Intro - Forskningsassistent

Jakob Jul Elben

22/02/2017

Contents

1 SAS **1**

2 Stata **2**

2.1 Batchfil . . . 2
2.2 Grundlæggende . . . 2
2.3 Databehandling . . . 3
2.3.1 Variable . . . 3
2.3.2 Sortering og gruppering af data . . . 5
2.3.3 Labels . . . 5
2.4 Analyse . . . 6
2.4.1 Dataoverblik . . . 6
2.4.2 Export af tabstat . . . 8
2.4.3 Regressioner . . . 8
2.5 Smart tricks . . . 9
2.6 Figurer . . . 10
2.7 Locals, globals og loops . . . 11
2.8 Postfile . . . 13
2.9 Esttab . . . 14
2.10 Frmtable . . . 15
2.11 Programmering . . . 19
2.12 Dato, tid og formattering . . . 20
2.12.1 Dato og tid . . . 20
2.12.2 Formattering . . . 22
2.13 Merge . . . 23

1 SAS

Min erfaring er, at i de fleste tilfælde kan det bedst betale sig at udføre databehandlingen i SAS, og selve analysen og datamipulationen i Stata. Grundet at 99 % af arbejdet foregår i Stata, så er der ikke vedlagt andet end et cheat sheet til SAS. Dette kan bruges til at merge data i SAS, hvorefte Stattransfer kan benyttes til at omdanne SAS-filen til Stata-format.

Der er lavet et SAS-program til at trække og merge data for personer, som ikke kan finde ud af SAS.

2 Stata

Stata er formentligt den mest udbredte software på ØI.

Denne guide bygger på Benjamin Ly Serena[1] do-files til Stata.

Husk altid at arbejde med DO-files, og aldrig i kommandolinjen.

2.1 Batchfil

Det er vigtigt, at man lader Stata opbevare sine midlertidige filer på et drev med forholdsvis meget plads, når der arbejdes med registerdata. Dette skyldes, at når der f.eks. skal køres regressioner, så Stata pladsen til at opbevare sine midlertidige filer. Hvis dette ikke er tilfældet, så vil Stata stoppe med at køre. Dette løses ved den guide der er vedlagt bagerst i dette dokument.

2.2 Grundlæggende

set more off, cd, clear all

Den sti Stata skal benytte defineres vha. *cd*. Herved vil Stata benytte den sti til at gemme figurerer og tabeller, samt indlæse datasæt, hvis en sti ikke er defineret, når disse gemmes eller indlæses. Stier skal altid være omgivet af *“”* i Stata. *clear all* fjerner alt der gemt i Stata. Vi begynder med *clear all* for at sikre, at der ikke er gamle globals mv. gemt i Stata. Herefter defineres stien og datasættet. Kommandoen *set more off* sikrer, at Stata udskriver altid udskriver hele output umiddelbart. *set autotabgraphs* medfører, at grafer vil blive åbnet i samme vindue som tabs.

```
clear all
set more off, perm
set autotabgraphs off, perm

cd "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent"
```

use

I Stata indlæses datasæt vha. af kommandoen *use* hvis man har specificeret en sti, kan der blot skrives navnet på det datasæt der ønskes loaded i den respektive sti. I dette tilfælde ligger *auto8* i vores hovedsti. Derfor kan vi enten loade datasættet ved blot at skrive navnet på dette, eller specificere hele stien. Begge fremgangsmåder er vist herunder.

```
// Dataset from main directory
use auto8.dta

// Dataset by defining directory
use "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\auto8.dta"
```

browse, if

[1] Tidligere forskningsassistent, og nuværende på ph.d. på ØI. Benjamin.Ly.Serena@econ.ku.dk

For at vise datasættet, så benyttes kommandoen *browse* i Stata. Det er også muligt, at betinge på hvilke observationer der skal vises. Dette gøres ved *if*, og det er ligeledes muligt at vælge bestemte variable der skal vises.

```
// Browse is used to view the dataset
br
// Browse if the car length exceed 180 inches.
br if length>180
// Browse price and miles per gallon
br price mpg
```

Af koden fremgår det, at jeg benytter kommandoen *br* i stedet for *browse*. Dette skyldes, at Stata tillader at forkorte alle variable og kommandoer mv. til det kortest mulige, hvor Stata stadig entydigt kan bestemme objektet. Det vil sige, at vi ikke ville kunne skrive *br*, hvis vi havde en variabel *bro*, da Stata ikke kan skelne mellem om vi henviser til variablen eller kommandoen.

Den logiske syntaks i Stata følger nedenstående tabel. Den logiske syntaks benyttes, når der skal skrives betingelser.

Betydning	Stata
A lig B	A == B
A ikke lig B	A != B
A større end B	A > B
A mindre end B	A > B
A større end eller lig B	A >= B
A større end eller lig B	A >= B
A større end B og B større end C	A > B & B > C
A større end B eller B større end C	A > B | B > C

help

Endeligt, så er det muligt at tilgå hjælp til *alle* kommandoer og funktioner i Stata ved at skrive *help*-kommandoen f.eks.

```
// help with the use command
help use
```

2.3 Databehandling

Stata har sin styrke i sin nemme syntaks, og effektive databehandlingen. Der findes en del andre programmer, der er væsentligt bedre til at fremstille analysedatasæt. Altså det datasæt man udfører sin analyse på baggrund. Af software kan f.eks. SAS og R nævnes selvom der selvsagt findes flere.

2.3.1 Variable

generate, drop, drop if, replace, replace if, keep, keep if

En variabel konstrueres i Stata ved kommandoen *generate*. Det er muligt, at benytte de simple matematiske operatorer, samt at benytte disse på andre variable. F.eks. finder vi bilernes vægt på længde i nedenstående stykke kode.

```
// Generate variable, which contains the weight/inch ratio
gen weight_inch = weight / length
```

En dummy variabel er et andet ord for indikator variabel. En dummy kan konstrueres i Stata på nedenstående måde. Hvis betingelse omringet af parenteserne er opfyldt, så vil variable antage værdien 1, hvis betingelsen ikke er opfyldt vil variablen antage værdien 0. Det vil sige, hvis vi ønsker at vide, hvorvidt miles per gallon er større eller mindre end 20 kan dette gøres ved nedenstående stykke kode.

```
// Generate a dummy for whether mpg exceeds 20.
gen mpg20_dummy = (mpg > 20)
```

Hvis vi ønsker at erstatte værdier i en variabel benyttes kommandoen *replace*. I nedenstående eksempel erstatter vi vores dummy's 0-værdier med skalaren 2. Hvis en variabel ønskes slettet kan vi benytte kommandoen *drop*. Herunder fjerne vi vores dummy. Det er naturligvis muligt som i langt de fleste tilfælde i Stata, at betinge på hvilke værdier der skal erstattes eller hvilke værdier skal slettes. For at gøre dette benyttes *if* blot. Dette er vist for *replace*-kommandoen, men ikke for *drop*-kommandoen.

```
// Replaces all values, which equals zero with the value 2
replace mpg20_dummy = 2 if mpg20_dummy == 0
// we drop variable weight_inch
drop weight_inch
```

I visse tilfælde, så er vi ikke interesserede i at droppe variable (observationer), men derimod beholde nogle bestemte variabel (observationer). Til dette findes der en kommando, som kan ses som den inverse til *drop* (*drop if*). Nemlig kommandoen *keep* (*keep if*). Denne kommando har samme fremgangsmåde som *drop* (*drop if*), men istedet for at slette de specificerede variable (observationer) beholder Stata udelukkende de specificerede observationer.

(COND1 & COND2 | COND3)

I nogle situationer kan det være meget smart at opdele betingelser. Forestil dig et scenarie, hvor vi ønsker at finde alle biler, der har en bestemt miles per gallon statistik, men vi ønsker at differentiere intervallet for dem. F.eks. kunne det være, at vi ønsker at markere alle biler der har en salgspris på 4000 og 10,000 dollars og som kører mere end 15 miles per gallon, og så ønsker vi at alle biler der har en salgspris større end 10,000 dollars som kører mere end 20 miles. Dette ville tage flere linjer kode, hvis ikke vi opdelte vores betingelser.

```
//Find specific cars
gen MpgPricedummy = ( (4000 < price & price < 10000 & 15 < mpg) | (10000 < price & 20 < mpg) )
```

egen

Generate tillader kun forholdsvis enkle dannelser af variable, men kommandoen *egen* fungerer i stor grad på samme måde som *gen*, men tillader mange flere funktioner. Så som gennemsnit, max, min osv. For at se alle *egen*s funktioner så skriv *help egen*.

```
// generates max, mean, and min price
egen pricemean = mean(price)
```

```
egen pricemin = min(price)
egen pricemax = max(price)
```

2.3.2 Sortering og gruppering af data

sort, by, bysort

Det er sjældent særligt overskueligt, at arbejde med registerdata, da der er så ufatteligt mange observationer. Det er dog muligt, at gørre det lidt mere overskueligt. Det er muligt at sortere data vha. *sort*-kommandoen. Dette er også vigtigt, når man sikrer sig, at de variable man har dannet rent faktisk gør, hvad de bør gøre. Kommandoen *sort* sorterer data efter opadgående rækkefølge for de valgte variable. F.eks. hvis vi ønsker at sortere bilerne efter om de kommer fra ind- eller udland kan *sort*-kommandoen benyttes.

```
// sorting cars after price and miles per gallon
sort foreign
```

Det er også muligt, at danne variable efter grupperinger. Dette gøres ved, at danne en variabel, der angiver det respektive tilhørsforhold, hvorefter kommandoen *by* returnerer et resultat for hvert outcome af tilhørsforholdsvariablen. Hvis vi ønsker, at finde gennemsnitsprisen for henholdsvis indenlandske og udenlandske biler kan *by*-kommandoen benyttes.

```
// generates mean price by foreign
by foreign: egen pricemean2 = mean(price)
```

Disse kommandoer kan også samles i en, hvilket kommandoen *bysort* gør. Hvis vi ønsker gennemsnitsprisen givet x miles per gallon kan vi benyttes *bysort*-kommandoen. Det er en god vane at benytte bysort kommandoen altid, da dette sikrer at Stata ikke går i stå. Dette skyldes, at når visse kommandoer benyttes, så skal data være sorteret, hvilket sker, når *bysort* benyttes frem for *by*.

```
// generates and sorts price by miles per gallon
bysort mpg: egen pricemean3 = mean(price)
```

2.3.3 Labels

label data, variables, value

En virkelig nyttig funktion ved Stata er dens label-kommandoer. Labels bliver benyttet til at beskrive datasæt, variable og variablers værdier. Dette har den smarte funktion, at når der dannes grafer, så vil Stata automatisk benytte disse labels til grafernes akser, og danne overskrifter, når grafer laves på baggrund af gruppering.

I mange tilfælde er det væsentligt, at have nogle centrale informationer om et datasæt. Det er muligt at vedhæfte op til 80 tegn til et datasæt i Stata. Når et datasæt har vedhæftet en *data label*, så vil denne label blive vist, når datasættet indlæses. F.eks. fremgik *"(1978 Automobile Data)"*, da vi indlæste datasættet *auto8.dta*. Dette skyldes, at denne label var vedhæftet datasættet.

Det er ligeledes muligt, at vedhæfte labels til variable. Det smarte ved at gøre dette er, at når der dannes grafer og benyttes kommandoer i Stata, så vil stata benytte labelnavnet i stedet for variabelnavnet. Dette gør det væsentligt hurtigere og mere sikkert at arbejde i Stata.

Endeligt er der en sidste type af labels, *value labels*. Value labels benyttes til at vedhæfte en label til forskellige udfald i en variabel. Dette betyder f.eks. at man kan vedhæfte om det er mand eller kvinde der har værdien 1 eller 0, og der lignende fordele i forbindelse med grafer som beskrevet herover.

I nedenstående kode er der eksempler på alle tre typer labels.

```
// Data and variable labels
label data "This is my first data label"
label variable price "This is my first variable label"
// Defining value labels and attaching those
label define firstlabel 1 "Miles per gallon exceeds 20" 2 "Miles per gallon does not exceed 20"
label values mpg20_dummy firstlabel
```

2.4 Analyse

Inden man går igang med sin analyse er det vigtigt, at gøre sig nogle tanker om, hvilket data man har med at gøre.

2.4.1 Dataoverblik

describe

Stata har flere funktioner til at beskrive data. f.eks. kommer vi ikek til at gennemgå *codebook* og *inspect*, men derimod *describe* og *summarize*. *describe* benyttes til at beskrive formatteringen af data. Kommandoen siger intet om outcomes, fordelingen mv., men angiver hvordan disse observationer er opbevaret i Stata. En anden smart funktion ved describe er, at det angiver data og variable labels, samt angiver navnet på value labels der eventuelt er vedhæftet variable. For at se de respektive value labels kan nedenstående kode benyttes.

```
//describes data
desc
// Shows value labels
label list firstlabel
```

Det er endvidere muligt, at benytte kommandoen for udelukkende et udpluk af variable, og ligeledes for andre datasæt, hvilke ikke er indlæst i Stata. Dette gøres ved nedenstående stykke kode.

```
// describes the specified variables in the loaded dataset
desc price mpg
// describes the specified dataset
desc using "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\auto8.dta"
// describes the specified variables in the specified dataset
desc price mpg using "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\auto8.dta"
```

summarize

Efter at have fået et overblik over, hvordan datasættet og dets variable er bygget op, så vil det næste naturlige skridt være, at få et overblik over den grundlæggende fordeling af variablene i datasættet. Til dette benyttes kommandoen *summarize*. Summarize returnerer five-number summary for alle variable i det respektive datasæt. Det er dog også muligt, at specificere hvilke variable der ønskes en five-number summary.

```
// five−number summary for all variables
sum
// five−number summary for price and mpg
summ price mpg
```

Det er også muligt, at få Stata til at returnere et mere detaljeret overblik over fordelingen af data. Dette gøres ved, at benytte nedenstående kommando. Hvis *detail* betingelse benyttes, så vil Stata også returnere *skewness* og *kurtosis*, samt en del percentiler. Det er endvidere også muligt, at benytte *bysort*-kommandoen. Hvis dette gøres, så vil Stata returnere et styk output for hvert unikt outcome i identifikationsvariablen (*foreign* i nedenstående tilfælde).

```
// detailed summary for price and mpg
summ price mpg, detail
// summary for price and mpg by foreign
bysort foreign: summ price mpg
```

Efter *summarize*-kommandoen er udført, så vil Stata lagre nogle resultater internt. Disse kan bruges til at danne variable med. Det er muligt at tilgå en liste med hvilke resultater Stata lagrer ved nedenstående kommando, dog skal *summarize*-kommandoen først være kørt.

```
return list
```

Disse resultater kan nemt gemmes i variable, hvilket kan gøres ved:

```
gen num=r(N)
gen sum=r(sum)
```

tabulate, tabstat

Den sidste vigtige kommando til at danne sig et overblik over data er *tab*, hvilken danner en frekvenstabel over de specificerede variable. Det er naturligvis også muligt at kombinere *tab* med *if* og *bysort*. Desuden er det muligt, at få *tab*-kommandoen til at danne en dummy for hvert niveau tabellen. Det vil sige, at hvis vi danner en dummy for hvert niveau af *trunk*, så vil der blive dannet 18 dummy-variable for hvert niveau.

```
// tabulate trunk
tab trunk
// tabulate trunk by foreign
bysort foreign: tab trunk
//tabulate trunk, and create dummies
tab trunk , gen(var9)
```

tabstat er en af de vigtigste kommandoer, der er i Stata. Kommandoen danner en tabel over de specificerede variable og de specificerede statistikker. F.eks. hvis vi ønsker en tabel, hvor vi finder standardafvigelsen, variance, middelværdi og skævhed, så kan *tabstat* benyttes.

```
tabstat price mpg weight, stat(mean variance sd skew)
```

Det er også muligt, at få variablene vist som rækker, og statistikkerne vist som søjler, samt at danne statistikken på baggrund af tilhørsforhold.

```
tabstat price mpg weight, by(foreign) stat(mean variance sd skew) columns(stat)
```

2.4.2 Export af tabstat

Der er nogle forholdsvis gode pakker til at eksportere tabstat i forskellige formater. *latabstat* eksporterer *tabstat*-outputtet til LaTeX-format. Syntaksen er fuldstændig den samme som for tabstat. Herefter bliver outputtet vist i evalueringsområdet i Stata. Hvis man blot skal bruge et hurtigt output kan man blot benytte log-filen. mht. logfilen, så gøres det på følgende måde

```
cap log close
log using logfil, replace text
tabstat price
log close
```

Endeligt, hvis outputtet ønskes i Stata, så anbefales det, at man enten benytter *collapse* eller laver en *postfile* (bliver gennemgået længere fremme), hvorefter kommandoen *xmlsave* benyttes.

2.4.2.1 dubletter

dublicates

I forbindelse med registerdata kan der opstå dubletter. Det vil sige, at der er fejl i enten kodningen eller fremstillingen af data. Dubletter kan have betydning for estimering med videre. Stata har forskellige kommandoer til dette. Kommandoen *duplicates* giver blandt andet mulighed for at slette dubletter, rapportere dubletter og generere en variabel, der angiver antallet af dubletter for den respektive observation. Hvis der ikke er nogle dubletter vil denne variabel naturligvis udelukkende antage værdien 1.

```
// drop duplicates
duplicates drop
// report on duplicates
duplicates report
// generates a varible, which tells how many duplicates
duplicates tag, gen(indikationsvar)
```

2.4.3 Regressioner

regress

OLS-regressionen udføres i Stata ved *regress*-kommandoen. Først angives afhængige variabel, hvorefter de uafhængige variable angives. Når en regression i Stata er udført gemmes relevante resultater fra regressionen internt i Stata (på samme måde som ved *summarize*). Disse kan naturligvis benyttes på samme måde som i *summarize*. De internt gemte resultater kan ses ved nedenstående stykke kode.

```
// Regress price on trunk
reg price trunk
// return results
ereturn list
```

Nedenstående kode er nyttig at forstå. Det vil sige, at vi danner en tabstat på baggrund af de observationer, der er blevet brugt i den regression, der blev kørt sidst.

```
tab price if e(sample)==1
```

Det er også muligt at gemme regressionerne permanent i Stata, således at disse kan tilgås ved et senere tidspunkt. Dette gøres ved kommandoen *eststo*[2].

eststo, esttab

```
// Stores the following linear regressions
eststo reg1: reg price trunk
eststo reg2: reg price mpg
// returns the most important results from regression reg1 and reg2
esttab reg1 reg2
```

2.5 Smart tricks

Stata har nogle smarte kommandoer, der gør arbejdsprocesserne væsentligt nemmere. Disse kommandoer vil blive gennemgået nærværende afsnit.

quietly, capture, inrange, missing, _n, _N

I situationer, hvor man f.eks. kun benytter en enkelt statistik fra *summarize*-kommandoen kan det være fordelagtigt, at tilgå informationen, men undertrykke outputtet. Dette kan gøres ved kommandoen *quietly*. Denne kommando gør, at Stata kører koden internt, hvilket gør, at man kan tilgå resultaterne. Resultaterne bliver dog ikke outputtet. F.eks. kunne vi ønske os, at vide, hvor mange biler kører mere end 20 miles per gallon. Dette er den eneste information vi er ude efter. Dette kan gøres ved nedenstående kode.

```
// summarize quietly
qui: sum price if mpg > 20
// display number of observations
disp r(N)
```

Kommandoen capture kan benyttes til at sikre at koden bliver kørt. Denne kommando skal man være forsigtig med, da den kan ødelægge analysen, hvis kommandoen bliver misbrugt. Det kommandoen gør er, at den undertrykker alt output, herunder fejlmeddelelser. Det vil sige, at hvis f.eks. en variabel ønskes genereret, men denne allerede eksisterer, så vil Stata blot køre over denne linje kode.

```
gen j = 1
//supresses error
cap gen j = 2
```

Kommandoen *inrange* sikrer blot, at en given variabel ligger i det specificerede interval. Dette kan lette læsningen af betingelser, og gør kodningen mere sikker.

[2]The package *estout* has to be installed. To install *estout* write the following in Stata command terminal *ssc install estout, replace*

```
// The following to lines of code do the same
sum price if trunk<=21 & trunk>=10
sum price if inrange(trunk,10,21)
```

I forbindelse med registerdata oplever man ofte, at der er missing values. Dette skal man huske at tage højde for, da missing values opfører sig som uendelig store værdier. Dette vil sige, at hvis man f.eks. ønsker sig alle personer, der ældre en 50 år, og der eksisterer personer med en missing values i alders variablen vil disse også blive talt med. Kommandoen *missing* er en logisk kommando, hvilken angiver 1, hvis missing, og 0 ellers.

```
// summarize if mpg is missing
sum price if mi(mpg)
// summarize if mpg is not missing
sum price if !mi(mpg)
```

Udtrykket *_N* angiver det samlede antal observationer, og udtrykket *[_n+1]* angiver observation nummer n+1. Dette kan vi f.eks. bruge i nedenstående stykke kode, hvor vi beholder den første observationer af hver værdi af *trunk*.

```
// keep only the first observations by trunk
bysort trunk: keep if _n == 1
```

2.6 Figurer

I dette afsnit vil der udelukkende blive gennemgået tre forskellige typer figurer, histograms, bar plots og scatter plots. For mere avancerede figurer og formatteringen af figurerne anbefales det, at benyttes sig af det vedlagte cheat sheet for data visualisering bagerst i nærværende dokument.

histograms

Histogrammer dannes blot ved nedenstående kode. Bin-option angiver antallet af søjler.

```
// histogram over price
hist price, bin(10)
```

barplot

Barplot er et søjlediagram. Dette er også forholdsvis enkelt at danne. Dette gøres normalt over grupperinger, da outputter ellers blot bliver en enkelt søjle.

```
// barplot over price
graph bar (median) price, over(trunk)
```

scatter plot Et scatter plot dannes ved *twoway scatter*-kommandoen. Først angives den afhængige variabel og derefter den uafhængige variabel. Det vil sige, at ved nedenstående kode plotter vi *price* mod *mpg*.

```
// we plot price against miles per gallon
twoway scatter price mpg
```

Hvis vi benytter kommandoen *describe*, så kan vi se, at både *price* og *mpg* har tilknyttet labels. Havde dette ikke været tilfældet, så ville akserne blot have de respektive variable som navne. Lad os illustrere dette med et tilfælde. Vi danner en dummy for, hvorvidt trunk er større end 15. Vi benytter by kommandoen i *twoway*, hvilket resulterer i to seperate grafer placeret ved siden af hinanden.

```
// dummy. Is trunk larger than 15
gen trunk_dummy = (trunk > 15)
// we plot price against miles per gallon by trunk_dummy
twoway scatter price mpg, by(trunk_dummy)
```

Hvis vi giver trunk_dummy en label, og dens værdier en label, så bliver grafen meget mere forståelig.

```
// variable label
label variable trunk_dummy "Turn cicle larger than 15"
// value label
label define label_trunk_dummy 0 "Turn cicle is not larger than 15 feet" 1 "Turn cicle is larger than 15
    feet"
label values trunk_dummy label_trunk_dummy
// new plot
twoway scatter price mpg, by(trunk_dummy)
```

graph save, graph combine, graph export

Det er muligt at gemme grafer internt i Stata, således at disse kan tilgås på et senere tidspunkt. Dette gøres ved at benytte *graph save* ligesom i nedenstående kode. Herved vil Stata vide, at dette navn henviser til følgende graf. *Replace*-option tilføres for at overskrive eventuelle gemte grafer.

```
// the plot from before
twoway scatter price mpg, by(trunk_dummy)
// The plot is saved
graph save "plot1.gph", replace
// a histogram
hist price, bin(5)
// The plot is saved
graph save "plot2.gph", replace
```

Vi har nu gemt de to plots i Stata. Det er muligt at kombinere de to plots med *graph combine*, og ligeledes er det muligt at eksportere grafer til en sti eller til den definerede sti. Begge dele er vist i nedenstående stykke kode.

```
// We combine the two graphs
graph combine plot1.gph plot2.gph
// we export the graph to our specified directory
graph export Plot.pdf, replace
```

2.7 Locals, globals og loops

```
use auto8.dta, clear
```

local, global

Locals kan betragtes som en slags midlertidige variable, der slettes, når koden er kørt. Det er normalt at locals indeholder skalarer og strenge, tekstykker. Vi kan f.eks. lade nedenstående local indeholde skalaren 1. Locals og globals kan vises ved kommandoen *display*. For at Stata kan forstå, at der er tale om en local skal denne omgives af *'local'*.

```
local tal = 1
disp `tal'
```

Vi kunne også have ladet ovenstående local indeholde en tekststring.

```
local string = "Dette er en local"
disp "`string'"
```

Globals bliver modsat locals ikke slettet, når et stykke kode er kørt, men slettes når programmet bliver lukket. Globals kan indeholde mere information end locals, men de har nogenlunde samme funktion på nær, at globals som sagt ikke bliver slettet. For at Stata kan forstå, at der er tale om en global skal denne omgives af *${globalnavn}*

```
global tal = 1
disp ${tal}
global string = "Dette er en global"
disp "${string}"
```

loops

Locals og globals er særdeles nyttige, når man bruger loops. Loops er meget nemme at lave i STATA, og der findes mange forskellige slags. For at lave et simpelt loop, kan 'forvalues' bruges.

```
forvalues 'iterationsvariabel'=1/10 {
'kommando'
}
```

Hvor 'iterationsvariabel' er en local, der skiftevis er lig 1,2,3,4,5,6,7,8,9 og 10. Imellem de to 'tuborgklammer' {}, skrives den kommando man ønsker at udføre for hver iteration. Hvis jeg for eksempel gerne vil lave en 'summarize' af variablen 'mpg' (miles per gallon) for hver niveau af 'rep78' (Repair record 1978), der antager værdierne 1-5, kan dette udføres ved at skrive:

```
forvalues rep=1/5 {
sum mpg if rep78==`rep'
}
```

Alternativt kan 'rep=1/5' skrives som 'rep=1(1)5', hvor '(1)' angiver at rep skal stige med 1 for hver iteration.

Loops kan også laves med funktionen 'foreach'. 'foreach' tillader blandt andet at bruge en strengvariabel, som iterationsvariabel. For eksempel, hvis jeg ønsker at lave en 'summarize' for hver af variablene 'trunk', 'mpg', 'length' og 'weight'. Her skrives:

```
foreach var in trunk mpg length weight {
sum `var'
```

```
}
```

En meget smart funktion som kan bruges i forbindelse med 'foreach' er 'levelsof'. 'levelsof' tæller antallet af unikke værdier i en given variabel, og gemmer denne information i en local. Dette gøres ved at skrive: levelsof 'variabel' , local('localnavn').

Derefter kan man bruge 'foreach' til at lave et loop for alle disse værdier ved at skrive:

```
forvalues 'iterationsvariabel' of local 'localnavn' {
'kommando'
}
```

Hvis man ønsker at lave ovenstående loop med 'summarize' af 'mpg' for hver niveau af 'rep78', kan der f.eks. skrives:

```
levelsof rep78, local(repniveauer)

foreach rep of local repniveauer {
sum mpg if rep78==`rep'
}
```

Globals kan bruges på samme vis som locals blev brugt i eksemplet foroven. Derudover findes der et hav af forskellige muligheder med 'foreach'. Tjek STATA's hjælpeside for mere info.

2.8 Postfile

Postfile er meget brugbar når der skal laves figurer i STATA. Det giver mulighed for hurtigt at lave nogle forholdsvis advancerede figurer, samt resultatdatasæt, hvis man skulle have behov det. Ideen er at gemme resultater fra for eksempel regressioner i et nyt datasæt. Derefter åbnes dette datasæt og resultaterne kan nemt tegnes som en figur.

Syntaksen er som følger:

```
    postfile 'postfilenavn' 'var1' var2' 'var3' ... using 'datasetnavn' , replace

    'kommando'

    post 'postfilenavn' ('input til var1') ('input til var2') ('input til var3') ...

    postclose 'postfilenavn'
```

Her kunne jeg for eksempel være interesseret i at plotte OLS-koefficienten og konfidensbånd fra en regression af 'mpg' på 'trunk' - for hver niveau af 'rep78'. I dette tilfælde ønsker jeg derfor at gemme fire variable; rep78-niveauet, koefficienten, nedre konfidensbånd og øvre konfidensbånd. For at udføre selve estimationen kan jeg bruge et loop. Koden ser ud som følger:

```
postfile fedfigur rep koef nedre oevre using "C:\Users\vzx151\Dropbox\Research Assistant\
    IntroForskningsassistent\figurdata", replace
```

```
forvalues rep=1/5 {

reg mpg trunk if rep78==`rep'

post fedfigur (`rep') (_b[trunk]) (_b[trunk]−_se[trunk]∗1.96) (_b[trunk]+_se[trunk]∗1.96)

}

postclose fedfigur
```

For at tegne figuren åbnes det nye datasæt

```
clear all
use "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\figurdata", clear

twoway (line nedre oevre rep, lpattern(dash dash)) (line koef rep), ///
scheme(s2mono) graphregion(color(white)) title("Postfile figur") ///
legend(order(3 1) label(1 "Konfidensbaand") label(3 "Koefficient: Milage paa trunk size")) xtitle("Repair
    record 1978, hvad end det betyder") ytitle("Miles per gallon")

graph export fedpostfilefigur.pdf, replace
```

Hovedfordelen ved postfile er helt klart fleksibiliteten. Derudover er det markant hurtigere, hvis man arbejder med store datasæt. I så fald ville jeg benytte et STATA-program til at generere figurdataen og et til at tegne figurerne, så data ikke skal indlæses på ny.

2.9 Esttab

Vi indlæseser data igen.

```
use auto8.dta, clear
```

Esttab er en nem og overskuelig måde at lave flotte regressionstabeller i STATA. Ideen er at man gemmer resultaterne af en række regressioner og derefter input'er dem i en tabel. Hvis jeg for eksempel ønsker at at smide resultaterne af ovenstående regressioner af 'mpg' på 'trunk' for niveau af rep78 i en tabel, kan det gøres på følgende måde:

```
eststo reg1: reg mpg trunk if rep78==1
eststo reg2: reg mpg trunk if rep78==2
eststo reg3: reg mpg trunk if rep78==3
eststo reg4: reg mpg trunk if rep78==4
eststo reg5: reg mpg trunk if rep78==5
```

Resultaterne gemmes som vist ved at skrive: eststo 'regnavn': 'regressionskommando'. I dette tilfælde er det oplagt at bruge et loop.

```
forvalues i=1/5 {
eststo reg`i': reg mpg trunk if rep78==`i'
```

```
}
```

Herefter omdannes regressionsoutputtet til en tabel ved at bruge esttab

```
esttab reg1 reg2 reg3 reg4 reg5 using "C:\Users\vzx151\Dropbox\Research Assistant\
    IntroForskningsassistent\flotesttabtabel1.rtf", replace ///
title("Awesome Esttab Tabel")
```

Hvis man ønsker at teste om trunk size har selvstændig betydning for milage kunne man for eksempel lave følgende tabel:

```
eststo reg1: reg mpg trunk
eststo reg2: reg mpg trunk weight
eststo reg3: reg mpg trunk length
eststo reg4: reg mpg trunk weight length

esttab reg1 reg2 reg3 reg4 using "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\
    flotesttabtabel2.rtf", replace ///
title("Awesome Esttab Tabel: Har bagagerumsstoerrelse betydning for hvor langt en bil koerer paa literen?
    ") label ///
scalars("r2 R-squared" "r2_a Adjusted R-squared" )
```

Estout muliggør også output i LaTeX. Ovenstående regressionsoutput ville i LaTeX-format blive dannet ved nedenstående kode.

```
eststo reg1: reg mpg trunk
eststo reg2: reg mpg trunk weight
eststo reg3: reg mpg trunk length
eststo reg4: reg mpg trunk weight length

esttab reg1 reg2 reg3 reg4 using "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\
    flotesttabtabel2.tex", replace ///
title("Awesome Esttab Tabel: Har bagagerumsstoerrelse betydning for hvor langt en bil koerer paa literen?
    ") label ///
scalars("r2 R-squared" "r2_a Adjusted R-squared" )
```

Dette danner følgende tabel

Det er utroligt nyttigt at kigge nærmere ind i denne pakke.

2.10 Frmtable

Frmttable er en til tider upraktisk, men utrolig fleksibel måde at tabeller i STATA. Ideen er at gemme resultater fra diverse funktioner i STATA i en matrix. Derefter konverterer frmttable denne til en flot regressionstabel. Fremgangsmåden er som følger. Først genereres en matrix med det ønskede antal rækker og kolonner. Hvis man ønsker standardfejl eller andre under-resultater, er det en god ide at fordoble (tredoble, hvis man har to under-resultater) antallet af kolonner, og lade være med at tælle disse med i antallet af rækker. Ved at inputte disse resultater i kolonnen ved siden af hovedresultatet, kan man ved hjælp af 'sub'-funktionen

Table 2: Awesome Esttab Tabel: Har bagagerumsstoerrelse betydning for hvor langt en bil koerer paa literen?

	(1)	(2)	(3)	(4)
	Mileage (mpg)	Mileage (mpg)	Mileage (mpg)	Mileage (mpg)
Trunk space (cu. ft.)	-0.787***	-0.0962	-0.00967	-0.0323
	(-6.07)	(-0.75)	(-0.07)	(-0.24)
Weight (lbs.)		-0.00565***		-0.00388*
		(-8.06)		(-2.42)
Length (in.)			-0.205***	-0.0742
			(-7.56)	(-1.23)
Constant	32.12***	39.69***	60.04***	47.40***
	(17.20)	(24.02)	(15.20)	(7.33)
Observations	74	74	74	74
R-squared	0.338	0.654	0.633	0.662
Adjusted R-squared	0.329	0.645	0.623	0.647

t statistics in parentheses

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$

i frmttable få STATA til at smide standardfejlene under resultaterne og pakke dem ind i parenteser eller andre 'brackets'. Når man har kørt de regressioner, summary statistics osv. som man ønsker, og gemt de resultater som man skal bruge, kan man benytte frmttable til at konstruere en flot tabel med aksetitler, noter og meget mere. For at illustrere fremgangsmåden, vil jeg nu genskabe ovenstående tabel.

Jeg skal bruge 7 rækker (for trunk, weight, length, konstantleddet, observationer, R-i-anden og justeret R-i-anden) og 4*2=8 kolonner (da jeg gerne vil inkludere standardfejl). Resultatmatricen 'X' laves således:

Koden er lidt rodet i dette dokument. Det anbefales, at kigge i do-filen for dette eksempel.

```
mat X=J(7,8,.) /* Laver en tom matrix med 7 raekker og 8 kolonner. */

* Herefter laver jeg hver af regressionerne og inputter resultaterne loebende i matricen*

* Reg 1 *
reg mpg trunk

mat X[1,1]=_b[trunk] // Smider regressionskoefficienten for 'trunk' i cellen i raekke 1, kolonne 1.
mat X[1,2]=_se[trunk] // Smider standardfejlen for 'trunk' i cellen i raekke 1, kolonne 2.
mat X[4,1]=_b[_cons] // Smider regressionskoefficienten for konstantleddet i cellen i raekke 4, kolonne 1.
mat X[4,2]=_se[_cons] // Smider standardfejlen for konstantleddet i cellen i raekke 4, kolonne 2.

mat X[5,1]=e(N) // Smider antal observationer i cellen i raekke 5, kolonne 1.
mat X[6,1]=e(r2) // Smider r-i-anden i cellen i raekke 6, kolonne 1.
mat X[7,1]=e(r2_a) // Smider justeret r-i-anden i cellen i raekke 7, kolonne 1.

* Reg 2 *
reg mpg trunk weight

mat X[1,3]=_b[trunk] // Smider regressionskoefficienten for 'trunk' i cellen i raekke 1, kolonne 3.
```

```
mat X[1,4]=_se[trunk] // Smider standardfejlen for 'trunk' i cellen i raekke 1, kolonne 4.
mat X[2,3]=_b[weight] // Smider regressionskoefficienten for 'weight' i cellen i raekke 2, kolonne 3.
mat X[2,4]=_se[weight] // Smider standardfejlen for 'weight' i cellen i raekke 2, kolonne 4.
mat X[4,3]=_b[_cons] // Smider regressionskoefficienten for konstantleddet i cellen i raekke 4, kolonne 3.
mat X[4,4]=_se[_cons] // Smider standardfejlen for konstantleddet i cellen i raekke 4, kolonne 4.

mat X[5,3]=e(N) // Smider antal observationer i cellen i raekke 5, kolonne 3.
mat X[6,3]=e(r2) // Smider r-i-anden i cellen i raekke 6, kolonne 3.
mat X[7,3]=e(r2_a) // Smider justeret r-i-anden i cellen i raekke 7, kolonne 3.



* Reg 3 *
reg mpg trunk length

mat X[1,5]=_b[trunk] // Smider regressionskoefficienten for 'trunk' i cellen i raekke 1, kolonne 5.
mat X[1,6]=_se[trunk] // Smider standardfejlen for 'trunk' i cellen i raekke 1, kolonne 6.
mat X[3,5]=_b[length] // Smider regressionskoefficienten for 'length' i cellen i raekke 3, kolonne 5.
mat X[3,6]=_se[length] // Smider standardfejlen for 'length' i cellen i raekke 3, kolonne 6.
mat X[4,5]=_b[_cons] // Smider regressionskoefficienten for konstantleddet i cellen i raee 4, kolonne 5.
mat X[4,6]=_se[_cons] // Smider standardfejlen for konstantleddet i cellen i raekke 4, kolonne 6.

mat X[5,5]=e(N) // Smider antal observationer i cellen i raekke 5, kolonne 5.
mat X[6,5]=e(r2) // Smider r-i-anden i cellen i raekke 6, kolonne 5.
mat X[7,5]=e(r2_a) // Smider justeret r-i-anden i cellen i raekke 7, kolonne 5.

* Reg 4 *
reg mpg trunk weight length

mat X[1,7]=_b[trunk] // Smider regressionskoefficienten for 'trunk' i cellen i raekke 1, kolonne 7.
mat X[1,8]=_se[trunk] // Smider standardfejlen for 'trunk' i cellen i raekke 1, kolonne 8.
mat X[2,7]=_b[weight] // Smider regressionskoefficienten for 'weight' i cellen i raekke 2, kolonne 7.
mat X[2,8]=_se[weight] // Smider standardfejlen for 'weight' i cellen i raekke 2, kolonne 8.
mat X[3,7]=_b[length] // Smider regressionskoefficienten for 'length' i cellen i raekke 3, kolonne 7.
mat X[3,8]=_se[length] // Smider standardfejlen for 'length' i cellen i raekke 3, kolonne 8.
mat X[4,7]=_b[_cons] // Smider regressionskoefficienten for konstantleddet i cellen i raekke 4, kolonne 7.
mat X[4,8]=_se[_cons] // Smider standardfejlen for konstantleddet i cellen i raekke 4, kolonne 8.

mat X[5,7]=e(N) // Smider antal observationer i cellen i raekke 5, kolonne 7.
mat X[6,7]=e(r2) // Smider r-i-anden i cellen i raekke 6, kolonne 7.
mat X[7,7]=e(r2_a) // Smider justeret r-i-anden i cellen i rae 7, kolonne 7.

* Tabel *

frmttable using "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\flotesttabtabel1.
    rtf", replace ///
title("Awesome Frmttable Tabel: Har bagagerumsstoerrelse betydning for hvor langt en bil koerer paa
    literen?") ///
```

```
/* Vigtige options */ ///
/* 1 */ statmat(X) /// /* Siger hvilken matrix resultaterne skal komme fra. */
/* 2 */ sub(1) /// /* Angiver antallet af underresultater - her 1 fordi jeg oensker standardfejl. */
/* 3 */ ctitle("", "(1)", "(2)", "(3)" , "(4)" \ "", "Milage(mpg)", "Milage(mpg)", "Milage(mpg)", "Milage(mpg)
    ") /// /* Specificerer kolonnetitler, bemaerk at det er muligt at lave flere linjer/raekker med tekst ved
     brug af '\'.*/
/* 4 */ rtitle("Trunk space (cu ft.)" \ " " \ "Weight (lbs.)" \ " " \ "Length (in.)" \ " " \ "Constant" \ " " \ "
    Observations" \ "" \ "R-squared" \ "" \ "Adjusted R-Squared" \ "" ) /// /* Specificerer raekketitler
    */
/* 6 */ sdec(5 \ 5 \ 5 \ 5 \ 5 \ 5 \ 5 \ 5 \ 0 \ 2 \ 2) /// /* Specificerer antal decimaler - kan specificeres
     for hver enkel celle ved at bruge ',' som kolonne-adskiller og '\' som raekke-adskiller." */
/* 7 */ hlines( 1 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 1) /* Specificerer hvor der skal vaere horisontale linjer.
    Default er bund og top. Der findes en tilsvarende funktion til vertikale linjer kaldet 'vlines'. */
```

Givet de mange linjer kode og de besværlige options er det selvfølgelig fuldstændig tåbeligt at bruge frmttable til denne tabel. Den kan som vist laves noget nemmere i Esttab. Der er dog tabeller, der ikke kan laves i Esttab, hvor kendskab til frmttable derfor er en nødvendighed. En sådan tabel er præsenteret nedenfor. Denne viser summary statistics for en række variable for hver niveau af rep78. Her kan der også bruges loops, hvilket gør arbejdet med matricen nemmere.

```
global varliste "mpg trunk weight length price" /* Vaelg variable */

mat Y=J(5,20,.) /* Jeg oensker at lave en tabel med 5 raekker og 5 kolonner, men med tre under-
    resultater - standard afvigelsen, minimum og maksimum. */

local i=1 /* Angiver raekkenummer, startende paa 1. */

foreach var of global varliste {

    local j=1 /* Angiver kolonnenummer, startende paa 1. */

    forvalues rep=1/5 {

    sum `var' if rep78==`rep'

    mat Y[`i',`j']=r(mean)
    mat Y[`i',`j'+1]=r(sd)
    mat Y[`i',`j'+2]=r(min)
    mat Y[`i',`j'+3]=r(max)


    local j=`j'+4 /* Spring fire kolonner over i naeste iteration */

    }

local i= `i'+1 /* Gaa til naeste raekke i naeste iteration */

}
```

```
frmttable using "C:\Users\okoBS\Documents\Intro Forskningsassistenter\flotfrmttableTabel2_ny.rtf",
    replace ///
title("Table: Summary statistics") statmat(Y) sub(3) brackets("" \ (,) \ [,] \ [,]) nocenter a4 ///
coljust(l c) addfont(Times New Roman) basefont(fnew1 fs12) statfont(fs12) sdec(2 \ 2 \ 2 \ 2 \ 2 \ 2 \
    2 \ 0 \ 0 \ 0 \ 0 \ 2 \ 2 \ 2 \ 2 \ 0 \ 0 \ 0 \ 0 ) ///
ctitle("", "Repair Record 1978" "" "" "" "" \ "", "1", "2", "3" , "4" , "5") multicol(1,2,5) ///
rtitle("Milage(mpg)" \ " " \ " " \ " " \ " " \ "Trunk space (cu ft.)" \ " " \ " " \ " " \ " " \ "Weight (lbs.)" \ " " \ " " \ " " \ "
    " \ "Length (in.)" \ " " \ " " \ " " \ " " \ "Price" \ " " \ " " \ " " \ " " ) ///
```

2.11 Programmering

Det kan være super smart at lave sine egne funktioner i STATA. Især hvis man skal udføre den samme kommando mange gange. Som med så mange andre ting i STATA, er der rigtig mange muligheder. Jeg viser kun hvad jeg ved om programmering i STATA.

Først og fremmest er det smart at sikre, at der ikke allerede er defineret et program med det navn ønsker at kalde sit. Det kan gøre ved simpelthen at skrive navnet i browseren og se om STATA kan genkende det. Man starter definitionen af et program ved at skrive: program 'programnavn'. For definere et program skal man vælge syntaxen af det man ønsker at inputte. Hvis det f.eks. er en variabelliste skrives: syntax varlist. Herunder kan der specificeres hvor mange variable der højest må angives ved at skrive: syntax varlist(max='maxtal').

Herefter kan der skrives: [if]. på denne måde understøtter dit program STATA's 'if'-funktion. Når der skrives 'if' gemmes en local med betingelsen, hvor denne afhænger af hvad brugeren skriver. Alle kommando'er indeni programmet skal derfor slutte med 'if'.

Hvis også ønsker at definere en række options for programmet kan dette gøres ved at skrive [, 'optionnavn1'('option1') 'optionnavn2'('option2')]. 'option' er et tal, navn eller en variabel som brugeren skriver. Resultatet heraf gemmes i en local 'optionnavn', som man kan referere til i programmet. I programmet skrives om 'option' er et navn (name), et tal (numlist(max='maxnum')) eller en variabel (varlist(max='maxnum'))

Nedenfor gives et simpelt eksempel:

Programmet laver for hver angivet variabel, en variabel indeholdende gennemsnittet.

```
cap program drop ben1

program ben1

syntax varlist(max=20) [if] [, prefix(name) replace] /* Definerer syntax. Tillad tyve inputvariable, tillad at
    brugeren selv kan definere navnet og tillad at erstatte eksisterende variable med samme navn. */

foreach var of local varlist {

qui sum `var' `if' /* Lav 'summarize' for hver variabel i variabellisten */

if "`prefix'" =="" { /* Lav default navn, hvis 'prefix' ikke er specificeret */
local prefix "mean"
```

```
di as txt "Default prefix 'mean' chosen."
}

if "`replace'" !="" {
cap drop `prefix'`var' /* Drop den eksisterende variabel, hvis der er skrevet 'replace' som option. */
}


qui gen `prefix'`var'=r(mean) `if' /* Gem resultat i nyvariabel */

la var `prefix'`var' "Mean of `:variable label `var'"" /* Giv ny variabel label'en 'Mean of 'variablens label''
      */
}

end
```

Benyt programmet

```
ben1 price mpg trunk weight length, replace
```

Normalt ville jeg gemme koden, der definerer programmet i en særskilt do-fil, så jeg kan køre denne i programmet, men et simpelt kodestykke. For eksempel har jeg et program kaldet program1, som jeg ofte bruger. Når jeg starter STATA skriver jeg derfor altid følgende:

```
qui do "C:\Users\vzx151\Dropbox\Research Assistant\IntroForskningsassistent\program.do"
program1 price mpg trunk weight length, replace
```

2.12 Dato, tid og formattering

2.12.1 Dato og tid

En ting man lige så godt kan venne sig til, er at statistikprogrammer generelt er utrolig ufleksible, når det kommer til datoer. Generelt, så er kommandoen *describe* ens ven, når man har med datoer at gøre. Her er det muligt, at se hvilket datoformat en variable er kodet. Dette gøre behandlingen lidt nemmere. Der er 8 datoformater i Stata.

stata	almindelig	stata format
datetime/c	millisekunder siden 01jan1960 00:00:00:000	%tc
dato	dage siden 01jan1960 (01jan1960=0)	%td
uge	uger siden 1960w1	%tw
måned	måneder siden 1960m1	%tm
kvartal	kvartaler siden 1960q1	%tq
halvår	halvår siden 1960h1	%th
år	pr siden 0000	%ty

Fordi Stata betragter datoer som numeriske, så bliver man også nødt til at betragte dem som numeriske. Imidlertid, så kan det være svært at holde styr på, hvor mange dage er det siden, at det var 25. november 1972. For at løse dette problem er der følgende funktioner.

stata	funktion	stata format
datetime/c	tc = ndyhms(M,D,Y,h,m,s)	%tc
datetime/c	tc = dhms(td, h,m,s)	%tc
datetime/c	tc = hms(h,m,s)	%tc
dato	td = mdy(M,D,Y)	%td
uge	tw = yw(Y,W)	%tw
måned	tm = ym(Y,M)	%tm
kvartal	tq = yq(Y,Q)	%tq
halvår	th = yh(Y,H)	%th

Endeligt er der også følgende måde, at konvertere datoer på. Det er utroligt vigtigt, at dette bliver gjort rigtigt. Hvis ikke dette gøres rigtigt, så får man helt forkerte datoer. F.eks. hvis man benytter en funktion der konverterer datoer om til måneder, men variablen er kodet som måneder, så bliver det selvsagt helt forkert.

From	datetime	date
datetime/c		td = dofc(tc)
date	tc = cofd(td)	
week		td = dofw(tw)
month		td = dofm(tm)
quarter		td = dofq(tq)
half-year		td = dofh(th)
year		td = dofy(ty)

From	week	month	quarter
date	tw = wofd(td)	tm = mofd(td)	tq = qofd(td)

From	half-year	year
date	th = hofd(td)	ty = yofd(td)

Endeligt er det nyttigt[3] at kende til, hvordan man udtrækker år, måned og dag fra en datovariabel (dvs. et format hvor man har år, måned og dato). Her findes følgende kommandoer, som er nyttige at kende til

Komponent	function	eksempel
år	year(td)	2013
måned	month(td)	5

[3]Det er naturligvis også nyttigt at kunne danne datoer fra stringe, men dette bliver ikke gennemgået i denne note, da alle register er kodet som dato-variable.

Komponent	function	eksempel
dag	day(td)	37
dage i året	doy(td)	187
uger i året	weel(td)	43
kvartaler i året	quarter(td)	3
halfår i året	halfyear(td)	2

2.12.2 Formattering

For at formattere en variabel i Stata er det første man skal vide, hvilken type variabel man har at gøre med. I stata findes der 6 typer af variable. De 5 af typerne er til at håndtere tal med, og den sidste type er til at håndtere strenge(tekstvariable). Det eneste man bør vide er, at *byte*, *int* og *long* kun kan opbevare heltal, hvorimod *float* og *double* kan opbevare decimaltal ligeledes. Strenge opbevares i den type variabel der kaldes *str"længde"*. Det vil sige, at variabel der indeholder udtrykket *"jeg"* som den længste string vil have formatet *str3*, og en variabel der har *"hvis"* som længste string vil have variabeltypen *str4* osv. Stata kan håndtere string-observationer op til ufatteligt mange tegn. Det vil sige op til et par milliarder. Stata kalder dog stringe på mere end 2045 tegn *strL*

I stata formatteres variable ved at angive variabel navn, og hvilken formattering der ønskes. Det er ligemeget, hvilken rækkefølge format og variabel kommer i koden. Nedenstående er altså evaluerer altså det samme.

```
format varlist %fmt
format %fmt varlist
```

Standard indstillingerne i Stata er som følger

variabel-type	format
byte	%8.0g
int	%8.0g
long	%12.0g
float	%9.0g
double	%10.0g
str#	%#s
strL	%9s

De vigtigste formater er følgende

numerisk

variabel-type	format
%#.#g	generel
%#.#f	fixed
%#.#gc	generel, med komma
%#.#fc	fixed med komma
double	%10.0g
str#	%#s

variabel-type	format
strL	%9s

strenge

variabel-type	format
%#s	højrejusteret
%-#s	center
%~#s	venstrejusteret

2.13 Merge

Selvom, at Stata ikke er særligt godt at forberede datasæt i, så kan det være nødvendigt, at kunne udføre simple merging-procedurer i Stata. Der er grundlæggende fire typer af merging i Stata, men det er kun de tre af dem, som er brugbare i forbindelse med registerdata. Grundlæggende er merging, at man samler to datasæt på baggrund af en identifikationsvariabel der er unik, eller en kombination af variable. Dette medfører, at vi er sikre på, at det er de rigtige observationer, der bliver sat sammen. F.eks. i en situation, hvor vi har to datasæt på årsniveau, så vil PNR alene ikke være nok til at kunne identificere de rigtige observationer. Det vil dog være muligt, at merge de to datasæt sammen, hvis vi både har PNR og år (der må altså kun være en observation pr pnr pr år). I stata merger man 1:1 på følgende måde.

Til følgende eksempel danner jeg to relevante datasæt. Til at begynde med, så danner vi et identifikationsnummer til hver observation, hvilket vi skal bruge til at merge. Det vil sige, at vi har to forskellige datasæt, hvor identifikationsnummeret er unikt for hver observation. Altså, det er udelukkende den respektive observation, der har dette identifikationsnummer.

```
use auto8.dta, clear
gen iden = [_n]
keep make price mpg rep78 iden
save merge1.dta, replace

use auto8.dta, clear
gen iden = [_n]
drop make price mpg rep78
save merge2.dta, replace
```

Efter vi har forberedt datasættet, så vil vi gerne danne datasættet *auto8.dta*. Dette kan vi gøre ved følgende kode. Vi merger datasæt *merge2* på datasættet vi har loaded *merge1* på baggrund af variablen *iden*.

```
use merge1.dta, clear
merge 1:1 iden using merge2.dta
```

Udover 1:1 merge findes der også 1:m og m:1 merge. m:1 merge vil vi gennemgå til at begynde med. Her vil identifikationsvariablen unikt identificere observationer i using-datasættet. Det vil sige, at det er det datasæt der *IKKE* er loaded. Det vil sige, at for hver observation, i master-datasættet (altså det datasæt vi har

. merge 1:1 pid time using *filename*

master + *using* = *merged result*

pid	time	x1
14	1	0
14	2	0
14	4	0
16	1	1
16	2	1
17	1	0

pid	time	x2
14	1	7
14	2	9
16	1	2
16	2	3
17	1	5
17	2	2

pid	time	x1	x2	_merge
14	1	0	7	3
14	2	0	9	3
14	4	0	.	1
16	1	1	2	3
16	2	1	3	3
17	1	0	5	3
17	2	.	2	2

Figure 1: 1:1 merge

loaded) vil der blive merget oplysninger fra den unikke observation i using-datasættet. Et godt eksempel på, hvornår dette bliver brugt er f.eks. hvis vi ønsker at merge noget regionalt data på individ observationer.

. merge m:1 region using *filename*

master + *using* = *merged result*

id	region	a
1	2	26
2	1	29
3	2	22
4	3	21
5	1	24
6	5	20

region	x
1	15
2	13
3	12
4	11

id	region	a	x	_merge
1	2	26	13	3
2	1	29	15	3
3	2	22	13	3
4	3	21	12	3
5	1	24	15	3
6	5	20	.	1
.	4	.	11	2

Figure 2: m:1 merge

. merge 1:m region using *filename*

master + *using* = *merged result*

region	x
1	15
2	13
3	12
4	11

id	region	a
1	2	26
2	1	29
3	2	22
4	3	21
5	1	24
6	5	20

region	x	id	a	_merge
1	15	2	29	3
1	15	5	24	3
2	13	1	26	3
2	13	3	22	3
3	12	4	21	3
4	11	.	.	1
5	.	6	20	2

Figure 3: 1:m merge

1:m er det modsatte. Det vil sige, at i master-datasættet vil der være en unik observation, som bliver merget på hver observation i using datasættet, når der er et match.

Version June 28, 2016
Simon Halphen Boserup

How to move the location of Stata temporary data files

At startup, Stata creates a temporary file on the hard drive, which it uses for some of its procedures, e.g., merging datasets. This may fill up the C-drive, the default location of the temporary file. To avoid this, you have to tell Stata to place it somewhere more suitable. This guide instructs you how to do this.

- Open Explorer and navigate to `X:\StataWork` (CAM1 doesn't have an X-drive, use `F:` here instead)
- Create a subfolder using you DST ident and 4-digit project number, e.g., YYXX9999
- In that folder, right-click and create a new text document – rename it (and thereby change file type) to `runstata.bat`
- Right-click the file and choose 'Edit'
- The content of the file should be the following lines

```
set STATATMP="X:\StataWork\YYXX999"
cd "X:\StataWork\YYXX999"
X:
"C:\Program Files (x86)\Stata14\StataMP-64.exe"
```

- Save the file and exit the editor
- Double-click the bat-file. Stata 14 should open. To check that your hard labor panned out, do the following two checks

  1. In Stata's command line, type ``pwd''. You should get an answer saying that the new folder on the X-drive is your present working directory
  2. Also in Stata's command line, first execute the line ``tempfile f'', and then the line ``di "`f'"''. Again, you should see the new folder on the X-drive as a result.

You have now succeeded in moving the location of the temp file. Each time you run Stata, you should do it by clicking the `runstata.bat` file. To make it easier on you, do the following.

- Right-click and drag `runstata.bat` to the start menu in the lower left corner of your screen and release.
- Open the start menu – you should see a shortcut to the `runstata.bat` file at the top of the list somewhere.

NOTE! If you want to open an existing do-file, you'll need to click `runstata.bat` first and open that file via the do-file editor. If you just click the do-file directly in Explorer/Stifinder, you'll open Stata with a temporary file on the C-drive.

Data Processing with Stata 14.1 Cheat Sheet

For more info see Stata's reference manual (stata.com)

Useful Shortcuts

F2 — keyboard buttons
describe data

Ctrl + **9**
open a new .do file

Ctrl + **8**
open the data editor

Ctrl + **D**
highlight text in .do file, then ctrl + d executes it in the command line

clear
delete data in memory

At Command Prompt

PgUp **PgDn** scroll through previous commands

Tab autocompletes variable name after typing part

cls clear the console (where results are displayed)

Set up

pwd
print current (working) directory

cd "C:\Program Files (x86)\Stata13"
change working directory

dir
display filenames in working directory

fs *.dta
List all Stata data in working directory

capture log close
close the log on any existing do files
(underlined parts are shortcuts – use "capture" or "cap")

log using "myDoFile.txt", **replace**
create a new log file to record your work and results

search mdesc
find the package mdesc to install

ssc install mdesc
install the package mdesc; needs to be done once
(packages contain extra commands that expand Stata's toolkit)

Import Data

sysuse auto, clear
load system data (Auto data)
(for many examples, we use the auto dataset.)

use "yourStataFile.dta", **clear**
load a dataset from the current directory

import excel "yourSpreadsheet.xlsx", **/***
***/ sheet**("Sheet1") **cellrange**(A2:H11) **firstrow**
import an Excel spreadsheet
(frequently used commands are highlighted in yellow)

import delimited"yourFile.csv", **/***
***/ rowrange**(2:11) **colrange**(1:8) **varnames**(2)
import a .csv file

webuse set "https://github.com/GeoCenter/StataTraining/raw/master/Day2/Data"
webuse "wb_indicators_long"
set web-based directory and load data from the web

Basic Syntax

All Stata functions have the same format (syntax):

[**by** varlist1**:**] *command* [varlist2] [=*exp*] [*if exp*] [**in** range] [weight] [**using** filename] [,*options*]

- [**by** varlist1**:**] — apply the *command* across each unique combination of variables in *varlist1*
- *command* — function: what are you going to **do** to *varlists*?
- [varlist2] — column to apply *command* to
- [=*exp*] — save output as a new variable
- [*if exp*] — condition: only apply the function *if* something is true
- [**in** range] — apply to specific rows
- [weight] — apply weights
- [**using** filename] — pull data from a file (if not loaded)
- [,*options*] — special options for *command*

```
bysort rep78 : summarize price if foreign == 0 & price <= 9000, detail
```

In this example, we want a *detailed* summary with stats like kurtosis, plus mean and median

To find out more about any command – like what options it takes – type **help** *command*

Basic Data Operations

Arithmetic

- **+** add (numbers) combine (strings)
- **–** subtract
- ***** multiply
- **/** divide
- **^** raise to a power

Logic

== tests if something is equal
= assigns a value to a variable

- **&** and
- **!** or **~** not
- **|** or
- **==** equal
- **!=** or **~=** not equal
- **<** less than
- **<=** less than or equal to
- **>** greater than
- **>=** greater or equal to

if foreign != 1 & price >= 10000

make	foreign	price
Chevy Colt	0	3,984
Buick Riviera	0	10,372
Honda Civic	1	4,499
Volvo 260	1	11,995

if foreign != 1 | price >= 10000

make	foreign	price
Chevy Colt	0	3,984
Buick Riviera	0	10,372
Honda Civic	1	4,499
Volvo 260	1	11,995

Explore Data

View Data Organization

describe make price
display variable type, format, and any value/variable labels

count
count if price > 5000
number of rows (observations)
Can be combined with logic

ds, has(type string)
lookfor "in."
search for variable types, variable name, or variable label

isid mpg
check if mpg uniquely identifies the data

See Data Distribution

codebook make price
overview of variable type, stats, number of missing/unique values

summarize make price mpg
print summary statistics (mean, stdev, min, max) for variables

inspect mpg
show histogram of data, number of missing or zero observations

histogram mpg, **frequency**
plot a histogram of the distribution of a variable

Browse Observations within the Data

browse or **Ctrl** + **8**
open the data editor
(Missing values are treated as the largest positive number. To exclude missing values, use the !**missing**(varname) syntax)

list make price **if** price > 10000 & !**missing**(price)
list the make and price for observations with price > $10,000

clist ... (compact form)

display price[4]
display the 4th observation in price; only works on single values

gsort price mpg (ascending) **gsort –**price **–**mpg (descending)
sort in order, first by price then miles per gallon

duplicates report
finds all duplicate values in each variable

assert price**!=.**
verify truth of claim

levelsof rep78
display the unique values for rep78

Change Data Types

Stata has 6 data types, and data can also be missing:

no data	true/false	words	numbers
missing	**byte**	**string**	**int long float double**

To convert between numbers & strings:

1 → "1"
gen foreignString = **string**(foreign) → "1"
tostring foreign, **gen**(foreignString)
decode foreign , **gen**(foreignString) → "foreign"

1 → "1"
gen foreignNumeric = **real**(foreignString) → "1"
destring foreignString, **gen**(foreignNumeric)
encode foreignString, **gen**(foreignNumeric) → "foreign"

recast double mpg
generic way to convert between types

Summarize Data

tabulate rep78, **mi gen**(repairRecord)
(include missing values; create binary variable for every rep78 value in a new variable, repairRecord)
one-way table: number of rows with each value of rep78

tabulate rep78 foreign, **mi**
two-way table: cross-tabulate number of observations for each combination of rep78 and foreign

bysort rep78**: tabulate** foreign
for each value of rep78, apply the command tabulate foreign

tabstat price weight mpg, **by**(foreign) **stat**(mean sd n)
create compact table of summary statistics

table foreign, **contents**(mean price sd price) f(%9.2fc) **row**
(displays stats for all data; formats numbers)
create a flexible table of summary statistics

collapse (mean) price (max) mpg, **by**(foreign) – replaces data
calculate mean price & max mpg by car type (foreign)

Create New Variables

generate mpgSq = mpg^2 **gen byte** lowPr = price < 4000
create a new variable. Useful also for creating binary variables based on a condition (**generate byte**)

generate id = **_n** **bysort** rep78**: gen** repairIdx = **_n**
_n creates a running index of observations in a group

generate totRows = **_N** **bysort** rep78**: gen** repairTot = **_N**
_N creates a running count of the total observations per group

pctile mpgQuartile = mpg, **nq** = 4
create quartiles of the mpg data

egen meanPrice = **mean**(price), **by**(foreign)
calculate mean price for each group in foreign
(see **help egen** for more options)

Tim Essam (tessam@usaid.gov) • Laura Hughes (lhughes@usaid.gov)
follow us @StataRGIS and @flaneuseks

inspired by RStudio's awesome Cheat Sheets (rstudio.com/resources/cheatsheets)

geocenter.github.io/StataTraining

updated January 2016

Data Transformation with Stata 14.1 Cheat Sheet

For more info see Stata's reference manual (stata.com)

Select Parts of Data (Subsetting)

Select Specific Columns

drop make
remove the 'make' variable

keep make price
opposite of drop; keep only variables 'make' and 'price'

Filter Specific Rows

drop if mpg < 20 **drop in** 1/4
drop observations based on a condition (left) or rows 1-4 (right)

keep in 1/30
opposite of drop; keep only rows 1-30

keep if inrange(price, 5000, 10000)
keep values of price between $5,000 – $10,000 (inclusive)

keep if inlist(make, "Honda Accord", "Honda Civic", "Subaru"**)**
keep the specified values of make

sample 25
sample 25% of the observations in the dataset (use **set seed** # command for reproducible sampling)

Replace Parts of Data

Change Column Names

rename (rep78 foreign**) (**repairRecord carType**)**
rename one or multiple variables

Change Row Values

replace price = 5000 if price < 5000
replace all values of price that are less than $5,000 with 5000

recode price (0 / 5000 = 5000**)**
change all prices less than 5000 to be $5,000

recode foreign (0 = 2 "US"**)(**1 = 1 "Not US"**), gen(**foreign2**)**
change the values and value labels then store in a new variable, foreign2

Replace Missing Values

mvdecode _all, **mv(**9999**)** useful for cleaning survey datasets
replace the number 9999 with missing value in all variables

mvencode _all, **mv(**9999**)** useful for exporting data
replace missing values with the number 9999 for all variables

Label Data

Value labels map string descriptions to numbers. They allow the underlying data to be numeric (making logical tests simpler) while also connecting the values to human-understandable text.

label define myLabel 0 "US" 1 "Not US"
label values foreign myLabel
define a label and apply it the values in foreign

label list
list all labels within the dataset

note: data note here
place note in dataset

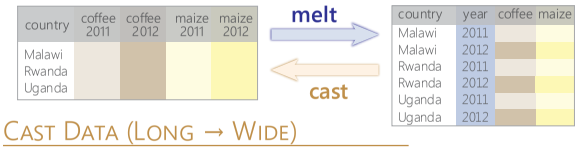
Reshape Data

webuse set https://github.com/GeoCenter/StataTraining/raw/master/Day2/Data
webuse "coffeeMaize.dta" load demo dataset

Melt Data (Wide → Long)

reshape variables starting with coffee and maize | unique id variable (key) | create new variable which captures the info in the column names

reshape long coffee@ maize@, **i(**country**) j(**year**)** — new variable
convert a wide dataset to long



Tidy Datasets have each observation in its own row and each variable in its own column.

Cast Data (Long → Wide)

create new variables named coffee2011, maize2012... | what will be unique id variable (key) | create new variables with the year added to the column name

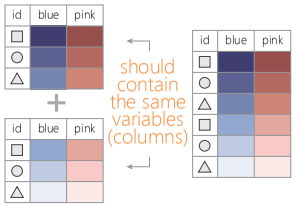
reshape wide coffee maize, **i(**country**) j(**year**)**
convert a long dataset to wide

When datasets are tidy, they have a consistent, standard format that is easier to manipulate and analyze.

xpose, clear varname
transpose rows and columns of data, clearing the data and saving old column names as a new variable called "_varname"

Combine Data

Adding (Appending) New Data

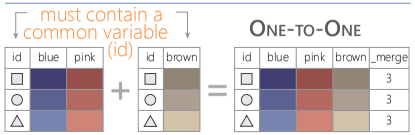


webuse coffeeMaize2.dta, **clear**
save coffeeMaize2.dta, **replace**
webuse coffeeMaize.dta, **clear** load demo data

append using "coffeeMaize2.dta", **gen(**filenum**)**
add observations from "coffeeMaize2.dta" to current data and create variable "filenum" to track the origin of each observation

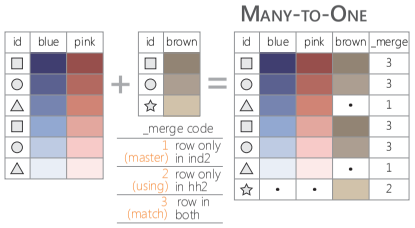
Merging Two Datasets Together

must contain a common variable (id)



webuse ind_age.dta, **clear**
save ind_age.dta, **replace**
webuse ind_ag.dta, **clear**

merge 1:1 id **using** "ind_age.dta"
one-to-one merge of "ind_age.dta" into the loaded dataset and create variable "_merge" to track the origin



webuse hh2.dta, **clear**
save hh2.dta, **replace**
webuse ind2.dta, **clear**

merge m:1 hid **using** "hh2.dta"
many-to-one merge of "hh2.dta" into the loaded dataset and create variable "_merge" to track the origin

Fuzzy Matching: Combining Two Datasets without a Common ID

reclink match records from different data sets using probabilistic matching ssc install reclink
jarowinkler create distance measure for similarity between two strings ssc install jarowinkler

Manipulate Strings

Get String Properties

display length("This string has 29 characters"**)**
return the length of the string

charlist make * user-defined package
display the set of unique characters within a string

display strpos("Stata", "a"**)**
return the position in Stata where a is first found

Find Matching Strings

display strmatch("123.89", "1??.?9"**)**
return true (1) or false (0) if string matches pattern

display substr("Stata", 3, 5**)**
return the string located between characters 3-5

list make **if regexm(**make, "[0-9]"**)**
list observations where make matches the regular expression (here, records that contain a number)

list if regexm(make, "(Cad.|Chev.|Datsun)"**)**
return all observations where make contains "Cad.", "Chev." or "Datsun"

compare the given list against the first word in make

list if inlist(word(make, 1**)**, "Cad.", "Chev.", "Datsun"**)**
return all observations where the first word of the make variable contains the listed words

Transform Strings

display regexr("My string", "My", "Your"**)**
replace string1 ("My") with string2 ("Your")

replace make = **subinstr(**make, "Cad.", "Cadillac", 1**)**
replace first occurrence of "Cad." with Cadillac in the make variable

display stritrim(" Too much Space"**)**
replace consecutive spaces with a single space

display trim(" leading / trailing spaces "**)**
remove extra spaces before and after a string

display strlower("STATA should not be ALL-CAPS"**)**
change string case; see also **strupper**, **strproper**

display strtoname("1Var name"**)**
convert string to Stata-compatible variable name

display real("100"**)**
convert string to a numeric or missing value

Save & Export Data

compress
compress data in memory

save "myData.dta", **replace** Stata 12-compatible file
saveold "myData.dta", **replace version(12)**
save data in Stata format, replacing the data if a file with same name exists

export excel "myData.xls", /* */ **firstrow(variables) replace**
export data as an Excel file (.xls) with the variable names as the first row

export delimited "myData.csv", **delimiter(",") replace**
export data as a comma-delimited file (.csv)

Tim Essam (tessam@usaid.gov) • Laura Hughes (lhughes@usaid.gov)
follow us @StataRGIS and @flaneuseks

inspired by RStudio's awesome Cheat Sheets (rstudio.com/resources/cheatsheets)

geocenter.github.io/StataTraining

updated March 2016

Data Visualization with Stata 14.1 Cheat Sheet

For more info see Stata's reference manual (stata.com)

Basic Plot Syntax: **graph** <plot type> y1 y2 ... yn x **[in]** **[if],** <plot options> **by(var)** **xline(**xint**) yline(**yint**) text(**y x "annotation"**)**

title("title**") subtitle("**subtitle**") xtitle("**x-axis title**") ytitle("**y axis title**") xscale(range(**low high**) log reverse off noline) yscale(**<options>**)**

scheme(s1mono**) play(**customTheme**) xsize(**5**) ysize(**4**) saving("**myPlot.gph**", replace)**

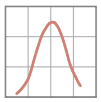
<options: marker, line, text, axis, legend, background options>

One Variable

sysuse auto, clear

Continuous

histogram mpg, **width(**5**) freq kdensity kdenopts(bwidth(**5**))**
histogram
bin(#) • width(#) • density • fraction • frequency • percent • addlabels addlabopts(<options>) • normal • normopts(<options>) • kdensity kdenopts(<options>)

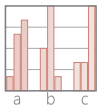


kdensity mpg, **bwidth(**3**)**
smoothed histogram
bwidth • kernel(<options> normal • normopts(<line options>)

← main plot-specific options; see help for complete set

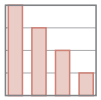
Discrete

graph bar (count**), over(**foreign**, gap(***0.5**)) intensity(***0.5**)**
bar plot **graph hbar** draws horizontal bar charts
(asis) • (percent) • (count) • over(<variable>, <options: gap(*#) • relabel • descending • reverse>) • cw •missing • nofill • allcategories • percentages • stack • bargap(#) • intensity(*#) • yalternate • xalternate

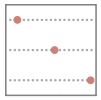


graph bar (percent**), over(**rep78**) over(**foreign**)**
grouped bar plot **graph hbar** ...
(asis) • (percent) • (count) • over(<variable>, <options: gap(*#) • relabel • descending • reverse>) • cw •missing • nofill • allcategories • percentages • stack • bargap(#) • intensity(*#) • yalternate • xalternate

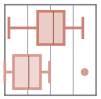
Discrete X, Continuous Y



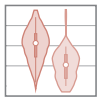
graph bar (median**) price, over(**foreign**)** **graph hbar** ...
bar plot (asis) • (percent) • (count) • (stat: mean median sum min max ...) over(<variable>, <options: gap(*#) • relabel • descending • reverse sort(<variable>)>) • cw • missing • nofill • allcategories • percentages stack • bargap(#) • intensity(*#) • yalternate • xalternate



graph dot (mean**) length headroom, over(**foreign**) m(**1, ms(S)**)**
dot plot (asis) • (percent) • (count) • (stat: mean median sum min max ...) over(<variable>, <options: gap(*#) • relabel • descending • reverse sort(<variable>)>) • cw • missing • nofill • allcategories • percentages linegap(#) • marker(#, <options>) • linetype(dot | line | rectangle) dots(<options>) • lines(<options>) • rectangles(<options>) • rwidth



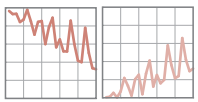
graph hbox mpg, **over(**rep78, descending**) by(**foreign**) missing**
box plot **graph box** draws vertical boxplots
over(<variable>, <options: total • gap(*#) • relabel • descending • reverse sort(<variable>)>) • missing • allcategories • intensity(*#) • boxgap(#) medtype(line | line | marker) • medline(<options>) • medmarker(<options>)



violplot price, **over(**foreign**)** ssc install violplot
violin plot over(<variable>, <options: total • missing)>) • nofill • vertical • horizontal • obs • kernel(<options>) • bwidth(#) • barwidth(#) • dscale(#) • ygap(#) • ogap(#) • density(<options>) bar(<options>) • median(<options>) • obsopts(<options>)

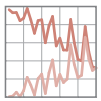
Plot Placement

Juxtapose (Facet)



twoway scatter mpg price, **by(**foreign, norescale**)**
total • missing • colfirst • rows(#) • cols(#) • holes(<numlist>) compact • [no]edgelabel • [no]rescale • [no]yrescal • [no]xrescale [no]iyaxes • [no]ixaxes • [no]iytick • [no]ixtick [no]iylabel [no]ixlabel • [no]iytitle • [no]ixtitle • imargin(<options>)

Superimpose

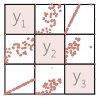


graph combine plot1.gph plot2.gph...
combine 2+ saved graphs into a single plot

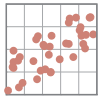
scatter y3 y2 y1 x, marker(i o i) mlabel(var3 var2 var1)
plot several y values for a single x value

graph twoway scatter mpg price in 27/74 || scatter mpg price /* */ if mpg < 15 & price > 12000 in 27/74, mlabel(make) m(i)
combine twoway plots using ||

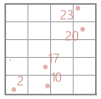
Two+ Continuous Variables



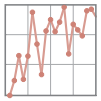
graph matrix mpg price weight, **half**
scatter plot of each combination of variables
half • jitter(#) • jitterseed(#) diagonal • [aweights(<variable>)]



twoway scatter mpg weight, **jitter(**7**)**
scatter plot
jitter(#) • jitterseed(#) • sort • cmissing(yes | no) connect(<options>) • [aweight(<variable>)]

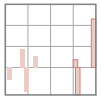


twoway scatter mpg weight, **mlabel(**mpg**)**
scatter plot with labelled values
jitter(#) • jitterseed(#) • sort • cmissing(yes | no) connect(<options>) • [aweight(<variable>)]

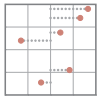


twoway connected mpg price, **sort(**price**)**
scatter plot with connected lines and symbols
jitter(#) • jitterseed(#) • sort see also **line**
connect(<options>) • cmissing(yes | no)

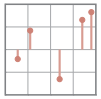
twoway area mpg price, **sort(**price**)**
line plot with area shading
sort • cmissing(yes | no) • vertical, • horizontal base(#)



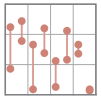
twoway bar price rep78
bar plot
vertical, • horizontal • base(#) • barwidth(#)



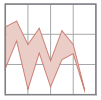
twoway dot mpg rep78
dot plot vertical, • horizontal • base(#) • ndots(#) dcolor(<color>) • dfcolor(<color>) • dlcolor(<color>) dsize(<markersize>) • dsymbol(<marker type>) dlwidth(<strokesize>) • dotextend(yes | no)



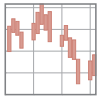
twoway dropline mpg price **in** 1/5
dropped line plot
vertical, • horizontal • base(#)



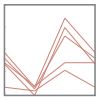
twoway rcapsym length headroom price
range plot (y_1 ÷ y_2) with capped lines
vertical • horizontal see also **rcap**



twoway rarea length headroom price, **sort**
range plot (y_1 ÷ y_2) with area shading
vertical • horizontal • sort cmissing(yes | no)



twoway rbar length headroom price
range plot (y_1 ÷ y_2) with bars
vertical • horizontal • barwidth(#) • mwidth msize(<marker size>)



twoway pcspike wage68 ttl_exp68 wage88 ttl_exp88
Parallel coordinates plot **(sysuse nlswide1)**
vertical, • horizontal

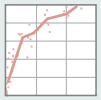
twoway pccapsym wage68 ttl_exp68 wage88 ttl_exp88
Slope/bump plot **(sysuse nlswide1)**
vertical • horizontal • headlabel

Three Variables

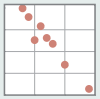
twoway contour mpg price weight, **level(**20**) crule(**intensity**)**
3D contour plot
ccuts(#s) • levels(#) • minmax • crule(hue | chue| intensity) • scolor(<color>) • ecolor (<color>) • ccolors(<colorlist>) • heatmap interp(thinplatespline | shepard | none)

regress price mpg trunk weight length turn, **nocons**
matrix regmat = **e(V)** ssc install plotmatrix
plotmatrix, mat(regmat**) color(**green**)**
heatmap mat(<variable) • split(<options>) • color(<color>) • freq

Summary Plots

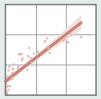


twoway mband mpg weight **|| scatter** mpg weight
plot median of the y values
bands(#)

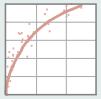


binscatter weight mpg, **line(**none**)** ssc install binscatter
plot a single value (mean or median) for each x value
medians • nquantiles(#) • discrete • controls(<variables>) • linetype(lfit | qfit | connect | none) • aweight[<variable>]

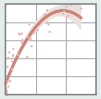
Fitting Results



twoway lfitci mpg weight **|| scatter** mpg weight
calculate and plot linear fit to data with confidence intervals
level(#) • stdp • stdf • nofit • fitplot(<plottype>) • ciplot(<plottype>) • range(# #) • n(#) • atobs • estopts(<options>) • predopts(<options>)

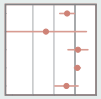


twoway lowess mpg weight **|| scatter** mpg weight
calculate and plot lowess smoothing
bwidth(#) • mean • noweight • logit • adjust

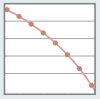


twoway qfitci mpg weight, **alwidth(**none**) || scatter** mpg weight
calculate and plot quadriatic fit to data with confidence intervals
level(#) • stdp • stdf • nofit • fitplot(<plottype>) • ciplot(<plottype>) • range(# #) • n(#) • atobs • estopts(<options>) • predopts(<options>)

Regression Results



regress price mpg headroom trunk length turn
coefplot, drop(_cons**) xline(**0**)** ssc install coefplot
Plot regression coefficients
baselevels • b(<options>) • at(<options>) • noci • levels(#) keep(<variables>) • drop(<variables>) • rename(<list>) horizontal • vertical • generate(<variable>)



regress mpg weight length turn
margins, eyex(weight**) at(**weight = (1800(200)4800)**)**
marginsplot, noci
Plot marginal effects of regression
horizontal • noci

Laura Hughes (lhughes@usaid.gov) • Tim Essam (tessam@usaid.gov)
follow us @flaneuseks and @StataRGIS

inspired by RStudio's awesome Cheat Sheets (rstudio.com/resources/cheatsheets)

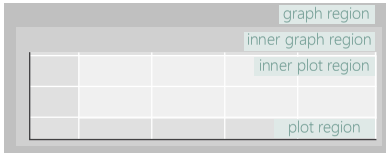
geocenter.github.io/StataTraining

updated February 2016

Plotting in Stata 14.1

Customizing Appearance

For more info see Stata's reference manual (stata.com)

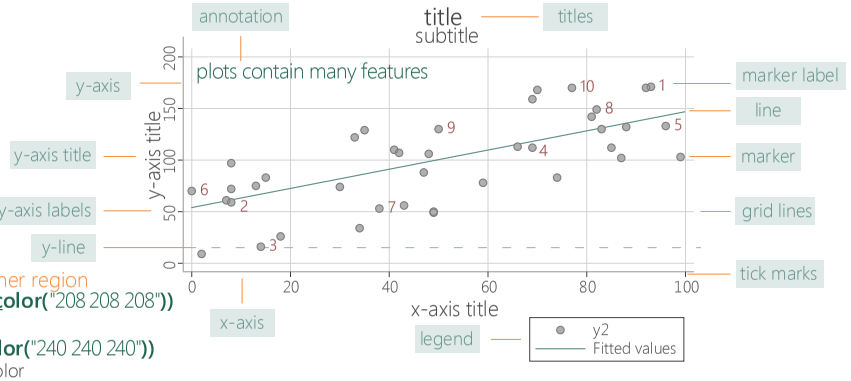


outer region inner region

scatter price mpg, **graphregion(fcolor**("192 192 192") **ifcolor**("208 208 208"))
specify the fill of the background in RGB or with a Stata color

scatter price mpg, **plotregion(fcolor**("224 224 224") **ifcolor**("240 240 240"))
specify the fill of the plot background in RGB or with a Stata color

Anatomy of a Plot



Syntax

marker <marker options>

arguments for the plot objects (in green) go in the options portion of these commands (in orange)

for example:
scatter price mpg, **xline**(20, **lwidth**(vthick))

line <line options> → xline(...) yline(...)

marker <marker options>

legend legend(region(...))

axes xscale(...) yscale(...)

tick marks grid lines xlabel(...) ylabel(...)

marker label <marker options>

annotation text(...)

titles title(...) subtitle(...) xtitle(...) ytitle(...)

axis labels xlabel(...) ylabel(...)

legend legend(...)

Color

mcolor("145 168 208") **mcolor**(none)
specify the fill and stroke of the marker in RGB or with a Stata color

mfcolor("145 168 208") **mfcolor**(none)
specify the fill of the marker

lcolor("145 168 208") **lcolor**(none)
specify the stroke color of the line or border

marker **mlcolor**("145 168 208")
tick marks **tlcolor**("145 168 208")
grid lines **glcolor**("145 168 208")

color("145 168 208") **color**(none)
specify the color of the text

marker label **mlabcolor**("145 168 208")
axis labels **labcolor**("145 168 208")

Size / Thickness

msize(medium) specify the marker size:

ehuge, vhuge, huge, vlarge, large, medlarge, medium, medsmall, small, vsmall, tiny, vtiny

lwidth(medthick) specify the thickness (stroke) of a line:

marker **mlwidth**(thin)
tick marks **tlwidth**(thin)
grid lines **glwidth**(thin)

vvvthick, vvthick, vthick, thick, medthick, medium, medthin, thin, vthin, vvthin, vvvthin, none

size(medsmall) specify the size of the text:

marker label **mlabsize**(medsmall)
axis labels **labsize**(medsmall)

28 pt. vhuge
20 pt. huge
16 pt. vlarge
14 pt. large
12 pt. medlarge
11 pt. medium
10 pt. medsmall
8 pt. small
6 pt. vsmall
4 pt. tiny
2 pt. half_tiny
1.3 pt. third_tiny
1 pt. quarter_tiny
1 pt minuscule

Appearance

msymbol(Dh) specify the marker symbol:

O, D, T, S, o, d, t, s, Oh, Dh, Th, Sh, oh, dh, th, sh, +, X, p, none, i

line, axes, grid lines **lpattern**(dash) **glpattern**(dash) specify the line pattern

solid, longdash, longdash_dot, dash, shortdash, shortdash_dot, dot, dash_dot, blank

axes **noline**
tick marks **noticks**
grid lines **nogrid** **nogmin** **nogmax**

axes **off** no axis/labels
tick marks **tlength**(2)

marker label **mlabel**(foreign)
label the points with the values of the foreign variable

axis labels **nolabels**
no axis labels

axis labels **format**(%12.2f)
change the format of the axis labels

legend **off**
turn off legend

legend **label**(# "label")
change legend label text

Position

jitter(#)
randomly displace the markers

jitterseed(#)
set seed

tick marks **xlabel**(#10, **tposition**(crossing))
number of tick marks, position (outside | crossing | inside)

marker label **mlabposition**(5)
label location relative to marker (clock position: 0 – 12)

Apply Themes

Schemes are sets of graphical parameters, so you don't have to specify the look of the graphs every time.

Using a Saved Theme

twoway scatter mpg price, **scheme**(customTheme)

help scheme entries
see all options for setting scheme properties

Create custom themes by saving options in a .scheme file

adopath ++ "~/<location>/StataThemes"
set path of the folder (StataThemes) where custom .scheme files are saved

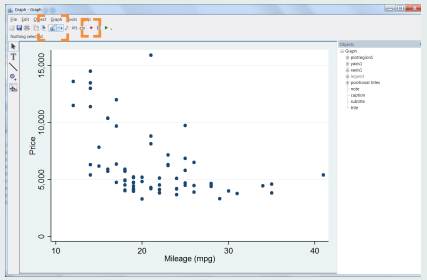
set as default scheme

set scheme customTheme, **permanently**
change the theme

net inst brewscheme, from("https://wbuchanan.github.io/brewscheme/") replace
install William Buchanan's package to generate custom schemes and color palettes (including ColorBrewer)

Using the Graph Editor

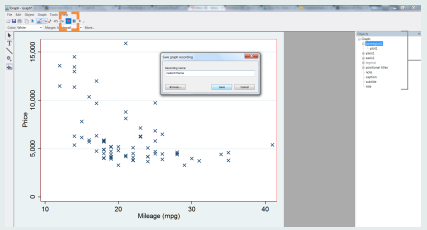
twoway scatter mpg price, **play**(graphEditorTheme)



Select the Graph Editor

Graph Editor

Click Record



Double click on symbols and areas on plot, or regions on sidebar to customize

Unclick Record

Save theme as a .grec file

Save Plots

graph twoway scatter y x, saving("myPlot.gph") **replace**
save the graph when drawing

graph save "myPlot.gph", **replace**
save current graph to disk

graph combine plot1.gph plot2.gph...
combine 2+ saved graphs into a single plot

graph export "myPlot.pdf", **as(.pdf)**
export the current graph as an image file

see options to set size and resolution

Laura Hughes (lhughes@usaid.gov) • Tim Essam (tessam@usaid.gov)
follow us @flaneuseks and @StataRGIS

inspired by RStudio's awesome Cheat Sheets (rstudio.com/resources/cheatsheets)

geocenter.github.io/StataTraining

updated June 2016

Data Analysis with Stata 14.1 Cheat Sheet

For more info see Stata's reference manual (stata.com)

Results are stored as either r -class or e -class. See Programming Cheat Sheet

Summarize Data

Examples use auto.dta (sysuse auto, clear) unless otherwise noted

univar price mpg, **boxplot** — ssc install univar
calculate univariate summary, with box-and-whiskers plot

stem mpg
return stem-and-leaf display of mpg

summarize price mpg, **detail** — frequently used commands are highlighted in yellow
calculate a variety of univariate summary statistics

ci mean mpg price, **level**(99) — for Stata 13: **ci** mpg price, **level** (99)
r compute standard errors and confidence intervals

correlate mpg price
return correlation or covariance matrix

pwcorr price mpg weight, **star**(0.05)
return all pairwise correlation coefficients with sig. levels

mean price mpg
estimates of means, including standard errors

proportion rep78 foreign
estimates of proportions, including standard errors for
e categories identified in varlist

ratio
estimates of ratio, including standard errors

total price
estimates of totals, including standard errors

Statistical Tests

tabulate foreign rep78, **chi2 exact expected**
tabulate foreign and repair record and return chi^2
and Fisher's exact statistic alongside the expected values

ttest mpg, **by**(foreign)
estimate t test on equality of means for mpg by foreign

r **prtest** foreign == 0.5
one-sample test of proportions

ksmirnov mpg, **by**(foreign) **exact**
Kolmogorov-Smirnov equality-of-distributions test

ranksum mpg, **by**(foreign) **exact**
equality tests on unmatched data (independent samples)

anova systolic drug — **webuse** systolic, clear
analysis of variance and covariance

e **pwmean** mpg, **over**(rep78) **pveffects mcompare**(tukey)
estimate pairwise comparisons of means with equal
variances include multiple comparison adjustment

Declare Data

By declaring data type, you enable Stata to apply data munging and analysis functions specific to certain data types

Time Series

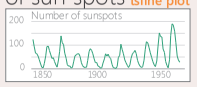
webuse sunspot, clear

tsset time, **yearly**
declare sunspot data to be yearly time series

r **tsreport**
report time series aspects of a dataset

generate lag_spot = L1.spot
create a new variable of annual lags of sun spots

tsline spot — tsline plot
plot time series of sunspots



e **arima** spot, **ar**(1/2)
estimate an auto-regressive model with 2 lags

Time Series Operators

L.	lag x_{t-1}	L2.	2-period lag x_{t-2}
F.	lead x_{t+1}	F2.	2-period lead x_{t+2}
D.	difference x_t-x_{t-1}	D2.	difference of difference x_t-x_{t-1}-(x_{t-1}-x_{t-2})
S.	seasonal difference x_t-x_{t-1}	S2.	lag-2 (seasonal difference) x_t-x_{t-2}

Useful Add-Ins

tscollap	compact time series into means, sums and end-of-period values
carryforward	carry non-missing values forward from one obs. to the next
tsspell	identify spells or runs in time series

Survival Analysis

webuse drugtr, clear

stset studytime, **failure**(died)
declare survey design for a dataset

r **stsum**
summarize survival-time data

e **stcox** drug age
estimate a cox proportional hazard model

Panel / Longitudinal

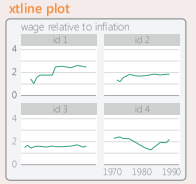
webuse nlswork, clear

xtset id year
declare national longitudinal data to be a panel

xtdescribe
report panel aspects of a dataset

r **xtsum** hours
summarize hours worked, decomposing standard deviation into between and within components

xtline ln_wage if id <= 22, **tlabel**(#3) — xtline plot
plot panel data as a line plot



e **xtreg** ln_w c.age##c.age ttl_exp, **fe vce**(robust)
estimate a fixed-effects model with robust standard errors

Survey Data

webuse nhanes2b, clear

svyset psuid [**pweight** = finalwgt], **strata**(stratid)
declare survey design for a dataset

r **svydescribe**
report survey data details

svy: mean age, **over**(sex)
estimate a population mean for each subpopulation

svy, subpop(rural)**:** mean age
estimate a population mean for rural areas

e **svy:** tabulate sex heartatk
report two-way table with tests of independence

svy: reg zinc c.age##c.age female weight rural
estimate a regression using survey weights

1 Estimate Models

stores results as e -class

regress price mpg weight, **robust**
estimate ordinary least squares (OLS) model
on mpg weight and foreign, apply robust standard errors

regress price mpg weight **if** foreign == 0, **cluster**(rep78)
regress price only on domestic cars, cluster standard errors

rreg price mpg weight, **genwt**(reg_wt)
estimate robust regression to eliminate outliers

probit foreign turn price, **vce**(robust)
estimate probit regression with
robust standard errors

logit foreign headroom mpg, **or**
estimate logistic regression and
report odds ratios

bootstrap, reps(100)**: regress** mpg /*
*/ weight gear foreign
estimate regression with bootstrapping

jackknife r(mean)**, double: sum** mpg
jackknife standard error of sample mean

Additional Models

pca ←	built-in Stata command	principal components analysis
factor		factor analysis
poisson • nbreg		count outcomes
tobit		censored data
ivregress	ivreg2	instrumental variables
diff	user-written	difference-in-difference
rd	ssc install ivreg2	regression discontinuity
xtabond	xtabond2	dynamic panel estimator
psmatch2		propensity score matching
synth		synthetic control analysis
oaxaca		Blinder-Oaxaca decomposition

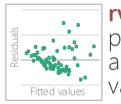
2 Diagnostics

not appropriate after robust cluster( )

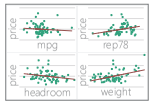
r **estat hettest** test for heteroskedasticity
ovtest test for omitted variable bias
vif report variance inflation factor

dfbeta(length)
calculate measure of influence

Type help regress postestimation plots for additional diagnostic plots



rvfplot, yline(0)
plot residuals against fitted values



avplots
plot all partial-regression leverage plots in one graph

3 Postestimation

commands that use a fitted model

regress price headroom length — Used in all postestimation examples

display _b[length] **display** _se[length]
return coefficient estimate or standard error for mpg
from most recent regression model

margins, dydx(length) — returns e-class information when post option is used
r return the estimated marginal effect for mpg

margins, eyex(length)
return the estimated elasticity for price

predict yhat if **e**(sample)
create predictions for sample on which model was fit

predict double resid, **residuals**
calculate residuals based on last fit model

test mpg = 0
r test linear hypotheses that mpg estimate equals zero

lincom headroom - length
test linear combination of estimates (headroom = length)

Estimation with Categorical & Factor Variables

more details at http://www.stata.com/manuals14/u25.pdf

Continuous Variables
measure something

Categorical Variables
identify a group to which an observations belongs

Indicator Variables
T F denote whether something is true or false

Operator	Description	Example	
i.	specify indicators	regress price **i.**rep78	specify rep78 variable to be an indicator variable
ib.	specify base indicator	regress price **ib(3).**rep78	set the third category of rep78 to be the base category
fvset	command to change base	fvset base frequent rep78	set the base to most frequently occurring category for rep78
c.	treat variable as continuous	regress price i.foreign**#c.**mpg i.foreign	treat mpg as a continuous variable and specify an interaction between foreign and mpg
o.	omit a variable or indicator	regress price **io(2).**rep78	set rep78 as an indicator; omit observations with rep78 == 2
#	specify interactions	regress price mpg c.mpg**#c.**mpg	create a squared mpg term to be used in regression
##	specify factorial interactions	regress price c.mpg**##c.**mpg	create all possible interactions with mpg (mpg and mpg^2)

Tim Essam (tessam@usaid.gov) • Laura Hughes (lhughes@usaid.gov)
follow us @StataRGIS and @flaneuseks

inspired by RStudio's awesome Cheat Sheets (rstudio.com/resources/cheatsheets)

geocenter.github.io/StataTraining

updated June 2016

Programming with Stata 14.1 Cheat Sheet

For more info see Stata's reference manual (stata.com)

1 Scalars — both r- and e-class results contain scalars

scalar x1 = 3
create a scalar x1 storing the number 3

scalar a1 = "I am a string scalar"
create a scalar a1 storing a string

Scalars can hold numeric values or arbitrarily long strings

2 Matrices — e-class results are stored as matrices

matrix a = (4\ 5\ 6)
create a 3 x 1 matrix

matrix b = (7, 8, 9)
create a 1 x 3 matrix

matrix d = b' transpose matrix b; store in d

matrix ad1 = a \ d
row bind matrices

matrix ad2 = a , d
column bind matrices

matselrc b x, c(1 3) findit matselrc
select columns 1 & 3 of matrix b & store in new matrix x

mat2txt, matrix(ad1**) saving(**textfile.txt**) replace**
export a matrix to a text file ssc install mat2txt

Displaying & Deleting Building Blocks

[**scalar** | **matrix** | **macro** | **estimates**] [**list** | **drop**] b
list contents of object b or drop (delete) object b

[**scalar** | **matrix** | **macro** | **estimates**] **dir**
list all defined objects for that class

matrix list b — list contents of matrix b
matrix dir — list all matrices
scalar drop x1 — delete scalar x1

3 Macros — public or private variables storing text

Globals — available through Stata sessions — Public

global pathdata "C:/Users/SantasLittleHelper/Stata"
define a global variable called pathdata

cd $pathdata — add a **$** before calling a global macro
change working directory by calling global macro

global myGlobal price mpg length
summarize $myGlobal
summarize price mpg length using global

Locals — available only in programs, loops, or .do files — Private

local myLocal price mpg length
create local variable called myLocal with the strings price mpg and length

summarize `myLocal' add a ` before and a ' after local macro name to call
summarize contents of local myLocal

levelsof rep78, **local**(levels)
create a sorted list of distinct values of rep78, store results in a local macro called levels

local varLab: **variable label** foreign can also do with value labels
store the variable label for foreign in the local varLab

Tempvars & Tempfiles — special locals for loops/programs

tempvar temp1 — initialize a new temporary variable called temp1
generate `temp1' = mpg^2 — save squared mpg values in temp1
summarize `temp1' — summarize the temporary variable temp1

tempfile myAuto create a temporary file to be used within a program
save `myAuto' — see also tempname

Building Blocks — basic components of programming

R- and E-Class: Stata stores calculation results in two* main classes:

r return results from general commands such as **summary** or **tabulate**

e return results from estimation commands such as **regress** or **mean**

To assign values to individual variables use:

1 Scalars — r individual numbers or strings
2 Matrices — e rectangular array of quantities or expressions
3 Macros — e pointers that store text (global or local)

* there's also s- and n-class

4 Access & Save Stored r- and e-class Objects

Many Stata commands store results in types of lists. To access these, use **return** or **ereturn** commands. Stored results can be scalars, macros, matrices or functions.

summarize price, detail
r **return** list
returns a list of scalars

mean price
e **ereturn** list
returns list of scalars, macros, matrices and functions

```
scalars:
      r(N)    =  74
      r(mean) =  6165.25...
      r(Var)  =  86995225.97...
      r(sd)   =  2949.49...
                 ...
```

Results are replaced each time an r-class / e-class command is called

```
scalars:
      e(df_r)   =  73
      e(N_over) =  1
      e(N)      =  73
      e(k_eq)   =  1
      e(rank)   =  1
```

generate p_mean = r(mean)
create a new variable equal to average of price

generate meanN = e(N)
create a new variable equal to obs. in estimation command

preserve create a temporary copy of active dataframe
restore restore temporary copy to original point — set restore points to test code that changes data

Accessing Estimation Results

After you run any estimation command, the results of the estimates are stored in a structure that you can save, view, compare, and export

regress price weight
estimates store est1
store previous estimation results est1 in memory

Use **estimates store** to compile results for later use

eststo est2: **regress** price weight mpg — ssc install estout
eststo est3: **regress** price weight mpg foreign
estimate two regression models and store estimation results

estimates table est1 est2 est3
print a table of the two estimation results est1 and est2

Exporting Results

The **estout** and **outreg2** packages provide numerous, flexible options for making tables after estimation commands. See also **putexcel** command.

esttab est1 est2, se star(* 0.10 ** 0.05 *** 0.01) label
create summary table with standard errors and labels

esttab using "auto_reg.txt", replace plain se
export summary table to a text file, include standard errors

outreg2 [est1 est2] using "auto_reg2.txt", see replace
export summary table to a text file using outreg2 syntax

Additional Programming Resources

bit.ly/statacode
download all examples from this cheat sheet in a .do file

adoupdate
Update user-written .ado files

adolist — ssc install adolist
List/copy user-written .ado files

net install package, from (https://raw.githubusercontent.com/username/repo/master**)**
install a package from a Github repository

https://github.com/andrewheiss/SublimeStataEnhanced
configure Sublime text for Stata 11-14

Loops: Automate Repetitive Tasks

Anatomy of a Loop — see also **while**

Stata has three options for repeating commands over lists or values: **foreach**, **forvalues**, and **while**. Though each has a different first line, the syntax is consistent:

```
foreach x of varlist var1 var2 var3 {
    command `x', option
    ...
}
```

objects to repeat over
open brace must appear on first line
temporary variable used only within the loop
requires local macro notation
command(s) you want to repeat can be one line or many
close brace must appear on final line by itself

Foreach: Repeat Commands over Strings, Lists, or Variables

```
foreach x in|of [ local, global, varlist, newlist, numlist ] {
    Stata commands referring to `x'
}
```

list types: objects over which the commands will be repeated

Strings

```
foreach x in auto.dta auto2.dta {
    sysuse "`x'", clear
    tab rep78, missing
}
```

same as...

loops repeat the same command over different arguments:

```
sysuse "auto.dta", clear
tab rep78, missing
sysuse "auto2.dta", clear
tab rep78, missing
```

Lists

```
foreach x in "Dr. Nick" "Dr. Hibbert" {
    display length ( "`x'" )
}
```

```
display length("Dr. Nick")
display length("Dr. Hibbert")
```

When calling a command that takes a string, surround the macro name with quotes.

Variables

```
foreach x in mpg weight {
    summarize `x'
}
```

```
foreach x of varlist mpg weight {
    summarize `x'
}
```

must define list type

- **foreach in** takes any list as an argument with elements separated by spaces
- **foreach of** requires you to state the list type, which makes it faster

```
summarize mpg
summarize weight
```

ForValues: Repeat Commands over Lists of Numbers

```
forvalues i = 10(10)50 {
    display `i'
}
```

iterator
numeric values over which loop will run

Use display command to show the iterator value at each step in the loop

```
display 10
display 20
...
```

Iterators

```
i = 10/50          ⟶ 10, 11, 12, ...
i = 10(10)50         10, 20, 30, ...
i = 10 20 to 50      10, 20, 30, ...
```

Debugging Code

set trace on (off) — see also **capture** and scalar _rc
trace the execution of programs for error checking

Putting It All Together — sysuse auto, clear

```
generate car_make = word(make, 1)
levelsof car_make, local(cmake)
local i = 1
local cmake_len : word count `cmake'
foreach x of local cmake {
    display in yellow "Make group `i' is `x'"
    if `i' == `cmake_len' {
        display "The total number of groups is `i'"
    }
    local i = `++i'
}
```

pull out the first word from the make variable
calculate unique groups of car_make and store in local cmake
store the length of local cmake in local cmake_len
define the local i to be an iterator
tests the position of the iterator, executes contents in brackets when the condition is true
increment iterator by one

Tim Essam (tessam@usaid.gov) • Laura Hughes (lhughes@usaid.gov)
follow us @StataRGIS and @flaneuseks

inspired by RStudio's awesome Cheat Sheets (rstudio.com/resources/cheatsheets)

geocenter.github.io/StataTraining

updated June 2016