

# Stata-Einführung für den SFB 580

Ulrich Kohler  
 kohler@wz-berlin.de  
 23.-24. September 2005

## Inhaltsverzeichnis

<b>1</b>	<b>Allgemeine Grundlagen</b>	<b>1</b>
1.1	Die Eingabe von Befehlen . . . . .	1
1.2	Hilfefunktionen . . . . .	2
1.3	Beispielanalysen . . . . .	3
1.4	Stata-Grammatik . . . . .	6
1.5	Do-Files . . . . .	20
1.6	Wie wird eine Befehlsdatei gestartet? . . . . .	21
<b>2</b>	<b>Variablen erzeugen und verändern</b>	<b>22</b>
2.1	generate und replace . . . . .	22
2.2	Label . . . . .	27
2.3	Spezielle Recodierungs-Befehle . . . . .	30
2.4	Missing values . . . . .	34
2.5	Rekodieren in hierarchischen Datensätzen . . . . .	35
<b>3</b>	<b>Erstellen von Grafiken</b>	<b>41</b>
3.1	Struktur des Grafik-Befehls . . . . .	41
3.2	Grafikelemente innerhalb der Datenregion . . . . .	44
3.3	Informationen außerhalb der Plotregion . . . . .	49
3.4	Multiple Grafiken . . . . .	51
3.5	Ausdruck und Export . . . . .	53
<b>4</b>	<b>Regressionsmodelle</b>	<b>54</b>
4.1	Statistische Modelle mit Stata . . . . .	54
4.2	Logistische Regression . . . . .	57
4.3	Regressionsdiagnostik . . . . .	64
4.4	Publikationsfertigen Output erzeugen . . . . .	70
<b>5</b>	<b>Modelle für hierarchische Daten</b>	<b>71</b>
5.1	Einfache Fixed-Effects Modelle . . . . .	73
5.2	Einfache Random-Effects Modelle . . . . .	75
5.3	Generalized Estimation Equation (GEE) . . . . .	76
5.4	Multilevel mixed-effects linear regression . . . . .	77
5.5	GLLAMM . . . . .	79
<b>6</b>	<b>Literatur</b>	<b>81</b>

Slide 1

## 1 Allgemeine Grundlagen

### 1.1 Die Eingabe von Befehlen

Stata wird durch eine Kommandosprache gesteuert. In der Kommandozeile werden Befehle als Worte, Buchstaben oder Zahlen eingegeben. Hier ist ein Beispiel:

```
. describe
```

Stata-User kommunizieren, indem sie sich Befehlszeilen zuschicken.  
Z.B.:

Q: „Wie bekomme ich standardisierte Regressionskoeffizienten?“

A: `. reg, beta`

Slide 2

### 1.2 Hilfefunktionen

Stata stellt drei Hilfefunktionen zur Verfügung:

- `help` wird verwendet, wenn man einen Befehlsnamen kennt, aber nicht weiß wie er funktioniert

```
. help describe, nonew
```

- `search` wird verwendet, wenn man einem Befehl für eine spezifische Anwendung sucht.

```
. search multilevel
```

- Das Menü wird verwendet, wenn man absolut nicht weiß, was man jetzt tun soll.

Slide 3

### 1.3 Beispielanalysen

#### 1.3.1 Daten laden

Mit dem Befehl `use` werden Stata-Systemdatensätze geladen. Nach dem Kommando wird der Dateiname angegeben. Wenn keine Erweiterung angegeben wird, wird „`dta`“ vorausgesetzt.

```
. use mydata
. use c:/programme/stata/auto
. use "c:/user/daten/stata/Automobile 1978"
. use http://www.stata-press/data/kk/data1.dta
```

Für später:

```
. save data1.dta, replace
```

Slide 4

#### 1.3.2 Daten kennenlernen

Einen allerersten Überblick über einen Datensatz verschaffen die Befehle `describe`, `ds` und `lookfor`.

```
. describe
. ds
. lookfor party
```

Slide 5

### 1.3.3 Einige oft verwendete statistische Verfahren

*Just to get the look & feel:*

```
. sum
. tab np9401
. tab np9402 np9403
. tabstat np11701 np0105, statistics(mean) by(state)
. reg np9403 hhinc ybirth
. ologit np9403 hhinc ybirth
. mlogit np9402 hhinc ybirth
. xtreg np9403 hhinc ybirt, i(state)
```

Slide 6

## 1.4 Stata-Grammatik

Fast alle Stata-Kommandos bestehen aus folgenden, in spezifischer Reihenfolge angeordneten Bausteinen:

```
command [varlist] [if] [in] [weight] [, options ]
```

Durch die Online-Hilfe bekommt man einen Hilfetext zu den Bausteinen:

```
. help varlist, nonew
. help if, nonew
. help in, nonew
. help weights, nonew
. help options, nonew
```

Slide 7

### 1.4.1 Welcher Baustein ist erlaubt?

Man kann einen Baustein nur verwenden, wenn er für einen Befehl *zugelassen* oder *vorgeschrieben* ist. Die Information darüber, erhalten Sie über die Syntax-Angabe der Online-Hilfe. Ein Baustein gilt als *zugelassen*, wenn er bei der Syntax-Angabe genannt wird. Erfolgt die Nennung *nicht* in eckigen Klammern, so ist der Baustein zwingend vorgeschrieben. Bausteine, die nicht angegeben werden, sind nicht zugelassen.

Die Syntaxangabe des Befehls `summarize` lautet:

```
[ by varlist: ] summarize [ varlist ] [ weight ] [ if ] [ in ] [, detail |
    meanonly format ]
```

Alle oben genannten Bausteine sind *zugelassen*, nur der Befehl ist zwingend vorgeschrieben.

Slide 8

### 1.4.2 Der Befehl

Mit dem Befehl wird festgelegt, welche Prozedur angefordert wird. Manche Befehle können abgekürzt werden. In der Online-Hilfe wird die maximal mögliche Abkürzung farblich hervorgehoben. Zwischen der maximal möglichen Abkürzung und dem Ausschreiben des gesamten Befehls sind alle Variationen möglich:

```
. su
. sum
. summ
. summa
. summar
. summari
. summariz
```

Slide 9

### 1.4.3 Die Variablenliste

Die Variablenliste (*varlist*) ist eine von Leerzeichen unterbrochene Liste von Variablennamen.

```
. sum ybirth hhinc np9403
. d ybirth hhinc np9403
```

Anzahl und Reihenfolge der Variablennamen einer *varlist* sind beliebig.

Slide 10

### 1.4.4 Abkürzungen der Variablenliste

Alle Variablennamen können abgekürzt werden. Es genügt, nur so viel des Namens einzugeben, dass die Variable eindeutig identifizierbar wird.

```
. sum w
```

Aber:

```
. sum y
```

Aber:

```
. sum y-h
```

Slide 11

### 1.4.5 Angabe von Variablenbereichen

In Variablenlisten gibt es drei Möglichkeiten Variablenbereiche zu spezifizieren:

- Variablen, die im Datensatz hintereinander stehen, können durch einen Bindestrich gemeinsam angesprochen werden.

```
. sum kitchen-phone
```

- Variablen mit gleichen Anfangsbuchstaben können durch die *Wildcard* „\*“ gemeinsam angesprochen werden.

```
. d np*
. d *inc*
```

Slide 12

- Variablen, die sich nur in einem Zeichen unterscheiden können mit ? gemeinsam angesprochen werden

```
. d np9?01
. d np9?0?
```

Slide 13

#### 1.4.6 Die „if“-Bedingung

Mit der „if“-Bedingung wird die Ausführung eines Befehls auf diejenigen Fälle eingegrenzt, für die der in der „if“-Bedingung gegebene *Ausdruck* wahr ist.

Ein einfaches Beispiel für eine „if“-Bedingung ist:

```
. sum *inc* if state==15
```

Hier wird der Befehl `sum shhinc` für diejenigen Fälle ausgeführt, für die der Ausdruck „state==15“ *wahr* ist. Das sind alle Thüringer.

Slide 14

#### 1.4.7 Variationsmöglichkeiten von if-Bedingungen

Durch die Verwendung von *Operatoren* und *Funktionen* ergeben sich zahlreiche Variationsmöglichkeiten von if-Bedingungen

```
. sum *inc* if state >= 11
. sum *inc* if state >= 11 & gender == 1
. sum *inc* if sqfeet/rooms <= 300
. sum *inc* if uniform() > .5
```

Ein Liste der Operatoren und Funktionen erhält man durch `help functions` bzw. `help operators`

Slide 15

**Eine Warnung** *Missings* werden in Stata auf  $+\infty$  gesetzt. Sie werden darum durch Ausdrücke mit den relationalen Operatoren für  $>$  und  $\geq$  eingeschlossen.

Slide 16

#### 1.4.8 Die Gewichtsanzweisung

Die Gewichtsanzweisung führt dazu, dass jeder Beobachtung des Datensatzes ein bestimmtes Gewicht zugewiesen wird. Die Gewichtsanzweisung hat folgende Syntax:

```
[weight_type = varname]
```

Dabei existieren drei Gewichtungstypen:

- `fweight` frequency weights
- `aweight` analytic weights
- `pweight` sampling weights

```
. sum varlist [aweight = varname]
```

Slide 17

### 1.4.9 Optionen

Nahezu jeder Stata-Befehl kann durch sogenannte „Optionen“ näher spezifiziert werden. Optionen werden durch ein Komma vom eigentlichen Kommando abgetrennt. Bei mehreren Optionen steht *nur ein* Komma. Das Komma leitet also nicht eine Option, sondern eine Liste von Optionen ein. Die Reihenfolge der Optionen eines Befehls ist beliebig.

```
. sum hhinc, detail
. tab np9402 np9403, row col
. d, short
```

Slide 18

### 1.4.10 Das Befehlsprefix „by“

Der Prefix „by“ besteht aus dem Prefix und einer Variablenliste. Es bewirkt, dass der eigentliche Stata-Befehl für alle Kategorien der Variablen aus der Liste wiederholt wird.

Die Anwendung des „By“-Prefixes setzt voraus, dass der Datensatz nach den Variablen der „By“-Liste sortiert wurde.

```
. by state, sort: sum np9403
. by state gender, sort: sum np9403
```

Slide 19

### 1.4.11 Schleifen

Stata erlaubt auf einfache Weise die Konstruktion sog. *Schleifen*. Diese werden eingesetzt, wenn eine bestimmte Prozedur, geringfügig abgewandelt häufiger wiederholt werden muss.

Im folgenden Beispiel wird eine Kreuztabelle aller „Satisfaction“-Variablen gegen das Geschlecht erzeugt:

```
. foreach SFB of varlist np9501-np9507 {
.     tabulate 'SFB' gender
. }
```

Aside: Im *Ernstfall* wird man das Wort „SFB“ als Elementnamen vermeiden. Ich verwende regelmäßig „var“.

Slide 20

### 1.5 Do-Files

Alle Stata-Befehle können sowohl interaktiv eingegeben als auch in eine Do-File („Syntax-Datei“) geschrieben werden. Die Befehlsdatei kann mit jedem beliebigen Texteditor geschrieben werden. Mit dem Befehl `doedit` wird der Stata Do-File Editor aufgerufen.

Hier wird der Do-File-Editor zusammen mit der Datei *an1.do* aufgerufen.

```
. copy http://www.stata-press.com/data/kk/anikk.do an1.do, replace
. doedit an1.do
```

Slide 21

## 1.6 Wie wird eine Befehlsdatei gestartet?

Befehlsdateien werden mit dem Befehl `do filename` gestartet. Als Dateiname wird der Name der jeweiligen Befehlsdatei verwendet. Wird keine Erweiterung angegeben, wird `.do` vorausgesetzt.

```
do an1
```

startet die Befehlsdatei `an1.do`.

Der Befehl `do` kann auch innerhalb von Do-Files verwendet werden, d.h. Do-Files können ineinander verschachtelt werden.

Slide 22

## 2 Variablen erzeugen und verändern

### 2.1 generate und replace

`generate` erstellt neue Variablen, `replace` verändert den Inhalt einer bereits vorhandenen Variable. Die Syntax der beiden Befehle ist identisch:

```
[by varlist:]generate varname=exp[in][if]
```

```
[by varlist:]replace varname=exp[in][if]
```

Slide 23

### 2.1.1 Variablennamen

Die Namen von Variablen, die mit `generate` gebildet werden können aus 32 Zeichen bestehen. Als Zeichen dürfen Buchstaben Zahlen und der Unterstrich (`_`) verwendet werden. Folgende Namen sind nicht erlaubt:

```
_all double long _rc
_b float _n _se
byte if _N _skip
_coef in _pi using
_cons int _pred with
```

Slide 24

### 2.1.2 Einfache Beispiele

```
. use data1, clear
. generate control = ybirth - ymove
. gen age = 1997 - ybirth
. gen age2 = ybirth^2
. gen log2inc= log(hhinc)/log(2)
```

Slide 25

### 2.1.3 generate/replace und if

Häufig wird `generate` und `replace` zusammen mit einer `if`-Bedingung verwendet.

```
. gen fulltime = 1 if emp <= 1
. replace fulltime = 0 if emp == 2
```

Slide 26

### 2.1.4 Relationale Operatoren

Hinter dem „make equal“- Zeichen können alle in Stata erlaubten Ausdrücke stehen. Also z.B. auch alle Ausdrücke mit relationalen Operatoren.

```
. gen men = gender == 1
. tab men
```

Wahre Ausdrücke bekommen in Stata den Wert 1, falsche den Wert 0.

Vorsicht: `gender==1` ist auch dann falsch, d.h. 0, wenn keine Angabe zum Geschlecht vorliegt:

```
. gen women = gender == 2 if gender < .
```

Slide 27

## 2.2 Label

### 2.2.1 Variablen-Label

```
. lab var fulltime "Vollzeit erwerbstaetig j/n"
```

Bei der Eingabe der Label kann man auf die Anführungszeichen verzichten, wenn das „Label“ keine besonderen Zeichen – Bindestriche, Leerzeichen, Kommata usw. – enthält. Umlaute und „ß“ werden nicht mit jedem verfügbaren Schriftsatz korrekt angezeigt. Man sollte sie darum nicht verwenden.

Die „Label“ können max. 80 Zeichen lang sein.

Slide 28

### 2.2.2 Value-Label

Value-Labels werden in zwei Schritten vergeben. Die Reihenfolge der Schritte ist beliebig:

- Die Definition eines Behälters, der Werte und zugeordnete „Label“ enthält, durch `label define`
- Die Verknüpfung dieses Behälters mit der gewünschten Variable durch `label value`

```
. lab def yesno 0 "nein" 1"ja"
. lab val men yesno
```

Slide 29

Ein Behälter kann mit mehreren Variablen verknüpft werden.

```
. lab val women yesno
```

Auf `lab val` kann auch verzichtet werden:

```
. gen worker:yesno = inlist(egp,7,8,9,10)
```

Slide 30

## 2.3 Spezielle Recodierungs-Befehle

### 2.3.1 recode

Spezialbefehl zur Zusammenfassung von Werten in Variablen mit vielen Ausprägungen.

```
. recode egph (1/3=1) (4 5=2) (6= 3) (nonmissing = 4), gen(egph4)
```

Für metrische Daten gibt es als Alternative die `recode()`-Funktion:

```
. gen agegroup = recode(age,20,40,60,98)
. tab agegroup
```

Slide 31

### 2.3.2 egen

Erweiterter `generate`-Befehl. Verwendet anstelle der generellen Stata-Funktionen die speziellen `egen`-Funktionen.

```
egen varname= egen-fcn(arg) [in][if][, options ]
```

Beispiele:

```
. egen worries = rsum(np95*)
. egen quality = neqany(kitchen-phone), values(2)
```

Slide 32

### 2.3.3 xtile

Befehle zur Gruppierung von Variablen an Perzentilen:

```
. xtile age15 = age, n(15)
```

Slide 33

### 2.3.4 separate

Erzeugt neue Variablen mit dem Inhalt einer alten Variable für unterschiedliche Gruppen. Der Befehl wird oft für Grafiken von Aggregatdaten benötigt:

```
. preserve
. collapse (mean) np11701, by(age15 gender)
. separate np11701, by(gender) veryshortlabel
. tw connected np11701? age15
. restore
```

Slide 34

### 2.4 Missing values

Fehlende Werte werden durch einen *Punkt* angesprochen und vergeben:

```
. replace egph = 99 if egph >= .
. replace egph = .a if egph == 98
. replace egph = .b if egph == 99
```

Um für mehreren Variablen gleichzeitig einen fehlenden Wert zu definieren können Schleifen konstruiert werden oder der Befehl `mvdecode` verwendet werden. `mvencode` wird für die umgekehrte Richtung verwendet.

Slide 35

### 2.5 Rekodieren in hierarchischen Datensätzen

Alle Datensätze sind „hierarchisch“. In *data1.dta* gibt es z.B. die Angaben von Befragten „innerhalb“ von Interviewern

```
. sort intnr
. list persnr intnr in 1/12
```

Beachten Sie die Ähnlichkeit dieser Datenstruktur zu Paneldaten!

Slide 36

#### 2.5.1 Einführendes Beispiel

Es soll eine Variable erzeugt werden, die für jeden Interviewer die Anzahl der von ihm interviewten ermittelt.

```
. by intnr, sort: generate intcount = _N
```

Slide 37

**2.5.2 `_n` und `by`:**

`_n` ist ein Platzhalter für die aktuelle Position einer Beobachtung im Datensatz.

```
. generate index = _n
. list persnr intrnr index in 1/12
```

Zusammen mit dem Prefix `by`: ist `_n` die Position eines Falles innerhalb der jeweiligen Kategorie der „`by`-Variablen“.

```
. by intrnr: generate intIndex =_n
. list persnr intrnr index intIndex in 1/12
```

Slide 38

**2.5.3 `_N` und `by`:**

`_N` ist der höchste Wert von `_n`. Über alle Fälle betrachtet ist `_N` die Fallzahl insgesamt.

```
. display _N
```

Hinter `by`: steht `_N` für den höchsten Wert von `_n` innerhalb der jeweiligen Kategorie der „`by`-Variablen“.

```
. list index persnr intrnr intIndex intcount in 1/12
```

Slide 39

**2.5.4 Rekodieren mit expliziten Subscripten**

In Stata können Variablen mit Subscripten versehen werden. Subscripte werden in eckige Klammern eingeschlossen und direkt an den Variablenamen angehängt.

```
. display ybirth[1]
. sort ybirth
. display ybirth[_N/2+1]
```

Slide 40

**2.5.5 Beispiele**

- Haushaltseinkommen

```
. use http://www.stata-press.com/data/kk/hierarch, clear
. list
. by hhnr, sort: generate hhinc=sum(income)
. by hhnr: replace hhinc = hhinc[_N]
```

- Klassenzugehörigkeit des Haushaltsvorsitzenden in einem Haushalt:

```
. by hhnr (hhpos), sort: generate occ_h = occ[1] if hhpos[1] == 1
```

### 3 Erstellen von Grafiken

#### 3.1 Struktur des Grafik-Befehls

1. Das Grafik-Kommando besteht normalerweise aus zwei Elementen: dem Befehl `graph` und der Angabe eines Grafik-Typs. Im folgenden Befehl ist `box` der Grafik-Typ:

```
. use data1 if hsize==1, clear
. graph box rent
```

2. Beim Grafik-Typ `twoway` muss ein drittes Element, der sog. Plot-Typ, spezifiziert werden. Hier ist ein Beispiel mit dem Plot-Typ `scatter`:

```
. graph twoway scatter rent sqfeet
```

Slide 41

Bei `twoway` kann der Befehl `graph` auch weggelassen werden. Zudem kann man bei den Plot-Typen `scatter` und `line` auf die Angabe von `twoway` verzichten:

```
. twoway scatter rent sqfeet
. scatter rent sqfeet
```

3. Die Plot-Typen des Grafik-Typs `twoway` können „überlagert“ werden:

```
. graph twoway (scatter rent sqfeet) (lfit rent sqfeet)
. scatter rent sqfeet || lfit rent sqfeet
```

4. Das Aussehen von Grafiken wird durch sog. Grafik-Schemata

Slide 42

beeinflusst. Die Änderung des Grafik-Schemas kann dieselbe Grafik mit einem deutlich anderen Aussehen erzeugen:

```
. set scheme economist
. scatter rent sqfeet
. set scheme s2color
```

Slide 43

#### 3.2 Grafikelemente innerhalb der Datenregion

##### 3.2.1 Grafiktyp

1. Balkendiagramme

```
. graph bar sqfeet, over(area)
```

2. Kuchendiagramme

```
. graph pie, over(area)
```

3. Punktdiagramme („Dot-Charts“)

```
. graph dot (mean) sqfeet, over(area)
```

Slide 44

Slide 45

## 4. Box-and-Whisker-Plots (Box-Plots)

```
. graph box sqfeet, over(area)
```

## 5. twoway-Grafiken

```
. graph twoway scatter rent sqfeet
. graph twoway function y = sin(x), range(1 20)
. graph twoway histogram rent
```

## 6. Scatterplot-Matrizen

```
. graph matrix np0105 rent sqfeet
```

Slide 46

## 3.2.2 Markersymbole

Unterschiedliche Datenbereiche werden mit unterschiedlichen Markersymbolen gekennzeichnet. Form, Größe und Farbe der Markersymbole können verändert werden. Dasselbe gilt für Farbe, Linienform und Liniendicke der Umrandung der Marker.

```
. generate rent_w = rent if state <= 9
. generate rent_e = rent if state >= 10 & state < .
. scatter rent_w rent_e sqfeet
. scatter rent_w rent_e sqfeet, msymbol(Oh T)
. scatter rent_w rent_e sqfeet, msymbol(Oh 0) msize(*.5 2)
```

Slide 47

## 3.2.3 Linien

An Stelle einer einzelnen Markierung für jeden Punkt der Grafik werden in Linien-Grafiken die Punkte der Grafik mit einer Linie verbunden.

```
. preserve
. use http://www.stata-press.com/data/kk/ka_temp, clear
. scatter mean year, msymbol(i) connect(direct) sort
```

der Plot-Typ `line` ist ein Synonym für den Scatterplot mit unsichtbarem Markersymbol und der Option `connect(direct)`:

```
. line mean year, sort
. restore
```

Slide 48

## 3.2.4 Referenzlinien und Beschriftungen

## • Referenzlinien

```
. scatter rent sqfeet, xline(1000) yline(1000)
```

## • Beschriftung von Markern

```
. preserve
. use http://www.stata-press.com/data/kk/data2agg, clear
. scatter lsat inc, mlabel(wave) mlabposition(12) mlabsize(small)
```

## • Arbiträrer Text

```
. sort wave
. local coor = lsat[1]
. line lsat wave, text('coor' 1984 "Happiness", placement(e))
```

Slide 49

### 3.3 Informationen außerhalb der Plotregion

- Beschriftung der Achsen

```
. restore
. scatter rent sqfeet, ylabel(0(400)2800)
```

- Tick-Lines

```
. scatter rent sqfeet, ytick(minmax) xmtick(##10)
```

- Achsentitel

```
. scatter rent sqfeet, ytitle("Rent (Monthly)" "in USD") ///
> xtitle("Home size" "in sqft.")
```

Slide 50

- Die Legende

```
. scatter rent_w rent_e sqfeet, legend(cols(1) ring(0) position(1))
```

- Grafik-Titel

```
. scatter rent sqfeet, ///
> title("Rent by Home Size") ///
> subtitle("Scatterplot") ///
> note("Data: GSOEP" "Randomized Public Use File") ///
> caption("This graph is used to demonstrate graph-titles.")
```

Slide 51

### 3.4 Multiple Grafiken

- Grafiken des Typs `twoway` können übereinander gelegt werden.
- Grafiken, die mit der Option `by()` in Teilgrafiken aufgelöst und dann gemeinsam dargestellt werden.
- Beliebige Grafiken, die mit Hilfe des Befehls `graph combine` zu einer gemeinsamen Darstellung zusammengeführt werden.

Slide 52

#### 3.4.1 Beispiele

```
. twoway (scatter rent sqfeet) ///
> (lfit rent_w sqfeet) ///
> (lfit rent_e sqfeet)
.
. scatter rent sqfeet, by(state, total)
.
. scatter rent_w sqfeet, name(west, replace)
. scatter rent_e sqfeet, name(east, replace)
. graph combine west east
```

Slide 53

### 3.5 Ausdruck und Export

Der Ausdruck von Grafiken erfolgt mit dem Stata-Befehl `graph print`

Der Export einer Grafik in eine Präsentationsprogramm erfolgt mit „Copy-and-Paste“ oder durch den Befehl `graph export`

Slide 54

## 4 Regressionsmodelle

### 4.1 Statistische Modelle mit Stata

Die Berechnung statistischer Modell mit Stata erfolgt immer nach dem gleichen Muster. Zuerste werden die Variablen entsprechend der Hypothesen *vorbereitet*. Danach wird ein Modell berechnet:

```
[ by varlist: ] modelcmd depvar varlist [ if ] [ in ] [ , options ]
```

Nach der Berechnung des Modells stehen jeweils weitgehend einheitliche Befehle zur Weiterbearbeitung der Ergebnisse zur Verfügung.

Slide 55

#### 4.1.1 Beispiel lineare Regressionsanalyse

```
. use data1, clear
. gen men:yesno = gender == 1
. gen fulltime:yesno = emp == 1 if emp <= 2
. gen ost:yesno = state <= 11 if state < .
. reg income yedu men fulltime ost
```

Slide 56

#### 4.1.2 Beispiel logistische Regressionsanalyse

```
. gen owner = renttype == 1 if renttype < .
. logit owner ost hsize hhinc
. probit owner ost hsize hhinc
. scobit owner ost hsize hhinc
```

Slide 57

## 4.2 Logistische Regression

### 4.2.1 Vorbemerkung

Stata hat zwei Kommandos zur Berechnung der logistischen Regression: `logit` und `logistic`. Der Befehl `logit` berichtet *b*-Koeffizienten, `logistic` berichtet Odds-Ratios. Im folgenden wird durchgehen `logit` verwendet.

Manche Anwender der logistischen Regression, insbesondere Bio-Statistiker und Mediziner, präferieren die Interpretation der Ergebnisse in Form der Odds-Ratios und verwenden deshalb eher `logistic`. Wer dem folgen möchte ersetze nachfolgend

Slide 58

### 4.2.2 Abhängige Variable

Bei der abhängigen Variable der logistischen Regression muss mindestens eine Kategorie 0 sein, da `logit` die logarithmierte Chance modelliert, dass die abhängige Variable von Null verschieden ist. Üblich ist eine abhängige Variable mit den Werten 0 („failure“) und 1 („success“):

```
. tab owner
```

Slide 59

### 4.2.3 Die unabhängigen Variable

Zur Erleichterung der Interpretation müssen die unabhängigen Variablen in der Regel *vor* der Berechnung des Modells rekodiert werden:

```
. generate age = 1997-ybirth
. summarize age if !mi(hhinc,owner,ost)
. generate cage = age - r(mean) if !mi(hhinc,owner,ost)
. summarize hhinc if !mi(age,owner,ost)
. generate chhinc = hhinc - r(mean) if !mi(age,owner,ost)
```

Slide 60

### 4.2.4 Der Modellbefehl

Die Syntax für logistische Regressionen ist die Gleiche wie bei allen statistischen Modellen in Stata. Zuerst der Befehl, dann die abhängige Variable dann eine Liste der unabhängigen Variablen:

```
. logit owner cage chhinc
```

Slide 61

#### 4.2.5 Post-Estimation Kommandos

Nach der Berechnung eines Regressionsmodells stehen die sog. Post-Estimation Kommandos zur Verfügung, mit denen die Ergebnisse des Modells weiterbearbeitet werden können.

```
. predict Phat
. lstat
. lfit, group(10)
. estimates store small
.
. logit owner cage chhinc ost
. estimates store big
. lrtest small full
.
. estimates table full small
```

Slide 62

#### 4.2.6 Marginaleffekte

Marginaleffekte können nach allen statistischen Modellen durch den „Post-Estimation“-Befehl `mfx` ermittelt werden. Marginaleffekte erleichtern die Interpretation der Koeffizienten

```
. mfx
. mfx, at(0 0 1)
```

Slide 63

#### 4.2.7 Conditional Effects Plots

Eine mächtige Alternative zur Interpretation mit Marginaleffekten sind Conditional Effects Plots. Sie können ebenfalls mit Hilfe von Post-Estimation Kommandos erstellt werden:

```
. gen Lhatw = _b[_cons] + _b[cage] * 0 + _b[chhinc] * chhinc
. gen Lhato = _b[_cons] + _b[cage] * 0 + _b[chhinc] * chhinc + _b[ost] * 1
. gen Phatw = exp(Lhatw)/(1 + exp(Lhatw))
. gen Phato = exp(Lhato)/(1 + exp(Lhato))
. line Phat? hhinc, sort
```

Siehe hierzu auch `search spostado`.

Slide 64

### 4.3 Regressionsdiagnostik

#### 4.3.1 Linearität

Die funktionale Form des Zusammenhangs bei der logistischen Regression ist linear bzgl. der Logits und S-förmig bzgl. der Wahrscheinlichkeiten.

- Lokale Mean-Regression

```
. generate groupage = autocode(age,15,16,90)
. egen mowner = mean(owner), by(groupage)
. graph twoway (scatter owner age, jitter(2)) (line mowner age, sort)
```

Slide 65

- LOWESS

```
. lowess owner age, jitter(2) bwidth(.5)
```

- Dummy-Variablen

```
. tabulate groupage, gen(aged)
. logit owner aged2-aged15 chhinc ost
. matrix b = e(b)'
. svmat b, names(b)
. generate index = _n
. graph twoway line b1 index in 1/14, sort
```

Slide 66

#### 4.3.2 Einflussreiche Beobachtungen

- Discrepancy vs. Leverage Plot

```
. logit owner cage chhinc ost
. predict leverage, hat
. predict spres, rstandard
. summarize leverage
. local a = r(mean)
. local b = 2 * r(mean)
. local c = 3 * r(mean)
. scatter spres leverage, xline('a' 'b' 'c') yline(-2 0 2) ms(oh)
```

Slide 67

- $\Delta\beta$  vs. Predicted

```
. predict db, dbeta
. separate db, by(owner)
. graph twoway scatter db0 db1 Phat
```

- $\Delta\chi$  vs. Predicted

```
. predict dx2, dx2
. separate dx2, by(owner)
. graph twoway scatter dx20 dx21 Phat, yline(4)
```

Slide 68

#### 4.3.3 Dummies/Interaktionseffekte/Transformationen

Generell müssen die unabhängigen Variablen *vor* der Modellschätzung aufbereitet werden. Dazu gibt es u.a. folgende Hilfsmittel:

- Dummy-Variablen:

```
. tab marital, gen(marital)
. logit owner cage chhinc ost marital2-marital6
. xi i.marital
. logit owner cage chhinc ost _I*
. xi: logit owner cage chhinc ost i.marital
```

Slide 69

- Interaktionseffekte

```
. xi: logit owner cage chhinc i.ost*i.marital
```

Das benutzerdefinierte Programm `xi3` erweitert die Funktionalität von `xi` auf Mehrwegsinteraktionseffekte.

- Transformationen

```
. qladder hhinc
. bcskew0 hhincbc = hhinc
```

Slide 70

#### 4.4 Publikationsfertigen Output erzeugen

Das benutzerdefinierte Programm `estout` dient dazu gespeicherte Ergebnisse von verschiedenen Modellen so zu formatieren, wie in Zeitschriften üblich.

```
. * net sj 5-3
. * net install st0085
. estout small full
> , stats(r2_p N, fmt(%3.2f %3.0f) labels("Pseudo R2" N)) ///
> cells( b(star fmt(%3.2f)) t(par fmt(%3.2f)) ) ///
> starlevels(* 0.05 ** 0.01)
```

Slide 71

## 5 Modelle für hierarchische Daten

Gegeben hierarchisch gegliederte Daten, wie z.B. Schüler innerhalb von Klassen innerhalb von Schulen, oder wiederholte Messungen eines Items an einer Person

```
. use persnr np95* using data1, clear
. reshape long np950, i(persnr) j(item)
. l in 1/20
```

oder Paneldaten:

```
. use http://www.stata-press.com/data/kk/data2w, clear
. reshape long mar hour inc lsat, i(persnr) j(wave)
. l persnr wave lsat hour in 1/30
```

Berechnet man für derartige Daten das einfache Regressionsmodell

Slide 72

```
. reg lsat hour
```

zu zeigt sich, dass die Residuen eines Befragten („Clusters“) einander ähnlicher sind als die Residuen von Beobachtungen aus verschiedenen Clustern. Anders formuliert: die Residuen korrelieren untereinander („Autokorrelation“):

```
. preserve
. predict res, resid
. reshape wide lsat mar hour inc res, i(persnr) j(wave)
. corr res198?
```

Slide 73

## 5.1 Einfache Fixed-Effects Modelle

Bei FE Modellen werden Dummy-Variablen der Clusterausprägungen in das Modell eingeschlossen:

```
. use http://www.stata-press.com/data/kk/beatles, clear
. describe
. list
. tab persnr, gen(d)
. regress lsat age d2-d4
```

Dies ist nur bei kleinem  $n$  möglich. Bei großem  $n$  kann man sich eines Tricks bedienen, der in `xtreg`, `fe` implementiert ist.

```
. xtreg lsat age, i(persnr) fe
. restore
. xtreg lsat hour, i(persnr) fe
```

Slide 74

### 5.1.1 Conditional Logit

Für binäre abhängige Variablen gibt es `xtlogit`, `fe`, welches äquivalent mit einem Conditional Logit ist:

```
. preserve
. use persnr np95* using data1, clear
. reshape long np950, i(persnr) j(item)
. gen sorgen = np950 == 1 if !mi(np950)
. tab item, gen(item)
. xtlogit sorgen item2-item6, i(persnr) fe
. restore
```

Note: die Koeffizienten dieses Modells lassen sich als Item-Schwierigkeit eines Rasch-Modells interpretieren. Entsprechend *wären* die Koeffizienten der Personen-Dummies die Personenparameter, die im Modell jedoch nicht geschätzt werden.

Slide 75

## 5.2 Einfache Random-Effects Modelle

Bei Random-Effects Modellen wird die Ähnlichkeit der Beobachtungen aus einem Cluster nicht durch einen Koeffizienten kontrolliert, sondern dadurch, dass man eine sie als Folge einer normalverteilten „latenten“ Variable ansieht.

```
. xtreg lsat hour, re i(persnr)
```

Die gemessene Lebenszufriedenheit wird hier als Folge einer „latenten“ Variable Lebenszufriedenheit und eines konstanten Effekts der Arbeitsstunden modelliert.

Für binäre Outcomes funktioniert `xtlogit`, `re` entsprechend.

Slide 76

## 5.3 Generalized Estimation Equation (GEE)

Beim Random-Effect Modell wird die Korrelation der Residuen eines Clusters als konstant angesehen. Mit GEE lässt sich diese Korrelation weiter strukturieren. Im folgenden wird z.B. angenommen, dass die Ähnlichkeit der Beobachtungen eines Befragten umso stärker ist, je näher die Beobachtungen einander zeitlich sind:

```
. xtgee lsat hour, i(persnr) t(wave) corr(ar1)
```

`xtgee` erlaubt die Verwendung der aus dem GLM-Ansatz bekannten Linkfunktionen und Verteilungsannahmen. Die Struktur der Residuen-Korrelationsmatrix kann frei spezifiziert werden.

## 5.4 Multilevel mixed-effects linear regression

Dies sind direkte Verallgemeinerungen der einfachen Random-Effects Modelle. Es können hier nicht nur Random-Effects für die Konstante („Random-Intercept“) Modell spezifiziert werden, sondern auch für die Koeffizienten („Random-Coefficient“). Das Modell erlaubt darüber hinaus die Verwendung mehrerer Hierarchieebenen.

Slide 77

- Random Intercept Model
 

```
. xtmixed lsat hour || persnr:
```
- Random Coefficient Model
 

```
. xtmixed lsat hour || persnr: hour
```

- 3-Level Random Intercept Model,
 

```
. xtmixed lsat hour || persnr: || hhnr:
```
- 3-Level Random Coefficient Model,
 

```
. * xtmixed lsat hour || persnr: hour || hhnr: hour
```

Slide 78

## 5.5 GLLAMM

„Generalized Latent Linear and Mixed Models“ sind direkte verallgemeinerungen von Linear Mixed Models. Mit ihnen können Latente-Variablen Modelle mit beliebige GLMs berechnet werden.

Slide 79

- Random Intercept Logit Model

```
. * ssc install gllamm
. egen id = group(persnr)
. keep if id < 20
. replace lsat = lsat > 6
. gllamm lsat hour, i(persnr) link(logit) family(binom)
```

- Random Slope Logit Model

```
. gen con = 1
. eq int: con
. eq slope: hour
. * gllamm lsat hour, i(persnr) nrf(2) eqs(int slope) adapt ///
>          link(logit) family(binom)
```

Slide 80

## 6 Literatur

Der Großteil der Beispiele stammt aus Kohler und Kreuter (2005), welches in Kürze auch in deutscher Sprache erscheint.

Für die logistische Regression mit Stata empfiehlt sich hinaus die Lektüre von Long und Freese (2001). Zur Modellierung für hierarchische Daten mit Stata siehe insbesondere Rabe-Hesketh und Skrondal (2005). Mitchell (2004) ist eine Einführung in Stata-Grafiken.

Alle genannten Bücher sind über den deutschen Distributor von Stata zu beziehen.

Slide 81

## Literatur

Kohler, Ulrich und Frauke Kreuter (2005). *Data Analysis Using Stata*. College Station: Stata Press.

Long, J. Scott und Jeremy Freese (2001). *Regression Models for Categorical Dependent Variables Using Stata*. College Station: Stata Press.

Mitchell, Michael N. (2004). *A Visual Guide to Stata Graphics*. College Station: Stata Press.

Rabe-Hesketh, Sophia und Anders Skrondal (2005). *Multilevel and Longitudinal Modeling Using Stata*. Stata-Press.