

Introduction à SAS



Programmation & Logiciels Statistiques

Cours 3

Introduction à SAS

- SAS est un système intégré pour la *manipulation*, l'*analyse* et la *présentation* des données
- C'est un système *modulaire*, de nombreux modules pouvant être ajoutés au système de base : **SAS Base**
- Dans cette introduction, on se concentrera sur **SAS Base** et sur les modules **SAS Stat** et **SAS Graph**
- Les principaux modules de SAS sont les suivants :

Introduction à SAS

- **SAS Base** : manipulation de données, statistiques descriptives, édition de rapports. Il permet de programmer en langage SAS, en SQL, en SCL (SAS Component Language) et en langage Macro.
- **SAS/STAT** : modélisation (régressions diverses), classification, statistiques descriptives avancées. Depuis SAS 9.1, permet la production de certains graphiques statistiques sans licence SAS/GRAPH.

Introduction à SAS

- **SAS/GRAPH** : production de nuages de points, courbes, diagrammes circulaires et en barres, cartes géographiques, etc.
- **SAS/ACCESS** : série de modules autorisant l'importation et l'exportation de données depuis et vers d'autres applications.
- **SAS/CONNECT** : permet de faire fonctionner SAS en mode client/serveur.

Introduction à SAS

- **SAS/ETS** : analyse économétrique et séries temporelles (ETS = Econometrics and Time Series).
- **SAS/AF** : permet la construction d'interfaces clique-bouton par-dessus SAS, de façon analogue aux interfaces Visual Basic pour les applications Microsoft (AF = Application Facility).
- **SAS/EIS** : permet la constitution et la visualisation de **tableaux de bord synthétiques** regroupant des statistiques et des graphiques (EIS = Executive Information System).

Introduction à SAS

- **SAS/ASSIST** : interface clique-bouton pour la création de programmes SAS.
- **SAS/IML** : permet d'utiliser un langage matriciel dans les programmes SAS (IML = Interactive Matrix Language).
- **SAS/OR** : Optimisation et recherche opérationnelle.
- **SAS/QC** : Contrôle de Qualité

Introduction à SAS

- SAS/INSIGHT : analyse statistique interactive
- SAS/OLAP, SAS/LAR LAB, SAS/MDDDB
COMMON PRODUCTS, SAS/SHARE...etc.

Le langage SAS

- Au cœur de SAS, il y a un *langage de programmation* composé d'instructions spécifiant la façon dont les données doivent être traitées et analysées
- Un programme SAS consiste en une suite d'instructions *SAS regroupées en blocs*
- Ces blocs sont appelés **étapes**

Le langage SAS

- Il y a deux types d'étapes : les **étapes data** et les **étapes proc** (pour PROCédure)
- Un **étape data** sert à préparer les données pour l'analyse
- Elle permet de *créer un tableau de données SAS, d'organiser les données et éventuellement de les modifier*
- Une **étape proc** est utilisée pour *analyser* les données d'un tableau SAS

Le langage SAS

- Un programme typique peut consister en une **étape data** pour lire des données brutes, suivie de plusieurs **étapes proc** pour les analyser
- Si, au cours de l'analyse, les données doivent être modifiées, une autre **étape data** est requise pour ce faire
- *Apprendre le langage SAS, c'est essentiellement apprendre les instructions requises pour effectuer une analyse et apprendre la façon de les structurer en étapes*

Le langage SAS

- Quelques principes généraux à retenir sont les suivants
- La plupart des instructions SAS commencent par un ***mot-clef*** identifiant le type d'instruction dont il s'agit
- ***Toutes les instructions SAS doivent se terminer par un point-virgule***
- Une instruction SAS peut s'étendre sur plusieurs lignes. Toutefois, ***il est préférable de se limiter à une instruction par ligne***. Cela permet d'éviter les erreurs et d'identifier plus facilement celles-ci le cas échéant

Le langage SAS

- Les instructions SAS peuvent être classées en **quatre catégories** selon la partie d'un programme où elles peuvent être utilisées
 - i. Les instructions d'étape data
 - ii. Les instructions d'étape proc
 - iii. Les instructions communes aux étapes data et étapes proc
 - iv. Les instructions *globales*, s'appliquant à toutes les étapes qui les suivent

Le langage SAS

- Un exemple simple d'instruction globale est l'instruction **title** qui définit un titre à imprimer sur les sorties des procédures et sur les graphes
- Le même titre est utilisé jusqu'à qu'il soit modifié ou réinitialisé
- Les étapes **data** et **proc** commencent respectivement par l'instruction **data** et **proc** et se terminent à l'instruction **data** ou **proc** suivante ou à l'instruction **run** suivante

Le langage SAS

- Lorsque les données sont incluses dans une étape data, l'étape se termine après les données (des exemples seront vus dans la suite)
- Il est important de comprendre où commence et s'achève chaque étape car *les programmes SAS ne sont pas exécutés instruction par instruction mais étape par étape*
- **Lors des premières utilisations de SAS, il est utile de marquer la fin de chaque étape par une instruction run**

Le langage SAS

- Une **règle importante** concerne les noms donnés aux variables et tableaux de données
- Ceux-ci peuvent contenir des lettres, des nombres et `_`, mais **ne peuvent être de longueur supérieure à huit caractères et ne peuvent commencer par un nombre**
- Il est possible, mais déconseillé, d'utiliser des noms réservés par SAS, par exemple des noms de fonctions

Le langage SAS

- Lorsqu'on doit référer à une liste de variables dans un programme SAS, on peut souvent utiliser une **forme abrégée**
- Par exemple, une liste de variables de la forme **sexe - - poids** réfère aux variables sexe et poids et à toutes les variables positionnées entre elles dans le tableau de données

Le langage SAS

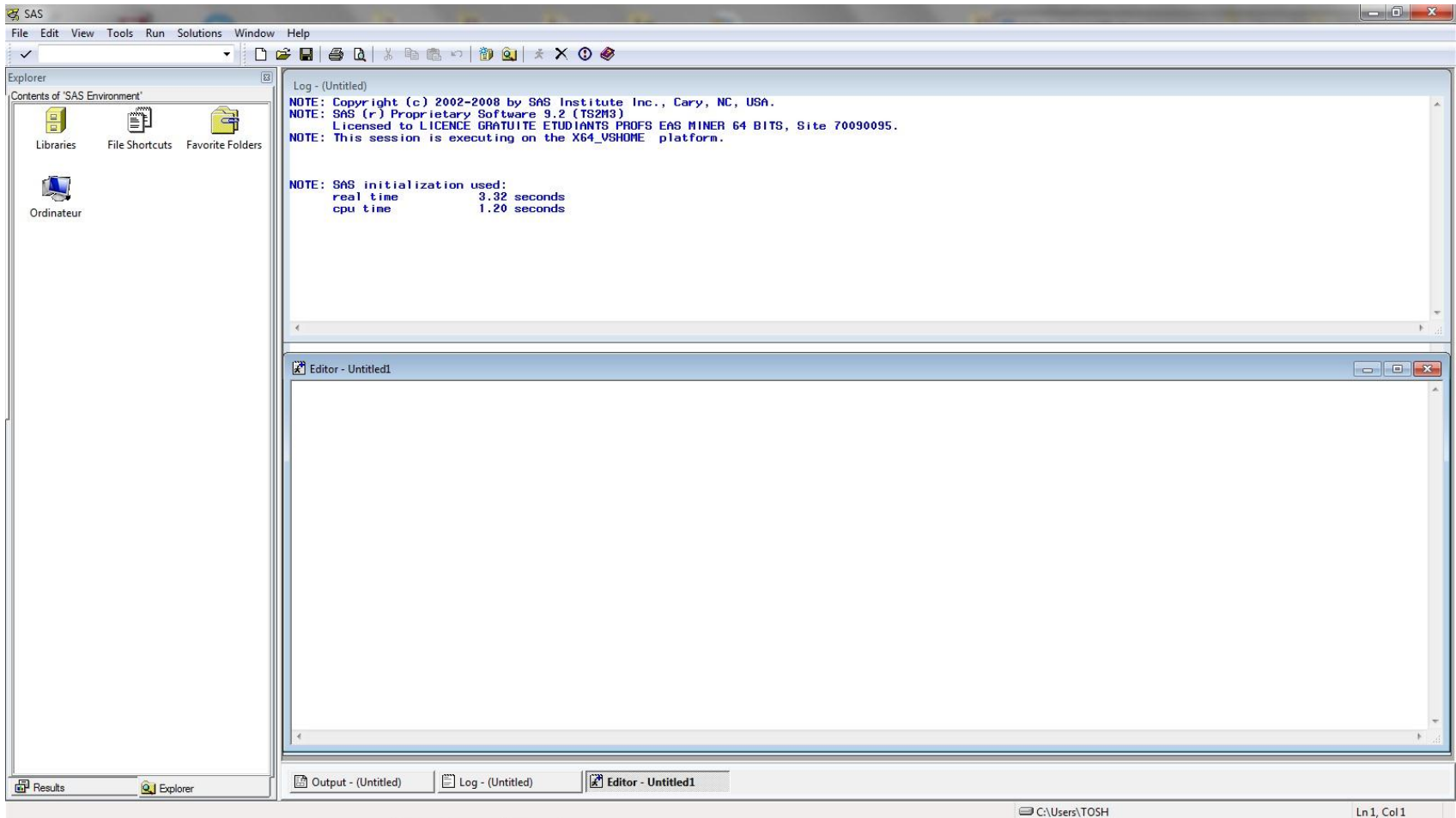
- Un autre exemple : lorsque des variables possèdent des noms de la forme score1, score2, ... score10, c'est-à-dire qu'ils ont une racine en commun mais se terminent par des nombres consécutifs, on peut y référer par une liste de variables de la forme **score1-score10** ; les variables n'ont pas à être contiguës dans le tableau de données

Le langage SAS

Un exemple simple

```
data bodyfat;
    input age pctfat sexe $;
cards;
    23 27.9 H
    39 31.4 F
    41 25.9 F
    49 25.2 H
    50 31.1 H
    53 34.7 H
    53 42.0 F
    54 29.1 H
    56 32.5 H
    57 30.3 H
    58 33.0 F
    58 33.8 F
    60 41.1 F
    61 34.5 F
;
proc print data = bodyfat;
run;
proc corr data = bodyfat;
run;
```

L'interface Windows



L'interface Windows

- Cet écran est désigné sous le nom de *Display Manager System* (DMS)
- Il est composé de cinq fenêtres principales
 - I. La fenêtre **Editeur (Program editor)** : elle permet de saisir les programmes et d'en demander l'exécution. Les différentes commandes saisies s'affichent en différentes couleurs.
 - II. La fenêtre **Journal (Log)** : s'y affichent les instructions soumises et les commentaires de SAS sur les programmes exécutés, ainsi que les messages d'erreurs et diverses notifications

L'interface Windows

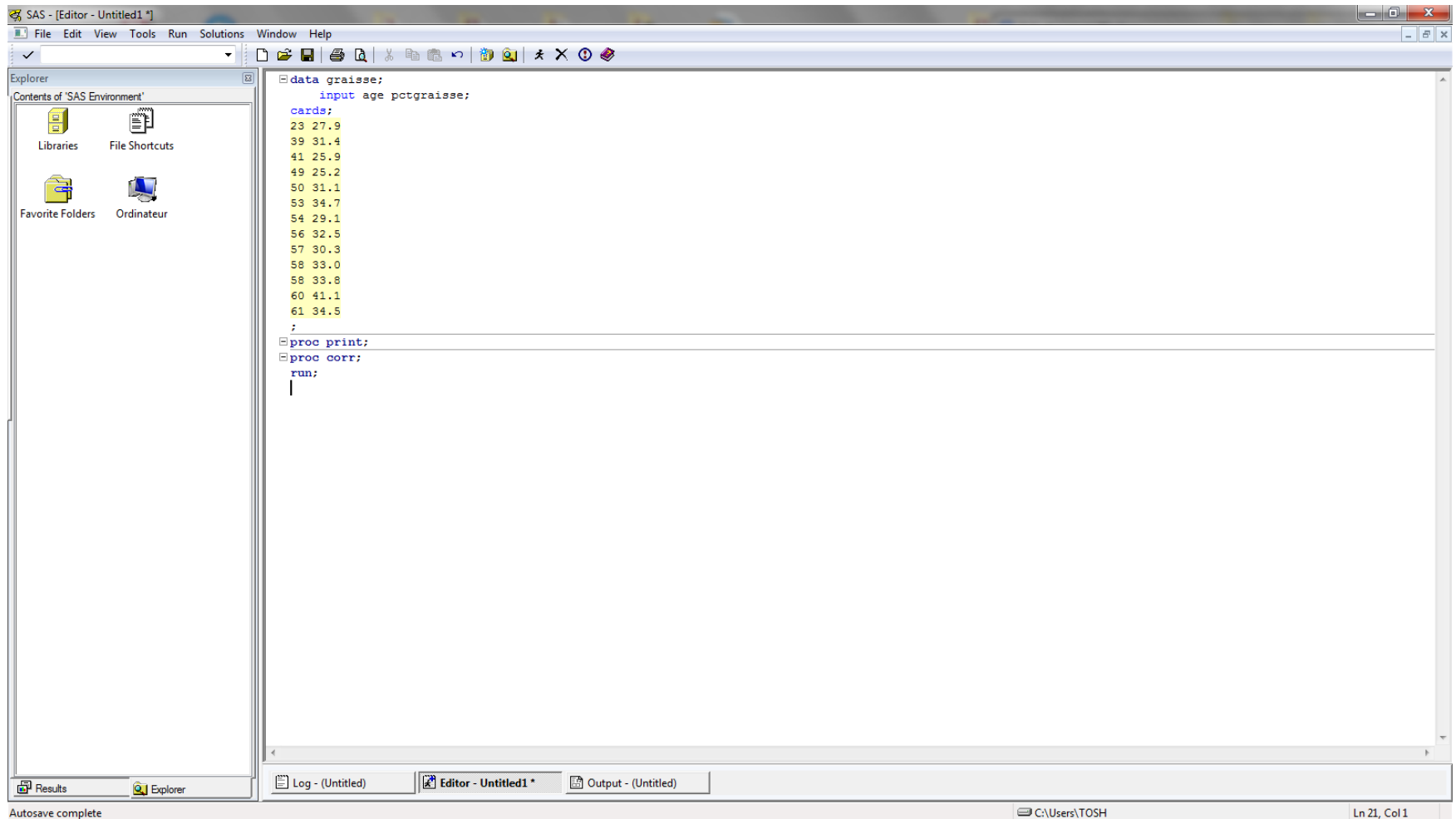
- III. La fenêtre **Sortie (Output)** : s'y affichent les sorties des programmes exécutés
- IV. La fenêtre **Explorateur (Explorer)** : comme l'explorateur de Windows, cette fenêtre permet de *naviguer* dans les diverses **bibliothèques (libraries)** de SAS et dans le poste de travail. On trouve dans le dossier Favoris les répertoires Mes Documents et le Bureau. Dans le dossier Raccourcis de Fichiers, on peut créer des raccourcis vers différentes fichiers du disque dur

L'interface Windows

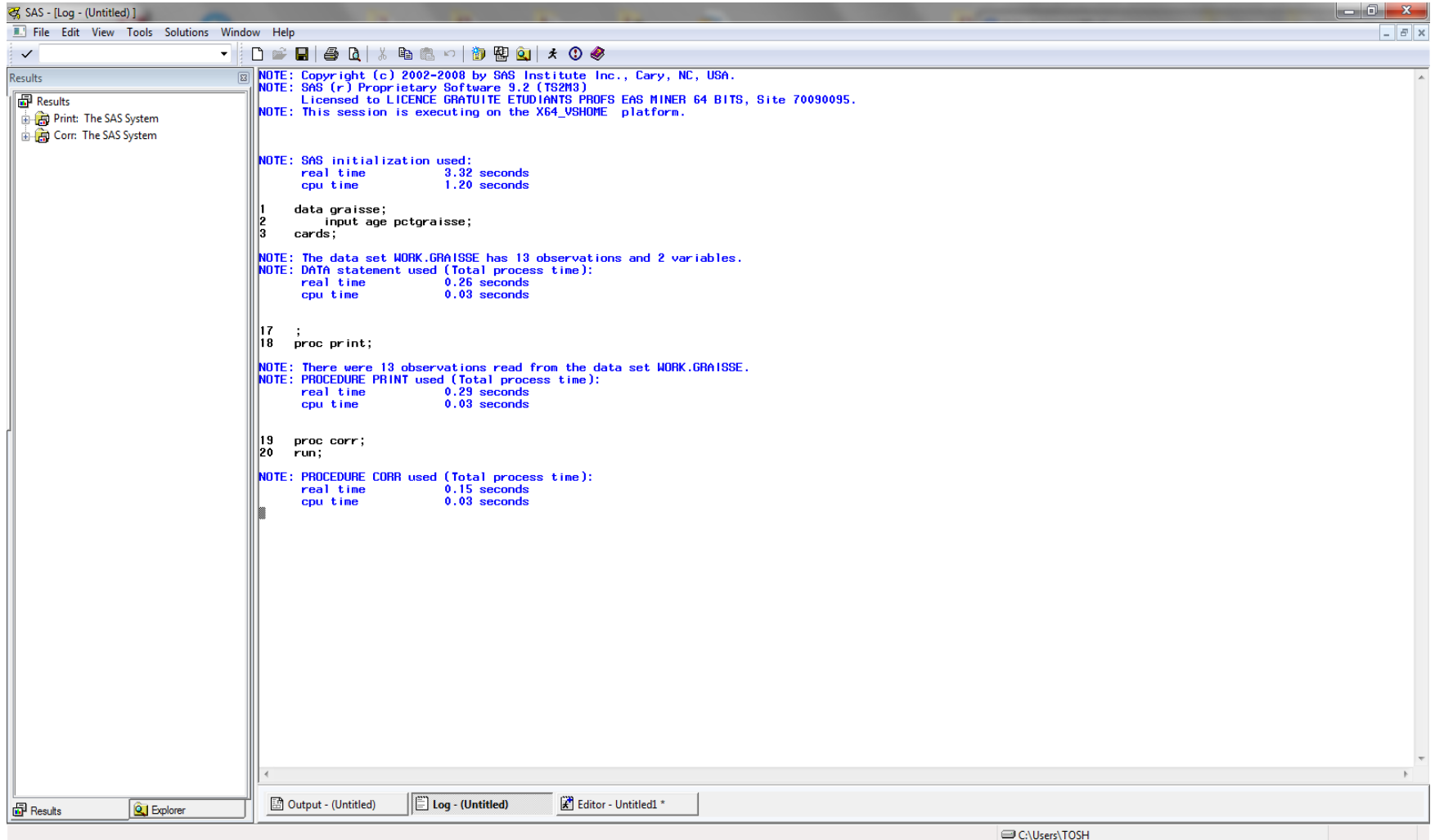
V. La fenêtre **Résultats (Results)** : elle permet de naviguer entre les divers résultats qui s'affichent dans la fenêtre Sortie et d'effacer certaines sorties pour ne conserver que les plus utiles

Les graphiques produits sont envoyés vers une nouvelle fenêtre **Graph**. Les sorties générées par ODS sont envoyées dans une fenêtre **Results Viewer** et peuvent être envoyées en même temps dans la fenêtre Sortie.

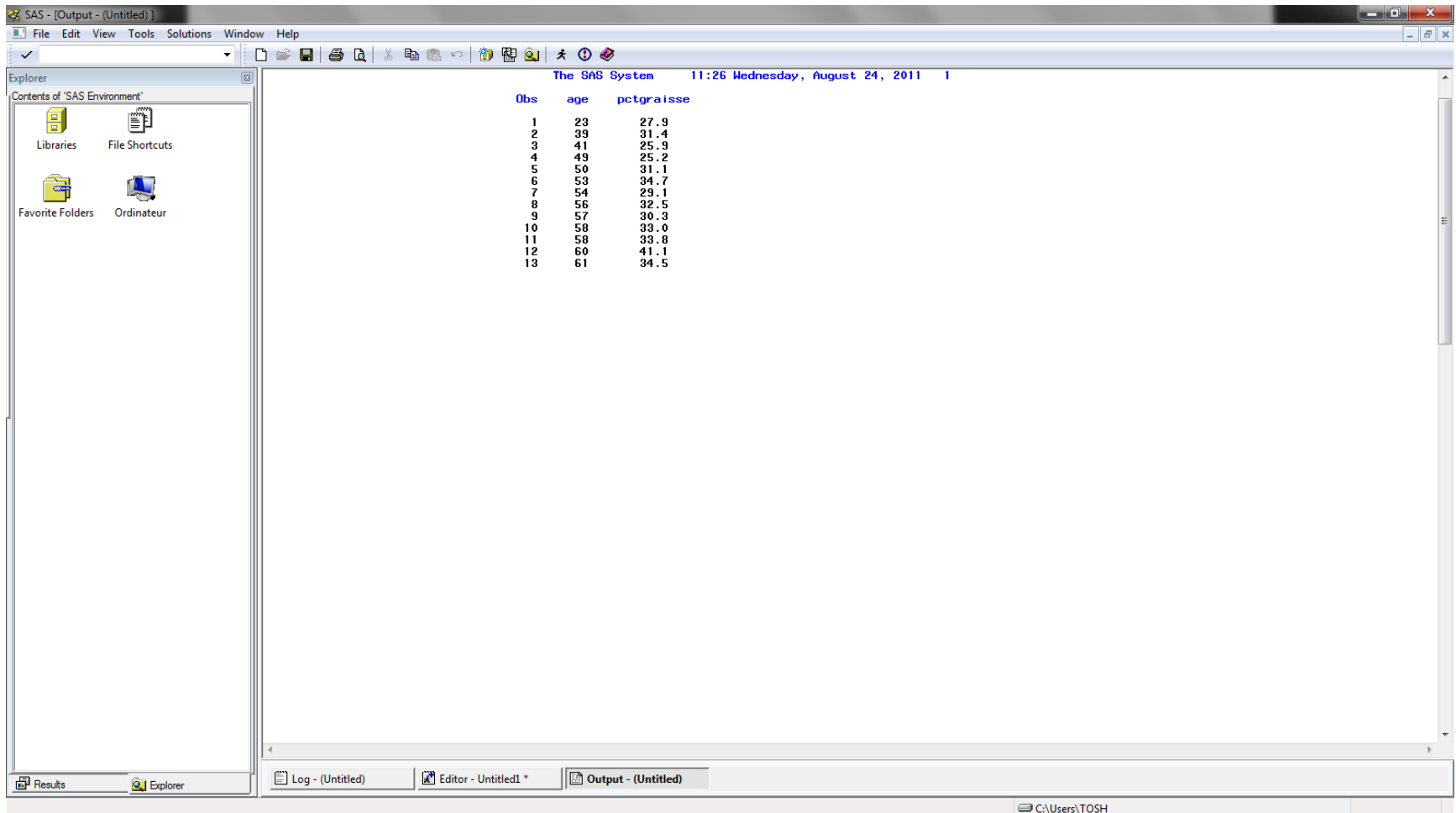
Exemple : fenêtre Editeur



Exemple : Fenêtre log



Exemple : Fenêtre Sortie



The screenshot shows the SAS Output window titled "SAS - [Output - (Untitled)]". The window displays the following data table:

Obs	age	pctgraisse
1	23	27.9
2	39	31.4
3	41	25.9
4	49	25.2
5	50	31.1
6	53	34.7
7	54	29.1
8	56	32.5
9	57	30.3
10	58	33.0
11	58	33.8
12	60	41.1
13	61	34.5

The window also shows the Explorer pane on the left with "Contents of 'SAS Environment'" and "The SAS System" window at the top right displaying the date and time: "11:26 Wednesday, August 24, 2011". The taskbar at the bottom shows the "Output - (Untitled)" window is active.

Exemple : Fenêtre Sortie

The SAS System 11:26 Wednesday, August 24, 2011 2

The CORR Procedure

2 Variables: age pctgraisse

Simple Statistics

Variable	N	Mean	Std Dev	Sum	Minimum	Maximum
age	13	50.69231	10.74232	659.00000	23.00000	61.00000
pctgraisse	13	31.57692	4.19566	410.50000	25.20000	41.10000

Pearson Correlation Coefficients, N = 13
Prob > |r| under H0: Rho=0

	age	pctgraisse
age	1.00000	0.57559 0.0396
pctgraisse	0.57559 0.0396	1.00000

Results Explorer Log - (Untitled) Editor - Untitled1 * Output - (Untitled) C:\Users\TOSH

L'étape data

- Une **étape data** est utilisée pour créer une **table SAS** pour une analyse subséquente.
- Les données peuvent être des **données brutes** ou une **table SAS** précédemment créée.

Etape data

1. Lecture de données brutes

- Ci-contre : données sur les membres d'un club de natation avec :
numéro de licencié,
équipe, poids au début,
dernier poids

```
1023 rouge 189 165
1049 jaune 145 124
1219 rouge 210 192
1246 jaune 194 177
1095 bleu 135 127
1352 vert 156 137
1262 bleu 196 180
1057 bleu 146 132
```

- Les données sont sauvegardées dans le fichier **wgtclub.dat**

Etape data

Lecture de données brutes

- Ci-contre : instructions SAS requises pour créer une table SAS
- L'instruction **data** spécifie le nom de la table créée : ici **wgtclub**
- L'instruction **infile** spécifie l'emplacement des données brutes

```
data wgtclub;  
    infile "d:\data\wgtclub.dat";  
    input idno equipe $ poids_deb  
    poids_fin;  
run;
```

Etape data

Lecture de données brutes

```
data wgtclub;  
infile "d:\data\wgtclub.dat";  
input idno equipe $ poids_deb  
poids_fin;  
run;
```

- Un certain nombre d'options peuvent être spécifiées pour l'instruction **infile**
- Celles-ci permettent d'altérer la façon dont les données sont lues
- L'option **pad** permet d'ajouter des espaces à la fin de chaque ligne de données lorsqu'elle est lue jusqu'à ce que la ligne soit de longueur égale au **logical record length**

Etape data

Lecture de données brutes

- C'est particulièrement utile en conjonction avec l'option **lrecl** = qui permet de spécifier le **logical record length**
- Ainsi **lrecl = 100 pad** aurait pour effet de rajouter des espaces à la fin de chaque ligne jusqu'à ce que sa longueur soit de 100 caractères
- L'option **expandtabs** a pour effet de transformer des caractères de tabulation en autant d'espaces

Etape data

Lecture de données brutes

```
data wgtclub;  
infile "d:\data\wgtclub.dat";  
input idno equipe $ poids_deb  
poids_fin;  
run;
```

- L'instruction **run** complète l'étape **data**
- Lorsqu'un programme est soumis, l'exécution est faite étape par étape plutôt qu'instruction par instruction
- Sans une instruction **run** à la fin du programme, il est possible que la dernière instruction ne soit pas exécutée
- **En pratique, il est conseillé d'inclure une instruction run après chaque étape**

Etape data

Lecture de données brutes

```
data wgtclub;  
infile "d:\data\wgtclub.dat";  
input idno equipe $ poids_deb  
poids_fin;  
run;
```

- L'instruction **input** spécifie que quatre variables doivent être lues dans le fichier de données brutes : **idno, equipe, poids_deb, poids_fin**
- Le signe **\$** après **equipe** signale qu'il s'agit d'une variable de type caractère
- **SAS a seulement deux types de variables : numérique et caractère**

Etape data

Lecture de données brutes

- Ici, les données sont séparées par des espaces
- Ceci justifie la forme simple de l'instruction **input** qui est utilisée : *les noms des variables sont listés dans l'ordre et les variables caractères sont identifiées par le signe \$*
- **On dit qu'il s'agit d'un input de la forme « liste »**

Etape data

Lecture de données brutes

- SAS admet trois formes principales d'input
 - I. Mode liste
 - II. Mode colonne
 - III. Mode formaté
- Il en existe d'autres qui ne seront pas traitées dans cette introduction
- Le mode liste est le plus simple et doit être préféré pour cette raison

Etape data

Lecture de données brutes

- La condition que les données doivent être séparées par des espaces a deux conséquences importantes
 - a) Les valeurs manquantes ne peuvent être représentées par des espaces; on utilise un point (.) à la place
 - b) Les valeurs caractères ne peuvent contenir d'espaces

Etape data

Lecture de données brutes

- Une autre caractéristique importante du mode liste est que la longueur par défaut pour les variables caractère est de 8
- Lorsqu'on utilise un input de mode liste, il faut **toujours** examiner le Log SAS pour vérifier que le nombre correct de variables et d'observations a été lu
- Le message « SAS went to a new line when INPUT statement reached past the end of a line » indique souvent un problème

Etape data

Lecture de données brutes

- Si c'est le cas, il peut être nécessaire d'avoir recours aux options **lrecl** et **pad** de **infile**
- Pour de petits tableaux de données, il est conseillé de les imprimer en sortie à l'aide de la **proc print** pour vérifier que les données ont été lues correctement
- Si le mode liste n'est pas approprié, on peut utiliser le **mode colonne**

Etape data

Lecture de données brutes

- Ci-contre : données sur les membres d'un club de natation avec :
identifiant, nom, équipe, poids au début, dernier poids
- Les données sont sauvegardées dans le fichier **wgtclub2.dat**

1023	David Shaw	rouge	189	165	H
1049	Amelia Serrano	jaune	145	124	F
1219	Alan Nance	rouge	210	192	H
1456	Ravi Sinha	jaune	194	177	H
2015	Richard Rose	bleu	135	127	H
2016	Hisashi Ito	vert	156	137	H
2045	Raoul Sanchez	bleu	196	180	H
2630	Yao Chen	bleu	146	132	H

Etape data

Lecture de données brutes

- Ci-contre : instructions SAS requises pour créer la table SAS **wgtclub2**
- La différence avec le mode liste est **qu'on spécifie les numéros des colonnes contenant les données** après le nom de chaque variable (ou après le \$ le cas échéant)
- Les numéros de colonnes de début et de fin sont séparées par un tiret

```
data wgtclub2;  
  infile "d:\data\wgtclub2.dat";  
  input idno 1-4 nom $6-19 equipe $21-25  
  poids_deb 27-29 poids_fin 31-33 sexe  
  $35;  
  
run;  
proc print;  
run;
```

- Pour une variable limitée à une colonne, on se contente de spécifier le numéro correspondant

Etape data

Lecture de données brutes

The screenshot displays the SAS interface. The main window shows the output of a SAS program, which is a table of data. The table has five columns: 'Obs', 'name', 'equipe', 'poids_deb', and 'poids_fin'. The data is as follows:

Obs	name	equipe	poids_deb	poids_fin
1	David Shaw	rouge	189	165
2	Amelia Serrano	jaune	145	124
3	Alan Nance	rouge	210	192
4	Ravi Sinha	jaune	194	177
5	Richard Rose	bleu	135	127
6	Hisashi Ito	vert	156	137
7	Raoul Sanchez	bleu	196	180
8	Yao Chen	bleu	146	132

Below the table, the SAS code used to generate the output is shown in a separate window titled 'weightclub2*':

```
data wgtclub2;
  infile "d:\data\wgtclub2.dat";
  input name $ 1-14 equipe $ 16-20
        poids_deb 22-24 poids_fin 26-28;
run;
proc print;
run;
```

The bottom of the window shows the status bar with 'Autosave complete', the user name 'C:\Users\TOSH', and the current position 'Ln 7, Col 5'.

Étape data

Lecture de données brutes

- En mode formaté, chaque variable est suivie de son « format » (de son **informat** en jargon SAS)
- Alternativement, une liste de variables entre parenthèse est suivie d'une liste de formats correspondants entre parenthèses
- Par exemple, pour lire les données précédentes en mode formaté, on procède de la façon suivante

Etape data

Lecture de données brutes

```
1023 David Shaw      rouge 189 165 H
1049 Amelia Serrano  jaune 145 124 F
1219 Alan Nance      rouge 210 192 H
1456 Ravi Sinha      jaune 194 177 H
2015 Richard Rose    bleu  135 127 H
2016 Hisashi Ito     vert  156 137 H
2045 Raoul Sanchez   bleu  196 180 H
2630 Yao Chen        bleu  146 132 H
```

```
data wgtclub3;
    infile "d:\data\wgtclub2.dat";
    input idno 5. nom $15. equipe $6.
    poids_deb 4. poids_fin 4. sexe
    $1.;
run;
proc print;
run;
```

- Le mode formaté est le plus flexible, en partie parce qu'un vaste choix d'**informats** est disponible

Etape data

Lecture de données brutes

- L'**informat** pour une variable **caractère** consiste en un **\$**, le **nombre de colonnes** occupées par les données et **un point**
- La forme la plus simple d'**informat** pour une variable **numérique** est simplement le **nombre de colonnes** occupées par les données et **un point**
- Il est important de noter que **les espaces séparant les données doivent être pris en compte** dans l'**informat**

Etape data

Lecture de données brutes

- Lorsque des données numériques contiennent un point décimal non explicite l'**informat** a un second nombre après le point, indiquant le nombre de chiffres à **droite** du point décimal
- Par exemple, un **informat** de 5.2 s'interprète comme 5 colonnes de données numériques plus insertion du séparateur décimal après 2 colonnes à partir de la droite
- **Lorsque les données contiennent un point décimal explicite celui-ci a priorité par rapport à l'informat**

Etape data

Lecture de données brutes

- Lorsque des données utilisent la virgule comme séparateur de milliers, on peut utiliser l'informat `comma` de la façon suivante

```
data commas;  
    input bignum comma6. ;  
  
cards ;  
1,860  
;
```

Étape data

Lecture de données brutes

- Le mode formaté doit être utilisé dès lors que les données ne sont pas en format numérique standard
- Ce type de données est rare en pratique
- L'usage le plus commun d'**informats** SAS sont les **informats de date**
- **Lorsqu'une date est lue en utilisant un informat de date, le résultat est le nombre de jours écoulés depuis le 1^{er} janvier 1960**

Etape data

Lecture de données brutes

- L'étape **data** ci-contre illustre l'utilisation de l'**informat** de date **ddmmyyw.** . Le **w** correspond à la largeur est doit être compris entre 6 et 32 colonnes.
- Pour des dates au format américain, on utilise l'**informat** **mmddyw.** .

```
data days;
            input day ddmmyy8.;
cards;
231090
23/10/90
23 10 90
23101990
;
run;
proc print data=days;
run;
proc print data=days;
            format day ddmmyy10.;
run;
```

Obs	day	Obs	day
1	11253	1	23/10/1990
2	11253	2	23/10/1990
3	11253	3	23/10/1990
4	11253	4	23/10/1990

Etape data

Lecture de données brutes

- Le mode formaté permet d'être beaucoup plus concis que le mode colonne, particulièrement quand des variables successives sont de même format
- Par exemple, si les premières 20 colonnes d'une ligne de données contiennent les réponses à 20 questions, chacune consistant en un seul chiffre, les données peuvent être lues par

```
input (q1-q20) (20*1.);
```

Etape data

Lecture de données brutes

- Les **informat**s de la liste sont répétés en les préfixant avec **n***, où **n** est le nombre de fois où l'**informat** doit être répété
- La liste entière d'**informat**s est réutilisée s'il y a moins d'**informat**s que de variables. Par exemple, l'instruction précédente pourrait être réécrite sous la forme

```
input (q1-q20) (1.);
```

Etape data

Lecture de données brutes

- La propriété précédente est particulièrement utile quand les données contiennent des groupes qui se répètent. Si les réponses aux 20 questions occupent alternativement 1 et 2 colonnes, on peut les lire avec

```
input (q1-q20) (1.2.);
```

- **Les différents modes d'input peuvent être combinés dans une même instruction input, bien que cela soit rarement nécessaire**

Etape data

Lecture de données brutes

- Dans une instruction `input` lorsque les données d'une observation occupent plusieurs lignes, le caractère `/` (slash) indique où commencer à lire les données de la ligne suivante
- Une autre possibilité est d'écrire une instruction `input` séparée pour chaque ligne de données, puisque SAS passe automatiquement à la ligne suivante après avoir exécuté chaque instruction `input`

Etape data

Lecture de données brutes

- Dans certains cas, il est utile d'empêcher SAS de passer automatiquement à la ligne suivante
- Cela peut être fait en ajoutant **@@** à la fin de l'instruction **input**.
- Ces propriétés seront illustrées en TD.

Etape data

Données délimitées

- Il y a deux types de données qui méritent une mention particulière : les données **séparées par des tabulations** et celles **séparées par des virgules (.csv)**
- Bien que l'input de mode liste soit en général utilisé pour lire des données **séparées par des espaces**, il peut être utilisé pour lire ces types de données délimitées
- Un problème important dans ce contexte est le traitement à réserver à l'occurrence de deux délimiteurs consécutifs

Etape data

Données délimitées

- Afin que l'occurrence de deux délimiteurs consécutifs soit interprétée comme l'occurrence d'une **valeur manquante**, on peut utiliser l'option `dsd` de l'instruction `infile`
- La façon la plus simple de lire des données séparées par des tabulations est d'utiliser un input de mode liste avec l'option `expandtabs` dans l'instruction `infile`
- Cela a pour effet de remplacer les tabulations par des espaces

Etape data

Données délimitées

- Si deux tabulations consécutives indiquent une valeur manquante, on doit spécifier les options `delimiter=` et `dsd`; par exemple,

```
infile 'nomfichier' delimiter='09'x dsd;
```
- La valeur 09 correspond au code hexadécimal du caractère tabulation dans l'alphabet ASCII
- Pour des données séparées par des virgules, l'option `dsd` suffit en général puisque le délimiteur par défaut pour `dsd` est la virgule

Etape data

Données délimitées

- Certains fichiers csv peuvent contenir des valeurs entre guillemets lorsque les données contiennent des virgules
- Ce cas est pris en charge par l'option `dsc` qui supprime les virgules entre guillemets puis supprime les guillemets lors de la lecture des données
- L'option `misover` est recommandée pour lire des fichiers csv; cela permet d'éviter que SAS ne revienne à la ligne lorsque la dernière valeur est manquante

Etape data

Données délimitées

- Les fichiers csv peuvent contenir les noms des variables en première ligne. Afin de ne pas lire celle-ci, on peut utiliser l'option `firstobs=2`
- La forme recommandée de l'instruction `infile` pour des données csv est donc

```
infile 'nomfichier' dsd missover;
```

ou

```
infile 'nomfichier' dsd missover firstobs=2;
```

lorsque la première ligne contient le nom des variables.

Etape data

L'alternative Proc Import

- Pour des données séparées par des tabulations ou des virgules, en particulier lorsque la première ligne contient le nom des variables, on peut utiliser `proc import` pour lire les données plutôt qu'une étape **data**
- Par exemple, pour lire un fichier de données délimitées par des tabulations avec les noms des variables en première ligne, on peut utiliser

Etape data

L'alternative Proc Import

```
proc import datafile='nomfichier'  
    out=tableausas dbms=tab replace;  
    getname=yes;  
run;
```

- Pour des données séparées par des virgules, spécifier **dbms=csv**
- La **proc import** détecte en général de façon correcte si les variables sont de type numérique ou caractère mais il est préférable de vérifier le résultat obtenu

Etape data

L'alternative Proc Import

- Une façon alternative d'utiliser **la proc import** est de faire appel à l'utilitaire d'importation de données
- Celui-ci est accessible à partir du menu **Fichier/Import Data...**

Etape data

SAS/ACCESS

- SAS dispose de divers modules permettant de lire des données issues de bases de données propriétaires
- Ceci requiert la licence SAS/ACCESS et ne sera pas détaillé dans ce cours
- Mentionnons simplement que le module SAS/ACCESS permet à la `proc import` de lire des classeurs au format Access, Excel, Dbase et Lotus. L'écran 1 de l'utilitaire d'importation fournit la liste des sources de données autorisées

Etape data

2. Lecture de données dans une table SAS

- Pour lire des données dans une table SAS plutôt que dans un fichier de données brutes, l'instruction **set** est utilisée à la place de **infile** et **input**
- Le code

```
data wgtclub2;  
    set wghtclub;  
run;
```

Etape data

Lecture de données dans une table SAS

- Est utilisé pour créer une table SAS **wgtclub2** en lisant les données de la table **wghtclub**
- Il est possible de conserver le nom de la table, l'instruction data précédente s'écrivant alors

```
data wghtclub;  
    set wghtclub;  
run;
```

- **C'est généralement réservé au cas où l'étape data modifie les données**

Etape data

3. Sauvegarde de tables SAS sur disque

- Jusqu'à présent, les exemples considérés ont concerné des table SAS temporaires
- Elles sont temporaires car elles sont effacées lorsqu'on quitte SAS
- Pour sauvegarder des tables SAS de façon permanente sur le disque et pour y accéder par la suite, on utilise l'instruction **libname** et on se réfère à la table de façon différente :

Etape data

Sauvegarde de tables SAS sur disque

```
libname dir "d:\data\sasdata";  
data dir.wghtclub;  
    set wghtclub;  
run;
```

- L'instruction **libname** spécifie que la **libref dir** réfère au répertoire **d:\data\sasdata**
- Après cela, toute table SAS préfixée avec **dir.** référera à une table stockée dans ce répertoire
- L'étape data ci-dessus lit les données de la table SAS temporaire **wghtclub** et les sauvegarde dans une table permanente de même nom

Etape data

Sauvegarde de tables SAS sur disque

- L'instruction **libname** est une **instruction globale**
- Le lien entre la libref **dir** et le répertoire **d:\data\sasdata** demeure tout au long de la session SAS ou jusqu'à réinitialisation
- Si on quitte SAS puis qu'on le relance, l'instruction **libname** doit être soumise de nouveau

Etape data

Sauvegarde de tables SAS sur disque

- Le répertoire **work** est celui où sont stockées les tables SAS temporaires
- Il est automatiquement référencé par SAS et les tables qui y sont stockées sont effacées lorsqu'on quitte SAS
- On peut utiliser la fenêtre **Explorateur** de SAS pour examiner le contenu d'une table temporaire ou les variables qui y apparaissent
- Pour cela, double cliquer sur **libraries** puis sur **work**
- Pour les tables sauvegardées de façon permanente, double cliquer sur **libraries** puis sur la **libref** correspondante

Etape data

Exportation de table SAS

- Outre la lecture de données brutes, l'étape **data** permet également d'**exporter** des tables SAS. Par exemple, pour obtenir un fichier .csv à partir de la table bodyfat,

```
data _NULL_;  
  set bodyfat;  
  file 'd:\data\bodyfat.csv' dlm=',';  
  put pctfat sexe;  
  
run;
```

Etape data

L'alternative Proc Export

- Une alternative consiste à utiliser la **proc export** ou l'utilitaire d'exportation, accessible dans le menu File/Export Data...
- Par exemple, pour sauvegarder la table **bodyfat** au format **.csv**,

```
proc export data = bodyfat  
    outfile = 'd:\data\bodyfat2.csv'  
    dbcs = CSV;  
  
run;
```

Etape data

4. Création et Modification de variables

- L'étape data permet de créer des tables SAS mais permet également de **modifier** celles-ci de diverses façons
- L'instruction d'affectation = peut être utilisée pour créer de nouvelles variables mais également pour modifier des variables existantes

Etape data

Création et Modification de variables

- L'instruction

```
perte = poids_deb-poids_fin;
```

permet de **créer une nouvelle variable perte** définie comme différence entre poids de départ et dernier poids

- L'instruction

```
perte = perte*0.4536;
```

permet de convertir la perte de poids de livres en kilos

Etape data

Création et Modification de variables

- SAS dispose des opérateurs arithmétiques usuels +, -, /, * et ** (élévation à une puissance)
- SAS dispose également de diverses fonctions arithmétiques, mathématiques et statistiques,
- Certaines seront vues dans la suite du cours et en TD
- **Le résultat d'une opération arithmétique sur une valeur manquante est une valeur manquante**
- Lorsque c'est le cas un message d'avertissement apparaît dans le Log

Etape data

Création et Modification de variables

- Les valeurs manquantes des variables numériques sont représentées par un point (.) et une variable peut être fixée à une valeur manquante par une instruction de la forme

```
age = .;
```

Etape data

Création et Modification de variables

- Pour assigner une valeur à une variable caractère, la chaîne de caractère doit être entre guillemets, par exemple

```
equipe = "vert";
```

- Pour assigner une valeur manquante à une variable caractère, on procède de la façon suivante

```
equipe = '';
```

Etape data

Création et Modification de variables

- Pour modifier la valeur d'une variable pour certaines observations et pas les autres, ou pour effectuer des modifications différentes pour des groupes d'observations différents, l'instruction = peut être utilisée avec l'instruction **if then** :

```
recompense = 0;  
if perte > 8 then recompense = 1;
```

Etape data

Création et Modification de variables

- Au lieu d'une affectation suivie d'une condition, on aurait pu utiliser directement

```
if perte > 8 then recompense = 1;  
else recompense = 0;
```

- La condition dans l'instruction **if then** peut être une simple comparaison de deux valeurs. Les formes de comparaisons élémentaires sont listés dans le tableau suivant :

Etape data

Création et Modification de variables

Opérateur		Signification	Exemple
EQ	=	égal à	a=b
NE	≠	différent de	a ne b
LT	<	strict. Inférieur à	a < b
GT	>	strict. supérieur à	a gt b
GE	>=	supérieur ou égal à	a >= b
LE	<=	inférieur ou égal à	a le b

Etape data

Création et Modification de variables

- Des comparaisons peuvent être combinées pour former des conditions plus complexes à l'aide de **and** (&), **or** (|) et **not** :

```
if equipe = "rouge" and perte > 8  
then recompense =2;
```

- Dans certains cas, il est préférable de regrouper les conditions à l'aide de parenthèses pour clarifier les choses

Etape data

Création et Modification de variables

- Certaines conditions faisant intervenir **une seule variable** peuvent être simplifiées.
- Par exemple, les deux instructions suivantes sont équivalentes :

```
if age >= 18 and age <= 40 then  
    agegroupe = 1;
```

```
if 18 <= age <= 40 then agegroupe = 1;
```

- De même, des conditions de la forme

```
x = 1 or x = 3 or x = 5
```

peuvent être abrégées sous la forme

```
x in (1, 3, 5)
```


Etape data

Création et Modification de variables

- Lorsque les données contiennent des valeurs manquantes, il est important de les prendre en considération lors d'un recodage.
- Pour les comparaisons numériques, les valeurs manquantes sont considérées comme **inférieures à toute valeur**. Par exemple,

```
if age >= 18 then adulte = 1;  
    else adulte = 0;
```

Etape data

Création et Modification de variables

- a pour effet d'assigner la valeur 0 à **adulte** si **age** est manquant, alors qu'il serait plus approprié d'assigner une valeur manquante
- Pour ce faire on peut utiliser la fonction **missing** de la façon suivante

```
if missing(age) then adulte=.
```

Etape data

Création et Modification de variables

- Lorsqu'on effectue des comparaisons entre des **variables caractères**, faire attention aux points suivants :
- Capitalisation : Majuscules/Minuscules
- Présence d'espaces en début de chaîne

Etape data

Création et Modification de variables

- Un groupe d'instructions peut être exécuté **conditionnellement** en les plaçant entre une instruction **do** et une instruction **end** :

```
if poids_deb > 10 and poids_fin < 140 then do;  
    objectif = 1;  
    recompense = 1;  
    equipe = "bleue";  
end;
```

Etape data

Création et Modification de variables

- Lorsqu'une même opération doit être effectuée sur plusieurs variables, il est souvent pratique d'utiliser un **array** combiné avec une boucle **do**
- Par exemple, supposons qu'on dispose de 20 variables q1 à q20 pour lesquelles la modalité « non pertinent » a été codée par -1 et qu'on souhaite remplacer cette valeur par une valeur manquante

Etape data

Création et Modification de variables

- On peut procéder de la façon suivante

```
array qall{20} q1-q20;  
do i=1 to 20;  
    if qall{i}=-1 then qall{i}=.;  
end;
```

- L'instruction `array` définit un array (tableau) en spécifiant son nom, le nombre de variables qu'il contient et la liste des variables à inclure. **Toutes les variables doivent être du même type** : numérique ou caractère.

Etape data

Création et Modification de variables

- La boucle **do** exécute les instructions comprises entre **do** et **end** un nombre fixé de fois, l'indice de la variable étant incrémenté à chaque itération
- Un **array** est une façon concise de référer à un groupe de variables. En fait, il fournit un **alias** à chaque variable de façon à ce qu'on puisse y référer en spécifiant le nom de l'array et la position de la variable dans celui-ci
- **Un array n'existe que durant l'étape data dans laquelle il est défini**

Etape data

Suppression de variables

- Des variables peuvent être **supprimées** du tableau de données en utilisant les instructions **drop** ou **keep**
- L'instruction **drop** spécifie une liste de variables à exclure du tableau de données
- L'instruction **keep** spécifie les variables à conserver, toutes les autres étant exclues

Étape data

Suppression de variables

- Ainsi, l'instruction `drop x y z;` dans une étape data a pour résultat que le tableau de données ne contient pas les variables x, y et z
- L'instruction `keep x y z;` a pour résultat que le tableau de données ne contient que ces trois variables

Etape data

Suppression d'observations

- Il peut être nécessaire de supprimer des **observations** du tableau de données parce qu'elles contiennent des erreurs ou qu'on souhaite effectuer l'analyse sur un sous-ensemble de données
- La suppression d'observations erronées est en générale effectuée en combinant une instruction **if then** avec une instruction **delete**

Etape data

Suppression d'observations

```
if perte > poids_deb then delete;
```

- Dans ce type de situation, il peut être utile d'afficher un message fournissant des informations sur l'observation erronée

```
if perte > poids_deb then do;  
    put "Erreur dans les données de  
    poids " idno = poids_deb = perte =;  
delete;  
end;
```

Etape data

Suppression d'observations

- L'instruction `put` affiche le texte entre guillemets et les valeurs des variables dans la fenêtre log.
- Si on souhaite analyser un sous-ensemble des données, il est souvent plus pratique de **créer un nouveau tableau de données** contenant uniquement les observations pertinentes.
- Ceci peut être effectué à l'aide des instructions `if` ou `where`.

Etape data

Suppression d'observations

- L'utilisation de `if` à cet effet consiste simplement à faire suivre le mot-clef de la condition requise
- Seules les observations pour lesquelles la condition est vérifiée sont incluses dans le tableau de données créé.

```
data hommes;  
    set wgtclub2;  
    if sexe= "H";  
run;
```

Etape data

Suppression d'observations

- L'instruction `where sexe= "H"`; peut être utilisée pour obtenir le même résultat.
- La distinction entre les instructions `if` et `where` peut être ignorée dans la plupart des cas, si ce n'est que `where` peut également être utilisée dans les étapes **proc**.

Etape data

Concaténation de tableaux

- Deux tables SAS ou davantage peuvent être combinées en une seule à l'aide de l'instruction `set`. Par exemple,

```
data sondage;  
    set hommes femmes;  
run;
```

- Cela fournit également une façon simple d'augmenter une table SAS avec de nouvelles observations.

Etape data

Concaténation de tableaux

- Il suffit de lire les nouvelles données dans une table SAS, puis de les combiner aux anciennes à l'aide de `set`. Par exemple,

```
data sondage;  
    set sondage nouvobs;  
run;
```


Etape data

Merge (Jointure)

- Dans le cadre d'une même étude, on peut disposer de plusieurs sources de données.
- Par exemple, on peut souhaiter combiner des données démographiques avec des résultats de tests médicaux pour les mêmes personnes.
- Dans un tel cas, on commence par lire les données dans des tables SAS différentes puis on les combine à l'aide d'un **merge** en utilisant l'**identifiant** des individus comme **clef**

Etape data

Merge (Jointure)

- Par exemple,

```
proc sort data = demographics;  
    by idnumber;  
proc sort data = labtests;  
    by idnumber;  
data combined;  
    merge demographics (in=indem) labtests  
        (in=inlab);  
    by idnumber;  
    if indem and inlab;  
run;
```

Etape data

Merge (Jointure)

- On commence par trier les deux tables SAS selon l'identifiant `idnumber`. Cette variable doit être de même type et de même longueur dans les deux tables
- L'instruction `merge` de l'étape data spécifie les tables à combiner. L'option `in` entre parenthèses après le nom de table a pour effet de créer une variable temporaire indiquant si la table correspondante a contribué à la table obtenue par `merge` pour chaque observation

Etape data

Merge (Jointure)

- L'instruction **by** spécifie la variable-clef, selon laquelle le **merge** est effectué
- L'instruction **if** permet de spécifier que seules les observations pour lesquelles les données démographiques et de laboratoires sont toutes deux présentes doivent être incluses dans la table finale
- Sans cette instruction, la table finale pourrait contenir des observations incomplètes

Etape data

Merge (Jointure)

- Une alternative peut consister à afficher des messages dans la fenêtre log pour les observations correspondantes; par exemple,

```
if not indem then put idnumber 'no  
    demographics';
```

```
if not inlab then put idnumber 'no lab  
    results';
```

- Notons que des **merge** plus compliqués peuvent être effectués, notamment dans des situations où il y a des correspondances multiples entre les données des tables SAS considérées

Etape data

Fonctionnement

- Il est utile, en plus de connaître les instructions pouvant être utilisées lors de l'étape **data**, de comprendre comme celle-ci fonctionne.
- Les instructions qui composent l'étape data forment une suite selon l'ordre dans lequel elles apparaissent. La suite commence avec l'instruction data et s'achève à la fin de l'étape data et est exécutée de façon répétitive jusqu'à épuisement de la source des données.

Etape data

Fonctionnement

- En partant de l'instruction **data**, une étape data typique lit des données suite à une instruction **input** ou **set** et les combine pour construire une observation.
- Les instructions suivantes sont alors appliquées à cette observation. Les données correspondant à l'observation peuvent être modifiées ou augmentées au cours du processus.
- A la fin de l'étape data l'observation est rajoutée au tableau de données créé.

Etape data

Fonctionnement

- Le processus est alors recommencé à partir de l'instruction data et itéré jusqu'à épuisement des données.
- L'étape data est alors achevée et l'exécution du programme passe à l'étape suivante.
- En fait, l'étape data consiste en une boucle d'instructions exécutée de façon répétitive jusqu'à ce que toutes les données soient traitées.

Etape data

Fonctionnement

- Le numéro de l'itération est sauvegardé dans la variable SAS `_n_` mais cette dernière n'est pas rajoutée au tableau de données.
- Le point où SAS rajoute une observation au tableau de données peut être contrôlé à l'aide de l'instruction `output`.
- Lorsqu'une étape data inclut une ou plusieurs instructions `output`, une observation est rajoutée au tableau de données à chaque fois qu'une instruction `output` est exécutée, mais pas à la fin de l'étape data.
- De cette façon, les données lues peuvent être utilisées pour former **plusieurs** observations.