il faut signaler qu'on peut associer plusieurs macro-variables dans une même invocation de premier niveau. Considérons l'exemple suivant :

```
DATA MYDATA ;
SET MYDATA ;
DESTINATION_TOURISTIQUE_1="Cary" ;
DESTINATION_TOURISTIQUE_2="New York" ;
```

```
DESTINATION_TOURISTIQUE_3="Chicago" ;
DESTINATION_TOURISTIQUE_4="Los Angeles" ;
DESTINATION_TOURISTIQUE_5="Austin" ;
DESTINATION_TOURISTIQUE_6="Boston" ;
DESTINATION_TOURISTIQUE_7="Orlando" ;
DESTINATION_TOURISTIQUE_8="Dallas" ;
DESTINATION_TOURISTIQUE_9="Knoxville" ;
DESTINATION_TOURISTIQUE_10="Asheville" ;
;
RUN ;
```

Pour raccourcir cette écriture, on peut décider de créer deux macro-variables nommées respectivement d et t définies comme suit ;

```
%LET d= DESTINATION_;
%LET t= TOURISTIQUE_;
```

Ainsi, on peut invoquer les deux macro-variables lors de la création des 10 variables comme suit :

```
/*Création de la macro-variable */
%LET d= DESTINATION_;
/* Création des variables finales*/
DATA MYDATA ;
SET MYDATA ;
&d&t.1="Cary";
&d&t.2="New York";
&d&t.3="Chicago";
&d&t.4="Los Angeles";
&d&t.5="Austin";
&d&t.6="Boston";
&d&t.7="Orlando";
&d&t.8="Dallas";
&d&t.9="Knoxville";
&d&t.10="Asheville";
RUN ;
```

Toutes les deux macro-variables font objet d'une même invocation de premier niveau. La macro d est un préfixe alors que la macro t se trouve en milieu de texte. On constate que l'invocation de la macro d n'est pas suivie par un point alors que celle de la macro t l'est. En effet, puisque la fin de l'invocation de la macro d est marquée par le début de l'invocation d'une autre macro (marquée par le &), alors, il

n'est pas nécessaire d'indiquer un point. En revanche, pour la macro t, la fin de son invocation est suivi par du texte qui n'est pas de la macro alors, il faut obligatoirement indiquer un point pour signifier la fin du texte d'invocation de la macro t.

III.3.3. Invocation indirecte de second niveau: utilisation du double Ampersand (&&)

Une invocation indirecte de la macro-variable avec indexation de second niveau survient lorsque la valeur renvoyée par la macro-variable doit d'abord être associée (soit comme préfixe ou comme suffixe) au nom d'autre macro-variable qui sera ensuite invoquée pour obtenir l'argument principal du programme. En effet, dans l'invocation indirecte de premier niveau précédemment présentée, lorsque la macro- est invoquée, la valeur obtenue est associée à un texte pour définir l'argument principal du programme. Mais dans l'invocation indirecte de second niveau, la valeur de la macro-variable obtenue est d'abord associée à un texte pour donner le nom d'une macro-variable qui sera ensuite invoquée afin d'aboutir finalement à l'argument principal du programme (c'est-à-dire la valeur entrant dans l'exécution de la tâche).

Tout comme dans l'invocation de premier niveau, dans l'invocation du second niveau la macro-variable peut intervenir soit en tant que préfixe, en tant que suffixe ou intervenir en milieu de texte. Ces trois cas sont détaillés ci-dessous.

III.3.3.1. Cas où la macro-variable est invoquée comme un préfixe

Pour introduire ce cas, supposons qu'on veuille créer, en utilisant une étape DATA, une variable qui s'appelle DESTINATION_TOURISTIQUE1 qui prend la valeur "New York". Utilisons pour cela deux macros-variables définies comme suit :

```
%LET city= ville;
%LET ville_dest= DESTINATION_TOURISTIQUE1;
```

D'abord, on constate que la valeur de la macro-variable city est ville. On constate aussi que la valeur de la macro-variable ville_dest est DESTINATION_TOURISTIQUE1. Cela signifie qu'on peut créer la variable la variable DESTINATION_TOURISTIQUE1 dans une étape DATA en invoquant la macro ville_dest. Par ailleurs, on peut constater que le nom de la macro-variable ville_dest est préfixé par le mot ville qui correspond à la valeur de la macro-variable city. Dès lors, on peut faire la conclusion suivante : Pour créer la variable DESTINATION_TOURISTIQUE1, il faut invoquer d'abord la

macro-variable ville_dest et pour constituer ce nom on peut utiliser le mot dest en le préfixant par la valeur de la macro-variable city.

Bien que cet exemple soit moins intuitif, il représente un cas typique d'utilisation d'une invocation de second niveau. La macro-variable étant un préfixe dans le nom de la macro-variable ville_dest, on va utiliser la formulation suivante :

```
%LET city= ville;
%LET ville_dest= DESTINATION_TOURISTIQUE1;
DATA MYDATA;
SET MYDATA ;
&&&city._dest="New York ";
RUN ;
```

Dans cette formulation, le macro-processeur de SAS interprète d'abord les deux premiers ampersands pour les convertir en un seul (voir le document de référence SAS pour sur les macros). Ensuite, le troisième ampersand associé, au mot city, sera interprété jusqu'au point. Cette interprétation donne la valeur de la macro-variable city qui est égale à ville. Ensuite le mot _dest est interprété comme tel car il ne contient aucun ampersand.

Le résultat final de cette interprétation de premier niveau sera alors : &ville_dest. Ensuite SAS recommence l'interprétation dès le début. Ainsi, en interprétant l'unique ampersand et le texte associé ville_dest, SAS rend compte qu'il s'agit d'une macro-variable, alors il renvoie la valeur associée à cette macro-variable. Ici, la valeur renvoyée est DESTINATION_TOURISTIQUE1. Ainsi, le résultat final de l'invocation sera :

```
DATA MYDATA;
SET MYDATA ;
DESTINATION_TOURISTIQUE1="New York ";
RUN ;
```

En somme, dans une invocation indirecte de second niveau, lorsque la macro-variable initiale est un préfixe alors l'invocation se fait en commençant par une double ampersand suivi par l'invocation de la macro-variable avec un ampersand. Au final, cela fera trois ampersands en début d'invocation. Il ne faut pas oublier de marquer la fin de l'invocation de la macro-variable initiale par un point.

III.3.3.2. Cas où la macro-variable est invoquée comme un suffixe

Pour introduire ce cas, supposons qu'on veuille créer, en utilisant une étape DATA, une variable qui s'appelle DESTINATION_TOURISTIQUE1 qui prend la valeur "New York". Utilisons pour cela deux macros-variables définies comme suit :

```
%LET city= ville;
%LET dest_ville= DESTINATION_TOURISTIQUE1;
```

D'abord, on constate que la valeur de la macro-variable city est ville et que la valeur de la macro-variable dest_dest est DESTINATION_TOURISTIQUE1. Ce qui signifie qu'on peut créer la variable DESTINATION_TOURISTIQUE1 en invoquant la macro dest_ville. De plus on peut aussi constater que le nom de la macro-variable dest_ville est préfixé par le mot ville qui correspond à la valeur de la macro-variable city. Dès lors, pour créer la variable DESTINATION_TOURISTIQUE1 il faut invoquer la macro-variable dest_ville et pour constituer ce nom on peut utiliser le mot dest en le suffixant par la valeur de la macro-variable city.

Ainsi, la macro-variable étant un suffixe dans le nom de la macro-variable dest_ville, on va utiliser la formulation suivante :

```
%LET city= ville;
%LET dest_ville= DESTINATION_TOURISTIQUE1;
DATA MYDATA;
SET MYDATA ;
&&dest_&city="New York";
RUN ;
```

Dans cette formulation, le macro-processeur interprète d'abord les deux premiers ampersands pour les convertir en un seul. Ensuite le mot dest_ est interprété comme tel car il ne contient aucun ampersand. Ensuite, l'unique ampersand sera associé au mot city pour être interprété. Cette interprétation donne la valeur de la macro-variable city qui est égale à ville.

Le résultat final de cette interprétation de premier niveau sera alors : &dest_ville. Ensuite SAS recommence l'interprétation dès le début. Ainsi, en interprétant l'unique ampersand et le texte associé dest_ville, SAS se rend compte qu'il s'agit d'une macro-variable, alors la valeur associée à cette macro-variable est renvoyée. Ici, la valeur renvoyée est DESTINATION_TOURISTIQUE1. Ainsi, le résultat final de l'invocation sera :

```
DATA MYDATA;
SET MYDATA ;
DESTINATION_TOURISTIQUE1="New York ";
RUN ;
```

En conclusion, dans une invocation indirecte de second niveau, lorsque la macro-variable initiale est un suffixe alors l'invocation se fait en commençant par une double ampersand suivi par l'invocation de la macro-variable avec un ampersand. Ici puisque la macro-variable initiale est un suffixe, il n'est pas nécessaire de marquer sa fin par un point.

III.3.3.3. Cas où la macro-variable est invoquée en milieu de texte

Pour introduire ce cas, considérons toujours l'exemple où l'on veut créer, en utilisant une étape DATA, une variable nommée DESTINATION_TOURISTIQUE1 et qui prend la valeur "New York". Et utilisons pour cela deux macros-variables définies comme suit :

```
%LET city= ville;
%LET tour_ville_dest= DESTINATION_TOURISTIQUE1;
```

D'abord, la valeur de la macro-variable city est ville et e la valeur de la macro-variable tour_ville_dest est DESTINATION_TOURISTIQUE1. Cela signifie aussi qu'on peut créer la variable DESTINATION_TOURISTIQUE1 en invoquant la macro tour_ville_dest. De plus on peut aussi constater que le nom de la macro-variable tour_ville_dest contient le mot ville qui correspond à la valeur de la macro-variable city. Ainsi, pour créer la variable DESTINATION_TOURISTIQUE1 il faut invoquer d'abord la macro-variable tour_ville_dest et pour constituer ce nom on peut utiliser le mot tour_$$$$_dest indiquant au milieu la valeur de la macro-variable city.

Ainsi, la macro-variable étant en milieu de mot dans le nom de la macro-variable tour_ville_dest, on va utiliser la formulation suivante :

```
%LET city= ville;
%LET tour_ville_dest = DESTINATION_TOURISTIQUE1;
DATA MYDATA;
SET MYDATA ;
&&tour_&city._dest="New York";
RUN ;
```

Dans cette formulation, le macro-processeur interprète d'abord les deux premiers ampersands pour les convertir en un seul. Ensuite le mot tour_ est interprété comme tel car il ne contient aucun ampersand. Ensuite, l'unique ampersand quit ce mot sera

associé au mot city pour être interprété comme une macro-variable. Cette interprétation s'arrête au niveau du point indiqué à cet effet. L'interprétation de &city donne la valeur de la macro-variable qui est égale à ville. Ensuite le mot _dest est interprété comme tel car il ne contient aucun ampersand.

Le résultat final de cette interprétation de premier niveau sera alors : &tour_ville_dest. Ensuite SAS recommence l'interprétation dès le début. Ainsi, en interprétant l'unique ampersand et le texte associé tour_ville_dest, SAS se rend compte qu'il s'agit d'une macro-variable, alors la valeur associée à cette macro-variable est renvoyée. Ici, la valeur renvoyée est DESTINATION_TOURISTIQUE1. Ainsi, le résultat final de l'invocation sera :

```
DATA MYDATA;
SET MYDATA ;
DESTINATION_TOURISTIQUE1="New York ";
RUN ;
```

En conclusion, dans une invocation indirecte de second niveau, lorsque la macro-variable initiale est en milieu de texte alors l'invocation se fait en commençant par une double ampersand suivi par l'invocation de la macro-variable avec un ampersand. Cette invocation est terminée par un point pour marquer la fin du texte traduisant le nom de la macro-variable.

III.3.3.4. Association de plusieurs macro-variables dans une même invocation de second niveau

Considérons toujours l'exemple où l'on veut créer, en utilisant une étape DATA, une variable nommée DESTINATION_TOURISTIQUE1 et qui prend la valeur "New York".

Et utilisons pour cela deux macros-variables définies comme suit :

```
%LET val1= tour_;
%LET val2= ville_;
%LET tour_ville_dest= DESTINATION_TOURISTIQUE1;
```

D'abord, la valeur de la macro-variable val1 est tour_, la valeur de la macro-variable val2 est ville_ et la valeur de la macro-variable tour_ville_dest est DESTINATION_TOURISTIQUE1. Cela signifie aussi qu'on peut créer la variable DESTINATION_TOURISTIQUE1 en invoquant la macro tour_ville_dest. De plus, on peut aussi constater que le nom de la macro-variable tour_ville_dest contient le mot tour_ qui correspond à la valeur de la macro-variable val1 et le mot ville_ qui

correspond à la valeur de la macro-variable val2. Ainsi, pour créer la variable DESTINATION_TOURISTIQUE1 dans la table, il faut d'abord invoquer la macro-variable tour_ville_dest et pour constituer ce nom on peut utiliser le mot dest en le préfixant respectivement par les macro-variables val1 et val2.

Dans ce cas, la macro-variable val1 sera un préfixe et la macro-variable val2 sera en milieu de texte. Pour les invoquer, on adoptera la formulation suivante :

```
%LET val1= tour_;
%LET val2= ville_;
%LET tour_ville_dest= DESTINATION_TOURISTIQUE1;
DATA MYDATA;
SET MYDATA ;
&&&val1&val2.dest="New York";
RUN ;
```

Toutes les deux macro-variables val1 et val2 font objet d'une même invocation de second niveau. La macro val1 est un préfixe alors que la macro val2 se trouve en milieu de texte. On constate que l'invocation de la macro val2 n'est pas suivie par un point alors que celle de la macro val2 l'est. En effet, puisque la fin de l'invocation de la macro val2 est marquée par le début de l'invocation d'une autre macro (marquée par le &), alors, il n'est pas nécessaire d'indiquer un point. En revanche, pour la macro val2, la fin de son invocation est suivi par du texte qui n'est pas de la macro alors, il faut obligatoirement indique un point pour signifier la fin du texte d'invocation de la macro t.

En termes d'interprétation, le macro-processeur interprète d'abord les deux premiers ampersands pour les convertir en un seul. Ensuite, le troisième ampersand, associé au mot val1, sera interprété comme une macro-variable. Cette interprétation donne la valeur de la macro-variable val1 qui est égale à tour_. Ensuite, le quatrième ampersand, associé au mot val2, sera interprété comme une macro-variable jusqu'au point qui marque la fin de la macro val2. Cette interprétation donne la valeur de la macro-variable val1 qui est égale à ville_. Ensuite le mot dest est interprété comme tel car il ne contient aucun ampersand.

Le résultat final de cette interprétation de premier niveau sera alors : &tour_ville_dest. Ensuite SAS recommence l'interprétation dès le début. Ainsi, en interprétant l'unique ampersand et le texte associé tour_ville_dest, SAS se rend compte qu'il s'agit d'une macro-variable, alors la valeur associée à cette macro-variable est renvoyée. Ici, la

valeur renvoyée est DESTINATION_TOURISTIQUE1. Ainsi, le résultat final de l'invocation sera :

```
DATA MYDATA;
SET MYLIB.MYDATA ;
DESTINATION_TOURISTIQUE1="New York ";
RUN ;
```

III.3.4. Invocation d'une macro-variable en milieu d'une chaine de caractères: utilisation de guillemets simples ' ' et de guillemets doubles " "

Pour renvoyer la valeur d'une macro-variable au milieu d'une chaîne de caractères, cette chaîne de caractères doit être spécifiée à l'intérieur des guillemets doubles et non des guillemets simples. Par exemple, en considérant la macro-variable suivante nommée joursem1 définie comme suit :

```
%LET joursem1 = Lundi Mardi Mercredi Jeudi Vendredi Samedi Dimanche;
```

Pour afficher les valeurs de cette macro avec la phrase telle que « les jours de la semaine sont… », on fait :

```
%PUT "Les jours de la semaine sont &joursem1";
```

On obtient alors le résultat suivant :

"Les jours de la semaine sont Lundi Mardi Mercredi Jeudi Vendredi Samedi Dimanche"

Mais en invoquant la macro-variable telle que :

```
%PUT 'Les jours de la semaine sont &joursem1' ;
```

On obtient exactement la formulation d'origine c'est-à-dire :

'Les jours de la semaine sont &joursem1'

Il est donc important de savoir dans quel contexte faut-il utiliser les doubles guillemets et les simples guillemets. D'une manière générale, les doubles guillemets fonctionnent partout où les simples guillemets fonctionnent. Mais la réciproque n'est pas toujours vraie.

Il faut aussi noter que pour traiter la valeur d'une macro-variable comme du texte pur, ce texte doit être spécifié à l'intérieur des guillemets doubles. Par exemple, on

veut créer une variable qui s'appelle ville qui doit prendre la valeur New-York. Pour cela, on peut faire

```
%LET city= New-York;
DATA MYDATA;
SET MYDATA ;
ville="&city";
RUN ;
```

Puisque la variable ville est une variable en caractères, la macro-variable city dont la valeur est New-York doit être invoquée ici entre double guillemet.

En utilisant les simples guillemets, &city sera renvoyée comme valeur et en utilisant pas du tout de guillemets on obtient des résultats encore plus étranges.

III.4. Utilisation des macro-fonctions

D'une manière générale, une macro-fonction est une fonction SAS principalement conçue pour être utilisée dans le cadre d'un macro-programme. Les macro-fonctions sont destinées spécialement à traiter les macro-variables. Elle se distingue d'une fonction classique par le symbole % indiqué devant le nom de la fonction. Toutefois, malgré cette spécificité, une macro-fonction peut être à la fois utilisée dans un macro-programme mais également en code libre.

Le tableau ci-dessous fournit la liste des principales macro-fonctions actuellement disponibles sous SAS (exceptions faite des nombreuses autres fonctions élaborées par des contributeurs utilisateurs) :

Macro-fonction	Description
%BQUOTE, %NRBQUOTE	Masque les caractères spéciaux et les opérateurs mnémoniques dans une valeur résolue lors de l'exécution de la macro.
%EVAL	Évalue les expressions arithmétiques et logiques en utilisant l'arithmétique des nombres entiers.
%INDEX	Renvoie la position du premier caractère dans une chaîne.
%LENGTH	Renvoie la longueur d'une chaîne.
%QUOTE, %NRQUOTE	Masque de caractères spéciaux et les opérateurs mnémoniques dans une valeur résolue lors de l'exécution de la macro. Les guillemets incomplets (" "), les apostrophes (') et les parenthèses (()) doivent être précédés par un % pour être caché par la fonction.
%SCAN, %QSCAN	Recherche d'un mot spécifié par son numéro. Quant à %QSCAN, elle masque les caractères spéciaux et les opérateurs mnémoniques dans son résultat.
%STR, %NRSTR	Masque les caractères spéciaux et les opérateurs mnémoniques dans le texte constant lors de la compilation de la macro. Les guillemets incomplets (" "), les apostrophes (') et les parenthèses (()) doivent être précédés par un % pour être caché par la fonction.
%SUBSTR, %QSUBSTR	Rencoie une chaîne d'une chaîne de caractères. %QSUBSTR masque les caractères spéciaux et les opérateurs mnémoniques dans son résultat.
%SUPERQ	Masques tous les caractères spéciaux et les opérateurs mnémoniques lors de l'exécution de la macro, mais empêche la résolution de la valeur.
%SYMEXIST	Renvoie une indication quant à savoir si le nom de la macro variable existe ou pas.
%SYMGLOBL	Renvoie une indication quant à savoir si le nom de macro variable est de type global ou pas.
%SYMLOCAL	Renvoie une indication quant à savoir si le nom de macro variable est de type local ou pas.
%SYSEVALF	Évalue les expressions arithmétiques et logiques en utilisant l'arithmétique des nombres décimaux (voir %EVAL).
%SYSFUNC, %QSYSFUNC	Exécute des fonctions SAS ou des fonctions écrites par l'utilisateur. %QSYSFUNC masque les caractères spéciaux et les opérateurs mnémoniques dans le résultat.
%UNQUOTE	Démasque tous les caractères spéciaux et les opérateurs mnémotechniques pour une valeur.
%UPCASE, %QUPCASE	Convertir les caractères en majuscules. %QUPCASE masque les caractères spéciaux et les opérateurs mnémoniques dans son résultat.

III.5. Introduction aux macro-programmes

Un macro-programme est un ensemble d'instructions visant à exécuter une ou plusieurs tâches. Il est généralement constitué d'un ensemble de macro-commandes formée lui-même d'un ensemble de macro-variables, de macro-fonctions ainsi que des procédures classiques (DATA et PROC).

L'élaboration d'un macro-programme SAS peut s'avérer utile dans plusieurs contextes de travail sur les données car il permet d'améliorer considérablement l'efficience du programme général. En générale, lorsqu'il s'agit d'effectuer plusieurs opérations de même nature sur un ensemble de variables, un ensemble d'observations ou un ensemble de tables, on peut penser à l'écriture d'un programme plutôt que de choisir un code libre. Dans cette section, nous donnerons quelques cas où l'utilisation macro-programme apparaît comme la manière la plus efficiente pour exécuter certaines tâches.

III.5.1. Structure de base d'un macro-programme : Définition et exécution

Dans sa structure la plus simple, un macro-programme est constitué de trois principales parties : la déclaration du début de programme, les instructions du programme (le corp du programme) et la déclaration de fin de programme. Chacune de ces parties est caractérisée par des éléments particuliers. Les lignes ci-dessous donnent la structure générale d'un programme :

```
%MACRO MYMACRO(arg1=, arg2=,..., argN=);
instruction1;
Instruction2;
…
…
…
instructionN;
%MEND MYMACRO;
```

La première ligne %MACRO MYMACRO déclare une nouvelle macro nommée MYMACRO définie sur N arguments arg1, arg2,..., argN. Les arguments d'un programme sont les informations initiales qu'on fournit au programme et qu'il doit utiliser pour exécuter la ou les tâches demandées. Un programme peut avoir un ou plusieurs arguments. Par exemple, si le programme est conçu pour exécuter un certain nombre d'opérations sur un ensemble de tables de données, alors le premier

argument pourrait être la liste des tables à traiter. Tandis que le second argument pourrait être la liste des variables concernées, etc. D'une manière générale, les arguments d'un programme sont traités par la suite comme des macro-variables.

De même qu'il peut avoir plusieurs arguments, un macro-programme peut avoir plusieurs instructions. Les instructions définies après la déclaration de début de programme constitue le corps du programme. Ces instructions accueillent d'abord les arguments en tant que macro-variables et les utilise conformément aux directives fixées par le programmeur. Les instructions sont généralement formulées en utilisant les macro-fonctions mais surtout les procédures classiques (DATA et PROC). Par exemple, un utilisateur souhaite STANDARDISER toutes les variables numériques sur les tables qu'il dispose. L'instruction sera alors d'ouvrir chaque table, d'identifier les variables numériques et d'appliquer PROC STANDARD sur chacune d'entre elles. Bien entendu, le principal argument de ce programme sera la liste de toutes les tables disponibles dans le répertoire fixé (notons que lorsque la liste des fichiers ne peut pas être spécifiée manuellement, on peut se servir des fonctions disponibles pour constituer automatiquement cette liste. Dans ce cas l'argument principal du programme sera défini selon un autre critère).

Quant à la troisième partie du programme, elle est matérialisée par %MEND qui signifie la fin de la définition du programme. Dans la structure présentée ci-dessous, nous avons écrit %MEND MYMACRO qui signifie la fin de la définition de la macro MYMACRO. Notons qu'il n'est pas nécessaire d'indiquer le nom de la macro après %MEND car on suppose qu'aucune macro n'a été définie à l'intérieur de la macro MYMACRO. Donc %MEND qui sera rencontré lors de l'exécution sera nécessairement pour MYMACRO. Par contre, si une macro a été déclarée (il ne s'agit pas ici d'invoquer) à l'intérieur de la macro MYMACRO alors, il faut absolument indique la fin de la déclaration par %MEND MYMACRO. Mais il est très rare de rencontrer des cas où une macro est définie à l'intérieur d'une autre macro. Généralement les macros sont séquentiellement définies de sorte que les unes puissent invoquer les autres pour exécuter correctement le programme.

L'exemple ci-dessous définit un programme qui s'appelle MYPLOT qui permet de tracer le nuage de points entre deux variables.

```
%MACRO MYPLOT(xvar=,yvar=,source=);
PROC PLOT DATA=&source;
PLOT &yvar*&xvar ="°";
RUN;
```

%MEND MYPLOT;

Ce programme a trois arguments et une instruction.

Les trois arguments sont xvar, yvar et source. En analysant le corps du programme, on constate que le troisième argument est invoqué dans la ligne de commande PROC PLOT DATA=&source. Ce qui signifie alors que *source* est une macro-variable qui doit renvoyer le nom de la table de données SAS.

La seconde ligne de commande PLOT &yvar*&xvar ="°" invoque les deux premiers arguments xvar et yvar. Cette ligne correspond à une PROC classique construite ici simplement avec les macro-variables. En analysant la structure de cette ligne yvar doit correspondre à la variable représentée sur l'axe des ordonnées alors que xvar doit être représentée sur l'axe des abscisses. Notons simplement que l'égalité ="°" est une option classique de PROC PLOT qui signifie que les points doivent représentés par ce symbole. Bien entendu, on pouvait utiliser un autre symbole.

Maintenant que nous avons compris la structure de ce programme, la seconde étape sera son exécution.

Pour exécuter un programme, on écrit le nom du programme avec ses arguments en le faisant précédée par le symbole%.

%***MYMACRO***(arg1=val1, arg2=val2,..., argN=valN); **quit**;

Exécutons ce principe en utilisant la macro MYPLOT en considérant xvar comme la variable revenu, la variable yvar comme le niveau d'éducation et la source de donnée comme MYLIB.MYDATA. Alors on a :

%***MYPLOT***(xvar=revenu, yvar=education, source=mylib.mydata); **quit**;

En sélectionnant cette ligne et en l'exécutant, Eh hop ! le graphique déboule comme par magie !

Sinon, pour être tout à fait sérieux, l'invocation du programme peut aussi être spécifiée autrement. En effet, au lieu d'attribuer les valeurs aux arguments en utilisant le signe d'égalité ; on peut aussi omettre cette indication et adopter la formulation suivante :

%***MYPLOT***(revenu, education, mylib.mydata); **quit**;

SAS reconnaît parfaitement que la première valeur indiquée correspond à la valeur du premier argument, que la seconde valeur indiquée correspond au deuxième argument, etc... Attention dans ce cas à ce que des valeurs ont été prévues pour tous

les arguments. Les arguments qui n'auront pas de valeurs correspondantes auront par défaut la valeur NULL (vide). Ce qui risque de compromettre l'exécution du programme. C'est d'ailleurs pour faire face à de telles situations que des valeurs par défaut sont définies dans beaucoup de programmes de traitements et d'analyses de données. Pour fixer les valeurs par défaut des arguments d'un programme celles-ci doivent être déclarées en même temps que le programme. Par exemple, en reprenant le programme MYPLOT ci-dessous, on peut définir des valeurs par défaut comme suit :

```
%MACRO MYPLOT(xvar=revenu,yvar=education,source=mylib.mydata);
PROC PLOT DATA=&source;
PLOT &yvar*&xvar ="°";
RUN;
%MEND MYPLOT;
```

Ainsi lorsque la macro MYPLOT est invoquée sans argument ou avec un des arguments manquants, SAS utilise les valeurs par défaut pour continuer l'exécution du programme.

Toutefois, il est fortement déconseillé d'utiliser des valeurs par défaut lorsque ces valeurs ne sont pas universelles c'est-à-dire quel que soit le contexte d'application du programme.

En effet, dans un programme, les valeurs par défaut ne doivent être définies que pour les arguments optionnels. Il est conseillé de rendre obligatoire certains arguments du programme et de renvoyer des messages d'erreur lorsque les valeurs ne remplissent pas un certain nombre de conditions bien définies (par exemple, l'argument ne doit pas être NULL, on ne doit pas utiliser une variable en caractère là une variable numérique est prévue, etc…).

S'agissant des valeurs par défaut, il est conseillé de les définir uniquement pour des arguments optionnels. Ces arguments concernent généralement les paramètres constants dans une modélisation par exemple. Supposons par exemple, qu'on veuille tester la stationnarité d'une série temporelle en utilisant le test Augmented Dickey-Fuller, c'est l'utilisateur qui décide du choix du nombre de retard à inclure dans l'équation de test. Néanmoins, celui qui a programmé le package permettant de réaliser ce test peut fixer une valeur par défaut de sorte que même si l'utilisateur n'indique pas le nombre de retard, que le test puisse être réalisé dans un premier. Ensuite, au vu de ces résultats primaire l'utilisateur peut se raviser pour fixer dans un second temps un nombre de retard qui lui semble optimale.

Au final, un bon programme doit être construit autour des arguments obligatoires et optionnels mais aussi un ensemble de contrôles et de messages d'erreur obligeant l'utilisateur à fournir les bonnes informations aux programmes. Malheureusement dans ce document, nous n'allons pas trop nous appesantir sur les aspects de programmation. Nous abordons simplement le strict minimum permettant de faire face aux problèmes courants de traitement de données.

III.5.2. Utilisation des macro-fonctions et des boucles %DO LOOP dans la construction de macro-programmes

La construction d'un macro-programme n'est intéressant que s'il est conçu pour exécuter des tâches nécessitant l'utilisation des macro-fonctions mais aussi des boucles %DO LOOP, %WHILE ou %UNTIL. Les macro-programmes sont généralement des successions et des imbrications des boucles %DO LOOP qu'il faut pouvoir déceler afin de comprendre la structure du programme.

Dans cette section, nous montrons comment utiliser les macro-fonctions et les boucles %DO afin de construire un macro-programme évolué. Pour cela, nous partirons des cas les plus simples qui peuvent être facilement généralisés.

Pour commencer, considérons l'exemple où l'on souhaite calculer (en utilisant DATA STEP classique) dix variables définies comme suit :

```
DATA MYDATA ;
SET MYDATA ;
DESTINATION_1="Cary";
DESTINATION_2="New-York";
DESTINATION_3="Chicago";
DESTINATION_4="Los-Angeles";
DESTINATION_5="Austin";
DESTINATION_6="Boston";
DESTINATION_7="Orlando";
DESTINATION_8="Dallas";
DESTINATION_9="Knoxville";
DESTINATION_10="Asheville";
RUN ;
```

On peut rendre efficiente ce code en utilisant une macro-commande (un mini macro-programme). Pour cela, nous devons d'abord définir des macro-variables.

En regardant ce code, on peut définir deux macro-variables : une première qui contient la liste des villes et une seconde qui est en fait un compteur allant de 1 à 10. La macro-variable correspondant à ce compteur est un suffixe pour le mot DESTINATION_ tandis que l'ensemble des formeront une macro-variable. Essayons maintenant d'écrire cette tâche sous forme de macro-commande. Nous proposons le programme suivant :

```
/*Définition de la macro*/
%macro macro_cities;
%LET cities= Cary New-York Chicago Los-Angeles Austin Boston Orlando Dallas
Knoxville Asheville;
%let k = %SYSFUNC(countw(&cities, ' '));
DATA MYDATA;
SET MYDATA;
%Do i=1 %to &k ;
%let city_&i = %scan(&cities,&i,' ');
DESTINATION&i="&&city_&i";
%end ;
Run;
%mend macro_cities;
/*Execution de la macro*/
%macro_cities; quit;
```

Comme on peut le constater le programme construit ici est un programme qui ne demande pas de spécifier les arguments. Tous les arguments sont déjà disponibles dans le corps du programme. La structure du programme est le suivant :

D'abord, on stocke la liste des villes dans une macro nommée cities.

Ensuite, nous créons une macro-variable nommée k qui compte le nombre de mots rencontré dans la macro-variable cities. Ce comptage est fait en utilisant la macro-fonction countw(). Mais pour exécuter cette fonction, on utilise une autre macro fonction %SYSFUNC() dont le rôle est d'exécuter toutes les macro-fonctions qu'on lui fournit comme arguments. En effet, étant donné que certaines macro-fonctions (telles que celles écrites par les utilisateurs) ne s'exécutent pas automatiquement, on est souvent amené à utiliser %SYSFUNC ou %QSYSFUNC pour exécuter ces types de macros-fonctions. C'est le cas ici pour la macro-fonction COUNTW. Ainsi, le nombre de mots trouvés par la fonction COUNTW est stocké dans la macro-variable k. Ce nombre représente alors la valeur maximale de l'indice de la boucle. Toutefois, avant définir la boucle, nous invoquons une étape DATA afin d'activer la table à modifier.

Ainsi, on ouvre la boucle avec la clause %DO %TO en introduisant un compteur i variant de 1 à k. Notons que, par définition dans un macro-programme, un compteur défini par %Do est une macro-variable. Il sera donc traité comme pendant toute sa durée d'utilisation.

Une fois que le compteur i est déclaré, on crée une nouvelle macro-variable indexée sur i qui contient le mot se trouvant à i-ième position dans la liste définie par la macro-variable cities. Cet i-ième mot est renvoyé en utilisant la macro-fonction %SCAN(). Une fois que ce mot est renvoyé, on créer la i-ième variable correspondant à la i-ième destination à laquelle on assigne la valeur contenue dans la macro city&i (c'est-à-dire la i-ième ville de la liste des villes définie par cities. Au final, en exécutant cette macro-commande, on obtient les mêmes résultats qu'un DATA STEP avec code libre. Cependant, on peut constater que la macro-commande telle que spécifiée a presque le même nombre de lignes que la première formulation en code libre (sinon un peu plus). On peut alors se demander quelle a été le véritable apport de l'utilisation de la macro-commande. En effet, l'utilisation de la macro-commande est avantageuse que lorsque le nombre de tâches à réaliser devient important. Par exemple supposons maintenant qu'il y ait 5 tables au lieu d'une et qu'il faille réaliser les mêmes opérations sur les mêmes tables. A ce moment, l'écriture en macro-commande prend systématiquement avantage par rapport au code. Par exemple, le code ci-dessous réalise les mêmes opérations en considérant cinq tables nommée tab1 jusqu'à tab5.

```
/*Définition de la macro*/
%macro macro_cities;
%LET cities= Cary New-York Chicago Los-Angeles Austin Boston Orlando Dallas
Knoxville Asheville;
%let k = %SYSFUNC(countw(&cities, ' '));
%Do i=1 %to 5 ;
DATA tab&i;
SET tab&i;
%Do j=1 %to &k ;
%let city_&j = %scan(&cities,&j,' ');
DESTINATION&j="&&city_&j";
Run;
%end ;
%end ;
%mend macro_cities;
/*Execution de la macro*/
%macro_cities; quit;
```

Cette macro-commande est bien la preuve que l'efficience d'un macro-programme augmente lorsque le nombre de tâches à effectuer est très élevé.

En conclusion, le choix de l'outil de codage doit être fait en fonction de l'efficience. Par conséquent l'utilisation de macro-programmes n'est pas nécessaire dans tous les contextes. Un code libre peut souvent fournir les meilleurs résultats de par leur clarté et leur simplicité.

CHAPITRE IV : LES METHODES D'ANALYSES DE DONNEES

Ce chapitre est consacré à la description des procédures SAS permettant la mise en œuvre des principales méthodes d'analyses de données se présentant sous forme de tables. L'objectif ici n'est pas de faire une présentation détaillée de ces méthodes en elles-mêmes mais simplement de donner un aperçu général sur leurs mises en œuvre sous SAS. Etant donné que la description de ces méthodes relève du seul domaine de la statistique théorique, nous supposons alors à priori que l'utilisateur a une idée précise de la méthode qui répond à ses préoccupations d'analyse. Dans ce chapitre, nous nous limiterons seulement à une brève description des procédures SAS permettant de réaliser telle ou telle analyse.

Le chapitre est organisé selon trois niveaux d'analyse : les analyses descriptives et les analyses multidimensionnelles (qui sont essentiellement d'ordre exploratoires) mais aussi les démarches de modélisations.

IV.1. Statistiques descriptives

IV.1.1. Analyses descriptives univariées

La description unidimensionnelle des données est la toute première phase de l'analyse des données. Elle est mise en œuvre à travers la présentation des indicateurs statistiques de base, des tableaux de synthèse ainsi que des représentations graphiques sur chaque variable afin de résumer l'information contenues dans chaque variable. Le choix du type de statistique à présenter dépend de la nature des variables en jeux (variables quantitatives ou variables qualitatives).

IV.1.1.1. Analyses descriptives univariées sur des variables quantitatives

IV.1.1.1.1. Tableaux de statistiques descriptives

D'une manière générale, pour réaliser la description univariée d'une variable quantitative, on utilise les caractéristiques de tendances centrales et de dispersion (moyenne, écart-type, etc..).

Sous SAS, on peut obtenir ces statistiques en utilisant PROC MEANS ou PROC SUMMARY. L'exemple ci-dessous est une illustration :

```
PROC MEANS DATA=MYTABLE MEAN MEDIAN STD  STDERR MIN MAX RANGE   N
KURT SKEW PROBT MAXDEC=3 ORDER=FORMATTED FW=8  VARDEF=DF ;
VAR    MYVAR        ;
/*OUTPUT OUT=MYSTAT MEAN= SUM= /AUTONAME; */
RUN ;
```

Cette commande présente respectivement la moyenne, la médiane, l'écart-type, l'erreur standard de la moyenne, le minimum, le maximum, l'étendue, le nombre d'observations, le coefficient d'aplatissement, le coefficient d'asymétrie, et la p-value du test de nullité de la moyenne de la variable MYVAR. Il y a également quelques options supplémentaires liées à la mise en forme des statistiques présentées. Notons aussi qu'on pouvait utiliser la procédure PROC SUMMARY qui est d'ailleurs un peu plus flexible que PROC MEANS. On l'utilisera dans d'autres contextes.

Par ailleurs, il faut noter aussi, pour la plu part des PROC SAS laissent la possibilité d'exporter les résultats obtenus dans une nouvelle table en utilisant l'instruction OUTPUT OUT=.

IV.1.1.1.2. Représentations graphiques de la distribution de variables quantitatives : Histogramme et Box-plot

Les représentations graphiques les plus couramment utilisées dans une analyse univariées pour illustrer la distribution des variables quantitatives sont l'histogramme et la boîte à moustaches. Les deux exemples ci-dessous

Histogramme

```
PROC UNIVARIATE DATA=MYTABLE  ;
HISTOGRAM   /   NORMAL CTEXT= BLUE  VAXISLABEL="Fréquence" ;
INSET MEAN="Moyenne" P50="Médiane" QRANGE="IIQ"  STD SKEWNESS KURTOSIS
NORMALTEST PNORMAL /HEADER="Caractéristiques POSITION=nw  ;
VAR MYVAR ;
RUN;
```

Box-plot

```
DATA MYTABLE; SET MYTABLE; BOXPLOT= "_";  RUN;
PROC BOXPLOT DATA = MYTABLE ; PLOT (MYVAR)*BOXPLOT ; RUN;
DATA MYTABLE; SET MYTABLE; DROP BOXPLOT;  RUN;
```

Ici, on fait PROC sur la variable nommée MYVAR (représentation astucieuse !).

Pour réaliser un Box-Plot horizontal, on ajoute l'option HORIZONTAL à l'instruction PLOT

```
DATA MYTABLE; SET MYTABLE; BOXPLOT= "_";  RUN;
PROC BOXPLOT DATA = MYTABLE ; PLOT (MYVAR)*BOXPLOT / HORIZONTAL ; RUN;
DATA MYTABLE; SET MYTABLE; DROP BOXPLOT;  RUN;
```

IV.1.1.2. Analyses descriptives univariées sur des variables qualitatives (nominales et ordinales)

IV.1.1.2.1. Tableaux de fréquences univariées (Tri à plat)

D'une manière générale, pour réaliser la description univariée d'une variable qualitative (nominale ou ordinale), on utilise les fréquences absolues et les fréquences relatives. Pour réaliser le tableau de fréquence sous SAS, on utilise PROC FREQ (ou PROC TABULATE). Voir exemple ci-dessous en utilisant MYVAR.

```
PROC FREQ DATA=MYTABLE;
TABLES MYVAR      ;
RUN;
```

On pouvait aussi utiliser PROC TABULATE qui est beaucoup plus flexible. Mais on reviendra sur l'utilisation de cette procédure dans les analyses bivariées.

IV.1.1.2.2. Représentations graphiques : Barres de fréquences et diagrammes circulaires

On peut représenter graphiquement la distribution des variables qualitatives en utilisant soit les barres de fréquences, soit les diagrammes circulaires.

Diagrammes de fréquences

Barres verticales

La commande suivante permet de réaliser les barres verticales sur les modalités de la variable MYVAR :

```
PROC GCHART DATA=MYTABLE;
VBAR MYVAR/ DISCRETE

        TYPE=FREQ
       INSIDE=PERCENT  WIDTH=20
       SUMVAR =MYVAR;
RUN;
QUIT;
```

On peut aussi utiliser une seconde méthode qui consiste d'abord à faire PROC FREQ pour récupérer les statistiques. Ensuite utiliser ces statistiques pour réaliser les barres (voir exemple).

```
PROC FREQ DATA=MYTABLE; TABLES MYVAR / OUT=FREQ_DATA ;RUN;
PROC GCHART DATA=FREQ_DATA;
VBAR MYVAR /   DISCRETE FREQ=PERCENT /* ou FREQ=COUNT*/  ;RUN;  QUIT;
```

Barres horizontales

```
PROC GCHART DATA=MYTABLE;
HBAR MYVAR/ DISCRETE
        TYPE=FREQ
        INSIDE=PERCENT  WIDTH=20
        SUMVAR =MYVAR;
RUN;
QUIT;
```

Diagramme circulaire

```
PROC GCHART DATA=MYTABLE;
PIE MYVAR  /TYPE=FREQ
        /*TYPE=SUM SUMVAR=VALUE*/
         SLICE=OUTSIDE VALUE=INSIDE
         NOHEADER
        /* EXPLODE= MODALITE1*/
 ;
RUN;
```

IV.1.2. Analyses descriptives bivariées et multivariées

IV.1.2.1. Liaison entre deux variables quantitatives (coefficient de corrélation linéaire)

IV.1.2.1.1. Coefficients de corrélation et matrice de corrélations

Pour examiner la liaison entre deux variables quantitatives, on utilise généralement le coefficient de corrélation (de Pearson ou de Spearman). Pour obtenir la corrélation entre deux variables sous SAS, on utilise PROC CORR. Voir exemple ci-dessous :

```
/* Coefficients de PEARSON*/
PROC CORR DATA=MYTABLE PEARSON NOSIMPLE  ;
VAR MYVAR1 MYVAR2 ;
```

```
 RUN;
/* Coefficients de SPEARMAN*/
PROC CORR DATA=MYTABLE PEARSON NOSIMPLE  ;
VAR MYVAR1 MYVAR2 ;
 RUN;
```

IV.1.2.1.2. Examen graphique de la corrélation : le nuage de points

On peut examiner graphiquement la corrélation entre deux variables quantitatives en se basant sur le graphique du nuage de points. La première manière pour obtenir le graphique du nuage de points entre deux variables est d'ajouter l'option PLOTS=SCATTER à la procédure PROC CORR. Lorsqu'il s'agit de la corrélation entre plusieurs variables simultanément, on utilise PLOTS=MATRIX. L'exemple ci-dessous illustre cette première manière :

```
PROC CORR DATA=MYTABLE PEARSON NOSIMPLE PLOTS=SCATTER  ;
VAR MYVAR1 MYVAR2 ;
 RUN;
```

La seconde manière pour obtenir le nuage de points entre deux variables est d'utiliser la procédure PROC GPLOT (voir exemple ci-dessous) :

```
/* Préparation de la mise en forme du graphique*/
symbol1 value=circle interpol=none height=3 color=blue;
symbol1 font='albany amt/unicode' value='2640'x interpol=none height=4.5
color=blue;
axis1 label= ("Yvar label") order=(50 to 75 by 5) minor=none offset=(0,0);
axis2 label= ("Xvar label") order=(40 to 160 by 20) minor=none offset=(0,0);
symbol2 font='albany amt/unicode' value='2642'x height=4.5 interpol=none
color=red;
legend1 position=(top left inside) shape=symbol(.,4)
repeat=1 mode=protect cborder=graydd;
/* Représentation*/
PROC GPLOT DATA=MYTABLE;
PLOT MYVAR1*MYVAR2 /
/* ou plot MYVAR1*MYVAR2=MYVAR3 /  en cas de croisement avec une variable
catégorielle*/
LEGEND=LEGEND1
VAXIS=AXIS1 HAXIS=AXIS2 NOFRAME
AUTOVREF CVREF=GRAYDD
AUTOHREF CHREF=GRAYDD;
RUN;
```

Lorsqu'il s'agit d'analyser simultanément plusieurs variables, on utilise PROC SGSCATTER comme suit :

```
PROC SGSCATTER DATA=MYTABLE;
MATRIX MYVAR1  MYVAR3… MYVARN;
RUN;
```

IV.1.2.1.3. Représentation graphique de la tendance d'une variable

Lorsqu'on souhaite analyser la tendance d'une variable afin d'apprécier par exemple son évolution au cours du temps, on peut également se servir la procédure PROC GPLOT en modifiant simplement l'apparence des points. Par exemple, supposons qu'on veuille analyser l'évolution d'une variable MYVAR en fonction du temps matérialisé par une variable nommée TIMEVAR. On peut adopter la formulation suivante :

```
/* Préparation de la mise en forme du graphique*/
symbol1 value=none interpol=joint height=3 color=blue;
symbol2 font='albany amt/unicode' value='2642'x height=4.5 interpol=none
color=red;
axis1 label= ("Yvar label") order=(50 to 75 by 5) minor=none offset=(0,0);
axis2 label= ("Xvar label") order=(40 to 160 by 20) minor=none offset=(0,0);
legend1 position=(top left inside) shape=symbol(.,4)
repeat=1 mode=protect cborder=graydd;
/* Représentation*/
PROC GPLOT DATA=MYTABLE;
PLOT MYVAR*TIMEVAR /
LEGEND=LEGEND1
VAXIS=AXIS1 HAXIS=AXIS2
AUTOVREF CVREF=GRAYDD
AUTOHREF CHREF=GRAYDD;
RUN;
```

D'abord, on remarque la structure de PROC GPLOT reste la même chose que celle qui a été utilisée dans la représentation d'un nuage de points. La différence réside simplement dans la mise en forme du graphique notamment l'option symbol1. En effet, pour le nuage de points, on utilise par exemple l'option : symbol1 value=circle interpol=none height=3 color=blue; alors que la courbe d'évolution, on utilise symbol1 value=none interpol=joint height=3 color=blue;
Dans cette dernière formulation, on fait disparaître les symboles en indiquant value=none tandis qu'on joint les positions des points en utilisant interpol=joint. Ce qui n'est pas fait dans le nuage de point.

IV.1.2.2. Liaison entre deux variables qualitatives

IV.1.2.2.1. Tableau de contingence ou tri croisé

Pour analyser la distribution de l'échantillon sur deux caractéristiques qualitatives données, on réalise un tableau de contingence. Le tableau de contingence est un tableau qui permet d'obtenir les fréquences de chaque modalité sur une variable par un croisement avec les modalités d'une autre variable, on dit alors qu'on effectue un tri croisé.

Pour réaliser un tri croisé sous SAS, on peut se servir en premier lieu de la procédure PROC FREQ. L'exemple ci-dessous est une illustration :

```
PROC FREQ DATA=MYTABLE;
 TABLES MYVAR1*MYVAR2;
RUN;
```

Noter aussi qu'on peut aussi utiliser la procédure PROC TABULATE qui est beaucoup plus flexible et qui permet de réaliser des croisements de plusieurs niveaux notamment autorisant l'analyse des variables quantitatives. Nous introduirons cette procédure dans la section suivante.

IV.1.2.2.2. Mesure du degré d'association entre deux variables qualitatives

Il existe plusieurs indicateurs pour mesurer le lien entre deux variables qualitatives nominales. On dénombre notamment la distance de Khi-deux, le coefficient de contingence de Pearson, le coefficient V de Cramer ainsi que le coefficient Phi (qui est l'équivalent du coefficient de corrélation linéaire pour variables qualitatives). On peut obtenir l'ensemble de ces statistiques en ajoutant CHISQ à l'instruction TABLE de PROC FREQ entre deux variables qualitatives. Voir-ci-dessous un exemple :

```
PROC FREQ DATA=MYTABLE;
 TABLES MYVAR1*MYVAR2 / CHISQ ;
RUN;
```

Remarque

Notons aussi que l'option CHISQ permet aussi de réaliser le test Exact de Fisher qui est le test à interpréter en cas où il y a des cellules du tableau de contingence dont la proportion est inférieure à 5 pour cent.

Par ailleurs lorsque les deux variables sont des variables qualitatives ordinales, il est possible de calculer leur degré d'association en utilisant le coefficient de corrélation

de rang de Spearman ou le coefficient de KENDALL. Il suffit pour cela d'utiliser PROC CORR comme cela a été présenté un peu plus haut (cas de variables quantitatives).

IV.1.3. Croisement des variables de plusieurs niveaux : utilisation de PROC TABULATE

PROC TABULATE est l'une des procédures de tabulation les plus puissantes de SAS. En effet, elle permet de réaliser des croisements de variables plusieurs niveaux y compris permettant de prendre en compte la distribution des variables quantitatives. Elle permet également de produire les statistiques ordinaires telles que le feraient PROC SUMMARY ou PROC MEANS. De par cette flexibilité, PROC TABULATE a de nombreux avantages par rapport à PROC FREQ. En effet, même si PROC FREQ permet de réaliser des croisements de plusieurs niveaux. Elle se limite à fournir les fréquences absolues et relatives. PROC TABULATE permet par exemple d'analyser la distribution d'une ou de plusieurs variables quantitatives selon les valeurs croisées de deux ou plusieurs variables qualitatives. Dans cette section, nous allons présenter quelques exemples d'utilisation de cette procédure. Ces exemples seront présentés de manière séquentielle de manière à pouvoir détailler toute la flexibilité offerte pas par PROC TABULATE.

IV.1.3.1. Analyse d'une variable quantitative selon les modalités d'une qualitative

Le premier avantage de PROC TABULATE par rapport à PROC FREQ réside dans sa capacité à croiser la statistique obtenue à partir d'une variable quantitative (moyenne, médiane) en fonction des modalités d'une ou de plusieurs variables qualitatives. Pour être concrète, commençons la démonstration par une analyse quantitative dans laquelle, on veut étudier le montant moyen de loyers par type de logement dans les grandes villes aux Etats-Unis. La première variable d'intérêt est nommée RENT qui capte le montant du loyer dans chaque cas analysé (l'exemple utilisé ici est tiré de l'article de Lauren Haworth intitulé *Anyone Can Learn PROC TABULATE* publié dans SUGI 27)

Tableau à simple entrée

D'abord, considérons la variable RENT et faisons la PROC TABULATE suivante :

```
PROC TABULATE DATA=TEMP;
VAR RENT;
TABLE RENT;
RUN;
```

Par défaut SAS fournit le total de la variable RENT. On peut modifier ce comportement en spécifiant les statistiques souhaitée (ex : Moyenne, Médiane, MIN, Max, Nombre d'observations, etc…). Exemple :

```
PROC TABULATE DATA=TEMP;
VAR RENT;
TABLE RENT*MEAN;
RUN;
```

Il faut signaler qu'on peut spécifier plusieurs instructions TABLE dans une même PROC TABULATE ou même invoquer dans une même instruction TABLE plusieurs statistiques sur la même variable ou sur d'autres variables. Les deux exemples ci-dessous sont des illustrations :

```
PROC TABULATE DATA=TEMP;
VAR RENT;
TABLE RENT*MEAN;
TABLE RENT*N;
RUN;
```

Ou encore

```
PROC TABULATE DATA=TEMP;
VAR RENT;
TABLE RENT*MEAN  RENT*N;
RUN;
```

La différence entre les deux formulations est que dans le premier cas, SAS créer deux tables et dans le second cas les statistiques sont présentées dans une même table (ce qui est logique !)

Il faut signaler que la seconde formulation peut être améliorée encore un peu plus car la variable RENT est spécifiée à chaque fois qu'une statistique est invoquée. Pour rendre efficient cette écriture, on peut utiliser une parenthèse. Dans ce cas on aura la présentation suivante :

```
PROC TABULATE DATA=TEMP;
VAR RENT;
TABLE RENT*(  N  MEAN);
RUN;
```

Dans cette formulation, on obtient exactement le même résultat que la précédente. Le tableau de résultat se présente comme suit :

Rent	
N	Mean
115.00	1416.50

La moyenne présentée dans ce tableau est une moyenne générale. Elle peut cacher des disparités au sein de l'échantillon. Par exemple, il est probable que le montant du loyer soit plus élevé dans les grands agglomérations que dans les petites. C'est pourquoi, il peut être intéressant de présenter les montants moyens des loyers par ville. Pour réaliser cela, on ajoute ce qu'on appelle variable de classification ou variable de croisement. Le croisement entre deux variables se fait en utilisant le signe *. Remarquons aussi que le signe * est aussi utilisé pour afficher une statistique sur une variable. Dès lors pour afficher la moyenne de la variable RENT par ville, il faut croiser d'abord croiser RENT avec le mot MEAN, ensuite croiser cet ensemble avec la variable ville. C'est cet exemple qui est illustré ci-dessous :

```
PROC TABULATE DATA=TEMP;
CLASS CITY;
VAR RENT;
TABLE RENT*MEAN*CITY;
RUN;
```

Le résultat obtenu par cette formulation ci-dessous se présente comme suit :

Rent		
Mean		
City		
Orlando	San Francisco	Seattle
837.13	2440.89	1099.55

La moyenne est beaucoup plus élevée à San Francisco comparativement à Orlando ou à Seattle. Ce qui justifie la pertinence de la catégorisation de la moyenne.

Remarquons que la prise en compte de la variable de classification se fait en ajoutant l'instruction CLASS. Cette instruction est très importante car elle définit le niveau de croisement entre les variables. Un croisement de deux niveaux correspond à la spécification de deux variables de classification.

On peut ajouter autant de variables de classification que nous voulons à l'instruction CLASS et réaliser les croisements en conséquence. Cependant quel que soit le nombre de variable ajouté à l'instruction CLASS, le tableau de résultats obtenu reste toujours un tableau à une seule dimension (en l'occurrence un tableau à plusieurs colonnes et une ligne).

Par ailleurs, il est possible d'afficher la moyenne générale en ajoutant le mot clé ALL comme suit :

```
PROC TABULATE DATA=TEMP;
CLASS CITY;
VAR RENT;
TABLE RENT*MEAN*(CITY ALL);
RUN;
```

Cela signifie qu'on veut les moyennes pour chaque ville mais aussi pour l'ensemble. Ce mot peut être utilisé pour toutes les statistiques à afficher.

Tableau à double-entrée

Dans de nombreux cas, on est face à des situations nécessitant l'élaboration de tableaux à double entrée ou des tableaux à plusieurs dimensions.

Pour réaliser un tableau à double-entrée, on sépare les arguments TABLE par une virgule. Par exemple, pour réaliser un tableau à double-entrée entre CITY et la moyenne de RENT, on fait :

```
PROC TABULATE DATA=TEMP;
CLASS CITY;
VAR RENT;
TABLE CITY, RENT*(N MEAN);
RUN;
```

Maintenant, nous souhaitons généraliser ce cas en analysant la moyenne du loyer par ville selon le nombre de pièces de l'appartement. Ainsi, on peut adopter la formulation suivante :

```
PROC TABULATE DATA=TEMP;
VAR RENT;
CLASS BEDROOMS CITY;
TABLE BEDROOMS, RENT*CITY*MEAN;
RUN;
```

Cette tabulation donne le résultat suivant :

	Rent		
	City		
	Orlando	San Francisco	Seattle
	Mean	Mean	Mean
Bedrooms			
1 Bedroom	726.85	2143.53	902.00
2 Bedrooms	947.40	2721.72	1297.10

Dans la formulation ci-dessus, la variable de ligne est BEDROOMS qui prend deux modalités (1 Bedroom et 2 Bedrooms). La variable de colonne est la moyenne de RENT croisée avec la variable CITY.

Il faut signaler qu'on peut effectuer un autre croisement de la moyenne de RENT sur une autre variable de ligne sans avoir besoin d'invoquer une autre PROC TABULATE. Pour cela, il suffit d'ajouter cette variable à la suite de BEDROOMS. Par exemple on souhaite faire une analyse du prix moyen par ville selon que l'appartenant dispose d'une connexion internet ou pas. Voir exemple ci-dessous :

```
PROC TABULATE DATA=TEMP;
VAR RENT;
CLASS BEDROOMS CITY INTERNET;
TABLE BEDROOMS INTERNET, RENT*CITY*MEAN;
RUN;
```

Comme il n'y a pas de signe * entre BEDROOMS et INTERNET, alors il ne s'agit pas d'un croisement entre les deux variables. Il s'agit plutôt d'un croisement des deux variables une à une avec le croisement de CITY avec la moyenne de RENT. Le résultat issu de ce croisement se présente comme suit :

	Rent		
	City		
	Orlando	San Francisco	Seattle
	Mean	Mean	Mean
Bedrooms			
1 Bedroom	726.85	2143.53	902.00
2 Bedrooms	947.40	2721.72	1297.10
Internet			
No	769.40	2292.89	855.55
Yes	904.85	2616.63	1343.55

Dans ce tableau les résultats des croisements avec BEDROOMS et INTERNET sont présentés sur des lignes séparées. Pour effectuer des croisements en imbriquant les deux variables, il faut légèrement modifier le code en ajoutant une étoile entre les eux variable telle que :

```
PROC TABULATE DATA=TEMP;
VAR RENT;
CLASS BEDROOMS CITY INTERNET;
TABLE BEDROOMS*INTERNET, RENT*CITY*MEAN;
RUN;
```

Le résultat obtenu se présente alors comme suit :

		Rent		
		City		
		Orlando	San Francisco	Seattle
		Mean	Mean	Mean
Bedrooms	Internet			
1 Bedroom	No	613.80	2012.78	743.30
	Yes	839.90	2290.63	1060.70
2 Bedrooms	No	925.00	2545.00	967.80
	Yes	969.80	2942.63	1626.40

Cela permettra par exemple d'afficher toutes les statistiques souhaitables sans avoir à écrire plusieurs PROC TABULATE. Mais abandonnons d'abord cette idée et concentrons-nous uniquement sur la moyenne de la variable RENT.

Là aussi, on peut définir autant de niveau de croisement que l'on souhaite tant que ces variables sont déclarées dans l'instruction CLASS comme des variables catégorielles.

Aussi, il ne faut pas oublier que pour toutes les statistiques affichées ci-dessus, on peut ajouter le mot ALL pour calculer les totaux soit en ligne soit en colonnes.

Par ailleurs, il faut signaler que le croisement peut être effectué en utilisant n'importe quelle autre statistique à la place de la moyenne.

IV.1.3.2. Croisements des variables qualitatives

Dans cette section, nous allons réaliser les tris à plat et les tris croisés en utilisant PROC TABULATE. L'objectif est de montrer que les résultats obtenus sont équivalents à ceux obtenus dans une PROC FREQ classique.

Considérons toujours l'exemple sur le montant des loyers aux Etats-Unis précédemment présenté.

Tableau de fréquences absolues d'une variable

Etablissons le tableau de fréquence sur CITY. Alors on fait :

```
PROC TABULATE DATA=TEMP;
CLASS CITY;
TABLE  CITY*N;
RUN;
```

Ainsi, pour obtenir un tableau de fréquence simple, on utilise simplement CLASS et TABLE.

Tableau croisés entre deux variables

Pour faire le tableau croisé entre deux variables (ex : CITY et BEDROOMS), on fait :

```
PROC TABULATE DATA=TEMP;
CLASS CITY BEDROOMS;
TABLE  BEDROOMS, CITY*N;
RUN;
```

On peut aussi afficher le total en ajoutant ALL par catégorie de BEDROOMS comme suit :

```
PROC TABULATE DATA=TEMP;
CLASS CITY BEDROOMS;
TABLE  BEDROOMS, (CITY ALL)*N;
RUN;
```

Ou encore le total par ville comme suit :

```
PROC TABULATE DATA=TEMP;
CLASS CITY BEDROOMS;
TABLE  BEDROOMS ALL, CITY*N;
RUN;
```

IV.1.3.3. Les croisements à trois dimensions

Tous les croisements précédemment effectués sont soit des croisements à une dimension (tableau à simple entrée), ou des croisements à double dimension (tableau à double entrée). A présent, il s'agit de réaliser un croisement trois dimensions. Pour cela, nous allons utiliser INTERNET, BERDROOMS et CITY en fonction desquelles nous

analysons la moyenne de RENT. La formulation de ce tableau à Triple-dimension est la suivante :

```
PROC TABULATE DATA=TEMP;
CLASS INTERNET BEDROOMS CITY ;
VAR RENT;
TABLE INTERNET, BEDROOMS, (CITY ALL)*RENT*MEAN;
RUN;
```

Dans cette formulation, les trois dimensions sont séparées par des virgules. Il faut noter que la troisième dimension ici est INTERNET car la troisième dimension correspond toujours à la première variable déclarée dans l'instruction TABLE.

Aussi, en termes de résultats, il faut signaler qu'en pratique, un tableau à trois dimensions est un tableau à deux dimensions répété pour chaque valeur de la troisième dimension. De ce point de vue, il s'agit simplement d'une généralisation d'un tableau à double entrée.

IV.1.3.4. Mise en forme des résultats du PROC TABULATE

Par défaut PROC TABULATE affiche les noms des variables et des statistiques dans l'intitulé des tableaux. Celui-ci peut alors très surchargé ou illisible lorsque le niveau de croisement devient plus élever. C'est pourquoi alors utiliser les options de mise en forme afin d'améliorer l'apparence des résultats.

La première option de mise en forme des résultats concerne l'intitulé des variables ou des statistiques. D'une manière générale, pour modifier l'intitulé d'une variable ou d'une statistique, on ajoute un signe ='texte ou rien' au nom de la variable ou de la statistique. Voir exemple ci-dessous :

```
PROC TABULATE DATA=TEMP;
CLASS CITY BEDROOMS;
TABLE  BEDROOMS=' ', CITY ='Ville'*N=' ';
RUN;
```

Dans cet exemple, nous indiquons que l'intitulé de la variable BEDROOMS et de la statistique N doit être vide en utilisant =' '. Par contre nous voulons que l'intitulé pour la variable CITY soit égal à Ville. De ce point de vue la définition des libellés pour les statistiques et les variables es très simple.

De même, on peut définir le format d'affichage de chaque statistique présentée dans la table. Pour cela, on utilise simplement l'option format en le croisant avec la statistique à produire. Voir exemple ci-dessous :

```
PROC TABULATE DATA=TEMP;
CLASS INTERNET BEDROOMS CITY ;
VAR RENT;
TABLE INTERNET, BEDROOMS, (CITY ALL)*RENT*MEAN*Format= best12.;
RUN;
```

Ici, on a choisi le format best12 qui est l'un des formats prédéfinis de SAS. On pouvait aussi définir notre propre format en utilisant PROC FORMAT avec l'instruction VALUE ou l'instruction PICTURE. Ensuite, venir associer ce format à la statistique désirée.

IV.1.4 Graphiques croisés : utilisation de PROC GCHART

La plupart des graphiques dans les analyses descriptives multivariés sont des graphiques croisés. Par exemple, on peut vouloir représenter la graphiquement la moyenne d'une variable quantitatives selon les modalités d'une ou de plusieurs variables qualitatives. Il peut aussi s'agir des totaux ou même des fréquences. Pour réaliser ces types de représentation sous SAS, on utilise PROC GCHART qui est, en quelque sorte, l'équivalent de PROC TABULATE pour les graphiques. Dans cette section, nous allons étudier plusieurs variantes de l'utilisation de PROC GCHART.

Pour bien comprendre la construction des graphiques croisés avec GCHART, commençons d'abord par un graphique de fréquence univarié en utilisant la variable QUALVAR1. On a :

```
PROC GCHART DATA=MYDATA;
VBAR QUALVAR1/ DISCRETE
        TYPE=FREQ
       SUMVAR =QUALVAR1
        INSIDE=PERCENT  WIDTH=20
        ;
RUN;
QUIT;
```

Dans cette formulation, nous indiquons à SAS que la variable QUALVAR est une variable qualitative avec l'option DISCRETE. Ensuite, nous lui indiquons que la statistique à produire est une fréquence. TYPE=FREQ. Avec SUMVAR (SUMMARY VARIABLE) nous indiquons que ces fréquences doivent être calculées en se basant sur

les valeurs de la variable QUALVAR1. Et pour le reste, l'instruction INSIDE permet de dire à SAS d'afficher les valeurs des fréquences à l'intérieur des barres dont les tailles seront fixées à 20.

Maintenant, nous voulons faire un graphique qui présente la moyenne d'une variable QUANTVAR1 selon les modalités de la variable QUALVAR1. Alors, on fait :

```
PROC GCHART DATA=MYDATA;
VBAR QUALVAR1/ DISCRETE
        TYPE=MEAN
       SUMVAR =QUANTVAR1
        INSIDE= MEAN  WIDTH=20
        ;
RUN;
QUIT;
```

On peut par exemple utiliser ce type de graphique lorsqu'il s'agit par exemple le montant moyen des loyers par région. Les modalités de la région seront représentées par QUALVAR1 et les montants des loyers seront représentés par QUANTVAR1.

Supposons maintenant qu'on nous demande de présenter les montants moyens par région en comparant à l'intérieur de chaque région le montant par zone. Pour cela, nous adopterons la formulation suivante :

```
PROC GCHART DATA=MYDATA;
VBAR QUALVAR1/ DISCRETE
        SUBGROUP=QUALVAR2
       TYPE=MEAN
       SUMVAR =QUANTVAR1
        ;
RUN;
QUIT;
```

Il s'agit ici d'un croisement de premier niveau car la zone (rurale ou urbaine) est incluse dans région.

Admettons qu'on se rende compte que ce graphique ne correspond pas à notre. On souhaiterait plutôt représenter la moyenne par zone (rurale VS urbaine) dans chaque région c'est-à-dire une barre distincte pour chaque zone à l'intérieur de chaque région. Alors on adoptera la formulation suivante :

```
PROC GCHART DATA=base0;
VBAR QUALVAR2 / DISCRETE
        TYPE=MEAN
        SUMVAR =QUANTVAR
                GROUP=QUALVAR1
                INSIDE=MEAN  WIDTH=20
         ;
RUN;
QUIT;
```

Il s'agit d'un croisement de second niveau où l'on peut distinguer la moyenne pour chaque zone à l'intérieur de chaque région. C'est le cas le plus couramment rencontré dans la pratique.

Remarque

Les exemples présentés ci-dessus sont basés uniquement sur VBAR (vertical BAR), on peut aussi utiliser VBAR3D, HBAR, HBAR3D et PIE.

IV.1.5. Test d'égalité de la moyenne à une valeur de référence

Le test d'égalité de la moyenne à une valeur de référence est un test d'hypothèse dans lequel on cherche à vérifier si une moyenne calculée sur l'échantillon est-elle conforme ou pas à une valeur de référence. Par exemple, le service contrôle qualité d'une entreprise de tuyauterie souhaite étudier la conformité d'un échantillon d'un modèle de tuyau produit dans une série donnée. Ce modèle de tuyau doit être d'un diamètre de 15 cm. En se basant sur l'échantillon qu'il dispose, le contrôleur qualité souhaite donc savoir si les tuyaux produits sont conformes à cette exigence. Dans ce cas, il utilise un test d'égalité de la moyenne à une valeur de référence. Ce test est encore appelé test de conformité.

Théoriquement, la mise en œuvre de ce test est basée sur la statistique de Student car la variance de la population n'est pas connue et qu'elle doit être estimée aussi à partir de l'échantillon.

Pour réaliser le test sous SAS, on utilise la procédure PROC TTEST. L'exemple ci-dessous est une illustration :

```
PROC TTEST DATA = MYTABLE H0=15;
        VAR MYVAR;
RUN;
```

L'argument H0 représente l'hypothèse nulle à tester. Elle indique la valeur de référence à laquelle on doit tester la conformité de la moyenne calculée sur l'échantillon. Cette moyenne empirique est calculée sur la variable MYVAR.

IV.1.6. Comparaison de moyennes sur deux échantillons : test de Student

On distingue deux principaux cas de comparaison de moyennes sur deux échantillons :le test de comparaison de moyenne sur échantillons indépendant et le test de comparaison sur échantillons appariés.

IV.1.6.1. Comparaison de moyennes sur deux échantillons indépendants

Deux échantillons sont indépendants lorsque les unités d'analyses qui les constituent sont distinctes d'un échantillon à un autre. Par exemple, un physiologiste veut évaluer l'efficacité d'un produit A sur l'augmentation de la force musculaire. La force musculaire est mesurée par la force de préhension c'est-à-dire la pression exercée sur un certain objet. Ce physiologiste réalise une expérience d'un mois sur deux groupes d'individus. Un premier groupe auquel on donne le produit pendant un mois et un second qui n'a reçu aucune dose sur la période. De tels échantillons sont considérés comme des indépendants.

Pour évaluer l'efficacité du produit A, on va comparer la moyenne de la force de préhension entre les deux groupes et tester si cette différence est significative.

Pour mettre en œuvre ce type de test, nous utilisons le test de Student encore appelé Test-T. Ce test est mis en œuvre sous SAS en utilisant la commande suivante :

```
PROC TTEST DATA=MYTABLE PLOT=NONE;
        CLASS GROUPVAR;
        VAR MYVAR;
  RUN;
```

Dans cette formulation, GROUPVAR est la variable qui représente les deux groupes et MYVAR est la valeur dont la moyenne doit être comparée.

IV.1.6.2. Comparaison de moyennes sur deux échantillons appariés

Deux échantillons sont appariés lorsque les unités d'analyse sont les mêmes d'un échantillon à un autre. Par exemple le physiologiste veut savoir si la force musculaire

est accrue par l'absorption d'un produit A. Il réalise l'expérience sur un échantillon de 10 personnes choisies au hasard et mesure leurs forces de préhension **avant** de les donner les produit. Ensuite **après** de consommation du produit A, le physiologiste repasse pour relever les forces de préhensions des individus. De tels échantillons sont appelés « échantillons appariés ». Car il s'agit de mesurer la même information sur les mêmes individus mais simplement à des périodes différentes.

Pour évaluer l'efficacité du produit, il faut alors comparer la moyenne de la force de préhension avant et après l'absorption du produit. Nous allons alors réaliser un test de Student sur échantillons appariés. La commande ci-dessous illustre sa mise en œuvre sous SAS.

```
PROC TTEST DATA = MYTABLE;
        PAIRED MYVAR1 * MYVAR2;
RUN;
```

Ici MYVAR1 est la première mesurée réalisée sur la variable de résultat et MYVAR2 est la seconde mesure réalisée sur la variable de résultat. Théoriquement, il s'agit alors de voir si la moyenne de MYVAR1 est égale à la moyenne de MYVAR2.

IV.1.7. Les analyses de variance : Anova, Manova, Ancova et Mancova

Les analyses de variances (de covariances) sont généralement des analyses qui mettent en relation une variable quantitative avec des variables qualitatives. Dans ce contexte, les variables qualitatives sont souvent qualifiée de facteurs (ou variables d'expérimentation) ; les variables quantitatives sont, elles, des variables de résultat. Par exemple, on souhaite évaluer l'effet de la saison sur la performance des athlètes en lancer de poids. Pour cela, on mesure les performances de 100 athlètes en lancer de poids (m) mesurées à deux moments de la saison (l'hiver et l'été). La variable de résultat ici est la distance atteinte par le poids lancé par l'athlète (variable quantitative) et le facteur est la saison (variable qualitative à deux modalités hiver et été). La saison représente ici le facteur. Ce facteur peut être de nature diverse et est fixé par l'expérimentateur. Par exemple les patients qui ont suivi un certain traitement, les clients qui ont été ciblés par une campagne marketing donnée, etc.. Dans chacun des cas, il s'agit d'analyser la différence en termes de résultats entre les individus qui ont reçu un traitement particulier comparativement et ceux qui n'ont pas reçu. Méthodologiquement, il s'agit de mettre en œuvre une Analyse de Variance (ANOVA) ou une analyse de covariance ANCOVA. Dans le cas où l'on dispose de

plusieurs variables de résultats à analyser simultanément, il s'agira d'un MANOVA ou d'une MANCOVA. Nous dirons quelques mots sur chacune des analyses et dans quel cas faut-il les utiliser.

IV.1.7.1. ANOVA

La méthode ANOVA est une méthode dans laquelle on compare la moyenne d'une variable quantitative selon les modalités d'une variable qualitatives. C'est un test de comparaison de moyenne qui permet de généraliser le test de Student lorsque le nombre de groupes est supérieur à deux. Contrairement au test de Student, l'ANOVA est fondée sur l'hypothèse que la variable dépendante quantitative est distribuée selon une loi normale. Par conséquent, elle n'est plus valide lorsque l'hypothèse de normalité n'est pas vérifiée. Il faut alors penser à des tests non paramétriques (Kruskal–Wallis, Wilcoxon, Mann-Whitney). L'exemple ci-dessous illustre la mise en place de l'ANOVA sous SAS.

```
PROC ANOVA DATA = MYTABLE;
        CLASS GROUPVAR;
        MODEL DEPVAR = GROUPVAR;
        MEANS GROUPVAR / BON  /* ou encore hovtest welch */;
RUN;
QUIT;
```

GROUPVAR est la variable qui représente les groupes d'analyse tandis que DEPVAR est la variable de résultat qu'on souhaite comparer. L'estimation est réalisée en utilisant l'instruction MODEL. On peut aussi ajouter des options supplémentaires par exemple MEANS qui propose une estimation des moyennes selon différentes méthodes d'ajustement.

L'exemple présenté ci-dessus est appelé ANOVA à un seul facteur.

Une ANOVA à deux facteurs est une ANOVA où il y a deux variables expérimentales ou deux facteurs. Par exemple, pour estimer une ANOVA à deux facteurs on fait :

```
PROC ANOVA DATA = MYTABLE;
        CLASS GROUPVAR1 GROUPVAR2;
        MODEL DEPVAR = GROUPVAR1 GROUPVAR2;
        RUN;
QUIT;
```

Il ne faut pas oublier qu'à partir du moment où le modèle contient deux variables explicatives, on peut examiner les effets d'interaction entre ces variables. Par exemple, on peut faire:

```
PROC ANOVA DATA = MYTABLE;
        CLASS GROUPVAR1 GROUPVAR2;
        MODEL DEPVAR = GROUPVAR1 GROUPVAR2 GROUPVAR1*GROUPVAR2;
        RUN;
QUIT;
```

Lorsque cette interaction est significative, elle doit être gardée dans le modèle pour la suite de l'analyse.

IV.1.7.2. MANOVA

Lorsqu'on veut comparer simultanément plusieurs variables de résultats (quantitatives) selon les modalités d'une variable expérimentale (qualitative), on utilise MANOVA. Celle-ci permet de prendre en compte les différentes corrélations qui peuvent exister entre les différentes variables de résultats. Pour mettre en œuvre cette démarche sous SAS, on peut utiliser PROC GLM telle que décrite ci-dessous :

```
PROC GLM DATA= MYTABLE;
  CLASS GROUPVAR;
  MODEL DEPVAR1 DEPVAR DEPVAR3= GROUPVAR / SS3;
  MANOVA H=_ALL_;
  RUN;
```

Tout comme pour l'ANOVA à deux facteurs, on peut aussi avoir une MANOVA à deux facteurs. Ces facteurs doivent alors déclarés dans l'instruction CLASS avant de les inclure dans l'instruction MODEL. On peut aussi penser à créer des interactions entre les facteurs. Exemple :

```
PROC GLM DATA= MYTABLE;
  CLASS GROUPVAR1 GROUPVAR2 ;
  MODEL DEPVAR1 DEPVAR DEPVAR3= GROUPVAR1 GROUPVAR2
GROUPVAR1*GROUPVAR2 / SS3;
  MANOVA H=_ALL_;
  RUN;
```

IV.1.7.3 ANCOVA

Une ANCOVA est une ANOVA en prenant en compte l'effet d'autres variables quantitatives. Il s'agit alors de comparer la moyenne de la variable de résultat selon

les modalité de la variable d'expérience après avoir contrôlé l'effet d'autres caractéristiques de nature quantitative. L'exemple ci-dessous illustre l'estimation d'une ANCOVA sous SAS :

```
PROC GLM DATA=MYTABLE;
  CLASS GROUPVAR;
  MODEL DEPVAR = GROUPVAR INDEPVAR1 INDEPVAR2 … / SOLUTION;
RUN; QUIT;
```

Tout comme dans les précédents cas, on peut aussi avoir une ANCOVA à deux ou plusieurs facteurs. Il suffit alors de les indiquer dans l'instruction CLASS avant de les spécifier dans l'instruction MODEL. De plus, il faut penser à créer des effets d'interaction en ajoutant des termes multiplicatifs.

IV.1.7.4 MANCOVA

Une MANCOVA est une ANCOVA réalisée sur plusieurs variables de résultat pris simultanément afin de contrôler d'éventuelles corrélations. Son estimation sous SAS se présente comme suit :

```
PROC GLM DATA=MYTABLE;
  CLASS GROUPVAR;
  MODEL DEPVAR1 DEPVAR2 DEPVAR3 = GROUPVAR INDEPVAR1 INDEPVAR2 … /
SOLUTION;
RUN; QUIT;
```

On peut aussi avoir une MANCOVA à deux ou plusieurs facteurs. Dans ce cas il faut les indiquer dans l'instruction CLASS avant de les spécifier dans l'instruction MODEL. Penser aussi à créer des effets d'interaction en ajoutant des termes multiplicatifs pour observer leur significativité.

IV.2. Analyses multidimensionnelles et Datamining

Les analyses multidimensionnelles sont des techniques exploratoires visant à résumer l'information contenue dans un vaste ensemble de données se présentant de telle sorte qu'il est impossible d'en dégager des structures générales en utilisant les outils de la statistique unidimensionnelle.

Les analyses multidimensionnelles sont souvent désignées sous le vocable de statistique exploratoire, du Datamining ou encore simplement de l'Analyse de Données (ADD). Elles sont très généralement utilisées dans les études marketing et connaissance-client dans le but de dégager ou de valider de structures de données, de réaliser du scoring, etc..

Les analyses multidimensionnelles se distinguent selon qu'elles soient orientées vers les variables ou orientées vers les individus. Une analyse multidimensionnelle orientée vers les variables vise généralement à réduire la dimension de la table de données en regroupant les variables (selon leur degré de corrélation) dans un nombre limité de catégories. Ces variables sont généralement désignées comme des axes. En revanche, une analyse multidimensionnelle orientée vers les individus vise à regrouper les individus (selon leur degré de ressemblance sur des critères définis par les variables) dans un nombre limité de classes. On dit alors qu'il s'agit d'une classification.

D'une manière générale, le choix d'une méthode particulière d'analyse multidimensionnelle dépend selon qu'on veuille réduire la dimension des données en regroupant les variables sur des axes ou bien regrouper les individus dans des classes.

Parmi les méthodes d'analyses multidimensionnelles les plus couramment utilisée on distingue les analyses en composante principales, les analyses factorielles des correspondances simples, les analyses des correspondances multiples et les méthodes de classification. Les trois premières analyses visent à réduire la dimension des données tandis que la dernière vise à regrouper les individus dans des classes selon leur degré de ressemblance. Dans cette section, nous présentons la mise en œuvre de chacune de ces méthodes sous SAS.

IV.2.1. Analyses en composantes principales (ACP)

IV.2.2. Mise en œuvre d'un ACP et calcul des scores

On utilise l'analyse en composantes principales lorsque les variables sur lesquelles porte l'analyse sont toutes de nature quantitative. Pour mettre en œuvre cette méthode sous SAS, on utilise PROC PRINCOMP. L'exemple ci-dessous est une illustration.

```
PROC PRINCOMP DATA=MYTABLE OUT=MYTABLE N=2 PLOTS=ALL;
VAR MYVAR1 MYVAR2…MYVARN ;
RUN;
```

Dans ce code, On réalise l'ACP sur les variables allant de MYVAR1 à MYVARN à partir de la table MYTABLE. Ensuite, on indique à SAS retenir les deux facteurs et créer les scores des individus en retenant deux composantes (N=2) et ajouter ces scores à la table de données. La nouvelle table obtenue est enregistrée sous le nom de MYTABLE afin d'écraser la première table. Mais on pouvait aussi donner un autre nom à la table

créée si l'on ne souhaitait pas modifier la table initiale. L'option PLOT=ALL permet de donner les représentations graphiques des différentes statistiques disponibles à après une ACP. Les deux représentations les plus intuitives sont celle des variables sur les axes factorielles mais aussi la représentation des individus par rapport aux axes.

Il est important de signaler qu'avant de faire une ACP, il est souhaitable de standardiser d'abord les variables afin d'éliminer de potentiel effets d'échelle. La standardisation permet de mettre toutes les variables sur la même échelle avec une moyenne de 0 et un écart-type de 1. Pour standardiser les variables on peut utiliser PROC STANDARD comme suit :

```
PROC STANDARD DATA=MYTABLE MEAN=0 STD=1 OUT=MYTABLE;
VAR MYVAR1 MYVAR2…MYVARN;
RUN;
```

Une fois que les variables sont standardisées, on peut effectuer l'ACP en utilisant PROC PRINCOMP.

Par ailleurs, il est possible d'ajouter une option supplémentaire à PROC PRINCOMP pour exporter dans une nouvelle table les statistiques obtenues en cours d'analyse. Il s'agit de l'option OUTSTAT. L'exemple ci-dessous est une illustration.

```
PROC PRINCOMP DATA=MYTABLE OUT=MYTABLE N=2 PLOTS=ALL
OUTSTAT=TAB_STAT ;
VAR MYVAR1 MYVAR2…MYVARN ;
RUN;
```

L'utilisation de l'option OUTSTAT peut s'avérer utile dans de nombreuses situations. Par exemple dans les études de scoring, il arrive qu'on utilise les paramètres d'un modèle déjà validé pour calculer les scores sur de nouveaux individus qui n'appartenant à l'échantillon au moment de l'estimation du modèle. Dès lors, il est important de spécifier chaque fois l'option OUTSTAT afin d'enregistrer les paramètres du modèle pour des utilisations futures (nous reviendrions sur ces aspets).

Par ailleurs, il faut aussi noter qu'en spécifiant l'option PLOTS=ALL SAS fournit les différentes graphiques disponibles après une ACP. Mais lorsque ces graphiques ne vous apparaissent pas très élaborées, on peut construire ses propres graphiques. Par exemple supposons qu'on veuille refaire la représentation des variables et des observations sur les deux composantes retenues. Alors, on se sert des tables produites par l'option OUTSTAT et OUT.

Représentation des variables

Pour représenter les variables à partir des statistiques obtenues, on suit les étapes suivantes (en utilisant les résultats du précédent code) :

```
/* Garder uniquement les scores des variables sur les deux composantes retenues */
DATA TAB1_STAT ; SET TAB_STAT ;  IF _TYPE_="SCORE"; run;
/* Transposer cette table pour mettre les variables en lignes*/
PROC TRANSPOSE DATA=TAB1_STAT OUT=TAB2_STAT; run;
/* Représentation graphique*/
PROC PLOT DATA=TAB2_STAT ; PLOT PRIN2*PRIN1$_NAME_ / VPOS=30 ;RUN;
```

Il faut simplement noter que PRIN1 et PRIN2 sont les noms automatiquement données aux deux composantes retenues. Il s'agit alors d'une représentation classique. On peut ainsi mettre en forme ce graphique à notre guise.

Représentation des individus

Pour représenter les individus sur les deux premières composantes retenues, on fait PROC PLOT (en se basant sur les résultats de l'ACP réalisée précédemment) :

```
PROC PLOT DATA=MYTABLE;
PLOT PRIN2*PRIN1$ID / VPOS=20 ;
PLOT PRIN2*PRIN1=Y / CONTOUR=2 VPOS=20;
```

Ici PRIN1 et PRIN2 sont les noms automatiquement données aux deux composantes retenues traduisant les scores des individus alors que les variables ID et Y sont des variables de mise en forme du graphique qui sont déjà disponibles dans la table initiale.

IV.2.2. Scoring à partir d'un modèle d'ACP pré-validé

Dans les exemples précédents, nous avons calculé les scores des individus appartenant à l'échantillon sur lequel l'ACP a été réalisée. Mais très souvent, on nous demande de calculer les scores des individus en partant d'un modèle déjà estimé et validé à partir d'un autre échantillon. Il s'agit alors d'utiliser les paramètres de ce modèle pour calculer les scores des individus appartenant au nouvel échantillon. Pour réaliser cette tâche, il faut utiliser la procédure PROC SCORE en se basant sur la table produite par l'option OUTSTAT de PROC ACP. Pour être concret, supposons qu'on veuille calculer les scores d'un groupe de clients d'une banque. Mais le responsable nous impose d'utiliser d'abord une ACP sur un premier groupe clients. Ensuite d'utiliser les paramètres de cette modélisation pour calculer les scores des clients du

second groupe. Pour simplifier, supposons que les informations sur le premier groupe de client soit enregistrées dans une table nommée MYTABLE1 et celles sur les groupe 2 dans une table nommée MYTABLE2. Pour réaliser ce scoring, nous allons suivre les étapes suivantes.

D'abord, on réalise une ACP sur le groupe 1 en faisant :

```
PROC PRINCOMP DATA=MYTABLE1 OUT=MYTABLE1 N=1 NOPRINT PLOTS=none
OUTSTAT=STAT_G1 ;
VAR MYVAR1 MYVAR2…MYVARN ;
RUN;
```

Notons que dans cette ACP, nous avons retenu une seule composante avec l'option N=1 et nous enregistrons les paramètres estimés dans la table STAT_G1.

Dans la seconde étape, nous utilisons PROC SCORE pour calculer les scores des individus du groupe 2 en utilisant STAT_G1. Ainsi, on a :

```
PROC SCORE DATA=MYTABLE2 SCORE=STAT_G1 OUT=MYTABLE2 ;
VAR MYVAR1 MYVAR2…MYVARN ;
RUN;
```

Notons ici que les variables utilisée dans PROC SCORE doivent être exactement les mêmes que celles se trouvant dans la table STA_G1 (c'est-à-dire celle utilisée lors de l'ACP). Aussi, l'option OUT de PROC SCORE permet de créer une nouvelle copiée à partir de la table initiale en ajoutant les variables de score. Toutefois, on peut donner le même nom que la table initiale s'il n'est pas nécessaire de créer une nouvelle table.

IV.2.2. Analyses factorielles des correspondances simples (AFC)

Simplement présentée, on dira que l'analyse factorielle des correspondances simples (AFC) est une ACP lorsque les variables analysées sont de toutes de nature qualitatives (nominale ou ordinales). Toutefois L'AFC diffère de l'ACP par la définition des unités statistique et la notion de distance utilisée pour comparer ces unités. En effet, contrairement à l'ACP, les unités statistiques de l'AFC ne sont pas les individus qui forment les lignes de la table de données (ex : personnes interrogées), mais les répartitions des réponses selon les modalités des questions (prises deux à deux). Les unités statistiques sont définies à part des répartitions conditionnelles des modalités, encore appelés profils lignes et profils colonnes. L'indicateur ainsi utilisé pour apprécier la distance entre deux profils est la distance de khi-deux. La description de

chaque ensemble de profils est effectuée comme en analyse en composantes principales. On recherche les axes les plus proches en se basant sur la distance du khi-deux. Les coordonnées sur ces axes définissent des variables appelées ici souvent facteurs au lieu de composantes principales. Ainsi, en dehors de la phase de construction des tableaux profil-ligne et profil-colonnes, les AFC répond aux même principale que l'ACP. Il n'est donc pas surprenant qu'on trouve des similitudes dans leur mise en œuvre et leur utilisation.

Pour mettre en œuvre l'AFC sous SAS, on utilise la procédure PROC. L'exemple ci-dessous est une illustration :

```
PROC FACTOR DATA=MYTABLE METHOD=P PRIORS=ONE ROTATE=PROMAX N=2
PLOTS=ALL OUT=MYTABLE OUTSTAT=STAT_DATA;
VAR MYVAR1 MYVAR2…MYVARN;
RUN;
```

Comme on peut le constater de nombreuses options sont disponibles dans proc FACTOR. Mais les principales sont METHOD qui spécifie la méthode d'extraction des facteurs. Ici, on choisit la méthode composante principale. PRIORS qui est une option complémentaire à METHOD. La valeur de celle-ci doit être fixée en fonction de la valeur fixée dans METHOD. ROTATE indique la méthode de rotation à choisir. Ici, on choisit PROMAX pour autoriser une certaine corrélation entre les facteurs obtenus. Il y a d'autres valeurs telles que VARIMAX, OBLIMIN, etc…, l'option N=2 indique qu'on veut retenir deux facteurs. PLOT affiche les différents graphiques. Quant aux options OUT et OUTSTAT, elles permettent d'exporter respecter les scores des individus dans la table MYTABLE et les paramètres estimés dans la table STAT_DATA.

Il faut noter que les paramètres estimés se situant dans la table STAT_DATA peuvent être utilisés pour calculer les scores des individus qui n'appartiennent pas initialement à l'échantillon (voir les détails dans le cas de l'ACP). Pour utiliser les paramètres afin d'estimer les scores pour les nouveaux individus, on utilise PROC SCORE comme suit :

```
PROC SCORE DATA=MYTABLE2 SCORE= STAT_DATA OUT=MYTABLE2 ;
VAR MYVAR1 MYVAR2…MYVARN ;
RUN;
```

IV.2.3. Analyses des correspondances multiples

L'analyse des correspondances multiples (ACM) est utilisée lorsque la liste des variables à analyser est constituée à la fois des variables quantitatives et des variables qualitatives. Il faut noter que l'ACM est une généralisation de l'AFC dans la mesure où

elle transforme d'abord toutes les variables quantitatives en des variables catégorielles définies par des intervalles de valeurs. La table de données obtenues sera donc une table constituée uniquement des variables de type qualitatives comme c'est le cas dans une AFC. En effet, comme dans une AFC, l'ACM. décrit les relations deux à deux entre les variables qualitatives à travers une représentation des groupes d'individus correspondant aux diverses modalités. Une fois que les tableaux de profil-lignes et de profil-colonne sont obtenus, une ACP est sur ces tableaux en utilisant la métrique du Khi-deux comme indicateur de distance entre (comme en A.F.C.).
Toutefois, il faut noter que dans l'interprétation des résultats d'une ACM, les pourcentages d'inertie n'ont pas un grand un intérêt. La sélection et l'interprétation des axes factoriels se feront essentiellement à l'aide des contributions obtenue sur les des variables actives.

D'un point de vue pratique, pour mettre en œuvre l'ACM sous SAS, on utilise la procédure PROC CORRESP. L'exemple ci-dessous est une ilustration

```
PROC CORRESP DATA=MYTABLE MCA OUTC=COORD_TAB OUTF=FREQ_TAB
PLOT=ALL;
TABLES MYVAR1 MYVAR2…MYVARN;
RUN;
```

Dans cette formulation, on indique qu'on souhaite réaliser une analyse en correspondance multiple (MCA) au lieu d'une analyse en correspondance simple. Nous indiquons la liste des variables dans l'instruction TABLE. Cette liste peut contenir des variables aussi bien qualitatives que quantitatives. Dans les options, on crée deux tables supplémentaires, une qui contient les coordonnées des différentes modalités (OUTC) et une qui contient les fréquences issues du tableau de Burt.

IV.2.4. Typologies et classifications ascendantes hiérarchiques

Les méthodes de typologie et de classification sont très couramment utilisées dans les études markéting notamment dans les études de segmentation et de ciblage de la clientèle. Cette section a pour but de présenter brièvement les méthodes les plus couramment utilisées.

IV.2.4.1. Typologie : k-means clustering

D'abord s'agissant de la typologie, la procédure la plus couramment utilisée sous SAS PROC FASTCLUS. Cette procédure permet de scinder les individus d'un échantillon en

nombre bien défini de classes. Les individus sont regroupés (en cluster) sur des critères de ressemblance définis sur la notion de distance par rapport aux centroïdes. Un cluster est alors définie comme un regroupement d'individu autour d'une valeur centrale appelée centroïde qui est en fait la moyenne du groupe. Ainsi, en spécifiant un nombre de cluster k, l'algorithme de regroupement détermine d'abord k valeurs de moyennes. Ensuite, il regroupe les individus autour de ces k-valeurs selon la distance par rapport à ces 5 valeurs. D'une manière générale, un individu est assigné à un centroïde lorsque sa distance avec ce centroïde est le plus faible par rapport aux autres centroïde. Ainsi, les individus assignés à une même centroïde forment un cluster. Cette classification ou typologie est généralement qualifiée de k-mean clustering. Pour mettre en œuvre cette démarche, on utilise la procédure PROC FASTCLUS comme suit :

```
PROC FASTCLUS DATA=MYTABLE MAXC=3 OUT=CLUSDATA;
   VAR MYVAR1 MYVAR2…MYVARN;
RUN;
```

Dans la commande ci-dessous, on réalise le k-means clustering en utilisant les variable MYVAR1 jusqu'à MYVARN. Ensuite dans les options, on indique qu'on veut constituer trois clusters au maximum. Nous indiquons également qu'on veut exporter les données avec les clusters constitués dans une table nommée CLUSTDATA. Dans cette table on retrouvera deux variables supplémentaires, la variable CLUSTER qui identifie le cluster auquel appartient l'individu mais aussi la variable DISTANCE qui indique la distance de l'individu par rapport au centre de classe.

IV.2.4.2. Classification Ascendante Hiérarchique (CAH)

La classification ascendante hiérarchique (CAH) est une méthode itérative de regroupement des individus qui obéit aux étapes suivantes. Dans un premier temps, on calcule la dissimilarité entre les N individus deux à deux en utilisant une notion de distance définie à partir de leur caractéristiques. Les individus les plus dissimilaires auront des distances plus élevées alors que les individus les plus semblables auront des distances faibles entre eux. Dans un second temps, on regroupe les deux individus dont le regroupement minimise un critère d'agrégation donné, créant ainsi une classe comprenant ces deux individus. On calcule ensuite la dissimilarité entre cette classe et les N-2 autres individus en utilisant le critère d'agrégation. Puis on regroupe les deux individus ou classes d'individus dont le regroupement minimise le critère d'agrégation. On continue ce processus ainsi jusqu'à ce que tous les individus soient regroupés. Ces regroupements successifs produisent alors un arbre de

classification dont la racine correspond à la classe regroupant l'ensemble des individus. Cet arbre de classification appelé dendrogramme représente une hiérarchie de partitions. On peut alors choisir une partition en tronquant l'arbre à un niveau donné afin d'aboutir à un nombre de classes distinctes. Le choix de ce niveau de coupure peut être arbitrairement fait par l'analyste en fonction de ses propres contraintes ou à partir des critères plus opérationnels.

Pour mettre en œuvre la CAH sous SAS, on utilise la procédure PROC CLUSTER. L'exemple ci-dessous est une illustration.

```
PROC CLUSTER DATA=MYTABLE METHOD=WARD STANDARD OUTTREE=TAB_TREE
PRINT=5 PLOTS(MAXPOINTS=500);
VAR MYVAR1 MYVAR2…MYVARN;
RUN;
```

Dans cette formulation, on réalise PROC CLUSTER en utilisant les variables allant de MYVAR1 à MYVARN. Nous indiquons à SAS de standardiser d'abord les données avant de les utiliser dans l'analyse. La méthode de classification suivie est celle de la distance de WARD. Il y a à peu près une dizaines de critère de distance pour réaliser. Mais la méthode de WARD est la plus couramment utilisée. Ensuite, on indique à SAS d'exporter les résultats dans une table nommée TAB_TREE en utilisant OUTTREE. C'est cette table qui sera utilisée pour réaliser l'arbre de classification afin de choisir le nombre de classe à retenir. Nous décidons ici de fixer l'exportation des résultats sur 5 niveaux de l'arbre. Nous augmentons, aussi, par précaution le nombre de point à afficher dans le dendrogramme. Cette option peut être utile dans certains contextes. Par exemple lorsque le nombre de cluster initial est très élevé (>200), SAS ne pourra plus afficher le dendrogramme car la valeur maximale par défaut est 200. C'est pourquoi, il faut souvent lire le fichier log pour prendre connaissance du message affiché à la suite de l'exécution. Bien entendu, il y d'autres options utiles qu'il faut consulter dans la base de connaissance de SAS.

Une fois que la procédure PROC CLUSTER est exécutée, la seconde étape est de réaliser le dendrogramme en utilisant la procédure complémentaire PROC TREE. Celle-ci doit être appliquée sur la table de sortie définie dans l'option OUTTREE. La commande alors se présente comme suit :

```
PROC TREE DATA= TAB_TREE  NCL=3   OUT=CLUSTERDATA ;  RUN;
```

Dans cette commande, nous indiquons à SAS de regrouper les individus en trois clusters. Les résultats seront alors exportés dans une table nommée CLUSTERDATA.

Notons que la fixation du nombre de clusters doit être faite après avoir examiné le dendrogramme. C'est pourquoi, il faut exécuter PROC TREE avec un nombre arbitraire de clusters. Ainsi, après avoir examiné le dendrogramme, on l'exécute une nouvelle fois en ayant la valeur finale du nombre de cluster.

IV.2.4.3. Quelques règles de bonnes pratiques dans les typologies et les classifications

Les typologies et les classifications étant réalisées à partir des notions de distance, il est important d'adopter quelques règles de bonnes pratiques afin d'améliorer l'efficience et la qualité des méthodes utilisées.

En effet, il est fortement conseillé d'utiliser d'abord des analyses en composantes principales ou des analyses factorielles afin de réduire la dimension des données à un nombre limité de composantes ou de facteurs. Au cas où l'ACP serait utilisé il faut d'abord veiller à standardiser les variables afin d'éliminer de potentiels effets d'échelle. Ainsi, une fois que ces composantes ou ces facteurs sont obtenus, on peut alors réaliser la typologie ou la classification en considérant ces variables de score. Il faut toutefois veilleur à ce que ces composantes ou ces facteurs retenus représentent une part significativement importante de l'inertie totale des données.

IV.4. Les modèles de régressions linéaires et logistiques

Dans cette section, nous étudions la mise en œuvre de deux principaux modèles: le modèle de régression linéaire multiple et le modèle de régression logistiques binaire. Dans le premier cas, il s'agit de modéliser un phénomène de nature quantitative par des variables qui peuvent être à la fois de type quantitatif ou qualitatif. Quant au second, il consiste à modéliser (sous forme de probabilité) un phénomène de type aléatoire binaire en utilisant des facteurs explicatifs qui peuvent être de nature quantitatif ou qualitatif. L'objectif de cette section est de faire une brève présentation de chacun des deux méthodes en montrant dans quel contexte on peut avoir recours à telle ou telle méthode. Nous discutons surtout de leur mise œuvre pratique sous SAS.

IV.4.1. Les modèles de régressions linéaires multiples

Le modèle de régression multiple est un modèle dans lequel on chercher à mesurer le lien entre une variable quantitative et un ensemble variables qui peuvent être soit quantitatives ou qualitatives. Dans ce cas on dit qu'on cherche à expliquer la variable.

Le modèle de régression est donc constitué de deux catégories de variables. D'une part, une variable dépendante c'est-à-dire celle qu'on cherche à expliquer et d'autre part une ou des variables indépendantes c'est-à-dire celles dont on veut mesurer l'influence sur la variable dépendante. Il existe aussi différentes qualifications pour distinguer les deux types de variables. La variable dépendante est souvent appelée variable expliquée et tandis que les variables indépendantes sont qualifiées de variables explicatives. Dans la modélisation prédictive, la variable dépendante est aussi qualifiée de variable prédite alors que les variables explicatives sont appelées facteurs prédicteurs. Ces qualifications sont donc interchangeables à tout point de vue.

Les modèles de régressions multiples sont très fréquemment rencontrés dans les études marketing. Par exemple, supposons qu'un banquier s'interroge sur les facteurs qui déterminent le montant des retraits d'espèces les clients par mois. Il soumet alors son interrogation au chargé d'études statistiques. Celui-ci décide alors d'estimer un modèle de régression linéaire multiple car la variable à expliquer est le montant total des retraits par mois par client. Parmi les variables explicatives, il peut choisir l'âge, le sexe, le type de contrat de travail, la situation familial, le nombre de produits bancaires auquel il a souscrit, etc… Le but final de l'analyse sera alors de déterminer parmi l'ensemble des facteurs choisis ceux qui influencent significativement le montant des retraits.

Pour mettre en œuvre une telle modélisation sous SAS, on utilise la procédure PROC REG. Dans sa structure la plus simple, cette procédure peut se présenter comme suit :

```
PROC REG DATA=MYTABLE ;
MODEL  DEPVAR= INDEPVAR1 INDEPVAR2 INDEPVARN / SELECTION=NONE
COLLIN VIF TOL  ;
RUN;
```

Dans cette formulation, le modèle est défini tel que DEPVAR représente la variable dépendante et INDEPVAR1 jusqu'à INDEPVARN représente les variables indépendantes. L'option SELECTION indique la méthode à utiliser pour la sélection des variables significatives. Cette option montre son utilité lorsque le nombre de

variables explicatives est très élevé et qu'on ne peut plus procéder par sélection manuelle. Quant aux options COLLIN VIF et TOL, ce sont des tests de diagnostiques sur le modèle permettant d'examiner la multicolinéarité entre les variables.

Par ailleurs, il faut signaler que même si le modèle linéaire permet de prendre en compte les variables qualitatives, celles-ci doivent être transformées en variables binaires définie sur chaque modalité de la variable. Par exemple, pour une variable qualitatives définie sur 4 modalités, on créer trois variables binaires correspondant par exemple aux trois premières modalités. Ces trois variables binaires sont alors incluses dans le modèle pour représenter la variable qualitative en question.

Quant à la qualité du modèle obtenu, elle doit être appréciée en se basant sur différents indicateurs notamment le R-carré ajusté mais sur différents tests de diagnostiques sur les résidus estimés (ces tests de diagnostic ne seront pas présentés ici).

IV.4.2. Régressions logistiques binaires

IV.4.2.1. Estimation d'un modèle logistique binaire LOGIT

Les modèles de régressions logistiques binaires sont les modèles les plus couramment utilisés dans les modélisations explicatives lorsque la variable à expliquer est de type binaire (oui ou non). Pour montrer dans quel genre de contexte on peut avoir recours à une modélisation logistique binaire. Prenons l'exemple d'un banquier qui propose à ses clients un nouveau produit financier. Pour simplifier faisons l'hypothèse que la banque a informé ses clients par tous les canaux habituels (courrier, Email, téléphone, en agence, publicité) de telle sorte que tous les clients sont au courant de la disponibilité du produit. Après six mois de campagne, au vu du taux de souscription, le responsable marketing souhaite maintenant comprendre quels sont les facteurs qui influencent (positivement ou négativement) la décision de souscription au produit en se basant sur les caractéristiques des clients. En confiant ce travail au Chargé d'études statistiques. Celui-ci décide alors de mettre en place un modèle de régression logistique.

Dans ce modèle, la variable dépendante est une variable binaire qui vaut 1 lorsque le client a souscrit au produit et 0 s'il n'a pas souscrit. Les potentiels variables explicatives choisies peuvent alors être les caractéristiques sociodémographiques classiques (âge, sexe, éducation, etc..) mais aussi d'autres variables liées aux activités bancaires (ancienneté, le nombre de produits financiers préalablement souscrit, le

revenu mensuel, la détention d'un titre financier de type action/obligation, etc…). Le but alors est d'identifier les principaux facteurs (significatifs) qui motivent ou qui démotive le client dans son choix de souscrire ou pas au produit en question.

Pour estimer un modèle de régression logistique binaire sous SAS, on utiliser la procédure PROC LOGISTIC. La commande ci-dessous est une illustration :

```
PROC LOGISTIC DATA=MYTABLE DESCENDING;
MODEL  DEPVAR= INDEPVAR1 INDEPVAR2 INDEPVARN /LINK=LOGIT;
RUN ;
```

Dans cette modélisation, la variable dépendante est une variable binaire qui prend 1 lorsque le phénomène étudié se réalise et 0 sinon. Par défaut SAS modéliser la plus petite valeur. C'est pourquoi nous mettons l'option DESCENDING pour qu'il commence la modélisation par la plus grande valeur c'est-à-dire 1 au lieu de 0. L'option LINK permet de spécifier qu'il s'agit d'un modèle LOGIT qui est la valeur qui représente le modèle logistique binaire. Il y a également d'autres valeurs telles que PROBIT ou NORMIT qui permettent d'estimer un modèle PROBIT.

Dans la formulation présentée ci-dessus, le modèle logistique est estimé dans la même structure que le modèle linaire. La seule différence réside dans le fait que les coefficients obtenus dans le modèle logistique ne sont pas directement interprétables. Il faut calculer les effets marginaux qui mesurent comment varie la probabilité de survenue de l'événement étudiée lorsque les caractéristiques sont modifiées. A ce propos, il faut signaler qu'il existe une autre procédure qui permet d'estimer le modèle LOGIT en permettant d'obtenir directement les effets marginaux. Il s'agit de la procédure PROC QLIM. Elle se présente comme suit :

```
PROC QLIM DATA=MYTABLE ;
MODEL  DEPVAR= INDEPVAR1 INDEPVAR2 INDEPVARN / DISCRETE(D=LOGIT);
OUTPUT OUT=EST_DATA MARGINAL; RUN;
```

Dans cette estimation, on indique que les résultats des estimations doivent être stockés dans une table nommée EST_DATA en calculant les effets marginaux individuels. Ainsi, pour calculer les effets marginaux moyens pour chaque variable on peut utiliser PROC MEANS suivant :

```
PROC MEANS DATA= EST_DATA NOLABELS MEAN MAXDEC=4
ORDER=FORMATTED FW=8  VARDEF=DF  ;
VAR  MEFF_P2_: ;
RUN;
```

Par ailleurs, il ne faut pas oublier que tout comme dans le modèle de régression linéaire, les variables explicatives spécifiées dans la clause MODEL doivent, toutes, être mises dans un format quantitatif ou binaire. Ainsi, toutes les variables qualitatives multimodales doivent être éclatées en des variables binaires pour être ensuite introduite dans le modèle.

IV.4.2.2. Scoring et segmentation à partir d'un modèle logistique

La modélisation logistique est très couramment utilisée dans les études marketing notamment pour le scoring et la segmentation des clients. Ce type de modèle est beaucoup plus utilisé dans les calculs de scores d'appétence, de score d'attrition ou même dans l'évaluation de risques de défaut d'engagement. Dans le domaine bancaire, par exemple, notamment dans l'analyse des risques crédit, le modèle logistique binaire peut être utilisé pour attribuer des scores aux emprunteurs qui déterminent les niveaux de risque de défaut de remboursement ou de paiement en fonction.

Pour introduire cette section, commençons par présenter un cas concret dans lequel le modèle logistique peut être utilisé afin d'ajuster une campagne marketing. Un opérateur téléphonique constate une baisse significative de son chiffre d'affaire à cause d'une augmentation du taux de résiliation des contrats par les clients. Il décide de mettre en place un projet de scoring visant à identifier les clients les plus susceptibles de résilier prochainement leur contrat. Il confie alors ce travail au Chargé d'études statistiques et markéting. Celui-ci décide alors de mettre en place une méthodologie de scoring en se basant sur un modèle logistique binaire.

La mise en œuvre d'une méthodologie de scoring à partir d'un modèle logistique se fait en trois grandes étapes.

Dans la première étape, on scinde d'abord l'échantillon initial en deux sous échantillons : un échantillon d'apprentissage et un échantillon de validation. L'échantillon d'apprentissage est l'échantillon qui sert de base à l'estimation du modèle logistique tandis que l'échantillon de validation sert à examiner la qualité du modèle obtenu. Il sert à valider ou à invalider le modèle. Généralement l'échantillon de validation doit avoir une taille de l'ordre de 25 à 30% de l'échantillon initial. Tandis que les 70 à 75% restant serviront à l'estimation du modèle.

Dans la seconde étape, on utilise l'échantillon d'apprentissage pour estimer un modèle logistique dans lequel la variable dépendante est la variable binaire, qui est dans notre cas ici, égale à 1 lorsque le client a résilié son contrat et 0 sinon. Pour

cette modélisation, on choisit toutes les variables explicatives potentielles (qualitatives et quantitatives). Dans cette étape on doit veiller à discrétiser toutes les variables quantitatives afin de les rendre sous formes de variables qualitatives, qui seront ensuite éclatées en des variables binaires. Toutes les variables qualitatives doivent également être transformées de sorte à obtenir des variables binaires. Au final, le modèle logistique estimé doit avoir comme variables explicatives essentiellement que des variables binaires également. Une fois que le traitement et l'organisation des données effectués, on peut estimer le modèle logistique. Pour cela, on utilise la procédure PROC LOGISTIC telle que décrite précédemment. Toutefois, lors de l'estimation du modèle, on peut aussi directement calculer les scores (ou les propensions) en ajoutant des options supplémentaires. La ligne de commande ci-dessous fournit un exemple :

```
PROC LOGISTIC DATA=MYTABLE1  DESCENDING PLOTS=NONE;
MODEL DEPVAR= INDEPVAR1 INDEPVAR2 INDEPVARN /LINK=LOGIT ;
OUTPUT OUT=MYTABLE1 P=P;
SCORE DATA =MYTABLE2 OUT = MYTABLE2 FITSTAT OUTROC=ROCDATA;
RUN;
```

Cette procédure contient de nombreux éléments qu'il faut un peu détailler. En effet dans la même procédure nous avons estimé le modèle mais aussi calculer les statistiques permettant la validation du modèle.

D'abord, signalons que le modèle logistique est estimé en considérant MYATBLE1 qui représente l'échantillon d'apprentissage. Ensuite, nous ajoutons la clause OUTPUT OUT afin de calculer les scores (probabilité de résiliations) pour tous les individus l'échantillon d'apprentissage.

Dans la clause SCORE, nous utilisons l'échantillon de validation MYTABLE2 afin de calculer les scores (probabilités) des individus appartenant à cet échantillon. Ces scores sont ajoutés à la table MYTABLE2 et enregistrée sous le même nom dans l'option OUT. Ensuite, nous spécifions l'option OUTROC afin d'indiquer une table dans laquelle les statistiques sur la qualité du modèle doivent stockées. Il s'agit notamment des statistiques servant à tracer la courbe ROC, sensibilité, spécificité, valeur prédictive positive, valeur prédictive négative, et… Bien sûr ces statistiques sont calculées à partir de l'échantillon de validation afin d'apprécier le pouvoir prédictif du modèle. A présent, nous avons toutes les informations nécessaires pour examiner la qualité et le pouvoir prédictif du modèle.

La troisième étape de la démarche est la validation du modèle en examinant les indicateurs de qualité du modèle. Pour valider un modèle plusieurs critères peuvent être examinés. Ces critères sont fixés selon qu'il s'agisse d'examiner la qualité de l'ajustement ou le pouvoir prédictif du modèle. S'agissant de la qualité de l'ajustement les principaux critères sont les tests (Likelihood, Score, Wald), les critères (AIC, SC, etc..), les R-Square (pseudo, McFadden, etc..). Pour ce qui concerne le pouvoir prédictif les indicateurs utilisés sont la sensibilité, la spécificité, la valeur prédictive positive, la valeur prédictive négative, le taux de bien classés / vrais positifs, la courbe ROC, la courbe de lift ainsi que l'indice de Gini. Nous allons nous intéresser principalement aux indicateurs sur le pouvoir prédictif.

Le pourcentage de prédictions correctes ou taux de bien classés

Le pourcentage de prédictions correctes (ou taux de bien classés) est le premier indicateur sur le pouvoir prédictif d'un modèle logistique binaire. Pour obtenir ce taux, on se base sur le tableau de contingence obtenu en croisant la variable de classification effectuée par le modèle avec la classification selon les données observées. Pour obtenir ce tableau, on peut utiliser la commande suivante :

PROC FREQ DATA=MYTABLE2; TABLE I_*DEPVAR**F_*DEPVAR*; **RUN**;

Où MYTABLE2 est la table dans laquelle sont enregistrées les probabilités estimées sur l'échantillon de validation. Cette table est obtenue avec l'option OUT de l'instruction précédemment illustrée. Dans cette table, on trouvera deux nouvelles variables nommée en suffixant le nom de la variable dépendante initiale. Si cette variable est nommée DEPVAR alors les deux variables de classification seront nommées I_DEPVAR et F_DEPVAR. I_DPVAR est la classification obtenue à partir du modèle tandis que F_DEPVAR est la classification basée les faits observés. Elle équivaut donc la variable initiale DEPVAR. Ainsi, en faisant un croisement entre les deux variables on obtient un tableau présentant sous la forme d'un tableau à double-entrée. Le tableau ci-dessous est un exemple illustratif. Le phénomène étudié est la résiliation du contrat par le client. La variable est donc définie de telle sorte qu'elle vaut lorsque l'individu a résilié et 0 s'il n'a pas résilié.

		Réalité		
		0	**1**	Total
Modèle	**0 (-)**	461	14	475
	1 (+)	49	56	105
Total		510	70	580

Dans ce tableau on peut remarquer que sur un échantillon de 580 clients, 510 n'ont pas résilié leur contrat alors que 70 ont résilié. En croisant ces résultats avec les prédictions du modèle, on peut faire les remarques suivantes :

- Le modèle prédit que 461 clients n'ont pas résilié leur contrat et effectivement dans la réalité ils n'ont pas résilié.
- Le modèle prédit que 14 clients n'ont pas résilié leur contrat alors que dans la réalité ils ont résilié.
- Le modèle prédit que 49 clients ont résilié leur contrat alors que dans la réalité, ils n'ont pas résilié
- Et enfin le modèle prédit que 56 clients ont résilié leur contrat et effectivement dans la réalité, ils ont résilié.

En se basant sur ces différents résultats on peut élaborer un certain nombre d'indicateur permettant de mesurer la capacité prédictive du modèle. Parmi ces indicateurs, on a notamment :

- **La sensibilité** qui mesure le pourcentage de positifs parmi ceux qui ont résilié. Soit 56/70=80%.
- **La spécificité** qui est égale au pourcentage de négatifs parmi ceux qui n'ont pas résilié. Soit 461/510=90%

Les valeurs de la sensibilité et de la spécificité permettent de faire une première appréciation sur la qualité prédictive du modèle. Un bon modèle est un modèle pour lequel ces deux valeurs sont élevées.

On peut aussi ajouter à la sensibilité à la spécificité les indicateurs de valeur prédictive positive et de valeur prédictive négative.

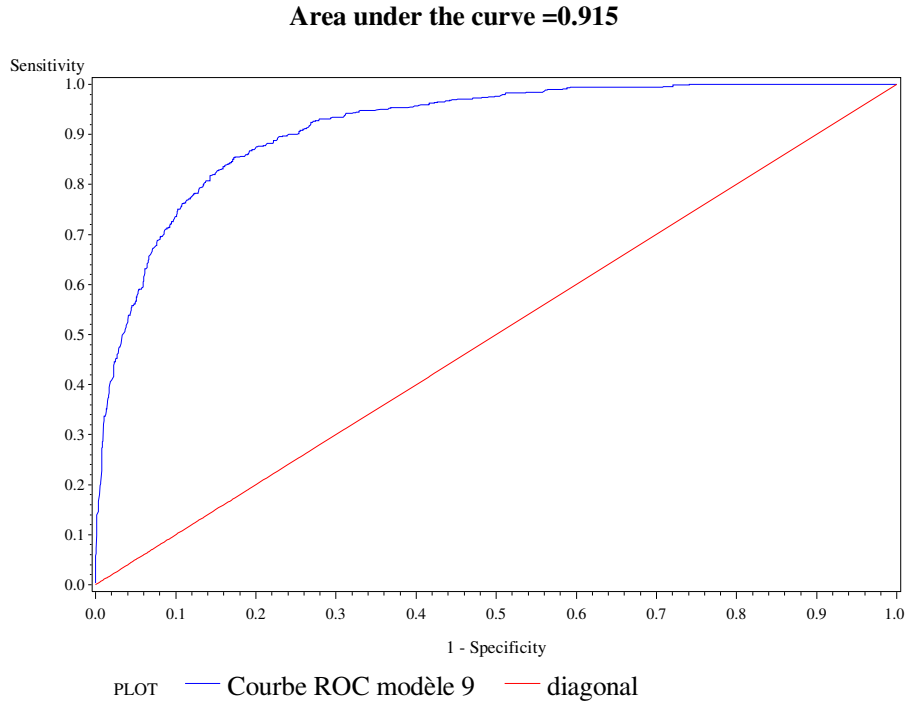
- **La valeur prédictive positive** (VPP) est le pourcentage de résiliés par les positifs. Soit 56/105=53%. On constate que la VPP est la réciproque de la sensibilité.

- **La valeur prédictive négative** est le pourcentage de non résilié par les négatifs. Soit 461/475=93%. Là également la VPN est la réciproque de la spécificité.

Au final connaissant la valeur de la sensibilité et de la spécificité, on peut maintenant calculer **le pourcentage de prédictions correctes** ou le taux de bien classés. En effet, en regardant le tableau ci-dessus, les bien classés sont ceux pour lesquels le modèle correspond à la réalité c'est-à-dire même conclusion (négatif-non résilié et positif-résilié). Le nombre total de prédictions correctes est alors égal à 461+56=517. En divisant ce total par le nombre total de clients dans l'échantillon 571/580=89%. Ce qui représente une valeur relativement élevée pour que le modèle soit considéré comme de bonne capacité prédictive.

La courbe ROC et l'AUC

La courbe ROC (Receiver Operating Characteristic) est un indicateur sur la qualité prédictive du modèle estimé. C'est une représentation graphique qui donne le taux de vrais positifs (pourcentage des positifs qui ont effectivement résilié, sensibilité) en fonction du taux de faux positifs (pourcentage positifs n'ont pas résilié, 1-spécificité). Le graphique ci-dessous illustre l'allure de la courbe ROC.



Pour obtenir cette courbe sous SAS, on utilise la procédure PROC GPLOT sur la table ROCDATA obtenue avec l'option OUTROC de l'instruction SCORE lors de l'estimation

du modèle (voir plus haut). Cette table contient la sensibilité, ainsi que 1-spécificité. On utilise alors ces deux indicateurs pour représenter la courbe ROC comme suit :

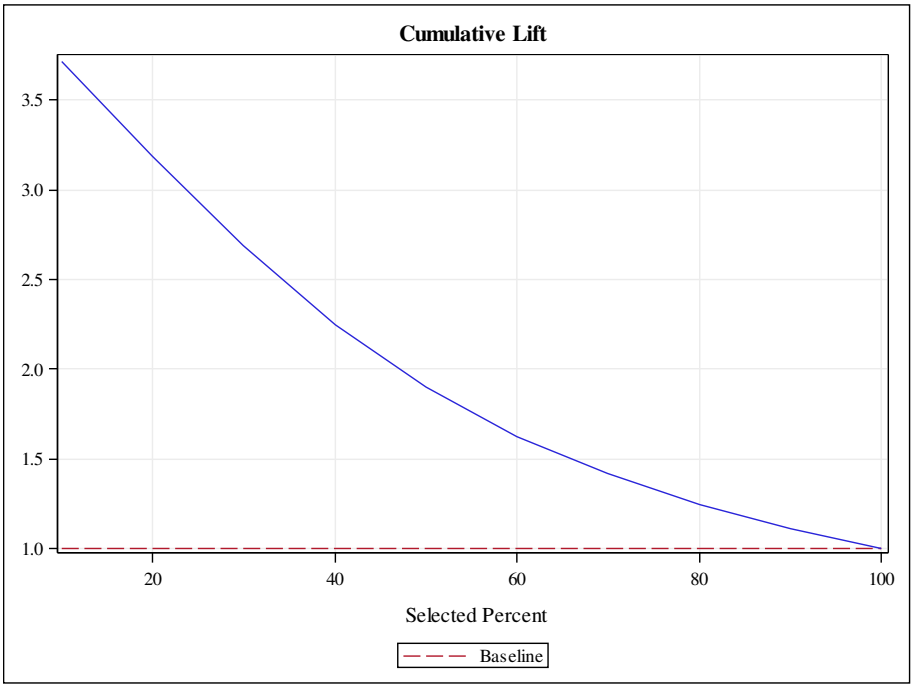
```
DATA ROCDATA ; SET ROCDATA; DIAGONAL=_1MSPEC_ ; RUN ;
PROC GPLOT DATA = ROCDATA ;
SYMBOL1 INTERPOL=JOIN VALUE=NONE COLOR=BLUE;
SYMBOL2 INTERPOL=JOIN VALUE=NONE COLOR=RED;
LEGEND1 VALUE=(HEIGHT=2 "COURBE ROC");
TITLE  HEIGHT=2  "AREA UNDER THE CURVE";
PLOT ( _SENSIT_ DIAGONAL )*_1MSPEC_  / LEGEND=LEGEND1   OVERLAY  ; RUN;
```

Notez aussi qu'on peut obtenir la courbe ROC directement à partir de l'estimation du modèle en ajoutant l'option à PROC LOGISTIC l'option PLOTS(ONLY)=(ROC(ID= CASENUM)).

S'agissant de la qualité, le modèle est meilleur lorsque la courbe s'éloigne de plus en plus de la diagonale et se rapproche de plus en plus de l'angle droit à gauche. Par ailleurs, la qualité du modèle est jugée à partir de l'aire sous la courbe AUC (Area Under Curve). Plus cette valeur est proche de 1 plus le modèle est de meilleur qualité.

La courbe lift

La courbe lift est l'un des critères les plus importants pour juger de la performance d'un modèle de score. Il correspond à la concentration d'individus ayant réalisé l'évènement étudié en fonction du nombre d'individus sélectionnés. La courbe ci-dessous est une illustration :



Pour obtenir cette courbe sous sas, on peut utiliser la macro **Gain lift** téléchargeable à partir du site de support SAS (http://support.sas.com/kb/41/683.html)

La courbe LIFT est un indicateur de la performance du modèle dans le ciblage des clients en fonction de leur score. Le graphique ci-dessus montre par exemple qu'en sélectionnant 20% de l'échantillon en utilisant le modèle, le nombre résiliés détectés est 3.2 fois plus élevés qu'avec une sélection aléatoire (sans modèles). Et en sélectionnant 40% de l'échantillon avec le modèle, le nombre de résiliés détectés est 2.25 fois plus élevé qu'avec une sélection aléatoire. La courbe LIFT étant décroissante, elle cela signifie que l'efficacité du modèle est maximale lorsque la proportion d'échantillon sélectionnée est faible

Efficacité du score calculé

La seconde phase de l'évaluation d'une modèle est l'examen de l'efficacité des scores calculés. Examiner l'efficacité du score revient à répondre à la question : est-ce que les clients ayant des scores élevés ont plus résilié que les clients ayant des scores faibles ? Pour pouvoir répondre à cette question, il faut constituer et comparer 2 groupes : ceux qui ont un score élevé et ceux qui ont un score faible. Ensuite, calculer le taux de résiliation pour chacune de ces populations. Néanmoins, une telle démarche nécessite donc d'intégrer aussi dans la campagne de ciblage les clients moins enclin à résilier leur contrat.

Bibliographie

Burlew, Michele M.(2006), SAS® Macro Programming Made Easy, Second Edition. Cary, NC: SAS Institute Inc.,.

Cody, R. P., & Smith, J. K. (2006). Applied statistics and the SAS programming language (5th ed.). Upper Saddle River, NJ: Pearson/Prentice-Hall.

Constable, N. (2007). SAS programming for Enterprise Guide users. Cary, NC: SAS Institute.

Davis, J. B. (2007). Statistics using SAS Enterprise Guide. Cary, NC: SAS Institute.

Der, G., & Everitt, B. S. (2007). Basic statistics using Enterprise Guide: A primer. Cary, NC: SAS Institute.

Dilorio, F., et. al., (2004) "Dictionary Tables and Views: Essential Tools for Serious Applications" Proceedings of the Twenty Ninth SAS Users Group International Conference, 237-29

Gamst, G., Meyers, L. S., & Guarino, A. J. (2008). Analysis of variance designs: A conceptual and computational approach with SPSS and SAS. New York: Cambridge University Press.

Hatcher, L. (2003). Step-by-step basic statistics using SAS: Student guide and exercises. Cary, NC: SAS Institute.

Hatcher, L., & Stepanski, E. J. (1994). Step-by-step approach to using the SAS system for univariate and multivariate statistics. Cary, NC: SAS Institute.

Knuth, D.E. (1973), The Art of Computer Programming, Volume 3. Sorting and Searching, Reading, MA: Addison-Wesley.

Peng, C. Y. J. (2009). Data analysis using SAS. Thousand Oaks, CA: Sage.

SAS (2011), SAS® 9.2 Language Reference: Dictionary, Fourth Edition. Cary, NC: SAS Institute Inc.

SAS (2011), SAS® 9.3 Macro Language: Reference. Cary, NC: SAS Institute Inc.

SAS (2011), SAS® 9.3 SQL Procedure User's Guide. Cary, NC: SAS Institute Inc

Schlotzhauer, S., & Littell, R. (1997). SAS system for elementary statistical analysis (2nd ed.). Cary, NC: SAS Institute.

Slaughter, S. J., & Delwiche, L. D. (2006). The little SAS book for Enterprise Guide 4.1. Cary, NC: SAS Institute