Analysen politikwissenschaftlicher Datensätze mit Stata

JOHANNES GUTENBERG UNIVERSITÄT MAINZ

Sitzung 3: Syntax und Variablen

Vorbereitung

- bitte starten Sie Stata (z:\profile.do)
- bitte laden Sie z:\daten\kohler-kreuter\data1.dta

Bausteine eines Stata-Befehls

- Bausteine
  - Präfix
  - Befehl selbst
  - Variablenliste
  - Nummernliste
  - Gewichtungsanweisung
  - if-Klausel
  - in-Klausel
  - Optionen
- Bausteine können verpflichtend oder optional sein
- Beispiel:
  - `help summarize` führt zu:
  - `[by varlist:] summarize [varlist] [weight] [if exp] [in range] [,[detail|meanonly] format separator(#) ]`

Befehl selbst

- interne und externe Befehle
- Befehle können abgekürzt werden

Variablenliste

- Kann häufig weggelassen werden
  - in einigen Fällen: Ausführung für alle Variablen
  - in anderen Fällen: Wiederholung des letzten Befehls
  - _all oder * steht für alle Variablen
- Abkürzungen
  - Variablen können bis zur Eindeutigkeit abgekürzt werden
  - * dient als Jokerzeichen (`d *9*; d *94*; d np*`)
  - mit dem Bindestrich wird ein Bereich von Variablen ausgewählt, die hintereinander stehen (`d persnr-zimmer`)
- Bei Modellbefehlen muß zuerst die abhängige Variable genannt werden
  - `help regress`
  - `regress depvar [varlist]`

in-Bedingung

- Beschränkt Befehl anhand der internen Ordnungsnummer auf bestimmte Fälle
- # (erster Fall) / # (letzter Fall)
  - erster Fall muß vor letztem Fall stehen
  - Zählung von Beginn der Datei
  - Minus-Zeichen: Zählung vom Ende der Datei
  - f steht für ersten Fall im Datensatz, l für letzten Fall im Datensatz

if-Bedingung

- wählt Fälle aufgrund logischer Bedingungen aus
- Relationale Operatoren
  - > größer
  - < kleiner
  - >= größer oder gleich
  - <= kleiner oder gleich
  - == Gleichheit
  - ~= Ungleichheit; alternativ: !=
- Logische Verknüpfungen
  - ~ nicht; alternativ: !
  - & und
  - | oder

if-Bedingung

- `help operators`
- Vorsicht: “missing” wird intern als sehr große Zahl gespeichert
  - `numlabel bil, add`
  - `tab bil`
  - `tab bil if bil>6`
  - `tab bil if bil >6 & bil ~=.`

Funktionen

- Arithmetische Operatoren (`+ - * /`) können verwendet werden, auch in if-Bedingungen
- `tab hst if hhein/hhgr >=2 * eink`
- Außerdem gibt es eine Unzahl von echten Funktionen, z.B. Quadratwurzel=`sqrt()`, Absolutwert=`abs()`, die an vielen Stellen verwendet werden können
- `display sqrt(2)+3/5`; `disp sqrt(2 + 3/5)`
- `help functions`
- Oft ist es sinnvoll, die Hierarchie durch Klammern zu verdeutlichen oder zu verändern
  - `disp sqrt(2 + (3/5))`
  - `disp sqrt((2 + 3) /5)`

Gewichtung

- Gewichtungsanweisung kann an viele Befehle angehängt werden
- Verschiedene Typen:
  - Häufigkeitsgewichte
  - analytische Gewichte
  - Sampling-Gewichte
  - (Importance-Gewichte)

Häufigkeitsgewichte

- Ein Fall im Datensatz steht für eine Vielzahl von identischen Fällen in der Wirklichkeit
- häufig benötigt bei Analysen auf Grundlage (amtlicher) Tabellen
  - summ gebjahr
  - preserve
  - use z:\daten\kohler-kreuter\freqwe, replace
  - summ gebjahr
  - list
  - summ gebjahr [fw=n]
  - restore

Analytische Gewichte

- Nützlich für Aggregatanalysen
- Üblicherweise: Ein Fall im DS repräsentiert einen Mittelwert über viele Beobachtungen
- Je weniger Beobachtungen ein Fall repräsentiert, desto größer ist seine Varianz
- Fälle, die auf vielen Beobachtungen basieren, sind wichtiger, da sie mehr Information beinhalten
- Analytische Gewichte berücksichtigen dies (vgl. Übung bei Kohler und Kreuter)

Sampling-Gewichte

- Umfragedaten basieren meist nicht auf einfacher, sondern auf mehrstufiger Zufallsauswahl
  - verzerrte Punktschätzer
  - (meist) zu optimistische Standardfehler
- „Normale“ Gewichtung korrigiert Punktschätzer, aber
  - Fälle stehen nicht für identische Fälle
  - Fälle mit hohem Gewicht sind nicht besonders „zuverlässig“
  - Sampling-Gewichte können Punktschätzung und Standardfehler korrigieren

Importance-Gewichte

- Keine statistisch definierte Bedeutung

Optionen

- modifizieren Befehle
- durch Komma abgetrennt
- schließen sich teilweise wechselseitig aus

Nummernliste

- Liste von Zahlen, die als Zahlen *verstanden* werden
- 1,2,3,4
  - `1,2,3,4` (nach Möglichkeit vermeiden) oder
  - `1 2 3 4` oder
  - `1/4`
- 8, 6, 4, 2
  - `8 6 to 2` oder
  - `8 6: 2` oder
  - `8(-2)2`
- Plus weitere Möglichkeiten: `help numlist`

by-Präfix

- führt ein Kommando für Subgruppen aus, die durch eine oder mehrere Variablen definiert sind
- Daten vorher sortieren oder `bysort` verwenden
- Nicht verwechseln mit spezifischer *by-Option* mancher Befehle
  - `bysort sex: tabstat eink`
  - `tabstat eink,by(sex)`

Makros

- Stata kann auf viele Weisen programmiert werden
- einfache Möglichkeit: „lokale Makros“
  - fungieren als „Behälter“, speichern numerische Werte oder Zeichenfolgen
  - lokal, d.h. nur innerhalb des laufenden Programms (Eingabesitzung oder do-File) gültig
  - `local drei=3`
    - definiert das Makro „drei“
    - weist ihm den numerischen Wert 3 zu
    - Kann überall verwendet werden: `display `drei' *4`
  - Akzent und Apostroph neben Backspace/Return
  - In profile.do an F4 / F5 gebunden

Schleifen

- stumpfsinnige Arbeiten schnell und möglichst fehlerfrei erledigen
- Wichtigstes Kommando: foreach
- Für jeden Wert aus einer Liste
  - setze Makro x auf diesen Wert {
  - für eine Reihe von Anweisungen aus
  - greife dabei evtl. auf x zurück
  - }

Beispiele

- Geschlecht rekodieren: `gen female=sex-1`

```
foreach variable of varlist np9501-
  np9507 {
reg `variable' female
}
```

- führt einen ansonsten identischen Regressionsbefehl mit sechs verschiedenen abhängigen Variablen aus

Beispiele

- Zehn leere Variabeln mit Namen von a1 bis a10 erzeugen:

```
foreach var of newlist a1-a10 {
gen `var'=.
}
```

Beispiele

- Eine Nummernliste durchlaufen

```
foreach i of numlist  27(9)133{
displ `i'
}
```

- Kein besonders sinnvolles Beispiel; jedoch häufig benötigt, um Funktionen zu plotten oder andere Berechnungen durchzuführen

Beispiele

- Elf (zunächst leere) kategoriale Altersvariablen erzeugen

```
foreach kat in 18bis24 25bis29
  30bis34 35bis39 40bis44 45bis49
  50bis54 55bis59 60bis64 65bis69
  70plus {
     gen altkat`kat'=.
     }
```

Zusammenfassung

- `foreach x of varlist` um eine Variablenliste abzuarbeiten
- `foreach x of newlist` um eine zu erzeugende Variablenliste abzuarbeiten
- `foreach x of numlist` um eine Nummernliste abzuarbeiten (evtl. `forvalues`)
- `foreach x in ...` um eine beliebige Aufzählung abzuarbeiten
- `{` am Ende des foreach Befehls
- Schleifenbefehle in eigenen Zeilen
- Beenden mit `}` in separater Zeile

Ergebnisse von Statistik-Kommandos

- Einfache Statistik-Befehle (`summarize`) lassen ihre internen Resultate im Speicher zurück
- Auf diese kann durch den Ausdruck `r()` zugegriffen werden
- Analog kann mit `e()` auf die Resultate von Modell-Befehlen (`regress`) zurückgegriffen werden
- Resultate
  - werden durch neue Befehle überschrieben
  - können in Makros oder Variablen gespeichert oder in Ausdrücken verwendet werden

Beispiel: Einkommen zentrieren

- `summ eink`
- Was ist verfügbar: `return list`
- `gen zenteink=eink-r(mean)`
- `summ zenteink`
- Wenn Sie mit Zwischenresultaten arbeiten wollen
  - nie abtippen
  - immer r() verwenden

Beispiel Konfidenzintervall

- `summ eink`
- Standardfehler berechnen: Wurzel aus (Varianz / Fallzahl)
- `local stdfhl=sqrt(r(Var)/r(N))`
- `local konfup=r(mean)+1.96*`stdfhl'`
- `local konflow=r(mean)-1.96*`stdfhl'`
- `displ `konfup'`
- `displ `konflow'`

generate Befehl

- erzeugt eine Variable, der zugleich ein Wert zugewiesen werden muß (z.B. missing)
- Variablennamen
  - bis zu 31 Zeichen lang
  - A-Z, a-z, 0-9, _
  - Müssen mit Buchstaben oder _ beginnen
  - verboten: _all, _b, _byte, _coef ...
  - vermeiden: Beginn mit _, e, Beginn mit I
  - am besten: kurze Namen aus Kleinbuchstaben/Ziffern

replace Befehl

- Verändert bestehende Variablen
- Ansonsten gleiche Syntax wie generate

Beispiele

- Pro-Kopf Haushaltseinkommen gen `pheink=hhein/hhgr`
- logarithmiertes Einkommen gen `logeink=log(eink)`
- Einkommen nach Miete: gen `raweink=hhein-miete`
- Zufallsvariable (Wertebereich 0-1) gen `r=uniform()`
- standardnormalverteilte Zufallsvariable `gen x=invnorm(uniform())`

Beispiele

- Einen Dummy für das Bundesland Bremen erzeugen
  - `tab bul`
  - `d bul`
  - `label list bul`
  - `gen bremen = bul==4`
  - `tab bremen`
- Logisch wahre Aussagen haben den numerischen Wert 1, logische falsche den Wert 0
- Deshalb kann z.B. das Ergebnis eines Vergleichs einer Variablen zugewiesen werden!

Beispiele

- Vorsicht mit missings
  - `gen hanse = ((bul == 2) | (bul ==4)) if bul ~=.`
  - `tab hhein`
  - `gen hhein3000 = hhein >=3000`
  - `replace hhein3000=. if hhein ==.`
- Oft ist es sinnvoll, eine neue Variable auf missing zu setzen und dann schrittweise zu verändern
  - `gen hanse= .`
  - `replace hanse =1 if bul==2`
  - `replace hanse =1 if bul==4`

by, _n und _N

- ermöglichen sehr elegante Rekodierungen
- Wie viele Interviews hat jeder Interviewer geführt?
- Wer hat wen interviewt?
  - `sort intnr`
  - `list persnr intnr`
  - benötigt wird eine Variable, die für jeden Interviewer mit intnr die Zahl der Interviews enthält
  - `by intnr: gen intcount=_N`

Was ist passiert?

- _n enthält für jede Variable ihre laufende Nummer im Datensatz
  - `gen lfd=_n`
  - `list lfd persnr intnr`
- Nach dem by-Präfix ist _n die laufende Nummer in der aktuellen Kategorie
  - `by intnr: gen intlfd=_n`
  - `list lfd persnr intlfd intnr`
- _N ist der höchste Wert, den _n annimmt (im Datensatz oder in der Kategorie)
  - `display _N`

Explizite Subskripte

- Beschränken generate/replace Befehle auf Fall mit einer bestimmten Nummer
  - innerhalb Kategorie oder
  - Datensatz
- Nützlich für hierarchisch strukturierte Daten

Beispiele

- Haushaltseinkommen als Summe einzelner Einkommen
  - `sort hhnr`
  - `by hhnr: gen hheink=sum(eink)`
  - `by hhnr: replace hheink=hheink[_N]`
- Beruf des HVs wird allen HH-Mitgliedern zugewiesen
  - `sort hhnr hast`
  - `by hhnr: gen bst_hv= bst[1] if hst[1]==1 & (bst == … | bst== …)`

Spezielle Rekodierungsbefehle

- recode
  - mächtig, aber etwas langsam
  - kann bestehende Variablen verändern oder in neue Variablen umkodieren
  - im zweiten Fall können gleich Werte-Labels vergeben werden
  - was nicht unter Regeln fällt, bleibt unverändert (z.B. missings)
  - Regeln: /, min, max, miss, nonmiss, else *
  - `recode bul (1/4 = 1 nord) (5/7=2 west) (8/9=3 sued) (10/max 0 = 4 ost), gen(region)`
- egen (extended generate)
  - faßt Erweiterungen zusammen
  - Funktionen nur mit egen verwendbar
  - einige weitere Einschränkungen
  - `help egen`

Hausaufgabe

- Schreiben Sie unter Verwendung von muster.do ein do-File, das
  - den kumulierten ALLBUS-Datensatz z:\daten\allbus1980-2000.dta öffnet
  - aus dem Erhebungsjahr und dem Geburtsjahr der Befragten deren Alter in Jahren errechnet
  - aus den vier ursprünglichen Items eine dreistufige Variante des Inglehart-Indexes erzeugt
    - vgl. zur Konstruktion http://www.politik.uni-mainz.de/kai.arzheimer/Lehre-Stata/InglehartIndex.html
    - es gibt mehrere Möglichkeiten;
      - die ersten beiden Präferenzen genügen für eine eindeutige Einstufung
      - möglichst wenig missings erzeugen
    - Zur Kontrolle enthält der Datensatz bereits eine Index-Variable
- Schicken Sie die Lösung bis zum 26. Mai an do-files@politik.uni-mainz.de; verwenden Sie das bekannte Schema