Programmation avec Stata

Marc Thévenin

2023-08-29

Table des matières

Présentation de la formation

> Septembre 2023
>
> La fin de la mise à jour approche, avec le début du dernier chapitre. Penser à vous reporter sur la formation dédiée au graphiques, ces éléments ayant été retiré de ce support (encore quelques élement dans le chapitre 4).

- Ce support n'intéressera peut-être pas les plus aguerri.e.s à Stata…. quoique. On trouve toujours, après des années pratique, de nouvelles commandes qui facilitent vraiment les choses (`sencode`, `fre`, `tabm`,…., commandes du package `gtools`). Un introduction aux **frames** introduites avec la version 16 et qui revisitent les fusions de bases, mais qui ne semble pas encore très utilisé, a été ajouté.
- La formation est téléchargeable en format pdf. Pour accéder à cette version aller sur l'icône pdf à droite de la barre de navigation.
- Si la grande majorité des éléments inclus à ce support sont classiques, on peut se reporter sur l'accueil principal pour trouver des fiches, encore en nombre limités, sur des fonctions que j'ai programmé ou sur des éléments de mise à jour du logiciel.
- Maj de la formation (de la plus ancienne à la plus récente):
  - 28/07/22 : **Les variables**. Chapitre 5 [todo: faire un topo sur les formats date avant fin 2023]
  - 07/10/22 : **Introduction**. Chapitre 1
  - 07/10/22 : **L'environnement**. Chapitre 2
  - 22/02/23 : **Le langage Stata**. Chapitre 3
  - 22/02/23 : **Les bases de données**. Chapitre 4 [introduction aux *frame* introduites avec la v16]
  - 09/08/23 : **Manipulation bases de données** (fusion, transpositions…). Chapitre 6

Les programmes des exemples traités dans les chapitres (à partir du troisième) sont téléchargeables:

- Chapitre3
- Chapitre4
- Chapitre5
- Chapitre6
- Chapitre7

- Support réalisé avec Posit-Rstudio avec l'outil d'édition Quarto.
- Version Stata: 18-SE


100% HUMAN
NO AI
CREATED CONTENT

1 Présentation de Stata

1.1 Historique de Stata

L'entreprise **Stata Corp Lp** a été fondée par William Gould. Il est toujours à la tête de l'entreprise.

- Première version (sous MS Dos) en janvier 1985.
- Première version Windows en 1995 (Stata 4).
- La version la plus récente est la 17 [printemps 2021].
- Le rythme de changement de version est de 2 ans. La version 18 devrait donc sortir au printemps 2023. Attention tout de même, il s'agit plutôt de grosses mises à jour, dont l'acquisition systématique s'avère généralement dispensable.

Version 18 et 17

Version 18

- Une autosave des programmes (enfin)
- Nouveau format .dtas pour enregistrer une liste de frames générées dans un programme
- Graphiques (se reporter à la formation graphique):
  - Nouvelle palette qualitative (enfin)
  - Nouveau thème par défaut (enfin)
  - Une nouvelle option `colorvar` (cf `fill`, `color` dans R ou python) qui permet d'empiler automatiquement ou presque plusieurs objet graphiques (autrefois un objet par valeur d'une condition)

Version 17
Une présentation a été faite il y a plus d'un an [Lien].
En résumé pour cette 17ème version:

- Amélioration de la vitesse d'exécution (je confirme)
- Suite **`collect`** pour gérer et générer les outputs des commandes statistiques. Rapidement testé pour des régressions. C'est plutôt pas mal même si comme d'habitude avec Stata "qui se raccroche aux branche", cela peut vite toutner à l'usine à gaz [rapide présentation]

1.2 Stata à l'Ined

- Version 18 Window SE en cours de déploiement, la version 17 Windows SE a été déployée au printemps 2022.
- Version 17 SE sous Linux (serveur margaux).

Petit rappel pour l'Ined: Quel que soit le logiciel choisi sous environnement Windows (SAS v9.4 ou Stata) vous avez accès à une version de l'autre application sous Linux via le serveur Margaux (SAS Studio ou STATA v17 SE).

Version SE: Standard Edition

- Nombre d'observations: 2.14 milliards
- Nombre de variables dans les bases: 32 767
- Nombre de variables RHS (limite nombre de colonnes des matrices: 10998)

Il existe également des versions dites MP (Multiple Process), dont la plus puissante gère 120000 variables et 20 milliards d'observations).

1.3 Les ressources

1.3.1 Les ressources externes

- Les manuels édités par STATA. Certains sont disponibles au GED [http://www.stata.com/bookstore/books on-stata]
- *__Stata Journal__, la revue trimestrielle éditée par Stata [http://www.stata-journal.com]. Depuis 2015 accès total aux versions PDF (via portail GED maintenant). L'accès et le sommaire des 4 derniers numéro est disponible ici: Lien
- Une chaîne Youtube [http://www.youtube.com/user/statacorp/videos?view=0] qui propose des tutoriels via les manips par les boites de dialogue.
- Un forum particulièrement actif (https://www.statalist.org/)
- Auto-promo: le support que j'ai mis en place à l'été 2022 et qui héberge cette formation introductive Lien

1.3.2 Les ressources internes

- Le manuel de Stata est directement intégré au logiciel. Pour y accéder à partir du menu : `help => PDF documentation`. ***Il est vraiement de très bonne qualité***.
- Pour une instruction, on peut obtenir son aide en tapant dans la fenêtre *command* de l'environnement principal **`help nom_commande`**.

– Pour les commandes internes, un lien permet d'accéder à l'entrée du manuel officiel, plus riche.

– Dans les fenêtre de dialogue, on peut accéder directement au fichier d'aide en cliquant sur le point d'interrogation.

Listing 1.1 `Utiliser l'aide interne`

```
help generate
```

L'aide comprend généralement:

- Le nom de la commande
- Le lien vers l'entrée du manuel si commande officielle. Pour les commandes externes, un lien vers le sitee support peut-être proposé.
- La description de la syntaxe
- Le détail des options
- Des exemples avec éventuellement une exécution directe (voir avec `help tw`)
- Les informations sauvegardées temporairement (jusqu'à l'exécution d'une autre commande) => *stored results*.
- L'instruction `findit nom_command` permet d'obtenir la liste de toutes les ressources disponibles d'une commande, internes ou externes. Exemple: `findit fitstat`

> ⚠ Warning: compatibilité entre les versions
>
> Des problèmes de compatibilité entre les versions de Stata se sont posés. Cela affecte la lecture des bases entre les différentes versions du logiciel. C'est le cas entre la version 14 et les versions antérieures (dont la version 13 qui date de 2012).
> Version 14 (idem 15,16): refonte total du système d'encodage vers le standard UTF8.
> Les versions inférieures ne peuvent pas ouvrir les bases au format actuel, qui doivent être enregistrées avec une commande particulière (`saveold`) sous Stata 14 à 17. Les accents ne sont pas lu après cette conversion, et ce qui nécessite d'exécuter un programme pour les convertir (au secours!!!!).
> A l'Ined, depuis le déploiement de la version 14 et le passage à la version 15 Linux, il n'y a plus de problème. Attention toutefois pour les collaborations avec l'extérieur.

1.4 Les users packages

Stata fonctionne avec un système hybride de commandes officielles ou usines et de commandes externes (idem R, Python…).
Commandes externes ou *user package*:

- Il s'agit d'instructions créees par les utilisateurs

- Ces instructions couvrent le data management et l'analyse statistique (descriptive, modélisation, graphiques)
- Même syntaxe que les instructions officielles
- La liste (triée par année) de ces instructions se trouve à cette adresse [http://ideas.repec.org/s/boc/bocode.h Elle n'est pas exhaustive, seulement celles sur le dépôt de Stata. Les dépôts dans les gits, en particulier *Github* tendent à se développer. Tout comme R, les dépôts usines ne sont pas forcément les plus à jour.

Installation directe d'une commande externe
Pour télécharger un package, on peut utiliser la ligne de commande:

Listing 1.2 `Installation d'une commande`

```
ssc install nom_commande, replace
```

Il est possible d'installer un package après avoir consulté les fichiers avec la commande `findit` (on peut donc consulter l'aide en amont)

Les programmes des packages externes se trouve dans un répertoire spécifique **plus/ado**, souvent installé dans la racine du lecteur `C:`. On peut déplacer ce répertoire, et c'est conseillé si on dispose, comme à l'Ined, d'une sauvergarde régulière sur un lecteur. On peut vérifier la localisation de ce répertoire avec la commande `sysdir`, et plus généralement la localisation des répertoires d'utilisation de Stata.

```
sysdir
```

```
   STATA:  C:\PROGRA~1\Stata17\
    BASE:  C:\PROGRA~1\Stata17\ado\base\
    SITE:  C:\PROGRA~1\Stata17\ado\site\
    PLUS:  D:\D\ado\plus\
PERSONAL:  D:\D\ado\personal\
OLDPLACE:  c:\ado\
```

Exercice

Ouvrir la base d'exemple *auto*
Dans la fenêtre *Command*: **`sysuse auto, clear`**

- [1] Accéder à l'aide (pdf) de l'instruction `tabulate`. Faire un tableau croisé entre la variable *foreign* et *rep78*
- [2] Installer le package externe `findname`, exécuter `findname t*` et chercher les variables de type chaine de caractère (string)

profile.do

Attention: Avec la version 18, la technique permettant avec un profile.do de récupérer une sauvegarde d'un programme n'a plus aucune utilité, Stata ayant (enfin) une autosave des programmes.

Paramétrer vos sessions avec un fichier ***profile.do*** [Lien]

2 L'environnement

Commandes et fonction introduites
En italique, commandes externes

Sections	Commandes
Menus et raccourcis	`db`
L'éditeur de programme	`run do #delimit log` *`log2do2`*

2.1 Les (principaux) types de fichier

- Les bases de données: **.dta**
- L'éditeur de programme : **.do** (**.ado** pour la programmation de commandes)
- Le log de session : **.smcl** ou **.log** (conversion possible en **.txt**, **.doc** ou **.pdf**)
- Les graphiques : **.gph** (+ enregistrement en format image: png, jpg, svf…)
- Les fichiers d'aides: **.sthlp**

2.2 Où passer ses instructions ?

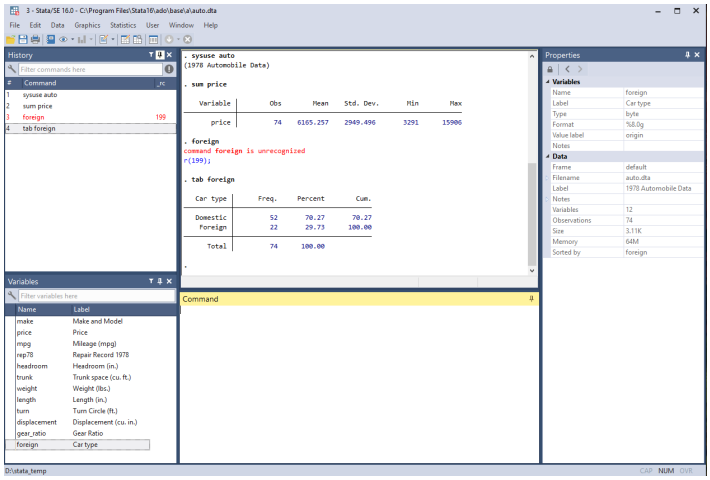
- Dans l'éditeur de programme (.do ou .ado ou écritue d'un fichier d'aide)
- Dans la fenêtre "Command" de l'environnement principal
- Par les menus (ouverture d'une boîte de dialogue). On récupéré la syntaxe de l'instruction directement dans l'interface principale, et par copier-coller dans l'éditeur de programme. Plutôt pratique.

2.3 L'interface principale

Boite output

Affiche les commandes exécutées et les résultats

Boite review



- C'est un log de toutes les instructions passées par la fenêtre command
- En cliquant de dessus, l'instruction est de nouveau affichée dans la fenêtre command
- Avec un clic-droit on peut copier, supprimer les instructions. Les instructions en rouge déclarent celles avec une erreur
- Les instructions passée par l'éditeur de programme ne sont reportée

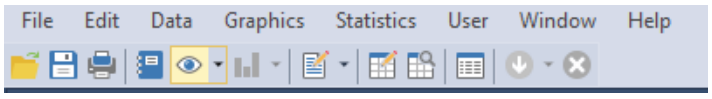
Boite variables

- Liste les variables présentent dans la base chargée avec éventuellement leur label
- En cliquant dessus, là ou les variables sont affichées dans la fenêtre command
- Avec un cliqu-droit, une ou plusieurs variables peuvent être copiées, conservées ou supprimées

Boite properties

- Pour les variables affiche leurs propriétés: nom, label, format, type….
- Si le cadenat est délocké, on peut modifier ces propriétés (voir variable manager). Le code qui exécute la modification est affichée dans la fenêtre command et la boîte review

2.4 Menus et raccourcis



Menus (boîtes de dialogue)

- File: gestion des fichiers => création, ouverture, import/export…
- Edit: utile pour la modification des préférences (couleurs interface, thèmes graphique)
- Data: manipulation des données
- Graphics: création de graphiques

- Statistics: instructions stat: descriptifs, modèles
- User: accès à vos boîtes de dialogue programmée en java
- Windows: activation des boîtes de l'interface principale
- help: infos sur le logiciel et accès à des ressources (internes et liens)
  On peut accéder directement à une boîte de dialogue avec la commande `db nom_commande`

Listing 2.1 `Syntaxe`

```
db logit
```

Raccourcis (dans l'ordre de gauche à droite)

- Ouverture d'un fichier
- Enregistrement de la base de données active
- Impression de la fenêtre output
- Ouverture d'un log
- Ouverture de la fenêtre d'aide
- Ouverture de l'éditeur de graphique (modification d'un graphique créé)
- Création ou ouverture de l'éditeur de programme (.do .ado . sthlp)
- Ouvre la base de données active en mode édition
- Ouvre la base de données active en mode lecture
- Ouvre le variable manager
- Déblocage du défilement de l'output pendant l'exécution
- Arrête l'exécution d'un programme

2.5 L'éditeur de programme

2.5.1 Soumettre un programme ou un bloc de programme

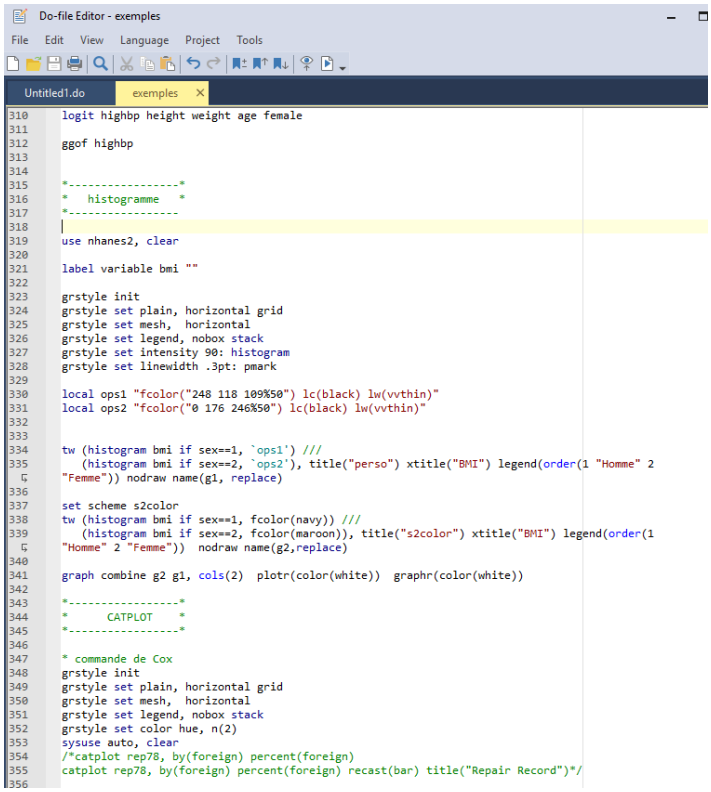
- On ne sélectionne aucune ligne: tout le programme est exécuté
- On sélectionne une partie du programme: seul le bloc est exécuté

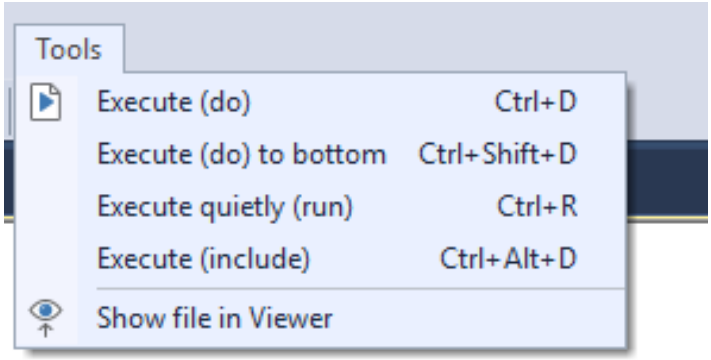
- **execute (do)**: les commandes et les résultats sont affichés dans l'output de l'interface principale
- **execute quietly (run)**: les commandes et les résultats ne sont pas affichés dans l'output de l'interface principale

`do` et `run` dans un programme

On peut exécuter un ou plusieurs programme enregistré dans un programme

```
do "path/nom_prog.do"  // avec affichage de l'output
run "path/nom_prog.do" // sans affichage de l'output
```


Do-file Editor - exemples
File Edit View Language Project Tools
Untitled1.do
exemples
logit highbp height weight age female
ggof highbp
----
* histogramme *
----
use nhanes2, clear
label variable bmi ""
grstyle init
grstyle set plain, horizontal grid
grstyle set mesh, horizontal
grstyle set legend, nobox stack
grstyle set intensity 90: histogram
grstyle set linewidth .3pt: pmark
local ops1 "fcolor("248 118 109%50") lc(black) lw(vvthin)"
local ops2 "fcolor("0 176 246%50") lc(black) lw(vvthin)"
tw (histogram bmi if sex==1, `ops1') ///
(histogram bmi if sex==2, `ops2'), title("perso") xtitle("BMI") legend(order(1 "Homme" 2
"Femme")) nodraw name(g1, replace)
set scheme s2color
tw (histogram bmi if sex==1, fcolor(navy)) ///
(histogram bmi if sex==2, fcolor(maroon)), title("s2color") xtitle("BMI") legend(order(1
"Homme" 2 "Femme")) nodraw name(g2,replace)
graph combine g2 g1, cols(2) plotr(color(white)) graphr(color(white))
----
* CATPLOT *
----
* commande de Cox
grstyle init
grstyle set plain, horizontal grid
grstyle set mesh, horizontal
grstyle set legend, nobox stack
grstyle set color hue, n(2)
sysuse auto, clear
/*catplot rep78, by(foreign) percent(foreign)
catplot rep78, by(foreign) percent(foreign) recast(bar) title("Repair Record")*/


Tools
Execute (do) Ctrl+D
Execute (do) to bottom Ctrl+Shift+D
Execute quietly (run) Ctrl+R
Execute (include) Ctrl+Alt+D
Show file in Viewer

2.5.2 delimiter

Par défaut une ligne = une instruction (**`#delimit cr`**).

Stata ne comprend pas la deuxième ligne de:

```
keep  X1 X2
      X3 X4
```

On peut contourner cela dans un programme pour écrire des instructions longue sur plusieurs lignes avec l'instruction **`#delimit ;`**

```
#delimit ;
keep  X1 X2
      X3 X4;

tabulate X1;
```

Le délimiteur s'appliquera tant qu'on ne repasse pas à `#delimit cr`

```
#delimit ;
keep  X1 X2
      X3 X4;

#delimit cr
tabulate X1
```

Le délimiteur `;` est sourtout utilisé pour les graphiques qui peuvent âtre assez gourmants en options (je le conseille).

La solution la plus utilisée est `///` à la fin d'une ligne, à l'exception de la dernière de la commande

```
keep  X1 X2 ///
      X3    ///
      X4

tabulate X1
```

2.5.3 Les commentaires

Si le commentaire est sur une seule ligne

```
* Commentaire
```

Si le commentaire est sur une ligne et suit une instruction

```
instruction  // Commentaire
```

Si le commentaire est sur plusieurs lignes

```
/* commentaire 1
   commentaire 2 */
```

⚠ Pas d'autosave des programmes...jusqu'à Stata 18

Enfin, enfin, enfin....
Depuis avril 2023 une autosave des programmes a été implémentée au logiciel. Pour faire court:

- Penser à ce que le fichier soit déjàsauvegardé.
- On peut paramétrer la fréquence de sauvegarde dans les préférences de l'éditeur [edit=>preference=>onglet Advanced]. Par défaut la fréquence est de 4 secondes, ce qui semble suffisant.
- Après un crash, ou autre mésaventure, il sera proposer d'ouvrir le fichier sauvegardé automatiquement.
- Penser également à enregistrer le programme de récupération avec le nom original.

Solution pour les version antérieures: le log de session
On peut sauver les meubles en générant un log de session en tête de programme ou mieux en générant un log à chaque ouverture de session dans un fichier profile [[marche à suivre]https://mthevenin.github.io/stata_fr/articles/index/posts/profile/profile.html)].

- Pour générer un log de session qui enregistrera en continu l'output de l'interface, et donc les lignes de commandes exécutée (sauf si exécution en mode `run`):
  - **file => log => begin**, puis choisir un emplacement pour l'enregistrement du fichier *log*.
  - `log using "path/nom_log.smcl"`
- Commande externe `log2do2`: à partir d'un fichier log, permet de conserver seulement les lignes de programmes en supprimant les éléments de l'output de types tableaux, warning et autre messages. Attention les lignes comportant des erreurs seront également conservée.
  `ssc install log2do2`

2.5.3.1 Interactions entre l'interface principale, les boîtes de dialogue et l'éditeur de programme

DEMONSTRATION PENDANT LA FORMATION

3 Le langage Stata

Programme du chapitre

Commandes et expressions introduites

SECTION	COMMANDES ET EXPRESSIONS
Opérateurs	`= == < <= > >= != & | + - * / ^`
Valeurs manquantes	`.` **`mdesc mvpatterns misschk`**
Suppression de l'output, et affichage d'une expression	`quietly display`
Sélection groupées	*`drop keep`* `* -`
Macros et répétition	`local global foreach forvalue return list i` *`regress`*

- **En gras, commandes externes**
- *En italique, commandes associées à un chapitre ultérieur*

3.1 La syntaxe générique

Remarque sur les crochets

> **i** […] expression entre crochets
>
> A ma connaissance, à l'exception des pondérations il n'y a pas d'utilisation de crochets dans la syntaxe des commandes usines. Dans les fichiers d'aide et pour cette formation, ils indiquent les expressions optionnelles d'une ligne d'instructions.

Forme concise de la syntaxe Stata pour une instruction portant sur des variables:

Exemple : `tabulate var1 var2, nofreq row`

Ce qui ce traduit par: produire un tableau croisé entre *var1* (en ligne) et la *var2* (en colonne) en affichant la répartition en % de *var2* pour chaque valeur de *var1* et pour l'ensemble des donnnées renseignées .

```
<IPython.core.display.HTML object>
```

```
(1978 automobile data)
```

```
tabulate rep78 foreign, nofreq row
```

```
    Repair |
    record |      Car origin
      1978 |  Domestic    Foreign |     Total
-----------+----------------------+----------
         1 |    100.00       0.00 |    100.00
         2 |    100.00       0.00 |    100.00
         3 |     90.00      10.00 |    100.00
         4 |     50.00      50.00 |    100.00
         5 |     18.18      81.82 |    100.00
-----------+----------------------+----------
     Total |     69.57      30.43 |    100.00
```

- Forme générique de la syntaxe STATA

```
[prefix:] command varlist [ [type_weight=var] if/in, options]
```

Expression conditionnelle (sélection): `if` (sélection de valeurs) ou `in` (sélection d'observations)

Exemple : **`bysort var2: summarize var1 if var4!=1, detail`**
Ce qui ce traduit par: pour chaque valeur de *var2*, des statistiques descriptives détaillées pour la variable *var1* si la valeur de *var3* est différente de 1.

```
bysort foreign: summarize price if rep78!=1, detail
```

```
-------------------------------------------------------------------------------
-> foreign = Domestic

                            Price
-------------------------------------------------------------
      Percentiles      Smallest
 1%         3291            3291
 5%         3667            3299
10%         3892            3667       Obs                  50
25%         4181            3799       Sum of wgt.          50

50%       4782.5                       Mean            6132.74
                        Largest       Std. dev.       3143.481
75%         6303           13466
90%        11441           13594       Variance        9881473
95%        13594           14500       Skewness       1.717717
99%        15906           15906       Kurtosis       4.857758
```

```
-------------------------------------------------------------------------------
-> foreign = Foreign

                            Price
-------------------------------------------------------------
      Percentiles      Smallest
 1%          3748           3748
 5%          3798           3798
10%          3895           3895       Obs                  22
25%          4499           3995       Sum of wgt.          22

50%          5759                      Mean           6384.682
                        Largest       Std. dev.      2621.915
75%          7140           9690
90%          9735           9735       Variance       6874439
95%         11995          11995       Skewness       1.215236
99%         12990          12990       Kurtosis       3.555178
```

Si la commande implique une base de données, le nom de la base est généralement précédée de **'using"** (sauf pour les instructions officielle d'ouverture/sauvegarde d'une base):

```
command using nom_base [,options]
```

3.2 Autres langages pris en charge

Internes

- Langage matriciel (MATA)
- Editeur de texte (SMCL) pour rédiger les aides ou paramétrer la forme des outputs (c'est une une horreur)
- Création de boites de dialogue (proche du Java)
- La programmation de commande dispose d'un certain nombre d'éléments de langage dédié. Pour une réutilisation ultérieure automatisée, le programme est enregistré dans un fichier *.ado*

Externes

R

Via une commande externe (`rsource`), on peut exécuter du R. R doit ête bien évidemment installé. Juste pour information:

```
rsource, terminator(END_OF_R)

library(readr)
trans <- read.csv("https://raw.githubusercontent.com/mthevenin/analyse_duree/master/bases/tra
```

```
head(trans)
table(trans$died)

END_OF_R
```

```
  id year age died stime surgery transplant wait mois compet
1 15   68  53    1     1       0          0    0    1      1
2 43   70  43    1     2       0          0    0    1      1
3 61   71  52    1     2       0          0    0    1      1
4 75   72  52    1     2       0          0    0    1      1
5  6   68  54    1     3       0          0    0    1      2
6 42   70  36    1     3       0          0    0    1      1

table(trans$died)

 0  1
28 75
```

Python

Depuis la version 16 on peut programmer intéractivement en Python. Cette intégration est suffisamment permet à Python de reconnaitre les macros Stata dans son code…c'est très utile. Python doit néanmoins être installé manuellement.

Vérification de l'installation

```
python query
```

```
------------------------------------------------------------------------------
    Python Settings
      set python_exec      C:\Users\thevenin_m\AppData\Local\Programs\Python\Py
> thon310\python.exe
      set python_userpath

    Python system information
      initialized          yes
      version              3.10.5
      architecture         64-bit
      library path         C:\Users\thevenin_m\AppData\Local\Programs\Python\Py
> thon310\python310.dll
```

Utilisation de python

```
python:

a = 4
b = 2
a*b

end
```

```
python:
import pandas as pd
trans = pd.read_csv("https://raw.githubusercontent.com/mthevenin/analyse_duree/master/bases/t

trans.head(10)
trans.info()
end
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103 entries, 0 to 102
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   id          103 non-null    int64
 1   year        103 non-null    int64
 2   age         103 non-null    int64
 3   died        103 non-null    int64
 4   stime       103 non-null    int64
 5   surgery     103 non-null    int64
 6   transplant  103 non-null    int64
 7   wait        103 non-null    int64
 8   mois        103 non-null    int64
 9   compet      103 non-null    int64
dtypes: int64(10)
memory usage: 8.2 KB
```

Autres

- Depuis la version 17 de Stata, on peut également programmer intéractivement en *Java*... mais là je n'y connais absolument rien.
- Intégration de l'édition en *markdown* pour produire des rapports en html ou pdf. Egalement possibilité de générer des documents word ou excel avec des commandes dédiées. L'intégration de *latex* est également possible via une commande externe. Toutes ces possibilité me semble néanmoins nettement en très en deçà de ce qui est réalisable actuellement avec les outils proposés par *Posit* (ex Rstudio), en particulier avec le nouvel outil `Quarto` associé au notebook Jupyter (noyau nbstata). Ce sont ces qui sont utilisées pour faire ce support.

3.3 Les opérateurs

Opérateurs
`&` Et
`|` [Alt+Gr6] **Ou**
Opérateurs d'affectation
`=`
Opérateurs pour expressions conditionnelles
`==`
`!=`
`>`
`>=`
`<`
`<=`
Opérateurs arithmétique
`+` , `-` , `/` , $\widehat{}$ (puissance)
Opérateur chaîne de caractères
`+`

[1] `+` concatène des variables caractères qui n'ont pas de valeur manquante. Sinon utiliser **`concat`** associée à la commande **`egen`** (voir chapitre sur la création de variables).

3.4 Les valeurs manquantes

! Statut des valeurs manquantes

La valeur d'une observation manquante dépasse la plus grande valeur observée d'une variable. Ceci doit être pris en compte dans les expressions conditionnelles impliquant par exemple des regroupement de variables ordinales ou mésurées (âge, revenus...):
Si une variable numérique X a des observations manquantes, la condition `if X>valeur` conservera ces informations.
Si $X = (1, 2, 3, 4, 5, 6, .)$:

- `...if x>4` conserve $x = (5, 6, .)$
- `...if x>4 & x<.` ou `...if x>4 & x!=.` regroupe seulement $x = (5, 6)$

Les valeurs manquantes utilisateurs
On peut rendre la valeur manquante informative en lui ajoutant une lettre: **.a** , **.b**, **.c** etc....

Exemple:

- $.a$ = Ne sait pas.
- $.b$ =Refus.

- *.c* =Pas de réponse.

Repérage des valeurs manquantes

Des commandes, comme `tabulate` avec l'option `mis` pour les variables catégorielles, permettent de repérer et d'afficher le nombre d'observations manquantes.

Il y a aussi plusieurs commandes qui permettent d'analyser ce type d'observations observations globalement.

- Commande externe **`mdesc`**: affiche pour chaque variable de la base ou une sélection de celle, le nombre et le % d'observations manquantes.
  - installation: `ssc install mdesc`
  - syntaxe: `mdesc [varlist]`

```
sysuse auto.dta, clear

mdesc
```

```
(1978 automobile data)

    Variable    |     Missing          Total     Percent Missing
----------------+-----------------------------------------------
           make |           0             74           0.00
          price |           0             74           0.00
            mpg |           0             74           0.00
          rep78 |           5             74           6.76
       headroom |           0             74           0.00
          trunk |           0             74           0.00
         weight |           0             74           0.00
         length |           0             74           0.00
           turn |           0             74           0.00
   displacement |           0             74           0.00
     gear_ratio |           0             74           0.00
        foreign |           0             74           0.00
----------------+-----------------------------------------------
```

- commandes externes `mvpatterns` et `misschk` pour analyser les différents patterns de valeurs manquantes (une même observation peut avoir des valeurs manquantes sur plusieurs variables). `misschk` ne scanne que les variables de type numérique, et permet de générer deux variables pour indiquer le nombre et le pattern de valeurs manquantes pour chaque observation.
  - installation: `ssc install mvpatterns` et `ssc install misschk`
  - syntaxe: `mvpatterns [varlist]` et `misschk [varlist], gen(nom)`

```
mvpatterns
```

```
variables with no mv's: make price mpg headroom trunk weight length turn
                    displacement gear_ratio foreign

Variable     | type     obs   mv   variable label
-------------+-------------------------------------
rep78        | int       69    5   Repair record 1978
-------------------------------------------------

Patterns of missing values

  +------------------------+
  | _pattern   _mv   _freq |
  |------------------------|
  |        +     0      69 |
  |        .     1       5 |
  +------------------------+
```

misschk

```
Variables examined for missing values

   #  Variable        # Missing   % Missing
-------------------------------------------
   1  price                   0         0.0
   2  mpg                     0         0.0
   3  rep78                   5         6.8
   4  headroom                0         0.0
   5  trunk                   0         0.0
   6  weight                  0         0.0
   7  length                  0         0.0
   8  turn                    0         0.0
   9  displacement            0         0.0
   10 gear_ratio              0         0.0
   11 foreign                 0         0.0

Warning: this output does not differentiate among extended missing.
To generate patterns for extended missing, use extmiss option.

   Missing for |
         which |
    variables? |      Freq.     Percent        Cum.
---------------+-----------------------------------
 __3__ _____ _ |          5        6.76        6.76
 _____ _____ _ |         69       93.24      100.00
---------------+-----------------------------------
```

```
         Total |          74      100.00

Missing for |
   how many |
 variables? |      Freq.     Percent        Cum.
------------+-----------------------------------
          0 |          69       93.24       93.24
          1 |           5        6.76      100.00
------------+-----------------------------------
      Total |          74      100.00
```

3.5 Casse et troncature

3.6 Sensibilité à la casse

Comme R ou Python, Stata est intégralement sensible à la casse pour les instructions, seulement en minuscules. Par exemple `TABULATE X` renverra un message d'erreur.

3.6.1 Troncature des instructions et des options

Dans le fichier d'aire d'une commande usine ou externe, le niveau de troncature est indiqué par un soulignement dans l'instruction: par exemple **`tabulate`** est souligné au niveau de **ta** : **tabulate** = **tabulat** = **tabula** = **tabul** = **tabu** = **tab** = **ta**. On utilise généralement `tabulate` ou `tab`.

A manier avec une certaine précaution car le programme peut devenir rapidement incompréhensible, surtout s'il est partagé entre personnes dont la pratique diffère à ce niveau:

```
tabulate var1 var2, nofreq row miss

* est équivalent à:

ta var1 var2, nof r m
```

3.7 Suppression de l'output, et affichage d'une expression

On peut rendre le résultat d'une commande invisible dans la fenêtre output avec `quietly` (`qui`).

```
    tab rep78
qui tab rep78
```

```
     Repair |
record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        2.90        2.90
          2 |          8       11.59       14.49
          3 |         30       43.48       57.97
          4 |         18       26.09       84.06
          5 |         11       15.94      100.00
------------+-----------------------------------
      Total |         69      100.00
```

Remarque: ne fonctionne pas avec les graphiques où l'on doit utiliser l'option `nodraw`

display (di)

Ce n'est pas une commande à proprement parler, mais l'instruction `display` (`di`) permet d'afficher dans l'output, entres autres, des opérations arithmétiques (c'est donc une calculatrice).

```
di exp(1)/(1+exp(1))
di "SALUT LES GENS!!!!"
```

```
.73105858
SALUT LES GENS!!!!
```

Elle est également utilisé pour vérifier le contenu d'une macro variable, de préférence lorsque cette macro implique des valeurs.

3.8 Sélection groupées de variables

Commandes associées pour filtrer: **`keep`**, **`drop`** [pour sélectionner des obseravations: `keep if`, `drop if`]

On peut sélectionner un ensemble de variables qui ont une racine commune, par exemple **c**, en écrivant : `*c*`.
Exemple:
television, *telephone*, *table* ont comme racine **t**. Pour supprimer ces variables, on peut exécuter `drop t*` au lieu de `drop television telephone table`.
Si on souhaite supprimer *television* et *telephone* seulement : `drop tele*`.

Si on veut sélectionner des variables occurencées ou comme dans la base *auto* les 5 variables qui se suivent [*headroom*, *trunk*, *weight*, *length*, *turn*]: `keep headroom-turn` . Pour des variables occurencées de x1 à x5: `keep x1-x5`.

```
sum t*
```

```
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       trunk |         74    13.75676    4.277404         5         23
        turn |         74    39.64865    4.399354        31         51
```

3.9 Macros et répétition

3.9.1 Introduction au macros

Juste une introduction…Vu la simplicité du langage Stata, il est conseiller de se mettre rapidement à la manipulation des expressions dites `macro`.

- Une macro, dans sa version la plus simple, est une expression qui est utilisée une ou plusieurs fois dans un programme. Elle sont de type temporaire (`local`) ou enregistré en dur (`global`).
- Les commandes, en particulier sur les opérations statistiques, enregistre un certains nombre d'objet de type macro qui peuvent être utilisés ultérieurement. On peut récupérer leur liste à la fin du fichier d'aide, et les visualiser les valeurs enregistrées après avoir exécuté une commande avec `return list`, `ereturn list`…

- Un autre type d'objet, appelé **`scalar`** ressemble à une macro mais n'en est pas. Il s'agit de pseudo variables. Ils ne seront pas traités.
- Le contenu d'une macro peut être affiché avec `display` ou `macro list` (`mac list`)
  - privilégier `display` pour afficher le contenu d'une macro de type valeur
  - privilégier `mac list` pour afficher le contenu d'une macro de type chaîne de caractère lorsque des doubles quotes (“) doivent rester apparentes. La gestion des” dans les macros peut s'avérer particulièrement retord.
    - macro *local*: mac list _macroname
    - macro *global*: mac list macroname

Macro temporaire

L'instruction `local` permet de définir des macros variables temporaire (disparaissent après l'exécution du programme):

```
local nom_macro expression
* ou
local nom_macro = expression numérique

* ou

local nom_macro :  fonction macro // hors contenu de la formation
```

```
local a = 2

di `a'

di 5^(`a')
```

```
2
25
```

```
local var mpg foreign

di "`var'"
mac list _var

sum `var'

regress price `var'
```

```
mpg foreign
_var:           mpg foreign

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74     21.2973    5.785503         12         41
     foreign |         74    .2972973    .4601885          0          1

      Source |       SS           df       MS      Number of obs   =        74
-------------+----------------------------------   F(2, 71)        =     14.07
       Model |   180261702         2  90130850.8   Prob > F        =    0.0000
    Residual |   454803695        71  6405685.84   R-squared       =    0.2838
-------------+----------------------------------   Adj R-squared   =    0.2637
       Total |   635065396        73  8699525.97   Root MSE        =    2530.9

------------------------------------------------------------------------------

       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]

-------------+----------------------------------------------------------------

         mpg |

   -294.196     55.692    -5.28   0.000     -405.242    -183.149
```

```
     foreign |   1767.292    700.158     2.52   0.014      371.217    3163.368
       _cons |   11905.415   1158.634    10.28   0.000     9595.164   14215.667
------------------------------------------------------------------------------
```

On remarque tout de suite l'utilité de ces expression: si on veut changer la liste de variables pour les instructions `sum` (troncature de `summarize`) et pour `regress`, on le fais une seule fois (dans la définition de la macro) au lieu de deux.

Avec Stata peut définir également des macros dites *global* qui sont sauvegardées et s'appliqueront à tous les programmes (on peut les supprimer). Leur utilisation est moins courante: `global nom_macro expression`, le nom de la macro dans l'expression s'écrira `$nom_macro`.
Il est conseillé de les supprimer en fin de programme avec `macro drop` + noms des macros.

```
global var mpg weight length turn

di "$var"
mac list var

sum $var
regress price $var, noheader

macro drop var
```

```
mpg weight length turn
var:            mpg weight length turn

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
         mpg |         74     21.2973    5.785503         12         41
      weight |         74    3019.459    777.1936       1760       4840
      length |         74    187.9324    22.26634        142        233
        turn |         74    39.64865    4.399354         31         51
------------------------------------------------------------------------------

       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |    -94.651     80.879    -1.17   0.246     -256.000      66.697
      weight |      5.030      1.154     4.36   0.000        2.728       7.332
      length |    -73.147     40.212    -1.82   0.073     -153.368       7.074
        turn |   -323.861    126.882    -2.55   0.013     -576.983     -70.738
       _cons |   19581.418   6005.223     3.26   0.002     7601.327   31561.509
------------------------------------------------------------------------------
```

La liste des macro enregistrées en dur est donnée par l'instruction: `macro dir`

```
mac dir
```

```
S_E_depv:       price
S_E_cmd:        regress
S_2:            1
S_1:            __000003
Rterm_options:  --vanilla
Rterm_path:     C:\Program Files\R\R-4.1.2\bin\R.exe
tmp:            D:\D\stata_temp\
user:           C:\Users\thevenin_m\
S_level:        95
F1:             help advice;
F2:             describe;
F7:             save
F8:             use
S_ADO:          BASE;SITE;.;PERSONAL;PLUS;OLDPLACE
S_StataSE:      SE
S_CONSOLE:      console
S_OS:           Windows
S_OSDTL:        64-bit
S_MACH:         PC (64-bit x86-64)
_width:         78
_width_col1:    13
_var:           mpg foreign
_a:             2
S_FN:           C:\Program Files\Stata18/ado\base/a/auto.dta
S_FNDATE:       13 Apr 2022 17:45
```

3.9.2 Objets sauvegardés lors de l'exécution d'une commande

- Ces objets de type macro ne sont conservés en mémoire qu'entre 2 commandes exécutés.
- On peut donc les manipuler qu'à ce moment là, en particulier les enregistrer sous forme de macro standard pour les utiliser ultérieurement (exemple: normaliser automatiquement une pondération, reporter des moyennes dans un graphique etc….)

```
qui sum price

return list

local mprice = r(mean)

di `mprice'
```

```
scalars:
                  r(N) =  74
              r(sum_w) =  74
               r(mean) =  6165.256756756757
                r(Var) =  8699525.974268788
                 r(sd) =  2949.495884768919
                r(min) =  3291
                r(max) =  15906
                r(sum) =  456229
6165.2568
```

> **Note**
>
> Une application typique est la normalisation d'une pondération brute (somme des poids = nombre d'observation dans l'échantillon).
> Si *wb* est la pondération brute (somme des poids = population cible), et *wn* les poids que l'on souhaite normaliser:

```
qui sum wb
generate wn = wb/`r(mean)'
```

> Tout changement de la variable *wb* modifiera automatiquement cette normalisation.
> Pour la commande **`gen` (ou `generate`) se reporter au chapitre 5.

3.10 Répétition avec des boucles

- **`forvalues`**: valeurs occurencées, compteur

`for num 1/n: commande` est de plus en plus abandonnée (aide Stata supprimée). On lui préfère maintenant l'instruction `forvalues` pour effectuer des boucles sur des occurences numériques.

Si l'on souhaite par exemple changer le nom des variables x1 à x9 en var1, var2,…., var9:

```
forvalues i=1/9 {

rename x`i' var`i'

}
```

- **`foreach`**: termes d'une expression enregistrée sous la forme d'une macro

Par l'exemple, et juste une petite introduction. On veut faire une régression linéaire entre la variable *price* et la variable *foreign* en ajoutant une seule autre variable dans cette liste: *mpg*, *headroom*, *trunk*.

Au lieu d'exécuter:

```
regress price foreign mpg
regress price foreign headroom
regress price foreign trunk
```

On génère une macro variable temporaire qui liste ces 3 variables , et on exécute une boucle avec l'instruction `foreach`.

```
local var mpg headroom trunk

foreach x of local var {

regress price foreign `x', noheader
}
```

```
------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
     foreign |   1767.292    700.158     2.52   0.014      371.217    3163.368
         mpg |   -294.196     55.692    -5.28   0.000     -405.242    -183.149
       _cons |  11905.415   1158.634    10.28   0.000     9595.164   14215.667
------------------------------------------------------------------------------
------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
     foreign |    577.812    787.566     0.73   0.466     -992.549    2148.174
    headroom |    491.575    428.405     1.15   0.255     -362.641    1345.791
       _cons |   4522.071   1412.097     3.20   0.002     1706.430    7337.711
------------------------------------------------------------------------------


------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
     foreign |   1190.155    760.805     1.56   0.122     -326.847    2707.157
       trunk |    262.772     81.852     3.21   0.002       99.564     425.980
       _cons |   2196.541   1267.857     1.73   0.088     -331.494    4724.576
------------------------------------------------------------------------------
```

….et on peut aller plus loin… Juste pour information car cela se complique (et pas qu'un peu), avec une technique de macro empilée, on ajoute les les variables une à une au modèle.

```
local j mpg weight length turn headroom trunk

foreach j2 of local j {
local x `x' `j2'
```

```
di  "covariables introduites = `x'"  // pour afficher ce qui est lu dans la macro

regress price `x' , noheader
}
```

```
covariables introduites = mpg

------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |   -238.894     53.077    -4.50   0.000     -344.701    -133.088
       _cons |  11253.061   1170.813     9.61   0.000     8919.088   13587.033
------------------------------------------------------------------------------
covariables introduites = mpg weight
------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |    -49.512     86.156    -0.57   0.567     -221.302     122.278
      weight |      1.747      0.641     2.72   0.008        0.468       3.025
       _cons |   1946.069   3597.050     0.54   0.590    -5226.245    9118.382
------------------------------------------------------------------------------
covariables introduites = mpg weight length

------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |    -86.789     83.943    -1.03   0.305     -254.209      80.630
      weight |      4.365      1.167     3.74   0.000        2.036       6.693
      length |   -104.868     39.722    -2.64   0.010     -184.090     -25.646
       _cons |  14542.434   5890.632     2.47   0.016     2793.940   26290.929
------------------------------------------------------------------------------
covariables introduites = mpg weight length turn
------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |    -94.651     80.879    -1.17   0.246     -256.000      66.697
      weight |      5.030      1.154     4.36   0.000        2.728       7.332
      length |    -73.147     40.212    -1.82   0.073     -153.368       7.074
        turn |   -323.861    126.882    -2.55   0.013     -576.983     -70.738
       _cons |  19581.418   6005.223     3.26   0.002     7601.327   31561.509
------------------------------------------------------------------------------
covariables introduites = mpg weight length turn headroom

------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
```

```
-------------+----------------------------------------------------------------
         mpg |    -96.145     80.259    -1.20   0.235     -256.300      64.010
      weight |      5.015      1.145     4.38   0.000        2.730       7.300
      length |    -60.922     40.791    -1.49   0.140     -142.319      20.476
        turn |   -332.592    126.045    -2.64   0.010     -584.111     -81.073
    headroom |   -538.252    373.149    -1.44   0.154    -1282.859     206.356
       _cons |  19317.280   5961.554     3.24   0.002     7421.185   31213.375
------------------------------------------------------------------------------
covariables introduites = mpg weight length turn headroom trunk
------------------------------------------------------------------------------
       price | Coefficient  Std. err.      t    P>|t|     [95% conf. interval]
-------------+----------------------------------------------------------------
         mpg |    -94.063     80.371    -1.17   0.246     -254.484      66.357
      weight |      5.079      1.148     4.42   0.000        2.788       7.371
      length |    -73.487     43.011    -1.71   0.092     -159.338      12.364
        turn |   -327.070    126.311    -2.59   0.012     -579.188     -74.952
    headroom |   -731.292    427.369    -1.71   0.092    -1584.324     121.740
       trunk |     98.275    105.721     0.93   0.356     -112.745     309.295
       _cons |  20447.251   6090.068     3.36   0.001     8291.424   32603.078
------------------------------------------------------------------------------
```

4 Les bases de données

Programme du chapitre

Commandes et expressions introduites

SECTION	COMMANDES ET EXPRESSIONS
Affectation d'un répertoire de travail	`cd`
Ouverture et sauvegarde d'une base	`use` `webuse` `[set]` `sysuse` `save` `saveold` `import` `export` **`usesas`** **`savasas`** `set` `obs` `insobs`
Décrire le contenu d'une base	`describe` `list` `codebook` `labelbook` `label list`
Tri, doublon, position des variables	`sort` `gsort` `duplicates` `order`
Decription statistique des variables	`summarize` `mean` `tabstat` **`violinplot`** **`heatplot`** **`gjoint`** `tabulate` **`fre`** **`tabm`** **`catplot`**
Introductions aux frames	`frame dir` `frame list` `frame create` `frame rename` `frame copy` `frame change` `frame drop` `frame reset`

- **En gras, commandes externes**
- *En italique, commandes associées à un chapitre ultérieur*

4.1 Affectation du répertoire de travail

La commande `cd` (Current Directory), permet d'indiquer le chemin d'accès du répertoire où se trouve la base à ouvrir ou à enregistrer. Si aucun chemin d'accès n'est spécifié, Stata ira chercher la base dans le répertoire par défaut (normalement C: ou D:).

Syntaxe:

```
cd "path"
```

Remarque
Avec cette commande, un seul repertoire de travail est actif. On peut élargir les possibilités en affectant des répertoires avec des macros variables locales ou globales

4.2 Ouverture et sauvegarde d'une base

4.2.1 Ouverture

Commande `use`

Syntaxe sans chargement d'un répertoire:

```
use "path/nom_base.dta" [,clear]
```

L'option `clear` permet d'effacer une base en cours d'utilisation. Il est conseillé de mettre cette option systématiquement. On peut également utiliser `clear` comme instruction avant d'ouvrir une base. on ne supprime pas la base du répertoire (commande `erase`), elle est juste écrasée dans la session.

Syntaxe avec affectation d'un répertoire:

```
cd "path"
use "nom_base.dta", clear
```

ou

```
cd "path"
clear
use "nom_base.dta"
```

Remarque: pour les bases d'exemples préinstallées, on utilise la commande `sysuse`. Dans les fichiers d'aide, des exemples font également appels a des bases localisées sur des serveurs qui s'ouvrent avec la commande `webuse`.

```
sysuse auto, clear
```

> **Note**
>
> **Ouverture d'une base stockée sur un git [github, gitlab ...]**
> Dans un premier temps, comme pour `cd` il faut charger le répertoire où se trouve localisé la base, avec la commande **`webuse set`**. Par exemple sur mon dépôt git, une base d'exemple (*logement.dta*) pour une commande se trouve à cette adresse: https://github.com/mthevenin/stata_graphiques/tree/main/programmation/gjoint
> Pour charger ce répertoire à distance:
>
> ```
> webuse set  "https://raw.githubusercontent.com//mthevenin/stata_graphiques/master/ressource
> ```
>
> On remarque que le chemin n'est pas identique au simple lien (spécificité des dépôt de type git).
> Il suffit ensuite de charger la base avec `webuse`

```
webuse logement.dta, clear
```

On revient au dépôt officiel de stata avec `webuse set`
En résumé avec un seul bloc d'instructions:

```
webuse set  "https://raw.githubusercontent.com//mthevenin/stata_graphiques/master/ressource

webuse "logement.dta", clear

webuse set
```

```
<IPython.core.display.HTML object>

(prefix now "https://raw.githubusercontent.com//mthevenin/stata_graphiques/mast
> er/ressources/gjoint")

(prefix now "https://www.stata-press.com/data/r18")
```

4.2.2 Sauvegarde

Commandes `save` ou `saveold`

```
save "path/nom_base.dta" [, replace]
```

L'option `replace` permet d'écraser une version antérieure de la base. Obligatoire à partir de la 2ème sauvegarde, mais on peut l'utiliser dès la première sauvegarde (un message d'avertissement s'affiche).

```
sysuse auto, clear

save auto, replace
```

```
(1978 automobile data)
file auto.dta saved
```

`saveold` permettra d'ouvrir une bases avec une version ancienne de Stata non compatibles avec la version courante. Cela commence à devenir moins critique, mais avec `save` il ne sera plus possible d'ouvrir une base avec une version inférieure à la 13 ou inférieur à la 13 (passage à l'encodage Utf8 avec la version 14).

```
 saveold "path/nom_base.dta", [replace] [version(#)] // # = numéro de la version de Stata:
```

Remarque:

- Ecrire l'extension .dta n'est pas obligatoire
- Les doubles quotes ne sont obligatoires que s'il y a un espace dans le chemin d'accès et/ou dans le nom de la base

```
use "ma base", clear
use ma_base,   clear
```

4.2.3 Autres formats

4.2.3.1 Importation/Exportation

Excel et fichiers textes (.txt, .csv)

- Le plus simple est passer par le menu: files + [*Import* ou *Export*] qui dispose d'une fenêtre de prévisualisation.
  - Pour excel les commandes sont `import excel` et `export excel`
  - Pour des fichiers textes type csv (R), les commandes sont `import delimited` et `export delimited`

**Exemples

```
* exportation csv
export delimited using "D:\D\stata_temp\export_csv.csv", replace

* exportation xls
export excel using "D:\D\stata_temp\export_excel.xls", firstrow(variables) replace

* importation cxv
import delimited "D:\D\stata_temp\export_csv.csv",  clear


* importation xls
import excel "D:\D\stata_temp\export_excel.xls", sheet("Sheet1") firstrow clear
```

SAS

- Depuis la version 16 de Stata il est possible d'importer directement des formats `sas7bdat`. Pas d'exportation possible.
- Pour les versions antérieure, la solution installée via *sasxport* n'est pas satisfaisante. Il est alors conseillé d'utiliser le package externe `savasas`
  - Sas => Stata [importation]: commande `usesas`
  - Stata => Sas [exportation] : commande `savasas`

– Si le chemin d'accès à l'exécutable de SAS Windows n'est pas reconnu, il faut récupérer et éditer le fichier sasexe.do (à partir de la ligne 169), dont l'accès est donné dans le fichier d'aide.

– Pour l'exportation, on peut générer un catalogue de format,en dur, avec l'option `format`.

SPSS

- Depuis la dernière version de Stata (16), il est possible d'importer directement des bases de ce format.
- Pas de possibilité d'exportation directe.

4.3 Création d'une base de donnée, ajout d'observations

On peut créer une base de donnée, "vide", avec la commande **`set obs n_obs`**

```
clear
set obs 100
```

A une base existante, on peut ajouter des observations (en valeurs manquante) avec la commande `insobs nbre_observation`. Par défaut ces observations s'ajouteront après la dernière ligne (option `before/after(position)` pour renseigner la position de la première observation ajoutée.

```
insobs 10  // ajout de 10 observations à la base
```

5 Décrire le contenu d'une base

5.0.1 Commande `describe`

Permet, sous forme de tableau, d'avoir des renseignement sur une base de donnée: taille en mémoire, nombre d'observations, descriptif des variables (nom, format, labels). La commande est régulièrement tronquée jusqu'à `des`

Base courante (ouverte)

Syntaxe:

```
describe [varlist , short]
```

```
sysuse auto.dta, clear

des
```

```
(1978 automobile data)

Contains data from C:\Program Files\Stata18/ado\base/a/auto.dta
 Observations:            74                  1978 automobile data
    Variables:            12                  13 Apr 2022 17:45
                                              (_dta has notes)
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
make            str18   %-18s                 Make and model
price           int     %8.0gc                Price
mpg             int     %8.0g                 Mileage (mpg)
rep78           int     %8.0g                 Repair record 1978
headroom        float   %6.1f                 Headroom (in.)
trunk           int     %8.0g                 Trunk space (cu. ft.)
weight          int     %8.0gc                Weight (lbs.)
length          int     %8.0g                 Length (in.)
turn            int     %8.0g                 Turn circle (ft.)
displacement    int     %8.0g                 Displacement (cu. in.)
gear_ratio      float   %6.2f                 Gear ratio
foreign         byte    %8.0g      origin     Car origin
-------------------------------------------------------------------------------
Sorted by: foreign
```

Base stockée (non ouverte)

On peut également décrire le contenu d'une base en format .dta en mémoire et non ouverte avec l'argument `using "path/nombase"`

```
describe using  "https://www.stata-press.com/data/r17/census2.dta"
```

```
Contains data                                  1980 Census data by state
 Observations:            50                   2 Dec 2020 09:21
    Variables:            15
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
state           str13   %-13s                 State
state2          str2    %-2s                  Two-letter state abbreviation
region          byte    %-8.0g     cenreg     Census region
pop             long    %12.0gc               Population
poplt5          long    %12.0gc               Pop, < 5 year
pop5_17         long    %12.0gc               Pop, 5 to 17 years
pop18p          long    %12.0gc               Pop, 18 and older
pop65p          long    %12.0gc               Pop, 65 and older
popurban        long    %12.0gc               Urban population
medage          float   %9.2f                 Median age
death           long    %12.0gc               Number of deaths
marriage        long    %12.0gc               Number of marriages
divorce         long    %12.0gc               Number of divorces
drate           int     %9.0g                 Deathrate
age             byte    %9.0g                 Age
-------------------------------------------------------------------------------
Sorted by:
```

5.0.2 Autres commandes

- **Affichage de la base dans l'output**

Commande `list`

Syntaxe:

```
 list [varlist] [expression]
```

```
list price mpg turn foreign in 1/10
```

```
     +--------------------------------+
     |  price   mpg   turn    foreign |
     |--------------------------------|
  1. |  4,099    22     40   Domestic |
  2. |  4,749    17     40   Domestic |
  3. |  3,799    22     35   Domestic |
  4. |  4,816    20     40   Domestic |
  5. |  7,827    15     43   Domestic |
     |--------------------------------|
  6. |  5,788    18     43   Domestic |
  7. |  4,453    26     34   Domestic |
  8. |  5,189    20     42   Domestic |
  9. | 10,372    16     43   Domestic |
 10. |  4,082    19     42   Domestic |
     +--------------------------------+
```

Sauf exceptions, comme la petite base d'exemple utilisée ici, penser à bien filtrer les informations souhaitées en termes de variables et d'observations.

- **Information sur les labels affectés aux variables**

Commande `labelbook` et `label list`

`labelbook`: affiche les informations sur les labels affectés aux modalités des variables.

```
labelbook [nom_label]
```

```
labelbook origin
```

```
-------------------------------------------------------------------------------
Value label origin
-------------------------------------------------------------------------------

      Values                                  Labels
       Range:  [0,1]                  String length:  [7,8]
           N:  2                 Unique at full length:  yes
        Gaps:  no                  Unique at length 12:  yes
  Missing .*:  0                           Null string:  no
                              Leading/trailing blanks:  no
                                   Numeric -> numeric:  no
  Definition
           0   Domestic
           1   Foreign

   Variables:  foreign
```

`label list [nom_label]` donne seulement l'affectation des labels aux valeurs.

```
label list origin
```

```
origin:
           0 Domestic
           1 Foreign
```

5.1 Tri, doublon, position des variables

5.1.1 Tri d'une base

Commande `sort`

```
sort varlist
```

- La commande `sort` n'effectue que des tris croissants. Pour faire un tri décroissant, on peut utiliser la commande `gsort`. Tris croissants et décroissant peuvent se succéder dans une logique de cluster.
  - `sort varlist` => tri croissant
  - `gsort + var1` => croissant *var1*
  - `gsort - var1` => décroissant *var1*
  - `gsort + var1 - var2` => croissant *var1* et décroissant *var2* dans chaque strate de *var1*
- Le tri d'une peut/doit être réalisé lorsqu'on veut répéter une instruction avec le préfixe `bysort`: `bysort varlist: instruction`. Il est imposé lorsqu'on souhaite apparier des bases [voir chapitre 6]

```
* tri croissant sur la variable price
sort price

* tri décroissant sur la variable prix pour chaque niveau de la variable foreign
gsort + foreign - price
```

5.1.2 Repérage et suppression des doublons

Repérage et suppression des doublons

Commande `duplicates list/tag/drop [varlist]`

Permet de lister, repérer (avec `gen(varname)`) ou supprimer des observations répliquées. Si la liste de variables n'est pas renseignée, elles toutes sont utilisées.

Syntaxe:

```
duplicates list [varlist]

duplicates tag [varlist], gen(var)

duplicates drop [varlist]
```

5.1.3 Modifier la place des variables dans la base

Commande `order`

Syntaxe:

```
order varlist, [first/last] [after/before(varname)]
```

```
order foreign, first
order rep78, after(foreign)

des
```

```
Contains data from C:\Program Files\Stata18/ado\base/a/auto.dta
 Observations:            74                  1978 automobile data
    Variables:            12                  13 Apr 2022 17:45
                                              (_dta has notes)
------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g      origin     Car origin
rep78           int     %8.0g                 Repair record 1978
make            str18   %-18s                 Make and model
price           int     %8.0gc                Price
mpg             int     %8.0g                 Mileage (mpg)
headroom        float   %6.1f                 Headroom (in.)
trunk           int     %8.0g                 Trunk space (cu. ft.)
weight          int     %8.0gc                Weight (lbs.)
length          int     %8.0g                 Length (in.)
turn            int     %8.0g                 Turn circle (ft.)
displacement    int     %8.0g                 Displacement (cu. in.)
gear_ratio      float   %6.2f                 Gear ratio
------------------------------------------------------------------------------
Sorted by: foreign
```

Cette opération est particulièrement pour organiser sa base de données après la création de nouvelles variables.

5.2 Description statistique des variables

Dans les menus de l'interface principale: **Statistics => Summaries, tables & tests**

5.2.1 Variables quantitatives

5.2.1.1 Tableaux d'indicateurs

> **Note**
>
> Les commandes qui sont rapidement décrites afficheront des indicateurs communs, typiquement la moyenne. Elles se distinguent par la forme de leur output facilitant plus ou moins les comparaisons, les possibilités offertes en termes de pondération, et sur la récupération des résultats (macro).

Commande `summarize`

Comme son l'indique, la commande `summarize`, avec l'option `detail` (`d`) donne un résumé complet de la distribution d'une variable quantitative: moyenne, variance, quantiles, symétrie, applatissement ..nom l'indique l.)

Syntaxe:

```
summarize varlist [, detail]
```

Si on indique pas le nom d'au moins une variable, toutes les variables de la base seront sélectionnées.

La commande peut-être tronquée jusqu'à `sum` [**Warning**: il existe également une fonction `sum` pour générer des cumuls lors d'une création de variable - voir chapitre 5].

```
sum price

sum
```

```
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906

    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
     foreign |         74    .2972973    .4601885         0          1
       rep78 |         69    3.405797    .9899323         1          5
        make |          0
       price |         74    6165.257    2949.496       3291      15906
         mpg |         74     21.2973    5.785503         12         41
-------------+---------------------------------------------------------
```

```
    headroom |         74    2.993243    .8459948        1.5          5
       trunk |         74    13.75676    4.277404          5         23
      weight |         74    3019.459    777.1936       1760       4840
      length |         74    187.9324    22.26634        142        233
        turn |         74    39.64865    4.399354         31         51
-------------+---------------------------------------------------------
displacement |         74    197.2973    91.83722         79        425
  gear_ratio |         74    3.014865    .4562871       2.19       3.89
```

```
sum price length, d
```

```
                            Price
-------------------------------------------------------------
      Percentiles      Smallest
 1%         3291            3291
 5%         3748            3299
10%         3895            3667       Obs                  74
25%         4195            3748       Sum of wgt.          74

50%       5006.5                       Mean           6165.257
                        Largest       Std. dev.      2949.496
75%         6342           13466
90%        11385           13594       Variance       8699526
95%        13466           14500       Skewness       1.653434
99%        15906           15906       Kurtosis       4.819188

                        Length (in.)
-------------------------------------------------------------
      Percentiles      Smallest
 1%          142             142
 5%          154             147
10%          157             149       Obs                  74
25%          170             154       Sum of wgt.          74

50%        192.5                       Mean           187.9324
                        Largest       Std. dev.      22.26634
75%          204             221
90%          218             222       Variance       495.7899
95%          221             230       Skewness      -.0409746
99%          233             233       Kurtosis       2.04156
```

```
bysort foreign: sum price, d
```

```
-------------------------------------------------------------------------------
-> foreign = Domestic

                          Price
-------------------------------------------------------------
      Percentiles      Smallest
 1%         3291            3291
 5%         3667            3299
10%         3955            3667       Obs                  52
25%         4184            3799       Sum of wgt.          52

50%       4782.5                       Mean           6072.423
                        Largest       Std. dev.      3097.104
75%         6234           13466
90%        11385           13594       Variance        9592055
95%        13594           14500       Skewness       1.777939
99%        15906           15906       Kurtosis       5.090316

-------------------------------------------------------------------------------
-> foreign = Foreign

                          Price
-------------------------------------------------------------
      Percentiles      Smallest
 1%         3748            3748
 5%         3798            3798
10%         3895            3895       Obs                  22
25%         4499            3995       Sum of wgt.          22

50%         5759                       Mean           6384.682
                        Largest       Std. dev.      2621.915
75%         7140            9690
90%         9735            9735       Variance        6874439
95%        11995           11995       Skewness       1.215236
99%        12990           12990       Kurtosis       3.555178
```

- ***Avantage***: récupération des résultats sous forme de macro rapide.
- ***Inconvénients***: pas de sélection des indicateurs avec l'option `detail`, output pas adapté aux comparaisons.

Extrait de l'aide summarize (`help summarize`)

```
summarize stores the following in r():

Scalars
  r(N)          number of observations
  r(mean)       mean
```

```
      r(skewness)    skewness (detail only)
      r(min)         minimum
      r(max)         maximum
      r(sum_w)       sum of the weights
      r(p1)          1st percentile (detail only)
      r(p5)          5th percentile (detail only)
      r(p10)         10th percentile (detail only)
      r(p25)         25th percentile (detail only)
      r(p50)         50th percentile (detail only)
      r(p75)         75th percentile (detail only)
      r(p90)         90th percentile (detail only)
      r(p95)         95th percentile (detail only)
      r(p99)         99th percentile (detail only)
      r(Var)         variance
      r(kurtosis)    kurtosis (detail only)
      r(sum)         sum of variable
      r(sd)          standard deviation
```

Informations enregistrées pour la variable *price*

```
qui sum price, d

return list
```

```
scalars:
                  r(N) =  74
              r(sum_w) =  74
               r(mean) =  6165.256756756757
                r(Var) =  8699525.97426879
                 r(sd) =  2949.495884768919
           r(skewness) =  1.653433511704859
           r(kurtosis) =  4.819187528464004
                r(sum) =  456229
                r(min) =  3291
                r(max) =  15906
                 r(p1) =  3291
                 r(p5) =  3748
                r(p10) =  3895
                r(p25) =  4195
                r(p50) =  5006.5
                r(p75) =  6342
                r(p90) =  11385
                r(p95) =  13466
                r(p99) =  15906
```

mean

N'affiche que la moyenne et ses statistiques associées. L'option `over` permet de comparer les valeurs moyennes des modalités d'une variable catégorielle (`over(varname)`) ou un croisement des modalités de plusieurs variables (`over(varlist)`)

```
mean price
mean price, over(foreign)
```

```
Mean estimation                          Number of obs = 74

------------------------------------------------------------

             |       Mean   Std. err.     [95% conf. interval]

-------------+----------------------------------------------

       price |

   6165.257    342.872      5481.914    6848.600

------------------------------------------------------------


Mean estimation                            Number of obs = 74

------------------------------------------------------------------
                 |       Mean   Std. err.     [95% conf. interval]
-----------------+------------------------------------------------
c.price@foreign |
      Domestic  |

   6072.423    429.491      5216.449    6928.398
       Foreign  |   6384.682    558.994      5270.608    7498.756
------------------------------------------------------------------
```

- **Avantage**: output synthétique si la moyenne de plusieurs groupes comparées
- **Inconvénients**: récupération des résultats via une matrice (on oublie)

tabstat

Permet de sélectionner les indicateurs avec l'option `stat()` (par défaut la moyenne). L'option `by()` permet de comparer le ou les indicateurs pour chaque niveau d'une variable catégorielle. Dans ce cas `tabstat` affiche également les résultats sur l'ensemble des observations (sinon ajouter l'option `nototal`).

```
tabstat price
tabstat price, by(foreign)
```

```
    Variable |      Mean
-------------+----------
       price |  6165.257
------------------------

Summary for variables: price
Group variable: foreign (Car origin)

 foreign |      Mean
---------+----------
Domestic |  6072.423
 Foreign |  6384.682
---------+----------
   Total |  6165.257
--------------------
```

Extrait de l'aide tabstat (`help tabstat`)

```
       mean            mean
       count           count of nonmissing observations
       n               same as count
       sum             sum
       max             maximum
       min             minimum
       range           range = max - min
       sd              standard deviation
       variance        variance
       cv              coefficient of variation (sd/mean)
       semean          standard error of mean (sd/sqrt(n))
       skewness        skewness
       kurtosis        kurtosis
       p1              1st percentile
       p5              5th percentile
       p10             10th percentile
       p25             25th percentile
       median          median (same as p50)
       p50             50th percentile (same as median)
       p75             75th percentile
       p90             90th percentile
       p95             95th percentile
       p99             99th percentile
```

```
        iqr             interquartile range = p75 - p25
        q               equivalent to specifying p25 p50 p75
```

Si on souhaite ajouter la médiane

```
tabstat mpg, by(foreign) stat(mean median)
```

```
Summary for variables: mpg
Group variable: foreign (Car origin)

 foreign |      Mean       p50
---------+--------------------
Domestic |  19.82692        19
 Foreign |  24.77273      24.5
---------+--------------------
   Total |   21.2973        20
------------------------------
```

5.2.1.2 Graphiques

[**MAJ EN COURS: bcp de nouveautés**]

Juste une rapide présentation de quelques visualisations permettant d'explorer des distributions.

Une seule distribution: **graph box/hbox** , **histogram**, **`violinplot`** (externe)
Plusieurs distributions: **`violinplot`** (externe) Deux distribution croisée: **`hexplot`** (externe), **`gjoint`** (externe)

Boxplot

```
graph hbox mpg,
graph hbox mpg, over(foreign)
```

Histogramme

```
histogram mpg, percent
histogram mpg, percent by(foreign)
```

- Dans le langage de Stata, ce type de graphique est appelé *oneway*.
- L'altération de son aspect avec les options n'est pas très flexible, surtout au niveau des couleurs.
- Pour les histogrammes, il y a une version *twoway* qui permet d'empilé plusieurs histogrammes dans un même graphique, mais généralement la visualisation n'est pas optimale. Préférer en ce cas là une approche par les **densités** (voir `violinplot` - comparaison III).
- Conseil de sémiologie graphique: garder l'axe quantitatif/continu sur les abcisses .

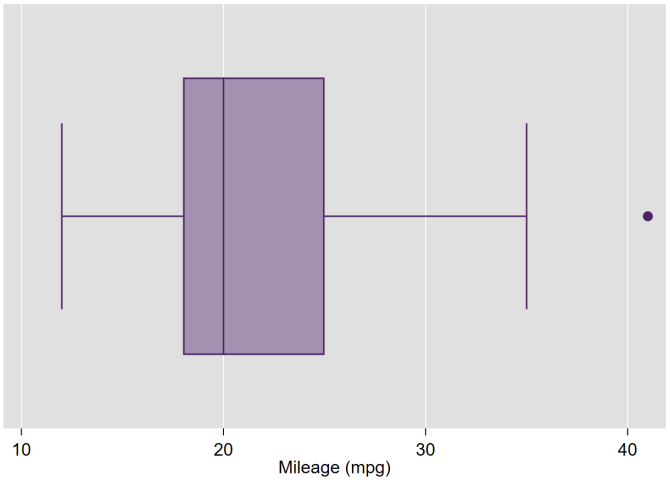


Figure 5.1: Sans comparaison

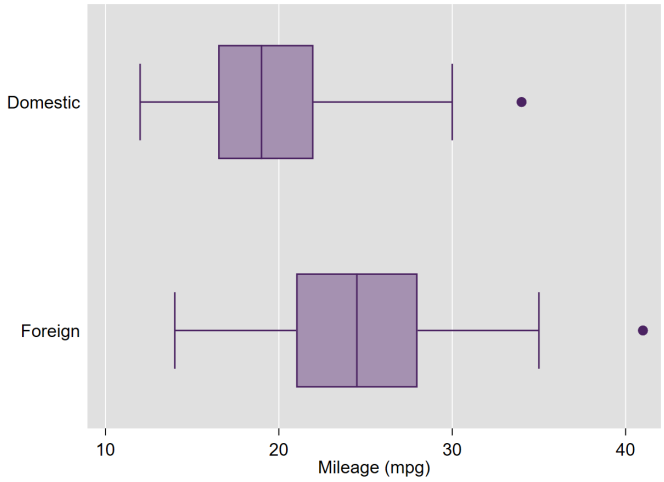


Figure 5.2: Avec comparaison

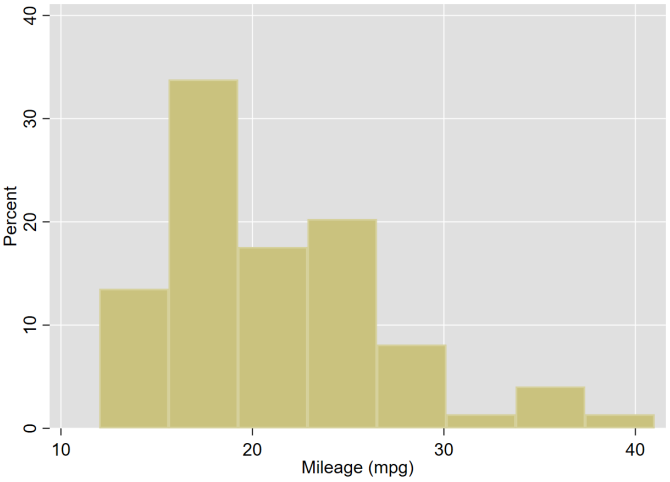


Figure 5.3: Sans comparaison

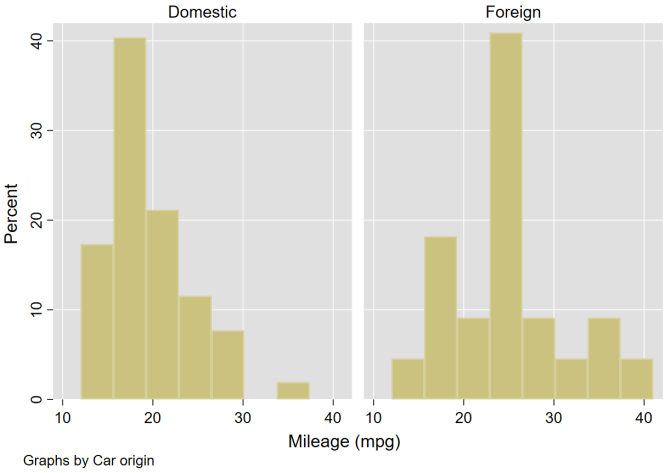


Figure 5.4: Avec comparaison

Violinplot (Ben Jann)

Toujours beaucoup d'options dans les commandes de *magik* B.Jann. Se reporter à son tutoriel sur github pour l'installation (nécessite l'installation de dépendances) Lien

```
violinplot mpg, fill

violinplot mpg, nobox over(foreign) left overlay nomedian  dscale(.)

violinplot mpg, fill over(foreign)

violinplot mpg, fill split(foreign) horizontal
```

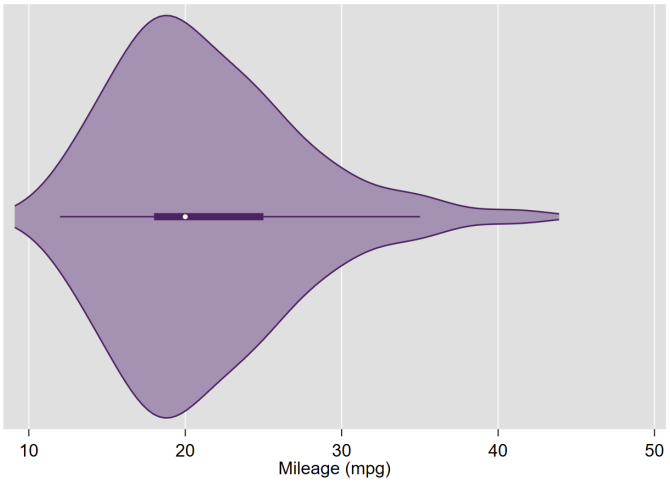


Figure 5.5: Sans comparaison

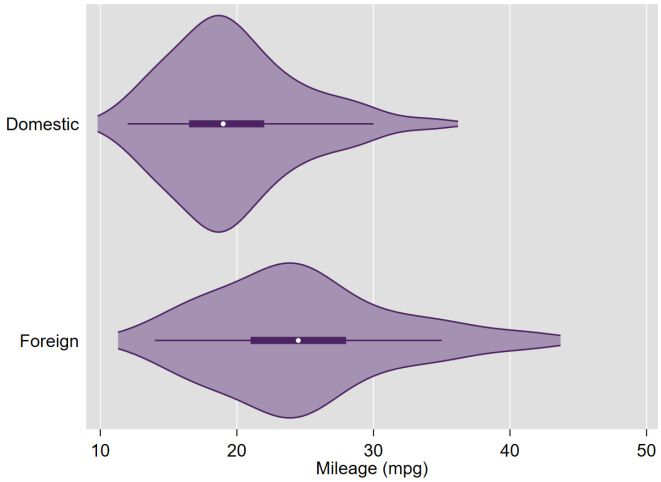


Figure 5.6: Comparaison I

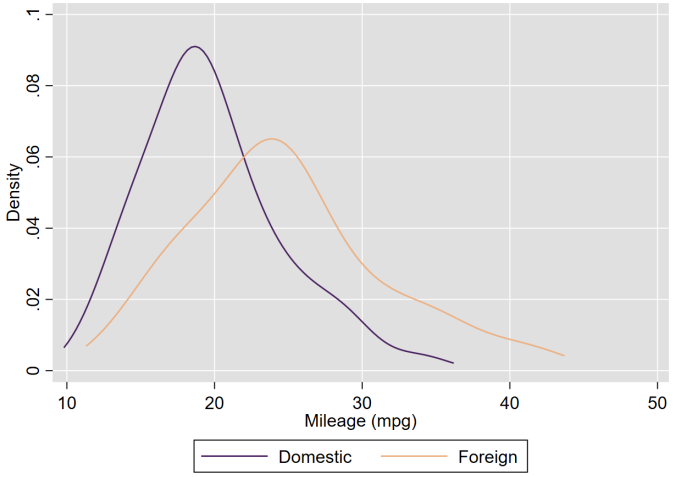


Figure 5.7: Comparaison II

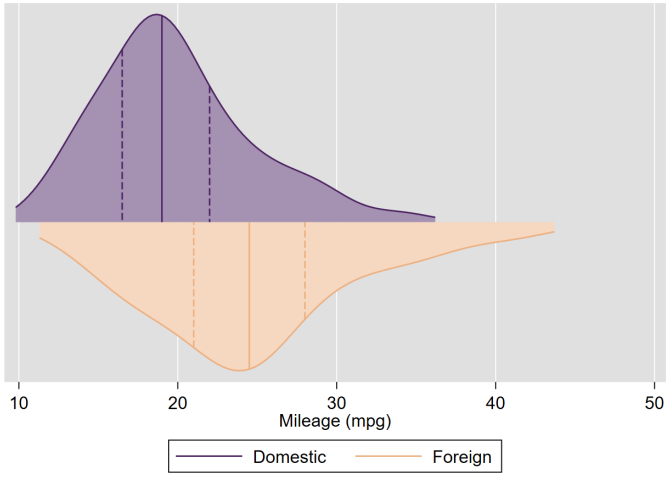


Figure 5.8: Comparaison III

Deux variables quantitatives

- Une nuage de point pêche rapidement lorsque le nombre d'observations augmente, par exemple audelà de 200.

- Solutions:
  - Courbes de niveaux
  - `Heatplot/hexplot`: l'idée est de visualiser un histogramme "vu du dessuss", la hauteurs des barres étant données par un différentiel de couleur issues d'une palette séquentielle (du clair au foncé par exemple).

```
ssc install heatplot, replace
```

- Il peut-être intéressant d'ajouté les distributions marginales des deux variables. J'ai programmé une petite commande (encore en version très alpha): **`gjoint`**. Tout le mérite revient à B.Jann pour la commande `hexplot` (j'ai juste combiné `hexplot` avec des histogrammes).

```
net install gjoint, from("https://raw.githubusercontent.com/mthevenin/stata_graphiques/master
```

```
hexplot price mpg, colors(flare)
gjoint price mpg, palette(flare)
```

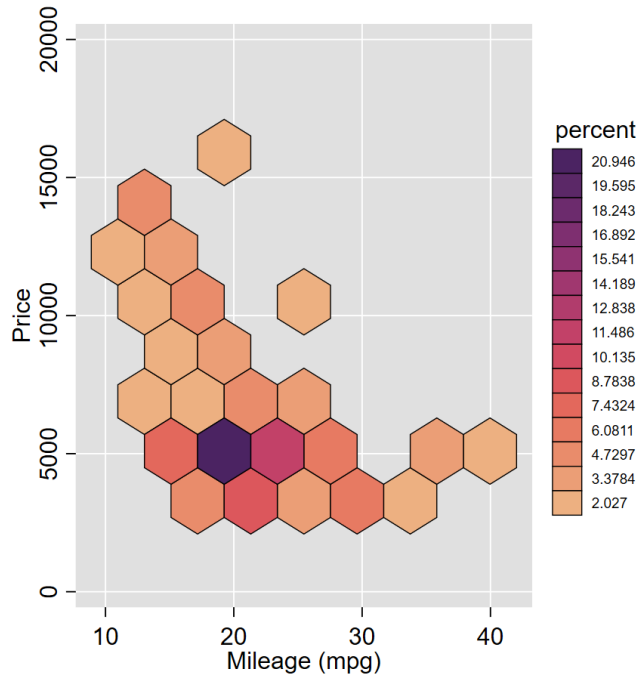


Figure 5.9: hexplot

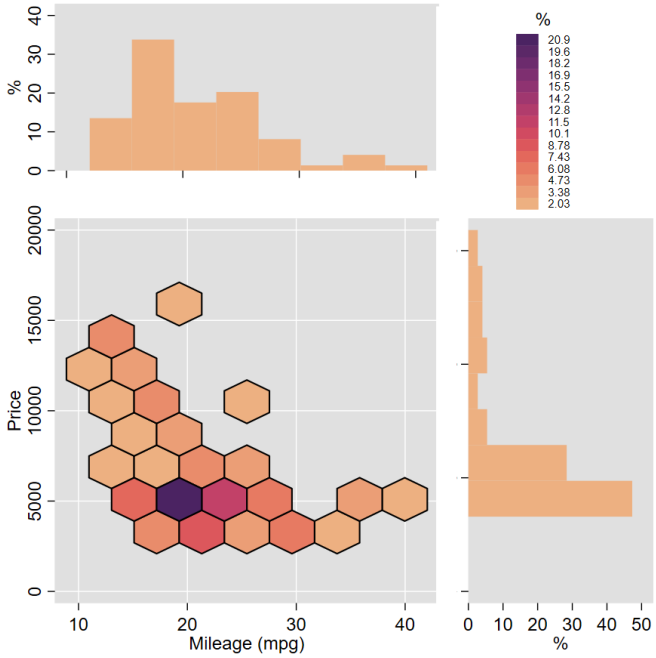


Figure 5.10: hexplot

5.2.2 Variables catégorielles

La principale commande est `tabulate` (`tab`). On peut l'utiliser avec des variables de type string.

Syntaxe (tableau croisé):

```
tab var1 var2 [, mis nofreq row col sort ......]
```

- Par défaut, l'ordre d'affichage suis la valeur de la modalité si la variable est de type numérique et l'ordre alphabétique pour une variable de type caractère. On peut utiliser l'option `sort` pour afficher par ordre croissant des effectifs observés [ou utiliser la commande externe **`tabsort`** qui offre plus de possibilités]
- Autres commandes (externe): **`fre`** [B.Jann], **`tabm`** [NJ.Cox], **`tabsort`** [NJ.Cox]

```
ssc install fre
ssc install tabsort
ssc install tabm
```

```
tab rep78, mis
tab rep78, mis sort
tab rep78 foreign, nolab mis

fre rep78

tabsort rep78
```

```
     Repair |
record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |          2        2.70        2.70
          2 |          8       10.81       13.51
          3 |         30       40.54       54.05
          4 |         18       24.32       78.38
          5 |         11       14.86       93.24
          . |          5        6.76      100.00
------------+-----------------------------------
      Total |         74      100.00

     Repair |
record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
          3 |         30       40.54       40.54
          4 |         18       24.32       64.86
          5 |         11       14.86       79.73
          2 |          8       10.81       90.54
          . |          5        6.76       97.30
          1 |          2        2.70      100.00
------------+-----------------------------------
      Total |         74      100.00
```

```
    Repair |
    record |      Car origin
      1978 |         0          1 |     Total
-----------+----------------------+----------
         1 |         2          0 |         2
         2 |         8          0 |         8
         3 |        27          3 |        30
         4 |         9          9 |        18
         5 |         2          9 |        11
         . |         4          1 |         5
-----------+----------------------+----------
     Total |        52         22 |        74
```

```
rep78 -- Repair record 1978
-------------------------------------------------------------
              |      Freq.    Percent      Valid       Cum.
--------------+----------------------------------------------
Valid   1     |          2       2.70       2.90       2.90
        2     |          8      10.81      11.59      14.49
        3     |         30      40.54      43.48      57.97
        4     |         18      24.32      26.09      84.06
        5     |         11      14.86      15.94     100.00
        Total |         69      93.24     100.00
Missing .     |          5       6.76
Total         |         74     100.00
-------------------------------------------------------------
```

```
     Repair |
record 1978 |      Freq.     Percent        Cum.
------------+-----------------------------------
          3 |         30       43.48       43.48
          4 |         18       26.09       69.57
          5 |         11       15.94       85.51
          2 |          8       11.59       97.10
          1 |          2        2.90      100.00
------------+-----------------------------------
      Total |         69      100.00
```

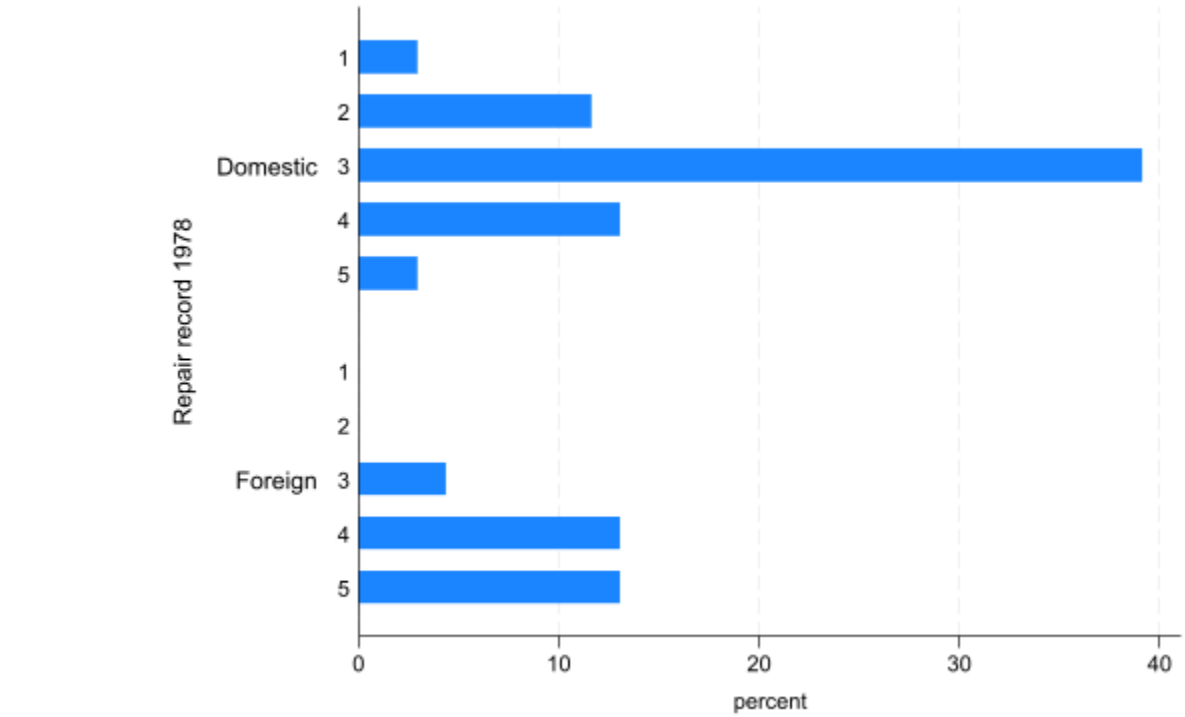
Graphiques

Niveau graphique, les possibilités restent toujours assez limitées pour les variables discrètes (et on abandonne l'idée des horribles camemberts).

On privilégiera ici la commande de NJ.Cox, **`catplot`**

```
ssc install catplot
```

```
catplot rep78, percent
catplot rep78, percent over(foreign)
```



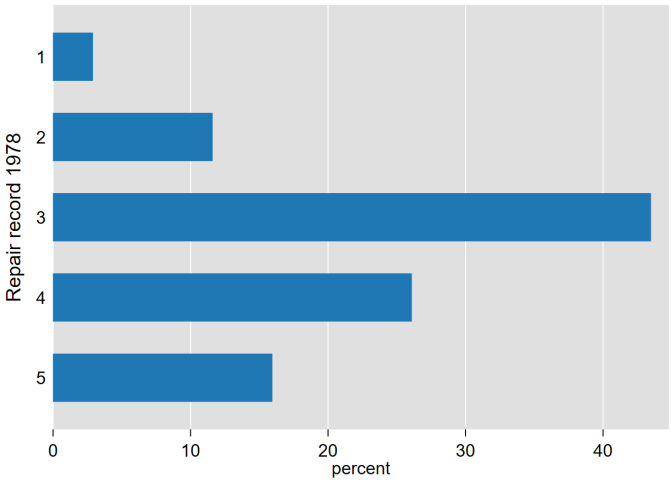


Figure 5.11: Sans comparaison

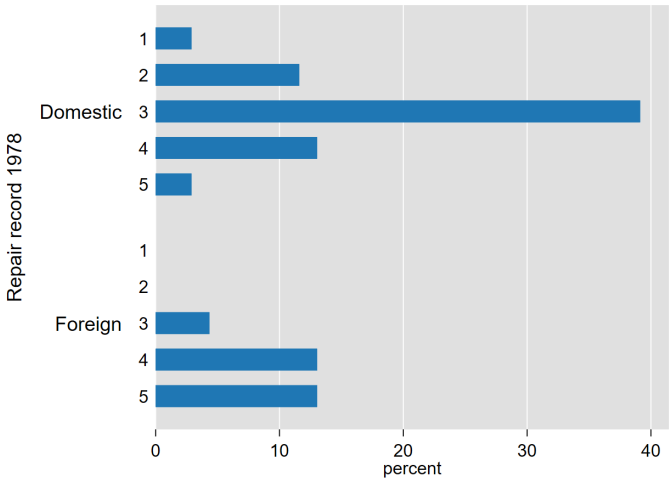


Figure 5.12: Avec comparaison

5.3 Introduction aux frames

- Depuis la version 16 (2019)
- Les frames permettent de travailler en parallèle sur plusieurs bases, sans switcher avec des opérations successives d'enregistrement/ouverture (`save`/`use`).
  - Sur l'interface principale, le contenu d'une frame (base de données) est affiché de manière traditionnelle. On peut faire des opérations sur les autres frames déclarées simultanément.
  - La première base ouverte lors de l'ouverture d'une session est déclarée comme frame par défaut.
- Les frames peuvent être liées entre elles avec une clé d'identification commune.
  - Importations partielles de variables ou d'observations d'une frame à l'autre.
  - Permet de générer une variable dans une frame en utilisant des variables d'une ou plusieurs autres frames. Il n'est donc pas nécessaire d'apparier les bases entre elles en amont.

On ne verra ici que quelques manipulations de bases, la liaison de frames sera traitée dans le chapitre 6.

`frame dir` ou **`frame mist`**

```
frame dir
frame list
```

```
  default  74 x 12; 1978 automobile data
  default  74 x 12; 1978 automobile data
```

```
sysuse auto, clear
frame dir
```

```
(1978 automobile data)
  default  74 x 12; 1978 automobile data
```

`frame rename`

On renomme une frame avec `ancien nom nouveau nom`

```
frame reset   // voir plus loin - commande ici seulement nécessaire pour compatibilité avec C
sysuse auto, clear
frame rename default auto

frame dir
```

```
(1978 automobile data)
  auto  74 x 12; 1978 automobile data
```

> **Note**
>
> Autre façon de procéder (et surement meilleure):

```
frame create auto
frame auto: sysuse auto, clear
```

`frame nom_frame:` ou **`frame nom_frame {}`**

- On peut exécuter une commande en la préfixant par **`frame nom_frame:`**
- Pour une série de commandes, il suffit d'enchasser cette série dans des crochets

```
frame auto: mean price
```

```
Mean estimation                           Number of obs = 74

------------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+----------------------------------------------------
       price |   6165.257    342.872      5481.914    6848.600
------------------------------------------------------------------
```

```
frame auto {
mean mpg
table rep78 foreign
}
```

```
Mean estimation                           Number of obs = 74

------------------------------------------------------------------
             |       Mean   Std. err.     [95% conf. interval]
-------------+----------------------------------------------------
         mpg |     21.297      0.673        19.957      22.638
------------------------------------------------------------------
```

```
-------------------------------------------------
                  |          Car origin
                  |  Domestic   Foreign   Total
------------------+------------------------------
Repair record 1978 |
```

```
  1                 |         2                2
  2                 |         8                8
  3                 |        27          3     30
  4                 |         9          9     18
  5                 |         2          9     11
  Total             |        48         21     69
------------------------------------------------
```

frame copy

Permet de copier à l'idendique une frame **frame copy nom_frame nouveau_nom**.

Dans l'exemple qui suit on va une frame à partir de la base auto, frame *prix*, en conservant avec seulement les variables *foreign* et *price*.

```
frame copy auto prix

frame dir
```

```
  auto  74 x 12; 1978 automobile data
  prix  74 x 12; 1978 automobile data
```

Comme indiqué précédemment, on est pas obligé de charger une base déclarée en frame pour effectuer des opérations dessus. On peut donc conserver les deux seules variables *foreign* et *price* tout en gardant la base *auto* chargée.

```
frame prix: keep foreign price
frame dir
```

```
  auto  74 x 12; 1978 automobile data
* prix  74 x 2; 1978 automobile data

Note: Frames marked with * contain unsaved data.
```

La frame *prix* ne comporte donc plus que deux variables.

On va supprimer la variable *make* de la base/frame *auto*

```
drop make
```

frame change

Permet de switcher d'une frame à une autre. Ici ce sont les informations de la frame *prix* qui seront chargée dans l'interface de Stata

```
frame change prix

mean price, over(foreign)
```

```
Mean estimation                                 Number of obs = 74

------------------------------------------------------------------
                |       Mean   Std. err.     [95% conf. interval]
----------------+-------------------------------------------------
c.price@foreign |
      Domestic  |   6072.423    429.491      5216.449    6928.398
       Foreign  |   6384.682    558.994      5270.608    7498.756
------------------------------------------------------------------
```

Du côté de la frame *auto*, base chargée initialement, on note que la suppression de la variable *make* a bien été enregistrée malgré le change de frame.

```
frame auto: des
```

```
Contains data from C:\Program Files\Stata18/ado\base/a/auto.dta
 Observations:            74                  1978 automobile data
    Variables:            11                  13 Apr 2022 17:45
                                              (_dta has notes)
------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
price           int     %8.0gc                Price
mpg             int     %8.0g                 Mileage (mpg)
rep78           int     %8.0g                 Repair record 1978
headroom        float   %6.1f                 Headroom (in.)
trunk           int     %8.0g                 Trunk space (cu. ft.)
weight          int     %8.0gc                Weight (lbs.)
length          int     %8.0g                 Length (in.)
turn            int     %8.0g                 Turn circle (ft.)
displacement    int     %8.0g                 Displacement (cu. in.)
gear_ratio      float   %6.2f                 Gear ratio
foreign         byte    %8.0g      origin     Car origin
------------------------------------------------------------------------------
Sorted by: foreign
     Note: Dataset has changed since last saved.
```

`frame drop` et **`frame reset`**

Permettent de supprimer une ou la totalité des frames

`frame drop nom_frame`, permet de supprimer une frame, à l'exception de celle chargée dans l'interface.

```
frame change auto

frame drop prix

frame dir
```

```
* auto  74 x 11; 1978 automobile data

Note: Frames marked with * contain unsaved data.
```

On a chargé dans l'interface la frame *auto*, puis on a supprimé la frame prix. **Par contre il n'est pas possible de supprimer la frame active dans l'interface**.

> **i** Stata 18
>
> Avec la nouvelle version, il est possible de supprimer plusieurs frames avec `frame drop`, mais toujours à l'exception de celle qui est active.

`frame reset`

On peut supprimer toutes les frames, dont celle chargée dans l'interface avec **`frame reset`**. Dans ce cas il n'y a plus de base chargée dans la session.

```
frame reset
sysuse auto, clear
frame rename default auto
```

```
(1978 automobile data)
```

```
frame reset

des
```

```
Contains data
 Observations:             0
    Variables:             0
Sorted by:
```

6 Les variables

Programme du chapitre

Commandes et expressions introduites

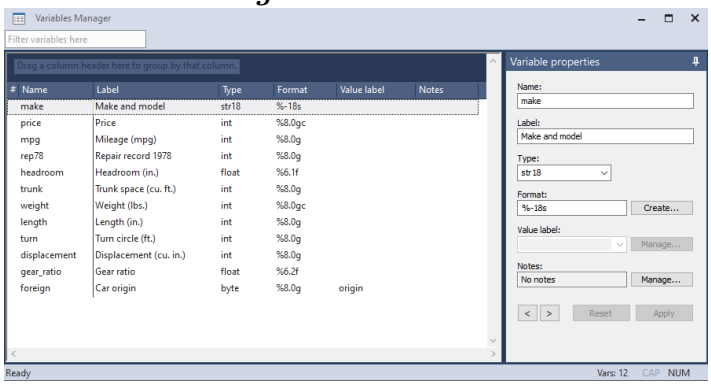
SECTION	COMMANDES
Types-formats	`recast` - `format` - `tostring` - `destring` - `decode` - `encode` - **`sencode`**
Création variable	`generate` - `replace` - `egen` [+ exemples de fonctions associées)
Variables de comptage	`_n` et `_N`
Sélection et recodage	`inlist` - `inrange` - `recode`
labels	`label variable` - `label define` - `label value` - `label list` - `label drop` - **`xaxis`**

- **En gras, commandes externes**

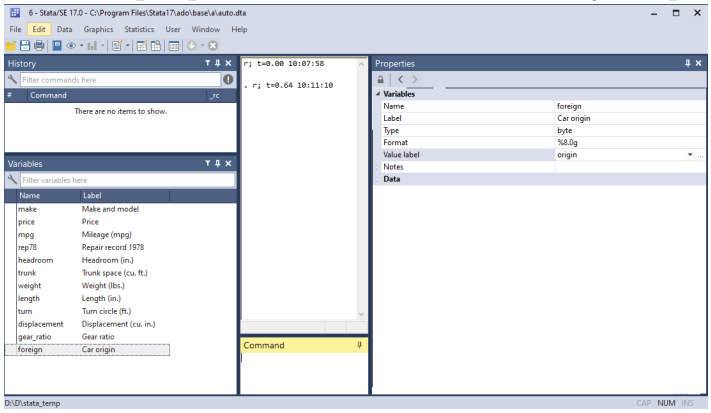
Pour accéder aux infomations sur les variables d'une base:

- Utiliser le **variables manager** ou la fenêtre d'information **variables properties** si elle est ancrée à l'interface. Ces deux outils permettent de faire des modifications et de récupérer la ligne de commande dans la fenêtre *command*. Le verrou de la fenêtre **variables properties** doit-être retiré .
- Utiliser la commande `describe` [`des`] ou la commande `ds` pour un usage plus avancé (sélection de variables selon leur type par exemple).

Variables manager



Fenêtre properties à droite de l'interface principale (mode dévérouillé)



Commande describe

```
sysuse auto, clear

describe
```

```
<IPython.core.display.HTML object>


(1978 automobile data)

Contains data from C:\Program Files\Stata18/ado\base/a/auto.dta
 Observations:            74                  1978 automobile data
    Variables:            12                  13 Apr 2022 17:45
                                              (_dta has notes)
------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
make            str18   %-18s                 Make and model
price           int     %8.0gc                Price
mpg             int     %8.0g                 Mileage (mpg)
rep78           int     %8.0g                 Repair record 1978
headroom        float   %6.1f                 Headroom (in.)
trunk           int     %8.0g                 Trunk space (cu. ft.)
weight          int     %8.0gc                Weight (lbs.)
length          int     %8.0g                 Length (in.)
turn            int     %8.0g                 Turn circle (ft.)
displacement    int     %8.0g                 Displacement (cu. in.)
gear_ratio      float   %6.2f                 Gear ratio
foreign         byte    %8.0g       origin    Car origin
------------------------------------------------------------------------------
Sorted by: foreign
```

6.1 Types et format

6.1.1 Types

Stata gère tous les types de variables: numérique, caractère, date. Un type de variable est un type de stockage.

- Types numériques: *float*, *long*, *double*, *int* et *byte*.
- Types caractère: *str#* et *strL* (très grandes chaînes de caractères). # est la longueur de la chaîne de caractère, elle ne peut pas excéder 2046 pour le type *str*.

Plus d'informations: `help data types`

Modification du type de variable

- Optimisation du poids en mémoire avec `compress`
- Commande **`recast`**

Optimisation du poids de la base

```
compress
```

```
  variable mpg was int now byte
  variable rep78 was int now byte
  variable trunk was int now byte
  variable turn was int now byte
  variable make was str18 now str17
  (370 bytes saved)
```

Passage de la variable make en str3 avec `recast`

En réduisant le type, on va tronquer les chaînes de caractères qui ne garderont que les 3 premières lettres, à manipuler avec prudence donc. Pour cette opération, Stata impose une confirmation avec l'option `force`.

Variable d'origine

```
des make
list make in 1/10
```

```
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
make            str17   %-17s                 Make and model

     +---------------+
```

```
     | make          |
     |---------------|
  1. | AMC Concord   |
  2. | AMC Pacer     |
  3. | AMC Spirit    |
  4. | Buick Century |
  5. | Buick Electra |
     |---------------|
  6. | Buick LeSabre |
  7. | Buick Opel    |
  8. | Buick Regal   |
  9. | Buick Riviera |
 10. | Buick Skylark |
     +---------------+
```

Modification du type

```
recast str3 make, force
des make
list make in 1/10
```

```
make:  74 values changed

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
make            str3    %-9s                  Make and model

     +------+
     | make |
     |------|
  1. | AMC  |
  2. | AMC  |
  3. | AMC  |
  4. | Bui  |
  5. | Bui  |
     |------|
  6. | Bui  |
  7. | Bui  |
  8. | Bui  |
  9. | Bui  |
 10. | Bui  |
     +------+
```

6.1.2 Format

Il s'agit du format d'affichage des valeurs des variables. Ils peuvent être modifiés sans que le type soit changé (décimales, alignement….).

Variables numérique:
- Format *g*: *général* (définition un peu obsure selon moi). - Format *f*: *fixe*. - Plusieurs format d'affichage pour les variables de type dates: *%td* (date avec jour-mois-année), *%tm* (mois), *%tq* (trimestre), *%tw* (semaine). Les dates et leur manipulation sont un domaines très riche, et feront l'objet d'une courte présentation en fin de chapite.

On peut changer le format d'affichage avec la commande **`format`**. Si le format est de type général (g), il est préférable de passer à un format de type fixe (f). On peut affecter un même format à une liste de variables.

Syntaxe:

```
format %format varlist
```

Exemple: changement du nombre de décimales

Dans la base *auto*, la variable *gear_ratio* est de format fixe à 2 décimales (%6.2f). Pour supprimer, à l'affichage, les deux décimales:

```
list gear_ratio in 1/10
```

```
     +----------+
     | gear_r~o |
     |----------|
  1. |     3.58 |
  2. |     2.53 |
  3. |     3.08 |
  4. |     2.93 |
  5. |     2.41 |
     |----------|
  6. |     2.73 |
  7. |     2.87 |
  8. |     2.93 |
  9. |     2.93 |
 10. |     3.08 |
     +----------+
```

```
format %6.0f gear_ratio
list gear_ratio in 1/10
```

```
     +----------+
     | gear_r~o |
     |----------|
  1. |        4 |
  2. |        3 |
  3. |        3 |
  4. |        3 |
  5. |        2 |
     |----------|
  6. |        3 |
  7. |        3 |
  8. |        3 |
  9. |        3 |
 10. |        3 |
     +----------+
```

Exemple: aligner le nombre décimal reporté avec `summarize` sur le format de la variable

Avec l'option `format` appliquée à la commande `summarize` on peut automatiquement réduire le nombre de décimales reportées dans l'output

```
sum gear_ratio
sum gear_ratio, d
```

```
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
  gear_ratio |         74    3.014865    .4562871       2.19       3.89

                          Gear ratio
-------------------------------------------------------------
      Percentiles      Smallest
 1%         2.19           2.19
 5%         2.28           2.24
10%         2.43           2.26       Obs                  74
25%         2.73           2.28       Sum of wgt.          74

50%        2.955                      Mean           3.014865
                        Largest       Std. dev.      .4562871
75%         3.37           3.78
90%         3.72           3.78       Variance       .2081979
95%         3.78           3.81       Skewness       .2191658
99%         3.89           3.89       Kurtosis       2.101812
```

Avec l'option **`format`**

```
sum gear_ratio, format
sum gear_ratio, d format
```

```
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
  gear_ratio |         74           3           0          2          4

                          Gear ratio
-------------------------------------------------------------
      Percentiles      Smallest
 1%            2              2
 5%            2              2
10%            2              2       Obs                  74
25%            3              2       Sum of wgt.          74

50%            3                      Mean                  3
                        Largest       Std. dev.             0
75%            3              4
90%            4              4       Variance              0
95%            4              4       Skewness              0
99%            4              4       Kurtosis              2
```

6.2 Modification du type

Il est possible de basculer d'un type caractère à un type numérique et inversement

De numérique à caractère

Si la variable numérique n'a pas de label affecté sur les modalités, ou qu'on ne veut pas conserver l'information données par les labels, on peut utiliser la commande `tostring`. On peut créer une nouvelle variable avec l'option `gen()` ou remplacer la variable numérique d'origine avec l'option `replace`. Une des deux options doit être nécessairement renseignée.

Syntaxe:

```
  tostring [varlist], gen(nom_varlist)
  tostring [varlist], replace
```

Exemple avec la variable *foreign* qui prend les valeurs 0 ou 1 avec les labels "domestic" (0) et "foreign" (1)

```
tostring foreign, gen(foreign_str)
des foreign foreign_str

tab foreign foreign_str
```

```
foreign_str generated as str1

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g      origin     Car origin
foreign_str     str1    %9s                   Car origin

           |      Car origin
Car origin |         0          1 |     Total
-----------+----------------------+----------
  Domestic |        52          0 |        52
   Foreign |         0         22 |        22
-----------+----------------------+----------
     Total |        52         22 |        74
```

Si la variable numérique a des labels affectés aux modalités modalités et qu'on souhaite conserver cet information, on utilise la commande **decode**

Syntaxe:

```
decode variable, gen(nom_var)
```

Exemple avec la variable *foreign*

```
capt drop foreign_str

decode foreign, gen(foreign_str)

des foreign foreign_str

tab foreign_str
```

```
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g      origin     Car origin
foreign_str     str8    %9s                   Car origin

 Car origin |      Freq.     Percent        Cum.
------------+-----------------------------------
   Domestic |         52       70.27       70.27
    Foreign |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00
```

De caractère à numérique

Si la variable caractère est a une forme numerique (une suite de nombre comme des années, des âges…), on utilise la commande `destring`. Lorsqu'il y a des des valeurs manquantes à la variable, on doit uiliser l'option `force`.

Syntaxe:

```
destring [varlist] , gen(nom_varlist)
destring [varlist] , replace [force]
```

Exemple avec la variable *rep78* qui est transformé dans un premier temps en variables caractère avec `tostring` puis de nouveau transformé en format numérique avec `destring`

```
tostring rep78, replace
des rep78

destring rep78, replace
des rep78
```

```
rep78 was byte now str1

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
rep78           str1    %9s                   Repair record 1978
rep78: all characters numeric; replaced as byte
(5 missing values generated)


Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
rep78           byte    %10.0g                Repair record 1978
```

Si la variable caractère n'est pas de forme numérique et que l'on souhaite récupérer les labels sur les modalités, on peut utiliser la commande **`encode`** ou la commande externe **`sencode`** (`net install st0043_2, force`). La seconde permet de remplacer directement la variable d'origine, option particulièrement pratique. Egalement, cette commande permet plus de souplesse sur le codage de la variable (`help sencode` pour plus de détail).

Avec `encode` ou `sencode` sans l'option `gsort`, le numéro de la modalité suivra l'ordre alphabétique des chaînes de caractère de la variable: si la variable caractère à pour valeur ("Homme", "Femme"), "femme" sera automatiquement codée 1 et "homme" 2.

Syntaxe:

```
encode variable, gen(nom_variable)
```

Syntaxe:

```
sencode variable, gen(nom_variable) replace gsort()
```

Exemple avec la variable *foreign_str* (variable caractère créée précédemment à partir de la variable foreign)

```
encode foreign_str, gen(foreign2)

tab foreign2
tab foreign2, nolab
```

```
 Car origin |      Freq.     Percent        Cum.
------------+-----------------------------------
   Domestic |         52       70.27       70.27
    Foreign |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00

 Car origin |      Freq.     Percent        Cum.
------------+-----------------------------------
          1 |         52       70.27       70.27
          2 |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00
```

> ⚠ Type de variable pour les modèles
>
> Les variables de type caractères ne sont pas acceptées, Stata renvoie alors un message d'erreur avec **no observation**. Si c'est le cas, les commandes `destring` et `encode` vont s'avérer particulièrement utiles.

6.3 Création d'une variable

6.3.1 generate - replace

La création d'une nouvelle variable se fait avec la commande `generate` généralement tronquée jusqu'à `gen` (voire `ge` pour les plus radicaux de la troncature de syntaxe).

Syntaxe:

```
gen nom_variable=valeur/fonction [expression: if in inlist inrange...]
```

Pour remplacer la valeur variable existante on utilise la commande `replace`. Le nom n'est malheureusement pas tronquable.

Syntaxe:

```
replace nom_variable=valeur/fonction [expression: if in inlist inrange...]
```

- On peut utiliser le préfixe `bysort`
- Pour utiliser une fonction mathématique (log, exp,….) => `help math_functions`
- Pour afficher la liste complète des fonctions (variables caractères, statistiques, pseudo nombre aléatoire, dates.): `help function`

Rappel: attention entre l'opérateur d'affectation (=) et l'expression conditionnelle (==).

```
gen     x=valeur if y==valeur
replace x=valeur if y==valeur
```

Création d'une indicatrice (0,1)

On peut rapidement générer des indicatrices (0,1) à partir d'une expression conditionnelle:

Syntaxe:

```
gen x= expression_conditionnelle
```

Exemple avec la variable *rep78*. On génère la variable *rep2* qui prend la valeur 1 si *rep78*>3, 0 sinon. Comme il y a des valeurs manquantes dans la variable d'origine, on corrige l'information pour l'indicatrice dont les valeurs manquantes ont été automatiquement affectées à la valeur 0.

```
gen rep2 = rep78>3
replace rep2 = . if rep78==.
tab rep78 rep2
```

```
(5 real changes made, 5 to missing)

    Repair |
    record |         rep2
      1978 |         0          1 |     Total
-----------+----------------------+----------
         1 |         2          0 |         2
         2 |         8          0 |         8
         3 |        30          0 |        30
         4 |         0         18 |        18
         5 |         0         11 |        11
-----------+----------------------+----------
     Total |        40         29 |        69
```

Remarque: Avec la commande `tabulate` on peut créer une série d'indicatrices à partir d'une variable catégorielle avec l'option `gen(nom_variable)`

Syntaxe:

```
tab x, gen(nom_variable)
```

Exemple avec la variable *foreign*. Avec `tabulate` on va générer deux indicatrices: *origine1* si *foreign*=0, et *origine2* si *foreign*=1. Un label à la variable est automatiquement créé indiquant la valeur de la variable d'origine.

```
tab foreign, gen(origine)

des origine1 origine2

tab origine1 foreign
tab origine2 foreign
```

```
 Car origin |      Freq.     Percent        Cum.
------------+-----------------------------------
   Domestic |         52       70.27       70.27
    Foreign |         22       29.73      100.00
------------+-----------------------------------
      Total |         74      100.00

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
origine1        byte    %8.0g                 foreign==Domestic
origine2        byte    %8.0g                 foreign==Foreign

foreign==D |      Car origin
   omestic |  Domestic    Foreign |     Total
-----------+----------------------+----------
         0 |         0         22 |        22
         1 |        52          0 |        52
-----------+----------------------+----------
     Total |        52         22 |        74

foreign==F |      Car origin
    oreign |  Domestic    Foreign |     Total
-----------+----------------------+----------
         0 |        52          0 |        52
         1 |         0         22 |        22
-----------+----------------------+----------
     Total |        52         22 |        74
```

6.3.2 egen

egen: extented generate
egenmore: package programmé par NJ.Cox qui ajoute des fonctions associée à `egen` [`ssc install egenmore`]. On utilise la commande `egen` une fois le package installé.

Réservé à l'utilisation de fonctions. Pour obtenir la liste `help egen` ou `help egenmore`.

Exemple: on va créer dans un premier la variable *mprice* qui reporte pour chaque observation la moyenne de la variable *price*. Dans un second temps, on va créer la variable *mprice_or*, mais avec le prix moyen des voitures selon leur origine (*foreign*). La fonction utilisée est la fonction **`mean()`**.

```
egen mprice = mean(price)

list  make price mprice in 1/10
```

```
     +--------------------------+
     | make    price     mprice |
     |--------------------------|
  1. | AMC     4,099   6165.257 |
  2. | AMC     4,749   6165.257 |
  3. | AMC     3,799   6165.257 |
  4. | Bui     4,816   6165.257 |
  5. | Bui     7,827   6165.257 |
     |--------------------------|
  6. | Bui     5,788   6165.257 |
  7. | Bui     4,453   6165.257 |
  8. | Bui     5,189   6165.257 |
  9. | Bui    10,372   6165.257 |
 10. | Bui     4,082   6165.257 |
     +--------------------------+
```

```
bysort foreign: egen mprice_or = mean(price)

list  make price mprice in 1/5
list  make price mprice in 66/70
```

```
     +-------------------------+
     | make   price     mprice |
     |-------------------------|
  1. | AMC    4,099   6165.257 |
  2. | AMC    4,749   6165.257 |
  3. | AMC    3,799   6165.257 |
  4. | Bui    4,816   6165.257 |
```

```
 5. | Bui    7,827   6165.257 |
    +-------------------------+

    +-------------------------+
    | make   price     mprice |
    |-------------------------|
66. | Sub    3,798   6165.257 |
67. | Toy    5,899   6165.257 |
68. | Toy    3,748   6165.257 |
69. | Toy    5,719   6165.257 |
70. | VW     7,140   6165.257 |
    +-------------------------+
```

⚠ Les fonctions `sum()` et `total()`

Sans aucune justification, la fonction `sum()` qui permet d'obtenir une somme incrémentale n'est pas associée à `egen` mais à `generate`. En revanche la fonction `total()` est associée à `egen`. Il faut s'en souvenir.

Exemple: on veut créer un identifiant numérique (variable *id*) pour chaque voiture (dans la base on a une seule voiture par nom de voiture).

```
gen x = 1
gen id = sum(x)

list id make in 1/5
list id make in 66/70
```

```
    +-----------+
    | id   make |
    |-----------|
 1. |  1   AMC  |
 2. |  2   AMC  |
 3. |  3   AMC  |
 4. |  4   Bui  |
 5. |  5   Bui  |
    +-----------+

    +-----------+
    | id   make |
    |-----------|
66. | 66   Sub  |
67. | 67   Toy  |
68. | 68   Toy  |
69. | 69   Toy  |
70. | 70   VW   |
    +-----------+
```

Si on veut reporter le nombre total d'observations dans la base (variable N), avec la fonction `total`:

```
egen N = total(x)

list id N make in 1/5
list id N make in 66/70
```

```
     +----------------+
     | id    N   make |
     |----------------|
  1. |  1   74   AMC  |
  2. |  2   74   AMC  |
  3. |  3   74   AMC  |
  4. |  4   74   Bui  |
  5. |  5   74   Bui  |
     +----------------+

     +----------------+
     | id    N   make |
     |----------------|
 66. | 66   74   Sub  |
 67. | 67   74   Toy  |
 68. | 68   74   Toy  |
 69. | 69   74   Toy  |
 70. | 70   74   VW   |
     +----------------+
```

On va le voir, ces deux variables qui viennent d'être générées peuvent l'être directement avec des *variables internes dites de comptage*.

Package gegen (M.Caceres)

- Pour les volumétries dépassant le million d'observations il est fortement conseillé d'utiliser la commande **gegen** associée au package de *Mauricio Caceres* **gtools**:
- https://gtools.readthedocs.io/en/latest/
- https://mthevenin.github.io/stata_programmation/speedup/gtools.html

7 Les variables internes de comptage

Très utile avec des données longitudinales, de durées ou toute base avec des données avec clusters.

Deux variables de comptage: `_n` et `_N`

- **`_n`** : renvoie le rang de l'observation
- **`_N`**: renvoie le nombre total d'observations

Application: On veut créer un identifiant sur l'ensemble des observations de la base auto.

```
capture drop id

gen id = _n
list make foreign id in 1/10
```

```
     +----------------------+
     | make    foreign   id |
     |----------------------|
  1. | AMC    Domestic    1 |
  2. | AMC    Domestic    2 |
  3. | AMC    Domestic    3 |
  4. | Bui    Domestic    4 |
  5. | Bui    Domestic    5 |
     |----------------------|
  6. | Bui    Domestic    6 |
  7. | Bui    Domestic    7 |
  8. | Bui    Domestic    8 |
  9. | Bui    Domestic    9 |
 10. | Bui    Domestic   10 |
     +----------------------+
```

On peut associer `bysort` à la création de ce type de variables, par exemple pour générer un id unique aux personnes composant un ménage.

lead & lag

Par défaut, pour toutes les instructions le rang sélectionné est celui de la ligne de l'observation `x=x[_n]`

- Lag(-1): Sélection du rang inférieur d'une variable x: $x = x[_n-1]$. La première observation est une valeur manquante.
- lead(+1): Sélection du rang supérieur d'une variable x: $x = x[_n-1]$. La dernière observation est une valeur manquante.

```
gen lag_make  = make[_n-1]
gen lead_make = make[_n+1]

list make lag_make lead_make in 1/10
```

```
(1 missing value generated)
(1 missing value generated)
```

	make	lag_make	lead_m~e
1.	AMC		AMC
2.	AMC	AMC	AMC
3.	AMC	AMC	Bui
4.	Bui	AMC	Bui
5.	Bui	Bui	Bui
6.	Bui	Bui	Bui
7.	Bui	Bui	Bui
8.	Bui	Bui	Bui
9.	Bui	Bui	Bui
10.	Bui	Bui	Cad

7.1 Sélection de plusieurs modalités, recodage

inlist
Pour sélectionner plusieurs modalités d'une variable dans une expression conditionnelle `inlist(variable, valeur1, valeur2,..)`.
Utile si la variable n'est pas de type ordinale.
Exemple: `gen Y=1 if inlist(X,1,3,6,10)` L'expression reste hélas toujours limitée à 10 valeurs pour les variables caractères.

inrange
Pour sélectionner un intervalle dans une expression `inrange(variable, valeur1,valeur2)` avec $valeur1 < valeur2$.
Exemple: `gen Y= inrange(X,5,10)` pour obtenir $Y = 1$ si $50 \leq x \leq 10$ 0 sinon.

recode
Permet de changer les valeurs d'une variable numérique:

`recode variable (anciennes_valeurs=nouvelle_valeur) (ancienne_valeurs=nouvelles_valeur).....`

Application: recoder la variable foreign 0=>1 et 1=>2.

7.2 Les labels

La création et la modification peut se faire directement viala boite de dialogue du **variable manager**.

Label des variables

Syntaxe:

```
label nom_variable "label"
```

On peut modifier/écraser un label existant

Exemple:

```
des foreign
label variable foreign "Origine de la voiture"
des foreign
```

```
Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g      origin     Car origin

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
foreign         byte    %8.0g      origin     Origine de la voiture
```

Label sur les modalités des variables

Deux étapes: la création des label et leurs affectation à une ou plusieurs variables.

Création du label: `label define`

Syntaxe:

```
label define nom_label val1 "label1" val2 "label2"... [,modify]
```

Exemple variable binaire (0,1) labélisée “No-Yes” avec comme nom de label NY : `label define NY 0 "non" 1 "oui`

S’il y a beaucoup de modalités à labelliser, on peut affecter ligne par ligne un label par modalité et utiliser l’option `modify`

Syntaxe:

```
label define nom_label 1 "nom1", modify
label define nom_label 2 "nom2", modify
label define nom_label 3 "nom3", modify
label define nom_label 4 "nom2", modify
```

Affectation du label: `label value`

Syntaxe:

```
label value nom_variable nom_label
```

Exercice: créer une variable indicatrice qui regroupe de la variable rep78: 0 si rep78<4 et 1 si rep78>3. Affecter un label à la variable (au choix) et des labels aux modalités (au choix).

```
#| code-fold: true
#| code-summary: "Show the code"

gen rep78b = rep78<4
replace rep78b=. if rep78==.

label define rep78b 0 "1-3 réparations" 1 "Plus de 3 réparations", modify
label value rep78b rep78b
des rep78b
tab rep78 rep78b
```

```
Unknown #command
Unknown #command


(5 real changes made, 5 to missing)

Variable      Storage   Display    Value
    name         type    format    label      Variable label
------------------------------------------------------------------------------
rep78b          float   %22.0g     rep78b

    Repair |
    record |        rep78b
      1978 | 1-3 répar  Plus de 3 |     Total
```

```
----------+----------------------+----------
        1 |         0          2 |         2
        2 |         0          8 |         8
        3 |         0         30 |        30
        4 |        18          0 |        18
        5 |        11          0 |        11
----------+----------------------+----------
    Total |        29         40 |        69
```

Fichiers de labels et multilangue

`label save`
On peut générer un fichier (.do) donnant le programme qui génère les labels (existants) d'une base: commande `label save` (par le menu: data => data utilities => label utilities => *save labels as do file*).

multilangue
Pour des enquêtes internationales, on peut générer des jeux de labels en plusieurs langues et switcher de l'une à l'autre (exemple MAFE l'Ined). La commande doit être installée, elle est externe à Stata (`ssc install mlanguage` - auteur *J.Weesie*).

::: callout_note ## TODO Faire un rapide topo sur les variables de type dates

8 Manipulations des bases de données

Programme du chapitre

Commandes et expressions introduites

SECTION	COMMANDES
Fusion	`append` - `merge` - `frlink` - `ffrval`
Transposition	`reshape long` `reshape wide`
Allongement	`expand`
Base d'indicateurs	`collapse` `contract`

- **En gras, commandes externes**

8.1 Fusion de bases

- Deux types de fusions:
  - La fusion verticale non controlée - empilement - (`append`)
  - la fusion horizontale contrôlée - appariement - (`merge`).

8.1.1 Append

- Consiste simplement à ajouter des observations entre plusieurs bases, avec ou non un même jeu de variables.

Avant empilement

base1		
id	v1	v2
A	8	2
B	1	2
C	2	4

base2			
id	v1	v2	v3
D	2	5	10
E	12	1	8

Après empilement

base3			
id	v1	v2	v3
A	8	2	.
B	1	2	.
C	2	4	.
D	2	5	10
E	12	1	8

On va générer les deux bases de données avec la commande `input` (non traité dans cette formation: `help input`).

```
clear
input str6 id v1 v2
  "A" 8 2
  "B" 1 2
  "C" 2 4
end

list

save base1, replace
```

```
<IPython.core.display.HTML object>
```

```
     +--------------+
     | id   v1   v2 |
     |--------------|
  1. |  A    8    2 |
  2. |  B    1    2 |
  3. |  C    2    4 |
     +--------------+
file base1.dta saved
```

```
clear
input str20 id v1 v2 v3
  "D" 2 5 10
  "E" 12 1 8
end
list

save base2, replace
```

```
     +-------------------+
     | id   v1   v2   v3 |
     |-------------------|
  1. |  D    2    5   10 |
  2. |  E   12    1    8 |
     +-------------------+
file base2.dta saved
```

La syntaxe de la commande `append` consiste à ajouter une ou plusieurs bases à la base active avec l'argument `using`.

```
append using base1
sort id
list
```

```
     +-------------------+
     | id   v1   v2   v3 |
     |-------------------|
  1. |  A    8    2    . |
  2. |  B    1    2    . |
  3. |  C    2    4    . |
  4. |  D    2    5   10 |
  5. |  E   12    1    8 |
     +-------------------+
```

On peut sélectionner les variables de la base qui sera empilée à la base active avec l'option `keep`.
Dans l'exemple, si la base active est *base1*, on peut ne pas vouloir ajouter la variable *v3* seulement renseignée pour les observations de *base2*.

```
use base1, clear
append using base2, keep(id v1 v2)
list
```

```
(variable id was str6, now str20 to accommodate using data's values)

     +--------------+
     | id   v1   v2 |
     |--------------|
  1. |  A    8    2 |
  2. |  B    1    2 |
  3. |  C    2    4 |
  4. |  D    2    5 |
  5. |  E   12    1 |
     +--------------+
```

Si les informations précédentes étaient ventilées dans trois bases, une par variable v, et avec le même niveau d'observation (A,B,C,D,E dans les 3 bases), l'utilisation de `append` conduirait à une structure empilée non souhaitable avec une réplication des *id*.

Pour obtenir la base finale proprement appariée, il convient de faire une fusion horizontale contrôlée par une une clé d'identification.

8.1.2 Merge

Stata demande que les bases soient soit triées (`sort`) sur la clé d'appariement en amont de l'opération. Sinon un message d'erreur sera renvoyé.

- La base active (ouverte) est appelée **base master**
- La base qui sera appariée à la base ouverte est appelée base **using** [1]

Syntaxe minimale 1 avec préfixes:

```
merge [1:1] [1:m] [m:1] id_variables(s) using nom_base
```

- Ici on peut apparier plus de deux bases.
- On dispose d'une sécurité si les niveaux d'identification sont différents.

[1] Cela peut être plusieurs bases.

8.1.2.1 Même niveau d'identification

Partons des informations suivantes: - *Base1* comprend la variable d'identification *id* (observations A,B,C) et de deux variables numériques *v1* et *v3* - *Base2* comprend la même variable d'identification *id* (observations B,C,D) et de la variable numérique *v3*

Avant appariement

base1 (using)		
id	v1	v2
A	8	2
B	1	2
C	2	4

base2 (active)	
id	v3
B	10
C	8
D	10

Après appariement

base 3					
Id	v3	v1	v2	_merge	
A	.	8	2	using	base 1 seulement
B	10	1	2	match	bases 1 et 2
C	8	2	4	match	bases 1 et 2
D	10	.	.	master	bases 2 seulement

Le niveau d'identification est identique dans les deux bases. Il s'agit donc d'un **`merge 1:1`** [**One to One**]

On va de nouveau générer les bases avec `input`.

```
clear
input str1 id v1 v2
"A" 8 2
"B" 1 2
"C" 2 4
end
list

sort id
save base1, replace
```

```
     +--------------+
     | id   v1   v2 |
     |--------------|
  1. |  A    8    2 |
  2. |  B    1    2 |
  3. |  C    2    4 |
     +--------------+
file base1.dta saved
```

Rappel: bien faire le `sort` sur la base using

```
clear
input str1 id v3
"B" 10
"C" 8
"D" 10
end
list

sort id
save base2, replace
```

```
     +---------+
     | id   v3 |
     |---------|
  1. |  B   10 |
  2. |  C    8 |
  3. |  D   10 |
     +---------+
file base2.dta saved
```

```
merge 1:1 id using base1
```

```
    Result                      Number of obs
    -----------------------------------------
    Not matched                             2
        from master                         1  (_merge==1)
        from using                          1  (_merge==2)

    Matched                                 2  (_merge==3)
    -----------------------------------------
```

- L'output affiche le résultat de l'appariement à l'aide d'un t ltrer si nécessaire les observations selon le résultat de l'apariement. Contrairement à d'autres applications, cette opération n'est pas effectuée en amont avec des fonctions où des options spécifiques. Par exemple avec R: `left_join`, `right_join`, `inner_join`. `_merge = 1` : observations qui se trouvent seulement dans la base active (master) `_merge = 2` : observations qui se trouvent seulement dans la base using (appariée) `_merge = 3` : observations communes aux bases master et using.
- Les variables de la base master/active sont positionnées en tête de colonnes.

```
sort id
list
```

```
     +-----------------------------------+
     | id   v3   v1   v2           _merge |
     |-----------------------------------|
  1. |  A    .    8    2    Using only (2) |
  2. |  B   10    1    2       Matched (3) |
  3. |  C    8    2    4       Matched (3) |
  4. |  D   10    .    .   Master only (1) |
     +-----------------------------------+
```

Si on souhaite seulement conserver les observations communes aux deux bases (`_merge=3`):

```
keep if _merge==3
list
```

```
(2 observations deleted)

     +---------------------------------+
     | id   v3   v1   v2        _merge |
     |---------------------------------|
  1. |  B   10    1    2   Matched (3) |
  2. |  C    8    2    4   Matched (3) |
     +---------------------------------+
```

> **Variable _merge et appariements successifs**
>
> Pensez à supprimer la variable *_merge* si plusieurs opérations d'appariement sont effectués. La commande ne prévoit pas d'écraser la variable de la fusion précédente.

Situation avec plus d'une base à apparier

On ne peux pas utiliser la syntaxe avec préfixe (ici `merge 1:1`).

On va ajouter une nouvelle base qui sera appariée avec les deux premières, qui seront donc les deux bases de type *using*.

```
clear
input str1 id str3  v4
"A" "Non"
"B" "Oui"
"C" "Oui"
end
```

```
list

sort id
```

```
     +----------+
     | id    v4 |
     |----------|
  1. |  A   Non |
  2. |  B   Oui |
  3. |  C   Oui |
     +----------+
```

```
merge id using base1 base2

order id v1 v2 v3 v4 _merge1 _merge2 _merge

list
```

```
(you are using old merge syntax; see [D] merge for new syntax)

     +-------------------------------------------------------+
     | id   v1   v2   v3    v4   _merge1   _merge2   _merge |
     |-------------------------------------------------------|
  1. |  A    8    2    .   Non         1         0        3 |
  2. |  B    1    2   10   Oui         1         1        3 |
  3. |  C    2    4    8   Oui         1         1        3 |
  4. |  D    .    .   10               0         1        2 |
     +-------------------------------------------------------+
```

On obtient maintenant 3 variables _merge:

- *_merge1. *Donne le résultat de l'appariement entre la nouvelle base et* base1*: 0 si seulement dans une seule des deux bases (D), 1 si dans les deux bases (A,B,C).
- *_merge2. *Donne le résultat de l'appariement entre la nouvelle base et* base2*: 0 si seulement dans une seule des deux bases (A,D), 1 si dans les deux bases (B,C).
- *_merge*. Résume rapidement le matching entre les bases: on retrouve au moins une fois les observations (A,B,C) dans l'un des deux appariement (_merge=3), on trouve une observation (D) qui ne se trouve que dans une base *using* (_D_merge=2).

Si l'on souhaite conserver les observations communes aux trois bases, on peut sommer les valeurs de *_merge1* et *_merge2* et conserver les observations dont la valeurs de cette somme est égale au nombre d'appariements; ou faire une sélection des observations avec un filtre conditionnel, ici:

```
keep if _merge1==1 & _merge2==1
list

drop _merge*
```

```
(2 observations deleted)

     +-----------------------------------------------------+
     | id   v1   v2   v3    v4   _merge1   _merge2   _merge |
     |-----------------------------------------------------|
  1. |  B    1    2   10   Oui         1         1        3 |
  2. |  C    2    4    8   Oui         1         1        3 |
     +-----------------------------------------------------+
```

Commande join du package ftools

[A tester]

- Documentation.
- Permet de gagner 70% de durée d'exécution lorsque la volumétrie dépasse 100000 observations
- Gère en amont le tri des bases appariée.

8.1.2.2 Niveaux d'identification différents

Un merge de type 1:1 n'est pas possible. Dans l'exemple qui suit la base *period_act* liste pour deux personnes le statut d'activité observé pour plusieurs périodes soit des observations multiples pour chaque individus, et la base *sexe* donne une caractéristique unique pour chaque individu. Selon le statut des bases appariée (master ou using), l'appariement est de type 1:m ou m:1.

- Si la base active est à observations multiples sur la clé d'identification: m:1
- Si la base active est à observations uniques sur la clé d'identification: 1:m

Base period_act

id	Periodes	Activité
1	1	Emploi
1	2	Emploi
1	3	Chômage
2	1	Chômage
2	2	Chômage
2	3	Emploi
2	4	Chômage

Base sexe

id	Sexe
1	Homme
2	Femme

Bases appariées

id	Periodes	sexe	Activité
1	1	Homme	Emploi
1	2	Homme	Emploi
1	3	Homme	Chômage
2	1	Femme	Chômage
2	2	Femme	Chômage
2	3	Femme	Emploi
2	4	Femme	Chômage

On va de nouveau générer les données avec `input`

```
clear
input id périodes str8 Activité
1 1 "Emploi"
1 2 "Emploi"
1 3 "Chômage"
2 1 "Chômage"
2 2 "Chômage"
2 3 "Emploi"
2 4 "Chômage"
end
list
sort id
save "period_act", replace
```

```
     +--------------------------+
     | id   périodes   Activité |
     |--------------------------|
  1. |  1          1     Emploi |
  2. |  1          2     Emploi |
  3. |  1          3    Chômage |
  4. |  2          1    Chômage |
  5. |  2          2    Chômage |
     |--------------------------|
  6. |  2          3     Emploi |
  7. |  2          4    Chômage |
     +--------------------------+
file period_act.dta saved
```

```
clear
input id str6 sexe
1 "Homme"
2 "Femme"
end
list
sort id
save "sexe", replace
```

```
     +------------+
     | id    sexe |
     |------------|
  1. |  1   Homme |
  2. |  2   Femme |
     +------------+
file sexe.dta saved
```

Si on effectuait un merge 1:1, Stata renverrait le message d'erreur suivant:

```
merge 1:1 id using activités

variable id does not uniquely identify observations in the using data
r(459);
```

Ici la base active est la base *sex*. Le prefixe qui doit être utilisé est donc **`1:m`** [2]

```
merge 1:m id using period_act
sort id période
list
```

```
    Result                      Number of obs
    -----------------------------------------
    Not matched                             0
    Matched                                 7  (_merge==3)
    -----------------------------------------

     +-------------------------------------------------+
     | id    sexe   périodes   Activité        _merge |
     |-------------------------------------------------|
  1. |  1   Homme          1     Emploi   Matched (3) |
  2. |  1   Homme          2     Emploi   Matched (3) |
  3. |  1   Homme          3    Chômage   Matched (3) |
  4. |  2   Femme          1    Chômage   Matched (3) |
  5. |  2   Femme          2    Chômage   Matched (3) |
     |-------------------------------------------------|
  6. |  2   Femme          3     Emploi   Matched (3) |
  7. |  2   Femme          4    Chômage   Matched (3) |
     +-------------------------------------------------+
```

[2] **`m:1`** renvoit un message d'erreur. Dans ce sens, la base active doit être *period_act* et la base using *sexe*.

> **Astuce**
>
> Le tri de la base est régulièrement modifié après ce type d'appariement. Penser donc à retrier les données proprement, surtout quand il s'agit comme ici d'informations biographiques (`sort id périodes`)

De nouveau les préfixes sont optionnels, et permettent seulement de contrôler l'appariement. On peut sans soucis fusionner des informations contextuelles avec des informations multiples avec seulement `merge`. Un avertissement se renvoyé à l'exécution de la commande

```
use sexe, clear
merge id using period_act
sort id périodes
list
```

```
(you are using old merge syntax; see [D] merge for new syntax)
variable id does not uniquely identify observations in period_act.dta
```

	id	sexe	périodes	Activité	_merge
1.	1	Homme	1	Emploi	3
2.	1	Homme	2	Emploi	3
3.	1	Homme	3	Chômage	3
4.	2	Femme	1	Chômage	3
5.	2	Femme	2	Chômage	3
6.	2	Femme	3	Emploi	3
7.	2	Femme	4	Chômage	3

8.1.2.3 Appariement avec des frames

L'utilisation des frames présentent plusieurs avantages:

- Il n'est pas nécessaire de trier les bases concernées par l'appariement.
- On peut sélectionner avec la commande `frget` la ou les variables qui seront récupérées dans la base master. On apparie donc pas des bases en tant que telles, on récupère de l'information de frames liées.
- Mieux encore, on peut réaliser des opérations entre observations individuelles et observations contextuelles sans passer par un appariement. Avec les frames, l'opération d'appariement doit être plutôt compris comme un système de liaison entre bases, le transfert d'informations n'étant qu'une opération optionnelle.

Au niveau des désavantages:

- Si on ne travaille pas exclusivement sous frames, les bases devront être transformées en frame (voir exemple)
- Absence de variable de type *_merge* qui permet de contrôler le résultat de l'appariement.
- les prefixes sont uniquement 1:1 et m:1. Cela signifie dans le second cas que la frame active lors de l'opération de liaison doit toujours être celle dont la clé d'identification est de type multiple (niveau individuel).
- Peut-être le plus embêtant est l'absence d'appariement pour les informations correspondant à **_merge=2** (Informations seulement présentes dans la base using). Le dernier exemple illustre ce point.

On reprend l'exemple précédent, en transformant dans un premier temps les deux bases en frames.

```
frame reset

frame create period_act
frame period_act: use period_act
frame create sexe
frame sexe: use sexe

frame dir
```

```
  default     0 x 0
  period_act  7 x 3; period_act.dta
  sexe        2 x 2; sexe.dta
```

On doit se positionner sur la frame *period_act* (type m)

```
frame change period_act
```

Pour lier les frames on utilise la commande `frlink`.

Syntaxe

```
frlink 1:1/m:1 id_variable(s), frame(nom_frame) gen(variable_lien)
```

Ici on fait un appariement de type m:1, la clé d'identification est de nouveau *id*. On lie la frame active à la frame *sexe* et la variable de liaison (ici un alias de la variable id) est appelée *link*.

```
frlink m:1 id, frame(sexe) gen(link)
```

```
(all observations in frame period_act matched)
```

Pour importer la variable sexe dans la frame *period_act*, on utilise la commande **`frget`**, en précisant la ou les variable que l'on souhaite récupérer, ainsi que la variable de liaison (une même frame peut avoir plusieurs liaisons. Voir plus loin).

```
frget sexe , from(link)

frame period_act: order link, last
list
```

```
(1 variable copied from linked frame)

     +---------------------------------------+
     | id   périodes   Activité    sexe   link |
     |---------------------------------------|
  1. |  1          1     Emploi   Homme      1 |
  2. |  1          2     Emploi   Homme      1 |
  3. |  1          3    Chômage   Homme      1 |
  4. |  2          1    Chômage   Femme      2 |
  5. |  2          2    Chômage   Femme      2 |
     |---------------------------------------|
  6. |  2          3     Emploi   Femme      2 |
  7. |  2          4    Chômage   Femme      2 |
     +---------------------------------------+
```

⚠ Liaison des frames en présence d'information incomplète

La liaison de frames peut être problématique en présence d'informations incomplètes. Pour faire simple, la liaison des frames permet de faire des appariements de type **_merge=1** et **_merge=3** (présence dans la master seulement ou présence dans la master et la using) mais ne permet pas de récupérer des informations présentes seulement dans la base using).
Pour illustrer cela on va générer une nouvelle frame, de type individus-périodes, avec une variable additionnelle *tvc*.

- Pour id= 1, on a pas d'information dans la frame *period_act* pour période=4.
- Pour id= 2, on a pas d'information dans la frame *tvc* pour les périodes 3 et 4.

Création de la nouvelle frame (voir le .do, la compilation pour générer ce support complexifie un peu l'opération):

```
frame create tvc
frame change tvc



clear
input id périodes tvc
1 1 0
1 2 0
1 3 1
1 4 0
2 1 1
2 2 0
end

list

save tvc, replace
frame tvc: use tvc
```

```
     +---------------------+
     | id   périodes   tvc |
     |---------------------|
  1. |  1          1     0 |
  2. |  1          2     0 |
  3. |  1          3     1 |
  4. |  1          4     0 |
  5. |  2          1     1 |
     |---------------------|
  6. |  2          2     0 |
     +---------------------+
file tvc.dta saved
```

Liaison des frames et récupération de la variable tvc dans *period_act*

```
frame change period_act
frlink 1:1 id périodes, frame(tvc) gen(link2)

frget tvc, from(link2)

list
```

```
(2 observations in frame period_act unmatched)
(2 missing values generated)
```

```
(1 variable copied from linked frame)
```

	id	périodes	Activité	sexe	link	link2	tvc
1.	1	1	Emploi	Homme	1	1	0
2.	1	2	Emploi	Homme	1	2	0
3.	1	3	Chômage	Homme	1	3	1
4.	2	1	Chômage	Femme	2	5	1
5.	2	2	Chômage	Femme	2	6	0
6.	2	3	Emploi	Femme	2	.	.
7.	2	4	Chômage	Femme	2	.	.

On voit bien que la valeur de tvc pour id=1 et périodes=4 n'a pas été importée (**_merge=2** dans un appariement classique). En revanche, pour id=2, l'incomplétude de l'information dans la base *tvc* pour les périodes 3 et 4 est bien visible.
Avec un merge classique (on suppose que *period_act* n'a pas été appariée à *sexe*):

```
use tvc, clear
sort id périodes
save tvc, replace
use period_act, clear
sort id périodes
merge 1:1 id périodes using tvc
sort id périodes
list
```

```
file tvc.dta saved

    Result                      Number of obs
    -----------------------------------------
    Not matched                             3
        from master                         2  (_merge==1)
        from using                          1  (_merge==2)

    Matched                                 5  (_merge==3)
    -----------------------------------------
```

	id	périodes	Activité	tvc	_merge
1.	1	1	Emploi	0	Matched (3)
2.	1	2	Emploi	0	Matched (3)
3.	1	3	Chômage	1	Matched (3)

```
  4. |  1          4                0    Using only (2) |
  5. |  2          1    Chômage     1       Matched (3) |
     |-------------------------------------------------|
  6. |  2          2    Chômage     0       Matched (3) |
  7. |  2          3     Emploi     .   Master only (1) |
  8. |  2          4    Chômage     .   Master only (1) |
     +-------------------------------------------------+
```

On a bien ici l'ajout de l'information correspondant à _merge=2 (*Using only*)

Un des intérêts des frames, est de faire des opérations entre informations individuelles et contextuelles sans passer par un appariement en amont. Par l'exemple, nous allons voir comment un appariement peut être évité lorsqu'on travaille sur ce genre d'information.

On va générer 2 bases, une individuelle et une contextuelle. La première contient un identifiant individuel (*id*), le nom de la zône d'appartenance (*zone*) et les valeurs observées d'une variable *x*. La seconde contient le nom des zônes et la valeur moyenne de la variable x dans ces espaces.

Création des frames:

```
frame reset

clear
input id str6 zone x
1 "zoneA" 10
2 "zoneA" 15
3 "zoneB" 9
4 "zoneB" 12
5 "zoneB" 10
6 "zoneB" 15
7 "zoneC" 6
8 "zoneC" 13
9 "zoneC" 16
end
list
save indiv, replace
```

```
     +-----------------+
     | id    zone    x |
     |-----------------|
  1. |  1   zoneA   10 |
  2. |  2   zoneA   15 |
  3. |  3   zoneB    9 |
  4. |  4   zoneB   12 |
```

```
  5. |  5   zoneB   10 |
     |-----------------|
  6. |  6   zoneB   15 |
  7. |  7   zoneC    6 |
  8. |  8   zoneC   13 |
  9. |  9   zoneC   16 |
     +-----------------+
file indiv.dta saved
```

```
clear
input str6 zone xmean
"zoneA" 11
"zoneB" 12
"zoneC" 13
end
list
save zone, replace
```

```
     +---------------+
     |  zone   xmean |
     |---------------|
  1. | zoneA      11 |
  2. | zoneB      12 |
  3. | zoneC      13 |
     +---------------+
file zone.dta saved
```

```
frame create indiv
frame indiv: use indiv
frame create zone
frame zone: use zone
```

Après avoir lié les deux frames (m:1), on va calculer directement la différence entre la valeur observée pour chaque individu de la variable x et sa moyenne par zone ($xmean$). On utilise la fonction **`frval`** comme argument de la commande **`generate`**.

```
frame change indiv
frlink m:1 zone, frame(zone) gen(link)
list
```

```
(all observations in frame indiv matched)
```

```
     +-----------------------+
     | id    zone    x   link |
     |-----------------------|
  1. |  1   zoneA   10      1 |
  2. |  2   zoneA   15      1 |
  3. |  3   zoneB    9      2 |
  4. |  4   zoneB   12      2 |
  5. |  5   zoneB   10      2 |
     |-----------------------|
  6. |  6   zoneB   15      2 |
  7. |  7   zoneC    6      3 |
  8. |  8   zoneC   13      3 |
  9. |  9   zoneC   16      3 |
     +-----------------------+
```

```
gen = var1 - frval(nom_link, var2)
```

```
gen diffx = x - frval(link, xmean)
list
```

```
     +-------------------------------+
     | id    zone    x   link   diffx |
     |-------------------------------|
  1. |  1   zoneA   10      1     -1 |
  2. |  2   zoneA   15      1      4 |
  3. |  3   zoneB    9      2     -3 |
  4. |  4   zoneB   12      2      0 |
  5. |  5   zoneB   10      2     -2 |
     |-------------------------------|
  6. |  6   zoneB   15      2      3 |
  7. |  7   zoneC    6      3     -7 |
  8. |  8   zoneC   13      3      0 |
  9. |  9   zoneC   16      3      3 |
     +-------------------------------+
```

8.2 Transposition d'une base

8.2.1 Syntaxe et exemples

Cette opération permet d'**allonger** ou d'**élargir** une base, généralement sur des variables occurencées. Ces occurences peuvent être des séquences ou points chronologiques (valeur d'une variable sur plusieurs années), ou des individus composant un ménage.

Avec Stata, ces opérations de transpositions sont effectuées avec la commande **`reshape`**

- De large à long: **reshape long**
- De long à large: **reshape wide**

A noter que la seconde opération est plus gourmande en durée d'exécution. De nouveau si la volumétrie de la base est élevée, disons plus d'une million d'observations, on peut se reporter sur la commande **greshape** du package **gtools**. On peut trouver un benchmark sur des données simulées [liens].

Au niveau de la syntaxe:

- Il est nécessaire d'avoir une variable d'identification pour réaliser l'opération: cela peut être un identifiant individuel[3] si la variations des observations est relatives à des périodes, ou un identifiant ménage si la source de la variation sont les personnes le composant. Ce peut bien évidemment fonctionner avec des zônes géographiques: régions-départements, régions-communes, départements-communes.
  Cette variable d'identification doit être renseignée en option: **i(var_id)**
- On indique dans l'expression principale le racine des variables occurencées: si la base est en format large avec les variables *revenu1980*, *revenu1981*,….,*revenu1990*, la racine sera donc **revenu**. Les occurences peuvent être des lettres (A,B,D…) ou des mots (un,deux,trois…).
- Information sur les occurences: selon le type de transposition on doit indiquer en option la variable qui contiendra ou qui contient les occurences. Cette option est **j(nom_variable)**
  - si la base est en format large et qu'on souhaite l'allonger, on indique obligatoirement la variable qui sera créée et qui reportera les valeurs des occurences.
  - si la base est en format long et qu'on souhaite l'élargir, on indique obligatoirement la variable qui contient les occurences.
- Selon la transposition, le nom de commande est suivi de **long** ou **wide**

Syntaxe de large à long:

```
reshape long racines_variables_occurencées, i(var_id) j(var_occurences)
```

Syntaxe de long à large:

```
reshape wide racines_variables_occurencées, i(var_id) j(var_occurences)
```

Exemple

On part de la base suivante

```
clear
input id x1 x2 x3 x4
1 10 20 12 25
2 12 22 15 30
3 15 25 33 30
4 21 17 22 27
5 13 15 14 18
```

[3]Cela peut être une zône géographique

```
end

list
```

```
     +------------------------+
     | id   x1   x2   x3   x4 |
     |------------------------|
  1. |  1   10   20   12   25 |
  2. |  2   12   22   15   30 |
  3. |  3   15   25   33   30 |
  4. |  4   21   17   22   27 |
  5. |  5   13   15   14   18 |
     +------------------------+
```

On allonger la base sur les variables *x1* à *x4*. La racine est donc *x*. Pour le choix de la nouvelle variable qui aura pour chaque id les valeurs 1 à 4, on ne peux pas choisir x, qui sera créée automatiquement. Selon le type d'information contenu dans l'occurence, on peut utiliser un nom indiquant une période, un membre de ménage ou une zône géographique. Ici on ca suposer que les occurences sont de nature temporelle, et on choisira *t* comme nom à la variable de l'option `j()`.

```
reshape long x , i(id) j(t)
```

```
(j = 1 2 3 4)

Data                               Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                5   ->   20
Number of variables                   5   ->   3
j variable (4 values)                     ->   t
xij variables:
                           x1 x2 ... x4   ->   x
-----------------------------------------------------------------------------
```

On remarque que Stata donne quelques informations sur le résultats de l'opération: variables créées, nombre d'observations dans le nouveau format

```
list
```

```
     +-------------+
     | id   t    x |
     |-------------|
  1. |  1   1   10 |
  2. |  1   2   20 |
  3. |  1   3   12 |
  4. |  1   4   25 |
  5. |  2   1   12 |
     |-------------|
  6. |  2   2   22 |
  7. |  2   3   15 |
  8. |  2   4   30 |
  9. |  3   1   15 |
 10. |  3   2   25 |
     |-------------|
 11. |  3   3   33 |
 12. |  3   4   30 |
 13. |  4   1   21 |
 14. |  4   2   17 |
 15. |  4   3   22 |
     |-------------|
 16. |  4   4   27 |
 17. |  5   1   13 |
 18. |  5   2   15 |
 19. |  5   3   14 |
 20. |  5   4   18 |
     +-------------+
```

On peut repasser au format de départ (large) avec `reshape wide`

```
reshape wide x , i(id) j(t)
```

```
(j = 1 2 3 4)


Data                               Long   ->   Wide
-----------------------------------------------------------------------------
Number of observations               20   ->   5
Number of variables                   3   ->   5
j variable (4 values)                 t   ->   (dropped)
xij variables:
                                      x   ->   x1 x2 ... x4
-----------------------------------------------------------------------------
```

```
list
```

```
     +-----------------------+
     | id   x1   x2   x3   x4 |
     |-----------------------|
  1. |  1   10   20   12   25 |
  2. |  2   12   22   15   30 |
  3. |  3   15   25   33   30 |
  4. |  4   21   17   22   27 |
  5. |  5   13   15   14   18 |
     +-----------------------+
```

Bien évidemment les variable fixes ne doivent pas être renseigné dans la commande, les valeurs sont conservées

```
clear
input id x1 x2 x3 x4 fixe
1 10 20 12 25 0
2 12 22 15 30 1
3 15 25 33 30 0
4 21 17 22 27 1
5 13 15 14 18 0

end
list

reshape long x, i(id) j(t)
list
```

```
     +-------------------------------+
     | id   x1   x2   x3   x4   fixe |
     |-------------------------------|
  1. |  1   10   20   12   25      0 |
  2. |  2   12   22   15   30      1 |
  3. |  3   15   25   33   30      0 |
  4. |  4   21   17   22   27      1 |
  5. |  5   13   15   14   18      0 |
     +-------------------------------+
(j = 1 2 3 4)

Data                               Wide   ->   Long
```

```
-----------------------------------------------------------------------------
Number of observations                5   ->   20
Number of variables                   6   ->   4
j variable (4 values)                     ->   t
xij variables:
                           x1 x2 ... x4   ->   x
-----------------------------------------------------------------------------
```

	id	t	x	fixe
1.	1	1	10	0
2.	1	2	20	0
3.	1	3	12	0
4.	1	4	25	0
5.	2	1	12	1
6.	2	2	22	1
7.	2	3	15	1
8.	2	4	30	1
9.	3	1	15	0
10.	3	2	25	0
11.	3	3	33	0
12.	3	4	30	0
13.	4	1	21	1
14.	4	2	17	1
15.	4	3	22	1
16.	4	4	27	1
17.	5	1	13	0
18.	5	2	15	0
19.	5	3	14	0
20.	5	4	18	0

8.2.2 Mise en garde

Complétude du nom de la racine

Bien penser à mettre l'intégralité de la racine, partie fixe de la variable occurencée:

```
clear
input id x_1 x_2 x_3 x_4
1 10 20 12 25
2 12 22 15 30
```

```
3 15 25 33 30
4 21 17 22 27
5 13 15 14 18
end

list
```

```
     +---------------------------+
     | id   x_1   x_2   x_3   x_4 |
     |---------------------------|
  1. |  1    10    20    12    25 |
  2. |  2    12    22    15    30 |
  3. |  3    15    25    33    30 |
  4. |  4    21    17    22    27 |
  5. |  5    13    15    14    18 |
     +---------------------------+
```

```
reshape long x , i(id) j(t)
```

renverra le message d'erreur suivant:

```
variable t contains all missing values
r(498);
```

Omission de variables occurencée

Contrairement à l'allongement, l'élargissement est plus contraignant, toutes les variables non fixes doivent être renseignées.

Si on omet des variables occurencées dans l'allongement, elle sont conservées tel quel et les valeurs sont répliquées d'une ligne à l'autre:

```
clear
input id x1 x2 y1 y2 fixe
1 10 20 12 25 0
2 12 22 15 30 0
3 15 25 33 30 1
4 21 17 22 27 1
5 13 15 14 18 0
end
list

reshape long x , i(id) j(t)
list
```

```
     +------------------------------+
     | id   x1   x2   y1   y2   fixe |
     |------------------------------|
  1. |  1   10   20   12   25      0 |
  2. |  2   12   22   15   30      0 |
  3. |  3   15   25   33   30      1 |
  4. |  4   21   17   22   27      1 |
  5. |  5   13   15   14   18      0 |
     +------------------------------+
(j = 1 2)

Data                                Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                 5   ->   10
Number of variables                    6   ->   6
j variable (2 values)                      ->   t
xij variables:
                                    x1 x2   ->   x
-----------------------------------------------------------------------------

     +------------------------------+
     | id   t    x   y1   y2   fixe |
     |------------------------------|
  1. |  1   1   10   12   25      0 |
  2. |  1   2   20   12   25      0 |
  3. |  2   1   12   15   30      0 |
  4. |  2   2   22   15   30      0 |
  5. |  3   1   15   33   30      1 |
     |------------------------------|
  6. |  3   2   25   33   30      1 |
  7. |  4   1   21   22   27      1 |
  8. |  4   2   17   22   27      1 |
  9. |  5   1   13   14   18      0 |
 10. |  5   2   15   14   18      0 |
     +------------------------------+
```

En revanche si on part d'une base longue avec plusieurs dimensions variables

```
(j = 1 2)
```

```
Data                               Wide   ->   Long
-----------------------------------------------------------------------------
Number of observations                5   ->   10
Number of variables                   6   ->   5
j variable (2 values)                     ->   t
xij variables:
                                  x1 x2   ->   x
                                  y1 y2   ->   y
-----------------------------------------------------------------------------
```

```
list
```

```
     +-------------------------+
     | id   t    x    y   fixe |
     |-------------------------|
  1. |  1   1   10   12      0 |
  2. |  1   2   20   25      0 |
  3. |  2   1   12   15      0 |
  4. |  2   2   22   30      0 |
  5. |  3   1   15   33      1 |
     |-------------------------|
  6. |  3   2   25   30      1 |
  7. |  4   1   21   22      1 |
  8. |  4   2   17   27      1 |
  9. |  5   1   13   14      0 |
 10. |  5   2   15   18      0 |
     +-------------------------+
```

```
reshape wide x, i(id) j(t)
```

renverra le message d'erreur suivant:

```
(j = 1 2)
variable y not constant within id
Your data are currently long. You are performing a reshape wide. You typed something like

. reshape wide a b, i(id) j(t)

    There are variables other than a, b, id, t in your data. They must be constant within i
    of information.

    The variable or variables listed above are not constant within id. Perhaps the values a

    Either that, or the values vary because they should vary, in which case you must either
```

8.3 Allongement d'une base

Section très courte. Pariculièrement utile lorsqu'on manipule des données biographiques avec des durées, et pour faire la mise en forme nécessaire pour une analyse à durée discrète. La commande ***expand*** permet de répliquer les lignes, sur une valeur fixe qu'on indique ou sur des valeurs non constantes renseignés dans une variable.

Dans le premier cas la syntaxe est: **expand valeur** Dans le second cas la synataxe est: **expand nom_variable**

Exemple:

```
clear
input id duree e
1  3 0
2  4 1
3  2 1
end

list
```

```
     +----------------+
     | id   duree   e |
     |----------------|
  1. |  1       3   0 |
  2. |  2       4   1 |
  3. |  3       2   1 |
     +----------------+
```

***Allongement de la base*:**

```
expand duree
```

```
(6 observations created)
```

Si on veut faire une analyse à durée discrère, avec les variables de comptage (chapitre 5):

```
bysort id: gen t=_n
bysort id: replace e=0 if t<_N
list
```

```
(4 real changes made)
```

```
     +--------------------+
     | id   duree   e   t |
     |--------------------|
  1. |  1       3   0   1 |
  2. |  1       3   0   2 |
  3. |  1       3   0   3 |
  4. |  2       4   0   1 |
  5. |  2       4   0   2 |
     |--------------------|
  6. |  2       4   0   3 |
  7. |  2       4   1   4 |
  8. |  3       2   0   1 |
  9. |  3       2   1   2 |
     +--------------------+
```

Remarque: si la valeur sur laquelle est allongée la base a une valeur négative (par exemple des durées négatives), un message indique leur présence.

8.4 Créer des bases d'indicateurs

Dans ce qui suit il est fortement recommandé d'utiliser les frames (Stata 16 minimum). Pour faire ce type d'opérations deux commandes sont disponibles:

- la plus utilisée, **`collapse`** permet de créer une base d'indicateurs dédiées aux variables quantitatives: moyenne, médiane et autes quantiles, ….
- la moins utilisée, **`contract`**, est dédiée aux variables catégorielles (effectifs et effectif cumulés, proportions et proportions cumulées).

Pour les pondérations admises, se reporter à l'aide des commandes[4].

> ⚠ Ecrasement de la base d'origine
>
> Attention la base sur laquelle on travaille va être écrasée. Si ce n'est pas souhaité:
>
> - Utiliser les commandes `preserve` `restore` avant et après l'opération.
> - Générer une frame avec les variables qui seront transformées en indicateurs. On pourra conserver les deux bases dans la sessions, et les utiliser en parallèle.

[4] La question des pondérations sera traitée dans le chapitre suivant

8.4.1 `collapse`

Les indicateurs disponibles sont les suivants:

```
        mean         means (default)
        median       medians
        p1           1st percentile
        p2           2nd percentile
        ...          3rd-49th percentiles
        p50          50th percentile (same as median)
        ...          51st-97th percentiles
        p98          98th percentile
        p99          99th percentile
        sd           standard deviations
        semean       standard error of the mean (sd/sqrt(n))
        sebinomial   standard error of the mean, binomial (sqrt(p(1-p)/n))
        sepoisson    standard error of the mean, Poisson (sqrt(mean/n))
        sum          sums
        rawsum       sums, ignoring optionally specified weight except observations with a
        count        number of nonmissing observations
        percent      percentage of nonmissing observations
        max          maximums
        min          minimums
        iqr          interquartile range
        first        first value
        last         last value
        firstnm      first nonmissing value
        lastnm       last nonmissing value
```

- Par défaut c'est la moyenne qui est utilisée.
- Les résultats peuvent être stratifiées avec une option `by()`.

Syntaxe avec un seul indicateur

```
collapse [(statistique autre que moyenne) varlist [, by(varlist)]
```

Dans les exemples, on utilisera `preserve restore` pour retrouver la base de départ.

Exemples

```
clear
sysuse auto

preserve
collapse price
list
restore
```

```
preserve
collapse price mpg, by(foreign)
list
restore

preserve
collapse (median) price mpg, by(foreign)
list
restore

preserve
collapse (median) price mpg if rep78!=., by(foreign rep78)
list
restore
```

(Note: Below code run with echo to enable preserve/restore functionality.)

```
. clear

. sysuse auto
(1978 automobile data)

. preserve

. collapse price

. list

     +---------+
     |   price |
     |---------|
  1. | 6,165.3 |
     +---------+

. restore

. preserve

. collapse price mpg, by(foreign)


. list

     +-----------------------------+
     |  foreign     price       mpg |
```

```
     |-----------------------------|
  1. | Domestic   6,072.4   19.8269 |
  2. |  Foreign   6,384.7   24.7727 |
     +-----------------------------+

. restore

. preserve

. collapse (median) price mpg, by(foreign)

. list

     +---------------------------+
     |  foreign     price    mpg |
     |---------------------------|
  1. | Domestic   4,782.5     19 |
  2. |  Foreign     5,759   24.5 |
     +---------------------------+

. restore

. preserve

. collapse (median) price mpg if rep78!=., by(foreign rep78)

. list

     +---------------------------------+
     | rep78    foreign     price   mpg |
     |---------------------------------|
  1. |     1   Domestic   4,564.5    21 |
  2. |     2   Domestic     4,638    18 |
  3. |     3   Domestic     4,749    19 |
  4. |     4   Domestic     5,705    18 |
  5. |     5   Domestic   4,204.5    32 |
     |---------------------------------|
  6. |     3    Foreign     4,296    23 |
  7. |     4    Foreign     6,229    25 |
  8. |     5    Foreign     5,719    25 |
     +---------------------------------+

. restore

.
```

On voit que la variable indicateur prend le nom de la variable. On ne peut donc pas générer une liste

d'indicateurs sans renommer les variables.

Syntaxe avec plusieurs indicateurs

Dans l'expression principal, on doit donner un nom différent à chaque variable pour chaque indicateur…ce n'est pas très pratique, Stata aurait pu prévoir un moyen de générer par défaut des nom de variable comme mean_varname, min_varname….

Dans le cas de deux indicateurs (*median*, *min*) pour deux variable (*price*, *mpg*).

```
collapse [(stat1) varname11 = var1 varname21= var2  (stat2 ) varname12 = var1 varname22= var2
```

```
preserve
collapse (median) pricemed = price mpgmed=mpg  (min) pricemin = price mpgmin= mpg , by(foreig
list
restore
```

```
(Note: Below code run with echo to enable preserve/restore functionality.)

. preserve

. collapse (median) pricemed = price mpgmed=mpg (min) pricemin = price mpgmin=
> mpg , by(foreign)

. list

     +-------------------------------------------------+
     |  foreign   pricemed   mpgmed   pricemin   mpgmin |
     |-------------------------------------------------|
  1. | Domestic    4,782.5       19      3,291       12 |
  2. |  Foreign      5,759     24.5      3,748       14 |
     +-------------------------------------------------+

. restore

.
```

Remarque: pour des variables codées sous forme d'indicatrice, on peut générer des proportions ou des pourcentages facilement, ce qui rend la commande `contract` caduque avec deux modalités (exemple: variable foreign).

8.4.2 `contract`

Même principe, mais le nombre d'indicateurs est limité (effectifs ou proportion, cumulées ou non). Il n'y a pas d'option by mais on peut directement croiser les dimensions avec plusieurs variables. Je n'ai jamais utilisé cette commande en dehors de la formation, donc je n'en donnerai que deux exemples:

```
preserve
contract rep78 foreign
list
restore


preserve
contract rep78 foreign,  percent(percentage)
list
restore
```

(Note: Below code run with echo to enable preserve/restore functionality.)

```
. preserve

. contract rep78 foreign

. list

     +--------------------------+
     | rep78    foreign   _freq |
     |--------------------------|
  1. |     1   Domestic       2 |
  2. |     2   Domestic       8 |
  3. |     3   Domestic      27 |
  4. |     3    Foreign       3 |
  5. |     4   Domestic       9 |
     |--------------------------|
  6. |     4    Foreign       9 |
  7. |     5   Domestic       2 |
  8. |     5    Foreign       9 |
  9. |     .   Domestic       4 |
 10. |     .    Foreign       1 |
     +--------------------------+

. restore

. preserve

. contract rep78 foreign, percent(percentage)

. list

     +-----------------------------------+
     | rep78    foreign   _freq   percen~e |
     |-----------------------------------|
```

```
  1. |     1   Domestic       2       2.70 |
  2. |     2   Domestic       8      10.81 |
  3. |     3   Domestic      27      36.49 |
  4. |     3    Foreign       3       4.05 |
  5. |     4   Domestic       9      12.16 |
     |-------------------------------------|
  6. |     4    Foreign       9      12.16 |
  7. |     5   Domestic       2       2.70 |
  8. |     5    Foreign       9      12.16 |
  9. |     .   Domestic       4       5.41 |
 10. |     .    Foreign       1       1.35 |
     +-------------------------------------+

. restore

.
```

9 Analyse statistique avec Stata

[Début maj 30 Aout 2023]