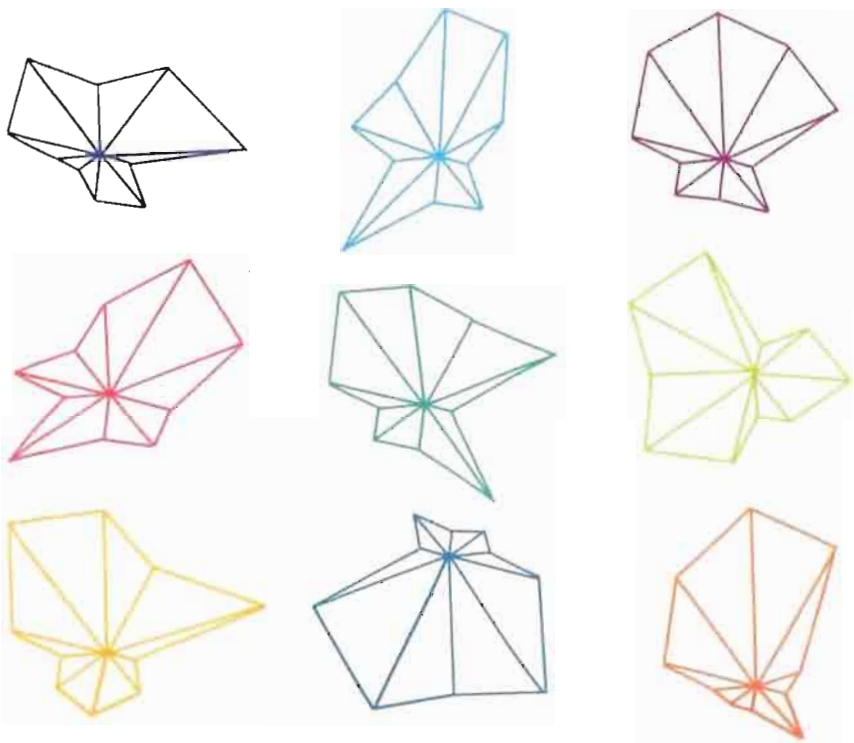
L'essentiel de STATA®

Philippe BOCQUIER



GLOBAL DESIGN

Ritme Informatique

RITME INFORMATIQUE

L'essentiel de

GLOBAL DESIGN
Marketing et Edition

Global Design
10, place Vendôme
75001 Paris

RITME INFORMATIQUE

Ritme Informatique
34, boulevard Haussmann
75009 Paris

Avant-Propos

Ce manuel s'adresse aux nouveaux utilisateurs du logiciel Stata, qu'ils soient étudiants, chercheurs ou professionnels des sciences biologiques, économiques ou sociales. Son but est d'indiquer concrètement aux utilisateurs de Stata les moyens de traiter leurs données statistiques. Il ne remplace pas les manuels de référence en anglais fournis par Stata mais permet au néophyte francophone de mieux s'orienter dans les nombreuses fonctions du logiciel. En outre, pour aider le lecteur, chaque section est agrémentée d'exercices et ponctuée d'astuces pour une meilleure utilisation du logiciel.

Un premier chapitre présente globalement le logiciel et ses différentes versions en fonction du matériel utilisé. Le deuxième chapitre résume ce que Stata sait faire et ce qui fait son originalité. Le troisième chapitre explique comment prendre en main le logiciel : configuration de la mémoire, chargement et sauvegarde des fichiers, syntaxe des commandes.

La gestion des fichiers de données fait l'objet d'un chapitre à part entière : création d'un nouveau fichier, transfert d'un fichier à partir d'un autre logiciel, gestion de la mémoire, lisibilité des données, combinaison de fichiers hiérarchisés ou non. Le chapitre suivant explique comment créer et modifier les variables : opérateurs et fonctions, utilisation de l'éditeur, adressage par ligne, calcul sur les fichiers hiérarchisés.

Les statistiques simples et leur représentation graphique font l'objet du plus long chapitre de ce manuel. Nous y expliquons comment produire des tableaux complexes croisant plusieurs variables, comment utiliser les quatre types de pondération dans le cas de données échantillonnées ou non, et comment obtenir des graphiques très informatifs.

Dans le dernier chapitre de ce manuel, nous montrons comment s'orienter dans le dédale complexe de modèles proposés par Stata, à partir d'une méthode originale basée sur la classification de la variable dépendante dans la régression.

Table des Matières

1 MAIS QU'EST-CE QUE C'EST STATA, DIS-DONC ?

1.1 D'abord, comment ça se prononce ?

Comme la plupart des logiciels de statistique commerciaux, Stata est un logiciel qui a été conçu par des anglophones et qui s'adresse, dans leur langue, aux anglophones. Cela ne veut pas dire que les autres n'y comprendront rien : il n'est heureusement pas nécessaire d'avoir l'anglais pour langue maternelle pour utiliser le logiciel.

En fait, c'est un peu la règle générale en informatique : on apprend un logiciel, en même temps qu'on apprend le langage spécifique dans lequel il est écrit. Si cela peut vous rassurer, les anglophones ne comprennent pas toujours le langage qui est utilisé dans les logiciels informatiques. De nouveaux sens sont parfois associés à des mots bien connus, et de nouveaux mots ou concepts sont souvent inventés lorsque les anciens ne conviennent pas aux concepteurs des logiciels.

Mais il faut admettre qu'une connaissance de base de l'anglais écrit sera utile, et il est probable aussi que l'apprentissage du logiciel vous fera apprendre de nouveaux mots, quel que soit votre niveau en anglais.

Mais, comment prononce-t-on Stata ? Ce mot rime avec le mot *data* qui veut dire « données » en anglais, et que l'on peut prononcer aussi bien « daata » que « dèïta ». En somme, pour une fois, les francophones peuvent s'en tenir à la prononciation à la française de Stata, car ils seront à peu près sûr d'être compris de tout le monde.

1.2 Et puis, à quoi ça sert ?

Stata est un logiciel statistique commercial pour manipuler, analyser et représenter graphiquement des données. Il associe puissance de calcul et convivialité. Il est particulièrement bien adapté à la recherche médicale, aux biostatistiques, aux mathématiques appliquées aux sciences sociales (sociologie, démographie, psychologie...), et à l'économétrie, et en général au traitement des données d'enquête.

Stata est un logiciel interactif qui possède de nombreuses procédures d'exploration des données, notamment des graphiques très informatifs. Il propose aussi des procédures statistiques avancées : méthodes d'estimation robustes (régression sur les quantiles, *bootstrapping*...), analyse de survie (modèle de Cox avec troncatures à gauche et sorties temporaires d'observation...), analyse de séries temporelles, table d'épidémiologie, etc.

Il est écrit dans un langage de programmation matriciel accessible à l'utilisateur averti, qui peut ainsi modifier des procédures déjà existantes ou créer de nouvelles procédures statistiques. Ainsi, l'utilisateur peut évoluer et devenir un concepteur, en proposant ses propres programmes aux autres utilisateurs du logiciel dans le *Stata Technical Bulletin*, bimensuel qui diffuse les nouveaux programmes auprès de ses abonnés. Cela fait de Stata un produit particulièrement évolutif puisque le logiciel se nourrit en quelque sorte du travail de ses utilisateurs les plus chevronnés.

1.3 Encore un nouveau logiciel ?

Bon, pas si nouveau que ça : Stata existe depuis 1985. Mais il a considérablement évolué depuis ce temps-là, essentiellement par l'introduction de nouvelles procédures, et aussi par l'adaptation à l'environnement Windows.

Certains logiciels offrent une plus grande variété de statistiques et de possibilités graphiques et cartographiques. Mais rares sont ceux qui vont aussi loin dans le domaine de l'analyse des données d'enquêtes. De plus, le maniement des autres logiciels

nécessite souvent d'écrire de longues procédures, et leur exécution est lente. Il faut en outre attendre plusieurs mois, voir des années avant que de nouvelles procédures statistiques soient intégrées à ces logiciels. Au contraire, Stata évolue très rapidement, en fonction des besoins de ses utilisateurs.

Stata propose deux versions de son logiciel :

1. Une version simple (*Small Stata*) pour des ordinateurs de petite capacité et qui ne peut accueillir des fichiers de plus de 1 000 observations et 99 variables ; cette version n'est vraiment utile que pour dispenser des cours sur des ordinateurs de première génération.

2. Une version standard (appelée encore *Intercooled Stata* mais qu'on appellera simplement Stata) qui peut accueillir des fichiers largement plus gros que ceux que vous traitez habituellement ; en particulier, le nombre d'observations n'est limité que par la mémoire vive de votre ordinateur.

Pour analyser vos propres données, vous n'utiliserez jamais *Small Stata*, et nous n'y ferons plus référence dans la suite de ce manuel. Si vous n'avez pas la version standard (*Intercooled*) de Stata, achetez-la immédiatement. Si vous n'êtes pas convaincu, vous le serez après avoir consulté ce manuel.

1.4 Est-ce que ça marche sur mon ordinateur ?

Stata charge les fichiers de travail en mémoire vive. Cela a un petit inconvénient et un gros avantage.

L'inconvénient est que la taille des fichiers qui peuvent être lus et modifiés dépend directement de la taille de la mémoire vive disponible, c'est-à-dire non utilisée par le système d'exploitation de votre ordinateur.

Si vous disposez de suffisamment de mémoire vive, alors Stata conserve son avantage majeur : la rapidité d'exécution, dont vous serez étonné si vous êtes un nouvel utilisateur, et dont vous aurez du mal à vous passer si vous avez déjà goûté à Stata. De plus, Stata utilise un format de stockage particulièrement condensé qui permet de conserver des fichiers deux à trois fois

moins volumineux que ceux des logiciels statistiques équivalents.

Le problème de la taille de la mémoire vive peut être contourné en utilisant la mémoire virtuelle, mais avec une incidence directe sur la rapidité d'exécution. En somme, si vous avez à vous plaindre, ce sera plutôt de votre ordinateur que de Stata.

La question de l'espace disponible en mémoire ne se posera véritablement que pour les rares utilisateurs de gros fichiers de données. Pour ceux-là, nous conseillons de n'utiliser que les variables ou les observations qui sont utiles pour une analyse spécifique : une procédure existe dans Stata pour une telle sélection, tout en maintenant l'intégrité du fichier original.

Stata fonctionne sous DOS, sous Windows 3.1 ou 95, sous Macintosh, sous Unix et sous d'autres systèmes moins courants... à condition évidemment de commander la version compatible avec l'un ou l'autre de ces systèmes d'exploitation. Dans le présent manuel, nous ne nous étendrons pas sur les différences entre les versions de Stata dans la mesure où elles ne concernent que la mise en marche du logiciel et l'adressage des fichiers. Ces procédures sont expliquées dans le volume des manuels Stata intitulé « *Getting started* » écrit pour chaque système d'exploitation.

Sinon, une fois le fichier chargé en mémoire, les procédures sont exactement les mêmes d'un système d'exploitation à l'autre. De plus, les fichiers de données sont parfaitement lisibles d'un système à l'autre : votre voisin utilise un Mac, pas de problème, vous pourrez quand même lire son fichier sous Windows à la maison, ou sous Unix au bureau.

2 CE QUE STATA PEUT FAIRE

Ce logiciel offre un large éventail de procédures, allant de la simple calculette à la mise au point par l'utilisateur de ses propres modèles d'estimation par le maximum de vraisemblance, en passant par une grande variété de commandes statistiques.

2.1 Des calculs directs

Avant, après ou pendant le traitement des données, Stata permet le calcul arithmétique et l'accès à différentes fonctions statistiques :

```
. display 25+(3443/(30^2))
28.825556

. display round(2*(1-normprob(1.96)),.01)
.05
```

Les résultats des commandes de Stata peuvent être réutilisés pour faire des calculs particuliers :

```
. summarize pop

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
     pop |      50     4518149     4715038      401851   2.37e+07

. di _result(6)-_result(5)
23266051
```

On peut aussi accéder directement aux coefficients des modèles de régression pour calculer de nouvelles variables :

```
. regress pctact pcturb

  Source |       SS       df       MS                  Number of obs =      50
---------+------------------------------                F(  1,    48) =   10.70
   Model |  .003514414     1  .003514414                Prob > F      =  0.0020
Residual |   .01575887    48   .00032831                R-squared     =  0.1823
---------+------------------------------                Adj R-squared =  0.1653
   Total |  .019273284    49  .000393332                Root MSE      =  .01812

------------------------------------------------------------------------------
  pctact |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  pcturb |    .058773   .0179636     3.272   0.002       .0226548    .0948912
   _cons |   .5628098   .0122964    45.770   0.000       .5380861    .5875334
------------------------------------------------------------------------------

. generate ajpctact= _b[_cons] + _b[pcturb]*pcturb
```

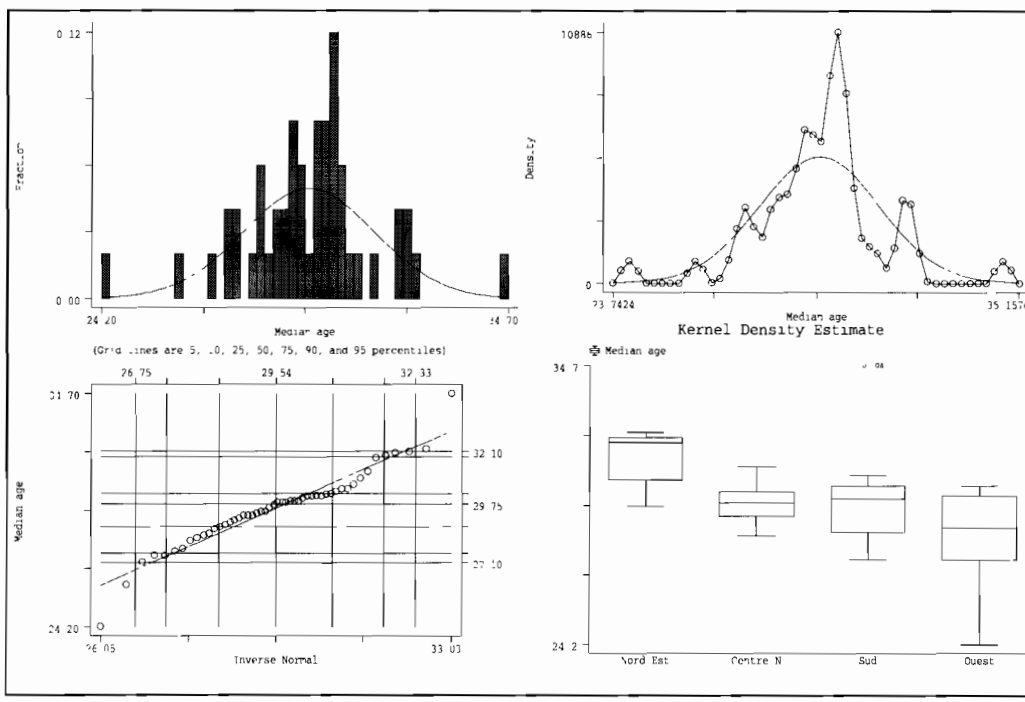
2.2 *Des analyses exploratoires graphiques et statistiques*

Stata permet d'obtenir rapidement une description des données, à l'aide de statistiques de distribution uni-, bi-, ou multivariée :

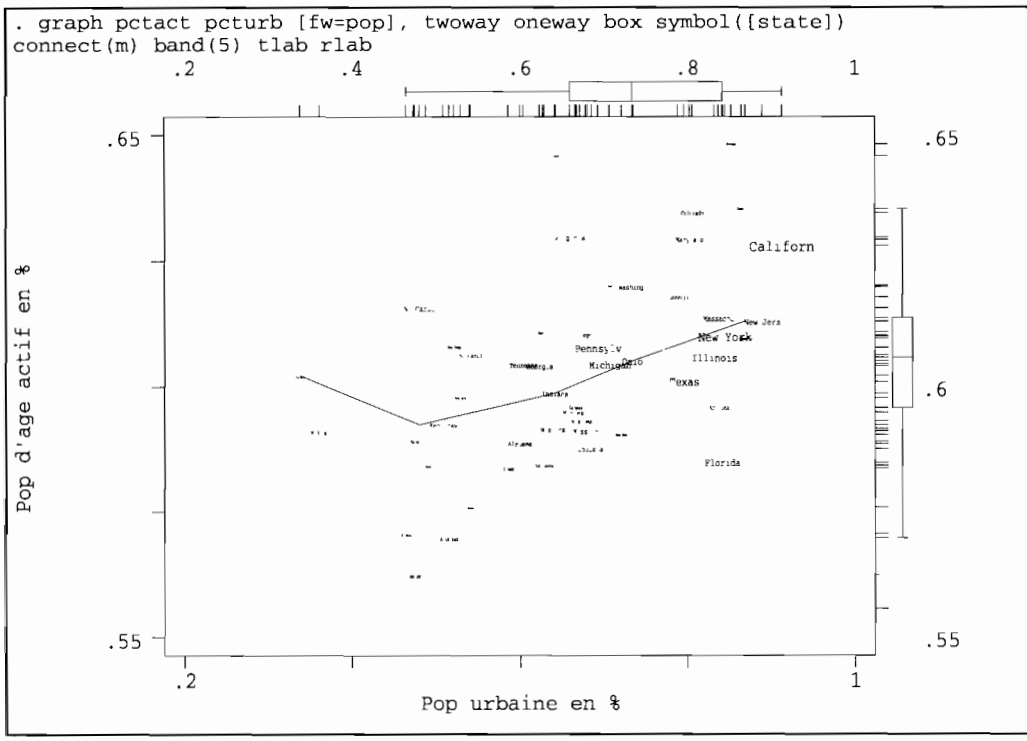
```
. summarize medage, detail
                        Median age
-------------------------------------------------------------
      Percentiles      Smallest
 1%         24.2            24.2
 5%         27.1            26.1
10%         27.5            27.1       Obs                  50
25%         28.7            27.4       Sum of Wgt.          50

50%        29.75                       Mean              29.54
                       Largest         Std. Dev.       1.693445
75%         30.2              32
90%        31.85            32.1       Variance        2.867755
95%         32.1            32.2       Skewness       -.1178646
99%         34.7            34.7       Kurtosis        5.013541

. tabulate actif region, all exact

Pop active | Census region
sup a 60%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+--------------------------------------------+----------
       Non |         1          9          8          5 |        23
       Oui |         8          3          8          8 |        27
-----------+--------------------------------------------+----------
     Total |         9         12         16         13 |        50

          Pearson chi2(3) =   8.8735   Pr = 0.031
 likelihood-ratio chi2(3) =   9.7154   Pr = 0.021
               Cramer's V =   0.4213
                    gamma =  -0.0581   ASE = 0.206
          Kendall's tau-b =  -0.0369   ASE = 0.131
           Fisher's exact =                 0.030
```

```
. table region act urb, contents(mean medage) row col center format(%5.2f) scol

----------+-----------------------------------------------------------------------
          |                Pop urbaine sup a 75% and Pop active sup a 60%
Census    | ------- Non -------      ------- Oui -------      ------ Total ------
region    |   Non     Oui    Total      Non     Oui    Total      Non     Oui    Total
----------+-----------------------------------------------------------------------
 Nord Est | 30.40   30.53   30.50             31.82   31.82    30.40   31.34   31.23
 Centre N | 29.52   29.35   29.49             29.90   29.90    29.52   29.53   29.52
      Sud | 29.23   29.35   29.28    34.70    29.25   31.07    29.91   29.33   29.62
    Ouest | 28.00   28.30   28.17    26.70    29.27   28.42    27.48   28.79   28.28

    Total | 29.24   29.31   29.27    29.37    30.38   30.18    29.25   29.79   29.54
----------+-----------------------------------------------------------------------
```

La commande graphique de Stata produit huit types de graphiques (histogramme, bâton, trait, boîte, point, matrice, étoile, camembert) qui peuvent être combinés, par exemple comme ceci :



En outre, on peut associer diverses courbes de lissage aux graphiques uni- et bivariés (densité de Kernel, méthode des intervalles, etc.). La combinaison des diverses techniques de représentations sur un même graphique permet de synthétiser les informations :



Par ailleurs, de nombreuses commandes graphiques répondent à des problèmes spécifiques (diagnostic de conformité à la loi normale, repérage des observations extrêmes, contrôle de qualité, courbes de survie, etc.).

Sous Windows, Macintosh et Unix, tous les graphiques peuvent être facilement copiés vers d'autres applications (traitement de texte, tableur, etc.).

2.3 *Des modèles de régression très diversifiés*

Les modèles proposés par Stata vont de la simple régression statistique jusqu'aux régressions les plus complexes pour traiter les données de panels. Stata est particulièrement adapté au traitement des données d'enquêtes, qu'elles soient de nature épidémiologique, économique, sociologique ou démographique. Des applications originales comme la régression sur les variables continues à intervalles ouverts ou sur les variables ordonnées, seront particulièrement appréciées en sciences sociales comme en biologie et en épidémiologie.

Les techniques d'estimation robuste sont disponibles pour pratiquement tous les modèles de régression. On appréciera entre autres les options sur la structure de corrélation des erreurs (par exemple, pour tenir compte des erreurs corrélées au sein des grappes) et la régression sur les quantiles comme alternative à la régression simple.

Les pondérations sont prises en compte de quatre manières différentes (analytique, par fréquence, selon un plan d'échantillonnage, par simple importance) selon le type de données utilisées. Mais surtout, les estimations des fréquences, moyennes, ratios, proportions ainsi que les coefficients de régression (simple, logit ou probit) peuvent être obtenus pour des données d'échantillonnage à plusieurs degrés, stratifié ou non.

Les régressions les plus courantes (simple, logit, probit, cox, etc.) sont accessibles avec des commandes simples, associées à de nombreuses commandes de diagnostic graphique. L'utilisateur averti peut aussi combiner, dans la mesure du possible, jusqu'à six familles de distribution des erreurs (gaussienne, inverse gaussienne, binomiale, binomiale négative, Poisson, gamma) avec huit types de fonction liant les variables dépendantes et indépendantes (identité, log-linéaire, logit, probit, log-log, ratio des puissances, puissances simples, binomiale négative). La plupart de ces distributions et fonctions valent pour les modèles de régression sur les données issues de panels.

Les techniques les plus avancées en matière d'analyse de survie permettent non seulement les troncatures à droite mais aussi à gauche, ainsi que des sorties temporaires d'observation. De plus, les variables indépendantes peuvent varier dans le temps, et les sujets, après avoir connu une première fois l'événement, peuvent entrer à nouveau dans la population soumise au risque et ainsi connaître plusieurs événements successifs jusqu'à la date d'enquête.

Enfin, les techniques d'analyse multivariée (analyse factorielle, canonique, en composantes principales) dans leur version anglo-saxonne sont aussi disponibles.

2.4 De la manipulation de fichiers et de la programmation avancée

Les fichiers de données Stata sont chargés en mémoire vive. Cela permet non seulement une remarquable rapidité d'exécution mais aussi l'adressage en ligne de chaque valeur d'une variable. La manipulation des données (vérification et correction) s'en trouve facilitée, notamment pour les données complexes, faisant intervenir plusieurs niveaux hiérarchiques (par exemple, un fichier de naissances pour chaque mère au sein des ménages classés par îlot d'habitation). Il n'est pas nécessaire de créer des fichiers intermédiaires pour créer des variables relatives à chaque niveau hiérarchique : de cette façon les informations collectées à un niveau supérieur sont directement disponibles aux niveaux inférieurs.

L'instrument le plus puissant de Stata est certainement son langage de programmation. Plus que de simples macros, Stata vous permet d'écrire des programmes spécifiques, accessibles de la même façon que n'importe quelle autre commande du logiciel, avec arguments et options.

Ce langage combine à la fois les opérations courantes de la programmation matricielle, l'accès à des procédures d'estimations par le maximum de vraisemblance et l'accès à des procédures déjà existantes dans Stata (comme le ferait une simple macro). En outre, dans les versions pour Windows, Macintosh et Unix, vous pouvez introduire des boîtes de dialogue dans les programmes.

Reprenons l'exemple donné dans le *User's Guide* de Stata, à la section *24.12.5 Development of a sample ado-command*, du chapitre consacré à la programmation *[U] 24 Programming Stata*. Dans ce programme qui calcule une statistique d'influence pour chaque observation dans une régression simple, nous avons indiqué en commentaire les aspects spécifiques de la programmation sous Stata :

```
program define hinflu
        version 5.0
        if "$S_E_cmd"~="fit" { error 301 }    /* Le programme ne peut s'exécuter
                                               qu'après la commande fit */
        tempvar Hi                             /* Des variables temporaires sont
                                               créées par souci d'économie
                                               de temps et d'espace en mémoire */

        local varlist "required new max(1)"    /* Le programme prévoit un seul
                                               argument qui correspond à la
                                               nouvelle variable à créer */

        parse "`*'"
        rename `varlist' `Hi'
        tempvar hii ei di2
        quietly {                              /* La suite du programme utilise
                                               des commandes existantes de Stata:
                                               fpredict, fit, generate...*/

                fpredict `hii', hat
                fpredict `ei', resid
                quietly fit
                                               /* Les résultats des commandes
                                               de Stata sont accessibles sous
                                               forme de variables systèmes:
                                               par exemple _result(#) */

                gen `di2' = (`ei'*`ei')/_result(4)
                replace `Hi' = ((_result(3)+1)/(1-`hii'))*(`di2'/(1-`di2')) + /*
                */ `hii'/(1-`hii')
        }
        rename `Hi' `varlist'
end                                            /* Les variables temporaires sont
                                               automatiquement éliminées après
                                               l'exécution du programme */
```

Pour exécuter ce programme, il suffira de taper le nom du programme suivi du nom de la nouvelle variable à calculer :

```
. hinflu nouvar
```

Ce mode de programmation a permis aux concepteurs de Stata de développer rapidement leur logiciel : chaque nouveau programme est d'abord diffusé rapidement *via* le bulletin bimensuel de liaison (*Stata Technical Bulletin - STB*) avant d'être éventuellement intégré dans une version supérieure du logiciel. De cette manière, les nouveaux programmes sont disponibles rapidement et sont testés par les utilisateurs eux-mêmes à une large échelle (*beta-test*).

2.5 *Pour en savoir plus*

En dépit du plaisir que vous aurez certainement à lire le présent manuel, nous devons admettre que les ouvrages de théorie statistique et les manuels de référence de Stata restent indispensables pour l'approfondissement des connaissances et pour un traitement avancé des données. Par ailleurs, le bulletin de liaison de Stata (*Stata Technical Bulletin - STB*) est un outil très efficace pour se mettre à la page des derniers développements en statistique. Il permet d'obtenir les programmes pour des nouvelles procédures à moindre frais.

Par ailleurs, Stata diffuse les dernières informations sur le logiciel sur le Web (**`http://www.stata.com`**). Les internautes peuvent également échanger leurs informations par l'intermédiaire du Statalist (*listserver* de Stata). Des cours payants de différents niveaux (NetCourses) sont aussi dispensés par Internet. Pour tout renseignement complémentaire et pour les inscriptions, adresser un courrier à **`stata@stata.com`**.

3 LA PRISE EN MAIN DU LOGICIEL

3.1 Les règles de base

Pour faciliter l'apprentissage du logiciel et de ses possibilités graphiques et statistiques, nous utiliserons principalement un fichier de données, « census.dta », tout au long de ce manuel. Ce fichier, fourni avec le logiciel Stata, contient des données du recensement des États-Unis de 1980.

3.1.1 Quelques conventions typographiques

Pour nous rapprocher des conventions utilisées dans les manuels de Stata, nous avons adopté les règles typographiques suivantes :

- Ce que Stata comprend **`ressemble a ca`** dans notre texte, c'est-à-dire à du caractère courrier gras. Notez l'absence d'accent et de caractères spécifiques au **`francais`**. Cela concerne aussi bien les commandes que vous taperez à l'écran que les résultats produits à l'écran. En somme, tout ce que Stata comprend est écrit en caractères **`courrier`** et avec un fort accent anglais !

- Lorsque après une commande, on veut indiquer que le lecteur doit taper un mot de son choix, par exemple, une liste de variables après la commande **`list`**, les *caractères italiques seront utilisés*, comme dans **`list`** *listevar*, **`by(`***vargroupe***`)`**.

- En plus des deux conventions précédentes, nous encadrerons ce que Stata montre à l'écran :

```
. list var1 var2

          var1     var2
1.           3       42
2.           5       23
3.           3       15
4.          10        4
5.           2       15
6.           0       10
```

3.1.2 Entrer et sortir de Stata

Avant d'entrer dans Stata, il est préférable de contrôler la place en mémoire vive qui sera allouée au fichier de données. En effet, par défaut, Stata n'allouera que 1 Mo de mémoire vive (RAM). C'est peu, et il serait dommage en plein travail de se trouver à court de mémoire et d'être obligé de sortir de Stata pour allouer de la mémoire supplémentaire.

Comme on l'a dit plus haut, la configuration optimale dépend des capacités de l'ordinateur et de la place occupée par les autres logiciels. Sachez par exemple que Windows 3.1 occupe au minimum 4 Mo de RAM. Pour Windows 95, 6 à 7 Mo de RAM doivent être réservé. Ainsi, lorsque l'ordinateur dispose de 16 Mo de RAM, on allouera au maximum 9 Mo de RAM pour Stata : au delà, Stata fonctionnera quand même, mais beaucoup plus lentement car il copiera une partie des fichiers sur le disque dur, ce qui accroît le temps de lecture et d'écriture.

Le tableau qui suit donne la formule approximative pour connaître le nombre maximum de Mo de RAM à allouer à Stata selon le système d'exploitation Windows (« Total » signifie le nombre total de Mo de RAM disponible sur l'ordinateur).

Windows 3.1	Windows 95
Total - 4	¾ Total - 3

Ainsi, avec 32 Mo de RAM sous Windows 95, un maximum de 21 Mo (= (32 * 3 / 4) - 3) devra être alloué à Stata.

Pour configurer la mémoire utilisée par Stata, on devra modifier les propriétés associées au programme exécutable de Stata. La

procédure diffère d'un système d'exploitation à l'autre (par exemple pour configurer à 9 Mo) :

- Pour DOS, il suffit de taper « **`C:>stata /k9000`** ».
- Pour Unix, il suffit de taper « **`% stata -k9000`** ».
- Pour Windows (3.1 et 95), les propriétés de l'icône de Stata doivent être modifiées. Dans Windows 95, on accède à la fenêtre des propriétés de Stata dans le sous-répertoire **`C:\Windows\Menu démarrer\Programmes\Stata`**.
  Pour obtenir la fenêtre des propriétés de l'icône, cliquer une fois sur l'icône, dérouler le menu **Fichier** et cliquer sur **Propriétés** (et **Raccourci** pour Windows 95). Ensuite, modifier la cible (appelée ligne de commande dans Windows 3.1) tel que « **`C:\Stata\wstata.exe /k9000`** » (au lieu de **`/k1000`** par défaut).
- Pour Macintosh, cliquer une fois sur l'icône correspondant à l'application de Stata. Dérouler le menu **Fichier** et cliquer sur **Informations**. Ensuite, sous la rubrique **Mémoire**, modifier la taille souhaitée à 9000 par exemple (au lieu de 1000 par défaut). Ne pas modifier la taille minimum.

Ensuite, le mode d'entrée dans Stata dépend du système d'exploitation. Le tableau qui suit indique quelles sont les commandes à exécuter pour entrer dans Stata. Cela suppose évidemment que Stata soit correctement installé sur l'ordinateur :

DOS	Windows 3.1	Windows 95	Macintosh	Unix
`c:>stata`	**Stata** *(icône)*	**Démarrer Programmes Stata**	**Stata.do** *(icône)*	**`% stata`**

3.1.3 Charger un fichier

Pour être chargé en mémoire vive par Stata, le fichier doit d'abord exister physiquement en mémoire morte, c'est-à-dire sur le disque dur de l'ordinateur. Stata travaille en mémoire vive sur

un fichier de travail qui est une copie du fichier original. Il appartient ensuite à l'utilisateur de sauvegarder correctement le fichier de travail.

Il n'y a qu'une manière de charger un fichier en mémoire vive. Ainsi, pour charger le fichier « census.dta » par exemple, il suffit de taper :

```
. use census
(1980 Census data by state)
```

Mais plutôt que le gentil message indiquant que vous venez de charger les données du recensement de 1980, vous obtiendrez quelque chose de plus méchant du genre :

```
. use census
file census.dta not found
r(601);
```

Sur un écran couleur, le message d'erreur indiquant que Stata n'a pas trouvé le fichier « census.dta » est en rouge, couleur qui suggère que Stata s'est fâché contre vous.

Ce n'est pas totalement injuste. Le fichier existe bel et bien mais pas dans le répertoire courant de travail. Lorsque vous avez installé Stata pour la première fois, il vous a été demandé de choisir un répertoire de travail, soit « **c:\data** » par défaut, soit un autre répertoire existant. Par la suite, à chaque fois que Stata sera exécuté, il se placera automatiquement dans ce répertoire de travail.

Or le fichier « census.dta » fait partie des fichiers copiés automatiquement lors de l'installation du logiciel, dans le répertoire « **c:\stata** » où figurent tous les fichiers (programmes ou données) fournis par Stata.

Pour lire « census.dta », il faut donc changer de répertoire. Pour cela, on peut procéder de deux façons :

- dans le cas où vous connaissez par cœur le chemin qui mène à votre fichier, taper simplement sous DOS ou sous Windows :

```
. cd\stata

. use census
(1980 Census data by state)
```

... ou bien sous Unix:

```
. cd /usr/local/stata

. use census
(1980 Census data by state)
```

- dans le cas où votre mémoire vous ferait défaut, utilisez les facilités du menu sous Windows, Macintosh ou Unix, et cliquez sur **File** puis **Open**, pour avoir accès au répertoire de fichiers de l'ordinateur et choisir ainsi le fichier tant désiré.

Sous Windows 95, le chemin d'accès doit le cas échéant tenir compte de l'abréviation des noms étendus de répertoires ou de fichiers. Ainsi pour rechercher le fichier **`C:\Mes données\mon fichier.dta`** *on devra taper :*

. cd\mesdon~1
. use monfic~1

Remarquez l'absence d'espace entre les mots, ainsi que le caractère ~ suivi d'un numéro (qui peut être différent s'il y a plusieurs fichiers qui commencent par 6 caractères identiques).

Comme on le voit, il faut être prudent en nommant les répertoires et les fichiers avec des noms étendus comme le permet Windows 95. Il est préférable d'utiliser de simples noms de 8 lettres (comme sous DOS ou Windows 3.1).

3.1.4 Sauvegarder un fichier

Les commandes pour sauvegarder un fichier sont parmi les premières que l'on apprend, et pourtant on se fait toujours avoir un jour ou l'autre en perdant tout ou partie de ses données. C'est ce qu'on appelle les joies de l'informatique.

La commande pour sauvegarder un fichier est pourtant simple : il suffit de taper **`save`** *`nomfichier`*. Mais regardons ce qui se passe :

```
. save census
file census.dta already exists
r(602);
```

Stata se fâche encore tout rouge pour vous dire que le fichier existe déjà. En effet, le fichier en mémoire vive n'est qu'une copie du fichier « census.dta » conservé en mémoire morte. En exécutant **`save`** vous devez indiquer si le fichier « census.dta »

doit être actualisé par la copie éventuellement modifiée en mémoire vive. Le fichier doit être sauvegardé soit sous un nouveau nom de fichier :

```
. save census2
file census2.dta saved
```

... soit sous le même nom de fichier avec l'option **replace**, ce qui aura pour effet de remplacer l'ancien fichier sur la mémoire morte de l'ordinateur :

```
. save census, replace
file census.dta saved
```

Remarquez dans les deux cas la teneur du message (en bleu) : Stata vous dit que le fichier est sauvegardé, mais il ne dit pas s'il a remplacé ou non un ancien fichier par une version plus récente. C'est à vous de savoir ce que vous faites en utilisant l'option **replace**, car il ne sera pas possible de récupérer l'ancien fichier, écrasé par le nouveau.

L'option **replace** peut tout aussi bien être utilisée sans spécifier le nom du fichier à remplacer :

```
. save, replace
file census.dta saved
```

Cette utilisation de l'option **replace** est très risquée : persuadé que vous travaillez sur le fichier « census2.dta », vous réalisez trop tard qu'il n'en est rien et que vous avez écrasé votre fichier original « census.dta » en ayant voulu économiser quelques secondes de votre temps précieux.

Pour sauvegarder un fichier, prenez l'habitude de toujours taper le nom du fichier :

. save *nom_fichier*, replace

Vous réduirez ainsi les risques d'erreurs.

3.1.5 Enregistrer les résultats d'une séance de travail

Sortir des résultats, c'est bien, les conserver, c'est encore mieux. Avant de commencer l'analyse de vos données, il est préférable de penser à conserver les résultats qui apparaissent à l'écran dans un fichier que vous pourrez consulter librement, en tapant la commande **log using** *nomfichier* :

```
. log using resultat
(suivent les commandes et les résultats)
```

Cette commande a pour effet d'ouvrir un fichier texte (format ASCII) avec extension « *nomfichier*.log » où seront conservées toutes les sorties à l'écran, c'est-à-dire non seulement les commandes que vous avez exécutées mais aussi les résultats produits par Stata. Ce fichier texte est récupérable dans n'importe quel éditeur de texte.

Pour fermer ce fichier texte, il suffit de taper :

```
. log close
```

Lors d'une séance de travail interactive, il est fort possible que les résultats obtenus ne vous satisfassent pas, ou plus souvent encore, que vous ayez fait des erreurs en tapant vos commandes. Dans ce cas, le fichier de résultats sera inutilement long. Il est donc préférable de vérifier d'abord l'utilité d'une commande ou d'un résultat avant de le faire figurer dans le fichier de résultats. Pour cela, on peut temporairement interrompre l'écriture sur le fichier « *.log » :

```
. log off

(suivent plusieurs commandes pour explorer les données et qui n'apparaîtront pas
dans le fichier des résultats)

. log on

(suivent les commandes utiles à conserver dans le fichier des résultats)
```

Les fichiers de résultats sont aussi très utiles pour ajouter des commentaires sur le traitement des données. Pour cela, placez une étoile avant le commentaire :

```
. * Ceci est un commentaire tres pertinent
```

Pour ajouter au fichier de résultats de la veille, les résultats de la session de travail du jour, on ajoute l'option **append** :

```
. log using resultat, append
```

Pour effacer un vieux fichier de résultat et rouvrir un nouveau fichier de résultat sous le même nom, on ajoute l'option **replace** :

```
. log using resultat, replace
```

Toutes ces commandes et options sont accessibles à partir du menu dans les versions de Stata pour Windows. Le Tableau

suivant indique la correspondance entre les termes utilisés dans les commandes interactives et les entrées du menu déroulant sous Windows :

options interactives	*menu déroulant* **Log**
`. log close`	Close log file
`. log off`	Suspend log file
`. log on`	Resume

L'avantage de Windows est de permettre d'amener à l'écran (dans une fenêtre) le fichier « *.log » en cours de session, même si vous ne l'avez pas encore fermé, temporairement ou définitivement. Pour cela, déroulez le menu **Log** et choisissez **Bring log window to top**. Vous pourrez ainsi voir les résultats que vous avez produits en début de session de travail, et qui n'apparaissent pas dans la fenêtre des résultats que vous venez de produire. Cela évite de répéter plusieurs commandes inutilement, et on peut plus facilement comparer les résultats entre eux, sans sortir de la session de travail.

Ajoutons enfin, que le fichier « *.log » en cours d'utilisation est automatiquement fermé en quittant Stata.

3.2 *La syntaxe des commandes de Stata*

Stata comme tous les logiciels, utilise un langage qui n'est ni de l'anglais, ni du français, mais son propre langage. Certes, les mots sont empruntés à la langue de Shakespeare, mais la syntaxe n'a rien à voir avec l'anglais. Hormis quelques exceptions, la syntaxe des commandes de Stata est :

```
[by listevar:] commande [listevar] [=exp]
  [if exp] [in intervalle] [pondération]
                  [, options]
```

Le nom de la commande est évidemment obligatoire, et il peut éventuellement être précédé d'un préfixe **by**, et le plus souvent il est suivi d'un ou de plusieurs suffixes. Dans le cas de commandes particulièrement usuelles, il peut parfois être abrégé, comme par exemple **d** pour **describe**. Les suffixes sont

entourés de crochets pour indiquer leur caractère optionnel : *listevar* correspond à une liste de variables, *exp* à une expression logique, *intervalle* à une série d'observations dans le fichier de données, et *pondération* à une expression indiquant la variable et le mode de pondération des données. Enfin, après une virgule, on peut ajouter une ou plusieurs *options* pour l'exécution de la commande.

La syntaxe complète pour chaque commande figure dans les manuels de référence de Stata, qui restent de ce point de vue irremplaçables. Mais puisque le préfixe **by** et les suffixes **if**, **in** et la pondération sont communs à la majorité des commandes, nous nous en tiendrons dans les chapitres suivants à exposer la syntaxe de base qui prend la forme :

commande [*listevar*] [=*exp*] [, *options*]

Dans le *User's Guide* de Stata, la syntaxe est expliquée à l'aide de la commande **summarize**. Plutôt que de reprendre le même exemple, nous préférons expliquer la syntaxe de la commande **list**, qui nous permettra par la même occasion de montrer comment on explore un fichier de données :

[**by** *listevar*:] **list** [*listevar*]
[**if** *exp*] [**in** *intervalle*]
[, [**no**]**display** **nolabel** **noobs**]

Cette commande n'utilise pas les suffixes [=*exp*] et [*pondération*], sur lesquels nous reviendrons dans les prochains chapitres à propos d'autres commandes.

3.2.1 Le suffixe [*listevar*]

Immédiatement après le nom de la commande, une liste de variables indique sur quelles variables doit s'effectuer la commande. Pour explorer le fichier « census.dta », on tapera :

```
. list state region pop

                state    region        pop
  1.          Alabama     South    3893888
  2.           Alaska      West     401851
  3.          Arizona      West    2718215
  4.         Arkansas     South    2286435
  5.       California      West   23667902
(suite de la liste)
```

La première colonne indique le numéro d'ordre de l'observation dans le fichier. Au sommet de chacune des colonnes suivantes est indiqué le nom de la variable.

Certaines commandes de traitement statistique nécessitent de spécifier au moins une variable. Dans ce cas, le manuel de Stata (et l'aide en ligne ou au menu) indique la liste de variables obligatoires sans crochets. Par exemple, **regress** *depvar [listevar1 (listevar2)]...* indique que le nom de la variable dépendante est nécessaire tandis que les noms des variables indépendantes entre crochets sont optionnels.

Dans les autres cas, la liste de variables par défaut est la totalité des variables du fichier. Ainsi, pour obtenir la liste de toutes les variables pour chaque observation, on tape :

```
. list

Observation 1

       state      Alabama       region        South          pop      3893888
      poplt5       296412     pop5_17       865836       pop18p      2731640
      pop65p       440015    popurban      2337713      medage        29.30
       death        35305    marriage        49018     divorce        26745

Observation 2

       state       Alaska       region         West          pop       401851
      poplt5        38949     pop5_17        91796       pop18p       271106
      pop65p        11547    popurban       258567      medage        26.10
       death         1604    marriage         5361     divorce         3517

Observation 3
(suite de la liste)
```

On remarque immédiatement que la présentation a changé : faute de place dans la fenêtre de résultats (qui fait 80 caractères de large), la commande **list** a choisi une présentation où la liste des variables (en vert sur l'écran couleur) et leur contenu (en jaune) sont indiqués en colonne pour chaque observation.

3.2.2 Le suffixe **[by** *listevar* **:]**

Ce préfixe avant la commande est une originalité de Stata qu'il est absolument nécessaire de connaître pour bien maîtriser les calculs complexes (notamment à l'aide de la fonction d'adressage) sur lesquels nous reviendrons plus loin.

Le préfixe **by** permet d'exécuter la commande pour chaque sous-ensemble d'observations défini pour chaque valeur de *listevar*. Avant d'exécuter la commande, le fichier doit d'abord être trié (avec la commande **sort** *listevar*) selon la même variable utilisée par le préfixe **by**. Par exemple, on aura :

```
. sort region

. by region: list region state pop medage

-> region=      NE
      region           state         pop     medage
  1.      NE   Massachusetts     5737037      31.20
  2.      NE    Rhode Island      947154      31.80
  3.      NE           Maine     1124660      30.40
  4.      NE        New York    17558072      31.90
  5.      NE         Vermont      511456      29.40
  6.      NE      New Jersey     7364823      32.20
  7.      NE    Pennsylvania    11863895      32.10
  8.      NE   New Hampshire      920610      30.10
  9.      NE     Connecticut     3107576      32.00

-> region= N Cntrl
      region           state         pop     medage
 10.  N Cntrl          Kansas     2363679      30.10
 11.  N Cntrl       S. Dakota      690768      28.90
 12.  N Cntrl       N. Dakota      652717      28.30
 13.  N Cntrl            Iowa     2913808      30.00
 14.  N Cntrl            Ohio    10797630      29.90
 15.  N Cntrl        Michigan     9262078      28.80
 16.  N Cntrl        Illinois    11426518      29.90
 17.  N Cntrl        Nebraska     1569825      29.70
 18.  N Cntrl       Wisconsin     4705767      29.40
 19.  N Cntrl        Missouri     4916686      30.90
 20.  N Cntrl         Indiana     5490224      29.20
 21.  N Cntrl       Minnesota     4075970      29.20
(suite de la liste)
```

Dans ce cas, il n'est pas utile d'inclure dans la liste la variable **region**, puisque chaque groupe de région est indiqué par son libellé « **-> region=** *libellé de la région* » : la variable **region** est identique dans chaque groupe d'États.

3.2.3 Le suffixe **[if** *exp***]**

Le suffixe **if** restreint l'exécution de la commande au sous-ensemble des observations pour lesquelles l'expression logique *exp* est vraie, c'est-à-dire différente de la valeur 0.

Nous reviendrons dans la section consacrée aux calculs sur la manipulation de ces expressions logiques, dites encore *booléennes*. Pour l'heure, un exemple suffit à comprendre le fonctionnement de ce suffixe :

```
. list region state pop medage if region==1

      region          state         pop     medage
  1.      NE  Massachusetts     5737037      31.20
  2.      NE   Rhode Island      947154      31.80
  3.      NE          Maine     1124660      30.40
  4.      NE       New York    17558072      31.90
  5.      NE        Vermont      511456      29.40
  6.      NE     New Jersey     7364823      32.20
  7.      NE   Pennsylvania    11863895      32.10
  8.      NE  New Hampshire      920610      30.10
  9.      NE    Connecticut     3107576      32.00
```

Le résultat est le même si l'on tape...

```
. list region state pop medage if region=="NE"
```

... c'est-à-dire si l'on mentionne le libellé exact de la région codée 1 pour « NE » (*North-East*).

3.2.4 Le suffixe `[in` *intervalle*`]`

Le suffixe **in** est moins courant dans la pratique, car il suppose de bien connaître l'ordre dans lequel sont classées les observations du fichier. Il permet d'exécuter la commande pour certaines observations, par exemple :

```
. list region state pop medage in 10/12

  10.  N Cntrl          Kansas     2363679      30.10
  11.  N Cntrl       S. Dakota      690768      28.90
  12.  N Cntrl       N. Dakota      652717      28.30
```

On peut aussi utiliser un code pour la première observation (« f » comme *first*) et la dernière (« l » comme *last*), ou bien des chiffres négatifs pour mentionner les observations depuis la dernière. Par exemple, pour faire la liste depuis l'avant-dernière observation jusqu'à la cinquième observation avant la fin, on exécutera la commande :

```
. list region state pop medage in -5/-2

      region          state         pop     medage
 46.    West         Alaska      401851      26.10
 47.    West        Wyoming      469557      27.10
 48.    West     New Mexico     1302894      27.40
 49.    West       Colorado     2889964      28.60
```

Ces subtilités sont en fait rarement utilisées car le résultat dépend totalement de l'ordre des observations dans le fichier. Ainsi, après avoir trié le fichier par ordre alphabétique des noms d'Etats, on obtiendra un résultat fort différent du précédent (remarquez les numéros d'ordre) :

```
. sort state

. list region state pop medage in -5/-2

      region          state        pop     medage
 46.    South       Virginia    5346818      29.80
 47.    South    W. Virginia    1949644      30.40
 48.     West     Washington    4132156      29.80
 49.  N Cntrl      Wisconsin    4705767      29.40
```

On préférera toujours sélectionner un sous-ensemble d'observation avec le suffixe **`if`** en fonction de variables bien connues et qui font sens, plutôt que de se fier à un ordre arbitraire des observations dans le fichier.

3.2.5 Les options

Ces suffixes, appelés plus communément *`options`*, modifient l'exécution de la commande mais pas le nombre d'observations sur lesquelles portent la commande. Les options sont spécifiques à chaque commande : il faut donc lire attentivement la syntaxe de chaque commande pour comprendre quel est l'objet de chaque option.

Ajoutons enfin que la commande **`list`** est la seule commande à avoir son équivalent sous Windows ou Macintosh : en effet le bouton **Browse** permet d'accéder à l'éditeur de Stata en mode lecture seule. Sans risquer de modifier les données originales, vous pouvez ainsi consulter les données comme vous le feriez dans un tableur.

Comparer les résultats des commandes suivantes :


```
. by region : list region state pop medage if region!=1

. list state pop poplt5 popurban medage death

. list state pop poplt5 popurban medage death, noobs

. list region state pop poplt5 popurban medage death

. list region state pop poplt5 popurban medage death, nodisplay

. list region state pop

. list region state pop, nolabel

. browse in 1/20

. browse if region==3, nolabel
```

(Placez-vous sur une colonne et taper sur la touche **Hide***)*

3.2.6 Les règles d'abréviation et de raccourcis

La plupart des noms de commandes, de variables, d'options, etc., peuvent être abrégés. En ce qui concerne les commandes et les options, le minimum de lettres requis pour l'abréviation est indiqué par un soulignement, comme vous l'avez peut-être remarqué à propos de la commande **`list`** expliquée ci-dessus. Par exemple, on pourra abréger :

```
. by region : list state pop, nolabel
```

... par :

```
. by region : l state pop, nol
```

... ou mieux encore :

```
. by reg : l st pop, nol
```

Stata ne prend aucun risque et ne confondra jamais une variable abrégée avec une autre. On remarque en particulier que **`pop`** n'est pas une abréviation pour **`popurban`**, pour **`pop65p`** ou pour toute autre variable commençant par les caractères *pop*, mais désigne bien la variable **`pop`** (population totale de l'État). L'abréviation **`popu`** ne pourra désigner que **`popurban`**. Il n'en est pas de même pour l'abréviation **`po`** qui pourrait désigner toutes les variables de population :

```
. by reg : l po

-> region=      NE  po ambiguous abbreviation
r(111);
```

Si votre intention était de faire la liste de toutes les variables qui commencent par les mêmes caractères, utilisez le caractère **`*`** :

```
. by reg : l po*

-> region=      NE
           pop      pop1t5     pop5_17      pop18p      pop65p    popurban
  1.   5737037      337215     1153174     4246648      726531     4808339
  2.    947154       56692      186159      704303      126922      824004
  3.   1124660       78514      242873      803273      140918      534072
  4.  17558072     1135925     3551938    12870209     2160767    14858068
  5.    511456       35998      109320      366138       58166      172735
  6.   7364823      463289     1527572     5373962      859771     6557377
  7.  11863895      747458     2375838     8740599     1530933     8220851
  8.    920610       62512      195570      662528      102967      480325
  9.   3107576      185188      637731     2284657      364864     2449774

-> region= N Cntrl
           pop      pop1t5     pop5_17      pop18p      pop65p    popurban
 10.   2363679      180877      468158     1714644      306263     1575899
 11.    690768       58446      147160      485162       91019      320777
(suite de la liste)
```

Remarquez que la liste des variables **pop*** apparaît dans l'ordre de classement dans le fichier de données. La commande précédente est absolument équivalente à :

```
. by reg : l pop-popu
```

Le tiret indique une suite de variables dans l'ordre du fichier.

On peut ainsi faire varier à l'infini les possibilités d'abréviations et de raccourcis au cours du traitement des données. En revanche, on risque de s'y perdre à la relecture des résultats : utilisez les abréviations, mais modérément et de manière compréhensible.

Comparer les résultats des commandes suivantes :


```
. l region state pop MedAge
. l reg state pop medage region
. l state popurban medage pop-pop5, noob
. l state pop* noob
. l d* medage-divorce
. l d* divorce-medage
. l pop* *age
. l pop* m* reg*
. l pop "Median age"
. l state popurbainedansetat
```

---

4 LES FICHIERS DE DONNÉES

Dans un fichier Stata (comme dans la plupart des autres logiciels statistiques), les lignes représentent les unités d'observation, et les colonnes représentent les caractéristiques de ces unités, appelées encore variables. Chaque variable est repérée par un nom, tandis que chaque observation *n* est repérée par un numéro d'ordre allant de 1 à *N*, le nombre total d'observations dans le fichier.

Les fichiers Stata sont toujours rectangulaires, c'est-à-dire que chaque ligne a le même nombre de colonnes, et chaque colonne le même nombre de lignes. Cela signifie en particulier que l'on ne peut avoir une case vide dans le fichier lorsqu'une variable n'est appliquée à aucune unité d'observation, ou lorsqu'une unité d'observation ne dispose d'aucune variable. Chaque fois que c'est le cas, Stata attribuera à cette case un code spécial correspondant à une valeur manquante, qui apparaît sous forme d'un point « . ».

Un fichier de données comprend donc une série de chiffres ou de caractères pour chaque individu et chaque variable. Mais pour faciliter sa lecture et sa manipulation, le fichier comprend aussi des libellés, des formats, des notes et d'autres caractéristiques, qui seront attribuées aux individus, aux variables ou à l'ensemble des données.

4.1 *Comment créer ou transférer un fichier de données sous Stata ?*

Stata n'est pas un logiciel de saisie, mais de traitement statistique des données. On peut bien sûr y introduire des données manuellement, mais il n'y a pas comme dans d'autres logiciels, de modules spécifiques permettant de mettre au point des feuilles de saisie originales. En revanche, il offre une grande flexibilité de lecture de données déjà saisies.

Trois cas de figure peuvent se présenter à l'utilisateur : le fichier n'est pas encore saisi sous forme de fichier, le fichier est saisi mais dans un format binaire, ou bien le fichier est saisi sous format ASCII.

4.1.1 Le fichier est à saisir

Les différentes possibilités offertes dans le cas d'un grand ou d'un petit fichier sont représentées dans la Figure 1.

Lorsque les données figurent dans *des questionnaires issus d'une enquête ou des dossiers* (telles qu'on en trouve dans les administrations et les entreprises) et que les informations concernent beaucoup de répondants et un nombre appréciable de variables, il est préférable de faire appel à un logiciel de saisie. La plupart des tableurs ont maintenant des modules de programmation qui permettent de mettre au point des feuilles de saisie. Il existe par ailleurs des logiciels statistiques qui offrent aussi la possibilité de mettre au point des masques de saisie.

Nous ne nous étendrons pas sur ces logiciels, ce qui dépasserait les objectifs de ce manuel. Simplement, il faut être conscient que pour une matrice de données de plus de 200 cases (c'est-à-dire 20 observations croisées avec 10 variables), la saisie devient un exercice fastidieux : il est préférable de s'entourer de toutes les précautions pour éviter les erreurs, surtout dans le cas de questionnaires qui présentent de nombreuses questions-filtres et autres renvois. Un programme de saisie permet en outre d'éviter les erreurs de saisie en spécifiant notamment les limites de valeurs pour chaque variable, les codes pour les données manquantes, etc. Une fois les données saisies à l'aide d'un

programme spécifique, reportez-vous plus bas à la section concernant les fichiers saisis en format binaire.

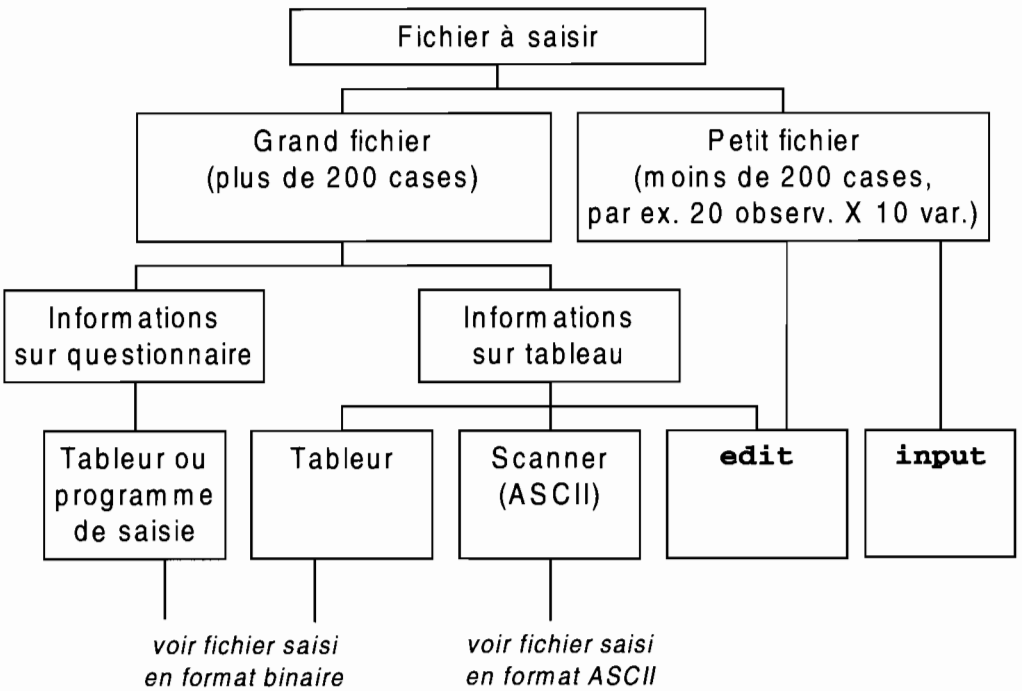


Figure 1 : Possibilités de saisie selon le type de données

Lorsque les données figurent dans un *tableau synthétique* (par exemple : une feuille de routine, un tableau publié dans un ouvrage, etc.), la mise au point d'un programme de saisie est la plupart du temps inutile. Il suffit alors de reproduire dans un tableur la forme du tableau de données, et de saisir ces données en ligne ou en colonne. Dans ce cas, après la saisie, reportez-vous plus bas à la section concernant les fichiers saisis en format binaire.

Une autre possibilité vous est offerte dans le cas où le tableau publié est suffisamment bien imprimé : le scanner, qui restitue (sous réserve de bonne qualité des caractères imprimés du tableau original) les données sous format ASCII. Une fois le tableau passé au scanner, reportez-vous au paragraphe sur les fichiers saisis en format ASCII.

Lorsque le fichier à saisir est petit (moins de 200 cases, comme on l'a expliqué plus haut), on peut alors utiliser deux procédures de Stata, **edit** et **input**, que nous décrivons plus loin.

4.1.2 Le fichier est saisi en format binaire

Lorsque le fichier est déjà saisi dans un format lisible seulement par un tableur (feuille de calcul) ou un autre logiciel statistique, on dit alors qu'il est saisi en format binaire.

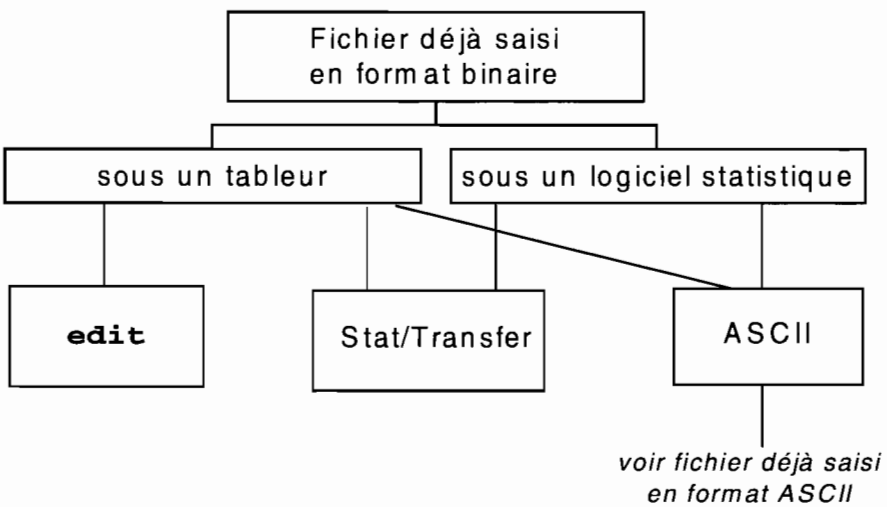


Figure 2 : Possibilités de conversion de fichier binaire

Les différentes possibilités offertes dans le cas d'un fichier saisi en format binaire sont représentées dans la Figure 2. Dans ce cas, le plus aisé est de faire appel au logiciel Stat/Transfer, qui permet de convertir les fichiers d'à peu près n'importe quel logiciel à n'importe quel autre[1], y compris Stata. Ce logiciel existe pour DOS, Windows et MacIntosh.

[1] Alpha Four, Clipper, Crunch, dBASE, Excel, FoxBASE, Gauss, Lotus 1-2-3, Paradox, Quattro Pro, SAS (fichiers *.tpt), S-Plus, SPSS (fichier *.xpt), Symphony, SYSTAT.

Si vous ne disposez pas de Stat/Transfer, la situation n'est pas aussi désespérée :

- utilisez la possibilité qu'offre certainement le tableur ou le logiciel statistique de produire un fichier en format ASCII ; reportez-vous ensuite à la section suivante sur les fichiers saisis en format ASCII ;
- dans le cas où les données sont saisies par un tableur, utilisez la commande **`edit`** de Stata en vous aidant des fonctions « couper-coller » des environnements Windows où MacIntosh, comme il est expliqué plus loin.

Dans le cas où vous ne disposeriez pas de toutes ces possibilités, votre situation est effectivement désespérée : le mieux est de courir acheter Stat/Transfer.

4.1.3 Le fichier est saisi en format ASCII

Lorsque le fichier est lisible par n'importe quel éditeur de texte, cela veut dire qu'il est en format ASCII. La Figure 3 représente la procédure à suivre lorsque le fichier est en format ASCII.

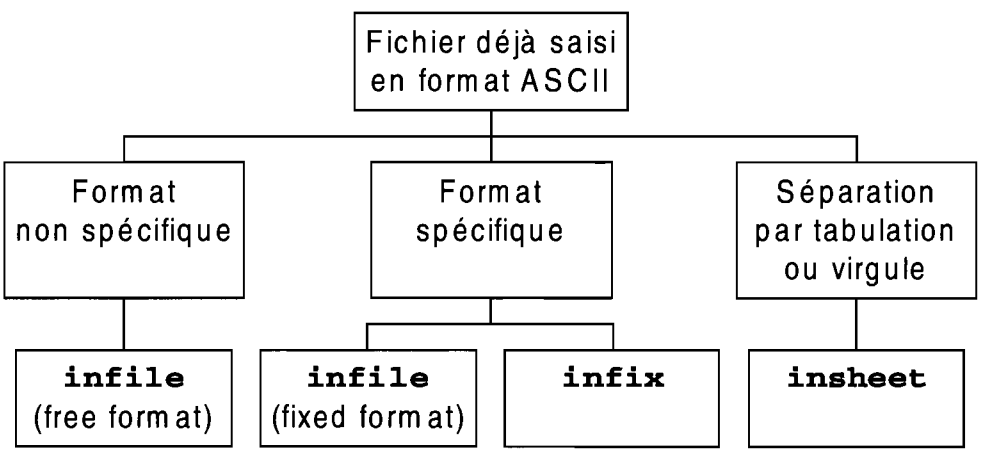


Figure 3 : Possibilités de lecture de fichier ASCII

Tout d'abord, rappelons que la plupart des logiciels statistiques et tableurs permettent de produire des fichiers en format ASCII. Quel que soit le mode de saisie que vous avez choisi (simple éditeur de texte, tableur, logiciel de saisie ou logiciel statistique), il vous faut savoir quel format exact de lecture des variables et des observations est utilisée dans votre fichier. Trois cas peuvent se présenter :

1. Le format est non spécifique : les valeurs de chaque variable sont séparées par des espaces (*blanks*) et chaque ligne d'observation peut avoir un nombre de caractères différent. Dans le cas où certaines variables seraient représentées par des caractères (et non des chiffres), les caractères pour une même variable doivent être entourés de guillemets, et ne doivent pas contenir des espaces (comme par exemple : Paul Durant, qui sera interprété comme représentant deux variables Paul et Durant), à moins que ces caractères soient entourés de guillemets ("Paul Durant"). Dans ces cas de formats non spécifiques, on utilisera la commande **`infile`** dans sa version pour format libre (*free format*).

2. Le format est spécifique : chaque variable occupe une place spécifique (une ou plusieurs colonnes) sur une ligne d'observation, et chaque observation est représentée par un nombre de lignes fixe (une ou plusieurs lignes). C'est généralement le format qui prend le moins de place dans un fichier ASCII, mais c'est aussi le plus complexe dans la mesure où il faudra utiliser un dictionnaire de variables écrit selon des conventions propres au langage de programmation des informaticiens. Dans ce cas, on utilisera la commande **`infile`** dans sa version pour format fixe (*fixed format*) ou bien la commande **`infix`**, plus simple d'utilisation.

3. Chaque ligne représente une observation et chaque variable est séparée par une marque de tabulation ou une virgule : c'est généralement sous cette forme que les tableurs permettent de sauvegarder les fichiers en format ASCII. Ce format est très simple à lire avec la commande **`insheet`**.

4.2 *Les commandes pour saisir et lire un fichier de données*

Si les données sont déjà saisies sous un autre logiciel et que vous avez judicieusement acheté Stat/Transfer pour convertir ces données dans Stata, cette section ne vous concerne pas. Sinon, lisez attentivement cette section en vous aidant d'abord des indications de la section précédente sur la marche à suivre pour créer ou transférer un fichier de données.

Les commandes décrites ici vont de la plus simple (**input**) à la plus complexe (**infile**). La commande **input**, le tableur intégré **edit** et la commande **insheet** sont toutes trois simples, alors que la commande **infile**, surtout dans sa version pour format fixe, est assez complexe. Si vous n'avez pas encore de notions de programmation informatique, **infile** constitue certainement une bonne initiation. Si vous êtes allergique à ceci : %10.2f, str20, 2.5E+02, %7e ou bien %3.0g, mais que vous devez quand même lire un fichier en format fixe, alors la commande **infix** est une version un peu simplifiée de **infile**.

4.2.1 Saisir des petits fichiers : la commande `input`

La commande **input** a une syntaxe très simple :

input [*listevar*]

Dans le cas où toutes les variables sont codées et qu'aucun libellé n'est nécessaire, il suffit de séparer la valeur de chaque variable par un espace, et de passer à la ligne pour chaque observation :

```
. input var1 var2

        var1       var2
  1. 23 3.4
  2. 21 0.6
  3. 32                                0.23900
  4.                    42 2.1
  5. end
```

On remarque immédiatement trois choses : tout d'abord, le numéro d'observation apparaît automatiquement à gauche de l'écran, ensuite le nombre d'espaces entre chaque valeur est indifférent, et enfin les caractères « *end* » sont réservés par Stata pour indiquer la fin de la saisie. Voyons le résultat :

```
. list

        var1        var2
  1.      23         3.4
  2.      21          .6
  3.      32        .239
  4.      42         2.1
```

Pour saisir une ou plusieurs observations supplémentaires, il suffit de taper immédiatement :

```
. input

        var1        var2
  5. 33 5.1
  6. 20 0.23
  7. end
```

Les nouvelles observations (5 et 6) sont générées à la suite des observations déjà saisies. Pour saisir une valeur manquante, on tapera un « . » au lieu de la valeur de la variable :

```
. input

        var1        var2
  7. 43 .
  8. . 2.21
  9. end
```

Pour saisir une nouvelle variable, on tapera :

```
. input var3

        var3
  1. 345
  2. 469
  3. 322
  4. .
  5. 145
  6. 99
  7. 432
  8. .
  9. end
```

Dans le cas où les valeurs des variables sont libellées, la syntaxe de la commande est plus complexe :

input [*type*] *listevar* [:*nomlib*]
[[*type*] *listevar* [:*nomlib*] ...]
[, **automatic label**]

Par exemple, pour saisir les variables **state**, **region** et **pop** du fichier « census.dta », on tapera :

```
. input str14 state int region:cenreg long pop, automatic

             state    region         pop
  1. Alabama South 3893888
  2. Alaska West 401851
  3. Arizona West 2718215
  4. Arkansas South 2286435
  5. California West 23667902
  6. Colorado West 2889964
  7. Connecticut NE 3107576
  8. Delaware South 594338
  9. Florida South 9746324
 10. Georgia South 5463105
(suite de la liste)
 39. "Rhode Island" NE 947154
 40. "S. Carolina" South 3121820
 41. "S. Dakota" "N Cntrl" 690768
 42. Tennessee South 4591120
 43. Texas South 14229191
 44. Utah West 1461037
 45. Vermont NE 511456
 46. Virginia South 5346818
 47. Washington West 4132156
 48. "W. Virginia" South 1949644
 49. Wisconsin "N Cntrl" 4705767
 50. Wyoming West 469557
```

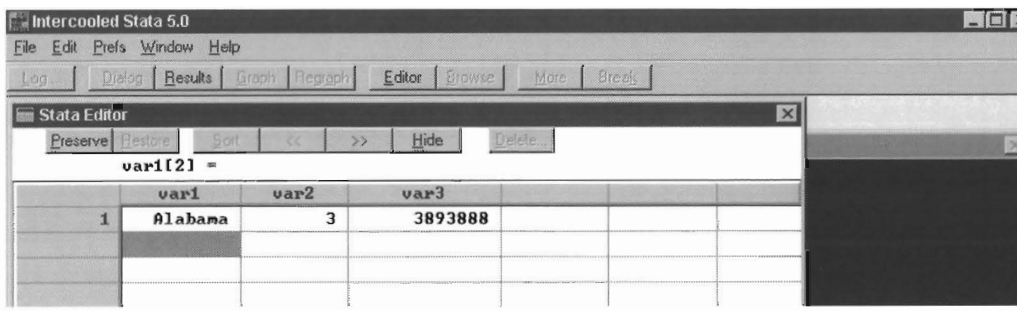
Vous n'êtes pas sans avoir remarqué que certains États (S. Carolina, W. Virginia...) ou régions (N Cntrl) ont leur libellé entouré de guillemets : lorsqu'un libellé contient des espaces, il est en effet nécessaire de spécifier quels sont les caractères qui doivent être inclus dans le libellé.

Cette forme de saisie demande beaucoup d'attention. Puisqu'il s'agit de définir en même temps que la saisie, les libellés et les types de variables, il est préférable d'utiliser la commande **`infile`** : dans ce cas, les données doivent être saisies indépendamment à l'aide d'un éditeur. Il est donc déconseillé d'utiliser la forme complexe de la commande **`input`** sans avoir d'abord compris **`infile`** ainsi que les formats numériques et alphanumériques de variables, et les libellés de valeur et de variable (voir la section *Comment rendre son fichier lisible ?*).

4.2.2 Saisir des petits fichiers : le tableur de Stata `edit`

Tout ce que vous venez de lire sur la commande **`input`**, vous allez vite l'oublier si vous travaillez dans un environnement Windows ou MacIntosh. Le tableur intégré de Stata est en effet beaucoup plus facile à utiliser, même s'il ne présente pas toutes les fonctions d'un tableur spécialisé.

Pour activer l'éditeur de Stata, cliquez sur le bouton **Editor**. Vous verrez apparaître une nouvelle fenêtre Stata Editor. Par exemple pour saisir le fichier « census.dta » :



Remarquez qu'on a tapé pour la variable région le code 3 et non « South ». L'éditeur de Stata ne peut automatiquement créer des libellés pour chaque valeur d'une variable (par exemple, « South » pour la valeur 3, « West » pour la valeur 4, etc.). Le libellé correspondant à ces valeurs devra être défini ultérieurement à l'aide de la commande **`label`**.

Cependant, Stata vous offre la possibilité d'introduire les libellés eux-mêmes : dans ce cas, l'éditeur considérera la variable région comme alphanumérique, et il vous faudra utiliser la commande **`encode`** pour la transformer en variable numérique avec des libellés correspondant à chaque valeur.

Avec l'éditeur, on peut déjà changer les noms des variables elles-mêmes qui ont été automatiquement nommées **`var1`**, **`var2`** et **`var3`**. Il suffit pour cela de cliquez deux fois (« double-clic ») sur la colonne correspondant à la variable. Une nouvelle fenêtre apparaît qui vous permet de changer le nom (« Name »), le libellé (« Label ») et même le format d'affichage de la variable (« Format »). On peut ainsi changer les noms **`var1`** par **`state`**, **`var2`** par **`region`** et **`var3`** par **`pop`**.

4.2.3 Couper dans la fenêtre d'un tableur ou d'un traitement de texte et coller sur le tableur intégré de Stata

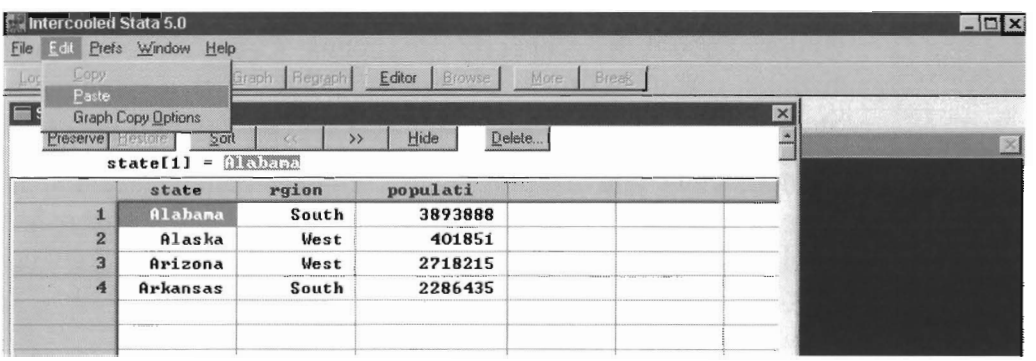
Le fameux couper-coller des environnements Windows et MacIntosh est certainement la plus agréable façon de transférer les données vers l'éditeur de Stata. L'inverse est également possible, ce qui double le plaisir.

Sélectionnez les données à copier dans le tableau d'une feuille de calcul ou simplement d'un traitement de texte. Par exemple, le tableau suivant contient les valeurs des trois premières variables pour les quatre premiers États du fichier « census.dta » :

State	Région	Population
Alabama	South	3893888
Alaska	West	401851
Arizona	West	2718215
Arkansas	South	2286435

Quelle que soit la présentation du tableau dans le tableur ou le traitement de texte, l'important est que dans la partie sélectionnée, chaque colonne définisse une variable (et une seule) et chaque ligne une observation (et une seule), à l'exception de la première qui définit le nom des variables.

Ensuite, copiez en mémoire la sélection de cellules, puis placez-vous dans la fenêtre de l'éditeur de Stata, et cliquez sur la première cellule en haut à gauche. Tirez sur le menu **Edit** de la fenêtre principale de Stata et cliquez sur **Paste** :



Notez que les noms des variables ont été transférés en minuscules, et que le « é » du mot « Région » n'apparaît plus, et que seuls les huit premiers caractères du mot « Population » ont servi à nommer la variable **`populati`**. Mais surtout, les valeurs pour la variable **`rgion`** sont stockées sous forme alphanumérique : il vous faudra utiliser la commande **`encode`** pour transformer la variable **`rgion`** en variable numérique avec des libellés correspondant à chaque valeur.

Le transfert dans le sens inverse, d'un fichier Stata vers un tableur ou un traitement de texte, se fait selon les mêmes principes. Simplement, vous devez spécifier si les noms des variables devront être copiées ou non avec leur contenu : pour cela, tirer sur le menu **Prefs**, sélectionner **General Preferences** et vérifiez dans **Editor Prefs** que l'option **Include variable names on copy to clipboard** est activée si vous voulez que les noms soient copiés. L'option **Copy value labels instead of numbers** est aussi utile lorsque vous voulez que les libellés des valeurs des variables soient copiées plutôt que les valeurs elles-mêmes (par exemple, le nom de la région « South » plutôt que 3).

4.2.4 Lire un fichier ASCII avec `insheet`

Les tableurs et les logiciels statistiques offrent souvent la possibilité de sauvegarder les fichiers en format ASCII, en séparant chaque variable par une virgule ou par une tabulation. Mais prenez garde aux virgules : si vous utilisez (selon la convention française) les virgules comme marque de décimale, il faudra d'abord les convertir en points « . ». La convention anglaise (décimale marquée par un point « . ») s'applique à Stata, comme à beaucoup d'autres logiciels statistiques.

Ensuite, vous pouvez utiliser les virgules comme séparateur entre variables, mais il est conseillé de plutôt utiliser les tabulations pour séparer chaque variable, car ces tabulations ne pourront jamais être confondues avec un caractère significatif.

Les valeurs des trois premières variables pour les quatre premiers États du fichier « census.dta » ont été inscrites dans un

fichier ASCII « census.txt » (le caractère → indique une marque de tabulation) :

```
State→Région→Population
Alabama→South→3893888
Alaska→West→401851
Arizona→West→2718215
Arkansas→South→2286435
```

Pour vérifier que ce fichier est bien en ASCII et que chaque variable est séparée par une tabulation, utilisez la commande **type** de Stata (où <T> indique une marque de tabulation) :

```
. type census.txt, showtabs
State<T>R.gion<T>Population
Alabama<T>South<T>3893888
Alaska<T>West<T>401851
Arizona<T>West<T>2718215
Arkansas<T>South<T>2286435
```

Ensuite, utilisez la commande **insheet** :

```
. insheet using census.txt
(3 vars, 4 obs)

. list

        state      rgion      populati
  1.   Alabama      South       3893888
  2.    Alaska       West        401851
  3.   Arizona       West       2718215
  4.  Arkansas      South       2286435

. describe

Contains data
  obs:             4
 vars:             3
 size:            84 (100.0% of memory free)
-------------------------------------------------------------------------------
   1. state       str8   %9s                 State
   2. rgion       str5   %9s                 R,gion
   3. populati    long   %12.0g              Population
-------------------------------------------------------------------------------
Sorted by:
     Note:  data has changed since last save
```

Notez que les noms des variables ont été transférés en minuscules, et que le « é » du mot « Région » n'apparaît plus, et que seuls les huit premiers caractères du mot « Population » ont servi à nommer la variable **populati**. Mais surtout, les valeurs pour la variable **rgion** sont stockées sous forme alphanumérique : il faudra utiliser la commande **encode** pour transformer la variable **rgion** en variable numérique avec des libellés correspondant à chaque valeur.

4.2.5 Lire un fichier ASCII avec `infile` en format libre

La commande **`infile`** lit des fichiers ASCII où les variables sont séparées par des virgules ou par des espaces. À la différence de la commande **`insheet`** elle ne peut identifier les caractères de tabulation comme séparateurs entre variables. Par contre, de la même façon que la commande **`input`**, elle peut tenir compte du type et du libellé des valeurs des variables. La différence essentielle entre les deux commandes est que **`input`** permet de saisir les données directement à l'écran, tandis que **`infile`** lit un fichier de données ASCII déjà existant. La commande **`infile`** est évidemment mieux adaptée à la lecture de gros fichiers.

La syntaxe de **`infile`** est la suivante :

`infile` [*type*] *listevar* [:*nomlib*] [_skip(#)]
[[*type*] *listevar* [:*nomlib*] [_skip(#)] ...]
`using` *nomfichier* [, **`automatic`**]

Par exemple, pour saisir les variables **`state`**, **`region`** et **`pop`** du fichier « census.txt » (où les espaces séparent les variables) :

```
. type census.txt, showtabs
state region pop
Alabama South 3893888
Alaska West 401851
Arizona West 2718215
Arkansas South 2286435
California West 23667902
 (suite de la liste)
"Rhode Island" NE 947154
"S. Carolina" South 3121820
"S. Dakota" "N Cntrl" 690768
Tennessee South 4591120
Texas South 14229191
Utah West 1461037
Vermont NE 511456
Virginia South 5346818
Washington West 4132156
"W. Virginia" South 1949644
Wisconsin "N Cntrl" 4705767
Wyoming West 469557
```

... on tapera :

```
. infile str14 state int region:cenreg long pop using census.txt, automatic
(50 observations read)
```

À chaque variable, on a attribué un type de stockage numérique (**`int`**, **`long`**...) ou alphanumérique (**`str14`**...). Pour plus de détails, voir la section *Comment rendre son fichier lisible ?*

Dans le fichier ASCII original, on a pris la précaution d'entourer de guillemets les libellés de certains États (`"S. Carolina"`, `"W. Virginia"`...) ou régions (`"N Cntrl"`) : lorsqu'un libellé contient des espaces, il est en effet nécessaire de spécifier quels sont les caractères qui doivent être inclus dans le libellé.

Ensuite, l'option **`automatic`** est nécessaire lorsque des libellés doivent être créés pour certaines variables. Par exemple, pour la variable **`region`**, on a besoin d'établir une correspondance entre les caractères contenus dans le fichier ASCII, et un code numérique. Cette correspondance est contenue dans un nom de libellé qu'on a appelé **`cenreg`** dans le dictionnaire. L'option **`automatic`** de la commande **`infile`** crée un code de 1 à *n* différent pour chaque chaîne de caractères différente rencontrée en lisant le fichier « census.txt ».

```
. label list
cenreg:
                1 South
                2 West
                3 NE
                4 N Cntrl
```

En outre, la commande **`infile`** ne tient compte que des espaces comme séparateurs entre variables. Elle n'est pas restreinte aux fichiers où chaque ligne représente une observation. **`infile`** permet de lire des fichiers complexes pour lesquels chaque observation figure sur plusieurs lignes ou au contraire une ligne peut contenir plusieurs observations : mais il est fortement déconseillé de travailler avec de tels fichiers, car les risques d'erreur sont grands et difficilement identifiables.

L'option **`_skip(#)`** permet de « sauter » une ou plusieurs variables dans le fichier ASCII. Par exemple, pour ne lire que les variables **`state`** et **`pop`**, on tapera :

```
. infile str14 state _skip(1) long pop using census.txt
(50 observations read)
```

Certaines variables peuvent être codées manquantes pour quelques individus. Comment devra-t-on s'y prendre pour coder une variable numérique manquante ? Et une variable alphanumérique manquante ? (Suggestion : utilisez les points « . » et les guillemets " ".)

4.2.6 Lire un fichier ASCII avec `infile` en format fixe

La commande **`infile`** permet de lire les fichiers ASCII les plus complexes, par l'intermédiaire d'un dictionnaire. Un dictionnaire est un fichier en format ASCII indiquant le format des données, c'est-à-dire indiquant quelles lignes doivent être lues et quelles positions sur chaque ligne correspondent à des variables.

La commande **`infile`** avec dictionnaire ne doit être utilisée qu'avec une bonne connaissance de la section 4.3 *Comment rendre son fichier lisible ?* À la différence de **`infile`** en format libre, les retours à la ligne sont considérés comme significatif : on devra spécifier dans le dictionnaire si chaque observation est représentée par plusieurs lignes. Lorsqu'au contraire chaque observation est représentée par une seule ligne et que chaque valeur est séparée par un espace, on utilisera la syntaxe de **`infile`** en format libre.

Le dictionnaire des variables a la syntaxe suivante :

`[infile] dictionary [using` *`nomfichierdonnées`*`]` {
** commentaires éventuels*
`_column(#)`
`_skip(#)`
[*`type`*] *`var`* [*`:nomlib`*] [*`%infmt`*] ["*`libellé variable`*"]
}
(les données peuvent suivre immédiatement le dictionnaire)

Le *`type`* de la variable est un format de stockage de la variable qui est expliqué dans la section *Les formats de stockage et d'affichage*. Le type de la variable ne doit pas être confondu avec le format de lecture (*`%infmt`*).

Le fichier de données peut apparaître directement à la suite de l'accolade } qui indique la fin du dictionnaire. Il est préférable cependant de conserver les données dans un fichier ASCII spécifié par l'option **`using`** *`nomfichier`* dans la mesure où vous pouvez avoir besoin de lire le même fichier de données avec des dictionnaires différents.

La commande **`infile`** avec dictionnaire comporte de nombreuses options pour lire une observation sur plusieurs

lignes. Nous ne les commenterons pas ici : si vous avez besoin de ces options, c'est que vous n'avez sans doute plus besoin de ce manuel. Simplement, nous montrerons en guise d'exemple, la procédure pour lire le « census.txt », écrit cette fois-ci en format fixe (notez l'absence de marque de tabulation) :

```
. type census.txt, showtabs
       Alabama  South  3893888
        Alaska   West   401851
       Arizona   West  2718215
      Arkansas  South  2286435
    California   West 23667902
(suite de la liste)
     S. DakotaN Cntrl   690768
     Tennessee  South  4591120
         Texas  South 14229191
          Utah   West  1461037
       Vermont     NE   511456
      Virginia  South  5346818
    Washington   West  4132156
   W. Virginia  South  1949644
     WisconsinN Cntrl  4705767
       Wyoming   West   469557
```

Dans le fichier de données ASCII, chaque ligne compte : vérifiez en particulier qu'aucune marque de paragraphe inutile ne figure en fin de fichier. Par exemple, si le fichier se termine ainsi (notez les marques de paragraphes ¶) :

```
(fin de la liste)
       Wyoming   West   469557¶
 ¶
```

... vous obtiendrez une observation de plus pour laquelle chaque variable sera codée manquante.

Le dictionnaire écrit en ASCII, de préférence avec une extension « *.dct » (comme dictionnaire), indiquant le format des données, aura la forme :

```
infile dictionary using census.txt {
               str14  state          %14s "State"
               byte  region:cenreg    %7s "Region"
_column(23)
               long     pop           %8f "Population"
}
```

Notez que chaque variable est située sur des positions précises, toujours les mêmes quelle que soit la ligne : le format est fixe. Dans le fichier de données, on remarque des lignes telles que :

```
     S. DakotaN Cntrl   690768
     WisconsinN Cntrl  4705767
```

Les 14 premiers caractères seront attribués à la variable **state**, (par exemple `Wisconsin`) tandis que les 7 suivants (`N Cntrl`)

sont attribués à la variable **region**. Dans un format libre, les deux chaînes de caractères devraient être séparées par un espace.

Ensuite, on exécute dans Stata la commande **infile**, pour lier le dictionnaire et les données :

```
. infile using census.dct, automatic

infile dictionary using census.txt {
                str14  state          %14s "State"
                byte   region:cenreg   %7s "Region"
_column(24)
                long      pop          %8f "Population"
}

(50 observations read)

. describe

Contains data
  obs:            50
 vars:             3
 size:         1,150 (100.0% of memory free)
-------------------------------------------------------------------------------
   1. state       str14  %14s                    State
   2. region      byte   %8.0g       cenreg      Region
   3. pop         long   %12.0g                  Population
-------------------------------------------------------------------------------
Sorted by:
     Note:  data has changed since last save
```

L'option **automatic** est nécessaire lorsque des libellés doivent être créés pour certaines variables. Par exemple, pour la variable **region**, on a besoin d'établir une correspondance entre les caractères contenus dans le fichier ASCII, et un code numérique. Cette correspondance est contenue dans un nom de libellé qu'on a appelé **cenreg** dans le dictionnaire. L'option **automatic** de la commande **infile** va créer un code de 1 à *n* différent pour chaque chaîne de caractères différentes rencontrées en lisant le fichier « census.txt ».

```
. label list
cenreg:
           1 South
           2 West
           3 NE
           4 N Cntrl
```

Par ailleurs, pour lire un tel fichier, on est libre de ne pas tenir compte de certaines positions. Ainsi, le chiffre de la population n'est lu qu'à partir de la colonne N°24, et occupe 8 positions. Si l'on voulait ne lire que les variables **state** et **pop**, on pourrait sauter les positions correspondant à la variable **region** en utilisant l'option **_column(#)** :

```
infile dictionary using census.txt {
                str14  state          %14s "State"
_column(23)
                long     pop           %8f "Population"
}
```

ou bien l'option **_skip(#)** :

```
infile dictionary using census.txt {
                str14  state          %14s "State"
_skip(8)
                long     pop           %8f "Population"
}
```

Dans le dictionnaire, on remarque que le format de lecture de la variable **region** *est alphanumérique (*`%7s`*) :*

```
byte  region:cenreg   %7s "Region"
```

Au contraire, dans le fichier Stata, le format de stockage est numérique (`%8.0g`*) :*

```
2. region      byte    %8.0g       cenreg     Region
```

Qu'est-ce qui explique cette différence ?

Par ailleurs, qu'est-ce qui distingue l'option **_skip(#)** *dans les deux versions de la commande* **infile**, *avec ou sans dictionnaire ?*

4.2.7 Lire un fichier ASCII avec `infix`

La commande **infile** est, admettons-le, destinée aux habitués de l'informatique. Pour le commun des mortels, Stata a créé une version simplifiée de **infile**, sans termes barbares tels que %10.2f, %20s, %2.5g ou bien %7.2e, mais qui peut néanmoins lire des données en format fixe, avec ou sans dictionnaire de variables.

La syntaxe de **infix** avec dictionnaire est la suivante :

infix using *nomdict* [, **using** *nomfichier*]

et la syntaxe du dictionnaire est :

infix dictionary [using *nomfichier***]** {
** commentaires éventuels*
[*type*] *listevar* #[-#]
}
(les données peuvent suivre immédiatement le dictionnaire)

Le `type` de la variable est un format de stockage de la variable qui est expliqué dans la section 4.3 *Comment rendre un fichier facilement lisible ?*

La commande **infix** avec dictionnaire comporte des options pour lire une observation sur plusieurs lignes. Nous ne les commenterons pas ici : si vous avez besoin de ces options, c'est que vous n'avez sans doute plus besoin de ce manuel.

Pour lire le fichier « census.dct », le dictionnaire aura la forme :

```
infix dictionary using census.txt {
              str state    1-14
              str region  15-21
              long pop    24-31
}
```

Comme on le voit, il suffit d'indiquer la position de la variable sur la ligne. Le dictionnaire **infix** est plus facile à utiliser si toutes les variables sont numériques dans le fichier de données.

En effet, les différences avec le dictionnaire **infile** se situent au niveau des libellés : **infix** ne crée pas de libellé de variables (par exemple, "Population" pour la variable **pop**), ni de libellé pour les valeurs des variables (**cenreg** pour les catégories de **region**). Cela a une conséquence importante : les valeurs pour la variable **region** ne peuvent être lues et stockées qu'en format alphanumérique (**str**) :

```
. describe

Contains data
  obs:            50
 vars:             3
 size:         1,450 (100.0% of memory free)
-------------------------------------------------------------------------------
   1. state        str14  %14s
   2. region       str7   %9s
   3. pop          long   %12.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  data has changed since last save
```

Il faudra utiliser la commande **encode** pour transformer la variable **region** en variable numérique avec des libellés correspondant à chaque valeur.

La commande **infix** peut aussi être utilisée sans dictionnaire. Dans ce cas, les spécifications de chaque variable sont inscrites directement sur la ligne de commande :

infix [*type*] *listevar* [#:] # [-#]
[[*type*] *listevar* [#:] # [-#] ...]
using *nomfichier*

Pour lire « census.txt », on aurait donc :

```
. infix str state 1-14 str region 15-21 long pop 24-31 using census.txt
```

Quels sont les résultats des commandes suivantes ?

```
. infix str state 1-14 byte region 15-21 using census.txt
. infix long pop 24-31 str region 15-21 using census.txt
. infix str state 1-16 str region 10-21 using census.txt
. infix str region 15-21 long pop 21-31 using census.txt
```

Testez ces commandes sur le fichier « census.txt » que vous aurez d'abord créé ou bien sur un fichier de votre choix contenant des variables alphanumériques.

4.2.8 « *Et si je n'ai pas assez de mémoire ?* »

Après avoir travaillé de longues heures à comprendre ce qui précède et à écrire un dictionnaire et/ou une commande compliquée pour lire votre fichier de données, vous tapez enfin sur la touche retour de chariot, non sans angoisse.

Vous entendez le disque dur travailler, et soudain, s'interrompre. Là, sur votre écran s'inscrit :

```
. (votre commande)
no room to add more observations
r(901);
```

« *Hein ?* ». Eh ! Oui, ça arrive : « *manque d'espace pour ajouter des observations* ».

Assurez-vous d'abord de la bonne configuration de la mémoire vive pour Stata (voir chapitre 3 sur *La prise en main du logiciel*). Si vous ne disposez réellement plus de mémoire, vérifiez que le type de chaque variable est bien défini d'une

manière optimale : si vous ne le définissez pas explicitement, le type sera **float** par défaut, ce qui prend quatre fois plus de place en mémoire qu'un type **byte** (voir la section suivante).

Si malgré tout la mémoire vous manque, considérez réduire le nombre de variables à lire dans le fichier de données (voir en particulier l'option **_skip**). Si toutes les variables sont vraiment indispensables, sachez que les commandes **infile** et **infix** prévoient la lecture d'une partie du fichier ASCII à l'aide des options **if** et **in** (voir *La syntaxe de base des commandes de Stata*). Par exemple, pour lire les 1000 premières lignes d'un fichier, on aura :

```
. infile (suivent les spécifications) in 1/1000, (options)
```

et pour les 1000 suivantes :

```
. infile (suivent les spécifications) in 1001/2000, (options)
```

Si l'on veut sélectionner les lignes pour une certaine valeur d'une variable (par exemple, les observations pour les hommes), on peut taper :

```
. infile (suivent les spécifications) if region==3, (options)
```

ou bien :

```
. infile (suivent les spécifications) if region=="South", (options)
```

Les mêmes procédures valent pour **infix**.

4.3 *Comment rendre un fichier facilement lisible ?*

La bonne compréhension des données statistiques commence par une bonne restitution visuelle de ces données. Pour savoir comment les chiffres sont restitués, et il faut d'abord connaître la différence entre type de stockage, format de lecture et format d'affichage des données numériques. Ensuite, ces données doivent être bien documentées, à l'aide de libellés et de notes. Enfin, nous terminons cette section par quelques conseils de toilettage du fichier, pour rendre son utilisation plus aisée, pour vous-mêmes et pour d'autres utilisateurs.

4.3.1 Les types de stockage

Les types de stockage numérique permettent d'optimiser la place d'un fichier en mémoire. A chaque variable on peut attribuer un espace en mémoire qui tient compte de valeurs extrêmes de la variable. Par exemple, il est inutile qu'une variable codée de 1 à 20 soit stockée dans un format qui autorise une variation de plus ou moins plusieurs millions.

Pour cela, Stata a défini un type de stockage pour les variables numériques du fichier. A chaque type correspond une *étendue* de mémoire, exprimée en nombre de *byte*, unité de mesure de stockage informatique. Cinq types de stockage sont définis, avec chacun des valeurs minimum et maximum :

Type	Étendue	Minimum	Maximum	Manquante
byte	1	-127	+126	+127
int	2	-32 768	+32 767	+32 768
long	4	-2 147 483 648	+2 147 483 646	-2 147 483 647
float	4	$\pm 10^{-37}$	$\pm 10^{37}$	2^{128}
double	8	$\pm 10^{-99}$	$\pm 10^{99}$	2^{333}

Les types **byte** et **int** ne peuvent stocker que des entiers, tandis que les autres types peuvent stocker tous les nombres entiers ou décimaux.

On voit immédiatement que les types **byte** et **int** prennent beaucoup moins de place que les autres. Stata établit généralement le type de stockage le plus économique, mais les nouvelles variables (notamment lors de la création d'un nouveau fichier à partir d'un fichier ASCII) sont de type **float** par défaut. Il est donc préférable autant que possible de réduire les types de stockage, ce qu'on fait simplement comme suit :

```
. compress
```

Si plus tard, une variable d'un type de stockage donné devait prendre un type supérieur (après quelques modifications), Stata identifierait automatiquement le type nécessaire au bon stockage de la variable.

Par ailleurs, on note que les valeurs manquantes sont toujours supérieures à n'importe quelle autre valeur de la variable pour un type donné. Cela a quelques conséquences que nous traiterons dans le chapitre suivant *Création et correction de variables*.

Quant aux variables alphanumériques (*string*), elles prennent autant de *bytes* que de caractères : par exemple, une variable de type **`str14`** prendra 14 *bytes* en mémoire. Les variables alphanumériques prennent généralement plus de place que les variables numériques. C'est pourquoi il est préférable de les codifier autant que possible, sans compter que les manipulations statistiques et les tabulations sont nettement plus aisées avec des variables numériques.

4.3.2 Les formats numériques de lecture et d'affichage

Les formats numériques sont utiles pour la lecture des données à l'aide de la commande **`infile`**, et pour la restitution des données à l'écran (avec **`list`**, ou avec l'éditeur intégré **`edit`**) ou dans les tableaux statistiques (avec **`tabulate`** ou avec des commandes statistiques plus complexes).

On doit bien distinguer les types (en référence à la place en mémoire de chaque variable) et les formats numériques, qui portent sur l'interprétation d'un nombre. Par exemple, on peut avoir besoin d'interpréter le nombre 2,5 comme deux unités et demie, deux millions et demi ou bien deux centièmes et demi. De plus, on peut avoir besoin de connaître ce nombre à la première décimale près, tout comme à l'unité (3), à la deuxième décimale (2,51) ou à la sixième décimale près (2,509432).

Pour définir un format numérique, il faut savoir combien de chiffres décimaux (après le point, qui marque la décimale dans Stata) seront affichés et sur combien d'espaces au maximum le chiffre tiendra (y compris le signe et le point de décimal). On exprime cela sous la forme *%w.d* où *w* (comme *width*, largeur ou étendue) est le nombre maximum d'espaces, et *d* le nombre de décimales. Logiquement *d* est inférieur à *w*.

À cela s'ajoutent trois formats numériques : **e**, **f** et **g**. Le format **e** signifie que le nombre est exprimé en puissance (positive ou négative) de 10. Par exemple, -2.5e+02 signifie -2.5 que multiplie 10^{+2}, soit -250, tandis que -2.5e-02 signifie -2.5 que multiplie 10^{-2}, soit -0.025.

Le format **f** est similaire au format de programmation en FORTRAN : les chiffres après la virgule sont arrondis de telle sorte à afficher toujours le même nombre de décimale.

Enfin, le format **g** est un format complexe qui restitue la meilleure précision possible (c'est-à-dire le plus de chiffres significatifs possible) en augmentant le nombre de décimales (*d* est dans ce cas un minimum) dans la limite du nombre d'espaces disponibles. Quelques exemples aideront à illustrer ces différents formats :

Chiffre exact	Format d'affichage %8.2e	%8.2f	%8.2g
2.509432	2.5e+00	2.51	2.50943
2509432	2.5e+06	2509432.00	2.5e+06
-100431.5	-1.0e+05	-100431.50	-100432
-10431.5	-1.0e+04	-10431.50	-10431.5
0.0000002	2.0e-07	0.00	2.0e-07

Le format **e** est généralement préféré par les physiciens ou des biologistes pour travailler sur des très grands nombres ou au contraire sur des fractions infinitésimales.

Le format **f** est plus adéquat pour les sciences humaines qui travaillent sur des chiffres à dimension... humaine.

Le format **g** est un compromis entre les formats **e** et **f** : il peut satisfaire tout le monde, bien qu'il puisse surprendre quelques-uns. Pourquoi, sous le format %8.2g qui est pourtant censé donner le plus de précision possible, le chiffre -100431.5 est-il arrondi à -100432 ? C'est que dans l'espace disponible prévu par l'affichage, il faut prévoir non seulement le chiffre après la

décimale, mais aussi la décimale elle-même (marquée par un point « . » dans Stata). Or avec un nombre maximum d'espaces de 8, et déjà 6 chiffres significatifs avant la décimale en plus du signe (« -100431 »), l'espace manque pour inscrire le point et le chiffre après la virgule (« .5 »). Le chiffre est donc arrondi à l'unité.

Ce qui est déjà déroutant avec l'exemple ci-dessus où le chiffre est négatif, l'est encore plus pour un chiffre positif avec le même nombre de chiffres significatifs. Il faut en effet savoir que le signe positif (« + ») est explicite, c'est-à-dire qu'il n'apparaît pas à l'affichage, mais qu'il est quand même pris en compte dans le décompte des espaces disponibles pour l'affichage. Les chiffres +100431.5 et -100431.5 seront tous les deux arrondis à l'unité avec un format %8.2g.

Ajoutons que le nombre d'espaces pour l'affichage est de 7 au minimum plus le signe (apparent quand il est négatif), soit 8 positions au total, et cela quel que soit le format spécifié. Il est donc inutile de fixer le nombre d'espaces à moins de 8 : le format de type %8... est le minimum requis. Comparez les formats précédents avec les formats où le nombre d'espaces est limité à 5 :

Chiffre exact	Format d'affichage %5.2e	%5.2f	%5.2g
2.509432	2.5e+00	2.51	2.51
2509432	2.5e+06	2509432.00	2.5e+06
-100431.5	-1.0e+05	-100431.50	-1.0e+05
-10431.5	-1.0e+04	-10431.50	-1.0e+04
0.0000002	2.0e-07	0.00	2.0e-07

Notez que le format `f` force l'affichage pour que le nombre de décimale soit toujours respecté.

Stata attribue un format par défaut à toute nouvelle variable, pour chaque type de stockage, pour assurer un affichage optimum dans la plupart des cas :

Type de stockage	Format par défaut
byte	**%8.0g**
int	**%8.0g**
long	**%12.0g**
float	**%9.0g**
double	**%10.0g**

Enfin, ces formats ont une signification différente pour la lecture d'un fichier de données (avec **infile**), et pour la restitution à l'écran ou dans des tableaux.

Lors de la lecture d'un fichier de données, le format indique la manière d'interpréter les chiffres inscrits sur des positions précises (*input formats*, noté *%infmts* par Stata). Ainsi, le chiffre 25 lu dans un fichier est interprété comme 25 par %2.0f, mais comme 2.5 par %2.1f.

La signification est différente pour les formats d'affichage (*display formats*, noté *%fmts* par Stata). Le chiffre original figurant en mémoire dans le fichier reste inchangé et garde la même précision. Les modifications du format d'affichage ne portent que sur la restitution de ce chiffre à l'écran ou dans les tableaux. Cela signifie en particulier que les calculs statistiques seront toujours faits de manière optimale, sur les chiffres originaux gardés en mémoire.

En guise d'exercice, créez le petit fichier suivant :


```
. input var1 var2

          var1       var2
  1. 0.034 1245000
  2. end
```

Testez les commandes :

```
. list
. format var1 %8.1f
. format var2 %8.1g
. generate var3=var1*var2
. list
. format var2 %10.3g
. list
. describe
. compress
. describe
```

4.3.3 Les libellés du fichier, des variables et des valeurs

Au moment du chargement du fichier en mémoire, le libellé du fichier apparaît entre parenthèses :

```
. use census
(1980 Census data by state)
```

Pour les nouveaux fichiers, aucun libellé n'apparaît par défaut : il est préférable d'en créer pour donner une courte description du fichier, en 31 caractères maximum :

```
. label data "USA: Recensement 1980 par Etat"
```

De même, le contenu des variables peut se comprendre par le choix judicieux d'un nom de variable, mais le plus souvent, comme les noms de variables sont limités à huit caractères, on préférera leur associer un libellé, en 31 caractères maximum :

```
. label var region "Region du recensement"
```

Mais on peut aller plus loin. Certaines variables représentent des catégories (ou modalités), codées chacune par un chiffre. La plupart du temps ces codes n'ont pas une signification en eux-mêmes et servent simplement à distinguer une catégorie d'une autre. Dans ce cas, il est préférable d'attribuer un nom à chacune de ces catégories (un libellé des valeurs), c'est-à-dire d'établir une correspondance entre les codes et les noms des catégories qu'ils représentent :

```
. label define cenreg 1 "Nord Est" 2 "Centre N" 3 Sud 4 Ouest, modify
```

L'option **`modify`** permet de corriger un label déjà existant, comme par exemple **`cenreg`** dans le fichier « census.dta ». Les libellés de chacune des catégories ne doivent pas dépasser 8 caractères. Les guillemets sont indispensables lorsqu'il y a des espaces dans le libellé. Pour ajouter un nouveau libellé, on tape :

```
. label define cenreg 9 Inconnue, add

. label list cenreg

cenreg:
           1 Nord Est
           2 Centre N
           3 Sud
           4 Ouest
           9 Inconnue
```

La correspondance définie par **label define** *corresp...*, ne doit pas être considérée comme une variable. En fait, *corresp* peut être attribuée à plusieurs variables :

```
. label define vrai 0 "Non" 1 "Oui"

. generate urbain=(popurban/pop)>.75

. generate actif=((pop18p-pop65p)/pop)>.6

. label values urbain vrai

. label values actif vrai

. tab actif urbain

           | urbain
    actif |       Non        Oui |     Total
----------+----------------------+----------
      Non |        20          3 |        23
      Oui |        15         12 |        27
----------+----------------------+----------
    Total |        35         15 |        50
```

On peut même attribuer à une correspondance entre valeurs et libellés le même nom qu'une variable du fichier :

```
. label define region 1 "Nord Est" 2 "Centre N" 3 Sud 4 Ouest 9 Inconnue

. label values region region
```

La commande **label define** *corresp...* définit des libellés pour une variable numérique. La commande **encode** fait en quelque sorte le travail inverse : à partir d'une variable alphanumérique (de type *string* **str#**) elle crée une nouvelle variable numérique, avec une correspondance entre les codes et les libellés. Cela est particulièrement utile à la suite d'une saisie avec **input**, d'un transfert de données avec l'éditeur intégré **edit**, ou bien d'une lecture de données avec **insheet** :

```
. rename region Aregion

. encode Aregion, gen(region)

. drop Aregion

. compress
region was float now byte
```

Effacer la variable alphanumérique **Aregion** n'est pas nécessaire, mais conseillé pour éviter d'encombrer le fichier, d'autant plus que les variables alphanumériques prennent beaucoup de place en mémoire. En outre, **compress** permet de réduire le type de stockage au minimum pour économiser de la place en mémoire.

4.3.4 Les notes associées au fichier ou aux variables

Sans doute trouvez-vous que le nombre de caractères autorisés pour les libellés est insuffisant pour bien documenter le fichier, les variables et les valeurs de ces variables. Vous avez raison. C'est pourquoi Stata permet l'ajout de notes, quasiment sans limite du nombre de caractères[2] et sans limite du nombre de notes. On doit les distinguer des simples commentaires précédés du caractère « * » qu'on peut faire apparaître dans un fichier de résultats « *.log » ou un programme « *.do ». Les notes associées aux données sont avant tout des pense-bêtes (sauf votre respect) qui peuvent notamment aider à communiquer des informations à des collègues qui utilisent le même fichier de données « *.dta » :

```
. note : Ce fichier est identique au fichier c:\stata\census.dta fourni par
Stata, mais j'ai traduit quelques expressions en français
```

Les numéros de notes apparaissent dans l'ordre où elles ont été saisies, mais il est souvent plus utile de mentionner la date avec les caractères réservés TS (comme *time stamps*) qui seront convertis en date et heure du jour :

```
. note : TS J'ai créé les variables urbain et actif
```

Notez que Stata n'aime pas les accents, donc ne soyez pas surpris de trouver quelques caractères bizarres à l'écran :

```
. note list

_dta:
  1.  Ce fichier est identique au fichier c:\stata\census.dta fourni par Stata,
>  mais j?ai traduit quelques expression en fran?ais
  2.  25 Apr 1997 11:11 J?ai cr?er les variables urbain et actif
```

Pour attacher des notes à des variables, il suffit d'indiquer le nom de ces variables. Hormis les libellés, ces notes sont particulièrement utiles pour indiquer comment les nouvelles variables ont été créées :

```
. label var urbain "Pop urbaine sup a 75%"

. note urbain : =(popurban/pop)>.75

. label var actif "Pop active sup a 60%"

. note actif : =((pop18p-pop65p)/pop)>.6
```

[2] En fait, 5400 caractères par note pour la version standard.

L'existence de notes est indiquée par une étoile « * » dans le descriptif du fichier, juste avant la colonne des libellés :

```
. describe

Contains data from census.dta
  obs:            50                          USA: Recensement 1980 par Etat
 vars:            14                          21 Mar 1997 15:02
 size:         3,400 (99.9% of memory free)   * _dta has notes
-------------------------------------------------------------------------------
   1. state       str14   %14s                 State
   2. region      int     %8.0g       cenreg   Region du recensement
   3. pop         long    %10.0g               Population
   4. poplt5      long    %10.0g               Pop, < 5 year
   5. pop5_17     long    %10.0g               Pop, 5 to 17 years
   6. pop18p      long    %10.0g               Pop, 18 and older
   7. pop65p      long    %10.0g               Pop, 65 and older
   8. popurban    long    %10.0g               Urban population
   9. medage      float   %9.2f                Median age
  10. death       long    %10.0g               Number of deaths
  11. marriage    long    %10.0g               Number of marriages
  12. divorce     long    %10.0g               Number of divorces
  13. urbain      float   %9.0g       vrai     * Pop urbaine sup a 75%
  14. actif       float   %9.0g       vrai     * Pop active sup a 60%
                                               * indicated variables have notes
-------------------------------------------------------------------------------
Sorted by:
     Note:  data has changed since last save
```

Enfin, pour supprimer des notes, on mentionne soit **_dta** pour indiquer les notes globalement associées au fichier, soit le nom de la variable, suivi éventuellement du numéro de note :

```
. note drop _dta in 2

. note drop urbain
```

Un conseil pour conclure : utilisez les libellés en prévision de la présentation des tableaux et des graphiques, et les notes pour une documentation plus technique, notamment lors de la création de nouvelles variables ou de la manipulation du fichier. Vous pourrez ainsi mieux repérer les erreurs... et les corriger !

4.3.5 Mettre en forme le fichier

La création de libellés et de notes aide considérablement la compréhension des données : c'est le travail principal de mise en forme du fichier. Outre cela, quelques petites améliorations esthétiques peuvent aider à réduire la taille du fichier et à le rendre plus lisible.

À la fin de chaque session de travail, avant de sauvegarder le fichier, prenez soin de compresser les formats de stockage :

```
. compress
region was int now byte
urbain was float now byte
actif was float now byte
state was str14 now str13
```

Il est en effet plus simple de taper **compress** que de définir *a priori* le format de stockage de chaque variable.

Ensuite, il est possible de regrouper les variables se rapportant au même thème en réorganisant l'ordre des variables dans le fichier :

```
. order state-pop65p actif popurban urbain
```

Cela ne modifie en rien le contenu des variables mais permet de les retrouver plus rapidement.

Enfin, il est souhaitable d'ordonner le fichier de manière la plus logique possible, pour repérer facilement une observation ou un groupe d'observations. Généralement, les fichiers contiennent un identifiant unique pour chaque observation :

```
. sort state
```

4.4 *Comment combiner plusieurs fichiers ?*

La combinaison des fichiers est une opération délicate que de nombreux utilisateurs craignent. S'apercevoir au moment de l'analyse que le fichier n'a pas été correctement construit est particulièrement énervant, puisqu'il faut alors tout recommencer depuis le début. Pour combiner des fichiers, trois principaux cas de figures peuvent se présenter :

- Les fichiers contiennent des informations sur des observations différentes, et on veut obtenir un fichier contenant l'ensemble des observations : par exemple, dans une enquête, les informations sur les individus d'un quartier peuvent être ajoutées aux informations recueillies dans un autre quartier. Dans ce cas, il faudra utiliser la commande **append**.

- Les fichiers contiennent des informations sur les mêmes observations, et on veut obtenir un fichier contenant l'ensemble des variables de tous les fichiers : par exemple, à la suite d'un deuxième passage auprès d'un échantillon d'individus, des informations supplémentaires ont pu être ajoutées aux informations déjà recueillies. Dans ce cas, la commande adéquate est **`merge`**. Les options **`update`** et **`replace`** permettent aussi d'actualiser les variables.

- Un des fichiers contient des observations sur un niveau hiérarchique inférieur à l'autre fichier, et on veut obtenir un fichier contenant les variables de tous les niveaux hiérarchiques pour l'ensemble des observations du niveau hiérarchique inférieur : par exemple, les informations sur les enfants peuvent être complétées par les informations sur leur mère, qui seront les mêmes dans le cas des frères et soeurs. On utilisera alors la commande **`merge`**.

Ces commandes permettent de combiner des fichiers deux à deux. Pour combiner plus de deux fichiers, il faut d'abord combiner deux fichiers, puis combiner le fichier obtenu avec le troisième fichier, etc. Par convention, on appellera le premier fichier « fichier maître », et le fichier ajouté « fichier appelé ».

4.4.1 La commande `append`

Pour se rappeler ce que fait la commande **`append`**, on peut penser au mot « appendice » : la combinaison des deux fichiers se fait par ajout des observations du fichier appelé à la suite des observations du premier fichier, le fichier maître en mémoire.

Supposez que le fichier maître soit composé des observations pour les dix premiers États des USA, et le fichier appelé des quarante autres États. Pour combiner ces deux fichiers, on place d'abord le premier fichier en mémoire et on ajoute les observations du deuxième :

```
. use census10
(Recensement 10 premiers Etats)

. append using census40
(label cenreg already defined)

. save census
file census.dta saved
```

Stata indique que le libellé **cenreg** est déjà défini dans le fichier maître « census10.dta » : c'est ce libellé qui sera utilisé dans le fichier final « census.dta ».

On aurait pu tout aussi bien combiner les fichiers dans l'ordre inverse :

```
. use census40
(Recensement 40 derniers Etats)

. append using census10
(label cenreg already defined)
```

Mais il faut se rappeler que les libellés et les notes du fichier maître (pour les variables, les valeurs des variables, et pour le fichier dans son ensemble) prévalent sur ceux du fichier appelé. On a donc tout intérêt à choisir comme fichier maître celui des deux fichiers qui contient les libellés et les notes les plus précis, ou les plus récents. On pourra toujours ensuite trier les observations comme il convient :

```
. sort state
```

L'ordre d'appel des fichiers n'a par contre aucun effet sur la précision des données : si une variable n'a pas le même type de stockage (voir plus haut) dans l'un et l'autre fichier, le type le plus précis est choisi, afin de ne pas perdre d'informations.

Mais qu'advient-il si les deux fichiers ne contiennent pas exactement les mêmes variables ? Par exemple, le fichier maître pourrait ne pas contenir la variable **divorce** tandis que le fichier appelé pourrait ne pas contenir la variable **marriage**. Dans ce cas, le fichier final contiendra toutes les variables, y compris les deux variables **divorce** et **marriage**, mais le code manquant « **.** » sera attribué aux observations pour lesquelles la variable n'est pas renseignée :

```
. list state marriage divorce

                state    marriage     divorce
  1.          Alabama       49018           .
  2.           Alaska        5361           .
  3.          Arizona       30223           .
  4.         Arkansas       26513           .
  5.       California      210864           .
  6.         Colorado       34917           .
  7.      Connecticut       26048           .
  8.         Delaware        4437           .
  9.          Florida      108344           .
 10.          Georgia       70638           .
 11.           Hawaii           .        4438
 12.            Idaho           .        6596
 13.         Illinois           .       50997
 14.          Indiana           .       40006
 15.             Iowa           .       11854
 16.           Kansas           .       13410
(suite de la liste)
 48.    W. Virginia           .       10273
 49.        Wisconsin         .       17546
 50.          Wyoming         .        4003
```

Après avoir combiné deux fichiers, posez-vous les questions suivantes et exécuté les commandes nécessaires pour mettre en forme le nouveau fichier :

- *Quel sera le libellé du nouveau fichier ? Le libellé du fichier maître ou bien celui du fichier appelé ?*
- *Sous quel nom le fichier sera-t-il sauvegardé si on exécute la commande suivante ?*

```
. save, replace
```

- *Quel contenu aura le nouveau fichier si aucune variable du fichier appelé n'existe dans le fichier maître ?*
- *Quel contenu aura le nouveau fichier si des observations du fichier appelé existent déjà dans le fichier maître ? Seront-elles dupliquées ou non ?*

4.4.2 La commande `merge` pour les fichiers non hiérarchisés

La traduction de *to merge* est « fusionner ». Mais cela donne une idée assez vague de ce que fait la commande **`merge`**. Il faut donc se souvenir que cette commande fusionne des variables pour un même ensemble d'observations, contrairement à **`append`** qui ajoute les observations d'un fichier au dessous d'un autre.

Supposez que le fichier maître contienne seulement, pour chaque États des USA, les effectifs de population, et que les autres

variables figurent dans le fichier appelé. En fusionnant ces deux fichiers, on ajoute les variables qui manquent :

```
. use censautr
(Autres variables du recensement)

. sort state

. use censpop
(Population au recensement)

. sort state

. merge using censautr
(label cenreg already defined)

. save census
file census.dta saved
```

Tout comme pour la commande **append**, les libellés et les notes du fichier maître prévalent sur ceux du fichier appelé. On a donc tout intérêt à choisir comme fichier maître celui des deux fichiers qui contient les libellés et les notes les plus précis, ou les plus récents. L'ordre d'appel des fichiers n'a par contre aucun effet sur la précision des données : le type de stockage (voir plus haut) le plus précis est choisi, sans perte d'informations.

Avant d'exécuter la commande **merge**, on a trié les fichiers maître et appelé. Pourquoi a-t-on pris cette peine ? Parce que **merge** fusionne les observations une à une dans l'ordre où elles sont dans les fichiers : la première observation du fichier maître est fusionnée avec la première observation du fichier appelé, la deuxième observation est fusionnée avec la deuxième, etc. On doit donc toujours s'assurer que les mêmes observations figurent bien dans les deux fichiers, et qu'elles sont bien triées dans le même ordre, avant d'exécuter **merge**.

Pour éviter toute erreur, il vaut mieux apparier les fichiers (*match merge*) selon l'identifiant de chaque observation. Pour cela, les fichiers doivent d'abord être triés selon une variable qui définit de manière unique chaque observation (**state** dans notre exemple). Cette variable servira pour l'appariement :

```
. merge state using censautr
(label cenreg already defined)
```

Mais que se passe-t-il si une observation n'existe pas dans un des deux fichiers ? Dans ce cas, les variables de ce fichier ne seront pas renseignées pour l'observation en question. Pour mieux contrôler le résultat d'une telle fusion, et repérer ainsi les observations qui ne figureraient pas dans les deux fichiers, la commande **merge** crée après chaque exécution une variable intitulée **_merge** codée comme suit :

1. si l'observation ne figure que dans le fichier maître,

2. si l'observation ne figure que dans le fichier appelé,

3. si l'observation figure dans les deux fichiers.

Par exemple, si la Californie manque dans le fichier appelé, on aura :

```
. list state popurban medage divorce _merge

             state    region    popurban     medage     divorce     _merge
  1.       Alabama     South     2337713      29.30       26745          3
  2.        Alaska      West      258567      26.10        3517          3
  3.       Arizona      West     2278728      29.20       19908          3
  4.      Arkansas     South     1179556      30.60       15882          3
  5.    California      West    21607606          .           .          1
  6.      Colorado      West     2329869      28.60       18571          3
  7.   Connecticut        NE     2449774      32.00       13488          3
(suite de la liste)
```

Les informations pour la Californie ne figuraient que dans le fichier maître (**_merge==1**), et par conséquent les variables **medage**, **death**, **marriage** et **divorce** sont codées manquantes pour cet État.

La commande **merge** possède une option particulière pour actualiser des données : **update**. Supposons par exemple, que les nombres de mariages et de divorces étaient provisoires dans le fichier maître et que le fichier appelé contienne les résultats définitifs après correction des erreurs dans les États de l'Arizona, de Californie et de Floride :

```
. use censact
(Actualisation: marriage divorce)

. sort state

. list in 1/10

             state    marriage     divorce
  1.        Alabama       49018       26745
  2.         Alaska        5361        3517
  3.        Arizona       30223       19908
  4.       Arkansas       26513       15882
  5.     California      210864      133541
  6.       Colorado       34917       18571
  7.    Connecticut       26048       13488
  8.       Delaware        4437        2313
  9.        Florida      108344       71579
 10.        Georgia       70638       34743

. save censact, replace
(Actualisation: marriage divorce)

. use censold
(1980 Census data by state)

. sort state

. list state marriage divorce in 1/10

             state    marriage     divorce
  1.        Alabama       49018       26745
  2.         Alaska        5361        3517
  3.        Arizona           .           .
  4.       Arkansas       26513       15882
  5.     California      199934      123432
  6.       Colorado       34917       18571
  7.    Connecticut       26048       13488
  8.       Delaware        4437        2313
  9.        Florida      100232           .
 10.        Georgia       70638       34743
```

On remarque en comparant les fichiers « censact.dta » et « censold.dta » que les variables **marriage** et **divorce** ont des valeurs manquantes pour les États de l'Arizona et de la Floride, et que les chiffres diffèrent pour les États de Californie et de Floride. Pour actualiser « censold.dta » avec les données de « censact.dta », on exécutera :

```
. merge state using censact, update replace

. tab _merge

    _merge |      Freq.     Percent        Cum.
-----------+-----------------------------------
         3 |         47       94.00       94.00
         4 |          1        2.00       96.00
         5 |          2        4.00      100.00
-----------+-----------------------------------
     Total |         50      100.00
```

```
. list state marriage divorce _merge in 1/10

                state    marriage     divorce     _merge
  1.          Alabama       49018       26745          3
  2.           Alaska        5361        3517          3
  3.          Arizona       30223       19908          4
  4.         Arkansas       26513       15882          3
  5.       California      210864      133541          5
  6.         Colorado       34917       18571          3
  7.      Connecticut       26048       13488          3
  8.         Delaware        4437        2313          3
  9.          Florida      108344       71579          5
 10.          Georgia       70638       34743          3
```

Avec l'option **update**, deux nouveaux codes ont été créés pour la variable **_merge** :

4. si l'observation contenait des valeurs manquantes dans le fichier maître qui ont été actualisées par les valeurs du fichier appelé,

5. si l'observation contenait des valeurs non manquantes différentes dans le fichier maître et dans le fichier appelé : en ajoutant l'option **replace**, les valeurs du fichier appelé actualisent les valeurs non manquantes du fichier maître ; sans l'option **replace**, les valeurs non manquantes du fichier maître ne sont pas actualisées.

Après avoir combiné deux fichiers, posez-vous les mêmes questions que dans l'exercice précédant pour la commande **append**, *puis répondez aux questions suivantes :*

- *Que se passe-t-il si on fusionne deux fichiers contenant les mêmes observations mais dans un ordre différent sans contrôler selon une variable d'appariement ? Comment sera codée la variable* **_merge** *?*
- *Après fusion de deux fichiers, comment seront codées les variables si le fichier contient une observation supplémentaire qui n'existe pas dans le fichier maître ? Comment sera codée la variable* **_merge** *pour cette observation ?*
- *Dans notre exemple d'actualisation avec l'option* **update**, *quelles seront pour l'État de Californie, les valeurs des variables* **marriage** *et* **divorce** *si l'on omet l'option* **replace** *? Celles du fichier maître ou bien celles du fichier appelé ?*
- *Dans le même exemple, pourquoi la variable* **_merge** *n'est elle pas codée* **4** *alors que la variable* **divorce** *était manquante dans le fichier maître ?*

4.4.3 La commande `merge` pour les fichiers hiérarchisés

L'appariement pour la fusion de fichiers non hiérarchisés est optionnel mais fortement recommandé, comme on l'a expliqué plus haut, pour éviter toute erreur liée à l'ordre des observations dans les fichiers maître et appelé. Pour la fusion de fichiers hiérarchisés, l'appariement selon une variable identifiant le niveau hiérarchique supérieur est indispensable.

Supposons que le fichier « censreg.dta » contienne les informations régionales issues du Recensement des USA, à la différence du fichier « census.dta » qui contient les informations sur les États. Pour fusionner les deux fichiers, on devra apparier selon la variable **`region`** qui représente un niveau hiérarchique supérieur au États :

```
. use census
(1980 Census data by state)

. sort region

. save, replace

. use censreg
(Recensement regional)

. list

      region        popreg
  1. Nord Est      49135284
  2. Centre N      58865672
  3.      Sud      74734032
  4.    Ouest      43172488

. sort region

. merge region using census

. sort region state

. list region popreg state pop _merge

      region        popreg          state         pop    _merge
  1. Nord Est      49135284    Connecticut     3107576         3
(suite de la liste)
  9. Nord Est      49135284        Vermont      511456         3
 10. Centre N      58865672       Illinois    11426518         3
(suite de la liste)
 21. Centre N      58865672      Wisconsin     4705767         3
 22.      Sud      74734032        Alabama     3893888         3
(suite de la liste)
 37.      Sud      74734032   W. Virginia     1949644         3
 38.    Ouest      43172488         Alaska      401851         3
(suite de la liste)
 50.    Ouest      43172488        Wyoming      469557         3
```

On remarque que la variable **_merge** est codée **3** pour toutes les observations : les quatre régions figuraient dans les deux fichiers. La variable **popreg** a été répliquée pour chacun des États. Le fichier final contient bien la totalité des observations du niveau hiérarchique inférieur, c'est-à-dire 50 États.

Dans le cas où une région ne figurerait pas dans le fichier des régions, la variable **popreg** serait codée manquante pour les États de cette région, et par conséquent la variable **_merge** sera codée **2** pour indiquer que l'information provient du fichier appelé « census.dta ».

À l'inverse, si le fichier des États ne contient pas les données sur les États d'une région (par exemple la région Sud), les variables concernant ces États seront codées manquantes. Une seule observation représentera cette région, et la variable **_merge** sera codée **1** pour cette observation, pour indiquer que l'information provient du fichier maître « censreg.dta » :

```
. list region popreg state pop _merge

      region        popreg          state        pop    _merge
  1. Nord Est     49135284    Connecticut    3107576         3
  2. Nord Est     49135284          Maine    1124660         3
(suite de la liste)
  9. Nord Est     49135284        Vermont     511456         3
 10. Centre N     58865672       Illinois   11426518         3
(suite de la liste)
 21. Centre N     58865672      Wisconsin    4705767         3
 22.      Sud     74734032              .          .         1
 23.    Ouest     43172488         Alaska     401851         3
 24.    Ouest     43172488        Arizona    2718215         3
(suite de la liste)
 37.    Ouest     43172488     Washington    4132156         3
 38.    Ouest     43172488        Wyoming     469557         3
```

Le fichier ne contient plus que 38 observations qui correspondent aux 37 États pour lesquels on a des informations, plus 1 observation pour les informations sur la région Sud.

Généralement, l'ordre d'appel des fichiers n'importe pas pour la fusion des fichiers hiérarchisés. Dans la mesure où une variable d'appariement est utilisée, les deux commandes :

```
. use census

. merge state using censreg
```

... donnent un résultat absolument identique aux deux commandes :

```
. use censreg

.. merge state using census
```

La seule différence entre les résultats est esthétique : dans le premier cas, les variables contenues dans le fichier appelé « censreg.dta » seront stockées à la suite des variables du fichier « census.dta », alors que ce sera l'inverse dans le second cas.

Il y a cependant une exception à cette règle : l'ordre d'appel importe dans le cas où la fusion est doublée d'une actualisation. En effet, comme pour les fichiers non hiérarchisés, on peut utiliser les options **`update`** et **`replace`**. Ces options serviront pour actualiser les informations d'un niveau hiérarchique à l'autre. Il faudra alors que le fichier maître corresponde au niveau hiérarchique à actualiser, et que le fichier appelé contienne les informations les plus récentes.

Par exemple, si les chiffres de population au niveau régional étaient le résultat d'un décompte provisoire du Recensement, et si le fichier « census.dta » contenait une variable **`popreg`** actualisée sur la base des décomptes définitifs de chaque État, alors, on choisirait « censreg.dta » comme fichier maître.

Répondez aux questions suivantes :

- *Dans notre exemple de fusion des fichiers hiérarchisés, qu'advient-il si les informations sur les États de l'Ouest manquent en plus des informations sur les États du Sud ? Combien d'observations contiendra le fichier final ?*
- *Dans l'exemple d'actualisation des données sur la population régionale avec les options* **`update`** *et* **`replace`**, *qu'advient-il si l'ordre des fichiers maître et appelé est inversé ? Dans le fichier final, la variable* **`popreg`** *proviendra de quel fichier ? Et si on inverse l'ordre mais sans l'option* **`replace`** *?*

Voici quelques conseils avant de combiner des fichiers :

- *Conservez toujours une copie des fichiers originaux dans un répertoire spécifique avant de procéder à une fusion avec* **append** *ou avec* **merge**.
- *Comparez toujours le nombre d'observations dans le fichier maître et dans le fichier appelé, et vérifiez que le nombre atteint dans le fichier final est bien celui attendu.*
- *Ne fusionnez jamais les fichiers avec* **merge** *sans appariement selon l'identifiant de chaque observation.*
- *Après avoir exécuter* **merge**, *vérifiez toujours le résultat à l'aide de la variable* **_merge**
- *Pour la fusion des fichiers hiérarchisés avec* **merge**, *utilisez le niveau hiérarchique supérieur comme fichier maître.*
- *Pour l'actualisation des fichiers avec l'option* **update** *de* **merge**, *utilisez l'option* **replace** *seulement après vous être assuré que le fichier appelé contient bien les informations les plus récentes.*
- *Avec* **append** *comme avec* **merge**, *triez le fichier selon l'identifiant de chaque observation. Pour les fichiers hiérarchisés, créez un identifiant pour chaque observation qui inclut l'identifiant de chaque niveau hiérarchique (par exemple :* **by region : generate ident=region*10 + _n**) *: ensuite, triez le fichier selon cet identifiant, ou bien selon les identifiants de chaque niveau hiérarchique par ordre d'importance (par exemple :* **sort region state**).

5 CRÉATION ET CORRECTION DE VARIABLES

5.1 *Les commandes* `generate` *et* `replace`

La commande **generate** crée de nouvelles variables. Elle a la syntaxe de base suivante :

[**by** *listevar*:] **generate** *var* = *exp*
[**if** *exp*] [**in** *intervalle*]

La commande **replace** utilise la même syntaxe, sauf qu'elle s'applique aux variables déjà existantes.

Comme on le voit, cette syntaxe est simple, ce qui n'est pas le cas de la forme que peut prendre *exp*. La première expression *exp* (après le signe =) spécifie le contenu de la variable, c'est-à-dire le plus souvent une valeur numérique. La seconde expression *exp* (après **if**) doit être formulée comme une expression logique dont le résultat est soit vrai soit faux : la création (ou le remplacement) de la variable est restreint aux observations pour lesquelles le résultat de l'expression est vrai.

Cela n'a l'air de rien, mais la confusion entre les deux expressions est certainement l'erreur la plus fréquente que peuvent faire les utilisateurs de Stata. À bon entendeur...

Avant de passer aux précautions à prendre pour éviter toute confusion, nous allons d'abord expliquer les opérateurs et les fonctions statistiques.

5.1.1 Les opérateurs arithmétiques, relationnels et logiques

Les opérateurs arithmétiques de Stata sont bien classiques : **+** (addition), **-** (soustraction), ***** (multiplication), **/** (division), **^** (puissance), tout comme les opérateurs relationnels **>** (supérieur), **<** (inférieur), **>=** (supérieur ou égal), **<=** (inférieur ou égal).

C'est peut-être moins le cas des opérateurs relationnels **==** (égal) ou **~=** (différent, que l'on peut écrire aussi **!=**), et des opérateurs logique **&** (et), **|** (ou bien), et **~** (non).

En effet, Stata distingue le signe **=** (affectation d'une valeur) du signe **==** (égalité entre deux valeurs). Dans le cas d'une affectation d'une valeur à une variable, la variable apparaît à gauche du signe **=** tandis que la valeur affectée apparaît à droite :

```
. generate 100*popurban/pop=pcturb
100 invalid name
r(198);

. generate popurban*100/pop=pcturb
popurban already defined
r(110);

. generate pcturb=100*popurban/pop
```

Au contraire, dans le cas d'une égalité entre deux valeurs, l'ordre peut être inversé sans que cela affecte l'égalité logique :

```
. generate ouest=region==4

. replace ouest=4==region
(0 real changes made)
```

Les expressions logiques sont particulièrement utiles pour créer des variables dichotomiques, c'est-à-dire qui ne prennent que deux valeurs, 0 et 1. En effet, une expression logique, c'est-à-dire une expression où interviennent les opérateurs relationnels **>**, **<**, **>=**, **<=**, **==**, **~=**, **!=**, ou bien les opérateurs logiques **&**, **|**, et **~**, est codée 1 lorsque son résultat est vrai, et codée 0 lorsque son résultat est faux :

```
. tabulate ouest region

           | Census region
     ouest |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+--------------------------------------------+----------
         0 |         9         12         16          0 |        37
         1 |         0          0          0         13 |        13
-----------+--------------------------------------------+----------
     Total |         9         12         16         13 |        50
```

Comme on le voit, Stata utilise une forme très condensée pour formuler une expression logique. Une forme moins condensée pour créer la variable dichotomique **ouest** serait :

```
. generate ouest=cond(region==4,1,0)
```

... ce qui se lit « si la condition **region==4** est vérifiée, affecter la valeur 1 à la variable **ouest**, sinon affecter la valeur 0 ». À la différence de la formulation condensée dont le résultat est soit 0 soit 1, la fonction **cond()** permet d'affecter n'importe quelles valeurs lorsque la condition est vraie ou fausse (par exemple 1 et 2 plutôt que 0 et 1).

La forme condensée est source de perplexité pour les nouveaux utilisateurs de Stata, particulièrement s'ils ont l'habitude des logiciels statistiques qui ne distinguent pas le signe d'affectation **=** et le signe d'égalité **==** et qui par conséquent utilisent systématiquement des fonctions telles que **cond()** pour évaluer le résultat d'une expression logique.

Si vous en avez l'habitude, vous pouvez continuer d'utiliser les fonctions logiques telles que **cond()**. Cependant, plus une commande est longue à taper, plus des erreurs risquent de s'y glisser qui peuvent avoir des conséquences fâcheuses sur les résultats. Par exemple :

```
. generate ouest=cond(region==4,0,1)
```

Dans ce cas (notez l'inversion des valeurs 0 et 1), aucun message d'erreur n'est renvoyé à l'écran puisque la syntaxe reste correcte. Mais le résultat est inverse à ce qui était attendu.

C'est pour éviter ce genre d'erreur d'inattention qu'il vaut mieux prendre l'habitude de la formulation condensée de Stata, tout aussi logique et plus économe. En outre, c'est cette formulation qui est utilisée par l'option **if** pour restreindre l'exécution de la commande à certaines observations.

Par ailleurs, l'ordre d'évaluation des opérateurs est **~** (non), **^**, **-** (négation), **/**, *****, **-** (soustraction), **+**, **~=** (ou **!=**), **>**, **<**, **<=**, **>=**, **==**, **&**, **|**. Comme pour toute formulation logique, les parenthèses doivent être utilisées le plus souvent possible, pour éviter les confusions.

Après avoir essayé de deviner le résultat des calculs, corrigez les commandes et comparer les résultats :

```
. gen ouest=cond(region==4,4,0)
. by region: gen ouest=region==4
. gen bidon=popurban/pop>.75
. gen bidon=.75<popurban/pop
. gen bidon=(.75<popurban)/pop
. gen bidon=popurban/pop>.75 | pop18p-pop65p/pop>.6
. gen bidon= popurban/pop | pop18p-pop65p
. gen bidon=popurban/ln((poplt5+pop5_17)/(pop-pop18p))
```

La commande **tabulate** *possède une option* **generate()** *bien pratique pour créer une série de variables dichotomiques à partir d'une variable polytomique. Exécutez la série de commandes :*

```
. tabulate region, generate(reg)
. describe reg*
. tabulate reg3 region
. gen nord=reg1+reg2
. tabulate nord region
```

5.1.2 Les fonctions mathématiques, statistiques, etc.

De nombreuses fonctions sont disponibles dans Stata : elles apparaissent dans des expressions (notées *exp*), et elles ont chacune un nom et un ou plusieurs arguments ordonnés :

nomfonction(arg1 [,arg2...])

Chaque argument peut lui-même être composé d'une expression ou d'une fonction. Ainsi, on peut combiner plusieurs fonctions (à condition que cela ait un sens), comme par exemple les fonctions **normd()**, **round()** et **log()** :

```
. generate bidon=normd(2+round(log(medage),.01))
```

Nous ne décrirons pas ici les nombreuses fonctions mathématiques et statistiques disponibles, ce que font très bien les sections [R] **functions** et [U] **20.3 Functions** du manuel de Stata. Nous mentionnerons cependant les plus utiles et les plus

communes pour la création de nouvelles variables : **cond()**, **int()**, et **round()**.

Nous avons déjà eu un aperçu de la fonction **cond(***exp,a,b***)** à propos des expressions logiques. Si l'expression *exp* est vrai, la fonction retient la valeur *a* et sinon la valeur *b*. Les valeurs *a* et *b* peuvent prendre la forme d'expressions, y compris la valeur manquante (symbolisée par un point « . »), comme par exemple :

```
. generate bidon=cond(region==3,., pop/2)
```

La fonction **int(***x***)** retient l'entier de *x* obtenu par troncature. Ainsi, la valeur 2,123 est évaluée à 2 et la valeur 1,89 à 1. La valeur de *x* peut être obtenue à l'aide d'une expression ; l'important est que le résultat de l'expression soit un nombre :

```
. generate bidon=int((pop18p-pop65p)/pop)
```

Le résultat est très différent avec la fonction **round(***x,y***)** qui arrondi la valeur 2,123 à 2 tout comme 1,89. La précision de l'arrondi est définie par *y*, c'est-à-dire que *x* est arrondi au plus proche multiple de *y* : ainsi **round(2.123,.1)** est évalué à 2,1 et **round(1.89,.2)** à 1,8 tandis que **round(2.123,3)** est évalué à 3 et **round(1,89,0.5)** à 2. Là aussi *x* et *y* peuvent être obtenus à l'aide d'expressions :

```
. generate bidon=round((pop18p-pop65p)/pop,10^-2)
```

5.1.3 Quelques précautions à prendre

Tout d'abord, il faut savoir que Stata attribue une valeur manquante à toute opération arithmétique impossible (par exemple une division par 0) ou pour toute opération sur une valeur manquante. C'est parfaitement logique, mais c'est quelque chose qu'on oublie facilement.

Une autre source très fréquente d'erreur, plus insidieuse celle-là, est l'utilisation des signes **>** ou **>=** dans le cas où la variable contient des valeurs manquantes. Comme il a été dit plus haut à propos des types de variables, les valeurs manquantes sont considérées dans Stata comme supérieures à n'importe quelle autre valeur. En conséquence, la commande :

```
. generate urbain=(popurban/pop)>.75
```

... génère la valeur 1 pour la variable **urbain** lorsque la variable **popurban** est manquante ou bien lorsque **pop** est manquante ou égale à 0. En effet, la valeur **(popurban/pop)** est manquante et par conséquent supérieure à la valeur 0,75.

De même, l'utilisation des signes **~=** ou bien **!=** (différent) est une source fréquente d'erreur. La commande :

```
. generate nord=region!=3 & region!=4
```

... génère la valeur 1 pour la variable **nord** lorsque la variable **region** est manquante, car alors la valeur de la variable est différente de 3 et de 4.

À moins d'être assuré qu'aucune variable n'est codée manquante pour aucune observation dans le fichier, il vaut mieux toujours effectuer les calculs pour les variables renseignées, c'est-à-dire non manquantes. L'option **if** est un excellent garde-fou :

```
. generate urbain=(popurban/pop)>.75 if popurban!=. & pop!=.

. generate nord=region!=3 & region!=4 if region!=.
```

Ceci nous amène à une autre source d'erreur que nous avions mentionnée au début de ce chapitre : la confusion entre l'expression (logique ou non) qui spécifie le contenu de la variable, et l'expression logique de l'option **if** qui restreint le calcul à certaines observations. Par exemple, la commande :

```
. generate ouest=region==4
```

... a une signification très différente de la commande :

```
. generate ouest=1 if region==4
```

Dans le premier cas, la variable **ouest** est égale à 1 pour les observations qui vérifient la condition logique **region==4**, et à 0 pour les autres observations. Dans le deuxième cas, la variable **ouest** est égale à 1 pour les observations qui vérifient la condition logique **region==4**, et elle sera codée manquante (un point « . ») pour les autres observations : le calcul est restreint aux États de la région Ouest.

Ce type de confusion provient du fait que de nombreux logiciels statistiques n'utilisent pas les expressions logiques : chaque

valeur d'une nouvelle variable était définie. Ainsi, on devait exécuter deux lignes de commande, du type :

```
. generate ouest=1 if region==4
. replace ouest=0 if region!=4
```

En somme, l'option **if** ne doit jamais être utilisée pour créer une variable de type dichotomique (0/1, vrai/faux). Elle est seulement nécessaire pour :

- restreindre le calcul aux variables renseignées, c'est-à-dire non manquante (ex : **gen ouest=reg==4 if reg!=.**) ;
- restreindre le calcul à certaines observations selon un critère différent de celui qui sert à créer la nouvelle variable (ex : **gen urbnord=(popu/pop)>.75 if reg!=3 & reg!=4**) ;
- corriger les valeurs d'une variable déjà existante (ex : **replace pop=23668031 if state=="California"**).

On peut ainsi édicter la règle suivante, qui vaut dans la grande majorité des cas : en exécutant **generate** =*exp* **if** *exp*, les variables qui apparaissent dans =*exp* ne doivent pas en principe figurer dans **if** *exp*.

5.1.4 Les corrections par Edit

Une alternative à **replace** sous Windows ou Macintosh est l'éditeur de Stata auquel on accède par le bouton **Edit** ou par la commande **edit**. Le bouton **Edit** est équivalent à la commande **edit** sans argument, c'est-à-dire que l'ensemble du fichier est accessible sans restriction sur les observations ou les variables. S'il s'agit de consulter simplement le fichier, sans le modifier, il est préférable alors d'utiliser le bouton **Browse** ou bien la commande **browse**, comme alternative à la commande **list**.

Lorsqu'il s'agit de modifier les données, il est préférable de restreindre l'affichage de l'éditeur aux seules variables et observations à modifier. Cela vous évitera des erreurs de manipulation que vous pourriez regretter. Pour cela, une fois dans l'éditeur, utilisez le bouton **Hide** pour cacher certaines variables.

Mais il est préférable d'utiliser les options **if** et **in** de la commande **edit** pour restreindre l'édition non seulement à certaines variables mais aussi à certaines observations. Ainsi, pour éditer les variables **pop** et **popurban** pour la région Ouest, on tapera :

```
. edit pop popurban if region==4
```

Le même principe vaut pour corriger une seule observation :

```
. edit pop popurban in 34
```

L'option **in** nécessite de connaître le numéro d'ordre de l'observation à modifier parmi l'ensemble des observations dans le fichier. L'observation N°34 ne correspond pas au même État selon que le fichier est classé par ordre alphabétique des noms des États, ou par région.

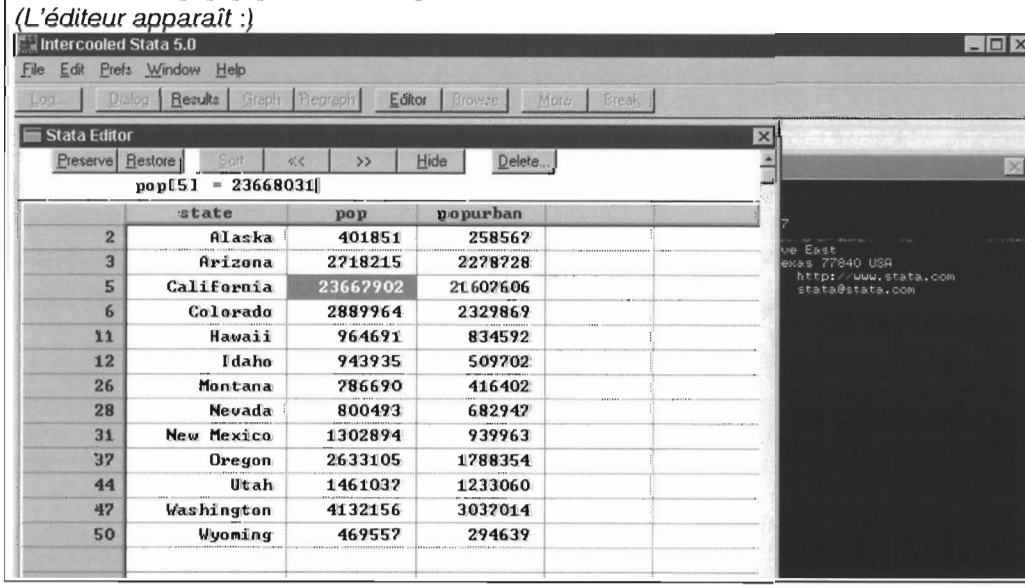
Quelle que soit la procédure de sélection (par **if** ou par **in**), pour bien s'assurer que l'édition portera sur les bonnes observations, il est préférable d'inclure l'identifiant des observations dans la liste des variables :

```
. edit state pop popurban if region==4

. edit state pop popurban in 34
```

Une fois dans l'éditeur, cliquez sur une cellule pour voir apparaître la valeur de la variable pour l'individu concerné :

```
. edit state pop popurban if region==4
```

(L'éditeur apparaît :)



	state	pop	popurban
2	Alaska	401851	258567
3	Arizona	2718215	2278728
5	California	23667902	21602606
6	Colorado	2889964	2329869
11	Hawaii	964691	834592
12	Idaho	943935	509702
26	Montana	786690	416402
28	Nevada	800493	682947
31	New Mexico	1302894	939963
37	Oregon	2633105	1788354
44	Utah	1461037	1233060
47	Washington	4132156	3037014
50	Wyoming	469557	294639

Dans l'exemple ci-dessus, la valeur de **pop** est 23667902 pour l'État de Californie, qui est le 5ème dans l'ordre du fichier : cette valeur **pop[5]** peut être remplacée par 23668031 dans la ligne d'édition immédiatement au dessous de la feuille de données.

Le bouton **Delete...** est une alternative à la commande **drop** : elle doit être utilisée avec les mêmes précautions. On peut ainsi éliminer soit une ou plusieurs variables soit des observations. **Delete** permet aussi d'éliminer les observations sur l'ensemble du fichier dont la valeur de la variable courante est égale à celle de l'observation courante : cette manipulation est équivalente à la commande :

```
. drop if region==4
```

L'avantage de l'éditeur sur les commandes **replace** et **drop** est qu'il permet une sauvegarde supplémentaire en cas d'édition répétées. Par exemple, supposez que les corrections suivantes ont été faites :

```
. generate urbain=(popurban/pop)>.75 if popurban!=. & pop!=.
(suivent d'autres commandes créant ou modifiant des variables)

. drop if region==4
```

Si la dernière commande **drop...** est une erreur, et que vous n'aviez pas auparavant sauvegardé votre fichier, il ne vous reste plus qu'à effacer le tout et à recommencer, en perdant toutes les modifications précédentes :

```
. clear

. use census
```

Par contre, si vous utilisez l'éditeur pour chaque correction, même si vous vous trompez, vous tapez sur le bouton **Restore** et vous conservez ainsi les corrections précédentes. En effet, la commande **use** *nomfichier* crée un fichier temporaire qui sera lui-même utilisé par **edit** pour créer un second fichier temporaire. En cliquant sur **Restore**, le second fichier temporaire est effacé et le premier est récupéré.

Une fois les corrections faites, vous avez le choix entre préserver ces corrections (bouton **Preserve**) ou bien restituer le fichier avant les corrections (bouton **Restore**). Dans les deux cas, pour sortir de l'éditeur, cliquez sur le bouton de fermeture de la

fenêtre, marqué d'une croix sous Windows. Notez qu'une liste des changements effectués apparaît dans l'écran des résultats :

```
. edit
- preserve
(suivent d'autres commandes créant ou modifiant des variables)
- drop if region == 4
- restore
```

Le premier message **preserve** indique que l'éditeur fait une copie de sauvegarde avant toute modification du fichier. Le dernier indique si les modifications ont été préservées (**preserve**) ou si le fichier a été restitué (**restore**).

L'éditeur de Stata est mieux adapté aux corrections individuelles, c'est-à-dire observation par observation. Si l'on veut faire une correction globale sur l'ensemble des observations, on lui préférera la commande **replace**. Par exemple, pour convertir tous les chiffres de population en millions avec une précision de deux chiffres après la virgule :

```
. replace pop=round(pop/10^6,.01)
```

Une telle correction n'est pas envisageable avec **edit**.

5.1.5 Les corrections à l'aide d'un fichier *.do

La plupart du temps, surtout lors de l'analyse exploratoire des données, le travail interactif suffit. On conserve simplement les résultats dans un fichier ASCII avec extension *.log (voir *La prise en main du logiciel*).

Mais vous pouvez aussi exécuter, à l'aide de la commande **do** *nomfichier*, une série de commandes contenue dans un fichier « *nomfichier*.do » en format ASCII. Les programmes *.do sont particulièrement utiles lorsqu'il s'agit d'exécuter une longue série de commandes. On distingue deux types de programmes :

- *Les programmes pour la création et la modification de variables*. Dans ce cas, il est préférable de conserver le fichier original dans un répertoire spécifique et d'exécuter le programme sur une copie du fichier dans un répertoire de travail : si des erreurs se sont glissées dans le programme, on pourra toujours récupérer le fichier de données original. Ces programmes pourront avoir la structure suivante :

```
* nom et date de création du programme
use c:\chemin du fichier original\nomfichier.dta
save nomfichier
(suivent d'autres commandes créant ou modifiant des variables)
save nomfichier, replace
```

- *Les programmes pour produire des résultats.* Dans ce cas, il est indispensable de faire précéder les tabulations, graphismes, et calculs statistiques de la commande **`log using`** *`nomfichier`* pour conserver les résultats dans un fichier ASCII. Ces programmes pourront avoir la structure suivante :

```
* nom et date de création du programme
log using nomfichier
use nomfichier
(suivent d'autres commandes de tabulations, graphiques et statistiques)
log close
```

Bien sûr, un même programme peut comporter une partie créant ou modifiant les variables et une autre partie pour les résultats. Considérez le programme suivant :

```
. type essai.do
* essai.do: ceci est un essai de tabulation au 29 avril 1997
use c:\stata\census
save census, replace /* copie le fichier dans le repertoire courant */
/* pour connaitre la repartition des Etats
a dominante urbaine */
generate urbain=(popurban/pop)>.75
log using taburb, replace
summarize popurban /* pop */ urbain
log close
```

Notez que les commentaires peuvent s'insérer librement pourvu qu'ils soient précédés du caractère **`*`** (si le commentaire tient sur une ligne) ou entouré des caractères **`/*`** et **`*/`** (si le commentaire tient sur plusieurs lignes). Dans un fichier programme, le commentaire peut aussi figurer au beau milieu d'une ligne de commande, pourvu qu'il soit entouré des caractères **`/*`** et **`*/`**. On peut aussi utiliser ces caractères pour inscrire une ligne de commandes sur plusieurs lignes :

```
summarize popurban pop urbain /*
      */ if region!=2 & pop<2000000
```

Par ailleurs, l'exécution d'un programme *.do s'interrompt lorsqu'il contient une erreur :

```
. do essai.do

* essai.do: ceci est un essai de tabulation au 29 avril 1997
. use c:\stata\census
(1980 Census data by state)

. save census, replace

. generate urbain=(popurban/pop)>.75

. log using taburb

. sumarize popurban urbain
unrecognized command
r(199);

end of do-file
r(601);
```

Dans ce cas, n'oubliez pas que le programme a ouvert un fichier « taburb.log ». Après l'interruption du programme, ce fichier reste ouvert. Si vous corrigez et relancez votre programme sans avoir fermé le fichier *.log, voilà ce qui arrive :

```
. do essai.do

* essai.do: ceci est un essai de tabulation au 29 avril 1997
. use c:\stata\census
(1980 Census data by state)

. save census, replace

. generate urbain=(popurban/pop)>.75

. log using taburb
log file already open
r(604);
```

Il faut donc fermer le fichier « taburb.log » et ajouter l'option **`replace`** dans la commande **`log`** :

```
. log close

. do essai.do

* essai.do: ceci est un essai de tabulation au 29 avril 1997
. use c:\stata\census
(1980 Census data by state)

. save census, replace

. generate urbain=(popurban/pop)>.75

. log using taburb, replace

. summarize popurban urbain

Variable |     Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
popurban |      50     3328253     4090178    172735   2.16e+07
  urbain |      50          .3      .46291         0          1

. log close
```

5.2 *L'adressage par ligne et la commande* egen

Chaque fois que Stata ouvre un fichier de données, deux variables systèmes sont créées :

- **_n** qui contient le rang courant de chaque observation dans l'ordre du fichier, et,
- **_N** qui contient le rang de la dernière observation.

Elles sont appelées « variables systèmes » parce qu'elles peuvent être utilisées mais ne peuvent pas être modifiées.

Par ailleurs, la valeur d'une variable pour une observation donnée peut être repérée par un numéro entre crochets. Par exemple, la valeur de la variable **region** pour l'observation N°32 est 17558072 :

```
. display region[32]
17558072
```

Il faut en effet se souvenir que Stata charge en mémoire le fichier sous la forme d'une matrice où chaque ligne est une observation et chaque colonne une variable. Les crochets indiquent l'adressage en ligne (*explicit subscripting*), c'est-à-dire le rang de la ligne dans le fichier, alors que le nom de la variable indique la colonne.

5.2.1 L'adressage par ligne d'enregistrement

L'originalité de ce système, c'est qu'il permet de combiner l'adressage par ligne d'observation avec les variables systèmes indiquant le rang courant ou le rang de la dernière observation. Pour expliquer comment fonctionne cette combinaison, nous allons calculer la somme des populations des États figurant dans le fichier « census.dta » et calculer ensuite la part de chaque État dans la population totale.

Pour cela on peut créer une variable que l'on va initialiser à 0 :

```
. generate cumul=0
```

... et calculer ensuite le cumul de la variable **pop** :

```
. replace cumul=pop[_n]+cond(_n==1,0,cumul[_n-1])
```

Observons attentivement la commande précédente :

- **pop[_n]** indique le contenu de la variable **pop** pour l'observation courante **_n** ; remarquez que dans ce cas l'adressage à la ligne courante est implicite : les expressions **pop** et **pop[_n]** sont absolument équivalentes. On aurait tout aussi bien pu écrire :

```
. replace cumul=pop+cond(_n>1,cumul[_n-1],0)
```

- La condition indique que l'on ajoute à **pop[_n]** la valeur du cumul mentionnée sur la ligne précédente **cumul[_n-1]**, seulement dans le cas où l'observation courante n'est pas la première du fichier (**_n>1**). Si l'observation courante est la première (**_n==1**), alors on n'ajoute rien.

On vérifie le résultat du calcul :

```
. list pop cumul

           pop       cumul
  1.   3893888     3893888
  2.    401851     4295739
  3.   2718215     7013954
  4.   2286435     9300389
  5.  23667902    3.30e+07
(suite de la liste)
 48.   1949644    2.21e+08
 49.   4705767    2.25e+08
 50.    469557    2.26e+08
```

Ainsi, la valeur de **cumul** sur l'observation N°3 est égale à la valeur de **cumul** sur l'observation N°2 (4295739) à laquelle on ajoute la valeur de **cumul** pour l'observation courante (2718215).

Cette procédure est, admettons-le, assez complexe, et c'est pourquoi il existe une fonction plus simple **sum(***var***)** pour faire la somme cumulée d'une variable :

```
. generate cumul=sum(pop)
```

Pour calculer la part de la population de l'État dans la population totale, on utilise la valeur de **cumul** pour la dernière observation **_N** :

```
. generate pourcent=100*pop/cumul[_N]

. format pourcent %8.2f

. list pop cumul pourcent

  1.    3893888    3893888      1.72
  2.     401851    4295739      0.18
  3.    2718215    7013954      1.20
  4.    2286435    9300389      1.01
  5.   23667902   3.30e+07     10.48
(suite de la liste)
 48.    1949644   2.21e+08      0.86
 49.    4705767   2.25e+08      2.08
 50.     469557   2.26e+08      0.21
```

L'adressage par ligne est surtout utile pour faire des calculs par groupe d'observations. Pour calculer la part de la population de chaque État dans la population totale de la région, on utilise le préfixe **by** après avoir trié le fichier par région :

```
. sort region

. by region: generate cumulreg=sum(pop)

-> region=Nord Est
-> region=Centre N
-> region=     Sud
-> region=   Ouest

. by region: generate pcentreg=100*pop/cumulreg[_N]

-> region=Nord Est
-> region=Centre N
-> region=     Sud
-> region=   Ouest
```

Exécutez les commandes suivantes pour repérer le plus grand État dans chaque région, et essayez de détecter l'erreur (ou les erreurs) :

```
. sort region pop

. by region: generate grand=pop>pop[_n-1]

. by region: list pop grand
```

Ensuite, exécutez les commandes suivantes, et donnez les significations des indices calculés **ind1** *et* **ind2** *:*

```
. sort region pop

. by region: generate ind1=_n==_N

. by region: generate ind2=_n==1

. by region: list pop ind*
```

5.2.2 La commande `egen`

L'adressage par ligne est utile pour calculer des indices situant une observation parmi l'ensemble des observations du fichier ou parmi un groupe d'observations dans le fichier (ex : pourcentage, rang, etc.). On peut aussi utiliser cet adressage dans les fichiers de séries temporelles, où l'on a souvent besoin de faire référence à la valeur d'une variable pour une année antérieure (valeurs décalées : *lag values*).

Il est parfois utile de reporter sur chaque observation un indice calculé pour un ensemble d'observations (dans un groupe ou pour le fichier tout entier). C'est le cas des moyennes mobiles dans les fichiers de séries temporelles. C'est aussi le cas des fichiers hiérarchisés, de type ménage-individus, ou bien mère-enfants : chaque observation peut être rattachée à un niveau hiérarchique plus élevé, et on a parfois besoin de calculer des indices au niveau hiérarchique supérieur (ex : nombre d'individus actifs par ménage, revenu moyen, etc.).

Tout cela nécessite de longs programmes de calculs. La commande **egen** peut dans ce cas vous simplifier la vie. Sa syntaxe prend deux formes :

egen *var = fonction(listevar)*
[**if** *exp*] [**in** *intervalle*]

Dans ce cas, **egen** calcule une nouvelle variable en fonction d'une liste de variables pour une même observation. Ainsi, à partir des valeurs contenues dans *listevar*, la fonction **rsum** calcule la somme totale (ex : dépense totale à partir des postes budgétaires), tandis que la fonction **rmean** calcule la moyenne (ex : moyenne de notes à différents examens).

Pour les séries temporelles, **egen** permet de calculer la moyenne mobile à l'aide d'une syntaxe particulière :

egen *var* = **ma**(*exp*)
[**if** *exp*] [**in** *intervalle*] [, **t(#) nomiss**]

Par exemple, pour calculer la moyenne mobile de la variable **var** sur 5 ans, on aura :

```
. egen moymob=ma(var), t(5)
```

L'option **nomiss** permet de forcer le calcul d'une moyenne non centrée en début ou en fin de série.

Mais **egen** est surtout utile pour calculer des indices sur un ensemble d'observations :

egen *var* = *fonction(exp)*
[**if** *exp*] [**in** *intervalle*] [, **by(***listevar***)**]

Parmi les nombreuses *fonctions* possibles de cette commande (pour plus de détails, voir la référence **egen** du manuel de Stata), nous allons expliquer celles qui permettent le mieux de comprendre l'utilité de **egen**. Tout d'abord, nous retrouvons une fonction qui ressemble à la fonction **sum()** que nous avions utilisée avec **generate** mais qui donne un résultat différent :

```
. sort region

. by region: generate cumulreg=sum(pop)

-> region=Nord Est
-> region=Centre N
-> region=     Sud
-> region=   Ouest

. egen sommereg=sum(pop), by(region)

-> region=Nord Est
-> region=Centre N
-> region=     Sud
-> region=   Ouest

. by region: list state pop cumulreg sommereg

-> region=Nord Est
             state         pop   cumulreg   sommereg
  1.         Maine     1124660    1124660   4.91e+07
  2.   Connecticut     3107576    4232236   4.91e+07
  3.  Pennsylvania    11863895   1.61e+07   4.91e+07
  4. New Hampshire      920610   1.70e+07   4.91e+07
  5.    New Jersey     7364823   2.44e+07   4.91e+07
  6. Massachusetts     5737037   3.01e+07   4.91e+07
  7.  Rhode Island      947154   3.11e+07   4.91e+07
  8.       Vermont      511456   3.16e+07   4.91e+07
  9.      New York    17558072   4.91e+07   4.91e+07

-> region=Centre N
             state         pop   cumulreg   sommereg
 10.      Illinois    11426518   1.14e+07   5.89e+07
 11.      Missouri     4916686   1.63e+07   5.89e+07
 12.       Indiana     5490224   2.18e+07   5.89e+07
(suite de la liste)
```

On remarque qu'avec **egen** le préfixe **by()** est renvoyé en option, et que la somme cumulée est reportée sur l'ensemble des

observations. Dans ce cas, pour calculer la part de chaque État, la formule sera différente pour **cumulreg** :

```
. by region: generate prcentreg=100*pop/cumulreg[_N]
```

... et pour **sommereg** :

```
. generate prcentreg=100*pop/sommereg
```

La commande **egen** permet des calculs plus élaborés tels que des moyennes :

```
. egen moyreg=mean(pop), by(region)
```

... des écarts types (*standard deviation*) :

```
. egen etreg=st(pop), by(region)
```

... ou des pourcentiles (médiane ou autre) :

```
. egen medreg=pctile(pop), p(75) by(region)
```

Dans ce dernier cas, l'option p(#) indique le pourcentile désiré (la médiane **p(50)** est la valeur par défaut).

D'autres fonctions existent pour la commande **egen**. *Exécutez les commandes suivantes :*

```
. sort region
. egen ind1=max(pop), by(region)
. replace ind1=pop==ind1
. egen ind2=min(pop), by(region)
. replace ind2=pop==ind2
```

Que signifient les variables **ind1** *et* **ind2** *?*
Comparez avec la procédure utilisée dans l'exercice précédant avec la commande **generate**.

Peut-être avez-vous remarqué que nous n'avons jamais spécifié l'option **[** *type***]** *des commandes* **generate** *et* **egen** *(voir le chapitre sur* Les fichiers de données*). Par défaut, le type de la variable sera* float. *Pour économiser de la place en mémoire après avoir créé les variables, il suffit d'exécuter la commande :*

```
. compress
```

---

6 LES STATISTIQUES SIMPLES ET LEUR REPRÉSENTATION GRAPHIQUE

Vous avez franchi avec succès les premières étapes de l'initiation à Stata : prise en main du logiciel, constitution d'un fichier, et création et correction de variables. Maintenant, les réjouissances peuvent commencer : vous êtes en mesure de vous amuser... pardon, d'explorer sérieusement les données.

Avant de mener des analyses à l'aide de modèle de régression et autres statistiques complexes, il est préférable de tirer le maximum de l'exploration des données et de statistiques simples. Cela a deux avantages :

- permettre de mieux connaître les données et donc de repérer leurs particularités et leurs éventuelles incohérences, ce qui pourra servir pour des analyses statistiques plus approfondies ;

- permettre de sélectionner des indices et des graphiques simples qui rendent le mieux compte des données afin de les restituer à un large public : les connaissances en statistique de la plupart des mortels ne dépassent guère le pourcentage, et de toute façon, même un public de spécialistes ne retiendra en définitive que les indices et les graphiques les plus simples.

Stata offre de nombreuses commandes pour l'analyse exploratoire des données, autant sous forme de tableaux que de graphiques. Comme dans les chapitres précédents, nous utiliserons le fichier « census.dta » pour illustrer ces commandes.

6.1 Les tableaux

La forme des tableaux dépend du nombre de variables croisées. Lorsqu'une seule variable est décrite, on parle de distribution univariée, ou de fréquence simple. Pour deux variables, on parle de distribution bivariée, et, au delà de deux variables, de distribution multivariée. Plus le nombre de variables est élevé, plus la lecture sera complexe : un tableau croisant plus de quatre variables est souvent incompréhensible pour une personne normalement constituée (y compris un statisticien).

6.1.1 Résumé des variables et fréquence simple

La première chose à faire avant de travailler sur un fichier est d'examiner l'ensemble des variables, pour détecter d'éventuelles erreurs à la saisie ou lors du transfert des données, et surtout pour se familiariser avec les données.

La commande **codebook** *listvar*, grâce à une programmation savante, donne une description adaptée aux variables alphanumériques et numériques. Parmi les variables numériques, elle distingue automatiquement les variables continues et discrètes. Pour être considérée comme discrète, une variable doit avoir au maximum 9 catégories distinctes : l'option **tabulate(#)** permet de changer cette valeur par défaut.

En insérant les résultats dans un fichier « *.log », on peut ainsi créer un document de référence où figureront une description à la fois des libellés et du contenu de chaque variable du fichier :

```
. log using descrip

. use census
(1980 Census data by state)

. codebook

state ------------------------------------------------------------------------ State
                  type:  string (str13)

        unique values:  50                       coded missing:  0 / 50

             examples:  "Georgia"
                        "Maryland"
                        "Nevada"
                        "S. Carolina"

              warning:  variable has embedded blanks
```

```
region ------------------------------------------------------------- Census region
                  type:  numeric (byte)
                 label:  cenreg

                 range:  [1,4]                        units:  1
         unique values:  4                    coded missing:  0 / 50

            tabulation:  Freq.   Numeric  Label
                             9         1  Nord Est
                            12         2  Centre N
                            16         3  Sud
                            13         4  Ouest

pop ---------------------------------------------------------------- Population
                  type:  numeric (long)

                 range:  [401851,23667902]            units:  1
         unique values:  50                   coded missing:  0 / 50
                  mean:   4.5e+06
             std. dev:   4.7e+06

           percentiles:        10%       25%       50%       75%       90%
                            671743   1.1e+06   3.1e+06   5.5e+06   1.1e+07
```

(suite de la liste)

```
. log close
```

Dans « census.dta », la variable **state** est alphanumérique et des espaces sont incrustés dans les intitulés (**embedded blanks**) comme par exemple entre **S.** et **Carolina**. La variable **region** est considérée comme discrète (moins de 9 valeurs distinctes : **unique values: 4**). Un tableau sommaire montre la fréquence (**Freq.**) pour chacune des quatre valeurs que peut prendre cette variable (**range: [1,4]**), ainsi que le libellé (**Label**) de chacune de ces valeurs. De plus, le nombre de valeurs manquantes parmi la totalité des observations est indiquée (**coded missing : 0 / 50**).

La variable **pop** est considérée comme continue parce qu'elle contient plus de 9 valeurs distinctes (**unique values: 50**) qui vont de 401 851 à 23 667 902, par unité de compte de 1 (**units: 1**). Dans ce cas, une description sommaire de la distribution de la variable est donnée par la moyenne (**mean**) et l'écart type (**std. dev**) et par les pourcentiles (la médiane est de 3 100 000 habitants, et 10 % des États ont plus de 11 000 000 d'habitants).

La commande **summarize** *listvar* permet aussi de résumer la distribution, en particulier pour les variables numériques continues. Cela n'aurait pas grand sens, par exemple, de calculer la moyenne d'une variable discrète comme **region** :

```
. summarize region

Variable |      Obs        Mean    Std. Dev.       Min        Max
---------+--------------------------------------------------------
  region |       50        2.66    1.061574         1          4
```

L'option **detail** permet une description plus précise des variables continues, incluant les pourcentiles, les quatre plus grandes (**Largest**) et plus basses (**Smallest**) valeurs, ainsi qu'un indice de dissymétrie (la valeur de **Skewness** est 0 pour la distribution normale) et de concentration (la valeur de **Kurtosis** est de 3 pour la distribution normale) :

```
. summarize pop, detail

                        Population
-------------------------------------------------------------
      Percentiles      Smallest
 1%       401851          401851
 5%       511456          469557
10%     671742.5          511456       Obs                  50
25%      1124660          594338       Sum of Wgt.          50

50%      3066433                       Mean            4518149
                        Largest       Std. Dev.       4715038
75%      5463105        1.19e+07
90%     1.11e+07        1.42e+07       Variance        2.22e+13
95%     1.42e+07        1.76e+07       Skewness        2.073804
99%     2.37e+07        2.37e+07       Kurtosis        7.729186
```

Ces statistiques montrent une distribution de **pop** nettement dissymétrique sur la droite (la médiane est inférieure à la moyenne comme l'indique aussi un indice de 2,07 pour **Skewness**) et très concentrée (**Kurtosis** de 7,73) sur quelques valeurs.

À l'inverse de la commande **summarize**, la commande **tabulate** est utile pour les variables discrètes.

```
. tabulate region

Census
region      |      Freq.     Percent        Cum.
------------+-----------------------------------
   Nord Est |          9       18.00       18.00
   Centre N |         12       24.00       42.00
        Sud |         16       32.00       74.00
      Ouest |         13       26.00      100.00
------------+-----------------------------------
      Total |         50      100.00
```

On remarque avec l'option **nolabel** (pour afficher les codes plutôt que les libellés), que les régions sont classées selon leur numéro de code :

```
. tabulate region, nolabel

Census
region |      Freq.     Percent        Cum.
------------+-----------------------------------
         1 |          9       18.00       18.00
         2 |         12       24.00       42.00
         3 |         16       32.00       74.00
         4 |         13       26.00      100.00
------------+-----------------------------------
     Total |         50      100.00
```

Dans le cas d'une variable discrète, le pourcentage cumulé n'a pas grand intérêt puisque l'ordre de classement des catégories de la variable est arbitraire. Le cumul n'a d'intérêt que pour les variables continues ou pour les variables comportant des catégories ordonnées, c'est-à-dire classées selon une échelle ordinale (par exemple : du meilleur au pire, du plus au moins satisfaisant, etc.).

La commande **tab1** *listvar* est une variation de **tabulate** pour établir les fréquences simples. Elle s'utilise comme **codebook** pour créer un document de référence sur la distribution de chacune des variables. Cependant, elle ne distingue pas de la même manière les variables alphanumériques, continues et discrètes : **codebook** est mieux adaptée à cet exercice.

Pour vous permettre de mieux comprendre l'utilisation des commandes décrites ci-dessus, répondez aux questions suivantes :

- *Pourquoi d'après* **codebook** *l'unité de compte de la variable* **medage** *est* **.1** *(***units: .1***) ?*
- *Observez le résultat de* **codebook** *sur la variable* **sommereg** *(voir chapitre précédent sur les créations de variables) : pourquoi cette variable continue (population régionale) est-elle présentée comme une variable discrète ?*
- *Exécutez la commande* **summarize state** *: pourquoi le nombre d'observations est-il égal à 0 ?*
- *Exécutez la commande* **tabulate pop** *: quel intérêt présentent le résultat ? Et le résultat de la commande* **tabulate medage** *? (Comparez la colonne* **Cum.** *avec les résultats de la commande* **summarize***)*

6.1.2 Tableaux croisés à deux variables

La commande **tabulate** devient vraiment intéressante pour croiser les distributions de deux variables discrètes. La syntaxe de base de cette commande est :

tabulate *varligne varcol*
[, *cell column row missing nofreq wrap nolabel all chi2 exact gamma lrchi2 taub V*]

Prenons pour exemple, la distribution des 50 États des USA par région selon qu'au moins trois quarts de leur population est urbaine ou non (voir chapitre sur *Les fichiers de données* pour la création de la variable **urbain**) :

```
. tabulate urbain region

Pop urbaine|  Census region
sup a 75%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+--------------------------------------------+----------
       Non |         4         11         13          7 |        35
       Oui |         5          1          3          6 |        15
-----------+--------------------------------------------+----------
     Total |         9         12         16         13 |        50
```

Les modalités de la première variable citée figurent en ligne, tandis que les modalités de la deuxième apparaissent en colonne. Des options permettent d'obtenir les pourcentages en ligne (**row**) , en colonne (**column**) ou par cellule (**cell**) du tableau :

```
. tabulate urbain region, column

Pop urbaine|  Census region
sup a 75%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+--------------------------------------------+----------
       Non |         4         11         13          7 |        35
           |     44.44      91.67      81.25      53.85 |     70.00
-----------+--------------------------------------------+----------
       Oui |         5          1          3          6 |        15
           |     55.56       8.33      18.75      46.15 |     30.00
-----------+--------------------------------------------+----------
     Total |         9         12         16         13 |        50
           |    100.00     100.00     100.00     100.00 |    100.00
```

Pour afficher les pourcentages sans les fréquences, on utilisera l'option **nofreq** :

```
. tabulate urbain region, row nofreq

Pop urbaine| Census region
sup a 75%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+--------------------------------------------+----------
       Non |     11.43      31.43      37.14      20.00 |    100.00
       Oui |     33.33       6.67      20.00      40.00 |    100.00
-----------+--------------------------------------------+----------
     Total |     18.00      24.00      32.00      26.00 |    100.00
```

Pour illustrer l'utilité de l'option **wrap**, nous allons d'abord créer une variable polytomique indiquant pour chaque région, les États considérés comme urbains ou ruraux, en combinant les quatre modalités de la variable **region** au deux modalités de la variable **urbain** :

```
. generate regurb=region*10+urbain

. label define regurb 10 "NE-rur" 11 "NE-urb" 20 "CN-rur" 21 "CN-urb" 30 "Sud-
rur" 31 "Sud-urb" 40 "Ouest-ru" 41 "Ouest-ur"

. label values regurb regurb

. label var regurb "Region par milieu"
```

La variable **regurb** contient 4x2 modalités. Si l'on veut croiser la variable **regurb** en colonne et **actif** en ligne (voir chapitre sur *Les fichiers de données* pour la création de la variable **actif**), on obtient un tableau scindé en deux :

```
. tabulate actif regurb

Pop active | regurb
sup a 60%  |    NE-rur     NE-urb     CN-rur     CN-urb    Sud-rur |     Total
-----------+---------------------------------------------------------+----------
       Non |         1          0          9          0          7 |        23
       Oui |         3          5          2          1          6 |        27
-----------+---------------------------------------------------------+----------
     Total |         4          5         11          1         13 |        50


Pop active | regurb
sup a 60%  |   Sud-urb   Ouest-ru   Ouest-ur |     Total
-----------+--------------------------------+----------
       Non |         1          3          2 |        23
       Oui |         2          4          4 |        27
-----------+--------------------------------+----------
     Total |         3          7          6 |        50
```

La commande **tabulate** prévoit par défaut un maximum de 5 modalités en colonnes, ce qui explique que le tableau soit scindé en deux afin de tenir sur les 80 caractères que comporte une ligne sur l'écran. Notez que la colonne **Total** fait référence au total général et non au total de chaque partie du tableau. Avec l'option **wrap**, le tableau n'est pas scindé en deux car aucune limite n'est imposée au nombre de colonnes :

```
. tab actif regurb, wrap

Pop active |  regurb
sup a 60%  |     NE-rur      NE-urb      CN-rur      CN-urb      Sud-rur      Sud-urb
>  Ouest-ru   Ouest-ur |      Total
-----------+----------------------------------------------------------------------
> -----------------------+----------
       Non |           1           0           9           0           7           1
>          3           2 |          23
       Oui |           3           5           2           1           6           2
>          4           4 |          27
-----------+----------------------------------------------------------------------
> -----------------------+----------
     Total |           4           5          11           1          13           3
>          7           6 |          50
```

L'option **wrap** semble donc parfaitement inutile pour une représentation à l'écran, mais si les résultats doivent être conservés dans un fichier « *.log », un tableau de plus de 5 colonnes pourra être imprimé sur une feuille de plus de 80 caractères (par exemple en format « paysage » plutôt que « portrait »).

Il peut arriver qu'une variable ne soit pas renseignée pour certaines catégories d'observations : dans ce cas, seules les observations pour lesquelles les variables croisées sont renseignées figureront dans le tableau. Pour que la totalité des observations (y compris celles dont les valeurs sont manquantes) figurent dans le tableau, on utilisera l'option **missing**. Dans ce cas une modalité intitulée « **.** » sera ajoutée aux colonnes ou aux lignes déjà présentes.

Diverses statistiques permettent d'évaluer le degré d'association entre les deux variables croisées dans le tableau :

```
. tabulate urbain region, all exact

Pop urbaine| Census region
sup a 75%  |  Nord Est   Centre N         Sud       Ouest |     Total
-----------+--------------------------------------------+----------
       Non |          4          11          13           7 |        35
       Oui |          5           1           3           6 |        15
-----------+--------------------------------------------+----------
     Total |          9          12          16          13 |        50

          Pearson chi2(3) =   8.0612   Pr = 0.045
 likelihood-ratio chi2(3) =   8.4497   Pr = 0.038
               Cramer's V =   0.4015
                    gamma =   0.0508  ASE = 0.255
          Kendall's tau-b =   0.0301  ASE = 0.151
           Fisher's exact =                 0.048
```

Chacune des statistiques produite par l'option **all** peut être obtenue par des options spécifiques : la statistique du Chi^2 selon le calcul standard (option **chi2**) ou selon le ratio de vraisemblance (option **lrchi2**) , ainsi que le **V** de Cramér qui varie de -1 à +1 dans le cas d'un tableau de 2 lignes sur 2 colonnes (2x2), et de 0 à 1 dans les autres cas. L'option **exact** calcule le test exact de Fisher, qui est basé sur la distribution de toutes les combinaisons possibles des cellules du tableau à partir de ses marges : ce calcul peut prendre un temps relativement long si le nombre de catégories et d'observations est grand et si l'ordinateur ne dispose pas d'un co-processeur rapide.

Ces statistiques sont valables pour toutes les variables polytomiques. Dans notre exemple, le caractère urbain de l'État semble bien être corrélé avec son appartenance à une des quatre régions : les valeurs des tests de Chi^2 sont inférieures à 5 %, ce que confirme le test de Fisher.

D'autres statistiques évaluent l'association entre deux variables ordinales : le Gamma de Goodman & Kruskal (option **gamma**) basé sur le nombre de paires d'observations concordantes et discordantes du point de vue des deux variables croisées, et le Tau de Kendall (option **taub**) qui s'interprète comme le gamma mais qui tient mieux compte du nombre de paires de même configuration. Dans notre exemple, ces statistiques n'ont aucun sens car la variable **region** n'est pas ordinale : la région 3 ne vaut pas plus que la région 2.

La commande **tab2** *listvar* est une variation de **tabulate** pour établir les tableaux croisés deux à deux pour toutes les combinaisons de variables de *listvar*. Elle s'utilise comme **codebook** pour créer un document de référence sur l'association entre les variables dicho- ou polytomiques du fichier de données. Il faut évidemment tenir compte du nombre important de tableaux croisés qui peut résulter de **tab2** : si trois variables sont croisées deux à deux, on n'obtient que 3 tableaux, mais si dix variables sont croisées deux à deux, on obtient 45 tableaux (voir *[U] 20.3.1 Mathematical functions*) :

```
. display comb(10,2)
45
```

Pour vous permettre de mieux comprendre l'utilisation de la commande **tabulate**, *exécutez les commandes suivantes :*

```
. generate caturb=recode(popurban/pop,.5,.6,.7,.8)

. tabulate actif caturb, all exact
```

La variable, calculée à l'aide de la fonction **recode()**, *distribue les États selon que le pourcentage de leur population urbaine est inférieur à 50 %, entre 50 et 60 %, etc. (voir* [U] 20 Functions and expressions*). Répondez aux questions suivantes :*

- *La variable* **caturb** *est-elle une variable continue ?*
- *Les valeurs du Gamma et du Tau ont-elles un sens ?*

6.1.3 Tableaux croisés à trois variables ou plus

Le préfixe **by** *listvar* est utilisé pour produire des tableaux croisés à deux variables pour chaque combinaison des modalités des variables énumérées dans *listvar*. Si une seule variable est énumérée dans *listvar*, on obtient un tableau croisé à trois variables, si deux variables sont énumérées, un tableau croisé à quatre variables, etc. Le préfixe **by** nécessite un tri préalable selon les variables énumérées :

```
. sort actif

. by urbain: tabulate actif region, all exact
-> urbain=      Non
Pop active | Census region
sup a 60%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+---------------------------------------------+----------
       Non |         1          9          7          3 |        20
       Oui |         3          2          6          4 |        15
-----------+---------------------------------------------+----------
     Total |         4         11         13          7 |        35

          Pearson chi2(3) =   5.0634   Pr = 0.167
 likelihood-ratio chi2(3) =   5.3683   Pr = 0.147
           Fisher's exact =                 0.164

-> urbain=      Oui
Pop active | Census region
sup a 60%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+---------------------------------------------+----------
       Non |         0          0          1          2 |         3
       Oui |         5          1          2          4 |        12
-----------+---------------------------------------------+----------
     Total |         5          1          3          6 |        15

          Pearson chi2(3) =   2.5000   Pr = 0.475
 likelihood-ratio chi2(3) =        .
           Fisher's exact =                 0.505
```

Les tests statistiques sont calculés pour chaque tableau croisé à deux variables. Ils concordent pour dire que parmi les États à dominante rurale, le caractère actif de l'État semble être corrélé avec son appartenance à une des quatre régions, quoique le niveau de significativité du test soit très faible : le test exact de Fisher est significatif à 16 % seulement.

Parmi les États à dominante urbaine, les tests ne sont pas significatifs. Le test du ratio de vraisemblance (**lrchi2**) n'est pas calculé car toutes les cases du tableau doivent comporter au moins une unité, ce qui n'est pas le cas ici. Mais il faut immédiatement ajouter que, de toute façon, aucun de ces tests ne permet de mesurer convenablement une association qui semble au premier abord très forte : tous les États des régions Nord Est et Centre Nord sont « actifs » alors qu'un tiers des États du Sud et de l'Ouest ne le sont pas. C'est un paradoxe des tests statistiques : ils ne permettent pas de mesurer par un degré de *significativité* une différence *signifiante* entre une proportion de 0 % et une proportion de 67 %, car aucun chiffre ne peut multiplier 0 % pour donner 67 %. On peut voir avec cet exemple, que même pour l'interprétation statistique, il est nécessaire de passer par une évaluation qualitative des données.

L'option **summarize(***var***)** est une alternative au préfixe **by** pour obtenir le croisement de trois variables, ou plutôt le résumé d'une troisième variable dans les cases d'un tableau croisant deux variables. Par exemple, si l'on veut obtenir la population moyenne par États dans chacune des régions selon leur caractère urbain ou rural, on exécutera :

```
. tabulate urbain region, summarize(pop) nostandard

                Means and Frequencies of Population

Pop urbaine|  Census region
sup a 75%  |   Nord Est    Centre N        Sud       Ouest |     Total
-----------+----------------------------------------------+----------
       Non |  3605155.3   4312650.2   3580118.4   1524312.6 |  3402042.9
           |          4          11          13           7 |         35
-----------+----------------------------------------------+----------
       Oui |  6942932.4     11426518   9397496.7   5417050.3 |  7122398.1
           |          5           1           3           6 |         15
-----------+----------------------------------------------+----------
     Total |  5459475.9   4905472.5   4670876.8   3320960.8 |  4518149.4
           |          9          12          16          13 |         50
```

On a ajouté l'option **nostandard** pour éviter le calcul des écarts types (*standard deviation*). Pour afficher seulement les

populations moyennes (sans les effectifs), on peut ajouter l'option **nofreq**.

L'option **summarize()** n'est pas seulement utile pour les variables continues, telles que les effectifs de population, les revenus, etc. Elle est aussi utile pour résumer les variables ordinales (voir exercice précédent) et aussi les variables dichotomiques. En effet, il faut savoir que *la moyenne d'une variable dichotomique est une proportion.* Par exemple, pour connaître la proportion d'États à dominante active selon la région et le caractère dominant urbain ou rural, on exécutera :

```
. format actif %4.2f

. tabulate urbain region, summarize(actif) nostandard nofreq

                    Means of Pop active sup a 60%

Pop urbaine| Census region
sup a 75%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+--------------------------------------------+----------
       Non |      0.75       0.18       0.46       0.57 |      0.43
       Oui |      1.00       1.00       0.67       0.67 |      0.80
-----------+--------------------------------------------+----------
     Total |      0.89       0.25       0.50       0.62 |      0.54
```

On changé le format d'affichage de la variable **actif** pour obtenir des proportions arrondies à deux chiffres après la virgule. Notez que l'option **summarize()** ne permet pas le calcul des tests statistiques réservés aux tableaux croisés à deux variables.

On peut utiliser le préfixe **by** pour résumer une variable dans un tableau croisant trois autres variables ou plus :

```
. by urbain: tabulate actif region, summarize(medage) nostandard nofreq

-> urbain=      Non
                        Means of Median age

Pop active | Census region
sup a 60%  |  Nord Est   Centre N        Sud      Ouest |     Total
-----------+--------------------------------------------+----------
       Non |      30.40      29.52      29.23      28.00 |      29.24
       Oui |      30.53      29.35      29.35      28.30 |      29.31
-----------+--------------------------------------------+----------
     Total |      30.50      29.49      29.28      28.17 |      29.27
```

```
-> urbain=      Oui

                        Means of Median age

Pop active | Census region
sup a 60%  |  Nord Est   Centre N        Sud       Ouest |      Total
-----------+----------------------------------------------+----------
       Non |         .          .      34.70       26.70 |      29.37
       Oui |     31.82      29.90      29.25       29.28 |      30.38
-----------+----------------------------------------------+----------
     Total |     31.82      29.90      31.07       28.42 |      30.18
```

Notez que la moyenne des âges médians **medage** n'est pas calculée pour les États à dominante urbaine et non active des régions du Nord Est et du Centre Nord : aucun État ne présente cette configuration-là comme nous l'avions déjà vu plus haut à propos des tests de significativité.

Exécutez les commandes suivantes :

```
. generate pcturb=popurban/pop

. tabulate urbain actif, summarize(region)

. tabulate actif region, summarize(pcturb)
```

Que signifie le résumé de la variable **region** *? Et le résumé de la variable* **pcturb** *? Quel est le sens du deuxième chiffre dans chaque case du tableau ?*

6.1.4 Présentation complexe des résumés de variables

Vous trouvez certainement les résultats de la commande **tabulate ..., summarize()** peu présentables, surtout avec le préfixe **by**. Vous avez raison, et c'est pourquoi Stata a prévu la commande **table** pour résumer diverses statistiques dans un seul tableau, même si ce tableau croise plus de deux variables. On peut inscrire dans ce tableau non seulement les effectifs, les moyennes et les écarts types, mais aussi d'autres statistiques plus complexes comme les pourcentiles, les valeurs minimum ou maximum, ou l'intervalle inter-quartile.

La syntaxe de base de la commande **table** est :

table *varligne [varcol [varsupercol]]*
,contents(*stat* [*listevar*] [*stat* [*listevar*]...])
[by(*varsuperligne***) row col scol format(***%fmt***)**
center left missing cellwidth(#) csepwidth(#)
scsepwidth(#) stubwidth(#)]

Comme on le voit, la syntaxe est assez complexe (et encore, nous n'avons mentionné que les options les plus utiles). Il est plus facile de la comprendre à l'aide d'exemples. Ainsi, la commande **tabulate** :

```
. by urbain: tabulate actif region, summarize(medage) nostandard nofreq
```
(voir tableau dans la section précédente)

... qui calcule la moyenne de l'âge médian **medage** des populations de chaque État selon le caractère urbain ou actif et l'appartenance à une région, a pour équivalent :

```
. table actif region, contents(mean medage) by(urbain) row col format(%5.2f)
stubwidth(25)

-------------------------+----------------------------------------------------
Pop urbaine sup a 75% and |                    Census region
Pop active sup a 60%      | Nord Est  Centre N       Sud     Ouest      Total
--------------------------+----------------------------------------------------
Non                       |
                      Non |    30.40      29.52     29.23     28.00      29.24
                      Oui |    30.53      29.35     29.35     28.30      29.31
                          |
                    Total |    30.50      29.49     29.28     28.17      29.27
--------------------------+----------------------------------------------------
Oui                       |
                      Non |                          34.70     26.70      29.37
                      Oui |    31.82      29.90     29.25     29.27      30.38
                          |
                    Total |    31.82      29.90     31.07     28.42      30.18
--------------------------+----------------------------------------------------
```

L'intitulé des lignes indique que la variable **urbain** a été placée en « super-ligne » par l'option **by()**, au dessus de la variable **actif** en ligne simple. Ainsi le **Non** et le **Oui** sur la marge gauche correspondent à la variable **urbain**, tandis que le **Non** et le **Oui** au dessus du **Total** correspondent à la variable **actif**. On a utilisé l'option **stubwidth(#)** pour donner le maximum de place à l'intitulé des lignes, soit 25 caractères.

Notez l'option **contents()** qui spécifie la statistique qui doit figurer dans le tableau (**mean**) et la variable sur laquelle doit

porter le calcul (**medage**). Les statistiques disponibles pour l'option **contents()** de la commande **table** sont :

- **freq** : fréquences pondérées (= **n** *var* si les données ne sont pas pondérées)
- **mean** *var* : moyenne de *var*
- **sd** *var* : écart type de *var*
- **sum** *var* : somme cumulée pondérée de *var*
- **rawsum** *var* : somme cumulée non pondérée de *var*
- **n** *var* : nombre d'observations pour lesquelles *var* est non manquante (= **count** *var*)
- **max** *var* : maximum de *var*
- **min** *var* : minimum de *var*
- **median** *var* : médiane de *var* (= **p50** *var*)
- **p#** *var* : #ème pourcentile de *var* (ex : **p34** *var*, 34ème pourcentile de *var*)
- **iqr** *var* : intervalle interquartile de *var*

Les totaux pour la variable en ligne (*varligne*) sont obtenus par l'option **row** et les totaux pour la variable en colonne (*varcol*) par l'option **col**. Par ailleurs, l'option **format()** permet de définir le format d'affichage des statistiques, soit deux chiffres après la décimale dans notre exemple.

On peut obtenir une présentation plus compacte en inversant les lignes et les colonnes, si la variable **urbain** intervient en tant que « super-colonne » (*varsupercol*) :

```
. table region actif urbain, contents(mean medage) row col center format(%5.2f)
cellw(6)

----------+-------------------------------------------------
          |  Pop urbaine sup a 75% and Pop active sup a 60%
Census    | --------- Non --------    --------- Oui --------
region    |   Non      Oui     Total      Non      Oui     Total
----------+-------------------------------------------------
 Nord Est |  30.40    30.53    30.50               31.82   31.82
 Centre N |  29.52    29.35    29.49               29.90   29.90
      Sud |  29.23    29.35    29.28     34.70     29.25   31.07
    Ouest |  28.00    28.30    28.17     26.70     29.27   28.42
          |
    Total |  29.24    29.31    29.27     29.37     30.38   30.18
----------+-------------------------------------------------
```

Le tableau est en fait composé de deux petits tableaux croisant la variable **region** (en ligne) et la variable **actif** (en colonne) : le premier tableau (les trois premières colonnes) correspond à la modalité **Non** de la variable **urbain**, tandis que le second (les trois dernières colonnes) correspond à la modalité **Oui**.

Si l'on veut ajouter les totaux selon la variable **actif** pour chacune des régions, on ajoutera l'option **scol** (super-colonne) :

```
. table reg act urb, contents(mean medage) row col center format(%5.2f) scol
```

Un maximum de trois variables peut être spécifié par **table**, et un maximum de quatre variables par l'option **by()**. Au total, on peut donc obtenir un tableau croisant au maximum sept variables, sans compter les différents résumés statistiques figurant à l'intérieur du tableau. Par exemple, on peut résumer non seulement la moyenne de la variable **medage** mais aussi le nombre d'États concernés et l'intervalle interquartile comme mesure de dispersion :

```
. table region actif urbain, contents(n medage mean medage iqr medage) row col
center format(%5.2f) cellw(6)
```

Complétez les commandes suivantes avec les options adéquates de manière à obtenir une présentation à la fois concise et lisible :

```
. table actif region urbain, contents(median pcturb) ...
. table urbain actif region, contents(median pcturb) ...
```

Dans ces tableaux, que signifie le résumé de la variable **pcturb** *? Peut-on obtenir le même résultat avec* **tabulate..., sum()** *?*

Comment peut-on obtenir avec **table** *un tableau indiquant la proportion d'États à dominante active selon la région et le caractère dominant urbain ou rural ?*

6.2 *Les pondérations et les extrapolations*

Dans cette section, on distingue :

- les *extrapolations* à partir des données d'enquêtes, qui font l'objet des commandes **svy**, et
- les *pondérations* qui peuvent être appliquées à n'importe quel type de données, y compris les données d'enquêtes, grâce à l'option `[xweight]` .

Pour évaluer la précision des estimations des moyennes, des proportions, des ratios, des totaux et des coefficients de régression, on doit tenir compte du plan de sondage (*sampling design*). Ainsi, avant d'exécuter tout autre commande sur les données d'une enquête, il est fortement recommandé de mesurer la précision des estimations, avec les commandes **svy**. S'il n'y a pas de plan de sondage (si les données ne proviennent pas d'une enquête par sondage), les pondérations simples avec l'option `[xweight]` suffisent.

6.2.1 L'option [*xweight*]

Toutes les commandes n'autorisent pas les pondérations, et tous les types de pondérations ne sont pas forcément autorisés. Pour ne pas alourdir l'exposé, nous n'avons pas mentionné les options de pondérations pour chaque commande : le lecteur se référera directement au manuel de Stata qui précise toujours dans la description de chaque commande quels types de pondération sont autorisés. Dans ce manuel, nous nous en tiendrons à un descriptif simplifié (que nous avons appelé « de base ») pour chaque commande.

Les pondérations sont introduites dans les commandes sous la forme d'un suffixe entre crochets, généralement après la liste de variables :

```
commande listevar [xweight=var] ...
```

Quatre types de pondérations sont autorisés dans Stata :

- **[fweight=***var***]** (*frequency weight*) : chaque observation dans le fichier représente plusieurs unités d'analyse ayant les mêmes caractéristiques, et en conséquence, *var* doit contenir des nombres entiers (si ce n'est pas le cas, Stata renvoie un message d'erreur). Le total des pondérations est supérieur au nombre d'observations dans le fichier et représente la population totale auprès de laquelle les données ont été recueillies.

- **[aweight=***var***]** (*analytic weight*) : chaque observation dans le fichier est pondérée relativement aux autres. Le total des pondérations est toujours égal au nombre d'observations dans le fichier : l'échelle de *var* n'est pas pertinente, seul compte le poids relatif d'une observation par rapport aux autres.

- **[pweight=***var***]** (*probability or sampling weight*) : chaque observation est pondérée par l'inverse de la probabilité de sondage : à la différence de **fweight**, *var* peut contenir des nombres décimaux.

- **[iweight=***var***]** (*importance weight*) : ce type de pondération diffère de **aweight** seulement parce qu'il tient compte de l'échelle de *var*, et de **fweight** parce qu'il n'est pas restreint aux nombres entiers.

On peut se demander, à la lecture de ce qui précède, à quoi sert **fweight** si **pweight** autorise les nombres décimaux, et à quoi sert de distinguer **iweight** de **pweight** alors que tous les deux tiennent compte de l'échelle des pondérations et qu'ils ne sont pas restreints aux nombres entiers.

Pour bien comprendre la différence entre **pweight** et les autres types de pondération, il faut se rappeler que pour un échantillon, la pondération **pweight** représente le poids recherché pour l'extrapolation et non pas le poids réel dans la population. Les données ont été recueillies auprès d'un échantillon (qu'on espère représentatif) et seulement sur cet échantillon : à chaque mesure est associée une marge d'erreur qui dépend du plan de sondage.

Au contraire avec les pondérations **fweight** et **iweight**, chaque observation est répliquée : plus la pondération est importante plus la précision est grande parce qu'on suppose alors que la mesure a été faite sur un nombre plus grand d'observations. C'est le raisonnement inverse pour la pondération **pweight**, où la précision est moins bonne lorsque la pondération est importante.

Dans le cas de la pondération **aweight**, le nombre d'observation n'est pas extrapolé. La différence avec une estimation sans pondération est que chaque observation n'a pas le même poids relativement aux autres observations. La précision est calculée en fonction du poids relatif des observations les unes par rapport aux autres.

Prenons le cas du calcul de la moyenne de **medage** dans les données du Recensement des USA. Pour obtenir une moyenne qui tienne compte du poids de chaque État, on utilisera la variable **pop**. Mais il ne s'agit pas de données provenant d'un échantillon : on n'a pas enquêté un individu par État. Chaque observation dans le fichier représente en fait l'ensemble des individus d'un État, pour lesquels on a calculé l'âge médian. Dans ce cas, l'utilisation de **pweight** est exclue.

Les données ne représentent pas cependant des individus répliqués : l'âge varie d'un individu à l'autre au sein d'un même État. La variable **medage** est un indice de valeur centrale calculé sur la distribution des âges des individus de chaque État. En conséquence, on ne doit pas non plus utiliser **fweight** (ou **iweight**) pour le calcul de la moyenne de **medage**.

Il ne reste plus que **aweight**. Ce type de pondération tient compte du fait que chaque observation est le résumé d'une distribution particulière. On sait que la distribution des âges est différente d'un État à l'autre, mais on ne connaît pas l'allure de ces distributions. Dans le jargon statistique, on parle d'*hétéroscédasticité* quand la variance n'est pas la même d'une observation à l'autre. Dans ce cas, **aweight** fait l'hypothèse que le résumé des distributions est plus précis quand il est calculé sur une population plus importante : le calcul de la variance tient compte du poids de chaque observation.

Mais voyons le résultat de ces différentes pondérations sur la moyenne de **medage**. Tout d'abord, les résultats sans pondération sont les suivants :

```
. table urbain region , contents(freq mean medage sd medage) col format(%5.2f)

----------+-------------------------------------------------
Pop       |
urbaine   |                  Census region
sup a 75% | Nord Est  Centre N       Sud     Ouest     Total
----------+-------------------------------------------------
      Non |        4         11         13         7        35
          |    30.50      29.49      29.28     28.17     29.27
          |     1.15       0.72       1.03      1.52      1.22
          |
      Oui |        5          1          3         6        15
          |    31.82      29.90      31.07     28.42     30.18
          |     0.38                  3.32      2.18      2.40
----------+-------------------------------------------------
```

Ces résultats non pondérés présentent le nombre d'États (**freq**), la moyenne des âges médians (**mean medage**) et l'écart type de ces âges (**sd medage**). Chaque observation dans le fichier vaut pour elle-même, comme si elle ne représentait qu'un individu. Tous les États ont le même poids, quel que soit l'effectif réel de leur population. On remarque que l'écart type ne peut être calculé pour le seul État à dominante urbaine de la région Centre Nord.

Si l'on pondère selon le poids de chaque État avec l'option **fweight**, on obtient :

```
. table urbain region [fw=pop], contents(freq mean medage sd medage) col
format(%5.2f)

----------+-------------------------------------------------
Pop       |
urbaine   |                  Census region
sup a 75% | Nord Est  Centre N       Sud     Ouest     Total
----------+-------------------------------------------------
      Non | 1.44e+07  4.74e+07  4.65e+07  1.07e+07  1.19e+08
          |    31.74      29.57      29.22     29.09     29.65
          |     0.78       0.64       0.94      1.25      1.16
          |
      Oui | 3.47e+07  1.14e+07  2.82e+07  3.25e+07  1.07e+08
          |    31.85      29.90      30.76     29.43     30.62
          |     0.31       0.00       2.95      1.23      1.94
----------+-------------------------------------------------
```

La somme de la population des États figure dans chaque case du tableau (soit par exemple un total de 14,4 millions d'habitants dans les quatre États à dominante rurale de la région Nord Est). Les moyennes et les écarts types sont très différents de ce qu'on obtenait sans pondération, car le poids de chaque État dépend de l'effectif de sa population.

Pour obtenir des écarts types qui tiennent compte de la précision de l'estimateur de la moyenne, on préférera l'option **aweight** qui tient compte du poids de la population pour calculer non seulement la moyenne mais aussi l'écart type :

```
. table urbain region [aw=pop], contents(freq mean medage sd medage) col
format(%5.2f)

----------+--------------------------------------------------
Pop       |
urbaine   |                    Census region
sup a 75% | Nord Est  Centre N       Sud     Ouest     Total
----------+--------------------------------------------------
      Non |     3.19      10.50     10.30      2.36     26.35
          |    31.74      29.57     29.22     29.09     29.65
          |     0.94       0.67      0.99      1.64      1.19
          |
      Oui |     7.68       2.53      6.24      7.19     23.65
          |    31.85      29.90     30.76     29.43     30.62
          |     0.34       0.00      3.22      1.32      1.98
----------+--------------------------------------------------
```

Les fréquences ne représentent plus la population totale, mais le poids relatif des groupes d'États parmi les 50 États qui forment le fichier. La moyenne est la même qu'avec l'option **fweight** : on a bien tenu compte du poids relatif de chaque État. Mais les estimations des écarts types différent sensiblement : elles sont toujours plus élevées surtout quand la catégorie concernée comporte de petits États. L'écart type a ainsi augmenté de plus de 20 % dans les États à dominante rurale de la région Nord Est, contre moins de 10 % dans les États plus peuplés à dominante urbaine de la même région. Le phénomène est encore plus accentué dans la région Ouest où les sept États à dominante rurale sont très petits (1,5 millions en moyenne, contre 5,4 millions dans les États à dominante urbaine de la même région).

Exécutez les commandes suivantes (notez la pondération) :

```
. gen pop2=pop+.5
. table region [fw=pop2], contents(freq mean medage) format(%5.2f)
. table region [fw=pop], contents(freq mean medage) format(%5.2f)
. table region [pw=pop], contents(freq mean medage) format(%5.2f)
. table region [pw=pop2], contents(freq mean medage) format(%5.2f)
```

*Comment expliquer les différences ou l'absence de différences entre les résultats ? Peut-on dans tous les cas calculer l'écart type (**sd medage**) ? (voir section suivante)*

6.2.2 Les commandes `svy` pour les données d'enquêtes

Nous ne traiterons ici que des commandes **`svy`** pour l'estimation des moyennes, proportions, ratios et sommes, car elles complètent utilement les commandes **`tabulate`** et **`table`**. Les mêmes principes valent pour les autres commandes **`svy`** que nous ne traiterons pas ici et qui estiment les coefficients de régression pour des données d'enquêtes, en tenant compte du plan de sondage : **`svyreg`** (régressions simples), **`svylogit`** (régressions logit) et **`svyprobt`** (régressions probit).

L'option **`pweight`** ne peut tenir compte des subtilités du sondage : elle pondère simplement les observations par l'inverse de la probabilité de sondage. Les commandes **`svy`** font plus que cela : elles tiennent compte de la stratification et du sondage à plusieurs degrés des grappes (*multi-stage cluster sampling*).

Supposons que les données de « census.dta » ne soient pas issues d'un recensement, mais d'une (mauvaise) enquête, où quelques individus auraient été tirés dans deux unités primaires par région. Pour calculer la moyenne des âges (c'est-à-dire la moyenne de **`medage`** interprétée ici comme l'âge de chaque individu), on pourra définir les régions comme des strates. Pour chacune de ces strates, on supposera qu'on a tiré au premier degré des individus dans une unité primaire d'un État à dominante urbaine et dans une unité primaire d'un État à dominante rurale. Dans chaque unité primaire, on aurait ensuite tiré aléatoirement au second degré un nombre variable d'individus (de 1 à 13).

Quel que soit le nombre de degrés de sondage, la variance des estimations est basée uniquement sur le premier degré, et autorise toute sorte de corrélation (ou effets de grappe) aux degrés inférieurs. Ainsi pour ses calculs, Stata n'a besoin que des variables définissant les strates et les unités primaires de sondage (*primary sampling units*) c'est-à-dire les grappes au premier degré de sondage, en plus, bien entendu, de l'inverse de la probabilité de sondage (c'est-à-dire **`pweight`**). On peut ensuite calculer la moyenne :

```
. svyset strata region

. svyset psu urbain

. svyset pweight pop

. svymean medage

Survey mean estimation

pweight:  pop                                   Number of obs    =        50
Strata:   region                                Number of strata =         4
PSU:      urbain                                Number of PSUs   =         8
                                                Population size  = 2.259e+08

------------------------------------------------------------------------------
    Mean |   Estimate    Std. Err.   [95% Conf. Interval]        Deff
---------+--------------------------------------------------------------------
  medage |   30.11047    .3317936    29.18927    31.03168    1.975249
------------------------------------------------------------------------------
```

On ne doit pas confondre l'erreur type (*standard error* : *Std. Err.*) de l'estimation par sondage qui figure dans ce tableau, avec l'écart type (*standard deviation* : *Std. Dev.*) de la distribution de la variable **medage** fourni par les commandes **table** ou **summarize**.

On aurait pu tout aussi bien définir les strates, les unités primaires et les pondérations à l'aide d'options :

```
. svymean medage [pw=pop], strata(region) psu(urbain)
```

L'avantage de les définir à l'avance par la commande **svyset** est que ces paramètres sont conservés en mémoire lors de la prochaine sauvegarde du fichier :

```
. save, replace
```

Par ailleurs, si tous les individus avaient été enquêtés dans chaque unité primaire, on aurait affaire à un sondage à un seul degré (*one stage sampling*). Dans le cas d'un tirage à un degré sans remise, on doit introduire un facteur de correction (**fpc** : *finite population correction*) en indiquant la variable qui comporte dans chaque strate le nombre d'unités primaires parmi lesquelles on a tiré les unités primaires échantillons. Par exemple, en supposant que toutes les unités primaires ont la même taille (égale au nombre d'individus enquêtés dans chaque strate), on aura :

```
. egen nbpsu=mean(pop) , by(region)

. svyset fpc nbpsu
```

```
. svymean medage

Survey mean estimation

pweight:  pop                                   Number of obs    =        50
Strata:   region                                Number of strata =         4
PSU:      urbain                                Number of PSUs   =         8
FPC:      nbpsu                                 Population size  = 2.259e+08

--------------------------------------------------------------------------------
    Mean |   Estimate    Std. Err.   [95% Conf. Interval]         Deff
---------+----------------------------------------------------------------------
  medage |   30.11047    .3317935    29.18927    31.03168     1.975249
--------------------------------------------------------------------------------
Finite population correction (FPC) assumes simple random sampling without
replacement of PSUs within each stratum with no subsampling within PSUs.
```

La correction n'a pratiquement aucun effet parce que le nombre de personnes enquêtées dans notre exemple est infime par rapport au nombre de personnes dans chaque strate : dans ce cas extrême, un tirage avec remise a pratiquement le même résultat qu'un tirage sans remise (*without replacement*). Dans le monde réel, cependant, la différence de précision peut être très importante : sans remise, le tirage fournit des estimations plus précises.

Mais ce type de correction ne doit jamais être utilisé pour les tirages à plusieurs degré (*multistage sampling*), même sans remise. Il n'est autorisé que pour les tirages à un degré comme l'indique le commentaire en note du tableau (*with no subsampling within PSUs*), ce qui en fait limite son usage étant donné que la plupart des sondages se font à plusieurs degrés.

Exécutez les commandes suivantes pour obtenir la moyenne par région :

```
. table region [pw=pop], contents(freq mean medage)

. svymean medage, by(region)
```

Comparez les résultats avec et sans stratification par région :

```
. svymean medage

. svyset strate, clear

. svymean medage

. svymean medage, by(region)
```

Comment expliquez-vous les différences d'intervalles de confiance ?

6.2.3 Récapitulatif sur les pondérations

Le choix du type de pondération dépend du type de données dont on dispose, et des statistiques que l'on veut obtenir. Pour vous aider à choisir la commande adéquate, reportez-vous au tableau page suivante qui fait la synthèse des sections précédentes.

On remarque que la commande **table** calcule les pourcentiles et les intervalles inter-quartiles alors que l'option **sum()** de la commande **tabulate** ne peut fournir que la moyenne et l'écart type.

Par contre, la commande **table** ne peut fournir directement une distribution à partir d'une variable polytomique comme **tabulate**. Il faut d'abord créer une série de variables dichotomiques avant d'en faire le résumé par **contents()**. Par exemple, dans une enquête, si **age** désigne le groupe d'âges en cinq catégories, on pourra obtenir la distribution par groupe d'âges selon le milieu et la région en créant d'abord une variable dichotomique pour chaque âge (voir le chapitre *Création et correction de variables*) :

```
. tabulate age, generate(age_)

. table urbain region [pw=pop], contents(mean age_1 mean age_2 mean age_3 mean
age_4 mean age_5)
```

Par ailleurs, l'estimation des indices de dispersion (écart type, intervalle inter-quartile...) et leur degré de précision (erreur type, intervalle de confiance) ne peuvent être obtenus à l'aide des procédures **svy** pour les données d'enquêtes. Il ne s'agit pas d'une insuffisance du logiciel : seules des procédures particulières (de type *bootstrapping*) peuvent fournir de tels indices car il n'y a pas de solution mathématique pour mesurer leur précision. L'explication de ces procédures nous mènerait bien au-delà du champ couvert par ce manuel. Le lecteur intéressé se reportera à la référence **[R] bstrap** dans le manuel de Stata.

Type de statistique	Exemples de statistiques	Données non échantillonnées	Données d'enquêtes par sondage
Distribution absolue (décompte des individus)	- Population totale - Distribution par groupes d'âges - Population par milieu	**tabulate [f/a/iw]** **table [f/a/iw]**	**svytotal** **tabulate [fw]** **table [f/i/pw]**
Distribution relative (proportions)	- Proportions par groupes d'âges - Proportions par milieu - Pourcentiles des revenus	**tabulate [f/a/iw]** *(construire variables dichotomiques :)* **table [f/a/iw]**	**svyprop** *(construire variables dichotomiques :)* **table [pw]**
Indice de valeur centrale	- Revenu moyen - Âge médian - Part de la population urbaine	*** tabulate [f/a/iw]** **table [f/a/iw]**	**svymean** **table [pw]**
Indice de dispersion	- Écart type de l'âge - Intervalle inter-quartile des revenus	*** tabulate [f/a/iw]** **table [f/a/iw]**	**table [f/iw]**
Somme	- Revenu total - Population urbaine - Beurre consommé	**table [f/a/iw]**	**svytotal** **table [pw]**
Ratio	- Part du revenu du capital sur le revenu total - Rapport entre le cholestérol et le beurre consommé	*(pas de procédure spécifique)*	**svyratio**

* Pour obtenir des indices de valeur centrale et de dispersion avec **tabulate**, utilisez l'option **sum()**.

6.3 *Les graphiques*

Tout comme les tableaux, les graphiques peuvent être univariés, bivariés ou multivariés. Mais ils ont l'avantage sur les tableaux qu'ils peuvent rendre compte plus facilement des variables continues, alors que les tableaux se limitent généralement aux variables discrètes (ou ordonnées) pour un nombre réduit de catégories.

Pour ce qui concerne les options communes à tous les types de graphiques, la syntaxe de base de la commande **graph** est :

graph [*listevar*]
[, *options spécifiques au type de graphique*
by(nomvar) *total*
x/y/r/tlabel x/y/r/ttick x/y/r/tline
x/y/rscale y/x/rlog
symbol(*s..s*) *connect*(*c..c*)
saving(*nomfichier*, [*replace*])]

Les options communes concernent essentiellement la mise en forme du graphique : libellés (*label*), graduations (*tick*), lignes (*line*), échelle des axes (*scale, log*), symboles (*symbol*), liaison des points (*connect*). Nous n'avons fait figurer ici que les options les plus courantes. Pour une description des options avancées pour les titres, la taille des caractères, l'épaisseur des traits et les couleurs, voir les exemples qui sont donnés dans le manuel de référence aux entrées **graph titles**, **graph textsize**, **graph pens**, **graph shading**.

Les options spécifiques au type de graphique sont décrites dans les sections suivantes, ainsi que les commandes spécifiques qui s'apparentent à la commande **graph**.

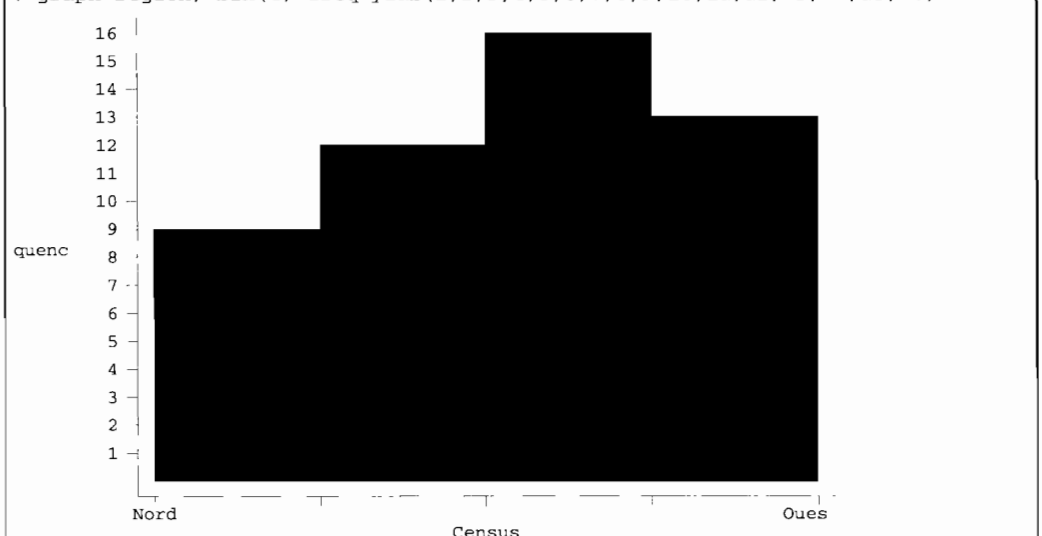
6.3.1 Graphiques de répartition discrète : histogrammes, barres, camemberts et étoiles

La commande **graph** produit par défaut un histogramme lorsqu'une seule variable figure dans la liste des variables. Certaines options sont spécifiques aux histogrammes :

graph *var*
[, **bin(#) freq Rescale by(***listevar***)**]

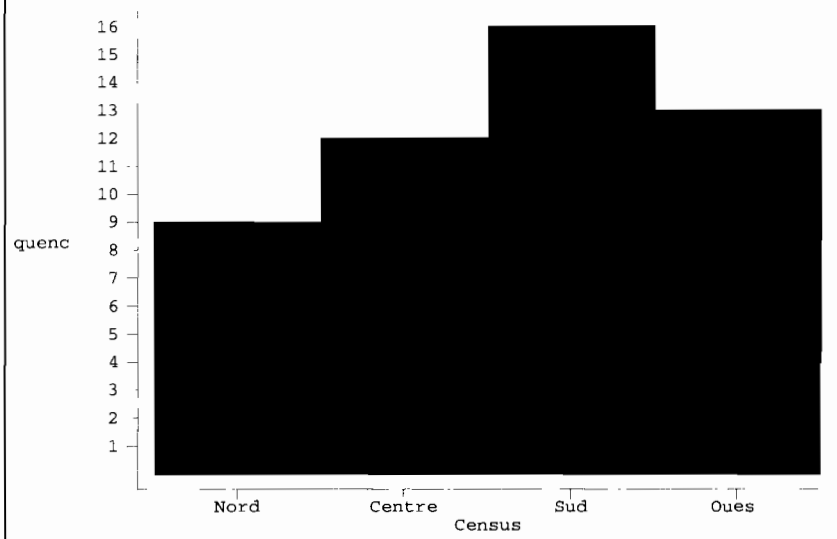
Le nombre d'intervalles par défaut est 5, et peut être changé pour une valeur maximum de 50. L'échelle des ordonnées est définie par défaut en pourcentage et l'option **freq** permet d'indiquer plutôt la fréquence. Ainsi, pour représenter le nombre d'États par région des USA, on aura :

```
. graph region, bin(4) freq ylab(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
```



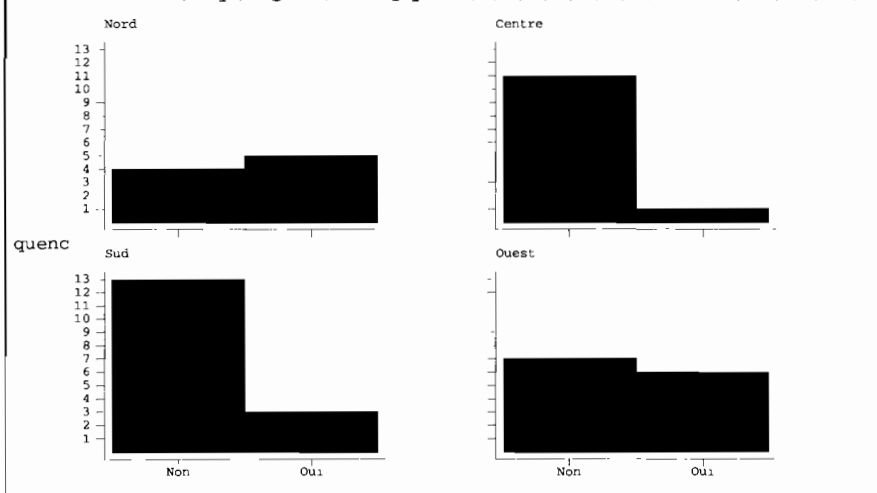
Un tel graphique n'est pas d'un très grand intérêt, surtout que l'on ne voit pas apparaître le nom de toutes les régions. Une alternative plus esthétique à la commande **graph** pour les histogrammes de variable discrète est la commande **hist**. Sa syntaxe est la même à la différence près que l'option **bin()** est inutile parce qu'automatiquement définie par le nombre de catégories de la variable décrite (remarquez les libellés de la variable **region** qui apparaissent sur l'axe des abscisses) :

```
. hist region, freq ylab(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
```



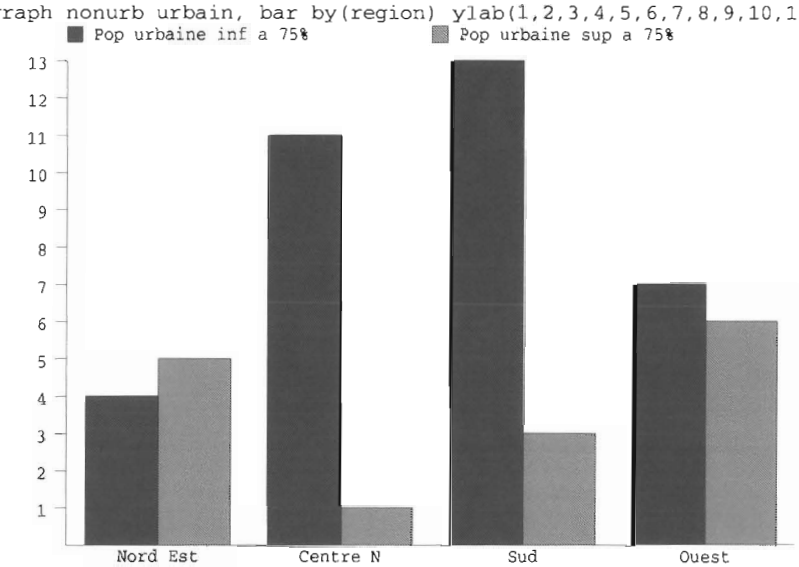
Le graphique a sans doute plus d'intérêt si l'on croise deux variables discrètes avec l'option **by()** après avoir trié le fichier selon la variable spécifiée (voir le chapitre *Création et correction de variables* pour la création de la variable **urbain**) :

```
. sort region

. hist urbain, by(region) freq ylab(1,2,3,4,5,6,7,8,9,10,11,12,13)
```



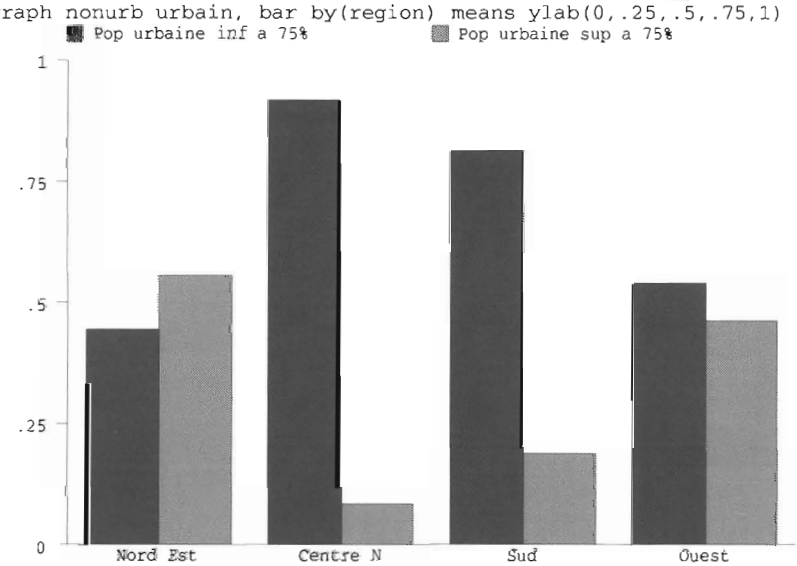
Le graphique aura une meilleure allure si l'on utilise l'option **bar** après avoir créé une variable pour chaque modalité de la variable dépendante :

```
. gen nonurb=urbain==0

. lab var nonurb "Pop urbaine inf a 75%"

. graph nonurb urbain, bar by(region) ylab(1,2,3,4,5,6,7,8,9,10,11,12,13)
```



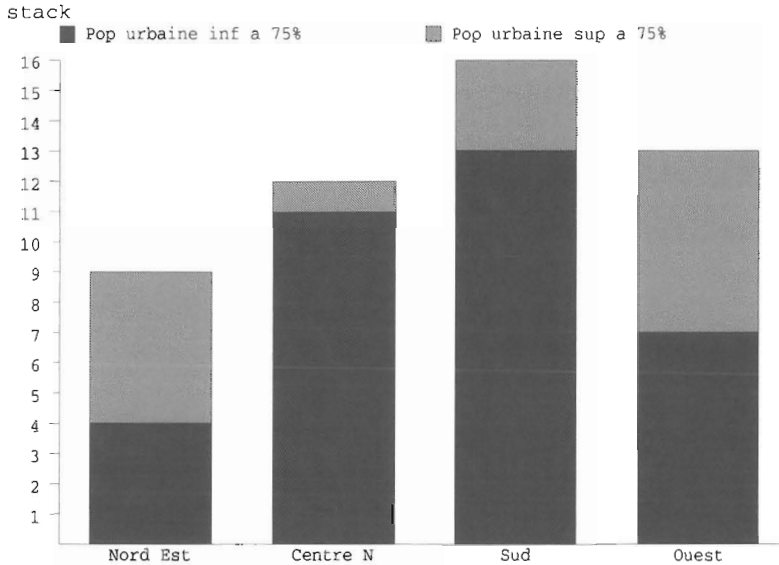
On peut ainsi spécifier jusqu'à six barres au maximum. La hauteur des barres peut être calculée en proportion pour chaque catégorie de la variable de contrôle avec l'option **means** :

```
. graph nonurb urbain, bar by(region) means ylab(0,.25,.5,.75,1)
```



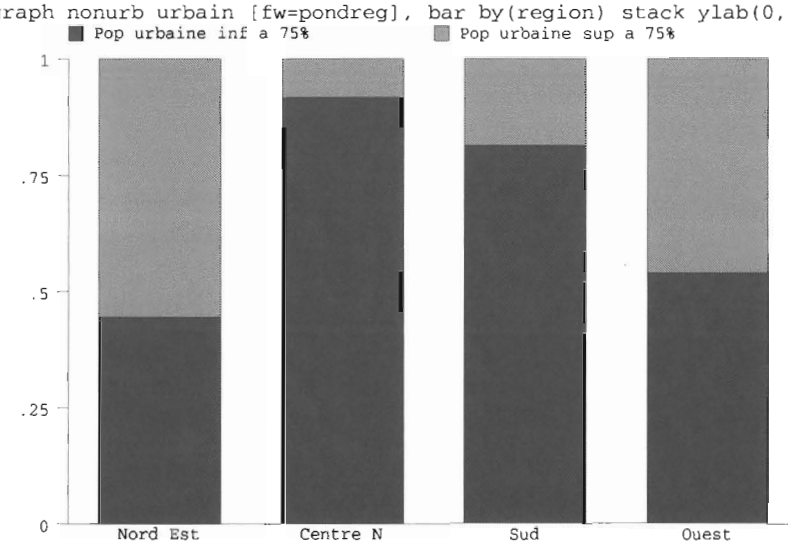
Pour représenter chacune des catégories empilées les unes sur les autres, on utilise l'option **stack** :

```
. graph nonurb urbain, by(region) ylab(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16)
bar stack
```



L'option **means** ne peut être combinée avec **stack** pour obtenir une distribution pourcentuelle. Cependant l'inconvénient peut être contourné en pondérant chacune des observations par l'inverse du poids de la catégorie de la variable de contrôle à laquelle elles appartiennent (voir chapitre *Création et correction de variable* pour plus de précision sur la commande **egen**) :

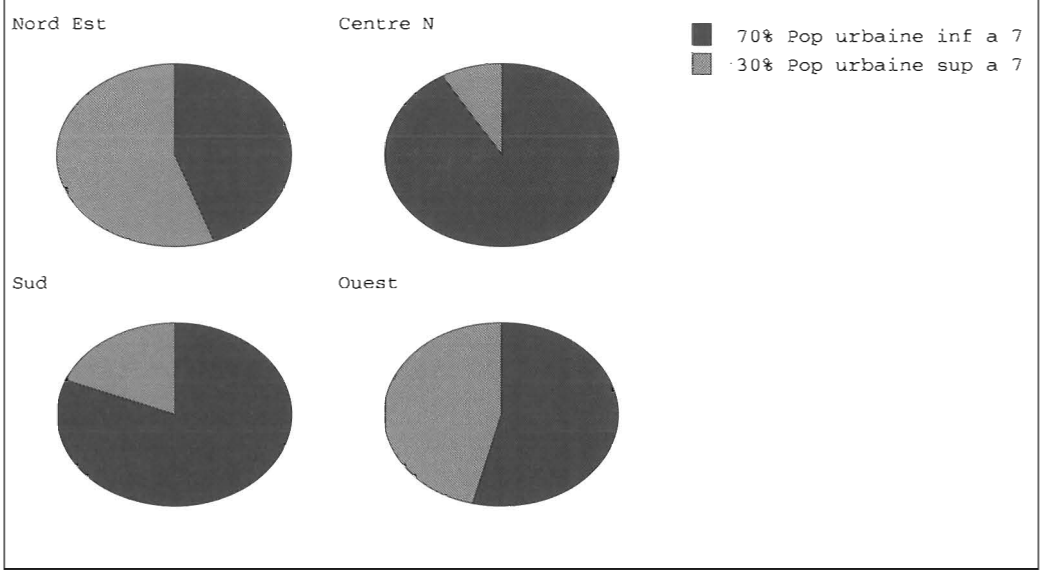
```
. egen pondreg=count(region), by(region)

. replace pondreg=1/pondreg

. graph nonurb urbain [fw=pondreg], bar by(region) stack ylab(0,.25,.5,.75,1)
```



De cette façon, on attribue un poids de 1/9 aux 9 États du Nord Est, de 1/12 aux 12 États de Centre Nord, etc.

La répartition relative (ou pourcentuelle) est obtenue plus simplement avec les fameux camemberts (*pie chart*) à l'aide de l'option **pie** :

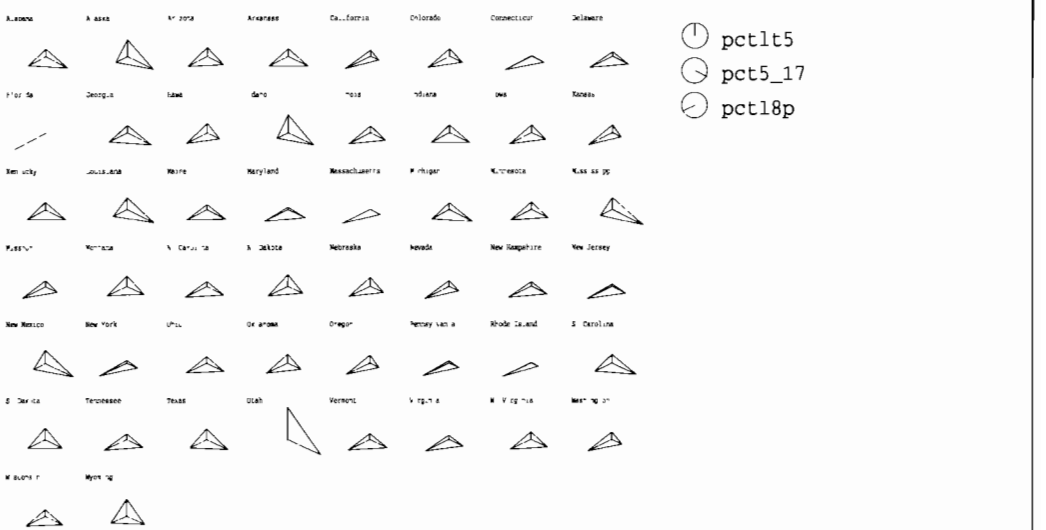
```
. graph nonurb urbain, pie by(region)
```



Remarquez que le graphique indique la répartition dans chaque catégorie **nonurb** et **urbain** pour l'ensemble des États, mais pas pour chaque région. Les barres sont plus informatives car on peut plus facilement voir le pourcentage de chaque catégorie sur l'échelle des ordonnées. En outre, on ne peut tracer des camemberts proportionnellement à la taille de chaque catégorie de la variable de contrôle (par exemple **region**).

Un autre type de graphique, rarement utilisé, est utile pour repérer les observations extrêmes ou aberrantes (celles qui s'éloignent le plus du comportement moyen). Le graphique en étoile (*star chart*) décrit pour chaque observation une série de variables ou de modalités d'une même variable. Par exemple, pour repérer les États qui ont une distribution par groupe d'âges particulière, on exécutera :

```
. gen pctlt5=poplt5/pop

. gen pct5_17=pop5_17/pop

. gen pct18p=pop18p/pop

. graph pctlt5 pct5_17 pct18p, star label(state)
```



Le centre de chaque étoile représente la valeur minimum de chaque catégorie, et la longueur de chaque trait est proportionnelle à la valeur relative de chaque catégorie. L'orientation de chaque trait correspond à une catégorie, tel qu'indiqué en haut à droite du graphique.

Ce graphique est, il est vrai, difficilement lisible étant donné le nombre d'observations. Pour faire des graphiques séparés pour chaque région, et mieux voir ainsi les États concernés, on doit trier le fichier selon la région et utiliser l'option **`select(#,#)`** :

```
. sort region

. graph pctlt5 pct5_17 pct18p, star label(state) select(1,9)
(graphique omis)
. graph pctlt5 pct5_17 pct18p, star label(state) select(10,21)
(graphique omis)
. graph pctlt5 pct5_17 pct18p, star label(state) select(22,37)
(graphique omis)
. graph pctlt5 pct5_17 pct18p, star label(state) select(38,50)
(graphique omis)
```

On peut voir ainsi que les États de l'Utah et de Floride ont vraiment une structure très différente de la moyenne, au contraire de l'État du Wyoming (dont les branches de l'étoile sont de longueur presque égale).

Exécutez les commandes suivantes pour comprendre l'option **Rescale** *de la commande* **hist** *:*

```
. hist urbain, by(region)
. hist urbain, by(region) Rescale
. hist urbain, by(region) freq Rescale
```

Quelle est la différence d'échelle entre les graphiques ? Pourquoi le deuxième et le troisième graphiques ont-ils la même allure, bien que l'échelle ne soit pas exprimée dans la même unité ?

Calculez à l'aide de **egen** *une variable de pondération pour les deux catégories d'États à dominante active et non active. Faites ensuite un graphique en barres empilées de la répartition pourcentuelle selon le caractère urbain des États et un autre graphique selon leur appartenance à l'une ou l'autre des régions.*

Faites des graphiques en barres empilées de la répartition absolue de la population par groupe d'âges pour chaque État regroupés par région. Faites les mêmes graphiques pour la répartition relative et comparez-les avec les graphiques en étoiles correspondant. Pouvez-vous discerner de la même façon les États aux structures extrêmes ?

Après avoir trié le fichier par région, comparez les deux graphiques :

```
. graph pctlt5 pct5_17 pct18p, star label(state) select(1,9)
. graph pctlt5 pct5_17 pct18p if region==1, star label(state)
```

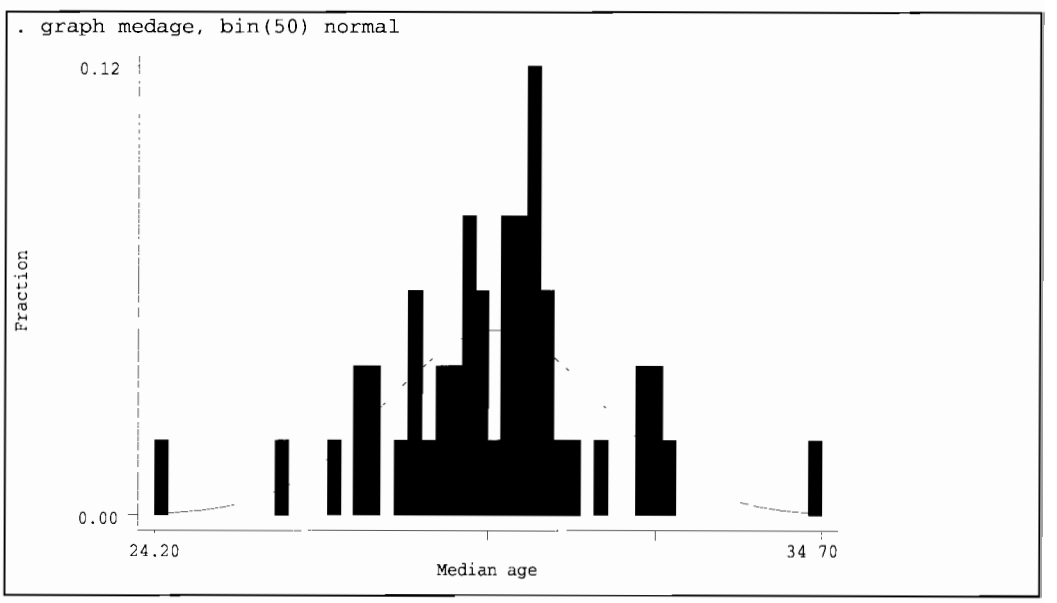
Pourquoi les résultats sont-ils différents ? Que représente le point minimum (le centre de l'étoile) dans chacun des cas ?

6.3.2 Graphiques de distribution univariée et ajustement des courbes

Il faut bien admettre que la distribution d'une variable discrète est assez peu informative, et que, dans ce cas, un simple tableau de fréquence suffit. En revanche, pour les variables continues ou ordonnées, plusieurs types de graphiques sont possibles avec la commande **graph**.

6.3.2.1 Les histogrammes et les boîtes

Les histogrammes sont obtenus de la même façon qu'avec les variables discrètes, mais on peut y ajouter l'option **normal**. Ainsi, pour représenter la distribution de l'âge médian dans chaque État, on aura :



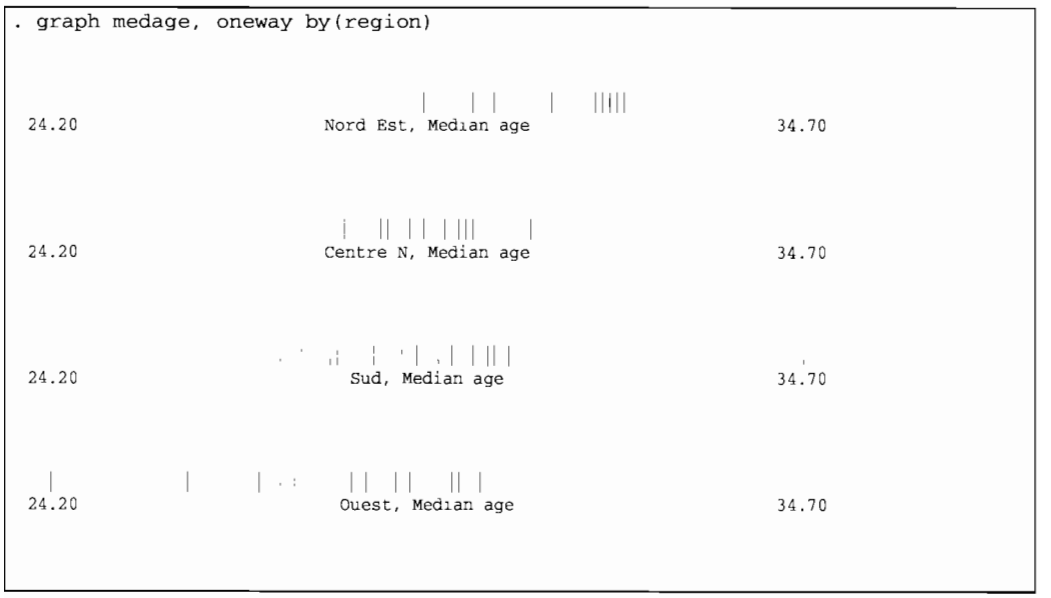
L'option **normal** permet de confronter la distribution empirique avec la distribution normale de même moyenne et de même écart type. D'après le graphique, il semble que la distribution de l'âge médian suive très approximativement une loi normale.

D'autres types de graphiques permettent de bien voir la distribution d'une variable continue :

graph *var*
[, **oneway jitter(#) box Rescale by(***listevar***)**]

On peut, par exemple, comparer la distribution de l'âge médian selon la région :

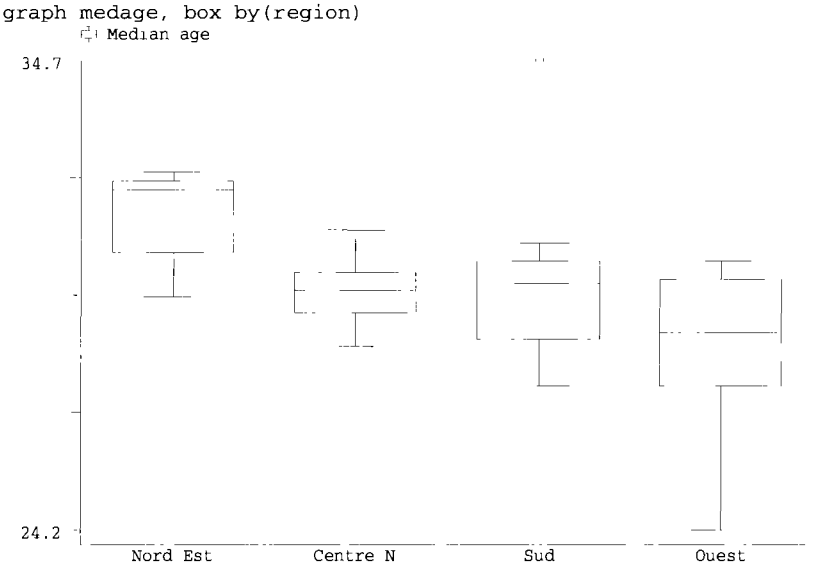
```
. graph medage, oneway by(region)
```



L'option **oneway** représente chaque observation (ici, un État) par un trait le long de l'axe des abscisses représentant l'âge médian.

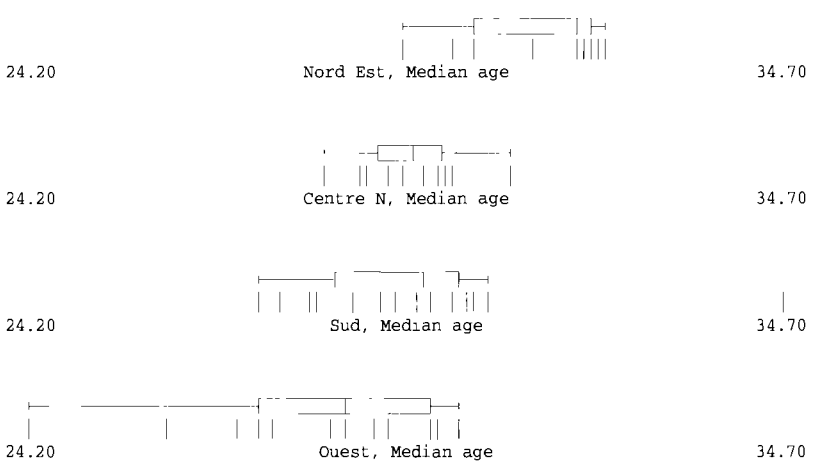
L'option **box** crée une « boîte » qui représente l'intervalle interquartile (voir la commande **summarize** expliquée dans la section précédante), dans laquelle un trait figure la médiane. Les « moustaches » (*wiskers*) de chaque côté de la boîte représentent l'extension de la distribution jusqu'aux observations de chaque côté de la boîte qui se situent à plus ou moins 1,5 fois l'intervalle interquartiles. Les points qui se situent hors de ces limites sont considérés comme « extrêmes » ou « aberrants » (*outside values*, *outliers*) par rapport au reste de la distribution :

```
. graph medage, box by(region)
```



Les deux options **oneway** et **box** peuvent être combinées sur un seul graphique :

```
. graph medage, box oneway by(region)
```



On voit bien que la distribution de l'âge médian est très différente selon les régions. Les âges médians sont particulièrement dispersés dans la région Ouest, alors que les distributions apparaissent plus homogènes dans les autres régions. Cependant, dans la région Sud, on voit clairement un État pour lequel l'âge médian est extrême (à 34,7 ans).

Les boîtes sont un bon moyen de représenter synthétiquement la distribution d'une variable, et d'identifier les valeurs extrêmes ou aberrantes. En revanche, elles « écrasent » les irrégularités de la distribution (les différentes valeurs modales en particulier). L'histogramme représente mieux ces irrégularités, mais cela peut devenir un inconvénient lorsque le nombre d'observations est trop important : l'arbre cache la forêt, le détail cache l'allure générale de la distribution.

6.3.2.2 Le lissage des histogrammes

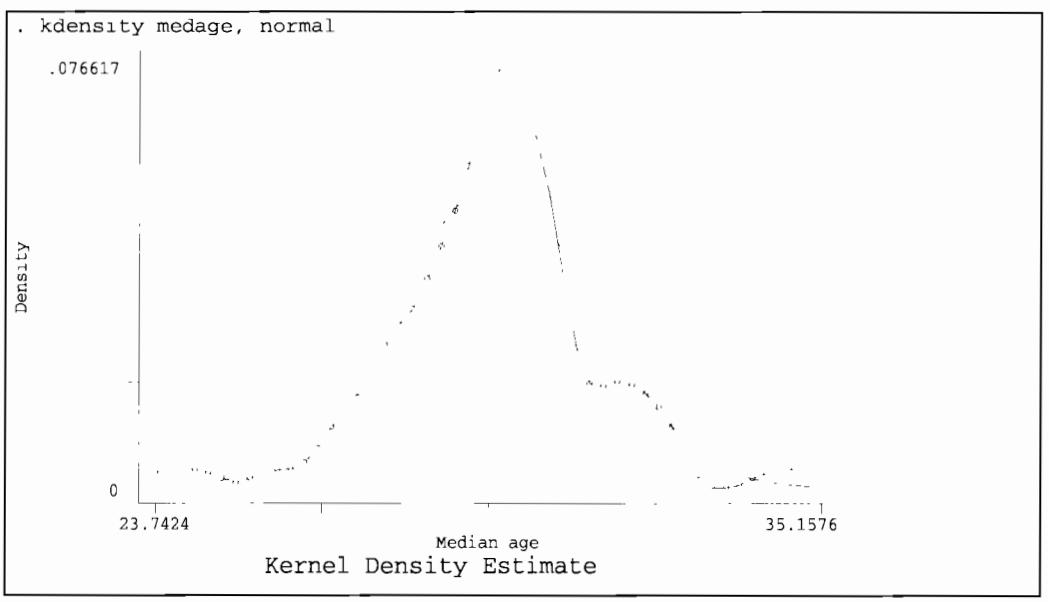
Il existe des techniques de lissage qui permettent justement de faire un compromis entre l'exigence de précision et l'exigence de généralisation. Dans un histogramme, la variable est divisée en un certain nombre d'intervalles équidistants et distincts (spécifiée par l'option **bin()** de la commande **graph**, soit 50 intervalles au maximum), et chaque intervalle est représenté par une barre verticale proportionnelle au nombre d'observations appartenant à cet intervalle. Le lissage consiste à construire des densités moyennes à partir de plusieurs éléments voisins dans des intervalles qui se chevauchent (à la différence de l'histogramme), et de relier ensuite ces densités moyennes pour représenter sous forme de courbe l'ensemble de la distribution. On appelle ce type de lissage une densité de kernel et la commande de base s'écrit :

kdensity *var* ,
[**generate(***varx vardensity***) width(#)**
biweight/cosine/epan/gauss/
parzen/rectangle/triangle
normal at(*varx***)**]

L'allure de la courbe dépend de l'intervalle qui servira à calculer chaque point de la densité de kernel : on appelle cet intervalle une fenêtre (*window*). Un calcul savant optimise la largeur de cette fenêtre en fonction du nombre d'éléments et des valeurs minimum et maximum de la distribution. On peut cependant forcer une largeur avec l'option **`width(#)`** où # représente la largeur voulue (voir exercices).

Par ailleurs, chaque point de la courbe est une moyenne des observations empiriques pondérée en fonction de leur éloignement du point central de l'intervalle. Les différentes fonctions de lissage attribuent des pondérations différentes à chacune des observations autour du point central : elles sont spécifiées par les options **`biweight`** à **`triangle`**, l'option par défaut étant **`epan`**.

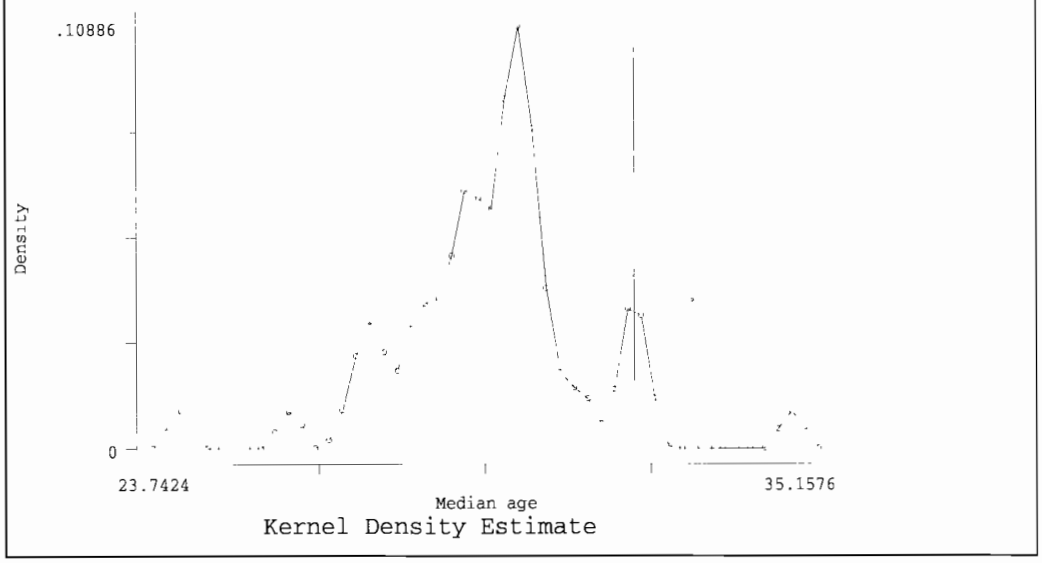
Ainsi, pour établir la densité de kernel de la variable **`medage`**, on exécutera :



La courbe de densité de kernel est indiquée par des petits cercles, tandis que l'option **`normal`** trace une courbe en trait continu. L'option **`epan`** par défaut a tendance à lisser un peu trop la distribution, de sorte que les différents modes disparaissent et que la densité de kernel apparaît suivre une loi (presque) normale. Mais pour la variable **`medage`**, les autres

options rendent mieux compte des irrégularités de la distribution, en particulier le mode autour de la valeur 32 (que nous avons représenté par un trait vertical) :

```
. kdensity medage, biweight xline(32)
```



La distribution apparaît maintenant plus irrégulière, et bimodale (si ce n'est multimodale).

Le résultat du lissage peut être conservé dans une nouvelle variable, ce qui est notamment utile pour comparer le résultat du lissage pour différentes catégories d'observations. La procédure est un peu complexe, mais elle vaut un petit effort d'attention. Ainsi pour comparer les distributions de **medage** par **region**, on aura :

(pour créer une variable indiquant les points pour lesquels la densité est calculée, ainsi qu'une variable contenant la valeur de la densité)

```
. kdensity medage, nograph gen(point fpoint) biweight
```

(pour créer une variable contenant la valeur de la densité pour la région Nord Est)

```
. kdensity medage if region==1, nograph gen(NE) at(point) biweight
```

(pour créer une variable contenant la valeur de la densité pour la région Centre Nord)

```
. kdensity medage if region==2, nograph gen(CN) at(point) biweight
```

(pour créer une variable contenant la valeur de la densité pour la région Sud)

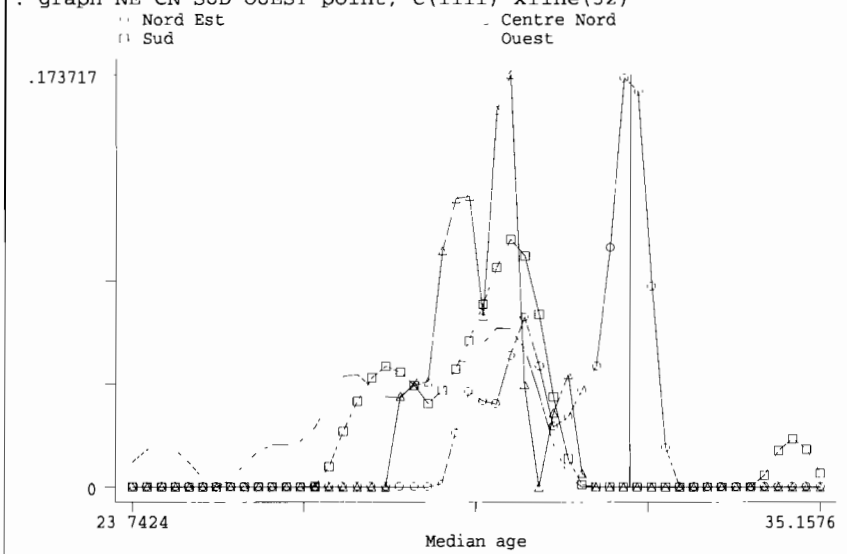
```
. kdensity medage if region==3, nograph gen(SUD) at(point) biweight
```

(pour créer une variable contenant la valeur de la densité pour la région Ouest)

```
. kdensity medage if region==4, nograph gen(OUEST) at(point) biweight
. label var NE "Nord Est"
. label var CN "Centre Nord"
. label var SUD "Sud "
. label var OUEST "Ouest "
```

(pour tracer sur un même graphique les courbes des quatre régions)

```
. graph NE CN SUD OUEST point, c(llll) xline(32)
```



Sur ce graphique (plus lisible à l'écran en couleur que sur papier en noir et blanc), on voit clairement que le mode à 32 ans est dû exclusivement à la région Nord Est dont les âges médians sont plus élevés que la moyenne des autres régions.

On pourrait être tenté d'utiliser l'option **by(** *var* **)** *de la commande* **graph** *pour produire une densité de kernel pour chaque catégorie de* *var* *:*

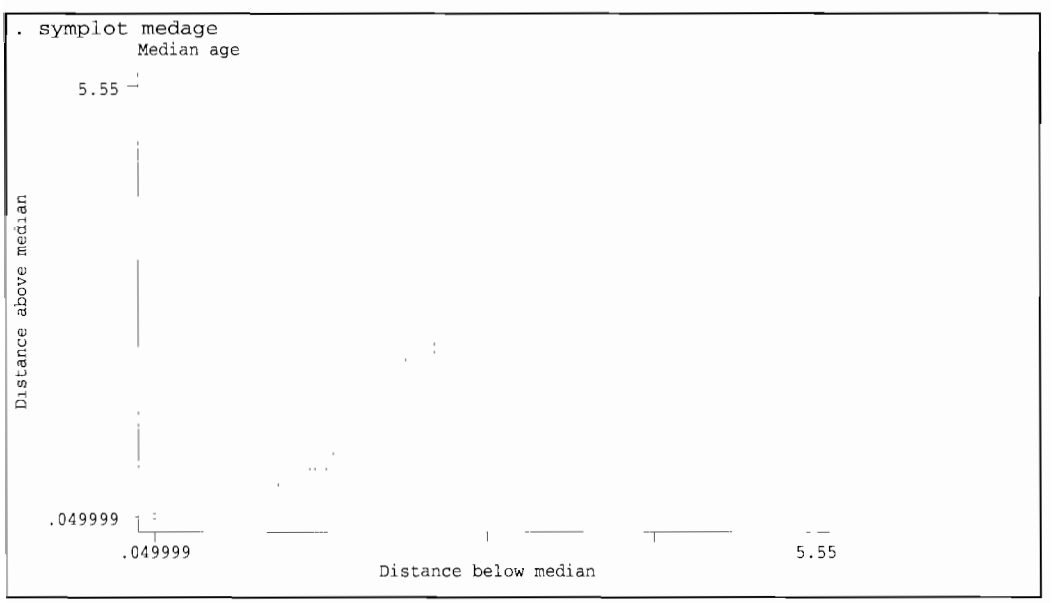
```
. kdensity medage, biweight by(region)
```

Cette méthode est absolument à proscrire, bien que cela ne soit pas explicité dans le manuel de Stata : vous pouvez vérifier que le graphique qui s'affiche à l'écran est totalement contradictoire avec le résultat obtenu avec la procédure décrite ci-dessus.

6.3.2.3 Diagnostics graphiques de symétrie et de conformité à la loi normale

Il existe plusieurs méthodes de diagnostic graphique pour évaluer la symétrie d'une distribution ainsi que sa conformité à la loi normale. Ces méthodes sont répertoriées sous le terme de *Distributional diagnostic plots* (entrée **diagplot** dans le manuel de Stata).

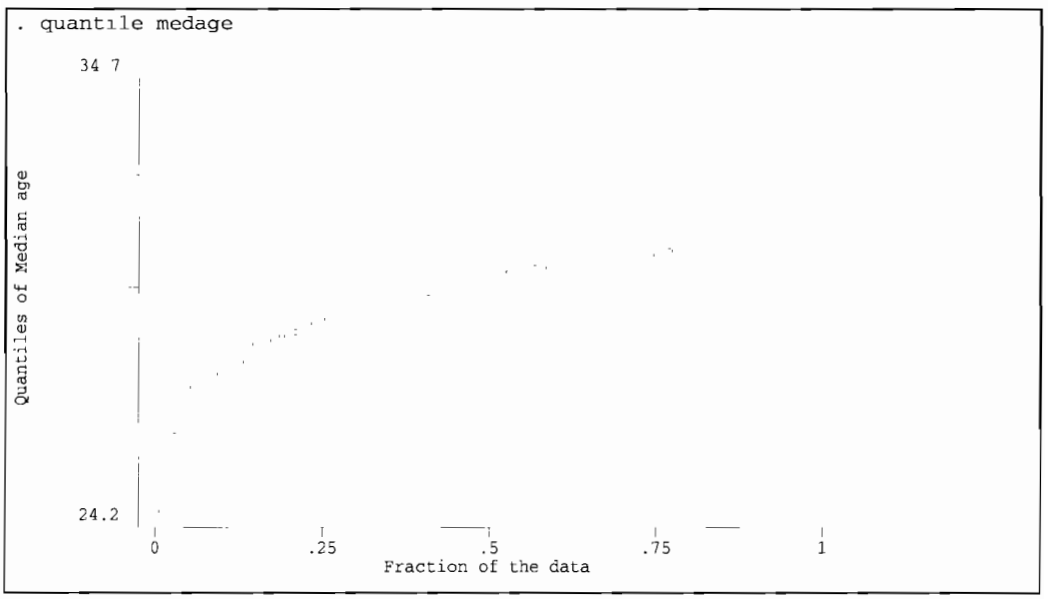
Un moyen de vérifier la symétrie d'une distribution est de comparer la différence entre la valeur médiane pour l'ensemble de la distribution et la valeur moyenne entre chaque paire d'observations constituées par les observations de rang opposé : par exemple on compare la médiane de **medage** avec la moyenne de **medage** pour l'observation de rang 1 et celle de rang N (soit 50), puis avec la moyenne de **medage** pour l'observation de rang 2 et celle de rang N-1, etc. Si la distribution était parfaitement symétrique, la moyenne pour chaque paire serait toujours égale à la médiane pour l'ensemble des observations. On obtient ainsi un graphique représentant la distance entre la médiane et la moyenne de chaque paire d'observations :



Lorsque les points se situent au dessus de la ligne de symétrie, la distribution est dissymétrique vers la droite (*skewed to the right*). Dans le cas de **medage**, le graphique indique une tendance dissymétrique à gauche.

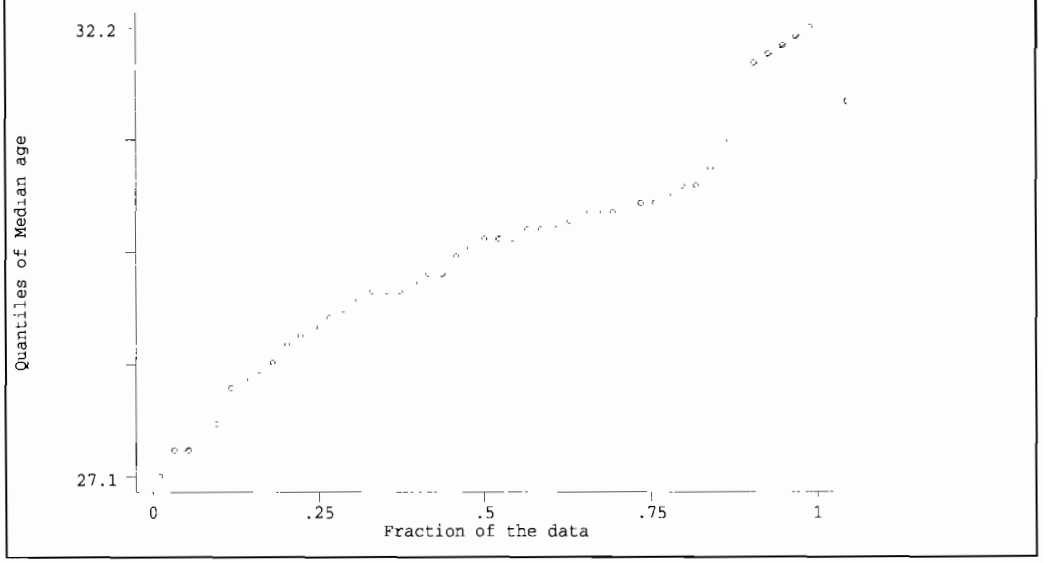
Cependant, ce diagnostic ne semble pas être confirmé par une autre méthode, qui reporte la valeur x de **medage** pour chaque observation (axe des ordonnées) contre la fraction d'observations (axe des abscisses) dont les valeurs sont inférieures à x :

```
. quantile medage
```



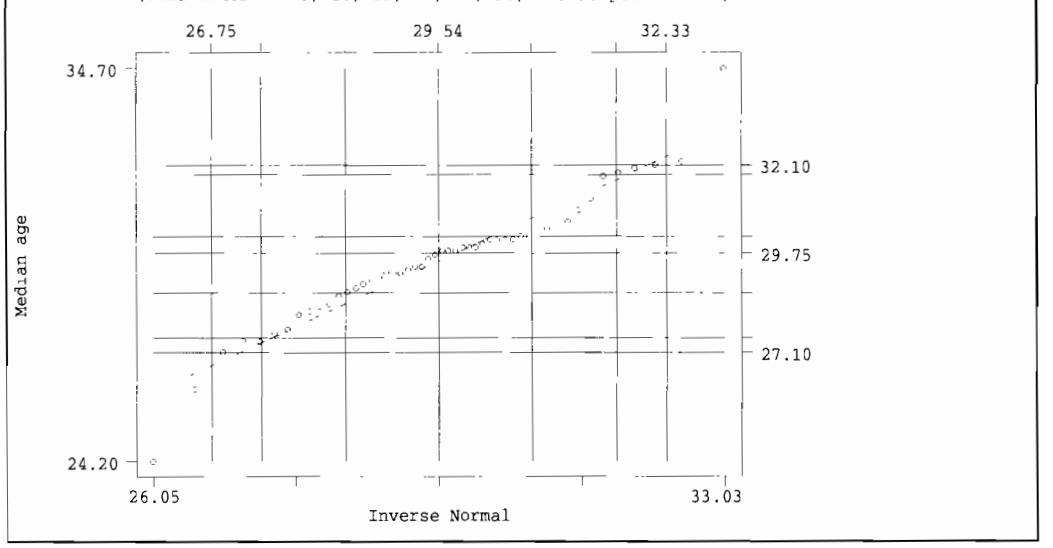
Dans ce graphique, les points se trouvent aussi bien au dessus qu'en dessous de la droite de référence. Cependant, cette droite joint les observations extrêmes, et ne rend pas bien compte d'une éventuelle dissymétrie par rapport à la médiane. Une astuce peut consister à éliminer les points extrêmes, qui ne s'alignent pas sur l'ensemble des points, pour mieux indiquer la dissymétrie :

```
. quantile medage if medage>26.2 & medage<34
```



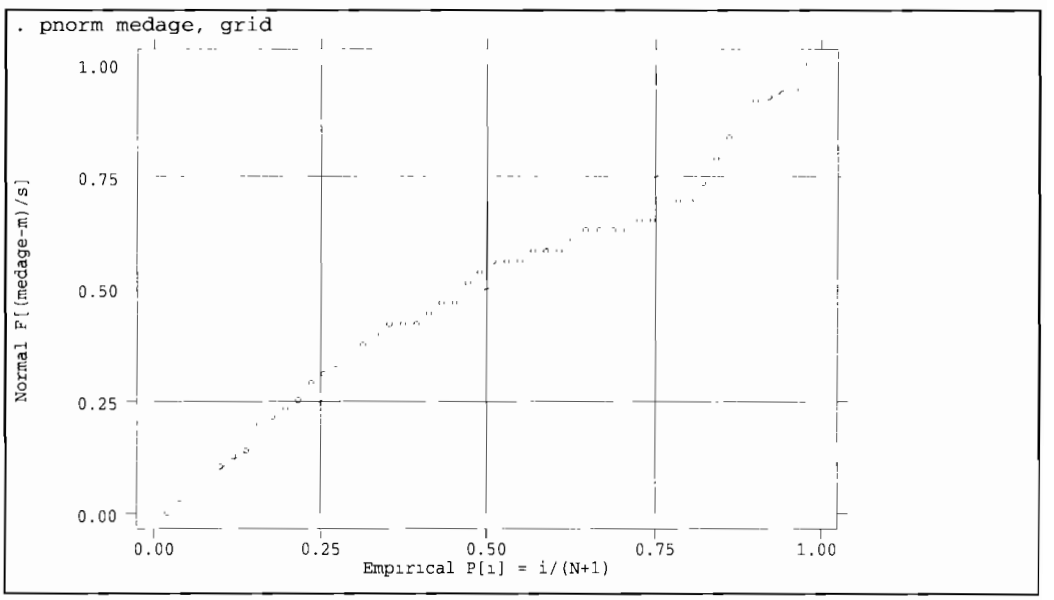
Un autre type de diagnostic consiste à confronter la distribution avec la loi normale. On a déjà vu que l'option **normal** de la commande **graph** superpose sur un histogramme une courbe normale de même moyenne et de même écart type que la distribution empirique. C'est la plupart du temps amplement suffisant mais on peut améliorer le diagnostic en comparant les quantiles des deux distributions (empirique et normale) :

```
. qnorm medage, grid
        (Grid lines are 5, 10, 25, 50, 75, 90, and 95 percentiles)
```



Les lignes verticales indiquent les quantiles de la distribution normale, et les chiffres en haut du graphique le 1er décile (26.75), la médiane (29.54) et le 9ème décile (32.33) correspondant à cette loi normale. Les lignes horizontales et les chiffres à droite du graphique représentent les quantiles de la distribution empirique. Ce graphique, mieux que le précédent, indique une relative symétrie (les quantiles sont répartis à peu près à égale distance de la médiane) et une bonne concordance avec la distribution normale (les points s'alignent assez bien sur la droite de référence, hormis les observations extrêmes).

Le même type de graphique peut être obtenu en confrontant la distribution empirique avec la distribution normale standardisée correspondante (c'est-à-dire transformée pour que sa moyenne soit égale à 0 et son écart type à 1) :



Les distorsions par rapport à la loi normale sont surtout visibles pour les valeurs supérieures à la médiane.

L'option **symbol()** *facilite le repérage des observations extrêmes dans les représentations graphiques :*

```
. graph medage, box by(region) symbol([state])
. qnorm medage, grid symbol([state])
```

A quels États correspondent les valeurs extrêmes ? Refaites les mêmes graphiques en excluant ces États (utilisez le suffixe **if***).*

Comparez le résultat des commandes suivantes :

```
. graph medage, xlab(24,26,28,30,32,34,35) normal freq
. graph medage, bin(11) xlab(24,26,28,30,32,34,35) normal freq
. graph medage, bin(50) xlab(24,26,28,30,32,34,35) normal freq
```

L'interprétation est-elle la même dans tous les cas ? Quel est l'intervalle qui donne les résultats les plus proches de la distribution réelle ?

Comparez les boîtes obtenues pour l'ensemble de la distribution avec les résultats suivants :

```
. graph med if med>26.2 & med<34, box by(region) ysc(24.2,34.7)
```

Comparez la densité de kernel par défaut (option **epan***) et les densités obtenues avec différentes fenêtres (option* **width(#)***) pour chaque fonction de densité (option* **biweight**, **cosine**, **gauss**, **parzen**, **rectangle**, **triangle***), par exemple :*

```
. kdensity medage, rectangle normal
. kdensity medage, rectangle normal width(1)
. kdensity medage, rectangle normal width(.1)
```

À quelle unité de mesure correspond la valeur # ? La largeur de la fenêtre a-t-elle plus ou moins d'influence que la fonction utilisée ?

Ecrivez une série de commandes pour comparer les densités de kernel de **medage** *selon le caractère* **urbain** *ou non des États. Confrontez ensuite la distribution empirique avec la distribution normale pour chaque catégorie de la variable* **urbain** *(en utilisant le suffixe* **if***) à l'aide de différentes méthodes (histogramme, quantiles). La distribution de* **medage** *suit-elle une loi normale et est-elle symétrique dans chaque catégorie ?*

6.3.3 Graphique de distribution croisée de deux variables continues

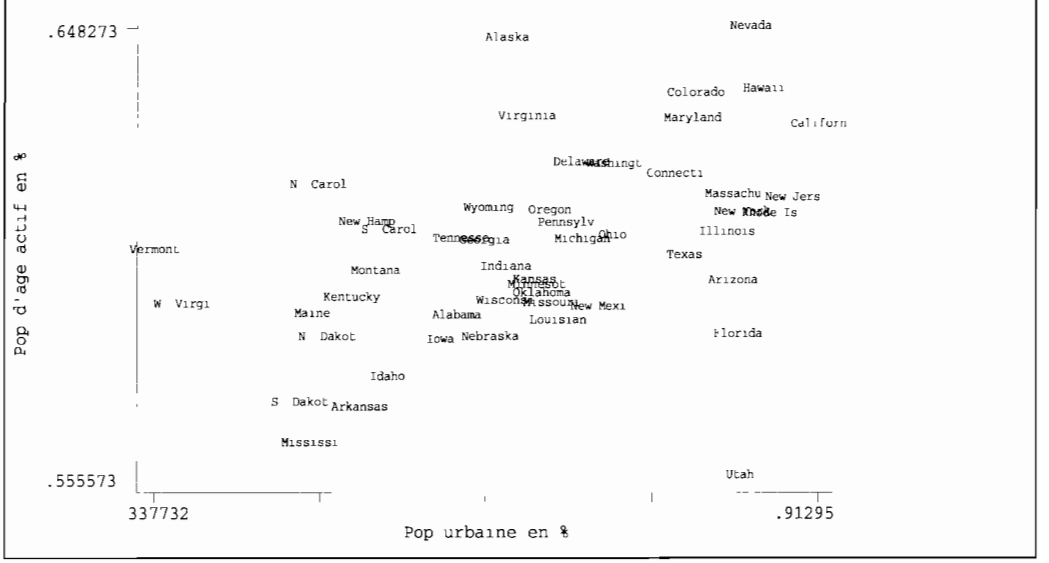
Les graphiques précédents croisent une variable dépendante (continue ou discrète) avec une deuxième variable discrète de contrôle : chaque graphique obtenu est un ensemble ou une superposition de graphiques univariés.

Dans la présente section, nous examinons un autre type de graphique qui croise deux variables continues. Le but est alors de représenter visuellement la relation entre deux variables. La syntaxe de base de **graph** pour produire ce graphique est :

graph *vary1 [vary2] varx*, **twoway**
[**connect(***./l/m/s/J/L/||/II***) band(#)**
by(*listevar***)**
oneway box rbox rescale Rescale jitter(#)]

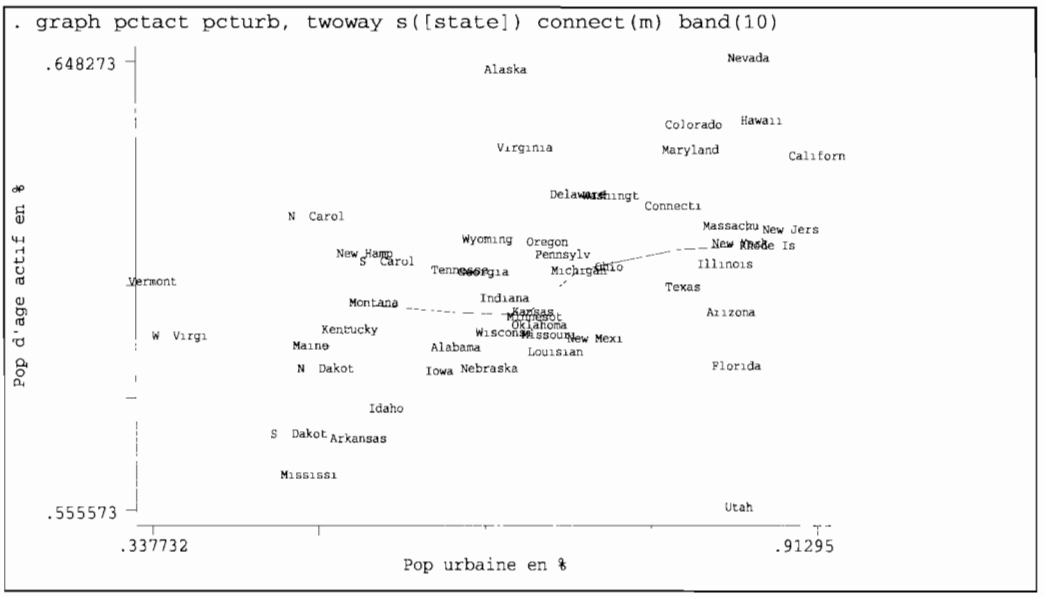
Prenons pour exemple la relation entre le pourcentage de la population en milieu urbain (**pcturb**) et le pourcentage de la population d'âge actif de 18 à 64 ans (**pctact**) :

```
. generate pctact=(pop18p-pop65p)/pop

. lab var pctact "Pop d'age actif en %"

. graph pctact pcturb, twoway s([state])
```



Il semble bien qu'il y ait une relation positive entre les deux variables ce qu'on peut mieux voir encore en ajoutant au graphique une courbe de lissage avec les options **connect(m)** et **band(#)** :

```
. graph pctact pcturb, twoway s([state]) connect(m) band(10)
```



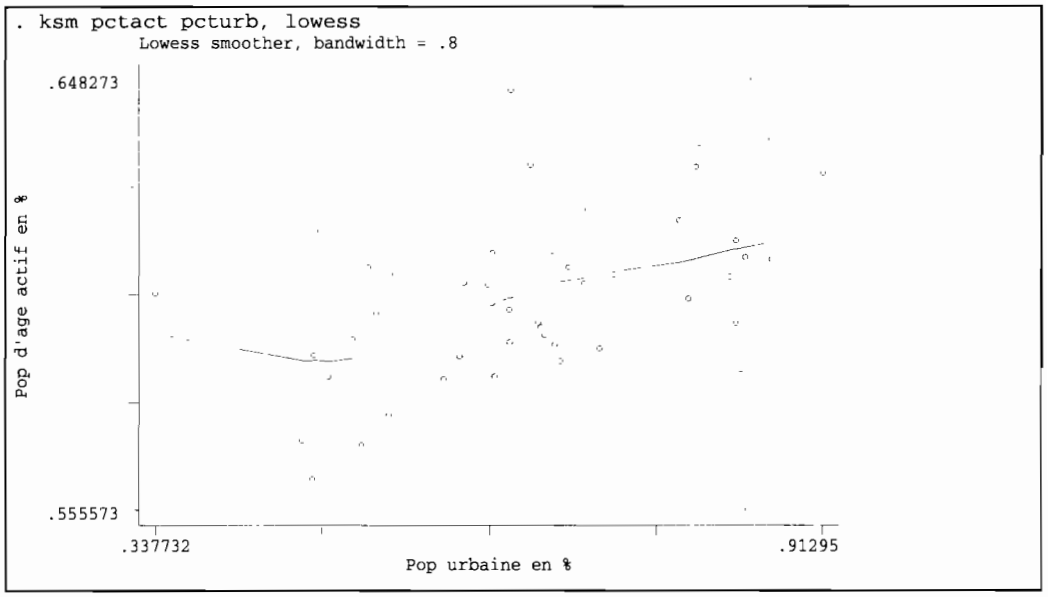
Le lissage consiste à diviser l'axe des abscisses en intervalles égaux (ici on en a pris 5), et de calculer les médianes des deux variables pour les observations comprises dans ces intervalles. Les médianes sont représentées par des nœuds et une ligne est ensuite tracée pour relier ces noeuds entre eux. Le lissage est d'autant plus important que le nombre d'intervalles est grand.

Un raffinement consiste à courber cette ligne brisée en utilisant l'option **connect(s)** plutôt que **connect(m)**. Les noeuds sont calculés de la même façon, seule diffère la forme de la courbe qui les relie entre eux.

Un autre raffinement consiste à faire un lissage de type *lowess* (*locally weighted scatterplot smoothing*) à l'aide de la commande **ksm**. Cette commande attribue à chaque point une valeur prédite par une régression linéaire pour l'ensemble des points voisins (pondérés en fonction de la distance avec le point prédit). Le principe est très proche du calcul des densités de kernel que l'on a vu plus haut. Le paramètre qui change l'intensité du lissage est le pourcentage de points inclus dans

chaque régression, avec l'option **bwidth(#)**. La valeur par défaut est de .8 (soit 80 %) :

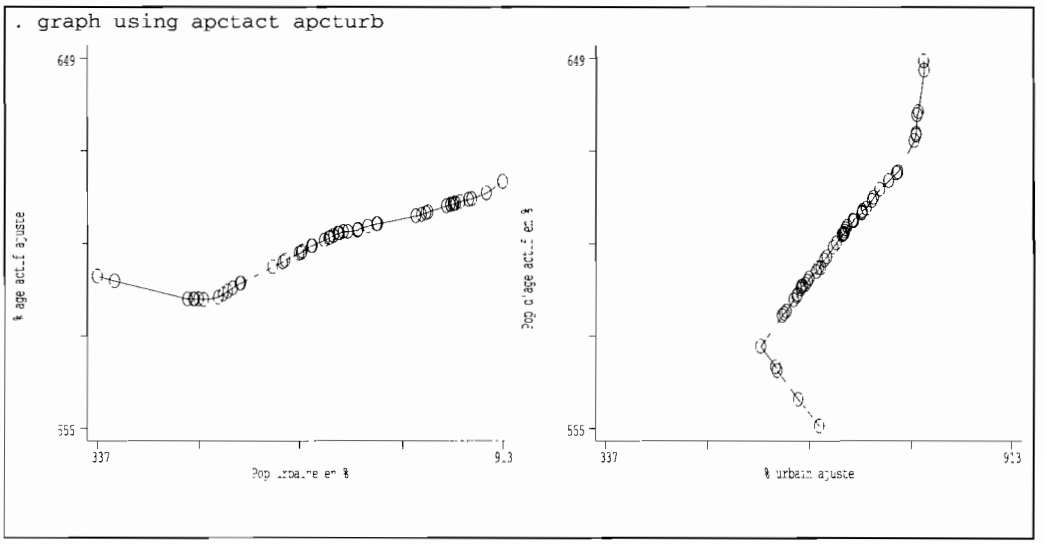
```
. ksm pctact pcturb, lowess
```



Il est important de comprendre que, quelle que soit la technique de lissage (valeurs médianes ou régression), la variable dépendante (ici **pctact**) est en ordonnée tandis que la variable indépendante ou explicative (ici **pcturb**) est en abscisse. Étant donné que le lissage se fait pour différents sous-ensembles d'observations définis selon la valeur de la variable indépendante, la courbe de lissage aura une allure différente si l'on inverse l'ordre des variables.

Pour s'en convaincre, il suffit de comparer les deux courbes de lissage, par exemple avec la technique de la régression, grâce à l'option **gen(*var*)** qui conserve les données ajustées de la variable dépendante :

```
. ksm pctact pcturb, lowess gen(apctact) nograph
. lab var apctact "% age actif ajuste"
. graph apctact pcturb, c(l) ysc(.555,.649) xsc(.337,.913) sort sav(apctact)
. ksm pcturb pctact, lowess gen(apcturb) nograph
. lab var apcturb "% urbain ajuste"
. graph pctact apcturb, c(l) ysc(.555,.649) xsc(.337,.913) sav(apcturb)
```



Comme on le voit immédiatement, la relation entre les deux variables n'est pas symétrique.

En somme, les techniques de lissage pour les distributions bivariées font l'hypothèse implicite que la première variable doit s'ajuster sur la deuxième, c'est-à-dire que les variations aléatoires sont attribuées à la première variable et non à la deuxième. Par exemple, en lissant les valeurs de **pctact**, on suppose que le pourcentage d'actifs dans un État dépend du caractère urbain de cet État, et non l'inverse. Le lissage est donc déjà une forme de modèle explicatif.

Les lecteurs intéressés par des techniques plus complexes de lissage à partir des valeurs médianes pourront se référer à la commande **smooth**, *que nous ne détaillerons pas ici. Mentionnons cependant (ce qui n'apparaît pas dans le manuel de Stata) qu'avant d'exécuter cette commande sur une variable (par exemple* **pctact**), *le fichier doit être trié selon la deuxième variable (***pcturb** *dans notre exemple). On est donc toujours en présence d'un modèle explicatif implicite où la variable de tri est la variable indépendante.*

La forme de la courbe dépend en grande partie des paramètres de lissage. Comparez les résultats en faisant varier l'option **band()** *ou bien l'option* **bwidth()** *:*

```
. graph pctact pcturb, twoway connect(m)

. graph pctact pcturb, twoway connect(m) band(5)

. graph pctact pcturb, twoway connect(m) band(20)

. ksm pctact pcturb, lowess bwidth(.5)

. ksm pctact pcturb, lowess bwidth(.2)
```

Si vous êtes familiers avec les techniques d'analyse de régression, les techniques qui précèdent ne sont pas sans vous rappeler quelque chose. Effectivement, les deux types de techniques sont très proches : le lissage peut être considéré comme une technique de régression non linéaire.
Exécutez les commandes suivantes pour comparer les résultats obtenus par les différentes méthodes sur un même graphique :

```
. regress pctact pcturb

. predict rpctact

. ksm pctact pcturb, lowess gen(apctact) nograph

. graph pctact apctact rpctact pcturb, c(mll) s(oii) sort band(10)
```

6.3.4 Graphique de distribution multivariée

On a déjà vu qu'un tableau croisant plus de quatre variables est difficilement compréhensible. Cependant, le cerveau humain perçoit mieux des nuances à travers une représentation graphique qu'à travers un raisonnement écrit ou chiffré. C'est pourquoi on recourt plus souvent aux graphiques dès que le nombre de dimensions dépasse deux. L'œil a une capacité extraordinaire pour analyser en très peu de temps l'aspect d'un graphique, et détecter les tendances globales en même temps que les extrêmes ou les exceptions.

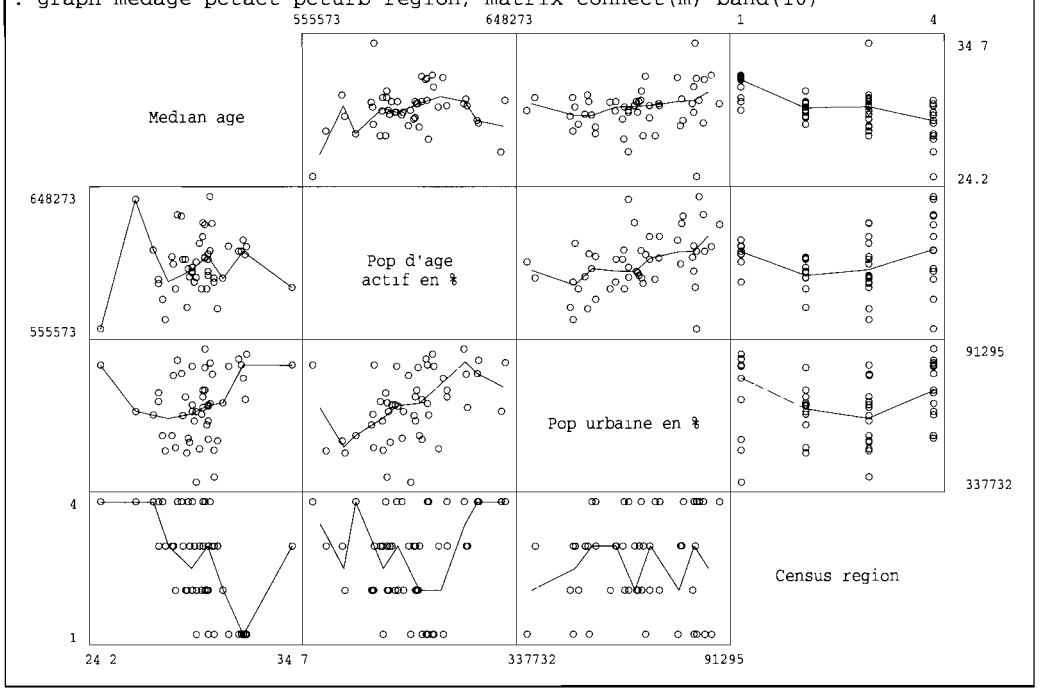
Cependant, il n'existe que très peu de techniques de représentation multivariée, pour une raison simple : la représentation sur un plan (à l'écran ou sur papier) est nécessairement en deux dimensions, même si des effets de

perspectives peuvent donner l'illusion d'une troisième dimension.

En fait, les graphiques dits multivariés sont des superpositions, ou des alignements de graphiques univariés ou bivariés. Nous avons déjà vu comment faire de tels graphiques avec l'option **by(***var***)** commune à de nombreuses commandes. Cette section est consacrée aux graphiques multivariés qui ne font pas appel à l'option **by**.

L'option **matrix** établit une série de graphiques bivariés croisant chaque paire de variables énumérées :

```
. graph medage pctact pcturb region, matrix connect(m) band(10)
```

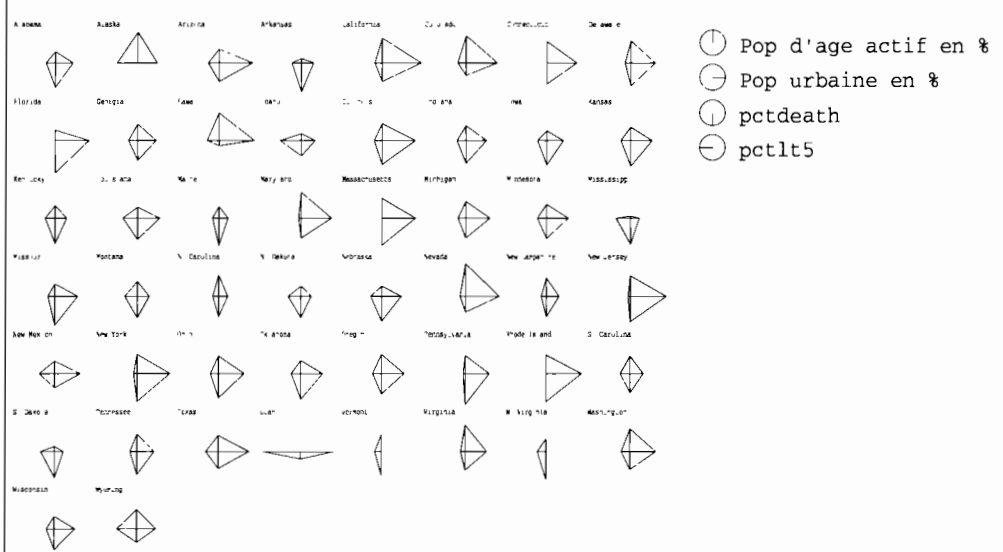


Les graphiques au dessus de la diagonale (qui contient les libellés des variables) sont la réplique inversée des graphiques en dessous de cette diagonale. Remarquez cependant que les résultats du lissage par l'option **connect(m)** ne sont pas symétriques. Comme on l'a dit plus haut, le lissage dépend de l'ordre des variables, et en conséquence certains lissages sont difficilement interprétables : par exemple, on ne voit pas très bien pourquoi l'appartenance d'un État à une région devrait

dépendre de l'âge médian, du pourcentage de population d'âge actif ou du taux d'urbanisation.

On a déjà vu l'application des graphiques en étoile pour chacune des modalités d'une même variable, quand on a décrit la répartition par groupe d'âges de la population de chaque État. Ce type de graphique permet aussi de représenter des variables de différentes natures : par exemple, on pourra classer chaque État selon sa part relative de population active, de population urbaine, d'enfants de moins de 5 ans et de décès :

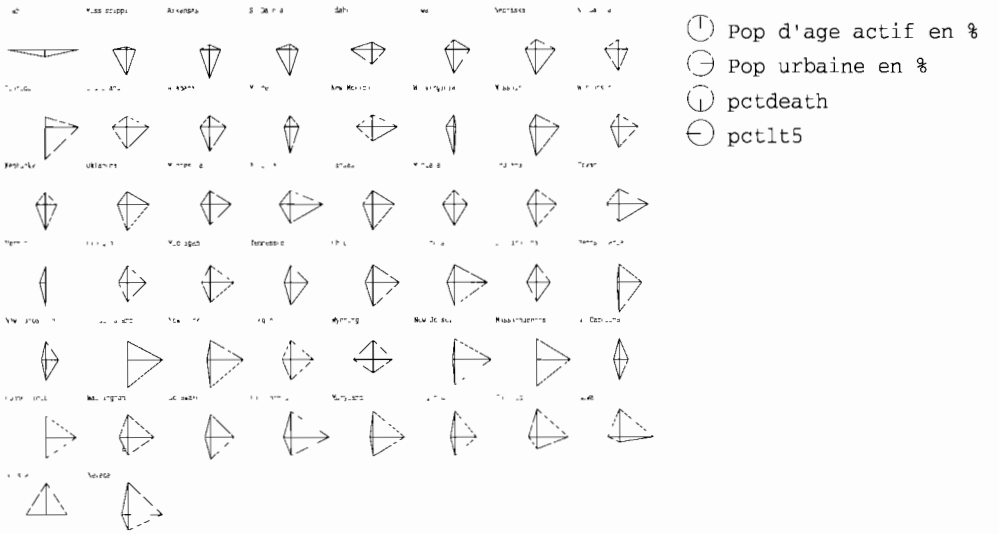
```
. gen pctlt5=poplt5/pop

. gen pctdeath=death/pop

. graph  pctact pcturb pctdeath pctlt5 , star label(state)
```



Au stade de l'analyse exploratoire des données, cette technique graphique est en quelque sorte une technique de classification multiple et de repérage des observations extrêmes ou aberrantes.

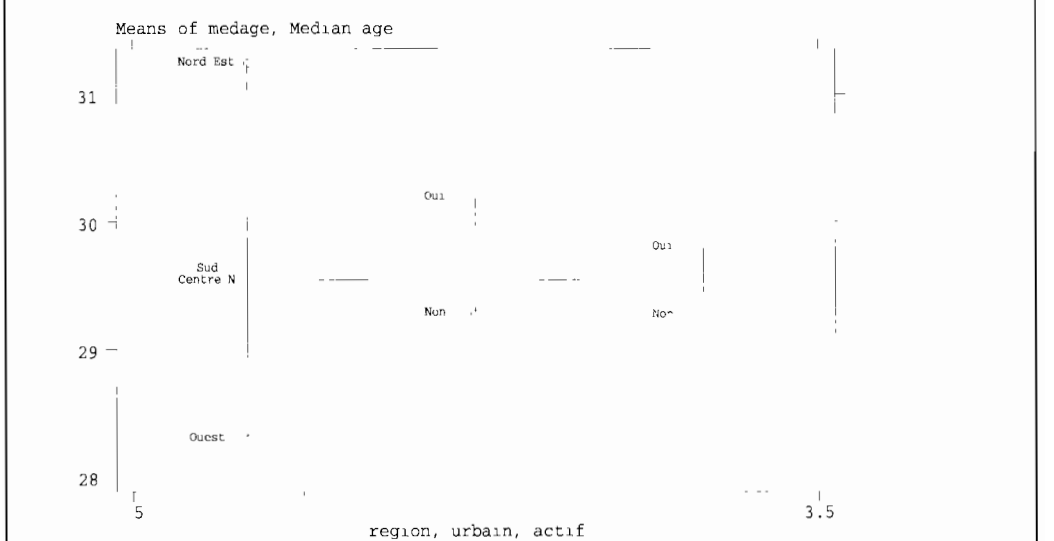
Le graphique ci-dessus a été obtenu à partir d'un fichier trié selon l'ordre alphabétique des États. Pour faciliter la lisibilité du graphique (c'est-à-dire le repérage des États aux structures particulières, et le regroupement des États aux structures similaires), il est conseillé de trier d'abord les observations selon les variables représentées :

```
. sort pctact pcturb pctdeath pctlt5

. graph  pctact pcturb pctdeath pctlt5 , star label(state)
```

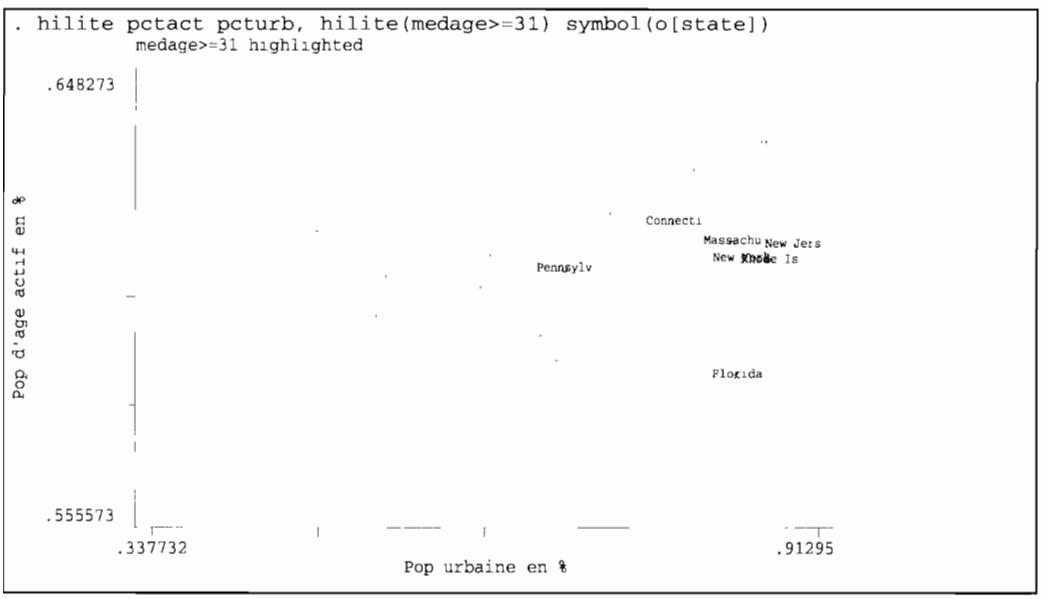


Enfin terminons ce chapitre avec deux commandes originales. La commande **`grmeanby`** permet de représenter graphiquement la moyenne (ou la médiane) d'une variable continue pour différentes modalités de plusieurs variables dichotomiques ou polytomiques :

```
. grmeanby  region urbain actif , summ(medage) ylab
```



La commande **hilite** permet de repérer plus facilement une catégorie d'observations dans un graphique bivarié. Sa syntaxe est très proche de la commande **graph** et l'option **symbol()** peut être judicieusement utilisée pour repérer les observations sélectionnées :



Faites un graphique en étoile en ajoutant la variable **region** *:*

```
. sort region pctact pcturb pctdeath pctlt5
. graph region pctact pcturb pctdeath pctlt5, star label(state)
```

Quel intérêt présente un tel graphique ? Les variables discrètes peuvent-elles être interprétées de la même façon que les variables continues ou ordonnées ?

L'option **symbol()** *pour la commande* **hilite** *peut aider à repérer les observations extrêmes classées selon une quatrième variable :*

```
. hilite pctact pcturb, hilite(medage>=31) symbol(o[region])
```

---

7 L'ANALYSE STATISTIQUE : APPROCHE PAR TYPE DE PROBLÈMES

Les chapitres précédents vous ont montré l'essentiel de Stata. En maîtrisant la manipulation des données, la création de variables, les tabulations et les graphiques, vous êtes autonome. Avec ces outils de base, vous pouvez maintenant aller plus en profondeur dans l'analyse statistique.

Ce chapitre a pour objectif de guider le lecteur au travers de la multitude des procédures statistiques proposées par Stata. Il s'agit principalement de commandes de modélisation statistique, depuis la simple régression linéaire jusqu'à la régression semi-paramétrique des données de survie, en passant par la régression logistique et ses variantes.

Plutôt que de reprendre une à une chacune des commandes (ce que le manuel de Stata fait déjà), ou d'aller en profondeur dans la statistique mathématique (ce qui nous mènerait bien au delà du propos de ce manuel), nous allons adopter une approche par type de problèmes. Il ne s'agit évidemment pas de recenser tous les types de problèmes possibles, mais, à partir de la définition très générale de la variable dépendante pour un problème donné, de guider le lecteur vers le ou les modèles statistiques adaptés à ce problème.

On sera peut-être surpris de constater que le choix du type de modèle ne dépend pas des variables indépendantes (explicatives). En fait, les procédures actuelles de régression traitent indifféremment les variables indépendantes, qu'elles soient continues, ordonnées ou discrètes. Par contre, il existe une

différence importante selon que les données sont individuelles ou agrégées : chaque fois que cela sera nécessaire, nous mentionnerons les modèles correspondants.

7.1 Les différents types de variables dépendantes

Pour l'analyse descriptive, nous avons fréquemment utilisé la distinction entre variables discrètes, ordonnées et continues. Le traitement statistique est différent dans chacun de ces cas, autant pour l'analyse descriptive que pour la modélisation.

Pour les besoins de ce chapitre, nous classerons les variables dépendantes (ce qu'on cherche à expliquer) selon deux catégories principales :

- les *variables discrètes*, qui sont des variables codées par un chiffre entier positif, allant de 0 à n ; une variable discrète est appelée *dichotomique* si elle ne prend que deux valeurs 0 et n (avec $n \neq 0$) et *polytomique* sinon ($n>1$) ;

- les *variables continues*, qui sont des variables codées par un chiffre réel, indifféremment entier ou décimal, positif ou négatif.

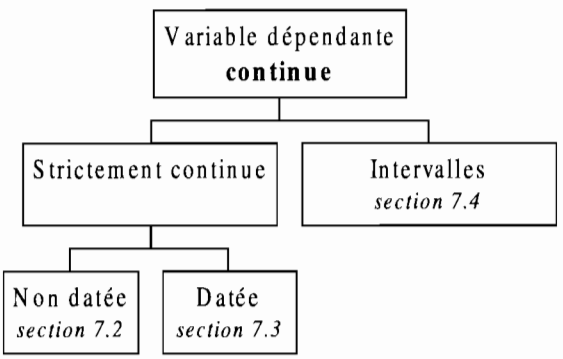
Cette différence est fondamentale du point de vue du traitement informatique. Mais qu'en est-il des variables ordonnées ? En fait, elles peuvent elles-mêmes être considérées comme faisant partie soit des variables discrètes soit des variables continues :

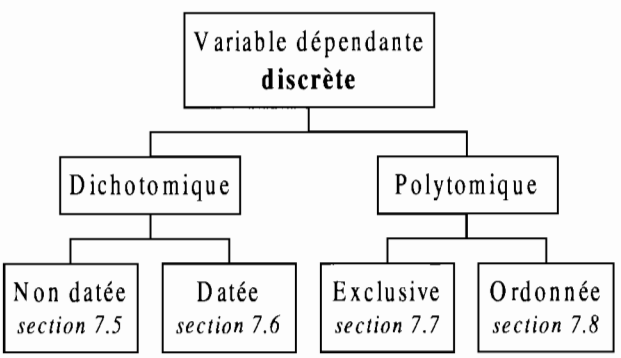
- par opposition aux variables dichotomiques ou polytomiques non ordonnées (dont les modalités représentent des catégories exclusives les unes des autres dans un ordre quelconque), les *variables polytomiques ordonnées* sont codées par un chiffre entier dont l'ordre a un sens, mais pas la valeur (par exemple, il est indifférent de coder ce type de variables de 0 à 4, de 1 à 5, de 102 à 118 par intervalle de 4, ou même 1,4,5,13,99) ;

- les *variables à intervalles multiples*, parfois ouverts, qui sont des approximations de variables continues (par exemple, des tranches de revenus, avec un intervalle ouvert pour les plus hauts revenus).

Enfin, il est fondamental de préciser si une date est associée à chaque valeur de la variable dépendante. La prise en compte du temps dans l'analyse définit une classe de modèle à part, autant pour les variables continues que discrètes.

Les deux diagrammes suivants résument tous ces paramètres, selon que la variable dépendante est discrète ou continue :





On remarque au passage que les variables dépendantes polytomiques (discrètes ou continues par intervalles) ne présentent pas de variation selon qu'elles sont datées ou non : à l'heure actuelle, il n'existe pas dans Stata de modèles qui prennent en compte le temps pour les variables dépendantes polytomiques.

Stata offre un nombre considérable de commandes pour la modélisation statistique. Dans ce manuel, nous en présentons environ une quarantaine. C'est largement trop pour en décrire tous les détails. En fait, moins d'une vingtaine présentent un

intérêt spécifique, les autres étant des variantes plus ou moins complexes.

Dans la suite de ce chapitre, chaque section correspond aux modèles présentés selon un classement par type de variable dépendante, tel qu'indiqué dans les diagrammes précédents. Ce classement (même s'il se justifie aussi d'un point de vue scientifique) a un but essentiellement pratique : ce chapitre n'a pas pour prétention de faire de la théorie statistique, mais entend guider le lecteur en fonction du type de problème auquel il est confronté. Nous laissons le lecteur se référer soit au manuel de Stata (qui offre, en anglais, de très bonnes explications sur les différents modèles) ou bien aux ouvrages spécialisés de statistiques.

Enfin, notre classification ne s'adapte pas bien aux modèles non régressifs, où la distinction entre variables dépendantes et indépendantes n'est pas valide : analyses factorielles, analyses en composantes principales, etc. Signalons cependant que la commande **`factor`** de Stata effectue ces analyses, dans leur version probabiliste anglo-saxonne (qui diffère sensiblement de la version française dans la tradition initiée par J.-P. Benzécri).

7.2 *Les modèles à variable continue non datée*

Le modèle de régression linéaire simple est à la fois le modèle fondateur de tous les autres modèles de régression, le plus simple de tous ces modèles et le modèle qui offre le plus de variantes. Sa simplicité apparente est en fait tributaire de nombreuses hypothèses simplificatrices : l'histoire de ce modèle est une longue succession de tentatives pour lever ou minimiser ces hypothèses.

En conséquence, un très grand nombre de procédures sont associées à ce modèle. Dans cette section, nous accorderons un intérêt tout particulier aux estimations robustes des coefficients de régression et de leur variance. En fin de section, nous aborderons le modèle de régression non linéaire (ou paramétrique).

7.2.1 Le modèle de régression linéaire

La commande de base pour la régression simple s'écrit simplement :

regress *vardep varindep*

La variable dépendante doit obligatoirement se situer en début de liste, et elle est suivie des variables indépendantes. Ce principe vaut d'ailleurs pour toutes les procédures de régression.

Les variables indépendantes peuvent être indifféremment continues ou discrètes. Dans le cas d'une variable polytomique, on peut soi-même créer une série de variables dichotomiques à l'aide de l'option **gen()** de la commande **tabulate** (voir le chapitre *Création et correction de variables*) et éliminer la catégorie de référence :

```
. tabulate region, gen(reg)
(tableau omis)

. drop reg1
```

Mais il est beaucoup plus aisé de faire appel au préfixe **xi** qui crée automatiquement une série de variables dichotomiques à partir des variables marquées d'un **i.** dans la commande :

```
. xi: regress pctact pcturb i.region
i.region              Iregio_1-4   (naturally coded; Iregio_1 omitted)

  Source |       SS       df       MS                  Number of obs =      50
---------+------------------------------                F(  4,    45) =    3.86
   Model |  .004920394     4  .001230098                Prob > F      =  0.0089
Residual |   .01435289    45  .000318953                R-squared     =  0.2553
---------+------------------------------                Adj R-squared =  0.1891
   Total |  .019273284    49  .000393332                Root MSE      =  .01786

------------------------------------------------------------------------------
  pctact |      Coef.   Std. Err.       t     P>|t|       [95% Conf. Interval]
---------+--------------------------------------------------------------------
  pcturb |   .0494978   .0188318      2.628   0.012       .0115686     .087427
Iregio_2 |  -.0128641   .0079323     -1.622   0.112      -.0288405    .0031123
Iregio_3 |  -.0052268    .007586     -0.689   0.494      -.0205057    .0100522
Iregio_4 |   .0014747   .0077856      0.189   0.851      -.0142063    .0171557
   _cons |   .5733959   .0143885     39.851   0.000       .5444161    .6023758
------------------------------------------------------------------------------

. describe I*

  42. Iregio_2    byte   %8.0g                 region==2
  43. Iregio_3    byte   %8.0g                 region==3
  44. Iregio_4    byte   %8.0g                 region==4
```

La région Nord Est (codée 1) a été omise et fait donc office de catégorie de référence. Le pourcentage d'habitants d'âge actif semble positivement corrélé au pourcentage d'habitants vivant dans les villes, mais peu à l'appartenance régionale : seul le coefficient pour la région du Centre Nord pourrait être significatif (au seuil, plutôt élevé, de 15 %).

7.2.1.1 Violation de l'hypothèse de linéarité

Dans ce modèle, les variables indépendantes, mais aussi la variable dépendante, peuvent être plus ou moins bien spécifiées. Par exemple, l'échelle de mesure de la variable dépendante n'est peut-être pas appropriée, et les observations extrêmes auront certainement une influence indue sur le résultat de la régression. Dans ce cas, l'hypothèse de linéarité (de la relation entre variables dépendante et indépendante) n'est pas forcément adéquate.

Une solution à ce type de problèmes est de transformer l'échelle de la variable dépendante pour vérifier l'hypothèse de linéarité. Une transformation classique est la transformation en une distribution proche d'une distribution normale à l'aide de la commande **boxcox** dont la syntaxe de base est :

boxcox *vardep* [*varindep*]
[, **mean median generate(***vardeptransformée***)**]

Si les variables indépendantes sont spécifiées, il suffit d'exécuter ensuite la commande **regress** sans argument pour obtenir les résultats de la régression avec la variable dépendante transformée. L'option **generate()** permet de conserver le résultat de la transformation dans une nouvelle variable. Les options **mean** et **median** sont appropriées dans le cas où la variable dépendante a une distribution symétrique.

Exécutez les commandes suivantes pour transformer la variable dépendante **pctact** *:*

```
. boxcox pctact pcturb Iregio*

. regress
```

Les résultats de la régression diffèrent-ils de la régression simple ? (comparez en particulier les coefficients de régression)
Avant de faire une telle transformation, il aurait été plus judicieux de voir d'abord la distribution empirique de **pctact** *et de la comparer à la loi normale (voir la section sur l'ajustement des courbes univariées) :*

```
. kdensity pctact, biweight normal

. qnorm pctact, grid
```

La transformation aurait peut-être été plus valide si l'on avait tenu compte de la relative symétrie de la distribution de **pctact** *:*

```
. boxcox pctact pcturb Iregio*, median

. regress
```

La transformation centrée a-t-elle plus d'intérêt ? Que vous inspire notamment le graphique obtenu par **qnorm** *et comment interprétez-vous en conséquence les différences ou l'absence de différence avec la régression simple ?*

7.2.1.2 Violation de l'hypothèse de mesure correcte des variables indépendantes

Certaines variables indépendantes peuvent être mesurées avec approximation. Dans ce cas, on voudrait pouvoir tenir compte de la fiabilité de ces mesures dans la régression. La commande **eivreg** (*Error-in-variables regression*) tient compte pour une ou plusieurs variables indépendantes d'une mesure de fiabilité, qui correspond au pourcentage de la variance totale mesurée que l'on peut attribuer à la variance réelle (le reste étant attribué aux erreurs de mesure). La syntaxe est la même que pour le modèle de régression simple, hormis l'option définissant la fiabilité :

eivreg *vardep varindep* ,
r(*varindep # [varindep # [...]]***)**

Le problème est que la plupart du temps la fiabilité n'est pas connue, mais suspectée. L'option **r()** est alors utile pour tester l'influence que pourrait avoir un erreur de mesure des variables

indépendantes sur les résultats de la régression. Autrement dit, on veut connaître la robustesse des résultats.

On va donc tester différents niveaux de fiabilité. Le niveau minimum est déterminé par la statistique du R^2 (*R-squared*) pour la régression de la variable indépendante à tester sur toutes les autres variables (dépendante et indépendantes) du modèle. Une fiabilité inférieure à ce R^2 signifierait que la variable indépendante à tester est connue avec une trop grande imprécision.

Pour évaluer le manque de fiabilité (supposé) du taux d'urbanisation on exécutera d'abord :

```
. regress pctact pcturb Iregio*
```

La statistique du R^2 est de 0,2337. On peut donc tester différentes mesures de fiabilité allant par exemple, de 0,25 (25 %) à 1 (100 %) :

```
. eivreg pctact pcturb Iregio* , r(pcturb .25)
. eivreg pctact pcturb Iregio* , r(pcturb .5)
. eivreg pctact pcturb Iregio* , r(pcturb .7)
. eivreg pctact pcturb Iregio* , r(pcturb .8)
. eivreg pctact pcturb Iregio* , r(pcturb .9)
. eivreg pctact pcturb Iregio* , r(pcturb 1)
```

Comparez ces résultats avec ceux de la régression simple.
À quoi correspond une fiabilité de 100 % ? Quel niveau de fiabilité pensez-vous pouvoir tolérer ?

7.2.1.3 Violation de l'hypothèse d'homoscédasticité (homogénéité de la variance de la variable dépendante)

Cette hypothèse est couramment violée et une très grande part de la littérature statistique est consacrée aux moyens d'évaluer l'hétéroscédasticité et de la contourner. La régression sur la médiane est, contrairement à la régression sur la moyenne, peu sensible à l'hétérogénéité de la variable dépendante : nous reviendrons sur cette technique plus loin.

Pour la régression sur la moyenne, si l'on suppose le différentiel de variance connu précisément, on peut alors en tenir compte

dans le calcul avec la commande **vwls** (*variance-weighted least square*), dont la syntaxe de base est :

vwls *vardep varindep* [, **sd(***var***)**]

Deux cas peuvent se présenter :

1. la variable dépendante est mesurée avec une erreur connue pour chaque observation (ce cas se présente essentiellement pour les sciences de la nature) : on utilise alors l'option **sd()** pour spécifier la variable qui indique l'écart type pour chaque observation ;

2. on mesure la variance dans chaque groupe constitué par le croisement des variables indépendantes (qui de ce fait doivent être nécessairement discrètes et non continues), et on fait la régression sur les observations ainsi groupées (ce cas est plus fréquent dans les sciences humaines et médicales) : l'option **sd()** n'est pas valide puisque l'écart type est calculé à partir des observations groupées.

Plus loin figure une autre technique d'estimation robuste de la variance pour les données échantillonnées.

Nous allons prendre ici un exemple pour illustrer le deuxième cas. Comparez les résultats de la régression simple avec la régression pondérée correspondante :

```
. regress pctact I* pcturb

. vwls pctact I* pcturb

. regress pctact I* urbain

. vwls pctact I* urbain
```

Pourquoi la deuxième commande renvoie-t-elle un message d'erreur ? Pourquoi la dernière régression porte-t-elle sur 49 États seulement ? Pour vous aider à répondre à cette question, considérez attentivement le tableau résumé suivant :

```
. table urbain region, c(mean pctact sd pctact freq)
```

Les exercices qui suivent ont pour but d'indiquer sommairement des commandes non mentionnées jusqu'à présent pour la régression linéaire simple.

Un nombre important de commandes sont disponibles pour le diagnostic de la régression linéaire. Pour cela, il faut utiliser la commande **fit** *plutôt que* **regress** *: les options de* **regress** *pour les estimations robustes empêchent certains diagnostics, et la commande* **fit** *a été créée pour éviter les confusions.*

Pour tester l'hétéroscédasticité, on peut effectuer après **fit** *:*

```
. fit pctact Iregio* pcturb

. hettest
```

*Pour repérer graphiquement les observations aberrantes (*outliers*), on représente les valeurs résiduelles contre les valeurs ajustées de la variable dépendante (*__rvfplot__*) :*

```
. rvfplot, symbol([state])
```

Quel est l'État le plus éloigné de la droite de régression ?

*On appelle les observations influentes celles dont le poids influe fortement sur la pente de la droite de régression. Si elles sont à la fois influentes et extrêmes (éloignées de la masse des autres observations), on parle alors d'effet levier (*leverage effect*). On peut déterminer à la fois les observations aberrantes et les observations influentes en exécutant la commande* **lvr2plot** *:*

```
. lvr2plot, s([state])
```

D'après ce graphique, les États aberrants sont-ils en même temps des États influents ?

Les graphiques de diagnostic utilisent certains indices que l'on peut obtenir avec la commande **fpredict** *après* **fit** *:*

```
. fpredict residu, residuals        (valeurs résiduelles)

. fpredict levier, leverage         (valeurs à fort effet levier)
```

Voir l'entrée [R] fit - Regression fit and diagnostics *du manuel de Stata pour d'autres graphiques et statistiques plus complexes.)*

L'extrapolation des données d'enquête peut être prise en compte dans la régression avec la commande **svyreg**.
Il est conseillé de définir d'abord la stratification et les unités primaires avec la commande **svyset** *(voir la section sur les pondérations et les extrapolations), mais on peut aussi les définir en option de la commande* **svyreg**. *À supposer que nos données soient issues d'un sondage, on aurait alors :*

```
. svyreg pctact Iregio* pcturb, strata(strate) psu(grappe)
```

Remarquez la différence entre l'extrapolation à partir des données d'enquête, et la simple pondération avec l'option **[a/f/i/pweight]**
Dans notre exemple, tiré du recensement, on utilisera le chiffre de la population pour pondérer les États. Dans ce cas, la pondération dite analytique est la plus appropriée :

```
. regress pctact Iregio* pcturb [aweight=pop]
```

La commande **cnsreg** *(à ne pas confondre avec la commande* **cnreg** *qui est traitée plus loin) permet d'imposer des contraintes dans les coefficients de régression linéaire.*
Par exemple, si l'on veut imposer le même coefficient pour deux catégories d'une même variable polytomique (ce qui revient à fusionner les deux catégories), on définit d'abord la contrainte par la commande **constraint** *avant d'exécuter* **cnsreg** *:*

```
. constraint define 1 Iregio_3=Iregio_4

. cnsreg pctact Iregio* pcturb, constraint(1)
```

La commande cnsreg peut être utilement associée aux tests sur les coefficients de la régression. Confrontez par exemple les résultats précédents avec le test suivant :

```
. test Iregio_3=Iregio_4
```

Les contraintes ne concernent pas seulement les relations entre les variables indépendantes (ou leur modalités) mais aussi la valeur des coefficients eux-mêmes. Par exemple, on peut être curieux de voir l'effet sur les coefficients par région, d'une constante définie à l'avance :

```
. constraint define 2 _cons=.5

. cnsreg pctact Iregio* pcturb, constraint(2)
```

(Remarquez l'absence d'erreur type et de statistiques de significativité pour la constante du modèle)

7.2.2 Les estimations robustes de la variance

On a vu plus haut le moyen de déceler les violations à l'hypothèse d'homogénéité de la variance de la variable dépendante (homoscédasticité) et d'y remédier en utilisant la variance observée dans chaque groupe avec la commande **vwls**.

Une autre méthode consiste à considérer systématiquement les données comme issues d'un échantillonnage (sujet à certaines variations aléatoires, notamment des erreurs de mesure) et non comme des résultats absolument exacts. Sans rentrer dans le détail du calcul d'estimation, disons simplement que les erreurs types des coefficients ainsi obtenues tiennent compte de l'hétérogénéité de la variance (c'est-à-dire de la dispersion au sein de chaque groupe d'observations).

Les estimations robustes de la variance sont données par l'option **robust** de la commande **regress** (cette option n'est pas disponible pour la commande **fit**). De plus, l'option **cluster(***var***)** associée à l'option **robust**, allège l'hypothèse d'indépendance entre les observations, pour chaque groupe défini par *var*. Pour calculer des estimations robustes de la variance, on tient alors compte de l'hétérogénéité de la variance entre les groupes d'observations, et non pas entre les observations d'un même groupe. Dans le contexte des données échantillonnées, l'option **cluster()** est donc une alternative utile à la commande **vwls** qui s'applique aux données où la variance est supposée exacte et précisément connue.

Cela n'a pas de sens de considérer les données de « census.dta » (issues du recensement) comme des données échantillonnées mais si c'était le cas on pourrait obtenir des estimations robustes pour les erreurs types des coefficients de la régression en faisant :

```
. regress pctact Iregio* pcturb, robust
```

Si les pourcentages d'actifs n'étaient pas indépendants par région, on pourrait considérer ces régions comme des grappes :

```
. regress pctact Iregio* pcturb, robust cluster(region)
```

Remarquez qu'à la différence de la commande **vwls**, *on peut utiliser des variables continues (telle que* **pcturb**) *dans la régression.*

7.2.3 Les estimations robustes des coefficients

Les données sur lesquelles on travaille ne sont pas toujours connues précisément, soit en raison d'erreurs de mesure, soit en raison d'un échantillonnage approximatif ou insuffisant. Dans ce cas, on peut attribuer aux observations extrêmes ou aberrantes un poids moins important que les observations proches du centre de la distribution. À la différence de la section précédente où l'on a calculé des variances robustes, on veut maintenant obtenir des coefficients robustes, sans trop se soucier des observations qui pourraient influer grandement sur les estimations.

Deux méthodes sont disponibles pour obtenir de telles estimations. Si l'on raisonne toujours sur les tendances centrales mesurées par la moyenne (régression classique), la commande **`rreg`** est appropriée :

`rreg` *vardep* [*varindep*]
[,**`iterate(#)`** **`tolerance(#)`** **`genwt(`***var***`)`**]

La procédure consiste à attribuer un poids moins important aux observations qui s'éloignent trop du nuage de points. Un poids initial est attribué à chaque observation et la procédure utilise une distance avec le point central pour attribuer un nouveau poids à l'observation lors de l'itération suivante, et ainsi de suite.

Le nombre d'itération maximum peut être contrôlé par l'option **`iterate(#)`** tandis que l'option **`tolerance(#)`** spécifie le critère de convergence entre deux itérations successives (valeur par défaut : 0,01). Le poids obtenu après estimation robuste pour chaque observation peut être conservé dans une nouvelle variable par l'option **`genwt(`***var***`)`**.

Une autre forme de régression moins sensible aux observations extrêmes ou aberrantes est la régression sur les quantiles. Au lieu de considérer la moyenne comme mesure de la tendance centrale pour le calcul des coefficients, la commande **`qreg`** utilise la médiane, très peu sensible aux observations en queue de distribution.

Cette méthode est excellente pour obtenir des coefficients de régression robuste, mais elle conduit à sous-estimer les erreurs types de ces coefficients. Aussi, quel que soit le problème envisagé, il vaut mieux toujours utiliser l'estimation par la méthode du *bootstrapping*, terme intraduisible qui signifie que les estimations des erreurs types sont obtenues par rééchantillonnage des observations :

bsqreg *vardep* [*varindep*]
[, **reps(#) iterate(#) quantile(#)**]

Le nombre de rééchantillonnage (réplication de l'échantillon) par défaut est de 20, ce qui suffit amplement à un stade exploratoire. Pour obtenir des estimations définitives, après sélection du modèle optimum, on peut augmenter le nombre de réplications (à 100 ou plus par exemple) avec l'option **reps(#)**.

L'originalité de la commande **qreg** (ou **bsqreg**) est de permettre la régression sur d'autres quantiles que la médiane : par exemple, le premier quartile ou le troisième, ou tout autre décile ou centile. Dans le cas où les observations présentent une relative homoscédasticité, les régressions sur les premier, deuxième et troisième quartiles devraient donner à peu près les mêmes résultats. Dans le cas contraire (forte hétéroscédasticité), les résultats différeront. Ce type de régression est donc aussi un bon moyen de voir si l'hypothèse d'homogénéité de la variance est respectée ou non.

Comparer les résultats de la régression classique et de la régression robuste sur la moyenne :

```
. regress pctact pcturb Iregio*

. rreg pctact pcturb Iregio*, genwt(probust)
```

Quels sont les coefficients les plus affectés par les observations extrêmes ou aberrantes ? La variable **probust** *contient le poids de chaque observation après estimation. Pour repérer les États dont on a diminué le poids pour estimer les coefficients robustes, on peut exécuter :*

```
. sort probust

. list state probust in 1/10
```

L'estimation sur la médiane donne aussi des résultats intéressants :

```
. bsqreg pctact pcturb Iregio*
```

Comparez avec la régression simple et avec la régression robuste sur la moyenne : quels sont les résultats les plus proches ? Pourquoi ? Pour évaluer l'homogénéité de la variance entre les États, comparez les résultats sur les trois quartiles :

```
. bsqreg pctact pcturb Iregio*, quant(.25)

. bsqreg pctact pcturb Iregio*, quant(.75)
```

D'après ces résultats, l'hétérogénéité est-elle forte ou non ? En d'autres termes, les États faiblement ou fortement actifs s'écartent-ils du centre de la distribution ? Sur quels États l'effet de l'urbanisation mesurée par **pcturb** *est-il le moins fort ?*

7.2.4 La régression multivariée et la régression multiple

Cette section s'adresse aux lecteurs avertis et rompus aux statistiques avancées. La régression multivariée est en effet d'un usage peu courant et sert essentiellement à faire des tests complexes. La régression multiple disponible dans Stata est d'un usage délicat : elle distingue une équation principale et une équation secondaire dans la régression.

Dans la régression multivariée, plusieurs variables dépendantes interviennent simultanément dans l'équation pour un même ensemble de variables indépendantes :

mvreg *vardep1 vardep2... :* [*varindep*] [**, corr**]

La procédure est absolument équivalente à faire une régression pour chaque variable dépendante, mais elle donne la possibilité de tester les valeurs des coefficients et d'obtenir la matrice de corrélation des résidus sur l'ensemble des régressions. Avec l'option **corr**, on peut voir si les variables dépendantes restent ou non indépendantes l'une de l'autre après la régression, et avec la commande **test** on peut tester si une ou plusieurs variables indépendantes sont significatives sur l'ensemble des régressions.

Le modèle précédent est en fait un cas particulier d'un modèle plus général où les variables indépendantes ne sont pas forcément les mêmes pour chaque régression. Cette fois-ci, pour chaque variable dépendante, un ensemble différent de variables indépendantes peut être choisi. La syntaxe de la commande est par conséquent plus complexe, car elle demande de définir d'abord chacune des équations du modèle :

```
eq vardep1 varindep
eq vardep2 varindep
...
sureg vardep1 vardep2 [, corr ]
```

Contrairement au modèle précédent, le modèle n'est pas équivalent à estimer plusieurs régressions séparément, sauf si les variables indépendantes sont les mêmes pour chaque équation (ce qui revient à exécuter **mvreg**). En revanche, on obtient la matrice de corrélation des variables dépendantes avec l'option **corr**, et on peut toujours tester avec **test** le degré de signification de certaines variables indépendantes.

Dans la commande **sureg**, chacune des équations est équivalente : elles ont la même valeur, le même niveau hiérarchique. La commande **regress** offre la possibilité de distinguer une équation principale (ou structurelle) où figure la variable dépendante d'intérêt, et une équation secondaire, « nichée » (*nested*) au sein de la première. Dans cette équation secondaire, une des variables indépendantes de l'équation principale est considérée comme variable dépendante dans une régression sur un autre ensemble de variables indépendantes. La syntaxe de base de la régression multiple est :

```
regress vardep1 vardep2 varindep1
        (varindep1 varindep2)
```

Pour distinguer les termes des deux équations, la variable dépendante de l'équation secondaire (*vardep2*) ne doit pas figurer dans la liste des variables indépendantes entre parenthèses : à ce titre *vardep2* est considérée comme une variable endogène (qui obéit à un processus propre), tandis que

les variables indépendantes de l'équation principale `varindep1` et secondaire `varindep2` sont considérées comme exogènes. On remarque que les variables indépendantes exogènes de l'équation principale (`varindep1`) figurent à la fois dans la liste des variables indépendantes de l'équation principale et dans la liste des variables exogènes entre parenthèses.

La régression multivariée permet de mesurer la corrélation entre les résidus de plusieurs régressions sur un même ensemble de variables indépendantes :

```
. mvreg pctact pcturb : Iregio*, corr
```

Le résultat du test est significatif : cela indique-t-il que les deux variables **pcturb** *et* **pctact** *sont indépendantes l'une de l'autre ? L'appartenance régionale semble très peu significative pour expliquer les variations autant du niveau d'urbanisation que du pourcentage d'actifs. Vérifier que l'appartenance à l'une quelconque des régions n'est pas significatif pour les deux régressions jointes :*

```
. test Iregio_2
. test Iregio_3
. test Iregio_4
```

Comparez les résultats des commandes **mvreg** *et* **sureg** *:*

```
. eq pcturb Iregio*
. eq pctact Iregio*
. sureg pctact pcturb, corr
```

La commande **sureg** *n'oblige pas à définir le même ensemble de variables indépendantes entre les équations :*

```
. eq pctact pcturb
. eq pcturb Iregio*
. sureg pctact pcturb, corr
```

Ce modèle multivarié, où chaque équation est indépendante l'une de l'autre est à bien distinguer de la régression multiple où les deux équations sont hiérarchisées :

```
. regress pctact pcturb (Iregio*)
```

Dans cette régression, quelle est la variable dépendante dans l'équation secondaire ? Quelles sont les variables exogènes ? Pourquoi la variable **pcturb** *ne figure-t-elle pas entre parenthèses ?*

7.2.5 Les régressions non linéaires

Rappelons que l'hypothèse principale de la régression classique est que la relation entre la variable dépendante et les variables indépendantes est... linéaire. Pour mieux se représenter l'importance de cette hypothèse, on peut imaginer un espace où chaque point représente la position de chaque observation : la régression linéaire consiste à tracer une droite à travers le nuage formé par ces points, de telle manière à minimiser la distance entre cette droite et ces points.

L'hypothèse principale de la régression non linéaire est tout aussi contraignante : elle stipule que la relation entre la variable dépendante et les variables indépendantes suit une forme donnée, une fonction définie par l'utilisateur. Cette forme n'est pas une droite, mais la régression consiste toujours à minimiser la distance entre la fonction de régression et les points d'observations.

La fonction de régression non linéaire doit être définie dans un sous-programme écrit dans le langage de Stata : cette procédure s'adresse bien évidemment aux utilisateurs avertis ayant une bonne connaissance des principes de programmation. Cependant, un certain nombre de fonctions courantes sont disponibles dans la version standard du logiciel, pour la régression sur une seule variable indépendante. La syntaxe de base de la régression non linéaire est :

nl *fonction vardep varindep*

Les fonctions courantes sont décrites dans le manuel de Stata. Il s'agit des fonctions exponentielles à deux ou trois paramètres (**exp2**, **exp2a** et **exp3**), des fonctions logistiques et de Gompertz à trois ou quatre paramètres (**log3** et **log4**, **gom3** et **gom4**).

Pour les fonctions plus complexes faisant intervenir plusieurs variables indépendantes, il faut écrire ses propres sous-programmes. Nous n'expliquerons pas ici cette procédure qui dépasse le cadre de ce manuel.

La régression non linéaire sur une seule variable indépendante est comparable aux procédures de lissage graphique que l'on a abordé dans le chapitre précédent. Comparez les résultats suivants :

```
. graph pctact pcturb, twoway s([state]) connect(m) band(10)
. ksm pctact pcturb, lowess
. nl log4 pctact pcturb
. nlpred ajlog4
. graph ajlog4 pctact pcturb, c(l.) sort
```

D'autres fonctions peuvent être testées, avec plus ou moins de succès :

```
. nl exp2 pctact pcturb
. nlpred ajexp2
. graph ajexp2 pctact pcturb, c(l.) sort
. nl exp2a pctact pcturb
. nlpred ajexp2a
. graph ajexp2a pctact pcturb, c(l.) sort
```

7.3 *Les modèles à variable continue datée*

Dans les modèles de régression classique, la relation entre les variables est considérée comme atemporelle. La mesure de chaque variable a en principe été faite au même moment, et par conséquent le temps n'intervient pas comme paramètre dans la régression.

Dans cette section, nous allons aborder des modèles où la variable dépendante est toujours continue, mais associée à une durée ou à une date.

7.3.1 Les processus de comptage

Dans le modèle de Poisson, le nombre d'événements dépend de la durée d'exposition (ou de la surface d'exposition, dans quelques cas de figure ne faisant pas référence au temps) et de certaines variables indépendantes. La syntaxe de base de la commande est :

poisson *vardep varindep*
[, exposure(#) offset(#) irr]

Le temps d'exposition peut être défini soit directement par l'option **exposure(#)** soit sous forme logarithmique par l'option **offset(#)**. Le rapport entre les taux d'incidence (*incidence rate ratio*, c'est-à-dire les coefficients sous forme exponentielle) sont obtenus avec l'option **irr**.

Très proche du modèle de Poisson, le modèle de régression binomiale négative tient compte d'une éventuelle dispersion entre les observations qui pourrait suivre une autre loi que celle de Poisson. En d'autres termes, la régression binomiale négative tient compte d'une hétérogénéité non observée, c'est-à-dire non captée par les variables indépendantes dans la régression de Poisson. Mesurer l'importance de cette hétérogénéité non observée est utile pour tester la conformité au processus de Poisson : si l'hétérogénéité non observée est significative, le modèle doit être rejeté. La syntaxe de base est :

nbreg *vardep varindep*
[**, exposure(#) offset(#) irr**]

Un modèle plus général encore permet de paramétriser la variance qui dépend d'un processus de Poisson et la variance supplémentaire qui s'ajoute à ce processus. Ce modèle permet aussi de tester la conformité au processus de Poisson, mais cette fois l'hétérogénéité supplémentaire est elle-même modélisée : elle est supposée dépendre d'un certain nombre de variables indépendantes. La syntaxe est plus complexe dans la mesure où il faut définir deux équations :

eq *vardep varindep1*

eq *eq2: varindep2*

gnbreg *vardep eq2*
[**, exposure(#) offset(#) irr**]

Enfin, dans le cas où plusieurs mesures auraient été faites sur les mêmes individus, on utilisera une commande qui lie les observations relevant d'un même individu :

xtpois *vardep varindep*
[**, exposure(#) i(***ident***) robust irr**]

Cette commande est équivalente à la commande **poisson**, hormis l'option **i()** qui spécifie l'identifiant des individus pour plusieurs observations. Il est en outre fortement conseillé d'utiliser l'option **robust** pour obtenir des erreurs types qui tiennent compte du fait que certaines observations ne sont pas indépendantes l'une de l'autre puisqu'elles se rapportent au même individu (voir plus haut la section sur *Les estimations robustes de la variance*).

La commande **xtpois** est à notre avis improprement classée parmi les commandes pour l'analyse des séries temporelles. En effet, les observations pour chaque individu ne sont pas classées dans le temps : le modèle ne tient pas compte du fait qu'une première observation d'un individu a pu avoir lieu un jour, des semaines, des mois ou des années avant une seconde ou une troisième observation. La durée d'exposition peut varier d'une observation à l'autre mais l'intervalle de temps entre les observations n'est pas pris en compte.

Supposons que la population totale d'un État soit une bonne évaluation du temps d'exposition en termes d'hommes-années. Pour tester si le nombre de décès dépend de la part d'actifs, du degré d'urbanisation et de l'appartenance régionale, on exécutera :

```
. poisson death pct1t5 pct5_17 pct65p pcturb Iregio*, exposure(pop)
```

Pour tester la conformité au processus de Poisson, on préferera :

```
. nbreg death pct1t5 pct5_17 pct65p pcturb Iregio*, exposure(pop)
```

Le test par rapport à la distribution de Poisson montre une différence très significative qui montre que l'on doit rejeter le modèle de Poisson. Le nombre de décès dépend peut-être plus de la composition par âges de la population que de l'urbanisation et de l'appartenance régionale. Dans ce cas, on peut tester un modèle plus général où ces dernières variables interviennent séparément par rapport au processus de Poisson.

```
. eq death pct1t5 pct5_17 pct65p

. eq region: pcturb Iregio*

. gnbreg death region, exposure(pop)
```

Après un nombre considérable d'itérations, que conclut le test ? Doit-on rejeter le processus de Poisson ? Voir plus loin le modèle logistique sur des données groupées pour une meilleure estimation.

Pour illustrer par des exemples l'utilisation des commandes pour l'analyse des séries temporelles et pour l'analyse de survie, nous vous demandons de bien vouloir saisir un fichier de données fictives à l'aide par exemple de l'éditeur intégré de Stata (assurez-vous qu'aucun fichier n'est en mémoire, puis cliquer sur **Editor** *dans le menu : voir le chapitre sur les fichiers de données) :*

Intercooled Stata 5.0

File Edit Prefs Window Help

Log... Dialog Results Graph Regraph Editor Browse More Break

Stata Editor

Preserve Restore Sort << >> Hide Delete...

ident[1] = 1

	ident	date	sexe	taux	even
1	1	1991	0	143	0
2	1	1992	0	132	0
3	1	1993	0	156	1
4	2	1990	1	123	0
5	2	1991	1	124	0
6	2	1992	1	127	0
7	2	1995	1	145	0
8	3	1993	1	98	0
9	3	1994	1	101	0
10	4	1990	0	102	0
11	4	1992	0	120	0
12	4	1993	0	122	0
13	4	1994	0	137	0
14	4	1995	0	160	1
15	5	1990	1	117	1

Après la saisie, exécutez les commandes suivantes :

```
. expand = 2
. expand = 2
. replace ident=ident+5 if _n>15 & _n<31
. replace ident=ident+10 if _n>30 & _n<46
. replace ident=ident+15 if _n>45 & _n<61
. replace taux=taux+round(uniform()*100,1) if _n>15
. save temporel
```

On dispose maintenant d'un fichier fictif « temporel.dta » composé de 20 individus. Supposez que le temps d'exposition soit calculé par la différence entre deux années d'observation consécutives (chaque individu ayant été observé depuis 1989) et que le nombre d'événements soit indiqué par **even** *:*

```
. sort ident date
. by ident: generate expo=cond(_n==1,date-1989,date-date[_n-1])
. xtpois even sexe taux, exposure(expo) i(ident) robust
```

*Notez que le nombre d'observations (60) n'est pas le même que le nombre de groupes (20) qui représentent en fait les individus : chaque individu est représenté par plusieurs observations qui sont liées par un identifiant (**ident**) propre à chaque individu. Dans notre exemple, chaque observation correspond à une période traversée par l'individu. Cependant, les résultats s'interprètent de la même façon que pour la commande **poisson** : seule la durée d'exposition est prise en compte et non la succession des différentes observations.*

7.3.2 Les séries temporelles auto-corrélées

Une variable enregistrée à intervalles réguliers sur une longue période de temps constitue une série temporelle. Généralement on veut expliquer l'allure de cette série par d'autres séries, qui constitueront autant de variables indépendantes. Ces variables peuvent être soit continues, soit discrètes (dichotomiques ou polytomiques).

La différence avec une régression classique est que les variables (dépendante et indépendantes) sont ordonnées dans le temps, en principe à intervalles réguliers. Dans ce cas, il peut arriver que l'observation à une date donnée ne soit pas indépendante de l'observation à la date antérieure, invalidant ainsi une des hypothèses de la régression classique : on parle alors d'autocorrélation. Une part importante de l'analyse consiste alors à tester l'existence d'une autocorrélation, et le cas échéant, à corriger en conséquence les résultats de la régression.

Le test d'autocorrélation le plus connu est celui de Durbin-Watson, que l'on obtient avec la commande suivante en même temps que le résultat de la régression classique :

regdw *vardep varindep*
[**, t(***vartemps***) force**]

Il vaut mieux définir la variable de temps avec l'option **t(***vartemps***)** plutôt que de laisser la commande prendre par défaut l'ordre des observations dans le fichier comme variable de temps. La commande **regdw** fournit exactement le même résultat que la commande **regress**, si ce n'est qu'elle le complète par le test de Durbin-Watson sur la corrélation, après régression, entre les résidus pour deux observations consécutives dans la série. Ce test s'interprète de la même façon qu'un test sur la loi normale (significatif au seuil de 5% lorsqu'il est supérieur à 1,96). L'option **force** est utilisée lorsque les intervalles de temps ne sont pas égaux, et que l'on veut néanmoins obtenir des résultats en ne considérant que l'ordre de la série (et non les intervalles de temps exacts).

Pour obtenir des résultats corrigés par la procédure de Cochrane-Orcutt, on exécute :

corc *vardep varindep*
[**, t(***vartemps***) force**]

Les coefficients de régression ainsi calculés tiennent compte de l'autocorrélation : le test de Durbin-Watson est calculé avant et après correction. D'autres procédures font le même travail mais sont plus adaptées à de petits échantillons :

prais *vardep varindep*
[**, t(***vartemps***) force**]

ou bien :

hlu *vardep varindep*
[**, t(***vartemps***) force**]

Le choix de l'une ou l'autre procédure est une affaire de haute théorie statistique que nous éviterons allègrement d'aborder ici. Les commandes **corc** et **prais** tendent à sous-estimer l'autocorrélation pour les petits échantillons : dans ce cas, la commande **hlu** est en principe plus efficiente.

Enfin, une dernière procédure permet de faire une estimation robuste de la variance des coefficients pour les séries temporelles. À la différence des commandes précédentes, elle ne corrige pas la valeur des coefficients de régression en fonction

de l'autocorrélation. Elle corrige seulement l'erreur type en tenant compte à la fois de l'hétéroscédasticité (voir l'option **robust** de la régression simple) et de l'autocorrélation sur un ou plusieurs intervalles de temps :

newey *vardep varindep* , **lag(#)**
[**t(***vartemps***) force**]

L'option **lag()**est obligatoire : lorsqu'elle est fixée à 1, l'autocorrélation est calculée par rapport à l'observation précédente ; fixée à 2, l'autocorrélation est calculée par rapport aux deux observations précédentes, etc.

Reprenons le fichier « temporel.dta » créé plus haut, mais cette fois-ci en considérant que chaque observation représente un individu :

```
. generate n=_n
```

Comparez les résultats des deux commandes suivantes :

```
. regress taux sexe
```

```
. regdw taux sexe, t(n)
```

On constate que les coefficients sont les mêmes : dans les deux cas il s'agit d'une régression simple. La deuxième commande indique en plus que le test de Durbin-Watson est significatif à 9,5%. En effet :

```
. display round((1-normprob(1.668804))*2*100,0.1)
```

On peut obtenir des intervalles de confiance des coefficients de régression qui tiennent compte d'une éventuelle autocorrélation sur des intervalles de temps d'une unité :

```
. newey taux sexe, lag(1) t(n)
```

Pour corriger les coefficients de régression en tenant compte de l'autocorrélation, plusieurs procédures sont disponibles :

```
. corc taux sexe, t(n)
```

```
. prais taux sexe, t(n)
```

```
. hlu taux sexe, t(n)
```

Remarquez les différences d'estimation des coefficients. Le test de Durbin-Watson devient significatif après correction pour l'autocorrélation par la commande **prais**, *et encore plus par la commande* **hlu**.

7.3.3 Les données issues de panels

Dans la section qui précède, une seule série était en jeu, un peu comme si l'échantillon n'était constitué que d'un seul individu. Dans la présente section, à chaque individu correspond une série temporelle, mais le questionnement est toujours le même : y a-t-il une relation entre une variable dépendante et une ou plusieurs variables indépendantes ? Ce type de données est généralement recueilli par panels, c'est-à-dire par des observations répétées au cours du temps sur un échantillon en principe représentatif. Ces données sont aussi dites « longitudinales » , par opposition aux données « transversales » recueillies en une seule fois. Elles sont aussi appelées « séries temporelles », quoique nous préférons réserver ce terme aux données agrégées (dans des annuaires statistiques, par exemple) plutôt qu'aux données d'enquêtes.

La première chose à faire avant d'exécuter un calcul de régression est de voir comment est organisé le fichier, notamment par rapport au temps :

xtdes [**, patterns(**#**) i(***ident***) t(***vartemps***)**]

L'identifiant qui permet de lier les observations entre elles pour un même individu est indiqué par l'option **i(***ident***)** et la variable de temps par l'option **t(***vartemps***)**. La commande **xtdes** décrit les configurations présentes dans l'échantillon : les données ne sont pas forcément disponibles pour tous les individus et pour chaque durée. Par défaut, la commande **xtdes** décrit neuf configurations (en indiquant, le cas échéant, s'il y en a plus de neuf), mais l'option **patterns(#)** permet d'augmenter le nombre de configurations décrites.

Exécuter la commande **xtdes** *est très utile en début de session de travail. Une fois exécutée, il n'est plus nécessaire de redéfinir l'identifiant de l'individu et la variable de temps pour les autres commandes commençant par* **xt**. *De plus, les paramètres définis par* **xtdes** *sont conservés sur le fichier après la sauvegarde, et il n'est donc plus nécessaire d'exécuter cette commande pour chaque nouvelle session, à moins d'avoir à redéfinir la variable de temps et l'identifiant des individus.*

La commande **xtgee** est certainement la commande la plus complète pour traiter les données longitudinales. Mais elle est aussi très complexe. Dans ce manuel, nous n'aborderons que les commandes les plus courantes pour traiter ces données : ces commandes sont en général des versions simplifiées de la commande **xtgee**. (qui signifie : *cross-sectional time-series using generalized estimating equation*). Leur syntaxe est plus simple et elles sont donc une première étape sur le parcours initiatique qui mène à la maîtrise de l'analyse des séries temporelles.

Moins complexe que **xtgee**, la commande **xtreg** (*cross-sectional time-series linear regression*) permet déjà de tester de nombreuses composantes de la variance grâce à ses options :

xtreg *vardep varindep*
[**, be fe re**]

Sans rentrer dans les détails de l'estimation statistique, rappelons que pour des données longitudinales, la variance se décompose en deux éléments : la variance entre les individus, et la variance entre les observations au cours du temps pour un même individu.

La commande estime une régression en introduisant dans le modèle une constante représentant la variation moyenne de la variable dépendante au cours du temps pour chaque individu. Par exemple, si la variable étudiée est le taux de cholestérol, on introduit dans la régression la variation moyenne de ce taux tout au long du suivi du patient. Les variantes du modèle dépendent de la façon de considérer cette variation moyenne.

Le modèle le plus simple consiste à traiter la variance moyenne comme fixe (option **fe** pour *fixed-effects estimator*, appelé encore *within estimator*), c'est-à-dire connue avec précision pour chaque individu. Au contraire, si cette moyenne est considérée comme une estimation (provenant d'une distribution aléatoire, et donc susceptible d'erreur), on utilise l'option **be** (pour *between estimator*). Dans ce cas, on doit faire l'hypothèse forte d'absence de corrélation entre la variation moyenne de la variable dépendante (pour un même individu au cours du temps) et les coefficients de la régression. Cette hypothèse est présente aussi

avec l'option **re** (pour *random-effects estimator*) mais de plus on fait l'hypothèse que la corrélation moyenne pour l'ensemble de l'échantillon peut s'appliquer de manière identique à chaque individu (il n'y a pas de variation d'un individu à l'autre). La corrélation entre les observations successives pour un même individu est censée être la même à la fois pour chaque individu et pour tous les individus de l'échantillon.

Le modèle à effets fixes (option **fe**) est focalisé sur l'effet des variables indépendantes qui varient au cours du temps pour un même individu (d'où le nom de *within estimator*, qui sous-entend une estimation des variations internes à chaque individu). L'effet des autres variables (constantes) n'est pas calculé.

Les deux autres modèles (option **be** et **re**), au prix de l'hypothèse d'absence de corrélation dans le temps, calculent les effets de tous les types de variables (constante pour chaque individu ou variant au cours du temps). On utilise généralement l'option **be** pour calculer la composante de la variance due aux différences entre individus, après avoir calculé la variance totale (inter- et intra-individuelle) dans le modèle plus général (option **re**).

Une variante des modèles précédents est obtenue par la commande **xtgls**. Son intérêt réside dans la possibilité d'obtenir des estimations qui tiennent compte de l'autocorrélation entre deux observations consécutives pour un même individu, ainsi que de la corrélation ou de l'hétéroscédasticité entre individus. Pour exécuter cette commande, il faut disposer au moins d'autant de périodes que d'individus dans l'échantillon. Sa syntaxe de base est :

xtgls *vardep varindep*
[**, panels(***typerreur***) corr(***corr***) force**]

On peut spécifier la structure des erreurs entre les différents individus (ou panels) : par défaut, elle est supposée homoscédastique (constante à travers les individus), mais il est utile de tester l'hétéroscédasticité avec l'option **panels(heteroscedastic)** ou plus simplement **p(h)**. Si de plus, on soupçonne une corrélation entre les individus, on peut spécifier **p(c)**.

Par ailleurs, l'autocorrélation entre deux observations consécutives peut être testée avec l'option **corr(ar1)** ou **c(a)** pour une autocorrélation globale ou bien avec l'option **corr(psar1)** ou **c(p)** pour une autocorrélation spécifique à chaque individu (*panel-specific*).

On remarquera qu'avec l'option **p(h)** ou **p(c)** et l'option **c(a)** ou **c(p)**, on est en présence d'un modèle très proche du modèle calculé par la commande **newey..., lag(1)** pour un seul panel.

Reprenons notre fichier « temporel.dta » en rapportant chaque panel aux individus correspondants. La commande **xtdes** *permet à la fois de décrire les données et de définir une fois pour toute la variable temps et l'identifiant des individus :*

```
. xtdes, i(ident) t(date)
```

Nous n'avons ici que 5 configurations possibles de dates. Tous les individus n'ont pas été observés à chaque date depuis 1989. Quel est l'indicateur du nombre total d'années couvertes par l'enquête ? Commençons par tester l'effet des variations dans le temps :

```
. xtreg taux sexe, fe
```

Ce modèle présente-t-il un intérêt ? Que signifient les valeurs du R-sq *pour la variation intra-, inter-individuelle et totale ? Ce modèle est mal adapté parce que nous n'avons pas de variable indépendante qui varie dans le temps : il ne peut y avoir de mesure de la variation intra-individuelle (*within panels*) et le modèle stipule de plus que l'effet des constantes n'est pas calculé. Pour obtenir une estimation de la différence selon le sexe, on préférera les autres modèles :*

```
. xtreg taux sexe, re

. xtreg taux sexe, be
```

Comment interprétez-vous les valeurs du R-sq *? Que concluez-vous sur l'effet du sexe ?*
Pour tenir compte de l'autocorrélation entre observations consécutives et de l'héréroscédasticité entre individus, on exécutera :

```
. xtgls taux sexe, p(h) c(a) force
```

Le nombre de périodes est-il suffisant pour ce modèle ? Pourquoi a-t-on utilisé l'option **force** *? Quelle est la signification de la note qui apparaît au début des résultats (voir aussi le nombre d'observations) ?*

7.4 Les modèles à variable continue par intervalles

Supposez que vous cherchiez à mesurer de quoi dépend le revenu, mais que dans votre échantillon, certaines personnes bien payées hésitent à déclarer leur revenu réel. Dans ce cas, il est préférable de mentionner pour ces personnes un revenu supérieur à une valeur x (à définir) plutôt que de se risquer à mentionner la déclaration biaisée des revenus de ces riches enquêtés.

Le modèle à variable continue par intervalles a pour but de répondre à ce type de problème. Il n'est d'ailleurs pas restreint aux intervalles ouverts à droite (dans les valeurs supérieures) et peut être généralisé à tous types d'intervalles (à gauche, à droite ou bien tout au long de la distribution). Dans tous les cas, il fait l'hypothèse que la distribution de l'erreur type de la variable dépendante suit une loi normale pour l'ensemble des observations : il faut souligner que cette hypothèse n'est pas forcément vérifiée, et que les cas d'intervalles ouverts sont fréquemment associés à une forte hétéroscédasticité.

La forme la plus générale de ce modèle est fournie par la commande **intreg**, dont la syntaxe de base est :

intreg *varinf varsup varindep*
[, **robust** **cluster**(*var*)]

Cette commande nécessite de définir les limites de chaque intervalle. Dans le cas où par exemple, les revenus ne seraient connus que par intervalles (pour ménager les susceptibilités ou pour simplifier l'enquête, on demande souvent aux enquêtés de se situer dans une classe de revenu, par exemple entre 5000 et 6499 FF, entre 6500 et 7999, etc.), on a besoin des bornes inférieures et supérieures pour chaque individu. Pour les intervalles ouverts à gauche (valeurs inférieures à un seuil), la borne inférieure est codée manquante (« **.** »), tout comme la borne supérieure dans le cas d'intervalles ouverts à droite. Lorsque les deux bornes ont la même valeur, cela signifie que la variable dépendante est connue précisément.

Pour limiter l'effet de l'hypothèse d'homoscédasticité, on peut spécifier les options **robust** et **cluster**, de la même manière que pour la régression classique (voir nos commentaires plus haut).

Lorsque la variable dépendante est connue sur une échelle ouverte à droite et/ou à gauche, mais que les données entre les bornes inférieure et supérieure sont connues avec précision, la commande **cnreg** (pour *censored-normal regression*) est plus adaptée :

cnreg *vardep varindep* , **censored(***varcens***)**

Il n'est nécessaire de définir qu'une seule variable dépendante, et l'option **censored(***varcens***)** indique si la valeur de la variable dépendante représente la borne inférieure de l'intervalle (*varcens*==**-1**), la borne supérieure (*varcens*==**+1**) ou bien si la valeur est connue précisément (*varcens*==**0**).

Une variante de ce modèle consiste à fixer pour toutes les observations les mêmes bornes supérieure et/ou inférieure :

tobit *vardep varindep*, **ll**[**(#)**] **ul**[**(#)**]

Si les limites ne figurent pas entre parenthèses pour les options **ll** (*lower limit*) et **ul** (*upper limit*), cela signifie que les valeurs minimum et maximum de la variable dépendante rencontrées dans le fichier sont considérées comme les limites effectives. On peut spécifier une seule limite, **ll** ou **ul**, si l'autre est connue pour chaque individu. Pour éviter les erreurs, et pour faciliter la lecture des résultats il est préférable de toujours mentionner explicitement les limites.

Notez, pour les commandes **tobit** et **cnreg**, l'absence des options **robust** et **cluster** : même si la commande **intreg** paraît à première vue plus complexe (du fait qu'elle nécessite la création de deux variables indiquant les bornes des intervalles), il est préférable de faire cet effort pour obtenir des estimations moins sensibles à l'hétéroscédasticité, grâce aux options **robust** et **cluster**.

Supposons que dans le fichier « temporel.dta », chaque observation représente un individu et que **taux** *soit la variable dépendante. Si les valeurs supérieures à 200 sont considérées comme douteuses, on pourra appliquer le modèle* **tobit** *et comparer les résultats avec un modèle de régression simple :*

```
. regress taux sexe

. tobit taux sexe, ul(200)
```

Supposons maintenant que la variable **even** *indique un intervalle ouvert à droite (***even==1***). Dans ce cas, l'intervalle peut être différent pour chaque individu, et la commande* **cnreg** *sera mieux adaptée :*

```
. cnreg taux sexe, censored(even)
```

Notre petit fichier de données fictives ne contient pas des limites inférieure et supérieure pour la variable **taux**, *mais en guise d'exercice vous pouvez créer artificiellement de tels intervalles pour chaque observation en considérant la variable* **taux** *comme la limite inférieure et en y ajoutant quelques unités aléatoirement :*

```
. generate tauxinf=taux

. generate tauxsup=taux+round(10*uniform(),1)
```

Pour agrémenter le problème, remplacez par des valeurs manquantes certaines limites inférieure et supérieure :

```
. replace tauxinf=. if _n<5

. replace tauxsup=. if _n>46
```

Le modèle correspondant à ces données serait alors :

```
. intreg tauxinf tauxsup sexe, robust
```

Remarquez l'utilisation de l'option **robust**.

Dans chacune des régressions obtenues, en dessous des coefficients, figure un indice **_se** *ou bien* **_sigma** *qui correspond à l'écart type de la variable dépendante après régression sous l'hypothèse d'une distribution normale. Une erreur type est associée à cet indice ainsi qu'un degré de significativité et un intervalle de confiance.*

7.5 Les modèles à variable dichotomique non datée

L'idée à la base des modèles logistique ou probit est de faire l'analyse non pas sur une variable dépendante continue qui peut prendre une infinité de valeurs, comme dans les modèles précédents, mais sur une variable dichotomique, codée 1 ou 0 selon que l'individu a ou n'a pas la caractéristique étudiée.

Ces modèles s'appliquent aux situations où certains individus ont une caractéristique que d'autres n'ont pas. La modélisation a pour objectif de déterminer les facteurs qui peuvent expliquer la présence de cette caractéristique. En principe, le temps est considéré comme nul ou inopérant : comme dans le modèle de régression simple, le moment de l'observation est supposé unique (voir plus loin les modèles qui s'inspirent du modèle logistique tout en tenant compte du temps).

7.5.1 Les modèles logistique et probit

Dans le modèle logistique (ou logit, nous verrons que ces deux termes sont équivalents), le ratio qui est modélisé est constitué du rapport entre la population qui a la caractéristique étudiée et celle qui ne l'a pas. Ce n'est donc pas exactement une proportion (rapport entre la population qui a la caractéristique étudiée et la population totale), mais les deux ratios sont proches lorsque la caractéristique étudiée est rare.

La différence entre les modèles logistique et probit réside dans la forme de la fonction qui lie la variable dépendante aux variables indépendantes : dans le modèle logistique, il s'agit de la distribution logistique cumulée, tandis que dans le modèle probit, il s'agit de la distribution normale cumulée. La conséquence se situe essentiellement au niveau de la distribution des variations aléatoires (l'erreur type).

Pour simplifier l'exposé, nous ne traiterons ici que du modèle logistique, en gardant à l'esprit que les procédures sont presque les mêmes pour le modèle probit. Dans le tableau ci-dessous figurent la correspondance entre les différentes commandes pour les deux types de modèle.

Modèle logistique	Modèle probit
logit	**probit**
logistic	**dprobit**
blogit	**bprobit**
glogit	**gprobit**
svylogit	**svyprobit**

La syntaxe de base du modèle logistique est :

logistic *vardep varindep*
[, **robust** **cluster**(*var*)]

Cette commande fournit les résultats sous forme de « rapports de chances » (*odds ratios*) : à chaque changement d'unité d'une variable indépendante correspond un coefficient multiplicatif, qui varie de 0 à plus l'infini et qui est centré sur la valeur 1, correspondant à un effet nul. Par exemple, à un rapport de chance calculé de 1,5 correspond une multiplication de la probabilité par 1,5. À un rapport de 0,5 correspond une réduction de 50% de la probabilité, ou bien, si l'on préfère, une division de la probabilité par 2.

Pour obtenir des coefficients sous forme additive, dont l'interprétation est moins immédiate mais qui varient symétriquement (autour de 0) de plus à moins l'infini, on exécutera la commande **logit**. Les deux commandes sont absolument équivalentes (c'est le même modèle qui est estimé) mais **logistic** est associée à des commandes de diagnostic de la régression qui ne sont pas disponibles après **logit**.

Comme pour la régression classique, les options **robust** et **cluster** permettent de tenir compte d'une éventuelle hétéroscédasticité.

En principe le modèle logistique est adapté à l'analyse des caractéristiques rares. Il arrive fréquemment que l'estimation ne converge pas lorsqu'une variable indépendante dichotomique ou bien une catégorie d'une variable indépendante polytomique définit un sous-ensemble d'individus dont aucun n'a la caractéristique étudiée (vardep==0). Dans ce cas une note apparaît avant les résultats de la régression pour dire que « nomvar~=0 predicts failure perfectly *». Les individus concernés sont alors éliminés de l'analyse ainsi que la variable indépendante qui correspond. Il en est de même si tous les individus d'une catégorie ont la caractéristique étudiée (vardep==1), et le message « nomvar~=0* predicts success perfectly *» apparaît.*

On assiste là à un véritable paradoxe : les individus pour lesquels le modèle s'ajuste parfaitement sont éliminés des calculs. Cela provient du fait que dans ce cas les coefficients pour ces individus pour le modèle logistique tendent vers 0 ou plus l'infini (ou bien vers plus ou moins l'infini pour le modèle sous forme logit).

Une astuce pour obtenir des estimations sur l'ensemble de l'échantillon est d'utiliser la commande **mlogit** *en y introduisant une contrainte de convergence : en effet cette commande (qui est décrite dans une section ultérieure) n'élimine pas les individus pour lesquels la prédiction est parfaite, et calcule les coefficients tendant vers plus ou moins l'infini (modèle additif) dont le degré de significativité n'est pas interprétable (l'erreur type tendant aussi vers l'infini). Pour les autres coefficients, les résultats sont les mêmes qu'avec la commande* **logit**, *mais l'avantage est que l'on peut estimer le gain de vraisemblance véritable compte tenu des prédictions parfaites.*

Dans le fichier « temporel.dta », créons une catégorie **bidon** *pour laquelle la variable dépendante* **even** *est toujours codée 1, et comparons les résultats :*

```
. generate bidon=cond(_n<16,even,0)

. logit even bidon sexe taux

. mlogit even bidon sexe taux, ltolerance(.000001)
```

Le gain de vraisemblance (Pseudo R^2) a presque doublé entre les deux estimations. Remarquez l'utilisation de l'option **ltolerance(#)** *ainsi que les valeurs du coefficient pour* **bidon**, *de son erreur type et de son degré de significativité.*

Une variante du modèle logistique de base permet de traiter les données groupées lorsque les données individuelles ne sont pas disponibles. Le cas se présente lorsqu'on connaît le nombre d'individus ayant la caractéristique étudiée ou non, classés par groupe selon les catégories des variables indépendantes, mais qu'on ne dispose pas d'un fichier où figurent pour chaque individu les valeurs des variables dépendante et indépendantes. Dans le fichier groupé, chaque observation représente plusieurs individus et correspond à une catégorie de l'échantillon définie par le croisement des variables indépendantes.

La syntaxe de base de la commande **blogit** est :

blogit *nbreponse poptotale*
[, **robust cluster(***var***) or**]

La syntaxe de **glogit** est la même excepté que les estimations robustes ne sont pas possibles. Le maximum de vraisemblance (*maximum-likelihood*) est la procédure d'estimation utilisée pour **blogit** (comme pour **logit** ou **logistic**) alors que **glogit** utilise la méthode des moindres carrés pondérés (*weighted least squares*). Le modèle **blogit** s'applique aux données issues d'un échantillon non pondéré, tandis que le modèle **glogit** est mieux adapté aux données exhaustives (de type recensement).

Enfin, la commande **svylogit** permet de traiter les données issues d'échantillonnage par grappes, stratifiées ou non (voir la section sur les pondérations dans le chapitre sur *Les statistiques simples et leurs représentations graphiques*).

Supposons que dans le fichier « temporel.dta », chaque observation représente un individu et que **even** *soit la variable dépendante. On peut alors appliquer un modèle logistique. Comparez les résultats obtenus avec les différentes commandes :*

```
. logit even sexe taux
. logit even sexe taux, robust
. logit even sexe taux, robust or
. logistic even sexe taux, robust
```

On peut appliquer le modèle logistique sur les données groupées du fichier « census.dta ». Comparez les résultats suivants :

```
. nbreg death pct1t5 pct5_17 pct65p pcturb Iregio*, exposure(pop)

. blogit death pop pct1t5 pct5_17 pct65p pcturb Iregio*

. glogit death pop pct1t5 pct5_17 pct65p pcturb Iregio*
```

Lequel de ces modèles est le plus adapté aux données du recensement ? Faites le même exercice avec les commandes **bprobit** *et* **gprobit**.

7.5.2 Le modèle conditionnel logistique appliqué aux données assorties (échantillon cas-témoin)

En biologie ou en épidémiologie (et parfois dans l'ingénierie), il arrive fréquemment de mesurer l'effet d'un produit ou d'un traitement en comparant des individus couplés par paires selon des caractéristiques communes et connues à l'avance. En d'autres termes, la variable dépendante distingue les cas (codés 1) et les témoins (0) au sein de groupes ayant des caractéristiques communes, et on cherche à connaître l'effet d'autres variables (indépendantes) au sein de ces groupes. La régression est dite conditionnelle (*conditional logistic regression*) parce que l'estimation est conditionnée par l'appartenance à un groupe. On ne s'intéresse pas à la variance entre les groupes (*between*) mais au sein des groupes (*within*).

Cette méthode a été généralisée de telle façon à ce que les cas et les témoins ne soient pas seulement assortis par paires (*matched case-control*) : un cas peut être assorti à plusieurs témoins et inversement, de sorte que toutes les configurations sont possibles entre le nombre de cas et le nombre de témoins.

La syntaxe de base de **clogit** est :

clogit *vardep varindep* , **group**(*ident*) [**or**]

Les résultats sont produits sous forme additive. La forme multiplicative (soit une présentation du même type que la commande **logistic**) est obtenue avec l'option **or** (*odds ratio*).

Créons dans le fichier « temporel.dta », une variable fictive indiquant si une observation est un cas ou un témoin, au sein de chaque groupe défini par la variable **ident**, *et appliquons le modèle* **clogit** :

```
. sort ident

. by ident: generate cas=round(uniform(),1)

. clogit cas expo sexe taux, group(ident)
```

La première note indique que les groupes ne sont pas uniquement constitués de paires. La deuxième note indique que certains groupes ont été éliminés de l'analyse car une seule catégorie (soit des cas, soit des témoins) est représentée dans ces groupes : il faut en effet au minimum une paire par groupe pour que l'analyse ait un sens. Enfin la variable sexe a été éliminée car chaque groupe est en fait constitué de personnes de même sexe : l'estimation porte sur la variance au sein des groupes (pour des variables indépendantes variant entre les individus d'un même groupe) et non entre les groupes.
Sur combien d'individus porte l'analyse de régression ? Pourquoi tout l'échantillon n'est-il pas analysé ?

7.6 *Les modèles à variable dichotomique datée*

7.6.1 Les modèles logistiques et probit appliqués aux données longitudinales

Les modèles logistique et probit s'appliquent aux données non datées, c'est-à-dire où chaque observation du fichier représente un individu. Une généralisation consiste à considérer plusieurs observations par individu (par exemple, plusieurs expérimentations d'un produit, plusieurs diagnostics, ou une série d'observations annuelles d'un même échantillon). Ce type de modèle est généralement appliqué aux données longitudinales, issues de panels. Rappelons cependant que les observations sur un même individu ne sont pas nécessairement ordonnées : il ne s'agit pas toujours de séries temporelles.

L'équivalent de la commande **logit** pour les données longitudinales est :

xtgee *vardep varindep* , **family(binomial)**
[**link(logit/probit)** **corr(***corr***)**
force robust eform]

Comme on l'a dit plus haut dans la section sur *Les données issues de panels*, il est préférable d'exécuter **xtdes** avant une commande de type **xt**, pour définir l'identifiant liant les observations entre elles pour un même individu, et la variable de temps.

En spécifiant l'option **family(binomial)**, le lien par défaut est de type logistique. L'option **link** n'est donc utile que pour spécifier un lien de type probit. Stata dispose d'une commande spécifique pour ce type de modèle, **xtprobit**, mais étant donné que cette commande fait appel aux mêmes options que **xtgee**, nous vous conseillons plutôt d'utiliser cette dernière, qui vous permettra de passer plus rapidement d'une variante du modèle à une autre.

L'option **corr** permet de définir jusqu'à sept types de structure de corrélation entre panels, c'est-à-dire entre les observations successives au cours du temps (*within correlation*). La corrélation par défaut est de type **exchangeable**. Dans ce cas une corrélation moyenne est appliquée pour tous les individus : la corrélation entre les observations pour un même individu est supposée constante, et cette corrélation est la même pour tous les individus de l'échantillon. Cette option est équivalente à l'option **re** (*random-effects estimator*) que nous avons vu plus haut dans la section sur *Les données issues de panels*. À l'opposé, l'option **corr(unstructured)** permet de ne poser aucune contrainte sur la corrélation entre observations successives au cours du temps (entre chaque panel).

Les autres options posent des contraintes intermédiaires. La plus intéressante sans doute correspond au modèle autorégressif AR(n) où les observations d'un panel sont supposées corrélées aux observations du ou des panels précédents. Les options **corr(ar1)** pour les commandes **xtgls** et **xtgee** sont équivalentes, mais l'autocorrélation n'est pas restreinte à deux panels consécutifs pour **xtgee** : dans le cas où n>1 l'autocorrélation est supposée régresser avec la valeur de n.

Une variante du modèle autorégressif est donnée par l'option **corr(stationary** *n*). Dans ce cas, l'autocorrélation entre un panel et les n précédents est constante, quel que soit le

nombre *n*. Avec l'option **corr(nonstationary** *n*), une autocorrélation spécifique est calculée pour chaque couple de panels formé par le panel courant et les *n* précédents.

L'option **corr(independent)** revient à considérer que les observations ne sont pas regroupées par individus, c'est-à-dire qu'elles ne sont plus issues de panels : la commande est alors équivalente à **logit** ou **probit**. Enfin, l'utilisateur peut spécifier sa propre matrice de corrélation (et non une matrice calculée à partir des données) avec l'option **corr(fixed** *matrice*) : cette option est rarement utilisée.

Il n'est pas très aisé de choisir *a priori* une structure de corrélation entre panels successifs. En pratique, on étudiera d'abord la structure de la matrice de corrélation avec l'option **corr(unstructured)**, pour voir si les données ne présentent pas une structure particulière dont on pourrait rendre compte avec une autre option de corrélation plus contraignante et plus simple à la fois.

L'option **robust** a été expliquée plus haut dans la section sur *Les estimations robustes de la variance*. Ajoutons que dans le cas de la commande **xtgee**, l'option **cluster(***ident***)** est implicite : une grappe correspond à un ensemble d'observations consécutives (panels) pour chaque individu.

L'option **force** est utilisée lorsque les intervalles de temps ne sont pas égaux, et que l'on veut néanmoins obtenir des résultats en ne considérant que l'ordre de la série (et non les intervalles de temps exacts).

Enfin, l'option **eform** permet d'obtenir les coefficients sous forme multiplicative (comme pour les commandes **logistic** ou **dprobit**), alors que sans cette option les résultats sont présentés sous forme additive (commandes **logit** et **probit**).

Reprenons pour exemple le fichier « temporel.dta », avec **even** *pour variable dépendante . Tous les individus n'ont pas été observés aux mêmes dates, de sorte que l'on est contraint d'utiliser l'option* **force**. *Le nombre d'individus est trop petit (et trop artificiel) pour tester toutes les options de corrélation, mais on peut tout de même comparer les résultats avec une corrélation moyenne (identique entre panels pour tous les individus) et avec une corrélation non structurée entre panels :*

```
. xtgee even sexe taux, family(bin) corr(exchang) force robust

. xtcorr

. xtgee even sexe taux, family(bin) corr(unstr) force robust

. xtcorr
```

La commande **xtcorr** *fournit la matrice de corrélation entre panels : l'option* **unstructured** *montre-t-elle une structure de corrélation particulière ? L'option de corrélation par défaut* **exchangeable** *est-elle raisonnable ?*

Comparez maintenant les résultats des deux commandes suivantes :

```
. logit even sexe taux, robust cluster(ident)

. xtgee even sexe taux, family(bin) corr(indep) force robust
```

Que vous inspirent ces résultats ? Que signifie l'option de corrélation **independent** *?*

7.6.2 Les modèles de survie

L'analyse de survie prend son origine en épidémiologie mais elle s'est étendue depuis à d'autres disciplines : le terme anglais *survival analysis* qui lui correspond le mieux, est de plus en plus remplacé par *event history analysis* pour bien signifier que ce type d'analyse s'applique aussi à des événements moins définitifs que le décès. En français, les termes d'analyse des biographies et d'analyse des transitions sont les plus courants.

Par souci de conformité avec le manuel de référence de Stata qui classe les modèles présentés dans cette section dans les rubriques **[R] st** (comme *survival time*), nous conserverons le terme générique de modèle de survie. Mais, le terme qui décrit le mieux leurs aspects techniques est celui de modèles à risques proportionnels (*proportional hazard models*). Mis au point la

première fois par D. R. Cox en 1972 (d'où l'appellation fréquente de « modèle de Cox »), ces modèles font l'hypothèse que l'effet des variables indépendantes est proportionnel à la probabilité de connaître l'événement dans chaque intervalle de temps.

Le modèle de survie se rapproche du modèle logistique sur les données longitudinales, par panels : ils traitent tous deux une variable dépendante dichotomique. En revanche, ils diffèrent fondamentalement l'un de l'autre dans leur façon de traiter le temps.

En effet, les modèles de type logistique ne considèrent pas explicitement la succession des panels dans le temps. Le temps n'intervient éventuellement que dans la structure de la corrélation entre périodes successives : c'est le cas par exemple avec les options de corrélation **ar**, **stationary** ou **nonstationary** où la corrélation est limitée à un nombre fixe de périodes successives. Pour les autres options de corrélation, le temps n'est pas pris en compte.

Dans les modèles de survie, au contraire, le temps écoulé entre la première et la dernière période fait partie intégrante de l'estimation. Le risque est calculé pour chaque unité de temps et non globalement pour l'ensemble des observations.

Ces modèles prennent explicitement en compte l'interruption du temps d'observation du fait de l'enquête, ou bien du fait de la déperdition de certains individus, qu'on appelle une « troncature à droite » (*right censoring*). Les troncatures à gauche (les entrées tardives en observation - *left censoring*) et même les interruptions en cours d'observation (*time gaps*) peuvent aussi être prises en compte. De plus, les variables indépendantes peuvent varier dans le temps (*time-varying variables*), et les sujets, après avoir connu une première fois l'événement, peuvent entrer à nouveau dans la population soumise au risque et ainsi connaître plusieurs événements successifs jusqu'à la date d'enquête : on parle alors d'événement renouvelé (*multiple failure* ou *recurring time failure*). De ces diverses manières, l'information sur le temps d'observation est mieux contrôlée.

Un nombre important de commandes sont disponibles pour déclarer, décrire et traiter les données de survie (*survival time data*). Sans les passer toutes en revue, il est indispensable de montrer comment déclarer les paramètres nécessaires à l'analyse de survie (de la même façon qu'on utilise la commande **xtset** pour les données de panels - *cross-sectional time series*). La syntaxe de base est la suivante :

stset *vartemps indtronc [f/p/iweight]*
, [**id(***ident***) t0(***datedébut***)**]

L'exécution de cette commande nécessite au minimum de connaître la durée d'observation de l'individu jusqu'à l'événement et un indice de troncature, qui par convention sera égal à 1 si l'événement a eu lieu avant la sortie d'observation, et à 0 sinon (troncature à droite).

Dans le cas où plusieurs observations ont été faites sur les mêmes individus, l'option **id** permet de déclarer la variable reliant les observations pour un même individu. Si les périodes correspondant à chacune de ces observations ne se suivent pas (s'il y a des interruptions au cours du temps d'observation - *gaps*), on doit alors déclarer la variable indiquant le début de période avec l'option **t0**.

Les paramètres définis par **stset** sont conservés sur le fichier après la sauvegarde : il n'est pas nécessaire d'exécuter cette commande pour chaque nouvelle session, à moins d'avoir à redéfinir les paramètres. Pour vérifier qu'ils ont été bien définis, et pour décrire le fichier, on exécutera la commande **stdes**.

Plusieurs modèles de survie (à risques proportionnels) sont disponibles dans Stata. Le modèle classique le plus courant est obtenu avec la commande **stcox**, dont la syntaxe de base est :

stcox *varindep*
[, **nohr strata(***var***) robust cluster(***var***)**]

Ce modèle a une composante non paramétrique, qui correspond à la fonction de survie : les quotients instantanés de survie - *hazard ratios* - sont calculés indépendamment pour chaque unité de temps, de sorte qu'aucune forme n'est imposée à la distribution de ces quotients. La composante paramétrique du

modèle est constituée du vecteur des variables indépendantes qui agissent proportionnellement sur toute la durée d'observation. En raison de cette double composante, le modèle est dit semi-paramétrique.

Pour ajuster la fonction de survie à une fonction de Weibull (où les quotients instantanés de survie sont de forme pt^{p-1} où t représente le temps et p un paramètre estimé à partir des données), on utilise la syntaxe de base :

stweib | **stereg** *varindep*
, [**nohr** **robust** **cluster(***var***)**]

Les commandes **stweib** et **stereg** diffèrent simplement en ce que **stereg** est un cas particulier de **stweib** où p=1. Dans ce cas la fonction de survie est dite exponentielle (quotients instantanés constants au cours du temps). Ces modèles sont dits paramétriques étant donné que les deux composantes du modèle (fonction de survie et variables indépendantes) sont paramétriques.

Le fichier « temporel.dta » est parfaitement adapté à l'analyse de survie. Les observations qui se rapportent à un même individu sont repérées avec la variable **ident**. *La variable* **even** *représente l'indice de troncature, et la variable* **date** *le moment de l'évènement.*
Dans un premier temps, on va considérer que les observations ont eu lieu depuis 1989. Pour déclarer les variables nécessaires à l'analyse de survie, on exécutera :

```
. generate duree=date-1989

. stset duree even, id(ident)

. stdes
```

La commande **stset** *crée automatiquement une nouvelle variable indiquant le moment d'entrée en observation :*

```
. sort ident duree

. by ident: list date t0 duree expo taux
```

Pourquoi **t0** *est-elle toujours égale à 0 pour la première observation pour chaque individu ? À quoi correspond* **t0** *pour les observations suivantes (comparez les variables* **t0**, **duree** *et* **expo**) *? Remarquez les valeurs de* **taux** *: comment appelle-t-on ce type de variable ?*

On pourrait considérer que chaque observation correspond en fait à une durée d'exposition d'un an. Dans ce cas, on sera en présence de troncatures à gauche (puisque tous les individus n'ont pas été observés en 1990) et aussi d'interruptions d'observations (certains individus ne sont pas observés tous les ans). Pour cela on devra définir explicitement la variable de début de période avec l'option **t0** *:*

```
. generate debut=duree-1

. stset duree even, id(ident) t0(debut)

. stdes

. sort ident duree

. by ident: list date debut duree
```

La commande **stdes** *nous confirme bien que tous les individus n'entrent pas en observation à la durée 0 (*first entry time>0*) et qu'il y a des interruptions d'observation (*subjects with gaps*).*

Comparez les résultats des modèles suivants :

```
. xtgee even sexe taux, family(bin) corr(exchang) force robust

. stcox sexe taux, robust nohr

. stweib sexe taux, robust nohr
```

Quel est le modèle qui correspond le mieux à la structure des données ? Dans quel modèle pose-t-on le moins d'hypothèses fortes ?

7.7 *Le modèle à variable polytomique exclusive*

Le modèle multinomial logit (ou logistique polytomique, les deux formulations sont équivalentes) est adapté aux situations où la variable dépendante est polytomique mais n'est pas ordonnée. Les modalités de la variable dépendante ne peuvent être classées selon un ordre de valeur. On sait simplement qu'elles sont exclusives l'une de l'autre, c'est-à-dire que les individus ne peuvent être classés à la fois selon l'une et l'autre modalités. À la différence du modèle logit, ce n'est pas la présence d'une seule caractéristique dont on mesure les chances, mais l'alternative entre plusieurs caractéristiques.

La syntaxe de base de la commande **mlogit** est :

mlogit *vardep varindep*
[, **basecategory(#) rrr**]

Le modèle multinomial logit est un système de *m*-1 équations, où *m* correspond au nombre de modalités de la variable dépendante. Une modalité de référence est choisie en fonction du nombre de cas le plus élevé dans l'échantillon, mais on peut imposer un autre choix avec l'option **basecategory**, en indiquant le numéro de modalité. En changeant de cette manière la catégorie de référence de la variable dépendante, on peut calculer autant de coefficients pour une même variable indépendante qu'il y a de combinaisons de deux modalités de la variable dépendante.

Par exemple, si *m*=6 on peut former jusqu'à $\frac{m!}{2!(m-2)!}=\frac{6\times5\times4\times3\times2}{2\times(4\times3\times2)}=15$ séries de coefficients de régression. On comprend que pour cette raison, les résultats du modèle multinomial logit sont plus difficiles à interpréter que ceux du modèle logit simple. Il faut non seulement raisonner avec plusieurs variables indépendantes (explicatives), elles-mêmes le plus souvent polytomiques, mais il faut aussi ajouter une dimension supplémentaire, puisque la variable dépendante comporte plusieurs modalités.

Ajoutons enfin que l'option **rrr** (*relative risk ratios*) sert à obtenir les résultats sous forme logistique.

*Supposons que l'on veuille déterminer l'appartenance d'un État à une des quatre régions des USA (*m*=4) à partir de certaines de ses caractéristiques contenues dans le fichier « census.dta » :*

```
. mlogit region pop actif urbain
```

Le nombre d'équations est bien de m*-1 c'est-à-dire 3, mais on peut former d'autres combinaisons de coefficients en changeant la catégorie de base :*

```
. mlogit region pop actif urbain, base(4)
```

Comparez les coefficients pour la région Ouest pour le premier modèle aux coefficients pour la région Sud dans le second : pourquoi les coefficients sont-ils symétriques ? Que mesurent les coefficients pour la région Nord-Est dans les deux modèles ? Pourquoi les coefficients de la région Centre-Nord sont-ils significatifs dans un cas et pas dans l'autre ?

Contrairement aux commandes **logit** *ou* **logistic**, *les individus ne sont pas éliminés de l'analyse lorsque la catégorie à laquelle ils correspondent prédit parfaitement le résultat. Par conséquent, comme il est indiqué plus haut dans la note qui suit les explications sur le modèle logistique, il est alors nécessaire d'introduire une contrainte de convergence en limitant le nombre d'itérations : dans ce cas,* **mlogit** *calcule, pour les catégories où la prédiction est parfaite, des coefficients de régression tendant vers plus ou moins l'infini (selon que les chances de succès sont totales ou nulles).*

Dans le fichier « census.dta », créons une catégorie **bidon** *égale à 1 pour certains États de la région Ouest, et analysons les résultats :*

```
. generate bidon=cond(region==4,round(uniform(),1),0)

. tabulate bidon region

. mlogit region actif urbain pop bidon, ltolerance(.00001)

. mlogit region actif urbain pop bidon, ltolerance(.00001) base(4)
```

Le coefficient **bidon** *pour la région Ouest tend vers l'infini. Par contre pour les régions Nord-Est et Centre-Nord, ils tendent vers une valeur nulle : en fait il faut se rappeler que la variable* **bidon** *est codée 0 pour la catégorie de référence, la région Sud. Par conséquent, les coefficients pour les régions Nord-Est et Centre-Nord, pour lesquelles la variable* **bidon** *est aussi codée 0, s"interprètent comme une différence entre risques nuls. On peut vérifier que c'est bien le cas en changeant la catégorie de référence pour la région Ouest :*

```
. mlogit region actif urbain pop bidon, ltolerance(.00001) base(4)
```

7.8 *Les modèles à variable polytomique ordonnée*

Dans les modèles logit ordonnés, la variable dépendante est évaluée par degré, comme par exemple un niveau de satisfaction, allant par paliers de « mauvais » à « excellent ». La variable dépendante est à la fois polytomique et ordonnée, sans toutefois être continue comme dans la régression simple.

La modélisation consiste à ajuster la répartition observée entre les différentes catégories de la variable dépendante, par régression sur un certain nombre de variables indépendantes (explicatives). Cela revient, sur une échelle allant de 0 à 1, à choisir des points de césure (*cut-points*) de façon à définir des intervalles de probabilité : le premier intervalle (entre 0 et le premier point de césure) correspond au premier niveau de l'échelle ordinale, le deuxième intervalle (entre le premier et le deuxième point de césure) au deuxième niveau, etc. Sur une échelle logistique, l'échelle varie en fait de plus à moins l'infini autour de la valeur 0.

La syntaxe de base de la commande est :

ologit *vardep varindep* [, **table**]

La commande **oprobit** pour les estimations sous forme probit a la même syntaxe. L'option **table** permet d'obtenir la distribution empirique qui correspond aux points de césure.

Supposons que dans le fichier « census.dta », nous ne connaissions pas l'âge médian, mais que nous disposions d'une variable ordinale **catage** *indiquant la catégorie d'âge médian où se situe chaque État (voir les exercices de la section sur les* Tableaux croisés à deux variables*) :*

```
. generate catage=recode(medage,27,28,29,30,31,32)
```

On peut alors estimer le modèle suivant :

```
. ologit catage pctact pcturb Iregio*, table
```

Les valeurs de la variable dépendante n'ont pas d'importance pourvu simplement qu'elles indiquent un ordre dans les catégories. Pour s'en convaincre, il suffit de relancer le modèle après avoir modifié la variable ainsi :

```
. replace catage=catage-26
```

Il vaut toujours mieux traiter les données exactes quand on peut le faire. Comparez les résultats précédents à ceux de la régression simple sur la variable originale :

```
. regress medage pctact pcturb Iregio*
```

Index

Autres ouvrages disponibles aux Editions Global Design :

- L'Essentiel de Scientific Word
- L'Essentiel de Mathematica 3
- Introduction à la programmation sous GAUSS

  Vol 1 : Méthodes Numériques en Mathématiques et Statistiques

  Vol 2 : Applications à la Finance et à l'Econométrie

Imprimé par Jouve, 18 rue St Denis,
75001 Paris
N°256816A — Dépôt légal : Avril 1998

L'essentiel de STATA

L'essentiel de Stata s'adresse principalement aux étudiants, chercheurs ou professionnels des sciences biologiques, économiques ou sociales. Il permet au lecteur de débuter rapidement avec Stata. Il explique d'abord la gestion des données, en particulier pour les fichiers hiérarchisés. Outre les tabulations complexes (croisant plus de trois variables), les différentes possibilités graphiques de Stata, en particulier les lissages de courbe, sont explicitées. Par ailleurs, ce manuel montre comment choisir parmi les nombreux modèles de régression proposés par Stata, à partir d'une méthode originale basée sur le type de variable dépendante. Ce manuel est agrémenté d'exercices et ponctué d'astuces pour une meilleure utlisation du logiciel.

L'auteur

Philippe BOCQUIER est Chargé de Recherche à l'Institut français de recherche scientifique pour le développement en coopération (ORSTOM). Il y mène des recherches dans les pays en développement sur les thèmes des migrations, de l'urbanisation, de l'insertion et de la mobilité professionnelle. Il utilise Stata pour l'enseignement et ses recherches. Spécialiste de l'analyse quantitative des biographies, il écrit des procédures pour faciliter ce type d'analyse.

GLOBAL DESIGN

RITME INFORMATIQUE

ISBN 2-911502-04-3
Prix 189 FF