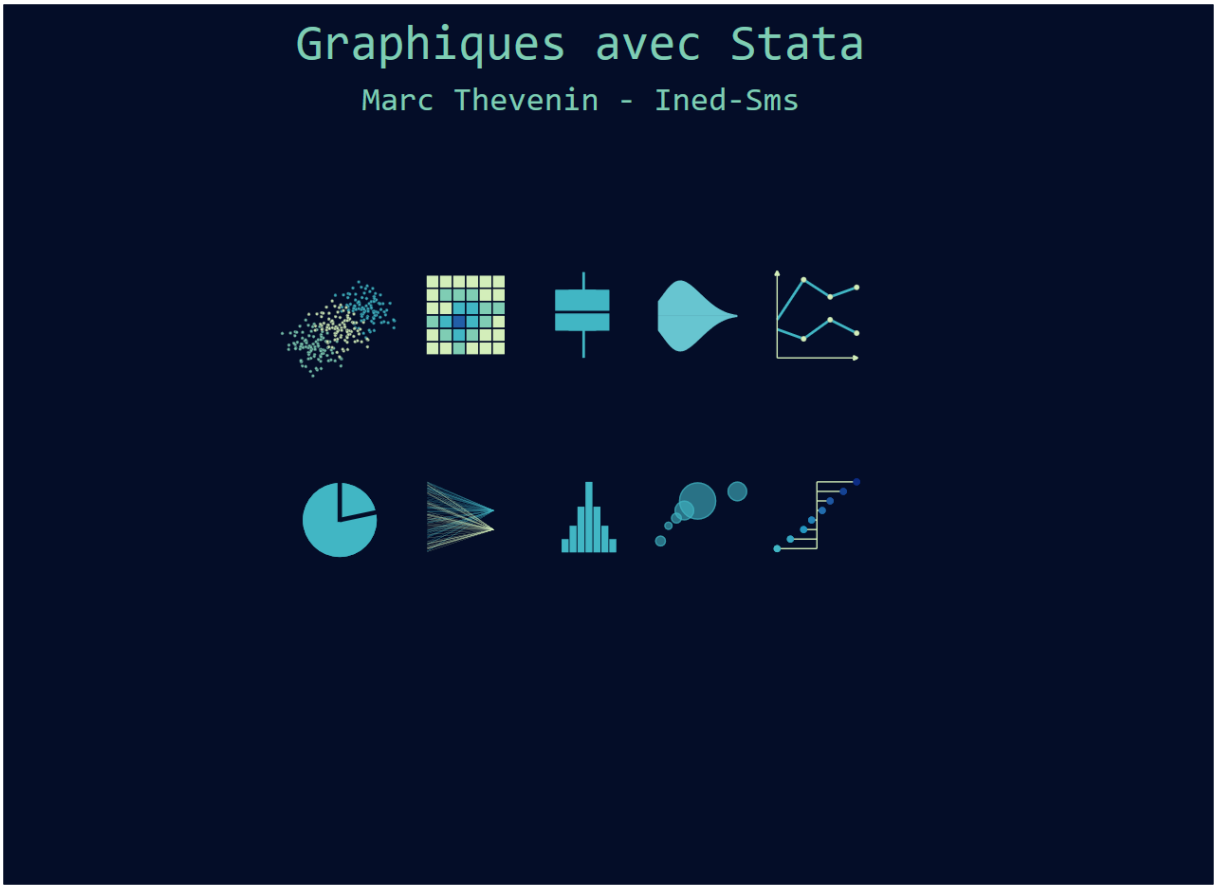


Version au 23 juin 2022 [chapitres 1-3 ok]

Table des matières

- Applications: Stata v16 – Python v3.8
- Conversion vidéo [version html] : ffmpg
- Relecture, conseils : Bénédicte Garnier – Ined Sms
- Impression : Philippe Olivier – Ined PLP

Introduction

[A faire]

Lien de la partie de Bénédicte Garnier (sémiologie graphique):
https://github.com/mthevenin/stata_fr/blob/master/semio_bene/2020.09.14.BGIntro_ppt.pdf

Lien site Dimitris Christodoulou (Graphique Stata) : https://graphworkflow.com/

Lien site Asjad Naqvi (Graphique Stata) : https://medium.com/the-stata-guide

Ouvrage prévu en 2022 : Demetris Christodoulou « *Visual Data Analytics* »

Générer et éditer un graphique

Eléments de syntaxe

Types de graphique

On peut identifier deux types de graphiques avec Stata.

Les graphiques de type one-way : coordonnées sur un seul axe avec ou non, un axe discret. Ils peuvent être préfixés par **gr** (**graph**) : **graph box** , **graph bar**, **histogram** …..

Les graphiques de type two-way : coordonnées sur deux axes, optionnellement selon le type de graphique un troisième axe peut être renseigné. Ils sont préfixés ou non par **tw** (**tw**oway) : **tw scatter**, **tw line**, …

Certains types de graphiques sont de type *oneway* ou *twoway*. C'est le cas des histogrammes avec une commande **histogram** et une commande **tw histogram**.

Les coordonnées sont généralement renseignées par des noms de variables. Pour certaines commandes graphiques, les valeurs sont directement renseignées: **tw scatteri** pour un nuage ou **tw pci** pour des segments [ici 4 coordonnées : (ymin, xmin) et (ymax, xmax)].

Un graphique peut être généré par une fonction : **tw function y = *f(x)***.

Liste et aides des commandes graphiques officielles : **help graph**.

Choix du délimiteur

Un graphique peut devenir assez gourmand en options. Par défaut avec Stata l'exécution est faite ligne par ligne avec l'option par défaut **#delimit cr**. Pour exécuter une commande sur plusieurs lignes, il est d'usage d'utiliser un triple slash `///`. Pour les graphiques, cette option manque de souplesse et peut provoquer des messages d'erreur lorsque `///` est collé par inadvertance au dernier caractère d'une ligne.

Lorsque le nombre de lignes devient assez conséquent, il est préférable d'utiliser **;** comme délimiteur. Le changement de délimiteur se fait avec **#delimit ;** et on indique la fin de l'exécution avec **;**. On retourne au délimiteur par défaut avec **#delimit cr**.

```
#delimit ;
tw scatter y x,
[option1]
[option2]
[option3]
.
.
[option n]
;

#delimit cr
* Suite du programm
```

Syntaxe

Pour des raisons de vocabulaire, nous allons utiliser le terme de **géométrie** issue de la syntaxe de l'incontournable librairie **ggplot2** de **R** pour identifier un type de graphique comme `scatter`, `line`…. Mais pas seulement. en fin de document je traite de l'utilisation de la librairie Python **plotnine**, qui est un wrapper relativement complet de **ggplot2**, directement exécutable dans un éditeur .do ou .ado. depuis la version 16 de Stata.

En première approche, la syntaxe d'un graphique avec Stata est particulièrement simple :

```
[tw/graph] type_géométrie coordonnées [if/in] [weight], [options géométrie]
[,[options du graphique]]
```

Un bloc d'objets graphiques (même type)

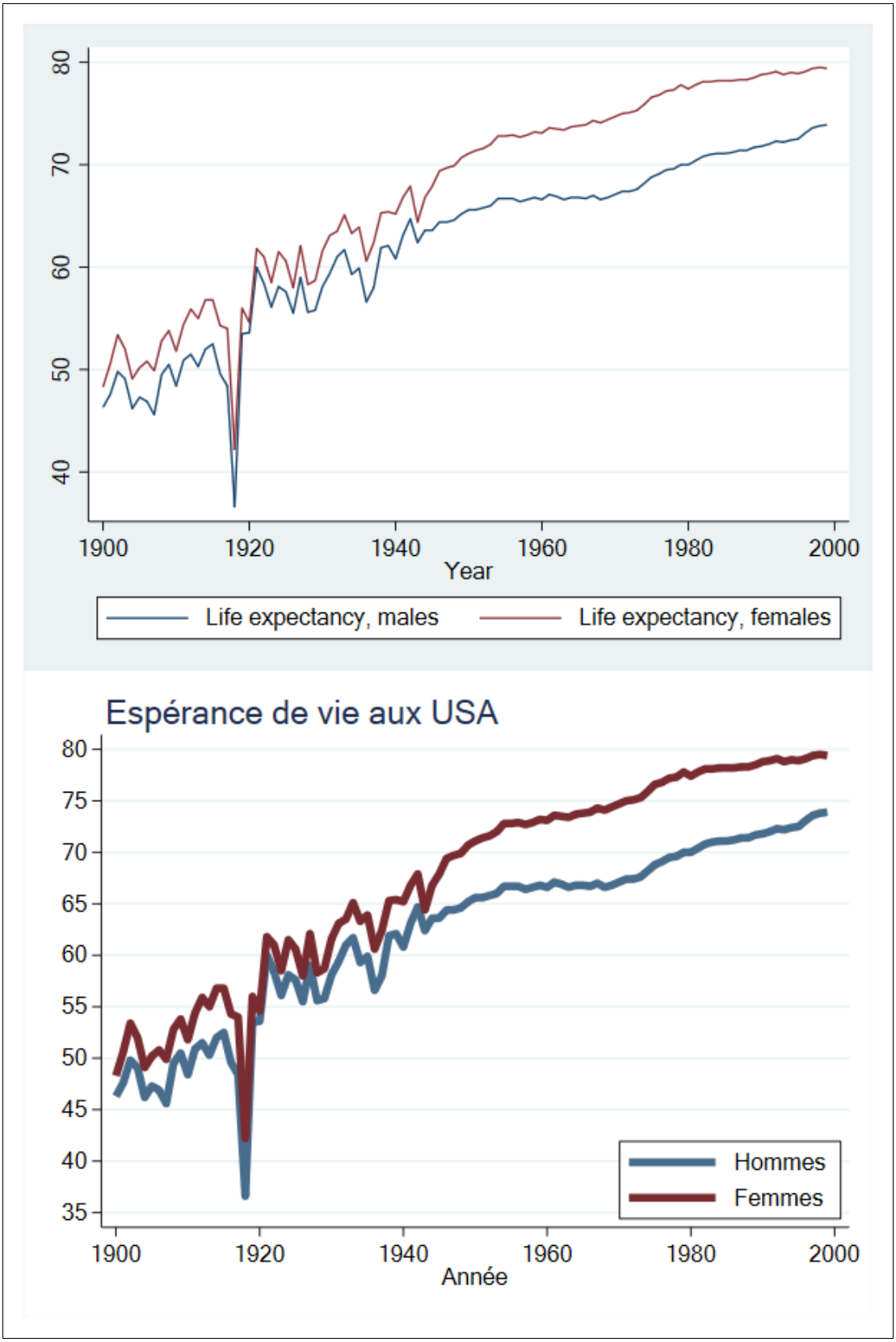
Un graphique comportant plusieurs géométries de même type peut être exécuté avec un seul bloc de coordonnées renseigné par des variables pour l'axe des ordonnées. Ces différentes coordonnées peuvent être des variables correspondant à plusieurs valeurs d'une variable additionnelle, souvent de type catégoriel [voir la commande **separate** plus bas].

```
[tw/graph] type_géométrie Y1 Y2 Y3.... [X] [, option1(Y1 Y2 Y3...)..options_du
 graphique]
```

Les options de la géométrie (plusieurs coordonnées pour Y) comme les couleurs ou les tailles/épaisseurs sont indiquées à la suite dans l'option choisies. Si on veut changer les couleurs de plusieurs courbes associées aux variable y1,y2 et y3: **color(couleurY1 couleurY2 couleurY3)**

Exemple

Graphique de type `line`, reportant les espérances de vie aux USA de 1900 à 2000. Le premier graphique est sans option, le second en comporte.



Premier graphique

```
tw line le_male le_female year
```

Second graphique

```
#delimit ;
tw line  le_male le_female year,

  lc(*.8 *1.2)
  lw(*4 *4)

  title("Espérance de vie aux USA", pos(11))
  legend(order(1 "Hommes" 2 "Femmes") pos(4) col(1) ring(0))
  ylabel(35(5)80, angle(0))
  xtitle("Année")
  graphr(color(white)) plotr(color(white))
;
```

Avec le changement de délimiteur, il est plus facile de distinguer les différents éléments du graphique, comme les options qui affectent directement les courbes (couleur, épaisseur) et les options générales du graphique (titre, légende, labels de y à l'horizontal, titre de x, couleur de fond).

Options des courbes

- **lc()**: je baisse la saturation de la courbe des hommes de 20% (plus claire) et j'augmente celle des femmes de 20% (plus foncée). Les couleurs par défaut sont conservées (palette Stata *s2color*).
- **lw()**: j'augmente fortement l'épaisseur des courbes (*4). Stata dispose de plusieurs unités pour altérer les tailles et épaisseurs, celle utilisée par défaut est une unité de type relative (voir la section dédiées plus bas).

Options du graphique

- J'ajoute un titre avec l'option **title(…)** que je positionne à 11 heures à l'extérieur avec l'argument **pos(11)**.
- Je change les labels de la légende dans l'option **legend()** avec l'argument **order()**. Je change la position de la légende en a mettant dans la zône du graphique avec **ring(0)** et à 4 heures avec **pos(4)**. Les labels sont reportés sur une colonne avec l'argument **col(1)**.
- Je modifie les labels des ordonnées avec l'option **ylabel()** , en changeant le delta des coordonnées reportées et en mettant les labels à l'horizontal avec l'argument angle(0).
- Je modifie le titre des abscisses avec l'option **xtitle()**.
- Je modifie les couleurs de fonds du graphique qui est composée de deux zônes : la zône où est réellement tracé le graphique avec l'option **plotr()** [**plotregion()**] et la zône externe avec **graphr()** [**graphregion**] où se trouve reporté par défaut les titres et les légendes. Pour changer la couleur de fond, on utilise l'argument **color()**.

Plusieurs blocs d'objets graphiques

Dans l'exemple précédent on avait deux variables, mais il est plus standard que les bases donnent des observations à partir d'une variable qui peut être croisées avec d'autres informations regroupées avec une variable discrète. Par rapport à d'autres langages graphiques, on pense bien évidemment à R avec ggplot, les choses se gâtent un peu du côté de Stata car plusieurs graphiques doivent être exécutés séparément dans la même commande. On donnera plus loin un moyen via les macros et les boucles d'automatiser cette exécution. Cela ne concerne que les éléments surperposés, Stata possédant en revanche une option pour générer des sous graphiques de type ***facettes*** (voir section sur les graphiques combinés).

Chaque sous graphique a son propre jeu d'options et ont doit bien indiquer leur séparation, soit avec des parenthèses soit avec des doubles barres horizontales.

```
(graph 1, options graph1) (graph2, options graph2)… (graph n, option graph
n) , options graphiques
```

ou

```
graph 1, options graph1 || graph2, options graph2 ||…. || graph n, option
graph n || , options graphiques
```

Ma préférence va plutôt à la deuxième solution pour éviter une surabondance de parenthèses déjà très présentes, et souvent imbriquées dans les options. Par ailleurs on visualise mieux la séparation entre les différents éléments graphiques.

Principe de la syntaxe avec ; comme délimiteur

Graphiques séparés par ()

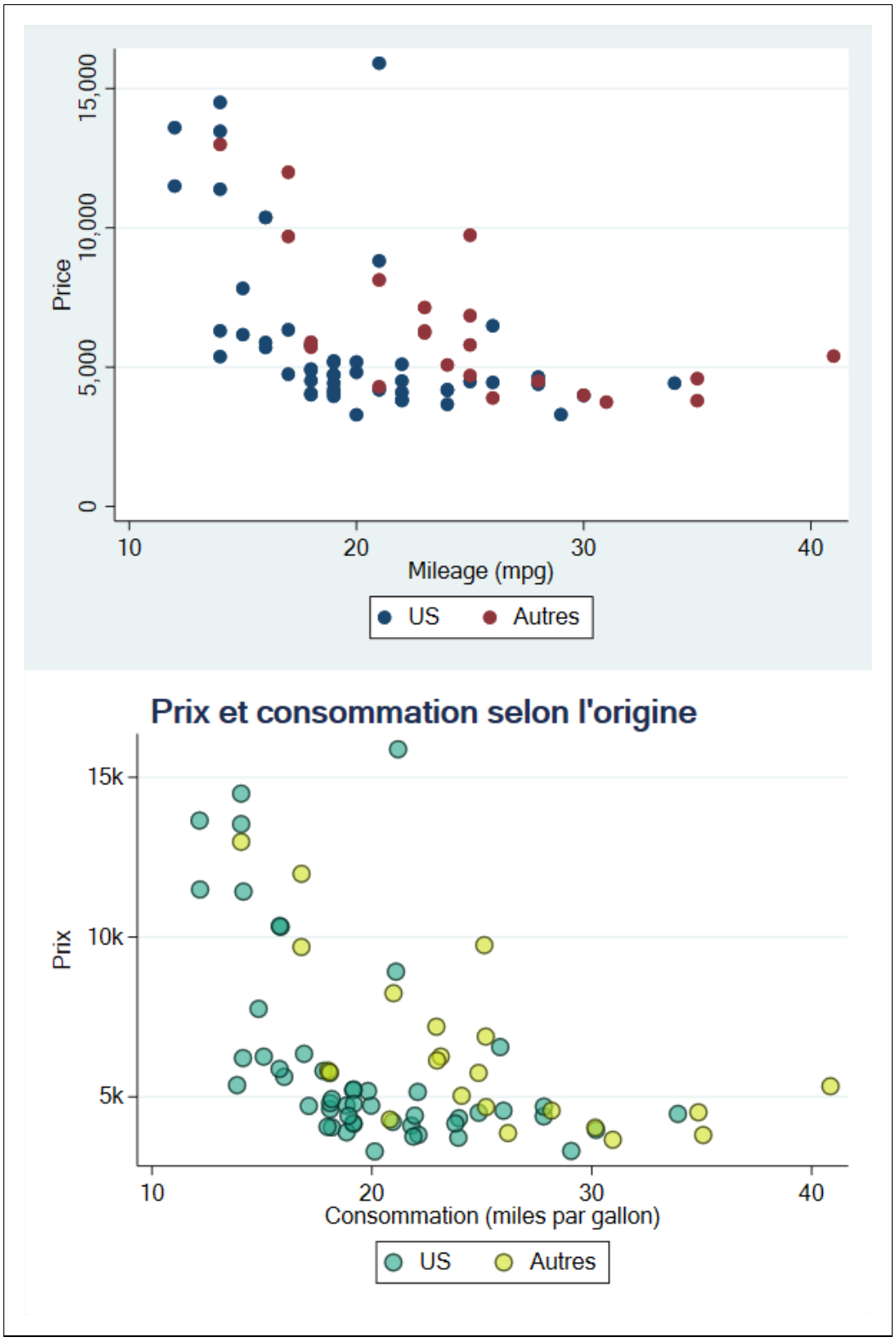
```
#delimit ;
[tw/graph]
(type_géométrie1 Y1 [X1] [Y2] [X2] [Z] [in if] [weight]
[, options(1)])

(type_géométrie2 Y2 [X1] [Y2] [X2] [Z] [in if] [weight]
[, options(2)])
...)

[, options_graphiques]
;
```

Graphiques séparés par ||

```
#delimit ;
[tw/graph]
   type_géométrie1 Y1 [X1] [Y2] [X2] [Z] [in if] [weight]
[, options(1)]

|| type_géométrie2 Y2 [X1] [Y2] [X2] [Z] [in if] [weight]
[, options(2)])

...
|| [, options_graphiques]
;
```

Exemple

Comme dans le premier graphique, celui du haut est sans option et celui du bas avec. On va afficher un nuage de point avec la base auto, entre les variables *price* et *mpg* (consommation d'essence en gallon) selon l'origine de la voiture (US ou autres pays).



Premier graphique

```
sysuse auto, clear

#delimit
tw
    scatter price mpg if !foreign
||  scatter price mpg if  foreign

||, legend(order(1 "US" 2 "Autres"))
;
```

Lorsqu'on superpose les sous graphiques de cette manière, Stata ne reconnait pas par défaut les labels à affecter à la légende, et affiche deux fois le label *price* pour chaque origine. On doit renseigner manuellement cette information.

Petite astuce pas forcément connue de toutes et tous. Lorsqu'une variable est de type indicatrice {0,1} il n'est pas nécessaire de préciser sa valeur.

!foreign est identique à foreign==0

foreign est identique à foreign==1

Second graphique

```
#delimit ;
tw
   scatter price mpg if !foreign,
   mc("31 161 135%60") msize(*1.5) mlc(black) mlw(.3) jitter(1)
|| scatter price mpg if foreign,
   mc("207 225 28%60") msize(*1.5) mlc(black) mlw(.3) jitter(1)

||, title("{bf: Prix et consommation selon l'origine}", pos(11))
    legend(order(1 "US" 2 "Autres"))
    ylabel(5000 "5k" 10000 "10k" 15000 "15k", angle(0))
    xtitle("Consommation (miles par gallon)")
    ytitle("Prix")
    graphr(color(white)) plotr(color(white))
;
```

Options des nuages

- **mc()** modifie la couleur des bulles. On verra par la suite plus en détail les altérations des couleurs, mais ici on a manuellement changé leur couleur avec un code RGB sur lequel on a réduit l'opacité à 60% (ou mis 40% de transparence).
- mlc() modifie la couleur du contour des bulles, ici en noir. Par défaut la couleur du contour est identique à la couleur de remplissage de la bulle.
- **msize()** modifie la taille de la bulle.
- **mlw()** modifie l'épaisseur du contour de la bulle.
- **jitter()** : très à la mode, les ***jitters*** sont des nuages dont les coordonnées ont été perturbées aléatoirement pour réduire les effets de superposition, ici liés à la variable *mpg*. A utiliser avec prudence, et éviter de trop modifier les coordonnées lorsque les coordonnées représentent des valeurs continues.

Options du graphique

- Le titre a été mis en gras avec une balise **SMCL** (*Stata Markup Control Langage*) : **"{bf :texte}"**.
- Les valeurs des prix reportés sur l'axe des ordonnées à été modifié manuellement avec des unités k (1k = 1000).
- Le reste des options sont de même nature que celles relative au graphique sur les espérances de vie.

Préférer un graphique avec un seul bloc d'objet ??

Plutôt que de multiplier le nombre de sous graphiques et d'options, on peut créer une variable pour chaque modalité avec la commande **separate** et générer plus facilement un graphique.

```
separate price, by(foreign)

#delimit ;
tw scatter price0 price1 mpg,
      mc("31 161 135%60" "207 225 28%60") mlc(black black)
      msize(*1.5 *1.5)  mlw(.3)
      jitter(1)

title("{bf: Prix et consommation selon l'origine}", pos(11))
legend(order(1 "US" 2 "Autres"))
ylabel(5000 "5k" 10000 "10k" 15000 "15k", angle(0))
xtitle("Consommation (miles par gallon)")
ytitle("Prix")
graphr(color(white)) plotr(color(white))
;

drop price0 price1
```

Encadré : boîte de dialogue et fenêtre d'édition

Générer un graphique :

- Ouverture d'une boîte dans la fenêtre *command* avec la commande *db*: db command
- Préférer *submit* à *OK* pour laisser la fenêtre ouverte

```
db tw
db histogram
```

Editer un graphique:

- On peut éditer manuellement le graphique après sa création dans la boîte d'édition du graphique, ou en le chargeant après sauvegarde
- On peut enregistrer les modifications faites dans l'éditeur avec *record* et les appliquer *play* pour une édition ultérieure. Un fichier au format *.grec* est enregistrée et peut être édité.
- Quelques améliorations de l'interface d'édition sont fournies avec la version 16 de Stata.

```
sysuse auto, clear
* sauvegarde en mémoire temporaire
tw scatter price mpg, name(g1, replace)
graph display g1 [, scheme() play()....]
* sauvegarde en dur
tw scatter price mpg, save(g1, replace)
graph display g1 [, scheme() play()....]
```

Options

Il s'agit ici d'un tour d'horizon forcément incomplet des options qui vont permettre de modifier et d'habiller les graphiques, afin d'en améliorer la visibilité. On va regarder principalement les options liées aux couleurs, tailles/épaisseurs ; et quelques paramètres de bases pour les axes, légendes, titres. L'idée est de ne traiter que des options qui sont utilisées au moins une fois dans le document.

Un très bon *cheat sheet*:

https://geocenter.github.io/StataTraining/pdf/StataCheatSheet_visualization15_Syntax_2016_June-REV.pdf

https://geocenter.github.io/StataTraining/pdf/StataCheatSheet_visualization15_Plots_2016_June-REV.pdf

Couleurs et épaisseurs

Couleurs

3 Eléments entrent dans les options relatives aux couleurs :

Nom - code couleur

Un nom de couleur prédéfini ou un code couleur numérique : la couleur *navy* (première couleur de la palette Stata *s2color*) a pour code **RGB** [Red-Green-Blue] "26 71 11".

Pour un élément de type ligne, courbe… : **`lc(navy) = lc("26 71 11")`**

Saturation/intensité

On peut modifier la saturation : clair (blanc) ⇔ foncé (noir). Une couleur par défaut ou renseignée manuellement a une intensité de 1. On tire vers le blanc en réduisant cette saturation, vers le noir en l'augmentant. Après le nom ou le code couleur, l'intensité est modifiée par ***#*** avec # supérieur ou égal à 0 (0=blanc).

Pour un élément de type barre, l'option pour modifier une couleur est donnée par fc() (f pour fill) : fc(*.5) réduit la saturation de 50% de la couleur par défaut**, `fc("26 71 11*1.2")`** augmente de 20% la saturation de la couleur navy, ici renseignée par son code RGB.

Transparence

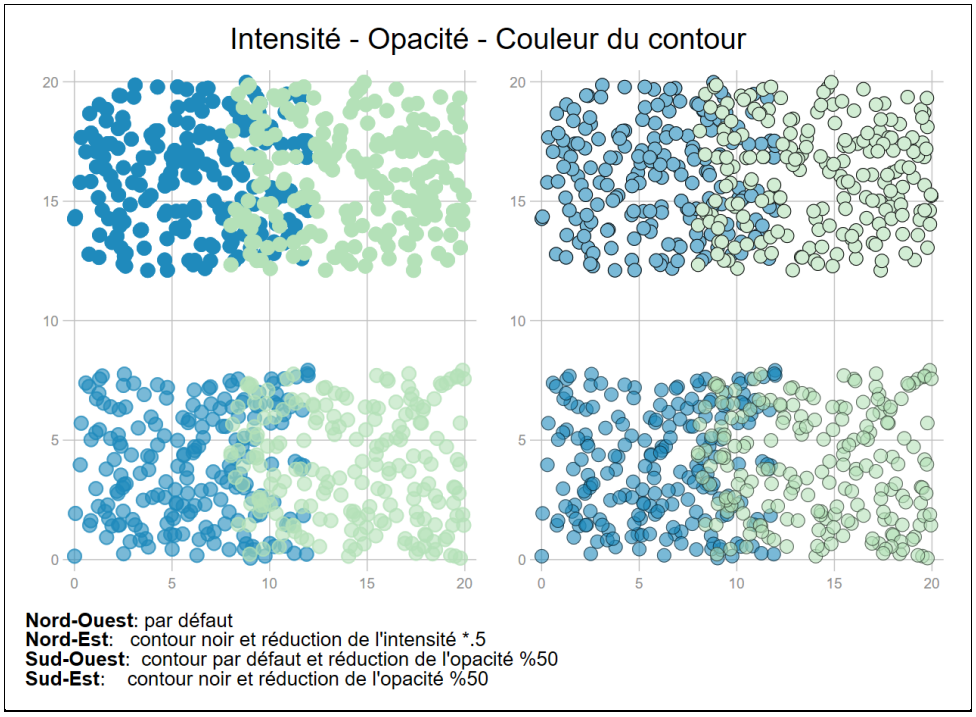
En retard sur ce point jusqu'à la version 15 de Stata, on peut maintenant modifier l'opacité (transparence) des couleurs.

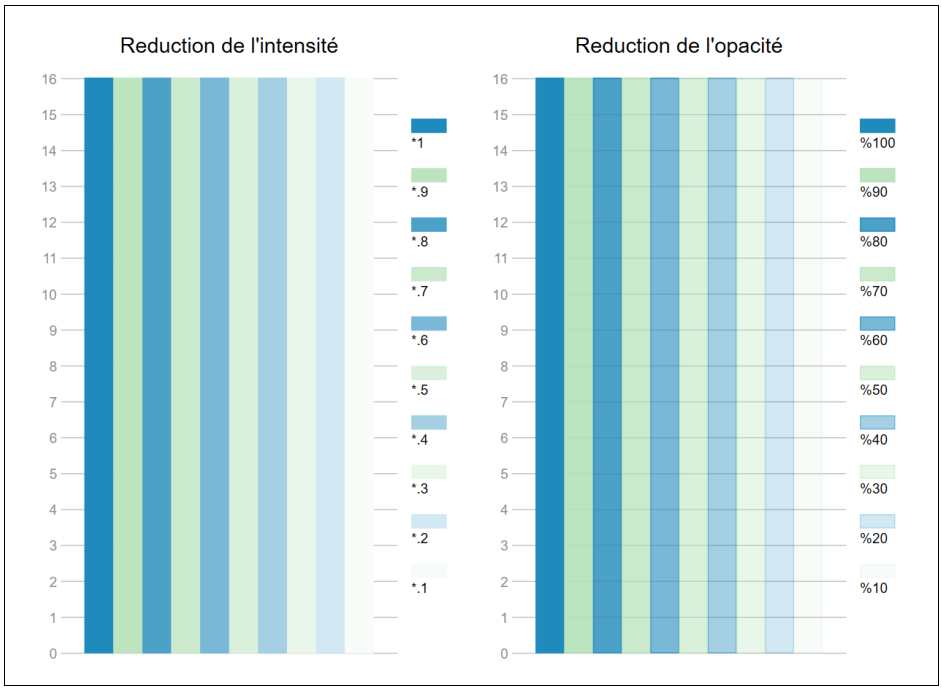
La transparence permet de gérer les effets de superposition, mais on doit se méfier des effets de flou qui peuvent se révéler assez désagréable, en particulier pour les nuages en l'absence de contour sur les bulles. L'argument de l'option exprime un pourcentage d'opacité avec valeur minimale 0 et une valeur maximale 100 (100 = valeur par défaut). Après le nom ou le code de la couleur, l'opacité est réduite par **%#**. On peut utiliser une transparence totale **(%0)** pour cacher des éléments d'un graphique.

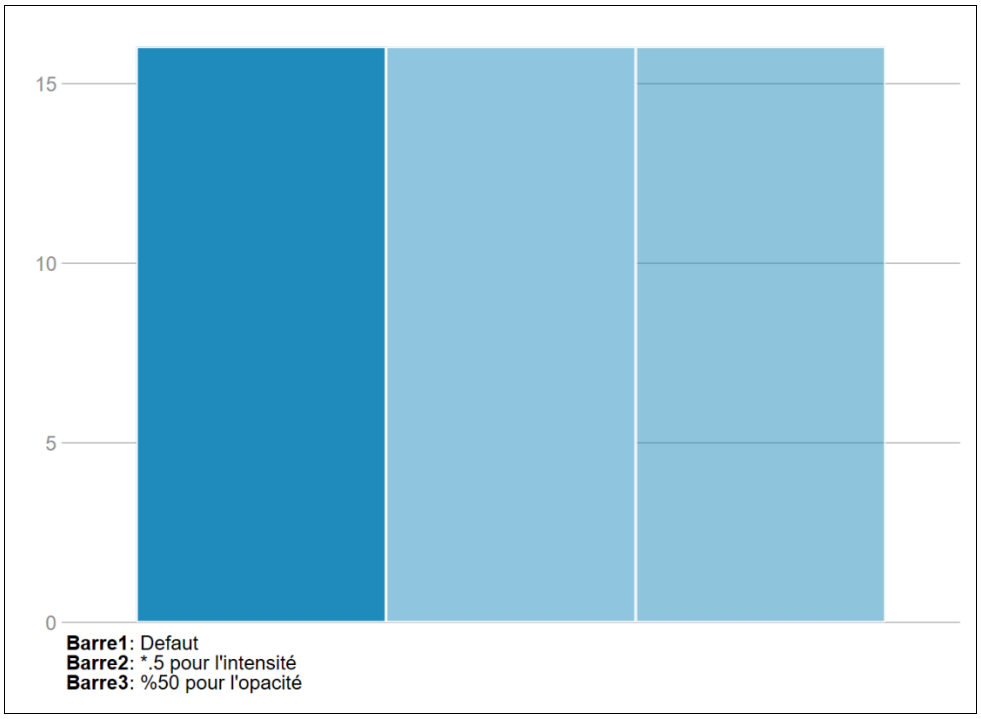
Pour un élément de type bulle : **`mc(%50)`** reduit de moitié l'opacité de la couleur par défaut, **`mc("26 71 11%70")`** applique 70% de transparence à la couleur *navy*.

Comme on le voit dans le second graphique ci-dessous, il n'y a pas trop de sens à baisser simultanément la saturation et l'opacité, une couleur plus transparente étant plus claire. On peut à l'inverse augmenter la saturation et baisser l'opacité.

Exemples







Le tableau qui suit donne le nom des options relatives aux couleurs, avec leur expression tronquée (`color` = `c`). Comme un nombre important de types de géométrie peuvent être associés à `tw line` dont ils partagent les options, ils ne sont pas forcément tous recensés (`help twoway`).

Graphiques twoway (non exhaustif)

`scatter -scatteri`	**`mc`**	Remplissage et contour
	`mfc`	Remplissage
	`mlc`	Contour
`line, pci` et associés	**`lc`**	Couleur de la courbe
`connected`		Options de scatter et line
`area - area - bar`	**`col`**	Remplissage et contour
	`fc`	Remplissage
	`lc`	Contour
	`fi`	Intensité du remplissage
`dropline`		Options de scatter et line
`dot`	**`dc`**	Remplissage et contour
	`dfc`	Remplissage
	`dlc`	Contour
`contour - contourline`	**`sc`**	Couleur de départ
	`ec`	Couleur d'arrivée
	`cc`	Liste de couleurs pour chaque niveau
	`crule`	Règle de transition entre couleurs
	`colorl`	Pour `contourline` seulement - une couleur par niveau
`lfitci`	**`clc`**	Couleur de la droite de régresion
	`acol`	Remplissage et contour de l'intervalle
	`fc`	Remplissage de l'intervalle
	`acl`	Contour de l'intervalle

Graphiques oneway

`graph matrix`		Options de `tw scatter`
`graph bar`		Options de `tw area…`
`graph dot`		Options de `tw dot`
`graph box`		Options de `tw bar - line - scatter`
`graph pie`	**`color`**	Couleur de la part

Axes

xlabel - ylabel	**`labc`**	Couleur des labels
	`tlc`	Couleur des coches
	`glc`	Couleur du grid
Xscale - yscale	**`lc`**	Couleur de la ligne de l'axe

Légende

	`color`	Couleur du texte
Dans la sous option `region()`	**`color`**	Couleur de remplissage et du contour de la zône
	`fc`	Remplissage de la zône
	`lc`	Contour de la zône

Exemple :

```
legend(order( 1 "A" 2 "B") title("Légende", color("red"))
 region(fc(yellow) lc(black)))
```

Titres (title, xtitle, ytitle…), notes (note, caption), texte libre (text)

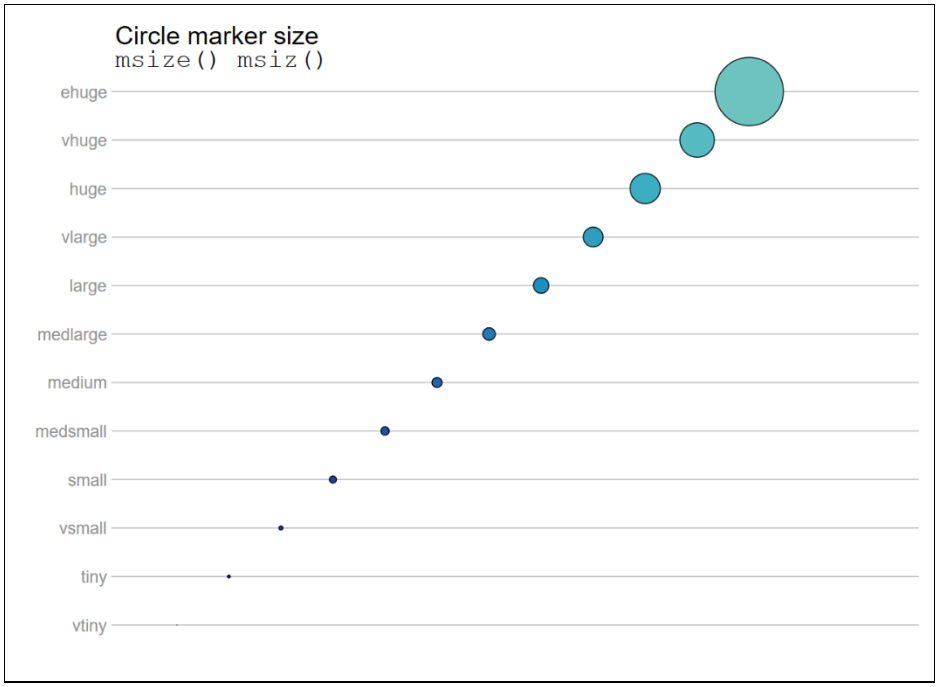
	`color`	Couleur du titre
Avec l'option box	`bc`	Remplissage et contour de la zône
	`fc`	Remplissage de la zône
	`lc`	Contour de la zône
Exemple: `title("title", box bc(red) color(white))`		

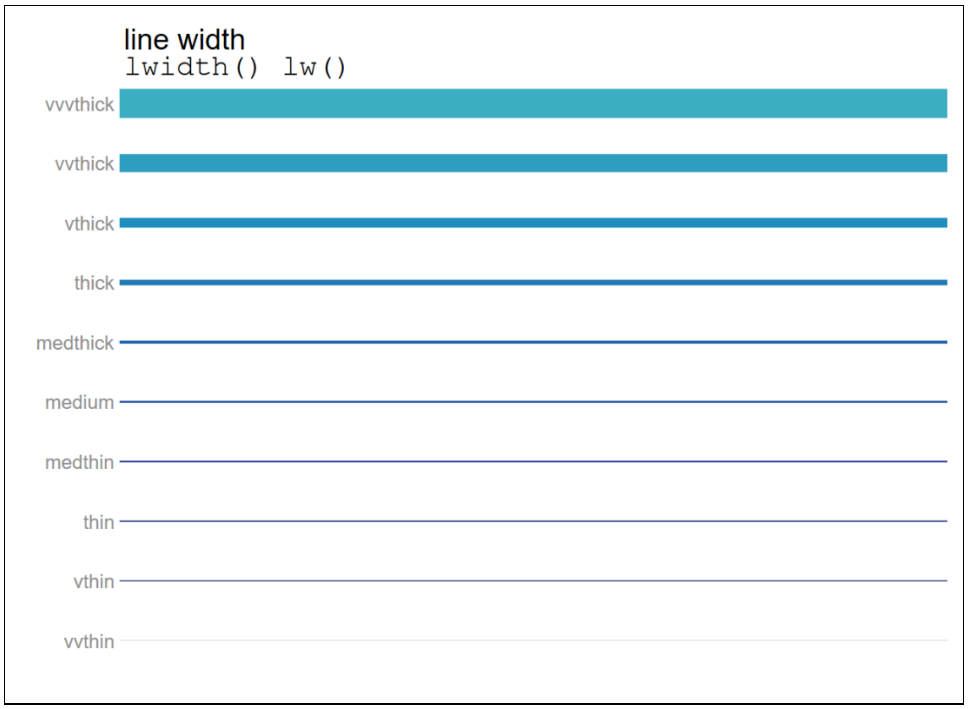
Tailles/Epaisseurs

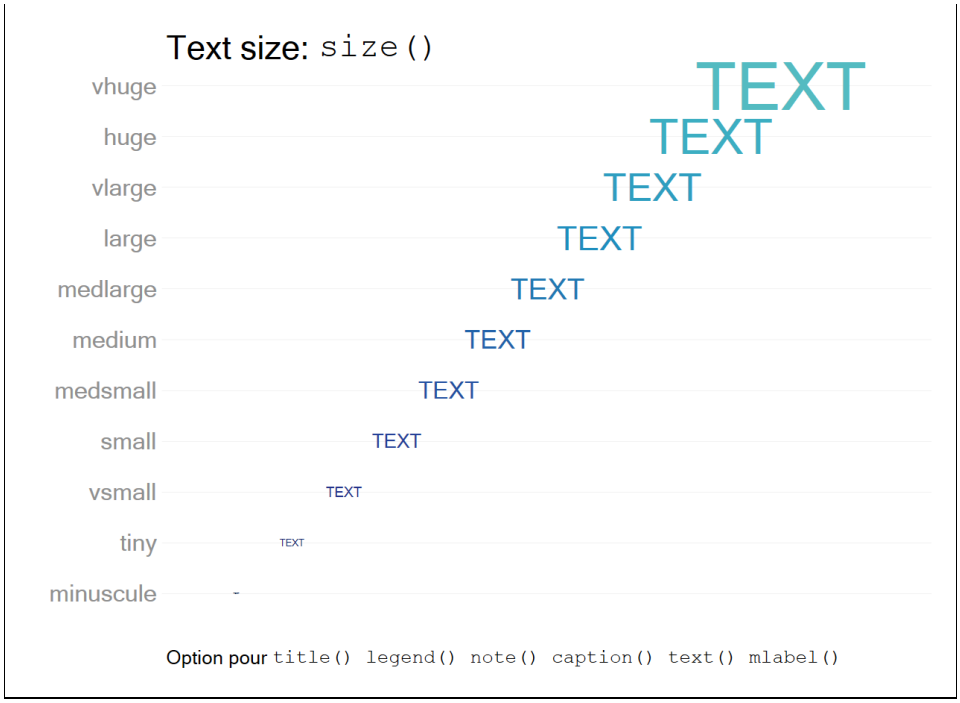
On peut altérer les tailles et épaisseurs d'éléments composants le graphiques en s'appuyant soit sur des arguments prédéfinis, soit en jouant sur les valeurs de plusieurs types d'unités, absolues (cm, point, pouce) ou relatives. Certaines ont été implémentées à la version 16 de Stata.

La mémorisation des épaisseurs et tailles prédéfinies pouvant s'avérer un peu laborieuse, avec une liste qui diffère selon le type d'objet, on pourra privilégier la modification des valeurs des unités directement.

Tailles prédéfinies







Valeurs paramétrables

Pour modifier directement les valeurs des tailles et épaisseurs Stata propose 6 unités : 3 absolues et 3 relatives, ces dernières étant un peu obscure au niveau de la définition de la valeur de référence.

donne la valeur de la modification.

Absolues (par ordre décroissant) : le pouce (#in), le centimètre (#cm), et le point (#pt)

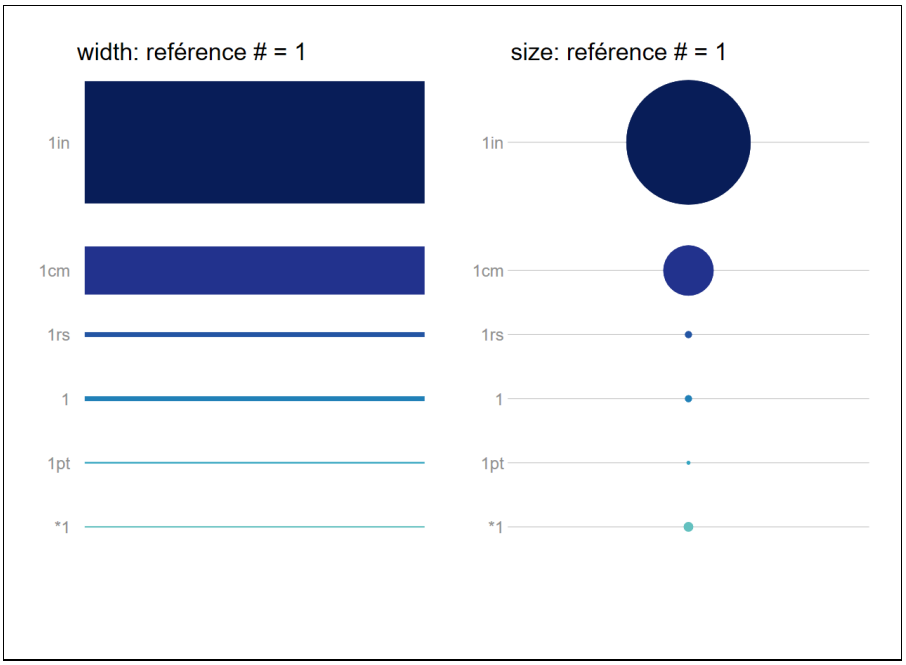
Relatives :

- **`option(#rs)`**: valeur relative par rapport au minimum de la longueur et de la largeur du graphique (=100). Si `option(.5rs)` et que le graphique fait 10 de largeur et 20 de longueur, la référence est associée à la largeur et est égale à 100. La taille ou l'épaisseur

de l'élément fera alors .5% de cette référence. L'aide de Stata ne s'attarde pas sur ce type de modification, pas forcément très claire.

- **`option(#)`** : également une valeur par rapport au minimum de la largeur et de la longueur, mais on garde cette valeur comme référence. Avec `option(.5)` et que le minimum est toujours égal à 10 (largeur), la taille ou l'épaisseur de l'élément est égale à .5% de 10.

- **`option(*#)`** : coefficient multiplicateur d'une taille ou épaisseur par défaut définie par le style (thème) du graphique. Je privilégie ce type de paramétrisation.



[Faire le même tableau que pour les couleurs]

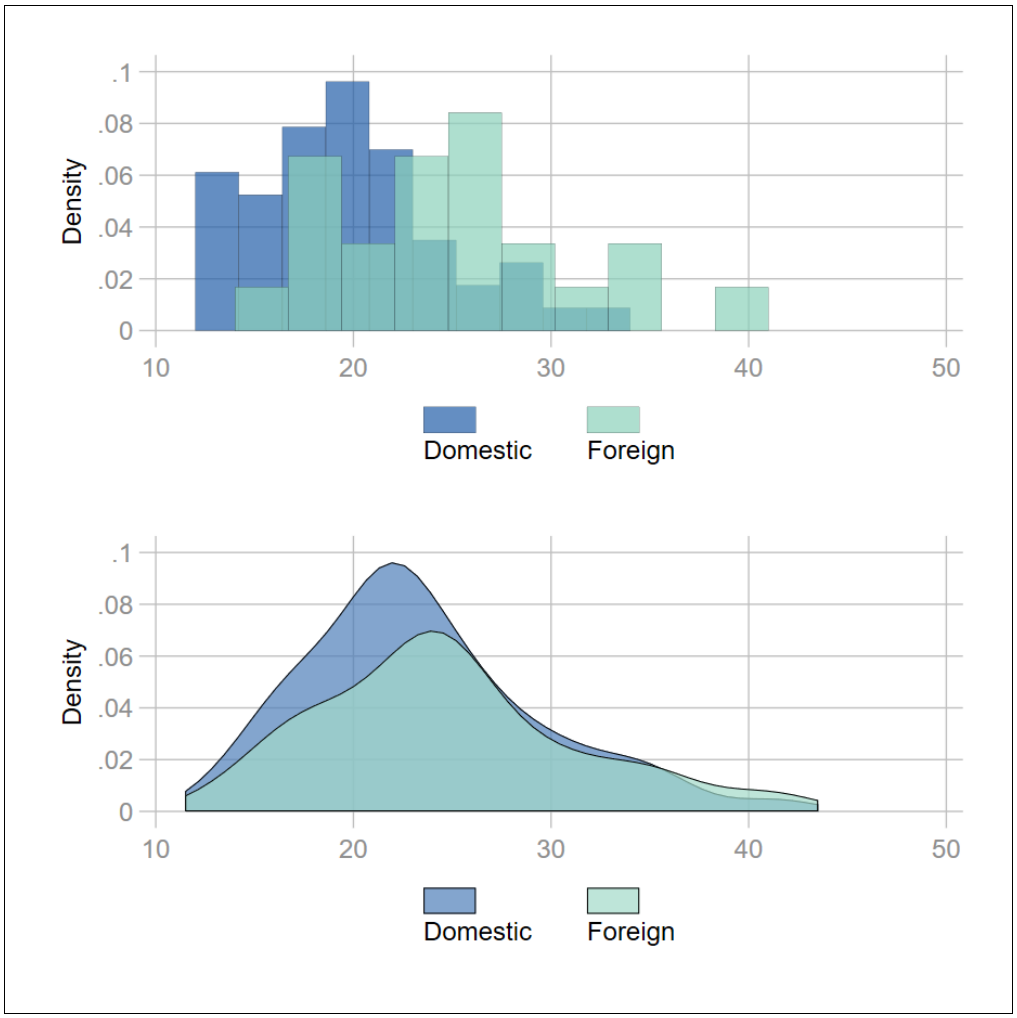
Exercice

Reproduire les graphiques ci-dessous avec la base *auto* [`sysuse auto.dta`]

Programmes : https://github.com/mthevenin/stata_fr/blob/master/exercices/exo1

Graphique(s) I

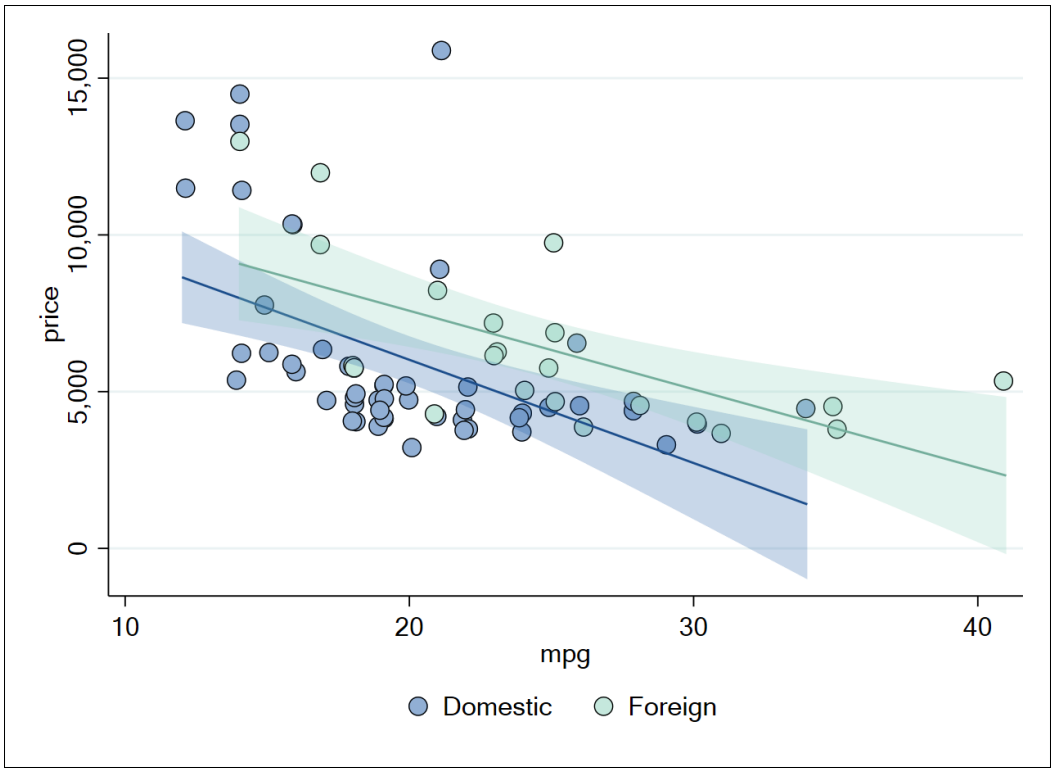
- Reproduire l'un et/ou l'autre de ces graphiques avec les variable *mpg* et *foreign* .
- Les histogrammes sont générés avec la fonction **`tw histogram`** (à préférer à la commande de type oneway **`histogram`**).
- Les densités sont estimées avec la fonction **`kdensity`** [`help kdensity`] et le graphique est généré avec **`tw area`** [`help twoway area`].
- Les couleurs sont issues de la palette de type séquentielle ***YlGnBu*** de la collection *Brewer* (voir le chapitre sur les palettes de couleur), les codes des deux couleurs sont "34 94 168" et "139 209 187".



Graphique II

Reproduire le graphique suivant :

- Nuage de points des variables **price** et **mpg** pour les deux modalités de la variable *foreign*, avec report des OLS.
- La droite de l'OLS est générée avec le type de graphique **tw lfitci**:
    - Couleur de la droite : option **`clc()`**
    - Couleur de remplissage l'intervalle de confiance: option **`fc()`**
    - Couleur du contour de l'intervalle de confiance: option **`alc()`**



Options pour les axes

Il y a 3 types options pour modifier les axes x/y x ou y) : x/ylabel(), x/ytitle(), x/yscale().

Une autre option permet de définir le choix de l'axe si un graphique comporte plusieurs définitions d'axes pour les abscisses et/ou pour les ordonnées : x/yaxis(#) avec # le numéro de l'axe, qui sera par la suite reporté dans les autres options relatives aux axes.

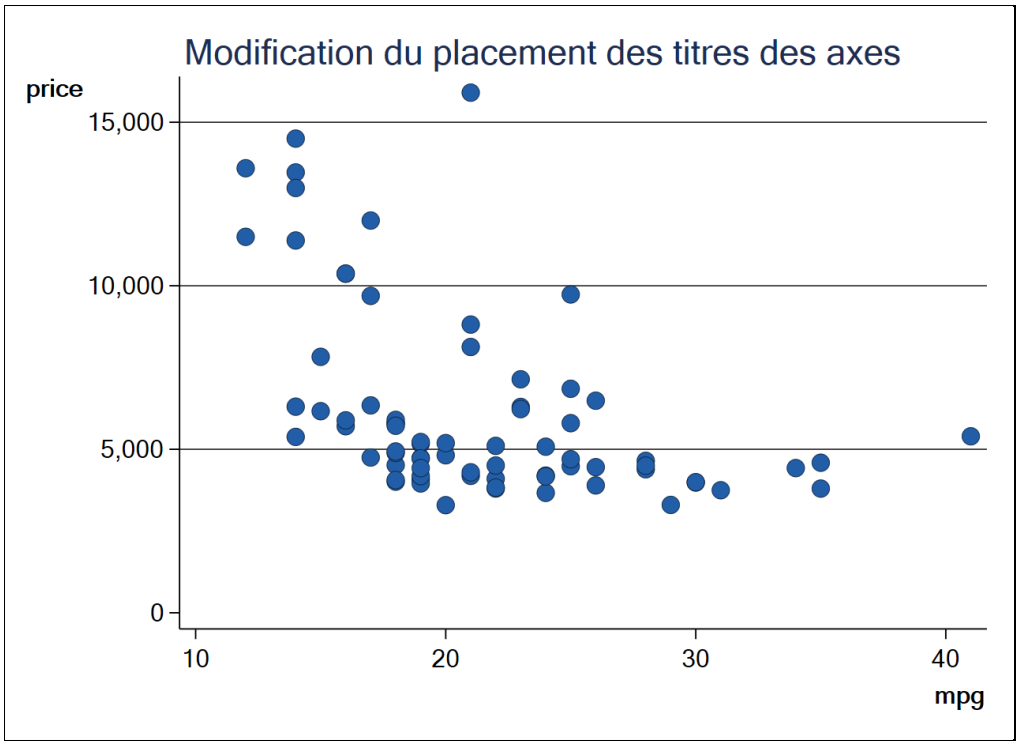
Titres

`xtitle()` `ytitle()` `ttitle()` `ztitle()`: `xtitle(["titres"] [, options])`

Principales options:

- Taille **size()**, couleur **color()**. Si on ajoute l'option box, on peut modifier l'apparence de la zone qui l'entoure.
- Titres sur plusieurs lignes: **xtitle("ligne1" ligne2"....[, options])**
- On peut modifier la position des titres des axes avec **`place(top/right/bottom/left)`** pour indiquer la position le long de l'axe (par exemple `right` pour *x* et `top` pour *y*) et contrôler leur éloignement de l'axe avec **width(#)** pour y et **height(#)** pour x. Le graphique suivant modifie les positions par défaut avec les options :

```
xtitle(", place(right) height(5))
ytitle(", orient(horizontal) place(top) width(5))
```



Coordonnées, labels, grid

`xlabel() ylabel() tscale() zscale()`: **`xlabel([coordonnées] [,options])`**

Modification des valeurs reportées sur les axes:

- Avec **valeur_min (delta) valeur_max**: **`xlabel(0(5)10)`**.
- En indiquant le nombre de valeurs à reporter sur l'axe : **`xlabel(#)`**.
- Manuellement: **`xlabel( #1 #2 "label" "label2"…)`** ou **`xlabel( #1 "label1" #2 "label2"...)`**.
  Exemple: **`xlabel(0 5 10)`** ou **`xlabel(0 "Zero" 5 "Cinq" 10 "Dix")`**.
- On peut mélanger les approches: **`xlabel(0 .5 1(1)10 15 20 "Vingt" )`** .

Principales options:

- Taille et couleur du texte avec **`labs()`** et **`labc()`**.
- Alterner les positions pour éviter les chevauchements avec **`alt`**.
- Modifier l'angle de report des labels: **`angle(#)`** avec # valeur de l'angle [0 (horizontal), 45, 90 (vertical) …].
- Modification du grid et des coches (tick): la couleur avec **`glc()`**et **`tlc()`** et l'épaisseur avec **`glw()`** et **`tlw()`**.

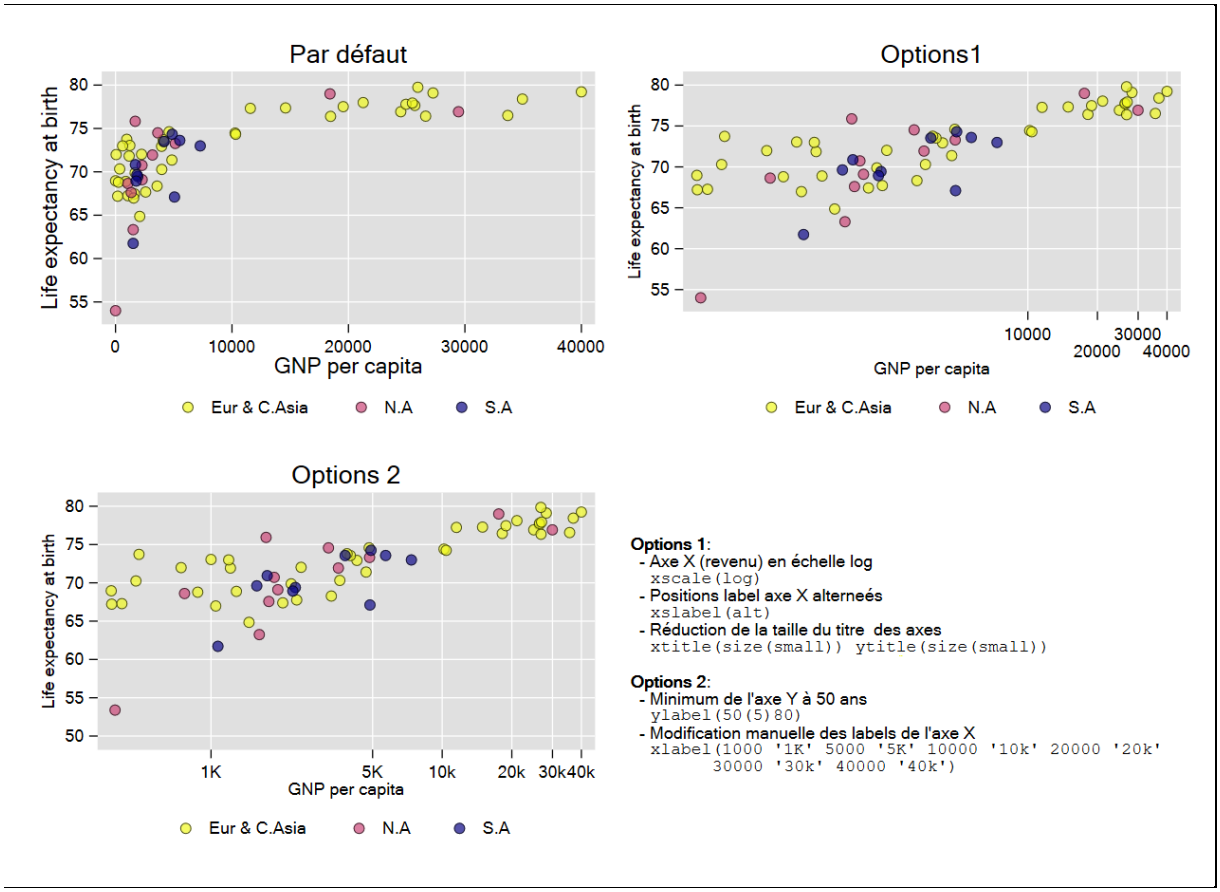
Echelles

`xscale() yscale() tscale() zscale()`

- Utilisation d'une échelle logarithmique: **`log`**. Seule l'échelle est modifiée, les valeurs d'origines sont reportées sur l'axe, ce qui est plutôt bien vu.

- Echelle en ordre décroissant : **`reverse`**. Pas conseillé, mais on peut trouver une certaine utilité aux graphiques combinés reportant des distributions croisées et marginales (voir la section dédiée aux graphiques combinés).
- Supprimer l'affichage des axes : **`off`**.
- Ne pas tracer la ligne parcourant l'axe : **`noline`**.
- Modifier les limites de l'axe : **`range(min max)`**.

Les graphiques qui suivent donnent quelques exemples de modification des options des axes. Les données sont issues de la base d'exemple *lifeexp* et les graphiques reportent sous forme de nuage les espérances de vie à la naissance (y) et le revenu par habitant (x).



Warning : pour la dernière option reportée la zône en bas à droite qui est un graphique « vide », le label doit-être indiqué par des doubles quotes "**label**" et non par **'label'** : `xlabel(1000 "1k" 5000 "5k" 10000 "10k" 20000 "20k" 30000 "30k" 40000 "40k")` et non `xlabel(1000 '1k'……)`. J'ai du composer avec la syntaxe permise pour les éléments de type texte dans un élément de type note.

Ajouter un point sur les axes multiples avec exemple

Options pour la légende

`legend(), clegend(), plegend: legend(contenu [options])`

Les options **`clegend()`** et **`plegend()`** sont réservées aux graphiques de type courbes de niveau avec une troisième dimension (hauteur, densité…): `clegend()` pour **`tw contour`** et `plengend()` pour **`tw contourline`**. Ces options ont des arguments spécifiques que je ne décrirais pas ici.

Stata utilise par défaut le label de la variable *y* pour alimenter le contenu de la légende.

- Ne pas afficher une légende: `legend(off)`.
- Position par défaut: 6 heures (Sud) – à l'extérieur (*graphregion*). Modifiable avec **`pos(#)`** et **`ring(0 ou 1)`**.
- Pour modifier le nombre de lignes et de colonnes: **`row(#)`** et **`col(#)`**
- Pour modifier les labels de la légende:
  order() : `order( 1 "label1" 2 "label2" .....)`
  ou
  lab() : `lab(1 "label1") lab(2 "label2") ...)`
  J'utilise de préférence `order` plus parcimonieux en parenthèses. L'apparence (gras, italique…) des labels est modifiable avec des balises **smcl** ; les tailles et couleurs sont également modifiables.
- Les labels peuvent être reportés sous les symboles avec l'option **`stack`**.

- On peut modifier l'aspect de la zône (couleur du fond, contour…) avec la sous option **`region()`** ou **`r()`**.

Encadré : le smcl

Le Smcl « Stata Markup Control Langage » est le langage d'édition de Stata, il faut le reconnaître un peu préhistorique (ce n'est pas *Markup* pour rien). Pour les graphiques, il permet de modifier les textes des titres, labels des axes, contenu des légendes : gras, italique… changement de police, ajout de formules.

Balise smcl : `"{type_option: texte}"`

Quelques options/modifications de base :

Gras : `"{bf: texte}"`

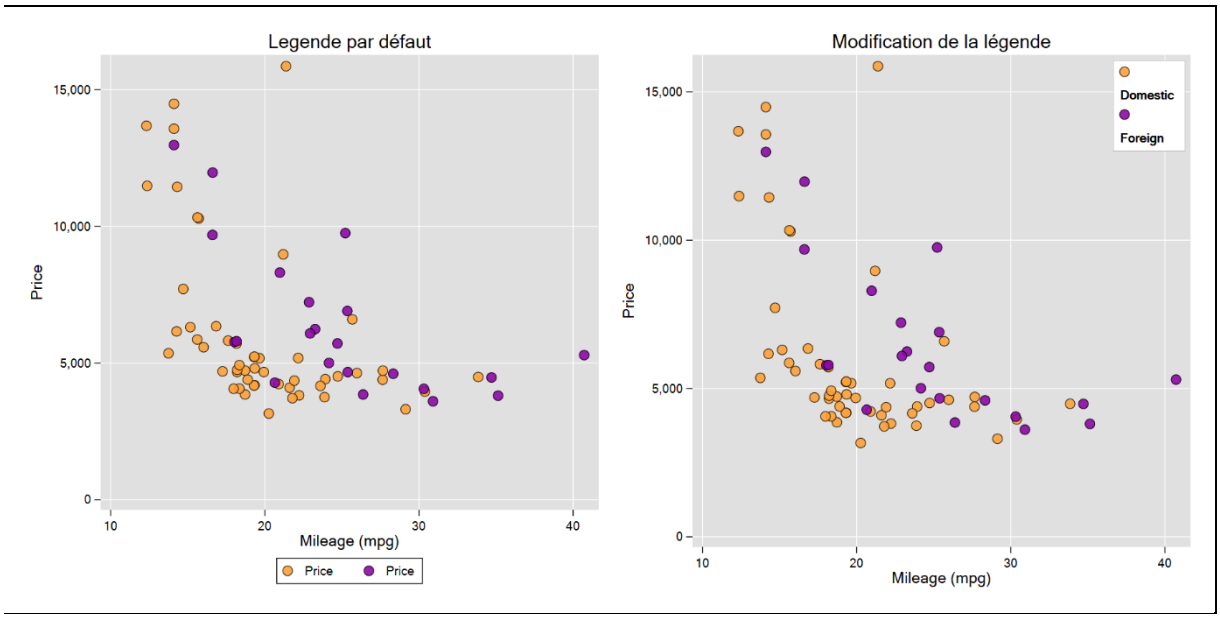
Italique : `"{it: texte}"`

Gras et italique : `"{bf:{it :texte}}"`

Le choix des polices dépend du système d'exploitation. Elles sont sélectionnées par leur type : **`"{stSans : texte}"`** pour type sans serif, **`"{stSerif : texte}"`** pour type serif, **`"{stMono: texte}"`** pour type mono, et **`"{stSymbol : texte}"`** pour les lettres grecques ou les symboles mathématiques.

Exemple police avec windows : **arial** (sans serif), **times new roman** (serif), **`courrier new`** (mono) et **symbol** (symbol).

On pourra utiliser toutes les typographies disponibles en les sélectionnant en amont, avec la commande **`graph set`**.



Options de la légende :

```
legend(order(1 "{bf:Domestic}" 2 "{bf:Foreign}") pos(1) ring(0) col(1)
region(lw(*.1)) stack )
```

- Le texte de labels a été mis en gras avec une balise Smcl.
- La légende a été positionnée à l'intérieur du graphique avec **`ring(0)`** et à 1 heure avec **`pos(1)`**.
- La légende est reportée sous forme de colonne avec **`col(1)`**.
- L'épaisseur du contour de la zône a été fortement réduit avec **`r(lw(*.1))`**.
- Le texte est positionné sous le symbole avec l'option **`stack`**.

Options pour les titres, notes, texte libre

La majeure partie des options sont déjà précisées plus haut pour les titres des axes. Les titres et les notes sont comme les légendes positionnés par rapport à des valeurs de l'horloge (de 1 à 12). Pour le texte libre on renseigne directement les coordonnées.

`title()` - **`subtitle()`**
Il y a également des options pour des titres qui sont positionnés à gauche, à droite, au dessus (mais en dessous du titre principal) ou en bas du graphique : `l1title()` et `l2title()` pour un titre à gauche (l pour left), `r1title()` et `r2title()` à droite (r pour right), `t1title()` et `t2title()` pour un titre au dessus (t pour top), et enfin `b1title()` et `b2title()` pour un titre en bas (b pour bottom) .

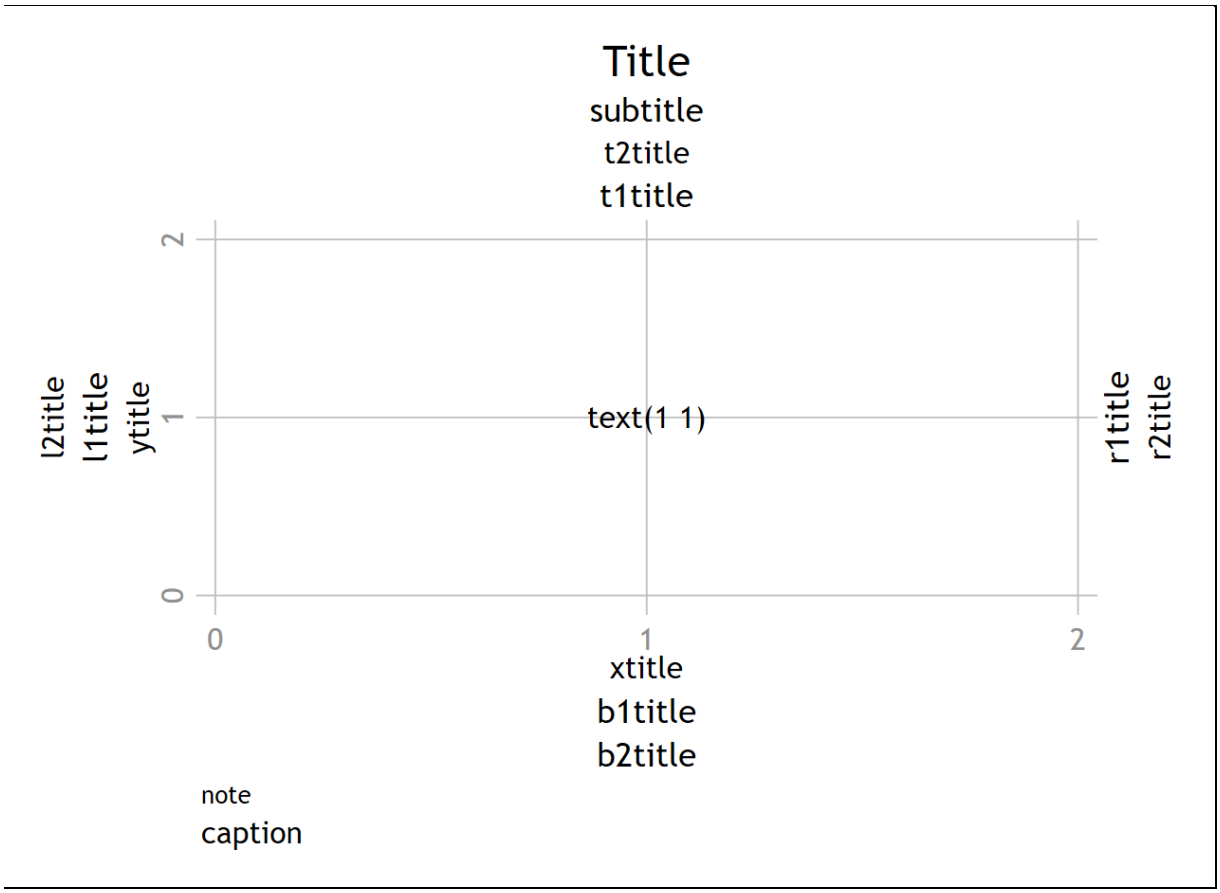
Par défaut le titre principal est positionné à 12 heures. Le sous titre est positionné en dessous du titre principal.

`note()` - **`caption()`**
`note()` est positionné par défaut à 6 heures, **`caption()`** en dessous de `note()`.

Ces options permettent de faire des notes de lecture ou de préciser une ou plusieurs sources au graphique.

text(coordonnées "texte" [,options])
Cette option permet d'afficher un texte dans la zône du graphique en entrant directement les coordonnées (*y*,*x*). Les sous options relatives à l'alignement peuvent être assez utiles si on ne veut pas tâtonner avec les coordonnées (on peut passer par les macros pour générer les coordonnées, voir le chapitre dédié).

Le graphique suivant présente les options par défaut (position, taille, couleur) des titres, dont le titre des axes, ainsi qu'un texte libre positionné aux coordonnées (1,1).



Autres

Changer les polices par défaut

On peut paramétrer en amont la typographie utiliser par les graphiques avec la commande graph set : **graph set window fontface[type_police] "nom_police"**

Le type de police est: sans, serif, mono, symbol.

Par défaut :

```
window           current
setting          default          choices
-------------------------------------------------------------
fontface         Arial             font name
fontfacesans     Arial             font name
fontfaceserif    Times New Roman   font name
fontfacemono     Courier New       font name
fontfacesymbol   Symbol            font name
-------------------------------------------------
```

Exemple: avec *Trebuchet MS* pour *fontface* et consolas pour *mono*

```
graph set window fontface"trebuchet MS"
graph set window fontfacemono "consolas"
```

```
window           current
setting          default          choices
-------------------------------------------------------------------------
fontface         trebuchet MS      font name
fontfacesans     Arial             font name
fontfaceserif    Minion Pro        font name
fontfacemono     consolas          font name
fontfacesymbol   Symbol            font name
-------------------------------------------------------------------------
```

Taille du graphique

Ces options sont particulièrement utiles pour les graphiques combinés lorsque l'empilement de plusieurs graphiques provoque un effet d'écrasement des axes (axes x plus court si plusieurs colonnes par exemple).

xsize(#) - **ysize(#)** : en pouce. Peut s'appliquer à un graphique individuel, ou plus souvent à la commande **graph combine** pour retrouver un bon ratio x/y pour les sous graphiques.

fxize(#) – **fysize(#)** : en %. A utiliser seulement pour combiner des graphiques. Avant la combinaison on peut appliquer ces options aux graphiques individuels pour contrôler les ratios x/y. Cela permet plus de liberté dans les compositions des graphiques combinés.

Une application sera donnée plus loin dans la section dédiée aux graphiques combinés.

Modifier automatiquement la taille du texte et des symboles

scale(#) - iscale(#)
Permet de modifier la taille des éléments types texte et symboles générés par un graphique. Utile lorsque le texte d'une légende sort du contour. Valeur de référence=1.
Pour la commande `graph combine`, on utilise l'option `iscale()` qui applique la modification à tous les sous graphiques.

Options d'affichage et d'enregistrement

nodraw

Equivalent de `quietly` pour les graphiques, il permet de ne pas afficher le graphique dans l'éditeur. Plutôt à utiliser pour les sous-graphiques qui seront combiner ou associer à l'option saving. Permet de gagner du temps d'exécution.

name() - graph display – saving() – graph use

name(nom_graph [,replace])
Permet de sauvegarder temporairement le graphique sur la session. On peut privilégier cette option pour combiner des graphiques ultérieurement.
Utiliser de préférence l'option replace pour écraser un graphique déjà en mémoire et avec un même nom, à défaut de quoi il faut supprimer le ou les graphiques en mémoire avec **graph drop noms_graphs** ou **graph drop _all** en amont.

graph display [nom_graph]
Permet d'afficher un graphique en mémoire (non sauvegardé en dur). Si le nom du graphique n'est pas précisé, le dernier graphique généré est ouvert dans l'éditeur de graphique.

Exemple:

```
tw line y x, name(g1,replace)
graph display g1
```

saving("path/nom"[, replace])
Sauvegarde en dur sur le disque.

graph use "nom_graph"
Permet d'afficher un graphique enregistré.

export nom_graph.format [, replace name(nom du graph ouvert dans l'éditeur)]
Stata peut convertir un graphique en plusieurs format : jpeg, png, gif (seulement mac), svg…. On peut le faire directement dans l'éditeur de graphique ou passer par la ligne de commande export. Pour utiliser cette commande, le graphique doit être ouvert dans l'éditeur.
Les formats d'exportation on des options propres.

Exemples:

```
tw line y x
graph export graphpng.png
tw line y x, name(g1,replace)
graph export graphpng.png, replace name(g1)
```

Combiner des graphiques

Facettes automatiques

Les **facettes** ou ***small multiples*** permet de combiner rapidement plusieurs graphiques répétés pour différentes valeurs d'une variable additionnelle, généralement catégorielle. Stata est plutôt efficace pour produire ce genre de visualisation avec l'option **by()** dans la commande graphique. On peut néanmoins regretter le manque de possibilité concernant l'habillage des sous graphiques avec, par exemple, une même couleur imposée à l'ensemble des sous graphiques.

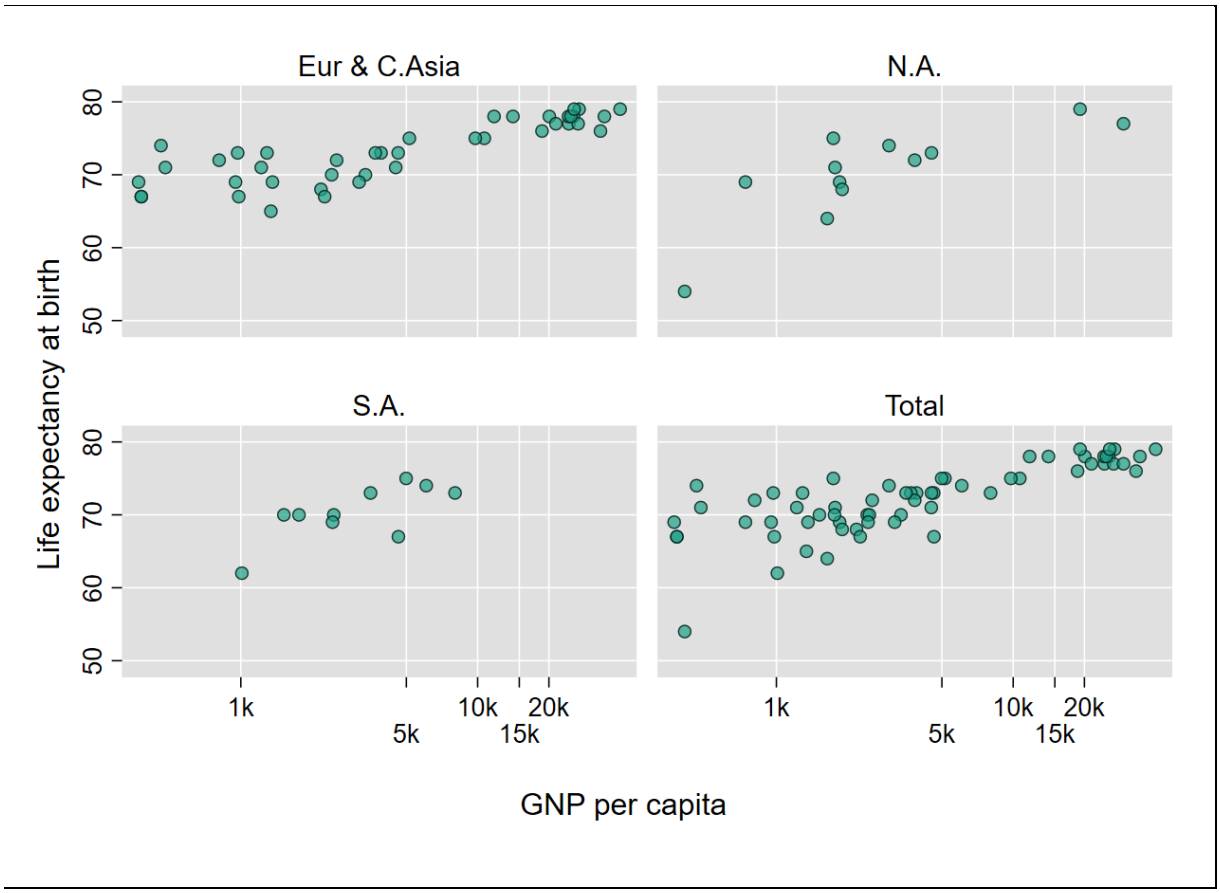
Syntaxe : **graphique, by(variable [, total col(#) row(#)…])…**

Avec l'argument **total** en option de **by()**, on peut ajouter un graphique additionnel représentant le graphique avec toutes les valeurs, donc sans l'option by. On peut également modifier l'affichage des sous-graphiques par ligne et colonne avec **row()** et **col()**.

Le graphique génère automatiquement une note « **Graphs by nom_variable** » plutôt dispensable. Si l'on souhaite supprimer cette note, on ajoute **note(" ")** en option de **by()**.

Exemple:

```
sysuse lifeexp
#delimit ;
tw scatter lexp gnppc,
by(region, total note(" "))
mc("31 161 135%70") mlc(black) mlw(*.6)

xlabel(1000 "1k" 5000 "5k" 10000 "10k" 15000 "15k" 20000 "20k" , alt)
xscale(log)
;
```



Combinaison libre

Deux commandes sont disponibles pour combiner plusieurs sous-graphique :

- **`graph combine`**: commande officielle.
- **`grc1leg`**: commande externe qui permet d'afficher une seule légende lorsque les sous graphiques partagent la même légende. Installation : `ssc install grc1leg`.

Conseils et options pour les sous-graphiques

- Les sous-graphiques devant être enregistrés, préférer **`name(nom, replace)`** à **`save(path/nom, replace)`**.
- Une fois les sous-graphiques validés, utiliser l'option **`nodraw`** pour raccourcir le temps d'exécution.
- **`margin(#1 #2 #3 #4)`** : si l'on souhaite modifier l'espace entre les sous graphiques on peut utiliser cette sous-option de l'option **`graphr()`**. Par défaut **`margin(0 0 0 0)`** avec **`margin(`***`left right top bottom`***`)`**. L'application en fin de section précisera par l'exemple l'intérêt de cette option. Voir également `imargin()` plus bas pour appliquer automatiquement une modification identique à tous les sous graphiques.
- Lorsqu'un graphique sera combiné à un autre, les options **`xsize()`** et **`ysize()`** sont sans effet. On peut alors utiliser les options **`fxsize()`** et **`fysize()`**. De même, l'option **`scale(#)`** est sans effet et sera remplacée par l'option **`iscale()`** du graphique combiné.

- Générer un graphique vide : en particulier lorsque le nombre de sous-graphiques est impair, on peut générer un ou plusieurs graphiques vides pour ajouter un titre, une note…., ou affiner la mise en page. La syntaxe de ce graphique est donnée ci-dessous, ainsi qu'un moyen de récupérer facilement sa syntaxe via une macro.

Syntaxe et récupération d'un graphique vide

Syntaxe

```
tw scatteri 1 1, ylab(,nogrid) xlab(,nogrid) mc(%0) xtitle("") ytitle("")
yscale(off noline) xscale(off noline) name(gv,replace)
```

Pour récupérer cette syntaxe on peut l'enregistrer dans une macro en dur, de type global. On peut soit exécuter directement ce graphique vide, soit le modifier pour ajouter, par exemple, un élément de type texte, ou modifier son habillage comme la couleur de fond.

Enregistrement dans une macro :

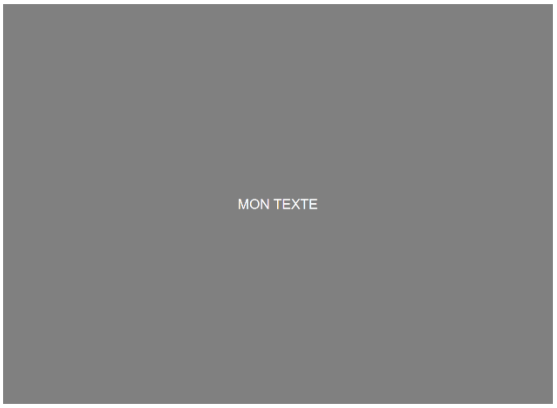
```
global gvide "scatteri 1 1, ylab(,nogrid) xlab(,nogrid) mc(%0) xtitle("")
ytitle("") yscale(off noline) xscale(off noline) name(gv,replace)"
```

Exécution directe du graphique vide sans affichage (ajout de `nodraw`) :

```
tw $gvide nodraw
```

Récupération de la syntaxe et ajout d'un texte libre et modifier l'habillage :

```
mac list gvide
tw $gvide text(1 1 "MON TEXTE", color(white)) ///
          graphr(color(gs8)) plotr(color(gs8))
```



Syntaxe et options

Syntaxe générique

```
graph combine liste_sousgraph [, col() row() xsize() ysize() iscale() xcommon
ycommon …..]

grc1leg graph combine liste_sousgraph [, legendfrom(nom_graph) col() row()
xsize() ysize() iscale() xcommon ycommon…..]
```

Options du graphique combiné

- **col(#)** et **row(#)** permettent d'indiquer le nombre de colonnes et de lignes pour la mise en page.
- **xsize()** et **ysize()** permettent de contrôler les effets d'écrasement si le nombre de colonnes et de ligne n'est pas identique.
- **iscale(#)** permet de réduire les éléments types texte et symbole pour l'ensemble des sous graphique (remplace `scale()` pour un graphique individuel).
- **imargin(#1 #2 #3 #4)** modifie pour chaque sous graphique la distance en la limite de la zône du graph (`graphregion`) et le bord du graphique. Dans l'ordre `imargin(`*`left right top bottom`*`)`. Par défaut imargin(0 0 0 0), des valeurs positives augmentent la distance, des valeurs négatives la réduise. On peut modifier cette distance pour le graphique combiné avec `graphr(margin(#1 #2 #3 #4)`.
- **xcommon** et **ycommon** forcent les graphiques à partager les même valeurs pour les axes.
- **legendfrom(nom_graph)**: **pour la commande `grc1leg` seulement**, permet d'afficher une seule légende lorsque les sous-graphiques partage la même.

Application : distributions croisée et marginales

L'idée est de proposé un graphique devenu assez standard en visualisation, représentant pour deux variables continues leur distribution croisée, ici sous forme d'un nuage, et leur distribution marginale, ici sous forme d'histogramme. Ce type de graphique est également souvent représenté dans une version combinant des courbes de niveaux et des densités simples.

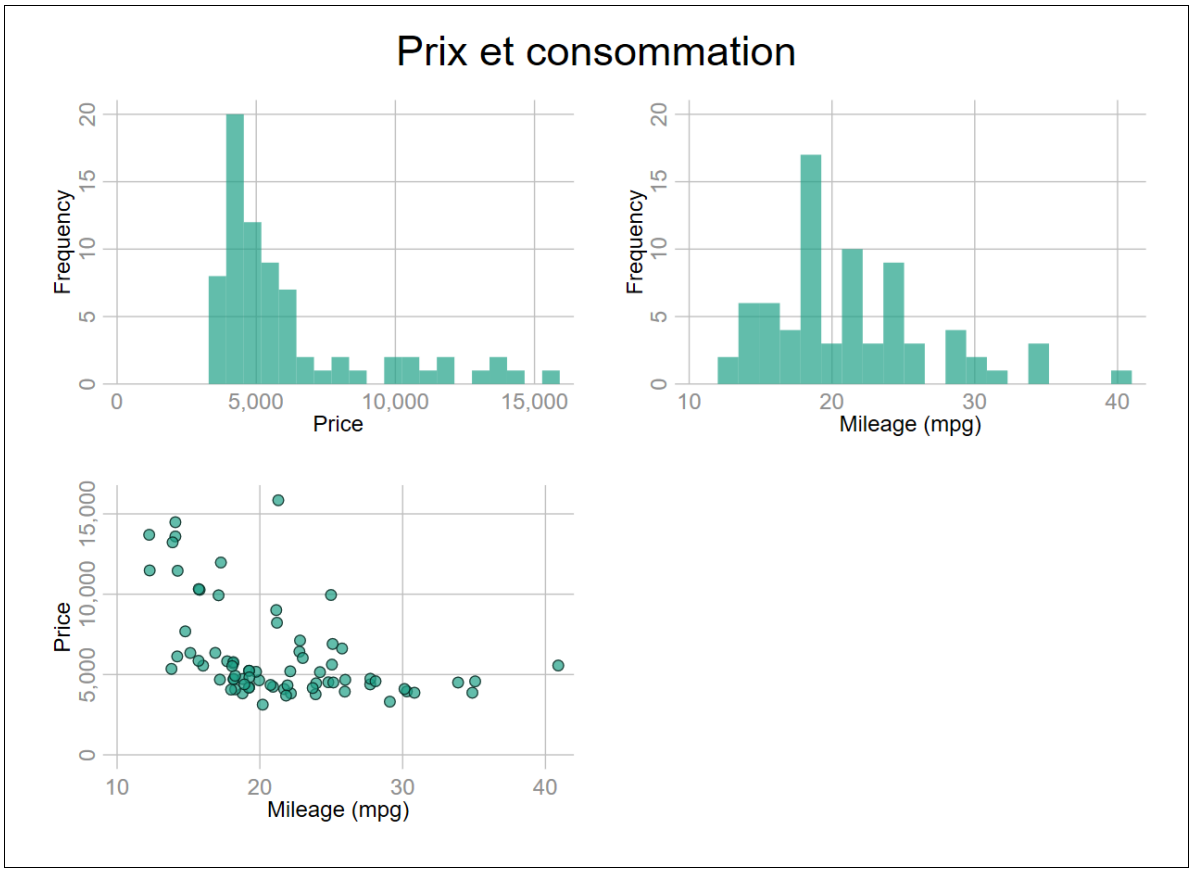
On prendra ici de nouveau les variables *price* et *mpg* de la base *auto*.

Dans un premier temps on combinera simplement les 3 graphiques, dans un second temps on ajoutera un graphique vide avec un élément titre à la seconde colonne de la première ligne, enfin on proposera un graphique définitif avec deux variantes, qui modifie la position du titre.

Au niveau des options je n'ai pas reporté les altérations du style par défaut pour alléger le report de la syntaxe.

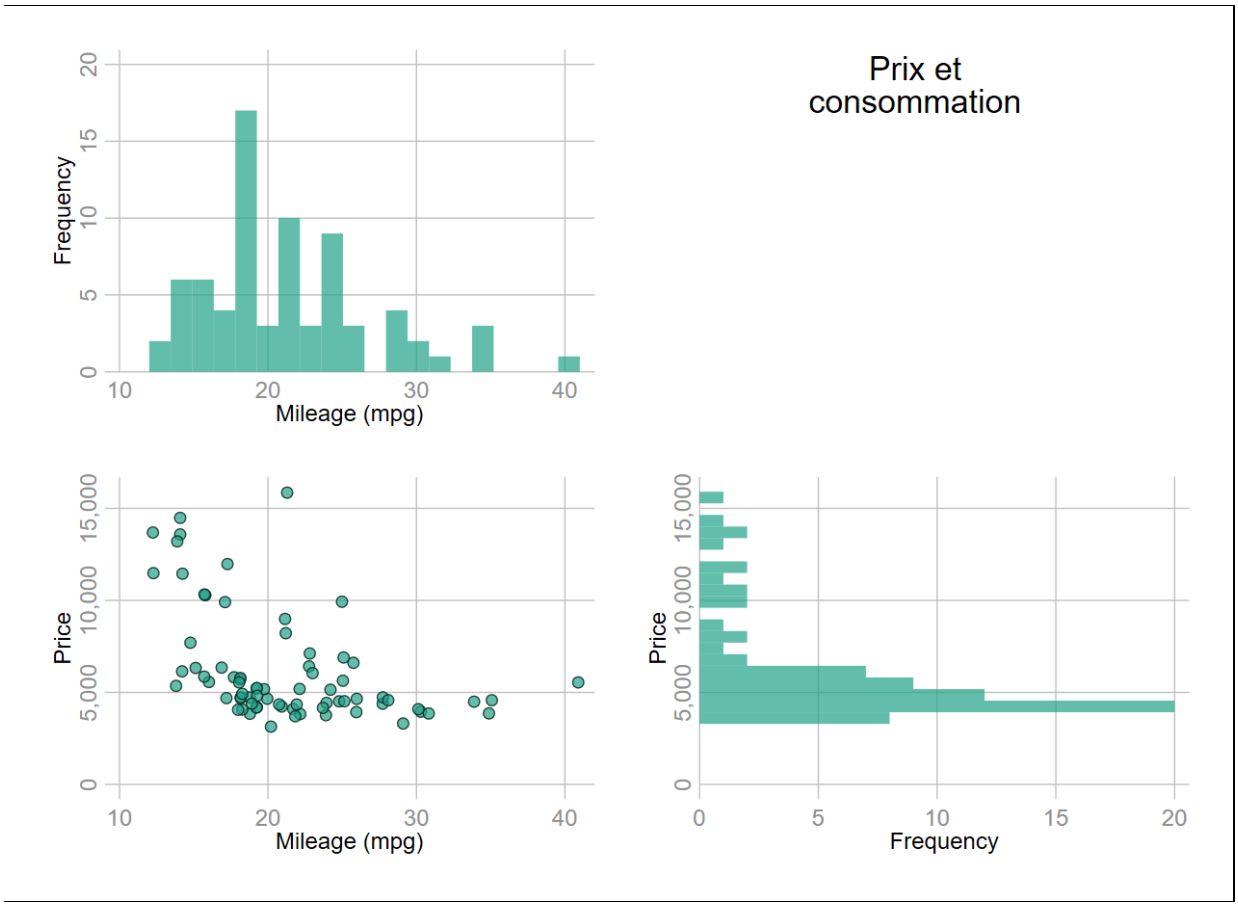
Graphique combiné simple

```
sysuse auto
#delimit ;
histogram price, bin(20) freq
fc("31 161 135%70") lc(%0) mlw(*6)
name(g1,replace)
;
histogram mpg, bin(20) freq
fc("31 161 135%70") lc(%0)
name(g2,replace)
;
tw scatter price mpg,
mc("31 161 135%70") mlc(black) mlw(*.6)
jitter(1)
name(g3,replace)
;
graph combine g1 g2 g3, title("Prix et consommation")
;
```



On ajoute un graphique vide dans lequel on indiquera le titre du graphique, et on positionne l'histogramme de la variable *price* à l'horizontal (option `horizontal`).

```
tw $gvide title("Prix et" "consommation")

#delimit ;
histogram price, bin(20) freq
fc("31 161 135%70") lc(%0) mlw(*6)
horizontal
name(g1,replace)
;

histogram mpg, bin(20) freq
fc("31 161 135%70") lc(%0)
name(g2,replace)
;

tw scatter price mpg,
mc("31 161 135%70") mlc(black) mlw(*.6)
jitter(1)
name(g3,replace)
;
graph combine g2 gv g3 g1 ;
```



Pour la version suivante, on va réduire la taille des abscisses pour g1 (*price*) et le graphique vide, la taille des ordonnées pour g2 (*mpg*) et le graphique vide avec `fxsize()` et `fysize()`. On va également cacher plusieurs titres et labels d'axes et réduire les distances d'espacement entre les sous graphiques avec l'option `imargin(#1 #2 #3 #4))` de **`graph combine`**. La distance d'espacement avec le bord du graphique combiné sera par contre augmentée pour compenser.

Titres et labels d'axes seront cachés et non pas supprimés. Si on les enlève, on modifie le ratio d'affichage du graphique et les axes reportés ne seront alors pas correctement ajustés entre le nuage et les histogrammes. Visuellement cela augmente la distance de séparation entre les graphiques, d'où sa réduction avec `imargin()`.

Modification des tailles des sous graphiques :

- **`histogramme price (g1): fxsize(50) fysize(100)`**
- **`histogramme mpg (g2): fxsize(100) fysize(50)`**
- **`graphique vide (gv): fxsize(50) fysize(50)`**

Titres et labels des axes cachés :

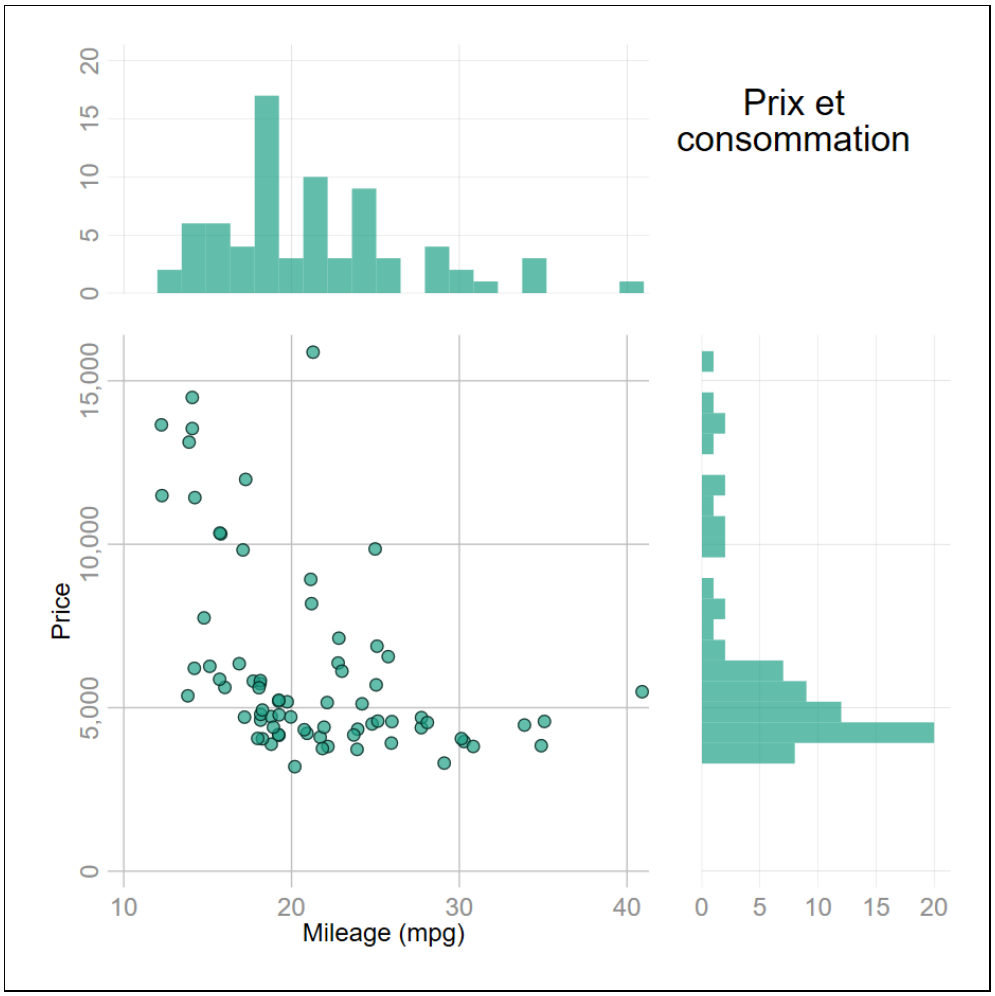
- **titre: `title(, color(0%)`**
- **label: `xlabel(,labc(%0)) ylabel(,labc(%0))`**

Modification des distances avec les bords:

Pour les sous graphiques, comme on applique la même modification, on utilise l'option `imargin(-5 -5 -5 -5)` de `graph combine`. C'est équivalent à utiliser pour chaque sous

graphique `graphr(margin(-5 -5 -5 -5)`. Pour que la distance entre les bords extérieurs des sous-graphiques ne soient pas trop proche de la limite du graphique combiné, on augmente la distance avec `graphr(margin(5 5 5 5))` appliqué à `graph combine`.

```
#delimit ;
histogram price, bin(20) freq
fc("31 161 135%70") lc(%0) mlw(*6)
horizontal
xtitle(, color(%0))
ytitle(, color(%0))
xlabel(,glw(*.2))
ylabel(, labc(%0) glw(*.3))
fxsize(50) fysize(100)
name(g1,replace)
;

histogram mpg, bin(20) freq
fc("31 161 135%70") lc(%0)
xtitle(,color(%0))
ytitle(,color(%0))
ylabel(, glw(*.2))
xlabel(, labc(%0) glw(*.3))
fxsize(100) fysize(50)
name(g2,replace)
;

tw scatter price mpg,
mc("31 161 135%70") mlc(black) mlw(*.6)  jitter(1)
fxsize(100) fysize(100)
name(g3,replace)
;

graph combine  g2 gv g3 g1,
xsize(20) ysize(20) col(2) imargin(-5 -5 -5 -5) graphr(margin(5 5 5 5))
;
```

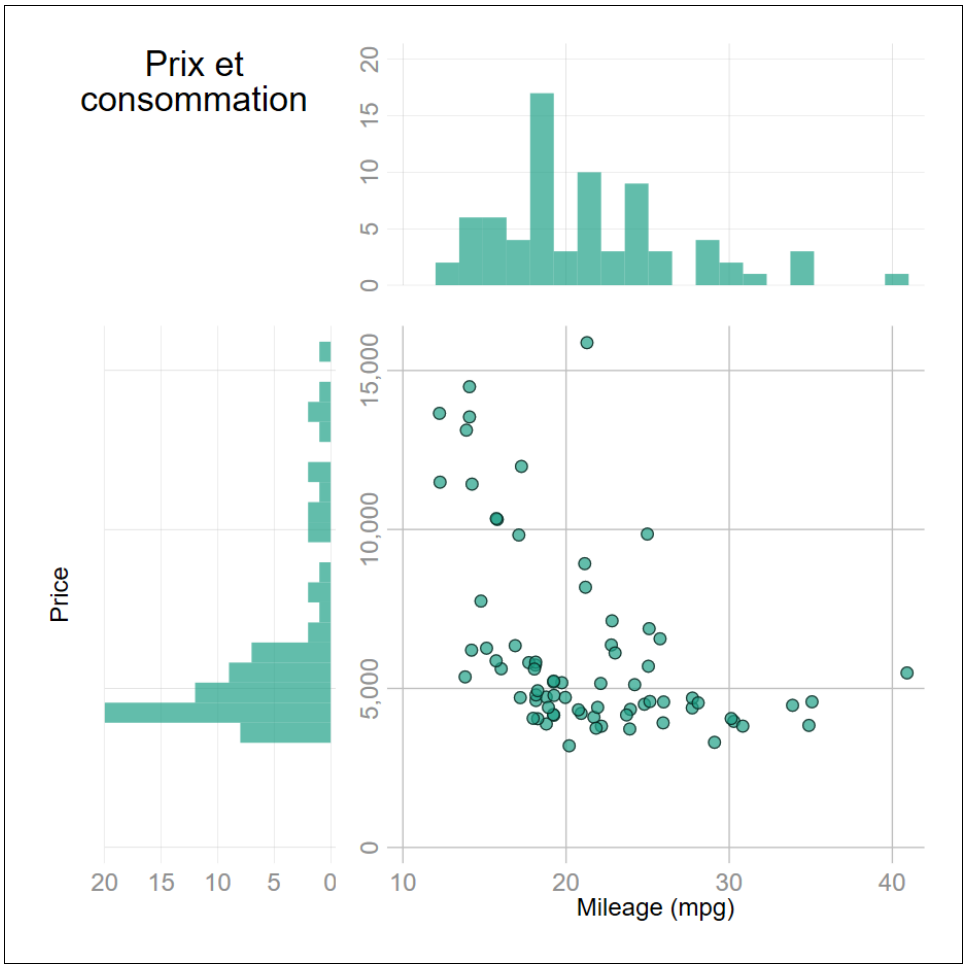


Le positionnement du titre, à droite, n'apparait pas optimal. En inversant le sens de l'axe x pour l'histogramme de *price*, en cachant le titre de l'axe de *price* pour le nuage, et en combinant différemment les graphiques, on obtient un meilleur résultat.

Inversion de l'axe x pour l'histogramme de price : **`xscale(reverse)`**

Cache du titre de l'axe price pour le nuage : **`ylabel(, labc(%0))`**

Modification de l'ordre des sous-graphique : **`graph combine gv g2 g1 g3`**



Exercice : représenter le graphique précédent avec une boxplot au lieu d'un histogramme pour les variables price et mpg.

Utilisation des macros

Rappels sur les macros

local – global

Les macros sont des objets qui enregistre une valeur ou une expression. On peut les utiliser une ou plusieurs fois dans un programme, et permettent de générer des blocs de programme en boucle.

Leur utilisation est essentielle pour les graphiques Stata et faciliter leur réutilisation. Elles permettent entre autres :

- d'alléger la syntaxe de l'habillage ;
- de reporter automatiquement des valeurs (moyenne, minimum, maximum…) dans un graphique, ou générer automatiquement une légende ou une série de labels qui sera reporté sur un axe discret ;
- de produire des graphiques complexes, en particulier en générant des macros empilées

Rappel important

Penser à tester/visualiser le contenu de la macro, avec **display** (**di**) ou **macro list** (**mac list**).

mac list renvoie la valeur ou l'expression qui a été réellement enregistré.

di peut être privilégier pour visualiser ce qui sera affiché dans la fenêtre de l'output.

à la fin de la session.

Macro temporaire: local

Les macros de type temporaire (**local**) sont conservées en mémoire seulement le temps de l'exécution.

Cependant en exécutant Stata avec Notebook Jupyter *(*librairie ***Stata Kernel*** – Kyle barron), ce type de macro vont rester en mémoire d'une exécution à une autre, ce qui peut présenter des avantages, en particulier pour les graphiques. Les macros seront effacées à la fin de la session. Voir la section dédiée à Python.

Syntaxe:

```
local nom [=] expression/opération
local nom [=] "expression"
local nom [=] `""expression""'
local nom : fonction_macro
```

Appel de l'expression dans un programme:

```
`nom'
"`nom'"
```

Test/visualisation de l'expression

```
mac list _nom
di `nom'
di "`nom'"
```

Exemple 1 : On veut enregistrer le résultat de l'opération 1+1

```
local  x  1 + 1

mac list _x
_x:  1 + 1
di `x'
2
```

`mac list` ne renvoie pas le résultat de l'opération, alors qu'il a été bien généré. On doit utiliser l'opérateur d'affectation =.

```
local x = 1 + 1

mac list _x
_x: 2
di `x'
2
```

```
local x "ABC"

mac list _x
_x: ABC
di "`x'"
ABC
```

Exemple 2 l'expression ABC est enregistré avec les doubles quotes: "ABC"

```
local x `""ABC""'

mac list _x
_x: "ABC"
di "`x'"
ABC"" invalid name
```

Remarque: **di** revoie un message d'erreur alors que la macro est correctement assignée.

Exemple 3 : chaque terme de l'expression est enregistré avec des doubles quotes

```
local x "A" "B" "C"

mac list _x
_x: A" "B" "C
di "`x'"
ABC
```

La solution précédente ne fonctionne pas.

```
local x `""A" "B" "C""'

mac list _x
_x: "A" "B" "C"
di "`x'"
A" "B" "C" " invalid name
```

Remarque: **di** renvoie un message d'erreur alors que la macro est correctement assignée.

Exemple 4 : On alterne valeur numérique et du texte en conservant les doubles quotes.

```
local no 0 "Non" 1 "Oui"

mac list _no
_no:     0 "Non" 1 "Oui"
di "`no'"
0 Non" 1 "Oui"" invalid name
```

Remarque: de nouveau **di** renvoie un message d'erreur alors que la macro est correctement assignée.

Macro en dur: global

Les macro de type global, sont enregistrées en dur, et conservées en mémoire d'une exécution à une autre et d'une session à une autre. Elles sont appelées avec **$**, et avec `mac list` le nom de la macro n'est pas précédé d'un underscore.

Test/visualisation de l'expression

```
mac list nom

di $nom

di "$nom"
```

La commande levelsof

Cette commande particulièrement utile, pour ne pas dire incontournable, récupère les valeurs d'une variable et les enregistre dans une macro temporaire. levelsof précède régulièrement l'utilisation d'une bouble de type **`foreach`**.

Syntaxe : **`levelsof nom_var [if/in], local(nom_macro)`**

Exemple

```
sysuse auto, clear
label list origin
origin:
           0 Domestic
           1 Foreign

levelsof foreign, local(f)
mac list _f
_f: 0 1
```

Attention : si la variable est de type string, l'ordre des expressions enregistrées dans la macro suivra l'ordre alphabétique ce qui ne correspondra pas forcément à ce qui est recherché.

Quelques fonctions associées à une macro

Il ne s'agit ici que de quelques fonctions utiles pour produire des graphiques et faciliter leur automatisation. L'ensemble des fonctions associées à une macro sont disponible dans l'aide (`help macro`).

Syntaxe :

`local/global nom_macro : nom_fonction expression`

word count

Permet de compter le nombre de termes contenus dans une expression enregistrée sous forme de macro. Utile pour compter le nombre de boucles à effectuer avec **`forvalue`**.

Syntaxe :

`Local/global : word count(nom_macro)`

Exemple

```
levelsof rep78, local(r)
1 2 3 4 5
local nombre: word count(`r')
mac list _nombre
_nombre:  5
```

value label

Récupère le nom d'un label affecté aux modalités d'une variable. Le nom de la variable peut-être enregistré en amont sous forme de macro.

Syntaxe:

local/global nom_macro : value label nom_variable/nom_macro

Exemple

```
local varname foreign
local labn : value label `varname'

mac list _labn
_labn : origin
```

label

Récupère l'expression de la modalité associée à une valeur d'un nom de label. Le nom du label peut avoir été enregistré en amont dans une macro (avec par exemple la fonction **value label**).

Syntaxe:

local/global nom_macro : label nom_label valeur

Exemple

```
local lab0 : label origin 0
local lab1 : label origin 0

mac list _lab0
lab0 : Domestic
mac list _lab1
lab1 : Foreign
```

En récupérant en amont le nom du label (origin).

```
local labn: value label foreign
local lab0: label `labn' 0
local lab1: label `labn' 1

mac list _lab0 _lab1
_lab0 : Domestic
_lab1 : Foreign
```

En récupérant en amont les valeurs de la variable avec levelsof, la modalité de chaque valeur est récupérée dans une boucle `foreach`.

```
levelsof foreign, local(v)
local labn: value label foreign

foreach val of local v {
local lab`val': label `labn' `val'
mac list _lab`val'
}

_lab0 : Domestic
_lab1 : Foreign
```

On peut en amont mettre une macro sur le nom de la variable, ce qui réduit le nombre de modification. Il est même possible de mettre dans la macro plusieurs variables, et effectuer l'opération dans une boucle principale.

```
local X foreign

levelsof `X', local(v)
local labn: value label `X'

foreach val of local v {
local lab`val': label `labn' `val'
mac list _lab`val'
}

_lab0 : Domestic
_lab1 : Foreign
```

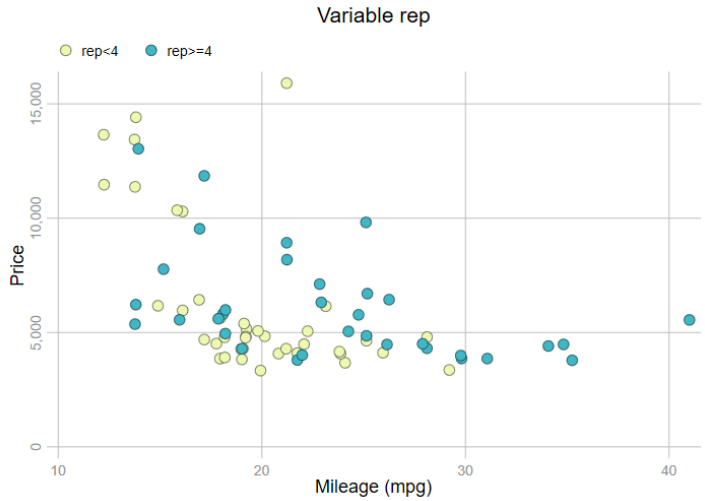
Graphique

On va récupérer automatiquement les modalités d'une variable pour renseigner la légende.

- La variable rep78 est regroupée en 2 modalités.
- Les modalités de la légende sont automatiquement insérées dans l'expression `legend(order(…))`.
- Pour utiliser une autre variable que rep, son nom est enregistré dans une macro.

```
sysuse auto, clear
gen rep= rep78<4
label define rep 0 "rep<4" 1 "rep>=4", modify
label value rep rep

*Récupération des modalités
local X rep
local labn: value label `X'
levelsof `X', local(l)
foreach l2 of local l {
local lab`l2': label `labn' `l2'
}
```

```
*Graphique
#delimit ;
tw scatter price mpg if `X',
   mc("237 248 177")   mlc(black) mlw(*.3)  msiz(*1.5) jitter(2)
|| scatter price mpg if !`X',
   mc("65 182 196")    mlc(black) mlw(*.3) msiz(*1.5) jitter(2)

|| , legend(order(1 "`lab0'" 2 "`lab1'") pos(11) region(color(%0)))
     title("Variable `X'")
;
#delimit cr
```

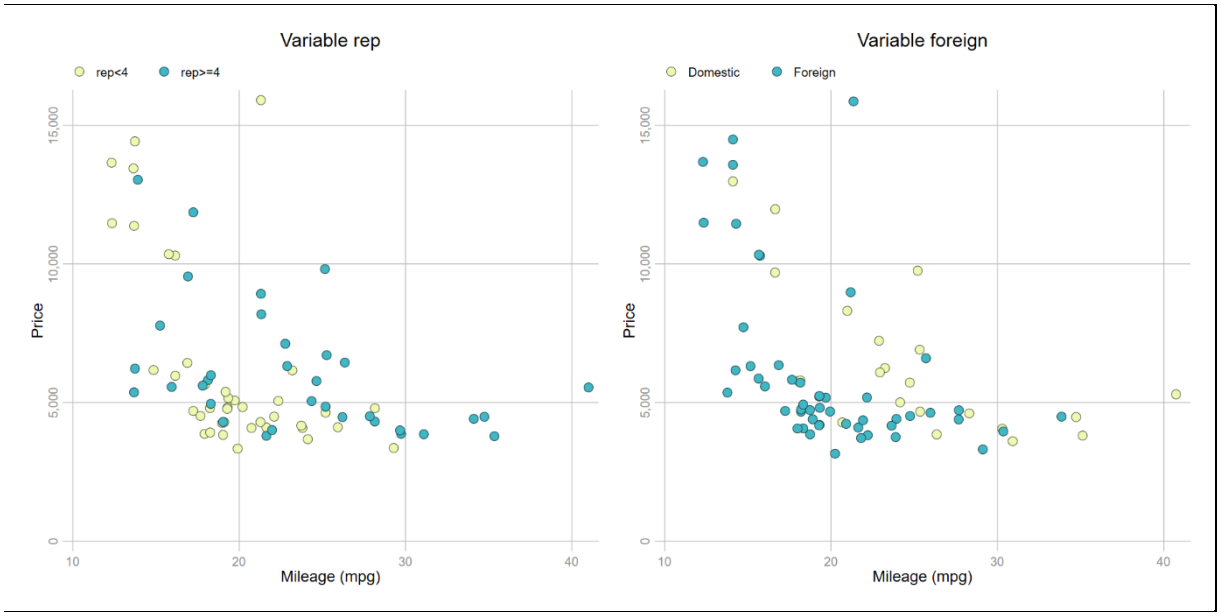


Il suffit de modifier la ligne qui enregistre le nom de la variable pour faire le même graphique avec la variable *foreign*: **`local X foreign`**.

Exercice

A partir du programme faire un graphique combiné des deux graphiques précédents en automatisant au maximum la sélection des variables *rep* et *foreign*. [lien github pour solution]

Il y a très peu d'ajouts ou de modifications à faire.



variable label

Récupère le label d'une variable sous forme du macro. Utile pour gérer une modification du titre des axes passer par les lignes de programme d'une graphique, et plus généralement pour donner de l'information sur variable de type continu ou de comptage sans label affecté au valeurs.

Syntaxe :

```
local/global nom_macro : variable label variable/macro
```

```
local labv:  variable label foreign
mac list _labv

_labv : car type
```

Modification du format d'une valeur numérique (changement du nombre de décimale)

Dans la section suivante, on va utiliser utiliser des objets de type macro qui sont générés après avoir exécuté une commande. Lorsque ces objets sont des valeurs statistiques comme des moyennes, estimateurs de regression etc., leur format enregistré comporte généralement un nombre important de décimales. Il est possible de modifier/réduire ce nombre de décimales avec la fonction **`display`** (**`di`**).

Syntaxe :

```
local/global nom_macro : di format valeur/macro
```

```
local dec: di %6.2f 0.123456789

di `dec'
.12
```

Objets macro return et ereturn

Après l'exécution d'une commande, un certain nombre d'objets sont conservés en mémoire jusqu'à l'exécution de la commande suivante. Leur liste est indiqué en bas du fichier d'aide.

- objets return: **r(nom_objet)**
  - Affichage de la liste: **return list**
- objets ereturn: **e(nom_objet)**
  - Affichage liste: **ereturn list**

```
sum price, 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

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
```

```
qui sum, d

di r(mean)
.54054054
di `r(mean)'
.54054054

local mean r(mean)
di `mean'
.54054054
mac list _mean
.5405405405405406

local mean `r(mean)'
di `mean'
.54054054
mac list _mean
.5405405405405406

local mean : di %6.2f `r(mean)'
di `mean'
0.54
mac list _mean
_mean : 0.54
```

Au niveau d'un graphique ces objets permettent :

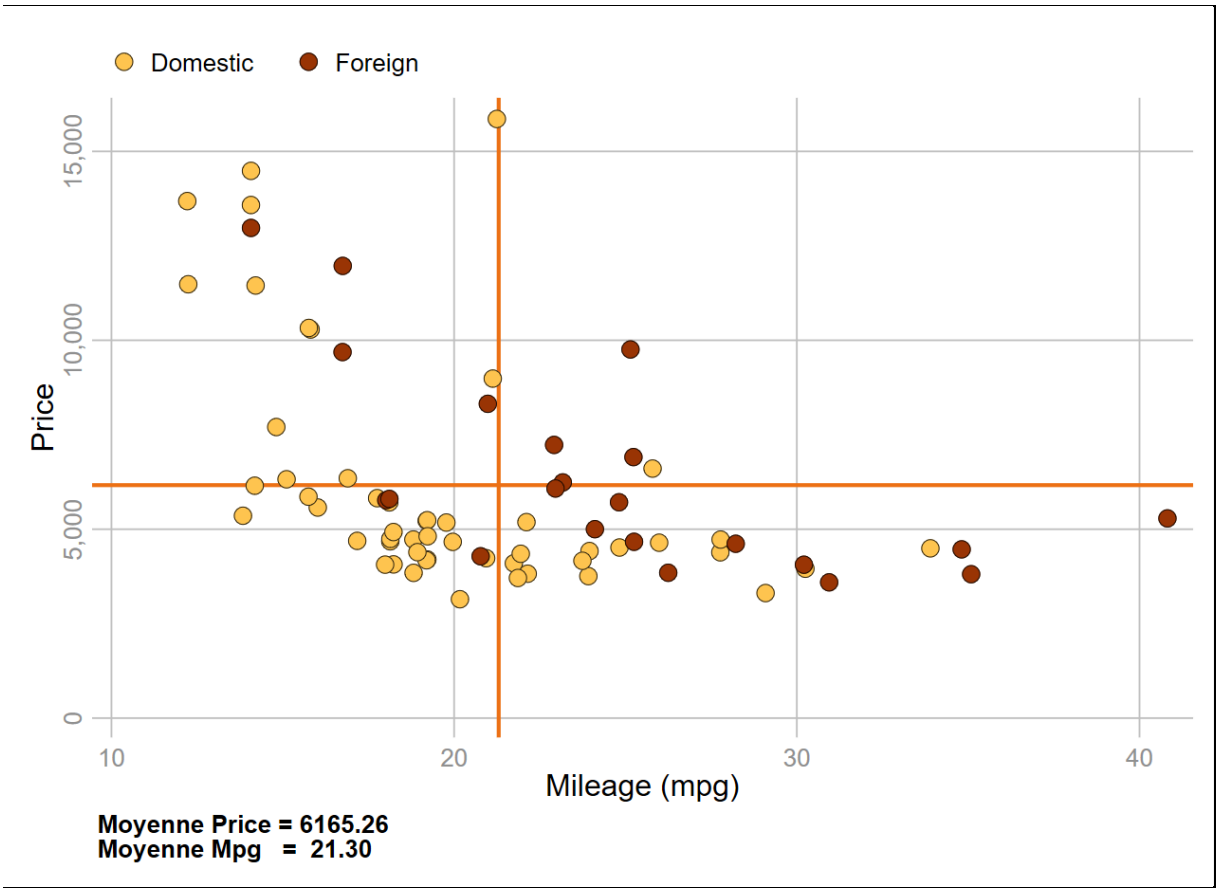
- d'afficher des valeurs dans un graphique;
- de générer automatiquement des éléments de type xline yline;
- de générer des graphiques de type scatteri ou pci qui entrent directement les coordonnées au lieu des variables;

Graphique

On va tracer une droite sur chaque axe qui reporte les moyennes des variables price et mpg, les valeurs de ces moyennes sont reportés en bas du graphique en tant que note

```
* Récupération des moyennes de price et mpg
* les noms des macros seront mprice et mmpg
local varlist price mpg
foreach v of local varlist {
qui sum `v'
local  m`v' : di %6.2f `r(mean)'
}

* Récupération des labels de la variable foreign
local X foreign
local labn: value label `X'
levelsof `X', local(l)
foreach l2 of local l {
local lab`l2': label `labn'  `l2'
```

```
}

* Graphique
#delimit ;

tw scatter price mpg if !foreign,
   mlc(black) mlw(*.3) mc("254 196 79") msiz(*1.5)  jitter(2)
|| scatter price mpg if foreign,
   mlc(black) mlw(*.3) mc("153 52 4")   msiz(*1.5)  jitter(2)

|| , legend(order(1 "`lab0'" 2 "`lab1'") pos(11) region(color(%0)))
     yline(`mprice', lc("236 112 20") lw(*1.5))
     xline(`mmpg'  , lc("236 112 20") lw(*1.5))
     note("{bf:Moyenne Price = `mprice'}"
          "{bf:Moyenne Mpg   = `mmpg'}")
;
#delimit cr
```



Compteurs i++

Les compteurs vont s'avérer très utiles dans les expressions en boucle pour générer une valeur incrémentale de type macro. C'est sur ce principe que fonctionne les boucles de type **forvalue**, mais il est possible de les générer dans une boucle de type **foreach**.

Exemple1 : on initialise un compteur i, et dans une boucle **forvalue** dont l'incrément va de 10 à 15 (delta=+1), on génère à chaque boucle une valeur allant de 1 = 6 avec la macro **`i++'**.

```
local i = 1

forvalue k = 10/15 {
di "k=`k' => i=`i++'"
}

k=10 => i=1
k=11 => i=2
k=12 => i=3
k=13 => i=4
k=14 => i=5
k=15 => i=6
```

Exemple 2 : on va afficher les noms d'une liste de variable en noms génériques dans une boucle **foreach**.

```
local i=1
local varlist price mpg turn length

foreach v of local varlist  {
di "variable`i++' = `v'"
}

variable1 = price
variable2 = mpg
variable3 = turn
variable4 = length
```

Point de vigilance : appels multiples d'un compteur dans une boucle.

Dans les exemples précédents, le compteur n'a été utilisé qu'une fois dans une itération. Il est important de noter que le compteur peut continuer à incrémenter lorsqu'il est appelé plusieurs fois dans une itération. Nous allons présenter les différents comportements du compteur, et le moyen de fixer sa valeur dans une itération avec des appels multiples.

Situation1 : Un seul appel du compteur (cas standard)

```
forv i=1/5 {
  di "iteration `i': i++ ="  `i++'
   }

iteration 1: i++ =1
iteration 2: i++ =2
iteration 3: i++ =3
iteration 4: i++ =4
iteration 5: i++ =5
```

Situation2 : Plusieurs appels dans une itération dans une boucle standard : la valeur du compteur est fixe dans chaque itération, mais pour la première et la dernière, le second appel est ignoré.

```
forv i=1/5 {
di "iteration `i': i++ ="  `i++'
di "iteration `i': i++ ="  `i++'
  }

iteration 1: i++ =1
iteration 2: i++ =2
iteration 2: i++ =2
iteration 3: i++ =3
iteration 3: i++ =3
iteration 4: i++ =4
iteration 4: i++ =4
iteration 5: i++ =5
iteration 5: i++ =5
iteration 6: i++ =6
```

Situation3 : On initialise le compteur en amont (**k**), il continue d'incrémenter dans chaque itération.

```
local k=1
forv i=1/5 {
  di "iteration `i': "  `k++'
  di "iteration `i': "  `k++'
  }

iteration 1: 1
iteration 1: 2
iteration 2: 3
iteration 2: 4
iteration 3: 5
iteration 3: 6
iteration 4: 7
iteration 4: 8
iteration 5: 9
iteration 5: 10
```

Situation4 : le compteur est appelé plusieurs fois. On veut fixer sa valeur pour qu'elle soit égale à la valeur de l'itération, sans rencontrer le problème de la situation2 avec un appel simple à la première et dernière itération. On va devoir initialiser un nouveau compteur à l'intérieur de la boucle, à chaque itération (compteur **j**).

```
local k=1
forv i=1/5 {
 local j = `k++'
 di "iteration `i': "  `j'
 di "iteration `i': "  `j'
  }

iteration 1: 1
iteration 1: 1
iteration 2: 2
iteration 2: 2
iteration 3: 3
iteration 3: 3
iteration 4: 4
iteration 4: 4
iteration 5: 5
iteration 5: 5
```

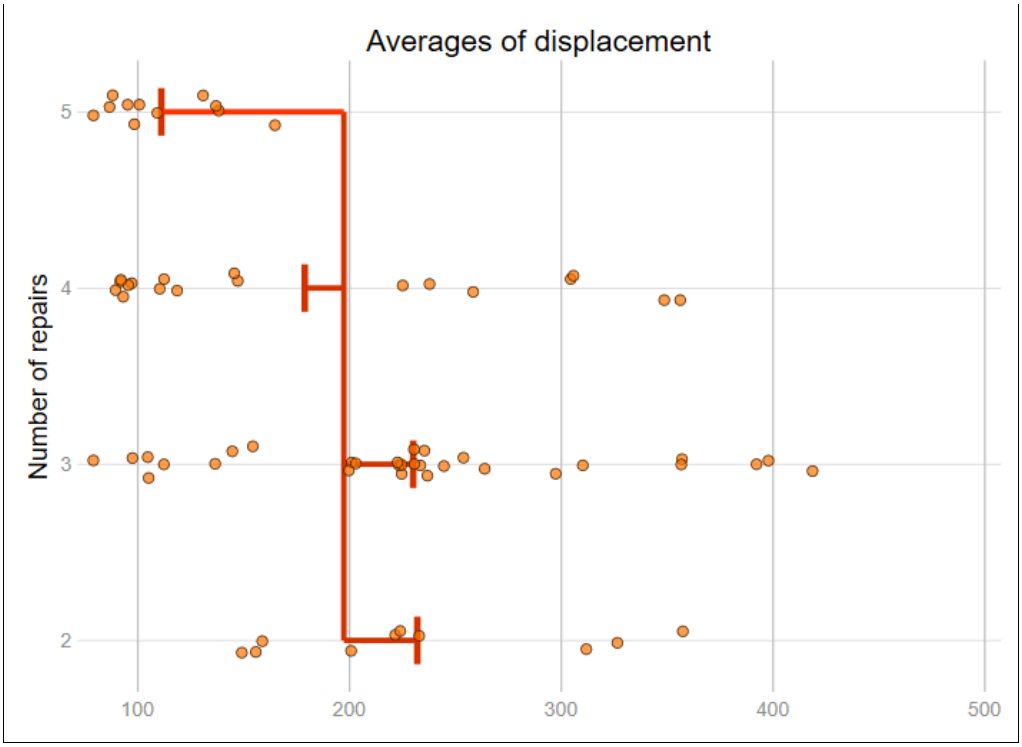
Si le compteur est utilisé plusieurs dans une itération et que sa valeur y doit être fixe, on devra donc utiliser cette dernière expression.

Graphique

Le graphique suivant donne la distribution de la variable *displacement* selon le nombre de réparation (variable rep78). Pour chaque valeur de *rep78*, la moyenne de *displacement* est reportée. Ces moyennes sont connectées à la moyenne calculée sur l'ensemble des voitures. On retrouve le principe d'un graphique de type « lollipop ».

```
local i=1
levelsof rep78, local(l)
foreach v of local l {
qui sum displacement  if rep78==`v'
local m`i++' = `r(mean)'
}

qui sum displacement
local mean `r(mean)'

#delimit ;
tw  scatteri 2 `m1',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)
||  scatteri 3 `m2',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)
||  scatteri 4 `m3',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)
||  scatteri 5 `m4',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)

|| pci 2 `mean' 5 `mean', lw(*2.5) lc("210 50 0")

|| pci 2 `mean' 2 `m1'  , lw(*2.5) lc("210 50 0")
|| pci 3 `mean' 3 `m2'  , lw(*2.5) lc("210 50 0")
```

```
|| pci 4 `mean' 4 `m3'  , lw(*2.5) lc("210 50 0")
|| pci 5 `mean' 5 `m4'  , lw(*2.5) lc("255 50 0")

||  scatter rep78 displacement,  mc("255 117 0%70") mlc(210 50 0) mlw(*.5) msize(*1)
jitter(5)

|| ,
xlabel(,glw(*1.2)) ylabel(,glw(*.5) angle(0)) yscale(range(1.8, 5.2))
legend(off)
title("Averages of displacement") ytitle("Number of repairs")
;
```



Pas à pas du programme

```
recode rep78 (1=2)

local i=1
levelsof rep78, local(l)
foreach v of local l {
qui sum displacement  if rep78==`v'
local m`i++' = `r(mean)'
}

qui sum displacement
local mean `r(mean)'
```

- On regroupe les valeurs les deux premières valeurs de *rep78* (peu d'observations).
- On initialise le compteur qui sera utilisé dans une boucle **`foreach`**.
- On récupère les valeurs de la variable rep78 avec **`levelsof`**.
- Dans la boucle on récupère les valeurs des moyennes de *displacement* pour chaque valeur de *rep78*. Elles sont enregistrées dans les expressions macro **m1** à **m4**. Les valeurs 1 à 4 sont générées par le compteur **i**.

- On récupère la moyenne pour l'ensemble des voitures. Elle est enregistrée dans la macro **mean**.

```
#delimit ;
tw  scatteri 2 `m1',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)
||  scatteri 3 `m2',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)
||  scatteri 4 `m3',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)
||  scatteri 5 `m4',  msymbol(|) mc("210 50 0") msize(*5) mlw(*3)

|| pci 2 `mean' 5 `mean', lw(*2.5) lc("210 50 0")

|| pci 2 `mean' 2 `m1'  , lw(*2.5) lc("210 50 0")
|| pci 3 `mean' 3 `m2'  , lw(*2.5) lc("210 50 0")
|| pci 4 `mean' 4 `m3'  , lw(*2.5) lc("210 50 0")
|| pci 5 `mean' 5 `m4'  , lw(*2.5) lc("255 50 0")

|| scatter rep78 displacement, mc("255 117 0%70") mlc(210 50 0) mlw(*.5)
   msize(*1) jitter(5)

|| , xlabel(,glw(*1.2)) ylabel(,glw(*.5) angle(0)) yscale(range(1.8, 5.2))
     legend(off)
     title("Averages of displacement") ytitle("Number of repairs");

#delimit cr
```

- Pour faciliter l'écriture et la lecture du programme on change de type de délimiteur.
- Les moyennes par niveau de réparation seront en arrière plan pour visualiser les valeurs de toutes les voitures.
- On utilise **`scatteri`** pour générer le nuage des moyennes pour chaque niveau de réparation. Les valeurs de ces dernières sont connues (de 2 à 5) et les moyennes de displacement on été enregistrées de m1 à m4. Pour le premier niveau de réparation le nuage est donc généré par : **`scatteri 2 `m1'`** [**`scatteri valeur_Y valeur_X`**]. On a utilisé un symbole de type *pipe* au lieu d'une bulle [**`msymbol(|)`**].
- On trace une droite verticale avec la fonction **`pci`** pour indiquer la moyenne de *displacement* pour l'ensemble des voitures. La valeur de cette moyenne (X) a été calculée en amont (macro **mean**) et on connait les valeurs de Y. Elles sont égales à 2 et. La droite est donc générée par **`pci 2 `mean' 5 `mean'`** [**`pci min_Y min_X max_Y max_X`**].
- On va connecter la valeur des moyennes pour chaque niveau de réparation avec la moyenne d'ensemble. On va ici tracer des droites horizontale, dont les coordonnées Y seront cette fois ci fixe, et le minimum de X toujours égal à la moyenne d'ensemble. Pour le premier niveau de réparation : **`pci 2 `mean' 2 `m1'`**.

Remarque : on pourrait améliorer le rendu de l'angle pour la connection des moyennes en modifiant légèrement les valeurs de Y_min et Y_max pour la droite verticale, avec par exemple : **`pci 1.985 `mean' 5.015 `mean', lw(*2.5) lc("210 50 0")`** .

Autres objets macro : token, tempvar

Token

Les tokens sont régulièrement utilisés dans la programmation de routines (.ado), mais peuvent dans un programme courant s'avérer utile pour transformer les noms variables en macros. Le token est une macro qui prend comme expression une liste de numéros : **`1', `2', `3'**……
On transforme une expression en token avec la commande **tokenize.**

```
tokenize price mpg
sum `1' `2'
/*
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
       price |         74    6165.257    2949.496       3291      15906
         mpg |         74     21.2973    5.785503         12         41
*/
```

Ou avec une expression sous forme de macro

```
local varlist price mpg
tokenize `varlist'
sum `1' `2'
```

Application pour un graphique :

```
local varlist price mpg foreign
tokenize `varlist'
tw scatter `1' `2' if !`3'
|| scatter `1' `2' if  `3'
```

Variables temporaires

Variables, fichiers… peuvent être créés de manière tempraire. On regardera seulement les variables temporaires générées avec la fonction **tempvar nom_variable**, elles sont appelées sous forme de macro temporaire : **commande `nom_variable'**. Principalement, elles permettent de générer des variables dont le nom n'entre pas en conflit avec des variables existantes, le nom enregistré étant de la forme _00000#, et elles n'alourdissent pas artificiellement le contenu de la base.

```
set obs 100
number of observations (_N) was 0, now 100

tempvar x
gen `x' = runiform()
des
------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
------------------------------------------------------------------------------
__000000        float   %9.0g
```

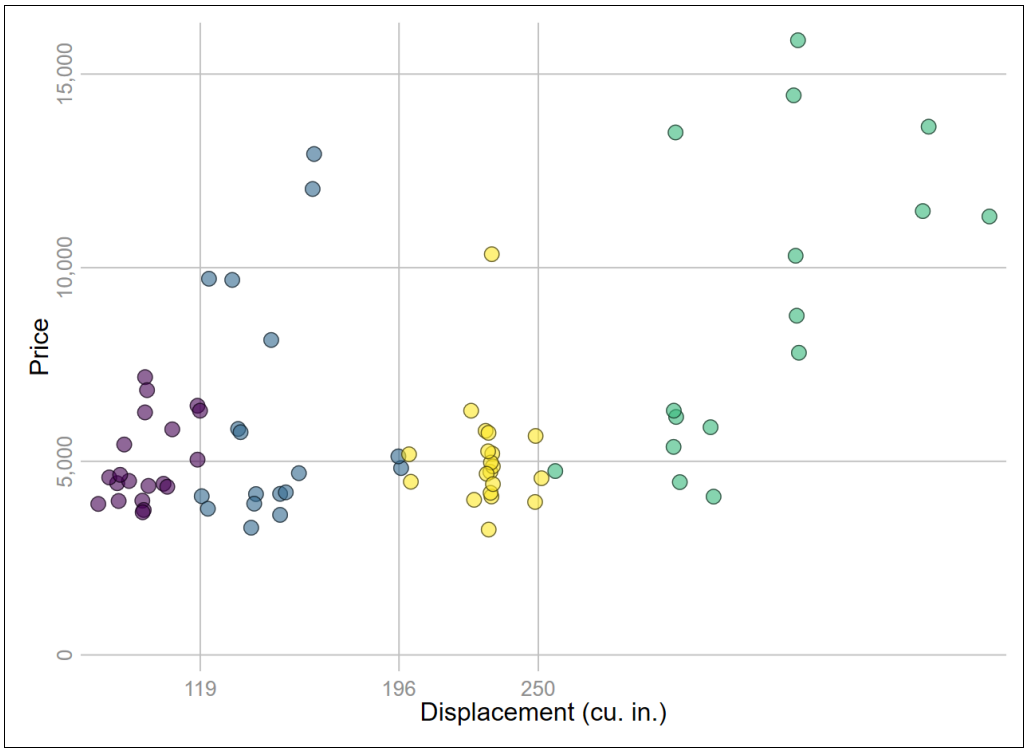
```
-------------------------------------------------------------------------------

sum `x'

    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
    __000000 |        100    .5253147    .2740814   .0073346   .9979447




tempvar x
gen `x' = runiform()
des
-------------------------------------------------------------------------------
              storage   display    value
variable name   type    format     label      variable label
-------------------------------------------------------------------------------
__000000        float   %9.0g
__000001        float   %9.0g
-------------------------------------------------------------------------------

sum `x'
    Variable |        Obs        Mean    Std. Dev.       Min        Max
-------------+---------------------------------------------------------
    __000001 |        100     .471744    .2926945   .0028599   .9711645
```

Graphique

On va de nouveau représenter un nuage de points, entre la variable *price* et la variable *displacement*. La couleur des bulles représente l'appartenance à un quartile de la variable *displacement*.

Programme

```
sysuse auto, clear
tempvar qdisp
local varlist price displacement `qdisp'
tokenize `varlist'
qui xtile `qdisp' = `2', n(4)
qui sum `2', d

#delimit ;
tw scatter `1' `2' if `3'==1,
   mlc(black) mlw(*.5) mc("68 1 84%60")    msiz(*1.5) jitter(1)
|| scatter `1' `2' if `3'==2,
   mlc(black) mlw(*.5) mc("49 104 142%60") msiz(*1.5) jitter(1)
|| scatter `1' `2' if `3'==4,
   mlc(black) mlw(*.5) mc("53 183 121%60") msiz(*1.5) jitter(1)
|| scatter `1' `2' if `3'==3,
   mlc(black) mlw(*.5) mc("253 231 37%60") msiz(*1.5) jitter(1)

||, legend(off)
    xlabel(`r(p25)' "`r(p25)'" `r(p50)' "`r(p50)'" `r(p75)' "`r(p75)'")
;
```



Alléger la syntaxe

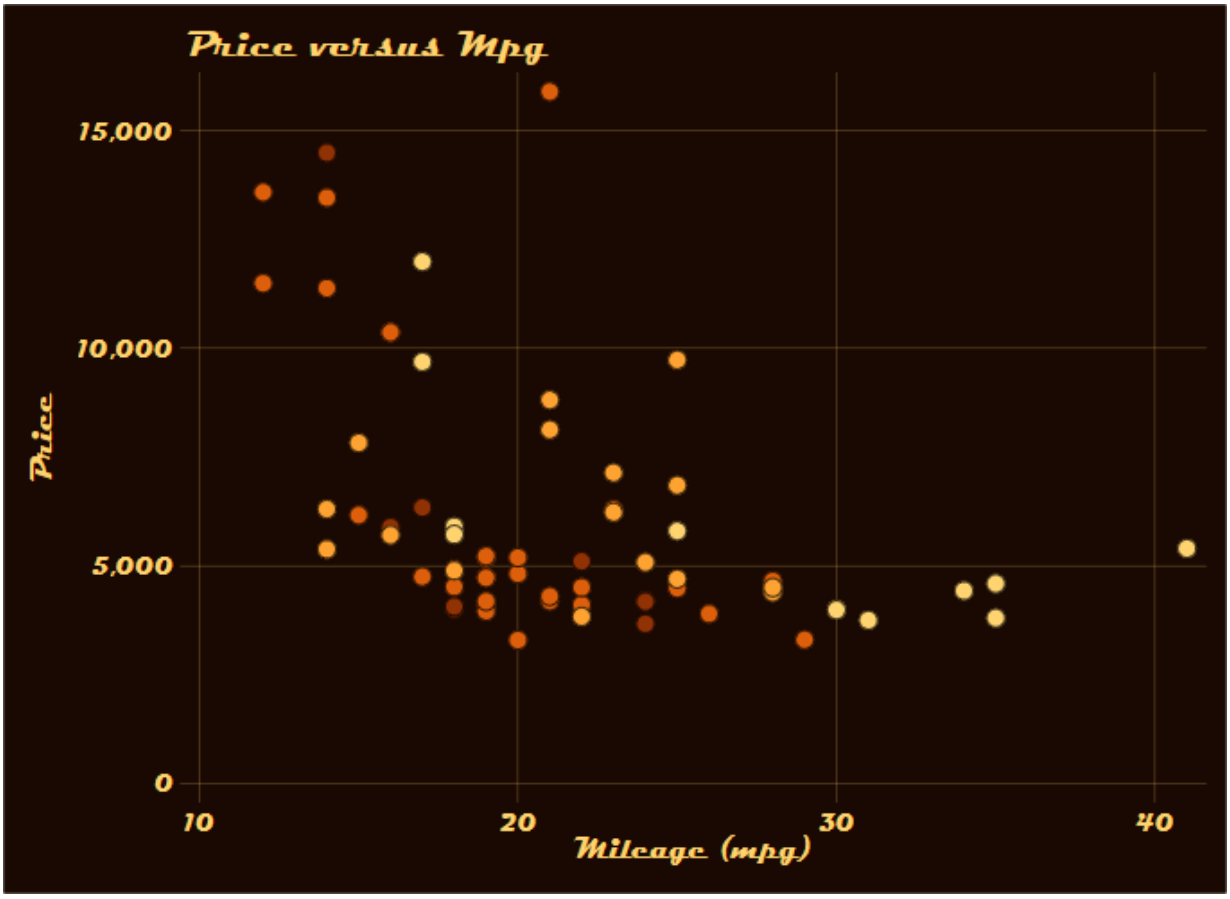
… et surtout faciliter les changements d'options

On le voit, les d'options sont répétées d'un objet graphique à un autre, et souvent en grand nombre. Les modifications de l'habillage du graphique avec les couleurs, les tailles/épaisseurs, peuvent alors s'avérer laborieuse. P

De manière très simple, les options communes à plusieurs objets ou éléments du graphique peuvent être renseignées en amont, sous forme de macros. Cela facilite grandement la modification des options.

Prenons le graphique suivant avec un habillage assez poussé qui modifie les couleurs par défauts des titres et couleurs de fonds. Pour chaque nuage la taille de la bulle, la couleur et l'épaisseur du contour sont identiques, les titres partagent les mêmes couleurs, ainsi que les deux éléments du background. Les options répétées sont repérées en rouge dans le programme (à l'exception de l'épaisseur du quadrillage).

```
graph set window fontface Magneto

tempvar rep
gen    `rep' = rep78
recode `rep' (1=2)

#delimit ;
tw scatter price mpg if `rep'==2, mc("145 50 5")    mlw(*.5) mlc(black)  msiz(*1.5)
|| scatter price mpg if `rep'==3, mc("221 95 11")   mlw(*.5) mlc(black)  msiz(*1.5)
|| scatter price mpg if `rep'==4, mc("254 162 50")  mlw(*.5) mlc(black)  msiz(*1.5)
|| scatter price mpg if `rep'==5, mc("254 211 112") mlw(*.5) mlc(black)  msiz(*1.5)

|| , legend(off)

ylabel(, labc("254 211 112") glc("254 211 112") glw(*.2) angle(0))
xlabel(, labc("254 211 112") glc("254 211 112") glw(*.2))

ytitle(, color("254 211 112"))
xtitle(, color("254 211 112"))
title("Price versus Mpg", color("254 211 112") pos(11))

graphr(color("102 37 6*4"))  plotr(color("102 37 6*4"))
;
```



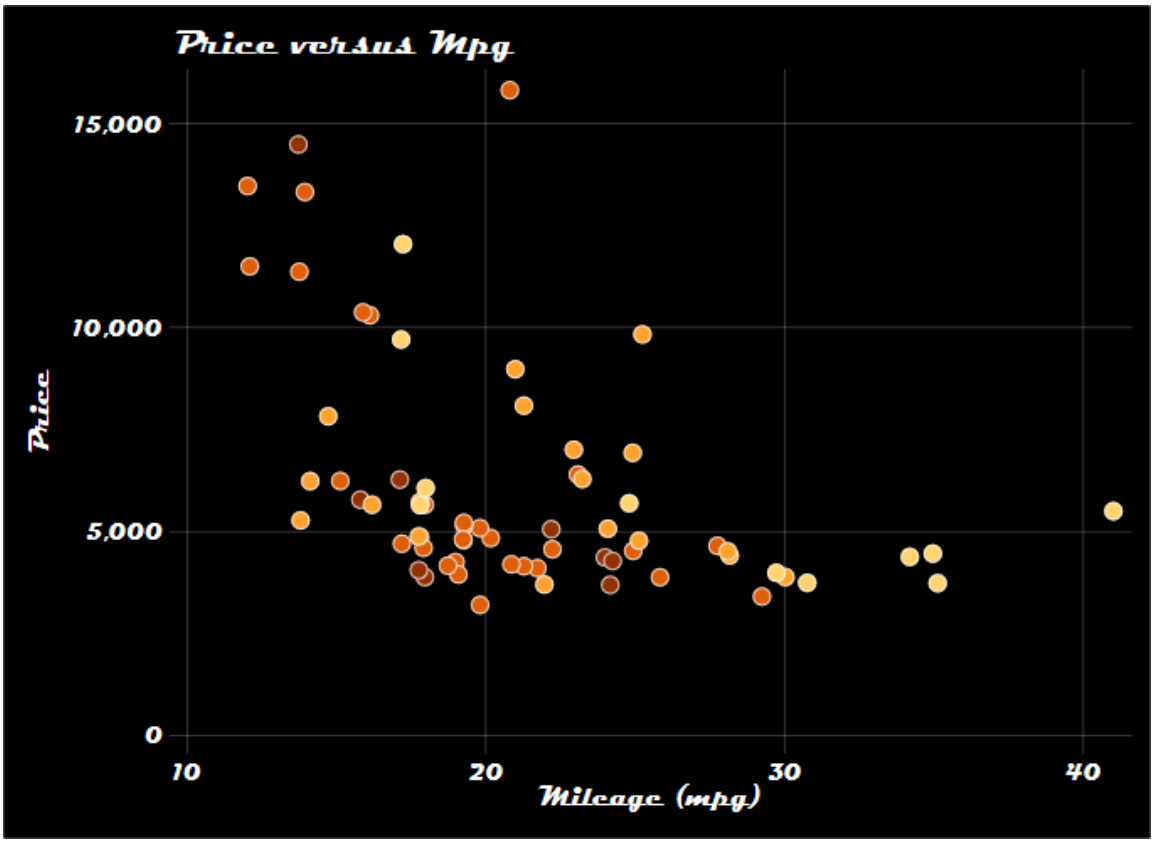
En mettant les options sous forme de macro et en les appelant dans la commande graphique, visuellement la syntaxe du graphique est nettement allégée :

```
local mopt "mlw(*.5) mlc(black) msiz(*1.5)"
local col1 "254 211 112"
local col2 "102 37 6*4"

#delimit ;
tw scatter price mpg if `rep'==2, mc("145 50 5")    `mopt'
|| scatter price mpg if `rep'==3, mc("221 95 11")   `mopt'
|| scatter price mpg if `rep'==4, mc("254 162 50")  `mopt'
|| scatter price mpg if `rep'==5, mc("254 211 112") `mopt'

|| , legend(off)
     ylabel(, labc(`col1') glc(`col1') glw(*.2) angle(0))
     xlabel(, labc(`col1') glc(`col1') glw(*.2))
     ytitle(, color(`col1')) xtitle(, color(`col1'))
     title("Price versus Mpg", color(`col1') pos(11))
     graphr(color(`col2'))     plotr(color(`col2'))
;
```

La modification des options est réalisée une seule fois en modifiant l'expression des macros.

En changeant quelques couleurs (coutours et titres) et en ajoutant un jitter:

```
local mopt "mlw(*.5) mlc(white) m()  msiz(*1.5) jitter(2)"
local col1 white
local col2 black

#delimit ;
tw scatter price mpg if `rep'==2, mc("145 50 5")    `mopt'
|| scatter price mpg if `rep'==3, mc("221 95 11")   `mopt'
|| scatter price mpg if `rep'==4, mc("254 162 50")  `mopt'
|| scatter price mpg if `rep'==5, mc("254 211 112") `mopt'

|| , legend(off)
     ylabel(, labc(`col1') glc(`col1') glw(*.2) angle(0))
     xlabel(, labc(`col1') glc(`col1') glw(*.2))
     ytitle(, color(`col1')) xtitle(, color(`col1'))
     title("Price versus Mpg", color(`col1') pos(11))
     graphr(color(`col2'))  plotr(color(`col2'))
;
```



Automatiser la programmation

Les informations qui vont suivre demande un minimum de pratique avec la manipulation des macros Stata.

Macros empilées

Le principe est plutôt simple, il s'agit lors d'une expression en boucle de conserver et empiler les expressions générées à chaque itération, et les enregistrer dans une macro.

Exemple : On veut conserver la chaine de caractère « A B C ».

Avec une macro standard :

```
local expression Expression_A Expression_B Expression_C

foreach e of local expression {
  local ms =  substr("`e'",12,1)
  mac list _ms
  }
_ms : A
_ms : B
_ms : C
```

```
mac list _ms
_ms: C
```

Dans chaque itération, chaque expression a bien été enregistrée, mais elle est écrasée à l'itération suivante. On pourrait à chaque itération enregistrer chaque macro et les empiler à l'extérieur de la boucle, mais il est possible de le faire automatiquement avec une macro dite « empilée ».

La syntaxe est simple, on appelle la macro dans l'expression qui la génère.

Syntaxe :

```
local nom_macro  `nom_macro'    expression
global nom_macro  $nom_macro    expression
```

Avec une macro empilée :

```
local expression Expression_A Expression_B Expression_C

foreach e of local expression {
local ms =  substr("`e'",12,1)

local me `me' `ms'
mac list _me
  }


*A chaque itération
_me:            A
_me:            A B
_me:            A B C

*Expression finale enregistrée
mac list _me
_me:            A B C
```

Automatiser la création du graphique

Pour un graphique, une application évidente est d'alléger et automatiser d'une syntaxe qui empile plusieurs objets graphiques de même nature.

Si on reprend un type de nuage de points déjà exécuté de nombreuses fois dans ce document :

```
tw scatter price mpg if rep78==1
|| scatter price mpg if rep78==2
|| scatter price mpg if rep78==3
|| scatter price mpg if rep78==4
|| scatter price mpg if rep78==5
```

Avec une macro empilée :

```
levelsof rep78, local(l)
foreach v of local l {
local scat `scat' scatter price mpg if rep78==`v' ||
}
* Exécution du graphique
tw `scat' , legend(off)
```

Ce qui est généré par la macro à chaque itération :

Itération 1 :

```
scatter price mpg if rep78==1 ||
```

Itération 2 :

```
scatter price mpg if rep78==1 || scatter price mpg if rep78==2 ||
```

Itération 3 :

```
scatter price mpg if rep78==1 || scatter price mpg if rep78==2 ||
scatter price mpg if rep78==3 ||
```

Itération 4 :

```
scatter price mpg if rep78==1 || scatter price mpg if rep78==2 ||
scatter price mpg if rep78==3 || scatter price mpg if rep78==4 ||
```

Itération 5 :

```
scatter price mpg if rep78==1 || scatter price mpg if rep78==2 ||
scatter price mpg if rep78==3 || scatter price mpg if rep78==4 ||
scatter price mpg if rep78==5 ||
```

C'est la dernière qui est finalement enregistrée et il ne manque plus qu'à l'appeler après tw.

Au-delà de l'allègement de la syntaxe du graphique cette manière de procéder automatise des modifications faites en amont, par exemple ici un regroupement de modalités de la variable **rep78**.

```
recode rep78 (1=2)

*Il n'est pas nécessaire de modifier le graphique
levelsof rep78, local(l)
foreach v of local l {
local scat `scat' scatter price mpg if rep78==`v' ||
}
tw `scat' , legend(off)
```

Générer une légende

Sur le même principe, on peut générer automatiquement une légende. L'opération peut néanmoins s'avérer un peu plus délicate, en raison du contrôle des doubles quotes pour les labels.

Dans un premier temps on ne va pas utiliser les fonctions macro qui permettent de récupérer automatiquement les noms et le contenu des labels.

Sans les fonctions macro

```
local l1 `""Domestic""'
local l2 `""Foreign""'
forvalue i=1/2 {
local ord `ord' `i' `l`i''
}

macro list _ord
* _ord: 1 "Domestic" 2 "Foreign"

#delimit ;
tw scatter price mpg if !foreign
|| scatter price mpg if  foreign
||, legend(order(`ord'))
;
```

Rappel : *pour la légende on doit explicitement récupérer les doubles quotes pour les expressions qui seront affichées dans la légende :* ***`local nom_macro `""expression""'***

Avec les fonctions macro

Les fonctions sont **`value label`** pour récupérer le nom du label et **`lab`** pour récupérer l'expression affectée à la modalité. Comme les expressions de la légende vont être récupérées via des macros on va accroître le nombre de quotes, ce qui peut s'avérer fastidieux pour lire le programme et éventuellement le modifier.

Dans le programme qui suit on va générer automatiquement la syntaxe principale du graphique avec une macro empilée, comme expliqué précédemment. Les noms des variables seront également transformées en token. Pour anticiper un changement de variable de stratification (ici f*oreign*), on va utiliser un compteur pour automatiser le nombre d'éléments contenu dans la légende.

```
local varlist price weight foreign
tokenize `varlist'

* Légende
local labn: value label `3' // on récupère le nom du label
levelsof `3', local(l)
local i=1
foreach l2 of local l {
local lab`l2': label `labn' `l2' // on récupère l'expressions pour chaque valeur
local lab`l2' `""`lab`l2''""'    // on transforme l'expressions en macro
local ord `ord' `i++' `lab`l2''  // on génère la syntaxe de la légende
}

* Graphique
local ops mlc(black) mlw(vthin)
foreach i of local l {
local scat `scat' scatter `1' `2' if `3'==`i', `ops' ||
}
```

```
tw `scat', legend(order(`ord'))
```

Routine (.ado)

- Manuel d'aide Stata [**entrée syntax**]
  https://www.stata.com/manuals/p.pdf#ppProgramming

- Principe des programmes qui exécutent des commandes (.ado)

- La routine/commande peut être programmée dans un programme principal (.do) ou sauvegardé comme .ado dans le répertoire où sont enregistrées les commandes externes (répertoire **ado**). Il est conseillé d'enregistrer ses propres commandes dans un sous répertoire **personal**, et de créer des sous répertoire par lettre de l'alphabet). Le nom du fichier .ado doit être identique au nom de la commande programmée.

Nous allons décrire seulement quelques principes qui utilisent les éléments vus dans ce chapitre. Il est difficilement envisageable de s'engager dans la programmation d'une commande sans utiliser le manuel d'aide officiel.

Syntaxe générique

```
capture program drop nom_commande
program define nom_routine [options]
syntax [arguments]
* ....programme de la routine...
end
```

Si la commande est enregistrée dans un .ado, la première ligne n'est pas nécessaire.

Exemple 1 : une simple division

Programmation de la routine

```
capture program drop division
program define division

syntax anything

tokenize `anything'
di ""
di as result "Le résultat de ma division est: " `1' / `2'

end
```

Exécution et résultat

```
division 1245 722

Le résultat de ma division est: 1.7243767
```

Application pour un graphique

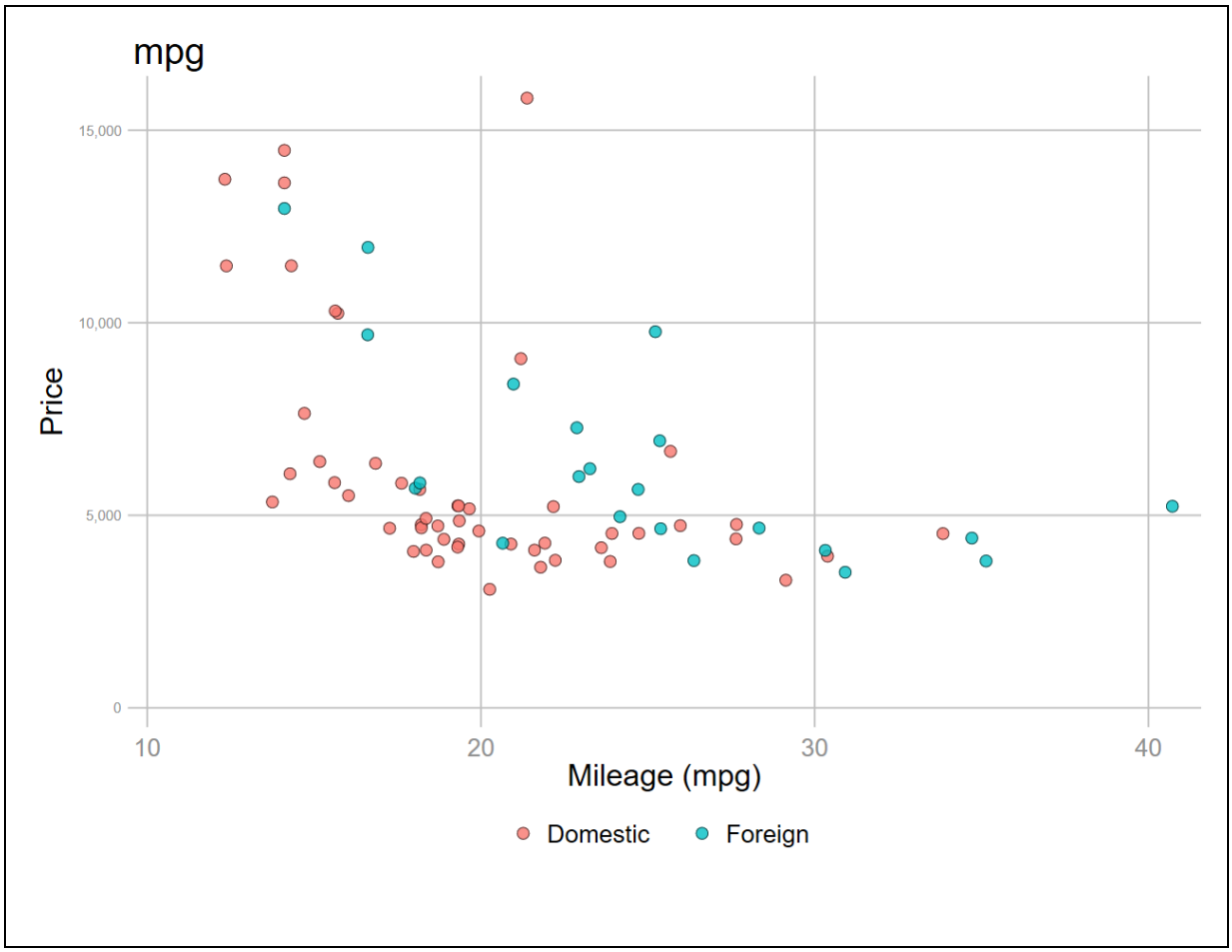
On va automatiser l'exécution du nuage de points avec une stratification, ici la variable *foreign*.

Programme

```
capture program drop sgraph
program define sgraph

syntax varlist(min=3 max=3), [title(string)] [palette(string)] [gops(string)]

tokenize `varlist'

* Légende
qui levelsof `3', local(l)
local i=1
local labn: value label `3'
foreach l2 of local l {
local lab`l2': label `labn' `l2'
local lab`l2' `""`lab`l2''""'
local ord `ord' `i++' `lab`l2''
}

* Syntaxe graphique
foreach l2 of local l{
local ops "mlc(black) mlw(*.3) jitter(3) mc(%80) msiz(*1) mlc(`mlc')"
local scat `scat' scatter `1' `2' if `3'==`l2', `ops' ||
}

* Exécution du graphique
tw `scat', legend(order(`ord') rows(1) pos(6) region(color(%0))) `title' `gops'

end
```

Exécution de la commande **sgraph**

```
sgraph price mpg foreign, gops(ylabel(,labs(*.6)) title("mpg") name(mpg, replace))
```



Programme pas à pas

```
capture program drop sgraph
program define sgraph

syntax varlist(min=3 max=3), [title(string)] [gops(string)]

tokenize `varlist'
```

- La commande s'appelle **sgraph.**
- Pour être exécutée, elle demande 3 arguments de type variable. On fixe un nombre obligatoire de variable égale à 3 : axe Y, axe X et la variable de stratification.
- La commande possède deux options "optionnelles". Comme pour les fichiers d'aide, le caractère optionnel est indiqué par [option].
- Le nom des options est précisé : un titre (**`title()`**) et d'autres options générales d'un graphique (**`gops()`**) qui seront listées .
- On indique le format attendu de l'option, ici une chaîne de caractères.

```
* Légende
qui levelsof `3', local(l)
local i=1
local labn: value label `3'
foreach l2 of local l {
```

```
local lab`l2': label `labn' `l2'
local lab`l2' `""`lab`l2''""'
local ord `ord' `i++' `lab`l2''
}

* Syntaxe graphique
foreach l2 of local l1{
local ops "mlc(black) mlw(*.3) jitter(3) mc(%80) msiz(*1) mlc(`mlc')"
local scat `scat' scatter `1' `2' if `3'==`l2', `ops' ||
}
```

La légende et la syntaxe du graphique sont générés avec une macro empilée

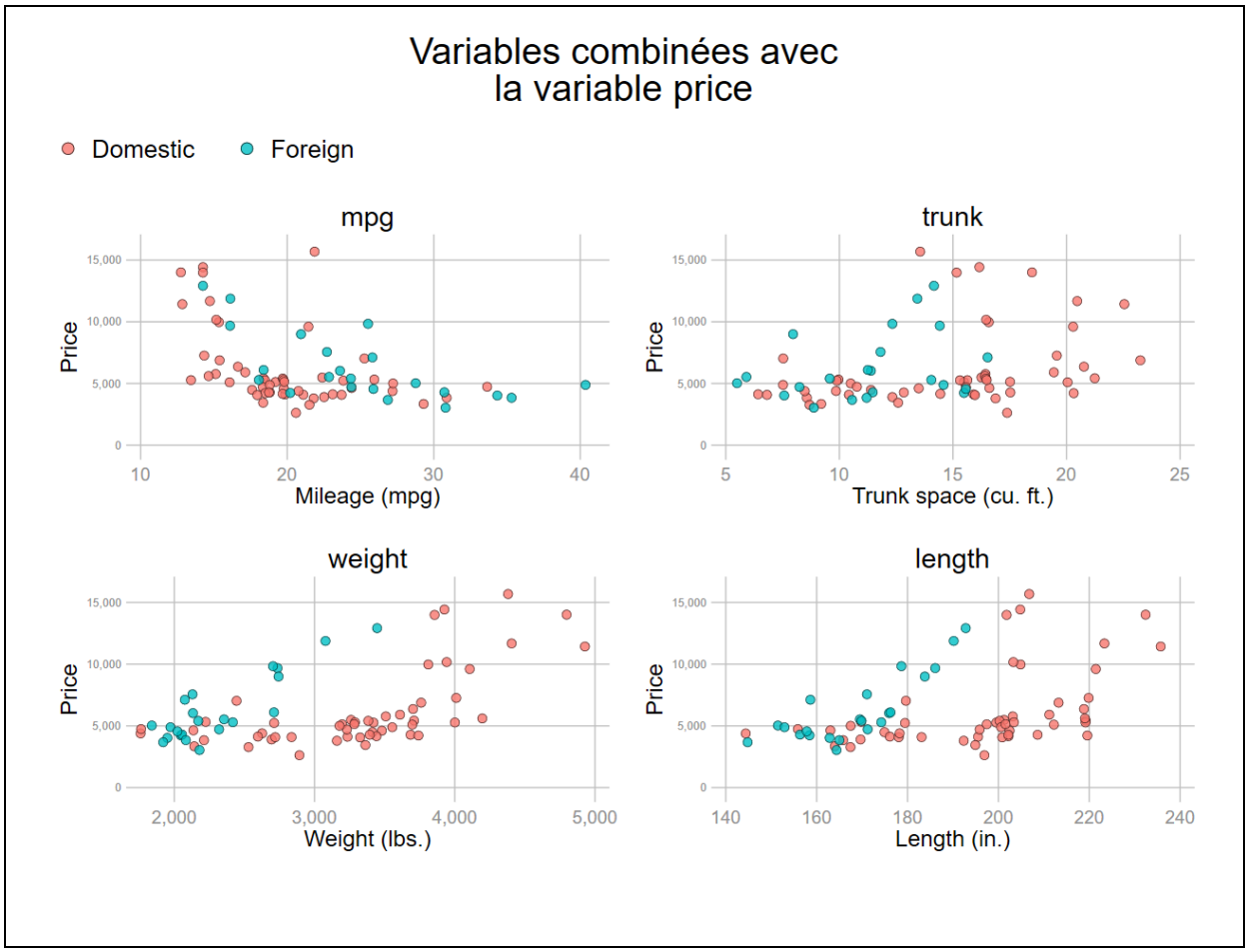
```
* Exécution du graphique
tw `scat', legend(order(`ord') pos(6) region(color(%0))) `title' `gops'

end
```

Le graphique est exécuté, les options qui ont été définies sont entrées sous forme de macro.

Remarque : le programme du graphique indique déjà des éléments optionnels liés à la légende, comme la position et l'absence de contour. Il est possible d'écraser ces arguments « par défaut » dans l'option **gopts**.

Exemple : générer rapidement un graphique qui combine le prix à plusieurs variables.

```
foreach X of varlist mpg trunk weight length {
#delimit ;
sgraph price `X' foreign,
       gops(  title("`X'")
              ylabel(, labs(*.6))
              name(`X', replace))
;

#delimit cr

local g `g' `X' //macro empilée pour lister les graphiques combinés
}

grc1leg `g', legendfrom(mpg) pos(11) title("Variables combinées avec" "la
variable price")
```



Exercice : ajouter une option ou plusieurs options pour modifier l'apparence des bulles.

Palettes de couleurs et styles

Lorsqu'ils sont générés, les graphiques appliquent un thème/style par défaut :

- Ces thèmes sont appelés **scheme**.

- Les options de la commande graphique visent à modifier les paramètres du thème.

- Les paramètres du style/thèmes sont appliqués aux couleurs, tailles/épaisseurs, positions des éléments de texte, définitions des axes….

- Stata dispose de 5 thèmes officiels. Des thèmes externes ont été programmés et peuvent être installés. La série de commandes associées à **`grstyle`** (Ben Jann) permet de paramétrer très facilement un thème, a minima en 3-3 lignes avec un nombre très réduit d'arguments.

- **Collection `schemepack` : récemment (2021), Asjad Naqvi a programmé une suite de thèmes à partir de la commande `grstyle`. Cette série offre une alternative très séduisante au thèmes usines vieillissants de Stata.**

- Les thèmes utilisent des palettes de couleurs. Il existe 3 types de palettes selon la nature des données :
  - **Palettes qualitatives** (*variables discrètes*).
  - **Palettes séquentielles** (variables ordonnées).
  - **Palettes divergentes (***variables ordonnées dans deux sens***).**

- Des commandes externes permettent d'augmenter le nombre de palettes mises à disposition par Stata, de les modifier ou de les créer.

- Pour la cartographie, la commande **`spmap`** (Maurizio Pisatti) intègre une quarantaine de palettes, principalement séquentielles.

- La librairie **`brewscheme`** (W.Buchanan) dispose de plusieurs commandes pour générer des palettes de couleurs.

- La commande **`colorpalette`** (B.Jann) charge plusieurs dizaine de palettes de couleurs, de tout type, permet de les modifier, d'en créer, et de les utiliser facilement dans la création de graphique. Elle est indispensable dès lors qu'on souhaite utiliser efficacement des couleurs différentes de celles fournies par le logiciel ? Cette commande sera longuement traitée dans ce chapitre, ainsi que la commande **grstyle** qui reprend la syntaxe pour générer de manière rapide un thème.

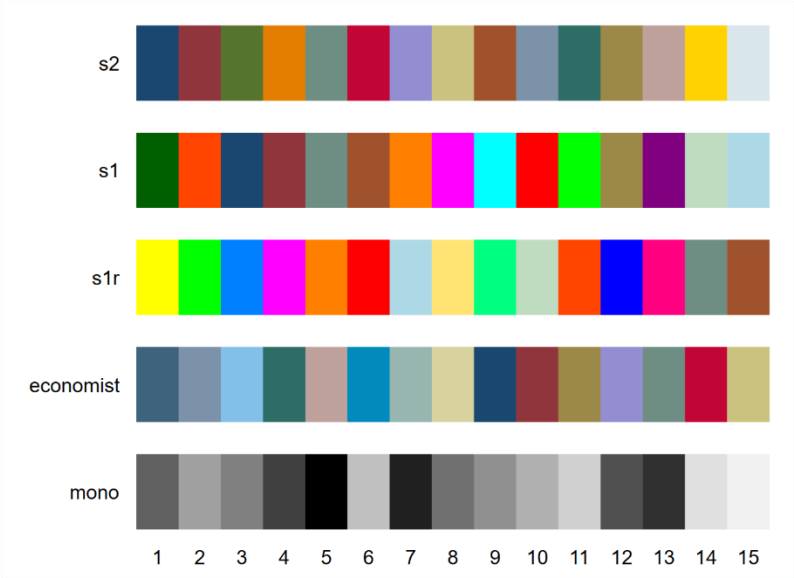
Couleurs et palettes de couleur

Les palettes Stata

Stata dispose de 5 palettes de couleurs usines qui sont associées à des thèmes graphiques. Par exemple la palette **s2** est associée au thème **s2color**.

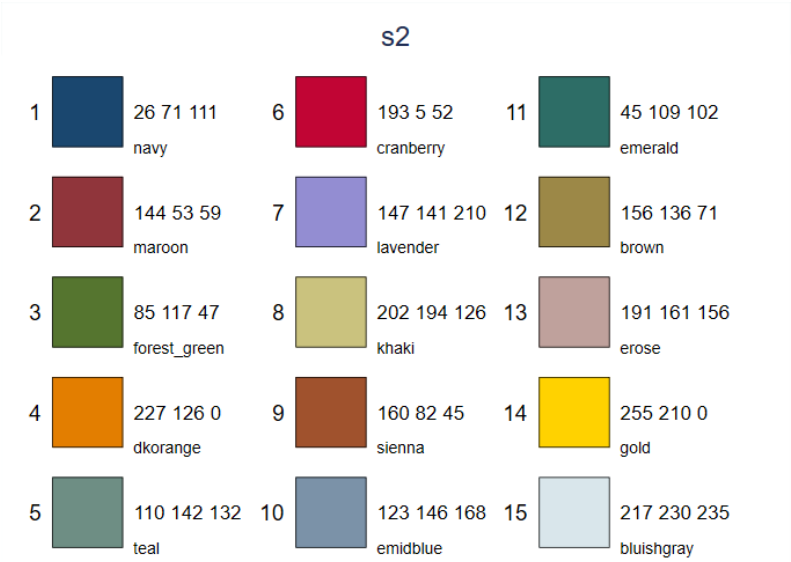
Les palettes associées aux thèmes stata sont de type qualitative. L'ordre d'utilisation des couleurs dans un graphique composé de plusieurs objets est pensé de tel sorte que des couleurs même type, par exemple le bleu, ne se succède pas.

Palettes Stata



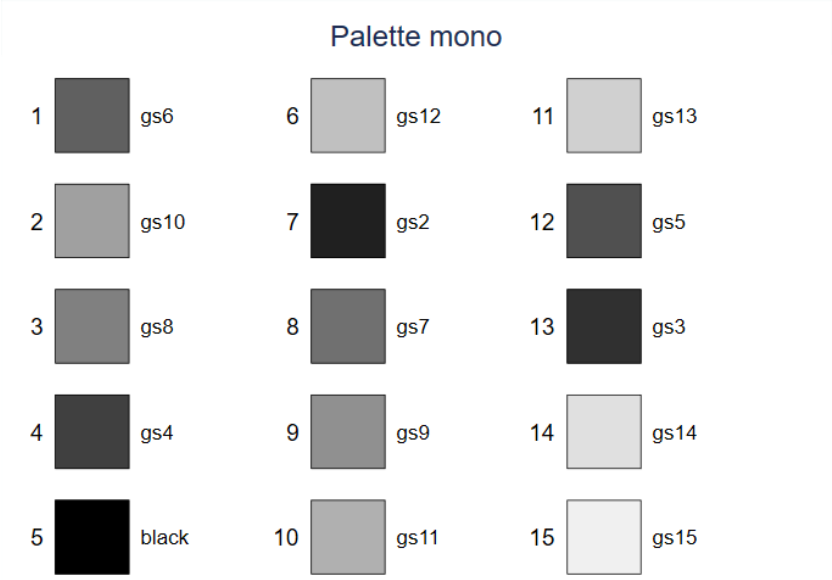
Les numéros sur l'abscisse correspondent à l'ordre par défaut d'utilisation des couleurs.

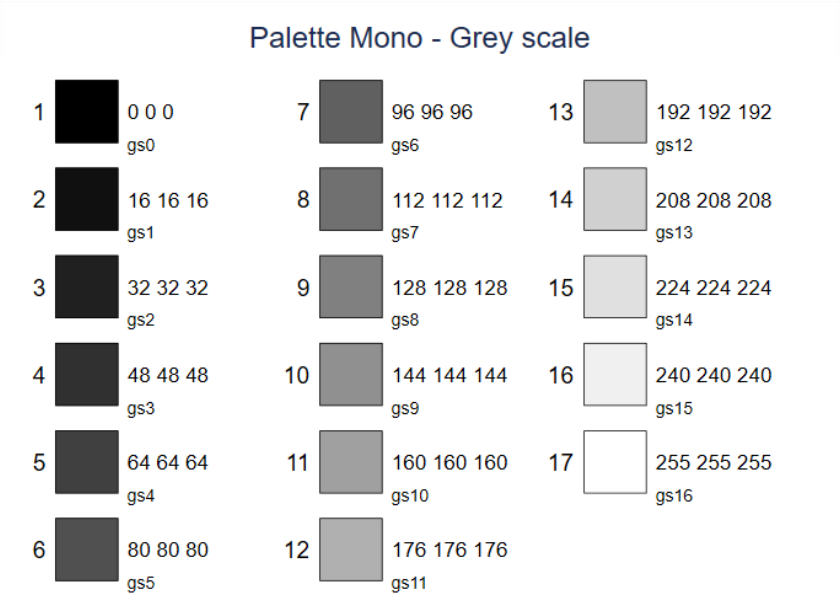
Les couleurs usines ont un nom, auquel correspond un code RGB.



La palette mono

La palette mono d'avère très utile pour modifier les couleurs des titres, labels, grids ou du background. Elle est ambivalente, l'ordre par défaut de la palette est plutôt qualitatif, mais le nom des couleurs correspond à une palette séquentielle nommée « grey scale » (gs# avec # allant de 0 – noir – à 16 – blanc).





Remarque : il est facile de générer une couleur appartenant à une palette correspondant à une échelle de gris : les trois valeurs du code RGB étant identique (compris entre 0 et 255)

On est vite limité avec les palettes usines de Stata. Malgré les efforts de plusieurs utilisateurs pour augmenter leur nombre (M.Pissatti, F.Briatte, D.Bischoff…), la commande **colorpalette** (Ben Jann) a récemment permis de démultiplier quasiment à l'infini les possibilités en terme de manipulation des couleurs.

colorpalette (Ben Jann)

Installation

```
ssc install palettes [, replace force]
ssc install colrspace [,replace force]
```

Syntaxe

```
colorpalette nom_palette/collection [, nom_sous_palette] [options]
```

Remarque : le nom d'une palette peut être une couleur ou une série de couleur

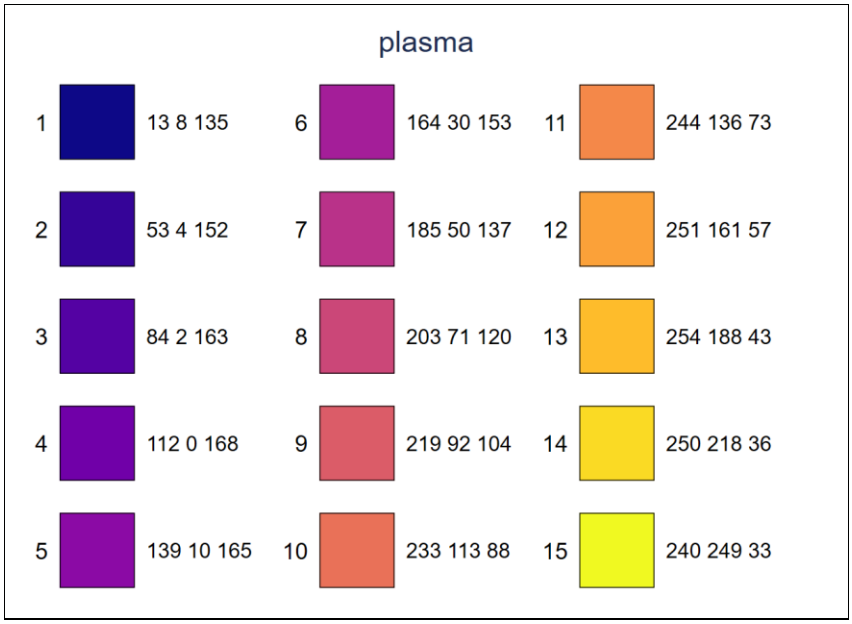
```
* Charge et affiche la palette s1 de Stata
colorpalette s1

* Charge et affiche la palette hue
colorpalette hue

* Charge et affiche la palette viridis
colorpalette viridis

* Charge et affiche la palette summer de la collection matplotlib
colorpalette matplotlib, summer

* Charge et affiche une palette composée des couleur bleu et rouge
colorpalette blue red
```

Pour visualiser les palettes disponibles : **help colorpalette**.

Exemple : la palette plasma

- Appartient à la collection viridis. Palettes Python visant à remplacer les palettes matplotlib
- Longueur par défaut : 15 couleurs
- Type : séquentielle

```
colorpalette plasma
```



Principales options et altération d'une palette

***Modifier la taille de la palette*: n(#)**

Important car la réduction de la taille pour certaines palettes ne consiste pas à prendre les # premiers éléments de la palette par défaut (cf palette **hue** ou **viridis**), mais forcer une palette non qualitative, donc plutôt séquentielle, à produire une palette réduite avec des types de couleurs contrastées plutôt qualitative.

***Modifier l'intensité ou l'opacité* : intensity(#), opacity(#) et ipolate(#)**

- On peut modifier l'intensité (saturation) ou l'opacité d'une palette avec **`intensity(#)`** ou **`intensity(numlist)`** et **`opacity(#)`**.
- On peut créer une palette séquentielle à partir d'une couleur de départ avec **intensity(#1(delta)#2)**.
- On peut créer une palette divergente avec des couleurs de départ, d'arrivée et de transition avec **ipolate(#)**.

Select - reverse - nograph

- On peut sélectionner des couleurs d'une palette avec **`select(numlist)`**. L'option autorise la répétition de couleurs, utile lorsque plusieurs objets graphiques partage une même couleur au sein d'un graphique (exemple nuages de point et courbes pour différentes valeurs d'une variable additionnelle : voir exemple avec grstyle).

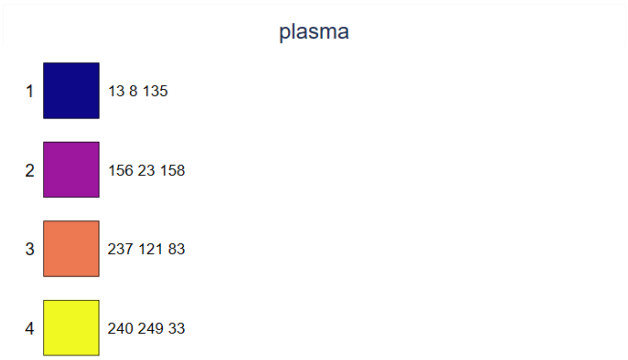
- On peut inverser les couleurs de la palette: **reverse**.

- Ne pas afficher la palette: **nograph** [vor utilisation de colorpalette pour générer un graphique]. Après avoir choisi les couleurs, à utiliser systématiquement pour charger la palette avant son utilisation dans un graphique. Cela réduit considérablement le temps d'exécution.

Exemples (à partir de la palette **plasma**)

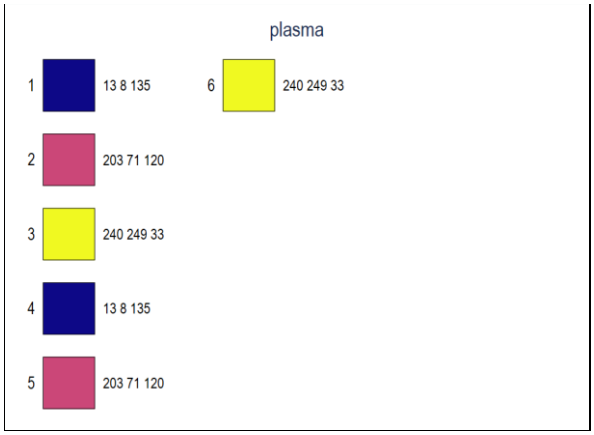
Création d'une palette séquentielle à partir de la couleur

```
colorpalette plasma, n(4)
```



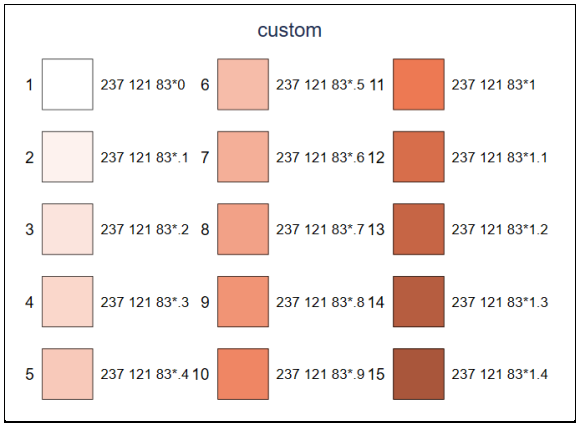
Répétition d'une liste de couleur avec select()

```
colorpalette plasma,  select(1 8 15 1 8 15)
```



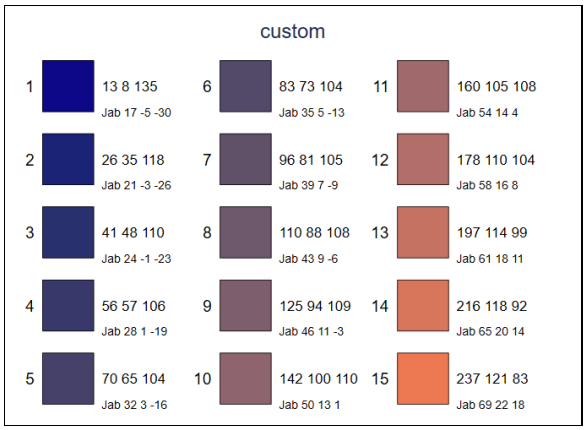
A partir de la couleur 3 (RGB= 237,121,83), création d'une palette sequentielle d'une longueur de 15, allant de "237,121,83*0" (proche blanc) à "237,121,83*15". La couleur de référence sera égale à "237,121,83*9".

```
colorpalette "237 121 83", intensity(0(.1)1.5)
```



Création d'une palette séquentielle d'une longueur 15 allant de la couleur 1 à la couleur 3 de la palette plasma à 4 couleurs issue de l'exemple 1.

```
colorpalette "13 8 135" "237 121 83", ipolate(15)
```

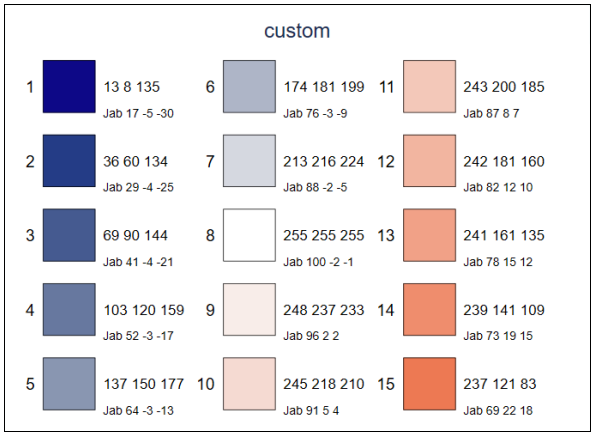


Si on ajoute la couleur 2 de la palette d'origine comme couleur de transition

```
colorpalette "13 8 135" "156 23 158"  "237 121 83", ipolate(15)
```

Si on utilise le blanc comme couleur de transition, on obtient une palette divergente.

```
colorpalette "13 8 135" white "237 121 83", ipolate(15)
```



Créer, enregistrer et charger sa propre palette

`colorpalette` permet de conserver en mémoire sa propre palette, en la générant dans un .ado ou dans le programme principal. Par exemple avec la palette issue du premier exemple (palette plasma à 4 couleur).

```
* Création de la palette mypal
capt program drop colorpalette_mypal
program colorpalette_mypal
c_local P 13 8 135, 156 23 158, 237 121 83, 240 249 33
c_local class qualitative
end

* On charge la palette avec la commande colorpalette_mypal
colorpalette_mypal

* On affiche la palette my_pal
colorpalette my_pal
```

Utilisation de colorpalette pour générer un graphique

On remarque, avec **return list**, que la commande génère une liste de macros qui enregistre les codes RGB des couleurs de la palette.

- la macro **r(p)** permet d'utiliser les couleurs de la palette dans un graphique à un seul bloc d'éléments.
- les macros **r(p#)** permettent d'utiliser les couleurs dans un graphique composé de plusieurs éléments.

La commande **colorpalette** n'est qu'un générateur de couleurs qui renvoie une liste de codes RGB sous forme de macros. Si pour des graphiques simples la technique du « copié/collé » de codes RGB reste possible, l'utilisation des macros va être une nouvelle fois fortement recommandée. On peut déjà néanmoins indiquer qu'avec le générateur de thème **grstyle** (voir la section « style »), également programmé par B.Jann, une sélection de couleurs pourra être directement intégrée à un graphique sans manipulation supplémentaire.

Macros générées par colorpalette

On va partir de la palette plasma précédente avec 4 couleurs et un % d'opacité de 80%. Avec **return list**, on affiche sous forme de macros la liste des codes RGB des couleurs

```
colorpalette plasma, n(4) opacity(80) nograph
return list

/*
scalars:
                  r(n) =  4

macros:
            r(ptype) : "color"
            r(pname) : "plasma"
            r(pnote) : "plasma colormap from matplotlib.org"
            r(psource) :
"https://github.com/matplotlib/matplotlib/blob/master/lib/matplotlib/_cm_liste
d.py"
        r(pclass) : "sequential"
        r(p) : ""13 8 135%80" "156 23 158%80" "237 121 83%80" "240 249 33%80""
                 r(p4) : "240 249 33%80"
                 r(p3) : "237 121 83%80"
                 r(p2) : "156 23 158%80"
                 r(p1) : "13 8 135%80"
*/
```

La liste renvoie la macro **r(p)** qui liste l'ensemble des codes couleurs. Cette macro est utilisée pour les graphiques avec un seul objet qui liste pour l'axe des ordonnées une série de variables. Par exemple :

```
colorpalette plasma, n(4) opacity(80) nograph

tw line le_wmale le_wfemale le_bmale le_bfemale year, lc(`r(p)')
```

Remarque : comme la liste `r(p)' deux doubles quotes, on ne doit pas enfermer cette liste par "".

La liste renvoie le code couleur de chaque élément de la palette, ici **r(p1)** à **r(p4)**. Ces macros sont utilisées pour les graphiques composées de plusieurs objets. Entrée directement dans la syntaxe d'un graphique, ces macros devront être enfermées dans des doubles quotes.

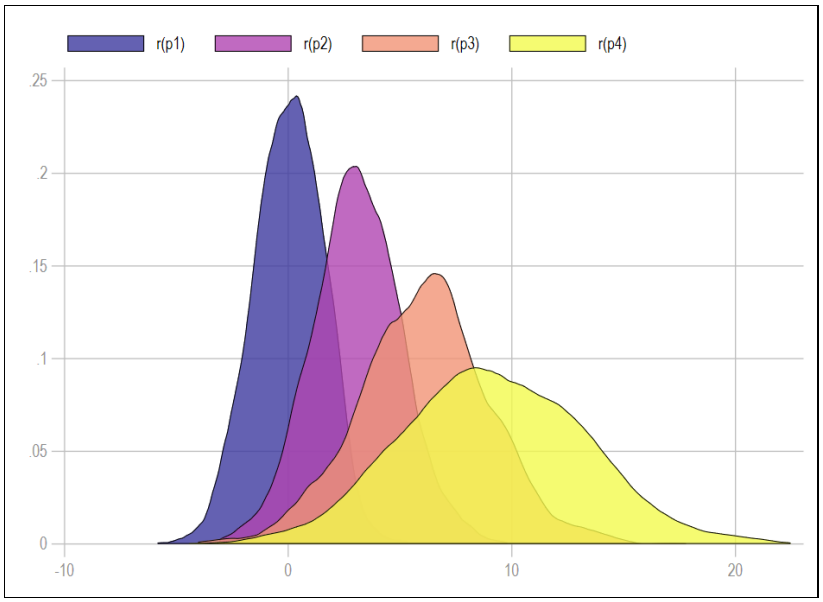
Exemple : On va afficher un graphique avec 4 densités sous forme d'aires générées aléatoirement. De nouveau on utilisera la palette plasma réduite à 4 couleurs et 80% d'opacité.

Données

```
clear
set obs 1000

gen y1= rnormal(0,1.5)
gen y2= rnormal(3, 2)
gen y3= rnormal(6, 3)
gen y4= rnormal(9, 4)

forv i=1/4 {
kdensity y`i', n(500) gen(x`i' d`i') nograph
}
```

Graphique (sans macros empilées pour faciliter la lisibilité)

```
Colorpalette plasma , n(4) opacity(80) nograph

local ops lc(black) lw(*.5)

#delimit ;
tw line d1 x1,  recast(area) `ops' fc("`r(p1)'")
|| line d2 x2,  recast(area) `ops' fc("`r(p2)'")
|| line d3 x3,  recast(area) `ops' fc("`r(p3)'")
|| line d4 x4,  recast(area) `ops' fc("`r(p4)'")

||, legend(order(1 "r(p1)" 2 "r(p2)" 3 "r(p3)" 4 "r(p4)") pos(11) row(1)
region(color(%0)))
;
```

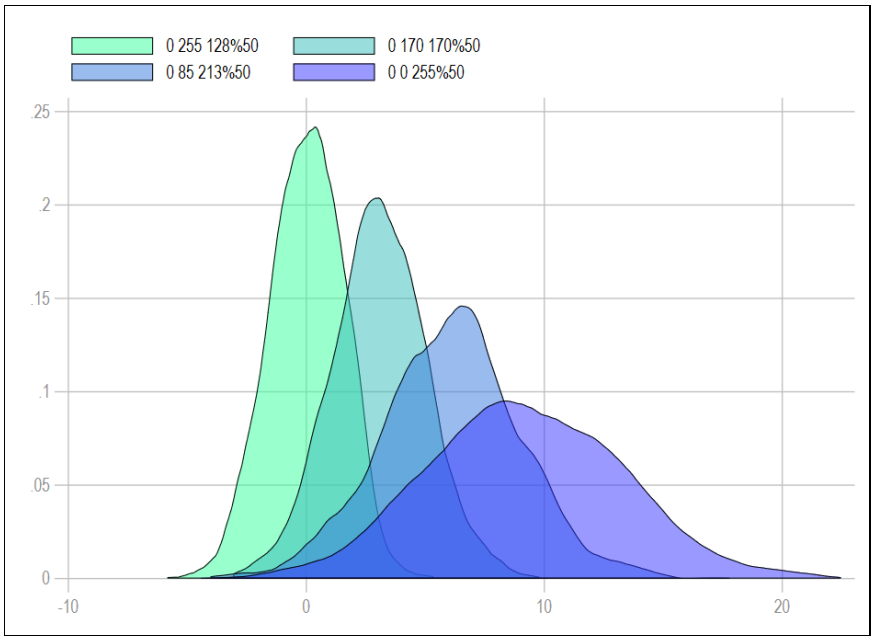


Sans modifier la syntaxe du graphique, on peut alors simplement changer de palette, en modifiant son nom et/ou en modifiant une ou plusieurs options comme l'opacité.

Exemple : palette winter de la collection matplotlib, avec 50% d'opacité et en inversant l'ordre des couleurs.

```
Colorpalette plasma , n(4) opacity(80) nograph

local ops lc(black) lw(*.5)

#delimit ;
tw line d1 x1,  recast(area) `ops' fc("`r(p1)'")
|| line d2 x2,  recast(area) `ops' fc("`r(p2)'")
|| line d3 x3,  recast(area) `ops' fc("`r(p3)'")
|| line d4 x4,  recast(area) `ops' fc("`r(p4)'")

||, legend(order(1 "`r(p1)'" 2 "`r(p2)'" 3 "`r(p3)'" 4 "`r(p4)'") pos(11)
row(1) region(color(%0)))
;
```

Remarque : en utilisant les macros dans la légende, on a affiché les codes couleurs



Quelques exemples de palettes

Sur le même principe que les deux graphiques précédents mais avec 6 éléments, on a mis à disposition une petite commande, **`testpal`**, qui permet de tester le rendu d'un graphique. La commande reporte 6 objets graphiques sous la forme de densités représentées par des aires et par des groupes. Les deux graphiques de la première colonne sont sur un fond blanc, ceux de la deuxième colonne sont sur un fond paramétrable (palette gs) par défaut très sombre.

Installation

```
net install testpal, from("https://mthevenin.github.io/stata_fr/ado/testpal/") replace
```

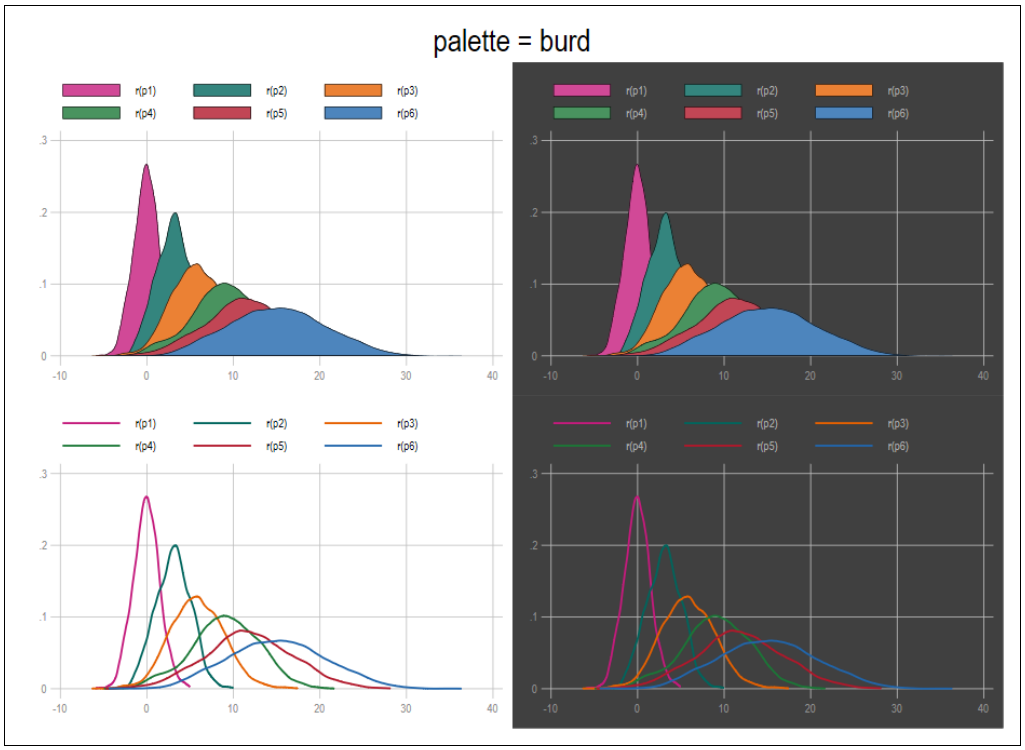
Syntaxe

```
testpal nom_palette , [rev] [op(#)] [bf(#)]
```

- **rev**: inverse l'ordre des couleurs de la palette.
- **op(#)**: modifie le pourcentage d'opacité des couleurs. Par défaut 80% (op(80)). # est compris entre 0+ et 100.
- **bf(#)**: permet de modifier la clarté des graphiques de la seconde colonne (blanc pour la première). # est une valeur comprise entre 1 (noir) et 14 (presque blanc). Par défaut, l'opacité des aires a été fixée à 80%.

Exemple avec la palette qualitative burd (F.Briatte)

```
testpal burd, rev op(100) bf(5)
```

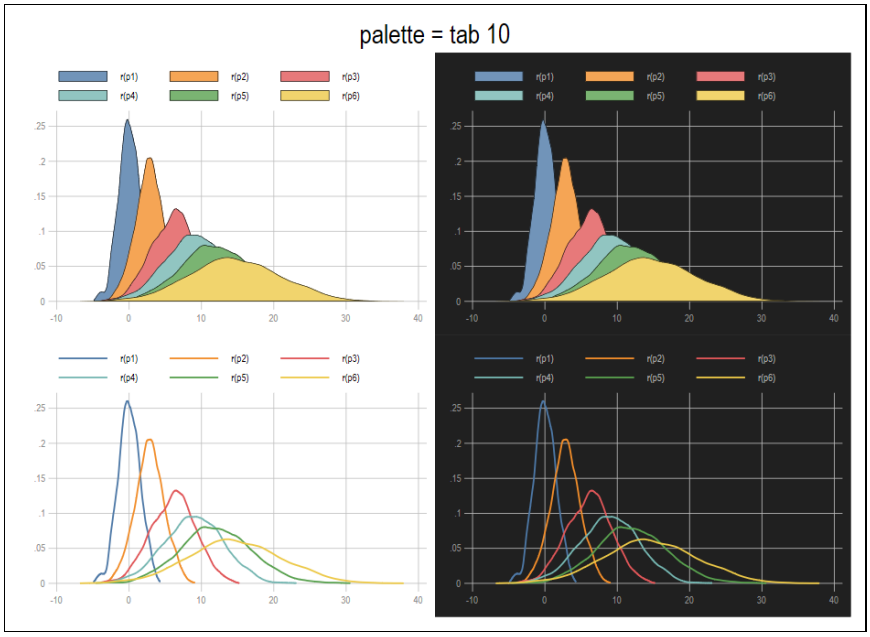


Pour les quelques palettes populaires qui vont être rapidement présentées, les options de la commande **`testpal`** n'ont pas été modifiées. Le nom de la palette est donné par le titre du graphique.

Palettes qualitatives

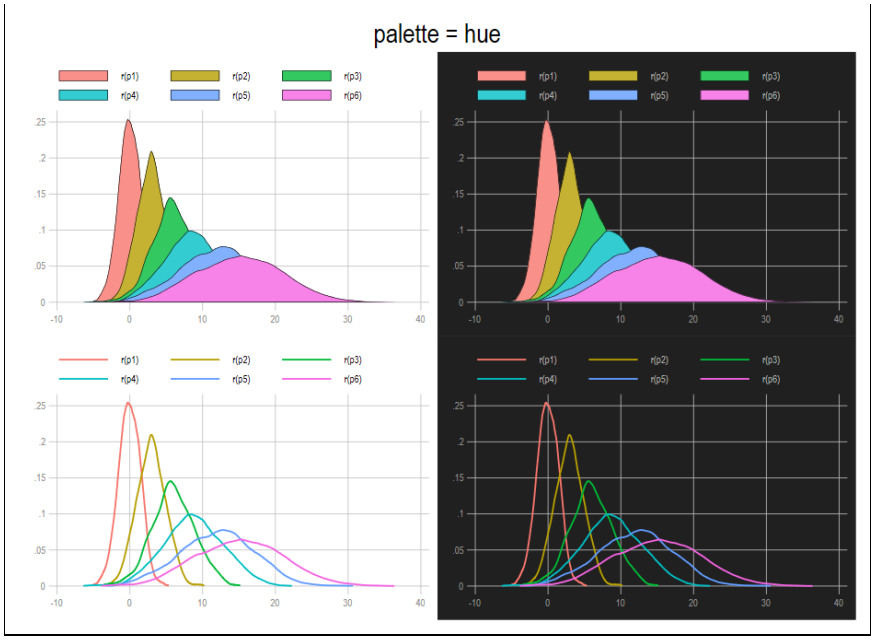
Tableau

https://www.tableau.com/about/blog/2016/7/colors-upgrade-tableau-10-56782



HUE

- Il s'agit de la palette par défaut de **ggplot2** (**R**)
- De plus en plus remplacée par la palette *Viridis* (voir plus bas)



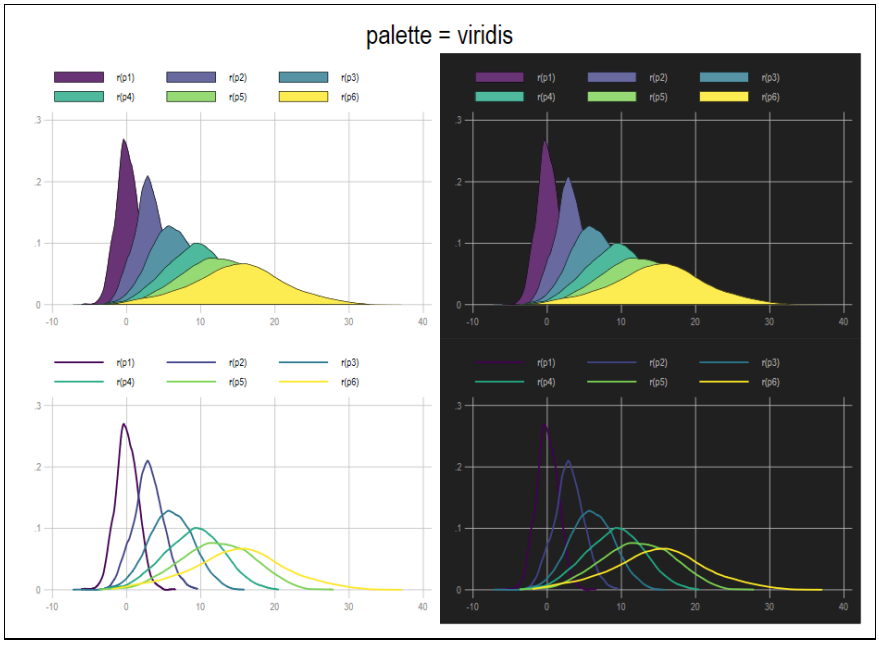
Palettes séquentielles

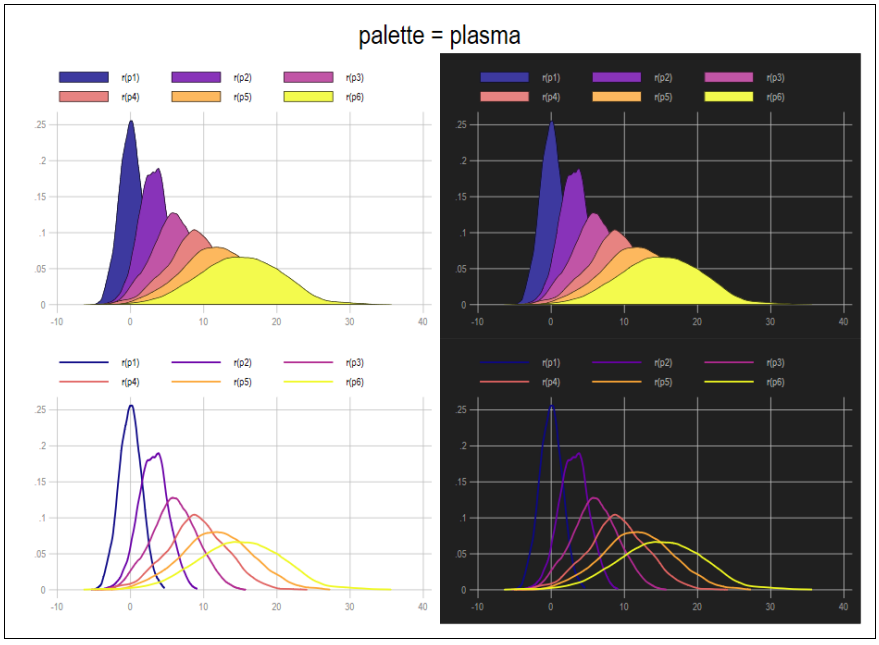
- Utilisées massivement en cartographie et, plus généralement pour représenter des fréquences (graphique type barre) ou des valeurs ordonnées
- Couleurs allant du plus clair au plus foncé (ou inversement)
  - Modification de l'intensité d'une couleur.
  - Une ou plusieurs gammes de couleurs : par exemples du jaune au rouge, du jaune au bleu…

En réduisant la taille de certaines palettes séquentielles ou divergentes, on peut obtenir une palette plutôt qualitative. C'est le cas de la collection, très en vogue, **Viridis**.

Collection viridis

- La palette Viridis et les 4 palettes associées, est la palette *star* du moment
- Développée pour Python pour donner une palette alternative à Matplotlib, elle est devenue la palette par défaut de la librairie graphique
- Avantages:
  - Même rendu, ou très proche, des couleurs sur toutes les parties d'un écran (uniformité).
  - Différences de couleurs maintenue à l'impression n&b.
  - Gère la plupart des formes de daltonisme.
- Limites : pour les courbes, un fond blanc ou très clair ou un fond noir ou très foncé, les couleurs aux extrémités passent difficilement. Prévoir un fond gris moyen ou ne pas sélectionner les couleurs aux extrémités de la palette.
- Les autres palettes : *plasma* est également très utilisée, en particulier pour le remplissage de surface (aire).
- Plus d'infos : https://rtask.thinkr.fr/fr/ggplot2-welcome-viridis/

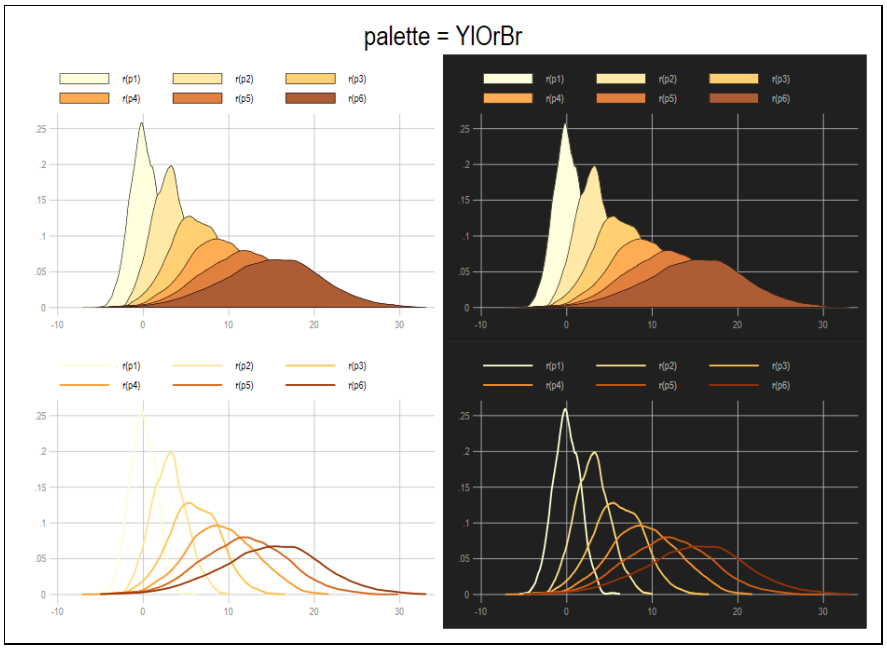




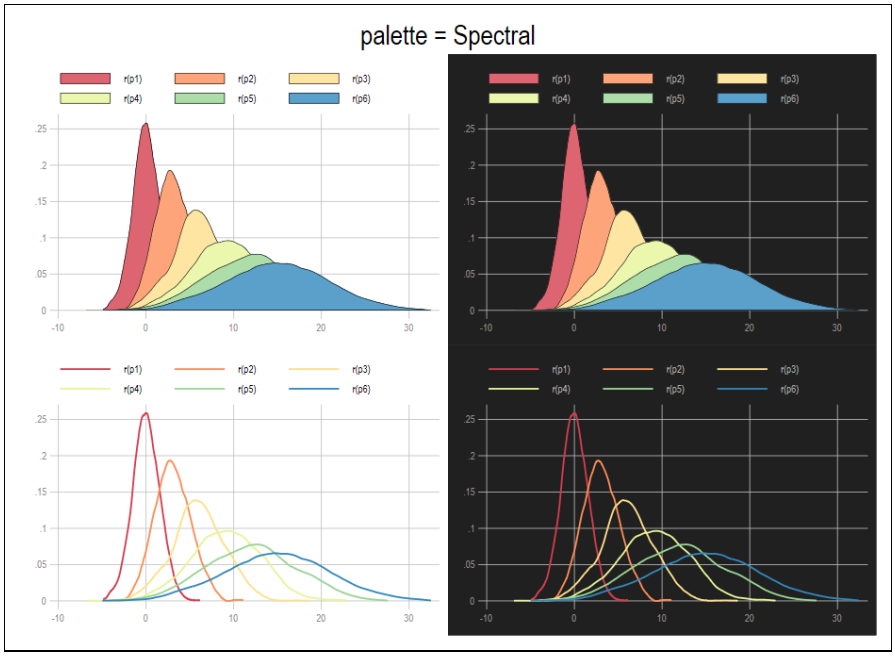
Collection colorbrewer

Très populaire en cartographie, sous certaines conditions palettes en particulier peu d'objets graphiques, certaines palettes peuvent avoir un caractère qualititatif (ex : YlGnBu)

https://colorbrewer2.org/#type=sequential&scheme=BuGn&n=3



On présentera enfin une seule palette de type divergente, issue de cette collection, la palette **spectral**.



Styles

Le thème est la « mise page » du graphique.

- Tous les graphiques ont un thème, appelé **scheme** qui paramètre tous les éléments composant le graphique : couleurs, épaisseurs, positions, contours, marges….
- Les options entrées dans le graphique visent à modifier ce paramétrage.
- Stata dispose de 11 thèmes internes, celui utilisé par défaut est **s2color** (factory scheme)

- Sans modifier des options, un thème peut ne pas être adapté à un graphique.

Changement de thème

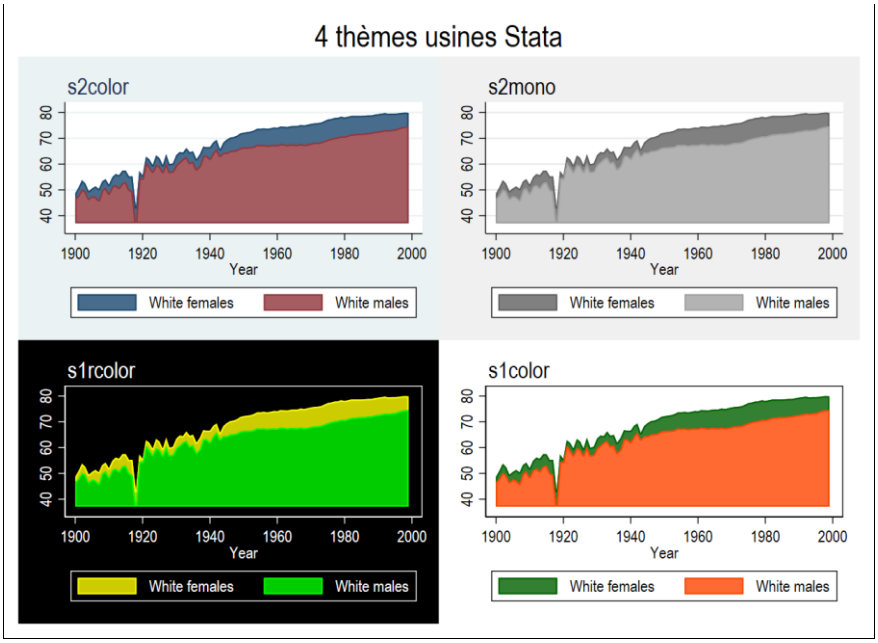
- Par les boîtes de dialogue: « **Edit** » => « **Préférences** » => « **Graph preferences** »
- Ligne de commande
  - On peut changer de thème en option d'un graphique : **`scheme(nom_scheme)`**
  - Dans un fichier .do ou .ado on peut charger un thème de façon temporaire (session) ou permanente: **set scheme nom_scheme [,permanently]**
  - Le choix du thème peut-être également intégré au fichier profile.do (charge le thème au début de chaque session)

Styles internes et externes

Styles internes

Stata fournit 11 thèmes internes dont la liste peut être obtenue avec `help scheme`.

Exemple de mise en page interne avec 4 de ces 11 thèmes) :



Styles externes

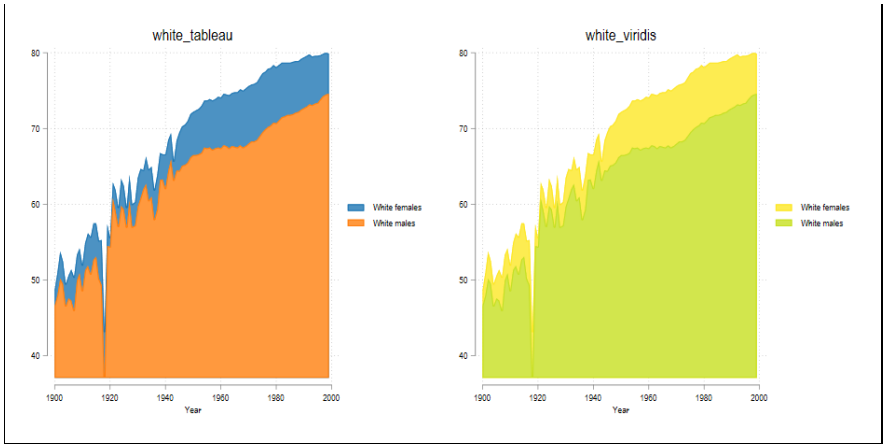
Plusieurs thèmes et collection de thèmes externe peuvent être installées, comme les collections de **F.Briatte** (*Burd*) et de **D.Bischoff** (*plottig*, *plotplainblind*…).
Récemment A.Naqvi a programmé une série de thèmes très intéressante, nommée schemepack, qui proposent pour plusieurs palettes de couleurs 3 variations : white_nompalette, black_nompalette et gg_nompalette.

Installation (version la plus récente) :

```
net install schemepack,  from("https://raw.githubusercontent.com/asjadnaqvi/Stata-schemes/main/schemes/") replace

help schemepack
```

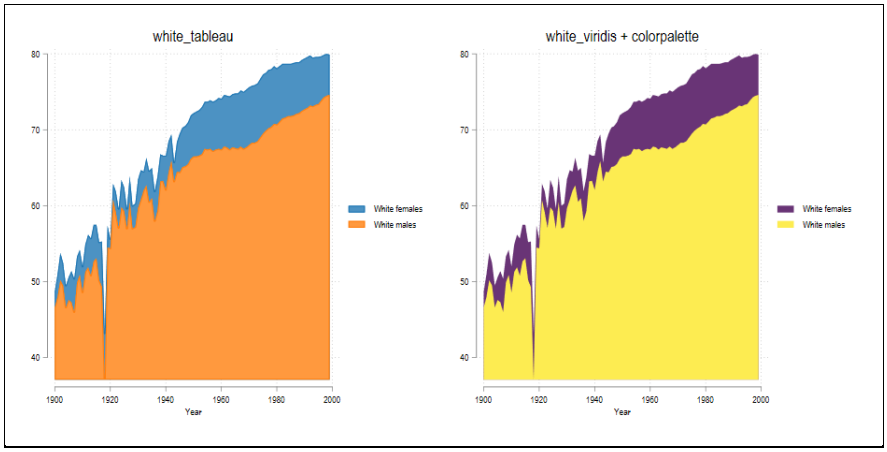
Attention seulement une palette est de type qualitative (tableau), les autres séquentielles. L'utilisation des thèmes associés aux palettes séquentielles peuvent donner un rendu peu satisfaisant lorsque le nombre d'objets graphique réduit (par exemple 2 courbes). Il conviendra alors d'utiliser en amont colorpalette. Egalement, la position par défaut de la légende a été fixée à 3 heures ce qui n'est peut-être pas le plus judicieux.

Exemple avec les thèmes white_tableau et white_viridis



Les deux couleurs se distinguent difficilement avec la palette viridis. On peut corriger rapidement ce problème avec **`colorpalette`**.

```
use uslifeexp, clear
colorpalette viridis, n(2) nograph
tw line le_wfemale le_wmale year, ///
fc(`r(p)') lc(`r(p)')  recast(area) scheme(white_viridis)
```



grstyle de B.Jann

Parallèllement à **colorpalette**, B.Jann a mis à disposition un générateur de thème dont l'utilisation, à minima, me semble particulièrement efficace. Ici nous présenteront que les quelques options que j'utilise quasi exclusivement.

Installation : **ssc install grstyle**

Initialisation et chargement

- Dans un programme on initialise le générateur avec **grstyle init**
- On peut enregistrer un thème avec un nom:

```
grstyle init nom_thème, [path] [replace]
```

- Si le thème a été enregistré, on peut le charger dans un graphique avec scheme(nom_scheme) ou avec la commande **set scheme**. Lors de sa création, le thème est chargé pendand la session.
- Les modifications de l'habillage et de la mise en page du graphique se fait avec une série de commande grstyle set éléments du graphique

Background du graphique, grid, légende

grstyle propose 3 options pour le background et le quadrillage : **plain**, **mesh** et **imesh**. Par exemples avec quelques sous options supplémentaires :

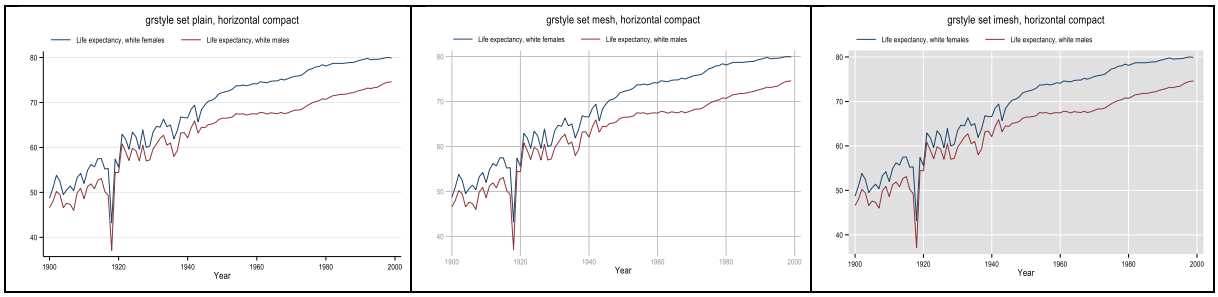
- `grstyle set plain, horizontal compact`
- `grstyle set mesh, horizontal compact`
- `grstyle set imesh, horizontal compact`

Légende (position et retirer le contour)

On peut retirer facilement le contour de la légende (on allège ainsi les options du graphique) avec nobox. En positionnant la légende à 11 heures :

- `grstyle set legend 2 , nobox`

Exemple



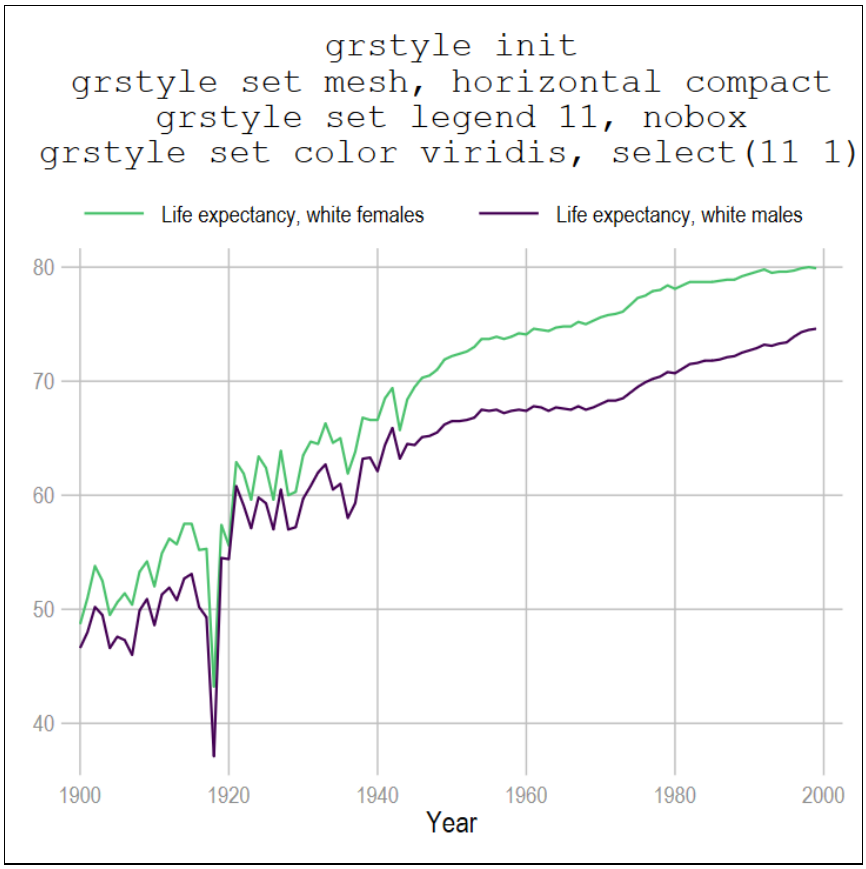
Couleurs

Même syntaxe que **colorpalette**, ici on associe le choix de la palette de couleurs à **grstyle set color**

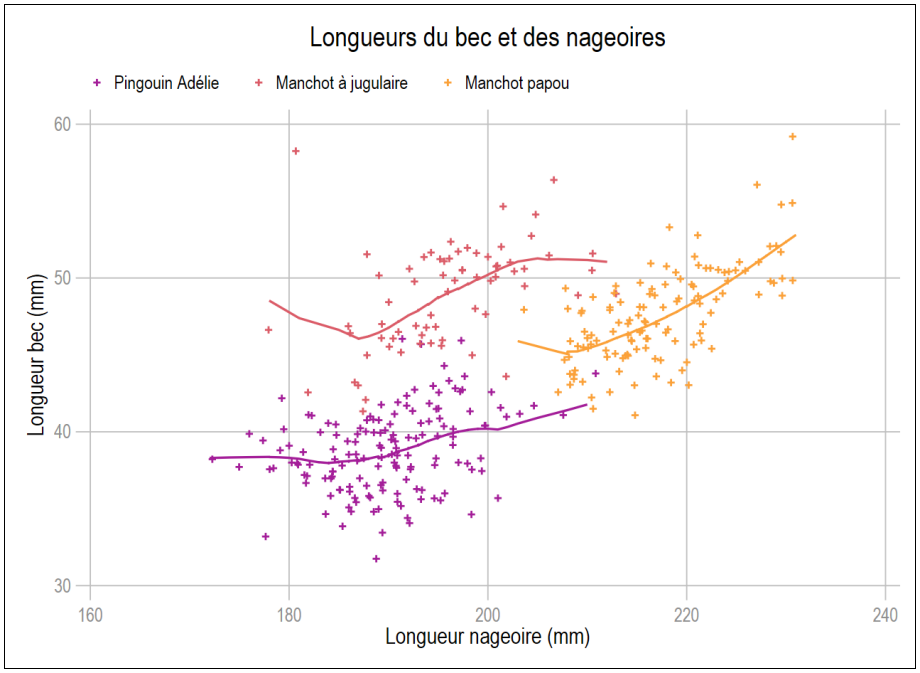
Avec la palette viridis réduite à deux couleurs, par exemple avec select(1 11)

- `grstyle set color viridis, select(1 11)`

Exemple avec le fond de type mesh, la légende à 11 heures sans contour et une sélection de deux couleurs de la palette viridis



Exemple avec répétition de couleurs



```
webuse set https://github.com/mthevenin/stata_graphiques/tree/main/bases
use pingouin, replace
webuse set

grstyle init
grstyle set mesh, horizontal compact
grstyle set legend 11, nobox
grstyle set color plasma, select(6 9 12 6 9 12)

local osl  lw(*1.2)
local osca   msize(*.8) msymbol(+)

#delimit ;
tw sca    bill_length_mm  flipper_length_mm if espèce==1,  jitter(3) `osca'
|| sca    bill_length_mm  flipper_length_mm if espèce==2,  jitter(3) `osca'
|| sca    bill_length_mm  flipper_length_mm if espèce==3,  jitter(3) `osca'
|| lowess bill_length_mm  flipper_length_mm if espèce==1,   `osl'
|| lowess bill_length_mm  flipper_length_mm if espèce==2,   `osl'
|| lowess bill_length_mm  flipper_length_mm   if espèce==3, `osl'

|| , legend(order(1 "Pingouin Adélie" 2 "Manchot à jugulaire" 3 "Manchot papou")
row(1))
     ytitle("Longueur bec (mm)") xtitle("Longueur nageoire (mm)")
     title("Longueurs du bec et des nageoires")
;

#delimit cr
```

Plus qu'un générateur de thème grstyle, avec une utilisation minimale, permet de paramétrer des options en amont du graphique et donc d'alleger sensiblement son programme et sa ses modifications.

Visualisation des données

Histogramme et densité

Histogramme

Densité

Boxplot, violin, courbes de crête (Ridge)

Boxplot

Bean et violin

Densités de Ridge

Application : probabilités assignées

Nuages, densités 2d et bubble plot

Nuages et overplotting

Densités 2d

Bubble plot

Courbes et associés

Line, lowess et connected

Application : effet spagetthi et popularité des prénoms

Barres, lollipop, haltères et coordonnées parallèles

Barres

Lollipop et haltères

Coordonnées parallèles

Application : gender wage gap

Graphiques pour variables catégorielles

Barres (catplot)

Mosaique – Merimekko (spineplot)

Pie (ou pas pie)

Graphiques pour résultats d'une régression

Forest plot

Effets marginaux

Diagnostique

Compléments

Graphiques animés et interactifs

Graphiques animés

Graphiques interactifs

Python (à partir de Stata v16)

Les principales librairies utilisables avec Stata.

Utiliser le notebook Jupyter

Plotnine et seaborn

Bibliographie