Formation au logiciel SAS

Statistical Analysis System

Odile Wolber (CNAM)

Objectif du cours : acquérir les principes du langage de programmation SAS

Etape DATA

- import des données
- saisie des données sous SAS
- création d'un fichier texte à partir d'une table SAS
- manipulation des données au cours d'une étape DATA
- chargement et fusion de tables SAS

Objectif du cours : acquérir les principes du langage de programmation SAS

Etape PROC

- imprimer les observations d'une table SAS : proc PRINT
- trier un fichier selon des clés : proc SORT
- afficher des comptages et des pourcentages : PROC FREQ
- calculer des statistiques simples : PROC MEANS
- calculer des corrélations : PROC CORR

Objectif du cours : acquérir les principes du langage de programmation SAS

Etape PROC

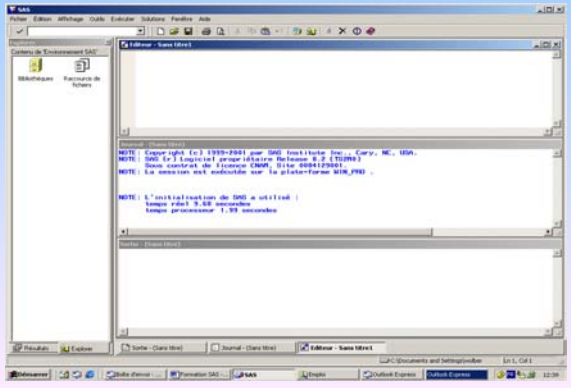
- analyser une variable en détail : PROC UNIVARIATE
- afficher le contenu d'une table : PROC CONTENTS
- effectuer des opérations globales sur les tables : PROC DATASETS
- créer des formats utilisateur : PROC FORMAT

Objectif du cours : acquérir les principes du langage de programmation SAS

Autre modules SAS

- langage matriciel : module SAS IML
- gestion de données informatiques des SGBD relationnels : SQL
- graphiques : SAS GRAPH
- macro langage SAS

Les fenêtres de SAS



Etape DATA

SAS possède son propre système de gestion de bases de données.
Les données sont structurées sous forme de tables SAS (DATA SET).
Une étape DATA débute par l'instruction :

DATA nom_de_table;

Si aucune table SAS n'est créée (recopie d'une table SAS dans un fichier texte) :

DATA _NULL_;

Ajout, suppression de variables ; fusion de tables SAS

Etape PROC

Analyse des tables SAS au moyen de procédures (ETAPE PROC)

Par exemple :
- imprimer les observations d'une table
- afficher la liste des variables contenues dans une table
- obtenir la distribution d'une variables
- demander le calcul d'une moyenne
- effectuer une analyse factorielle, une classification

I. Etape DATA

Table SAS : matrice croisant en lignes les observations et en colonnes les variables.

Exemple : 15 premières lignes de la table SAS Russet

Obs	Pays	gini	farm	rent	gnpr	labo	inst	ecks	deat	demo
1	Argentine	86.3	98.2	32.9	374	25	13.6	57	217	2
2	Australie	92.9	99.6	.	1215	14	11.3	0	0	1
3	Autriche	74	97.4	10.7	532	32	12.8	4	0	2
4	Belgique	58.7	85.8	62.3	1015	10	15.5	8	1	1
5	Bolivie	93.8	97.7	20	66	72	15.3	53	663	3
6	Brésil	83.7	98.5	9.1	262	61	15.5	49	1	3
7	Canada	49.7	82.9	7.2	1667	12	11.3	22	0	1
8	Chili	93.8	99.7	13.4	180	30	14.2	21	2	2
9	Colombie	84.9	98.1	12.1	330	55	14.8	47	316	2
10	Costa Rica	88.1	99.1	5.4	307	55	14.8	19	24	2
11	Cuba	79.2	97.8	53.8	361	42	13.6	100	2900	3
12	Danemark	45.8	79.3	3.5	913	23	14.6	0	0	1
13	Rép. Dominic.	79.5	98.5	20.8	205	56	11.3	6	31	3
14	Equateur	86.4	99.3	14.6	204	53	15.1	41	18	3
15	Egypte	74	98.1	11.6	133	64	15.8	45	2	3

Source : Les données de Russet – La régression PLS – Michel Tenenhaus – Editions Technip – page 172

I. Etape DATA

Tables permanentes / tables temporaires

Une table permanente est **stockée dans une librairie**. La librairie est définie par une **instruction LIBNAME** et assurant la correspondance entre le nom de librairie SAS et le nom physique (répertoire) de la librairie :

libname pays 'c:\Odile Wolber\Cours SAS';

La table temporaire est détruite à la fin de la session SAS en cours :

Par exemple, si on veut stocker de façon permanente la table temporaire Russet dans la librairie pays, la syntaxe est la suivante :

```
data pays.Russet;
  set Russet;
run;
```


I. Etape DATA

Import de données : création d'une table SAS à partir d'un fichier texte

Syntaxe :

```
DATA nom_de_table;
  INFILE fichier_en_entrée;
  INPUT format_de_lecture;
RUN;
```

DATA permet de nommer la table SAS à créer

INFILE indique la référence du fichier à lire en entrée

INPUT précise comment lire les données (position de la variable, son type – caractère ou numérique - , nom de la variable)

I. Etape DATA

Import de données : création d'une table SAS à partir d'un fichier texte

Options de l'instruction INFILE

FIRSTOBS=numéro_d'observation
Numéro de la première observation du fichier en entrée à prendre en compte dans la table SAS créée.

OBS=numéro_d'observation
Numéro de la dernière observation du fichier en entrée à prendre en compte dans la table SAS créée.

MISSOVER
En cas d'enregistrements de longueur variable, si toutes les variables de l'ordre INPUT n'ont pas été lues, les variables non renseignées sont mises à valeur manquante.

I. Etape DATA

Import de données : création d'une table SAS à partir d'un fichier texte

Exemple :

Le fichier Russet est initialement un fichier texte stocké dans le répertoire c:\Odile Wolber\Cours SAS\Pays.txt. On crée une table SAS permanente du même nom, stockée dans le même répertoire (librairie SAS pays précédemment définie par une instruction libname).

```
DATA pays.russet;
  INFILE 'c:\Odile Wolber\Cours SAS\Pays.txt';
  …;
RUN;
```

ou de manière équivalente, en utilisant l'instruction filename :

```
FILENAME russet 'c:\Odile Wolber\Cours SAS\Pays.txt';
DATA pays.russet;
  INFILE russet;
  …;
RUN;
```


I. Etape DATA

Import de données : création d'une table SAS à partir d'un fichier texte

Exemple :

Si on utilise les options firstobs= et obs= :

```
DATA pays.russet;
  INFILE 'c:\Odile Wolber\Cours SAS\Pays.txt' firstobs=2 obs=4;
  …;
RUN;
```

Seules les lignes 2, 3 et 4 sont recopiées dans la table SAS pays.russet :

Obs	Pays	Gini	Farm	Rent	Gnpr	Labo	Inst	Ecks	Deat	Demo
1	Australie	92.9	99.6	.	1215	14	11.3	0	0	1
2	Autriche	74.0	97.4	10.7	532	32	12.8	4	0	2
3	Belgique	58.7	85.8	62.3	1015	10	15.5	8	1	1

I. Etape DATA

Import de données : création d'une table SAS à partir d'un fichier texte

Instruction INPUT

L'instruction la plus sûre consiste à préciser la colonne de début et la colonne de fin d'une variable.

On distingue les variables numériques et caractères, en faisant précéder les variables caractères par un signe $.

Syntaxe :

```
INPUT nom_de_variable <$> <colonne_de_début> <colonne_de_fin>;
```

Si la variable n'occupe qu'une colonne, la colonne de fin peut être omise.

Exemple :

```
DATA pays.russet;
  INFILE 'c:\Odile Wolber\Cours SAS\Pays.txt';
  INPUT Pays $ 1-13 Gini 15-18 Farm 20-23 Rent 25-28…Demo 52 ;
RUN;
```


I. Etape DATA

Import de données : création d'une table SAS à partir d'un fichier texte

Instruction INPUT

Au fur et à mesure de l'exécution de l'instruction INPUT, un élément virtuel, appelé pointeur de colonne est positionné afin de lire la variable suivante. Ce pointeur peut être déplacé avec les instructions suivantes :

- **INPUT @n déplace le pointeur à la colonne n**
- **INPUT +n déplace le pointeur de n colonnes vers la droite**
- **INPUT / déplace le pointeur à la ligne suivante**

Ces instructions posent des problèmes lorsque le fichier contient des valeurs manquantes. C'est pourquoi on préférera préciser les colonnes de début et de fin d'une variable (à condition toutefois que les enregistrements soient de longueur fixe).

I. Etape DATA

Saisie des données sous SAS

Si les données sont saisies sous SAS, elles doivent :

- suivre directement l'étape **DATA**
- être précédées du mot clé **CARDS**
- être suivies d'un caractère **;** placé en début de ligne.

I. Etape DATA

Saisie des données sous SAS

Exemple :

```
data parc;
 input ZONE HERB $ BOSQUET $ EAU $ HIPPO $;
 cards;
 1 courte oui proche moyen
 2 haute oui éloignée aucun
 3 courte oui proche aucun
 4 courte oui proche important
 5 courte oui moyenne aucun
 6 courte oui proche aucun
 7 courte non éloignée aucun
 8 courte oui moyenne important
 9 courte non éloignée aucun
 10 courte non éloignée aucun
 ;
run;
```

I. Etape DATA

Export de données : création d'un fichier texte à partir d'une table SAS

Pour créer un fichier texte à partir d'une table SAS :

- assigner un fichier de sortie par une instruction **FILENAME**
- créer une étape **DATA _NULL_**

(l'objectif n'est pas de construire une table SAS, mais de recopier les éléments d'une table SA dans un fichier externe)

- formater les variables à l'aide de l'instruction **PUT**

I. Etape DATA

Export de données : création d'un fichier texte à partir d'une table SAS

Instruction PUT

L'instruction PUT s'utilise avec la même logique que l'instruction INPUT. Elle permet d'écrire les données en utilisant éventuellement un format d'écriture.

Syntaxe :

```
PUT <contrôle du pointeur d'écriture> <nom_de_variable1> <format_d'écriture>;
…
PUT <contrôle du pointeur d'écriture> <nom_de_variablen> <format_d'écriture>;
```

- **PUT @n var1 2.** déplace le pointeur à la colonne n
- **PUT +n var 1 2.** déplace le pointeur de n colonnes
- **PUT / var1 2.** déplace le pointeur au début de la ligne suivante

I. Etape DATA

Export de données : création d'un fichier texte à partir d'une table SAS

Exemple :

La table SAS pays.russet est recopiée dans un fichier texte.

```
FILENAME russet 'c:\Odile Wolber\Cours SAS\Russet.txt';
DATA _NULL_;
  SET pays.russet;
  FILE russet;
  put pays $ @15 Gini @20 Farm…@52 Demo;
RUN;
```

ou de manière équivalente :

```
DATA _NULL_;
  SET pays.russet;
  FILE 'c:\Odile Wolber\Cours SAS\Russet.txt';
  put pays $ @15 Gini @20 Farm…@52 Demo;
RUN;
```


I. Etape DATA

Manipulation des données au cours d'une étape DATA

- création de variables

Exemple : à partir des variables du fichier pays.russet, on construit les variables suivantes : ln(RENT+1), ln(FARM), Exp(INST-16.3). La variable DEMO étant qualitative, on la remplace par les trois variables indicatrices des états poltiques : DEMOSTAB, DEMOINST et DICTATUR :

```
DATA pays.russet;
   SET pays.russet;
   LRENT=ln(RENT+1);
   LFARM=ln(FARM);
   EINST=Exp(INST-16.3);
   if DEMO=1 then DEMOSTAB=1;
   else DEMOSTAB=0;
   …;
RUN;
```


I. Etape DATA

Manipulation des données au cours d'une étape DATA

- conversion d'une variable numérique en une variable caractère

Exemple : la variable DEMO étant qualitative, on veut la remplacer par une variable caractère CDEMO. On utilise l'instruction length :

```
DATA pays.russet;
   SET pays.russet;
   LENGTH CDEMO $ 1;
   CDEMO=DEMO;
RUN;
```


I. Etape DATA

Manipulation des données au cours d'une étape DATA

- **Sélection de certaines observations : instructions IF et WHERE**

Syntaxe :

```
IF condition THEN OUTPUT;
```

ou

```
IF condition;
```

L'instruction IF filtre les observations en ne prenant en compte que celles qui vérifient la condition spécifiée.

Exemple :

Dans le fichier pays.russet, on sélectionne uniquement les démocraties :

```
DATA demo;
     SET pays.russet;
     IF demo=1 or demo=2;
RUN;
```

I. Etape DATA

Manipulation des données au cours d'une étape DATA

De manière équivalente, on peut utiliser l'instruction WHERE :

```
DATA demo;
    SET pays.russet;
    WHERE demo=1 or demo=2;
RUN;
```

- **Suppression de certaines observations : instruction DELETE**

Syntaxe :

```
IF condition THEN DELETE;
```


I. Etape DATA

Manipulation des données au cours d'une étape DATA

- Sélection de certaines variables : instructions DROP, KEEP et RENAME

Ces mots-clés peuvent être utilisés :

- comme option de l'instruction **DATA** ou de l'instruction **SET**
- comme instruction à part entière

I. Etape DATA

Manipulation des données au cours d'une étape DATA

- Ne stocker qu'une partie des variables utilisées dans l'étape DATA : instruction KEEP

3 syntaxes possibles :

```
DATA nom_de_table(KEEP=liste_de_variables) ;
ou
DATA nom_de_table ;
SET nom_de_table(KEEP=liste_de_variables) ;
ou
DATA nom_de_table ;
SET nom_de_table ;
…
KEEP liste_de_variables;
```


I. Etape DATA

Manipulation des données au cours d'une étape DATA

- Ne pas stocker des variables créées au cours de l'étape DATA : instruction DROP

3 syntaxes possibles :

```
DATA nom_de_table(DROP=liste_de_variables) ;
ou
DATA nom_de_table ;
SET nom_de_table(DROP=liste_de_variables) ;
ou
DATA nom_de_table ;
SET nom_de_table ;
…
DROP liste_de_variables;
```


I. Etape DATA

Manipulation des données au cours d'une étape DATA

- Renommer une ou plusieurs variables : instruction RENAME

3 syntaxes possibles :

```
DATA nom_de_table(RENAME=(ancien_nom=nouveau_nom)) ;
ou
DATA nom_de_table ;
SET nom_de_table(RENAME=(ancien_nom=nouveau_nom)) ;
ou
DATA nom_de_table ;
SET nom_de_table ;
…
RENAME ancien_nom1=nouveau_nom1…ancien_nomn=nouveau_nomn;
```


I. Etape DATA

- Concaténation et chargement d'une table : instruction SET

Syntaxe :

SET <nom_de_table1>…<nom_de_tablen;

La table résultante contiendra toutes les variables provenant des tables en entrée.

- Utilisation des variables FIRST. et LAST. :

Après tri d'une table par une procédure SORT, l'appel de cette table par une instruction :

SET nom_de_table;BY liste_de_variables;

génère automatiquement, pour chaque observation, une variable FIRST.variable_clé mise à 1 s'il y a eu changement de clé depuis l'observation précédente, et une variable LAST.variable_clé mise à 1 s'il y a changement de valeur, pour la variable considérée, entre l'observation courante et l'observation suivante.

I. Etape DATA

- Utilisation des variables FIRST. et LAST. :

Exemple :

On a un fichier d'enquêtés, chaque individu étant rattaché à un ménage (repéré par la variable MENAGE). Pour chaque individu, on a collecté des informations aux 2 niveaux : ménage et individu. En outre, on possède ces informations pour deux années (repérées par la variable AN).
On ne souhaite conserver, pour chaque année, que les informations au niveau du ménage.

Syntaxe :

```
proc sort data=niv_ind;
      by menage an;
run;

data niv_men;
      set niv_ind;
      by menage an;
      if first.an;
run;
```


I. Etape DATA

- Utilisation de l'instruction RETAIN :

L'utilisation la plus courante consiste à créer à partir de l'instruction RETAIN une variable compteur.

Exemple :

On a un fichier contenant les caractéristiques des logements successifs occupés par des ménages (repérés par la variable MENAGE). On veut compter le nombre de logements successifs occupés par les ménages :

Syntaxe :

```
proc sort data=logement;
      by menage;
run;

data nb_log;
      set logement;
      by menage;
      retain nlog 0;
      if first.menage then nlog=0;
      nlog=nlog+1;
      if last.menage;
run;
```


I. Etape DATA

- Appariement de tables SAS : instruction MERGE

Syntaxe :

```
MERGE nom_de_table1 <IN=nom_de_variable>;
          …
          nom_de_tablen <IN=nom_de_variable>;
BY liste_de_variables;
```

Les fichiers doivent être triés, dans le même ordre, selon la liste de variables énoncés dans l'instruction BY.

Exemple :

On dispose de deux tables SAS, sur 10 zones d'un parc national, relatives aux caractéristiques physiques de ces zones (table parc1) et à la densité au km² de neuf herbivores dans ces 10 zones (table parc2). On veut fusionner ces deux tables.

I. Etape DATA

- Appariement de tables SAS : instruction MERGE

```
proc sort data=parc1;
      by Zone;
run;

proc sort data=parc2;
      by Zone;
run;

data parc;
      merge parc1 parc2;
      by Zone;
run;
```

L'option IN= n'est pas indispensable. Si par exemple, on ne veut sélectionner que les observations de la table1, les instructions seront les suivantes :

```
MERGE nom_de_table1 <IN=A>;
          …
          nom_de_tablen <IN=nom_de_variable>;
BY liste_de_variables;
IF A;
```


II. Etape PROC

- Une procédure de traitement des données est annoncée par le mot clé PROC suivi du nom de la procédure.
- Un certain nombre d'options et d'instructions permettent de définir le nom de la table en entrée, les variables à analyser, si les résultats sont stockés dans une autre table…
- Si aucun nom de table n'est indiqué après PROC et le nom de la procédure, SAS traite la dernière table créée.

II. Etape PROC

Imprimer les observations d'une table SAS : PROC PRINT

Cette procédure est utilisée pour imprimer tout ou une partie des valeurs d'une table SAS. Elle permet également d'imprimer des totaux et sous totaux pour des variables numériques.

Syntaxe :

```
PROC PRINT DATA=nom_de_table <liste d'options>;
  VAR liste de variables;
  ID liste de variables;
  BY liste de variables;
  PAGEBY variable;
  SUMBY variable;
  SUM liste de variables;
RUN;
```

II. Etape PROC

Imprimer les observations d'une table SAS : PROC PRINT

Exemple :

On veut imprimer le nombre de tués lors de manifestations violentes en fonction de l'instabilité politique :

```
proc sort data=pays.russet;
 by demo;
run;

proc print;
 var pays deat;
 by demo;
 sum deat;
run;
```


II. Etape PROC

```
---------- Demo=1 ----------

 Obs    Pays           Deat

   1    Australie         0
   2    Belgique          1
  ...
  12    Royaume-Uni       0
  13    Suède             0
  14    Suisse            0
  15    Uruguay           1
 ----                    ----
 Demo                     16

---------- Demo=2 ----------
```


II. Etape PROC

Trier une table SAS selon une clé : PROC SORT

Cette procédure est utilisée pour trier les observations d'une table SAS selon une ou plusieurs variables. Les observations, une fois triées, peuvent être stockées dans une nouvelle table (instruction OUT=).

Syntaxe :

```
PROC SORT DATA=nom_de_table OUT=nom_de_table;
  BY <DESCENDING> variable1…<DESCENDING> variablen;
RUN;
```

OUT= Table SAS en sortie. Si cette option est absente, la table d'origine est remplacée par la table triée.

<DESCENDING> Si ce mot-clé est précisé, les observations sont triées par ordre décroissant de la variable indiquée après DESCENDING

II. Etape PROC

Afficher des comptages et des pourcentages : PROC FREQ

Cette procédure produit des tableaux de fréquences unidimensionnelles ou des tris croisés multidimensionnels. Elle permet également de faire certains tests statistiques tels que le test du chi2.

Syntaxe :

```
PROC FREQ data=nom_de_table <liste d'options>;
  BY liste_de_variables;
  TABLES requête </liste d'options>;
  WEIGHT variable;
RUN;
```

Tables permet de définir la liste des distributions souhaitées. Les tableaux croisés sont obtenus en séparant les variables par des astérisques.

II. Etape PROC

Afficher des comptages et des pourcentages : PROC FREQ

Exemple :

Dans l'étude de la relation herbivores-environnement, on veut croiser la densité au km² des éléphants et la proximité de l'eau :

```
proc freq data=parc;
 tables A*EAU;
run;
```

Dans le cas de demande de plusieurs distributions simultanées, il suffit de séparer chaque distribution par un espace. Par exemple, TABLES a b correspond à la demande la distribution simple de la variable a, puis de celle de la variable b.

On peut stocker les résultats de la procédure dans une table SAS :

```
TABLES requête / OUT=nom_de_table;
```

La table résultante comprend, outre les variables d'origine, l'effectif (variable COUNT) et le pourcentage par rapport à l'ensemble des observations (PERCENT)

II. Etape PROC

Calculer des statistiques simples :

PROC MEANS

Cette procédure permet de calculer des statistiques simples telles que la moyenne, la variance, le minimum, le maximum ou l'étendue, sur une population ou sur des sous-populations.

Syntaxe :

```
PROC MEANS data=nom_de_table <liste d'options>;
        CLASS liste de variables;
        VAR liste de variables numériques;
        BY liste de variables;
        FREQ variable;
        WEIGHT variable;
        ID variable;
        OUTPUT OUT=nom_de_table;
RUN;
```

Seule l'instruction VAR est obligatoire.

II. Etape PROC

Calculer des statistiques simples : PROC MEANS

Comparaison des instructions CLASS et BY : l'instruction CLASS permet d'obtenir des sorties plus lisibles lorsque les sous-populations étudiées sont constituées à partir du croisement de deux variables ou plus. L'instruction CLASS permet également d'afficher les effectifs de chaque sous-population.

VAR= variables numériques pour lesquelles on sollicite la procédure
WEIGHT variable de pondération
OUTPUT permet de stocker les résultats dans une table
OUTPUT OUT=nom_de_table <statistiques à conserver>;
ID la valeur maximum de la variable mentionnée, au niveau de chaque groupe défini par l'instruction CLASS, sera conservée dans la table de sortie.

II. Etape PROC

Calculer des statistiques simples : PROC MEANS

Par défaut, PROC MEANS calcule les statistiques suivantes:
N nombre d'observations sur lesquelles sont basés les calculs
MEAN moyenne empirique
STD écart-type empirique
MIN valeur minimale
MAX valeur maximale

PROC MEANS peut également calculer d'autres statistiques, parmi lesquelles :
KURTOSIS coefficient de kurtosis
SKEWNESS coefficient de symétrie
SUM somme
T valeur de Student
VAR variance

II. Etape PROC

Calculer des statistiques simples : PROC MEANS

Tous les résultats issus de PROC MEANS peuvent être sauvegardés dans une table. Alors qu'à l'écran ne sont affichées que les statistiques au niveau le plus fin, toutes les statistiques aux niveaux agrégés sont conservées dans la table de sortie.

Syntaxe :

OUTPUT OUT=nom_de_table <statistiques_à_conserver>;

Trois formes de demandes statistiques

statistique= calcul des statistiques pour les variables précisées dans VAR= et donne les mêmes noms aux variables en sortie que les variables en entrée. Une seule statistique peut être sauvegardée par ce biais.

statistique=nom1…nomn calcul des statistiques pour les variables de VAR mais les stocke sous un nom différent. Les noms de la liste sont attribués dans l'ordre des variables de VAR.

statistique(var1…varn)=nom1…nomn applique les statistiques à un sous-ensemble des variables de VAR.

II. Etape PROC

Calculer des statistiques simples : PROC MEANS

Exemple :

Sur le fichier pays.russet, on calcule la moyenne et la variance des variables gini, farm et gnpr, en distinguant les trois groupes de pays (démocratie stable, démocratie instable et dictature). On stocke les résultats dans la table temporaire m_russet. Aucun résultat n'est imprimé dans la fenêtre OUTPUT.

```
proc means data=pays.russet mean var noprint;
 class demo;
 var gini farm gnpr;
 output out=m_russet
        mean(gini farm gnpr)=m_gini m_farm m_gnpr
        var(gini farm gnpr)=v_gini v_farm v_gnpr;
run;
```


II. Etape PROC

Calculer des statistiques simples : PROC MEANS

Contenu de la table m_russet :

Obs	Demo	_TYPE_	_FREQ_	m_gini	m_farm	m_gnpr	v_gini	v_farm	v_gnpr
1	.	0	47	71.3681	92.9319	559.26	206.817	45.0018	239904.11
2	1	1	15	63.8867	88.7600	1055.00	171.370	35.6183	272558.57
3	2	1	11	74.0636	94.2545	490.27	211.895	42.7907	88473.42
4	3	1	21	75.3000	95.2190	241.29	187.566	37.6676	23652.51

II. Etape PROC

Calculer des coefficients de corrélation : PROC CORR

PROC CORR permet le calcul des coefficients de corrélation entre les variabes (de PEARSON, de SPEARMAN, de KENDALL ou de HOEFFDING) et peut produire des matrices de produits croisés ainsi que des matrices de variance-covariance.

On peut aussi calculer des statistiques univariées simples et crée de nouvelles tables contenant ces statistiques univariées et les corrélations.

Syntaxe :

PROC CORR data=nom_de_table <liste d'options>;
BY liste de variables;
FREQ variable;
PARTIAL variable;
VAR liste de variables;
WEIGHT variable;
WITH liste de variables;
RUN;

II. Etape PROC

Calculer des coefficients de corrélation : PROC CORR

BY la procédure est exécutée pour chaque sous groupe.

VAR variables pour lesquelles les coefficients sont calculés.

WITH utilisé pour obtenir les corrélations uniquement pour certaines combinaisons de variables. Indiquer dans l'instruction WITH les variables dont on veut les corrélations avec les variables de VAR.

WEIGHT variable de pondération. A utiliser uniquement pour le calcul du coefficient de Pearson.

FREQ spécifie la variable numérique dont la valeur représente la fréquence de l'observation.

II. Etape PROC

Calculer des coefficients de corrélation : PROC CORR

Exemple 1 :

Sur le fichier pays.russet, on calcule la corrélation entre les variables gini farm rent. Les résultats sont stockés dans le fichier c_russet1 :

```
proc corr data=pays.russet outp=c_russet1;
 var gini farm rent;
run;
```

Obs	_TYPE_	_NAME_	Gini	Farm	Rent
1	MEAN		71.3681	92.9319	21.7477
2	STD		14.3811	6.7083	17.8461
3	N		47.0000	47.0000	44.0000
4	CORR	Gini	1.0000	0.9379	0.1556
5	CORR	Farm	0.9379	1.0000	0.1982
6	CORR	Rent	0.1556	0.1982	1.0000

II. Etape PROC

Calculer des coefficients de corrélation : PROC CORR

Exemple 2 :

Sur le fichier pays.russet, on calcule la corrélation des variables gini farm rent avec les variables gnpr et labo. Les résultats sont stockés dans le fichier c_russet2 :

```
proc corr data=pays.russet outp=c_russet2;
 var gini farm rent;
 with gnpr labo;
run;
```

Obs	_TYPE_	_NAME_	Gini	Farm	Rent
1	MEAN		71.3681	92.9319	21.7477
2	STD		14.3811	6.7083	17.8461
3	N		47.0000	47.0000	44.0000
4	CORR	Gnpr	-0.2997	-0.3483	-0.0591
5	CORR	Labo	-0.2438	0.2989	-0.0473

II. Etape PROC

Les procédures de gestion des données

PROC CONTENTS affiche la description d'une table : nombre d'observations, nombre de variables, liste et position des variables.

Syntaxe :

```
PROC CONTENTS data=nom_de_table;
     <OUT=nom_de_table>;
RUN;
```

OUT stocke le résultat de la demande dans une table SAS.

II. Etape PROC

Les procédures de gestion des données

PROC DATASETS permet d'effectuer des opérations globales sur les tables.

Syntaxe :

```
PROC DATASETS library=nom_de_librairie;
     <DELETE>;
RUN;
```

Les principales opérations couvertes par cette procédure sont :

CONTENTS liste la description d'une table (analogue à la PROC CONTENTS)
CHANGE ou AGE renomme une table
COPY copie ou déplace une ensemble de tables d'une librairie à une autre
DELETE détruit un ensemble de tables
APPEND concatène deux tables

II. Etape PROC

Notion de format

Un format permet de regrouper plusieurs modalités sous un même libellé. On distingue des formats caractères, appliqués à des variables caractères, et des formats numériques, appliqués à des variables numériques.

Deux étapes :
- création et gestion de formats : PROC FORMAT
- utilisation de formats : instructions de FORMAT existants dans de nombreuses procédures.

Syntaxe :

```
PROC FORMAT DATA=nom_de_table <liste d'options>;
          VALUE nom_de_format
          Liste_de_valeurs='valeur1 formatée'
          …
          Liste_de_valeurs='valeurn formatée'
          ;
RUN;
```

II. Etape PROC

Notion de format

Chaque liste de valeurs peut être composée :

- d'une valeur simple
  75='Paris'
- d'une liste continue de valeurs
  92-94='Petite couronne'
- d'une liste exhaustive de valeurs
  92.93.94='Petite couronne'
- d'une liste discontinue de valeurs
  75.77.78.91-95='Région parisienne

Mots clés réservés :

LOW plus petite valeur rencontrée dans la table
HIGH plus grande valeur rencontrée dans la table
OTHER toutes les valeurs non prévues par les autres classes

II. Etape PROC

Notion de format

Exemple 1 :

Regroupement d'âges en classes quinquennales :

```
PROC FORMAT;
   VALUE FQUIN   15-19='15 à 19 ans'
                 20-24='20 à 24 ans'
                 25-29='25 à 29 ans'
                 30-34='30 à 34 ans'
                 35-39='35 à 39 ans'
                 40-HIGH='40 ans et plus'
   ;
RUN;

PROC FREQ DATA=fic_age;
   TABLES age;
   FORMAT age FQUIN.;
RUN;
```


II. Etape PROC

Notion de format

Exemple 2 :

Regroupement des départements en région :

```
PROC FORMAT;
   VALUE $ REGION
   'Paris','Hauts-de-Seine','Seine-Saint-Denis','Val de
Marne','Yvelines',
   'Val d'Oise','Seine et Marne'='Ile-de-France'
   'Ardennes','Aube','Marne','Haute-Marne'='Champagne-Ardenne'
   …
   ;
RUN;

PROC FREQ DATA=fic_dpt;
   TABLES dpt;
   FORMAT dpt  $REGION.;
RUN;
```


II. Etape PROC

Réduction des variables quantitatives

PROC STANDARD permet de réduire ou « standardiser » les variables quantitatives

Syntaxe :

```
PROC STANDARD <liste d'options> ;
BY <descending> liste_de_variables ;
VAR liste_de_variables ;
      WEIGHT variables ;
```

BY : suivi du nom d'une variable qualitative indique que les statistiques sont calculées par groupe d'observations ; la table doit être préalablement triée.

VAR : variables standardisées et recopiées dans la table de sortie (instruction **OUT=**). Par défaut, toutes les variables numériques sont traitées.

WEIGHT : nom de la variable contenant les pondérations des observations.

II. Etape PROC

Transposition des matrices

PROC TRANSPOSE

Syntaxe :

```
PROC TRANSPOSE<liste d'options> ;
VAR liste_de_variables ;
COPY liste_de_variables ;
BY liste_de_variables ;
```

BY : une observation est créée pour chaque variable transposée et pour chaque groupe.

COPY : variables recopiées dans la table de sortie sans transposition.

VAR : variables transposées et recopiées dans la table de sortie ; par défaut, toutes les variables numériques sont traitées.

Options principales :

OUT = TABLE SAS spécifie le nom de la tables créée

NAME = spécifie le nom de la variable créée qui contiendra les noms des anciennes variables pour désigner les observations (par défaut _name_).

II. Etape PROC

Transposition des matrices

Exemple :

```
proc transpose data=DataOW.pays out=pays name=variable;
 var Gini Ecks Farm;
run;
```

Obs	variable	COL1	COL2	COL3 …	COL45	COL46	COL47
1	Gini	86.3	92.9	74.0	81.7	90.0	43.7
2	Ecks	57.0	0.0	4.0	1.0	36.0	9.0
3	Farm	98.2	99.6	97.4	96.6	99.3	79.8

III. Le langage matriciel

SAS IML : module spécialisé interprétant un langage de calcul matriciel.

L'objet de base de manipulation du langage est une matrice, un tableau bidimensionnel (nrow X ncolumn) de valeurs numériques ou de caractères. Un table SAS peut être lue dans une matrice ou, inversement, créée à partir d'une matrice.

SAS IML commence par l'instruction **PROC IML** et se termine par **QUIT.**

III. Le langage matriciel

Lire les colonnes d'une table SAS et les mettre en matrice

Syntaxe :

Pour copier les variables var1 var2 … varn dans la matrice X :

```
proc iml ;
     use nom_de_table ;
READ ALL VAR {var1 var2 … varn} INTO X;
QUIT;
```

Pour copier toutes les variables dans la matrice X :

```
proc iml ;
use table SAS ;
READ ALL VAR _ALL_ INTO X;
QUIT;
```


III. Le langage matriciel

Créer une table SAS à partir d'une matrice

Syntaxe :

Pour copier la matrice X dans une table SAS :

```
proc iml ;
… ;
CREATE nom_de_table FROM matrice_X ;
APPEND FROM matrice_X ;
CLOSE nom_de_table ;
quit;
```


III. Le langage matriciel

Définir un vecteur colonne ou une matrice

Pour définir un vecteur colonne au cours d'une procédure IML :

```
X={1 2 3} ;
```

Pour définir une matrice 2X3 :

```
A = {2 1 9, 1 3 4} ;
ou :
A = {2 1 9,
     1 3 4}
```

Pour transposer une matrice X :

```
Y=X` ;
ou :
Y=T(X) ;
```


III. Le langage matriciel

Les fonctions matricielles

La fonction **j(n1,n2,val)** peut être utilisée pour créer une matrice de n1 lignes et n2 colonnes dont tous les éléments prennent la valeur val :

```
A=j(1,5,1)
```

crée un vecteur ligne de 5 colonnes prenant la valeur 1.

Pour créer la matrice identité, on utilise la fonction **I** :

```
Ident=I(4)
```

crée une matrice identité 4X4.

La fonction **DIAG** permet de créer une matrice diagonale dont les éléments sont les termes diagonaux de la matrice argument.

Exemple :

```
proc iml;
  X={4 9 6, 2 5 7, 1 9 7};      =>
  D=diag(X);
  print D;
quit;
```

D

4	0	0
0	5	0
0	0	7

III. Le langage matriciel

Les fonctions matricielles

La fonction **VECDIAG** permet de créer un vecteur dont les éléments sont les éléments diagonaux de la matrice argument.

Le calcul des vecteurs et des valeurs propres d'une matrice s'effectue à partir des fonctions EIGVEC et EIGVAL.

La fonction EIGVAL retourne le vecteur colonne des valeurs propres, triées dans leur ordre décroissant.
La fonction EIGVEC retourne la matrice des vecteurs propres.

III. Le langage matriciel

Les fonctions matricielles

La fonction **TRACE** retourne la trace de la matrice.

La fonction **DET** calcule le déterminant d'une matrice.

L'inverse d'une matrice peut être utilisée à partir de la fonction **INV** :

Y=**INV**(X)

Le calcul d'une puissance α d'une matrice s'effectue à partir de l'opérateur ** :

Y=X**α

III. Le langage matriciel

Les résultats obtenus par la fonction **INV** ne sont pas toujours très stables. Ils ne le sont pas non plus lorsque l'on calcule une puissance négative d'une matrice.

C'est pourquoi pour calculer X^{α} (α négatif), il est préférable de procéder selon les étapes décrites ci-après :

III. Le langage matriciel

1. Calcul de L, matrice diagonale des valeurs propres de X :

```
PROC IML ;
  USE nom_de_table1 ;
  READ ALL VAR {var1 var2 var3} INTO X ;
  L=DIAG(EIGVAL(X)) ;
  CREATE nom_de_table2 FROM L ;
  APPEND FROM L ;
  CLOSE L ;
QUIT ;
```


III. Le langage matriciel

2. Calcul de L^{α} :

```
DATA nom_de_table2 ;
  SET nom_de_table2(KEEP=COL1-COL3) ;
  COL1=COL1**α ;
  COL2=COL2**α ;
  COL3=COL3**α ;
RUN ;
```


III. Le langage matriciel

3. Calcul de U, matrice des vecteurs propres de X, puis de $X^{\alpha} = U L^{\alpha} U'$:

```
PROC IML;
  USE nom_de_table2;
  READ ALL VAR _ALL_ INTO Lα;
  USE nom_de_table1;
  READ ALL VAR {var1 var2 var3} INTO X;
  U=EIGVEC(X);
  Xα=U*Lα*U` ;
QUIT;
```


IV. SAS SQL

Le **SQL** (Structured Query Language) est le langage universel de gestion des données informatiques des SGBD relationnels.

Extraction de données

Syntaxe :

```
PROC SQL ;
  CREATE TABLE nom_de_table (
  variable1 type1(long1) <LABEL='label1' FORMAT=fmt1
    INFORMAT=inf1>,
  variable2 type2(long2) <LABEL='label2' FORMAT=fmt1
    INFORMAT=inf1>,
  variable3 type3(long3) <LABEL='label3' FORMAT=fmt1
    INFORMAT=inf1>,
  );
QUIT ;
```

Exemple : créer la table PARC avec les variables :
- Zone (ZONE), entier sur 2 caractères
- Hauteur de l'herbe (HERB), caractère (longueur 6)
- Présence de bosquets (BOSQUET), caractère (longueur 3)
- Accès à l'eau (EAU), caractère (longueur 8)
- Présence d'hippopotames (HIPPO), caractère (longueur 8)

```
proc sql;
 create table parc(
 ZONE integer label='zone' format=2.,
 HERB char(6) label='hauteur de l"herbe',
 BOSQUET char(3) label='présence de bosquets',
 EAU char(8) label='accès à l"eau',
 HIPPO char(8) label='présence d"hippopotames');
quit;
```


La commande SELECT

La commande **SELECT** permet de rechercher des informations par une sélection :

Syntaxe :

SELECT variable <,variable>…
FROM nom_de_table ou nom_de_vue <,nom_de_table ou nom_de_vue>…
<**WHERE** expression>
<**GROUP BY** variable <,variable…>…>
<**HAVING** expression>
<**ORDER BY** variable <,variable>…>;

Exemple :

```
proc sql;
 select ZONE, HERB
 from parc3
 where ZONE>6;
quit;
```

Le résultat d'une requête SELECT est par défaut affiché dans l'OUTPUT

ZONE	HERB
7	courte
8	courte
9	courte
10	courte
11	courte
12	haute
13	courte

On peut aussi stocker le résultat d'une requête SELECT dans une table SAS :

Exemple : on sélectionne toutes les variables pour les zones 7 et plus.

```
proc sql;
 create table extract
 as select *
 from parc3
 where ZONE>6;
quit;
```


La commande WHERE

La commande WHERE permet de spécifier un critère de sélection.

Exemple : créer une table EXTRACT des zones 7 et plus et dont l'accès à l'eau contient la chaîne de caractère « loig ».

```
proc sql;
   create table extract
   as select *
   from parc3
   where   ZONE>6 and
           EAU like '%loig%';
run;
```

On obtient la table EXTRACT suivante :

Obs	ZONE	HERB	BOSQUET	EAU	HIPPO
1	7	courte	non	éloignée	aucun
2	9	courte	non	éloignée	aucun
3	10	courte	non	éloignée	aucun
4	12	haute	oui	éloignée	aucun

La clause ORDER BY

La clause **ORDER BY** permet de trier le résultat de la requête.

Options suivantes de la proc SQL permettent de restreindre le nombre d'observations traitées :

INOBS=n restreint à n le nombre d'observations de chaque table à incorporer (les n premières).

OUTOBS=n restreint à n le nombre d'observations en sortie (les n dernières).

La clause ORDER BY

Exemple : trier la table parc3 par la variable ZONE (ordre descendant) et ne conserver que les variables ZONE, HIPPO, EAU et la dernière observation pour laquelle il existe un bosquet.

```
proc sql outobs=1;
    select ZONE,HIPPO,EAU
    from parc3
    where BOSQUET='oui'
    order by ZONE desc;
run;
```

Il s'affiche dans la fenêtre OUTPUT :

ZONE	HIPPO	EAU
13	Bas CNAM	proche

Supprimer les doublons : option DISTINCT

Elimine les doublons sur les variables HERB, BOSQUET et EAU.

Exemple :

```
proc sql;
    select distinct HERB, BOSQUET, EAU
    from parc3
    order by HERB desc;
run;
```

Dans la fenêtre OUTPUT s'affiche le résultat suivant :

HERB	BOSQUET	EAU
haute	oui	éloignée
courte	non	éloignée
courte	oui	moyenne
courte	oui	proche

Regrouper les modalités d'une variable : la clause GROUP BY

La clause **GROUP BY** permet de regrouper les modalités d'une variable. Cette clause est utilisée uniquement lorsque la requête utilise une fonction somme.

SELECT fonction (variable)
FROM table
GROUP BY variable1, variable2…

L'expression HAVING permet d'opérer sur un sous-ensemble d'observations de la table. C'est l'équivalent du WHERE appliqué au groupe. La clause GROUP BY doit toujours précéder une expression HAVING et définit la groupe sur lequel l'expression HAVING va opérer. On utilise l'expression HAVING lorsque la requête utilise une fonction de groupe (AVG, COUNT…).

Exemple :

```
proc sql;
    select mean(Gini) as Mgini
    from pays.russet
    group by Demo
    having Mgini>70;
quit;
```

Regroupement par type de régime (Demo) et calcul de la moyenne des coefficients de gini. Sélection des observations pour lesquelles la moyenne (Mgini) est supérieure à 70.

Les fonctions

Ces fonctions sont les suivantes :
- MEAN, AVG : moyenne
- COUNT, N : compte le nombre de valeurs non manquantes
- CV : coefficient de variation
- MAX : donne la plus grande valeur d'une variable
- MIN : donne la plus petite valeur d'une variable
- SUM : somme
- VAR : variance

Syntaxe :

SELECT fonction1(variable) <**AS** nomvar1 **LABEL**='label'>,
fonction2(variable) <**AS** nomvar2 **LABEL**='label'>,
FROM table
GROUP BY variable

AS permet de donner un nom à la variable résultat lorsque l'on crée une table en sortie (CREATE TABLE).
LABEL permet d'associer un label à la colonne.

Exemple : calculer la moyenne du coefficient de gini, les coefficients minimal et maximal de gini par type de régime pour la table pays.russet.

```
proc sql;
  select Demo,
  avg(Gini) LABEL='Moyenne des coefficients de Gini',
  min(Gini) LABEL='Coefficient minimal',
  max(Gini) LABEL='Coefficient maximal'
  from pays.russet
  group by Demo;
quit;
```

On obtient la sortie suivante :

Demo	Moyenne des coefficients de Gini	Coefficient minimal	Coefficient maximal
1	63.88667	45.8	92.9
2	74.06364	47	93.8
3	75.3	43.7	93.8

L'expression when

Cette expression permet de changer la représentation des données en donnant un code chiffré à une chaîne de caractères ou inversement, en donnant à un code un intitulé plus significatif.

Exemple : on recode en binaire la variable HIPPO : 1 si présence d'hippopotames, 0 sinon.

```
proc sql;
   select *,
   case when HIPPO='aucun' THEN 0
   else 1
   end
   as BIN_HIP
   from parc3;
run;
```

On obtient la sortie suivante :

ZONE	HERB	BOSQUET	EAU	HIPPO	BIN_HIP
1	courte	oui	proche	moyen	1
2	haute	oui	éloignée	aucun	0
3	courte	oui	proche	aucun	0
4	courte	oui	proche	importan	1

Les jointures

1/ On sélectionne uniquement les observations communes.

```
PROC SQL ;
    SELECT a.*, b.*
    FROM  table1 AS a,
          table2 AS b
    WHERE a.var1=b.var1;
QUIT;
```

Cette jointure correspond au cas IF INA AND INB du merge en SAS de base.

Les jointures

2/ Jointure à gauche

```
PROC SQL ;
    SELECT a.*, b.*
    FROM  table1 AS a
          LEFT JOIN
          table2 AS b
    ON a.var1=b.var1;
QUIT;
```

Elle correspond au cas IF INA du merge.

Les jointures

3/ Jointure à droite :

```
PROC SQL ;
    SELECT a.*, b.*
    FROM  table1 AS a
          RIGHT JOIN
          table2 AS b
    ON a.var1=b.var1;
QUIT;
```

Elle correspond au cas IF INB du merge.

Les jointures

4/ Jointure « full »

```
PROC SQL ;
    SELECT a.*, b.*
    FROM  table1 AS a
          FULL JOIN
          table2 AS b
    ON a.var1=b.var1;
QUIT;
```

Elle correspond au *merge by* sans condition.

Les jointures

5/ Produit cartésien

```
PROC SQL ;
    SELECT a.*, b.*
    FROM  table1 AS a
          table2 AS b;
QUIT;
```

On fusionne chaque observation de la table A avec chaque observation de la table B. Si A contient *na* observations et B *nb* observations, le résultat contient *na*X*nb* lignes.

Les jointures

6/ Jointure d'une table avec elle-même

Il est possible de faire une jointure d'une table avec elle-même pour obtenir davantage d'informations.

Exemple : Nous avons la table EMPLOI avec les variables :

- Empnum (numéro de l'employé)
- Empnom (nom de l'employé)
- Emptitre (titre de l'employé)
- Numsup (numéro du supérieur)

EMPNUM	EMPNOM	EMPTITRE	NUMSUP
101	John	head man.	.
201	Betty	manager	101
213	Joe	respvent	201
214	Jeff	respvent	201
401	Sam	manager	101
412	Nick	respvent	401

Construire une table avec pour chaque employé le nom de son supérieur hiérarchique (nomsup) et son titre (titresup).


```
proc sql;
 select a.*,
 b.EMPNOM as SUPNOM,
 b.EMPTITRE as SUPTITRE
 from emploi as a,
      emploi as b
 where a.NUMSUP=b.EMPNUM;
quit;
```

EMPNUM	EMPNOM	EMPTITRE	NUMSUP	SUPNOM	SUPTITRE
101	John	head man.	.		
201	Betty	manager	101	John	head man.
213	Joe	respvent	201	Betty	manager
214	Jeff	respvent	201	Betty	manager
401	Sam	manager	101	John	head man.
412	Nick	respvent	401	Sam	manager

L'union

L'opérateur UNION est un opérateur ensembliste. Il effectue l'union des résultats de deux requêtes SELECT, c'est à dire qu'à partir de deux tables résultats, il en crée une troisième comportant l'ensemble des observations des deux tables de départ, en éliminant les doublons parfaits et en triant selon l'ordre des variables dans la commande select.

Exemple :

```
PROC SQL ;
     CREATE TABLE table3 AS
     SELECT *
     FROM table1
     UNION
     SELECT *
     FROM table2 ;
QUIT ;
```

Cette étape correspond dans ce cas à :

```
DATA table3 ;
     SET table1 table2 ;
RUN;
PROC SORT DATA=table3 noduplicates;
     BY _ALL_;
RUN;
```


La commande INSERT INTO

Cette commande est utilisée pour insérer des observations dans une table. Elle nécessite le nom de la table, le nom des variables à initialiser et la valeur à leur assigner.

INSERT INTO table

SET variable1=valeur1, variable2=valeur2…

Exemple : insérer une observation dans la table parc

```
proc sql;
    insert into parc3
    set ZONE=14, HERB='haute', BOSQUET='non',
    EAU='éloignée', HIPPO='moyen' ;
quit;
```


La commande INSERT INTO

Associée à une clause **SELECT**, cette commande permet de copier un sous-ensemble d'une table dans une autre. La table cible doit exister préalablement, et les types de données doivent correspondre de façon très précise.

Exemple : insérer des observations de la table parc4 (zone 14 à 20) dans la table parc3

```
proc sql;
    insert into parc3
    select *
    from parc4
    where ZONE>13;
quit;
```


La commande INSERT INTO

On peut également insérer les valeurs de chaque observation avec l'instruction **VALUES**.

INSERT INTO table

VALUES (valeur1, valeur2, valeur3…)

Exemple : insérer 5 observations dans la table parc3

```
proc sql;
    insert into parc3
    values(15,'haute','oui','proche','moyen')
    values(16,'courte','non','proche','moyen')
    values(17,'haute','oui','proche','aucun')
    values(18,'haute','non','éloignée','moyen')
    values(19,'courte','non','éloignée','importan');
quit;
```

La commande UPDATE

La commande UPDATE spécifie le nom de la table qui va être modifiée, suivi d'une clause SET définissant les modifications à apporter, et d'une clause WHERE avec le critère précisant les observations à modifier.

UPDATE TABLE
SET variable1=valeur1, variable2=valeur2…
WHERE condition

Exemple : remplacer la modalité 'éloignée' de la variable EAU par 'loin' :

```
proc sql;
   update parc3
   set EAU='loin'
   where EAU='éloignée';
quit;
```


La commande DELETE

Cette commande consiste en une clause DELETE FROM spécifiant le nom de la table concernée, et d'une clause WERE avec le critère précisant les observations à supprimer.

Exemple : détruire les observations des zones 14 et plus.

```
proc sql;
   delete
   from parc3
   where ZONE>13;
quit;
```


Gchart : tracé d'histogrammes, de camemberts, de graphiques étoilés

Gplot : graphiques à deux dimensions

Gmap : fonds de cartes

G3d : surfaces ou nuages de points en trois dimensions

Les graphiques sont stockés dans des catalogues graphiques que l'on pourra consulter à nouveau en utilisant la procédure **greplay**.

Tout texte ou figure géométrique complémentaires peuvent être rajoutés sur un graphe en les décrivant dans une table SAS spéciale dite d'annotation : **Annotate data set**.

Instructions générales pour la construction des graphiques

Structure du programme détaillée dans le polycopié de cours :

```
goptions …. ; /*orientation du graphique, bordures, dimensions, couleurs, police de caractère*/
title1…. ; /*édition de titres dans les graphiques*/
titlei…. ;
footnote1… ; /*notes de bas de graphique*/
footnotei… ;
legend1… ; /*légendes*/
legendi… ;
axis1...; /*options sur les axes*/
axisi...;
symbol1...; /*uniquement dans gplot*/
symboli… ;
pattern1… ; /*instructions sur les surfaces*/
patterni… ;
```


Structure du programme (suite) :

```
data annotate ; /*pour créer les instructions qui */
…               /*permettent d'écrire à l'intérieur du graphique*/
run ;

proc gplot data= ….gout=… ;
   plot y*x=n/<options>;
run ;
```

gout : envoi dans un catalogue graphique
y : axe des ordonnées
x : axe des abscisses
n : renvoi à un numéro de symbole

Annotate data set

Une *table d'annotations*, définie lors d'une étape data, est une table SAS contenant les descriptifs d'un ensemble de graphiques qui viendront se superposer aux résultats des procédures précédemment décrites (gchart, gplot,…).

Elle contient ainsi un certain nombre de variables de positions, de descriptions et de texte.

Annotate data set

Les variables de la table d'annotation ont des noms prédéfinis.

Les plus usuelles sont :

- les variables qui définissent les éléments de dessins : **function**.

function indique ce qu'il faut tracer : *bar, draw, frame, pie, symbol, label…*

- les variables de position qui spécifient les coordonnées des points où les tracés doivent être faits et le système de coordonnées :

x abscisse du point

y ordonnée du point

z si 3 dimensions

Annotate data set

- le système de coordonnées pour le tracé (se combine avec size) :

hsys unité à utiliser pour la taille du texte

xsys système de coordonnées pour x

ysys système de coordonnées pour y

zsys système de coordonnées pour z

Ces variables dont les noms sont prédéfinis sont des variables caractères ; leurs valeurs doivent être rentrées comme telles.

- les variables d'attribut :

color couleur du texte

style police de caractère

text contient le texte à écrire ou le nom de la variable du fichier qui contient les différents textes comme autant de valeurs

size détermine la hauteur du texte

position d'un texte par rapport au point (calé à gauche, centré…)

line type de ligne (par défaut, continue)

Exemple :

Représentation des resultants d'une analyse des correspondances des données de parc3 à partir d'un fichier de sortie de **proc corresp**.

```
data anno;
  set corr1(keep=_TYPE_ _NAME_ dim1 dim2);
  x=dim1; y=dim2;
  xsys='2'; ysys='2' ;
  if _TYPE_='OBS' then do;
        text=_NAME_;
        size=0.7;
        color='b1';
        style='hwdmx010';
  output; end;
  if _TYPE_='VAR' then do;
        text=_NAME_;
        size=0.7;
        color='b1';
        style='hwdmx011';
  output; end;
  label x='dimension 1' y='dimension 2';
  keep x y text xsys ysys size color style ;
run;
```


Annotate data set

Obs	x	y	xsys	ysys	text	size	color	style
1	-0.15047	-0.00000	2	2	1	0.7	b1	hwdmx010
2	0.82759	0.00000	2	2	2	0.7	b1	hwdmx010
3	0.93777	0.00000	2	2	3	0.7	b1	hwdmx010
…	…	…	…	…	…	…	…	…
14	-0.28133	0.00000	2	2	HERB	0.7	b1	hwdmx011
15	0.89370	0.00000	2	2	BOSQUET	0.7	b1	hwdmx011
16	-0.55856	0.96462	2	2	EAU	0.7	b1	hwdmx011
17	-0.55856	-0.32154	2	2	HIPPO	0.7	b1	hwdmx011

Dans les versions antérieures de SAS, l'enrichissement des graphiques était difficile : tout devait être programmé, en particulier à partir des fichiers *annotate*.

Maintenant, la présentation peut être très soignée en utilisant de façon conviviale l'éditeur graphique de SAS.

Procédure gchart

La procédure proc gchart a de nombreuses options de représentation :

- block barres à 3 dimensions
- vbar histogramme barres verticales
- hbar histogramme barres horizontales
- pie camemberts
- star graphiques étoilés

Syntaxe :

proc gchart <**data**=*fichier des données*>
<**annotate**=*fichier annotate*>
<**gout**=*catalogue graphique*> ;

Appel de l'option de tracé (exemple barres verticales) :

vbar variables à représenter
</**annotate**=*fichier annotate*>
<*options d'apparence*>
<*options statistiques*>
<*options de classes*>
<*options d'axes*>
<*autres options descriptives*>> ;

<**by** <variable> ;

Procédure gchart

Il est important de spécifier l'option **discrete** pour les variables qui ont été recodées par la procédure proc format, sinon le programme essaiera de créer des classes.

Lorsque la variable principale est une variable numérique continue, on peut laisser SAS créer les classes à représenter ; il faut spécifier alors combien de classes retenir, par l'options **levels=** .

Exemple : on a un fichier d'enquête avec le nombre de volés par âge et par sexe .

Le programme utilisera l'option sumvar= et l'option sum. Dans ce cas, les statistiques ne sont plus imprimées.

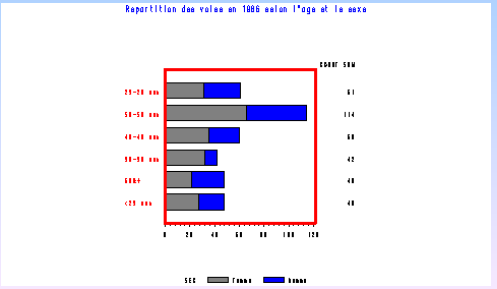
```
title1 f=hwdmx010 h=0.4 cm c=blue
'Repartition des voles en 1986 selon l"age et le sexe' ;
pattern1 c=gray v=s ;
pattern2 c=blue v=s;
axis1 w=3 c=red label=none length=8 cm;
axis2 w=3 label=none length=8 cm;
```

Procédure gchart

```
proc gchart data=chart;
hbar  age/discrete
      sumvar=count
      subgroup=sex
      sum
      frame
      axis=axis2
      maxis=axis1
      space=1
      coutline=bl;
run;
```


Procédure gchart

On obtient alors la représentation suivante :



Procédure gplot

La procédure **gplot** produit des graphiques à deux dimensions dans un système d'axes perpendiculaires haxis et vaxis.

Les points sont définis par leurs coordonnées sur les axes.

Syntaxe :

proc gplot **data**=*<nom du fichier des données>*

gout=*<nom du catalogue de sortie du graphique>*

annotate=*<fichier annotate valable pour tous les graphiques dans cette procédure>* par exemple si on utilise **by**

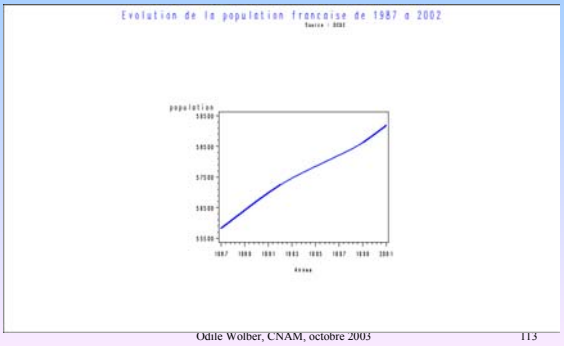
<uniform> *spécifie que les mêmes axes doivent être utilisés pour tous les graphiques de cette procédure* ;

Exemple avec une seule courbe :

```
goptions reset=all rotate=landscape border ftext=hwdmx013
   htext=0.35;
title1 c=blue f=hwdmx020 h=1.4 j=center 'Evolution de la population
   francaise de 1987 a 2002' ;
title2 c=bl f=hwdmx011 h=0.7     '                    Source : OCDE';
symbol1 v=none I=join l=1 c=blue w=2;
axis1  length=8 cm
       label=(h=1 f=hwdmx021 h=0.8)
       value=(f=hwdmx021 h=0.8)
       order=(1987 to 2002 by 2)
       minor=(h=0.2 w=1.0 n=4);
axis2  length=6 cm
       label=(h=1 f=hwdmx021 'population')
       value=(f=hwdmx021 h=0.8)
       order=(55500 to 59500 by 1000)
       major=(h=0.5 w=1.5)
       minor=(h=0.2 w=1.0 n=4);

proc gplot data=population gout=graf1;
plot France*Annee=1 / haxis=axis1 vaxis=axis2;
run;
```


Procédure gplot



Procédure gplot

Exemple de courbes multiples avec option overlay :

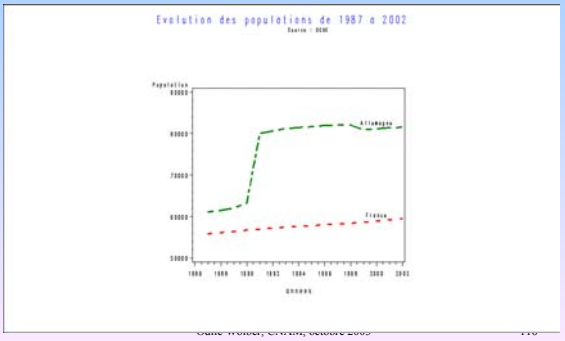
```
goptions reset=all rotate=landscape border ftext=hwdmx013
   htext=0.35 cm;
title2 c=blue f=hwdmx020 h=1.4 j=center 'Evolution des
   populations de 1987 a 2002';
title3 c=bl f=hwdmx011 h=0.7 '                    Source : OCDE';
symbol1 v=none i=j l=1 c=blue w=2;
symbol2 v=none i=j l=8 c=green w=2;
symbol3 v=none i=j l=34 c=red w=2;
axis1  length=10 cm
       label=(h=1 f=hwdmx021 'annees')
       value=(f=hwdmx021 h=0.8);
axis2  length=8 cm
       label=(h=0.8 f=hwdmx021 'Population')
       value=(f=hwdmx021 h=0.8)
       major=(h=0.5 w=1.5)
       minor=(h=0.2 w=1.0 n=4);
```

Procédure gplot

```
data anno;
 set population;
 length text $ 10;
 if Annee=2000 then do;
  xsys='2'; ysys='2';
  position='2'; size=0.8 ;
  color='bl' ; style='hwdmx020';
  x=Annee;
  y=France;text='France';output;
  y=Germany;text='Allemagne';output;
 end ;
 keep x y text xsys ysys size color style position;
run;

proc gplot data=population gout=graf2;
plot France*Annee=3 Germany*Annee=2 / overlay haxis=axis1
   vaxis=axis2 annotate=anno;
run;
```


Procédure gplot



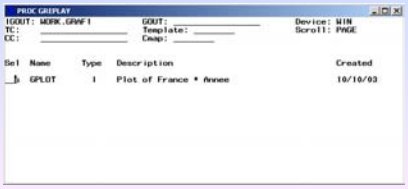
Sauvegarde des graphiques

Les graphiques sont sauvegardés dans des catalogues graphiques qui peuvent être temporaires ou assignés par un libname.

L'accès aux catalogues se fait par la procédure **proc greplay**.

Exemple : on effectue la procédure suivante :

```
proc greplay data=graf2;
run;
```



Dès que l'on souhaite écrire des programmes SAS généraux afin, par exemple, de les appliquer à différents jeux de données, il est nécessaire, par souci d'efficacité, de faire appel au macro langage SAS.

Les objets macros sont précédés des caractères & ou %.

Toute macro SAS débute par la déclaration :

%MACRO nom de la macro ;

et se termine par les 2 instructions ;

%MEND ;

%nom de la macro ;

Déclaration d'une macro-variable

L'instruction la plus simple pour créer une macro-variable est d'utiliser la commande **%LET**. Cette instruction permet d'associer une chaîne de caractères à un identificateur :

```
%LET nomvar1=taille ;
%LET nomvar2=poids ;
%LET varlist=csp sexe age taille poids revenu;
```

Une autre possibilité pour créer une macro- variable consiste à utiliser la routine **SYMPUT** dans une étape data. L'appel de cette routine se fait grâce à l'instruction CALL :

CALL SYMPUT(nommacro,valeur) ;

Exemple : on veut récupérer le nombre d'individus dans un fichier. On peut procéder de la manière suivante :

```
data _null_;
  set fichier;
  N=_N_;
  call symput('N',N);
run;
```


Interprétation des macro-variables

Une fois que la macro-variable est créée, il est possible de récupérer son contenu (valeur) à l'aide du code & suivi du nom de la macro variable désirée. Dans l'exemple précédent :

```
TITLE " Etude des variables &varlist " (avec " au lieu de ')
PROC PLOT ;
PLOT &nomvar1*&nomvar2 ;
RUN ;
```

Lorsque le processeur macro rencontre **&**, il recherche la macro variable qui suit ce & et remplace &nomvar1 par le texte contenu dans la macro. Si par exemple la valeur « taille » est affectée à la macro-variable &nomvar1 et la valeur « poids » à la macro-variable &nomvar2, le texte généré après la passage dans le processeur macro est le suivant :

```
PROC PLOT ;
PLOT taille*poids ;
RUN ;
```

Interprétation des macro-variables

L'interprétation de noms suffixés se fait différemment : supposons que l'on désire imprimer la première valeur d'une série de mesures effectuées à différentes dates ((taille1,poids1) ; (taille2,poids2) ; (taille3,poids3)) :

```
%LET nomvar1=taille ;
%LET nomvar2=poids ;
PROC PLOT;
PLOT &nomvar1.1*&nomvar2.1;
RUN;
```

Le texte généré par la processeur macro est le suivant :

```
PROC PLOT ;
PLOT taille1*poids1 ;
RUN ;
```

Le « . » situé après nomvar1 et nomvar2 indique au processeur macro la fin du nom de la macro-variable à chercher. Il est utilisé pour éviter des confusions, comme la recherche d'une macro-variable nomvar11.

Interprétation des macro-variables

La déclaration d'une macro-commande peut inclure des paramètres qui deviennent des macro-variables locales :

```
%MACRO PLOT(yvar,xvar) ;
PROC PLOT ;
PLOT &yvar*&xvar ;
RUN ;
%MEND ;
%PLOT(taille,poids) ;
```

On peut aussi définir des paramètres par défaut :

```
%MACRO PLOT(yvar=taille,xvar=poids) ;
PROC PLOT ;
PLOT &yvar*&xvar ;
RUN ;
%MEND ;
%PLOT ;
```


Macros instructions simples

Le macro-langage intègre des instructions des langages de programmation traditionnels pour permettre des traitements procéduraux (traitements conditionnels, boucles…).

Traitement conditionnel

L'instruction **%IF** permet de réaliser ce traitement. Sa syntaxe est la même que celle du langage SAS de base au % près et se présente comme suit.

Syntaxe :

```
%IF expression %THEN %DO;
        texte
%END;
%ELSE %DO;
        texte
%END;
```


Traitement itératif

Lorsqu'une séquence de commandes doit être répétée tant qu'une condition est réalisée, on utilise une des instructions suivantes :

Syntaxe 1 :

```
%DO %UNTIL(expression);
        texte
%END;
```

Le texte est répété tant que l'expression reste vraie.

Exemple : la macro suivante renvoie le nombre de mots contenus dans une chaîne de caractères :

```
%MACRO exemple;

 %MACRO Nbmot(chaine);

  %let i=1;
  %do %until(&lm=0);
   %let lm=%length(%scan(&chaine,&i));
   %let i=%eval(&i+1);
  %end; /* %do until(&lm=0) */
  %eval(&i-2)

 %MEND Nbmot;

 %let semaine=lundi mardi mercredi jeudi
              vendredi samedi dimanche;
 %put %Nbmot(&semaine);

%mend;
%exemple;
```


Traitement itératif

Syntaxe 2 (boucle avec compteur) :

```
%DO compteur=début %TO fin <BY increment> ;
        instructions
%END;
```

Exemple : la macro suivante renomme les colonnes du fichier :

```
%macro rencol;

data fichier;
 set fichier(keep=col1-col100);
 %do i=1 %to 100;
  rename col&i=var&i;
 %end;
run;

%mend;
%rencol;
```

Les entrées/sorties à l'écran

L'instruction **%PUT** permet d'écrire un texte dans la log. Cette instruction sert souvent à vérifier la valeur d'une macro-variable, d'une macro-expression.

L'instruction %INPUT permet de faire saisir des valeurs rentrées par l'utilisateur.

Syntaxe :

%INPUT <nom1 nom2…> ;

Exemple :

```
%macro accueil;
 %PUT Entrer votre nom et votre mot de passe :;
 %INPUT nom password;
%mend accueil;
%accueil;
```


Les entrées/sorties à l'écran

Le macro-processeur attribue les valeurs séparées par des blancs aux macro-variables dans l'ordre spécifié.

Si l'utilisateur tape un nom à particule, le programme précédent ne fonctionnera pas. Dans ce cas, il faut utiliser la macro &SYSBUFFR, comme suit :

```
%macro accueil;
  %put Entrer votre nom :;
  %input;
  %let nom=&SYSBUFFR;
  %put Entrer votre mot de passe :;
  %input password;
%mend accueil;
%accueil;
```


Environnements GLOBAL et LOCAL

Instruction %GLOBAL

%GLOBAL nommacro_1 <…nommacro_n>;

L'instruction %GLOBAL crée une macro-variable. Quel que soit l'endroit où est située cette ligne de code, la macro-variable nommacro est créée dans l'environnement global et sera utilisable pendant toute la session SAS.

Instruction %LOCAL

L'instruction %LOCAL n'est utilisable que dans un macro-programme. L'instruction %LOCAL est utilisée lorsque l'on désire conserver la valeur d'une macro-variable homonyme définie avant l'appel d'un macro-programme.

%LOCAL nommacro_1 <…nommacro_n> ;

Les entrées/sorties à l'écran

Exemple : dans l'exemple suivant, la macro variable locale index prend les valeurs 1 à 3 au cours du traitement de la macro keepm.

```
%let index=100;
%macro keepm(max=);
 %local index;
 %do index=1 %to &max;
  data period&index;
   set fichier;
   if period=&index;
  run;
 %end;
%mend;
%keepm(3);
```


La fonction %EVAL

Une macro-variable est toujours de type caractère, même si dans certains cas, son contenu peut-être interprété de manière numérique.

Exemple :

%LET x=5 ;
%LET y=7 ;
%LET z=&x+&y ;

L'exemple précédent affecte à z la valeur 5+7. Pour affecter la valeur 12 à z, il faut utiliser la fonction **%EVAL**.

%LET z=%EVAL(&x+&y) ;

Attention : le macro-langage ne traite que des entiers.

Quelques fonctions de traitement des chaînes de caractère

La fonction %LENGTH

La macro-fonction **%LENGTH** fournit la longueur d'une macro-expression.

Exemple :

```
%macro valeur;
 %let var1=Liste de caracteres;
 %let longueur1=%length(&var1);
 %put result1=&longueur1;     => result1 = 19
 %let var2=                   Test;
 %let longueur2=%length(&var2);
 %put result2=&longueur2;     => result2 = 4
```

VI. Le macro langage SAS

La fonction %LENGTH

Exemple :

```
%let var3=%str(                Test  );
 %let longueur3=%length(&var3);
 %put result3=&longueur3;        => result3 = 22

%mend;
%valeur;
```

La macro-fonction %LENGTH ignore les blancs externes aux macro-expressions. C'est pourquoi result2 vaut 2. En revanche, tous les espaces contenus entre deux caractères d'une macro-expression sont comptabilisés.

Pour que les espaces externes à une macro-expression soient pris en compte, il est nécessaire d'utiliser les macros-fonctions de "quoting". Dans l'exemple précédent, la macro-fonction %STR est utilisée pour pouvoir tenir compte des espaces externes à la valeur Test. C'est pourquoi result3 vaut 22

VI. Le macro langage SAS

La fonction %SCAN

La macro-fonction **%SCAN** extrait un mot (macro-expression) séparée par deux délimiteurs dans une macro-expression.

Syntaxe :

%LET retour=%SCAN(argument, nième,<delimiteurs>) ;

Exemple :

```
%let LF=1-2 1-3 ; /*liste de plans factoriels*/
%let I=1 ; /*nombre de plans factoriels à représenter*/
%do %until(&LM=0);
 %let LM=%length(%scan(&LF,&I," "));
 %let I=%eval(&I+1);
%end;
%let NPLANS=%eval(&I-2);
%do I=1 %to &NPLANS;
 %let P&I.AXH=%scan(%scan(&LF,&I," "),1,"-");
 %let P&I.AXV=%scan(%scan(&LF,&I," "),2,"-");
 %let P&I=%scan(%scan(&LF,&I," "),1,"-")_%scan(%scan(&LF,&I,"
"),2,"-");
%end;
```


VI. Le macro langage SAS

La fonction %SUBSTR

La macro-fonction **%SUBSTR** permet d'extraire une sous-chaîne de caractères d'une macro-expression.

Exemple : la macro suivante extrait les deux derniers caractères d'un mot :

```
%MACRO exemple;

 %macro Extcar(mot) ;
 %substr(&mot,%eval(%length(&mot)-1),2)
 %mend;

 %let period=annee88;
 %let annee=%Extcar(&period);
 %put &annee;

%mend;
%exemple;
```

=> Le résultat 88 s'affiche dans la fenêtre output.

VI. Le macro langage SAS

La fonction %UPCASE

La macro-fonction **%UPCASE** transforme une expression en majuscule.

Exemple :

%put %upcase(minuscules);

Le résultat est le suivant :

MINUSCULES

La fonction %STR

La macro-fonction %STR empêche le macro-langage d'interpréter les caractères de l'argument.

Exemple :

%LET var=%STR(or) ;

empêche l'interprétation de la valeur var : or comme opérateur logique.

VI. Le macro langage SAS

La fonction %NRSTR

La macro-fonction **%NRSTR** va encore plus loin que la macro-fonction %STR. Toute référence à une macro variable (&) ou macro-programme ou fonction (%) est ignorée.

Exemple :

```
%let var1=A;%put &var1;
%let var2=%str(&var1*B);%put &var2;
%let var3=%nrstr(&var1*B);%put &var3;
```

Le résultat est le suivant :

A

A*B

&VAR1*B