# PROJET AVEC LE LOGICIEL SAS

# INTRODUCTION AU LOGICIEL SAS

# Sommaire

# 1 Introduction

Le Logiciel statistique SAS permet le traitement de données à l'aide programmes écrits en langage SAS. Ces données peuvent être aussi bien des nombres que des chaînes de caractères alphanumériques. L'utilisateur doit créer :

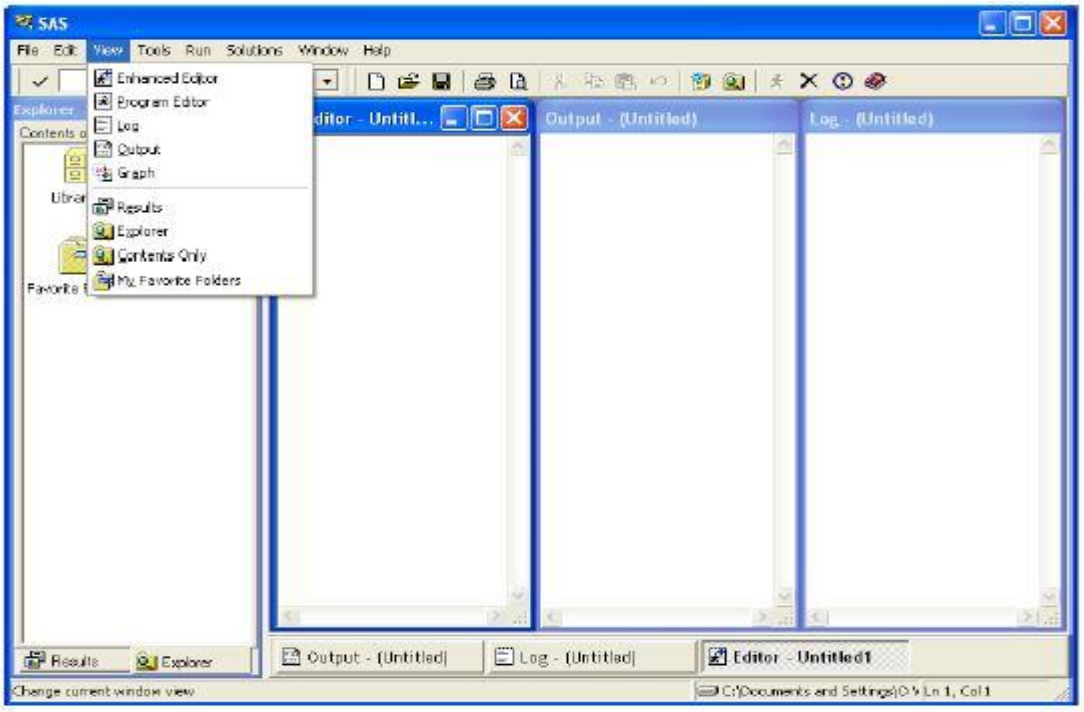
- Un fichier contenant les instructions du programme en langage SAS, à partir de l'éditeur de SAS.
- Éventuellement un fichier contenant les données alphanumériques à analyser, à partir d'un éditeur de texte.

La version SAS que nous utiliserons est la version 9. Il faut commencer par lancer SAS : **menu Démarrer → Programmes → SAS → SAS xx (Français)**. Plusieurs fenêtres apparaissent alors à l'écran, elles peuvent être déplacées à votre convenance.

# 2 L'environnement SAS

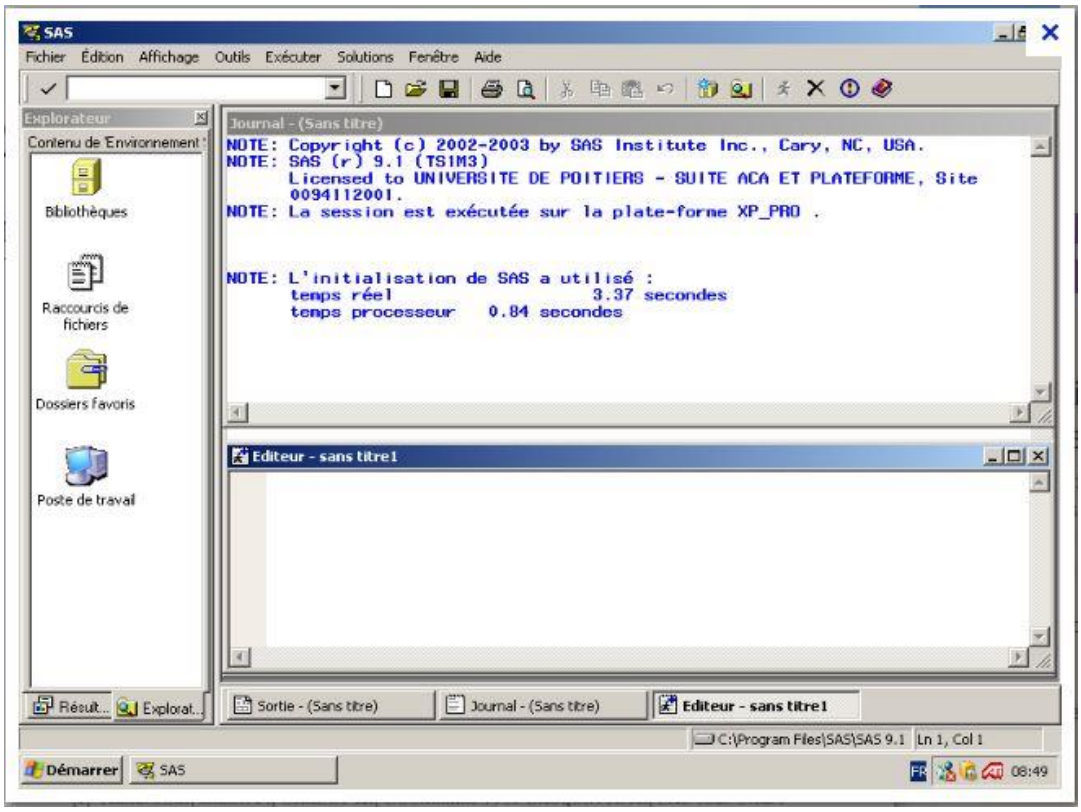
## 2.1 Les fenêtres SAS

Cinq fenêtres s'ouvrent au démarrage de SAS : Editeur (Program Editor), Sorties (Output), Journal (Log), Résultats (Results), Explorateur (Explorer) :




Présentation des cinq principales fenêtres.

- **SAS : EDITEUR** est un éditeur de texte classique pour entrer et modifier les programmes SAS avant d'en demander l'exécution. *Le contenu de cette fenêtre est entièrement généré par l'utilisateur.*

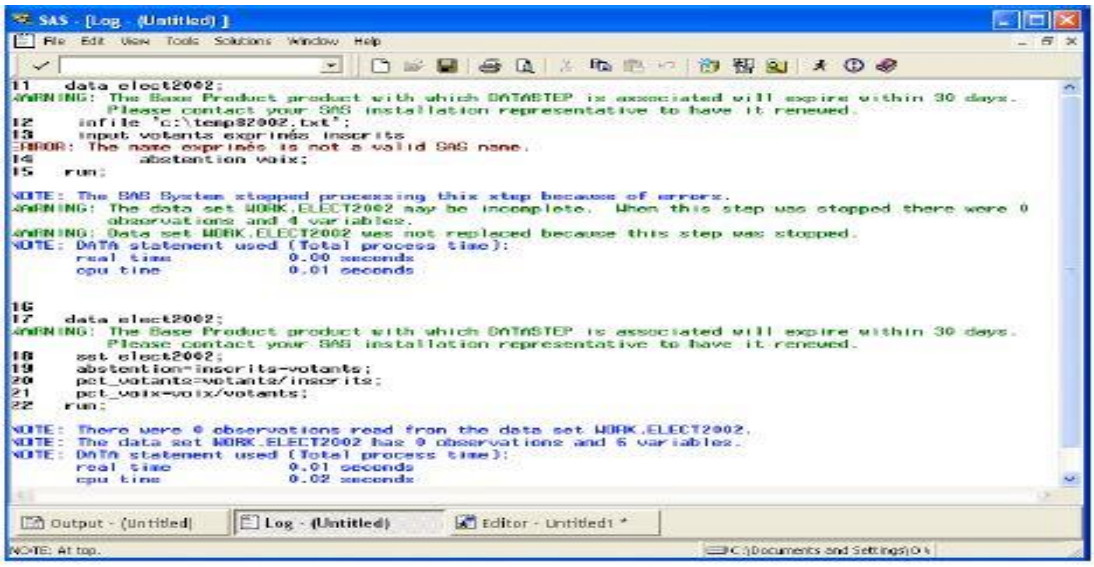



**SAS : JOURNAL**

La fenêtre **Journal** sert à l'affichage des commandes soumises ainsi qu'aux commentaires générés par le programme SAS. *Le contenu de cette fenêtre est également généré par* SAS. Elle contient notamment des **messages d'exécution** tels que :

- **NOTE** : Ce sont des messages d'information. Ils indiquent généralement que le programme s'est exécuté correctement et/ou fournissent des détails sur les actions entreprises par SAS.
- **AVERTISSEMENT (WARNING)** : Ces messages signalent un comportement inattendu ou potentiellement problématique. Le programme peut s'exécuter malgré tout, mais les résultats doivent être vérifiés attentivement.
- **ERREUR (ERROR)** : Ces messages indiquent qu'une erreur s'est produite. Le programme ne peut pas s'exécuter correctement, ou une étape a été ignorée. Ces erreurs doivent être corrigées pour garantir l'exactitude des résultats.

  De plus, SAS recopie en **noir** le code soumis par l'utilisateur, **précédé de numéros de lignes**. Ces numéros sont **réinitialisés à chaque nouvelle exécution** d'un programme dans la session en cours, ce qui permet de repérer facilement les instructions lors de la lecture du Journal.

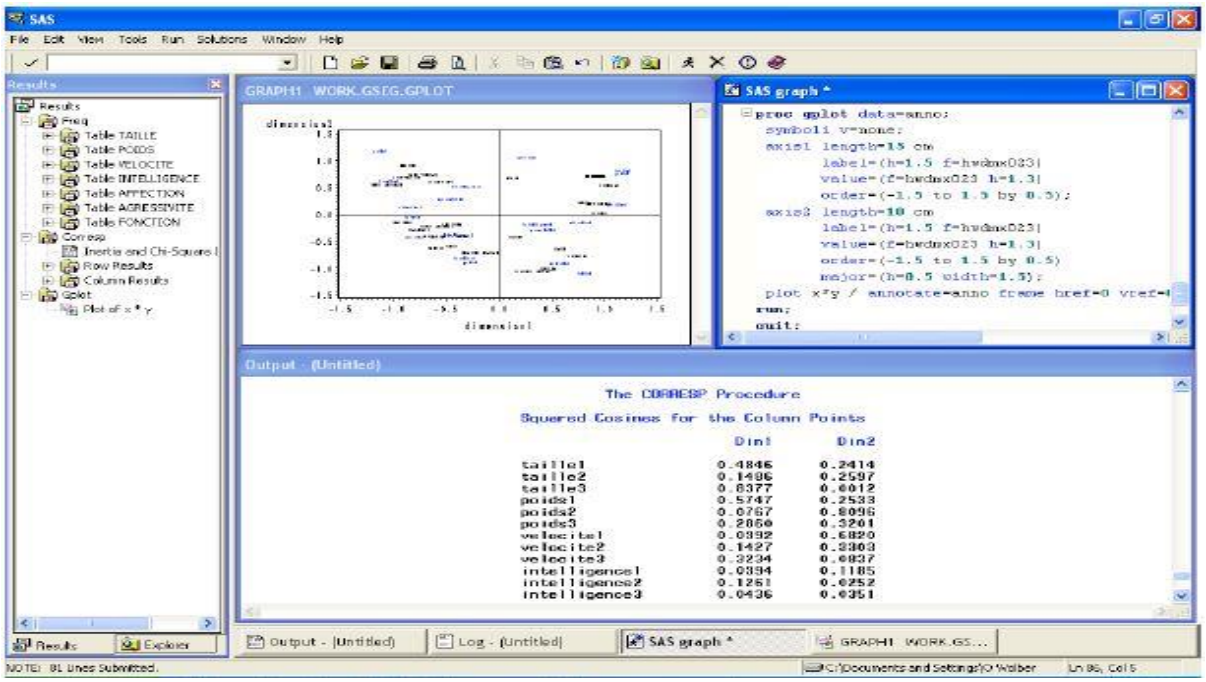



- **SAS : SORTIE, RESULTATS, GRAPH**

  La fenêtre **Sortie (Output)** affiche les **résultats textuels** générés après la compilation et l'exécution des programmes SAS. On y retrouve notamment les **listings**, **tableaux de résultats**, et **sorties statistiques** produits par les procédures.

  Elle est complétée par deux autres fenêtres importantes :

  - La **fenêtre Graph** : elle est dédiée à l'affichage des **graphiques** produits par les procédures de visualisation ou d'analyse graphique.
  - La **fenêtre Résultats (Results)** : elle propose une **arborescence hiérarchique** des différentes sorties générées par le programme. Elle permet de naviguer facilement entre les différentes parties du rapport ou des résultats, notamment lorsqu'un programme produit plusieurs tableaux ou graphiques.



- **SAS : EXPLORATEUR** permet d'accéder rapidement à vos fichiers. Vous pouvez voir et gérer vos fichiers SAS et créer des raccourcis vers des fichiers autres que SAS. On peut utiliser cette fenêtre pour créer de nouveaux répertoires et fichiers SAS, et pour déplacer, copier, supprimer des fichiers et en particulier de visualiser les tables SAS qui ont été créées.


## 2.2 Exécution d'un programme

Les menus disponibles sont variables en fonction de la fenêtre active. Le menu "RUN" n'est disponible que lorsque la fenêtre Editeur est active. La barre d'outils regroupe les icônes standards (nouveau, ouvrir, enregistrer, couper, copier, coller, annuler, imprimer, aperçu avant impression). Quand la fenêtre Editeur est active, l'icône "Soumettre/Submit" s'ajoute à la barre d'outils :

Il permet l'exécution du programme qui est écrit dans la fenêtre active ; si seule une partie du programme est sélectionnée, alors seul ce code sera soumis et exécuté par SAS.

Une fois votre programme mis en forme, il doit être exécuté en cliquant sur « **exécuter/soumettre** » dans le menu.

- **Si le programme est correct**, la fenêtre SORTIE apparaît au premier plan et contient les résultats des procédures choisies. La fenêtre SORTIE n'est pas nettoyée entre les procédures. Il est recommandé d'effacer le contenu de la fenêtre SORTIE par le menu ou en utilisant la commande « **Édition → Effacer tout** » entre les exécutions d'un même programme.
- **Si le programme est incorrect**, la fenêtre EDITEUR reste au premier plan et la fenêtre JOURNAL contient le compte rendu de l'exécution et la description des erreurs. La fenêtre JOURNAL n'est également pas nettoyée entre les procédures, il est recommandé d'effacer son contenu.

## 2.3 Les librairies

Une librairie SAS est un répertoire qui contient des fichiers SAS. Les librairies permettent de stocker les fichiers de données (tableaux SAS). Chaque librairie SAS est liée à un répertoire sur le disque dur ou sur le serveur utilisé. Elles permettent donc d'éviter la réécriture du chemin à chaque utilisation d'un nom de tableau SAS en indiquant avant chaque nom de tableau la librairie sous la forme suivante : **nomlib.nomdelatableSAS.**

Par exemple, si la librairie « **nomlib** » est associée au répertoire C:\mes documents\xxx, alors le tableau SAS **nomdelatableSAS** sera désigné par « **C:\mes\documents\xxx\ nomdelatableSAS.sas7.bdat** » dans Windows et par « **nomlib.nomdelatableSAS**» dans SAS.

Il y a d'origine deux bibliothèques sous **SAS**. Une temporaire (**WORK**) et une permanente (**SASUSER**). La bibliothèque **WORK** détruit les fichiers qu'elle contient dès que vous quittez **SAS**.

# 3 SAS Studio ou SAS en ligne

## 3.1 Definition

**SAS Studio** est une **interface web** développée par SAS (Statistical Analysis System) qui permet d'écrire, exécuter et visualiser du code SAS **directement dans un navigateur Internet**, sans avoir besoin d'installer un logiciel sur son poste.

## 3.2 Caractéristiques principals

- **Accessible via un navigateur** (Chrome, Firefox, etc.)
- Interface conviviale avec:
  - un éditeur de code SAS,
  - une fenêtre de résultats (log, output, graphiques),
  - une bibliothèque d'exemples et d'assistants pour générer du code.
- Permet la **gestion de données**, la **programmation**, l'**analyse statistique**, la **visualisation**, etc.
- Peut être utilisé dans des environnements **locaux** (installé sur un serveur) ou dans le **cloud** (via SAS OnDemand for Academics, par exemple).

## 3.3 Pourquoi l'utiliser en formation?

- Prise en main rapide pour les étudiants débutants.
- Aucun besoin d'installation complexe.
- Favorise la découverte du langage SAS de manière interactive.
- Utilisé dans de nombreux cursus en statistiques, data science ou ingénierie des données.

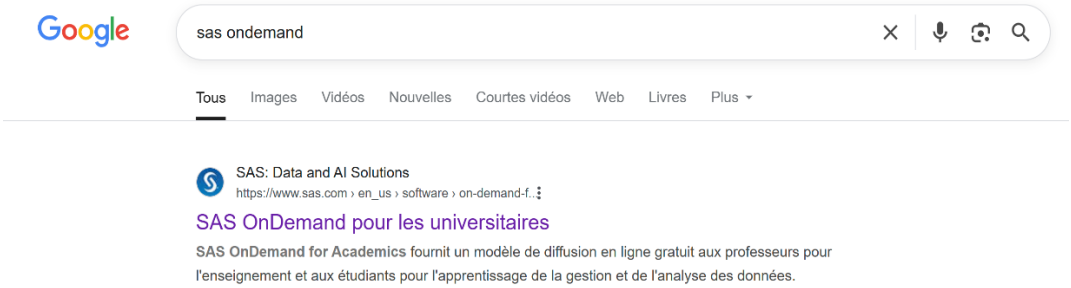
## 3.4 Creation de profile sas

Pour accéder à SAS studio, il vous faut un avoir un profile SAS et pour cela, il est nécessaire d'avoir une adresse email valide.

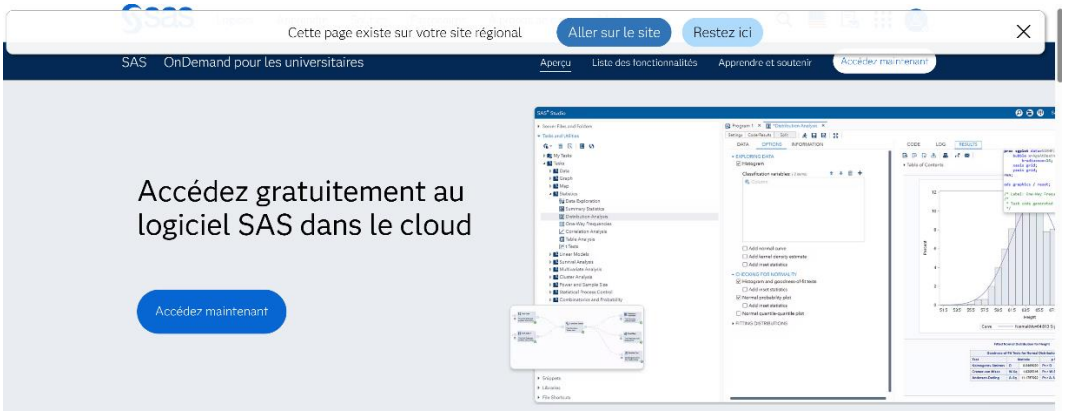
**Étapes:**

1. Va sur le site : 👉 https://welcome.oda.sas.com
2. Clique sur **"Sign Up"** si tu n'as pas encore de compte.
3. Crée un compte SAS avec ton adresse email (de préférence académique).
4. Une fois inscrit, connecte-toi sur : 👉 https://oda.sas.com
5. Choisis **"SAS Studio"** dans la liste des applications disponibles.

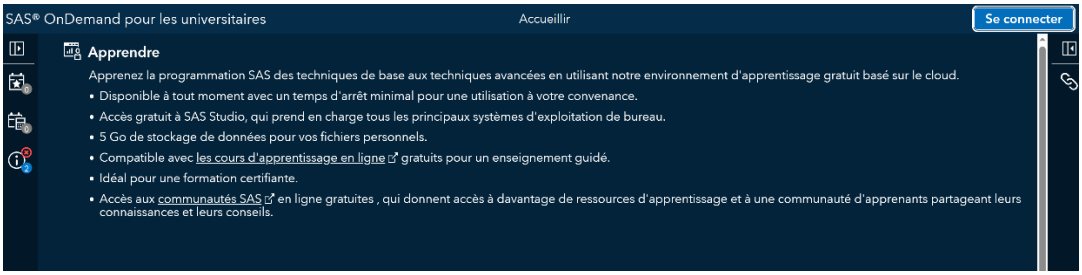
Aller sur google :



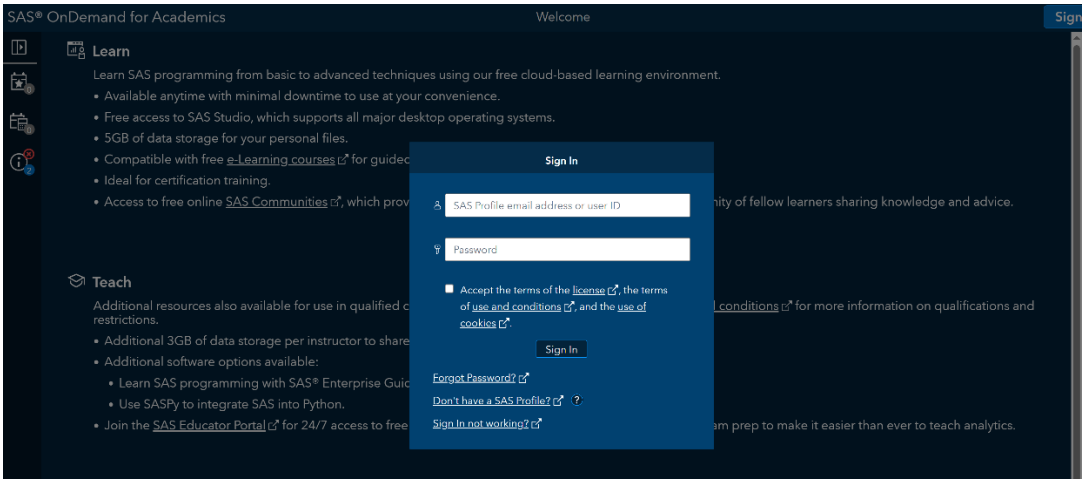

Cliquer sur SAS **OnDemande** pour les universitaires et sur « Accéder maintenant »:



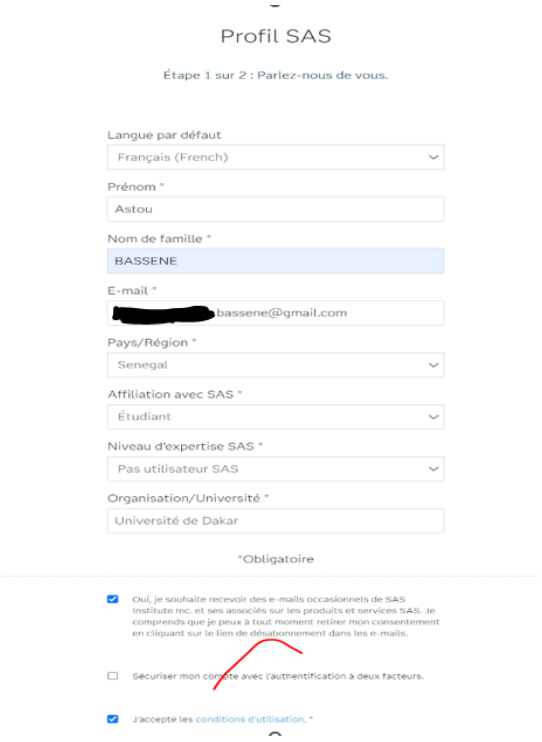

Cliquer sur « se connecter »




Cliquer sur « Don't have a SAS Profile »

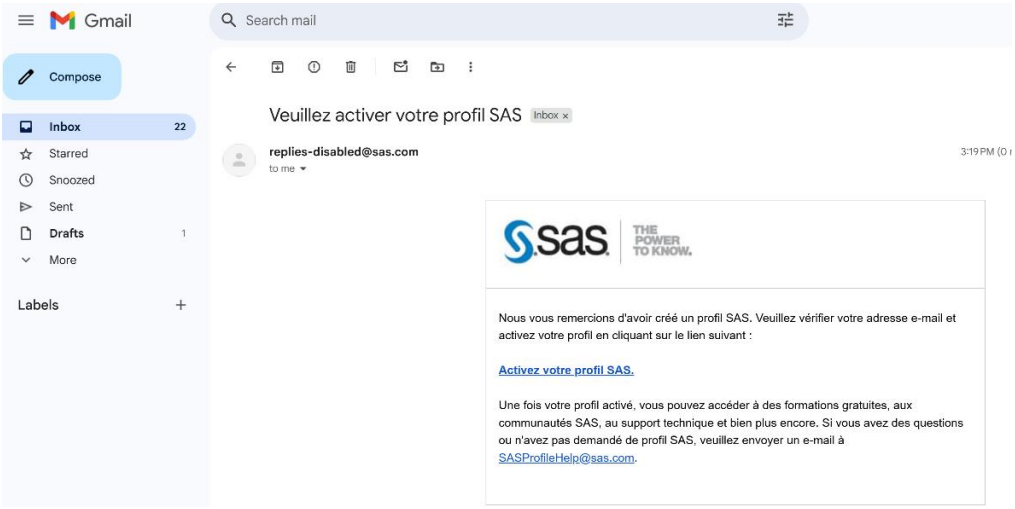


Remplissez le formulaire qui va s'afficher :

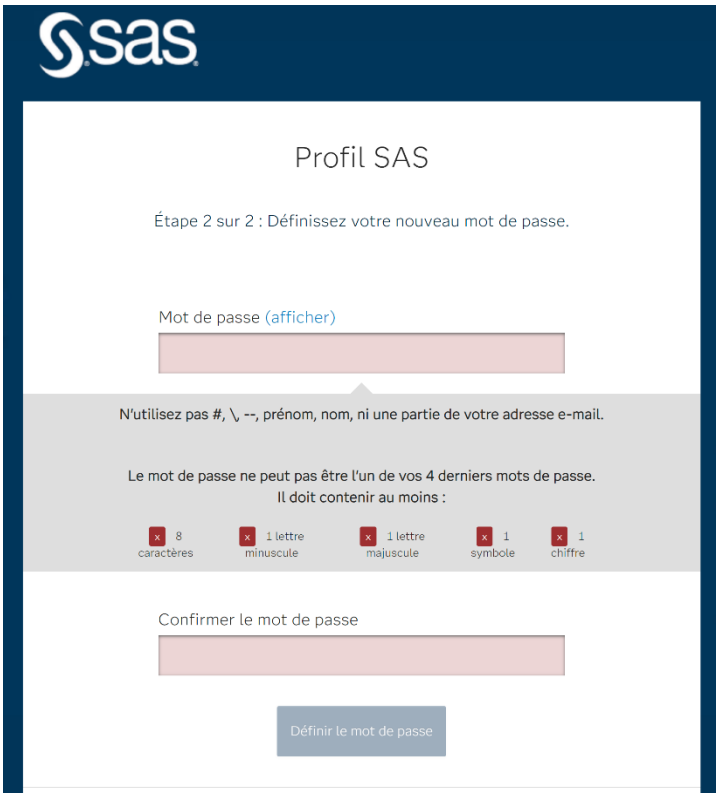



Voici le message qui va s'afficher :

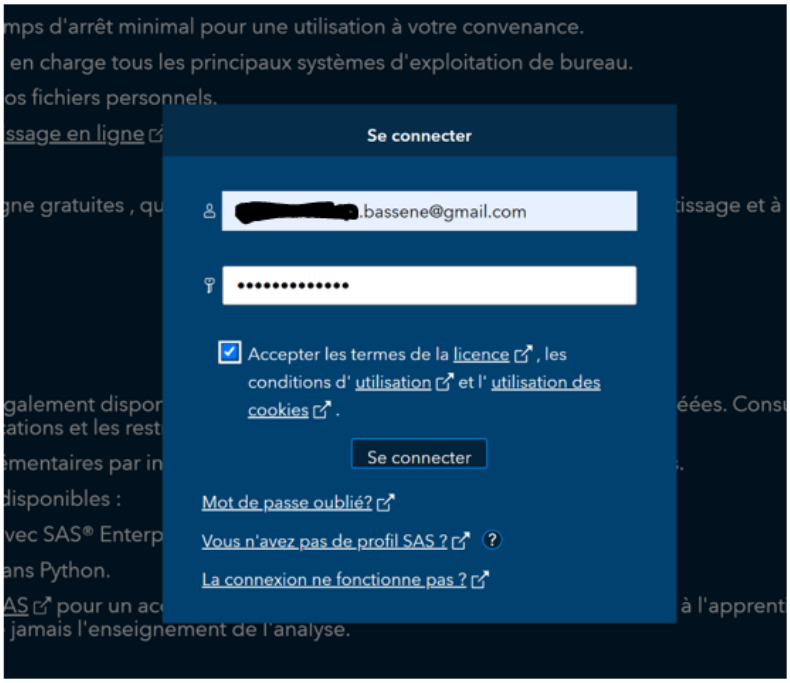
Voici le mail que vous allez recevoir :



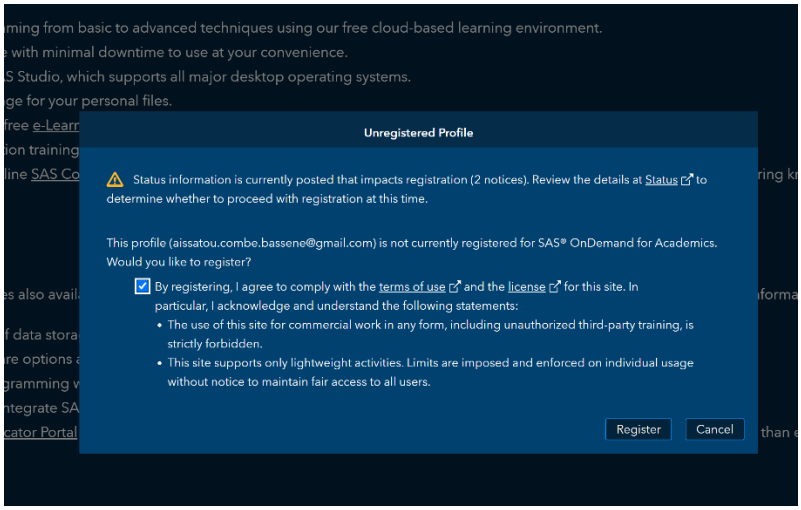

En cliquant sur « activer votre profile sas », vous allez tomber sur cette page :



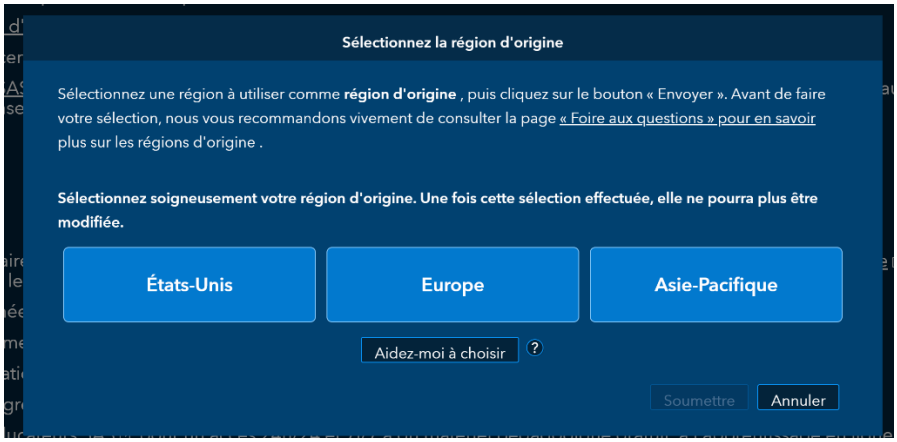

Après avoir créé votre mot de passe que je vous recommande de bien retenir, reconnecter à nouveau :



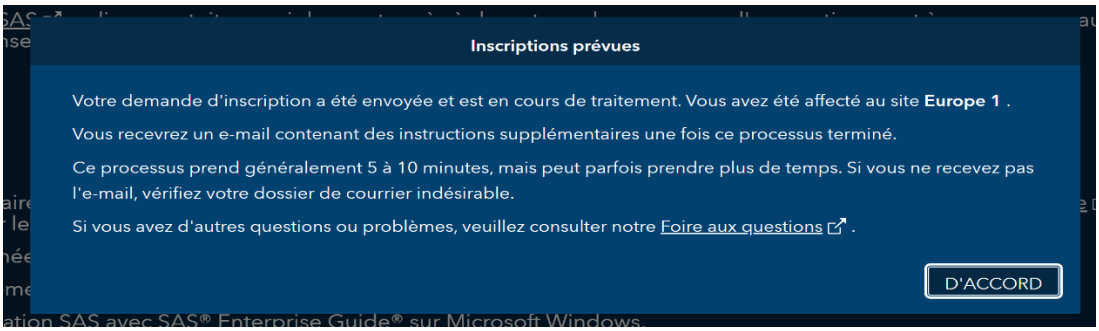

Cliquer sur « se connecter » et une nouvelle page s'affichera :



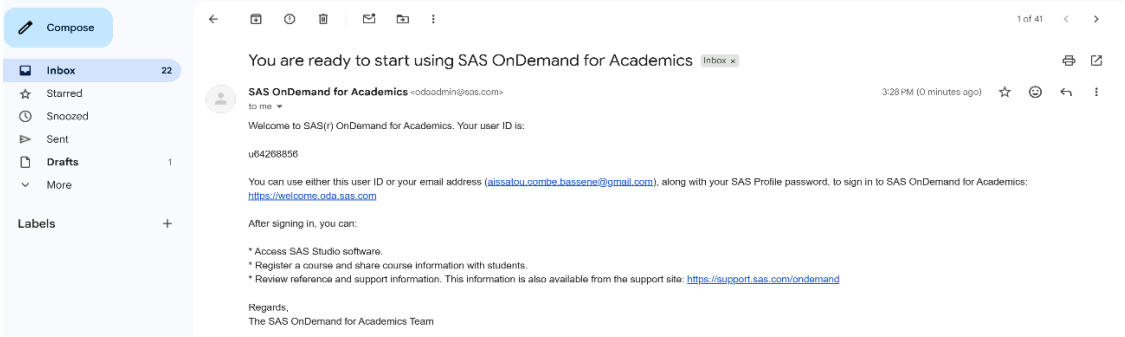

Après avoir cliqué sur « Register » la page ci-dessous s'affichera : sélectionner « Europe »



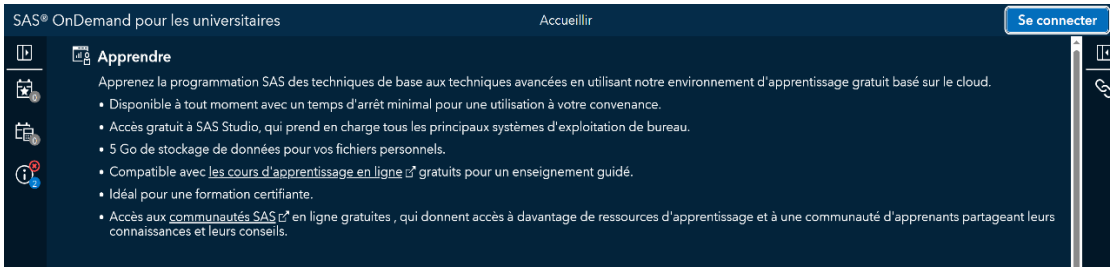
Cliquer sur d'accord pour finaliser votre profile :

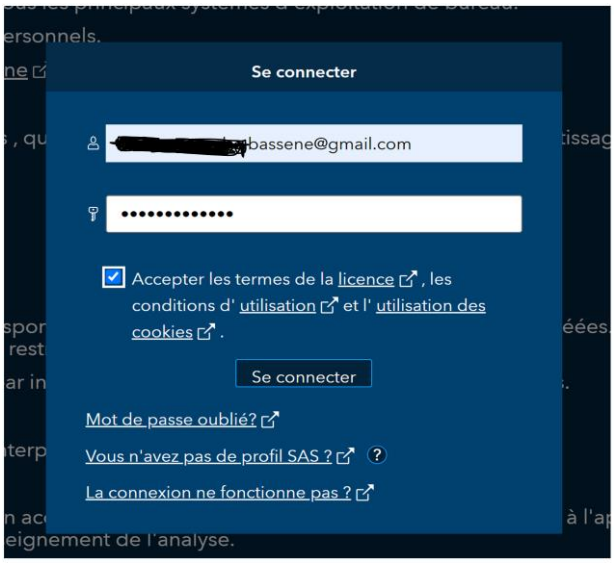


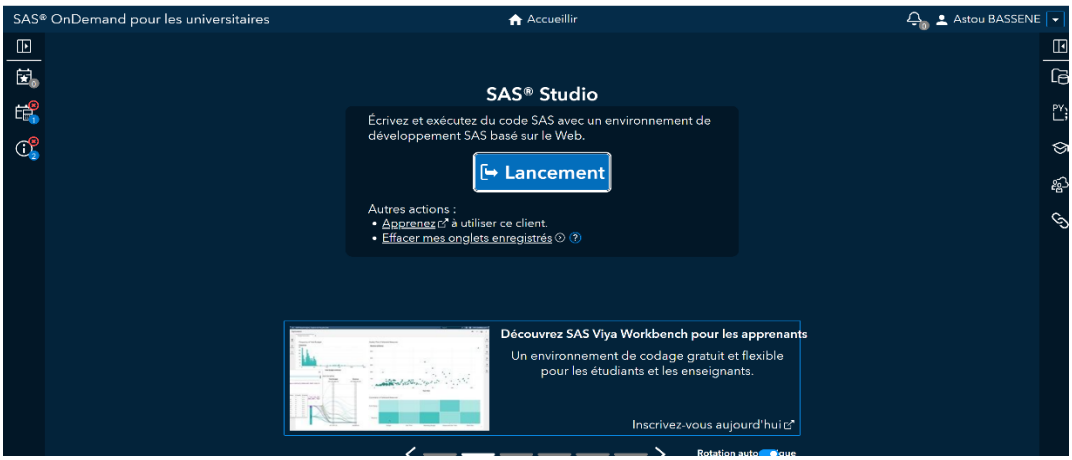

Un mail de confirmation vous sera envoyé :



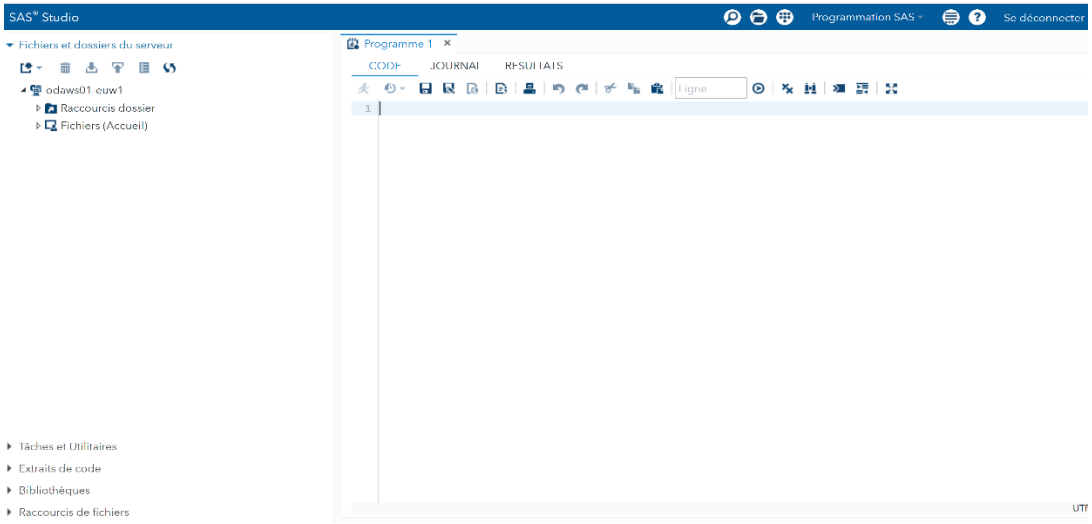

Vous êtes enfin prêt à utiliser SAS studio :






Cliquer sur « Lancement » :




## 4 Mise au point des programmes

La partie principale d'un programme SAS est composée d'étapes. Il existe deux types d'étapes :

- **L'étape data** (étape données) qui commence par le mot clé « DATA ». Il s'agit de la création d'un tableau de données. Ces données sont soit incluses dans le programme SAS, soit contenues dans un fichier texte (ascii) provenant d'un éditeur, d'un autre logiciel de statistique ou encore d'un gestionnaire de base de données.
- **L'étape proc** (étapes procédures) qui commence par le mot clé « PROC ». Il s'agit d'appliquer une procédure à un tableau de données préalablement crée par SAS dans une étape DATA.

Chaque étape est elle-même constituée d'une suite d'instructions. Une instruction SAS commence habituellement par un mot-clé SAS identifiant le type d'instruction : DATA, PROC, INPUT, … Le reste de l'instruction comprend des informations complémentaires décrivant la tâche à accomplir.

## 4.1 Etape DATA

Etape DATA : Afin de pouvoir traiter des données avec SAS, il faut les mettre sous un certain format, interne à SAS. SAS possède en effet son propre système de gestion de bases de données (étape DATA). L'objet créé dans une Etape DATA s'appelle une TABLE SAS.

Toute étape DATA débute par l'instruction DATA nom-de-table où nom-de-table est le nom de la table SAS créée. L'étape DATA se termine par l'instruction RUN.

Syntaxe :

```
DATA nom-de-table;
instructions;
RUN;
```

## 4.2 Etape PROC

Etape PROC : Un programme SAS est un enchaînement de procédures qui réalise un traitement sur les TABLES disponibles. Voici quelques exemples basiques :

- PRINT : affiche le contenu d'une table dans l'Output.
- SORT : tri d'une TABLE selon une variable
- MEANS, UNIVARIATE : résumés numériques pour variables quantitatives
- FREQ : résumés numériques pour variables qualitatives
- CORR : calcule des corrélations entre variables quantitatives
- PLOT, GPLOT : graphiques pour variables quantitatives
- CHART, GCHART : graphiques pour variables qualitatives
- BOXPLOT : boîtes à moustaches.

Syntaxe :

```
PROC nom-de-procedure <options>;
instructions;
RUN;
```

## 4.3 Sauvegarde d'un programme

Tant que l'on ne quitte pas le logiciel SAS, le programme en cours est conservé et peut être rappelé mais il sera effacé à la sortie de SAS. Il est donc nécessaire de sauvegarder les programmes que l'on veut conserver. Afin de sauver un programme, sélectionner la fenêtre EDITEUR, puis cliquer sur : ***Fichier → Enregistrer sous.***

## 4.4 Les tableaux SAS

Les bases de données ou tableaux SAS ont un format rectangulaire. Chaque ligne du tableau constitue une observation. Une observation est un ensemble de valeurs concernant un même individu. Les valeurs contenues dans une colonne du tableau constituent une variable. Une variable est un ensemble de valeurs concernant une même caractéristique.

SAS identifie les variables par leur nom. Un nom de variable SAS peut comporter de 1 à 32 caractères et doit commencer par une lettre ou le caractère souligné (_). Les autres caractères doivent être des lettres, des chiffres ou le caractère souligné. Un nom de variable ne peut pas comporter de caractères blancs (espace).

**Exemple : Création d'un fichier de données et procédure simple**

L'exemple suivant consiste à créer une petite base de données concernant des élèves (`data step`) puis à imprimer ces données (`proc step`).

```
data exemple1;
input nom $ sexe $ age taille poids ;
cards;
Marine F 21 169 60.0
Denis M 18 165 45.0
Antoine M 20 175 63.0
Sarah F 19 170 50.0
Mélanie F 22 172 55.0
Jean M 21 170 56.2
Pierre M 20 172 63.2
Anaïs F 18 162 44.6
;
run;

proc print ;
run;
```

Les informations concernant chaque élève - *nom, âge, sexe, taille, poids* - forment une **observation.** Un ensemble de valeurs concernant une même caractéristique**,** telle que l*e poids d'une personne*, forme une **variable**.

**Remarques :**

- Chaque instruction SAS doit obligatoirement se terminer par le caractère point-virgule **(;).**
- L'instruction qui permet d'associer un tableau de nom **nomtable** à une librairie nommée **nomlib** est :

```
DATA nomlib.nomtable;
Run ;
```

- L'instruction qui permet d'avoir le contenu d'une librairie est :

```
PROC DATASETS
lib=nomlib;
run ;
```

- Vous pouvez visualiser, dans la fenêtre SORTIE, un fichier de données en tapant dans la fenêtre EDITEUR :

```
PROC PRINT DATA= nomtable ;
RUN ;
```

# 5 Gestion des données

La gestion de données se fait principalement à l'aide des **data steps**. Elles consistent à créer ou transformer un tableau de données SAS. On peut créer des tables SAS permanentes ou temporaires. Les tables temporaires sont effacées lorsqu'on quitte la session. Par défaut, les tables sont créées dans la bibliothèque de travail WORK : elles sont temporaires.
Une TABLE est structurée en lignes (observations) et colonnes (variables) :

- Les observations sont les lignes d'une TABLE SAS. Chaque observation correspond à une unité statistique ou un individu. Chaque observation porte un numéro.
- Les variables sont les colonnes d'une TABLE SAS. Chaque variable correspond à une information connue sur les observations.

Toutes les variables ont trois attributs : nom, type (numérique ou caractères), longueur.

**Remarque** :

- La longueur maximale du nom est de 32 caractères ;
- SAS ne distingue pas les minuscules et les majuscules.

**Variables numériques :** SAS reconnaît les nombres sous les formes suivantes :

- Entiers : 1234 ;
- Entiers relatifs : -5 ;
- Réels : 5.85 ;
- Notation scientifique : 5.4E-1 ;
- Dates : '24aug90'd. Une date est ainsi définie le plus souvent comme numérique. Mais SAS stocke les dates comme un nombre de jours à partir du 1/1/1960.

## 5.1 Lecture de Données

Il existe 3 façons de créer une TABLE SAS à partir de données :

- Ecriture à l'intérieur du programme avec des données délimitées,
- Importation depuis un fichier externe avec l'instruction INFILE
- Utilisation d'une table préexistante dans un fichier lisible par SAS.

Dans les deux premiers cas, une déclaration des variables est obligatoire à l'aide de l'instruction INPUT :
Syntaxe :

```
DATA nom-de-table;
...
INPUT VAR1 VAR2 ...;
...
RUN;
```

### 5.1.1 Lecture de Données dans le code

Les données peuvent être écrites dans le programme après l'instruction DATALINES ou CARDS

```
Data exemple;
Input nom $ sexe $ age taille poids note ville $ group $ ;
cards;
Marine F 21 169 60.0 16 Paris A
Denis M 18 165 45.0 13 Paris A
Antoine M 20 175 63.0 12 Lille B
Sarah F 19 170 50.0 8 Lens A
Mélanie F 22 172 55.0 6 Lens B
Jean M 21 170 56.2 13 Lille C
Pierre M 20 172 63.2 19 Lille C
;
run;

PROC PRINT DATA=Exemple;
RUN;
```

- Une instruction **DATA** demande à SAS de lire des données et de les organiser en fichier SAS. Cette instruction comprend le mot-clé **DATA** et un nom de fichier choisi par l'utilisateur, dans ce cas-ci ***exemple*.**
- L'instruction **INPUT** fournit l'information à SAS pour organiser les données en fichier SAS. Elle débute par le mot-clé INPUT suivi de la liste des noms de variables, dans ce cas-ci *nom, sexe, age, taille et poids et note*. Les variables *nom, sexe, ville et groupe* sont suivies de $, pour indiquer que ces variables contiennent des valeurs alphabétiques. Les autres variables sont numériques. Cette forme de l'instruction INPUT suppose que **les valeurs manquantes sont indiquées par un point.**
- L'instruction **CARDS** indique que des lignes de données suivent, un **point-virgule** indique la fin des lignes de données**.**
- L'instruction **RUN** demande à SAS d'exécuter les instructions précédentes. Bien que SAS n'exige pas toujours l'instruction RUN après les lignes de données et le point-virgule, il est recommandé d'inclure l'instruction RUN dans cette section de vos programmes.
- L'instruction **PROC PRINT** demande à SAS d'imprimer les données.

### 5.1.2 Données délimitées

On soumet ensuite le code de l'étape DATA : la TABLE Exemple est créée dans la librairie WORK (temporaire). On peut la visualiser à partir de l'Explorateur SAS ou bien dans la fenêtre OUTPUT en utilisant la procédure PRINT.

```
Proc print data=Exemple;
run ;
```

On peut utiliser d'autres séparateurs que " ". Pour cela, il faut rajouter l'instruction INFILE avec des options :

- FIRSTOBS=num : numéro num de la première ligne lue.
- OBS=num : numéro num de la dernière ligne lue.
- DLM="." : indique le caractère séparateur : "." ou ";" ou "&"

Par défaut, le séparateur utilisé avec l'instruction INFILE est ",".

Exemple :

```
DATA Exemple;
INFILE datalines DLM="&";
INPUT nom$ age sexe$;
DATALINES;
Chris&36&M
Jane&21&F
Tom&30&M
Joe&49&M
Jerry&28&M
;
RUN;
```

**Gestion des données manquantes sur une ligne**. Le code suivant produit la TABLE Exemple2 :

```
DATA exemple;
INFILE datalines DLM="&";
INPUT nom$ age sexe$;
DATALINES;
Bob&m
Marie&25&f
Yves&45&m
;
RUN;
```

L'option DSD de l'instruction INFILE donne priorité au séparateur : deux séparateurs successifs seront interprétés comme une donnée manquante :

```
DATA exemple;
INFILE datalines DLM="&" DSD;
INPUT nom$ age sexe$;
DATALINES;
Bob&&m
Marie&25&f
Yves&45&m
;
RUN;
```

### 5.1.3 Les informats ou formats de lecture

Les informats sont des formats de lecture des données (**Attention ! !** à ne pas confondre avec les formats d'écriture cf. plus loin). Les informats sont spécifiées dans l'instruction INPUT et indiquent à SAS comment les données doivent être lues dans le programme ou dans le fichier externe :

- Variable numérique : Var :x. pour variables entières et Var :x.q pour variables décimales où x indique le nombre de caractères à lire (point compris) et q le nombre de décimales.
- Variable alpha-numérique : Var :$x. où x indique le nombre de caractères (8 par défaut).
- Variable de date : YYMMDD.

**Remarque** : Le " :" donne la priorité au délimiteur sur l'informat le long d'une ligne : cela signifie que lorsque SAS lit une ligne il peut passer d'une variable à l'autre sans forcément avoir atteint le nombre total de caractères déclarés.

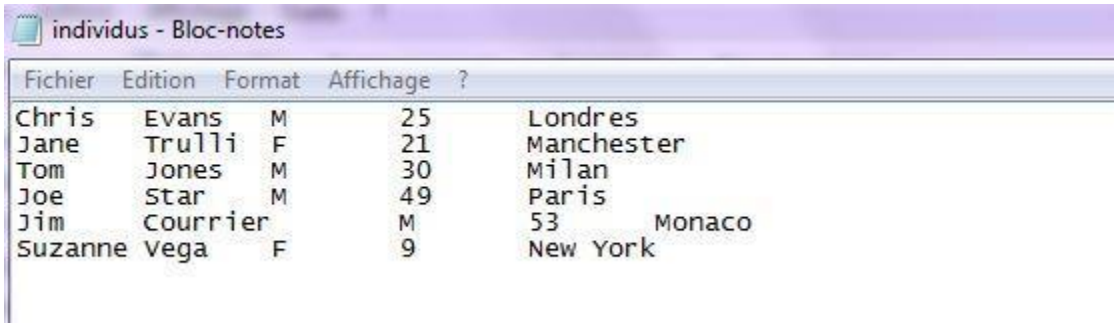
```
DATA Exemple;
INFILE datalines DLM="&";
INPUT nom$ age sexe$;
DATALINES;
Chris Evans&25&m
JaneTrulli&21&f
JoeStar&49&m
;
RUN;
```

Par défaut, seuls les 8 premiers caractères de nom ont été lus. Il faut donc utiliser un **informat** pour remédier à ce problème.

```
DATA exemple;
INFILE datalines DLM="&";
INPUT nom: $11. age sexe$;
DATALINES;
Chris Evans&25&m
JaneTrulli&21&f
JoeStar&49&m
;
RUN;
```

### 5.1.4 Lecture de données dans un fichier externe

Les données se trouvent dans un fichier externe nommé *individus.txt*. Ici les données sont délimitées par une tabulation :




Pour lire ces données, on utilise l'instruction `INFILE`

```
DATA tableindiv;
INFILE 'individus.txt' DSD DLM='09'x  TRUNCOVER;
INPUT prenom $ nom $ sexe $ age ville $10.;
RUN;
```

L'option DSD prévient d'éventuelles données manquantes, et l'option TRUNCOVER pour lire des lignes non complètes sans passer directement à la ligne suivante : ici sans l'option TRUNCOVER SAS ira à la ligne suivante sans lire Londres qui n'occupe pas 10 caractères. DLM='09'x est utilisé pour indiquer que le séparateur est ici une tabulation.

### 5.1.5 Manipulations de données

- **Utilisation de fichiers de données SAS existants : Instruction SET**

  L'instruction utilisée pour créer un tableau SAS à partir d'un autre ou tout simplement **modifier** un tableau SAS est l'instruction **SET**.

  **Syntaxe** :

```
data nouveaufichier(options);
set fichierexistant (options);
instructions complémentaires (IF, KEEP, DROP…);
run ;
```

  - **En ne conservant que certaines variables :** KEEP

```
data exemple ;
set exemple;
keep nom sexe; /*on ne conserve que le nom et le sexe */
run;

ou

data exemple ;
set exemple (keep= nom sexe); /*on ne conserve que le nom et le sexe */
run;
```

  - **En supprimant certaines variables :** DROP

```
data exemple ;
set exemple;
drop taille; /* on supprime la taille*/
run;
```

- **En conservant certaines observations:** IF, WHERE, OBS, FIRSTOBS

```
data exemple ;
set exemple;
if sexe='F'; /*on ne conserve que les personnes de sexe féminin*/
run;

ou

data exemple;
set exemple (where=(sexe='F'));
run;
```

- **Création de plusieurs fichiers simultanément dans SAS_OUTPUT**

  Dans SAS, il est possible de **créer plusieurs fichiers de données (datasets)** en une seule étape en les listant directement **après l'instruction DATA**. Cela permet de centraliser la logique de traitement dans un même bloc de code.

  Pour diriger chaque observation vers l'un des fichiers créés, on utilise ensuite l'instruction **OUTPUT nomdefichier**, qui indique explicitement dans quel fichier l'observation courante doit être stockée.

  Exemple :

```
DATA hommes femmes;
    SET population;
    IF sexe = "M" THEN OUTPUT hommes;
    ELSE IF sexe = "F" THEN OUTPUT femmes;
RUN;
```

  Dans cet exemple :

  - Deux fichiers sont créés : hommes et femmes.
  - Les données sont lues depuis le dataset population.
  - Selon la valeur de la variable sexe, chaque observation est dirigée vers le fichier approprié à l'aide de l'instruction OUTPUT.

Exemple 2 :

```
DATA jeunes adultes seniors;
    SET population;
    IF age < 18 THEN OUTPUT jeunes;
    ELSE IF age < 65 THEN OUTPUT adultes;
    ELSE OUTPUT seniors;
RUN;
```

Ce code crée trois fichiers à partir du dataset population, selon les tranches d'âge.

## 5.2 Créations de variables

### 5.2.1 Création d'une variable à partir des variables existantes

Lors de la création de nouvelles variables dans SAS, un certain nombre de formules et de fonctions peuvent être utilisées pour effectuer des calculs ou des transformations sur les données existantes. Ces fonctions peuvent appartenir à différentes catégories selon leur usage :

- **Fonctions arithmétiques**

  Permettent d'effectuer des **opérations mathématiques de base** : +, -, *, /, ** (exposant)
  Exemple : prix_total = quantite * prix_unitaire;

- **Fonctions statistiques**

Utilisées pour calculer des **résumés numériques** : MEAN(x1, x2, ...), SUM(x1, x2, ...), MIN(...), MAX(...), STD(...) (écart type), etc.
Exemple : moyenne_note = MEAN(note1, note2, note3);

- **Fonctions mathématiques**
  Utiles pour des calculs plus avancés : SQRT(x) (racine carrée), LOG(x) (logarithme naturel), EXP(x) (exponentielle), ROUND(x, multiple).
  Exemple : surface = 3.14 * RAYON**2;

- **Structure d'une formule dans SAS**

  Une **formule de calcul** peut contenir :
  - des **constantes** (ex. : "France", "01jan2020"d, 3.14)
  - des **noms de variables existantes**
  - des **fonctions SAS**
  - des **opérateurs arithmétiques**

  Exemple: age = YEAR(today()) - annee_naissance;

### 5.2.2 Création de nouvelles variables numériques avec des fonctions SAS

SAS offre un ensemble de **fonctions numériques** permettant de **créer de nouvelles variables** à partir d'opérations sur des variables existantes.

#### 5.2.2.1 Fonctions utiles

- INT(x) : Retourne la **partie entière** d'une valeur numérique (tronque la partie décimale)
  age_entier = INT(age);
- ROUND(x, multiple) : Arrondit une valeur au multiple spécifié.
  prix_arrondi = ROUND(prix, 0.05); /* arrondi au 5 centimes */
- SUM(x1, x2, ...) : Additionne plusieurs variables **en ignorant les valeurs manquantes**.
  total_ventes = SUM(vente1, vente2, vente3);
  Contrairement à l'opérateur +, SUM() ne retourne **pas de valeur manquante** si l'une des variables est manquante.
  revenu = salaire + bonus; → Si bonus est manquant → revenu = .
  revenu = SUM(salaire, bonus); → Si bonus est manquant → revenu = salaire.
  On peut aussi utiliser la liste implicite de variables, en utilisant le mot clé OF devant le premier nom de variable pour éviter la soustraction. Var=SUM(OF x1-xn)
- MEAN(x1, x2, ...) : Calcule la **moyenne** des variables spécifiées, sur les **valeurs non manquantes**.
  moyenne_note = MEAN(note1, note2, note3);

Comme pour SUM(), la fonction MEAN() **ignore les valeurs manquantes** et retourne la moyenne des valeurs disponibles. Si **toutes les valeurs** sont manquantes, le résultat est **manquant** (.).

On peut aussi modifier ou encore créer une variable à partir des variables existantes :

```
Data exemple;
set exemple;
If note>=10 then Note=Note+1; /*on ajoute 1 à la variable Note*/
Run;

Data exemple;
Set exemple;
 rapport=poids/taille ;
Run ;
```

# 6 Fusion de fichiers

Pour des raisons techniques (taille des fichiers de données) ou de provenance (sources différentes), il est parfois nécessaire de combiner des tableaux SAS pour n'en faire qu'un seul. Cependant, il existe de nombreuses façons de combiner les données (concaténation verticale, horizontale, mise à jour…).

## 6.1 Concaténation verticale: SET

Il s'agit d'additionner deux ou plusieurs fichiers de données en les ajoutant l'un à la suite de l'autre.

L'instruction utilisée est SET comme illustré dans l'exemple suivant :
Syntaxe :

```
DATA fusionverticale;
SET fichier1 fichier2 fichier3 … fichiern;
RUN;
```

Il est possible de concaténer plus de deux tableaux en même temps. En revanche, un problème se pose quant aux variables présentes dans chacun des fichiers. Les variables non communes aux deux tableaux seront incluses dans le tableau final mais ne seront pas renseignées comme elles ne l'étaient pas au départ.
De même, si les variables ne sont pas du même type, la concaténation ne fonctionnera pas. Il faudra transformer les données.

## 6.2 Concaténation horizontale: MERGE

Il arrive d'obtenir deux tableaux SAS concernant les mêmes individus mais comportant des variables différentes. Lorsqu'il est nécessaire de lier des variables contenues dans les deux fichiers, il est possible de réaliser une concaténation horizontale, c'est-à-dire de créer un seul fichier contenant toutes les variables. L'instruction utilisée dans ce cas est l'instruction MERGE.
Syntaxe

```
DATA fusionhorizontale;
MERGE fichier1 fichier3..fichiern;
BY Idenetifiant_commun_aux_n_fichiers ;
RUN;
```

### 6.2.1 Fusion horizontale sophistiquée : MERGE avec option BY

Le programme suivant permet de fusionner N fichiers par rapport aux individus :

```
DATA ensemble;
MERGE fichier1 fichier3…fichiern;
BY Idenetifiant_commun_aux_n_fichiers;
RUN;
```

**Attention** : Pour que BY fonctionne, les fichiers doivent avoir été triés par rapport à la variable contenue dans le BY (ici `Idenetifiant_commun_aux_n_fichiers`). Si tel n'est pas le cas, utilisez PROC SORT.

### 6.2.2 Fusion avec l'utilisation de l'option IN=

IN=Variable est une option des instructions SET et MERGE permettant de savoir d'où vient l'observation lorsque l'on fusionne plusieurs fichiers de données :

```
DATA ensemble;
MERGE fichier1 (in=A) fichier3 (in=B);
BY nom;
IF A;
RUN;
```

- `DATA ensemble` : On crée une nouvelle table appelée `ensemble`.
- `MERGE fichier1 (in=A) fichier3 (in=B)` : On fusionne deux tables `fichier1` et `fichier3` par la variable `nom`. Les options `(in=A)` et `(in=B)` créent deux variables indicatrices temporaires (valeurs 0 ou 1), qui indiquent si l'observation provient de `fichier1 (A=1)` ou de `fichier3 (B=1)`.
- `BY nom` : La fusion se fait par la variable `nom`, donc les deux jeux de données doivent être triés préalablement par `nom`.
- `IF A` : On garde uniquement les observations qui proviennent de `fichier1 (A=1)`, qu'elles soient ou non appariées avec `fichier3`

```
DATA ensemble;
MERGE fichier1 (in=A) fichier3 (in=B);
BY nom;
IF A AND B;
RUN;
```

- `MERGE fichier1 (in=A) fichier3 (in=B)` : Fusionne les deux fichiers selon la variable `nom`. Les indicateurs A et B permettent de savoir de quel fichier vient chaque observation.
- `BY nom` : Nécessite que les deux fichiers soient triés par `nom`.
- `IF A AND B` : Ne garde que les observations présentes dans les deux fichiers (donc les noms communs à `fichier1` et `fichier3`).

## 7 Overview

On détaille maintenant quelques procédures usuelles :

- PROC PRINT réalise des impressions,

- PROC IMPORT réalise des importations de données externes dans SAS
- PROC EXPORT réalise des exportations tables SAS vers d'autres types (txt, Excel, csv…)
- PROC SORT permet de trier un fichier
- PROC TABULATE tableaux croisés,
- PROC MEANS/SUMMARY calcul de statistiques descriptives,
- PROC FREQ tableaux de fréquence ou de contingence et tests du Chi-2,
- PROC CORR calculs de corrélations,
- PROC GPLOT permet de créer des graphiques 2D,
- PROC BOXPLOT permet de créer des **boîtes à moustaches,**
- PROC GCHART permet de générer différents types de **diagrammes,**
- PROC UNIVARIATE description et tests de la distribution d'une variable
- PROC FORMAT réalise des formats personnalisés
- PROC TTEST réalise des tests d'égalité de la variance et de la moyenne de deux échantillons
- PROC SQL permet d'extraire/modifier de l'information entre des bases/tables relationnelles

Toutes les procédures d'analyse statistique ont la même structure de syntaxe :

```
PROC Nom_PROC DATA=nom_de_table (WHERE=(expression));
VAR variable(s)_numérique(s);
BY <DESCENDING> variable(s);
CLASS variable(s);
RUN;
```

Pour effectuer une sélection des observations de la table SAS spécifiée en entrée d'une procédure SAS, on peut utiliser l'option WHERE= .

Pour sélectionner les variables analysées dans les procédures de statistique, on utilise l'instruction VAR. Elle est valable dans un grand nombre de procédures statistiques : MEANS, UNIVARIATE, CORR, etc. Les variables numériques sont les seules autorisées dans une instruction VAR. Pour mener une analyse sur plusieurs sous-groupes d'observations, on dispose des instructions CLASS et BY.

**Comparaison Instructions BY/CLASS**

- BY : Valable dans toutes les procédures d'analyse statistique. CLASS : Valable uniquement dans les procédures MEANS, UNIVARIATE et TTEST.
- BY : Les valeurs manquantes de chaque variable du BY constituent un sous-groupe pris en compte dans l'analyse. CLASS : Par défaut, les valeurs manquantes des variables de classe ne sont pas prises en compte dans l'analyse.
- BY : Nécessite un tri préalable de la table en entrée : tri par la même liste de variables que celle de l'instruction BY et dans le même ordre. CLASS : Ne nécessite pas de tri préalable de la table en entrée.

## 8 PROC PRINT

Edite les observations d'un tableau. La commande PROC PRINT peut être suivie de l'option DATA= puis des options, séparées chacune par un caractère blanc.

### 8.1 Exemple

```
proc sort data=exemple;
by sexe;
run;

PROC PRINT data=exemple UNIFORM;
VAR nom sexe age taille poids;
ID nom sexe age taille poids;
BY sexe;
PAGEBY sexe;
SUMBY sexe;
SUM age taille poids;
RUN;
```

sexe=F

| nom | sexe | age | taille | poids | nom | sexe | age | taille | poids |
|---|---|---|---|---|---|---|---|---|---|
| Marine-A | F | 21 | 169 | 60.0 | Marine-A | F | 21 | 169 | 60.0 |
| Sarah-V | F | 19 | 170 | 50.0 | Sarah-V | F | 19 | 170 | 50.0 |
| Mélanie | F | 22 | 172 | 55.0 | Mélanie | F | 22 | 172 | 55.0 |
| Anaïs | F | 18 | 162 | 44.6 | Anaïs | F | 18 | 162 | 44.6 |
| Jane | F | 21 | 145 | 56.8 | Jane | F | 21 | 145 | 56.8 |
| Emma | F | 21 | 172 | 77.1 | Emma | F | 21 | 172 | 77.1 |
| Jane | F | 21 | 170 | 67.3 | Jane | F | 21 | 170 | 67.3 |
| Elisa | F | 16 | 178 | 77.0 | Elisa | F | 16 | 178 | 77.0 |
| Amy | F | 21 | 170 | 72.6 | Amy | F | 21 | 170 | 72.6 |
| Sihui | F | 21 | 169 | 72.1 | Sihui | F | 21 | 169 | 72.1 |
| July | F | 20 | 171 | 78.0 | July | F | 20 | 171 | 78.0 |
| Chloe | F | 16 | 160 | 66.2 | Chloe | F | 16 | 160 | 66.2 |
| sexe | | | | | | | 237 | 2008 | 776.7 |

sexe=M

| nom | sexe | age | taille | poids | nom | sexe | age | taille | poids |
|---|---|---|---|---|---|---|---|---|---|
| Denis-M | M | 18 | 165 | 45.0 | Denis-M | M | 18 | 165 | 45.0 |
| Antoine | M | 20 | 175 | 63.0 | Antoine | M | 20 | 175 | 63.0 |
| Jean | M | 21 | 170 | 56.2 | Jean | M | 21 | 170 | 56.2 |
| Pierre | M | 20 | 172 | 63.2 | Pierre | M | 20 | 172 | 63.2 |
| Chris | M | 16 | 170 | 63.2 | Chris | M | 16 | 170 | 63.2 |
| Tom | M | 20 | 155 | 66.0 | Tom | M | 20 | 155 | 66.0 |
| Joe | M | 19 | 173 | 68.2 | Joe | M | 19 | 173 | 68.2 |
| Jerry | M | 18 | 168 | 65.0 | Jerry | M | 18 | 168 | 65.0 |
| Dan | M | 18 | 166 | 68.0 | Dan | M | 18 | 166 | 68.0 |

## 9 PROC IMPORT

Permet d'importer sous SAS des fichiers de données Excel, Lotus, Access, dBase, ….

Syntaxe simplifiée

```
PROC IMPORT DATAFILE='chemin\nomfichier'
OUT=nomfichiersas
DBMS=type_de_fichier <REPLACE> ;
<options selon type du fichier> ;
RUN;
```

DBMS peut prendre les valeurs : **EXCEL, TAB, …**
REPLACE : remplace un fichier existant. Si REPLACE n'est pas spécifié, IMPORT n'écrasera pas un fichier existant.

**Les options** selon type du fichier sont : Getnames=**Yes** (Si la première ligne de votre fichier de données comporte les noms des variables) ou **No** (sinon).

### 9.1 Exemple

```
PROC IMPORT DATAFILE= '/home/aladjibassene130/M1_MSI/msi_data.xlsx'
OUT= Clients DBMS=xlsx REPLACE;
```

```
SHEET="Clients";
GETNAMES=YES;
RUN;

PROC IMPORT DATAFILE='/home/aladjibassene130/M1_MSI/russett.txt'
   DBMS=TAB REPLACE OUT=TOTO.russet;
      GETNAMES=YES;
RUN;
```

# 10 PROC EXPORT

Permet d'exporter des fichiers de données SAS sous Excel, Access, ….

Syntaxe simplifiée

```
PROC EXPORT DATAFILE=nomfichiersas
OUTFILE='chemin\nomfichier'
DBMS=type de fichier <REPLACE> ;
<options selon type du fichier> ;
RUN;
```

Les options selon type du fichier sont :

DELIMITER='*caractère*': Identifie le délimiteur qui va séparer les colonnes de données. Par défaut c'est un espace.

## 10.1 Exemple

```
PROC EXPORT DATA=Clients
OUTFILE= "/home/aladjibassene130/M1_MSI/msi_data_exp.xlsx"
DBMS=xlsx REPLACE;
SHEET="Clients";
RUN;
```

# 11 PROC SORT

La procédure SORT permet de trier les observations d'un jeu de données selon l'ordre croissant (ou décroissant) des valeurs des variables spécifiées dans l'instruction BY**.**

Syntaxe **:**

```
PROC SORT DATA=nom_du_jeu_de_données_à_trier;
BY [DESCENDING] nom_variable ... [DESCENDING] nom_variable;
RUN;
```

## 11.1 Exemple

```
proc sort data=exemple;
by sexe;
run;
```

```
proc sort data=exemple;
by sexe groupe;
run;
```

```
proc sort data=exemple;
by descending sexe;
run;
```

```
proc sort data=exemple;
by descending sexe descending age;
run;
```

## 12 PROC TABULATE

La procédure TABULATE permet d'obtenir diverses statistiques descriptives disposées en une table. Les statistiques éditées sont sensiblement les mêmes que dans la PROC MEANS. La force de cette procédure réside dans la présentation des résultats.

Syntaxe :

```
PROC TABULATE DATA=tableSAS <options1>;
CLASS nom_variable(s);
VAR nom_variable(s);
TABLE (structure des lignes du tableau : page, colonnes, lignes),
(structure des colonnes du tableau)*(calculs apparaissant dans
les cases du tableau);
```
{Identifie, selon une des trois formes suivantes: var*var (croiser), var var (concatener), (var var) (grouper), les variables définissant les pages, les lignes et les colonnes des tables à imprimer. };
```
BY [NOTSORTED][DESCENDING] variable format...;
```
{Imprime des tables différentes pour les observations de chaque groupe.}
```
RUN;
```

**<options1>**

**FORMAT=format :** Identifie un format selon lequel on imprime les données dans chaque cellule de la table.
**FORMCHAR= 'onze_caractères' :** Identifie les caractères qui formeront les lignes de la table. Par défaut c'est |----|+|---.
**NOSEPS= :** Supprime l'impression des lignes horizontales.
**ORDER= :** À utiliser avec l'instruction CLASS, indique parmi DATA, FORMATTED, FREQ et INTERNAL l'ordre selon lequel on imprime les valeurs des variables. Par défaut c'est INTERNAL.
**MISSING :** Indique de tenir compte des valeurs manquantes.
**VARDEF**= Indique le diviseur utilisé lors des calculs de variances parmi :

DF (n-1) (choisi par défaut)
N (n) (choisi par défaut)
WDF (somme des poids moins 1)
WEIGHT (somme des poids).

**<Options2> :**

**BOX :** Identifie parmi _PAGE_, 'texte' et 'nom_variable' le(s) mot(s) à écrire dans la cellule vide du coin supérieur gauche de la table.
**CONDENSE :** Imprime plusieurs tables sur une même page.
**MISSTEXT=' texte' :** Identifie un maximum de 20 caractères qui seront imprimés dans les cellules contenant une valeur manquante.
**PRINTMISS :** Imprime toutes les combinaisons de valeurs possibles des variables de la table même s'il s'agit, par exemple, d'une colonne ne comportant que des valeurs manquantes.
**FUZZ=nombre :** Utilise 0 à la place des données (autre que les fréquences) dont la valeur absolue est plus petite que ce nombre dans les calculs et pour l'impression.

## 12.1 Exemple d'application

L'ensemble de données de Russett (Russett, 1964) est étudié dans Gifi (1990). Trois blocs de variables ont été définis pour 47 pays. Le premier bloc est lié aux « inégalités agricoles », le second au « développement industriel » et le dernier décrit l'« instabilité politique ». Russett a collecté ces données pour étudier les relations entre les inégalités agricoles, le développement industriel et l'instabilité politique. Les hypothèses de Russett peuvent être formulées comme suit : il est difficile pour un pays d'échapper à la dictature lorsque ses inégalités agricoles sont supérieures à la moyenne et son développement industriel inférieur à la moyenne.

- Inégalités agricoles
  - GINI : Inégalité de la répartition des terres,
  - FERME (farm) : Pourcentage d'agriculteurs qui possèdent la moitié des terres,
  - LOYER (rent) : Pourcentage d'agriculteurs qui louent toutes leurs terres.
- Développement industriel
  - PNBR (gnpr) : Produit national brut par habitant (1955 $),
  - LABO : Pourcentage de la main d'œuvre employée dans l'agriculture.
- Instabilité politique
  - INST: Instabilité de l'exécutif (45-61),
  - ECKS : Nombre d'incidents de guerre interne violents (46-61),
  - DÉCÈS (death) : Nombre de personnes tuées à la suite de violences de groupes civiques (50-62),
  - Demo : Démocratie stable, Démocratie instable, Dictature.

| | continent | pays | gini | farm | rent | gnpr | labo | inst | ecks | death | demo |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 2 | AMS | Argentina | 86.3 | 98.2 | 3.52 | 5.92 | 3.22 | 0.07 | 4.06 | 5.38 | instable |
| 3 | OCE | Australia | 92.9 | 99.6 | 3.4 | 7.1 | 2.64 | 0.01 | 0 | 0 | stable |
| 4 | EUR | Austria | 74 | 97.4 | 2.46 | 6.28 | 3.47 | 0.03 | 1.61 | 0 | instable |
| 5 | EUR | Belgium | 58.7 | 85.8 | 4.15 | 6.92 | 2.3 | 0.45 | 2.2 | 0.69 | stable |
| 6 | AMS | Bolivia | 93.8 | 97.7 | 3.04 | 4.19 | 4.28 | 0.37 | 3.99 | 6.5 | dictature |
| 7 | AMS | Brasil | 83.7 | 98.5 | 2.31 | 5.57 | 4.11 | 0.45 | 3.91 | 0.69 | instable |
| 8 | AMN | Canada | 49.7 | 82.9 | 2.1 | 7.42 | 2.48 | 0.01 | 3.14 | 0 | stable |
| 9 | AMS | Chile | 93.8 | 99.7 | 2.67 | 5.19 | 3.4 | 0.12 | 3.09 | 1.1 | instable |
| 10 | AMS | Colombia | 84.9 | 98.1 | 2.57 | 5.8 | 4.01 | 0.18 | 3.87 | 5.76 | instable |
| 11 | AMN | CostaRica | 88.1 | 99.1 | 1.86 | 5.73 | 4.01 | 0.18 | 3 | 3.22 | instable |

```
proc tabulate data=russet;
class demo;
table (demo
ALL="Tous")*(N="Effectif"
PCTN="%");
run;
```

| | demo | | | | | | Tous | |
|---|---|---|---|---|---|---|---|---|
| | dictature | | instable | | stable | | | |
| | Effectif | % | Effectif | % | Effectif | % | Effectif | % |
| | 20 | 42.55 | 12 | 25.53 | 15 | 31.91 | 47 | 100.00 |

```
proc tabulate data=russet;
class demo;
var gnpr gini farm rent;
table (demo ALL="Tous"),
gnpr*MEAN="Moyenne"
gini*sum="Somme"
farm*sum="Somme"
rent*MEAN="Moyenne";
run;
```

| | gnpr | gini | farm | rent |
|---|---|---|---|---|
| | Moyenne | Somme | somme | Moyenne |
| demo | | | | |
| dictature | 5.32 | 1498.70 | 1901.10 | 2.83 |
| instable | 6.00 | 898.40 | 1135.30 | 2.44 |
| stable | 6.77 | 958.30 | 1331.40 | 3.01 |
| Tous | 5.96 | 3355.40 | 4367.80 | 2.79 |

Dans le premier tableau, les statistiques affichées dans les cellules sont les effectifs (N) et les pourcentages (PCTN). Le mot-clé ALL crée une colonne récapitulative : il s'agit ici du total des effectifs. Dans le second tableau, la dernière partie de l'instruction TABLE concerne les calculs à effectuer : la syntaxe est de la forme nomVariabledeCalcul*statistique, où la variable de calcul est celle de l'instruction VAR et la statistique est : SUM, MEAN, STD, MAX, MIN, RANGE, MEDIAN, Q1, Q3, P1, P5, P10, P90, P95, P99…

## 13 PROC MEANS et PROC SUMMARY

Les procédures MEANS et SUMMARY permettent d'obtenir plusieurs statistiques descriptives. Elles sont très similaires. Leur différence majeure vient du fait que PROC MEANS imprime automatiquement les statistiques descriptives à moins de spécifier l'option NOPRINT contrairement à PROC SUMMARY qui ne les imprime pas à moins de spécifier l'option PRINT.

La PROC MEANS permet d'obtenir des statistiques descriptives élémentaires sur des variables quantitatives. Par défaut, elle calcule le nombre d'observations non manquantes, la moyenne, l'écart-type, la valeur minimum et la valeur maximum de toutes les variables numériques de la table (ou des variables indiquées par l'instruction VAR). Néanmoins, il existe des options permettant de demander un grand nombre de statistiques comme : somme, médiane, variance, skewness, kurtosis, quartiles, premier et dernier centile, premier et dernier décile.

Syntaxe :

```
PROC MEANS [ou SUMMARY] DATA=nom_du_jeu_de_données <options><statistiques>;
VAR nom_variables … ; {Liste des variables numériques pour lesquelles on veut les statistiques.}
CLASS nom_variables … ; {Liste des variables qui serviront à former des sous-groupes.}
FREQ nom_variable; {Identifie une variable numérique dont les valeurs représentent la fréquence de chaque observation.}
BY nom_variable … ; {Effectue des analyses statistiques séparées pour les observations de chaque groupe.}
OUTPUT OUT=nom <statistiques>; {Nomme un nouveau jeu de données qui contiendra les statistiques spécifiées.}
RUN ;
```

**<Options> :**
**ORDER**= À utiliser avec l'instruction CLASS, indique parmi DATA, FORMATTED, FREQ et INTERNAL l'ordre selon lequel on imprime les valeurs des variables. Par défaut c'est INTERNAL.
**DESCENDING** Imprime d'abord les sous-groupes, puis les groupes et finalement l'ensemble des données.
FW=I, identifie la largeur (I) des champs à utiliser pour l'impression de chaque statistique. Par défaut c'est 12.
**MAXDEC=n :** Identifie le nombre (0 <= n <= 8) maximal de décimales à imprimer pour les résultats.
**NOPRINT** : À utiliser avec PROC MEANS, supprime l'impression de toutes les statistiques descriptives.
**PRINT** (obligatoire pour SUMMARY) : À utiliser avec PROC SUMMARY, imprime toutes les statistiques descriptives.
**IDMIN :** À utiliser avec l'instruction ID, les variables serviront plutôt à identifier les valeurs minimales.
**MISSING** À utiliser avec l'instruction CLASS, indique de tenir compte des valeurs manquantes pour former les sous-groupes.
**NWAY :** Indique d'imprimer les statistiques que pour les sous-groupes (sinon les groupes), mais pas pour l'ensemble des données.
**VARDEF**= Indique le diviseur utiliser lors des calculs de variances et de covariances parmi :
DF (n-1) (choisi par défaut)

N (n) (choisi par défaut)
WDF (somme des poids moins 1)
WEIGHT (somme des poids).

**<Statistiques> :**
N, NMISS, NOBS, MIN, MAX, RANGE, SUM, SUMWGT, MEAN, CSS, USS, VAR, STD, STDERR, CV, SKEWNESS, KURTOSIS, T et PRT.

## 13.1 Exemple d'application:

```
PROC MEANS DATA=russet;
VAR gnpr;
CLASS demo;
RUN;
```

La procédure MEANS

| Variable d'analyse : gnpr | | | | | | |
|---|---|---|---|---|---|---|
| demo | N obs | N | Moyenne | Ec-type | Minimum | Maximum |
| dictature | 20 | 20 | 5.3210000 | 0.5689686 | 4.1900000 | 6.6400000 |
| instable | 12 | 12 | 5.9983333 | 0.5738361 | 5.1900000 | 6.9500000 |
| stable | 15 | 15 | 6.7746667 | 0.7895556 | 4.2800000 | 7.7600000 |

```
PROC SORT DATA=russet;
BY demo;
RUN;

PROC MEANS DATA=russet
mean nmiss;
VAR gnpr;
BY demo;
RUN;
```

La procédure MEANS

demo=dictature

| Variable d'analyse : gnpr | |
|---|---|
| Moyenne | Nbre manquant |
| 5.3210000 | 0 |

demo=instable

| Variable d'analyse : gnpr | |
|---|---|
| Moyenne | Nbre manquant |
| 5.9983333 | 0 |

demo=stable

| Variable d'analyse : gnpr | |
|---|---|
| Moyenne | Nbre manquant |
| 6.7746667 | 0 |

```
PROC MEANS DATA=russet mean kurtosis uclm lclm t prt alpha=0.05;
VAR gnpr;
CLASS demo;
RUN;
```

La procédure MEANS

| Variable d'analyse : gnpr | | | | | | | |
|---|---|---|---|---|---|---|---|
| demo | N obs | Moyenne | Kurtosis | Borne supérieur de l'IC à 95% pour la moyenne | Borne inférieure de l'IC à 95% pour la moyenne | Valeur du test t | Pr > \|t\| |
| dictature | 20 | 5.3210000 | 0.5671416 | 5.5872855 | 5.0547145 | 41.82 | <.0001 |
| instable | 12 | 5.9983333 | -0.9321540 | 6.3629314 | 5.6337353 | 36.21 | <.0001 |
| stable | 15 | 6.7746667 | 7.5502417 | 7.2119080 | 6.3374253 | 33.23 | <.0001 |

KURTOSIS : coefficient d'aplatissement (compare la forme de la courbe de distribution des observations à celle de la loi normale : un coefficient positif indique une plus forte concentration des observations ; un coefficient négatif indique une courbe plus aplatie).
SKEWNESS : coefficient d'asymétrie (indique si les observations sont réparties équitablement autour de la moyenne (le coefficient est alors nul) ou si elles sont plutôt concentrées vers les valeurs les plus faibles (coefficient positif) ou vers les valeurs les plus élevées (coefficient négatif).

T : valeur de la statistique Student.
PRT : p-value du test de Student de nullité de la moyenne.

## 14 PROC FREQ

La procédure FREQ permet d'obtenir des tables de fréquences et des tableaux de contingences ainsi que plusieurs statistiques descriptives et différents tests. C'est également dans cette procédure que l'on trouvera l'opportunité de faire des tests du Chi2.

**SYNTAXE :**

```
PROC FREQ DATA=nom_du_jeu_de_données <liste-option1>;
BY nom_variable ...; {Imprime des tables ou des tableaux différents pour les observations de chaque groupe.}
TABLES ligne_var[*col_var]...</ liste-option2>; {Produit les tables de fréquences et les tableaux de contingences demandés.}
WEIGHT nom_variable; {Identifie une variable dont les valeurs doivent être des entiers qui représentent le nombre de sujets correspondants à chaque observation. Par défaut, chaque observation vaut un dans le compte des fréquences.}
RUN ;
```

L'instruction TABLES permet de définir la liste des distributions souhaitées. Plusieurs distributions peuvent être demandées simultanément, chacune pouvant être unidimensionnelle ou multidimensionnelle. Dans le cas de demande de plusieurs distributions simultanées, il suffit de séparer chaque distribution par un espace. Si on veut croiser plusieurs variables entre elles, il suffit de les séparer par des astérisques.

Les options sont les suivantes :

**<Liste-option1> :**

**FORMCHAR (1,2,7) = 'trois_caractères'** : Identifie les caractères qui formeront les lignes de divisions (1) verticales, (2) horizontales et (7) les intersections. Par défaut c'est |-+.
**ORDER**= : Indique parmi DATA, FORMATTED, FREQ et INTERNAL l'ordre selon lequel apparaîtront les variables. Par défaut c'est INTERNAL.
**PAGE** Imprime une seule table ou un seul tableau par page.

**</ liste-option2> :**

**ALL** : Idem à CHISQ, MEASURES et CMH.
**ALPHA**=α : Identifie le niveau 100(1- α) % de l'intervalle de confiance. Par défaut α =0,05.
**CHISQ** : Imprime la valeur et la "p-value" des khi-deux de Pearson, du maximum de vraisemblance et de Mantel-Haenszel, ainsi que le test de Fisher, le coefficient de Phi, celui de contingence et la statistique V de Cramer.
**CMH** : Imprime la valeur et la "p-value" des trois statistiques (corrélation, ANOVA et association générale) de Cochran-Mantel-Hanszel.
**CMH1** : Idem à CMH, mais imprime seulement la statistique de corrélation.
**CMH2** : Idem à CMH, mais n'imprime pas la statistique d'association générale.
**EXACT** : Test de Ficher pour les tableaux plus grands que 2x2
**MEASURES** : Imprime la valeur et l'erreur standard asymptotique des statistiques Gamma, Tau-b de Kendall, tau-c de Stuart, D de Somers, corrélation de Pearson, corrélation de Spearman, lambda asymétrique et symétrique ainsi que le coefficient d'incertitude, la taille de l'échantillon et, pour les tableaux 2x2, les risques relatifs.
**CELLCHI2** : Imprime pour chaque cellule sa contribution à la statistique Chi-2.
**CUMCOL** : Imprime pour chaque cellule les pourcentages cumulatifs des colonnes.

**DEVIATION** : Imprime pour chaque cellule la différence entre sa fréquence et sa valeur espérée.
**EXPECTED** : Imprime les fréquences espérées sous l'hypothèse de l'indépendance.
**MISSPRINT** : Pour les tableaux, imprime les fréquences des valeurs manquantes sans toutefois les utiliser dans les calculs de statistiques.
**MISSING** : Inclut les valeurs manquantes dans les calculs.
**OUT**=nom : Nomme le jeu de données qui contiendra les valeurs et les fréquences des variables de la dernière table ou du dernier tableau.
**LIST** : Imprime les données en une liste plutôt que sous forme de table.
**NOCOL** : Supprime l'impression des pourcentages des colonnes.
**NOCUM** : Supprime l'impression des fréquences cumulatives et des pourcentages cumulatifs.
**NOFREQ** : Supprime l'impression des fréquences.
**NOPERCENT** : Supprime l'impression des pourcentages.
**NOPRINT** : Supprime l'impression des tables, mais imprime les statistiques demandées.
**NOROW** : Supprime l'impression des pourcentages de lignes

## 14.1 Exemple d'application:

```
PROC FREQ DATA=russet;
TABLES continent demo;
RUN;
```

Le script ci-dessus fournit les deux distributions marginales de continent et demo

La procédure FREQ

| continent | Fréquence | Pourcentage | Fréquence cumulée | Pourcentage cumulé |
|---|---|---|---|---|
| AFR | 2 | 4.26 | 2 | 4.26 |
| AMN | 8 | 17.02 | 10 | 21.28 |
| AMS | 11 | 23.40 | 21 | 44.68 |
| ASI | 6 | 12.77 | 27 | 57.45 |
| EUR | 18 | 38.30 | 45 | 95.74 |
| OCE | 2 | 4.26 | 47 | 100.00 |

| demo | Fréquence | Pourcentage | Fréquence cumulée | Pourcentage cumulé |
|---|---|---|---|---|
| dictature | 20 | 42.55 | 20 | 42.55 |
| instable | 12 | 25.53 | 32 | 68.09 |
| stable | 15 | 31.91 | 47 | 100.00 |

```
PROC FREQ DATA=russet;
TABLES continent*demo;
RUN;
```

Le script ci-dessus fournit la distribution conjointe de continent et demo (table de contingence).

La procédure FREQ

Fréquence
Pourcentage
Pct de ligne
Pct de col.

| | Table de continent par demo | | | |
|---|---|---|---|---|
| | demo | | | |
| continent | dictature | instable | stable | Total |
| AFR | 2<br>4.26<br>100.00<br>10.00 | 0<br>0.00<br>0.00<br>0.00 | 0<br>0.00<br>0.00<br>0.00 | 2<br>4.26 |
| AMN | 5<br>10.64<br>62.50<br>25.00 | 1<br>2.13<br>12.50<br>8.33 | 2<br>4.26<br>25.00<br>13.33 | 8<br>17.02 |
| AMS | 6<br>12.77<br>54.55<br>30.00 | 4<br>8.51<br>36.36<br>33.33 | 1<br>2.13<br>9.09<br>6.67 | 11<br>23.40 |
| ASI | 4<br>8.51<br>66.67<br>20.00 | 1<br>2.13<br>16.67<br>8.33 | 1<br>2.13<br>16.67<br>6.67 | 6<br>12.77 |
| EUR | 3<br>6.38<br>16.67<br>15.00 | 6<br>12.77<br>33.33<br>50.00 | 9<br>19.15<br>50.00<br>60.00 | 18<br>38.30 |
| OCE | 0<br>0.00<br>0.00<br>0.00 | 0<br>0.00<br>0.00<br>0.00 | 2<br>4.26<br>100.00<br>13.33 | 2<br>4.26 |
| Total | 20<br>42.55 | 12<br>25.53 | 15<br>31.91 | 47<br>100.00 |

**Test d'indépendance du Chi-2**

```
PROC FREQ DATA=russet;
TABLES continent*demo/nofreq
nopercent norow nocol nocum
Expected deviation cellchi2 chisq;
RUN;
```

On a effectué l'analyse sur un échantillon très petit, aussi le test du Chi-2 n'est pas très fiable. Cependant, le test du Chi-2 montre une probabilité de rejeter à tort l'hypothèse d'indépendance entre les deux variables de 6% (0,06383), ce qui est assez important : on ne peut pas rejeter cette hypothèse. Cette conclusion est corroborée par les valeurs des trois derniers coefficients, phi, contingence et V de Cramer : pour ce dernier, par exemple, la liaison observée correspond à 43,26% de liaison entre les deux variables (ce qui est assez peu).

La procédure FREQ

Attendu
Ecart
Khi-2 de cellule

Table de continent par demo

| continent | demo | | | |
|---|---|---|---|---|
| | dictature | instable | stable | Total |
| AFR | 0.8511<br>1.1489<br>1.5511 | 0.5106<br>-0.511<br>0.5106 | 0.6383<br>-0.638<br>0.6383 | |
| AMN | 3.4043<br>1.5957<br>0.748 | 2.0426<br>-1.043<br>0.5321 | 2.5532<br>-0.553<br>0.1199 | |
| AMS | 4.6809<br>1.3191<br>0.3718 | 2.8085<br>1.1915<br>0.5055 | 3.5106<br>-2.511<br>1.7955 | |
| ASI | 2.5532<br>1.4468<br>0.8199 | 1.5319<br>-0.532<br>0.1847 | 1.9149<br>-0.915<br>0.4371 | |
| EUR | 7.6596<br>-4.66<br>2.8346 | 4.5957<br>1.4043<br>0.4291 | 5.7447<br>3.2553<br>1.8447 | |
| OCE | 0.8511<br>-0.851<br>0.8511 | 0.5106<br>-0.511<br>0.5106 | 0.6383<br>1.3617<br>2.905 | |
| Total | 20 | 12 | 15 | 47 |

Statistiques pour la table de continent par demo

| Statistique | DDL | Valeur | Prob |
|---|---|---|---|
| Khi-2 | 10 | 17.5894 | 0.0623 |
| Test du rapport de vraisemblance | 10 | 19.8177 | 0.0310 |
| Khi-2 de Mantel-Haenszel | 1 | 10.5465 | 0.0012 |
| Coefficient Phi | | 0.6118 | |
| Coefficient de contingence | | 0.5218 | |
| V de Cramer | | 0.4326 | |
| WARNING: 89% des cellules ont un effectif théorique inférieur à 5. Le test du Khi-2 peut ne pas convenir. | | | |

Taille de l'échantillon = 47

## 15 PROC CORR

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

Syntaxe :

```
PROC CORR DATA=nom_du_jeu_de_données <options>;
BY nom_variable ...; {Effectue des analyses statistiques séparées pour les observations de chaque groupe.}
FREQ nom_variable;
PARTIAL liste_de_variables;
VAR nom_variable ...; {Liste des variables dont on veut les corrélations.}
WEIGHT nom_variable; {Identifie une variable dont les valeurs représentent le poids de chaque observation pour le calcul des coefficients de corrélation pondérés.}
RUN;
```

BY : la procédure est exécutée pour chaque sous-groupe. PARTIAL : permet de mesurer les corrélations partielles entre les combinaisons 2 à 2 d'une liste de variables numériques (instruction VAR) en éliminant l'influence des variables de l'instruction PARTIAL. VAR : variables pour lesquelles les coefficients sont calculés.

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.

Quelques tests :

- KENDALL : sélection du type de corrélation KENDALL (pour les variables ordinales).
- PEARSON : sélection du type de corrélation PEARSON (par défaut) (pour les variables numériques).
- SPEARMAN : sélection du type de corrélation SPEARMAN (pour les variables ordinales).

**Les options sont les suivantes :**

**OUTP=nom :** Nomme un nouveau jeu de données qui contiendra les coefficients de corrélation de Pearson.
**OUTH=nom :** Nomme un nouveau jeu de données qui contiendra les statistiques de Hoeffding.
**OUTK=nom :** Nomme un nouveau jeu de données qui contiendra les coefficients de corrélation de Kendall.
OUTS=nom
Nomme un nouveau jeu de données qui contiendra les coefficients de corrélation de Spearman. HOEFFDING : Imprime la statistique D de Hoeffding.
**KENDALL** : Imprime les coefficients tau-b de Kendall à condition que l`instruction WEIGHT ne soit pas utilisée. **PEARSON** : Affiche le coefficient de corrélation linéaire (option valide par défaut).
**SPEARMAN** : Imprime les coefficients de Spearman.
**NOMISS** : Dans les calculs, ne tient pas compte des observations ayant des valeurs manquantes.
**VARDEF**= Indique le diviseur utiliser lors des calculs de variances et de covariances parmi :

DF (n-1) (choisi par défaut)
N (n) (choisi par défaut)
WDF (somme des poids moins 1)
WEIGHT (somme des poids).

**SINGULAR=p :** À utiliser avec l'instruction PARTIAL, identifie le critère (p) de détermination de la singularité des variables.
**ALPHA** Imprime le coefficient alpha de Cronbach.
**COV** Imprime la matrice des covariances.
**CSSCP** Imprime la matrice des sommes de carrés corrigées et des produits croisés.
**NOCORR** Supprime l'impression des corrélations de Pearson.
**SSCP** Imprime la matrice des sommes de carrés et les produits croisés.
**BEST=n :** Imprime n corrélations des variables ayant les plus grandes valeurs absolues. Imprime les coefficients en ordre décroissant.
**NOPRINT** Supprime toutes les impressions.
**NOPROB** Supprime l'impression de la signification P du test H0 : rho=0 contre H1 : rho=/ 0.
**NOSIMPLE** Supprime l'impression des statistiques descriptives pour chaque variable.
**RANK** Imprime les coefficients de corrélations pour chaque variable en ordre décroissant de valeur absolue.

## 15.1 Exemple d'application:

```
PROC CORR data=russet ;
VAR gini farm rent gnpr labo inst
ecks death;
run;
```

Coefficients de corrélation de Pearson, N = 47
Proba > |r| sous H0: Rho=0

| | gini | farm | rent | gnpr | labo | inst | ecks | death |
|---|---|---|---|---|---|---|---|---|
| gini | 1.00000 | 0.93765 <.0001 | 0.41006 0.0042 | -0.30010 0.0404 | 0.25651 0.0818 | 0.13787 0.3554 | 0.29374 0.0451 | 0.42517 0.0029 |
| farm | 0.93765 <.0001 | 1.00000 | 0.47676 0.0007 | -0.37352 0.0097 | 0.30188 0.0392 | 0.11733 0.4322 | 0.35087 0.0156 | 0.41222 0.0040 |
| rent | 0.41006 0.0042 | 0.47676 0.0007 | 1.00000 | -0.08017 0.5922 | -0.18726 0.2075 | 0.07226 0.6293 | 0.09291 0.5345 | 0.29739 0.0424 |
| gnpr | -0.30010 0.0404 | -0.37352 0.0097 | -0.08017 0.5922 | 1.00000 | -0.81510 <.0001 | -0.13918 0.3508 | -0.54727 <.0001 | -0.51512 0.0002 |
| labo | 0.25651 0.0818 | 0.30188 0.0392 | -0.18726 0.2075 | -0.81510 <.0001 | 1.00000 | 0.24886 0.0916 | 0.49456 0.0004 | 0.51636 0.0002 |
| inst | 0.13787 0.3554 | 0.11733 0.4322 | 0.07226 0.6293 | -0.13918 0.3508 | 0.24886 0.0916 | 1.00000 | 0.32717 0.0248 | 0.08357 0.5765 |
| ecks | 0.29374 0.0451 | 0.35087 0.0156 | 0.09291 0.5345 | -0.54727 <.0001 | 0.49456 0.0004 | 0.32717 0.0248 | 1.00000 | 0.62754 <.0001 |
| death | 0.42517 0.0029 | 0.41222 0.0040 | 0.29739 0.0424 | -0.51512 0.0002 | 0.51636 0.0002 | 0.08357 0.5765 | 0.62754 <.0001 | 1.00000 |

```
PROC CORR data=russet KENDALL
SPEARMAN  PEARSON;
VAR gini farm rent gnpr labo inst
ecks death;
run;
```

Coefficients de corrélation de Pearson, N = 47
Proba > |r| sous H0: Rho=0

| | gini | farm | rent | gnpr | labo | inst | ecks | death |
|---|---|---|---|---|---|---|---|---|
| gini | 1.00000 | 0.93765 <.0001 | 0.41006 0.0042 | -0.30010 0.0404 | 0.25651 0.0818 | 0.13787 0.3554 | 0.29374 0.0451 | 0.42517 0.0029 |
| farm | 0.93765 <.0001 | 1.00000 | 0.47676 0.0007 | -0.37352 0.0097 | 0.30188 0.0392 | 0.11733 0.4322 | 0.35087 0.0156 | 0.41222 0.0040 |
| rent | 0.41006 0.0042 | 0.47676 0.0007 | 1.00000 | -0.08017 0.5922 | -0.18726 0.2075 | 0.07226 0.6293 | 0.09291 0.5345 | 0.29739 0.0424 |
| gnpr | -0.30010 0.0404 | -0.37352 0.0097 | -0.08017 0.5922 | 1.00000 | -0.81510 <.0001 | -0.13918 0.3508 | -0.54727 <.0001 | -0.51512 0.0002 |
| labo | 0.25651 0.0818 | 0.30188 0.0392 | -0.18726 0.2075 | -0.81510 <.0001 | 1.00000 | 0.24886 0.0916 | 0.49456 0.0004 | 0.51636 0.0002 |
| inst | 0.13787 0.3554 | 0.11733 0.4322 | 0.07226 0.6293 | -0.13918 0.3508 | 0.24886 0.0916 | 1.00000 | 0.32717 0.0248 | 0.08357 0.5765 |
| ecks | 0.29374 0.0451 | 0.35087 0.0156 | 0.09291 0.5345 | -0.54727 <.0001 | 0.49456 0.0004 | 0.32717 0.0248 | 1.00000 | 0.62754 <.0001 |
| death | 0.42517 0.0029 | 0.41222 0.0040 | 0.29739 0.0424 | -0.51512 0.0002 | 0.51636 0.0002 | 0.08357 0.5765 | 0.62754 <.0001 | 1.00000 |

Coefficients de corrélation de Spearman, N = 47
Proba > |r| sous H0: Rho=0

| | gini | farm | rent | gnpr | labo | inst | ecks | death |
|---|---|---|---|---|---|---|---|---|
| gini | 1.00000 | 0.91174 <.0001 | 0.25607 0.0823 | -0.29770 0.0421 | 0.25960 0.0780 | 0.31763 0.0296 | 0.34103 0.0190 | 0.50477 0.0003 |
| farm | 0.91174 <.0001 | 1.00000 | 0.24973 0.0905 | -0.36644 0.0113 | 0.30252 0.0388 | 0.26404 0.0729 | 0.34173 0.0187 | 0.49308 0.0004 |
| rent | 0.25607 0.0823 | 0.24973 0.0905 | 1.00000 | -0.03606 0.8098 | -0.12212 0.4135 | -0.05085 0.7343 | 0.14340 0.3362 | 0.26100 0.0764 |
| gnpr | -0.29770 0.0421 | -0.36644 0.0113 | -0.03606 0.8098 | 1.00000 | -0.84457 <.0001 | -0.19300 0.1937 | -0.49006 0.0005 | -0.59763 <.0001 |
| labo | 0.25960 0.0780 | 0.30252 0.0388 | -0.12212 0.4135 | -0.84457 <.0001 | 1.00000 | 0.20616 0.1644 | 0.47418 0.0008 | 0.60647 <.0001 |
| inst | 0.31763 0.0296 | 0.26404 0.0729 | -0.05085 0.7343 | -0.19300 0.1937 | 0.20616 0.1644 | 1.00000 | 0.32936 0.0238 | 0.29546 0.0438 |
| ecks | 0.34103 0.0190 | 0.34173 0.0187 | 0.14340 0.3362 | -0.49006 0.0005 | 0.47418 0.0008 | 0.32936 0.0238 | 1.00000 | 0.72643 <.0001 |
| death | 0.50477 0.0003 | 0.49308 0.0004 | 0.26100 0.0764 | -0.59763 <.0001 | 0.60647 <.0001 | 0.29546 0.0438 | 0.72643 <.0001 | 1.00000 |

Coefficients de corrélation du Tau b de Kendall, N = 47
Proba > |tau| sous H0: Tau=0

| | gini | farm | rent | gnpr | labo | inst | ecks | death |
|---|---|---|---|---|---|---|---|---|
| gini | 1.00000 | 0.76873 <.0001 | 0.18782 0.0638 | -0.20539 0.0426 | 0.17513 0.0845 | 0.22270 0.0329 | 0.23094 0.0243 | 0.37175 0.0005 |
| farm | 0.76873 <.0001 | 1.00000 | 0.16309 0.1083 | -0.24499 0.0158 | 0.19701 0.0528 | 0.20293 0.0524 | 0.22675 0.0273 | 0.33393 0.0019 |
| rent | 0.18782 0.0638 | 0.16309 0.1083 | 1.00000 | -0.03723 0.7136 | -0.08862 0.3833 | -0.03089 0.7675 | 0.11233 0.2739 | 0.19529 0.0688 |
| gnpr | -0.20539 0.0426 | -0.24499 0.0158 | -0.03723 0.7136 | 1.00000 | -0.65641 <.0001 | -0.14378 0.1687 | -0.36513 0.0004 | -0.45953 <.0001 |
| labo | 0.17513 0.0845 | 0.19701 0.0528 | -0.08862 0.3833 | -0.65641 <.0001 | 1.00000 | 0.16056 0.1251 | 0.33383 0.0012 | 0.46468 <.0001 |
| inst | 0.22270 0.0329 | 0.20293 0.0524 | -0.03089 0.7675 | -0.14378 0.1687 | 0.16056 0.1251 | 1.00000 | 0.23391 0.0270 | 0.20460 0.0643 |
| ecks | 0.23094 0.0243 | 0.22675 0.0273 | 0.11233 0.2739 | -0.36513 0.0004 | 0.33383 0.0012 | 0.23391 0.0270 | 1.00000 | 0.55888 <.0001 |
| death | 0.37175 0.0005 | 0.33393 0.0019 | 0.19529 0.0688 | -0.45953 <.0001 | 0.46468 <.0001 | 0.20460 0.0643 | 0.55888 <.0001 | 1.00000 |

## 16 PROC GCHART

La procédure GCHART permet de tracer des histogrammes (BAR Charts) horizontaux, verticaux et à colonnes segmentées, ainsi que des diagrammes en bâtons (BLOCK Charts), circulaires (PIE Charts) et en étoiles (STAR Charts).

Syntaxe :

```
PROC GCHART DATA=nom_du_jeu_de_données <liste-option1>;
BY nom_variable ..;
{Sur chaque diagramme, représente séparément les observations de chaque groupe.}
HBAR nom_variable ... </ liste-option2,3>;
{Liste des variables à représenter par un histogramme à barres horizontales.}
VBAR nom_variable ... </ liste-option2,3>;
{Liste des variables à représenter par un histogramme à barres verticales.}
BLOCK nom_variable ... </ liste-option2,3>;
{Liste des variables à représenter par un diagramme en bâtons (3D).}
PIE nom_variable ... [NOHEADER] </ liste-option2>;
{Liste des variables à représenter par un diagramme circulaire.}
STAR nom_variable ... [NOHEADER] </ liste-option2>;
{Liste des variables à représenter par un diagramme en étoile.}
RUN ;
```

Les options sont les suivantes :

**<liste-option1> :**

**FORMCHAR (1,2,7,9,16,20)= 'six_caractères'** :Identifie les caractères qui formeront les graphiques. Par défaut c'est |-+-/*.

**LPI=n** : Identifie la proportion des diagrammes circulaires ou en étoiles. Ici, n = (nombre de lignes par pouce / nombre de colonnes par pouce)*10 à l'impression

**</liste-option2> :**

**AXIS=min max** : Identifie les valeurs minimale et maximale des axes.

**DISCRETE** : À utiliser lorsque la variable numérique à représenter est discrète.

**FREQ=nom_variable :** Identifie une variable numérique dont les valeurs représentent la fréquence de chaque observation.

**LEVELS=nombre** : Indique le nombre de barres ou de sections qui représenteront la variable continue.
**MIDPOINTS=valeur** ... Pour des variables continues, identifie les valeurs formant les milieux des intervalles représentés par chaque barre ou section.

**MISSING :** Indique de tenir compte des valeurs manquantes des variables à représenter.
**SUMVAR=nom_variable :** Identifie une variable dont la somme de ses valeurs sera imprimée pour chaque barre ou section.

**TYPE=** : Indique parmi FREQ, CFREQ, PERCENT, CPERCENT, MEAN et SUM ce que les barres ou les sections représentent. Par défaut c'est FREQ

**</liste-option3> :**

**GROUP=nom_variable :** Trace les diagrammes de chaque groupe côte-à-côte.
**SUBGROUP=nom_variable** : Trace des diagrammes à colonnes segmentées selon chaque sous-groupe.
**NOSYMBOL** : À utiliser avec l'instruction SUBGROUP, supprime l'impression de l'identification des symboles.

**SYMBOL='caractère'** : Identifie le(s) caractère(s) à utiliser pour dessiner les barres ou les bâtons. Par défaut c'est l'astérix (*).

**NOSTATS** : Supprime l'impression des fréquences, des fréquences cumulatives, des pourcentages et des pourcentages cumulatifs au bout de chaque barre.

**FREQ (ou CFREQ)** : Imprime seulement la fréquence (cumulative) au bout de chaque barre.

**PERCENT (ou CPERCENT) :** Imprime seulement le pourcentage (cumulatif) au bout de chaque barre. **SUM (ou MEAN)** : Imprime le nombre total d'observations (ou la moyenne) que représente chaque barre.

**ASCENDING (ou DESCENDING)** : Imprime les barres en ordre croissant (ou décroissant) de grandeur à l'intérieur de chaque groupe.

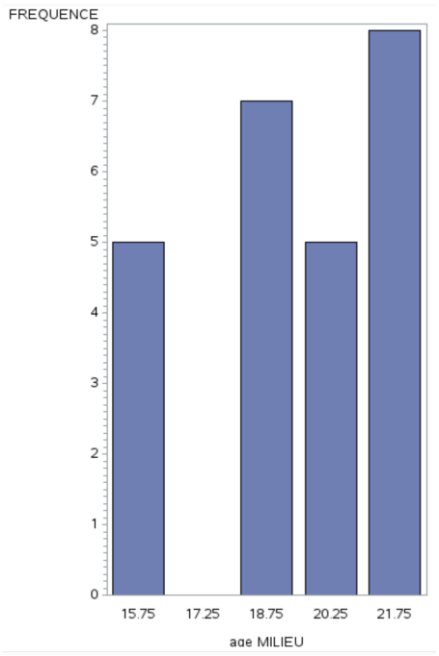
**NOZEROS** : Supprime l'emplacement des barres de longueur nulle.

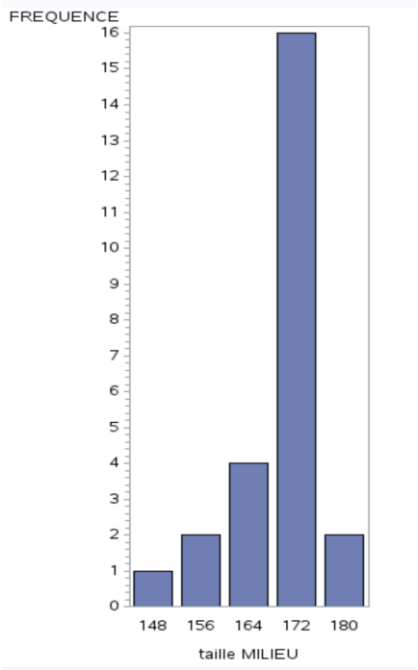
**REF=valeur** : Trace une droite perpendiculaire aux barres, vis-à-vis la valeur spécifiée de l'axe. **NOSPACE** : Imprime un histogramme à barres verticales collées si l'espace est insuffisant pour les séparer.

**NOHEADER** : Supprime l'impression de l'en-tête par défaut.

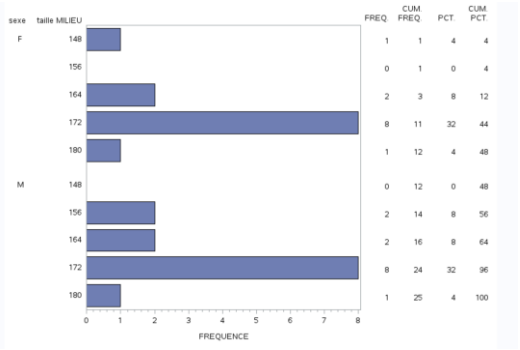
## 16.1 Exemple d'application

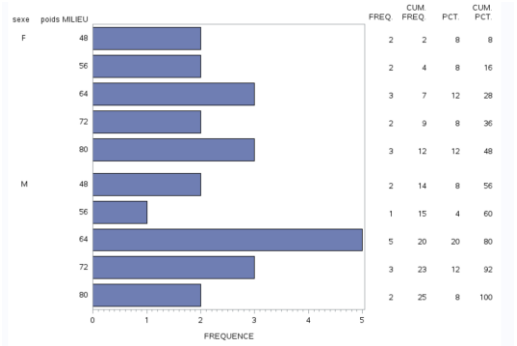
```
PROC GCHART DATA =
exemple;
VBAR age taille;
RUN ;
QUIT ;
```



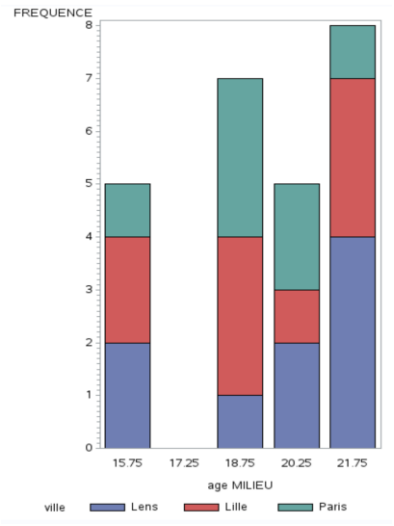



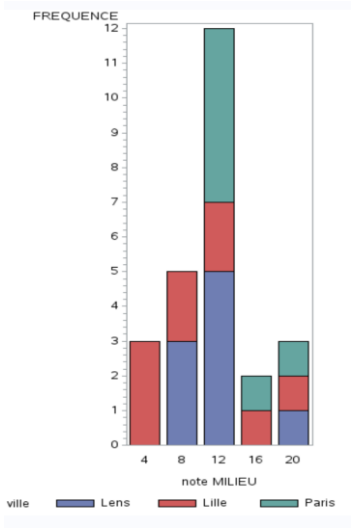
```
PROC GCHART DATA =
exemple;
HBAR taille poids /
GROUP = sexe;
RUN ;
QUIT ;
```





```
PROC GCHART DATA =
exemple;
VBAR age note /
SUBGROUP = ville;
RUN ;
QUIT ;
```




## 17 PROC BOXPLOT

PROC BOXPLOT est plus ancien, mais offre des options spécifiques à la qualité et au contrôle statistique. La procédure PROC BOXPLOT de **SAS** permet de créer des **boîtes à moustaches** (boxplots) pour représenter graphiquement la distribution d'une variable numérique selon une ou plusieurs catégories (groupes).

Elle est idéale pour:

- Visualiser la **dispersion**, la **médiane**, les **quartiles**, et les **valeurs aberrantes (outliers)**.
- Comparer une variable numérique entre plusieurs groupes.

Syntaxe :

```
PROC BOXPLOT DATA=mon_dataset;
   PLOT variable_numerique*variable_catégorique;
RUN;
```

- `DATA=` : nom du jeu de données.
- `variable_numerique` : variable continue (ex. : salaire).
- `variable_catégorique` : variable qui définit les groupes (ex. : département).

Options utiles :

**BOXSTYLE=schematic** : Affiche les outliers (valeurs aberrantes).

**ID=Identifiant** : Affiche le matricule des valeurs aberrantes.

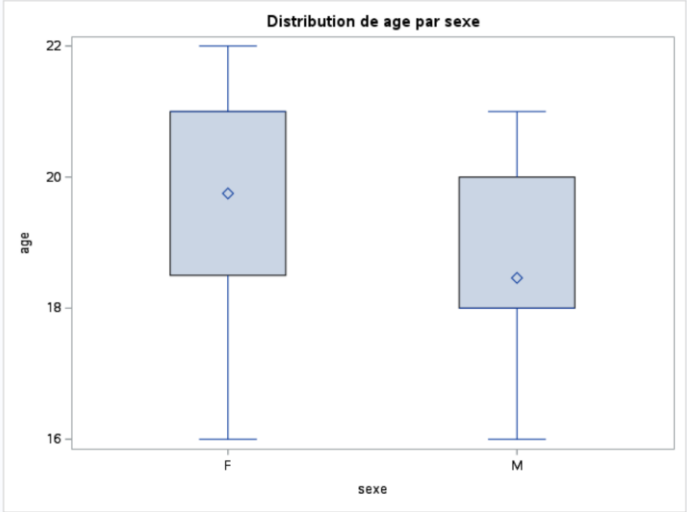
**HAXIS= / VAXIS=** : Personnalise les axes.

**NOTCH** : Ajoute une encoche autour de la médiane pour tester la différence entre médianes.

## 17.1 Exemple d'application

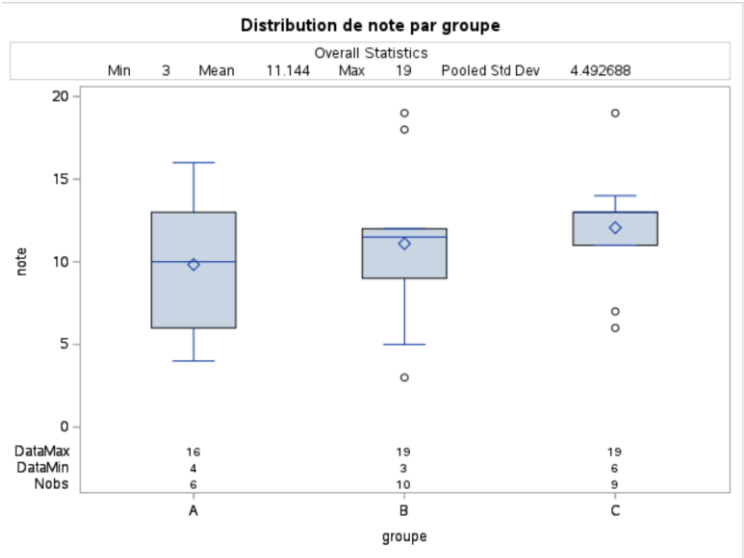
```
PROC SORT DATA= exemple;
      BY sexe;
RUN;

PROC BOXPLOT data=exemple;
      PLOT Age * sexe /
      CAXIS=black CTEXT=black
      CBOXES=black
BOXSTYLE=schematic
      IDCOLOR=black IDSYMBOL=dot;
RUN;
```




```
PROC SORT DATA=exemple;
      BY groupe;
RUN;

PROC BOXPLOT data=exemple;
      PLOT Note * groupe /
      CAXIS=black CTEXT=black
      CBOXES=black
BOXSTYLE=schematic
      IDCOLOR=black IDSYMBOL=dot;
      INSET MIN MEAN MAX STDDEV /
 HEADER='Overall Statistics'
POSITION=TM;
 INSETGROUP N min max NHIGH NLOW
NOUT /
HEADER='Extremes par groupe';
RUN;
```



## 18 PROC GPLOT

La procédure GPLOT permet de tracer le graphique d'une variable d'un jeu de données SAS en fonction d'une autre.

Syntaxe :

```
PROC GPLOT DATA=nom_du_jeu_de_données <liste-option1>;
BY nom_variable ...;
```

{Trace des nuages de points différents pour les observations de chaque groupe.}

```
PLOT Y*X='symbole' </ liste-option2>;
```

{Trace le graphique de la variable Y en fonction de la variable X en représentant chaque observation par le symbole spécifié.}

```
RUN ;
```

Les options sont les suivantes :

**<liste-option1> :**

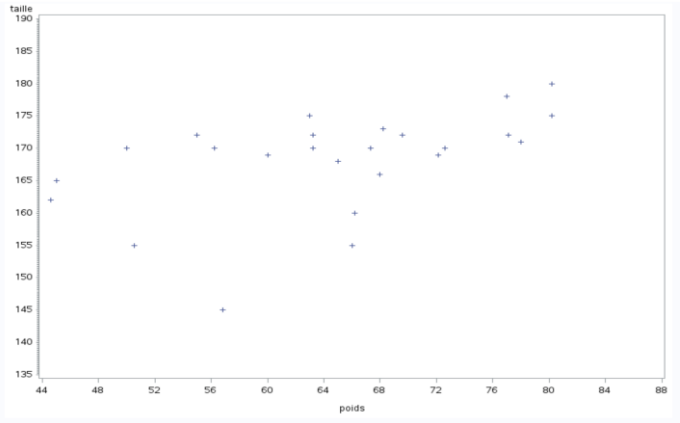
**UNIFORM** : Uniformise les échelles des axes de tous les graphiques.
**NOMISS :** Exclut du calcul des axes les observations ayant des valeurs manquantes.
**NOLEGEND** : Supprime l'impression de la légende au haut de chaque graphique.
**FORMCHAR=`onze_caractères'** : Identifie les caractères qui formeront le contour du graphique. Par défaut c'est |- - - -|+|- - - .
**HPERCENT=p** … Identifie le pourcentage (1 <= p <= 100) de la largeur de la page à utiliser pour chaque graphique.
**VPERCENT=p** : Identifie le pourcentage (1<= p <= 100) de la longueur de la page à utiliser pour chaque graphique

**</ liste-option2> :**

**HAXIS (ou VAXIS) =a to b by p** : Indique que la graduation de l'échelle sur l'axe des X (ou l'axe des Y) va de a à b par un pas p.
**HZERO (ou VZERO)** : Si les valeurs sont toutes positives et que l'option HAXIS (ou VAXIS) n'a pas été spécifiée, alors la graduation de l'échelle débutera à 0 pour l'axe des X (ou l'axe des Y).
**HREVERSE (ou VREVERSE) :** Renverse l'ordre des valeurs sur l'axe des X (ou l'axe des Y).
**HEXPAND (ou VEXPAND)** : Allonge l'axe des X (ou l'axe des Y) de façon à minimiser les marges de chaque côté du graphique et à maximiser la distance entre les graduations de l'échelle.
**HSPACE (ou VSPACE) =nombre** : Identifie le nombre de marques que l'on veut sur l'échelle entre les graduations pour l'axe des X (ou l'axe des Y).
**HREF (ou VREF) =valeur** … : Trace des droites perpendiculaires à l'axe des X (ou à l'axe des Y) vis-à-vis les valeurs spécifiées.
**HREFCHAR (ou VREFCHAR) = 'caractère'** : Indique le caractère à utiliser pour tracer les droites de l'option HREF (ou VREF). Par défaut c'est la barre verticale (|) (ou le tiret (-)).
**HPOS (ou VPOS) =l** : Indique la longueur (l) de l'axe des X (ou l'axe des Y). La valeur maximale pour l est celle de l'option LINESIZE (ou PAGESIZE) moins trois.
**BOX** : Trace une bordure tout autour du graphique.
**OVERLAY** : Supperpose tous les graphiques spécifiés dans l'instruction PLOT.
**CONTOUR** : Trace le graphique des courbes de niveaux à condition que l'instruction soit de la forme PLOT Y*X=Z; où Z est une variable numérique

### 18.1 Exemple d'application

```
PROC GPLOT DATA=exemple;
   PLOT taille*poids /
            HAXIS=44 TO 90 BY 4
            VAXIS=135 TO 190 BY 5
            HMINOR=100
            VMINOR=30;
 RUN;
 QUIT;
```




## 19 PROC UNIVARIATE

### 19.1 PROC UNIVARIATE: Statistiques descriptives

La procédure UNIVARIATE calcule des statistiques simples sur des variables numériques. Plus complète que la procédure MEANS, elle fournit :

- Des détails sur les valeurs extrêmes d'une variable.
- Le calcul de n'importe quel quantile de la distribution.
- Plus de tests : 2 tests pour l'égalité de la moyenne à une constante donnée, 1 pour le même test avec la médiane, 4 pour la normalité de la distribution, etc.
- La création de graphiques pour étudier la distribution (boxplot, histogrammes, courbes de densité).

SYNTAXE :

```
PROC UNIVARIATE DATA=nom_du_jeu_de_données <options>;
VAR liste_de_variables;{Liste des variables dont on veut des graphiques, des statistiques descriptives et des tests.}
BY nom_variable ...;{Effectue des analyses statistiques séparées pour les observations de chaque groupe.}
FREQ nom_variable; {Identifie une variable numérique dont les valeurs représentent la fréquence de chaque observation.}
ID nom_variable ...; {Liste des variables qui serviront à identifier les cinq plus grandes ainsi que les cinq plus petites observations.}
HISTOGRAM liste_de_variables / <options> ;
PROBPLOT liste_de_variables / <options> ;
OUTPUT OUT=nom <statistiques=>; {À utiliser avec l'instruction VAR, nomme un nouveau jeu de données qui contiendra les variables en plus des statistiques spécifiées.}
RUN ;
```

VAR : spécifie les variables numériques à analyser. Par défaut, toutes les variables numériques de la table en entrée sont analysées.
ID : variable d'identification pour les 5 plus petites et 5 plus grandes valeurs en sortie.
HISTOGRAM : trace un histogramme pour les variables spécifiées.
PROBPLOT : trace la distribution des variables, en la comparant à une distribution théorique spécifiée par l'utilisateur.

Les options et statistiques disponibles :

**<Options> :**

**FREQ** : Imprime la table des fréquences.
**NOPRINT** : Supprime toutes les impressions.
**NORMAL** : Teste l'hypothèse que les données proviennent d'une distribution normale.
**PLOT** : Imprime les représentations arborescentes, les diagrammes en boîtes et les graphiques de quantile-normale.
**ROUND**=unité ... Identifie les unités (>= 0) selon lesquelles arrondir les valeurs de chaque variable.
**VARDEF**= Indique le diviseur utiliser lors des calculs de variances parmi :

- **DF** (n-1) (choisi par défaut)
- **N** (n) (choisi par défaut)
- **WDF** (somme des poids moins 1)
- **WEIGHT** (somme des poids).

**<Statistiques> :**

N, NMISS, NOBS, MEAN, STDMEAN, SUM, STD, VAR, CV, USS, CSS, SKEWNESS, KURTOSIS, SUMWGT, MAX, MIN, RANGE, Q3, MEDIAN, Q1, QRANGE, P1, P5, P10, P90, P95, P99, MODE, T, PROBT, MSIGN, PROBM, SIGNRANK, PROBS, NORMAL et PROBN.

## 19.2 Exemple d'application:

```
PROC UNIVARIATE DATA=russet;
VAR gnpr;
ID pays;
HISTOGRAM gnpr / normal(mu=est sigma=est color=blue w=1);
PROBPLOT gnpr / normal(mu=est sigma=est color=blue w=1);
RUN;
```

Les sorties par défaut :

La procédure UNIVARIATE
Variable : gnpr

| Moments | | | |
|---|---|---|---|
| N | 47 | Somme des poids | 47 |
| Moyenne | 5.95787234 | Somme des observations | 280.02 |
| Ecart-type | 0.89247564 | Variance | 0.79651277 |
| Skewness | 0.01530328 | Kurtosis | -0.9070496 |
| Somme des carrés non corrigée | 1704.963 | Somme des carrés corrigée | 36.6395872 |
| Coeff Variation | 14.9797711 | Std Error Mean | 0.13018095 |

| Mesures statistiques de base | | | |
|---|---|---|---|
| Location | | Variabilité | |
| Moyenne | 5.957872 | Ecart-type | 0.89248 |
| Médiane | 5.860000 | Variance | 0.79651 |
| Mode | 4.890000 | Intervalle | 3.57000 |
| | | Ecart interquartile | 1.55000 |

Note: Le mode affiché est le plus petit des 6 modes avec un effectif de 2.

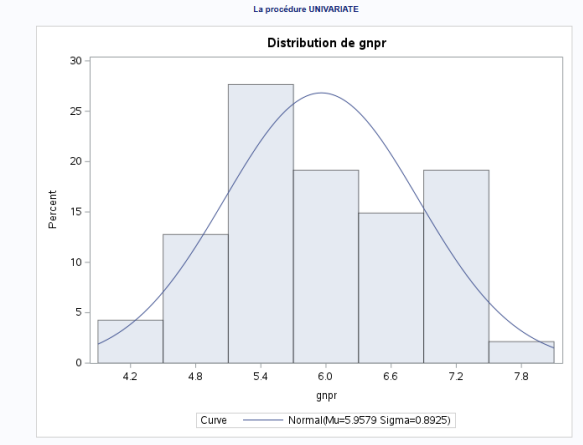
| Tests de tendance centrale : Mu0=0 | | | | |
|---|---|---|---|---|
| Test | Statistique | | p-value | |
| t de Student | t | 45.76608 | Pr > \|t\| | <.0001 |
| Signe | M | 23.5 | Pr >= \|M\| | <.0001 |
| Rang signé | S | 564 | Pr >= \|S\| | <.0001 |

**L'option ID:** `ID pays;`

| Quantiles (Définition 5) | |
|---|---|
| Niveau | Quantile |
| 100Max 100% | 7.76 |
| 99% | 7.76 |
| 95% | 7.14 |
| 90% | 7.10 |
| 75% Q3 | 6.85 |
| 50% Médiane | 5.86 |
| 25% Q1 | 5.30 |
| 10% | 4.89 |
| 5% | 4.50 |
| 1% | 4.19 |
| 0% Min | 4.19 |

| Observations extrêmes | | | | | |
|---|---|---|---|---|---|
| La plus petite | | | La plus grande | | |
| Valeur | pays | Obs | Valeur | pays | Obs |
| 4.19 | Bolivia | 1 | 7.10 | Australia | 33 |
| 4.28 | India | 37 | 7.11 | Switzerland | 44 |
| 4.50 | Libia | 10 | 7.14 | NewZealand | 41 |
| 4.88 | Taiwan | 18 | 7.42 | Canada | 35 |
| 4.89 | SouthVietnam | 16 | 7.76 | USA | 46 |

**L'option Histogram**: `HISTOGRAM gnpr / normal(mu=est sigma=est color=blue w=1) ;`



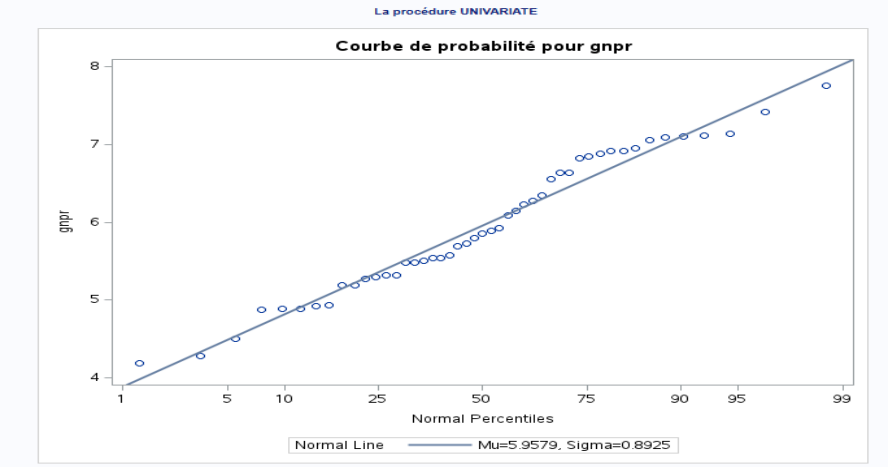

La procédure UNIVARIATE
Fitted Normal Distribution for gnpr

| Parameters for Normal Distribution | | |
|---|---|---|
| Paramètre | Symbole | Estimation |
| Mean | Mu | 5.957872 |
| Std Dev | Sigma | 0.892476 |

| Goodness-of-Fit Tests for Normal Distribution | | | | |
|---|---|---|---|---|
| Test | Statistique | | p-value | |
| Kolmogorov-Smirnov | D | 0.10957259 | Pr > D | >0.150 |
| Cramer-von Mises | W-Sq | 0.09503990 | Pr > W-Sq | 0.131 |
| Anderson-Darling | A-Sq | 0.57931650 | Pr > A-Sq | 0.130 |

| Quantiles for Normal Distribution | | |
|---|---|---|
| | Quantile | |
| Pourcentage | Observé | Estimé |
| 1.0 | 4.19000 | 3.88166 |
| 5.0 | 4.50000 | 4.48988 |
| 10.0 | 4.89000 | 4.81412 |
| 25.0 | 5.30000 | 5.35591 |
| 50.0 | 5.86000 | 5.95787 |
| 75.0 | 6.85000 | 6.55984 |
| 90.0 | 7.10000 | 7.10163 |
| 95.0 | 7.14000 | 7.42586 |
| 99.0 | 7.76000 | 8.03408 |

**L'option PROBPLOT:** `PROBPLOT gnpr/normal(mu=est sigma=est color=blue w=1) ;`



### 19.3 PROC UNIVARIATE : Test de normalité

SAS permet de réaliser quatre tests de normalité : les tests de Shapiro-Wilk, Kolmogorov-Smirnov, Cramer-von Mises et Anderson-Darling. Pour obtenir les résultats des quatre tests de normalité :

```
PROC UNIVARIATE DATA=tableSAS NORMAL;
VAR nom_variable;
RUN ;
```

### 19.1 Exemple d'application:

```
PROC UNIVARIATE DATA=russet Normal;
VAR gnpr;
RUN ;
```

| Tests de normalité | | | | |
|---|---|---|---|---|
| Test | Statistique | | p-value | |
| Shapiro-Wilk | W | 0.967763 | Pr < W | 0.2174 |
| Kolmogorov-Smirnov | D | 0.109573 | Pr > D | >0.1500 |
| Cramer-von Mises | W-Sq | 0.09504 | Pr > W-Sq | 0.1306 |
| Anderson-Darling | A-Sq | 0.579316 | Pr > A-Sq | 0.1299 |

## 20 PROC FORMAT

On peut avoir besoin de définir des formats personnalisés. Ces types de formats sont créés et gérés par la procédure FORMAT. Par exemple, un format permet de regrouper plusieurs modalités sous un même libellé. On distingue deux étapes : la création et gestion de formats : PROC FORMAT et l'utilisation de formats dans l'instruction de FORMAT existant dans de nombreuses PROC.

Syntaxe :

```
PROC FORMAT <options>;
VALUE nom_de_format
liste_de_valeurs="valeur1 formatée"
liste_de_valeurs="valeurn formatée"
;
RUN;
```

### 20.1 Exemple d'application:

On crée un format `regim` pour les valeurs de la variable caractère `demo` : cela permet de regrouper sous le même libellé `Democratie` les valeurs `stable` et `instable` :

```
Proc format;
Value $ regim
'stable','instable'='Democratie'
'dictature'='Dictature'
;
run;
```

# 21 PROC TTEST

La procédure TTEST réalise des tests d'égalité de la variance et de la moyenne de deux échantillons.
Remarque : On peut obtenir des résultats comparables avec la procédure NPAR1WAY si l'on travaille sur de petits échantillons, en utilisant alors des statistiques dites non-paramétriques.

Syntaxe

```
PROC TTEST DATA=tableSAS;
VAR variable(s)_Numérique(s);
CLASS variable_de_Classe;
RUN;
```

L'instruction VAR contient les variables dont les moyennes seront comparées. L'instruction CLASS permet de spécifier une variable dont les valeurs constituent les groupes d'observations à comparer. Cette variable ne doit avoir que deux valeurs

## 21.1 Exemple d'application:

On veut tester la différence des PIB moyens (Gnpr) entre démocraties et dictatures : on utilise la PROC FORMAT qui permet de définir un Libellé Démocratie et Dictature, ils constituent les deux groupes d'observations à comparer.

```
Proc format;
Value $ regim
'stable','instable'='Democratie'
'dictature'='Dictature'
;
run;

PROC TTEST DATA=russet;
VAR gnpr;
CLASS demo;
FORMAT demo $regim.;
RUN;
```

On obtient plusieurs tableaux et des graphiques en sortie de la `PROC TTEST` :

La procédure TTEST

Variable : gnpr

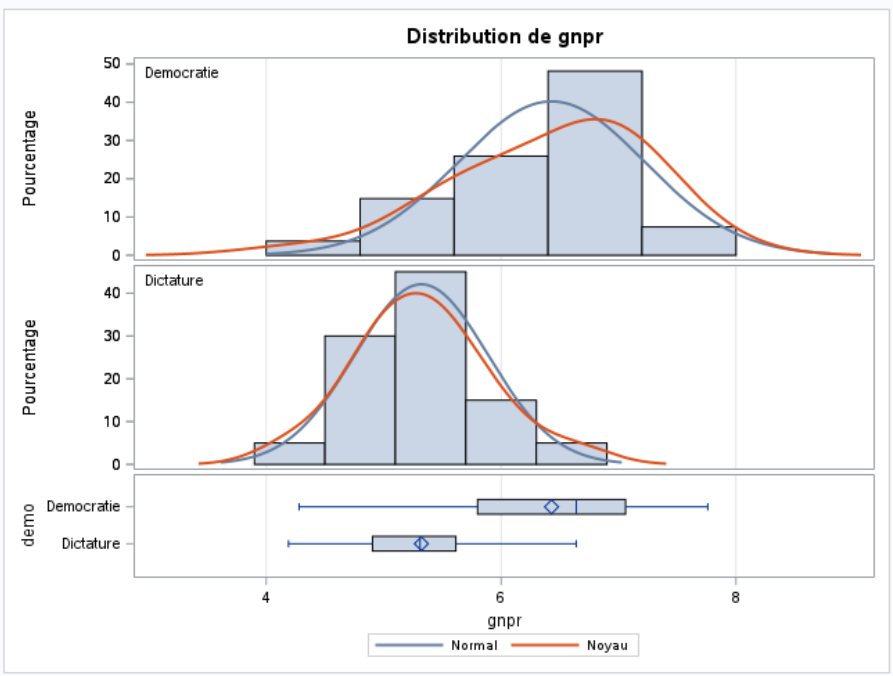
| demo | Méthode | N | Moyenne | Ec-type | Err. type | Minimum | Maximum |
|---|---|---|---|---|---|---|---|
| Democratie | | 27 | 6.4296 | 0.7934 | 0.1527 | 4.2800 | 7.7600 |
| Dictature | | 20 | 5.3210 | 0.5690 | 0.1272 | 4.1900 | 6.6400 |
| Diff (1-2) | Pooled | | 1.1086 | 0.7074 | 0.2087 | | |
| Diff (1-2) | Satterthwaite | | 1.1086 | | 0.1988 | | |

| demo | Méthode | Moyenne | IC à95% - Moyenne | | Ec-type | Ec.-type de l'IC à95% | |
|---|---|---|---|---|---|---|---|
| Democratie | | 6.4296 | 6.1158 | 6.7435 | 0.7934 | 0.6248 | 1.0873 |
| Dictature | | 5.3210 | 5.0547 | 5.5873 | 0.5690 | 0.4327 | 0.8310 |
| Diff (1-2) | Pooled | 1.1086 | 0.6883 | 1.5290 | 0.7074 | 0.5867 | 0.8910 |
| Diff (1-2) | Satterthwaite | 1.1086 | 0.7083 | 1.5089 | | | |

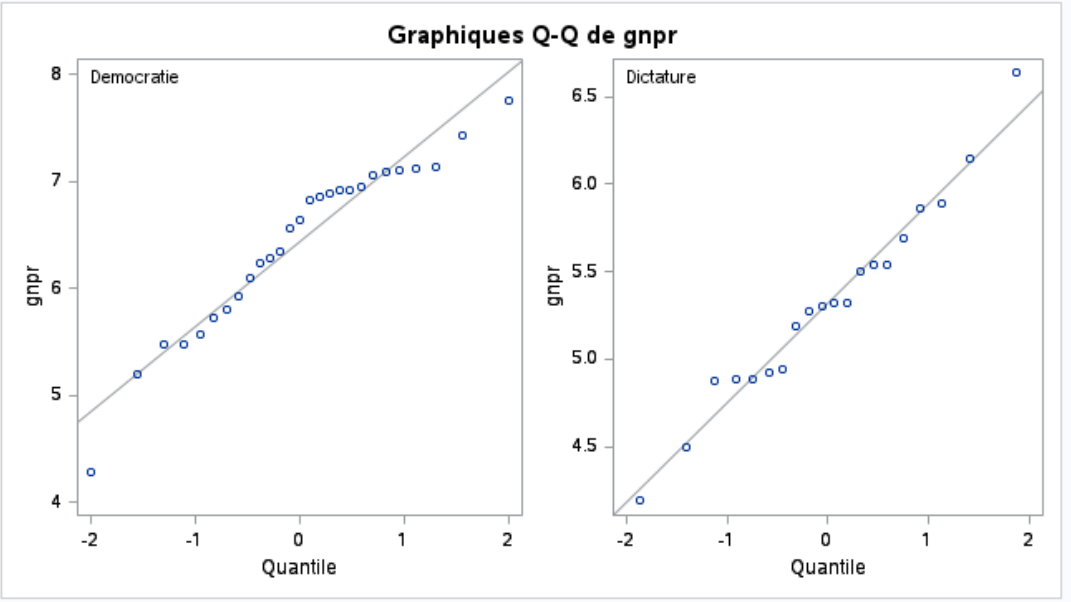
| Méthode | Variances | DDL | Valeur du test t | Pr > \|t\| |
|---|---|---|---|---|
| Pooled | Egal | 45 | 5.31 | <.0001 |
| Satterthwaite | Non égal | 44.971 | 5.58 | <.0001 |

| Egalité des variances | | | | |
|---|---|---|---|---|
| Méthode | DDL num. | DDL den. | Valeur F | Pr > F |
| Folded F | 26 | 19 | 1.94 | 0.1389 |

On obtient plusieurs tableaux et des graphiques en sortie de la `PROC TTEST` :




On obtient des tableaux et des graphiques en sortie de `la PROC TTEST` :



# 22 PROC SQL

## 22.1 Overview

Le langage SQL (Structured Query Language) est un langage utilisé à travers plusieurs applications pour extraire/modifier de l'information entre des bases/tables relationnelles. SAS a créé une procédure appelée la PROC SQL très puissante utilisant une syntaxe et une logique similaire au langage SQL, mais adapté au langage SAS. À travers un PROC SQL, nous serons capables de répliquer la majorité des traitements et procédures appris jusqu'à présent dans ce cours.

**Syntaxe** : Étant donné la quantité élevée de traitement possible avec un PROC SQL, nous verrons une syntaxe simplifiée à travers ce cours

```
PROC SQL;
CREATE TABLE BASESORTANTE AS
SELECT VAR1 as VAR1_new, VAR2 as VAR2_new, ...
FROM BASEENTRANTE
WHERE CONDITIONS
GROUP BY VAR1, VAR2,...
ORDER BY VAR1, VAR2,...;
QUIT;
```

**Note**

- L'ordre des déclarations est important.
- Selon vos besoins, plusieurs des étapes peuvent être tout simplement enlevés. Seulement le SELECT et FROM sont obligatoires.
- À noter que des virgules seront utilisés pour séparés les différentes variables

**CREATE TABLE :**

- Créer la table sortante, similaire à la déclaration DATA

**SELECT :**

- À cette étape, nous devons lister les variables que nous désirons garder dans la base de données, similaire à l'option KEEP.
- À noter que * peut être utilisé pour garder toutes les variables.
- L'option "as" peut être utilisé pour renommer une variable (optionel)
- Un format peut être créé pour chacune des variables en ajoutant format=... suivant la variable.
- Toutes nouvelles variables peuvent être créé à cette étape, il suffit d'écrire l'opération SAS pour la définir et lui définir un nom suivant le "as"

**FROM**

- Spécifie le nom de la (ou des) bases de données entrantes en lecture, similaire à la déclaration SET

**WHERE**

- Étape pour effectuer des filtres à la lecture, similaire à l'option WHERE

**GROUP BY**

- Étape pour spécifier le regroupement de variable par lequel on veut somatiser la base, similaire à la déclaration CLASS dans un PROC SUMMARY

**ORDER BY**

- Étape pour spécifier e trie voulu dans la base de données sortante, similaire à la déclaration BY d'un PROC SORT. Pour trier de façon décroissante, la variable devra être suivit de l'option "desc"

## 22.2 SQL - Query ou Requête

### 22.2.1 SQL - Requête simple:

La requête la plus simple est celle permettant de récupérer toutes les données d'une table. Toute requête d'interrogation de données commence par le mot clé SELECT et se termine normalement par un point-virgule (";"). Voici la requête permettant de récupérer l'ensemble des informations présentes dans la base de données Clients.

```
PROC SQL;
SELECT *
 FROM Clients;
QUIT;
```

**Explications :**

- Le terme SELECT indique donc que l'on veut récupérer des données ;
- Le caractère * indique que l'on veut tous les attributs de la table ;
- Le terme FROM est utilisé pour indiquer à partir de quelle table nous devons récupérer les données.
- Le terme QUIT permet de quitter la procédure SQL.

**Limitation des résultats :**

Il est parfois utile de n'avoir que les premières lignes d'un tableau, pour comprendre son contenu par exemple. Dans ce cas, il est possible d'ajouter au début de la procédure le terme OUTOBS suivi du nombre de lignes souhaité.

```
PROC SQL OUTOBS=3;
SELECT *
    FROM Clients
QUIT;
```

**Ordre des résultats :**

De même, nous sommes souvent amenés à trier les données (croissant ou décroissant). Le terme ORDER BY, à placer également à la fin de la requête permet d'effectuer un tel tri. Il est à noter que le tri se fait uniquement à l'affichage et que le tableau n'est en aucun cas modifié. De plus, cela peut être fait sur tout type de données.

Il est possible d'indiquer de deux manières le(s) attribut(s) à prendre en compte pour le tri :

- Par les noms des variables
- Par leur position dans le tableau de données

Les deux requêtes suivantes sont identiques et permettent toutes les deux d'avoir la liste des clients triée par ordre croissant de leur nom (Attribut `Nomcli` placé en 3ème position).

```
PROC SQL;
SELECT *
    FROM CLIENTS
    ORDER BY Nomcli;
QUIT;
```

```
PROC SQL;
SELECT *
    FROM CLIENTS
    ORDER BY 3;
QUIT;
```

```
PROC SQL;
SELECT *
    FROM CLIENTS
    ORDER BY NomCli DESC;
QUIT;
```

Il est toujours préférable d'utiliser la première option. Et pour trier, on utilise ASC pour l'ascendant(croissant), il est aussi possible d'ajouter le terme DESC pour indiquer que l'on veut un tri descendant(décroissant). Par défaut, c'est donc un tri croissant qui est effectué.

```
PROC SQL;
SELECT *
    FROM CLIENTS
    ORDER BY nomCLI DESC;
QUIT;
```

Il est possible d'avoir plusieurs critères de tri. Pour cela, séparez les différents attributs (nom ou position) par une virgule (,). La requête suivante permet donc de trier la table CLIENTS d'abord par leur nom, puis par leur prénom.

```
PROC SQL;
SELECT *
    FROM CLIENTS
    ORDER BY nomcli,preNomcli;
QUIT;
```

Voici la même requête que précédemment, avec le nom trié par ordre alphabétique décroissant puis par le prénom dans l'ordre croissant.

```
PROC SQL;
SELECT *
    FROM CLIENTS
    ORDER BY nomcli DESC,prenomcli;
QUIT;
```

**Restriction :**

Une restriction est une sélection de lignes d'une table, sur la base d'une condition à remplir, définie après le terme WHERE. Cette condition peut être une combinaison de comparaisons utilisant AND, OR et NOT (attention donc aux parenthèses dans ce cas).

**Opérateurs classiques :** Nous avons tous les opérateurs de comparaison classiques : =, <>, > ,>=, <, <=.

Cette requête permet de lister tous les clients habitant au 57500 :

```
PROC SQL;
SELECT *
    FROM CLIENTS
    WHERE cpcli=57500;
QUIT;
```

Si on veut le complément de cette demande, c'est-à-dire tous les clients qui n'habitent pas au code postal 57500, on utilise le symbole <> pour indiquer une non-égalité qui revient à faire `NON (cpcli=57500)`

```
PROC SQL;
SELECT *
    FROM CLIENTS
    WHERE cpcli <> 57500;
QUIT;
```

Comme mentionné précédemment, il est possible de combiner les comparaisons. La requête suivante vous permet d'avoir les clients de sexe masculin vivant à Saint-Avold avec un âge strictement inférieur à 23.

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE villecli= "Saint-Avold"
    AND sexe= "H"
    AND age < 23;
QUIT;
```

Pour les comparaisons de chaînes, il est important de faire attention à la casse (c'est-à-dire minuscules / majuscules). Par définition, donc, un "a" est différent d'un "A". Pour remédier à ce problème, il existe les fonctions UPPER() et LOWER() pour transformer une chaîne en majuscule et en minuscule respectivement.

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE UPPER(Villecli)="FORBACH";
QUIT;
```

**Données manquantes :** Les données manquantes dans SQL sont marquées d'un `NULL`. Il existe de nombreuses raisons, bonnes ou mauvaises, pour les données manquantes, et il est parfois utile de tester leur présence. Pour cela, nous utiliserons le terme `IS NULL` comme condition. Par exemple, pour lister les clients dont la ville n'est pas renseignée, il faut lancer la requête suivante.

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE Villecli IS NULL;
QUIT;
```

**Opérateurs spécifiques :** Les deux premiers opérateurs définis ci-dessous sont particulièrement utiles pour limiter la taille de la requête. Ce dernier est utile pour comparer une chaîne de caractères à une pseudo-chaîne.

BETWEEN : Cet opérateur permet de définir un intervalle fermé dans lequel l'attribut doit avoir sa valeur. La condition suivante est équivalente à `Age > = 18 AND Age <= 21`.

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE Age BETWEEN 18 AND 21;
QUIT;
```

IN : Cet autre opérateur permet de définir une liste de valeurs entre parenthèses et séparées par des virgules. La condition suivante est équivalente à `Sexe = 'H' OR Sexe = 'F'`

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE sexe IN ('H','F');
QUIT;
```

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE Age IN (18,19,20,22);
QUIT;
```

LIKE : Comme mentionné précédemment, l'opérateur LIKE permet de comparer une chaîne de caractères avec une pseudo-chaîne, dans laquelle on peut ajouter deux caractères spécifiques :

- % : une séquence de caractères, éventuellement zéro
- _ : un et un seul caractère

Par exemple, la requête suivante récupère les clients dont le nom de famille commence par un "L".

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE Nomcli LIKE 'L%';
QUIT;
```

La requête suivante permet d'avoir tous les clients qui ont un Nom composé de 5 lettres seulement

```
PROC SQL;
SELECT *
    FROM Clients
    WHERE NomCli LIKE '_____';
QUIT;
```

Notez que l'opérateur LIKE est insensible à la casse, c'est-à-dire qu'il ignore les lettres majuscules/minuscules.

**Projection :** Une projection est une sélection de colonnes d'un tableau, basée sur une liste d'attributs placés après le SELECT et séparés par une virgule. La requête suivante permet de n'avoir que les nom et prénom des clients.

```
PROC SQL;
SELECT Nomcli,
       Prenomcli
  FROM Clients;
QUIT;
```

```
PROC SQL;
SELECT Nomcli,
       Prenomcli
  FROM clients
 WHERE Age IN (18,22,20);
QUIT;
```

**Doublons :** Lors d'une projection, on est souvent en présence de doublons dans les résultats, c'est-à-dire deux ou plusieurs lignes identiques. Par exemple, quand on liste les villes des clients, on a plusieurs fois chacune des villes existantes.

```
PROC SQL;
SELECT Villecli
  FROM clients;
QUIT;
```

Cependant, dans ce cas, nous nous intéressons souvent à la liste des valeurs uniques. Pour l'obtenir, il est possible d'ajouter le terme DISTINCT juste après le SELECT, pour supprimer ces doublons.

```
PROC SQL;
SELECT DISTINCT Villecli
  FROM Clients;
 QUIT;
```

Cela fonctionne également lorsque plusieurs attributs ont été spécifiés dans le SELECT.

```
PROC SQL;
SELECT DISTINCT Villecli,
                sexe
           FROM clients;
QUIT;
```

**Renommer :** Pour améliorer la présentation, il est possible de renommer un attribut, le terme AS étant placé après l'attribut à renommer et suivi du nouveau nom.

```
PROC SQL;
SELECT Nomcli AS Nouveau_Nom
  FROM clients;
QUIT;
```

### 22.2.2 SQL – INNER JOIN - LEFT JOIN

- La commande INNER JOIN, également appelée EQUIJOIN, est un type de jointure très courant pour lier plusieurs tables entre elles. Cette commande renvoie des enregistrements lorsqu'au moins une ligne dans chaque colonne correspond à la condition.

```
PROC SQL;
    SELECT *
      FROM Clients cl
INNER JOIN Locations Lo
        ON cl.codecli=lo.codecli;
QUIT;
```

```
PROC SQL;
   SELECT *
      FROM Clients cl
INNER JOIN Locations Lo
        ON cl.codecli=lo.codecli
     WHERE cl.cpcli=57800;
QUIT;
```

- La commande LEFT JOIN (également appelée LEFT OUTER JOIN) est un type de jointure entre 2 tables. Cela vous permet de lister tous les résultats du tableau de gauche même s'il n'y a pas de correspondance dans le deuxième tableau.

```
PROC SQL;
   SELECT *
      FROM Locations lo
 LEFT JOIN Films fi
        ON lo.codefilm=fi.codefilm;
QUIT;
```

```
PROC SQL;
   SELECT *
      FROM Locations Lo
 LEFT JOIN Films fi
        ON lo.codefilm=fi.codefilm
     WHERE fi.codefilm IN (1,5,7);
 QUIT;
```

```
PROC SQL;
    SELECT *
      FROM Clients cl
INNER JOIN Locations Lo
        ON cl.codecli=lo.codecli
INNER JOIN Films fi
        ON lo.codefilm =fi.codefilm
     WHERE cl.cpcli=57800;
QUIT;
```

```
PROC SQL;
   SELECT *
      FROM Clients cl
 LEFT JOIN Locations Lo
        ON cl.codecli=lo.codecli
 LEFT JOIN Films fi
        ON lo.codefilm =fi.codefilm
     WHERE cl.cpcli=57800;
QUIT;
```

```
PROC SQL;
    SELECT *
      FROM Clients cl
 LEFT JOIN Locations Lo
        ON cl.codecli=lo.codecli
INNER JOIN Films fi
        ON lo.codefilm =fi.codefilm
     WHERE cl.cpcli=57800;
QUIT;
```

### 22.2.3 SQL – Tableau des produits

| Refprod | Nomprod | NoFour | CodeCateg | QteParUnit | PrixUnit | UnitesStock | UnitesCom | NiveauReap | Indisponible |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Chai | 1 | 1 | 10 boites x 20 sacs | 90 | 39 | 0 | 10 | 0 |
| 2 | Chang | 1 | 1 | 24 bouteilles (1 litre) | 95 | 17 | 40 | 25 | 0 |
| 3 | Aniseed Syrup | 1 | 2 | 12 bouteilles (550 ml) | 50 | 13 | 70 | 25 | 0 |
| 4 | Chef Antons Cajun Seasoning | 2 | 2 | 48 pots (6 onces) | 110 | 53 | 0 | 0 | 0 |
| 5 | Chef Antons Gumbo Mix | 2 | 2 | 36 boites | 106.75 | 0 | 0 | 0 | 1 |
| 6 | Grandmas Boysenberry Spread | 3 | 2 | 12 pots (8 onces) | 125 | 120 | 0 | 25 | 0 |
| 7 | Uncle Bobs Organic Dried Pears | 3 | 7 | 12 cartons (1 kg) | 150 | 15 | 0 | 10 | 0 |
| 8 | Northwoods Cranberry Sauce | 3 | 2 | 12 pots (12 onces) | 200 | 6 | 0 | 0 | 0 |
| 9 | Mishi Kobe Niku | 4 | 6 | 18 cartons (500 g) | 485 | 29 | 0 | 0 | 1 |
| 10 | Ikura | 4 | 8 | 12 pots (200 g) | 155 | 31 | 0 | 0 | 0 |
| 11 | Queso Cabrales | 5 | 4 | 1 carton (1 kg) | 105 | 22 | 30 | 30 | 0 |
| 12 | Queso Manchego La Pastora | 5 | 4 | 10 cartons (500 g) | 190 | 86 | 0 | 0 | 0 |
| 13 | Konbu | 6 | 8 | 1 boites (2 kg) | 30 | 24 | 0 | 5 | 0 |
| 14 | Tofu | 6 | 7 | 40 cartons (100 g) | 116.25 | 35 | 0 | 0 | 0 |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| 15 | Genen Shouyu | 6 | 2 | 24 bouteilles (250 ml) | 77.5 | 39 | 0 | 5 | 0 |
| 16 | Pavlova | 7 | 3 | 32 boites (500 g) | 87.25 | 29 | 0 | 10 | 0 |
| 17 | Alice Mutton | 7 | 6 | 20 boites (1 kg) | 195 | 0 | 0 | 0 | 1 |
| 18 | Carnarvon Tigers | 7 | 8 | 1 carton (16 kg) | 312.5 | 42 | 0 | 0 | 0 |
| 19 | Teatime Chocolate Biscuits | 8 | 3 | 10 boites x 12 pièces | 46 | 25 | 0 | 5 | 0 |
| 20 | Sir Rodneys Marmalade | 8 | 3 | 30 boites | 405 | 40 | 0 | 0 | 0 |
| 21 | Sir Rodneys Scones | 8 | 3 | 24 cartons x 4 pièces | 50 | 3 | 40 | 5 | 0 |
| 22 | Gustafs | 9 | 5 | 24 cartons (500 g) | 105 | 104 | 0 | 25 | 0 |
| 23 | Tunnbr | 9 | 5 | 12 cartons (250 g) | 45 | 61 | 0 | 25 | 0 |
| 24 | Guaran | 10 | 1 | 12 canettes (355 ml) | 22.5 | 20 | 0 | 0 | 1 |

## 22.3 SQL – Ordre des commandes SELECT

Cette commande SQL est relativement courante car il est très courant de devoir lire des données à partir d'une base de données. Il existe plusieurs commandes qui vous permettent de mieux gérer les données que vous souhaitez lire. Voici un petit aperçu des fonctionnalités possibles :

- Joindre un autre tableau aux résultats
- Filtrer pour ne sélectionner que certains enregistrements
- Classer les résultats
- Regroupez les résultats pour ne faire que des statistiques (score moyen, prix le plus élevé, etc.)

Une requête SELECT peut être assez longue. Juste pour référence, voici une requête SELECT qui a presque toutes les commandes possibles :

```
PROC SQL;
   SELECT *
      FROM table
     WHERE condition
  GROUP BY expression
    HAVING condition
{UNION | INTERSECT | EXCEPT}
  ORDER BY expression
    OFFSET start;
QUIT;
```

**Note** : cette requête imaginaire sert de rappel principal pour savoir dans quel ordre chacune des commandes est utilisée dans une requête SELECT.

**Combiner AND et OR :** Gardez à l'esprit que les opérateurs peuvent être combinés pour effectuer des recherches puissantes**.**

```
PROC SQL;
    SELECT *
      FROM produit
     WHERE (CodeCateg=1 AND PrixUnit > 50)
        OR (CodeCateg=3 AND PrixUnit < 90);
    QUIT;
```

## 22.4 SQL – Requêtes et calculs arithmétiques

**Calcul simple** : Il est possible d'effectuer des calculs arithmétiques dans un SELECT, en utilisant les opérateurs classiques : +, -, *, /, (). Voici un premier exemple de calcul dans une requête. Evidemment, vous pouvez aussi vouloir faire une projection en même temps, en n'affichant que la référence du produit.

```
PROC SQL;
SELECT *,
      UnitesStock + UnitesCom
 FROM produit;
 QUIT;
```

```
PROC SQL;
SELECT RefProd,
       UnitesStock + UnitesCom
  FROM produit;
QUIT;
```

**Renommer :** Pour plus de lisibilité, il est d'usage de renommer un calcul, grâce à AS.

```
PROC SQL;
SELECT RefProd AS Reference,
       UnitesStock + UnitesCom AS Available_units
  FROM produit;
QUIT;
```

Il est important d'avoir un code lisible (débogage plus simple, compréhension facile par un autre, réutilisation plus facile, …). Dans cet exemple, nous utilisons la multiplication * pour calculer la quantité en stock pour chaque produit, égale au prix unitaire multiplié par la quantité de produits en stock.

```
PROC SQL;
SELECT RefProd,
       PrixUnit * UnitesStock AS Amount_in_stock
 FROM produit;
QUIT;
```

**Combinaison de clauses :** Puisque nous sommes dans une requête SELECT, nous pouvons évidemment utiliser toutes les restrictions que nous voulons dans le WHERE.

```
PROC SQL;
SELECT RefProd,
       PrixUnit * UnitesStock AS
Amount_in_stock_unavailable;
 FROM  produit
WHERE  Indisponible=1;
QUIT;
```

```
PROC SQL OUTOBS=5;
SELECT RefProd,
       PrixUnit * UnitesStock AS
Amount_in_stock
  FROM produit
 WHERE Indisponible=1
ORDER BY 2 DESC
QUIT;
```

Et bien sûr, nous pouvons également trier le résultat, en utilisant ORDER BY, et nous limiter à n lignes, en utilisant OUTOBS. Nous avons donc ici les trois produits indisponibles avec la plus grande quantité en stock.

**Calculs complexes :** Les calculs peuvent être un peu plus complexes, grâce à l'utilisation de parenthèses. Par exemple, considérons que nous voulons conserver au moins 5 unités de chaque produit. Dans la requête suivante, nous calculons la quantité en stock directement disponible, en tenant compte de la contrainte précédente.

```
PROC SQL;
SELECT RefProd,
       PrixUnit *(UnitesStock-5)
  FROM produit
 WHERE UnitesStock >= 5;
QUIT;
```

Toute expression mathématique combinant des opérateurs classiques est donc acceptable à ce niveau.

**Arrondir :** Il est possible d'obtenir l'arrondi d'un nombre réel grâce à la fonction ROUND(). Dans l'exemple ci-dessous, nous calculons une augmentation de 50 % des prix des produits.

```
PROC SQL;
SELECT RefProd,
       ROUND(PrixUnit *1.5,2) AS Nouveau_prix
 FROM produit;
QUIT;
```

## 22.5 SQL – Traitement conditionnel

Nous avons vu comment se restreindre à un sous-ensemble d'une table via les restrictions de la partie WHERE d'une requête. Il existe également une possibilité de traitement conditionnel avec le terme CASE. Dans un SELECT, celui-ci va nous permettre de conditionner la valeur d'une colonne par les valeurs d'autres colonnes.

**Comparaison des valeurs (égalité) :** La première utilisation de cette commande est de comparer un attribut avec un ensemble de valeurs. La comparaison étant l'égalité, il s'agit principalement d'attributs de type texte ou à nombre restreint de valeurs. Dans ce cas, l'ordre des comparaisons utilisant WHEN n'a pas d'importance.

```
PROC SQL;
SELECT codecli,Nomcli,prenomcli,Sexe,
  CASE Sexe
      WHEN "H" THEN "Homme"
      WHEN "F" THEN "Femme"
      ELSE Sexe
  END AS Titre
 FROM Clients ;
QUIT ;
```

Vous pouvez également l'utiliser pour mettre un message en fonction de la valeur d'un attribut. Ci-dessous, nous affichons à partir de quel niveau de stock le produit doit être commandé pour le réapprovisionnement. S'il est à 0, nous indiquons qu'il n'y a pas de niveau minimum. Sinon, on utilise la valeur stockée dans LevelReap (NiveauReap) pour créer le message.

```
PROC SQL;
SELECT Refprod, Nomprod, NiveauReap,
 CASE NiveauReap
      WHEN 0 THEN Pas_de_niveau_minimum"
      ELSE "Réapprovisionnement à partir de"|| NiveauReap||"unités
restantes"
  END AS raapprovisionnement
 FROM Produit ;
QUIT ;
```

**Comparaison aux valeurs (infériorité ou supériorité) :** Pour pouvoir comparer un attribut avec des valeurs mais autrement que par égalité, il est également possible d'utiliser un CASE. Dans l'exemple de la diapositive suivante, nous comparons le prix unitaire des produits avec deux valeurs seuils. Si le prix est inférieur ou égal à 50, alors le produit est considéré comme faisant partie de la gamme de prix bas. Ensuite, on teste pour savoir s'il est inférieur ou égal à 500 et si c'est le cas, le produit sera dans la moyenne. Enfin, par défaut, le produit sera dans la gamme luxe

```
PROC SQL;
SELECT Refprod, Nomprod, PrixUnit,
 CASE
      WHEN PrixUnit<=50 THEN "Bas prix"
      WHEN 50 < PrixUnit <=50 THEN "Prix moyens"
      ELSE "Produits de luxe"
  END AS Gamme
 FROM Produit ;
QUIT ;
```

**Comparaison entre les attributs :** Enfin, il est également possible d'utiliser ce test CASE pour comparer plusieurs attributs entre eux. Dans l'exemple ci-dessous, nous souhaitons afficher un message en fonction de l'action à effectuer ou non pour le réapprovisionnement. Si le produit a déjà été commandé (c'est-à-dire UnitesCom> 0), alors il est indiqué. En revanche, si ce n'est pas le cas, mais que le stock est inférieur au niveau de réapprovisionnement, alors il est indiqué que le produit doit être commandé. Pour les produits qui ne sont plus en stock (et dont le niveau de réapprovisionnement est égal à 0), nous indiquons simplement qu'il n'y a plus de produits disponibles. Pour les autres, il n'y a rien à faire.

```
PROC SQL;
SELECT Refprod, UnitesStock, UnitesCom, NiveauReap,
 CASE
      WHEN UnitesCom > 0 THEN "Déjà commandé"
      WHEN UnitesCom < NiveauReap THEN "A Commander "
      WHEN UnitesCom == 0 THEN "N'est plus en stock "
      ELSE " Rien à faire "
  END AS Informations
 FROM Produit ;
QUIT ;
```

## 22.6 SQL – Agrégation

**Count** : Counts sont largement utilisés en SQL. D'une part, parce qu'il est important de savoir combien de clients, de commandes, … D'autre part, c'est un moyen intéressant de contrôler son travail lors de l'écriture d'un programme complexe, nécessitant plusieurs requêtes. C'est le premier agrégat (ou calcul agrégé) à voir. Nous utilisons la commande COUNT() pour cela :

- **Nombre de lignes dans un tableau :** Pour calculer le nombre de lignes d'un tableau, quelles que soient les valeurs de celui-ci, le plus correct est d'utiliser COUNT(*). L'étoile ici indique que nous prenons la table entière, comme dans une requête de table SELECT * FROM. On a donc ici le nombre de clients contenus dans la base de données :

```
PROC SQL;
SELECT COUNT(*)
  FROM clients;
QUIT;
```

- **Nombre de valeurs d'un attribut :** Il est possible de spécifier un attribut dans le COUNT(). Cela comptera le nombre de lignes avec une valeur dans cet attribut. Si nous l'utilisons sur une clé primaire, nous devrions obtenir le même résultat qu'avant :

```
PROC SQL;
SELECT COUNT(codecli)
  FROM clients;
 QUIT;
```

**Note** : Par contre, si on indique un attribut dans lequel il manque des valeurs (i.e. NULL en SQL), on n'aura pas le même résultat. On aura donc le nombre de valeurs non nulles de cet attribut.

- **Nombre de valeurs distinctes d'un attribut :** Pour aller plus loin, il est également possible d'ajouter la clause DISTINCT avant l'attribut, pour obtenir le nombre de valeurs distinctes de cet attribut. Nous avons donc ci-dessous le nombre de valeurs distinctes de l'attribut CodeCateg. Ce qui nous donne le nombre de CodeCateg :

```
PROC SQL;
SELECT COUNT(DISTINCT CodeCateg)
  FROM produit;
QUIT;
```

Ce nombre correspond au nombre de lignes dans le tableau résultant de la requête suivante :

```
PROC SQL;
SELECT DISTINCT CodeCateg
  FROM produit;
QUIT;
```

- **Restriction dans le dénombrement :** Pour compter des sous-ensembles d'une table, il est évidemment possible d'ajouter des restrictions à la requête. Par exemple, ci-dessous, nous calculons le nombre de produits indisponibles dans la base de données :

```
PROC SQL;
SELECT COUNT(*)
  FROM produit
 WHERE Indisponible=1;
 QUIT;
```

## 22.7 SQL –Calculs statistiques simples

Outre les comptages, il est possible de faire quelques calculs statistiques de base, tels que somme, moyenne, minimum et maximum

**Sum** : La fonction SUM (attribut) permet donc d'additionner la somme des valeurs non nulles de l'attribut passé en paramètre. Bien entendu, ce calcul peut se faire suite à une restriction, c'est-à-dire sur un sous-ensemble de la table. Dans la deuxième sélection, nous calculons le nombre d'unités en stock pour tous les produits de la catégorie 1.

```
PROC SQL;
SELECT SUM(UnitesStock)
  FROM Produit
 WHERE CodeCateg=1;
QUIT;
```

**AVG (Moyenne) :** Bien qu'avec un SUM() et un COUNT(), on puisse obtenir la moyenne, il existe la fonction AVG(attribut) (pour la moyenne) pour la calculer directement. La première requête ci-dessous permet de calculer le prix unitaire moyen des produits. Dans la plupart des cas, il faudra améliorer la lisibilité du résultat en arrondissant les valeurs, très souvent à 2 décimales, comme ci-dessous.

```
PROC SQL;
SELECT AVG(PrixUnit)
  FROM Produit;
QUIT;
```

```
PROC SQL;
SELECT ROUND(AVG(PrixUnit),2)
  FROM Produit;
QUIT;
```

**Minimum et maximum :** Enfin, deux autres fonctions utiles sont disponibles : MIN(attribut) et MAX(attribut), permettant d'obtenir respectivement le minimum et le maximum d'un attribut, sans prendre en compte les valeurs nulles. On obtient donc avec cette demande le prix minimum et le prix maximum

```
PROC SQL;
SELECT MIN(PrixUnit),
       MAX(PrixUnit)
  FROM Produit;
 QUIT;
```

## 22.8 SQL –Agrégats par attribut

Si l'on souhaite avec un calcul statistique selon un critère (que l'on appellera critère d'agrégation), il n'est pas nécessaire de répéter la requête autant de fois qu'il y a de valeurs pour le critère. Pour cela, il existe la commande GROUP BY, qui permet d'effectuer un calcul agrégé (COUNT(), SUM(), AVG(), ...) pour chaque valeur du critère d'agrégation. Le GROUP BY doit être placé après le WHERE dans la requête.

- **Agrégat simple :** Le critère d'agrégation est souvent un attribut unique (par exemple, on veut le nombre d'élèves de chaque sexe). Le premier exemple que nous allons voir est le comptage. Donc, nous aimerions voir l'attribut CodeCateg affiché en plus du compte.

```
PROC SQL;
SELECT CodeCateg, COUNT(*)
  FROM Produit ;
QUIT;
```

Une fois exécuté, on se rend compte qu'il ne retourne qu'une seule ligne, le nombre total de CodeCateg. Pour avoir un résultat correct, vous devez spécifier le critère d'agrégation (ici l'Indisponible) dans la clause GROUP BY, comme ci-dessous :

```
PROC SQL;
SELECT Indisponible, COUNT(*)
  FROM Produit
GROUP BY Indisponible;
QUIT;
```

```
PROC SQL;
SELECT Indisponible,
       COUNT(*) AS Nombre_de_clients
  FROM Produit
GROUP BY Indisponible
ORDER BY 2 DESC;
QUIT;
```

Ce mécanisme fonctionne bien évidemment avec tous les autres calculs agrégés que nous avons vu précédemment (SUM(), AVG(), …).

- **Combinaison d'agrégats :** Il est également possible de calculer directement plusieurs agrégats dans une seule requête, comme ci-dessous. Nous cherchons donc à avoir, pour chaque fournisseur : le nombre de produits, le prix moyen (arrondi à l'entier), le prix minimum et le prix maximum

```
PROC SQL;
SELECT NoFour,
       COUNT(*) AS Nombre_de_produits,
       ROUND(AVG(PrixUnit)) AS Prix_moyen,
       MIN(PrixUnit) AS Prix_Minimum,
       MAX(PrixUnit) AS Prix_Maximum
  FROM Produit
ORDER BY NoFour;
QUIT;
```

- **Agrégat complexe :** Par contre, il arrive que l'on souhaite avoir un critère d'agrégation prenant en compte plusieurs attributs. Dans ce cas, les attributs doivent être spécifiés à la fois dans le SELECT et dans le GROUP BY. Ci-dessous, nous cherchons donc à connaître le nombre de produits pour chaque fournisseur et chaque catégorie.

```
PROC SQL;
SELECT NoFour,
       CodeCateg,
       COUNT(*)
  FROM Produit
GROUP BY NoFour, CodeCateg;
QUIT;
```

Plus généralement, il est obligatoire que les attributs présents dans le SELECT soient également présents dans le GROUP BY. Sinon, le résultat ne correspondra pas à ce que l'on cherche à réaliser, et ce n'est pas toujours facile à repérer.

Parfois, nous voulons restreindre les résultats avec une condition sur un calcul agrégé.

Pour effectuer ces restrictions, il est nécessaire d'utiliser la clause HAVING, nécessairement située après le GROUP BY. Dans notre exemple, nous devons donc écrire la requête suivante :

```
PROC SQL;
SELECT NoFour,
       COUNT(*) AS Nombre_de_produits,
       ROUND(AVG(PrixUnit)) AS Prix_moyen,
       MIN(PrixUnit) AS Prix_Minimum,
       MAX(PrixUnit) AS Prix_Maximum
  FROM Produit
ORDER BY NoFour
HAVING COUNT(*) >=10;
QUIT;
```