

SOEP-spezifische Problemlösungen mit Stata

Ulrich Kohler
Universität Mannheim

10. Juli 2001

Slide 1

Die Datei „profile.do“

Der Do-File „profile.do“ wird beim Aufruf von Stata automatisch ausgeführt. Für die Arbeit mit dem SOEP empfiehlt es sich, in „profile.do“ einen *globalen* Makro für das Verzeichnis mit den SOEP-Daten zu setzen:

```

snip ⌘
global soepdir Pfad/zu/den/SOEP/Dateien
snip ⌘

```

Die Datei „profile.do“ sollte im Arbeitsverzeichnis stehen oder in einem Verzeichnis dass von Stata durchsucht wird. Beispiele sind „c:/ado/personal“ (Windows) oder „~/bin“ (Unix).

Die Definition eines „SOEP-Verzeichnisses“ hat folgende Vorteile:

- Sparen von Schreiarbeit: In allen Befehlen, in denen „Pfad/zu/den/SOEP/Dateien“ angegeben werden muss, kann nun „\$soepdir“ angegeben werden.
- Austauschbarkeit von Do-Files: Um Do-Files auf unterschiedlichen Rechnern laufen zu lassen genügt es, den globalen Makro auf dem jeweiligen Rechner zu setzen.

Die Datei „profile.do“ kann unbegrenzt viele Befehle enthalten. Hier ist ein Beispiel für eine typische „profile.do“-Datei:

```

profile.do

* Usage
set matsize 100
set memory 30m
quietly conren style 3

* Function-Keys
global F4 "dir *.dta;"
global F5 "dir *.do;"
global F6 "codebook;"
global F9 "do _marked;"

```

```
* Directories
global soepdir ~/data/soep
global allbdir ~/data/allbus/dta
global politdir ~/data/polit
exit
```

Mit „set matsize 100“ setzt man die maximale Größe von bearbeitbaren Matrizen auf 100×100 . Die Voreinstellung ist 40×40 , „set memory 30m“ dient zur Festlegung des Arbeitsspeichers, der Stata zur Verfügung gestellt wird. Mit „set conren 3“ werden die Farben für Stata für Unix (Console) festgelegt.

Mit „global“ werden so genannte globale Makros erstellt. Diese werden vor allem innerhalb von Programmen benötigt. Der Befehl kann aber auch dazu genutzt werden, um die Funktionstasten mit beliebigen Stata-Befehlen zu belegen. Zum Beispiel wird mit „global F4 ”dir*.dta”“ die Funktionstaste „F4“ mit dem Befehl „dir *.dta“ belegt. Dies bewirkt eine Auflistung der Stata-Systemdatensätze im Arbeitsverzeichnis.

Das Semikolon in den einzelnen Befehlen bewirkt, dass der Befehl unmittelbar durch die Funktionstaste ausgeführt wird. Ohne das Semikolon muss der Befehl noch mit der Eingabetaste abgeschlossen werden.

Die Funktionstaste „F9“ wird oben mit dem Befehl „do _marked“ belegt. Dies macht dann Sinn, wenn der für *Do-Files* verwendete Editor so eingestellt wurde, dass markierte Bereiche mit der „F9“-Taste unter dem Namen „_marked.do“ abgespeichert wird.

Gib Gas mit use

Das volle Syntax-Diagramm von „use“ lautet:

```
use [varlist] [if exp] [in range] using filename [, clear nolabel ]
```

Durch exakte Spezifikation der benötigten Daten lässt sich das Einlesen der Daten beschleunigen.

Slide 2

```
. set rmsg on
. use $soepdir/ppfad, clear
. keep hhnr persnr ?netto
. use hhnr persnr ?netto using $soepdir/ppfad, clear
.
. use $soepdir/ppfad, clear
. keep if psample==1
. use $soepdir/ppfad, clear, if psample==1
```

Die Optimierung der Geschwindigkeit sollte vor allem beim Schreiben von eigenen Programmen (*Ados*) beachtet werden.

Beim Retrieval von SOEP-Daten müssen oft viele Datensätze nacheinander eingelesen werden. Der Geschwindigkeitsgewinn in *Do-Files* kann dann groß werden.

Beispiel: Folgende *Do-Files* enthalten die „use“-Befehle aus dem Beispiel-Retrieval im *Desktop-Compagnion*. In diesem Beispiel werden insgesamt 13 „use“-Befehl verwendet.

In *retrival1.do* werden mit „use“ nur diejenigen Variablen geladen, welche benötigt werden. Die Angabe von „clear“ erfolgt als Option von „use“, nicht als eigenes Kommando.

```
. do retrival1

snip >⌘
end of do-file
r; t=1.11 14:22:34
```

In *retrival2.do* wird „clear“ als eigenes Kommando vor jeden neuen „use“-Befehl eingesetzt:

```
. do retrival2
```

```
snip >␣  
end of do-file  
r; t=1.33 14:32:02
```

In *retrival3.do* wird zunächst der gesamte Datensatz geladen, anschließend die benötigten Variablen mit „keep“ im Datensatz belassen.

```
. do retrival3
```

```
snip >␣  
end of do-file  
r; t=2.64 14:33:50
```

merge

Der Befehl „merge“ spielt einem im Arbeitsspeicher befindlichen Datensatz (*Master-Daten*) neue Variablen eines Datensatzes auf der Festplatte (*Using-Daten*) zu.

Slide 3

Bei der Arbeit mit dem SOEP wird man „merge“ normalerweise mit Schlüsselvariablen verwenden. Bei Schlüsselvariablen werden jeder Beobachtung der *Master-Daten* die Werte derjenigen Variablen zugespielt, welche dieselben Werte auf der Schlüsselvariablen aufweisen.

Master-Daten und *Using-Daten* müssen nach der Schlüsselvariable sortiert sein.

Personen : Personen

Slide 4

```

                                                                    _retrival4.do
1  use persnr atatzzeit using $soepdir/apgen, clear          /* Using 1 */
2  sort persnr
3  save apgen, replace
4  use persnr btazeit using $soepdir/bpgen, clear          /* Using 2 */
5  sort persnr
6  save bpgen, replace
7  use hhnr ahnr bhnr persnr anetto bnetto                  /* Master
8  */ using $soepdir/ppfad if anetto==1 & bnetto==1, clear
9  sort persnr
10 merge persnr using apgen, nokeep                          /* Merge */
11 assert _merge==3
12 drop _merge
13 sort persnr
14 merge persnr using bpgen, nokeep
15 assert _merge==3
16 drop _merge
17 for any apgen bpgen: erase X.dta                          /* Putzen */
18 exit
```

Es ist vorteilhaft zuerst die *Using*-Daten für das *Merging* vorzubereiten. Hierdurch spart man sich einen „save“- und einen „use“-Befehl.

Zur Vorbereitung des *Merging* werden zunächst die benötigten Variablen der *Using*-Daten ausgewählt. Danach wird nach der Schlüsselvariablen sortiert. Eine andere Reihenfolge kostet Zeit, da dann größere Datensätze sortiert würden.

Die *Using*-Daten werden einzeln aufgerufen, bearbeitet und danach temporär gespeichert.

Üblicherweise werden die Dateien „ppfad“ oder „hpfad“ als *Master*-Daten verwendet. Mit ihrer Hilfe wird die grundlegende Struktur des späteren Datensatzes vorgegeben. Im obigen Beispiel wird ein *Balanced* Paneldesign der Wellen *a* und *b* verwendet.

Die Struktur des „merge“-Befehls ist einfach. Nach dem Befehlsnamen erfolgt die Angabe der Schlüsselvariablen. Danach der „using“-Teil, in dem der Name der *Using*-Daten angegeben wird.

Die Option „nokeep“ sorgt dafür, dass Beobachtungen der *Using*-Daten, die nicht in den *Master*-Daten enthalten sind, nicht in den Datensatz aufgenommen werden.

Jeder „merge“-Befehl erzeugt die Variable *_merge*. In ihr wird abgelegt, ob eine Beobachtung nur in den *Master*-Daten, nur in den *Using*-Daten oder in beiden Da-

teien enthalten ist. Die Variable muss vor dem nächsten „merge“-Befehl gelöscht werden.

Im Beispiel sollte man erwarten, dass beide „merge“-Befehle zu einem Datensatz führen, in dem nur Beobachtungen aus beiden Dateien enthalten sind. Mit „assert“ wird dies überprüft. Der Befehl überprüft beliebige Ausdrücke. Falsche Ausdrücke erzeugen eine Fehlermeldung.

Jeder „merge“-Befehl löscht den Eintrag über die Sortierreihenfolge des Datensatzes. Deshalb muss der Datensatz vor jedem „merge“-Befehl erneut sortiert werden.

Nach abgeschlossenem *Merging* sollten die temporären Dateien gelöscht werden. Oben geschieht das mit Hilfe des „erase“-Befehls. Dieser wurde innerhalb einer „for“-Schleife gesetzt.

Slide 5

Personen : Haushalt

```
retrival5.do
```

```

1  use hhnr hhnrakt amieteg using $soepdir/ahgen, clear
2  sort hhnr hhnrakt
3  save ahgen, replace
4  do retrival4                      /* rerun previous as master */
5  ren ahhnr hhnrakt
6  sort hhnr hhnrakt
7  merge hhnr hhnrakt using ahgen, nokeep          /* Merge */
8  assert _merge==3
9  drop _merge
10 erase ahgen.dta                          /* Putzen */
11 exit

```

Enthalten die *Master*-Daten mehrere Beobachtungen für jede Ausprägung der Schlüsselvariablen, so wird jeder dieser Beobachtungen die Information der *Using*-Daten zugespielt.

Oben wird jeder Person aus dem Personen-Datensatz die ihm zugehörige Haushaltsinformation zugespielt.

Wird einem Haushaltsdatensatz ein Personendatensatz zugespielt ist das Ergebnis prinzipiell dasselbe, d.h. der Haushaltsdatensatz wird zum Personendatensatz.

```
retrival6.do
```

```

1  do retrival4                      /* rerun previous as using */
2  ren ahhnr hhnrakt
3  sort hhnr hhnrakt persnr
4  save ul, replace
5  use hhnr hhnrakt amieteg using $soepdir/ahgen, clear
6  sort hhnr hhnrakt
7  merge hhnr hhnrakt using ul, nokeep          /* Merge */
8  assert _merge==3 | _merge==1
9  drop _merge
10 erase ul.dta                          /* Putzen */
11 exit

```

(In Beispiel ist enthält der Haushaltsdatensatz einige Haushalte, die im Personenda-

tensatz nicht enthalten sind. Der Personendatensatz wird mit *retrival4.do* im *Balanced* Paneldesign gebildet. Der Haushaltsdatensatz enthält dagegen auch Haushalte, die in Welle b nicht mehr teilgenommen haben.)

Fehlerkontrolle

Der Befehl „assert“ wird zur Fehlerkontrolle verwendet. Der Befehl überprüft beliebige Ausdrücke. Falsche Ausdrücke erzeugen eine Fehlermeldung.

Manchmal sollen Do-Files nicht unmittelbar durch eine fehlgeschlagene *Assert*-Kontrolle abgebrochen werden.

Slide 6

```
retrival6a.do
snip >
merge hhnr hhnrakt using ul, nokeep           /* Merge */
capture assert _merge==3
if _rc ~= 0 {
    tab _merge
    erase ul.dta
    exit
}
drop _merge
erase ul.dta                                 /* Putzen */
```

Hier wird mit „assert“ untersucht, ob der neue Datensatz wirklich nur Beobachtungen enthält, die zuvor in den *Master*- und in den *Using*-Daten enthalten waren. Ist dies nicht der Fall wird eine Häufigkeitsauszählung der Variable *_merge* vorgenommen und die temporäre Datei *ul.dta* gelöscht. Erst danach wird der *Do-File* beendet.

Der Befehl „capture“ fängt den *Return-Code* von Befehlen ab und speichert ihn im Makro *_rc*. Der *Return-Code* ist Null, wenn der Befehl fehlerfrei war. Ist der *Return-Code* nicht Null, so führt der Befehl zu einer Fehlermeldung.

Mit „if“ (als Befehl) wird eine Verzweigung eingeleitet. Hier startet die Verzweigung, wenn der vorangegangene Befehl eine Fehlermeldung ergab.

Bitte verwechseln Sie nicht die *If*-Verzweigung mit der *If*-Bedingung (siehe Kohler/Kreuter, S. 367).

Motivation zur Schleifenbildung

In Datenretrivals aus dem SOEP muss dieselbe Befehlssequenz oft mehrmals hintereinander durchgeführt werden. Ein Beispiel ist diese Sequenz aus dem *Desktop-Companion*, die insgesamt 12 mal ausgeführt wird.:

Desktop-Companion Vers. 3.0, S. 105

Slide 7

```
snip >
/* ----- merge APGEN ----- */
replace hhnrakt=ahhnr;
sort    hhnr hhnrakt persnr;
merge   hhnr hhnrakt persnr
using   C:/TEMP/apgen;
drop    if _merge==2;
drop    _merge;
erase   C:/TEMP/apgen.dta;
snip >
```

Mit Schleifen lässt sich hier einige Arbeit sparen.

Im Beispiel für Datenretrivals aus dem Desktop-Companion werden insgesamt 12 Using-Daten zu den Master-Daten zugefügt.

apgen	ah	ap
bpgen	bhbrutto	bp
cpgen	cpbrutto	cp
mpgen	mpbrutto	mp

Für jede dieser Dateien muss obige Sequenz wiederholt werden, wobei alle Haushaltsdaten nach *hhnr hhnrakt* und alle Personendaten nach *hhnr hhnrakt persnr* sortiert. Es ist darum am einfachsten, eine Schleife für die Personendaten und eine Schleife für die Haushaltsdaten zu bilden.

Zum Aufbau der Schleife benötigt man 3 Werkzeuge:

- Lokale Makros
- Die *String*-Funktion „substr(s,n1,n2)“
- Den „foreach“-Befehl

Diese werden auf den folgenden Folien erläutert.

Lokale Makros

In ein lokales Makro werden *Zeichenketten* gespeichert, die immer wieder zur Verfügung stehen. Bei der *Definition* des lokalen Makros wird diesen Zeichen ein Makroname zugewiesen. Nach der Definition können Sie an Stelle der Zeichen den Makronamen eingeben.

Slide 8

```
. local a "eink bdauer gebjahr"  
. display "`a'"  
. local a dir *.  
. local b dta  
. local c do  
. `a'`b'  
. `a'`c'  
. local b `a'`b'  
. `b'
```

Eine ausführlichere Beschreibung von lokalen Makros findet sich in Kohler/Kreuter, S. 341-344.

substr(s,n1,n2)**Slide 9**

Die Funktion „substr(s,n1,n2)“ ist eine der *String*-Funktionen von Stata. Sie gibt aus einem beliebigen Text *s* die Zeichenkette Länge *n2* zurück, wobei bei *n1* begonnen wird.

```
. display substr("123456789",2,3)
. display substr("123456789",5,.)
. display substr("123456789",.,6)
. display substr("Die SOEP-Gruppe am DIW",5,4)
. display substr("apgen",1,1)
```

String-Funktionen können überall eingesetzt werden, wo Funktionen erlaubt sind. Den vollen Vorrat von *String*-Funktionen finden Sie unter

```
. help functions
```

Neben den allgemeinen *String*-Funktionen gibt es eine Reihe von Funktionen für *Strings*, die nur im Zusammenhang mit der Definition lokaler Makros einsetzbar sind. Diese werden unter den *Extendend Macro Functions* in [P] „macro“ beschrieben.

foreach

Mit „foreach“ werden die Befehle innerhalb der Klammern für jedes Element einer Liste, eines lokalen Makros, einer Variablenliste oder einer Nummernliste ausgeführt.

Slide 10

```
. foreach wort in Die SOEP-Gruppe am DIW {
2. di as res "`wort'"
3.}

. local x "Die SOEP-Gruppe am DIW"
. foreach wort of local x {
2. di as res "`wort'"
3. }

. foreach var of varlist _all {graph var}
. foreach nummer of numlist 2(2)100 {di as res `nummer'}
```

Beachte: „foreach Wort *in* ...“ und „foreach Wort of local ...“

Der „foreach“-Befehl existiert erst seit Stata 7. Er ersetzt für die meisten Anwendungsfälle den Befehl „while“. Beginn, Ende und Schrittweite von Schleifen werden mit ihm weitgehend automatisch festgelegt.

Mit der Syntax von Stata 6 würden diese Befehle wie folgt lauten:

```
tokenize "Die SOEP-Gruppe am DIW"
while "`1'" ~= "" {
    di as res "`Wort'"
    macro shift
}
```

```
local x "Die SOEP-Gruppe am DIW"
tokenize "`x'"
while "`1'" ~= "" {
    di as res "`1'"
    macro shift
}
```

```
unab varlist: _all
tokenize "`varlist'"
while "`1'" ~= "" {
    graph `1'
    macro shift
}
```

```
local i 2
while i <= 100 {
    di as res `i'
    local i = `i' + 1
}
```

An Stelle von

```
. foreach number of numlist 2(2)100 {di as res `nummer'}
```

lässt sich auch der syntaktisch einfachere Befehl „forany“ verwenden:

```
. forvalues i=2(2)100 {di as res `i'}
```

Die Befehle „foreach“ und „forany“ sollten nicht mit „for“ verwechselt werden. Bei „foreach“ und „forany“ handelt es sich um interne Stata-Befehl. Innerhalb der Klammern können (fast) beliebig viele Befehle ausgeführt werden und beliebig viele weitere Schleifen gestartet werden. Bei „for“ handelt es sich um einen externen Stata-Befehl (*Ado*). Die Zahl der Befehle innerhalb einer *For*-Schleife sowie die Möglichkeit weitere Schleifen zu starten sind begrenzt.

Prinzipiell sollten innerhalb von *Do-Files*, auf jeden Fall aber innerhalb von *Ado-Files*, die Befehle „foreach“ oder „forany“ an Stelle von „for“ verwendet werden. Wesentliche Gründe sind die höheren Rechengeschwindigkeiten und die bessere Lesbarkeit der Dateien.

Schleifenbildung

Beispiel der Schleife für den „merge“ der Personendaten.

retrival7.do

```

snip >%
gen hhnrakt = .
foreach file in apgen ap bpgen bp cpngen cp mpngen mp local pfiles {
    local wave = substr("`file'",1,1)
    replace hhnrakt = `wave'hhnr
    sort hhnr hhnrakt persnr
    merge hhnr hhnrakt persnr using `file', nokeep
    assert _merge==3 | _merge==1
    drop _merge
    erase `file'.dta
}
snip >%

```

Slide 11

Verwendet wird `foreach` mit einer „anylist“. Hier werden die Dateinamen der Dateien genannt, die zusammengeführt werden sollen.

Die Dateinamen stimmen hier mit den Original-SOEP-Dateinamen überein. Es handelt sich jedoch nicht um die Original-SOEP-Dateien sondern um die aus diesen gewonnenen temporären Dateien (siehe *retrival7.do*).

Die Schleife für die Haushaltsdaten lautet entsprechend. Hier wird jedoch lediglich nach *hhnr* und *hhnrakt* sortiert

retrival7.do

```

snip >%
foreach file in ah bhbrutto chbrutto mhbrutto {
    local wave = substr("`file'",1,1)
    replace hhnrakt = `wave'hhnr
    sort hhnr hhnrakt
    merge hhnr hhnrakt using `file', nokeep
    assert _merge==3 | _merge==1
    drop _merge
    erase `file'.dta
}
snip >%

```

append

Mit „append“ werden Beobachtungen *unten* an den Datensatz angehängt. Der Befehl kann nützlich sein, wenn man Daten im „langen“ Format benötigt:

Slide 12

```
. use persnr bfamstd using $soepdir/bpgen, clear
. gen welle = 85
. ren bfamstd famstd
. save bpgen, replace
. use persnr afamstd using $soepdir/apgen, clear
. gen welle = 84
. ren afamstd famstd
. append using bpgen
. erase bpgen.dta
```

Dies führt zu einem Datensatz mit folgender Struktur:

	persnr	famstd	welle
1.	101	[1] verh	84
2.	102	[1] verh	84
3.	103	[3] ledi	84
<i>snip</i> >*			
12291.	101	[1] verh	85
12292.	102	[1] verh	85
12293.	103	[3] ledi	85

Meistens sollen aus den Datensätzen zunächst bestimmte Beobachtungen ausgewählt werden. Dies geschieht in der Regel durch *Merging* mit den Metadateien *ppfad.dta* oder *hpfad.dta*. In diesem Fall empfiehlt sich oben gezeigtes Vorgehen nicht. Sinnvoll dürfte es aber bei Generierung von *langen* Datensätzen im *Unbalanced*-Paneldesign sein.

Slide 13

Installation von mmerge

Der Befehl „mmerge“ vereinfacht das *Mergen* von Datensätzen.

Der Befehl „mmerge“ wurde nicht von StataCorp programmiert. Der Code wurde jedoch im *Stata-Technical-Bulletin* Nr. 53 veröffentlicht. Es handelt sich daher um einen *STB-Ado*.

„STB-Ados“ gehören nicht zum offiziellen Lieferumfang von Stata. Das Programm muss nachinstalliert werden. Dazu benötigt man die STB-Eintragsnummer. Sie lautet: STB-53, dm75. Die Installation erfolgt daher mit:

```
. net stb-53  
. net install dm75
```

Autor von „mmerge“ ist Jeroen Weesie, Utrecht.

Ados sind externe Stata-Befehle. Die meisten Stata-Befehle sind externe Kommandos. Innerhalb der externen Kommandos unterscheidet man offizielle Stata-Kommandos und nicht offizielle Stata-Kommandos („Cool-Ados“). Offizielle Stata-Kommandos werden von StataCorp programmiert. StataCorp übernimmt bis zu einem gewissen Umfang eine Qualitätsgarantie für diese Programme.

Externe Stata-Befehle können jedoch von jedem Stata-Anwender programmiert werden. Dies sind die nicht offiziellen externen Befehle.

Jeder Stata-Anwender kann diese Programme anderen Stata-Anwendern zur Verfügung stellen. Dazu müssen die Programme in geeigneter Weise veröffentlicht werden. Nach Veröffentlichungsform kann man drei Arten von nicht offiziellen Stata-Befehlen unterscheiden:

- *STB-Ados*
- *Statalist-Ados*
- *Andere Ados*

Die nicht offiziellen *Ados* erweisen sich oft als außerordentlich hilfreiche Werkzeuge. Oft stehen Prozeduren dem Nutzer schon lange zur Verfügung, bevor sie

zum offiziellen Bestandteil von Stata werden. Dennoch hat die Verwendung nicht-offizieller *Ados* auch einige Nachteile. Hier die Wichtigsten:

- Allgemein fällt die Verlässlichkeit der Befehle, von den STB- zu den Anderen *Ados*. Arbeitet man mit den „Anderen“ *Ados* heißt das: *Check results carefully!*
- Kein Stata-Nutzer installiert alle Stata-*Ados* die es gibt. *Do-Files*, in denen *Cool-Ados* verwendet werden sind darum auf anderen Rechnern möglicherweise nicht lauffähig. Es kann sinnvoll sein, die Befehle zum Installieren der *Cool-Ados* direkt in den entsprechenden *Do-File* zu schreiben.

```
snip >␣
capture which hplot
if _rc ~= 0 {
    net stb-53
    net install dm75
}
```

- *Cool-Ados* werden bisweilen überarbeitet. Zumindest bei *Statalist-Ados* und Anderen *Ados* kann man sich dabei nicht darauf verlassen, dass sich die Syntax des Befehl dabei nicht ändert. Wer sicher gehen muss, sollte die jeweils verwendete Version in sein Arbeitsverzeichnis kopieren. Dann wird diese Version gegenüber späteren Neuinstallationen bevorzugt.

Informationen über Installation, Fundstellen und Suchmöglichkeiten finden sich in Kohler/Kreuter, Kap. 12.

Eigenschaften von mmerge

- Ausweisung von *Unmatched Records* auf Grund von *Missings*
- Anzeige der Namen von Variablen die in den *Master* und in den *Using*-Daten sind
- Automatische Sortierung
- Automatische Löschung von *_merge*
- Automatische Tabellierung von *_merge*
- Schlüsselvariablen können unterschiedlich heißen
- Auswahl von Beobachtungen in den *Using*-Daten
- etc.

Slide 14

Funktionsweise von mmerge

Die Syntax von „mmerge“ entspricht im Wesentlichen der von „merge“. Meist wird man zusätzlich die Optionen „type()“ und „unmatched“ angeben.

Slide 15

```
. use hhnr ahhnr bhhnr chhnr anetto bnetto cnetto persnr
using $soepdir/ppfad, clear
. keep if anetto==1 & bnetto==1 & cnetto==1
. drop anetto bnetto cnetto
. mmerge hhnr ahhnr persnr using $soepdir/ap, t(1:1) unmatched(none)
. mmerge hhnr ahhnr using $soepdir/ah, t(n:1) unmatched(none)
```

Mit „mmerge“ müssen „Master“- und „Using“-Daten vor dem *Merge* nicht mehr sortiert werden.

Mit „type()“ werden die Eigenschaften der Schlüsselvariablen festgelegt. Die Angabe der Option „type()“ ist nicht unbedingt notwendig. Sie sollte aber spezifiziert werden, damit „mmerge“ eine Überprüfung der Eigenschaften der Schlüsselvariablen vornehmen kann. „type()“ erlaubt die Angabe folgender Eigenschaften:

- auto Der Type wird automatisch bestimmt. Dies ist die Voreinstellung. Eine Überprüfung der Eigenschaften kann hier aus naheliegenden Gründen nicht stattfinden.
- 1:1 die Schlüsselvariablen in den „Master“- und „Using“-Daten sind Eineindeutig.
- n:1 die Schlüsselvariablen in den „Master“- Daten sind nicht eineindeutig, d.h. es gibt mehrere Fälle mit der gleichen Ausprägung(skombination) der Schlüsselvariable.
- 1:n die Schlüsselvariablen in den „Using“-Daten sind nicht eineindeutig.
- n:n keine der Schlüsselvariablen ist eineindeutig.
- spread Verwendet entweder 1 : n oder n : 1

Mit der Option „unmatched()“ wird angegeben, was mit nicht zuordbaren Beobachtungen geschieht:

- `none` Nicht zuordbare Beobachtungen werden gelöscht.
- `both` Alle nicht zuordbaren Beobachtungen bleiben im Datensatz.
- `master` Nur nicht zuordbare Beobachtungen der *Master*-Daten bleiben im Datensatz.
- `using` Nur nicht zuordbare Beobachtungen der *Using*-Daten bleiben im Datensatz.

Interessant sind auch die „u“-Optionen von „mmerge“. Die „u“-Optionen nehmen auf die „Using“-Daten Bezug, ohne diese zu verändern. Die wichtigsten sind:

- `ukeep(varlist)` Variablenliste mit Variablen die in den *Using*-Daten gehalten werden sollen.
- `udrop(varlist)` Variablenliste mit Variablen die in den *Using*-Daten gelöscht werden sollen.
- `uif(exp)` Beobachtungen die aus den *Using*-Daten entfernt werden sollen.
- `umatch(varlist)` Name der Schlüsselvariablen in den *Using*-Daten. Diese Option wird benötigt, wenn die Schlüsselvariablen in den *Using*-Daten anders heißen wie in den *Master*-Daten.

Die „u“-Optionen machen die Bearbeitung der *Using*-Daten vor dem *Merging* überflüssig. Daher lassen sich Datenretrivals aus dem SOEP erheblich vereinfachen. Hier ist das Datenretrival aus dem *Desktop Companion*:

retrival8.do

```

1  * Datenretrival Desktop-Companion
2  * -----
3
4  version 7.0
5  set more off
6
7  * Data Structure and Unit of Analysis
8  * -----
9
10 use hhnr persnr ahhnr bhnr chnr mhnr /*
11 */ anetto bnetto cnetto mnetto /*
12 */ sex gebjahr /*
13 */ psample msampreg /*
14 */ using $soepdir/ppfad, clear
15
16 keep if anetto==1 | bnetto==1 | cnetto==1 | (mnetto==1 & msampreg==1)
17 keep if psample==1 | psample==3

```

```
18
19 * Merge
20 * -----
21
22 mmerge hnr ahnr persnr using $soepdir/apgen, /*
23 */ ukeep(abilzeit aerwzeit) t(1:1) unmatched(master)
24 mmerge hnr ahnr using $soepdir/ah, /*
25 */ ukeep(abula) t(n:1) unmatched(master)
26 mmerge hnr ahnr persnr using $soepdir/ap, /*
27 */ ukeep(ap3301) t(1:1) unmatched(master)
28 mmerge hnr bhnr persnr using $soepdir/bpgen, /*
29 */ ukeep(berwzeit bbilzeit) t(1:1) unmatched(master)
30 mmerge hnr bhnr using $soepdir/bhbrutto, /*
31 */ ukeep(bbula) t(n:1) unmatched(master)
32 mmerge hnr bhnr persnr using $soepdir/bp, /*
33 */ ukeep(bp4301) t(1:1) unmatched(master)
34 mmerge hnr chnr persnr using $soepdir/cpgen, /*
35 */ ukeep(cerwzeit cbilzeit) t(1:1) unmatched(master)
36 mmerge hnr chnr using $soepdir/chbrutto, /*
37 */ ukeep(cbula) t(n:1) unmatched(master)
38 mmerge hnr chnr persnr using $soepdir/cp, /*
39 */ ukeep(cp5201) t(1:1) unmatched(master)
40 mmerge hnr mhnr persnr using $soepdir/mpgen, /*
41 */ ukeep(merwzeit mbilzeit) t(1:1) unmatched(master)
42 mmerge hnr mhnr using $soepdir/mhbrutto, /*
43 */ ukeep(mbula) t(n:1) unmatched(master)
44 mmerge hnr mhnr persnr using $soepdir/mp, /*
45 */ ukeep(mp4701) t(1:1) unmatched(master)
46
47 exit
```

mkdat/holrein

Mit „mkdat“ und „holrein“ sieht das Retrieval des *Desktop Companion* so aus:

```
retrival9.do
```

```
snip >
mkdat abilzeit bbilzeit cbilzeit mbilzeit /*
*/ aerzeit berzeit cerzeit merzeit using $soepdir, /*
*/ waves(a b c m) files(pgen) netto(-3,-2,-1,0,1,2,3,4,5) /*
*/ keep(sex gebjahr psample msampreg) clear

keep if anetto==1 | bnetto==1 | cnetto==1 | (mnetto==1 & msampreg==1)
keep if psample==1 | psample==3

holrein abula using $soepdir, files(h) waves(a)
holrein bbula cbula mbula using $soepdir, files(hbrutto) waves(b c m)
holrein ap3301 bp4301 cp5201 mp4701 using $soepdir, /*
*/ files(p) waves(a b c m)
```

Slide 16

„mkdat“ und „holrein“ wurden speziell für SOEP-Retrivals entwickelt. Mit ihnen wird die Itemkorrespondenzliste mehrere Variablen eines Filetyps zusammengeführt.

Die Befehl können mit

```
. net from http://www.sowi.uni-mannheim.de/lehrstuehle/lesas/ado
. net install mkdat
```

installiert werden. Autor der Programme ist Ulrich Kohler.

Allgemein lautet die Syntax wie folgt:

```
mkdat itemlist using soepdir, files(filetyp) waves(wavelist)
netto(design) keep(vars of ppfad)
```

Innerhalb eines „mkdat“-Befehls können beliebig viele Variablen angegeben werden. Alle Variablen müssen jedoch aus einem Filetyp stammen. Um Variablen anderer Filetypen einzulesen verwendet man „holrein“.

Als Filetypen werden momentan akzeptiert:

- p Personendatensätze
- h Haushaltsdatensätze

`pgen` Personendatensätze, generierte Variablen
`hgen` Haushaltsdatensätze, generierte Variablen
`pbrutto` Personen, brutto
`hbrutto` Haushalte, brutto
`kind` Kinderdatensätze
`pausl` Personen, Ausländer
`pkal` Personen, Kalendarien
`peigen` selbst generierte Personen-Variablen

Die Integration weiterer Filetypen ist geplant.

Die Itemkorrespondenzliste beider Befehle muss rechteckig sein. Bei unvollständiger Itemkorrespondenzliste müssen Wegwerfnamen eingesetzt werden. Die Wegwerfnamen müssen mit `_` beginnen, also z.B. `_X1`, `_X2` usw.

Die Wellen aus denen die Variablen stammen werden mit den Buchstaben a-o bezeichnet. Zwischen den Buchstaben steht ein Leerzeichen.

Mit der Option `„netto()“` wird das Design des Datensatzes spezifiziert. In die Klammer wird eine, durch Kommas getrennte, Liste von Werten der *?netto*-Variablen angegeben. Es werden nur diejenigen Beobachtungen verwendet, welche die angegebenen Werte auf der *?netto*-Variablen aufweisen. Voreinstellung ist `„netto(1)“`. Dies führt zu einem *Balanced* Paneldesign.

Beachte: Die Angabe konnte oben nicht zur Auswahl der Beobachtungen entsprechend des Beispiels im *Desktop-Compagnion* verwendet werden. Deshalb wurden zunächst alle Beobachtungen ausgewählt und die Auswahl von Hand vorgenommen.

Im Folgenden ein weiteres Beispiel zur Demonstration der Leistungsfähigkeit von `„mkdat“`.

crdm.do

```

1  * pid, bst, fam, bil, bdauer, hheink (14 Wellen, balanced)
2
3  version 7.0
4  set more off
5  clear
6
7  * Retrieval
8  * -----
9
10 #delimit ;
11
12 mkdat
  
```

```
13 /* PID */
14 ap5601 bp7901 cp7901 dp8801 ep7701 fp9301 gp8501 hp9001 ip9001
15 jp9001 kp9201 lp9801 mp8401 np9401
16 ap5602 bp7902 cp7902 dp8802 ep7702 fp9302 gp8502 hp9002 ip9002
17 jp9002 kp9202 lp9802 mp8402 np9402
18 ap5603 bp7903 cp7903 dp8803 ep7703 fp9303 gp8503 hp9003 ip9003
19 jp9003 kp9203 lp9803 mp8403 np9403
20 /* Arbeiter */
21 ap2801 bp3801 cp4601 dp3801 ep3801 fp3801 gp3701 hp4801 ip4801
22 jp4801 kp5101 lp4301 mp4101 np3501
23 /* Selbstaendige */
24 ap2802 bp3802 cp4602 dp3802 ep3802 fp3802 gp3702 hp4802 ip4802
25 jp4802 kp5102 lp4302 mp4102 np3502
26 /* Auszubildende */
27 ap2803 bp3803 cp4603 dp3803 ep3803 fp3803 gp3703 hp4803 ip4803
28 jp4803 kp5103 lp4303 mp4103 np3503
29 /* Angestellte */
30 ap2804 bp3804 cp4604 dp3804 ep3804 fp3804 gp3704 hp4804 ip4804
31 jp4804 kp5104 lp4304 mp4104 np3504
32 /* Beamte */
33 ap2805 bp3805 cp4605 dp3805 ep3805 fp3805 gp3705 hp4805 ip4805
34 jp4805 kp5105 lp4305 mp4105 np3505
35 /* frueher Arbeiter */
36 ap1601 bp24b01 cp24b01 dp22b01 ep22b01 fp20b01 gp23b01 hp25b01
37 ip25b01 jp25b01 _x1 _x11 _x21 _x21a
38 /* frueher Selbst. */
39 ap1602 bp24b02 cp24b02 dp22b02 ep22b02 fp20b02 gp23b02 hp25b02
40 ip25b02 jp25b02 _x2 _x12 _x22 _x22a
41 /* frueher Azubi */
42 ap1603 bp24b03 cp24b03 dp22b03 ep22b03 fp20b03 gp23b03 hp25b03
43 ip25b03 jp25b03 _x3 _x13 _x23 _x23a
44 /* frueher Angest. */
45 ap1604 bp24b04 cp24b04 dp22b04 ep22b04 fp20b04 gp23b04 hp25b04
46 ip25b04 jp25b04 _x4 _x14 _x24 _x24a
47 /* frueher Beamter */
48 ap1605 bp24b05 cp24b05 dp22b05 ep22b05 fp20b05 gp23b05 hp25b05
49 ip25b05 jp25b05 _x5 _x15 _x25 _x25a
50 using $soepdir, files(p) waves(a b c d e f g h i j k l m n)
51 keep(sex gebjahr) ;
52
53 holrein
54 /* Familienstand */
55 afamstd bfamstd cfamstd dfamstd efamstd ffamstd gfamstd hfamstd
56 ifamstd jfamstd kfamstd lfamstd mfamstd nfamstd
57 /* Schulbildung */
58 apsbil bpsbil cpsbil dpsbil epsbil fpsbil gpsbil hpsbil
59 ipsbil jpsbil kpsbil lpsbil mpsbil npsbil
60 /* Bildungsdauer in Jahren */
61 abilzeit bbilzeit cbilzeit dbilzeit ebilzeit fbilzeit
62 gbilzeit hbilzeit ibilzeit jbilzeit kbilzeit lbilzeit mbilzeit nbilzeit
63 using $soepdir, files(pgen) waves(a b c d e f g h i j k l m n) ;
64
65 holrein
66 /* Haushaltseinkommen */
67 ah46 bh39 ch51 dh51 eh42 fh42 gh42 hh48 ih49 jh49 kh49
68 lh50 mh50 nh50
69 using $soepdir, files(h) waves(a b c d e f g h i j k l m n) ;
70
71 #delimit cr
72
73 drop ?netto
74 save dm, replace
75
76 exit
```


Weites und Langes Datenformat

Slide 17

i	$X_{t=1}$...	$X_{t=T}$
1	x_{11}	...	x_{1T}
\vdots	\vdots	\ddots	\vdots
n	x_{n1}	...	x_{nT}

i	t	X
1	1	x_{11}
\vdots	\vdots	\vdots
n	1	x_{n1}
\vdots	\vdots	\vdots
1	T	x_{1T}
\vdots	\vdots	\vdots
n	T	x_{nT}

Paneldaten treten typischerweise im weiten oder langen Datenformat auf. Innerhalb dieser Datenformate treten typische Datenmanagementprobleme auf.

Das weite Datenformat lässt häufig die Anwendung von Schleifen sinnvoll erscheinen.

Das lange Datenformat erfordert häufig die Anwendung der Rekodierung mit „by“, „_n“, „_N“ und expliziten Subskripten. Da im SOEP Personen innerhalb von Haushalten vorliegen sind in diesem Sinne auch weite Daten bereits Daten im langen Format.

In Stata sollten Paneldaten normalerweise in das lange Format gebracht werden. Zuvor ist eine Bearbeitung des weiten Datensatzes jedoch meist unumgänglich.

Zur Darstellung der Datenmanagement-Konzepte soll der mit *crdm.do* erzeugte Datensatz *dm.dta* verwendet werden.

```
. do crdm
```

for

„for“ ist die einfachste Form von Schleifen innerhalb von Stata. Der Befehl ist sinnvoll, wenn *ein* Befehl mehrmals erledigt werden soll.

Slide 18

```
dml.do
```

```
snip ⌘
for varlist ap5603 bp7903 cp7903 dp8803 ep7703 fp9303 gp8503 hp9003 /*
*/ ip9003 jp9003 kp9203 lp9803 mp8403 np9403: replace X = 6 - X if X >= 0

for varlist ap5603 bp7903 cp7903 dp8803 ep7703 fp9303 gp8503 hp9003 /*
*/ ip9003 jp9003 kp9203 lp9803 mp8403 np9403: lab val X pii

label define pi3 1 "s. schwach" 2 "schwach" /*
*/ 3 "maessig" 4 "stark" 5 "s. stark"
snip ⌘
```

In „for“ wird zunächst eine Liste angegeben. Danach wird das Kommando nach dem Doppelpunkt für jedes Element der Liste ausgeführt.

Vor der eigentlichen Liste wird der *Listentyp* spezifiziert.

varlist Die Liste ist eine Liste von Variablen. Alle Abkürzungsmöglichkeiten von Variablenlisten sind erlaubt (siehe „help varlist“)

numlist Die Liste ist eine Nummernliste. Alle Abkürzungsmöglichkeiten von Nummernlisten sind erlaubt (siehe „help numlist“ sowie Kohler/Kreuter, S. 67)

anylist Die Liste ist eine durch Leerzeichen getrennte Liste von Zeichen

Wenn die Variablenliste einem einheitlichen Namensschema gehorcht, kann obiger „for“-Befehl viel kürzer gefasst werden. Im weiten Format kann es darum sinnvoll sein, die korrespondierenden Variablen einheitlich umzubenennen. Auch hierzu kann der „for“-Befehl eingesetzt werden. Dazu verwendet man den „for“-Befehl mit zwei *Forlisten*. Zwischen den *Forlisten* steht ein Backslash. Für die Elemente der ersten *Forlist* wird ein X im Stata-Befehl eingesetzt, für die Elemente der zweiten *Forlist* ein Y.

```
dml.do
```

```
snip ⌘
```

```
for numlist 84/97 \ /*
*/ varlist ap5603 bp7903 cp7903 dp8803 ep7703 fp9303 gp8503 hp9003 /*
*/ ip9003 jp9003 kp9203 lp9803 mp8403 np9403: ren Y piix
```

snip ⋈

Im „for“-Befehl können bis zu 9 *Forlisten* verwendet werden. Darüber hinaus können beliebig viele Kommandos nach dem Doppelpunkt verwendet werden. Zwischen den Kommandos steht ebenfalls ein Doppelpunkt.

An Stelle von „for“-Befehlen mit mehreren *Forlisten* und mehreren Stata-Befehlen sollte man besser „foreach“ oder „forvalues“ verwenden.

foreach

Mit „foreach“ können auf einfache Weise Schleifen über mehrere Kommandos gebildet werden:

dml.do

```

snip >
local i 84
foreach piece in ap56 bp79 cp79 dp88 ep77 fp93 gp85 {
    gen pid`i' = 1 if `piece'01 == 2
    replace pid`i' = 2 if `piece'02==1
    replace pid`i' = 3 if `piece'02>=2 & `piece'02<=4
    replace pid`i' = 4 if `piece'02==5
    replace pid`i' = 5 if `piece'02==6
    replace pid`i' = 6 if `piece'02==7 | `piece'02==8
    lab var pid`i' "Parteiidentifikation 19`i'"
    lab val pid`i' pid
    local i = `i' + 1
}
lab def pid 1 "Keine" 2 "SPD" 3 "CDU/CSU" 4 "FDP" 5 "B90/Gr" /*
*/ 6 "Andere P"
snip >

```

Slide 19

Mit „foreach“ kann auch der obige „for“-Befehl zum umbenennen von Variablen neu geschrieben werden:

```

local i 84
foreach var of varlist ap5601 bp7901 cp7901 dp8801 ep7701 fp9301 /*
*/ gp8501 hp9001 ip9001 jp9001 kp9201 lp9801 mp8401 np9401 {
    ren `var' pii`i'
    note pii`i': SOEP-Name `var'
    local i = `i' + 1
}

```

Diese Schleife ist die Grundlage des Programms „soepren“.

soepren

Das Programm „soepren“ benennt eine Liste korrespondierender Variablen nach einem einheitlichen Namensschema. Dies erleichtert die Überführung in einen Datensatz im langen Format und die Bearbeitung des Datensatzes mit „for“ und „forvalues“.

dml.do

Slide 20

```

snip >
soepren ah46-nh50, new(hhein) w(84/97)
soepren ?bilzeit, new(bdauer) w(84/97)
soepren ?psbil, new(bil) w(84/97)
soepren ?famstd, new(fam) w(84/97)
soepren ahnr-lhnr mhnr nhnr, new(hnr) w(84/97)

soepren ap5601 bp7901 cp7901 dp8801 ep7701 fp9301 gp8501 hp9001 /*
*/ ip9001 jp9001 kp9201 lp9801 mp8401 np9401, new(pi) w(84/97)

for any 01 02 03 04 05: soepren ap28X bp38X cp46X dp38X ep38X fp38X
/* gp37X hp48X ip48X jp48X kp51X lp43X mp41X np35X, new(bstX) w(84/97)
snip >

```

Installation von „soepren“ erfolgt durch

```

. net from http://www.sowi.uni-mannheim.de/lehrstuehle/lesas/ado
. net install soepren

```

soepren.ado

```

1  *! soepren.ado 0.1, ukohler@sowi.uni-mannheim.de
2  program define soepren
3  version 7.0
4  syntax varlist, NEWstub(string) Waves(numlist integer)
5
6  local nvars: word count `varlist'
7  local nwaves: word count `waves'
8  if `nvars' ~= `nwaves' {
9      display as error "lists have unequal number of elements"
10     exit 198
11 }
12
13 tokenize `waves'
14 foreach var of varlist `varlist' {
15     ren `var' `newstub'`1'
16     note `newstub'`1': SOEP-Name `var'
17     mac shift
18 }
19 end
20 exit

```

reshape

Die Umsetzung eines Datensatzes vom *weiten* in das *lange* Format erfolgt mit „reshape“.

„reshape“ ist zeit- und speicherintensiv. Man sollte daher Maßnahmen zum Sparen von Speicherplatz treffen (Unnötige Variablen entfernen, „compress“)

Am einfachsten ist die Überführung wenn alle Variablen einem einheitlichen Namensschema folgen.

dml.do

```

snip ⋈
compress
keep hnr persnr sex gebjahr hnr* bst* pi* pid* pii* bil* hhein* fam*

reshape long hnr bst01 bst02 bst03 bst04 bst05 pi pid pii bil hhein /*
*/ fam , i(persnr) j(welle)
snip ⋈

```

„reshape long“ überführt einen weiten Datensatz in einen langen Datensatz. „reshape wide“ überführt einen langen Datensatz in einen weiten.

„reshape“ generiert automatisch die Variable Welle mit der (zweistelligen) Angabe der Erhebungswelle. Die Werte diese Variable übernimmt „reshape“ aus dem Variablennamen der Variablen im weiten Datensatz. Dies setzt voraus, dass die Variablen entsprechend benannt wurden.

„reshape“ erlaubt auch andere Namensschemata als die hier verwendete. Allerdings wird die Syntax dann komplizierter.

Nach dem ersten „reshape“-Befehl können die Datensätze durch Eingabe von „reshape long“ bzw. „reshape wide“ ohne Variablenliste in das jeweilige Datenformat umgesetzt werden.

Datenmanagement im langen Format

Rekodierungen im langen Format erfordern häufig die Verwendung von „by“, „n“, „N“ und expliziten Subskripten.

„n“ ist die laufende Position im Datensatz.

„N“ ist die letzte Position im Datensatz.

„by“ unterteilt den Datensatz in mehrere Untergruppen. Nach „by“ steht „n“ für die laufende Position innerhalb der jeder Subgruppe und „N“ für die letzte Position in jeder Subgruppe

Explizite Subskripten werden durch die Konstruktion *varname[position]* angegeben. Mit ihnen kann die Position einer Beobachtung innerhalb eines Datensatzes oder einer Subgruppe spezifiziert werden.

Slide 22

Eine einführende Beschreibung der Rekodierung mit „by“, „n“ und „N“ findet sich bei Kohler/Kreuter, S. 83–90.

Beispiele 1

Wie viele gültige Werte hat jeder Befragte auf der Variable *Parteineigung*?

```
. sort persnr welle  
. by persnr: gen pivalid = sum(pid ~= .)  
. by persnr: replace pivalid = pivalid[_N]
```

Wie oft wechselten die Befragten ihre Haushaltsnummer?

```
. by persnr (welle): gen hnrchg = sum(hnr ~= hnr[_n-1] & _n ~= 1)  
. by persnr: replace hnrchg = hnrchg[_N]
```

Slide 23

Wahre Bedingungen erhalten den Wert 1, falsche den Wert 0. „pid ~= .“ ist darum 0 für alle Beobachtungen, mit einem Missing auf der Variable *pid* und 1 für alle Beobachtungen mit einem gültigen Wert. „sum(pid ~= .)“ ist die laufende Summe der Beobachtungen mit einem gültigen Wert. In der untersten Beobachtung jedes Befragten steht damit die Anzahl der gültigen Beobachtungen. Durch „replace pivalid = pivalid[_N]“ wird diese Zahl in die übrigen Beobachtungen desselben Befragten *hineinkopiert*.

Vorsicht ist geboten bei Ausdrücken wie im unteren Beispiel. „_n-1“ nimmt Bezug auf die vorangegangene Beobachtung. Der Wert der Beobachtung, die der ersten Beobachtung vorangeht ist *Missing*. Die Bedingung „hnr ~= hnr[_n-1]“ ist daher normalerweise wahr, d.h. 1. Dies ist oben nicht erwünscht.

Im zweiten Beispiel wird eine korrekte Lösung nur erzeugt, wenn der Datensatz nach *persnr* und *welle* sortiert ist. Darum wurde in „by“ die Variable *welle* in Klammern gesetzt. Dies dient der Überprüfung der Sortierung.

Beispiel 2

Ermittle die Dauer von Episoden mit gleicher Parteiidentifikation

```
. preserve
. gen start = welle
. by persnr (welle): gen episode = sum(pid ~= pid[_n-1])
. by persnr episode (welle), sort: gen end = welle[_N]
. by persnr episode: keep if _n == 1
. list persnr pid episode start end
. restore
```

Slide 24

Der Befehl „preserve“ speichert den momentanen Datensatz temporär ab. Mit „restore“ wird er wieder hergestellt.

Im ersten Schritt werden hier die Parteipräferenz-Episoden durchnummeriert. Im zweiten Schritt wird die Welle der letzten Beobachtung einer Episode den anderen Beobachtungen dieser Episode zugespielt. Am Ende werden nur die ersten Beobachtungen einer Episode aufbewahrt. Diese enthalten dann das Startjahr der ersten und das Endjahr der letzten Beobachtung.

Beachte, dass seit Stata 7.0 die Sortierung des Datensatzes auch innerhalb von „by“ vorgenommen werden kann.

Beispiel 3**Slide 25**

Erzeuge eine Variable mit dem Wert 1, für alle Wellen nach dem Ereignis „Erste Hochzeit“ (Wechsel von ledig zu verheiratet) und 0 in allen anderen Fällen.

```
. by persnr (welle): gen ehoch = sum(fam==1 & fam[_n-1] == 3)
```

Beispiel 4

Überprüfe, ob der Anteil von Frauen in 2 Personen-Haushalten, bei denen beide Personen verheiratet sind, immer 0.5 beträgt.

Slide 26

```
. preserve
. by welle hnr, sort: keep if _N == 2 & fam[1]==1 & fam[2]==1
. by welle hnr: gen pwomen = sum(sex==1)/sum(sex~=.)
. by welle hnr: assert pwomen == .5 if _n==_N
. restore
```

Slide 27**xt-Daten**

Zur Verwendung der Stata-Befehle zur Analyse von Paneldaten müssen die Daten als *xt*-Daten deklariert werden.

Als *xt*-Daten können Paneldatensätze im langen Format deklariert werden. Zur Deklaration wird erklärt, welche Variable die Personen und welche Variable die Zeit enthält.

```
. iis persnr  
. tis welle
```

Nach der Deklaration der *xt*-Daten können die *xt*-Kommandos angewandt werden. Die Liste der offiziellen *xt*-Kommandos finden Sie durch

```
. help xt
```

Weitere Kommandos finden sich durch

```
. search xt
```

bzw.

```
. net search xt
```

xtdes

„xtdes“ beschreibt das Teilnahmemuster der Befragten.

Slide 28

```
. do retrival9
. soepren ap3301 bp4301 cp5201 mp4701, new(eink) w(84/86 96)
. keep persnr gebjahr sex eink*
. reshape long eink, i(persnr) j(welle)
. replace eink = . if eink < 0
. xtde if eink~=., i(persnr) t(welle)
```

Die Deklaration der Datensätze als *xt*-Daten kann auch als Option der jeweiligen Befehle erfolgen. Die Deklaration gilt dann auch für die anderen *xt*-Kommandos.

longplot

Slide 29

„longplot“ zeichnet die Werte einer Variable gegen die Zeit, wobei die Werte eines Befragten durch ein Linie verbunden werden. Dies ist nur bei kleineren Datensätzen sinnvoll.

```
. longplot eink welle in 1/100, by(sex) id(persnr)
```

Der Befehl kann mit

```
. archinst longplot
```

installiert werden. Dies setzt voraus, dass zuvor das Ado-Paket „archutil“ installiert wurde:

```
. net stb-54
. net install ip29_1
```

„longplot“ ist kein *xt*-Kommando. Darum muss die *id()* Option angegeben werden, auch wenn die Daten als *xt*-Daten deklariert wurden.

Die von „longplot“ produzierte Grafik kann relativ einfach auch ohne „longplot“ erstellt werden.

xt1.do

```
1  * longplot von Hand
2  * -----
3
4  version 7.0
5  set more off
6  clear
7
8  * Retrieval
9  * -----
10
```

```
11 do retrieval9
12
13 * Wide -> Long
14 * -----
15 soepren ap3301 bp4301 cp5201 mp4701, new(eink) w(84/86 96)
16 keep persnr gebjahr sex eink*
17 reshape long eink, i(persnr) j(welle)
18
19 replace eink=. if eink<0
20
21
22 * Longplot
23 * -----
24
25 separate eink, by(sex)
26 sort persnr welle
27 graph eink1 eink2 welle, c(LL), in 1/100
28
29 exit
30
31
32
33
```

xtgraph

Slide 30

„xtgraph“ produziert Zeitreihenplots aus Paneldaten. Gezeigt werden Mittelwerte und Streuungswerte über die Zeit.

```
. iis persnr
. tis welle
. xtgraph eink
. xtgraph eink, av(median) bar(iqr) by(sex)
```

Der Befehl kann mit

```
. archinst xtgraph
```

installiert werden. Dies setzt voraus, dass zuvor das Ado-Paket „archutil“ installiert wurde:

```
. net stb-54
. net install ip29.1
```

„xtgraph“ ist ein zeitintensives Kommando. Die meiste Zeit nimmt dabei die Generierung der Daten in Anspruch. Zur Produktion von Präsentationsgrafiken kann es sinnvoll sein, die Daten von Hand zu erzeugen und anschließend die Graphik zu gestalten.

Die Erzeugung der Grafik von Hand kann z.B. so erfolgen:

```
1 * xtgraph von Hand
2 * -----
3
4 version 7.0
5 set more off
6 clear
7
8 * Retrieval
```

xt2.do

```
9 * -----
10
11 do retrival9
12
13 * Wide -> Long
14 * -----
15 soepren ap3301 bp4301 cp5201 mp4701, new(eink) w(84/86 96)
16 keep persnr gebjahr sex eink*
17 reshape long eink, i(persnr) j(welle)
18
19 replace eink=. if eink<0
20
21 * xtgraph-Daten
22 * -----
23
24 collapse (mean) meink=eink (sd) sdeink=eink (count) n = eink, by(welle)
25 gen ub = meink + 1.96* sqrt(sdeink^2/n)
26 gen lb = meink - 1.96* sqrt(sdeink^2/n)
27
28 * Graphik
29 * -----
30
31 graph meink ub lb welle, c(lII) s(oii)
32 exit
```

```
. graph
```

xttab, xttrans

„xttab“ ist eine verallgemeinerte Häufigkeitsauszählung, bei der die Häufigkeiten in „Between“ und „Within“-Komponenten zerlegt werden.

```
. use dml, clear  
. xttab pid, i(persnr)
```

„xttrans“ berechnet die Übergangswahrscheinlichkeiten von einem Zeitpunkt zum nächsten.

```
. xttrans pid, t(welle)
```

Slide 31

„xttrans“ ist lediglich eine Kreuztabelle der aktuellen Werte gegen die vorangegangenen:

```
. by persnr (welle), sort: gen pidlag = pid[_n-1]  
. tabulate pidlag pid, row
```

Panelmodelle

Slide 32

Die Panelregressionsmodelle von Stata folgen alle der in Stata üblichen Syntax von statistischen Modellen:

```
. Befehl depvar indepvarlist
```

Eine vollständige Liste der Panelregressionsmodelle erhält man mit

```
. help xt
```

xtreg	Fixed-, between- and random-effects, and population-averaged linear models
xtregar	Fixed- and random-effects linear models with an AR(1) disturbance
xtgls	Panel-data models using GLS
xtpcse	OLS or Prais-Winsten models with panel-corrected standard errors
xtrchh	Hildreth-Houck random coefficients models
xtivreg	Instrumental variables and two-stage least squares for panel-data models
xtabond	Arellano-Bond linear, dynamic panel data estimator
xttobit	Random-effects tobit models
xtintreg	Random-effects interval data regression models
xtlogit	Fixed-effects, random-effects, & population-averaged logit models
xtprobit	Random-effects and population-averaged probit models
xtclog	Random-effects and population-averaged cloglog models
xtpois	Fixed-effects, random-effects, & population-averaged Poisson models
xtnbreg	Fixed-effects, random-effects, & population-averaged negative binomial models
xtgee	Population-averaged panel-data models using GEE

Ereignisdaten

Zur Verwendung der Stata-Befehle zur Analyse von Ereignisdaten müssen die Daten als *st*-Daten deklariert werden. Dies geschieht mit dem Befehl „stset“.

Survival-Time-Daten können in unterschiedlichen Formaten vorliegen.

Eine Überblick über mögliche Formate gibt [R] stset.

Die Liste der offiziellen *st*-Kommandos finden Sie durch

```
. help st
```

Weitere Kommandos finden sich durch

```
. search st
```

bzw.

```
. net search st
```

stset	Declare data to be survival-time data
stdes	Describe survival-time data
stsum	Summarize survival-time data
stvary	Report which variables vary over time
stfill	Fill in by carrying forward values of covariates
stgen	Generate variables reflecting entire histories
stsplit	Split time-span records
stjoin	Join time-span records
stbase	Form baseline dataset
sts	Generate, graph, list, and test the survivor and cumulative hazard functions
stir	Report incidence-rate comparison
stci	Confidence intervals for means and percentiles of survival time
strate	Tabulate failure rate
stptime	Calculate person-time
stmh	Calculate rate ratios using Mantel-Haenszel method
stmh	Calculate rate ratios using Mantel-Cox method
stcox	Estimate Cox proportional hazards model
stphptest	Test of Cox proportional hazards assumption
stphplot	Graphical assessment of the Cox prop. hazards assumption
stcoxkm	Graphical assessment of the Cox prop. hazards assumption
streg	Estimate parametric survival models (exponential, weibull, gompertz, lognormal, loglogistic, gamma)
stcurve	Plot fitted survival functions

```
sttocc      Convert survival-time data to case-control data
sttocc      Convert survival-time data to count-time data
cttost      Convert count-time data to survival-time data
snapspan    Convert snapshot data to time-span data

st_is       Survival analysis subroutines for programmers
```

Zeitreihen

Zur Verwendung der Stata-Befehle zur Analyse von Zeitreihen müssen die Daten als *ts*-Daten deklariert werden. Dies geschieht mit dem Befehl „*tsset*“.

Survival-Time-Daten können in unterschiedlichen Formaten vorliegen. Eine Überblick über mögliche Formate gibt [R] *tsset*.

Die Liste der offiziellen *st*-Kommandos finden Sie durch

```
. help time
```

Weitere Kommandos finden sich durch

```
. search time series
```

bzw.

```
. net search time series
```

```
tsset      Declare dataset to be time series
tdates     Information on time-series dates and functions
tfmt       Information on time-series formats
```

Time-series analysis and graphics:

```
corrgram   Correlogram in table and character-based plot format
ac         Correlogram of the autocorrelations
pac        Correlogram of the partial autocorrelations
xcorr      Cross-correlogram for bivariate time series
pergram    Periodogram
cumsp      Cumulative spectral distribution graph
```

Time-series estimation commands:

```
arima      Autoregressive integrated moving average models
arch       Autoregressive conditional heteroskedasticity (ARCH) family of
           models
prais      Prais-Winsten regression
```

Time-series tests:

```
wntestb    Bartlett's periodogram-based test for white noise
wntestq    Portmanteau (Box-Pierce or Ljung-Box) test for white noise
dfuller    Augmented Dickey-Fuller unit root test
pperron    Phillips-Perron test for unit roots
dwstat     The Durbin-Watson d statistic after regress
```

Time-series programming commands:

```
tsrevar    Generate time-series operated temporary variables
```

Slide 34

```
tsunab    Unabbreviate variable list containing time-series operators  
tsreport  Report time gaps in a dataset or estimation sample
```

Advanced time-series programming command:

```
_crcarl   Compute AR(1) rho from residuals
```