Institute for Empirical Research in Economics
University of Zurich

Working Paper Series
ISSN 1424-0459

Working Paper No. 277

Einführung in die Statistiksoftware STATA

Andreas Kuhn and Oliver Ruf

March 2006

Einführung in die Statistiksoftware STATA®

Andreas Kuhn*
Oliver Ruf

Institut für Empirische Wirtschaftsforschung
Universität Zürich

29. März 2006

Zusammenfassung

STATA ist eine umfangreiche Statistiksoftware, welche sich sowohl zur Verwaltung von umfangreichen Datenbeständen als auch zur statistischen Analyse eignet. Diese kurze Einführung gibt einen Überblick über die allgemeine Syntax des Programmes sowie die wichtigsten Anweisungen zur Datenaufbereitung und –analyse. Diese Einführung dient der selbstädigen Einarbeitung in das Programm. Anhand eines Beispieldatensatzes können die meisten der beschriebenen Befehle angewendet und nachvollzogen werden.

Keywords: Statistiksoftware, STATA
JEL–Classification: A22, C87

*Kontakt: Institut für Empirische Wirtschaftsforschung, Blümlisalpstrasse 10, 8006 Zürich, kuhn@iew.unizh.ch, oliruf@iew.unizh.ch. Die verwendeten Beispieldatensätze sowie die entsprechenden Skripte können bei den Autoren angefordert werden.

Inhaltsverzeichnis

1 Einleitung

STATA ist ein umfangreiches Programm zur statistischen Analyse (Siehe die Seite des Herstellers zu weiteren Informationen: `www.stata.com`) und liegt mittlerweile in Version 9 .0 vor. Urspünglich war STATA fast ausschliesslich befehlsgesteuert (etwa im Gegensatz zu SPSS), nur sehr wenige Funktionen (z.B. das Öffnen und das Speichern von Datensätzen) liessen sich (auch) über Menüs aufrufen. Ab Version 8.0 können fast alle Befehle über das Menü aufrufen, das Arbeiten mit dem Menü wird aber nicht empfohlen, da die Programmsprache sehr einfach und intuitiv ist, was nach kurzer Zeit ein schnelles und exaktes Arbeiten erlaubt. Die Benutzeroberfläche von STATA besteht aus fünf wesentlichen Teilen bzw. Bildschirmen:

1. Der Menüleiste;
2. Dem Eingabefenster;
3. Dem Ausgabefenster;
4. Dem Variablenfenster und
5. Dem "review"–Fenster.

Über die Menüleiste kann man primär die folgenden Funktionen aufrufen: Öffnen und Speichern von Datensätzen, Aufrufen des Dateneditors sowie des do–file Editors. Die gesamte Steuerung von STATA erfolgt über das Befehlseingabefenster (bzw. über die Menüleiste). Sämtliche Outputs erscheinen auf dem Ausgabefenster (mit Ausnahme von Grafiken; diese erscheinen in einem separaten Fenster). Über das Variablenfenster können die im Datensatz vorhandenen Variablen leicht in das Befehlsfenster übertragen werden. Über das "review"-Fenster lassen sich die zuletzt benutzten Befehle nochmals aufrufen[1]. Laufende Berechnungen oder Hilfefenster lassen sich mit dem Break–Symbol in der Menüleiste abbrechen.

Daneben kann mit der Eingabe von `doedit` der Do–File Editor geöffnet bzw. dortin gewechselt werden. Mit `edit` gelandt man in den Dateneditor, mit `browse` gelangt man in den Daten–Browser (hier können die Daten nur angeschaut, aber nicht verändert werden).

1.1 Hilfe– und Suchfunktion

Weitere Informationen findet man in den verschiedenen Handbüchern zu STATA. Als Einstieg in das Programm eignet sich das `Getting Started` Manual sowie der `User's Guide`. Ausserdem findet sich auf der homepage von STATA (unter der Rubrik faq's) eine Antwort zu manch kniffligem Problem: `www.stata.com/support/faqs/`.
STATA verfügt über ein eigenes Hilfe–Menü (welches weitgehend den Einträgen in den Handbüchern entspricht), auf welches mit dem Befehl `help 'Name des Befehls'` zugegriffen werden kann.

Es gibt ausserdem die Suchfunktion `search 'Stichwort'`, mit welcher nach bestimmten Stichwörtern (lokal und auf dem Internet) gesucht werden kann. Mit `net search 'Stichwort'` kann nach zusätzlichen Programmen (sogenannten `ado-files`, siehe Abschnitt 1.3) gesucht werden, welche nicht standardmässig im Programm integriert sind.

Bei kniffligeren Problemen bietet es sich an, sich auf dem STATA–listserver anzumelden bzw. sich in dessen Archiv umzuschauen (`www.stata.com/statalist/`) oder allenfalls den technischen Support von STATAin Anspruch zu nehmen (`tech-support@stata.com`).

[1] Mit dem Befehl `#review k` können die letzten `k`$\leq$`100` Befehlseingaben ebenfalls wieder aufgerufen werden.

1.2 Arbeitsspeicher setzen

Da STATA den gesamten Datensatz in den Arbeitsspeichere einliest, muss entsprechend viel Arbeitsspeicher bereitgestellt werden (dabei ist zu berücksichtigen, dass man i.d.R. noch zusätzliche Variablen generieren möchte). Vor dem Einlesen eines Datensatzes sollte man die Grösse des zugewiesenen Arbeitsspeichers einstellen. Über `memory` erlangt man Auskunft über den bereits zugeteilten Arbeitsspeicher. Mittels `set memory #m` kann man die Anzahl Megabytes zuweisen. Man sollte jedoch nie mehr als $\frac{3}{4}$ des insgesamt verfügbaren Speichers an STATA zuweisen. Mit dem Befehl `about` erhält man Informationen über den vorhandenen Arbeitsspeicher auf dem benutzten Rechner.

1.3 Zusätzliche Programmroutinen (`ado-files`) installieren und verwalten

Neben den Programmroutinen, welche standardmässig in STATA vorhanden sind, können zusätzlich Routinen installiert werden, welche von anderen Anwendern geschrieben und zur Verfügung gestellt werden. In der STATA–Sprache heissen solche Programme `ado-files`.

Beispielsweise sind bisher noch keine Routinen zur Beschreibung der Einkommensverteilung standardmässig in STATA enthalten. Mit `net search inequality` kann man beispielsweise überprüfen, ob Programme zum Stichwort `inequality` existieren und diese allenfalls direkt installieren.

Mit `ado dir` kann man sich anschauen, welche `ado-files` man bereits installiert hat. Mit `ado describe` erhält man Informationen zu den installierten `ado-files`.

1.4 Updaten

STATA wird kontinuierlich erweitert, so dass man das Programm auch dann updaten kann, wenn keine neue Version des Programmes vorliegt. Mit `update query` erhält man Informationen zur aktuell installierten Version sowie zur offiziellen Version. Anschliessend erhält man eine Empfehlung zum Updaten. Ein vollständiges Update wird über `update all` durchgeführt.

1.5 Online Taschenrechner

Der Befehl `display` <exp> lässt sich als online-Taschenrechner verwenden. Siehe auch `help functions`.

```
. dis 5*6+(7-2)
35
. dis sqrt(27/(4-2))
3.6742346
. dis 3.4 ^2
11.56
```

Der Befehl `display` kann auch für das Programmieren (insbesondere für das Darstellen von Ergebnissen) verwendet werden.

2 Das Einlesen und Abspeichern von Daten

Wenn ein Datensatz geöffnet ist, erscheinen im Variablenfenster die Variablen des Datensatzes. Mit dem Befehl `clear` wird der aktuelle Datensatz aus dem Arbeitsspeicher gelöscht.

2.1 Definieren von Makros

Ein Makro ist ein Platzhalter. STATA verfügt über temporäre (`local`) und ein permanente (`global`) Makros. Ein `global` entspricht so lange seinem zugewiesenen Wert (dieser kann auch alphanumerisch sein), bis das STATA beendet wird. Ein `local` hingegen wird nach Beendigung eines do–Files (sofern es in einem do–File definiert wurde) wieder entfernt. Auf den Wert eines `global` kann mit dem Namen des `global` nach einem Dollar Zeichen zugegriffen werden. Auf den Wert eines `local` kann mit dem Namen in (einfachen) Anführungszeichen zugegriffen werden[2].

```
** Setzen eines permanenten und eines tempor"aren Makros
. global bsp "/mnt/disc3/STATA/introduction2stata/data"
. dis "$bsp"
/mnt/disc3/STATA/introduction2stata/
. local bsp "/mnt/disc3/STATA/introduction2stata/data"
. dis "`bsp'"
/mnt/disc3/STATA/introduction2stata/data
```

2.2 Wechseln des Arbeitsverzeichnisses

STATA selbst arbeitet in einem Arbeitsverzeichnis, wo es Datensätze speichert, wenn kein Pfad eingegeben wird. Dieses Arbeitsverzeichnis kann verändert werden. Mit Hilfe des Befehls `cd` (`change direcory`) ist es möglich, auf der STATA-Oberfläche selbst zwischen Laufwerken und Ordnern zu wechseln. Um das aktuelle Arbeitsverzeichnis anzuzeigen, kann `pwd` eingegeben werden. Mit `cd drive:/pathname` bzw. `cd pathname` kann ins entsprechende Laufwerk und/oder Verzeichnis gewechselt werden. Mit `cd ..` wird in das jeweilige übergeordnete Verzeichnis gewechselt. Mit dem Befehl `ls` wird der Inhalt des aktuellen Arbeitsverzeichnisses angezeigt.

2.3 Das Einlesen bestehender Daten

Das Einlesen von bereits bestehenden STATA-Datensätzen erfolgt entweder über das Funktionsfeld in der Menüleiste oder über den Befehl `use <filename.dta>`[3] im Eingabefeld. Beispiele:

```
. global data "/mnt/disc3/STATA/introduction2stata/data"
. cd $data/mnt/disc3/STATA/introduction2stata/
. use beispieldatensatz.dta
ODER
. use /mnt/disc3/STATA/introduction2stata/data/beispieldatensatz.dta
ODER
. use $data/beispieldatensatz.dta
```

[2] Das korrekte Setzen dieser Anfürhungszeichen ist etwas mühsam, da auf der linken Seite des `local` ein einfaches Anführungszeichen ‘ stehen muss und auf der rechten Seite des `local` ein Apostroph ’ stehen muss.

[3] <`filename.dta`> bezeichnet im folgenden jeweils den Dateinamen *und* den Pfadnamen, sofern notwendig (wenn etwa auf eine Datei zugegriffen werden soll, welche sich nicht im aktuellen Arbeitsverzeichnis abgelegt ist. Siehe Punkt 2.2 zum Wechseln des Arbeitsverzeichnisses.

2.4 Das Einlesen von Daten aus einem anderen Datenformat

Daten können auch eingelesen werden, wenn sie nicht im STATA–Format[4] vorliegen (text bzw. ASCII Format). Siehe dazu `help infiling`.

2.5 Das manuelle Einlesen von Daten

Das Einlesen neuer Daten erfolgt manuell über den Dateneditor, der in der Menüleiste angewählt oder mit dem Befehl `edit` aufgerufen werden kann. Alternativ können Daten auch (umständlicher) mit dem Befehl `input` manuell eingelesen werden.

2.6 Das Speichern von Daten

Das Speichern von Daten erfolgt entweder über die Menüleiste oder über `save <filename.dta>` um das file unter einem neuen Namen abzuspeichern bzw. `save <filename.dta>, replace` um das file unter dem bestehenden Namen abzuspeichern. Sobald man Variablen und / oder Beobachtungen aus dem Datensatz löscht, sollte man sich überlegen, den Datensatz unter einem neuen Namen abzuspeichern. Wird der geänderte Datensatz mit `save, replace` abgespeichert, werden alle Änderungen entsprechend übernommen!

```
. save  beispieldatensatz.dta, replace
file beispieldatensatz.dta saved
```

Mit dem Befehler `compress` wird der Datensatz – sofern möglich – komprimiert, indem Variablen in einem "platzsparenderen" Format abgespeichert werden (sofern damit kein Informationsverlust verbunden ist).

2.7 Weitere Befehle auf der Betriebssystemebene

Weitere wichtige Befehle auf der Betriebssystemebene sind `erase`, mit welchem Dateien gelöscht werden können, `mkdir`, mit welchem Ordner angelegt werden können, `copy`, mit welchem Dateien kopiert werden können, sowie `type`, mit welchem der Inhalt von Dateien angezeigt werden kann.

Auf einem Unix–basierten Rechner kann mittels einem Ausrufezeichen auf Unix Befehle zugegriffen werden. Beispielsweise kann ein Datensatz mittels `!compress <filename.dta>` bzw. `!uncompress <filename.dta.Z>` komprimiert bzw. dekomprimiert werden.

[4] An der Dateiendung `.dta` zu erkennen.

3 Zwei wichtige features: log-files und do-files

Bevor wir uns einigen wichtigen Befehlen zur Datenaufbereitung und –auswertung zuwenden, sollen zwei wichtige "features" von STATA dargestellt werden, welche die Arbeit mit dem Programm um einiges erleichtern und insbesondere die Reproduzierbarkeit der eigenen Ergebnisse garantieren.

3.1 Das Aufzeichnen von "sessions" mittels log-files

Ein wichtiges Hilfsinstrument bei der Arbeit mit STATA sind die sogenannten log-files, mit deren Hilfe sich ganze Arbeitssessions aufzeichnen lassen. Dies geschieht zu Beginn der Arbeit, indem man mittels dem Befehl `log using <filename.log>` ein solches log-file startet bzw. ein bestehendes log-file überschreibt `(, replace)` oder erweitert `(, append)`. Mittels `log off` und `log on` kann das log-file temporär unterbrochen und wiederaufgenommen werden. Beendet wird das log-file mit dem Befehl `log close`. Mit dem Befehl `view <filename.log>` der Inhalt des logfiles betrachtet werden. Ein log-file kann auch mit einem beliebigen Text-Editor angesehen werden.

```
. log using test.log, replace
(note: /mnt/disc3/STATA/introduction2stata/logfiles/test.log not found)
-------------------------------------------------------------------------------
 log:  /Users/oli/Lehrstuhl/STATA/STATA/introduction2stata/logfiles/test.log
 log type:  text
 opened on:  13 Mar 2006, 14:23:36


 Alles was hier geschrieben wird, wird im log-file gespeichert.


 . log close
       log:  /mnt/disc3/STATA/introduction2stata/logfiles//test.log
  log type:  text
 closed on:  13 Mar 2006, 15:03:30
-------------------------------------------------------------------------------
```

3.2 Das Erstellen und Aufrufen von Programmabläufen mittels do-files

Das zweite wichtige Hiflsmittel in STATA sind die sog. do-files, mit denen ganze Programmroutinen, aber auch kurze, aber umständliche Befehlszeilen (etwa das Erstellen gewisser Grafiken), als Skripte erstellt, abgespeichert und zu einem späteren Zeitpunkt abgeändert, ergänzt und wiederaufgerufen werden können. Über die Menüleiste oder mit dem Befehl `doedit` gelangt man in den Do-file editor, in dem solche Skripten in der normalen STATA-Programmsprache erstellt und abgespeichert werden können.
Am besten speichert man die do-files nach einem bestimmten Schema, beispielsweise: "Name do-file"."Benutzter Datensatz".do. Zudem erweist es sich oftmals als hilfreich, am "Kopf" jedes Skriptes die folgenden Informationen festzuhalten (solche Informationszeilen jeweils mit einem * beginnen; Längere Kommentare können auch mit /* ... */ angebracht werden)[5]. Beispielsweise:

```
*** Do-file <Name>:<Inhalt>
*** Datum, Autor(In)
```

Auch sinnvoll ist es, sowohl für die log-files als auch für die do-files einen eigenen Ordner zu benützen.

[5] Siehe auch das Beispiel in Anhang B.

3.3 Reproduzierbarkeit der eigenen Ergebnisse

Um die eigenen Ergebnisse zu einem späteren Zeitpunkt jederzeit wieder reproduzieren zu können (etwa weil nachträglich Änderungen im Design vorgenommen werden müssen oder durch Datenverlust etc.), ist es absolut notwendig, sämtliche Aufbereitungen und Berechnungen (inkl. Schätzungen und Grafiken, etc.) mittels do–files zu dokumentieren.

3.4 profile.do

Ein spezielles do–file ist das sogenannte "profile–dofile" (sic), welches bei jedem Aufstarten von STATA direkt ausgeführt wird. Dies ist insofern hilfreich, als damit beispielsweise direkt beim Starten des Programmes globale Makros definieren lassen, um einfach zwischen verschiedenen Ordnern hin und her zu wechseln, oder direkt der Arbeitsspeicher gesetzt werden kann.

Das `profile.do` ist ein normales do–file, abgesehen davon, dass es `profile.do` heissen **muss** und nicht an einem beliebigen Ort abgespeichert werden kann (siehe dazu `help profile`).

4 Allgemeine Befehlssyntax

In diesem Kapitel und den folgenden werden die wichtigsten Befehle für die Arbeit mit STATA erläutert. Es geht dabei einerseits um Befehle für die Datenaufbereitung und andererseits zur Datenanalyse. Zur besseren Veranschaulichung wird ein kleiner fiktiver Datensatz verwendet (`beispieldatensatz.dta`).

STATA besitzt eine einfache und konsistente Befehlssyntax, welche mit wenigen Ausnahmen der folgenden Basisform folgt (siehe `help language`):

```
[by varlist1:] command [varlist2] [weight] [if exp] [in #/#] [using file], [options]
```

Die meisten Befehle in STATA lassen sich abkürzen (was ebenfalls die Arbeitsgeschwindigkeit erhöht). Aus `help 'command'` wird ersichtlich ob und wie ein Befehl abgekürzt werden kann (der unterstrichene Teil des Befehls ist die minimal notwendige Eingabe für den entsprechenden Befehl). Der Befehl `summarize` etwa lässt sich sich auch als `su` schreiben.

4.1 Der `by` Befehl

Mit der `by` Anweisung wird der Befehl (innerhalb von `varlist1`) wiederholt ausgeführt. Damit das funktioniert, müssen die Daten nach `varlist1` sortiert sein. Alternativ kann auch `bysort` verwendet werden, was die vorgängie Sortierung erübrigt. Im folgenden Beispiel wird der `summarize` Befehl für Männer und Frauen separat durchgeführt:

```
. bysort sex: summarize inc

-------------------------------------------------------------------------------
-> sex = weiblich

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         inc |        10        4097    1274.005       2500       6500

-------------------------------------------------------------------------------
-> sex = maennlich

    Variable |       Obs        Mean    Std. Dev.       Min        Max
-------------+--------------------------------------------------------
         inc |        10        4283    1011.655       2100       5750
```

4.2 Variablenlisten (`varlist`)

Eine `varlist` ist eine Liste von (existierenden) Variablen. Beispiele:

```
. *** Eine einzelne Variable:
. quietly summarize sex
. *** Mehrere Variablen:
. quietly summarize sex educ birth
. *** Mehrere Variablen (alle Variablen zwischen sex und inc):
. quietly summarize sex - inc
. *** Mehrere Variablen (alle Variablen mit der Endung -c):
. quietly summarize *c
```

Bei der Verwendung von `var_von - var_bis` spielt die Anordnung der Variablen im Datensatz offensichtliche eine Rolle. Wie dies geändert werden kann, wird in Abschnitt 5.1.1 erläutert.

4.3 Gewichtung (`weight`)

Beobachtungen können mit der Option `weight` gewichtet werden. Es stehen dabei verschiedene Arten von Gewichtung zur Verfügung (siehe `help weights`). Hat man in einem Datensatz beispielsweise duplizierte Beobachtungen, können `frequency weights` verwendet werden, siehe dazu untenstehendes Beispiel:

4.4 Einschränken der Stichprobe (`if` und `in` Anweisungen)

Soll eine Anweisung nicht über den gesamten Datensatz ausgeführt werden, kann dies mit der `if` oder der `in` Anweisung gemacht werden.

```
. use $data/beispieldatensatz_haeufigkeiten.dta, clear
. summarize inc [fw = h]
```

Variable	Obs	Mean	Std. Dev.	Min	Max
inc	850	3913.165	1206.515	2100	6500

4.5 Verwenden eines zusätzlichen Datensatzes

Bestimmte Anweisungen verlangen – neben dem bereits geöffneten Datensatz – einen zweiten Datensatz. Dieser kann innerhalb der Option `using file` spezifiziert werden (inkl. Pfadangabe).

4.6 Weitere Optionen

Die meisten Befehle weisen zusätzliche (und offensichtlich unterschiedliche) Optionen auf. Diese sind immer mit `help command` einzusehen und sind durch ein Komma vom eigentlichen Befehl abgetrennt.

5 Datenaufbereitung

5.1 Informationen zum geöffneten Datensatz

Einen Überblick über alle Variablen im Datensatz und über die Grösse des Datensatzes (Anzahl Beobachtungen, Anzahl Variablen) erhält man über den Befehl `describe`.

```
. use /mnt/disc3/STATA/introduction2stata/data/beispieldatensatz.dta
. describe
Contains data from beispieldatensatz.dta
  obs:            20
 vars:             5                          13 Mar 2006 14:11
 size:           360 (99.9% of memory free)
------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
id              byte   %8.0g
sex             byte   %8.0g
educ            byte   %8.0g
inc             int    %8.0g
birth           str9   %9s
------------------------------------------------------------------------------
Sorted by:  id
```

Mit der Option `short` wird nur der obere Teil der Tabelle ausgegeben.

```
. describe, short
Contains data from beispieldatensatz.dta
  obs:            20
 vars:             5                          20 Mar 2006 14:24
 size:           360 (99.9% of memory free)
Sorted by:  id
```

5.1.1 Variablen ordnen

Mit `aorder` können die Variablen alphabetisch geordnet werden. Mit `order varlist` werden die Variablen in der angegebenen Reihenfolge an den Anfang des Datensatzes verschoben.

5.2 Anmerkungen zum Datensatz generieren

Mit dem Befehl `notes` können Anmerkungen zum Datensatz und / oder zu spezifischen Variablen an den Datensatz angefügt und eingesehen werden. Mit `TS` ("time stamp") kann zusätzlich Datum und Zeit eingefügt werden. Siehe dazu das folgende Beispiel:

```
. note: Das ist der Beispieldatensatz. /* Anmerkung erzeugen */
. note: TS - Das ist der Beispieldatensatz. /* Anmerkung mit einer "time stamp" (TS) erzeugen */
. note inc: Hier steht das Einkommen drin. /* Anmerkung zu einer Variablen erzeugen */
. notes /* Anmerkungen anzeigen */
_dta:
  1.  Das ist der Beispieldatensatz.
  2.  16 Mar 2006 18:01 - Das ist der Beispieldatensatz.
inc:
  1.  Hier steht das Einkommen drin.
```

Eine zweite Möglichkeit, Anmerkungen zum Datensatz zu generieren, ist über `label data ''Anmerkung''`. Dies eignet sich insbesondere dafür, den zentralen Inhalt des Datensatzes hineinzuschreiben (z.B. die Stichprobe, welche dem Datensatz zugrunde liegt).

5.3 Daten sortieren

Die Daten lassen sich mit dem Befehl `sort <varname(s)>` beliebig nach einer oder mehreren Variable(n) aufsteigend sortieren. Um den Befehl `[by <varlist>:]` verwenden zu können, müssen die Daten entsprechend sortiert sein.

Um auch in absteigender Reihenfolge sortieren zu können, muss der Befehl `gsort <+/- varname>` verwendet werden, wobei ein Minuszeichen vor einer Variablen angibt, dass innerhalb dieser Variablen absteigend sortiert werden soll.

5.4 Erstellen von neuen Variablen, Datentransformationen

Mit dem Befehl `generate <newvar>= exp` lassen sich neue Variablen berechnen, wobei `exp` für den entsprechenden Befehl oder die entsprechende Funktion steht. Bestehende Variablen lassen sich mit `replace <varname> = exp` ersetzen. Siehe `help functions` und `help egen` zu verschiedenen Funktionen.

5.4.1 Variablenformat

STATA verfügt über fünf verschiedene Variablenformate für numerische Variablen, diese können mit `help datatypes` eingesehen werden. Das Variablenformat ist von der Anzahl Stellen innerhalb einer Variablen abhängig – dies können Vor– oder Nachkommastellen sein. Dazu ein einfaches Beispiel:

```
. gen byte inc2 = inc
. fl in 1/2
```

	id	sex	educ	inc	birth	inc2
1.	1	weiblich	2	2500	28oct1967	.
2.	2	maennlich	1	2100	5aug1971	.

Da die Variable `inc` Werte aufweist, welche grösser als `100` sind, kann die neue Variable `inc2`, welche im `byte` Format erstellt wurde, diese Werte nicht korrekt speichern. Das Problem taucht nicht auf, wenn die neue Variable in diesem Beispiel im `int` Format erstellt wird:

```
. gen int inc3 = inc
. fl inc* in 1/2
```

	inc	inc2	inc3
1.	2500	.	2500
2.	2100	.	2100

Standardmässig werden neue Variablen im `float` Format erstellt (damit entsteht in der Regel obiges Problem nicht). Man kann dann vor dem Abspeichern des Datensatzes mit `compress` die Variablen komprimieren.

Alphanumerische Variablen werden im `string` Format gespeichert.

5.4.2 Logische Operatoren

Logische Operatoren sind:

& | > < == >= <= !=

Wobei & ein logisches UND, | ein logisches ODER und != ein logisches UNGLEICH bezeichnet[6]. Siehe `help operators` zu weiteren Informationen zu logischen Operatoren.

Anwendung: Das Generieren von Dummy-Variablen

Mit der Hilfe der logischen Operatoren können auf einfache Art und Weise {0,1}-kodierte Dummy-Variablen erstellt werden:

`generate <new dummy> = <logical exp>`

Beobachtungen, für welche die logischen Bedingungen wahr sind, wird der Wert 1 zugeschrieben, den übrigen Beobachtungen der Wert 0. Im Beispieldatensatz bezeichnet etwa der Wert 2 auf der Variablen sex, dass es sich um eine männliche Person handelt. Um nun eine dummy-Variable für Männer zu erstellen, wird folgende Anweisung ausgeführt:

```
. generate male = sex == 2
. tabulate sex
```

sex	Freq.	Percent	Cum.
1	10	50.00	50.00
2	10	50.00	100.00
Total	20	100.00	

n.b. Der Befehl `tabulate` tabelliert die entsprechende Variable und zeigt die entsprechenden Häufigkeiten auf.

5.4.3 Arithmetische Operatoren

Arithmetische Operatoren sind:

+ − * / ^ ( )

Beispiel: Erstellen einer Variable, welche das quadrierte Einkommen enthält:

```
. generate incsq = inc^2
```

Siehe `help expressions` zu weiteren Informationen zu arithmetischen Operatoren.

5.4.4 Mathematische Funktionen

Zu den wichtigsten verfügbaren mathematischen Funktionen gehören:

- `abs(x)`; der Absolutwert von `x`.
- `sqrt(x)`; die Wurzel aus `x`.
- `exp(x)`; entspricht e^x.
- `ln(x)`; der natürlich Logarithmus von `x`.
- `max(x1,x2,...,xn)`; das Maximum von `x1` bis `xn`.

[6] Hinweis: In der STATA-Befehlssprache wird = als Zuweisung benutzt und == als Gleichheit.

- `min(x1,x2,...,xn)`; das Minimum von `x1` bis `xn`.
- round(x, y); rundet `x` in Einheiten von `y`.

Siehe `help mathfun` zu weiteren mathematischen Funktionen.

5.4.5 Wahrscheinlichkeitsverteilungen und Dichtefunktionen

Unter `help probfun` sind die verfügbaren Funktionen zu Wahrscheinlichkeitsverteilungen und Dichtefunktionen einsehbar.

Beispiel:

```
. *** Berechnen von Pr(-1.96 <= Z <= 1.96)
. display norm(1.96) - norm(-1.96)
.95000421
. *** Berechnen von z, so dass Pr(Z <= z) = 0.95
. display invnorm(.95)
1.6448536
```

5.4.6 Funktionen für alphanumerische Variablen

Funktionen für alphanumerisch kodierte Variablen sind unter `help strfun` einzusehen. Die wichtigsten davon sind:

- `real()`: Enthält eine `string` Variable (versehentlich) nur numerische Informationen, kann mit `generate <newvar> = real(string var)` daraus eine numerische Variable generiert werden.
- `substr()`: Damit kann ein (beliebiger) Teil einer `string` Variablen extrahiert werden.
- `encode`: Mit `encode` kann eine Variable im `string` Format in eine numerische Variable überführt werden. Die neu kreierten numerischen Ausprägungen werden gleichzeitig mit den bisherigen alphanumerischen Ausprägungen bezeichnet (siehe Beispiel unten).
- `decode`: Macht das Umgekehrte von `encode`.

Beispiel:

```
. generate year = substr(birth,-4,4)  /* Extrahiert das Jahr aus der string-Variable birth */
. generate month = substr(birth,-7,3)  /* Extrahiert den Monat */
. fl birth year month in 1/3
```

	birth	year	month
1.	28oct1967	1967	oct
2.	5aug1971	1971	aug
3.	5dec1968	1968	dec

```
. generate year_real = real(year)  /* generiert eine numerische Variable */
. encode month, generate(month_real) /* generiert eine numerische Variable und erzeugt labels */
. fl birth year* month* in 1/3
```

	birth	year	year_real	month	month_~l
1.	28oct1967	1967	1967	oct	oct
2.	5aug1971	1971	1971	aug	aug
3.	5dec1968	1968	1968	dec	dec

5.5 Beschriftung von Variablen

Variablen lassen sich mit dem Befehl `rename <old varname> <new varname>` umbennen. Zudem lässt sich über den Befehl `label variable <varname> ''label''` jeder Variable ein label mit bis zu 80 Zeichen zuweisen, wohingegen es sinnvoll ist, die eigentlichen Variablennamen relativ kurz zu halten.

```
. rename edu Ausbildung
. label variable birth "Geburtsdatum"
. d birth
              storage  display     value
variable name   type   format      label      variable label
------------------------------------------------------------------------------
birth           str9   %9s                    Geburtsdatum
```

Die einzelnen Ausprägungen von kategorialen Variablen lassen sich ebenfalls bezeichnen. Dies erfolgt über zwei Schritte:

1. Man definiert mit dem Befehl `label define <labelname> # ''label'' [# ''label'']` zunächst eine label-Variable.
2. Danach weist man diese der entsprechenden Variable zu mittels `label values <varname> <labelname>.`

```
. label define label_Ausbildung 1 "Primaerstufe" 2 "Sekundaerstufe" 3 "Tertiaerstufe"
. label values Ausbildung label_Ausbildung
. tabulate Ausbildung
   Ausbildung |      Freq.     Percent        Cum.
--------------+-----------------------------------
 Primaearstufe |          5       25.00       25.00
Sekundaerstufe |         10       50.00       75.00
 Tertiaerstufe |          5       25.00      100.00
--------------+-----------------------------------
         Total |         20      100.00
```

5.5.1 Anzeigeformat von Variablen ändern

Unabhängig vom Format, in welchem Variablen gespeichert werden, kann das Format geändert werden, in welchem die Variablen im Output–Fenster angezeigt werden. Dies kann man mit dem Befehl `format var %fmt` tun.

Beispiel:

5.6 Löschen von Variablen und Beobachtungen

Einzelne oder wenige Variablen lassen sich mit dem Befehl `drop <varname(s)>` löschen. Wenn umgekehrt nur eine oder wenige Variablen nicht gelöscht werden sollen, verwendet man besser den `keep <varname(s)>` Befehl. Beispiel (Löschen der Variable sex):

```
. drop sex
```

Beide Befehle lassen sich analog für die Selektion von Beobachtungen verwenden, wobei hier die Befehle entsprechend mit dem `[if exp]` oder dem `[in range]` Befehl benutzt werden müssen.

Beispiel (Löschen aller weiblichen Beobachtungen):

```
. drop if sex == 1
(10 observations deleted)
```

Wie bereits erwähnt ist beim Löschen von Beobachtungen und / oder Variablen zu beachten, dass beim Überschreiben des Datensatzes die entsprechenden Änderungen definitiv übernommen werden und nicht mehr rückgängig gemacht werden können.

5.7 Erstellen eines Datensatzes von statistischen Kennwerten

Oftmals möchte man einen Datensatz komprimieren, indem man für verschiedene Gruppen entsprechende statistische Kennwerte berechnet (bsp. den Mittelwert über Gruppen von Personen). Der entsprechende Befehl lautet:

`collapse (stat) <newvarname1> = <varname1> [(stat) <varname2>], [by <varlist>].`

Mit dem `collapse`Befehl kann entweder eine Variable mit einem neuen Namen generiert werden (`<newvarname1>`) oder der bestehende Name übernommen werden[7]. `(stat) <varname>` bezeichnet dabei, welcher Kennwert für welche Variable berechnet werden soll, in `[by <varlist>]` wird angegeben, über welche Variable(n) die Daten komprimiert werden sollen. Aus dem Beispieldatensatz lässt sich etwa ein komprimierter Datensatz erstellen, welcher nur noch den Mittelwert und die Standardabweichung des Einkommens enthält, und zwar getrennt nach Geschlecht:

```
. collapse (mean) mean_inc = inc (sd) sd_inc = inc , by(sex)
. fl
```

	sex	mean_inc	sd_inc
1.	1	4097	1274.01
2.	2	4283	1011.65

5.8 Das Zusammenfügen von Datensätzen, wenn in beiden Datensätzen dieselben Variablen für unterschiedliche Beobachtungen vorliegen

Solche Datensätze lassen sich einfach über den Befehl `append using <filename.dta>` zusammenfügen, wobei der Datensatz in `<filename.dta>` am Ende des bereits geöffneten Datensatzes angefügt wird.

Beispiel (Männer und Frauen sind in unterschiedlichen Datensätzen gespeichert):

```
. global data /mnt/disc3/STATA/introduction2stata/data
. use  $data/beispieldatensatz_men.dta, clear
. tabulate sex
```

sex	Freq.	Percent	Cum.
maennlich	10	100.00	100.00
Total	10	100.00	

```
. append using  $data/beispieldatensatz_women.dta
(label educ already defined)
(label sex already defined)
```

[7] Wenn mehr als ein statistischer Kennwert derselben Variable berechnet werden soll, kann höchstens eine der Variablen den bisherigen Namen beibehalten.

```
. tabulate sex
```

sex	Freq.	Percent	Cum.
weiblich	10	50.00	50.00
maennlich	10	50.00	100.00
Total	20	100.00	

5.9 Das Zusammenfügen von Datensätzen, wenn für dieselben Beobachtungen unterschiedliche Variablen vorliegen

In diesem Fall müssen die beiden Datensätze mit dem `merge` Befehl zusammengefügt werden. Damit die Datensätze zusammengefügt werden können, muss damit mindestens eine Variable in beiden Datensätzen existieren, welche das entsprechende matching erlaubt. Der Befehl lautet:
`merge <varname> using <filename.dta>`
Unter `<varname>` wird die Variable angegeben (es kann hier auch mehr als eine Variable stehen), über welche die Daten verknüpft werden sollen. Unter `<filename.dta>` wird der Datensatz angegeben, welcher an den bereits geöffneten Datensatz angefügt werden soll.

Wichtig: Es müssen vor dem Zusammenfügen **beide** Datensätze nach der Variable in `<varname>` sortiert werden[8]. STATA generiert automatisch eine Variable namens `_merge`, welche die Informationen zu den matches enthält. Eine `3` bezeichnet Beobachtungen mit Daten aus beiden Datensätzen.

Sollen nur diejenigen Beobachtungen verwendet werden, welche im `master` Datensatz enthalten sind, kann dies mit der Option `nokeep` angefordert werden. In diesem Fall werden Beobachtungen, welche **nur** im `using` Datensatz enthalten sind (nicht aber im `master` Datensatz) nicht angefügt.

Beispiel: Das Geburtsdatum befindet sich in einem separaten Datensatz (`geburtstag.dta`) und wurde aus dem Beispieldatensatz gelöscht. Um diese Information wieder hinzuzufügen, müssen beide Datensätze nach der Verknüpfungsvariable (`id`) sortiert sein. Mit dem `merge`–Befehl wird dann der Geburtstag wieder an den Beispieldatensatz angefügt.

```
. global data /mnt/disc3/STATA/introduction2stata/data
. use  $data/geburtstag.dta
. sort id
. describe
Contains data from geburtstag.dta
  obs:            20
 vars:             2                          14 Mar 2006 13:55
 size:           280 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              byte   %8.0g
birth           str9   %9s
-------------------------------------------------------------------------------
Sorted by:  id
. save $data/geburtstag.dta, replace
file geburtstag.dta saved
. use $data/beispieldatensatz_ohne_birth.dta
. sort id
```

[8] Ab Version 9.0 kann mit der Option `sort` direkt der Datensatz sortiert werden, der an den bereits geöffneten Datensatz angefügt werden soll

```
. describe
Contains data from beispieldatensatz_ohne_birth.dta
  obs:            20
 vars:             4                          14 Mar 2006 13:56
 size:           180 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              byte   %8.0g
sex             byte   %8.0g
educ            byte   %8.0g
inc             int    %8.0g
-------------------------------------------------------------------------------
Sorted by:  id
. merge id using  $data/geburtstag.dta
. describe
Contains data from beispieldatensatz_ohne_birth.dta
  obs:            20
 vars:             6                          14 Mar 2006 13:56
 size:           380 (99.9% of memory free)
-------------------------------------------------------------------------------
              storage  display     value
variable name   type   format      label      variable label
-------------------------------------------------------------------------------
id              byte   %8.0g
sex             byte   %8.0g
educ            byte   %8.0g
inc             int    %8.0g
birth           str9   %9s
_merge          byte   %8.0g
-------------------------------------------------------------------------------
Sorted by:
     Note:  dataset has changed since last saved
. tab _merge
     _merge |      Freq.     Percent        Cum.
------------+-----------------------------------
          3 |         20      100.00      100.00
------------+-----------------------------------
      Total |         20      100.00
```

n.b. Wenn die Variable `_merge` wie in unserem Beispiel nur Dreien enthält, heisst dies, dass in beiden Datensätzen die selben Beobachtungen enthalten sind.

5.10 Transponieren eines Datensatzes und Wechseln zwischen dem "long"– und dem "wide"–Format

Ein Datensatz kann mittels `xpose` transponiert werden (d.h. Spalten und Zeilen werden getauscht).

Mittels dem `reshape` Befehl kann zwischen dem "long" (Jede Beobachtung steht in einer separaten Zeile) und dem "wide" Format (Ausprägungen derselben Beobachtung werden in dieselbe Zeile verschoben) hin– und her gewechselt werden.

Dazu ist ein Beispiel hilfreich. Im File `beispieldatensatz_long.dta` ist das Einkommen für jede Person zu zwei Jahren vorhanden. Der Datensatz ist im `long` Format, d.h. jede Person ist zweimal im Datensatz enthalten (da das Einkommen variiert zwischen den beiden Jahren). Im Beispiel ändern wir mit der `reshape wide` Anweisung in das `wide` Format, in welchem zwei unterschiedliche Einkommensvariablen erzeugt werden (für jedes Jahr eine). Danach wechseln wir wieder zurück in das `long` Format.

```
. use $data/beispieldatensatz_long.dta, clear
. sort id year
. fl in 1/4
```

	id	sex	educ	inc	birth	year
1.	1	weiblich	2	2500	28oct1967	2005
2.	1	weiblich	2	2575	28oct1967	2006
3.	2	maennlich	1	2100	5aug1971	2005
4.	2	maennlich	1	2163	5aug1971	2006

```
. reshape wide inc, i(id) j(year)
(note: j = 2005 2006)
Data                               long   ->   wide
-----------------------------------------------------------------------------
Number of obs.                       40   ->      20
Number of variables                   6   ->       6
j variable (2 values)              year   ->   (dropped)
xij variables:
                                    inc   ->   inc2005 inc2006
-----------------------------------------------------------------------------
. fl in 1/2
```

	id	inc2005	inc2006	sex	educ	birth
1.	1	2500	2575	weiblich	2	28oct1967
2.	2	2100	2163	maennlich	1	5aug1971

```
. reshape long inc, i(id) j(year)
(note: j = 2005 2006)
Data                               wide   ->   long
-----------------------------------------------------------------------------
Number of obs.                       20   ->      40
Number of variables                   6   ->       6
j variable (2 values)                     ->   year
xij variables:
                          inc2005 inc2006   ->   inc
-----------------------------------------------------------------------------
. fl in 1/4
```

	id	year	inc	sex	educ	birth
1.	1	2005	2500	weiblich	2	28oct1967
2.	1	2006	2575	weiblich	2	28oct1967
3.	2	2005	2100	maennlich	1	5aug1971
4.	2	2006	2163	maennlich	1	5aug1971

5.11 Indivudaldaten aus Häufigkeitsdaten erstellen und vice versa

Mit dem Befehl `expand exp` können aus Häufigkeitsdaten Individualdaten erstellt werden, sofern in `exp` die Häufigkeiten der Variablenmuster enthalten sind. Der Befehl `contract <varlist>` macht den umgekehrten Vorgang.

```
. use beispieldatensatz_haeufigkeiten.dta, clear
. expand h /* h beinhaltet die absoluten Haeufigkeiten */
(830 observations created)
. describe, short
Contains data from beispieldatensatz_haeufigkeiten.dta
  obs:           850
 vars:             6                          21 Mar 2006 11:54
 size:        16,150 (99.9% of memory free)
Sorted by:
     Note:  dataset has changed since last saved
. contract id-birth
. describe, short
Contains data from beispieldatensatz_haeufigkeiten.dta
  obs:            20
 vars:             6                          21 Mar 2006 11:54
 size:           380 (99.9% of memory free)
Sorted by:  id  sex  educ  inc  birth
     Note:  dataset has changed since last saved
```

5.12 Das Überführen von Datumsangaben in ein numerisches Format

Alphanumerische Datumsangaben lassen sich leicht in ein numerisches Format überführen, mit dem Berechnungen durchgeführt werden können. Dies geschieht mit der Funktion `generate <newvarname >= date(<string date>,''format'')`. Einzelne Daten können mittels der Funktion `mdy(m,d,y)` berechnet werden. Dadurch können dann einfach Zeitspannen berechnet werden. Dieses STATA-interne numerische Datumsformat weist für den 1. Januar 1960 den Wert 0 auf. Beispiel: (i) Umformen der alphanumerischen Geburtstagsangabe in das numerische Format, (ii) Generieren einer Variable mit dem Datum des 14. März 2002 (Stichtag), (iii) Berechnen einer Altersvariable (Alter zum Stichtag).

```
. global data /mnt/disc3/STATA/introduction2stata/data
. use  $data/beispieldatensatz.dta
* Generieren einer neuen Variable, welche das Datum im STATA Format enth"alt
* "dmy" heisst, day, month, year und definiert das Datumsformat.
. generate birthdate = date(birth, "dmy")
* Generieren eines locals, welcher das Datum vom 1.7.2000 enth"alt.
. local Datum = mdy(7,1,2000)
* Generieren einer Variable, welche das Datum vom 14.3.2000 enth"alt.
. generate date  = mdy(3,14,2006)
. generate age = (date - birthdate + 1) / 365.25
. fl birth birthdate date age in 1/5
```

	birth	birthdate	date	age
1.	28oct1967	2857	16874	38.379192
2.	5aug1971	4234	16874	34.609172
3.	5dec1968	3261	16874	37.273101
4.	24jul1972	4588	16874	33.639973
5.	25dec1969	3646	16874	36.219028

```
. save $data/beispieldatensatz_age.dta
file beispieldatensatz_age.dta saved
```

n.b. Der Befehl `in 1/5` bewirkt, dass nur die ersten 5 Beobachtungen aufgelistet werden.

6 Programmierung

Mittels einem dofile kann STATA kleine und grosse Programme ausführen. Dieser Abschnitt gibt eine kleine Übersicht über die wichtigsten Programmierroutinen. Siehe dazu auch das Handbuch `Programming`.

6.1 Die `more` Meldung ausschalten

Bei der Ausgabe von längerem Output pausiert STATA (was durch die `--more--` Nachricht angezeigt wird). Bei längeren Programmen möchte man diese Pausen ausschalten (da man jedes mal die Leertaste drücken muss, damit das Programm weiterläuft). Dies kann man mit `set more off` tun. Mit `set more on` kann man die Pause wieder einstellen.

6.2 Ende einer Befehlszeile

Standardmässig endet für STATA eine Befehlszeile mit einem Zeilenumbruch. Bei längeren Befehlszeilen ist dies teilweise etwas unpraktisch. Deshalb kann innerhalb eines do–files das Ende einer Befehlszeile auf ein Semikolon `;` umgestellt werden, so dass dieselbe Anweisung nun auf verschiedene Zeilen verteilt werden kann. Man erreicht dies mit dem Befehl `#delimit ;` (auf den normalen Zeilenumbruch kann mit dem Befehl `#delimit cr` zurückgestellt werden):

```
. #delimit ;
. use
. beispieldatensatz.dta,
. clear;
. #delimit cr
```

6.3 Schleifen ("loops")

Eine Schleife dient dazu um einen Befehl oder ein Teil eines Skriptes wiederholt ausgeführt werden soll. Eine Schleife erleichtert zum Beispiel das Berechnen von Arbeitslosenquoten für jedes Quartal von 1990 bis 1999. Das Vorgehen soll anhand eines Beispiel verdeutlicht werden.

```
* Der local Jahr nimmt in Einerschritten (der Wert in der Klammer) Werte an von 1990 bis 1999
. forvalues Jahr = 1990(1)1999 {
. forvalues Monat = 2(3)11 {
. local Datum = mdy(‘Monat’,10,‘Jahr’)
. dis ‘Datum’
. }
. }
```

Der `local` kann auch Werte aus einer (beliebigen, d.h. auch alphanumerischen) Liste annehmen:

```
. foreach Tag in Monat Dienstag Mittwoch Donnerstag Freitag Samstag Sonntag  {
. foreach Tageszeit in Morgen Abend  {
. . dis ‘Tag’ " am "  ‘Tageszeit’
. }
. }
```

Der `local` welcher in jedem Durchlauf einen neuen Wert annimmt, kann im Skript verwendet werden. Als weiters Anschauungsbeispiel dient das Skript `quartalsfiles.do` aus Abschnitt 11.3. Aus dem Beispiel wird auch ersichtlich, dass verschiedene Schleifen verschachtelt werden können.

6.4 `If` und `else (if)` Anweisungen

Sollen bestimmte Anweisungen nur dann ausgeführt werden, wenn bestimmte Bedinungen erfüllt sind, lässt sich dies mittels `if` bzw. `if` und `else` bzw. `if` und `else if` Anweisungen erreichen (siehe `help ifcmd`). Wiederum lässt sich dies am einfachsten anhand eines kleinen Beispiels illustrieren:

```
. local start = mdy(3,20,2006)
. local end = mdy(3,26,2006)
. forvalues datum = ‘start’(1)‘end’ {
  2. if dow(‘datum’) >= 1 & dow(‘datum’) <= 5 {
  3. dis day(‘datum’) " " month(‘datum’) " " year(‘datum’) ": " "Wochentag"
  4. }
  5. else {
  6. dis day(‘datum’) " " month(‘datum’) " " year(‘datum’) ": " "Wochenende"
  7. }
  8. }
20 3 2006: Wochentag
21 3 2006: Wochentag
22 3 2006: Wochentag
23 3 2006: Wochentag
24 3 2006: Wochentag
25 3 2006: Wochenende
26 3 2006: Wochenende
```

6.5 Verifizieren von Bedingungen

Ebenfalls nützlich sind Anweisungen, welche überprüfen, ob bestimmte Bedingungen erfüllt sind oder nicht (in Kombination mit `if`–Anweisungen). Der Befehl `assert exp` überprüft, ob `exp` erfüllt ist nicht. Ist die Bedingung nicht erfüllt, wird eine Fehlermeldung ausgegeben, ansonsten passiert nichts. Siehe dazu folgendes einfaches Beispiel:

```
. use $data/beispieldatensatz.dta, clear
. count
   20
. assert r(N) == 20
. assert r(N) == 19
assertion is false
r(9);
```

Das Problem hierbei ist offensichtlich, dass bei der Ausgabe einer Fehlermeldung ein do–file abgebrochen wird (siehe Beispiel oben). Damit die `assert` Anweisung sinnvoll verwendet werden kann, muss diese in Kombination mit dem `capture` Befehl verwendet werden, da dieser die Ausgabe von allfälligen Fehlermeldungen unterdrückt (und somit das do–file nicht unterbrochen wird). Bei der Verwendung von `capture`ist allerdings aus dieser Eigenschaft heraus grosse Vorsicht geboten! Intern wird allerdings die entsprechende Fehlermeldung (falls `exp` nicht erfüllt ist) in der Systemvariable `_rc` (steht für `return code`) gespeichert. Dazu folgendes Beispiel:

```
. quietly count
. cap assert r(N) == 20
. if _rc == 0 {
.         dis "Anzahl Beobachtungen == 20"
Anzahl Beobachtungen == 20
.         }
. else {
.         dis "Anzahl Beobachtungen != 20"
.         }
```

Mit dem Befehl `confirm` lässt sich u.a. überprüfen, ob ein bestimmter Datensatz existiert, ob bestimmte Variablen existieren oder ob eine Variable ein bestimmtes Format aufweist. Auch

dieser Befehl sollte innerhalb eines do–files zusammen mit der `capture` Anweisung verwendet werden. Dazu ebenfalls ein Beispiel:

```
. *** Der erste Datensatz existiert
. confirm file beispieldatensatz.dta
. *** Dieser Datensatz existiert nicht, deshalb gibt es eine Fehlermeldung
. confirm file beispieldatensatz_frauen.dta
file beispieldatensatz_frauen.dta not found
r(601);
. *** Die Variable existiert, ist allerdings nicht numerisch kodiert
. confirm numeric variable birth
'birth' found where numeric variable expected
r(7);
. *** ... sondern alphanumerisch
. confirm string variable birth
```

6.6 Implizite Nummerierung der Beobachtungen

Mit dem `by(sort)` Befehl können (unter anderem) die beiden Funktionen `_n` und `_N` verwendet werden, welche eine implizite Nummerierung innerhalb der `by` Anweisung erlauben. Diese Funktionen sind insbesondere dann hilfreich, wenn dieselbe Beobachtung mehrfach im Datensatz beobachtet wird. Die Funktionsweise kann anhand des Beispieldatensatzes erläutert werden:

```
. sort sex inc
. drop rank
. by sex: gen int rank = _n
. by sex: gen count = _N
. l id inc rank count if sex == 1
```

	id	inc	rank	count
1.	1	2500	1	10
2.	7	2750	2	10
3.	6	2800	3	10
4.	4	3400	4	10
5.	13	4000	5	10
6.	20	4150	6	10
7.	8	4750	7	10
8.	15	5000	8	10
9.	11	5120	9	10
10.	16	6500	10	10

6.7 Implizit gespeicherte Ergebnisse

Nach Auswertungen (z.B. Regressionen) stehen die meisten Ergebnisse zur weiteren Bearbeitung zur Verfügung, da diese intern abgespeichert werden (darauf basiert etwa die `outreg` Anweisung). STATA unterscheidet hierbei zwischen allgemeinen Befehlen und Schätzverfahren. Die Unterscheidung ist nur insofern wichtig, als Ergebnisse aus ersteren unter `r()` abgelegt werden und Ergebnisse aus letzteren unter `e()`.

Die folgenden beiden Beispiele illustrieren dies:

Beispiel 1: Allgemeiner Befehl (`summarize`, speichert Ergebnisse unter `r()`):

```
. *** quietly unterdr"uckt den Output
. quietly summarize inc
. *** Gespeicherte Ergebnisse anzeigen
. return list
```

```
scalars:
                  r(N) =  20
              r(sum_w) =  20
               r(mean) =  4190
                r(Var) =  1262726.315789474
                 r(sd) =  1123.710957403848
               r(min) =  2100
               r(max) =  6500
               r(sum) =  83800

. *** macro definieren
. local mean = r(mean)

. dis `mean'
4190
```

Beispiel 2: Ergebnis aus Schätzverfahren (Ergebnisse werden in `e()` gespeichert):

```
. xi: qui reg inc i.educ
i.educ            _Ieduc_1-3          (naturally coded; _Ieduc_1 omitted)

. eret list

scalars:
                  e(N) =  20
               e(df_m) =  2
               e(df_r) =  17
                  e(F) =  6.234071107521063
                 e(r2) =  .4231058111521436
               e(rmse) =  902.3084773154324
                e(mss) =  10151070
                e(rss) =  13840730
               e(r2_a) =  .3552359065818076
                 e(ll) =  -162.8527104528289
               e(ll_0) =  -168.3536745611955

macros:
             e(depvar) : "inc"
                e(cmd) : "regress"
            e(predict) : "regres_p"
              e(model) : "ols"

matrices:
                  e(b) :  1 x 3
                  e(V) :  3 x 3

functions:
             e(sample)

. *** Varianz--Kovarianz Matrix erzeugen
. matrix VCE = e(V)

. *** Inhalt der Matrix anzeigen
. matrix list VCE

symmetric VCE[3,3]
             _Ieduc_2    _Ieduc_3       _cons
_Ieduc_2   244248.18
_Ieduc_3   162832.12   325664.24
   _cons  -162832.12  -162832.12   162832.12
```

6.8 Stichprobenziehung

Mit dem Befehl `sample #` kann eine (einfache) `#` % Zufallsstichprobe aus einem bestehenden Datensatz gezogen werden (Achtung: Die nicht–gezogenen Beobachtungen gehen verloren!). Mit der Option `count` kann die absolute Anzahl Beobachtungen angegeben werden, welche gezogen werden sollen. Mit der Option `by()` kann eine geschichtete Stichprobe gezogen werden.

Achtung! Wenn man Ergebnisse aus Stichprobenziehungen reproduzieren möchte, muss man den Startwert des Zufallszahlengenerators **vor** der Stichprobenziehung auf einen bes timmten (aber beliebigen) Wert setzen. Dies kann mit dem Befehl `set seed #` tun, wobei `#` dem Startwert entspricht.

```
. set seed 68027468
. sample 5, count
(15 observations deleted)
. list
```

	id	sex	educ	inc	birth
1.	2	maennlich	1	2100	5aug1971
2.	10	maennlich	2	4050	24feb1947
3.	5	maennlich	3	4500	25dec1969
4.	18	maennlich	2	4890	26apr1965
5.	15	weiblich	3	5000	12oct1966

6.9 Temporäres Abspeichern und Wiederherstellen des Datensatzes

Mit `preserve` kann der Datensatz temporär (im Arbeitsspeicher) abgespeichert werden und mit `restore` (jederzeit) wiederhergestellt werden. Nach einer `restore` Anweisung wird der Datensatz allerdings aus dem Speicher gelöscht (dies kann man mit der Anweisung `restore, preserve` umgehen; hier wird der Datensatz wiederhergestellt und gleich wieder temporär abgelegt).

Mit diesem Befehl kann beispielsweise das Problem gelöst werden, den zugewiesenen Arbeitsspeicher bei einem geöffneten Datensatz zu ändern (was normalerweise nicht gemacht werden kann).

```
. describe, short
Contains data from beispieldatensatz.dta
  obs:            20
 vars:             5                          20 Mar 2006 14:24
 size:           360 (99.9% of memory free)
Sorted by:  id
. set mem 5m
no; data in memory would be lost
r(4);
. preserve
. clear
. set mem 5m
(5120k)
. restore
. describe, short
Contains data from beispieldatensatz.dta
  obs:            20
 vars:             5                          20 Mar 2006 14:24
 size:           360 (99.9% of memory free)
Sorted by:  id
```

6.10 Unterdrücken / Anzeigen von Ergebnissen

Bei längeren Programmen möchte man allenfalls nicht den gesamten Output anzeigen lassen. Dies kann mit dem Voranstellen der Anweisung `quietly` an einen Befehl erreicht werden. Ganze Programmroutinen können mit `quietly` unterdrückt werden. Mit `noisily` können wiederum einzelne Anweisungen innerhalb einer `quietly` Anweisung angezeigt werden. Dazu ebenfalls ein (hoffentlich illustratives) Beispiel:

```
quietly {

local jahr = 2006

forvalues monat = 1(1)12 {
local datum = mdy(‘monat’,10,‘jahr’)

if ‘monat’ == 8 {
         noisily display "Es ist August."
         }
else {
         display "Das wird nicht angezeigt."
         }
      }
}
```

6.11 MATA: Arbeiten mit Matrizen

Ab Version 9.0 steht die Matrix–Pogrammsprache MATA zur Verfügung. Siehe dazu das separate Manual `MATA`.

6.12 Maximum likelihood und nicht–lineare kleinste Quadrate

STATA hat eine sehr leistungsfähige und einfache Sprache zur Programmierung von eigenen ML– und NLS–Schätzungen (obwohl man dies in der Regel nicht brauchen wird). Siehe dazu `help ml` sowie `help nl`.

6.13 Zufallszahlen

Gleichverteilte (auf dem Einheitsintervall) und standardnormalverteilte Zufallszahlen lassen sich folgendermassen erzeugen:

```
. clear
. set obs 1000 /* 1’000 Beobachtungen generieren */
obs was 0, now 1000
. generate random = uniform() /* Erzeugt eine gleichverteilte Zufallsvariable */
. generate random2 = invnorm(uniform()) /* Erzeugt eine stand.normalverteilte ZV */
. summarize
```

Variable	Obs	Mean	Std. Dev.	Min	Max
random	1000	.5026516	.2919431	.0009839	.9995214
random2	1000	-.0219599	1.002332	-3.387047	3.318738

Analog zum Ziehen von Stichproben gilt hier, dass man den Startwert des Zufallszahlengenerators setzen muss, damit die Ergebnisse reproduzierbar sind.

6.14 Fehlersuche

Wenn ein do–file nicht komplett durchläuft und man den Fehler nicht findet, kann man mit `set trace on` zur Fehlersuche verwendet werden (mit `set trace off` stellt man diese Funktion wieder aus).

7 Deskriptive Kennwerte und einfache Testverfahren

7.1 Univariate Kennwerte

Die wichtigsten univariaten statistischen Angaben zu einzelnen Variablen erhält man über den Befehl `summarize <varname>`. Dadurch werden die folgenden Angaben ausgegeben: Anzahl (non-missing) Beobachtungen, Mittelwert, Standardabweichung, Minimum und Maximum. Über die Option `detail` werden weitere univariate Kennwerte wie etwa die Perzentile (inkl. dem Median) oder die Schiefe und die Kurtosis der Verteilung ausgegeben.

```
. global data /mnt/disc3/STATA/introduction2stata/data
. use $data/beispieldatensatz.dta
. summarize inc, detail
```

inc

Percentiles		Smallest		
1%	2100	2100		
5%	2300	2500		
10%	2625	2750	Obs	20
25%	3300	2800	Sum of Wgt.	20
50%	4320		Mean	4190
		Largest	Std. Dev.	1123.711
75%	4895	5000		
90%	5435	5120	Variance	1262726
95%	6125	5750	Skewness	-.0953087
99%	6500	6500	Kurtosis	2.534299

```
. sort sex
. by sex: summarize inc
```

-> sex = 1

Variable	Obs	Mean	Std. Dev.	Min	Max
inc	10	4097	1274.005	2500	6500

-> sex = 2

Variable	Obs	Mean	Std. Dev.	Min	Max
inc	10	4283	1011.655	2100	5750

Weitere und ähnliche Informationen zu einzelnen Variablen erhält man über den Befehl `codebook <varname>`. Zusätzliche Angaben erhält man mit dem Befehl `inspect <varname>`.

Mit `centile` können Perzentile (mit entsprechendem Vertrauensintervall) berechnet werden, mit `ci` können Vertrauensintervalle berechnet werden.

Beispiel (3. Quartil):

```
. centile inc, centile(75)
```

				— Binom. Interp. —	
Variable	Obs	Percentile	Centile	[95% Conf.	Interval]
inc	20	75	4897.5	4394.071	5743.532

7.2 Einfache Testverfahren

Mit `ttest` können einfache Mittelwertvergleiche durchgeführt werden.

```
. ttest inc, by(sex) unequal
Two-sample t test with unequal variances
```

Group	Obs	Mean	Std. Err.	Std. Dev.	[95% Conf.	Interval]
1	10	4097	402.8759	1274.005	3185.631	5008.369
2	10	4283	319.9134	1011.655	3559.306	5006.694
combined	20	4190	251.2694	1123.711	3664.087	4715.913
diff		-186	514.4449		-1270.8	898.7996

```
Satterthwaite's degrees of freedom:  17.1211
                    Ho: mean(1) - mean(2) = diff = 0
     Ha: diff < 0                 Ha: diff != 0                 Ha: diff > 0
       t =  -0.3616                 t =  -0.3616                 t =  -0.3616
   P < t =   0.3611           P > |t| =   0.7221           P > t =   0.6389
```

Mit `sdtest` kann ein einfacher Test auf Varianzhomogenität durchgeführt werden.

```
. sdtest inc, by(sex)
Variance ratio test
```

Group	Obs	Mean	Std. Err.	Std. Dev.	[95% Conf.	Interval]
1	10	4097	402.8759	1274.005	3185.631	5008.369
2	10	4283	319.9134	1011.655	3559.306	5006.694
combined	20	4190	251.2694	1123.711	3664.087	4715.913

```
                         Ho: sd(1) = sd(2)
             F(9,9) observed   = F_obs           =    1.586
             F(9,9) lower tail = F_L   = 1/F_obs =    0.631
             F(9,9) upper tail = F_U   = F_obs   =    1.586
  Ha: sd(1) < sd(2)          Ha: sd(1) != sd(2)          Ha: sd(1) > sd(2)
P < F_obs = 0.7486     P < F_L + P > F_U = 0.5029     P > F_obs = 0.2514
```

7.3 Bivariate und partielle Korrelation

Bivariate Korrelationen zwischen Variablen erhält man über den Befehl `correlate <varname(s)>`. Um das Signifikanzniveau der Korrelationen zu erhalten, muss man allerdings den alternativen Befehl `pwcorr <varname(s)>` benutzen. Partielle Korrelationen erhält man über den Befehl `pcorr <varname(s)>`.

```
. global data /mnt/disc3/STATA/introduction2stata/data
. use $data/beispieldatensatz_age.dta
. correlate age inc
(obs=20)
```

	age	inc
age	1.0000	
inc	-0.0686	1.0000

```
. pwcorr age inc, sig
```

	age	inc
age	1.0000	
inc	-0.0686	1.0000
	0.7737	

7.4 Tabellen von Kennwerten erstellen

Mit dem Befehl `table <varname>, contents (clist)` lassen sich für kategoriale Variablen Tabellen mit statistischen Kennwerten erstellen[9], wobei `clist (<statistic> <varname>)` verschiedene univariate Kennwerte von verschiedenen Variablen sein können. Beispiele für `clist` sind etwa der Mittelwert `(mean)` oder die Standardabweichung `(sd)`.

```
. global data /mnt/disc3/STATA/introduction2stata/data
. use $data/beispieldatensatz_age.dta
. table sex, c(mean age sd age mean inc sd inc)
```

sex	mean(age)	sd(age)	mean(inc)	sd(inc)
1	47.742916	10.00612	4097	1274.005
2	42.59822	11.36635	4283	1011.655

7.5 Erstellen von Kreuztabellen

Mit dem Befehl `tabulate <varname>` lässt sich eine kategoriale Variable auflisten. Mit dem Befehl `tabulate <varname1><varname2 >` lassen sich Kreuztabellen erstellen.
Mit dem Zusatz `, generate(<new varname>)` lässt sich aus einer kategorialen Variable ein Set von Indikatorvariablen erstellen (z.B. für die Verwendung in Regressionsrechnungen).
Mit der Option `all` lassen sich verschiedene Assoziationsmasse berechnen (Siehe auch `spearman` und `ktau` zu weiteren Assoziationsmassen). Mit den Optionen `cell, column` und `row` lassen sich verschiedene Prozentangaben ausgeben.

```
. use beispieldatensatz_age.dta
. tabulate sex educ, row
```

Key
frequency *row percentage*

	educ			
sex	1	2	3	Total
1	4 40.00	2 20.00	4 40.00	10 100.00
2	1 10.00	8 80.00	1 10.00	10 100.00
Total	5 25.00	10 50.00	5 25.00	20 100.00

[9] Mit dem `table` Befehl lassen sich analoge Ergebnisse produzieren wie mit dem `collapse` Befehl, mit dem Unterschied dass bei letzterem der Datensatz komprimiert wird. Der `table` Befehl verändert den Datensatz nicht.

8 Ökonometrische Modelle

Allgemeine Informationen zur Schätzung von ökonometrischen Modellen finden sich unter `help estcom`.

8.1 Signifikanzniveau setzen

Das Signifikanzniveau (und damit die Breite von Vertrauensintervallen) kann bei Schätzungen als Option `level(#)` eingegeben werden (wobei `#` $= 1 - \alpha$ entspricht). Mit dem Befehl `set level #` kann das Signifikanzniveau unabhängig vom Schätzverfahren und bis zur Beendigung des Programmes geändert werden (vorgegeben sind 95%).

8.2 Speichern, Verwalten und Darstellen von Schätzergebnissen

Mit den `estimates` Befehlen können Ergebnisse von Modellen gespeichert, verwaltet und dargestellt werden. Zur Illustration dient folgendes Beispiel:

```
. estimates clear /* (allenfalls) vorhandene Schaetzergebnisse loeschen */
. quietly regress ln_inc age
. estimates store model_1 /* Speichert Ergebnisse unter "model_1" */
. quietly regress ln_inc age age_sq
. estimates store model_2
. quietly regress ln_inc age age_sq male educ_2 educ_3
. estimates store model_3
. estimates table _all /* Alle Ergebnisse in einer Tabelle anzeigen */
```

Variable	model_1	model_2	model_3
age	.00350155	.05123008	-.04465442
age_sq		-.00051415	.00061164
male			.08029341
educ_2			.36181506
educ_3			.67394181
_cons	8.1441142	7.0936829	8.6145366

```
. estimates stats _all /* Statistiken zu allen Schaetzungen anzeigen */
```

Model	nobs	ll(null)	ll(model)	df	AIC	BIC
model_1	20	-3.296431	-3.129672	2	10.25934	12.25081
model_2	20	-3.296431	-2.996057	3	11.99211	14.97931
model_3	20	-3.296431	5.696722	6	.6065562	6.58095

8.3 Testen von Hypothesen und Kombinationen von Parametern

Nach dem Schätzen eines Modells können verschiedene Arten von Hypothesen getestet werden:

- Lineare Hypothesen: Lineare Hypothesen (z.B. herkömmlicher t– und F–Test in OLS) können mit dem Befehl `test` durchgeführt werden.
- Nichtlineare Hypothesen: `testnl`
- Lineare Kombinationen von Parametern: `lincom`
- Nichtlineare Kombinationen von Parametern: `nlcom`
- Likelihood–Ratio Test: `lrtest`

- Hausman–Test: `hausman`

8.4 Geschätzte Werte, Residuen, etc. generieren

Nach dem Schätzen eines Modells können über `predict newvar` verschiedene Grössen berechnet werden (z.B. die geschätzten Werte der abhängigen Variable, Residuen, etc.). Welche Grössen im einzelnen berechnet werden können, hängt natürlich vom geschätzten Modell ab. Unter `help command` ist jeweils aufgeführt, welche Grössen berechnet werden können.

```
. quietly regress ln_inc age age_sq male educ_2 educ_3
. predict yhat, xb /* Berechnet die geschaetzten Werte der abhaengigen Variable */
. predict resid, residual /* Berechnet die Residuen */
```

8.5 Marginale Effekte und Elastizitäten

Mit `mfx` können marginale Effekte sowie Elastizitäten (sowie deren Standardfehler) berechnet werden.

8.6 Lineare Regression

Lineare Regressionsmodelle lassen sich über den Befehl:

`regress <depvar> <indepvar(s)>, [options]` berechnen.

Es existiert eine Reihe von regressionsanalytischen tools zum linearen Regressionsmodell. Siehe dazu `help regdiag`.

8.7 IV und 2SLS

IV ("instrumental variable estimation") und 2SLS ("two–stage least squares") kann mittels `ivreg` berechnet werden:

`ivreg <depvar> [<varlist1>] (<varlist2> = <varlist_iv>), [options]`

Wobei `<depvar>` die abhängige Variable darstellt. `<varlist1>` enthält die exogenen (in der Sprache von IV) Regressoren, `<varlist2>` die endogenen Regressoren. Die Instrumente werden in `<varlist_iv>` spezifiziert.

8.8 Qualitative abhängige Variablen

- Logit–Modell (logistische Regression):

  `logit / logistic <depvar> <indepvar(s)>, [options]`

- Probit–Modell:

  `probit <depvar> <indepvar(s)>, [options]`

Für diese Modelle muss die abhängige Variable so kodiert sein, dass der Wert Null einen "Misserfolg" und ein Wert ungleich Null (typischerweise, aber nicht notwendigerweise der Wert Eins) einen "Erfolg" kennzeichnet. `logit` und `logistic` ergeben dieselben Resultate, allerdings werden unter `logistic` standardmässig die odds ratios ausgegeben und unter `logit` die Koeffizienten. Allerdings lassen sich unter beiden Befehlen sowohl die Koeffizienten als auch die odds ratios ausgeben.

- Multinomiales logit–Modell:

  `mlogit <depvar> <indepvar(s)>, [options]`

- Ordered logit–Modell:

  `ologit <depvar> <indepvar(s)>, [options]`

- Ordered probit–Modell:

  `oprobit <depvar> <indepvar(s)>, [options]`

Welche Werte die abhängige Variable annimmt, spielt bei den beiden zuletzt genannten Modellen effektiv keine Rolle, ausgenommen dass höhere Werte mit einem "höheren" Ereignis assoziiert werden. Ausserdem ist die Anzahl der Ereignisse auf 50 beschränkt.

8.9 Zähldaten

- Poisson–Modell:

  `poisson <depvar> <indepvar(s)>, [options]`

- Negatives Binomialmodell:

  `gnbreg <depvar> <indepvar(s)>, [options]`

- Zero–inflated Poisson Modell

  `zip <depvar> <indepvar(s)>, [options]`

8.10 Modelle für Variablen mit eingeschränktem Wertebereich

- Zensierung (censoring):

  `cnreg <depvar> <indepvar(s)>, ll[(#)]ul[(#)] [other options]`

- Corner solution; Tobit-Modell:

  `tobit <depvar> <indepvar(s)>, ll[(#)]ul[(#)] [other options]`

Mit Hilfe von `ll[(#)]` und / oder `ul[(#)]` können die Zensierungspunkte angegeben werden. `ll( )` kennzeichnet Linkszensierung, `ul( )` Rechtsensierung.

- Stutzung (truncation):

  `truncreg <depvar> <indepvar(s)>, ll(varname | #) ul(varname | #) [other options]`

`ll( )` und `ul( )` kennzeichnen Stutzungspunkte.

- Selbstselektion (Heckman selection model):

  `heckman <depvar> <varlist>, select([depvar_s = ] varlist_s) twostep [options]`

Die Option `select( )` kennzeichnet die Selektionsgleichung und ist nicht optional.

8.11 Ereignisdaten

Zur Analyse von Verweildauern muss der Datensatz zunächst als solcher deklariert werden. Dies geschieht mit Hilfe des Befehls `stset` (allgemeine Informationen zu "survival-time data" findet man unter `help st`).

Deskriptive Auswertungen sind anschliessend über die Befehle `stdes` und `stsum` verfügbar. "Überlebenstafeln" lassen sich über den Befehl `ltable` produzieren und mit der Option `, graph` auch grafisch darstellen (siehe auch `help sts`).

- Semiparametrisches Regressionsmodell (Cox–Regression):

  `stcox <varlist>, [options]`

- Parametrische Regressionsmodelle:

  `streg <varlist>, dist(distname) [options]`

Wobei `distname` eines der folgenden parametrischen Modelle bezeichnet:
`distname = (exponential | weibull | gompertz | lognormal | loglogistic)`.

Zur Analyse von Ereignisdaten gibt es ein separates Handbuch von STATA (`Survival Analysis and Epidemiological Tables`).

8.12 Paneldaten

Zur Analyse von Paneldaten muss der Datensatz zu Beginn ebenfalls als solcher deklariert werden. Dies geschieht mit dem Befehl `xt, i(varname) t(varname)`, wobei unter `i(...)` die Panelvariable und `t(...)` die Zeitvariable bezeichnet.

Anschliessend lassen sich über `xtdes`, `xtsum` und `xttab` deskriptive Auswertungen vornehmen.

- fixed–effects Modell:

  `xtreg <depvar><indepvars>, fe`

- random-effects Modell:

  `xtreg <depvar> <indepvars>, re`

Zur Analyse von Paneldaten gibt es ein eigenes Handuch von STATA (`Longitudinal / Panel Data`).

8.13 Zeitreihendaten

Zur Analyse von Zeitreihendaten muss der Datensatz – analog zu Panel– und Ereignisdaten – zunächst als solcher deklariert werden. Dies kann man mit dem Befehl `tsset` tun. Mit `arima` und `arch` können (offensichtlich) ARIMA sowie (G)ARCH Modelle geschätzt werden.
Ist ein Datensatz als Zeitreihendatensatz deklariert, können die folgenden Zeitreihenoperatoren verwendet werden:

- `Lk.<varname>`: k–ter lag von `<varname>`
- `Fk.<varname>`: k–ter lead von `<varname>`
- `Dk.<varname>`: k–te Differenz von `<varname>`

Einen mehr oder weniger vollständigen Überblick über Befehle zum Handling von Zeitreihendaten findet man mittels `help time`.
Zur Analyse von Zeitreihendaten gibt es ebenfalls ein eigenes Handbuch von STATA (`Times-Series`).

9 Weitere Verfahren

Es gibt eine ganze Reihe von weiteren statistischen Verfahren, welche in STATA implementiert sind, unter anderem die folgenden.

9.1 Varianzanalyse

Einfache (einfaktorielle) Varianzanalysen können via `oneway` geschätzt werden, allgemeine Varianzanalysen mittels `anova`.

9.2 Hauptkomponenten– und Faktorenanalyse

- Hauptkomponentenanalyse:
  `pca <varlist>, [options]`
  `score <newvarlist>`
- Faktorenanalyse:
  `factor <varlist>, [options]`
  `score <newvarlist>`

In beiden Fällen erfolgt über den ersten Befehl (`pca` bzw. `factor`) die Berechnung der Hauptkomponenten bzw. Faktoren. Mit dem zweiten Befehl (`score`) werden die Ausprägungen der Beobachtungen auf diesen berechnet.

9.3 Clusteranalyse

Es gibt eine ganze Reihe von Anweisungen zur Clusteranalyse, siehe `help cluster`. Wir wollen an dieser Stelle aber nicht näher darauf eingehen.

9.4 Bootstrapping

Mithilfe der `bootstrap` und `bstat` Anweisungen lassen sich praktische beliebige Kennwerte bootstrappen.

10 Tabellen und Grafiken

Beim Erstellen von Tabellen und Graphiken ist darauf zu achten, dass diese immer selbsterklärend sein müssen, sofern Sie innerhalb einer schriftlichen Arbeit dargestellt werden.[10] Grundsätzlich müssen Tabellen und Grafiken ohne Bezug auf den Text der Arbeit verständlich sein. Bezüglich Layout von Tabellen und Grafiken orientiert man sich am besten anhand von publizierten Arbeiten.

10.1 Das Erstellen von Tabellen

Um eine Tabelle mit Regressionsoutputs zu erstellen, bietet STATA das Tool `outreg`[11]. Dazu gibt es, wie zu jedem Befehl, ein STATA Help-File, auf welches direkt aus STATA über `help outreg` zugegriffen werden kann.

Tabelle 1: Lohnregressionen

Abhängige Variable:	Einkommen		
Durchschnitt:	4'190		
	(1)	(2)	(3)
Mann	186.000 (0.36)	444.761 (0.78)	374.827 (0.77)
Alter		417.447 (1.03)	-44.882 (0.14)
Alter^2		-4.306 (0.99)	0.9648 (0.28)
Sekundäerstufe			1,259.800 (2.22)*
Tertiärstufe			2,508.144 (4.08)**
Konstante	4,097.0000 (11.26)**	-5,628.983 (0.61)	2,698.525 (0.38)
Beobachtungen	20	20	20
R^2	0.01	0.08	0.58

Anmerkungen: **, * bezeichnet Signifikanz auf dem 1% bzw. 5% Niveau. t–Wert in Klammern.

Ein Problem welches bei der Verwendung von `outreg` auftreten kann ist, dass wenn man das

[10] Dies bedeutet insbesondere, dass alle Beschriftungen in derselben Sprache sind und dass entweder keine Abkürzungen oder in einer Fussnote die entsprechenden Erklärungen angegeben sind.

[11] Dieses muss allerdings zunächst installiert werden. Dies kann man mit `net search outreg` tun und dann den Anweisungen folgen.

`outreg` ins EXCEL Format einliest, EXCEL Ausdrücken in Klammern als negative Ausdrücke ohne Klammer interpretiert. Dies kann umgangen werden, indem man den `outreg output` ins das Word Format bringt. Die Tabelle muss dann in ein EXCEL File kopiert werden, wobei die Zellen im EXCEL vorher als *Text* definiert werden müssen. Mit minimalen Änderungen können mittels `outreg` schöne Tabelle erstellt werden.

Tabelle 10.1 in diesem Abschnitt ist ein Beispiel, wie man drei Lohnregressionen übersichtlich in einer einzelnen Tabelle darstellen kann. Der Output des Skriptes von STATA, mit welchem diese Tabelle erstellt worden ist, befindet sich anschliessend an die Tabelle.

Mit diesem Skript kann Tabelle 10.1 erzeugt werden:

```
. tabulate edu, generate(edu_)
```

educ	Freq.	Percent	Cum.
1	5	25.00	25.00
2	10	50.00	75.00
3	5	25.00	100.00
Total	20	100.00	

```
. reg inc male
```

Source	SS	df	MS		
Model	172980	1	172980	Number of obs	= 20
Residual	23818820	18	1323267.78	F(1, 18)	= 0.13
				Prob > F	= 0.7219
Total	23991800	19	1262726.32	R-squared	= 0.0072
				Adj R-squared	= -0.0479
				Root MSE	= 1150.3

inc	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
male	186	514.4449	0.36	0.722	-894.8086	1266.809
_cons	4097	363.7675	11.26	0.000	3332.753	4861.247

```
. outreg using $outreg/regression.doc, replace bdec(3)
. capt g age_2 = age^2
. reg inc male age age_2
```

Source	SS	df	MS		
Model	2028351.62	3	676117.205	Number of obs	= 20
Residual	21963448.4	16	1372715.52	F(3, 16)	= 0.49
				Prob > F	= 0.6925
Total	23991800	19	1262726.32	R-squared	= 0.0845
				Adj R-squared	= -0.0871
				Root MSE	= 1171.6

inc	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
male	444.7614	569.5808	0.78	0.446	-762.6961	1652.219
age	417.4479	406.9646	1.03	0.320	-445.2784	1280.174
age_2	-4.306484	4.354288	-0.99	0.337	-13.53716	4.924194
_cons	-5628.984	9163.811	-0.61	0.548	-25055.4	13797.43

```
. outreg using $outreg/regression.doc, append bdec(3)
```

```
. #delimit ;
delimiter now ;
. reg inc male age age_2 edu_2 edu_3 ;
```

Source	SS	df	MS		
Model	13966030	5	2793206.01	Number of obs	= 20
Residual	10025770	14	716126.425	F(5, 14)	= 3.90
				Prob > F	= 0.0201
Total	23991800	19	1262726.32	R-squared	= 0.5821
				Adj R-squared	= 0.4329
				Root MSE	= 846.24

inc	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
male	374.8273	486.4794	0.77	0.454	-668.5673	1418.222
age	-44.88218	318.2186	-0.14	0.890	-727.3932	637.6289
age_2	.9647887	3.431033	0.28	0.783	-6.394045	8.323623
edu_2	1259.8	566.9532	2.22	0.043	43.80629	2475.794
edu_3	2508.145	614.3953	4.08	0.001	1190.398	3825.892
_cons	2698.525	7020.869	0.38	0.706	-12359.74	17756.79

```
. #delimit cr
delimiter now cr
. outreg using $outreg/regression.doc, append bdec(3)
```

10.2 Das Erstellen von Grafiken

STATA unterstützt eine Vielzahl von Grafiktypen. Für das Arbeiten mit dem Graphik–Tool von STATA empfehlen wir als Nachschlagewerk das `Graphics Reference Manual`. Mitchell (2004) enthlt viele Beispiele fr die verschiedensten Grafiken.

Eine Übersicht findet sich über `help graph`. Ein paar wenige (bzw. die am häufigsten verwendeten) werden hier noch separat aufgelistet. In den Abbildungen 2 bis 5 finden sich vier beispielhafte Grafiken (diese wurden allerdings alle mit anderen Datensätzen erstellt).

- Histogramme (Darstellung von quantitativen Merkmalen): Siehe `help histogram` zu weiteren Informationen.
- Scatterplots / Steudiagramme (Darstellung der gemeinsamen Verteilung von zwei quantitativen Merkmalen): Siehe `help scatter` dazu.
- Kurvendiagramm (dient primär der Darstellung von Variablen über die Zeit): Siehe `help line` zu weiteren Informationen.
- Zwei Scatterplots (es ist teilweise auch möglich, verschiedene Graphen zu kombinieren) in derselben Graphik: Siehe `help graph twoway`
- Säulendiagramm (zur Darstellung von qualitativen oder diskreten Merkmalen): Siehe `help graph bar`
- Schätzung der Dichtefunktion eines Merkmales (Kerndichteschätzung): Siehe `help kdensity` dazu.

Um Graphiken in einem anderen Format zu speichern (z.B. im eps Format) siehe `help graph export`.[12]

[12] Das eps Format eignet sich insbesondere, um Grafiken in LaTeXzu integrieren. Siehe Abschnitt 12. Siehe auch `help eps_options`.

Es folgt noch ein Beispiel für das Erstellen einer Graphik (siehe Abbildung 1 auf der folgenden Seite).

```
. reg inc age
```

Source	SS	df	MS	Number of obs =	20
Model	350440.05	1	350440.05	F(1, 18) =	0.27
Residual	23641359.9	18	1313408.89	Prob > F =	0.6118
				R-squared =	0.0146
Total	23991800	19	1262726.32	Adj R-squared =	-0.0401
				Root MSE =	1146

inc	Coef.	Std. Err.	t	P>|t|	[95% Conf.	Interval]
age	12.63201	24.45488	0.52	0.612	-38.74578	64.00979
_cons	3619.405	1133.976	3.19	0.005	1237.01	6001.8

```
. predict y_hat
(option xb assumed; fitted values)
. label variable y_hat "gesch. Einkommen"
. label variable inc "Einkommmen"
. #delimit ;
delimiter now ;
. scatter inc y_hat age,  c(i l) clp(l "-") ms(o i) sort
                                          xlabel(30(5)60,grid)  ylabel(2000(1000)7000, alternate)
                                          subtitle(Untertitel) title(Haupttitel) xtitle("Alter")
                                          note(Beschriftung);
. #delimit cr
delimiter now cr
. graph export $graph/graph.eps
(file graph.eps written in EPS format)
```

Dieselbe Graphik kann auch einfacher erstellt werden mithilfe des `twoway` Befehls (damit können wie erwähnt verschiedene Graphen übereinander gelegt werden):

```
. #delimit ;
. tw (scatter inc age) (lfit inc age, clp("-")), xlabel(30(5)60, grid)
ylabel(2000(1000)7000, alternate) xtitle("Alter")
legend(label(2 "gesch. Einkommen"));
. #delimit cr
```

Abbildung 1: Beispielgraphik

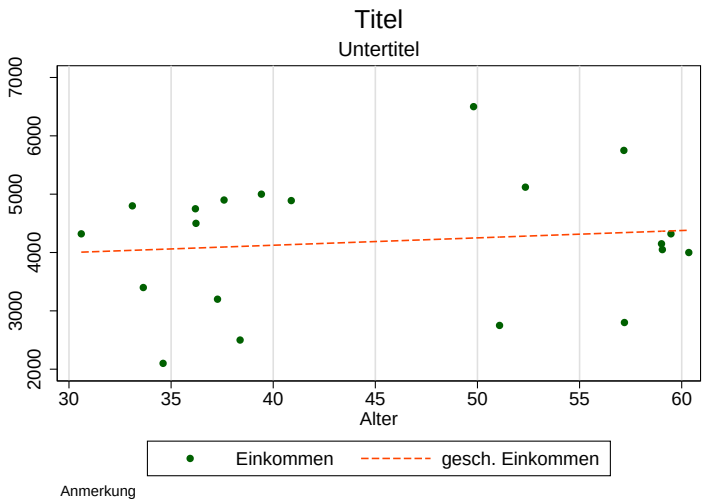
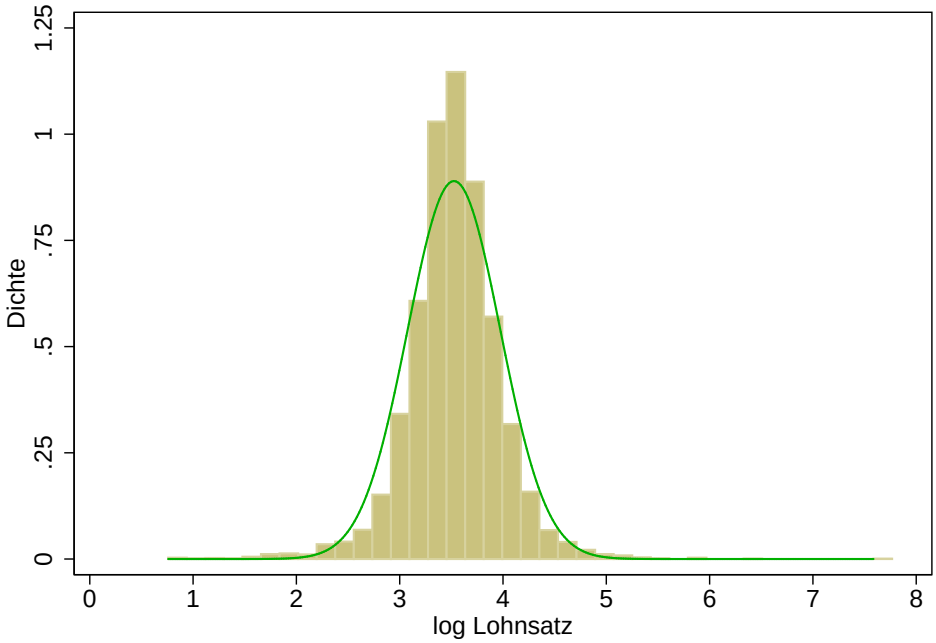
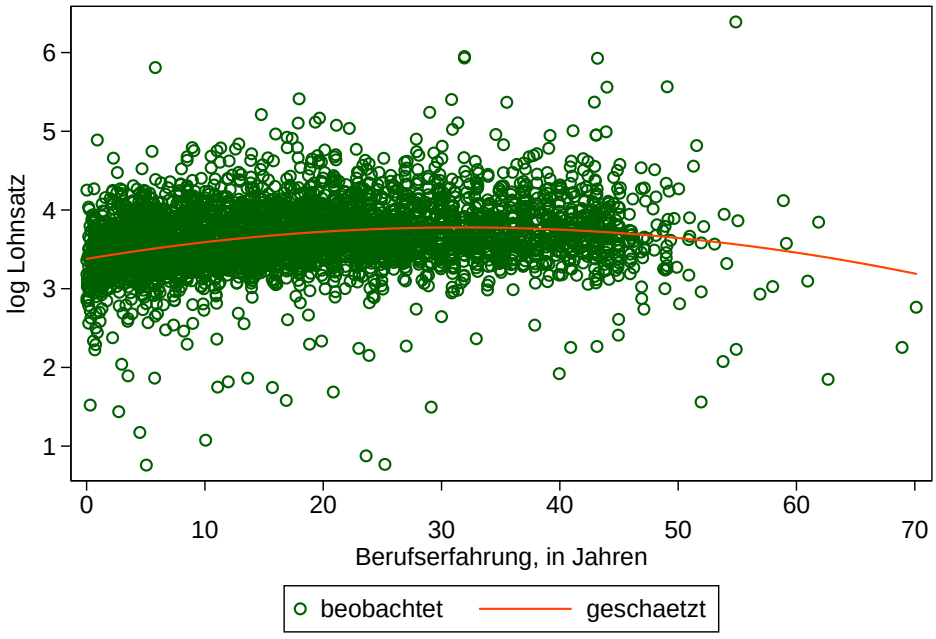


Abbildung 2: Beispiel für ein Histogramm



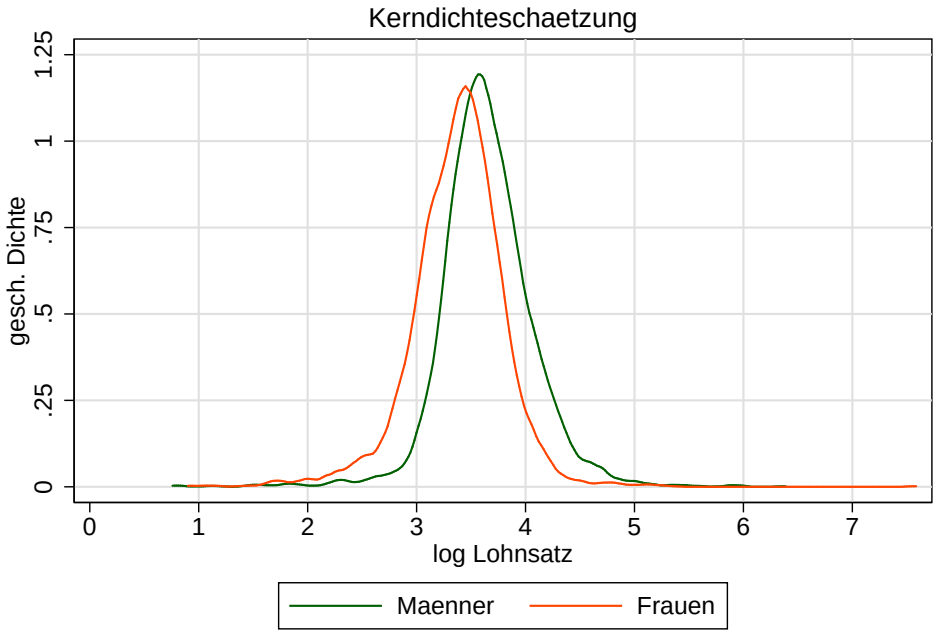
Quelle: SAKE, 1999

Abbildung 3: Beispiel für ein Streudiagramm



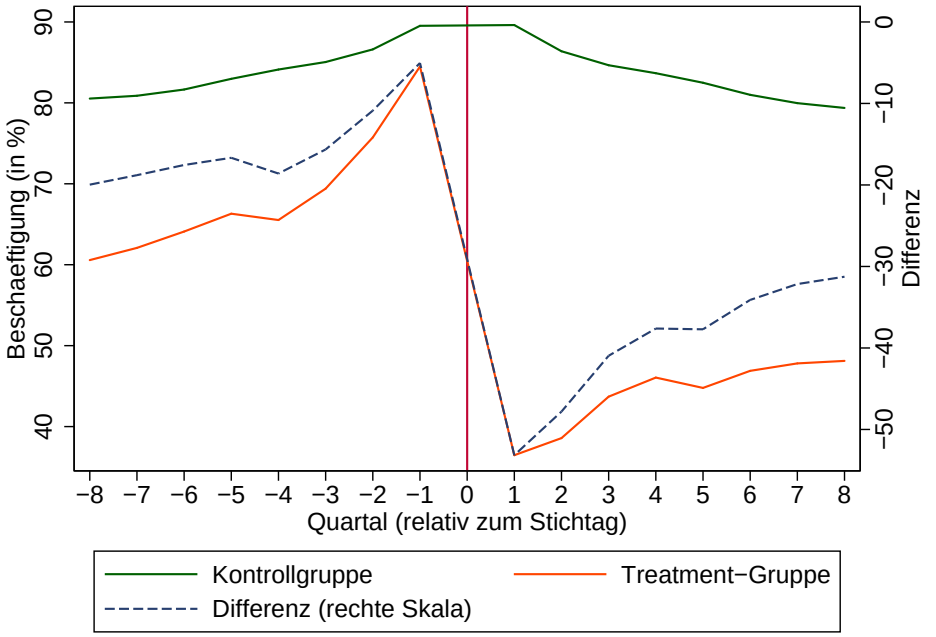
Quelle: SAKE, 1999

Abbildung 4: Beispiel für eine Kerndichteschätzung



Quelle: SAKE, 1999

Abbildung 5: Beispiel für ein Kurvendiagramm



Quelle: Hauptverband Oesterreich

11 Anwendungen

11.1 Arbeiten mit dem Übungs–dofile

Mit Hilfe des Skript `introduction2stata.do`kannst du alles was du bis jetzt gelernt hast, noch einmal ganz in Ruhe ausprobieren. Dazu musst du folgendermassen vorgehen:

1. Logge dich mittels deinem Studentenlogin über vnc auf dem Linux Server ein. Öffne das Skript `introduction2stata.do` direkt aus dem STATA. Das Skript liegt auf `/mnt/disc3/STATA/introduction2stata/dofiles`.

2. Fülle ganz oben im Skript folgendes ein: (i) bei NAME dein Name (ii) bei "PFAD DATA" den Pfad eines Ordners in deinem Ordner auf dem Linuux-Server für die Daten und (iii) in "PFAD LOGFILE" einen Pfad für das logfile. Dann speichere das Skript über `save as`in deinem Ordner auf dem Linux Server.

3. Nun kannst du das dofile Schritt für Schritt ausführen.

11.2 `time`–Skript

Mit diesem kleinen Programm kann unter Eingabe von `time <datum>` angezeigt werden, um welches Datum (Jahr, Monat) es sich handelt. Die Variable `datum` muss allerdings im STATA–internen Datumsformat vorliegen.

```
cap program drop time
program define time
args datum
nois dis as result year(‘datum’) " " month(‘datum’)
end
```

Am einfachsten ist es, den Programmcode in das `profile.do` zu kopieren, so dass bei jedem Start von STATA direkt dieses Programm geladen wird.

11.3 Erstellen von Quartalsdatensätzen

Das Skript `quartalsfile.do` erstellt aus den Qualifikations-Datensätzen Quartalsfiles. Du solltest jeden der Schritte in diesem Skript verstehen. Das Skript befindet sich auf `/mnt/disc3/STATA/dofiles`. Um das Skript ausführen zu können, musst du am Anfang noch einen Pfad für einen Ordner in deinem Ordner auf dem Linux Server eingeben und dann das Skript über `save as` in deinen Ordner speichern.

12 STATA und LaTeX

12.1 Grafiken

Das Einbinden von Grafiken aus STATA in ein LaTeX–Dokument ist kein Problem. Man muss die Grafiken lediglich in das EPS (Encapsulated PostScript) Format exportieren (mithilfe des `graph export` Befehls). In LaTeXkönnen EPS–Grafiken problemlos eingebunden werden (bsp. mit dem `epsfig` Befehl; dazu muss das gleichnamige Paket geladen werden).

12.2 Tabellen

Das Einbinden von Tabellen ist wesentlich umständlicher und lässt sich nach unserem Wissensstand noch nicht ganz vollständig automatisieren.

12.2.1 Verwendung von `outreg`

Eine erste (indirekte) Möglichkeit besteht darin, über `outreg` Tabellen im Word– oder Excel–Format zu erstellen und von dort aus weiterzubearbeiten und entsprechend in das TeX–File einzufügen.

12.2.2 Export in LaTeX

Es gibt auch verschiedene Programmroutinen, mit welchen verschiedene Arten von Tabellen mehr oder weniger direkt in ein LaTeX–File exportiert werden können (alle diese Routinen sind als `ado-files` vorhanden und müssen zunächst installiert werden).

- `sutex`: Dient dem Exportieren von deskriptiven Kennwerten in eine LaTeX–Tabelle.
- `outtable:` Export einer STATA–Matrix in eine LaTeX–Tabelle.
- `outtex`: Export eines files in ein LaTeX–File.
- `estout, est2tex`: Export von Schätzergebnissen (siehe dazu auch `estimates` in Abschnitt 8.2) in eine LaTeX–Tabelle. Auf diesem Prinzip basiert auch der Weg über `outreg`, allerdings kann dort kein LaTeX–File generiert werden.

Literatur

[1] Acock, Alan C. (2006). A Gentle Introduction to Stata. Stata Press.

[2] Baum, Christopher (2005). Introduction to Stata at Boston College. `www.u-cergy.fr/rech/pages/nguyenvan/ea/StataIntro.pdf`

[3] Cleves, Mario; William W. Gould, and Roberto Gutierrez (2004). An Introduction to Survival Analysis Using Stata, Revised Edition. Stata Press.

[4] Gould, William, Jeffrey Pitblado, and William Sribney (2006). Maximum Likelihood Estimation with Stata, 3rd Edition. Stata Press.

[5] Hamilton, Lawrence C. (2006). Statistics with Stata (Updated for Version 9.0). Brooks/Cole.

[6] Kohler, Ulrich and Frauke Kreuter (2005). Data Analysis Using Stata. Stata Press.

[7] Kopka, Helmut (2000). LaTeX Band 1: Einführung. 3. Auflage. Pearson Studium.

[8] Long, J. Scott and Jeremy Freese (2006). Regression Models for Categorial Dependent Variables Using Stata, 2nd Edition. Stata Press.

[9] Mitchell, Michael (2004). A Visual Guide to Stata Graphics. Stata Press.

[10] Wooldridge, Jeffrey M. (2005). Introductory Econometrics. South College Publishing.

Baum (2005) ist eine etwas ausführlichere Einführung. Einführende Bücher zu STATA sind Acock (2006), Hamilton (2006) und Kohler und Kreuter (2006). Daneben gibt es Bücher zu spezialisierten Themen (Cleves et al. 2004, Gould et al. 2006, Long und Freese 2006, Mitchell 2004).

Die mit Abstand beste Einführung in die unter Abschnitt 8 erwähnten Modelle ist Wooldridge (2006).

Mehr zu LaTeX findet man beispielsweise in Kopka (2000).

A Beispieldatensatz

	id	sex	educ	inc	birth
1.	1	1	2	2500	28oct1967
2.	2	2	1	2100	5aug1971
3.	3	2	2	3200	5dec1968
4.	4	1	2	3400	24jul1972
5.	5	2	3	4500	25dec1969
6.	6	1	1	2800	2jan1949
7.	7	1	1	2750	13feb1955
8.	8	1	3	4750	4jan1970
9.	9	2	2	4320	7aug1975
10.	10	2	2	4050	24feb1947
11.	11	1	3	5120	8nov1953
12.	12	2	2	4800	5feb1973
13.	13	1	1	4000	8nov1945
14.	14	2	2	4900	10aug1968
15.	15	1	3	5000	12oct1966
16.	16	1	3	6500	23may1956
17.	17	2	2	5750	12jan1949
18.	18	2	2	4890	26apr1965
19.	19	2	2	4320	23sep1946
20.	20	1	1	4150	11mar1947

B Beispielvorlage für ein do–file

```
***
*** "Inhalt des dofiles"
***
*** "Verfasser, Datum"
***

clear  /* loescht den Arbeitsspeicher */
cap log close  /* schliesst ein allenfalls geoeffnetes logfile */
set mem #m /* memory auf # MB setzen */
set more off /* more ausschalten */

*** globale Makros fuer die verwendeten Dateipfade eingeben

global data "PFAD ROHDATEN"
global path "PFAD EIGENE DATEN"
global log  "PFAD LOGFILES"
global do   "PFAD DOFILES"

*** logfile oeffnen und falls bereits vorhanden ueberschreiben

log using $log/00_template.log, replace

***
*** Hier beginnt das eigentliche do-file
***


*** Am Schluss logfile schliessen und more wieder einschalten

log close
set more on
```

C 50 wichtige Befehle...

welche man kennen sollte[13].

- Hilfefunktion, Suchfunktion:
  - `help, search`
- Betriebssystemebene:
  - `pwd, cd, sysdir, mkdir, dir / ls, erase, copy, type`
- Öffnen und Speichern von Datensätzen:
  - `use, clear, save, append, merge, compress`
- Dateneingabe:
  - `input, edit, infile, infix, insheet`
- Internet und Updaten von STATA:
  - `update, net, ado, news`
- Einfache Datenbeschreibung:
  - `describe, codebook, inspect, fastlist, browse, count, assert, summarize, table, tabulate`
- Datenmanipulation:
  - `generate, replace, egen, recode, rename, drop, keep, sort, encode, decode, order, by, reshape`
- Formatierung:
  - `format, label`
- Arbeitsaufzeichnung:
  - `log, notes`
- Taschenrechner:
  - `display`

[13] Übernommen von: `www.ats.ucla.edu/stat/stata/notes3/commands.htm`