

Data Analysis Using Stata

Third Edition



Copyright © 2005, 2009, 2012 by StataCorp LP
All rights reserved. First edition 2005
Second edition 2009
Third edition 2012

Published by Stata Press, 4905 Lakeway Drive, College Station, Texas 77845
Typeset in L^AT_EX 2_•

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN-10: 1-59718-110-2
ISBN-13: 978-1-59718-110-5

Library of Congress Control Number: 2012934051

No part of this book may be reproduced, stored in a retrieval system, or transcribed, in any form or by any means—electronic, mechanical, photocopy, recording, or otherwise—with or without the prior written permission of StataCorp LP.

Stata, **STATA**, Stata Press, Mata, **MATA**, and NetCourse are registered trademarks of StataCorp LP.

Stata and Stata Press are registered trademarks with the World Intellectual Property Organization of the United Nations.

L^AT_EX 2_• is a trademark of the American Mathematical Society.

Contents

List of tables	xvii
List of figures	xix
Preface	xxi
Acknowledgments	xxvii
1 The first time	1
1.1 Starting Stata	1
1.2 Setting up your screen	2
1.3 Your first analysis	2
1.3.1 Inputting commands	2
1.3.2 Files and the working memory	3
1.3.3 Loading data	3
1.3.4 Variables and observations	5
1.3.5 Looking at data	7
1.3.6 Interrupting a command and repeating a command	8
1.3.7 The variable list	8
1.3.8 The in qualifier	9
1.3.9 Summary statistics	9
1.3.10 The if qualifier	11
1.3.11 Defining missing values	11
1.3.12 The by prefix	12
1.3.13 Command options	13
1.3.14 Frequency tables	14
1.3.15 Graphs	15
1.3.16 Getting help	16

1.3.17	Recoding variables	17
1.3.18	Variable labels and value labels	18
1.3.19	Linear regression	19
1.4	Do-files	20
1.5	Exiting Stata	22
1.6	Exercises	23
2	Working with do-files	25
2.1	From interactive work to working with a do-file	25
2.1.1	Alternative 1	26
2.1.2	Alternative 2	27
2.2	Designing do-files	30
2.2.1	Comments	31
2.2.2	Line breaks	32
2.2.3	Some crucial commands	33
2.3	Organizing your work	35
2.4	Exercises	39
3	The grammar of Stata	41
3.1	The elements of Stata commands	41
3.1.1	Stata commands	41
3.1.2	The variable list	43
List of variables: Required or optional	43	
Abbreviation rules	43	
Special listings	45	
3.1.3	Options	45
3.1.4	The in qualifier	47
3.1.5	The if qualifier	48
3.1.6	Expressions	51
Operators	52	
Functions	54	
3.1.7	Lists of numbers	55

3.1.8	Using filenames	56
3.2	Repeating similar commands	57
3.2.1	The by prefix	58
3.2.2	The foreach loop	59
	The types of foreach lists	61
	Several commands within a foreach loop	62
3.2.3	The forvalues loop	62
3.3	Weights	63
	Frequency weights	64
	Analytic weights	66
	Sampling weights	67
3.4	Exercises	68
4	General comments on the statistical commands	71
4.1	Regular statistical commands	71
4.2	Estimation commands	74
4.3	Exercises	76
5	Creating and changing variables	77
5.1	The commands generate and replace	77
5.1.1	Variable names	78
5.1.2	Some examples	79
5.1.3	Useful functions	82
5.1.4	Changing codes with by, _n, and _N	85
5.1.5	Subscripts	89
5.2	Specialized recoding commands	91
5.2.1	The recode command	91
5.2.2	The egen command	92
5.3	Recoding string variables	94
5.4	Recoding date and time	98
5.4.1	Dates	98
5.4.2	Time	102

5.5	Setting missing values	105
5.6	Labels	107
5.7	Storage types, or the ghost in the machine	111
5.8	Exercises	112
6	Creating and changing graphs	115
6.1	A primer on graph syntax	115
6.2	Graph types	116
6.2.1	Examples	117
6.2.2	Specialized graphs	119
6.3	Graph elements	119
6.3.1	Appearance of data	121
Choice of marker	123	
Marker colors	125	
Marker size	126	
Lines	126	
6.3.2	Graph and plot regions	129
Graph size	130	
Plot region	130	
Scaling the axes	131	
6.3.3	Information inside the plot region	133
Reference lines	133	
Labeling inside the plot region	134	
6.3.4	Information outside the plot region	138
Labeling the axes	139	
Tick lines	142	
Axis titles	143	
The legend	144	
Graph titles	146	
6.4	Multiple graphs	147
6.4.1	Overlaying many twoway graphs	147

6.4.2	Option by()	149
6.4.3	Combining graphs	150
6.5	Saving and printing graphs	152
6.6	Exercises	154
7	Describing and comparing distributions	157
7.1	Categories: Few or many?	158
7.2	Variables with few categories	159
7.2.1	Tables	159
Frequency tables		159
More than one frequency table		160
Comparing distributions		160
Summary statistics		162
More than one contingency table		163
7.2.2	Graphs	163
Histograms		164
Bar charts		166
Pie charts		168
Dot charts		169
7.3	Variables with many categories	170
7.3.1	Frequencies of grouped data	171
Some remarks on grouping data		171
Special techniques for grouping data		172
7.3.2	Describing data using statistics	173
Important summary statistics		174
The summarize command		176
The tabstat command		177
Comparing distributions using statistics		178
7.3.3	Graphs	186
Box plots		187
Histograms		189

Kernel density estimation	191
Quantile plot	195
Comparing distributions with Q–Q plots	199
7.4 Exercises	200
8 Statistical inference	201
8.1 Random samples and sampling distributions	202
8.1.1 Random numbers	202
8.1.2 Creating fictitious datasets	203
8.1.3 Drawing random samples	207
8.1.4 The sampling distribution	208
8.2 Descriptive inference	213
8.2.1 Standard errors for simple random samples	213
8.2.2 Standard errors for complex samples	215
Typical forms of complex samples	215
Sampling distributions for complex samples	217
Using Stata’s svy commands	219
8.2.3 Standard errors with nonresponse	222
Unit nonresponse and poststratification weights	222
Item nonresponse and multiple imputation	223
8.2.4 Uses of standard errors	230
Confidence intervals	231
Significance tests	233
Two-group mean comparison test	238
8.3 Causal inference	242
8.3.1 Basic concepts	242
Data-generating processes	242
Counterfactual concept of causality	244
8.3.2 The effect of third-class tickets	246
8.3.3 Some problems of causal inference	248
8.4 Exercises	250

9	Introduction to linear regression	253
9.1	Simple linear regression	256
9.1.1	The basic principle	256
9.1.2	Linear regression using Stata	260
	The table of coefficients	261
	The table of ANOVA results	266
	The model fit table	268
9.2	Multiple regression	270
9.2.1	Multiple regression using Stata	271
9.2.2	More computations	274
	Adjusted R ²	274
	Standardized regression coefficients	276
9.2.3	What does “under control” mean?	277
9.3	Regression diagnostics	279
9.3.1	Violation of E(ϵ_i) = 0	280
	Linearity	283
	Influential cases	286
	Omitted variables	295
	Multicollinearity	296
9.3.2	Violation of Var(ϵ_i) = σ^2	296
9.3.3	Violation of Cov(ϵ_i, ϵ_j) = 0, $i \neq j$	299
9.4	Model extensions	300
9.4.1	Categorical independent variables	301
9.4.2	Interaction terms	304
9.4.3	Regression models using transformed variables	308
	Nonlinear relationships	309
	Eliminating heteroskedasticity	312
9.5	Reporting regression results	313
9.5.1	Tables of similar regression models	313
9.5.2	Plots of coefficients	316

9.5.3	Conditional-effects plots	321
9.6	Advanced techniques	324
9.6.1	Median regression	324
9.6.2	Regression models for panel data	327
From wide to long format	328	
Fixed-effects models	332	
9.6.3	Error-components models	337
9.7	Exercises	339
10	Regression models for categorical dependent variables	341
10.1	The linear probability model	342
10.2	Basic concepts	346
10.2.1	Odds, log odds, and odds ratios	346
10.2.2	Excursion: The maximum likelihood principle	351
10.3	Logistic regression with Stata	354
10.3.1	The coefficient table	356
Sign interpretation	357	
Interpretation with odds ratios	357	
Probability interpretation	359	
Average marginal effects	361	
10.3.2	The iteration block	362
10.3.3	The model fit block	363
Classification tables	364	
Pearson chi-squared	367	
10.4	Logistic regression diagnostics	368
10.4.1	Linearity	369
10.4.2	Influential cases	372
10.5	Likelihood-ratio test	377
10.6	Refined models	379
10.6.1	Nonlinear relationships	379
10.6.2	Interaction effects	381

10.7	Advanced techniques	384
10.7.1	Probit models	385
10.7.2	Multinomial logistic regression	387
10.7.3	Models for ordinal data	391
10.8	Exercises	393
11	Reading and writing data	395
11.1	The goal: The data matrix	395
11.2	Importing machine-readable data	397
11.2.1	Reading system files from other packages	398
Reading Excel files	398	
Reading SAS transport files	402	
Reading other system files	402	
11.2.2	Reading ASCII text files	402
Reading data in spreadsheet format	402	
Reading data in free format	405	
Reading data in fixed format	407	
11.3	Inputting data	410
11.3.1	Input data using the Data Editor	410
11.3.2	The input command	411
11.4	Combining data	415
11.4.1	The GSOEP database	415
11.4.2	The merge command	417
Merge 1:1 matches with rectangular data	418	
Merge 1:1 matches with nonrectangular data	421	
Merging more than two files	424	
Merging m:1 and 1:m matches	425	
11.4.3	The append command	429
11.5	Saving and exporting data	433
11.6	Handling large datasets	434
11.6.1	Rules for handling the working memory	434

11.6.2	Using oversized datasets	435
11.7	Exercises	435
12	Do-files for advanced users and user-written programs	437
12.1	Two examples of usage	437
12.2	Four programming tools	439
12.2.1	Local macros	439
Calculating with local macros	440	
Combining local macros	441	
Changing local macros	442	
12.2.2	Do-files	443
12.2.3	Programs	443
The problem of redefinition	445	
The problem of naming	445	
The problem of error checking	445	
12.2.4	Programs in do-files and ado-files	446
12.3	User-written Stata commands	449
12.3.1	Sketch of the syntax	451
12.3.2	Create a first ado-file	452
12.3.3	Parsing variable lists	453
12.3.4	Parsing options	454
12.3.5	Parsing if and in qualifiers	456
12.3.6	Generating an unknown number of variables	457
12.3.7	Default values	459
12.3.8	Extended macro functions	461
12.3.9	Avoiding changes in the dataset	463
12.3.10	Help files	465
12.4	Exercises	467
13	Around Stata	469
13.1	Resources and information	469
13.2	Taking care of Stata	470

13.3	Additional procedures	471
13.3.1	Stata Journal ado-files	471
13.3.2	SSC ado-files	473
13.3.3	Other ado-files	474
13.4	Exercises	475
	References	477
	Author index	483
	Subject index	487

Tables

3.1	Abbreviations of frequently used commands	42
3.2	Abbreviations of lists of numbers and their meanings	56
3.3	Names of commands and their associated file extensions	57
6.1	Available file formats for graphs	154
7.1	Quartiles for the distributions	176
9.1	Apartment and household size	267
9.2	A table of nested regression models	314
9.3	Ways to store panel data	329
10.1	Probabilities, odds, and logits	349
11.1	Filename extensions used by statistical packages	397
11.2	Average temperatures (in °F) in Karlsruhe, Germany, 1984–1990 . .	410

Figures

6.1	Types of graphs	118
6.2	Elements of graphs	120
6.3	The Graph Editor in Stata for Windows	138
7.1	Distributions with equal averages and standard deviations	175
7.2	Part of a histogram	192
8.1	Beta density functions	204
8.2	Sampling distributions of complex samples	218
8.3	One hundred 95% confidence intervals	232
9.1	Scatterplots with positive, negative, and weak correlation	254
9.2	Exercise for the OLS principle	259
9.3	The Anscombe quartet	280
9.4	Residual-versus-fitted plots of the Anscombe quartet	282
9.5	Scatterplots to picture leverage and discrepancy	291
9.6	Plot of regression coefficients	317
10.1	Sample of a dichotomous characteristic with the size of 3	352
11.1	The Data Editor in Stata for Windows	396
11.2	Excel file <code>popst1.xls</code> loaded into OpenOffice Calc	399
11.3	Representation of <code>merge</code> for 1:1 matches with rectangular data	418
11.4	Representation of <code>merge</code> for 1:1 matches with nonrectangular data	422
11.5	Representation of <code>merge</code> for m:1 matches	426
11.6	Representation of <code>append</code>	430
12.1	Beta version of <code>denscomp.ado</code>	465

Preface

As you may have guessed, this book discusses data analysis, especially data analysis using Stata. We intend for this book to be an introduction to Stata; at the same time, the book also explains, for beginners, the techniques used to analyze data.

Data Analysis Using Stata does not merely discuss Stata commands but demonstrates all the steps of data analysis using practical examples. The examples are related to public issues, such as income differences between men and women, and elections, or to personal issues, such as rent and living conditions. This approach allows us to avoid using social science theory in presenting the examples and to rely on common sense. We want to emphasize that these familiar examples are merely standing in for actual scientific theory, without which data analysis is not possible at all. We have found that this procedure makes it easier to teach the subject and use it across disciplines. Thus this book is equally suitable for biometricalians, econometricians, psychometricalians, and other “metricians”—in short, for all who are interested in analyzing data.

Our discussion of commands, options, and statistical techniques is in no way exhaustive but is intended to provide a fundamental understanding of Stata. Having read this book and solved the problems in it, the reader should be able to solve all further problems to which Stata is applicable.

We strongly recommend to both beginners and advanced readers that they read the preface and the first chapter (entitled *The first time*) attentively. Both serve as a guide throughout the book. Beginners should read the chapters in order while sitting in front of their computers and trying to reproduce our examples. More-advanced users of Stata may benefit from the extensive index and may discover a useful trick or two when they look up a certain command. They may even throw themselves into programming their own commands. Those who do not (yet) have access to Stata are invited to read the chapters that focus on data analysis, to enjoy them, and maybe to translate one or another hint (for example, about diagnostics) into the language of the statistical package to which they do have access.

Structure

The first time (chapter 1) shows what a typical session of analyzing data could look like. To beginners, this chapter conveys a sense of Stata and explains some basic concepts such as variables, observations, and missing values. To advanced users who already have experience in other statistical packages, this chapter offers a quick entry into Stata.

Advanced users will find within this chapter many cross-references, which can therefore be viewed as an extended table of contents. The rest of the book is divided into three parts, described below.

Chapters 2–6 serve as an introduction to the basic tools of Stata. Throughout the subsequent chapters, these tools are used extensively. It is not possible to portray the basic Stata tools, however, without using some of the statistical techniques explained in the second part of the book. The techniques described in chapter 6 may not seem useful until you begin working with your own results, so you may want to skim chapter 6 now and read it more carefully when you need it.

Throughout chapters 7–10, we show examples of data analysis. In chapter 7, we present techniques for describing and comparing distributions. Chapter 8 covers statistical inference and explains whether and how one can transfer judgments made from a statistic obtained in a dataset to something that is more than just the dataset. Chapter 9 introduces linear regression using Stata. It explains in general terms the technique itself and shows how to run a regression analysis using an example file. Afterward, we discuss how to test the statistical assumptions of the model. We conclude the chapter with a discussion of sophisticated regression models and a quick overview of further techniques. Chapter 10, in which we describe regression models for categorical dependent variables, is structured in the same way as the previous chapter to emphasize the similarity between these techniques.

Chapters 11–13 deal with more-advanced Stata topics that beginners may not need. In chapter 11, we explain how to read and write files that are not in the Stata format. At the beginning of chapter 12, we introduce some special tools to aid in writing do-files. You can use these tools to create your own Stata commands and then store them as ado-files, which are explained in the second part of the chapter. It is easy to write Stata commands, so many users have created a wide range of additional Stata commands that can be downloaded from the Internet. In chapter 13, we discuss these user-written commands and other resources.

Using this book: Materials and hints

The only way to learn how to analyze data is to do it. To help you learn by doing, we have provided data files (available on the Internet) that you can use with the commands we discuss in this book. You can access these files from within Stata or by downloading a zip archive.

Please do not hesitate to contact us if you have any trouble obtaining these data files and do-files.¹

1. The data we provide and all commands we introduce assume that you use Stata 12 or higher. Please contact us if you have an older version of Stata.

- If the machine you are using to run Stata is connected to the Internet, you can download the files from within Stata. To do this, type the following commands in the Stata Command window (see the beginning of chapter 1 for information about using Stata commands).

```
. mkdir c:\data\kk3  
. cd c:\data\kk3  
. net from http://www.stata-press.com/data/kk3/  
. net get data
```

These commands will install the files needed for all chapters except section 11.4. Readers of this section will need an additional data package. You can download these files now or later on by typing

```
. mkdir c:\data\kk3\kksoep  
. cd c:\data\kk3\kksoep  
. net from http://www.stata-press.com/data/kk3/  
. net get kksoep  
. cd ..
```

If you are using a Mac or Unix system, substitute a suitable directory name in the first two commands, respectively.

- The files are also stored as a zip archive, which you can download by pointing your browser to <http://www.stata-press.com/data/kk3/kk3.zip>.

To extract the file `kk3.zip`, create a new folder: `c:\data\kk3`. Copy `kk3.zip` into this folder. Unzip the file `kk3.zip` using any program that can unzip zip archives. Most computers have such a program already installed; if not, you can get one for free over the Internet.² Make sure to preserve the `kksoep` subdirectory contained in the zip file.

Throughout the book, we assume that your current working directory (folder) is the directory where you have stored our files. This is important if you want to reproduce our examples. At the beginning of chapter 1, we will explain how you can find your current working directory. Make sure that you do not replace any file of ours with a modified version of the same file; that is, avoid using the command `save`, `replace` while working with our files.

We cannot say it too often: the only way to learn how to analyze data is to analyze data yourself. We strongly recommend that you reproduce our examples in Stata as you read this book. A line that is written **in this font** and begins with a period (which itself should not be typed by the user) represents a Stata command, and we encourage you to enter that command in Stata. Typing the commands and seeing the results or graphs will help you better understand the text, because we sometimes omit output to save space.

As you follow along with our examples, you must type all commands that are shown, because they build on each other within a chapter. Some commands will only work if

2. For example, “pkzip” is free for private use, developed by the company PKWARE. You can find it at <http://pkzip.en.softonic.com/>.

you have entered the previous commands. If you do not have time to work through a whole chapter at once, you can type the command

```
. save mydata, replace
```

before you exit Stata. When you get back to your work later, type

```
. use mydata
```

and you will be able to continue where you left off.

The exercises at the end of each chapter use either data from our data package or data used in the Stata manuals. StataCorp provides these datasets online.³ They can be used within Stata by typing the command `webuse filename`. However, this command assumes that your computer is connected to the Internet; if it is not, you have to download the respective files manually from a different computer.

This book contains many graphs, which are almost always generated with Stata. In most cases, the Stata command that generates the graph is printed above the graph, but the more complicated graphs were produced by a Stata do-file. We have included all of these do-files in our file package so that you can study these files if you want to produce a similar graph (the name of the do-file needed for each graph is given in a footnote under the graph).

If you do not understand our explanation of a particular Stata command or just want to learn more about it, use the Stata `help` command, which we explain in chapter 1. Or you can look in the Stata manuals, which are available in printed form and as PDF files. When we refer to the manuals, [R] `summarize`, for example, refers to the entry describing the `summarize` command in the *Stata Base Reference Manual*. [U] **18 Programming Stata** refers to chapter 18 of the *Stata User's Guide*. When you see a reference like these, you can use Stata's online help (see section 1.3.16) to get information on that keyword.

Teaching with this manual

We have found this book to be useful for introductory courses in data analysis, as well as for courses on regression and on the analysis of categorical data. We have used it in courses at universities in Germany and the United States. When developing your own course, you might find it helpful to use the following outline of a course of lectures of 90 minutes each, held in a computer lab.

To teach an introductory course in data analysis using Stata, we recommend that you begin with chapter 1, which is designed to be an introductory lecture of roughly 1.5 hours. You can give this first lecture interactively, asking the students substantive questions about the income difference between men and women. You can then answer them by entering Stata commands, explaining the commands as you go. Usually, the students

3. They are available at <http://www.stata-press.com/data/r12/>.

name the independent variables used to examine the stability of the income difference between men and women. Thus you can do a stepwise analysis as a question-and-answer game. At the end of the first lecture, the students should save their commands in a log file. As a homework assignment, they should produce a commented do-file (it might be helpful to provide them with a template of a do-file).

The next two lectures should work with chapters 3–5 and can be taught a bit more conventionally than the introduction. It will be clear that your students will need to learn the *language* of a program first. These two lectures need not be taught interactively but can be delivered section by section without interruption. At the end of each section, give the students time to retype the commands and ask questions. If time is limited, you can skip over sections 3.3 and 5.7. You should, however, make time for a detailed discussion of sections 5.1.4 and 5.1.5 and the examples in them; both sections contain concepts that will be unfamiliar to the student but are very powerful tools for users of Stata.

One additional lecture should suffice for an overview of the commands and some interactive practice in the graphs chapter (chapter 6).

Two lectures can be scheduled for chapter 7. One example for a set of exercises to go along with this chapter is given by Donald Bentley and is described on the webpage <http://www.amstat.org/publications/jse/v3n3/datasets.dawson.html>. The necessary files are included in our file package.

A reasonable discussion of statistical inference will take two lectures. The material provided in chapter 8 shows necessary elements for simulations, which allows for a hands-on discussion of sampling distributions. The section on multiple imputation can be skipped in introductory courses.

Three lectures should be scheduled for chapter 9. According to our experience, even with an introductory class, you can cover sections 9.1, 9.2, and 9.3 in one lecture each. We recommend that you let the students calculate the regressions of the Anscombe data (see page 279) as a homework assignment or an in-class activity before you start the lecture on regression diagnostics.

We recommend that toward the end of the course, you spend two lectures on chapter 11 introducing data entry, management, and the like, before you end the class with chapter 13, which will point the students to further Stata resources.

Many of the instructional ideas we developed for our book have found their way into the small computing lab sessions run at the UCLA Department of Statistics. The resources provided there are useful complements to our book when used for introductory statistics classes. More information can be found at <http://www.stat.ucla.edu/labs/>, including labs for older versions of Stata.

In addition to using this book for a general introduction to data analysis, you can use it to develop a course on regression analysis (chapter 9) or categorical data analysis (chapter 10). As with the introductory courses, it is helpful to begin with chapter 1, which gives a good overview of working with Stata and solving problems using Stata's online help. Chapter 13 makes a good summary for the last session of either course.

Acknowledgments

This third American edition of our book is all new: We wrote from scratch a new chapter on statistical inference, as requested by many readers and colleagues. We updated the syntax used in all chapters to Stata 12 or higher. We added explanations for Stata's amazing new factor-variable notation and the very useful new commands `margins` and `marginsplot`. We detailed a larger number of additional functions that we found to be very useful in our work. And last but not least, we created more contemporary versions of all the datasets used throughout the book.

The new version of our example dataset `data1.dta` is based on the German Socio-Economic Panel (GSOEP) of 2009. It retains more features of the original dataset than does the previous one, which allows us to discuss inference statistics for complex surveys with this real dataset.

Textbooks, and in particular self-learning guides, improve especially through feedback from readers. Among many others, we therefore thank K. Agbo, H. Blatt, G. Consonni, F. Ebcinoglu, A. Faber, L. R. Gimenezduarte, T. Gregory, D. Hanebuth, K. Heller, J. H. Hoeffler, M. Johri, A. Just, R. Liebscher, T. Morrison, T. Nshimiyimana, D. Possenriede, L. Powner, C. Prinz, K. Recknagel, T. Rogers, E. Sala, L. Schötz, S. Steschmi, M. Strahan, M. Tausendpfund, F. Wädlich, T. Xu, and H. Zhang.

Many other factors contribute to creating a usable textbook. Half the message of the book would have been lost without good data. We greatly appreciate the help and data we received from the SOEP group at the German Institute for Economic Research (DIW), and from Jan Goebel in particular. Maintaining an environment that allowed us to work on this project was not always easy; we thank the WZB, the JPSM, the IAB, and the LMU for being such great places to work. We also wish to thank our colleagues S. Eckman, J.-P. Heisig, A. Lee, M. Obermaier, A. Radenacker, J. Sackshaug, M. Sander-Blanck, E. Stuart, O. Tewes, and C. Thewes for all their critique and assistance. Last but not least, we thank our families and friends for supporting us at home.

We both take full and equal responsibility for the entire book. We can be reached at `kkstata@web.de`, and we always welcome notice of any errors as well as suggestions for improvements.

1 The first time

Welcome! In this chapter, we will show you several typical applications of computer-aided data analysis to illustrate some basic principles of Stata. Advanced users of data analysis software may want to look through the book for answers to specific problems instead of reading straight through. Therefore, we have included many cross-references in this chapter as a sort of distributed table of contents.

If you have never worked with statistical software, you may not immediately understand the commands or the statistical techniques behind them. Do not be discouraged; reproduce our steps anyway. If you do, you will get some training and experience working with Stata. You will also get used to our jargon and get a feel for how we do things. If you have specific questions, the cross-references in this chapter can help you find answers.

Before we begin, you need to know that Stata is command-line oriented, meaning that you type a combination of letters, numbers, and words at a command line to perform most of Stata's functions. With Stata 8 and later versions, you can access most commands through pulldown menus. However, we will focus on the command line throughout the book for several reasons. 1) We think the menu is rather self-explanatory. 2) If you know the commands, you will be able to find the appropriate menu items. 3) The look and feel of the menu depends on the operating system installed on your computer, so using the command line will be more consistent, no matter what system you are using. 4) Switching between the mouse and the keyboard can be tedious. 5) And finally, once you are used to typing the commands, you will be able to write entire analysis jobs, so you can later replicate your work or easily switch across platforms. At first you may find using the command line bothersome, but as soon as your fingers get used to the keyboard, it becomes fun. Believe us, it is habit forming.

1.1 Starting Stata

We assume that Stata is installed on your computer as described in the *Getting Started* manual for your operating system. If you work on a PC using the Windows operating system, you can start Stata by selecting **Start > All Programs > Stata 12**. On a Mac system, you start Stata by double-clicking on the Stata symbol. Unix users type the command **xstata** in a shell.

After starting Stata, you should see the default Stata windowing: a Results window; a Command window, which contains the command line; a Review window; and a Variables window, which shows the variable names.

1.2 Setting up your screen

Instead of explaining the different windows right away, we will show you how to change the default windowing. In this chapter, we will focus on the Results window and the command line. You may want to choose another font for the Results window so that it is easier to read. Right-click within the Results window. In the pop-up menu, choose **Font...** and then the font you prefer.¹ If you choose the suggested font size, the Results window may not be large enough to display all the text. You can resize the Results window by dragging the borders of the window with the mouse pointer until you can see the entire text again. If you cannot do this because the Stata background window is too small, you must resize the Stata background window before you can resize the Results window.

Make sure that the Command window is still visible. If necessary, move the Command window to the lower edge of the Stata window. To move a window, left-click on the title of the window and hold down the mouse button as you drag the window to where you want it. Beginners may find it helpful to dock the Command window by double-clicking on the background window. Stata for Windows has many options for manipulating the window layout; see [GS] **2 The Stata user interface** for more details.

Your own windowing layout will be saved as the default when you exit Stata. You can restore the initial windowing layout by selecting **Edit > Preferences > Load Preference Set > Widescreen Layout (default)**. You can have multiple sets of saved preferences; see [GS] **17 Setting font and window preferences**.

1.3 Your first analysis

1.3.1 Inputting commands

Now we can begin. Type the letter **d** in the command line, and press *Enter* or *Return*. You should see the following text in the Results window:

```
. d
Contains data
obs:          0
vars:          0
size:          0
Sorted by:
```

You have now typed your first Stata command. The letter **d** is an abbreviation for the command **describe**, which describes data files in detail. Because you are not yet working with a data file, the result is not very exciting. However, you can see that entering a command in Stata means that you type a letter or several letters (or words) and press *Enter*.

1. In Mac OS X, right-click on the window you want to work with, and from **Font Size**, select the font size you prefer. In Unix, right-click on the window you want to work with, and select **Preferences...** to display a dialog allowing you to choose the fonts for the various windows.

Throughout the book, every time you see a word in **this font** preceded by a period, you should type the word in the command line and press *Enter*. You type the word without the preceding period, and you must preserve uppercase and lowercase letters because Stata is case sensitive. In the example below, you type **describe** in the command line:

```
. describe
```

1.3.2 Files and the working memory

The output of the above **describe** command is more interesting than it seems. In general, **describe** provides information about the number of variables and number of observations in your dataset.² Because we did not load a dataset, **describe** shows zero variables (vars) and observations (obs).

describe also indicates the size of the dataset in bytes. Unlike many other statistical software packages, Stata loads the entire data file into the working memory of your computer. Most of the working memory is reserved for data, and some parts of the program are loaded only as needed. This system ensures quick access to the data and is one reason why Stata is much faster than many other conventional statistical packages.

The working memory of your computer gives a physical limit to the size of the dataset with which you can work. Thus you might have to install more memory to load a really big data file. But given the usual hardware configurations today, problems with the size of the data file are rare.

Besides buying new memory, there are a few other things you can do if your computer is running out of memory. We will explain what you can do in section 11.6.

1.3.3 Loading data

Let us load a dataset. To make things easier in the long run, change to the directory where the data file is stored. In what follows, we assume that you have copied our datasets into **c:\data\kk3**.

To change to another directory, use the command **cd**, which stands for “change directory”, followed by the name of the directory to which you want to change. You can enclose the directory name in double quotes if you want; however, if the directory name contains blanks (spaces), you *must* enclose the name in double quotes. To move to the proposed data directory, type

```
. cd "c:\data\kk3"
```

With Mac, you can use colons instead of slashes. Alternatively, slashes can be used on all operating systems.

2. You will find more about the terms “variable” and “observation” on pages 5–6.

Depending on your current working directory and operating system, there may be easier ways to change to another directory (see [D] **cd** for details). You will also find more information about folder names in section 3.1.8 on page 56.

Check that your current working directory is the one where you stored the data files by typing **dir**, which shows a list of files that are stored in the current folder:

```
. dir
<dir> 1/24/12 19:22 .
<dir> 1/24/12 19:22 ..
0.9k 1/24/12 12:54 an1cmdkk.do
0.3k 1/24/12 12:54 an1kk.do
1.9k 1/24/12 12:54 an2kk.do
0.4k 1/24/12 12:54 an2_0kk.do
1.5k 1/24/12 12:54 analwe.dta
1.5k 1/19/12 17:10 anbeta.do
0.8k 1/17/12 12:50 anincome.do
2.4k 1/17/12 12:50 ansamples.do
2.1k 1/24/12 12:54 anscombe.dta
(output omitted)
```

Depending on your operating system, the output may look slightly different. You will not see the line indicating that some of the output is omitted. We use this line throughout the book to save space.

In displaying results, Stata pauses when the Results window fills, and it displays **—more—** on the last line if there are more results to display. You can display the next line of results by pressing *Enter* or the next page of results by pressing any other key except the letter *q*. You can use the scroll bar at the side of the Results window to go back and forth between pages of results.

When you typed **dir**, you should have seen a file called **data1.dta** among those listed. If there are a lot of files in the directory, it may be hard to find a particular file. To reduce the number of files displayed at a time, you can type

```
. dir *.dta
```

to display only those files whose names end in **.dta**. You can also display only the desired file by typing **dir data1.dta**. Once you know that your current working directory is set to the correct directory, you can load the file **data1.dta** by typing

```
. use data1
```

The command **use** loads Stata files into working memory. The syntax is straightforward: Type **use** and the name of the file you want to use. If you do not type a file extension after the filename, Stata assumes the extension **.dta**.

For more information about loading data, see chapter 11. That chapter may be of interest if your data are not in a Stata file format. Some general hints about filenames are given in section 3.1.8.

1.3.4 Variables and observations

Once you load the data file, you can look at its contents by typing

```
. describe
Contains data from data1.dta
obs:      5,411                               SOEP 2009 (Kohler/Kreuter)
vars:      65                                  13 Feb 2012 17:08
size:    568,155

variable   storage  display       value
name      type     format      label
                                variable label

persnr    long     %12.0g      Never changing person ID
hhnr2009  long     %12.0g      * Current household number
state     byte     %22.0g      state      * State of Residence
ybirth    int      %8.0g       * Year of birth
sex       byte     %20.0g      sex        Gender
mar       byte     %29.0g      mar        * Marital Status of Individual
edu       byte     %28.0g      edu        * Education
yedu     float    %9.0g       * Number of Years of Education
voc       byte     %40.0g      voc        Vocational trainig/university
emp       byte     %44.0g      emp        * Status of Employment
egp       byte     %45.0g      egp        * Social Class (EGP)
income    long     %10.0g      * Individual Labor Earnings
hhinc    long     %10.0g      * HH Post-Government Income
hysize   byte     %8.0g       * Number of Persons in HH
hysize0to14 byte    %8.0g       * Number of hh members age 0-14
rel2head  byte     %20.0g      rel2head   * Relationship to HH Head
ymove    int      %8.0g       * Year moved into dwelling
ybuild   byte     %21.0g      ybuild    * Year house was build
condit   byte     %24.0g      condit    * Condition of house
dsat     byte     %45.0g      scale11   * Satisfaction with dwelling
size     int      %12.0g      * Size of housing unit in ft.^2
seval    byte     %20.0g      seval     * Adequacy of living space in
                                         housing unit
rooms    byte     %8.0g       * Number of rooms larger than 65
                                         ft.^2
renttype  byte     %20.0g      renttype  * Status of living
rent     int      %12.0g      * Rent minus heating costs in USD
reval    byte     %20.0g      reval     * Adequacy of rent
eqpheaa byte     %20.0g      scale2    * Dwelling has central floor head
eqpter   byte     %20.0g      scale2    * Dwelling has balcony/terrace
eqpbas   byte     %20.0g      scale2    * Dwelling has basement
eqpgar   byte     %20.0g      scale2    * Dwelling has garden

(output omitted)
```

The data file `data1.dta` is a subset of the year 2009 German Socio-Economic Panel (GSOEP), a longitudinal study of a sample of private households in Germany. The same households, individuals, and families have been surveyed yearly since 1984 (GSOEP-West). To protect data privacy, the file used here contains only information on a random subsample of all GSOEP respondents, with minor random changes of some information. The data file includes 5,411 respondents, called observations (obs). For each respondent, different information is stored in 65 variables (vars), most of which contain the respondent's answers to questions from the GSOEP survey questionnaire.

Throughout the book, we use the terms “respondent” and “observations” interchangeably to refer to units for which information has been collected. A detailed explanation of these and other terms is given in section 11.1.

Below the second solid line in the output is a description of the variables. The first variable is `persnr`, which, unlike most of the others, does not contain survey data. It is a unique identification number for each person. The remaining variables include information about the household to which the respondent belongs, the state in which he or she lives, the respondent’s year of birth, and so on. To get an overview of the names and contents of the remaining variables, scroll down within the Results window (remember, you can view the next line by pressing *Enter* and the next page by pressing any other key except the letter *q*).

To begin with, we want to focus on a subset of variables. For now, we are less interested in the information about housing than we are about information on respondents’ incomes and employment situations. Therefore, we want to remove from the working dataset all variables in the list from the variable recording the year the respondent moved into the current place (`ymove`) to the very last variable holding the respondents’ cross-sectional weights (`xweights`):

```
. drop ymove-xweights
```

1.3.5 Looking at data

Using the command `list`, we get a closer look at the data. The command lists all the contents of the data file. You can look at each observation by typing

```
. list
```

1.	persnr 8501	hhnr2009 85	state N-Rhein-Westfa.	ybirth 1932	sex Male	mar Married
	edu Elementary			yedu 10		
	voc Vocational training			emp not employed		
	egp Retired		income . .	hhinc 22093	hhsiz 2	
	hhsiz~14 0		rel2head Head			

2.	persnr 8502	hhnr2009 85	state N-Rhein-Westfa.	ybirth 1939	sex Female	mar Married
	edu Elementary			yedu 8.7		
	voc Does not apply			emp not employed		
	egp Retired		income . .	hhinc 22093	hhsiz 2	
	hhsiz~14 0		rel2head Partner			

(output omitted)

In a moment, you will see how to make `list` show only certain observations and how to reduce the amount of output. The first observation is a man born in 1932 and from the German state North Rhine-Westphalia; he is married, finished school at the elementary level and had vocational training, and is retired. The second observation is a married woman, born in 1939; because she lives in the same household as the first observation, she presumably is his wife. The periods as the entries for the variable `income` for both persons indicate that there is no information recorded in this variable for the two persons in household 85. There are various possible reasons for this; for example, perhaps the interviewer never asked this particular question to the persons in this household or perhaps they refused to answer it. If a period appears as an entry, Stata calls it a “missing value” or just “missing”.

In Stata, a period or a period followed by any character *a* to *z* indicates a missing value. Later in this chapter, we will show you how to define missings (see page 11). A detailed discussion on handling missing values in Stata is provided in section 5.5, and some more general information can be found on page 413.

1.3.6 Interrupting a command and repeating a command

Not all the observations in this dataset can fit on one screen, so you may have to scroll through many pages to get a feel for the whole dataset. Before you do so, you may want to read this section.

Scrolling through more than 5,000 observations is tedious, so using the `list` command is not very helpful with a large dataset like this. Even with a small dataset, `list` can display too much information to process easily. However, sometimes you can take a glance at the first few observations to get a first impression or to check on the data. In this case, you would probably rather stop listing and avoid scrolling to the last observation. You can stop the printout by pressing *q*, for quit. Anytime you see —*more*— on the screen, pressing *q* will stop listing results.

Rarely will you need the key combination *Ctrl+Break* (Windows), *command+* (Mac), or *Break* (Unix), which is a more general tool to interrupt Stata.

1.3.7 The variable list

Another way to reduce the amount of information displayed by `list` is to specify a variable list. When you append a list of variable names to a command, the command is limited to that list. By typing

```
. list sex income
```

you get information on gender and monthly net income for each observation.

To save some typing, you can access a previously typed `list` command by pressing *Page Up*, or you can click once on the command `list` displayed in the Review window. After the command is displayed again in the command line, simply insert the variable list of interest. Another shortcut is to abbreviate the command itself, in this case by typing the letter *l* (lowercase letter L). A note on abbreviations: Stata commands are usually short. However, several commands can be shortened even more, as we will explain in section 3.1.1. You can also abbreviate variable names; see *Abbreviation rules* in section 3.1.2.

Scrolling through 5,411 observations might not be the best way to learn how the two variables `sex` and `income` are related. For example, we would not be able to judge whether there are more women or men in the lower-income groups.

1.3.8 The `in` qualifier

To get an initial impression of the relationship between gender and income, we might examine the gender of the 10 respondents who earn the least. It would be reasonable to first sort the data on the variable `income` and then list the first 10 observations. We can list only the first 10 observations by using the `in` qualifier:

```
. sort income
. list sex income in 1/10
```

	sex	income
1.	Male	0
2.	Male	0
3.	Male	0
4.	Female	0
5.	Male	0
6.	Female	0
7.	Male	0
8.	Female	0
9.	Female	0
10.	Male	0

The `in` qualifier allows you to restrict the `list` command or almost any other Stata command to certain observations. You can write an `in` qualifier after the variable list or, if there is no variable list, after the command itself. You can use the `in` qualifier to restrict the command to data that occupy a specific position within the dataset. For example, you can obtain the values of all variables for the first observation by typing

```
. list in 1
```

and you can obtain the values for the second to the fourth observations by typing

```
. list in 2/4
```

The current sort order is crucial for determining each observation's position in the dataset. You can change the sort order by using the `sort` command. We sorted by `income`, so the person with the lowest income is found at the first position in the dataset. Observations with the same value (in this case, `income`) are sorted randomly. However, you could sort by `sex` among persons with the same income by using two variable names in the `sort` command, for example, `sort income sex`. Further information regarding the `in` qualifier can be found in section 3.1.4.

1.3.9 Summary statistics

Researchers are not usually interested in the specific answers of each respondent for a certain variable. In our example, looking at every value for the `income` variable did not provide much insight. Instead, most researchers will want to reduce the amount of information and use graphs or summary statistics to describe the content of a variable.

Probably the best-known summary statistic is the arithmetic mean, which you can obtain using the `summarize` command. The syntax of `summarize` follows the same principles as the `list` command and most other Stata commands: the command itself is followed by a list of the variables that the command should apply to.

You can obtain summary statistics for income by typing

<code>. summarize income</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
income	4779	20540.6	37422.49	0	897756

This table contains the arithmetic mean (`Mean`) as well as information on the number of observations (`Obs`) used for this computation, the standard deviation (`Std. Dev.`) of the variable `income`, and the smallest (`Min`) and largest (`Max`) values of income in the dataset.

As you can see, only 4,779 of the 5,411 observations were used to compute the mean because there is no information on income available for the other 632 respondents—they have a missing value for income. The year 2009 average annual income of those respondents who reported their income is €20,540.60 (approximately \$30,000). The minimum is €0 and the highest reported income in this dataset is €897,756 per year. The standard deviation of income is approximately €37,422.

As with the `list` command, you can `summarize` a list of variables. If you use `summarize` without specifying a variable, summary statistics for all variables in the dataset are displayed:

<code>. summarize</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
persnr	5411	4692186	3096841	8501	1.11e+07
	5411	79260.42	48474.2	85	167012
	5411	7.899649	4.440415	0	16
	5411	1959.493	18.12642	1909	1992
	5411	1.522269	.49955	1	2
hhnr2009	5410	1.718854	1.020349	1	5
	5183	2.382597	1.392508	1	5
	5039	11.80419	2.676028	8.7	18
	4101	2.460619	1.870365	1	6
	5256	3.050038	1.868658	1	5
state	4789	9.29004	6.560561	1	18
	4779	20540.6	37422.49	0	897756
	5407	37149.97	26727.97	583	507369
	5411	2.610238	1.164874	1	5
	5411	.3418961	.70429	0	3
ybirth					
sex					
mar					
edu					
yedu					
voc					
emp					
egp					
income					
hhinc					
hsize					
hhsizeto14					
rel2head	5411	1.577342	.7471922	1	5

In chapter 7, we will discuss further statistical methods and graphical techniques for displaying variables and distributions.

1.3.10 The if qualifier

Assume for a moment that you are interested in possible income inequality between men and women. You can determine if the average income is different for men and for women by using the `if` qualifier. The `if` qualifier allows you to process a command, such as the computation of an average, conditional on the values of another variable. However, to use the `if` qualifier, you need to know that in the `sex` variable, men are coded as 1 and women are coded as 2. How you discover this will be shown on page 110.

If you know the actual values of the categories in which you are interested, you can use the following commands:

<code>. summarize income if sex==1</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
income	2320	28190.75	47868.24	0	897756
<code>. summarize income if sex==2</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
income	2459	13322.89	21286.44	0	612757

You must type a double equal-sign in the `if` qualifier. Typing a single equal-sign within the `if` qualifier is probably the most common reason for the error message “invalid syntax”.

The `if` qualifier restricts a command to those observations where the value of a variable satisfies the `if` condition. Thus you see in the first table the summary statistics for the variable `income` only for those observations that have 1 as a value for `sex` (meaning men). The second table contains the mean income for all observations that have 2 stored in the variable `sex` (meaning women). As you can see now, the average income of women is much lower than the average income of men: €13,322.89 compared with €28,190.75.

Most Stata commands can be combined with an `if` qualifier. As with the `in` qualifier, the `if` qualifier must appear after the command and after the variable list, if there is one. When you are using an `in` qualifier with an `if` qualifier, the order in which they are listed in the command line does not matter.

Sometimes you may end up with very complicated `if` qualifiers, especially when you are using logical expressions such as “and” or “or”. We will discuss these in section 3.1.5.

1.3.11 Defining missing values

As you have seen in the table above, men earn on average substantially more than women: €28,191 compared with €13,323. However, we have seen that some respondents have a personal income of zero, and you might argue that we should compare only those people who actually have a personal income. To achieve this goal, you can expand the `if` qualifier, for example, by using a logical “and” (see section 3.1.5).

Another way to exclude persons without incomes is to change the content of `income`. That is, you change the `income` variable so that all incomes of zero are recorded as a missing value, here stored with the missing-value code `.c`. This change automatically omits these cases from the computation. To do this, use the command `mvdecode`:

```
. mvdecode income, mv(0=.c)
      income: 1369 missing values generated
```

This command will exclude the value zero in the variable `income` from future analysis.

There is much more to be said about encoding and decoding missing values. In section 5.5, you will learn how to reverse the command you just entered and how you can specify different types of missing values. For general information about using missing values, see page 413 in chapter 11.

1.3.12 The `by` prefix

Now let us see how you can use the `by` prefix to obtain the last table with a single command. A prefix is a command that precedes the main Stata command, separated from it by a colon. The command prefix `by` has two parts: the command itself and a variable list. We call the variable list that appears within the `by` prefix the *bylist*. When you include the `by` prefix, the original Stata command is repeated for all categories of the variables in the *bylist*. The dataset must be sorted by the variables in the *bylist*. Here is one example in which the *bylist* contains only the variable `sex`:

```
. sort sex
. by sex: summarize income

```

		Obs	Mean	Std. Dev.	Min	Max
<i>-> sex = Male</i>						
Variable		1746	37458.5	51939.73	46	897756
<i>-> sex = Female</i>						
Variable		1664	19688.09	23330.87	163	612757

The output above is essentially the same as that on page 11, although the values have changed slightly because we changed the `income` variable using the `mvdecode` command. The `by` prefix changed only the table captions. However, compared with the `if` qualifier, the `by` prefix offers some advantages. The most important is that you do not have to know the values of each category. When you use `by`, you need not know whether the different genders are coded with 1 and 2 or with 0 and 1, for example.³ The `by` prefix saves typing time, especially when the grouping variable has more than two categories or when you use more than one grouping variable. The `by` prefix allows you to use

3. You can learn more about coding variables on page 413.

several variables in the *bylist*. If the *bylist* contains more than one variable, the Stata command is repeated for all possible combinations of the categories of all variables in the *bylist*.

The **by** prefix is one of the most useful features of Stata. Even advanced users of other statistical software packages will be pleasantly surprised by its usefulness, especially when used in combination with commands to generate or change variables. For more on this topic, see sections 3.2.1 and 5.1.4.

1.3.13 Command options

Let us go back to exploring income inequality between genders. You might argue that using the arithmetic mean of **income**, even when combined with its minimum and maximum, is an inadequate way to compare the two subgroups. These values are not sufficient to describe the income distribution, especially if, as you may suspect, the distribution is positively skewed (skewed to the right). You can obtain more statistics from the **summarize** command by specifying options. Options are available for almost all Stata commands.

In contrast to the prefix commands and qualifiers discussed so far, options are command specific. For most commands, a certain set of options is available with a command-specific meaning. You specify options at the end of the command, after a comma.

The **summarize** command has only a few options. An important one is **detail**. Specifying this option returns several types of percentiles, among them the median (the 50th percentile) and the first and third quartiles (the 25th and 75th percentiles, respectively); the already known mean, minimum, and maximum; and the second through fourth “moments”: the variance, skewness, and kurtosis.

Individual Labor Earnings				
	Percentiles	Smallest		
1%	548	46		
5%	1852	90		
10%	3864	134	Obs	3410
25%	10040	163	Sum of Wgt.	3410
50%	22841		Mean	28786.96
		Largest	Std. Dev.	41537.7
75%	36930	710920		
90%	53251.5	749421	Variance	1.73e+09
95%	69806	869446	Skewness	11.27167
99%	107981	897756	Kurtosis	188.2845

The **detail** option does the same thing, even if you use **if** and **in** qualifiers and the **by** prefix command. You can add any list of variables or any **if** or **in** qualifiers, as well as any prefix; the function of the option is always the same. You can check this yourself by typing

```
. by sex: summarize income if edu==4, detail
```

After entering this command, you will obtain the income difference between men and women for all respondents who at least have the *Abitur* (which is the German qualification for university entrance). Interestingly enough, the income inequality between men and women remains, even if we restrict the analysis to the more highly educated respondents.

More general information about options can be found in section 3.1.3.

1.3.14 Frequency tables

In addition to simple descriptive statistics, frequencies and cross-classified tables (univariate and bivariate frequency tables) are some of the most common tools used for beginning a data analysis. The Stata command for generating frequency tables is **tabulate**. This command must include a variable list, consisting of one or two variables. If you use one variable, you get a one-way frequency table of the variable specified in the command:

<code>. tabulate sex</code>				
Gender	Freq.	Percent	Cum.	
Male	2,585	47.77	47.77	
Female	2,826	52.23	100.00	
Total	5,411	100.00		

If you specify two variables, you get a two-way frequency table:

<code>. tabulate emp sex</code>				
Status of Employment	Gender		Total	
	Male	Female		
full time	1,346	695	2,041	
part time	59	540	599	
irregular	70	218	288	
not employed	1,014	1,314	2,328	
Total	2,489	2,767	5,256	

The first variable entered in the variable list of the **tabulate** command forms the row variable of the cross-classified table, and the second variable forms the column variable. Absolute frequencies are written in the table cells, the contents of which you can change using appropriate options. The crucial options for this command are **row** and **column**, which return row and column percentages. Other **tabulate** command options return information about the strength of the relationship between the two variables. As we explained for the **summarize** command, you can use options with **if** or **in** qualifiers, as well as command prefixes.

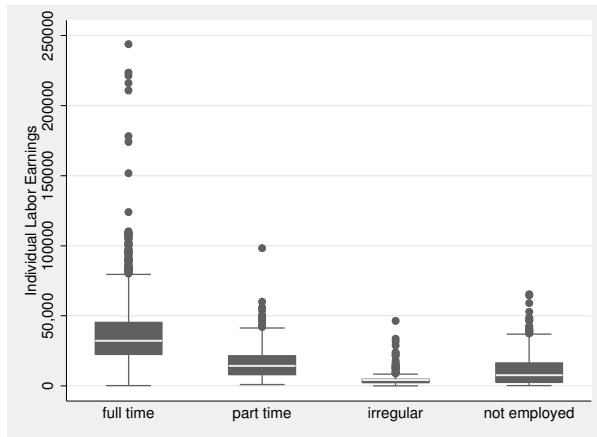
For more information about `tabulate`, see section 7.2.1. See below for an example of column percentages in a cross-classified table.

Status of Employment	Gender		Total
	Male	Female	
full time	54.08	25.12	38.83
part time	2.37	19.52	11.40
irregular	2.81	7.88	5.48
not employed	40.74	47.49	44.29
Total	100.00	100.00	100.00

1.3.15 Graphs

Graphs provide a quick and informative way to look at data, especially distributions. Comparative box-and-whisker plots are a nice way to compare a distribution of one variable, such as `income`, across different subgroups (in this case, employment status):

```
. graph box income if income <= 250000, over(emp)
```



This graph command is composed of the `graph box` command, the variable `income`, the `if` qualifier, and the option `over(emp)`, which specifies the grouping variable. (See chapter 6 for a discussion of other possible options.)

Looking at the four box plots shown in the Graph window, you can see that income is right-skewed for all subgroups, with many outliers. The median (indicated by the middle line in the box) for the full-time employees is, as we assumed, higher than that for all the other groups. If there are relatively more part-time working women represented in the dataset, the median gross income must be smaller. Therefore, we might observe that income inequality is due to the division of labor within couples rather than due to

To display the online help system, type `help` at the command line. You do not need Internet access to use the online help—all the necessary information is already stored on your machine. The online help contains help text for every Stata command. The help text will be displayed on the screen, and you can scan the pages of the help text as usual. To see the help file for a specific command, type `help` followed by the name of the command.

Help entries are all structured the same way. They begin with the structure of the command—the “syntax diagram”—followed by a fairly detailed explanation of the command’s function and in turn by a description of available options. At the end of each help entry, you will find examples showing how to use the command, together with cross-references to related commands.

For example, the description of `regress` tells you that it fits a model of *depvar* on *indepvars* using linear regression. To do this, you must type the command itself followed by the dependent (endogenous) variable and then type the list of independent (exogenous) variables. Below we give an example of linear regression, and we will explain regression in chapter 9. For more information about reading Stata syntax, see chapter 3.

1.3.17 Recoding variables

In our analysis of income inequality, income is the dependent variable while gender and employment status are independent variables. Unfortunately, the two independent variables are problematic:

1. Gender is a dichotomous nominal variable. Conventionally, such variables are included in a linear regression analysis coded 0 and 1 for the two outcomes. However, in our dataset, `sex` is coded as 1 and 2, so we recommend constructing a new variable in the required form:

```
. generate men = 1 if sex== 1  
. replace men = 0 if sex== 2
```

This set of commands says, “Generate the variable `men` with the value 1 if the value of the variable `sex` is equal to 1, or leave it as missing otherwise. Then replace the values in the variable `men` with 0 if the value of the variable `sex` is equal to 2.”

2. Employment status is a nominal variable. But because it is not dichotomous, we cannot use its values the way they appear in the data file. Fortunately, Stata’s regression command has a way to deal with nominal variables. However, we might want to limit our analysis of income to persons who were employed at the time of the survey. Therefore, we create a new variable `emp3` with valid values for full-time, part-time, and irregular employment and missing otherwise:

```
. generate emp3 = emp if emp <= 2  
. replace emp3 = 3 if emp == 4
```

You can check the results of these last commands by looking at a cross-tabulation of `emp` and `emp3` including the missing values:

Status of Employment	emp3				Total
	1	2	3	.	
full time	2,041	0	0	0	2,041
part time	0	599	0	0	599
irregular	0	0	288	0	288
not employed	0	0	0	2,328	2,328
.	0	0	0	155	155
Total	2,041	599	288	2,483	5,411

As you can see, full-time employed persons have value 1 on the new variable, part-time employed have value 2, and irregularly employed persons have value 3. All other respondents were assigned to the missing value.

You will find more examples and a detailed description of the commands for creating and changing variables in chapter 5. Recoding variables is probably the most time-consuming part of data analysis, so we recommend that you spend some time learning these commands.

1.3.18 Variable labels and value labels

Looking at the output of variable `emp3` just created, you realize that the rows of the table show the numerals 1 to 3. Without knowing what these numbers mean, you will not know how to interpret them. Assigning labels to the variables and their values—and even to the entire dataset—can help make their meaning clear. In Stata, you assign labels by using the `label` command.

See section 5.6 for a detailed explanation of the `label` command. Here we will simply show you how the `label` command works generally. First, let's create a label for the variable `emp3` holding "Status of employment (3 categories)":

```
. label variable emp3 "Status of employment (3 categories)"
```

The command is the same whether or not a label was previously assigned to the variable. To assign value labels to the values of `emp3`, we type in the command below.

```
. label define emp3 1 "full time" 2 "part time" 3 "irregular"
. label values emp3 emp3
```

You may want to reenter the `tabulate` command used above so that you can see the results of the labeling commands. The following command displays column percentages and suppresses the display of frequencies:

. tabulate emp3			
Status of employment (3 categories)	Freq.	Percent	Cum.
full time	2,041	69.71	69.71
part time	599	20.46	90.16
irregular	288	9.84	100.00
Total	2,928	100.00	

1.3.19 Linear regression

Now that you have generated all the variables you want to include in the regression model, the remaining task is simple. The command to compute a linear regression is `regress`. Just type `regress` and, after the command itself, the name of the dependent variable, followed by a list of independent variables. In the list of independent variables, type `i.` in front of nominal scaled variables.

. regress income men i.emp3						
Source	SS	df	MS	Number of obs = 2886		
Model	5.2597e+11	3	1.7532e+11	F(3, 2882) = 112.71		
Residual	4.4830e+12	2882	1.5555e+09	Prob > F = 0.0000		
Total	5.0089e+12	2885	1.7362e+09	R-squared = 0.1050		
				Adj R-squared = 0.1041		
				Root MSE = 39440		
income	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
men	12754.37	1675.482	7.61	0.000	9469.105	16039.63
emp3						
2	-15295.93	2074.688	-7.37	0.000	-19363.95	-11227.9
3	-28981.25	2688.387	-10.78	0.000	-34252.61	-23709.89
_cons	30950.82	1410.999	21.94	0.000	28184.15	33717.49

The average annual income of respondents with values of zero on all independent variables—that is, the full-time working women—is €30,950.82. Their part-time working colleagues earn on average around €15,295.93 less. Irregularly employed women earn on average €28,981.25 less than full-time employed women, and hence only €1,969.57 per year. Independent of the income differences between the three categories of employment, men still earn on average €12,754.37 more than women. Therefore, the income inequality between men and women cannot be explained by the higher proportion of part-time working women in the data file.

The validity of these conclusions depends on many different factors, among which are several statistical aspects that we will address in chapter 9.

1.4 Do-files

Suppose that the regression results computed in the preceding section are interesting enough that we want to keep them. When analyzing data, you want to make sure you can reproduce the results. We will discuss this basic rule of scientific work in chapter 2, but some of its aspects are so important to us that we want to introduce the basic procedure at this point.

To begin, type the command

```
. doedit
```

This command calls the Stata Do-file Editor, which allows you to create and modify Stata do-files and other ASCII text files. You can also use any text editor, such as Notepad, TextPad, UltraEdit, or (X)Emacs.⁴

The Do-file Editor is in the foreground. It is basically an empty sheet in which you can enter any text—including, of course, Stata commands, which is what we want to do now. To begin, type the following lines:

```
begin: an1.do
1: use data1, clear
2:
3: mvdecode income, mv(0=.a)
4:
5: generate men = 1 if sex == 1
6: replace men = 0 if sex == 2
7:
8: generate emp3 = emp if emp <= 2
9: replace emp3 = 3 if emp == 4
10:
11: label variable emp3 "Status of employment (3 categories)"
12: label define emp3 1 "full time" 2 "part time" 3 "irregular"
13: label values emp3 emp3
14:
15: regress income men i.emp3

```

end: an1.do

Be sure that you type only the plain text. Do not type the solid lines, `an1.do`, or the numbers with the colons at the beginning of each line. We have indicated the text to be entered by placing it between the two solid lines. The line numbers help us refer to specific lines in text files. The word `an1.do` shown at the top is the filename under which you should save your text when you finish.

The lines you typed are the commands that you will need to reproduce the regression analysis. The only new thing here is the option `clear` in the `use` command, which is needed because there are already data in memory that need to be replaced by the dataset you want to use.

4. See <http://fmwww.bc.edu/repec/bocode/t/textEditors.html> for a discussion of different text editors.

Now save the text using the filename `an1.do`. Make sure that you save the file in the same directory that you are using while working with Stata. (If you followed our earlier suggestion, this will be `c:\data\kk3`.) To save the file with the Stata Do-file Editor, select **File > Save As...**, or choose its equivalent if you use another text editor. In many Windows editors, you must specify the file type to be saved; be sure to save it as an ASCII text file with a `.do` extension.

Now switch back to Stata by clicking on any Stata window. Another way to switch between an editor and Stata is to use the key combination `Alt+Tab`. (On a Mac keyboard, you use the key combination `command+Tab`.)

As soon as you are back at the Stata command line, type

```
. do an1.do
```

You should see a list of all the Stata commands you entered in the Do-file Editor. The command `do` causes Stata to execute the do-file that contains all the Stata commands that you want to execute, so to run the do-file `an1.do`, you typed `do an1.do`. However, you can omit the extension `.do` and simply type `do an1`; Stata will understand that you are running a do-file (see section 3.1.8). If you are still in the Do-file Editor, you can select **Tools > Execute (do)**.

If you saw an error message, you most likely made a typing error in your do-file. As soon as Stata detects an error, it stops executing the do-file and displays the corresponding error message. In this case, switch back to your editor and carefully check the commands you entered. Correct the error, save the file, and switch back to Stata. Try the do-file again. Remember that you can repeat a command by pressing *Page Up*. It might be a good idea to repeat the sequence of switching to an editor, saving, switching back to Stata, and executing the do-file a few times, to make it a habit.

If you received the error message

```
file an1.do not found  
r(601)
```

you may not have saved `an1.do` in the correct directory. You can check whether the file is in your current working directory by typing

```
. dir an1.do
```

If `an1.do` does not appear in the list, type

```
. pwd
```

to determine the location of your current working directory. Then switch back to your editor and save the file in the correct directory.

1.5 Exiting Stata

Now you are at the end of your first Stata session. Before you quit, save all the commands you typed in this session. You save the commands by right-clicking anywhere in the Review window, selecting **Save All...** from the pulldown menu, and saving the contents to the file **an1cmd.do**. Make sure that this file is stored in the same working directory you have been using during this session! We will come back to this file in the next chapter.

Now you can finish your first Stata session. Type the command **exit**.

```
. exit
no; data in memory would be lost
r(4);
```

Well, that is obviously not the way to do it. The reason is that commands like **generate** and **replace** have changed the data. Exiting Stata without saving the changes in the dataset would cause those changes to be lost forever. To do this, you must explicitly request that Stata exit without saving your changes by using the **clear** option. If you want to save the data, you can use the **save** command. You should specify a filename after **save**, and it is fine to use a new filename, such as **mydata**:

```
. save mydata
```

The file will be saved in the current working directory with the name **mydata.dta**. If you pick a name that is already in use for another file, you will get an error message. Stata tries to ensure that you do not accidentally overwrite data. The only way to overwrite a file is to use the option **replace**, for example, **save mydata, replace**.

In most cases, you should avoid using the command **save, replace**, which will use the name of the file you currently have loaded and cause an irreversible loss of the original version of the data.

Now that you have saved the file, you can exit Stata with **exit**.

You do not really need to save the changes to the data because you have already saved the commands that created the changes. You can reproduce the current state of your work by running the do-file again on the original data. If you do not want to save the file—and we recommend that you not save it (see chapter 2)—you can simply exit Stata by typing

```
. exit, clear
```

1.6 Exercises

1. The command `cd` changes the working directory to the specified drive and directory. Explore the command by changing Stata's working directory to the following directories:

For Windows users

- `c:\Documents and Settings\All Users\Desktop`
- Starting from the desktop, go up one directory in the directory tree.
- `c:\Program Files`
- `c:\data\kk3\`

For Linux users

- `~/Desktop`
- Starting from the desktop, go up one directory in the directory tree.
- `/usr/local`
- `~/data/kk3/`

2. Try the following commands. Write a short description of the task they perform:

- `copy`
- `mkdir`
- `type`
- `winexec`

3. Load `data1.dta` into memory and try the following commands. Again write a short description of the task they perform:

- `codebook`
- `inspect`
- `browse`
- `edit`
- `lookfor`

4. Find the Stata commands to execute the tasks listed below. Give an example with `data1.dta` for each of these commands (you do not need to understand what the commands actually do).

- Delete variables
- Chi-squared test for bivariate tables
- Correlation matrix
- Cronbach's alpha
- Factor analysis
- Kernel density estimation
- Locally weighted scatterplot smoother

2 Working with do-files

Science claims to be objective. A central criterion for objectivity is *intersubjective traceability* (Popper 1994, 18); in other words, other people should be able to confirm the results by using the same methods or to criticize the results on the grounds of problematic details. This requires that you diligently document every decision you make during your research.

Few areas of scientific work are as easy to document as statistical data evaluation, yet every now and then, some students cannot show how they obtained a particular result. This problem is not restricted to students. When trying to reproduce 62 empirical economic studies from the renowned *Journal of Money, Credit, and Banking*, Dewald, Thursby, and Anderson (1986) found that only 22 of the addressed authors provided their data and programs. Twenty of them did not reply, and for 20 others, the data did not exist. Only one of the 22 articles for which the data and programs were available was well documented.¹

Poor organization is likely the cause of the cases of nonexistent data and the 21 analyses that were badly, or not at all, documented. This chapter will show you how to prevent such problems by using do-files. As discussed in section 1.4, do-files are simple text files containing Stata commands that are executed one after the other. Using do-files is the best way to guarantee reproducibility, but you must take care to organize your do-files properly.

2.1 From interactive work to working with a do-file

Even though do-files are important, we usually begin our analyses interactively. We try different models, delete outliers, transform variables, construct indices, and so on. As you work on an analysis, you should, however, try to document the essential steps you take in a do-file so you can reproduce them later. Stata provides two ways to record your steps.

1. Quoted from Diekmann (1998).

2.1.1 Alternative 1

At the end of chapter 1, we asked you to right-click on the Review window and save the review contents to the file `an1cmd.do`. Once you save the review contents to a file, you can produce a do-file from those saved commands. To do this, type the following command:²

```
. doedit an1cmd.do
```

The `doedit` command opens the Stata Do-file Editor. If you specify a filename with the command, that file is opened in the Do-file Editor. If you did not follow our example in chapter 1, you can open the file `an1cmdkk.do`, which you previously downloaded; see the *Preface*. Your file `an1cmd.do` should look something like our `an1cmdkk.do`, the first several lines of which contain

```
begin: an1cmdkk.do
1: d
2: describe
3: cd c:\data\kk3
4: dir
5: dir *.dta
6: use data1
7: describe
8: drop ymove - xweights
9: list
end: an1cmdkk.do
```

Whatever your file looks like, it reproduces the commands you typed while following the examples in chapter 1. It is a list of Stata commands, and hence nearly a complete do-file. But you will want to remove any commands that you do not need. Most obviously, you will want to remove all commands that returned an error. Those are spotted quickly if you scan for commands that were entered two times in a row; the second command usually will then be the correct one.

Some commands are unnecessary in a do-file, such as `describe`, `list`, and variations of `summarize`. The only really essential commands in our example are those that are directly related to the regression analysis, that is, those needed to reproduce the results or those that deliver important extra information. Decide for yourself which commands you want to delete. This is what your file might look like after you have deleted some commands:

2. Remember that your working directory is `c:\data\kk3`. For details, see page 3.

```

begin: an2.do
1: use data1
2: drop ymove - xweights
3: summarize income
4: mvdecode income, mv(0=.a)
5: sort sex
6: by sex: summarize income
7: summarize income, detail
8: generate men = 1 if sex == 1
9: replace men = 0 if sex == 2
10: generate emp3 = emp if emp <= 2
11: replace emp3 = 3 if emp == 4
12: label variable emp3 "Status of employment (3 categories)"
13: label define emp3 1 "full time" 2 "part time" 3 "irregular"
14: label values emp3 emp3
15: regress income men i.emp3

```

This do-file could now be run in Stata without error.

Before we discuss a second way to save your work as a do-file, please save your current file as a do-file. We recommend that you name it `an2.do`. Make sure that you save the file in the directory (for example, `c:\data\kk3`) in which you are working in Stata with this book.

2.1.2 Alternative 2

In the previous section, you preserved your interactive work by saving the contents of the Review window. Another way to preserve your work is to use the `cmdlog` command. We will show you an example of how to use this command, but be aware that the example uses advanced statistical techniques. Do not worry about the statistics; just concentrate on creating the do-file.

Our example extends the analysis from chapter 1, where we found that women generally earn less than men do. A multiple regression model showed that this inequality is only partly due to the higher rate of part-time employment among women.

You, however, still have doubts about your results. You argue:

- The income of the working population increases with age. At the same time, women are still more likely than men to give up working in favor of family responsibilities. The group of working women therefore is proportionally younger and therefore earns less than the group of working men.
- The income inequality between men and women is dying out. For many years, women ranked the objective of a career lower than that of starting a family, resulting in generally lower professional ambition and correspondingly lower incomes for working women. The situation is different today, with young women pursuing their careers as ambitiously as men do, so the income inequality is found only among older women.

To verify these hypotheses, you must first determine the necessary steps of the analysis. You can start by doing some interactive trials. Change to the Stata Command window. Before you begin, you should reproduce your original findings. To save some typing, we have made creating the two new variables slightly more elegant. You can find out more on page 80.

```
. use data1, clear
. mvdecode income, mv(0=.a)
. generate men = sex == 1
. generate emp3 = emp if emp!=5
. regress income men i.emp3
```

Now you can begin to turn your hypotheses into analyses. Because you already know that your trials should result in a do-file, you should record your interactive session. Type the following command:

```
. cmdlog using an2.do
file an2.do already exists
r(602)
```

Typing `cmdlog using` instructs Stata to create a file in which all subsequent commands will be recorded. Along with the command, you must specify the name of the file to be created (here `an2.do`). If you do not type the filename extension (`.do`), Stata will use the extension `.txt`.

An error message then informs you that the file `an2.do` already exists, which is true because you saved such a file above (page 27). As always, you cannot lose data in Stata unless you explicitly request to lose them. You can use the `replace` option to overwrite the previous do-file with a new one. However, that would not be good advice in this case. The file `an2.do` contains all your analyses up to now, which you would also like to keep. You need to use a different name, or better still, add the subsequent analysis directly to the previous analyses already saved in `an2.do`. This is what the option `append` is for:

```
. cmdlog using an2.do, append
```

You think the results are biased because young women with low incomes make up much of the group of working women. You should therefore control for age by creating an `age` variable from the year of birth:

```
. generate age = 2009 - ybirth
```

You should center the `age` variable (a continuous variable), that is, subtract the mean from every value; otherwise, the constant in the multiple regression gives you the estimated average income of 0-year-olds, which has no use.

When centering your `age` variable, you should subtract the mean age of those cases for which you run the regression analysis. These are the working persons with valid values for all the variables used in the regression model. There are many ways to do this, which we encourage you to find for yourself. If you like, you can type the command

```
. cmdlog off
```

You can then try out commands, which will not be saved in `an2.do`. When you have found a solution, type

```
. cmdlog on
```

and retype the command you want to use. After you type `cmdlog on`, your entries are once again saved in `an2.do`.

To center `age` based on the observations in the model, we take advantage of the function `missing()` to summarize `age` for those observations that are not missing on the variables used in the regression model (`income` and `emp3`):

<code>. summarize age if !missing(income,emp3)</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
age	2886	44.0395	11.227	18	87

Afterward, we create the new variable `age_c`, in which the mean age (44.0395) is subtracted from the age of every person:³

```
. generate age_c = age - 44.0395
```

For the second hypothesis, you believe that income inequality is not an issue among younger persons. This can be modeled by what is called an “interaction effect”, which we will describe thoroughly in section 9.4.2. The easiest way to create interaction effects in regression models is with factor-variable notation, which we will describe in sections 9.4.1 and 9.4.2. For now, it is enough to type the following command.

<code>. regress income i.emp3 c.men##c.age_c</code>						
Source	SS	df	MS	Number of obs = 2886 F(5, 2880) = 78.38 Prob > F = 0.0000 R-squared = 0.1198 Adj R-squared = 0.1182 Root MSE = 39127		
Model	5.9995e+11	5	1.1999e+11			
Residual	4.4090e+12	2880	1.5309e+09			
Total	5.0089e+12	2885	1.7362e+09			
	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
income						
emp3						
2	-16120.07	2076.971	-7.76	0.000	-20192.57	-12047.57
4	-29188.95	2669.66	-10.93	0.000	-34423.58	-23954.31
men						
age_c	12151.78	1668.855	7.28	0.000	8879.512	15424.05
c.men##c.age_c	190.1975	94.01761	2.02	0.043	5.84891	374.5461
_cons	412.3474	130.3497	3.16	0.002	156.7592	667.9356
	31363.58	1408.624	22.27	0.000	28601.57	34125.6

3. A much more elegant way of centering variables can be found in chapter 4.

Now you can evaluate your hypotheses: The model fits the mean annual income of average-aged, full-time working women as €31,363.58. The coefficient of 190.1975 for the variable `age_c` implies that the mean income of women increases by €190.20 for each 1-year increase in age. For men, the net effect of a 1-year increase in age is equal to the coefficient on `age_c` plus the coefficient on the interaction term `c.men#c.age_c` (412.3474), which equals 602.5449. Hence, the mean annual income of men increases by €602.54 for each 1-year increase in age. Therefore, the income inequality between men and women, which at the mean age amounts to €12,151.78, is higher among the older interviewees and lower among the younger interviewees.

This result also seems interesting enough to be preserved in a reproducible way. Because you have recorded your commands, that is no longer a problem. Close the log file, and change back to the Do-file Editor:

```
. cmdlog close
. doedit an2.do
```

You find the commands you entered last at the end of the file `an2.do`. The commands are already formatted for a do-file. Only faulty commands need to be deleted.

You might need to make some cosmetic changes to your do-file, which we will explain next.

2.2 Designing do-files

The do-file below should be very similar to your `an2.do` except that it will include some modifications: comments, line breaks, and some Stata commands that are useful for do-files. The do-file below also includes line numbers, which are not part of the do-file but are there to orient you when we talk about specific parts of the file. In the following paragraphs, we will walk you through these modifications.

```
begin: an2.do
1: * Income inequality between men and women in Germany (GSOEP-data)
2: * -----
3:
4: version 12
5: set more off
6: capture log close
7: log using an2, replace
8:
9: * Data: Subsample of GSOEP 2009
10: use data1, clear // -> Note 1
11: drop ymove - xweights
12:
13: * Descriptive statistic of income
14: summarize income
15: mvdecode income, mv(0=.a) // -> Note 2
16: sort sex
17: by sex: summarize income
18: summarize income, detail
19:
```

```

20: * Employment by sex
21: tabulate emp sex, column norefq           // -> Note 3
22:
23: * Preparation for regression analysis
24: generate men = sex == 1                   // Dummy for gender
25: generate emp3 = emp if emp!=5            // -> Note 4
26: label define emp3 ////
27:     1 "full time" ///
28:     2 "part time" ///
29:     3 "irregular"
30: label values emp3 emp3
31:
32: * Regression analysis I
33: regress income men i.emp3
34:
35: * Preparation for regression analysis II
36: generate age = 2009 - ybirth             // Age
37: summarize age if !missing(income,emp)    // -> Note 5
38: generate age_c = age - r(mean)
39:
40: * Regression analysis II
41: regress income i.emp c.men##c.age_c
42:
43: log close
44: exit
45:
46: Description
47: -----
48:
49: This is an analysis of income inequality between men and women in Germany.
50: Hypotheses: see Kohler/Kreuter (2012, ch. 1-2). The higher amount
51: of part-time working women is not a sufficient explanation for the inequality
52: in average income between men and women. In addition, even though there
53: is a higher income inequality among older people, younger women
54: are still affected.
55:
56: Notes:
57: -----
58:
59: 1) SOEP - Education sample of samples A-D and F without random group 5
60:    (created with crdata1.do).
61: 2) Respondents with zero incomes are excluded from further analysis.
62: 3) Women are more often part-time employed than men. It is reasonable to
63:    control for employment status.
64: 4) This command excludes all respondents who are not employed.
65: 5) Centering the age variable: see Aiken/West (1991).

```

end: an2.do

2.2.1 Comments

One change in `an2.do` is that we included comments to make it easier for you to understand the do-file. Using comments, you can insert titles, explain why an analysis was carried out, or attach keywords to the results.

There are three ways to add comments. 1) You can add a comment line by putting an asterisk (*) at the beginning of the line; see lines 1, 2, and 9 of `an2.do` for an example.

- 2) You can also add comments within a line by adding `//`. Everything that is written on a line after `//` is ignored when running the do-file; see lines 10, 15, and 24 for examples.
- 3) You can use the `/* */` syntax to add comments; Stata ignores everything between `/*` and `*/`, including line breaks, when running a do-file.

2.2.2 Line breaks

Now please take a look at the command `label define` in line 26 of `an2.do`. The command `label define` is an example of a command that can be very long. This is not a problem for Stata; it can deal with such long lines. But avoiding long lines makes your do-file more readable. Moreover, some text editors might split long lines into several lines, causing Stata to produce an error when running your do-file. So it is a good idea to restrict lines to 75–80 characters.

For long commands, you may have to continue the command over several lines. In Stata, commands end with a line break. If you put a line break at the end of a line and write part of a command on the next line, Stata thinks that you have already entered the entire command in the first line and interprets the second line as a new command.

There are two ways to continue commands over more than one line. The first method is to add three slashes, as shown in lines 26, 27, and 28 of our example do-file. You already know that after `//`, everything up to the line break is interpreted as a comment—but the line break itself is interpreted as a line break. With `///`, the line break is part of the comment, so Stata does not notice the line break, and the command continues on the next line.

The second method to continue commands across lines is by using the `#delimit` command. `#delimit` defines the character that indicates the end of a command. You can use either a line break (the default) or a semicolon. You can change delimiters within do-files. For example, you could type

```
#delimit ;
label define emp
  1 "Full-time"
  2 "Part-time"
  3 "Retraining"
  4 "Irregular"
  5 "Unemployed"
  6 "Milit. serv."
  7 "N. working";
#delimit cr
```

Typing `#delimit ;` changes the character marking the end of the command to the semicolon. The `label define` command now spans several lines. Typing `#delimit cr` changes the delimiter back to the line break (carriage return).

Some Stata users always use the semicolon to mark the end of commands in do-files. We do not do this because we often run only extracts of our do-file, which is much more cumbersome to do when using the semicolon as the command delimiter.

2.2.3 Some crucial commands

Some commands are recommended in all do-files. Every do-file should begin with a series of similar commands:⁴

```
begin: an2.do _____  
⌘  
4: version 12  
5: set more off  
6: capture log close  
7: log using an2, replace _____  
end: an2.do _____
```

With **version**, you specify the version of Stata for which the do-file was written. By doing so, you ensure that your do-file will still run without errors in future versions of Stata. When you include the command **version**, Stata works the same way as the specified version for the duration of the do-file. Thus **version** should always be the first command in a do-file.

After the **version** command, the sequence can vary. In our example, we have first deactivated the partitioned display of output. With **set more off**, the do-file runs without interruption. However useful it may be to break up the output into screen pages in interactive sessions, it is useless when you are running a do-file—at least if you are saving the results of the do-file in a file. You do not need to undo this setting at the end of the do-file. The specification is valid *locally*, that is, only for the do-file.

Now you should ensure that the results of your do-file are actually saved in a file. Generally, you will use the **log using** command for this, which is why we call files with Stata results “log files”. (We make sure that no log file is already open with the **capture log close** command, which we explain shortly.) **log using** works very much like **cmdlog using**. It instructs Stata to create a file in which the output of all subsequent commands will be written. If you specify a filename, the log file will have that name; here it is called **an2**. If you do not specify a filename extension, Stata uses the extension **.smcl**. After you enter **log using an2**, everything that appears on the screen is saved in the file **an2.smcl** until you enter **log close**.

Stata stores the results in a format called the Stata Markup and Control Language (SMCL). With SMCL, log files stored on the hard disk display in the same format as output in the Results window. You can view the log files by typing the **view** command, which opens a new window, the Stata Viewer, containing the contents of the specified file. For example, to view the file named **an2.smcl**, you type

```
. view an2.smcl
```

4. The recommendations presented here are taken from the Stata NetCourse 151. For information on the Stata Internet courses, see section 13.1.

SMCL is also useful for translating log files to HTML or L^AT_EX. If you want to insert Stata output into a word processor, such as Microsoft Word, you may want to translate the log file to plain ASCII with **translate**:

```
. translate an2.smcl an2.log
```

You can even save your log files as plain ASCII from the beginning by specifying the filename after **log using** with the extension **.log**.

Now let us return to our example do-file. Here we use **log using** without the extension because we want Stata to save the log file in SMCL. We always use the same filename for the log file as for the do-file; that is, the do-file **an2.do** stores its results in **an2.smcl**. Because the file **an2.smcl** may already exist—maybe created by an earlier version of **an2.do**—you should specify the **replace** option to overwrite the obsolete log file.

The **log using** command does not directly follow the **set more off** command. Instead, we use the **log close** command to ensure that no log file is already open. However, we placed **capture** before the **log close** command. If a log file had not been open when we ran the do-file, entering **log close** would have resulted in an error message—and aborted the do-file. So we included the **capture** command before the **log close** command. **capture** forces Stata to ignore the error message and continue running the do-file.

You can place **capture** before any Stata command to cause Stata to ignore any error messages from that command and continue running the command. In this case, it ensured that the do-file would continue to run if there were no open log file to be closed. Usually, you will not want to **capture** error messages; if a command does not work properly, you will want to know about it. However, using **capture** with **log close** within a do-file is an exception because **log close** leads to an error message only if no log file is open. In this case, you do not need **log close**, and the do-file can simply continue with the next command, here **log using an2, replace**.

If you had omitted **capture log close**, an error message would occur every time you ran this do-file with a log file already open. This can happen if you have previously written some interactively entered commands in a log file and have forgotten to close the log file.

The end of every do-file should contain the following commands:

```
begin: an2.do —————
```

```
*  
43: log close  
44: exit
```

```
————— end: an2.do —————
```

log close, as we explained, is necessary for closing the previously opened log file. **log close** is executed only if the do-file runs without errors. If the do-file does not run completely because of an error, the log file remains open.

`exit` is more interesting. Although chapter 1 presented `exit` as a command for exiting Stata, in do-files `exit` ends the do-file and returns you to the interactive command-entry mode. In fact, typing `exit` in a do-file is not necessary, but there are two good reasons for doing so. First, the last command in a do-file must include a line break. Had we not pressed *Enter* after typing `log close`, the log file would not have been closed, and everything we did after the do-file was finished would also be logged. By including the `exit` command in our do-file, we ensure that the previous command ends in a line break. Second, Stata stops executing do-files as soon as the `exit` command is reached, which allows us to include notes at the end of the file without having to use the comment syntax mentioned previously. Notice our description of the file and implementation notes at the end of our example do-file.

2.3 Organizing your work

Creating concise and easy-to-read do-files is only the first step toward achieving reproducibility of analyses. The second step is a suitable work plan to ensure that

- important files are not lost,
- you can find your do-files for a particular result again without problems,
- all steps are clearly documented, and
- all analyses can be reproduced easily.

To fulfill these objectives, we suggest the following procedure, which you can tailor to your needs, based on the distinction between two types of do-files: creation do-files and analysis do-files.⁵

It is impossible to distinguish completely between these two types of files. In an analysis do-file, the datasets may still need to be edited, but we try to avoid this and do as much editing as possible in the creation do-files. We will never use a creation do-file to carry out an analysis, and we will never save a dataset with an analysis do-file.

This distinction is useful only if it can be inferred from the filenames. For this reason, the names of all of our creation do-files begin with the letters `cr`, whereas the names of analysis do-files begin with the letters `an`.

The results of the analysis do-files are always recorded in a log file, which has the same name as the do-file but with the extension `log` or `smcl`—whichever you prefer. The analysis do-file we created in this chapter is called `an2.do`. It creates the log file `an2.smcl`.

You use creation do-files to create new datasets. This means that we read in a certain dataset, create new variables, recode existing variables, or delete observations

5. We have taken this suggestion from the Stata NetCourse 151. For details on the Stata Internet courses, see section 13.1. Also see Long (2009) for an alternative proposal.

and variables, etc. We then save the newly created dataset as a data file that has the name of the creation do-file without the letters `cr`. For example, we are using for most of our examples in this book a dataset from a creation do-file called `crdata1.do`.⁶ The data file associated with the do-file `crdata1.do` therefore has the name `data1.dta`.

The separation of creation and analysis do-files makes sense only if you are going to use a particular dataset for several analyses. Modifications of a dataset that apply only to one specific analysis are more likely to belong in the analysis do-file. Modifications that apply to several analyses are better stored in a creation do-file because you need to modify the file only once, saving a lot of time in some cases. You need to decide carefully which dataset modifications you want to save in a creation do-file.

Apart from the creation and analysis do-files, our work plan comprises a further type of do-file: the master do-file, or *master file* for short, which contains a list of do commands.

When we begin a new project, we create a do-file with the name `master.do`. In this do-file, we at first write only a title representing the name of our project. Because our master file is a do-file, this title must be marked as a comment (and the file must end with `exit`). The first few entries in `master.do` could, for example, look like this:

```
----- begin: master`example.do -----
1: // Exemplary analyses for Kohler/Kreuter, Data Analysis Using Stata
%<
17: exit
----- end: master`example.do -----
```

Then the actual work begins.

Suppose that we were conducting an analysis to use in a fictitious book. For the first chapter of our book, we wanted to present a small example analysis. We first needed to clarify the nature of the problem, so we did some interactive trials and then finally created a dataset containing data from the social sciences. To create the final version of this dataset, we wrote a do-file called `crdata1.do`. Of course, it took several attempts to create a do-file that ran absolutely error-free. Then we once again accessed the master file, added the line `do crdata1`, and commented this entry:

```
----- begin: master`example.do -----
1: // Exemplary analyses for Kohler/Kreuter, Data Analysis Using Stata
2: do crdata1                                // creation extract of SOEP`09
%<
17: exit
----- end: master`example.do -----
```

After creating `data1.dta` from `crdata1.do`, we carried out a plausibility analysis; that is, we intensively searched for errors in the dataset. Such error checking is important to avoid recoding errors. We decided to document our error checking in a do-file so that we can later reconstruct which error sources we checked (and which we did not). We called this `ancheck1.do`. When we created this do-file, it took several attempts to get

6. For documentation, we have included `crdata1.do` among the files you installed.

it to run error-free. We recorded each attempt in a log file called `ancheck1.smcl`, which therefore always contained the latest results. When we could run the do-file without error messages, we extended our `master.do` file as follows:

```
----- begin: master`example.do -----
1: // Exemplary analyses for Kohler/Kreuter, Data Analysis Using Stata
2: do crdata1                                // creation extract of SOEP`09
3: do ancheck1                                // error checks in data1.dta
%<
17: exit
----- end: master`example.do -----
```

Inspecting the results of our error checking, as recorded in `ancheck1.smcl`, we noticed a small error in the variable for occupational status. We corrected this in a new creation do-file (`crdata1V2.do`), which in turn now creates `data1V2.dta`. After completing `crdata1V2.do`, we again checked for errors and found no more. We did this error check with a do-file (`ancheck1V2.do`), and we included both do-files in the master file and added comments:

```
----- begin: master`example.do -----
1: // Exemplary analyses for Kohler/Kreuter, Data Analysis Using Stata
2: do crdata1                                // creation extract of SOEP`09
3: do ancheck1                                // error checks in data1.dta
4: do crdata1V2                               // correction of errors in data1.dta
5: do ancheck1V2                             // error checks in data1V2.dta
%<
17: exit
----- end: master`example.do -----
```

We then began the actual analyses. The first analysis dealt with the income inequality between women and men and is contained in `an1.do`. We also tested this do-file several times before including the line `do an1` in our file `master.do`.

These examples can be extended further. Whenever you complete an analysis, you can add the corresponding do-file to `master.do`, thus reproducing the sequence of all analyses done in the course of a project. From the comments in `master.do`, you can quickly determine the rough contents of the respective analyses and reconstruct the analysis from the beginning. You can repeat all the analyses from the beginning by running the master do-file, although this will usually not be necessary.

Once you have added a do-file to `master.do`, you should not modify the do-file again. If you discover an error in a do-file listed in the master file, correct the do-file and add it *under a new name* to the end of `master.do`. This applies especially if you have already carried out many analyses using a dataset that was created by a faulty creation do-file. For example, in our example above, we found in the do-file `ancheck1.do` that `data1.dta` contained an error. Instead of correcting the error in `crdata1.do` and running the do-file again, we corrected this error in a second version of that do-file (`crdata1V2.do`). We would do the same if we had already done several analyses between the time we created the dataset and the time we discovered the error. Assume that your master file contained the following sequence:

```
begin: master`example.do
1: // Exemplary analyses for Kohler/Kreuter, Data Analysis Using Stata
2: do crdata1                                // creation extract of SOEP'09
3: do ancheck1                                // error checks in data1.dta
4: do crdata1V2                               // correction of errors in data1.dta
5: do ancheck1V2                            // error checks in data1V2.dta
6:
7: do an1                                     // income inequality men/women
8: do anrent                                  // description of rents
9: do anpi                                    // partisanship by ownership
10:
17: exit
end: master`example.do
```

During your current analysis, you realize that `data1V2.dta` contains a previously undetected error resulting from a faulty recoding in `crdata1V2.do`. Instead of correcting the error in `crdata1V2.do`, you should document the error in a do-file (for example, `anerror.do`), correct it in a further do-file (for example, `crdata1V3.do`), and then repeat the analyses from `an1.do` to `anpi.do`. Because you now refer to the new dataset, `data1V3.dta`, you should save the corresponding modifications in `an1V2.do` to `anpiV2.do` under new filenames and document them accordingly in the master file. The result might then resemble the following:

```
begin: master`example.do
6:
7: do an1                                     // income inequality men/women
8: do anrent                                 // description of rents
9: do anpi                                    // partisanship by ownership
10:
11: // Error in data1V2, -> correction and repetition in an1 - anpi
12: do anerror                                // discovery of error in data1V2.do
13: do crdata1V3                               // correction of errors in data1V2.dta
14: do an1V2                                   // corrected results of an1.do
15: do anrentV2                                // corrected results of anmiete.do
16: do anpiV2                                  // corrected results of anpi.do
17: exit
end: master`example.do
```

If you follow this procedure, you can reproduce the entire process, even the mistake itself, at any time. You might think that this will not be useful, but a long time can pass between the error and its discovery, and you may have cited faulty figures in your writing (even some you have published). With the procedure we have outlined, it will be relatively easy, even after a long time, to detect the cause of erroneous figures. There are few things as frustrating as chasing after the determinants of a figure that no longer correspond to the results of a current analysis.

To conserve space on your hard disk, you may not want to create many permanent datasets. Delete every dataset that you have created with a creation do-file as soon as you no longer need it for your current analysis. Examples are the datasets `data1.dta` and `data1V2.dta` in the above do-file, which both had problems and were no longer needed after we corrected them. You can delete these datasets by using the Stata command `erase`. Be sure to document this in the master file:

```

----- begin: master`example.do -----
*  

5: do ancheck1V2           // error checks in data1V2.dta  

6: erase data1.dta  

7: do an1                  // income inequality men/women
----- end: master`example.do -----

```

If you later want to rerun an older analysis do-file, you may see the error message “file data1V2.dta not found”. You can, however, quickly reproduce the dataset with the corresponding creation do-file.

Likewise, you can also delete log files and then reproduce them accurately later. Finally, you need to make backup copies only of your do-files and the original data file. You can reproduce all the datasets and log files you have created during your project at any time by using the command `do master`.

2.4 Exercises

1. Open the Stata Do-file Editor.
2. Write a do-file that performs the following tasks:
 - a. opens a log file
 - b. opens `data1.dta`
 - c. generates a new variable with the name `satisfaction` that is equal to the sum of the existing variables `lsat`, `hsat`, and `dsat`
 - d. drops all variables except `satisfaction` and `income`
 - e. summarizes `satisfaction` and `income`
 - f. saves the new dataset with the name `2erase.dta`
 - g. closes the log file
3. Run your do-file in the following manner:
 - a. Use the button in the Do-file Editor two times in a row.
 - b. Save the do-file with the name `cr2erase.do` and run it from the Command window.
 - c. Exit and relaunch Stata, and then start the do-file from the command line without opening the Do-file Editor.
 - d. Exit Stata and copy the do-file to another directory. Relaunch Stata and start the do-file from the command line.
4. Think of strategies to create do-files that run under all the above conditions.
5. Open and print out the log file created by `cr2erase.do` using the Stata Viewer.
6. Create a do-file that starts `cr2erase.do`, and run it.

3 The grammar of Stata

3.1 The elements of Stata commands

As you have seen, Stata commands are made up of specific elements. This section describes these elements of the Stata language and the general rules governing them. These rules apply to all Stata commands, making it easy for you to learn quickly how to use new Stata commands.

Each element of a Stata command can be *required*, *permitted*, or *prohibited*. Obviously, you can use an element of a Stata command only when it is at least permitted. The online help provides information about these elements. For example, if you look at the online help for `summarize`, you will find the following syntax diagram:

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

Here you can find most of the elements of the Stata language. For example, *varlist* is an abbreviation for “variable list”, which is a language element used in almost every Stata command. *if* stands for the language elements “*if* qualifier” and “expression”. You can use any element that is displayed in the syntax diagram. Elements displayed in square brackets are optional; those displayed without them are required.

Now let us examine the elements one by one. We will first describe the command itself and the variable list used in the command. We will then describe options, which further specify the command, and qualifiers, which restrict the command processing to subsets of the data. We will discuss *weight* at the end of the chapter, after we introduce other commonly used commands.

To follow our examples, you will need to use our example dataset:¹

```
. use data1, clear
```

3.1.1 Stata commands

Each Stata command has a name that specifies the task Stata should do. Commands written by StataCorp are called official commands, and those that users write are called user-written commands. You might want to do something in Stata and find that there is no official Stata command for that. You may find that there is a user-written command

1. Be sure that your working directory is `c:\data\kk3` (see page 3).

written by someone else. Chapter 13 of this book shows you how to find and install such additional commands; chapter 12 teaches you how to program user-written commands.

Some official commands may be abbreviated. The shortest possible abbreviation for each command is shown in the syntax diagram of the online help with an underline beneath the smallest part of the command that needs to be typed. You can use every possible abbreviation between the shortest allowed up to the entire command. For example, from the syntax diagram on page 41, you learn that you can use each of the following abbreviations for `summarize`:

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

However, it is not always a good idea to use the shortest possible abbreviation for each command, especially in do-files, because overusing abbreviations can make your do-file hard to read. Table 3.1 shows some frequently used commands and their abbreviations with the shortest possible abbreviation underlined. We also show a recommended abbreviation, but this is not always the shortest possible abbreviation and sometimes is not an abbreviation at all. We have followed international usage drawn from postings to Statalist (see page 469), but these recommendations are only our opinion.

Table 3.1. Abbreviations of frequently used commands

Command	Recommended abbreviation	Usage
<u>d</u> escribe	d	Describe data in memory
<u>g</u> enerate	gen	Create new variables
<u>g</u> raph	graph	Graph data
<u>h</u> elp	h	Call online help
<u>l</u> ist	l	List data
<u>r</u> egress	reg	Linear regression
<u>s</u> ummarize	sum	Means, etc.
<u>s</u> ave	save	Save data in memory
<u>s</u> ort	sort	Sort data
<u>t</u> abulate	tab	Tables of frequencies
<u>u</u> se	use	Load data into memory

For ease of reading and following of examples, we try not to use abbreviations in this book.

3.1.2 The variable list

In most commands, you can use a variable list. Within the online help, this is indicated by the term *varlist*. A variable list is a list of variable names separated by spaces.

List of variables: Required or optional

Some commands allow a variable list but do not require one, whereas others require a variable list. You can learn whether a command requires a variable list by looking at the online help. If the term *varlist* appears in square brackets, the variable list is optional. If *varlist* appears without square brackets, it is required.

Many commands that do not require a variable list will use all existing variables unless you provide a list. For example, specifying `summarize` without a variable list returns the means and standard deviations of all the variables in the dataset. Other commands redisplay their previous results if you do not provide a list of variables (for example, `regress`).

Some commands require a list of variables. This is the case if you cannot repeat the command, or if it is not possible or not useful to apply the command to all variables. If you want to force Stata to apply these commands to all variables, you can specify `_all` instead of a variable list. The command `drop`, for example, deletes specified variables from the dataset. Thus the command

```
. drop ymove ybuild
```

deletes the variables `ymove` and `ybuild`. You cannot reverse this command. Once you drop a variable, the only way to get the variable back is to reload the data file. Therefore, using `drop` without a *varlist* will not apply the `drop` command to all variables, because this would mean dropping the entire dataset. If you really wanted to delete all the variables, you would use `drop _all`.²

Abbreviation rules

You can abbreviate the variable names in a list of variables to as few characters for a variable as you need to identify it uniquely. For example, the variable `condit` in `data1.dta` is uniquely identified by using the character `c`; no other variable in this dataset begins with `c`. The variable `ybirth`, on the other hand, cannot be distinguished by its first character from the variables `ymove`, `ybuild`, and `yedu`. You must type at least `ybi` to let Stata know you are referring to `ybirth`.

2. If you have typed `drop _all` right now, be sure to reload the file before you continue reading.

In addition, you can use the tilde (~) to omit one or more characters of a variable name. For example, you can type

```
. summarize y~h
```

to summarize `ybirth`, because there is only one variable that begins with `y` and ends with `h`.

Both types of abbreviations must match a single variable. If the abbreviation does not uniquely identify a variable, Stata displays an error message. To specify more than one variable, you must do so explicitly. This leads us to the second type of shortcut for variable lists: specifying more than one variable at once. There are three ways to do so:

1. You can use a question mark to specify variables that have the same names except for one character. For example,

```
. summarize pi?
```

summarizes all variables having names beginning with `pi` followed by one character. Question marks may be used anywhere in the variable name—at the beginning, in the middle, or at the end.

2. You can use the asterisk (*) as a wildcard character to specify variables that share parts of their names but differ in one or more characters. For example,

```
. summarize wor* e*
```

summarizes all variables that begin with `wor` or with `e`, regardless of how many characters follow. Just like question marks, wildcards may be used anywhere in a variable name.

3. You can use a hyphen to specify a range of variables that come one after another in the dataset. The order of the variables in the dataset can be found in the output of the `describe` command. From the output of

```
. describe
```

you can see that there are some variables describing the dwellings in the dataset. The first one is `condit` and the last one is `eqpnrj`. To summarize those variables, you can type

```
. summarize condit-eqpnrj
```

It is easy to overuse shortcuts for lists of variables. Instead of typing

```
. summarize ybirth voc emp egp
```

you could also have typed

```
. su y~h v~eg
```

This is nice if you are working interactively. However, if you need to preserve your work for later, the first version is easier to decipher. We recommend not using abbreviations for variable names in do-files.

Special listings

For some commands, you will see terms like *varname* or *depvar* in the syntax diagram. These terms are used for variable lists that consist of one variable. These single-variable lists are sometimes combined with general variable lists when the order of the variables matters. For example, in the command for linear regression, you must specify the dependent or endogenous variable by putting its name *before* the list of independent variables. Here the order of the variable list matters, so the syntax diagram of the command looks like this:

```
regress depvar [indepvars] ...
```

The term *depvar* stands for the dependent variable, whereas *indepvars* stands for the independent variables. As always, the order of the variables within the *indepvars* does not matter.

For every Stata command that fits a statistical model, you must specify the dependent variable before a list of independent variables.

3.1.3 Options

Options are used to change the default behavior of a command and are provided for almost every Stata command. In the syntax diagram, you can see that options are allowed if you find the word *options* after a comma. Below the syntax diagram, you find a list of the options available for the specific command. The syntax diagram of *summarize*, for example, allows *detail*, *meanonly*, *format*, and *separator()* options:

```
summarize ... [ , options ]
```

options	Description
<hr/>	
Main	
<u>detail</u>	display additional statistics
<u>meanonly</u>	suppress the display; calculate only the mean; programmer's option
<u>format</u>	use variable's display format
<u>separator(#)</u>	draw separator line after every # variables; default is <u>separator(5)</u>

Sometimes options are required. Then you will not find a square bracket before the comma in the syntax diagram.

You specify options by placing a *comma* at the end of the command and then listing the options with spaces between them. The comma begins a list of options, but you should not have more than one comma no matter how many options you have. The order of the options does not matter. Here is an example for the *tabulate* command. In the

`tabulate` syntax diagram (which you can display by typing `help tabulate twoway`), you find among many others the options `column` and `missing`.

`tabulate ... [, options]`

<i>options</i>	Description
Main	
...	
<code>column</code>	report relative frequency within its column of each cell
...	
<code>missing</code>	treat missing values like other values
...	

Using the `column` option with the `tabulate` command will give you the percentages within each column of the table (column percentages) in addition to the frequencies. If you add the `missing` option, the table includes respondents who have missing values on the variables specified in the `tabulate` command. In the example below, 26 respondents who refused to respond to the question and 66 persons for whom we do not know why the information is missing are added to the table. For more information about frequency tables, see section 7.2.1.

```
. tabulate wor06 sex, column missing
```

Key
<i>frequency</i> <i>column percentage</i>

Worried about consequences from climate change	Gender		Total
	Male	Female	
Very concerned	664 25.69	828 29.30	1,492 27.57
Somewhat concerned	1,385 53.58	1,597 56.51	2,982 55.11
Not concerned at all	493 19.07	352 12.46	845 15.62
.	32 1.24	34 1.20	66 1.22
Refusal	11 0.43	15 0.53	26 0.48
Total	2,585 100.00	2,826 100.00	5,411 100.00

Most options can be abbreviated, and the shortest abbreviation is underlined in the online help syntax diagram. Some commands have a long list of options, so you should carefully review the command-specific online help.

3.1.4 The *in* qualifier

The *in* qualifier, like the *if* qualifier discussed in the next section, is a command element that limits the execution of the command to a subset of observations. The *in* qualifier is allowed for every Stata command that displays [*in*] in its syntax diagram. The *in* qualifier is composed of the word *in* and a range of observations specified in terms of their position in the dataset. If you think about the dataset as a huge table with rows and columns, the number of an observation in the dataset refers to the number of a row in that table.

The range can be a single observation or a range from a certain observation (row) to a second observation below the former observation. You type a slash (/) between the first position and the final position in the range.

For example, you can look at the person ID, the gender, and the year of birth of the 10th observation in the dataset by restricting the command *list* to that observation:

```
. list persnr sex ybirth in 10
```

	<i>persnr</i>	<i>sex</i>	<i>ybirth</i>
10.	29101	Female	1954

And you can get the same information for observations 10–14 by using

```
. list persnr sex ybirth in 10/14
```

	<i>persnr</i>	<i>sex</i>	<i>ybirth</i>
10.	29101	Female	1954
11.	29102	Male	1956
12.	29103	Male	1985
13.	34901	Male	1951
14.	34902	Male	1955

There is no other way to specify a range other than those shown in these two examples. You cannot specify lists of observations or combinations of ranges. Commands such as *list sex in 1 15 26* or *list sex in 1/10 16* would result in an error message. However, you can use a minus sign to specify a range:

```
. list persnr sex ybirth in -5/-1
```

	persnr	sex	ybirth
5407.	11104701	Female	1963
5408.	11109701	Male	1934
5409.	11116001	Female	1937
5410.	11119401	Male	1984
5411.	11126401	Female	1938

Here `-1` means the observation on the last line of the dataset, that is, the last observation. Correspondingly, `-5` means the fifth observation from the last, so the command above shows the last five observations of the dataset. If the data have been sorted by year of birth, these will be for the youngest five persons.

When you use the minus sign, do not confuse the order within the range. The fifth-to-last observation precedes the last observation in the dataset, so you need to insert `-5` before `-1`. The command `list persnr sex ybirth in -1/-5` would be invalid. As long as you account for the order of the observations, you can even use combinations of the count order:

```
. list persnr sex ybirth in 5406/-5
```

is valid, because the fifth-to-last observation is 5,407th and therefore is after 5,406th. Instead of `1` and `-1`, you can use `f` and `l` to indicate the first and last observations in the dataset.

3.1.5 The if qualifier

The `if` qualifier is allowed for every Stata command that displays [`if`] in its syntax diagram. Like the `in` qualifier, the `if` qualifier is used to run a command only with data from certain observations. The `if` qualifier is composed of the word `if` and an *expression*. To take full advantage of the `if` qualifier, you need to be familiar with expressions; see section 3.1.6.

A command with an `if` qualifier uses only those observations for which the expression is true, or more precisely, for which the expression is not zero.

In chapter 1, we showed a simple example using the `if` qualifier:

```
. summarize income if sex == 1
```

Variable	Obs	Mean	Std. Dev.	Min	Max
income	2320	28190.75	47868.24	0	897756

In this example, the `if` qualifier is `if sex==1`. Note the double equal-sign. With this `if` qualifier, we tell Stata to evaluate the expression `sex==1` for each observation in the data and to use only observations for which the expression is true (not zero).

We have been speaking generally, but let us look at this example in more detail. First, let us again look at the income and gender of observations 10–14 of our dataset:

```
. list income sex in 10/14, nolabel
```

	income	sex
10.	31902	2
11.	50391	1
12.	4807	1
13.	34805	1
14.	46664	1

In the first row of the output, we find an observation with an income of €31,902 and a gender of 2. We wonder what the expression `sex==1` evaluates to for this observation. We would probably say that the expression is false for this observation, but in Stata-talk, the expression evaluates to 0 for this observation. For the second observation, you would say that the expression `sex==1` is true, whereas Stata says that the expression evaluates to 1.

If you add an `if` qualifier to a command, Stata evaluates the expression for every observation in the dataset the same way we just did. Stata then executes the command for all the observations for which the expression is nonzero.

Because the `if` qualifier consists of the word `if` and an expression, the general rules for expressions apply (see section 3.1.6). In practice, however, you will most often build expressions that can be true (1) or false (0). Such expressions usually will contain relational operators. Here are just three examples. First, the command

```
. summarize income if ybirth < 1979
```

shows the mean income for respondents who were born before 1979. All of these observations have a value for year of birth that is smaller than 1979. We can extend the first command to

```
. summarize income if ybirth <= 1979
```

showing the mean income for respondents born in 1979 and before. The command

```
. summarize income if ybirth != 1979
```

shows the mean income of all respondents except those born in 1979 (which would also include those where `ybirth` was missing, because missing values are also not equal to 1979). Finally, the command

```
. summarize income if ybirth != 1979 & !missing(ybirth)
```

would summarize the income of all of those not born in 1979 only among those who had a nonmissing year of birth.

We wrote above that a command with an `if` qualifier uses only observations for which the expression is not zero. You can take this literally by stating an `if` qualifier like this:

```
. summarize income if income
```

With this command, Stata summarizes income for all observations for which the expression behind `if` is not zero. These are respondents who have a value on income that is not zero. Note carefully that missings are counted as being not zero, too. However, in this case, missings on income are discarded from the analysis because the command in front of the `if`-qualifier discards them anyway.

The way we have coded the last command is very useful in connection with dummy variables, that is, variables that only contain values 0 and 1. In chapter 1, we created such a dummy variable with

```
. generate men = 1 if sex==1
. replace men = 0 if sex==2
```

which allows us to code

```
. summarize income if men
```

to summarize the income of men.³

There is one trap associated with the `if` qualifier that you should be aware of—and you might want to mark this paragraph with the brightest color you have. The difficulty arises in expressions that contain the name of a variable with missing values, which are set to $+\infty$ in Stata. Expressions having the relational operator `>` or `\geq` therefore evaluate to 1 for observations with missing values.

Take this command, for example:

<code>. tabulate edu, missing nolabel</code>			
Education	Freq.	Percent	Cum.
1	1,836	33.93	33.93
2	1,571	29.03	62.96
3	266	4.92	67.88
4	977	18.06	85.94
5	533	9.85	95.79
.a	228	4.21	100.00
Total	5,411	100.00	

3. This would also summarize income for those respondents with a missing value on men if there had been missing values on sex or men in our data.

We see that the variable `edu`—representing education—contains $977 + 533 = 1,510$ nonmissing observations with an education greater than or equal to 4 (whatever that means). There are 228 more observations for which the education is listed as `.a`, which is one of several ways to code that we have no information about the education of those persons. And we can see the dot only because we used the `missing` option with the `tabulate` command.

Now let us summarize the year of birth for those observations having an educational level of 4 or higher:

. summarize ybirth if edu >= 4					
Variable	Obs	Mean	Std. Dev.	Min	Max
ybirth	1738	1965.155	18.41194	1912	1992

Here we get a table based on 1,738 observations instead of 1,510. This is because the 228 observations with missing values are set to $+\infty$ inside Stata, and $+\infty$ clearly is higher than 4.

To summarize the observations without including the respondents with unknown education, we need to exclude them explicitly. We can do this by stating complicated expressions. Now we will explain expressions in more detail.

3.1.6 Expressions

Expressions are allowed or required wherever the syntax diagram of a Stata command displays the term *exp*, so expressions can be used at different places within a Stata command. A short version of the syntax diagram of the `generate` command, for example, looks like this:

```
generate newvar = exp [if] [in]
```

This command requires an expression after the command name. Expressions can also be found at several other places in the Stata language.

Think of an expression as an arithmetic problem, such as $2 + 3$. With Stata, this problem is easy to solve because Stata ships with its own pocket calculator: the `display` command.

```
. display 2+3  
5
```

That was a snap! Just like every pocket calculator, the `display` command calculates expressions and displays the result. And as with most pocket calculators, we can calculate somewhat more complicated expressions by combining operators and functions.

Operators

For an overview of the operators used in expressions, see the online help:

```
. help operators
```

From this list, we see that we can easily calculate the following expressions:

```
. display 2-3
-1
. display 2*3
6
. display 2/3
.66666667
. display 2^3
8
```

We can combine more than one expression, and we can use parentheses to change the order of the calculations:

```
. display 2*3 + 2/3 - 2^3
-1.3333333
. display 2*(3 + 2)/(3 - 2)^3
10
```

Using expressions with logical and relational operators may seem complicated, but it is easy: expressions with relational operators can be true or false, corresponding to 1 or 0, respectively, in Stata.

Stata uses a double equal-sign to test for equality.⁴ That is, the expression `2==3` makes Stata determine if 2 and 3 are equal:

```
. display 2==3
0
. display 2==2
1
```

As you can see, Stata says that the expression `2==3` is 0 because this expression is not true: two does not equal three. The expression `2==2` on the other hand is true and therefore evaluates to 1.

The logical operators are most often used to combine different expressions containing relational operators, just like the expressions `2==3` and `2==2`. Suppose that you wanted to find out if both expressions are true. You would have Stata evaluate if expression 1 and expression 2 are true:

```
. display 2==3 & 2==2
0
```

⁴. A single equal-sign instructs Stata to make two things equal. Single equal-signs cannot be used in expressions.

To find out if at least one of the expressions is true, you would have Stata evaluate if expression 1 or expression 2 is true:

```
. display 2==3 | 2==2
1
```

You can use many operators in an expression. If you build complicated expressions, you should use parentheses to indicate the order in which to perform the calculations—even if they are not strictly needed.

Stata expressions can also deal with *strings* (words and letters) if they are enclosed in quotation marks. Let us give an example by asking Stata the following crucial question:

```
. display ("SPSS"=="Stata")
0
```

Now we know that this is wrong.

Finally, let us illustrate some uses of expressions with operators in *if* qualifiers. To start with, each of the commands

```
. summarize ybirth if edu == 4 | edu == 5
. summarize ybirth if edu >= 4 & edu <= 6
. summarize ybirth if edu >= 4 & !missing(edu)
```

summarize *ybirth* for observations with values on *edu* above 4 without blundering into the trap of including the missing value. The command

```
. summarize yedu if (2009 - ybirth) < 18
```

summarizes *yedu* for all respondents below the age of 18, while

```
. summarize yedu if !((2009 - ybirth) < 18)
```

summarizes *yedu* for all respondents who are *not* below the age of 18. Prefixing an expression with the logical operator for “not” also works for expressions with dummy variables. The command

```
. summarize income if !men
```

summarizes *income* for all respondents who are *not* men (that is, respondents who are women). Again, if our data had missing value on gender, this would also include respondents for whom the variable *men* is missing, because for those, the value of *men* is not equal to 1.

If you have more than just one dummy variable, this approach can be further extended. To illustrate, we create a dummy variable for full-time employment:

```
. generate fulltime = 0 if !missing(emp)
. replace fulltime = 1 if emp == 1
```

We summarize the labor earnings of full-time employed men with

```
. summarize income if men & fulltime
```

Likewise, the average labor earning of full-time employed women and not full-time employed men and women can be obtained with

```
. summarize income if men & !fulltime
. summarize income if !men & fulltime
. summarize income if !men & !fulltime
```

Functions

Other important tools for building expressions are Stata's *functions*, which are rules that assign *one* specific value to each of a range of given values. A simple function can be expressed as $f(x) = 1$. This function assigns the value 1 to every number x . A slightly more complicated function is $f(x) = \sqrt{x}$, which assigns the square root of x to every number x . Here we will deal only with such nontrivial functions.

Stata has many predefined functions, including the square root, the logarithm, and the exponential. All of these functions consist of a function name followed by an argument within parentheses: *function_name(argument)*.

The function name describes its purpose. `sqrt()` calculates the square root, `ln()` calculates the natural logarithm, and so on.

The argument specifies the values for which the function should be calculated. The argument is itself an expression; that is, you can use simple numbers or complicated expressions of functions and operators inside the parentheses.

Let us try some examples of expressions with functions. The following commands calculate $\sqrt{2}$, $\sqrt{2 + (3/5)}$, and $e^{\sqrt{|-1|}}$, respectively:

```
. display sqrt(2)
1.4142136
. display sqrt(2 + 3/5)
1.6124515
. display exp(sqrt(abs(-1)))
2.7182818
```

You can find out about all of Stata's functions by typing

```
. help functions
```

Depending on your statistical knowledge, you may find this list too large to comprehend. However, if you read on, you will find us introducing more and more of these functions whenever they came in handy. For now, we just want to mention three functions that are often used in the context of `if` qualifiers.

A very helpful function is `missing(a,b,...)` or its shorthand version `mi(a,b,...)`. The function returns 1 if one of its arguments is the missing value and returns 0 otherwise. You can use `missing()` to summarize the income of those observations that are missing on `edu` with

```
. summarize income if missing(edu)
```

The more frequent use, however, is in connection with the “not” operator. The command

```
. summarize income if yedu >= 4 & !missing(edu)
```

is yet another way to circumvent the trap with missing values described on page 3.1.5. If you add further arguments to `missing()`, you can even restrict the execution of a command on those observations who are not missing on a list of variables.

```
. summarize income if !missing(edu,emp)
```

The functions `inlist(z,a,b,...)` and `inrange(z,a,b)` are also often used in `if` qualifiers. The function `inlist(z,a,b,...)` evaluates whether its first argument (`z`) is equal to one of the following arguments (`a, b, ...`). If this is the case, the function returns 1; otherwise it returns 0. The function can be used to restrict the execution of a command on observations that have a set of values on a certain variable. Here we use the command to summarize the income of observations who are married (`mar==1`), widowed (`mar==3`), or divorced (`mar==5`).

```
. summarize income if inlist(mar,1,3,5)
```

Negation of `inlist()` uses all other observations. This is neat, but be aware that the missing value is also neither 1, nor 3, nor 5:

```
. summarize income if !inlist(mar,1,3,5) & !missing(mar)
```

Similarly to `inlist(z,a,b,...)`, the function `inrange(z,a,b)` checks whether the first argument is inside the range specified by the two other arguments. Here we use this function to summarize the income of observations born between 1989 and 1992:

```
. summarize income if inrange(ybirth,1989,1992)
```

3.1.7 Lists of numbers

In the syntax diagrams of some commands, you will find the term *numlist*, which stands for a list of numbers that Stata understands as numbers. Because Stata is quite smart about understanding numbers, you can abbreviate lists of numbers as shown in table 3.2. Examples for useful applications of lists of numbers can be found in sections 3.2.2 and 6.3.4.

Table 3.2. Abbreviations of lists of numbers and their meanings

Input	Meaning
1,2,3,4	1, 2, 3, 4
1 2 3 4	1, 2, 3, 4
1/4	1, 2, 3, 4
2 4 to 8	2, 4, 6, 8
8 6 to 2	8, 6, 4, 2
2 4: 8	2, 4, 6, 8
8 6: 2	8, 6, 4, 2
2(2)8	2, 4, 6, 8
8(-2)2	8, 6, 4, 2
8/10 15 to 30 32 to 36	8, 9, 10, 15, 20, 25, 30, 32, 34, 36
8/10(5)30(2)36	8, 9, 10, 15, 20, 25, 30, 32, 34, 36

3.1.8 Using filenames

In the syntax diagrams of some Stata commands, you will find the element `using filename`, indicating that these are commands that read or write a file. You refer to the file by inserting the term `using` and the complete name of the file. Sometimes you can omit the term `using`.

Generally, a complete filename consists of a directory, a name, and an extension. The directory is where the file can be found. The name is the name of the file itself, and the extension is usually a file type indicator. The file most often used in this book, `c:\data\kk3\data1.dta`, is in the directory `c:\data\kk3`, has the name `data1`, and has the extension `.dta`, which is the extension for Stata datasets.

How you specify file addresses depends on your operating system: you use colons to separate folder names under Mac, slashes under Unix, and backslashes under Windows. Stata also lets you use the slash to separate folder names in *all* operating systems, which is why we generally use it.

You specify the filename the same way in all operating systems: just type the filename. If the filename (or address) contains blanks or other special characters, you will need to place quotation marks around the complete name. For example, to describe the (fictitious) file `AT & T.dta` in the folder `My Data`, you would type `describe using "c:\My Data\AT & T.dta"`.

Whenever you specify a filename, Stata must have the complete name, but this does not mean that you need to type the complete name. Here is what happens if you omit one part or another:

- If you type a filename without the directory, Stata looks in the working (current) directory. You can find the address of the working directory by typing the `pwd` (print working directory) command:

```
. pwd
```

- If you type a filename without the extension, Stata looks for a file with an extension that is appropriate for the specified command. Table 3.3 below shows commonly used commands that look for or save files with the given extension.

Table 3.3. Names of commands and their associated file extensions

Extension	Command
.dta	<code>use; save; append; merge; joinby; describe</code>
.raw	<code>infile (with varlist); infix (with varlist); insheet; outsheets; outfile</code>
.dct	<code>infile (without varlist); infix (without varlist)</code>
.smcl	<code>log (.smcl or .log depending on whether file format is SMCL or ASCII)</code>
.txt	<code>cmdlog</code>
.do	<code>do; run</code>
.gph	<code>graph using; graph, saving()</code>

Finally, the filename does not necessarily refer to a file on your computer but can also mean a file somewhere on the Internet. If your computer has a connection to the Internet, you can load a Stata dataset directly from an Internet location, for example:

```
. use http://www.stata-press.com/data/kk3/data1
```

The same applies for all commands that load a file. However, you cannot write to the web.

3.2 Repeating similar commands

In everyday data analysis, you will often need to retype similar commands over and over again. For example, our data file contains 12 variables that refer to people's concerns about different aspects of their lives. To compare the answers of men and women for these 12 variables, you would repeat a `tabulate` command 12 times:

```
. tabulate wor01 sex
. tabulate wor02 sex
. tabulate wor03 sex
. tabulate wor04 sex
. tabulate wor05 sex
. tabulate wor06 sex
. tabulate wor07 sex
. tabulate wor08 sex
. tabulate wor09 sex
. tabulate wor10 sex
. tabulate wor11 sex
. tabulate wor12 sex
```

Retyping similar commands is boring and, for complex commands, error-prone. To avoid this, you can benefit from the tools we describe here.

Stata has several tools for repeating similar commands. We will discuss three of them: the **by** prefix, the **foreach** loop, and the **forvalues** loop (also see Cox 2002a,b). The simplest one is the **by** prefix, which is used to repeat a single command for different observations. The other two are used to loop over elements of lists or number sequences.

3.2.1 The **by** prefix

The **by** prefix repeats a command for every group of observations for which the values in the variable list are the same. Most Stata commands accept the **by** prefix; this is indicated immediately following the syntax diagram and options table, for example,

by may be used with **summarize**; see [D] **by**.

The **by** prefix consists of the word **by** or **bysort**, a variable list (which we will call the *bylist*), and a colon. Using the **by** prefix generally requires that the dataset be sorted by the variables of the *bylist*, but there are ways to sort the data on the fly.

The best way to understand the **by** prefix is to use it. Let us summarize the income for each group of the variable **sex**. For this, we need to sort the data by gender and then issue **summarize** with the **by** prefix:

```
. sort sex
. by sex: summarize income


---


-> sex = Male
  Variable |       Obs        Mean     Std. Dev.      Min      Max
  income   |    2320    28190.75    47868.24         0    897756


---


-> sex = Female
  Variable |       Obs        Mean     Std. Dev.      Min      Max
  income   |    2459    13322.89    21286.44         0    612757
```

As you can see, the `by` prefix forces Stata to summarize the income for every group or category of the `bylist`. First, Stata summarizes the income of the first group (the men), and then it summarizes the income of the second group (the women).

This works equally well if we use a `bylist` with more than just two groups:

```
. by edu, sort: summarize income
```

We have used the option `sort` of the `by` prefix to sort the data by education (`edu`). `sort` is part of the `by` prefix and not part of the command after the colon.

Finally, you could have typed

```
. bysort edu: summarize income
```

because `bysort` is just `by` with the `sort` option. Choose whichever syntax you like—and stick with it.

If you provide more than one variable in the `bylist`, Stata does what you have told it to do for each possible combination formed by the values of the `bylist` variables. Therefore, the command

```
. by sex edu, sort: summarize income
```

first has Stata summarize the income of all men with elementary education. Then it does the same for all men with intermediate education, and so on. After summarizing the incomes of all educational groups of males, it does the same for each educational group of females.

Some Stata commands allow the `by()` option, which is easy to mistake for the `by` prefix. You need to know the difference: a `by()` option is part of a specific Stata command, and its function is defined in that command. In other words, the `by()` option works differently with different commands. The `by` prefix, on the other hand, does the same thing with every Stata command that allows it: `by` repeats the command for each group indicated by the variable list.

3.2.2 The foreach loop

The `foreach` loop is used to repeat a specific task for each element of a list. This task can be a single, consistently applied command—as with the `by` prefix—or a command that varies slightly within each replication, or even an entire series of Stata commands.

Because the `foreach` loop is so flexible, its syntax is slightly more complicated. A simplified syntax diagram of `foreach` is as follows:

```
foreach lname listtype list {
    commands
}
```

A **foreach** loop always has at least three lines: the first line, which begins the loop, ends with an opening brace. The second line is a Stata command (you can include more lines containing Stata commands). The last line contains a closing brace. You cannot place anything else in the line after the opening brace or in the line containing the closing brace (except for comments).

In the syntax diagram above, the first element of the first line is the command name: **foreach**. Following that are elements you must type: the element name (*lname*), the list type (*listtype*), and the **foreach** list (*list*). Next comes the opening brace.

The **foreach** list is a list of parameters, such as a variable list, a list of numbers, or a list of arbitrary parameters. But you need to tell Stata the list type. For example, if you want to specify a variable list, you use **of varlist**. Finally, you must specify a name for each element of the **foreach** list. The element name is used in the Stata commands between the braces to refer to the elements of the **foreach** list.

There is more to a **foreach** loop, but it is best explained with an example. In what follows, you will type some commands, but make sure you read the fine points carefully before you begin typing.

```
. foreach X of varlist wor01-wor12 {
    . tabulate `X' sex
    . }
```

Here are the fine points. The first line begins the loop. After you press *Enter*, you will see the number 2 on your screen. This is just to remind you that everything you are going to type is being processed by **foreach**. You do not need to do anything about that. Just type the **tabulate** command as usual, press *Enter*, and then type the third “command”, the closing brace. If you make an error somewhere, just press *Enter*, type the closing brace, and press *Enter* again; you will need to start over from the first line. But remember that you can access the commands you previously entered in the Review window by clicking on the command or by pressing *Page Up*.

Note the symbols before and after the **X** in this command. The symbol before the **X** is a single open quotation mark, and the symbol after the **X** is a single close quotation mark. The two symbols are *not* the same even though they look alike in some fonts. The open quote is a backtick or accent grave, whereas the closing right quote is a forward quote or apostrophe. On many American keyboards, the opening quote is found at the top left (near the *Esc* key), whereas the closing quote is found on the right (near the *Enter* key). On European keyboards, the position of both characters vary from country to country. The opening quote is often used to produce the French accent grave, which forces you to press the *Spacebar* before the symbol appears on the screen.

Now you may begin entering the commands. If everything works, you will get the output for 12 **tabulate** commands.

The **of varlist** declares the list type to be a variable list, meaning that the **foreach** list is a list of variable names. The element name is declared to be “**X**”, so we use **X** to represent the current element of the list. Stata requires that you put single quotation

marks around the element name in the part within the braces. After the closing brace, Stata begins the loop, successively replacing the ‘X’ in the second command with the name of each variable in the **foreach** list. Thus **tabulate ‘X’ sex** becomes **tabulate wor01 sex** in the first round of the loop, **tabulate wor02 sex** in the second round, and so on.

The types of **foreach** lists

As we said, you can specify different types of **foreach** lists:

- **of varlist** for lists of variables
- **of newlist** for lists of new variables
- **of numlist** for lists of numbers
- **in** for arbitrary lists of letters, words, and numbers separated by spaces

You have already seen an example of a **foreach** loop with a variable list. The following examples, which we have saved in **foreachkk.do**, show the other list types. You may want to open the file in the Do-file Editor to play with these examples:

```
. doedit foreachkk.do
```

You might want to save the edited version of **foreachkk.do** to, say, **myforeach.do** and run that do-file, or you can type the commands interactively. You will find that in our example do-file, we have indented the part between the opening and the closing brace. Doing so is good style and lets you more easily see what code the **foreach** loop affects. It is also good style to align the closing brace with the first letter of **foreach**. For more about style issues with Stata commands, type

```
. view http://fmwww.bc.edu/repec/bocode/s/stylerules.hlp
```

Our first example uses the list type **of newlist**, which you will find in lines 11–13 of **foreachkk.do**. Here we generate 10 variables containing uniformly distributed random numbers between 0 and 1 by using the Stata function **runiform()**, which creates random numbers:

```
————— begin: foreachkk.do —————
*<
10: // Example with new varlist
11: foreach var of newlist r1-r10 {
12:     generate `var' = runiform()
13: }
————— end: foreachkk.do —————
```

Instead of using the list type **of newlist**, we could have used the list type **of numlist** in this example (see the next example). However, with **of newlist** Stata checks the validity of the variable names to be generated before beginning the loop.

Next we use `of numlist` to replace the variables with newly generated random numbers. Because the variables `r1` to `r10` already exist, we need to use `replace` instead of `generate` within the loop:

```
————— begin: foreachkk.do —————
*  

15: // Example with numlist  

16: foreach num of numlist 1/10 {  

17:     replace r`num' = runiform()  

18: }
————— end: foreachkk.do —————
```

Finally, here is an example with an arbitrary list:

```
————— begin: foreachkk.do —————
*  

20: // Example with anylist  

21: foreach piece in This sentence has 5 pieces {  

22:     display "`piece'"  

23: }
————— end: foreachkk.do —————
```

Several commands within a foreach loop

You can put more than one command within a `foreach` loop, as shown in the next example, in which we generate a centered version of the variables for income and year of birth. To produce a centered version of a variable, we need to subtract its arithmetic mean from each of its values:

```
————— begin: foreachkk.do —————
*  

25: // Example with more than one line  

26: foreach var of varlist ybirth income {  

27:     summarize `var', meanonly  

28:     generate `var'_c = `var' - r(mean)  

29:     label variable `var'_c "`var' (centered)"  

30: }
————— end: foreachkk.do —————
```

We begin by calculating the arithmetic mean (line 27), and then we generate a new variable (line 28) that has the name of the old variable with `_c` appended. The term `r(mean)` refers to a *saved result* of the `summarize` command; the saved result contains the value of the mean we just calculated. We will discuss saved results in chapter 4. Finally, we define a variable label for each of the new variables.

3.2.3 The forvalues loop

Once we understand the `foreach` loop, the `forvalues` loop is easy: it is just a shorter way to set up a `foreach` loop with `of numlist` as the *listtype*.

The simplified syntax diagram of **forvalues** is as follows:

```
forvalues lname = range {
    commands
}
```

This looks very much like the syntax diagram of **foreach**. Again there are at least three lines: the first line begins the loop. The second line is a Stata command (and may be followed by more lines with Stata commands). The last line contains only a closing brace. You cannot place anything in the first line after the opening brace or in the same line as the closing brace other than comments.

In the first line of the syntax diagram, you find the command itself—**forvalues**—followed by an element name (*lname*), an equal-sign, a *range*, and then the opening brace. The *range* is similar to a numlist. You can specify a range of numbers by using the rules you have learned from section 3.1.7 except that you cannot list single numbers or more than one range. For example, you can specify 1(1)10 for all integer numbers from 1 to 10, but you cannot specify 1(1)10 15 19 or 1(1)10 12(2)20.

The element name is an arbitrary name for each number of the *range*. It is used in the Stata commands between the braces to refer to the specified numbers.

Let us try an example. Type the following lines:

```
. forvalues num=1/10 {
. replace r`num' = runiform()
. }
```

num is enclosed in single quotes just like the element name in the **foreach** loop. After you type the third line, Stata should begin the loop, successively replacing ‘*num*’ with each number of the specified *range*: Stata will replace once more the contents of the variables **r1** to **r10**.

The **forvalues** loop may seem unnecessary, given that it is the same as a **foreach** loop with the list type of **numlist**. However, **foreach** with a numlist cannot have more than 1,600 numbers in the list. There is no such limitation with **forvalues**. Moreover, **forvalues** is more efficient for Stata to process.

3.3 Weights

This section is slightly more difficult than the previous sections. We will explain weights as simply as possible, but you may get lost anyway. If so, just skip this section and return to it later.

Weights are allowed for every Stata command that has [*weight*] in its syntax diagram. Think of weights as a way for Stata to treat some observations as more important than others when calculating statistics.

You specify weights through the following general syntax:

[*weighttype* = *varname*]

You must always enclose weights within square brackets. These square brackets have nothing to do with the ones in the syntax diagram that identify optional arguments; they are part of the notation of the weights themselves. Inside the square brackets, you specify the weight type and the name of the variable that contains the weights (the weight variable). You must specify the correct weight type. Make sure you really understand the different weight types before using them (and be somewhat suspicious of results of software packages that do not have different weight types).

There are three main weight types:

- **fweight** frequency weights,
- **aweight** analytic weights, and
- **pweight** sampling weights.

If you simply specify **weight** as the weight type, Stata chooses a type for you; different commands have different default weight types. Some commands also use the weight type **iweight**, or *importance weights*. These weights do not have a formal statistical definition, so you should carefully read the description of commands allowing importance weights to learn how they are used. Importance weights are used in programming contexts and are not needed by regular users. The three main weight types have clear statistical definitions, which we describe next.

Frequency weights

Frequency weights are used for weight variables that contain the number of equal observations in the dataset. Because this may sound a bit puzzling, we will explain it with an example. To begin, type the following command:

<code>. summarize ybirth</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
ybirth	5411	1959.493	18.12642	1909	1992

This command displays a table summarizing data on the year of birth. The calculation is based on 5,411 observations, each of which is a respondent that occurs in the dataset only once.

Now please load and describe `freqwe.dta` from our file package:

```
. use freqwe, clear
(Frequency weighted data from data1.dta)
. describe
Contains data from freqwe.dta
    obs:                 82                               Frequency weighted data from
                                         data1.dta
    vars:                  2                               13 Feb 2012 17:08
    size:                328

variable name  storage  display   value
          type   format   label   variable label
ybirth        int      %8.0g      * Year of birth
n            int      %12.0g      Frequency
                                         * indicated variables have notes
Sorted by: ybirth
```

You can see that these data contain only 82 observations and 2 variables—year of birth (`ybirth`) and `n`.

Now `summarize` again the year of birth but use the weights this time:

```
. summarize ybirth [fweight = n]
Variable |     Obs      Mean   Std. Dev.      Min      Max
ybirth  | 5411  1959.493  18.12642  1909    1992
```

Comparing these results with those above, we see that they are the same. Again we have 5,411 observations with a mean of 1959.49 and a standard deviation of 18.13.

Because our dataset contains only 82 observations, it may come as a surprise that Stata reports 5,411 observations. We should take a closer look at our dataset. Because the dataset is sorted by year of birth, we might profit from simply listing some observations one by one:

```
. list in 1/5
```

	ybirth	n
1.	1909	1
2.	1912	4
3.	1913	1
4.	1914	2
5.	1915	2

In the dataset, there is only one observation for each year of birth, that is, only one observation for year 1909, only one observation for 1912, and so on. But each observation is weighted according to the variable `n`. The observation for year 1909 is weighted with the factor 1; the observation for 1912 is weighted with the factor 4; and if you skip to other observations, you will find that the observation for 1967 is weighted with 106.

If you use frequency weights, Stata interprets the weighting variable as if each observation existed in as many copies as are specified in the weighting variable. The `summarize` command above therefore sees not only one but four observations for 1912, and so on.

`freqwe.dta` contains the same information about the year of birth as the entire `data1.dta`. But rather than listing people with the same year of birth one by one—that is, one observation for 1909, four observations for 1912, etc.—`freqwe.dta` lists identical observations only once, indicating how many such observations exist. Using frequency-weighted data is therefore a more parsimonious way to store the same data and may be more useful for handling datasets that are too big for the working memory of your computer.

To change a dataset from being unweighted to being frequency weighted, you can use the command `contract`; the command `expand` is used for the reverse operation. `expand` is useful if a specific command does not allow frequency weights.

Analytic weights

We will explain the idea of analytic weights with yet another example. Load `analwe.dta` into Stata. Summarize the variable `ybirth` once with frequency weights and then a second time with analytic weights:

```
. use analwe
(Example data with analytic weights)
. summarize ybirth [fweight = n]
      Variable |       Obs        Mean    Std. Dev.        Min        Max
                 ybirth |     5411    1959.547     1.790374   1954.894   1962.191
. summarize ybirth [aweight = n]
      Variable |       Obs        Weight         Mean    Std. Dev.        Min        Max
                 ybirth |        14          5411    1959.547     1.857787   1954.894   1962.191
```

You get the same mean with both weight types. But take a look at the standard deviation (Std. Dev.), which is about 1.79 with frequency weights and 1.86 with analytic weights. Not only are the two values different, but they both differ considerably from the standard deviations calculated in the last section. Which one is correct?

The answer depends on how the data were created. If the weights reflect the number of respondents (observations) born at the respective year of birth, the frequency weighted result would be correct. But our example dataset was created differently. Let us take a look at some of the data:

```
. list in 1/5
```

	state	ybirth	n
1.	Berlin	1962.19	208
2.	Schleswig-Hols.	1956.61	166
3.	Hamburg/Bremen	1954.89	101
4.	Lower Saxony	1957.38	412
5.	N-Rhein-Westfa.	1960.82	1145

The first observation is from Berlin, a German city that is also a German state. The second observation is from Schleswig-Holstein, another state in the far north of Germany. The complete dataset contains 14 observations, one for each state of Germany.⁵ For each state, there is a variable for the average year of birth and a weighting variable. The year of birth is the mean of all respondents from the same state, and the weighting variable is the number of respondents used to calculate that mean. Such data sometimes are called *aggregate data*.

Clearly, not all respondents used to calculate the mean for each state have the same year of birth. We do not have 5,411 observations; rather, we have 14 means made from different numbers of observations.⁶ You need to use *analytic weights* for such data. With analytically weighted data, each observation on a variable is itself a mean, and the weight represents the sample size used to compute the mean.

Sampling weights

The third weight type is probably the most interesting—and may be one of your reasons for preferring Stata over other statistical packages.

Often you analyze data using a sample from a population and use the sample to infer something about the larger population. Many statistical tools assume that the sample is chosen by simple random sampling with replacement, meaning that each element of the population has the same sampling probability. In practice, you rarely have simple random samples. Complex samples—that is, samples with observations of different sampling probabilities—are much more common.

5. Actually, Germany has 16 states. However, we merged the data from Hamburg and Bremen and the data from Rheinland-Pfalz and Saarland into one category for data protection.

6. See the file `cranalwe.do` for how we created `analwe.dta` from `data1.dta`.

If you use statistical tools that assume simple random samples but your dataset comes from complex samples, you can make two mistakes. First, you get biased point estimators; that is, the means, medians, regression coefficients, etc., do not reflect the true values in the population. Second, you incorrectly calculate the sampling distribution of your point estimators, leading you to evaluate confidence intervals erroneously, and so on (Kish 1965).

To avoid the first mistake, you could weight your data with the reciprocal of the sampling probability. Then it would not matter if you use frequency weights or analytic weights, because both lead to the same results. With the second mistake, the issue is more complicated. Both frequency and analytic weights lead to incorrect results. An observation cannot be regarded as referring to more than one respondent, nor can one observation be regarded as an aggregate measure of more than one respondent. Each observation is just one simple observation. Stata's probability weights give correct standard errors in this case.

Stata's tools for dealing with probability weights are part of a broader class of commands for dealing with complex samples. These are called survey commands, which we will introduce in chapter 8.

3.4 Exercises

1. Compute a mean and a standard deviation for each of the following variables in our data file `data1.dta`:
 - All variables that begin with “wor”.
 - All variables with information on the quality of dwellings.
 - All variables related to “satisfaction”.
2. Abbreviate the following commands as much as you can (and so that Stata will still recognize them):


```
. summarize ymove ybuild condit eqphea area1 area2
. regress wor01 ybirth income
```
3. Use the command `gen inc = hhinc` to create a new variable `inc` that is equal to `hhinc`, and rerun your last command. Explain why the same command now produces a different result.
4. Read `help set varabbrev` and consider the advantages and disadvantages of `set varabbrev off`.
5. List the person ID, interviewer number, year of birth, household income, and life satisfaction of the 10 observations with the lowest household income. Then rerun the command without the observation numbers and add a line for the means of the variables in your list.

6. Create a table with gender as a column variable and life satisfaction as a row variable. Include only respondents from West Germany (`state<10`) in the table. Have Stata display only column percentages.
7. Create a table with gender as a column variable and life satisfaction as a row variable, separate for each state.
8. The following sequence of commands creates a standardized variable for year of birth:

```
. summarize ybirth  
. generate ybirth_s = (ybirth - r(mean))/r(std)
```

Create a loop to standardize the following variables in a similar fashion: `ymove`, `size`, `rooms`, `lsat`, `rent`, `hhsiz`, `yedu`, `hhinc`, and `income`.

9. Include a command in your loop that labels the variables above (`ymove`, `size`, etc.) as “standardized”.

4 General comments on the statistical commands

In this book, we refer to all commands that do some statistical calculation as statistical commands. In chapter 1, for example, we used the statistical commands `summarize`, `tabulate`, and `regress`. Despite their different purposes, statistical commands have one feature in common: they store their results internally. This chapter describes these saved results and what you can do with them. You can do further calculations with them, save them as variables, export them to other programs, or just display them on the screen.

With saved results, Stata distinguishes between regular statistical commands (r-class) and estimation commands (e-class). We start our discussion with regular statistical commands and move over to estimation commands afterward.

To follow our examples, load `data1.dta`:¹

```
. use data1, clear
```

4.1 Regular statistical commands

All the names of stored results used for regular statistical commands have the format `r(name)`, where *name* varies between the various results stored by one command.

The command `summarize`, for example, is an r-class command that stores its results in the following repositories:

<code>r(N)</code>	number of observations	<code>r(sd)</code>	standard deviation
<code>r(sum_w)</code>	sum of the weights	<code>r(min)</code>	minimum
<code>r(mean)</code>	arithmetic mean	<code>r(max)</code>	maximum
<code>r(Var)</code>	variance	<code>r(sum)</code>	sum of variable

Specifying the `detail` option with the `summarize` command saves more statistics.

Why do we need saved results? They have many uses. Generally, whenever you need the result of a command to set up another command, you should use the saved

1. You may want to check that your current working directory is `c:\data\kk3`; see page 3.

results. The rule is this: Never transcribe the results of commands by hand into other commands; use the saved results instead. Here are some examples with the saved results of **summarize**.

You can use the saved results to center variables, that is, to subtract the mean of a variable from each of its values:

```
. summarize income
. generate inc_c = income - r(mean)
```

You can also use the saved results to standardize variables. You first center your variable and then divide it by its standard deviation:

```
. summarize income
. generate inc_s = (income - r(mean))/(r(sd))
```

You can use the saved results to make the high values of your variable low and vice versa (mirroring):

```
. summarize income
. generate inc_m = (r(max) + 1) - income
```

You can also use the saved results to calculate the lower and upper bounds of the 95% confidence interval around the mean. Under standard assumptions, the boundaries of the 95% confidence interval are given by $CI = \bar{x} \pm 1.96 \times \sqrt{s^2/n}$, where \bar{x} is the mean, s^2 is the variance, and n is the number of observations. These figures can be found in the saved results of **summarize**:

```
. summarize income
. display r(mean) + 1.96 * sqrt(r(Var)/r(N))
. display r(mean) - 1.96 * sqrt(r(Var)/r(N))
```

Every statistical command of Stata saves its main results. To work with a specific result, you must know where Stata stores that result. You can find where saved results are stored in the following sources:

- The *Reference* manuals and help files: The entry for each statistical command has a section called *Saved results*, which states the names and meanings of the saved results.
- The command **return list** shows the names and contents of the saved results of the last issued r-class command:

```
. summarize income
. return list
```

Each new r-class command deletes the saved results of the previous r-class command. For example, the commands

```
. summarize income
. tabulate edu sex
. generate inco_c = income - r(mean)
```

do not generate a centered version of `income` because `tabulate` deletes all saved results of `summarize` and saves its own results. Some results from `tabulate` have the same names as those of `summarize`, and others do not. `tabulate` does not save `r(mean)`, for example, so the new variable `inco_c` contains missing values.

The transitory nature of the saved results often forces us to save those results more permanently. You can do this using local macros, which unlike saved results are not overwritten until you explicitly do so.²

You define local macros using the command `local`. For example, typing

<code>. summarize income</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
income	4779	20540.6	37422.49	0	897756
<code>. local x = r(mean)</code>					

stores the contents of the saved result `r(mean)` in a local macro called `x`. You can use another name as long as the name is not longer than 31 characters. After defining the local macro, you can use the name of the local macro whenever you need to refer to the result of the `summarize` command above. The contents of the local macro will not be overwritten unless you explicitly redefine the macro.

To use the local macro, you must explicitly tell Stata that `x` is the name of a local macro by putting the name between single quotes. To simply display the content of the local macro `x`, you would type

```
. display `x'  
20540.6
```

If this command does not work for you, you probably incorrectly typed the opening and closing quotes. The opening quote is a backtick or accent grave, whereas the closing right quote is a forward quote or apostrophe. On many American keyboards, the opening quote is found at the top left (near the *Esc* key), whereas the closing quote is found at the middle right (near the *Enter* key). On European keyboards, the position of both characters varies from country to country. The opening quote is often used to produce the French accent grave, which forces you to press the *Spacebar* before the sign appears on the screen.

2. At the end of a do-file or program, any local macros created by that do-file or program are dropped; see chapter 12.

Now let us turn to a practical application. You can calculate the difference between the means of a variable for two groups as follows:

```
. summarize income if sex == 1
      Variable   Obs    Mean    Std. Dev.    Min    Max
      income     2320  28190.75  47868.24      0  897756
. local meanincmen = r(mean)
. summarize income if sex == 2
      Variable   Obs    Mean    Std. Dev.    Min    Max
      income     2459  13322.89  21286.44      0  612757
. display r(mean) - `meanincmen'
-14867.866
```

The `display` command takes the mean of income computed for women (stored in `r(mean)`) and subtracts the previously computed mean of income computed for men (which we stored in `meanincmen`), yielding a difference of € –14,867.866. Unfortunately, you cannot store all saved results of statistical commands in local macros because sometimes results are stored as matrices. To permanently store the results in matrices, you need to use the Stata matrix commands, which are mainly programming tools, so we will not deal with them here. For more information, see [U] **14 Matrix expressions** or type

```
. help matrix
```

4.2 Estimation commands

We use the term estimation command or, more specifically, e-class command for all commands that store their results in repositories with names of the format `e(name)`. Almost always these are commands that fit statistical models, such as linear or logistic regression (see chapters 9 and 10).

Most of what we discussed for r-class commands also holds for e-class commands, but there are a few subtle differences.

- While the manual and help file entries list the saved results of estimation commands under *Saved results* just as for the r-class commands, you have to use the command `ereturn list` instead of `return list` to see the names and contents of the saved results of the last issued e-class command:

```
. regress income yedu
. ereturn list
```

- Like r-class commands, each new e-class command deletes the saved results of the previous e-class command. However, while each new e-class command also deletes all saved results of the previous r-class command, e-class command results are not overwritten by r-class commands.

- The names of repositories of r-class commands differ wildly between commands, while the names of repositories of e-class commands tend to be the same for all estimation commands. Specifically, all Stata estimation commands save at least two specific results, `e(b)` and `e(V)`. These saved results are the vector of the fit model coefficients and their variance–covariance matrix. Type

```
. matrix list e(b)
. matrix list e(V)
```

to see what is in these results.

- You can access a single piece of the values stored in `e(b)` with `_b[name]`. For example, to access the estimated regression coefficient for `yedu`—the first value in the column headed with `Coef.` of the regression output—you type

```
. display _b[yedu]
```

- In the output of the statistical models, the column headed with `Std. Err.` contains the standard error (see section 8.2.2) of the regression coefficient. The value is equal to the square root of the entry in the main diagonal of the variance–covariance matrix left behind as `e(V)`. You can access that standard error with `_se[name]`, where `name` is the name of the coefficient you are interested in:

```
. display _se[yedu]
```

In a nutshell, Stata handles the saved results of statistical models with more care than those of regular statistical commands. The reason for this is that data analysts often want to investigate further the results of statistical models. It is therefore necessary that the results of statistical models are kept longer than those of regular statistical commands.

If normal durability of e-class results is not long enough, Stata allows for saving the regression results even more permanently. Specifically, you can use `estimates store` to save the results of statistical models in the computer’s memory:

```
. estimates store myreg1
```

During a Stata session, estimation results stored with `estimates store` can be restored with `estimates restore`:

```
. estimates restore myreg1
(results myreg1 are active now)
. ereturn display
```

income	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
yedu	3900.475	204.9923	19.03	0.000	3498.589 4302.362
_cons	-24845.36	2497.585	-9.95	0.000	-29741.86 -19948.85

This ability to store and restore results of regression models will turn out to be very useful if you wish to compare the coefficients between several statistical models (see section 9.5).

4.3 Exercises

1. Load a subset from the National Longitudinal Survey into Stata memory by typing

```
. webuse labor, clear
```

2. Request a list of the stored results that are available after running the following commands:

```
. summarize whrs
. summarize whrs hhrs
. summarize hhrs, detail
. tabulate k16
. tabulate cit k16
. count if cit
. correlate whrs ww
. correlate whrs ww faminc
```

3. Find out whether the following commands are e-class or r-class:

```
. summarize whrs
. mean whrs
. ci whrs
. regress ww whrs we
. logit cit faminc
```

4. Generate a variable containing a centered version of the variable **faminc**; do not input the statistics by hand.
5. Generate a variable containing a *z*-standardized version of the variable **faminc**; do not input the statistics by hand. Note: The *z*-standardized values of the variable *x* are calculated with $z_i = (x_i - \bar{x})/s_x$, where \bar{x} is the mean of variable *x* and s_x is the standard deviation of *x*.
6. Create a local macro containing the number of cases in the dataset, and display the content of this macro.
7. Display the range of **faminc**.

5 Creating and changing variables

In everyday data analysis, creating and changing variables takes most of the time. Stata has two general commands for these tasks: `generate` and `replace`, which are the bread and butter of Stata users. `egen` and `recode` are also often used; they allow some shortcuts for tasks that would be tedious using `generate` and `replace`.

When we work with data that have not been prepared and preprocessed nicely, we sometimes run into complex data manipulation problems. There is no single technique for solving these problems—which can challenge experienced users as well as newcomers—though `generate` and `replace` are useful. Although we cannot describe all possible data manipulation challenges you might encounter, we will give you some general advice for generating variables for use in Stata. This chapter will give you the tools to solve even the most difficult problems.¹

To use our examples, load `data1.dta`:²

```
. use data1, clear
```

Then execute the command

```
. numlabel _all, add
```

which we will explain in detail in section 5.6 below. For now it is enough to know that the command makes creating and changing variables somewhat more comfortable.

5.1 The commands `generate` and `replace`

`generate` creates a new variable, whereas `replace` changes the contents of an existing variable. To ensure that you do not accidentally lose data, you cannot overwrite an existing variable with `generate` and you cannot generate a new variable with `replace`. The two commands have the same command syntax: you specify the name of the command followed by the name of the variable to be created or replaced. Then you place an equal-sign after the variable name, and specify an expression to be created or replaced. The `by` prefix is allowed, as are the `if` and `in` qualifiers.

1. Difficult recoding problems appear almost weekly in Statalist. For practice, you should try solving some of them and read the solutions suggested by other readers. For more about Statalist, see chapter 13.

2. You may want to check that your current working directory is `c:\data\kk3`; see page 3.

You can use `generate` to create new variables. For example,

```
. generate newvar = 1
```

generates the new variable called `newvar` with the value of 1 for each observation:

. tabulate newvar			
newvar	Freq.	Percent	Cum.
1	5,411	100.00	100.00
Total	5,411	100.00	

You can also create a new variable out of existing variables, for example,

```
. generate pchinc = hhinc/hhsiz
```

where you divide the household income (`hhinc`) by the number of persons in the household (`hhsiz`). `generate` automatically does this for every observation in the data, and the results of the calculations are written to the variable for household income per capita (`pchinc`).

```
. list hhinc hhsiz pchinc in 1/4
```

	hhinc	hhsiz	pchinc
1.	22093	2	11046.5
2.	22093	2	11046.5
3.	62078	2	31039
4.	62078	2	31039

`replace` changes the content of a variable. Below we change the content of the variable `newvar` to 0.

```
. replace newvar = 0
```

The variable now contains the value 0 for all observations instead of the value 1:

. tabulate newvar			
newvar	Freq.	Percent	Cum.
0	5,411	100.00	100.00
Total	5,411	100.00	

5.1.1 Variable names

When working with `generate`, remember some rules about variable names. The names of variables can be up to 32 characters long. However, it is a good idea to keep the names concise to save time when you have to type them repeatedly. To ensure that

your Stata displays the entire variable name in the output of `describe`, you may want to restrict yourself to names up to 14 characters long.

You can build your names with letters (A–Z and a–z), numbers (0–9), and underscores (_), but you cannot begin the variable name with a number. The following names are not allowed:

<code>_all</code>	<code>double</code>	<code>long</code>	<code>_rc</code>
<code>_b</code>	<code>float</code>	<code>_n</code>	<code>_skip</code>
<code>byte</code>	<code>if</code>	<code>_N</code>	<code>str#</code>
<code>_coef</code>	<code>in</code>	<code>_pi</code>	<code>using</code>
<code>_cons</code>	<code>int</code>	<code>_pred</code>	<code>with</code>

Some variable names are not recommended even though they are allowed. Avoid using the single letter `e` to prevent confusion with the “e” in scientific notation. Avoid names beginning with an underscore, because such names are used by Stata for internal system variables. Future enhancements may lead to conflicts with such names, even if they are not on the above list.

5.1.2 Some examples

Variables created or modified with `generate` or `replace` are assigned the value of the expression after the equal-sign. The general rules for expressions given in section 3.1.6 also apply here.

When generating new variables, you can perform simple calculations. For example,

```
. generate hhinc_USD = hhinc * 1.46028
(4 missing values generated)
```

multiplies all values in the variable `hhinc` (household income) with the 2009 average exchange rate³ of 1.46028 to convert the values from Euros into U.S. dollars. The new variable `hhinc_USD` has four missing values because `hhinc` had four missing values.

```
. generate age = 2009-ybirth
```

determines the age of each respondent at the time the survey was conducted.

You can also use mathematical functions to generate new variables. For example,

```
. generate loginc = log(income)
(2001 missing values generated)
```

calculates the natural logarithm of personal income for each observation and stores the result in the variable `loginc`. The command

```
. replace loginc = log(income)/log(2)
(3410 real changes made)
```

3. <http://www.oanda.com/lang/de/currency/average>

produces the base-2 logarithm instead of the natural logarithm. The result of the calculation overwrites the variable `loginc`.

Or you can use statistical functions, such as

```
. generate r = runiform()
```

which generates a random variable from a uniform distribution with values ranging from 0 to nearly 1. A random variable with a standard normal distribution (a variable with a mean of 0 and a standard deviation of 1) is generated with

```
. generate r01 = rnormal()
```

You can use every function introduced in section 3.1.6 with `generate` or `replace`, and you can combine them with the operators `replace` with the algebraic operators addition, subtraction, multiplication, division, and power is quite straightforward. You might be surprised, however, that you can also use relational operators with `generate` and `replace`. Let us walk through some examples of using `generate` and `replace` with relational operators.

Suppose that we need to generate the variable `minor` to indicate respondents who had not reached the age of legal adulthood in 2009, the time the survey was conducted. This means the respondent will have a value greater than 1991 on the variable for year of birth (`ybirth`). The new variable `minor` must contain the value 1 for all respondents younger than 18 and the value 0 for all other persons. One way to create this variable is to generate the variable `minor` having a value equal to 0 for all respondents, and then to replace the values of this newly generated variable with 1 for all respondents younger than 18.

```
. generate minor = 0
. replace minor = 1 if ybirth > 1991 & !missing(ybirth)
(66 real changes made)
```

Another way to create the variable `minor` is based on what we explained in section 3.1.5: expressions with relational operators can be true (1) or false (0). Knowing this, we can create a variable indicating minors in a single line; we will call the variable `minor2` this time.

```
. generate minor2 = ybirth > 1991
```

The expression `ybirth > 1991` is false (0 in Stata) for all interviewees born before 1991, so all those interviewees get the value 0 for the new variable `minor2`. For those born in 1991 and later, the expression `ybirth > 1991` is true (1 in Stata); therefore, the new variable becomes a 1 for these observations:

. tabulate minor2				
minor2	Freq.	Percent	Cum.	
0	5,345	98.78	98.78	
1	66	1.22	100.00	
Total	5,411	100.00		

Note that `minor2` will also equal 1 for any observations for which `ybirth` is missing, because Stata treats missing values as very large numbers. The command

```
. generate minorshh = ybirth > 1991 & hysize == 1 if !missing(ybirth)
```

generates the variable `minorshh` that equals 1 for all respondents younger than 18 who lived alone at the time of the interview (`hysize == 1`) and that equals 0 elsewhere.

As we just demonstrated, you can easily generate variables with the values 0 and 1. Such variables are called *dummy variables*, or sometimes just *dummies*. You will use them quite often, but be careful if there are missing values among the variables from which you build your dummy. We recommend that you restrict the command to the valid cases by using an `if` qualifier. For more about handling missing values, see section 5.5.

You can mix relational and algebraic operators in your expressions. For example, you can construct an additive index for quality of a dwelling by summing expressions for all the characteristics of a dwelling. Using `generate` and what you have learned about expressions with relational operators that are either true or not true, you can create the additive index for dwelling quality like this:

```
. generate quality = (eqphea==1) + (eqpter==1) + (eqpbass==1)
> + (eqpgar==1) + (eqpalm==1) + (eqpsol==1) + (eqpair==1)
> + (eqplif==1) + (eqpnaj==1)
```

```
. tabulate quality
```

quality	Freq.	Percent	Cum.
0	7	0.13	0.13
1	87	1.61	1.74
2	464	8.58	10.31
3	1,709	31.58	41.90
4	2,560	47.31	89.21
5	504	9.31	98.52
6	72	1.33	99.85
7	5	0.09	99.94
8	3	0.06	100.00
Total	5,411	100.00	

But note that

```
. generate quality2 = eqphea==1 + eqpter==1 + eqpbas==1
> + eqpgar==1 + eqpalm==1 + eqpsol==1 + eqpair==1 + eqplif==1 + eqpnjrj==1
. tabulate quality2
   quality2 |      Freq.    Percent      Cum.
   0          |     5,411     100.00    100.00
   Total      |     5,411     100.00
```

would not have led to the result desired.

You do need parentheses around the relational operators in this example. Addition has priority over equality testing, so in Stata the expression `kitchen == 1 + shower == 1 + ...` reads as `kitchen == (1 + (shower == 1 + ...))`. So be careful with complicated expressions! Use parentheses to make complicated expressions clear, even in cases when the parentheses are not required by Stata.

Of course, you can use `generate` and `replace` not only to create dummy variables but also for all kinds of recoding. Say, for example, that we would like to reduce our quality index to three categories: poor, medium, and high, where poor indicates two or fewer amenities, medium is three to five, and high is more than five.

It is easy to forget to look for missing values when you generate a new variable from another. Therefore, we recommend generating a new variable with all 0s to start with. After that, you can do all your codings with `replace`.

```
. generate quality3 = 0
. replace quality3 = 1 if quality <= 2
(558 real changes made)
. replace quality3 = 2 if quality > 2 & quality <= 5
(4773 real changes made)
. replace quality3 = 3 if quality == 6 | quality == 7 | quality == 8
(80 real changes made)
```

Use `tabulate` to check your work. For the example above, this would be `tabulate quality quality3`. If there still are any cases with 0 in the new variable, you may have forgotten to specify a value for some cases. You can set the remaining cases to missing by replacing the 0 with a dot.

5.1.3 Useful functions

We wrote above that you can place any expression after the equal-sign of `generate` or `replace` and that the general rules for expressions given in section 3.1.6 apply. From this, it follows that you can use all general Stata functions mentioned in [D] **functions**. In this section, we will briefly describe some functions that are often used in the context of `generate` and `replace` but seldom described. If you like this section, you will find more of this in Cox (2011).

Sometimes one wishes to round numbers up or down. Several functions exist for this purpose, but `round(a, b)` seems to be the most natural choice. If you use the function with just one argument, it rounds that argument to the nearest integer. For example, the variable `hhinc_USD` created on page 79 contains many digits behind the decimal point. The command

```
. generate hhinc_USDr1 = round(hhinc_USD)
(4 missing values generated)
```

rounds these values to the nearest integer value. By adding the second argument to the function, you can also round to multiples of that second argument. Here we round the values in `hhinc_USD` to multiples of 500:

```
. generate hhinc_USDr2 = round(hhinc_USD,500)
(4 missing values generated)
. list hhinc_USDr1 hhinc_USDr2 hhinc_USD in 1/5
```

	hhinc_~1	hhinc_~2	hhinc_~D
1.	32262	32500	32261.97
2.	32262	32500	32261.97
3.	90651	90500	90651.27
4.	90651	90500	90651.27
5.	35891	36000	35890.76

Dichotomous variables with values 0 and 1 (dummy variables) are often used in data analysis. The functions `inlist(z, a, b, ...)` and `inrange(z, a, b)`, which we already described in section 3.1.6, are very useful for creating such dummy variables from existing polytomous or continuous variables. Take as a first example the variable `pib`, which holds the political party that each respondent supports.

Political party supported		Freq.	Percent	Cum.
1. SPD		733	34.89	34.89
2. CDU/CSU		795	37.84	72.73
3. FDP		144	6.85	79.58
4. Greens/B90		211	10.04	89.62
5. Linke		145	6.90	96.53
6. DVU, Rep., NPD		32	1.52	98.05
7. Other parties		29	1.38	99.43
8. Several parties		12	0.57	100.00
Total		2,101	100.00	

Of these parties, the “SPD”, the “Greens/B90”, and the “Linke” are usually considered to be left parties. If you wish to create a dummy variable for left parties, you can type

```
. generate left1 = (pib==1) | (pib==4) | (pib==5) if !missing(pib)
(3310 missing values generated)
```

or, much shorter using `inlist(z, a, b, ...)`,

```
. generate left2 = inlist(pib,1,4,5) if !missing(pib)
```

Likewise, the function `inrange(z, a, b)` can be used to create a dummy variable for persons aged between 25 and 65:

```
. generate age25to65 = inrange(age,25,65) if !missing(age)
```

The function `inlist()` has a hidden gem: You can use the function not only for checking whether the values of a variable are equal to one of the specified values, but also for checking the other way around, that is, checking whether any of a list of variables contains a specified value. Here we use this idea to check whether any of the respondents has the value 3 (“not concerned at all”) on at least one of the variables about worries for the future:

```
. generate optimist = inlist(3,wor01,wor02,wor03,wor04,wor05,wor06,wor07,  
> wor08,wor09,wor10,wor11,wor12)
```

Another useful function is `cond(a, b, c)`. The function returns the result of expression *b* if the expression *a* is true (nonzero) and returns the result of expression *c* if expression *a* is false (zero). In the following example, we use the function to create a variable that is equal to the smaller value of the two variables `wor02` (“worried about finances”) and `wor03` (“worried about stability of financial markets”). Hence, the variable holds the respondents’ worries about finances if they are more concerned about the stability of financial markets, while it holds the respondents’ worries about the stability of financial markets if they are more concerned about finances.

```
. generate wor_finance = cond(wor02<wor03,wor02,wor03) if !missing(wor02,wor03)  
(122 missing values generated)
```

Finally, there are `recode(a, b, ...)`, and `irecode(a, b, ...)`. Both of these functions bin neighboring values together. Let us begin with an example of the `recode()` function. The command

```
. generate income_groups = recode(income,10000,20000,50000,100000)  
(632 missing values generated)
```

creates the new variable `income_groups`, which becomes 10,000 for all observations with an income of €10,000 or below (that is, between €0 and €10,000), €20,000 for all observations between €10,001 and €20,000, and €50,000 for all observations between €20,001 and €50,000. All nonmissing observations with an income above €50,000 become €100,000 on `income_groups`.

Once you understand `recode()`, `irecode()` is easy. It does almost the same as `recode()` but with two differences: First, `irecode()` returns values $1, 2, \dots, k$ instead of the limits of the various groups themselves. Second, while `recode()` returns the last argument specified by the user for all values greater than the last argument, `irecode()` creates another category for those values. You will see the difference by comparing the last category in the output of

<code>. tabulate income_groups</code>				
income_grou ps	Freq.	Percent	Cum.	
10000	2,219	46.43	46.43	
20000	666	13.94	60.37	
50000	1,489	31.16	91.53	
100000	405	8.47	100.00	
Total	4,779	100.00		

with the last two categories of

<code>. generate income_groups2 = irecode(income,10000,20000,50000,100000)</code> (632 missing values generated)				
<code>. tabulate income_groups2</code>				
income_grou ps2	Freq.	Percent	Cum.	
0	2,219	46.43	46.43	
1	666	13.94	60.37	
2	1,489	31.16	91.53	
3	357	7.47	99.00	
4	48	1.00	100.00	
Total	4,779	100.00		

You can solve most problems using the simple commands described here and in the previous sections. Many data manipulation problems can also be solved using shortcuts. For example, you can easily create the `quality3` variable by using the `recode` command shown in section 5.2.1. You could have created the additive index `quality` using `egen` (see section 5.2.2). But being fluent in the use of `generate` and `replace` will help you find a solution even when there are no such shortcuts. This is especially true if you know the concepts we explain in the following section. They are a bit more difficult, but it is worthwhile to know them.

5.1.4 Changing codes with *by*, *_n*, and *_N*

Let us intrigue you with a small example. Suppose that you want to generate a variable that represents the number of persons interviewed by each interviewer. At the moment, your data look as follows:⁴

4. The order in which respondents were interviewed by each interviewer may differ in your dataset.

```
. sort intnr
. list persnr intnr in 1/8, sepby(intnr)
```

	persnr	intnr
1.	4035001	10
2.	4035002	10
3.	3934401	10
4.	3934402	10
5.	10144301	10
6.	4035003	10
7.	8501	18
8.	8502	18

For each respondent in the dataset (indicated through `persnr`), you can get the identification number of the interviewer (`intnr`) conducting the interview. Thus you can easily see that the interviewer with the identification number 10 has conducted six interviews and that interviewer 18 has done only two. You can write these results into your data, and then your data will look like this:

	persnr	intnr	newvar
1.	4035001	10	6
2.	10144301	10	6
3.	3934402	10	6
4.	4035003	10	6
5.	3934401	10	6
6.	4035002	10	6
7.	8501	18	2
8.	8502	18	2

But how would you do this automatically? That is, how would you generate this variable within Stata? Think about it before reading on.

If you have an idea, most likely you have already worked with statistical software packages before. You probably thought of somehow combining the observations for each interviewer into what many software programs call an *aggregate*. If you want to do it that way with Stata, you can use `collapse`. But Stata provides a smarter solution:

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

This solution has many advantages. It is short and fast, keeps the original data file, and requires you to understand only basic concepts, most of which you already know. You know `generate`, and you know the `by` prefix. What you do not know is `_N`. So, let us explain.

To understand *_N*, you need to understand the system variable *_n*, which contains the position of the current observation in the data. You can use *_n* to generate a running counter of the observations:

```
. generate index = _n
. list persnr intnr index in 1/8, sepby(intnr)
```

	persnr	intnr	index
1.	4035001	10	1
2.	4035002	10	2
3.	3934401	10	3
4.	3934402	10	4
5.	10144301	10	5
6.	4035003	10	6
7.	8501	18	7
8.	8502	18	8

For the first observation, the new variable *index* contains the number 1; for the second, the number 2; for the eighth, the number 8; and so on. Used with *by*, the system variable *_n* is the position within each by-group. This way you can generate a running counter for the number of interviews by each interviewer:

```
. by intnr: generate intIndex = _n
. list persnr intnr index intIndex in 1/8, sepby(intnr)
```

	persnr	intnr	index	intIndex
1.	4035001	10	1	1
2.	4035002	10	2	2
3.	3934401	10	3	3
4.	3934402	10	4	4
5.	10144301	10	5	5
6.	4035003	10	6	6
7.	8501	18	7	1
8.	8502	18	8	2

After learning *_n*, *_N* is easy: it indicates the highest value of *_n*. Without the prefix *by*, *_N* has the same value as *_n* for the last observation, which is just the number of observations in the dataset:

```
. display _N
5411
```

In combination with the `by` prefix, `_N` contains the value of `_n` for the last observation within each by-group. The `_n` for the last observation in the by-group of interviewer 10 in our example is 6. Therefore, `_N` equals 6 in this by-group, and `by intnr: generate intcount = _N` is equivalent to `generate intcount = 6`. In the by-group of interviewer 18, the value of `_n` in the last observation is 2. `by intnr: generate intcount = _N` is now equivalent to `generate intcount = 2`, and so forth. This is what we want to have:

```
. list persnr intnr index intIndex intcount in 1/8, sepby(intnr)
```

	persnr	intnr	index	intIndex	intcount
1.	4035002	10	1	1	6
2.	4035001	10	2	2	6
3.	4035003	10	3	3	6
4.	3934401	10	4	4	6
5.	3934402	10	5	5	6
6.	10144301	10	6	6	6
7.	8502	18	7	1	2
8.	8501	18	8	2	2

Recoding with `by`, `_n`, and `_N` differs slightly from the process of recoding described earlier. It takes some time to get used to it, but once you do, you will find it very useful. Here is one more example.

Suppose that you need a variable that contains a unique value for each combination of marital status and education. The variable should be 1 for married interviewees (`mar==1`) with the lowest educational level (`edu==1`), 2 for married interviewees with secondary education (`edu==2`), etc. The standard way to generate such a variable is

```
. generate maredu = 1 if mar == 1 & edu == 1
. replace maredu = 2 if mar == 1 & edu == 2
```

and so on. After typing 25 commands, you will get what you want. Using the concepts explained above, however, you need only two lines:

```
. by mar edu, sort: generate maredu2 = 1 if _n == 1
. replace maredu2 = sum(maredu2)
```

Why? Consider the following fictitious data:

	mar	edu	step1	step2
1.	1	1	1	1
2.	1	1	.	1
3.	1	1	.	1
4.	1	1	.	1
5.	1	1	.	1
6.	1	2	1	2
7.	2	1	1	3
8.	2	1	.	3
9.	2	1	.	3
10.	2	2	1	4
11.	2	2	.	4
12.	2	2	.	4

The data file contains the variables representing marital status and education, each of which has two categories. The data file is sorted by marital status and education. If you type `by mar edu: generate maredu2 = 1 if _n == 1`, you get the variable of step 1. The command assigns 1 to the first observation (`if _n == 1`) of each by-group and assigns missing otherwise. Because there are two `by` variables in the command, a new by-group begins when one or both of the variables change. The second command calculates the running sum of the variable generated in the first step. Missing values are treated as 0 in this calculation.

5.1.5 Subscripts

Subscripts are commonly used in statistical formulas to specify which observation of a variable you refer to. For example, in the term x_3 the number 3 is a subscript; used in a statistical formula, the term would refer to the third observation of the variable x . Likewise, we use subscripts in Stata to refer to a specific observation of a variable. You specify a subscript by putting brackets after a variable name. Inside the brackets, you use an expression to specify the observation in question. For example, `ybirth[1]` would refer to the first observation of the variable `ybirth`; `income[_N]` would refer to the last observation of the variable `income`.

Subscripts are helpful in datasets with hierarchical structures. Such hierarchical structures are present in almost every dataset. The file `data1.dta` has a hierarchical structure because we have observed persons within households. We have, for example, information of the labor earnings (`income`), the social class (`egp`), and the relationship to the head of the household (`rel2head`) of every person above the age of 15 in each household.

The hierarchical structure of `data1.dta` can bee seen very clearly in the following subset of the data:

```
. sort hhnr2009
. list hhnr2009 income egp rel2head in 26/35, sepby(hhnr2009)
```

	hhnr2009	income	egp	rel2head
26.	738	57064	2. Service class 2	2. Partner
27.	738	27601	9. Semi- and unskilled manual workers	3. Child
28.	738	.	.b. Does not apply	3. Child
29.	738	47501	2. Service class 2	1. Head
30.	794	19638	9. Semi- and unskilled manual workers	3. Child
31.	794	943	15. unemployed	3. Child
32.	794	2112	.b. Does not apply	2. Partner
33.	794	26810	2. Service class 2	1. Head
34.	850	20206	15. unemployed	1. Head
35.	850	.	18. Retired	2. Partner

Suppose that you would like to have a variable equal to the total labor earnings for each household. How can you get this? Try to find a solution yourself before you continue reading.

Our solution is the following:

```
. by hhnr2009: generate hhearn=sum(income)
. by hhnr2009: replace hhearn = hhearn[_N]
```

You already know the command `generate hhearn = sum(income)`, which calculates the running sum of income. Here you do not sum from the first observation to the last but sum only from the first observation of each household to the last observation of that household. You do this by typing `by hhnr2009:` before the command. Then you can find the sum of personal incomes for all adults of the household in the last observation (`_N`) of each household. This number is copied to all other observations of the same household by using the command `by hhnr2009: replace hhearn = hhearn[_N]`.

Similarly, you can find the social class (`egp`) of the head of the household (`rel2head==1`) and assign this value to all members of the same household with

```
. sort hhnr2009 rel2head
. by hhnr2009: generate egph = egp[1] if rel2head[1] == 1
```

You can even combine these two lines into one line by using the `by` construction shown below, where the sorting takes place for all variables in the `by` statement. However, when Stata forms the groups to which `generate` will apply, it will ignore the variables in parentheses.

```
. bysort hhnr2009 (rel2head): generate egphV2 = egp[1] if rel2head[1] == 1
```

This approach does not copy the contents of the last observation of each household to the other observations but copies the contents of the *first* observation. Using the `if` qualifier ensures that the social class of the first observation is indeed the class of the head of the household. This is important because some heads of household may not have been interviewed.

Generating variables with subscripts is often useful when you are working with panel or time-series data. However, for many panel or time-series transformations, Stata offers specific tools for managing your data (see `help xtset` and `help tsset`).

This section and the previous section were harder to understand than the other sections in this chapter. You may ask yourself if there is an easier solution, and the answer is often yes. Highly specialized commands for specific tasks exist. The following section shows you some of them. Unfortunately, these commands often do not help you to better understand the basic concepts of generating variables with Stata. If you cannot find a special command for a specific task, do not give up; instead remember that there might be a solution in Stata (also see Cox [2002b]).

5.2 Specialized recoding commands

5.2.1 The recode command

You will often need to combine several values of one variable into a single value. Earlier, we created the dummy variable `minor` to indicate respondents who had not reached the legal age of adulthood by the time the survey was conducted. Instead of using the procedure described on page 80, you could also use the command `recode`. You would type

```
. recode ybirth (min/1991 = 0) (1992/max = 1), generate(minor3)
```

With `recode`, you assign new values to certain observations of a new variable according to a coding rule. Using the `generate()` option stores the results in a new variable instead of overwriting `ybirth`. For example, above we assigned the value 0 to respondents born before or in 1991, and we assigned the value 1 to those born in 1992 or after. The result is stored in the new variable `minor3`. Here `min` refers to the lowest value of the variable `ybirth`, and `max` refers to the highest value.⁵ Missing values in this case do not count as the highest value.

⁵. To categorize variables into quartiles or other percentiles, use `xtile` or `pctile` (see `help pctile`).

If you wanted to create another variable with three categories, separating out respondents between age 18 and age 20, the `recode` command would look like this:

```
. recode ybirth (min/1987 = 0) (1988/1991 = 1) (1992/max = 2), generate(minor4)
```

Values specified in `recode` do not have to be in consecutive order.

The `recode` command also allows you to specify a variable list to be recoded, so you can recode several variables in one step. The following command generates copies of the variables on the appliances in dwellings but with code 2 changed to 0:

```
. recode eqp* (2=0), generate(head terr basm gard alrm solr airc lift engy)
```

5.2.2 The `egen` command

`egen` provides a large and constantly growing number of extensions to the `generate` command. These extensions are based on nothing more than one or more `generate` and `replace` commands. `egen` can be seen as a feature designed to spare you from having to think about more-complicated command sequences.

The structure of `egen` is similar to that of `generate`. The command is followed by a variable name (which is the name of the variable that should be generated), an equal-sign, and finally the `egen` function. Unlike the general Stata functions discussed in section 3.1.6, the `egen` functions are available only within the `egen` command.

A useful `egen` function is `anycount(varlist)`, which creates a variable containing the count of variables whose values are equal to any of the integer values in the `values()` option. Therefore, this `egen` function provides an easy way to form an index from several other variables. Look at the following example, which contains information on how concerned respondents are about several items:

```
. tab1 wor*
```

You may wish to form an index of respondents' concern about the future. One way to do this is to count the number of items that respondents said they were strongly concerned about, where "strongly" is coded as 1. You can do this easily by using the `egen` function `anycount()`:

see notes	Freq.	Percent	Cum.
0	1,016	18.78	18.78
1	453	8.37	27.15
2	521	9.63	36.78
3	692	12.79	49.57
4	641	11.85	61.41
5	524	9.68	71.10
6	408	7.54	78.64
7	367	6.78	85.42
8	266	4.92	90.33
9	190	3.51	93.85
10	148	2.74	96.58
11	89	1.64	98.23
12	71	1.31	99.54
13	25	0.46	100.00
Total	5,411	100.00	

You can compute the number of missing values for each respondent with the `egen` function `rowmiss(varlist)`:

```
. egen worried_m = rowmiss(wor*)
```

You can use the new variable `worried_m` in different ways, such as to display the table above only for those observations who have no missing values for any of the variables used in the `egen` command:

see notes	Freq.	Percent	Cum.
0	567	18.62	18.62
1	260	8.54	27.16
2	315	10.34	37.50
3	411	13.50	51.00
4	380	12.48	63.48
5	307	10.08	73.56
6	208	6.83	80.39
7	200	6.57	86.96
8	139	4.56	91.53
9	95	3.12	94.65
10	65	2.13	96.78
11	48	1.58	98.36
12	25	0.82	99.18
13	25	0.82	100.00
Total	3,045	100.00	

There are many other `egen` functions, a growing number of which have been created by users.⁶ If you have a specific problem, see the list of `egen` functions (`help egen`). Keep in mind that several `egen` functions are simple applications of `generate`—simple if you have learned how to look at problems from a Stata perspective.

5.3 Recoding string variables

The data manipulation tools you have learned so far can help you solve many problems. However, sometimes you will be confronted with variables that contain strings (letters or words), which require special attention.

Because our previous datasets did not contain strings, we will use a new dataset to provide examples. `mdb.dta` contains information about all German politicians who were members of the parliament (*Bundestag*) between 1949 and 1998 (des Deutschen Bundestages 1998). For each politician, the dataset contains his or her name, party affiliation, and the duration of his or her membership in parliament.

```
. use mdb, clear
```

String variables can be easily identified in the output of `describe`, where an entry in the column “storage type” beginning with “str” refers to a string variable:

<code>. describe</code>					
Contains data from <code>dta/mdb.dta</code>					
obs:	7,918 <th>vars:</th> <td>14</td> <th>size:</th> <td>934,324</td>	vars:	14	size:	934,324
				MoP, Germany 1949–1998 16 Jul 2012 11:39 (_dta has notes)	
variable	storage type	display format	value label	variable label	
index	int	%8.0g		Index-Number for Parliamentarian	
name	str63	%63s		Name of Parliamentarian	
party	str10	%10s		Fraction-Membership	
period	str2	%9s		Legislative Period	
pstart	int	%d		Start of Legislative Period	
pend	int	%d		End of Legislative Period	
constituency	str4	%9s		Voted in Constituency/Country Party Ticket	
birthyear	int	%9.0g		Year of Birth	
birthmonth	byte	%9.0g		Month of Birth	
birthday	byte	%9.0g		Day of Birth	
deathdate	int	%d		Date of Death	
begindate	str15	%15s		Begin of Episode	
enddate	str11	%11s		End of Episode	
endtyp	byte	%25.0g	endtyp	Reason for Leaving the Parliament	

Sorted by: pstart name

6. Chapter 13 explains where you can find those functions.

Our dataset contains the following string variables: `name`, `party`, `period`, `constituency`, `begindate`, and `enddate`. String variables can be tabulated just like the already familiar numeric variables but they cannot be summarized:

. summarize party						
Variable	Obs	Mean	Std. Dev.	Min	Max	
party	0					

Because you cannot do calculations with string variables, you may want to change them to numeric variables. You can do this by using one of two methods. First, you can use `encode` to generate a numeric variable with value labels according to the contents of a string variable. After typing the command, type the name of the string variable to be converted and then type the name of the numeric variable you want to create within the parentheses of the `generate()` option:

```
. encode party, generate(party_n)
```

`encode` attaches the value 1 to the category that comes first alphabetically, which is not always desirable. Specifically, you should not use `encode` for strings containing numbers that merely happen to be stored as strings. This is the case for the variable `period`, which contains numerals from 1 to 13; however, because they are stored in a string variable, Stata does not know that these numerals represent numbers. String variables like `period` should be converted to values with the command `destring`, which mirrors the syntax of `encode`:

```
. destring period, generate(period_r)
```

You may not always want to convert strings to numeric variables, because many data-management tasks are easier done with string variables. You can use many of the tools you have learned about with string variables. For example, to construct a variable that is 1 for all politicians who are members of the Christian Democratic Union and 0 for all others, type

```
. generate cdu = party == "CDU" if !missing(party)
```

That is, you can use *strings* in expressions if you enclose them in double quotes. However, be careful. Unlike numbers, strings can be lowercase and uppercase. And although we humans tend to be sloppy about whether we use uppercase or lowercase letters, Stata is not. If your strings contain the same information but mix lowercase and uppercase, you should harmonize them by using either the `lower()` function or the `upper()` function. The following example uses `upper()` to convert the contents of `party` on the fly, when constructing a variable that is 1 for all politicians who are members of the Social Democratic Party (SPD):

```
. generate spd = upper(party) == "SPD" if !missing(party)
```

An important string function is `strpos(s1, s2)`, which checks whether one string, *s2*, is part of another string, *s1*, and returns the position where *s2* is first found in *s1*. For example, using `strpos()` you can find out that the string “Stata” contains the string “a” for the first time at the third position:

```
. display strpos("Stata","a")
3
```

Now take a look at the variable `name`:

```
. list name in 1/5
```

	name
1.	Adenauer, Dr. Konrad
2.	Agatz, Willi
3.	Ahrens, Adolf
4.	Albers, Johannes
5.	Albertz, Luise

As expected, the variable contains the names of all the politicians. If you go through the list of names, you will notice that some politicians have a doctoral degree. You can easily extract this small piece of information about the politicians’ education by using the `strpos()` function: just check whether the name of a politician contains the string “Dr.”:

```
. generate dr = strpos(name,"Dr.") > 0
. tabulate dr
      dr |   Freq.    Percent      Cum.
      ---+-----+-----+-----+
        0 | 5,598     70.70     70.70
        1 | 2,320     29.30    100.00
      Total | 7,918    100.00
```

The `strpos()` function is also an important tool if you need to extract a specific part of a string. Suppose that you want to construct a variable that contains only the last names of the parliamentarians. In the variable `name`, first and last names are separated by a comma. The last name of each parliamentarian begins with the first letter and ends one character before the comma. Therefore, the first step in extracting the last name is to know the position of the comma. By typing

```
. generate comma = strpos(name,",")
```

you store this information in the variable `comma`.

Now you need to extract the substring of the variable `name` from the first position up to the position in the variable `comma`. You can do this using the `substr()` function. Inside the parentheses of this function are three arguments, separated by commas: the string from which the substring should be extracted, the position where the substring

begins, and the length of the substring. For example, to extract a string of length 4 beginning at the fifth position from the string “Hello world”, type

```
. display substr("Hello world",5,4)
o wo
```

To extract the last two characters of a string, you can count backward by using a negative second argument:

```
. display substr("Hello world",-2,.)
ld
```

That is, negative values for the second argument are used to indicate that the beginning position is counted from the end of the string. A dot in the third argument is used to extract everything from the beginning position to the end of the string.

To extract the last names, you need to extract a substring of the string in the variable `name` from the first position to the position of the comma, which is stored in the variable `comma`. The length of that substring must therefore be the position of the comma minus 1. Hence,

```
. generate str fname = substr(name,1,comma-1)
. list fname in 1/5
```

fname	
1.	Adenauer
2.	Agatz
3.	Ahrens
4.	Albers
5.	Albertz

returns the required result.

Clearly, you can use the same approach to extract the part of the name after the comma:

```
. generate str fname = substr(name,comma+1,.)
. list fname in 1/5
```

However, to extract just the first names of the politicians requires more work. In addition to the first names, you will find doctoral degrees and titles of nobility after the comma. To get rid of these suffixes, you can use the `subinstr()` function to replace a substring of an arbitrary string with another string. In the following example, we use this function to replace the string “Dr.” in the string variable `firstname` with an empty string—that is, we delete the string “Dr.” from the strings in `firstname`.

```
. replace firstname = subinstr(firstname,"Dr.","","",.)
```

The first argument within the parentheses is the string or variable that contains the substring to be replaced, the second argument is the substring to be replaced, and the

third argument is the replacement string. The fourth argument indicates the number of occurrences to replace; missing means all occurrences.

5.4 Recoding date and time

Besides string variables, variables holding information about date and time of day are a second group of variables that requires special attention. We talk about dates when there is information on the day, month, and year of the respective event. Similarly, we talk about time when there is, at minimum, information on the hour and the minute of an event.

Dealing with dates and times requires some steps that are quite different from what we discussed so far. We will explain both, starting with dates.

5.4.1 Dates

The file `mdb.dta` on parliamentarians uses information on dates heavily. Along with each person's name, our dataset contains information on each person's date of birth (and death), the date he or she became a member of the parliament, and the date he or she left the parliament. Here the various dates happen to be stored in different ways. The birth date is stored in three variables: `birthyear` holds the year in which the parliamentarian was born. Likewise, `birthmonth` holds the month of birth, and `birthday` holds the day of birth.

```
. list name birth* in 1/4
```

	name	birthyr	birthm-h	birthday
1.	Adenauer, Dr. Konrad	1876	1	5
2.	Agatz, Willi	1904	6	10
3.	Ahrens, Adolf	1879	9	17
4.	Albers, Johannes	1890	3	8

The dates each member entered and left parliament are stored as single string variables (`begindate` and `enddate`):

```
. list name begindate enddate in 1/4
```

	name	begindate	enddate
1.	Adenauer, Dr. Konrad	07 September 49	Sep 07 1953
2.	Agatz, Willi	07 September 49	Sep 07 1953
3.	Ahrens, Adolf	07 September 49	Sep 07 1953
4.	Albers, Johannes	07 September 49	Sep 07 1953

Suppose that you want to know the age at which politicians entered parliament. Generally, you would want to subtract the beginning date from the birth date. However, this is not as easy as it would seem, because you cannot subtract anything from a string

variable such as `begindate`. You might, therefore, decide to store only the year of the exit date and to subtract the beginning year from that variable. But this would not be very accurate, because it makes a difference whether the date is at the beginning or at the end of the beginning year.

When dealing with dates, you need to use *elapsed dates*, which are integer numbers counting the days from a standard date. If, for example, January 1, 1960, is used as day 0, then January 2, 1960, is day 1, and November 9, 1989, is day 10,905.

Elapsed dates are clearly preferable to the other ways of storing dates, so Stata has special functions to convert dates from other formats into elapsed dates. To convert dates stored in separate variables into elapsed dates, the function `mdy()` is used. Inside the parentheses, you list the names of the variables containing the month, the day, and the year. List the names in the order month, day, year, and separate the names with commas. This function returns the elapsed dates for the birth dates as follows:

```
. generate birthdate = mdy(birthmonth,birthday,birthyear)
. list name birth* in 1/4
```

	name	birthyr-r	birthm-h	birthday	birthd~e
1.	Adenauer, Dr. Konrad	1876	1	5	-30676
2.	Agatz, Willi	1904	6	10	-20293
3.	Ahrens, Adolf	1879	9	17	-29325
4.	Albers, Johannes	1890	3	8	-25500

In Stata, elapsed dates are counted from January 1, 1960, so all dates before then are negative numbers.

You may well be convinced that in Stata it is good to have elapsed dates. However, you may find elapsed dates to be cryptic. You might know something about, say, November 9, 1989—the date when the Berlin Wall fell—but the number 10,905 is inscrutable. Therefore, it would be helpful for Stata to store dates as elapsed dates but to show them in a way that humans can understand. You can do this by setting the display format of a variable containing elapsed dates to the Stata standard date format:

```
. format birthdate %td
```

You could even build fancier display formats by using the user-specified elapsed date formats described in `help dates and times`:

```
. format birthdate %tdM_d._CY
```

After you have set the display format, commands that respect display formats will show elapsed dates in a way that users can understand. Among the commands that respect display formats is `list`:

```
. list name birthdate in 1/4
```

	name	birthdate
1.	Adenauer, Dr. Konrad	January 5. 1876
2.	Agatz, Willi	June 10. 1904
3.	Ahrens, Adolf	September 17. 1879
4.	Albers, Johannes	March 8. 1890

To convert dates stored in string variables, you can use the `date(a,b)` function. Inside the parentheses, you first state a string or the name of the string variable containing the dates, and then, separated by a comma, you specify a *pattern string*, which describes the format of the dates in the string variable. If the string variable contains dates like “1989 November 9th”, the pattern string will be YMD because the order of the date is year (Y), month (M), day (D).

To understand the `date()` function, you should try it with `display`. We use `display` in connection with the format statement `%td` so that the returned elapsed date is shown in an easily readable format:

```
. display %td date("11/9/1989","MDY")
09nov1989
. display %td date("9/11/1989","DMY")
09nov1989
. display %td date("1989/11/9","YMD")
09nov1989
```

`date()` is tolerant about how the dates are stored in your string variable. The `date()` function understands the proper English words and usual abbreviations for the months, and does not care about the characters between the different parts of the dates. Here are some further examples, and you might try even more:

```
. display %td date("November 9 1989","MDY")
09nov1989
. display %td date("Nov9/1989","MDY")
09nov1989
. display %td date("11. 9/1989","MDY")
09nov1989
```

`date()` is also smart about two-digit years. If the years of the dates have only two digits and you know that those years are all from the same century, then you can put that information into the pattern string as follows:

```
. display %td date("November 9 `89","MD19Y")
09nov1989
```

The situation is more complicated for two-digit years referring to more than one century. In the special situation of values from two centuries, there is a solution if the century is then often clear from context. Think of the birth dates of the respondents of a survey from 2009 on a population aged 16 and older. The birth year '94 cannot be 1994 then, because the respondent would be only 15 years old. Hence, '94 must be 1894, or more generally, all birth years before and including '94 should be interpreted as belonging to the nineteenth century. It is therefore possible to add a third argument to `date(a, b, z)` to specify the maximum year that should be returned. Try this:

```
. display %td date("November 9 `92", "MDY", 1993)
09nov1992
. display %td date("November 9 `93", "MDY", 1993)
09nov1993
. display %td date("November 9 `94", "MDY", 1993)
09nov1894
. display %td date("November 9 `95", "MDY", 1993)
09nov1895
```

With all this in mind, we can now use `date()` to transform `begindate` and `enddate` into an elapsed date format. The entries in `begindate` are in the order day, month, year, where year is two digits long. From the data source, we know that there should be no year greater than 1999, so we use

```
. generate begin = date(begindate, "DMY", 1999)
```

The variable `enddate` has the order month, day, year, where year is four digits long. Hence,

```
. generate end = date(enddate, "MDY")
```

which leads to

```
. format begin end %tdM_d._CY
. list name begin end in 1/4
```

	name	begin	end
1.	Adenauer, Dr. Konrad	September 7. 1949	September 7. 1953
2.	Agatz, Willi	September 7. 1949	September 7. 1953
3.	Ahrens, Adolf	September 7. 1949	September 7. 1953
4.	Albers, Johannes	September 7. 1949	September 7. 1953

After you convert the beginning and ending dates of each politician into elapsed dates, calculating the time span between the two dates is easy. You can calculate the age (in days) at which each politician became a member of parliament:

```
. generate entryage = begin - birthdate
```

These ages in days can be rescaled to ages in years through division by 365.25. In the following command, we additionally round downward to the closest integer by using the `floor()` function.

```
. replace entryage = floor(entryage/365.25)
```

In our dataset, there is one complication: politicians are listed more than once if they served in more than one legislative period. Regarding age of entry into parliament, it is therefore sensible to keep only the observations for the first entry. This is yet another job for recoding with `by`, `_n`, and `_N` (see sections 3.2.1 and 5.1.4).

```
. bysort name (period_r), sort: keep if _n==1
```

These are the names and ages of those politicians who were youngest when they were first elected into the German Bundestag:

```
. sort entryage
. list name entryage begin party in 1/5
```

	name	entryage	begin	party
1.	Berninger, Matthias	23	November 10. 1994	B90/GRU
2.	Nolte, Claudia	24	October 3. 1990	CDU
3.	Schoeler, Andreas von	24	December 13. 1972	FDP
4.	Bury, Hans Martin	24	December 20. 1990	SPD
5.	Homburger, Birgit	25	December 20. 1990	FDP

5.4.2 Time

In Stata, times are handled similarly to dates. To create elapsed times, milliseconds are counted since January 1, 1960 (this is Stata's standard; see page 99). A value of 1 on an elapsed time variable means 1 millisecond into January 1, 1960. November 9, 1989, at 6:53 PM—which is the time Günter Schabowski declared the opening of the border between East and West Germany—happened 942,432,780,000 milliseconds after January 1, 1960, at 12:00 AM.

Time diaries are a common example of datasets that include times. In time diaries, respondents are asked to note the start and end time of each activity within a day. For illustrative purposes, we use the time diary available in the file `diary.dta`. This dataset contains a (fictitious) schedule for one day in the life of a child.⁷

7. The fictitious day followed the questionnaire for children used in the Panel Study of Income Dynamics.

```
. use diary, clear
. list in 1/10
```

	activity	bhour	bmin	estring	location
1.	Sleeping	0	0	7:30	at home
2.	Getting up	7	30	7:40	at home
3.	Using the bathroom	7	40	7:45	at home
4.	Eating	7	45	8:15	at home
5.	On the way	8	45	9:05	in car
6.	In school	9	5	15:15	school
7.	Playing basketball	15	15	17:00	YMCA
8.	On the way	17	0	17:30	in car
9.	Watching TV	17	30	18:00	at home
10.	Eating	18	0	18:25	at home

The dataset contains two variables for the beginning of each activity: one variable captures the hour (`bhour`) and the other captures the minute (`bmin`). The end of an activity is stored in another variable as a string. We will show how to convert both start and end data into an elapsed time format.

Knowing that our fictitious child got up at 7:30 AM is not enough. To calculate the elapsed time (the number of milliseconds that have passed since midnight on January 1, 1960), we need one more piece of information: the date for which we recorded the activities of this child. Let us assume our data recorded activities on November 9, 1989.⁸ We can add a variable containing that information to the dataset by using one of the date functions we discussed in the previous section:

```
. generate date = date("9.11.1989","DMY")
```

Once the date is known, it is fairly easy to create elapsed times. A couple of functions allow you to do this (see `help dates`). For our example, we will use `dhms(d,h,m,s)`. Here you specify the day as an elapsed date (`d`), then the hour of the event (`h`), the minute (`m`), and finally the seconds (`s`) if you know them. The elapsed time variable `begin` for the start of each activity will be created as follows:

```
. generate double begin = dhms(date,bhour,bmin,0)
```

Let us explain because this command might not be intuitive. First, the keyword “double” ensures that enough storage space is reserved for the elapsed time variable. Remember that times are stored in milliseconds; thus they often contain large numbers. To avoid rounding errors, you must save those time variables with a special storage type (`double`). You can find more on storage types in section 5.7.

8. This is an arbitrary date that we picked only for this example.

Second, you might notice that we added 0 at the very end of the command. Our data did not contain a variable with information on the second the activity started; thus we chose to enter 0 instead. Finally, we were fortunate to have the date already stored as an elapsed date variable. However, in the absence of an elapsed date you could use the function `mdyhm()`. Within the parentheses, you would enter month, day, year, hour, minute, seconds.

To convert the end time of the activities into an elapsed time variable, we need to use a different function. As mentioned above, in our example dataset the end time is stored as a string variable. To convert string variables into elapsed time, you would use the function `clock()`. Its functionality is similar to `date()`. Within the parentheses, you first list the string variable, which contains the entire time point (that is, date and time). Then you give information on the structure of this string variable.

The string variable in our example dataset does not yet contain a variable with all information for the time point. Before we can use `clock()`, we need to add a date to our time information.

```
. generate efull = "9.11.1989 " + estrng
. generate double end = clock(efull,"DMYhm")
```

The key `DMYhm` indicates that the newly created string variable contains day, month, year, hour, and minutes, respectively. Other options can be found in `help dates`.

It is reasonable to define a display format for your time variable, just like you did earlier for dates. The standard format for times is `tc`,

```
. format begin end %tc
```

Once elapsed times are created, we can find out which activity in the life of our fictitious child took the most time. We calculate the difference between the start and end times. The function `minutes()` allows users to express the difference between the start and end times in minutes.

```
. generate time = minutes(end-begin)
```

Afterward, you proceed as usual:

```
. by activity, sort: replace time = sum(time)
(5 real changes made)
. by activity: keep if _n == _N
(5 observations deleted)
. sort time
. list activity time
```

	activity	time
1.	Getting up	10
2.	Listening to bedtime story	20
3.	Playing computer games	30
4.	Reading book from library	35
5.	On the way	50
6.	Eating	55
7.	Watching TV	60
8.	Using the bathroom	65
9.	Playing basketball	105
10.	In school	370
11.	Sleeping	450

Now we know that the life of our child is dominated by sleeping and school.

5.5 Setting missing values

Before reading on, please load `data1.dta`:

```
. use data1, clear
```

Variables in Stata may contain missing values, which can arise for a number of reasons. For example, in earlier examples in this book, we created variables where we intentionally left certain observations out, or we changed existing variables to exclude certain values. With `generate` and `replace`, this is done by specifying a dot behind the equal sign. For example, by typing

```
. replace income = . if income == 0
```

you change all occurrences of 0 in the `income` variable to a *missing value*.

In addition to the dot, there are 26 other codes for missing values, namely, `.a`, `.b`, ..., `.z`. Observations with these codes are also excluded from statistical analyses. These codes are used to distinguish between different reasons for values being missing. For example, many surveys distinguish between the answer “do not know” and an explicit refusal to answer. In this case, it makes sense to code “do not know” as `.a` and explicit refusal to answer as `.b`. This way, both types of missing data are excluded from statistical analyses, but the information on the different causes is kept in the dataset.

You can assign the special codes for missing values the same way you assign the single dot. For example, we could assign incredibly low individual labor earnings to yet another missing value code, say, `.c` for “inconsistent”:

```
. replace income = .c if income <= 120
```

Often you may want to change multiple instances of the same value to missing in a dataset. For example, in the original GSOEP data, refusals to answer are coded as `-1`. In data analysis, refusals are usually treated as missing values. Therefore, it makes sense to change `-1` in all variables to missing simultaneously. To do so, you can set up a `foreach` loop (see section 3.2.2) or use the `mvdecode` command:

```
. mvdecode _all, mv(-1=.a)
```

`mvdecode` replaces the values of a *varlist* according to the rule specified in the `mv()` option. The rule can be read as, “Make the value before the equal-sign equal to the missing-value code after the equal-sign”. If you do not specify a missing-value code, the simple dot is used. In our example, we used `_all` instead of a variable list; `_all` executes the command on all variables in the data file. However, our example does not change the data, because we have already set `-1` to missing. Before you use `_all`, you should verify that there are no variables in your dataset for which `-1` is a valid value. In our case, we know no variables include `-1` as a value to be used in statistical computations.

Of course, you can change more values to missing. In the GSOEP, the response “does not apply” is always coded as `-2`, which means, “This question has not been asked of this respondent”. Usually, you would want to change this value to missing, as well:

```
. mvdecode _all, mv(-2=.b)
```

Missing values can also arise unintentionally. This happens when you create values with a function that is not defined for some values. For example, the command

```
. generate dvisits_log = log(dvisits)
(1595 missing values generated)
```

creates 1,595 missing values. Of these missing values, 98 were already present in `dvisit`, but all the other missing values originated because the logarithm of 0 is not defined.

Another frequent source of unintended missing values is ill-specified `if` qualifiers. The commands

```
. generate minor = 0 if (2009-ybirth) > 18
. replace minor = 1 if (2009-ybirth) < 18
```

are an attempt to create a dummy variable to distinguish between respondents above and below the age of 18, but unintentionally set respondents of age 18 to the missing value. Moreover, if there were already any missing values in `ybirth`, those values would have been assigned to 0 because for them the expression `2009-ybirth` evaluates to missing, which is a value greater than 18 (see section 3.1.5). The commands

```
. drop minor
. generate minor = (2009-ybirth) < 18 if !missing(ybirth)
```

are a better way to create the dummy variable `minor`.

Sometimes you might want to restore a missing definition. You can do this using `replace` or `mvencode`. `mvencode` has the same syntax as `mvdecode` but performs the opposite function. For example, by typing

```
. replace income = 0 if income == .
```

you change the missing value of `income` to the numeric value 0. By typing

```
. mvencode _all, mv(.a=-1)
```

you translate the missing value `.a` of all variables to `-1`.

These examples also show how problematic assigning missing-value codes can be. In the beginning, we have assigned all incomes of 0 to the missing value. This assignment is irreversible in that there is no way to distinguish the missing values that were already present from those newly assigned. Such problems arise often in assigning missing values to the data, although they also can occur in recoding variables.

To avoid this problem, we recommend that you duplicate the variables and apply the missing-value statements only to the copies. And remember, you are always in luck if you work with do-files in the way we suggested in chapter 2.

5.6 Labels

You have not finished generating a variable until you label it. Labels identify the contents of a variable. There are three main ways to label a variable: the *variable name*, the *variable label*, and the *value labels*.⁹

The variable name is not a label in the strictest sense. It is the name you give the new variable in `generate`. You can use variable names up to 32 characters long, but you typically will not use variable names of that length. Often you will need to type the names repeatedly, which will be tedious if the names are long.

9. There is one more concept in Stata that is less widely used—that of *notes*. We have used notes in `data1.dta` to store the variable name and the record type of the variable in the GSOEP database. You can look at our notes by typing `notes`. See `help notes` for more information.

There are two approaches used for choosing variable names. Often you will find datasets with *logical* variable names, where a logical key is used to name the variables. You will quite often find variable names composed of the letter `v` and the number of a question in a survey. Take the variable `v1` for example. This variable has the answers to the first question in our survey. The variable `v2` would contain the answers to the second question, and so on. In the GSOEP, logical variable names are used to express the year of the survey, the survey type, and the question number (see section 11.4.1). However, in a day-to-day data analysis, descriptive variable names can be quite helpful. Descriptive names directly indicate the contents of the variable, for example, `state` in our example dataset.

We recommend that you use logical variable names when entering data from a questionnaire (see chapter 11) but that you use descriptive variable names when preparing variables for a specific analysis.

Short variable names may not describe the contents of a variable clearly. To provide more information, you can use the variable label. For examples of variable labels, look at the right column of the output of `describe`. You also can find variable labels in the output of some statistical procedures.

Variable labels serve no purpose in Stata other than for users to understand the contents of a variable. That said, leaving out variable labels can be very annoying to users, especially in a dataset with logical variable names.

To label a variable, value, or an entire dataset, you use the `label` command.¹⁰ If you want to use `label` to label a variable, you need to specify a keyword. Type `label variable`, then type the name of the variable you want to label, and follow this by the label itself. You can use up to 80 characters for the label.¹¹

To label the variable `dvisits_log` that we created above, you would use the following command:

```
. label variable dvisits_log "Log of number of doctoral visits"
```

You do not need the quotation marks for labels that do not have special characters—dashes, blanks, commas, etc. Some characters, such as letters with accents, are displayed properly only if you use a font that can display those characters. If you are concerned about portability, you may not want to use such characters.

10. For information about labeling datasets, see section 11.5.

11. See `help notes` if you wish to store in the dataset information about the variable beyond the limit of 80 characters; for example, you may want to store the wording of the survey question.

As stated, variable labels are intended to help the Stata user. The same is true of value labels, which indicate the meanings of the values of a variable. This is important for variables with nominal scaling. The numbers 1 and 2 in the variable for gender (`sex`) are meaningless until we know that 1 stands for “male” and 2 stands for “female”. To store such meanings in the dataset, we define value labels.

We again use the command `label`, but this time with two steps:

1. Define a label that contains the values and their meanings.
2. Attach the label to the variable.

Consider the variable `minor` generated on page 80. To label the values of `minor`, we first define the contents of the label by typing `label define`, followed by the name of the label we want to define (at most 32 characters) and then the definitions of the values. We do this last step by specifying a value and stating its meaning:

```
. label define minor_lb 0 "Adult" 1 "Minor"
```

Again quotation marks are needed only if the value labels contain special characters, but we recommend that you always use them. Also avoid using letters with accents and the like. You can use labels up to 32,000 characters long to specify the meaning of a value, but we doubt that labels that long are useful.¹²

Defining a value label has no effect until we attach the contents of the label to a variable. This is done with `label values varname labelname`:

```
. label values minor minor_lb
```

Now the values of the variable are connected to the definitions specified in the label. The output of statistical commands, such as `tabulate`, then presents the meaning of the numbers instead of the numbers themselves:

<code>. tabulate minor</code>				
minor	Freq.	Percent	Cum.	
Adult	5,345	98.78	98.78	
Minor	66	1.22	100.00	
Total	5,411	100.00		

This two-step process may seem unnecessarily complicated, but it has its advantages. The main advantage is that you can use the same value label to label the values of more than one variable. In our dataset, we have connected all variables indicating features of dwellings with the label `scale2`. This label defines 1 as meaning “yes” and 2 as meaning “no”. Whenever variables share the same values, we can use the value label `scale2` again.

12. For compatibility with older versions of Stata and other programs, you might want to restrict yourself to 80 characters. Always put the most significant part of a label into the first few characters.

There is an even easier way to link value labels and variables. Define the value label `yesno`:

```
. label define yesno 0 "no" 1 "yes"
```

As already described, you can link `yesno` with `label value` to any variable. However, for new variables you can use `generate`. To do so, add a colon and then the name of the value label after the name of the new variable. This way you can generate and label a variable in one command:

```
. generate married:yesno = mar==1 if !missing(mar)
. tabulate married
```

married	Freq.	Percent	Cum.
no	2,331	43.08	43.08
yes	3,080	56.92	100.00
Total	5,411	100.00	

Finally, you can use the command `label list` to get information about the contents of a value label. To look at the contents of the value label `yesno`, you would type

```
. label list yesno
yesno:
    0 no
    1 yes
```

If you do not specify a label name, `label list` returns a list of the contents of all the labels we have.

A helpful alternative to `label list` is `numlabel`, which allows you to include the value in a value label. For example, the value label can be “1 yes” instead of “yes”. This way, commands such as `tabulate` show both the label and its numeric value. The following commands make this change for any value labels in the dataset and show the effect for `married`.

```
. numlabel _all, add
. tabulate married
```

married	Freq.	Percent	Cum.
0. no	2,331	43.08	43.08
1. yes	3,080	56.92	100.00
Total	5,411	100.00	

These “numlabels” are particularly useful during the data-management tasks described in this chapter. However, for statistical analysis and presentation of results, numlabels are less useful and sometimes even disturbing. It is therefore also possible to remove them:

```
. numlabel _all, remove
```

Type `help numlabel` for more information.

5.7 Storage types, or the ghost in the machine

There is a frustrating problem that you will encounter sooner or later. Before we can explain this problem, we must make a short technical introduction. Stata distinguishes alphanumeric variables (*strings*) from numerical variables (*reals*). Strings contain letters and other characters (including numerals that are not used as numbers). Reals are numbers. For both types of variables, Stata distinguishes between different storage types:

```
. help datatypes
```

The storage types differ in the amount of memory they use. To store an observation in a variable of storage type `byte`, Stata uses exactly 1 byte of memory. For the storage type `double`, Stata uses 8 bytes. To save memory, you should use the most parsimonious storage type. You can provide a keyword with `generate` to specify the storage type of the new variable.¹³ If you do not specify a keyword, Stata defaults to using floats for numbers or a string of sufficient length to store the string you specify, so you will usually not need to care about storage types. However, there is one exception. To illustrate, try typing the following:

```
. generate x = .1
. list if x == .1
```

You would expect a list of all observations now, because the variable `x` is 0.1 for all observations. But what you got is nothing.

Here is what has happened: Stata stores numbers in binary form. Unfortunately, there is no exact binary representation of many floating-point numbers. Stored as floating-point variables, such numbers are precise up to about seven digits. The number 0.1 is stored as 0.1000000149011612 as a floating-point variable, for example. Likewise the number 1.2 is stored as 1.200000047683716. The problem therefore is that the variable `x` does not really contain 0.1. The variable contains instead 0.1000000149. On the other hand, when Stata calculates, it is always as precise as it can be. That is, Stata does calculations with a precision of about 16 digits. For this reason, Stata handles the number 0.1 in a calculation as 0.10000000000000014.... If you compare this 0.1 with the 0.1 stored as a `float`, as in the above `if` qualifier, the two numbers

¹³. The keywords `byte`, `int`, `long`, `float`, and `double` cannot be used as variable names for this reason (see section 5.1.1).

are not equal. To avoid this problem, you can store numeric variables as `double`. You can get around the problem we described by rounding the decimal in the `if` qualifier to `float` precision:

```
. list if x == float(.1)
```

Type `help data types` for more information.

5.8 Exercises

- Load a subset of the National Health and Nutrition Examination Study (NHANES) into Stata memory by typing

```
. webuse nhanes2, clear
```

- Create the variable `men` with value 0 for female observations and 1 for male observations. Label your variable with “Men y/n” and the values of your variables with “no” for 0 and “yes” for 1.
- Correspondingly, create fully labeled dummy variables (indicator variables coded with 0 and 1) for “being in excellent health” and “being 70 years of age or more”.
- Assuming that the formula to calculate the body mass index (BMI) is

$$\text{BMI} = \frac{\text{Weight in kg}}{\text{Height in m}^2} \quad (5.1)$$

create a fully labeled BMI variable for the NHANES data.

- Create a fully labeled version of BMI, where the values have been classified according to the following table:

Category	BMI
Underweight	< 18.5
Normal weight	18.5–24.9
Overweight	25.0–29.9
Obese	30.0–39.9
Severely obese	> 40

- Create a fully labeled version of BMI, where the values of BMI have been classified into four groups with approximately the same number of observations.
- Create a fully labeled version of BMI, where the values of BMI have been classified into three groups defined as follows: Group 1 contains observations with a BMI lower or equal to one standard deviation below the gender-specific average. Group 2 contains observations with values higher than group 1 but lower than one

standard deviation above the gender-specific average. Finally, group 3 contains observations with values above group 2. Please create this variable with the help of `egen`.

8. Create the variable of the last problem without using `egen`.
9. With the help of the command `egen`, create a variable that enumerates all possible covariate combinations formed by the variables `region` and `smsa`.
10. Create the variable of the last problem without using `egen`.
11. Create a variable that indicates for each respondent how many people were interviewed in the same strata where the respondent lives.

6 Creating and changing graphs

In modern data analysis, graphs play an increasingly important role. Unfortunately, some authors regard data analysis using graphs as disjoint from traditional *confirmatory* data analysis. In contrast to confirmatory data analysis, which presents and tests hypotheses, graph-based data analysis is often perceived as a technique solely for generating hypotheses and models. However, as Schnell (1994, 327–342) convincingly spells out, this division between exploratory and confirmatory data analysis is misleading. It seems more sensible for us to regard graphs as tools for data analysis in general, and maybe even primarily for hypothesis-driven data analysis. We therefore use many graphical features in the chapters on distributions (chapter 7) and regression and logistic regression (chapters 9 and 10). There we discuss how to use and interpret the different graphs.

To take full advantage of the applications shown in these later chapters and create your own graphs, you need to understand Stata's basic graphics capabilities, which we explain in this chapter. Naturally, we will not cover every possibility, but after reading this chapter, you should be able to understand the logic behind Stata's graphics features. We strongly recommend that you read [G-1] **graph intro**, the online help, and last but not least, the excellent book on Stata graphs by Mitchell (2012).

The examples in this chapter use single-person households with moderate rents from **data1.dta**. Therefore, please enter¹

```
. use data1 if hsize == 1 & rent < 2000, clear
```

6.1 A primer on graph syntax

The syntax of the graph commands is different from that of most other Stata commands. We will explain this briefly here, and we will go into more detail as we progress:

- A Stata command for creating graphs comprises two elements: the **graph** command and a graph type. Here, for example, **box** is the graph type:

```
. graph box rent
```

1. Make sure that your working directory is **c:\data\kk3**; see page 3.

- For the **twoway** graph type, a plottype must also be specified. Here is an example with the plottype **scatter**:

```
. graph twoway scatter rent size
```

For the **twoway** graph type, you can leave out **graph** to save typing. For the plot-types **scatter** and **line**, you can even leave out **twoway**. The following commands are therefore identical to the one given above:

```
. twoway scatter rent size
. scatter rent size
```

- The plottypes of the **twoway** graph type can be overlaid. Here is an example with **scatter** and **lfit**:

```
. graph twoway (scatter rent size) (lfit rent size)
```

Here both types are set in parentheses. However, you can also separate the plot-types with **||** as in the following example, where we also leave out **graph** and **twoway**:

```
. scatter rent size || lfit rent size
```

- Occasionally, you will find options such as **xlabel(#20, angle(90))** or **xscale(range(0 300) reverse alt)** in your graph commands. That is, options of the **graph** command can contain suboptions or a list of options.
- The overall look of a Stata graph is specified by a graph scheme, so changing the graph scheme can change the look of the graph considerably:

```
. set scheme economist
. scatter rent size
```

To obtain the same graphs as shown in the book, you must switch to the **s2mono** scheme:

```
. set scheme s2mono
```

6.2 Graph types

For the most part, the **graph** command is composed of the same building blocks used in other Stata commands. However, you must include a subcommand for the graph type. When creating statistical graphs, you must first decide on the graph type, after which you can use the options to design the graph.

6.2.1 Examples

Among others, the following graph types are available in Stata:

- Bar charts

```
. graph bar size, over(area1, label(angle(45))) title(bar chart)
```

- Pie charts

```
. graph pie, over(area1) title(pie chart)
```

- Dot charts

```
. graph dot (mean) size, over(area1) title(dot chart)
```

- Box-and-whisker plots (box plots)

```
. graph box size, over(area1, label(angle(45))) title(box-and-whisker plot)
```

- Twoway (graphs in a coordinate system). This graph type allows for various plottypes, such as scatterplots, function plots, and histograms.

```
. graph twoway scatter rent size, title(scatterplot)
```

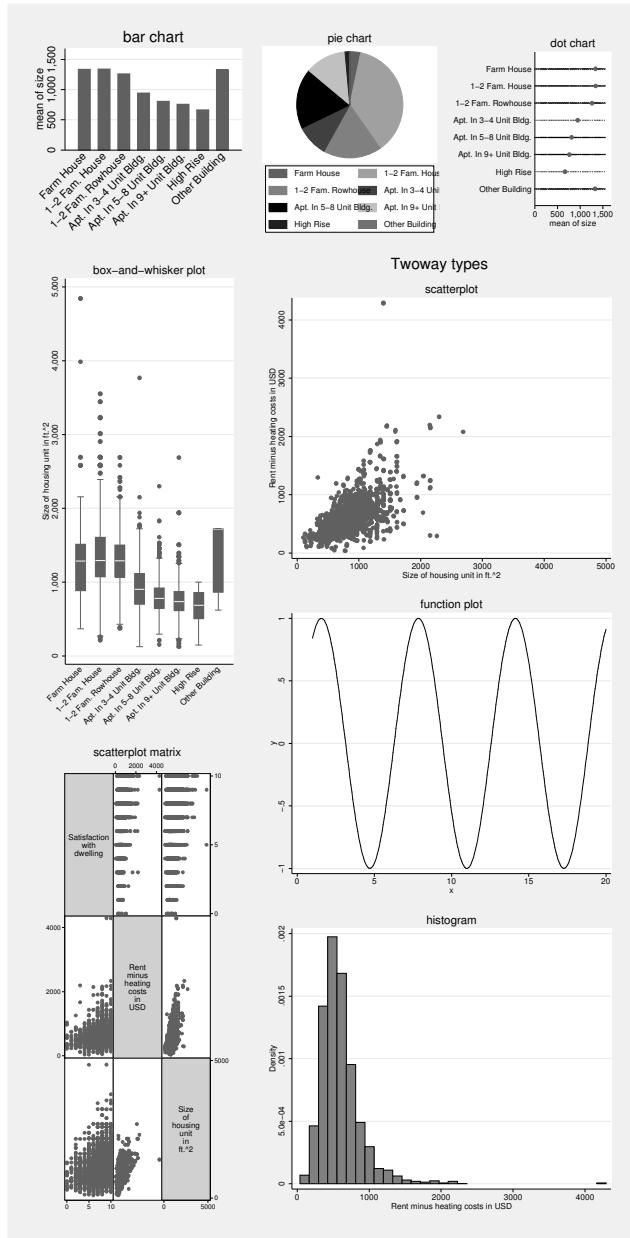
```
. graph twoway function y = sin(x), range(1 20) title(function plot)
```

```
. graph twoway histogram rent, title(histogram)
```

- Scatterplot matrices

```
. graph matrix dsat rent size, title(scatterplot matrix)
```

You will find examples for each of these graph types in figure 6.1.



Source: `gratypes.do`

Figure 6.1. Types of graphs

6.2.2 Specialized graphs

In addition to the basic graphs shown in figure 6.1, there are roughly 50 statistical graph commands, such as distributional diagnostic plots, plots designed for regression diagnostics, time-series graphs, and graphs for survival analysis. All of these graphs use the basic graph commands and require specific statistical interpretations or special data preparation. Some are only available immediately after statistical procedures, whereas others are designed to help you prepare a statistical analysis, such as

```
. gladder income
```

which displays how the variable `income` would be distributed after nine different power transformations (see section 9.4.3). For a list of all statistical graph commands, type `help graph other`; you will see several applications of different graph types in the rest of this book.

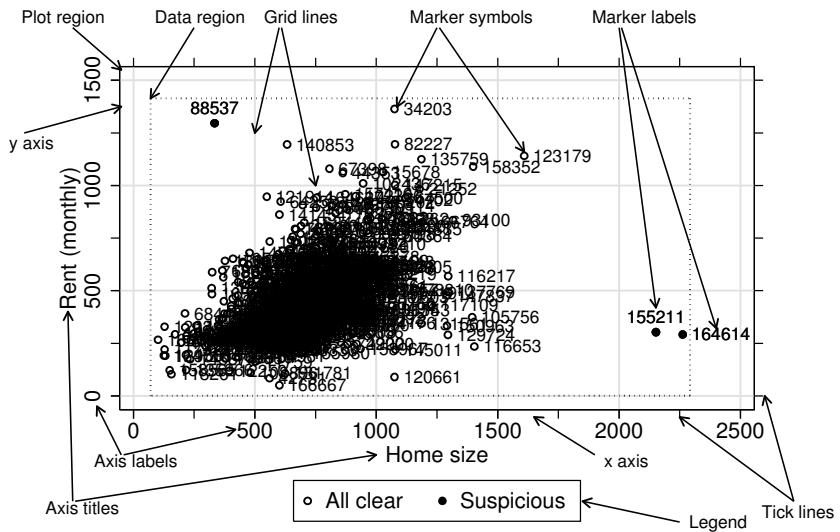
Some software packages include 3-D graphs, meaning illustrations that appear to be three-dimensional. In 3-D graphs, rectangles of a bar chart may be depicted as blocks, lines in line graphs as tapes or snakes, and the circular segments from pie charts as slices of pie or cake. Graphs such as these might be helpful in business presentations, but most of them are not suitable for presenting statistical data. Therefore, there are no such illustrations in Stata.

6.3 Graph elements

The appearance of graphs is defined by a series of elements, shown in figure 6.2 (see Cleveland [1994, 22–25]). In the rest of this chapter, we discuss these elements, which can be roughly subdivided as follows.

- Elements that control the display of data, including the shape, color, and size of the “marker symbols”, as well as lines, bars, and other ways to display data.
- Elements that control the size and shape of the graph, including the “graph region” and “plot region”. The graph region is the size of the entire graph, including titles, legends, and surrounding text. The plot region is the space that can hold data points. Just inside the plot region is the data region. The plot region’s size is determined by the axes, whereas the data region’s size is determined by the minimums and maximums of the variables being plotted.
- Elements that convey additional information within the graph region, including, for instance, reference lines for crucial values, marker symbol labels, or any other text in the plot region.
- Information outside the plot region, which affects the appearance of the axes that border the graph region on the left (*y* axis), bottom (*x* axis), top (upper axis), and right (right axis). The appearance of information outside the plot region is controlled by various elements, for example, tick lines, axis labels, and axis titles.

Often displayed here are a legend, which explains the symbols in the graph, and the title and description of the graph.



Source: grclleveland.do

Figure 6.2. Elements of graphs

You can change all of these graph components in various ways with two different tools:

- The options available for the `graph` command and the options available for each of the specialized graph or plottypes. This would be in line with the usual way of operating Stata through its command language.
- The Graph Editor. The Graph Editor allows you to change the graph using the mouse. You start the Graph Editor by selecting **File > Start Graph Editor** within an open Graph window. Likewise, you end the Graph Editor by selecting **File > Stop Graph Editor**. You cannot enter commands in the Command window while the Graph Editor is open.

Which of the two tools is the better one? It is difficult to give an answer that applies to all cases, but we generally prefer the first option, largely because of its possibility for replication. While the Graph Editor does have a feature known as the Graph Recorder to keep track of your edits, we prefer to store graph commands in a do-file. This makes re-creating graphs, perhaps with revised data, trivially easy.

On the other hand, we do acknowledge that there are layout choices that are difficult to achieve (if not impossible) using the command line. This is especially true for adding text within a graph. The seemingly easy task of adding an arrow with a label to a graph, for example, as seen in figure 6.2, requires considerable programming that only command line junkies will find reasonable.² For tasks of this kind, the Graph Editor is of tremendous help. We would recommend the following: See if you can create and edit your graph to your liking with the command line options. Use the Graph Editor only if you get stuck or the effort far exceeds the output.

Sticking to this principle, we will put heavy emphasis on command line options for the rest of the chapter. We will introduce the Graph Editor only in places where it seems particularly useful, such as in section 6.3.3, where we explain how to add text within the plot region.

One last note: Stata offers many design possibilities, but you should follow some basic rules and accepted standards in designing statistical graphs. An excellent summary of the design of statistical graphs in general can be found in Cleveland (1994, 23–118).

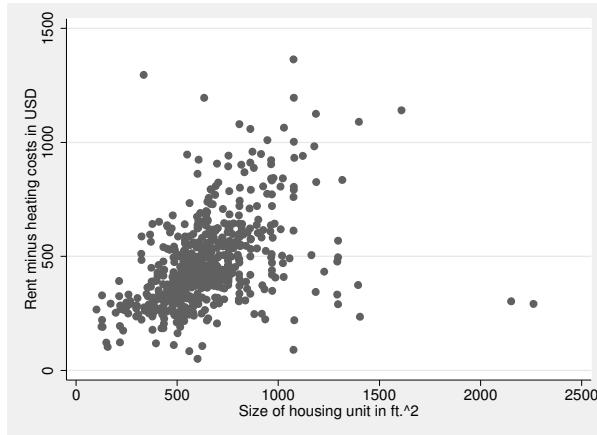
6.3.1 Appearance of data

When first constructing your graph, you specify how your data appear in the graph by indicating the basic graph and plottype. For instance, the `twoway line` plottype generates a line, the `twoway scatter` plottype generates round symbols, and the `bar` graph type generates vertical bars.

The `twoway` graph type has far more design possibilities, first of all because you specify a variable list. For twoway graphs, the *last* variable in a variable list always forms the *x* axis. All other variables form the *y* axis. For example, the following scatterplot uses a variable list with two variable names, `rent` and `size`. Therefore, `rent` (the variable named first) forms the *y* axis, and `size` (the variable named last) forms the *x* axis.

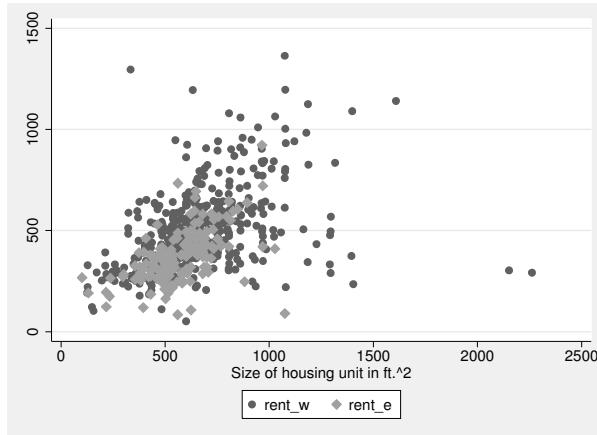
2. Should you count yourself in this group, take a look at the do-file `grcleveland.do`, which is part of our data package.

```
. scatter rent size
```



If you specify more than one y variable, the data points for the different y variables are marked with various symbols so that all data ranges are visible. For example, we can plot the rental rates of West and East Germany separately:

```
. generate rent_w = rent if state<= 9
. generate rent_e = rent if state>= 10 & !missing(state)
. scatter rent_w rent_e size
```



You can change the shape, size, and color of marker symbols, including the borders of the markers. All options affecting the markers begin with the letter `m` and contain a code word for the part of the marker to be changed. For example, the `msymbol()` option changes the marker symbol's shape, the `msize()` option changes its size, and the `mlabel()` option changes its label. The next section describes the most important marker options in more detail; for more information, see [G-3] **marker_options** or type `help marker_options`.

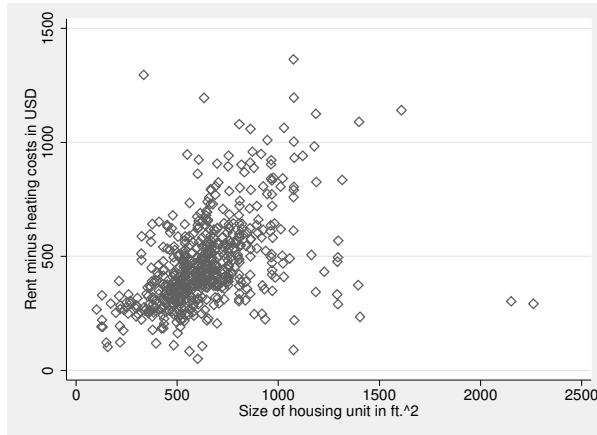
Choice of marker

The `msymbol` option specifies the symbol used to mark observations. You specify a shape inside the parentheses, which in many cases can be abbreviated to a single letter. Below are the basic shapes for the plot symbols and their one-letter abbreviations.

<code>circle</code>	<code>o</code>
<code>triangle</code>	<code>T</code>
<code>diamond</code>	<code>D</code>
<code>plus</code>	<code>+</code>
<code>x</code>	<code>X</code>
<code>point</code>	<code>p</code>
<code>none</code>	<code>i</code>

With the exception of `point` and `none`, these basic shapes can be further modified: for example, the letters `sm` before a basic shape transform a large symbol into a small one. The code word `smcircle` therefore indicates a small circle. Alternatively, you can specify a small circle using the lowercase letter `o`. Similarly, `t` and `d`, respectively, stand for `smtriangle` and `smdiamond`. Also you can add `_hollow` to the name to use hollow marker symbols. If you used code letters instead of the symbol names, an `h` is added to the other code letters; thus, `circle_hollow` or `0h` indicates a hollow circle. The following command results in a graph that uses a hollow diamond as a marker symbol instead of a circle:

```
. scatter rent size, msymbol(diamond_hollow)
```



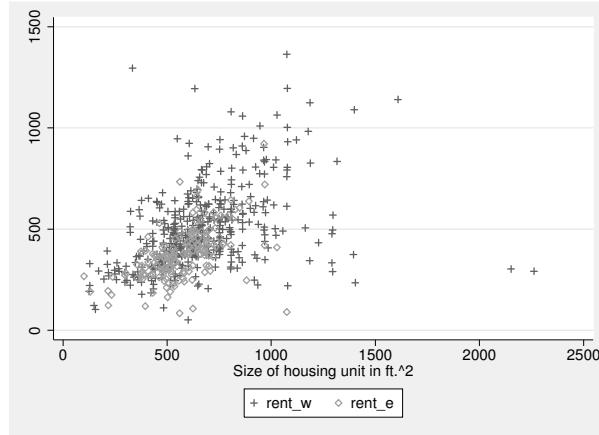
Depending on the Stata version you have, more marker symbols may be available. For a list of available marker symbols, type

```
. graph query symbolstyle
```

If you have designated many y variables in your graph command, the first y variable stands for the first series, the second y variable stands for the second series, and so on. In this case, Stata automatically uses different marker symbols for each series. To change the marker symbols for individual series, specify one letter for each series in the parentheses of `msymbol()`, separated by a space. The first code letter stands for the first y variable, the second letter for the second y variable, and so on. Typing

```
. scatter rent_w rent_e size, msymbol(+ dh)
```

assigns a large plus sign to the values for `rent_w` and a small hollow diamond for the values for `rent_e`:



Marker colors

You can change the color of the markers with the `mcolor()` option. The command for changing the color of the marker symbols is nearly the same as that for changing marker symbols. You simply enter the desired color for each data area in the parentheses. In the following example, we use a large lavender-colored triangle for the rent in the West and a large lime-colored plus sign for the rent in the East.

```
. scatter rent_w rent_e size, msymbol(T +) mcolor(lavender lime)
```

You can obtain a list of predefined colors and gray tones with `help colorstyle` or you can define your own colors. You can change a color's intensity by multiplying the color by a factor (for example, `mcolor(green*.8)`); you can also mix colors by specifying either RGB or CMYK values. To specify the RGB values, you type three numbers, standing for red, green, and blue, respectively, between 0 and 255.

```
. scatter rent_w rent_e size, msymbol(T +) mcolor(lavender "255 0 0")
```

To specify the CMYK values, you specify four numbers between 0 and 255, representing cyan, magenta, yellow, and black, respectively.³

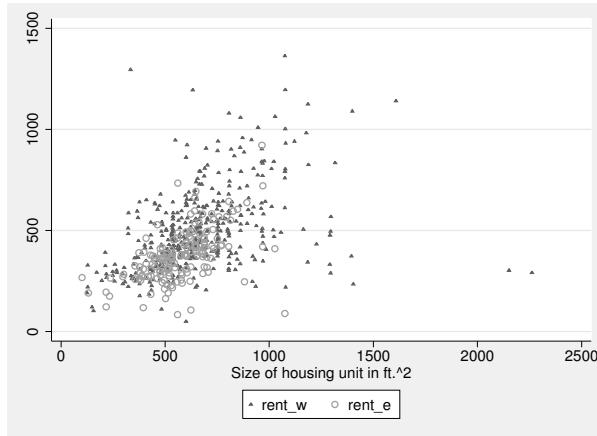
You can set a plot symbol's outline and fill colors separately. You can also set the color of the marker symbol fill using `mfcolor()` and outline using `mlcolor()`. You can change the outline's line thickness and line style; see the descriptions of the `mlstyle()`, `mpattern()`, and `mlwidth()` options under `help marker_options` or in [G-3] ***marker_options***.

3. According to Wikipedia, the K in CMYK stands for “key” because in four-color printing, cyan, magenta, and yellow printing plates are carefully aligned (or keyed) with the black “key plate”.

Marker size

In addition to using the `sm` preset marker sizes, you can more finely adjust the size of the marker symbol by using the `msize()` option. You can enter either absolute or relative sizes. An absolute size is one of 12 predefined sizes that range from `vtiny` (very tiny) to `ehuge` (extremely huge); see `help markersizestyle`. Relative sizes allow you to multiply existing symbol sizes by any number, for example, `msize(*1.5)` for a 1.5-fold increase in symbol size. You can also make the diameter of the symbol relative to the size of the graph. In the example below, we specify `msize(*.5 2)`. The triangle for rent in the West is thus half as large as it was displayed previously, and the circle for rent in the East has a diameter that is 2% of the height of the graph.

```
. scatter rent_w rent_e size, msymbol(th oh) msize(*.5 2)
```



Lines

With line graphs, instead of having one marker symbol for each data point of the graph, the data points are connected with a line. Line graphs are often used in time series, so our example will use the `ka_temp.dta` file, which contains the average yearly temperatures in the German city of Karlsruhe for the period 1779–2004. However, before loading this dataset, please issue the command

```
. preserve
```

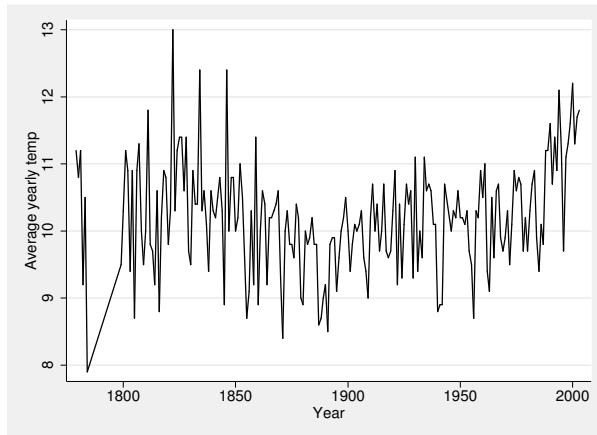
which stores the dataset as it is, in a temporary file. After preserving the dataset, you can safely try out whatever you want and then simply type `restore` to get your preserved data back.

Having preserved the dataset as it is right now, we can safely load our time-series dataset.

```
. use ka_temp, clear
```

For this line graph, we could use the `connect()` option with the `scatter` plottype. To do this, you must enter a line type in the parentheses of `connect()`. In the following example, we use the `direct` line type, which connects the points of a graph with a straight line. We will also specify the invisible marker symbol (`i`) and use the `sort` option—more on that later (page 128).

```
. scatter mean year, msymbol(i) connect(direct) sort
```



You can generate the same graph in a much easier way. The `line` plottype is a synonym for the scatterplot with the invisible marker symbol and the `connect(direct)` option; thus, you can use the `line` command:

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

Accordingly, the plottype `connected` is a synonym for the scatterplot with the `connect(direct)` option and visible plot symbols. The following commands therefore generate the same graph:

```
. scatter mean year, connect(direct) sort
. twoway connected mean year, sort
```

Regardless of how you generate your line graph, you can always specify many *y* variables. Using the `connect()` option, you can designate the line type for each *y* variable. You can also specify a line type that is analogous to the `msymbol()` option, which again can also be abbreviated. Here we provide an overview of the available line types; for a complete list, type `graph query connectstyle`.

`direct` or `l` connects successive data points with a straight line.

`ascending` or `L` connects successive data points with straight lines if the *x* variable shows increasing values.

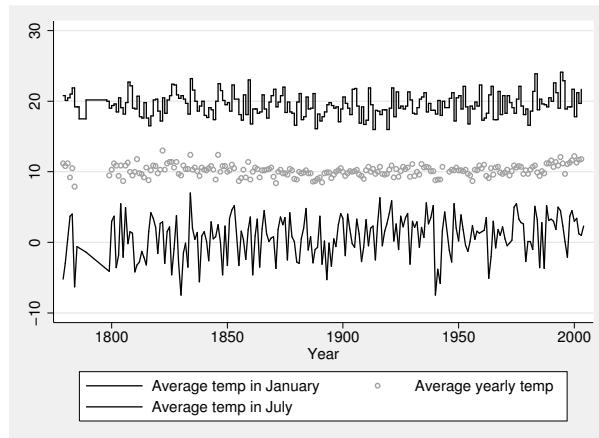
`stairstep` or `J` connects successive data points with a steplike line, which is first horizontal and then vertical.

`stepstair` connects successive data points with a steplike line, which is first vertical and then horizontal.

`none` or `i` means that no line should be drawn (invisible).

The following graph shows examples of these two line types and the invisible line. We will generate a graph where the average temperatures for July are connected with a steplike line and the average temperatures in January use a straight line. The average yearly temperatures will remain unconnected:

```
. scatter jan mean jul year, connect(l i J) msymbol(i oh i) sort
```



In addition to changing the line type, you can also modify the lines' color, thickness, and pattern; for details, type `help connect_options`. Mostly, these options work the same way as the equivalent options for marker symbols. We strongly recommend that you read [G-2] **graph twoway line** to better understand these options.

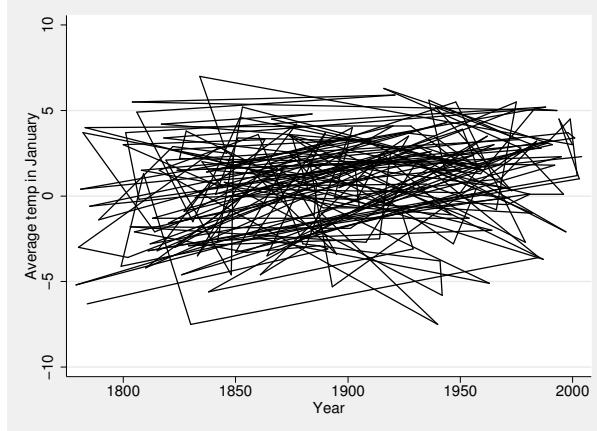
Why sort?

We have used the `sort` option in all the line graphs. It is not necessary, but we have good reason for doing so. All line types connect successive data points *in the dataset*, not successive along the *x* axis. This feature occasionally leads to complications. To demonstrate this, try sorting the observations in the dataset by the average yearly temperature:

```
. sort mean
```

You should then generate the following graph:

```
. line jan year
```



This is not the type of graph that you would associate with a time series. Yet in principle, this is the same process we used in generating the graphs earlier. After all, the data points are being connected with straight lines! However, the points are connected in the order in which they appear in the dataset; that is, the line goes from the first data point in the dataset to the second data point in the dataset. However, given the way that the data are sorted, the first point in the dataset is the one with the lowest average yearly temperature and not the one with the smallest value on the x variable.

In time-series plots, and indeed in most other graphs where the points should be connected with lines, the points should be connected in the order of the values of the x axis. You can do this by doing a preliminary sorting of the data or by using the `sort` option. We use the `sort` option, but this is a matter of taste. However, regardless of your method, when you generate a line graph, make sure you have adequately sorted the data. Also make sure that you can reproduce the graph regardless of any inadvertent resorting of the original data.

Before you continue reading, you should reload your original dataset using the command

```
. restore
```

6.3.2 Graph and plot regions

The conclusions drawn from the content of a graph are often affected by the design of the data region. For example, the smaller the data region is in proportion to the graph region, the larger any correlations in a scatterplot will appear. The smaller the relationship between the height and width of a graph region, the weaker any changes

over time will appear, and so on. Therefore, you should design the data region carefully. Pay specific attention to these three elements: the aspect-ratio of the graph region (that is, the relationship between the height and width of the graph), the margin around the plot region, and the scale of the two axes.

Graph size

You control the overall size of the graph by using the options `xsize()` and `ysize()`, with the height and width of the graph (including all labels and the title) in inches specified in the parentheses. You normally will not change the graph size, because it is better to do this when printing it or importing it into a presentation or desktop publishing program. However, the options are of interest because they allow for changes in the ratio between height and width (the aspect ratio) and therefore in the overall design of the graph and plot regions.

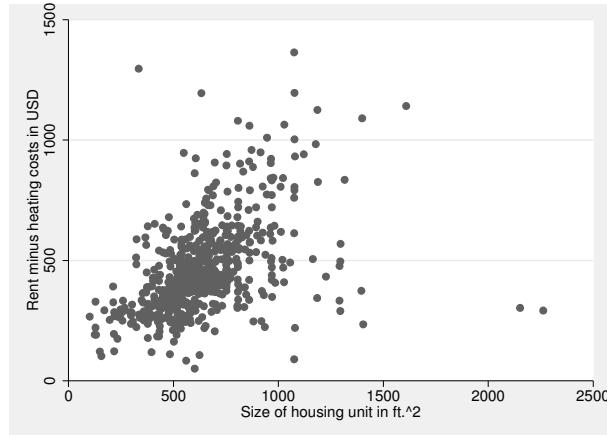
The default size of a graph in scheme `s2color` is 4×5.5 , that is, 4 inches high and 5.5 inches wide (1 inch \sim 2.5 cm). If you change only the width or the height, the design of the graph region and the plot region will also change. Here are two sufficiently extreme examples:

```
. scatter rent size, xsize(1)
. scatter rent size, ysize(1)
```

Plot region

In the scheme currently available for the graphs, the data region is slightly smaller than the plot region; that is, there is a small margin between the smallest and largest data points and their allocated axes. This is advantageous in that no data points are drawn exactly on the axis. You change the default setting using `plotregion()`, which you can use to change various plot region characteristics (such as color and shading; see [G-3] *region_options*). For example, to change the margin around the plot region, you would use the `margin()` suboption. You enter the margin size in the parentheses with a codeword, such as `tiny`, `small`, or `medium`. Here is an example that does not use a margin:

```
. scatter rent size, plotregion(margin(zero))
```

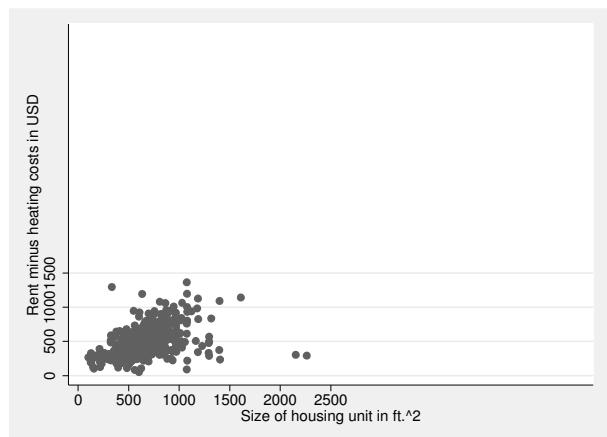


The larger the margin, the smaller is the plot region (relative to the graph region) and vice versa. For information about specifying the plot region margin, type `help marginstyle`.

Scaling the axes

Axis scaling affects the size and shape of the data region, so it may heavily influence the conclusions drawn from a graph. Within the options `yscale()` and `xscale()`, you can determine how axes are scaled (arithmetic, log, reversed), the range of the axes, and the look of the axis lines. To change the upper and lower boundaries of an axis, you use the `range()` suboption, specifying in the parentheses a number for the lower boundary followed by a second number for the upper boundary of the range of that axis.

```
. scatter rent size, xscale(range(0 5000)) yscale(range(0 5000))
```

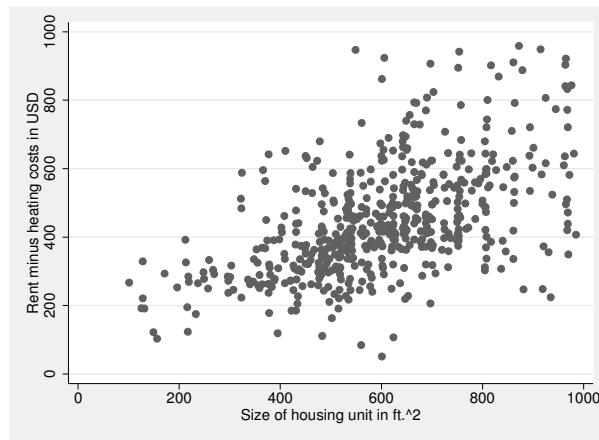


The lower the value of the lower boundary and the higher the value of the upper boundary, the smaller the data region will be in relation to the plot region. Cleveland (1994, 92–95) suggests that the data region take up as much of the plot region as possible, although many authors recommend always starting the axis at 0 (Huff 1954). Cleveland reasons that you can assess the extent of any differences between the values of the y variables more easily. When analyzing data, it is often helpful to have a graph that provides as much space as possible for the data region and therefore allows for an accurate reading of the individual values.

Stata's default setting takes Cleveland's suggestions into account. The lower boundary of each axis is the minimum of the respective variable that forms the axis. Correspondingly, the upper boundary of each axis is defined by the highest value of the variable it is based on. When changing the range of the axis, you can make the data region smaller than the default setting, but you cannot make it bigger. This means that you can have more white space between the data region and the axis, but you cannot cut out data points. If you specify a smaller axis upper boundary than the maximum of the variable this axis is based on, Stata will ignore the request and the graph will be generated with the default setting with no error messages. This feature ensures that all data points can always be drawn.

However, sometimes you may want to enlarge the data region beyond the default size. For example, when a certain area of a scatterplot is densely occupied, you may need to enlarge that area so that you can explore the relationships between observations in that part of the graph. Here you can reduce the data to be displayed by adding an `if` qualifier to the graph command.

```
. scatter rent size if size <= 1000 & rent <= 1000
```



You can also change the appearance of the plot region by transforming the variables to be plotted. After all, a transformation simply converts the unit of measure into a different unit. For example, temperatures in degrees Celsius are transformed into

degrees Fahrenheit by $F = C \times 1.8 + 32$. Therefore, you can always change the axis unit of your graphs by using a variable regenerated by the respective transformation (see chapter 5). You can also perform two standard unit transformations, taking the log and mirroring, by using the following suboptions in `xscale()` and `yscale()`:

- `xscale(log)` and `yscale(log)` for logarithmic scales.
- `xscale(reverse)` and `yscale(reverse)` to draw axes from the data maximum to the data minimum.

Also among the suboptions of `yscale()` and `xscale()` are some that do not affect the appearance of the displayed data. These include omitting the axes altogether and the placement of the axes; a complete list of these options can be found in `help axis_scale_options`. We will explain this in more detail in section 6.4.3.

6.3.3 Information inside the plot region

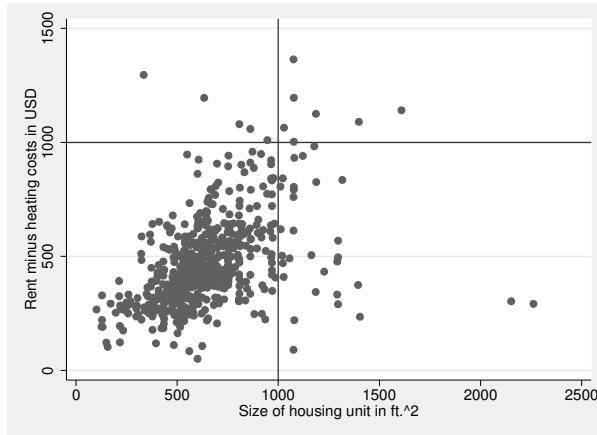
A glance at the graphs in any publication shows that the plot region is often used for information that goes well beyond the actual data. Quite often, this is merely “chart junk” (Tufte 2001, 107), that is, visual elements that often hinder the actual aim of the graph, namely, the depiction of data. Usually, it is sensible to limit the use of the graph region to displaying data. Nevertheless, you may find it useful to draw lines at important points and use individual labels, as long as you do this sparingly.

Reference lines

You can add reference lines to the plot by using the `xline()` and `yline()` options, typing in the parentheses a list of numbers (section 3.1.7) where the lines will be drawn.

The following command draws vertical and horizontal lines at the position 1000. Note the difference between the reference lines and the *grid lines* drawn along the y axis in the graph scheme `s2mono`. Grid lines are drawn at points where the axes are labeled. These grid lines are linked to the options for labeling axes and are discussed in further detail in section 6.3.4.

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



You can use reference lines to visualize internally saved results, such as the mean or median (see chapter 4). Here is an example of a horizontal line on the mean value of the monthly rents:

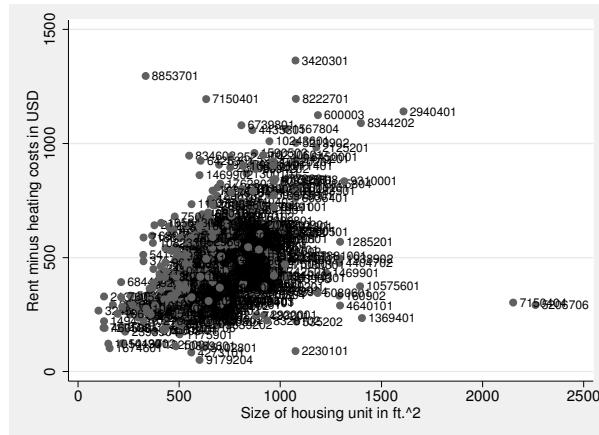
```
. summarize rent if !missing(rent)
. local rentmean = r(mean)
. scatter rent size, yline(`rentmean`)
```

Labeling inside the plot region

Use labels *inside* the plot region sparingly because they can easily be mistaken for data points, making it harder to interpret the data; they may also hide the actual data points. Information about the meaning of plotted data should usually be provided outside the plot region, for example, with a legend. Sometimes it can be useful to mark data inside the plot region, for example, when you want to label marker symbols with the content of a variable or place text at a given x - y coordinate. Let us begin by labeling the marker symbols.

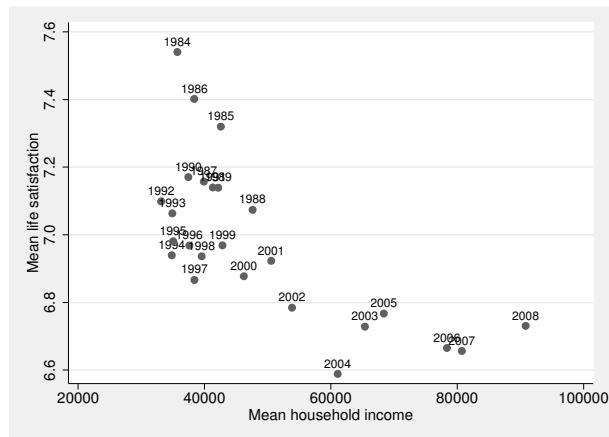
To label the markers, we use the `mlabel()` option with a variable inside the parentheses, which will display the content of the variable next to the marker symbol for each observation. In the following example, the markers are labeled with the individual person's ID.

```
. scatter rent size, mlabel(persnr)
```



Even with moderate sample sizes, labeling markers usually means that labels are placed on top of each other and on top of the data points, making the graph illegible. In data analysis, it is often sufficient to be able to read the labels for observations that lie far apart. For presentations, however, you should use marker labels only with small sample sizes, such as with aggregate data. The following example illustrates this. In this scatterplot, the average values for “life satisfaction” (`lsat`) are plotted against average values for “annual income” (`inc`) for 26 survey years from the German Socio-Economic Panel. Here we use the survey year as the marker label:

```
. preserve
. use data2agg, clear
. scatter lsat hhinc, mlabel(wave) mlabposition(12) mlabsize(small)
```



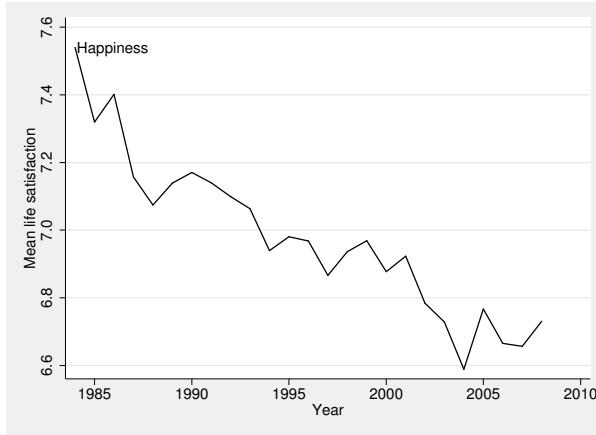
Note especially the two options `mlabposition()` and `mlabsize()`. The `mlabsize()` option allows us to adjust the size of the marker label. You can enter the same arguments in the parentheses that you would for the `msize()` option described above (see section 6.3.1). The `mlabposition()` option allows us to change the position of the marker label. The number entered in the parentheses is called the “clock position”. If you think of the space surrounding a marker as the face of a clock, the number entered reflects the given position on the face of the clock. The number 12 is equivalent to the position directly above the marker symbol; the number 6 is directly below the marker symbol; and so on.

Unfortunately, some of the values are so close together that the annual figures are slightly in each other’s way (here 1999 collides with 1998). In such graphs, changing the position of the marker label for various symbols is often unavoidable. Here we use the `generate` command and two `replace` statements to build the variable `pos` that indicates the position of the individual labels. Again we use clock position. The newly created variable is then specified in the `mlabvposition()` option.

```
. generate pos = 12
. replace pos = 3 if inlist(wave,1989,1998,2007)
. replace pos = 7 if inlist(wave,1991)
. replace pos = 9 if inlist(wave,1990,1993,1994,1995,1997,2006)
. scatter lsat hhinc, mlabel(wave) mlabvposition(pos)
```

You can also place labels inside the plot region by using the `text()` option. The `text()` option allows you to enter text at any x - y coordinate in the plot region. This is useful if, for instance, you wish to label a line graph directly inside the plot region.

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



This adds the label at $x = 1984$ and $y =$ the value of `lsat` for the first observation. The `placement()` option tells Stata how to orient the text relative to the point you specify; see [G-3] **added_text_options** or `help added_text_options`.

Using the Graph Editor

It is much easier to place text within the plot region with the help of the Graph Editor instead of command options. To start the Graph Editor, select **File > Start Graph Editor** from the menu bar within the Graph window. On the right side, you will see the Object Browser, which allows you to select single elements in the graph. On the left side of the graph, you see a vertical menu bar containing tools that you can use to change the graph. The first tool with the little arrow pointer is automatically activated when you launch the Graph Editor. It is used to mark and move objects.

To get a sense of the different tools, click on each of them one by one. You will see that the gray area above the graph changes each time you select a new tool. Each tool is connected to the Contextual Toolbar where you can further specify the functionality of the tools.

To add text, use the Add Text tool, which is marked with a capital **T**. After selecting the tool (with a mouse click), you can click on any position within the graph. This brings up a new window in which you can enter the text that should be added to the graph. Clicking on **Apply** allows you to examine the change that will be applied to the graph without closing the dialog.

Drawing lines and arrows with the Graph Editor is just as easy. You can use the third tool in the vertical menu bar, indicated by a diagonal line. After selecting this tool, you can use the context menu to decide on the characteristic of the line you are about to draw. You can, for example, decide on the width of the line or choose whether to have an arrowhead on one or both ends of the line. Once you specify the characteristics, you can click on the graph at the position where you want to start the line and drag your cursor to its endpoint. When you let go of the mouse, the line will be part of the graph.

Figure 6.3 shows the graph window of the Graph Editor after we used the tools described.

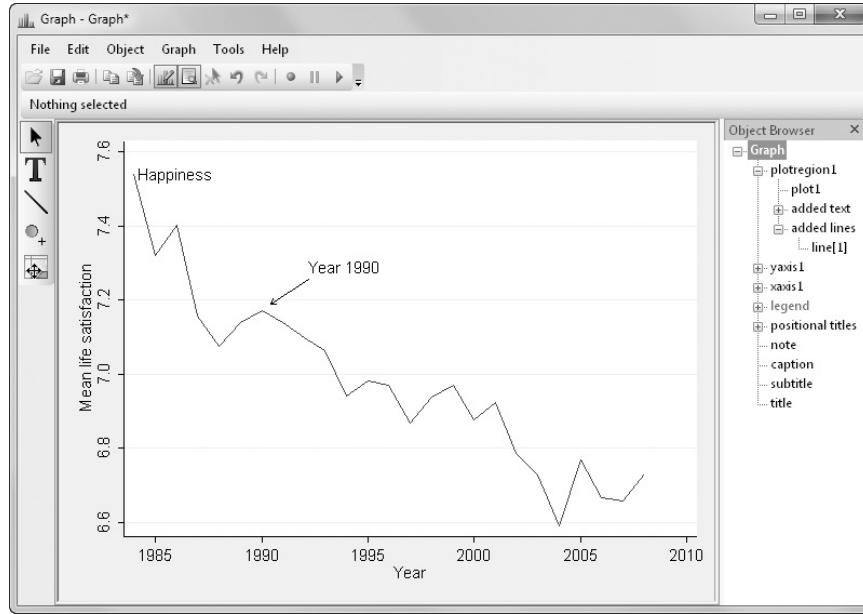


Figure 6.3. The Graph Editor in Stata for Windows

Before you continue reading, please close the Graph Editor without saving the graph. The changes you made are now lost. That is fine for our purpose. If you do want to keep the changes, you should save the graph using **File > Save As....** When you save the graph, keep in mind that you only save the final product, not the path that lead you there.⁴

6.3.4 Information outside the plot region

In addition to using labels inside the plot region, you also have many possibilities for graph design outside the plot region. Changes you make there may not affect the data presentation; nevertheless, some changes outside the plot region can affect the plot region and therefore the data region. This is primarily because the information outside the plot region takes available space from the plot region. In extreme cases, this can affect the shape and size of the plot region, making it harder to interpret its content. Also some of the following options may affect the axis scaling.

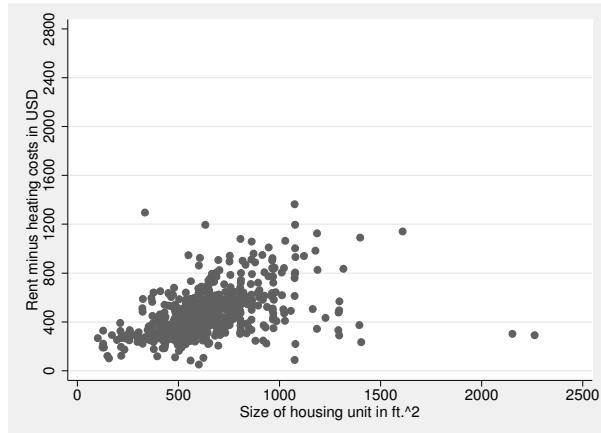
4. The item **Recorder** in the **Tool** menu allows you to store the changes that you have made to your graph. Storing changes allows you to redo the same changes to another version of the same graph.

Labeling the axes

Axis labels are the values that are placed next to the tick lines along the axes (see figure 6.2 on page 120). Although the default setting labels the axes reasonably well, you can change the settings by using the options `xlabel()` and `ylabel()` for the *x* and *y* axes, respectively. Within the parentheses, you can indicate one of three ways to specify the values to be labeled:

- Specify a list of numbers (see section 3.1.7):

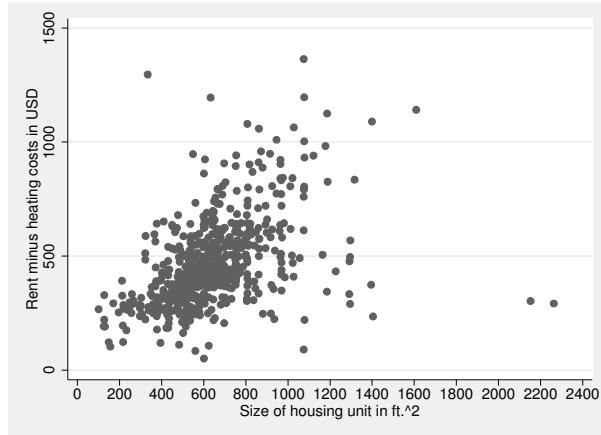
```
. restore  
. scatter rent size, ylabel(0(400)2800)
```



A list of numbers whose value area goes beyond that of the minimum or maximum values of the data will change the scaling of the axes.

- Specify an approximate number of values to be labeled. This number is placed after the pound sign:

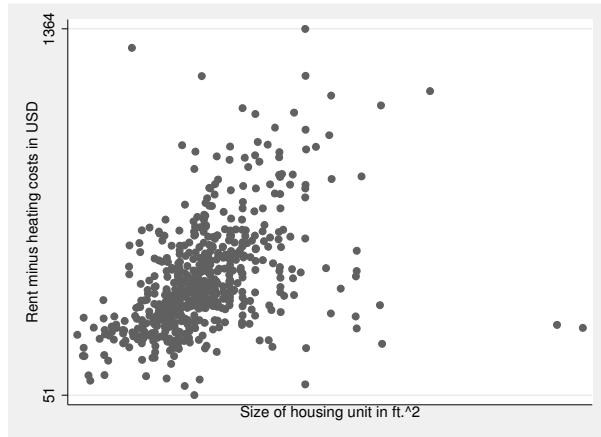
```
. scatter rent size, xlabel(#15)
```



By using this method, you are specifying only an *approximate* number of values to be labeled. Stata still tries to find reasonable values. So do not be surprised that in our example, we end up having 14 value labels instead of 15.

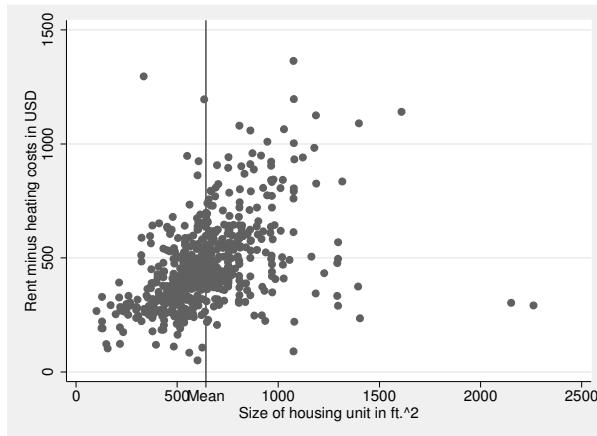
- Specify no labels with the keyword `none` or specify to label the minimum and maximum with `minmax`.

```
. scatter rent size, xlabel(None) ylabel(minmax)
```



The axes are normally directly labeled with the numbers in `xlabel()` or `ylabel()`. However, you can also enter text after one or more values of the number list. The number will then be displayed as text in the graph. The text must be set in quotation marks. Here is an example:

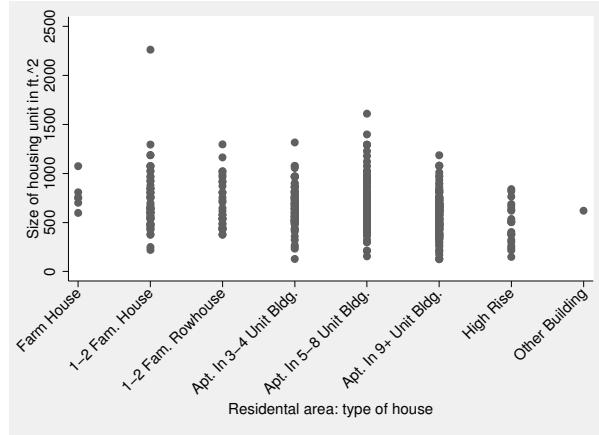
```
. summarize size if !missing(rent)
. local sizemean = r(mean)
. scatter rent size, xline(`sizemean')
> xlabel(0 500 `sizemean' "Mean" 1000(500)2500)
```



The `xlabel()` and `ylabel()` options also have a series of suboptions that may be entered into the parentheses and separated by a comma. For example, depending on the chosen graph scheme, grid lines are often drawn onto the labels. You can suppress grid lines by using the `nogrid` suboption.

The suboption `valuelabel` is part of the `xlabel()` and `ylabel()` options and enables you to replace the number with the value label assigned to the variable forming the axis. In the following example, we will simultaneously use the `angle(45)` suboption for the `x` axis. This places the axis label at a 45° angle to the axis. For the `y` axis, we specify `nogrid`. `nogrid` is a suboption and is therefore entered after a comma within `ylabel()`.

```
. scatter size area1, xlabel(1(1)8, valuelabel angle(45)) ylabel(,nogrid)
```



For a complete list of available suboptions, type `help axis_label_options` or see [G-3] *axis_label_options*.

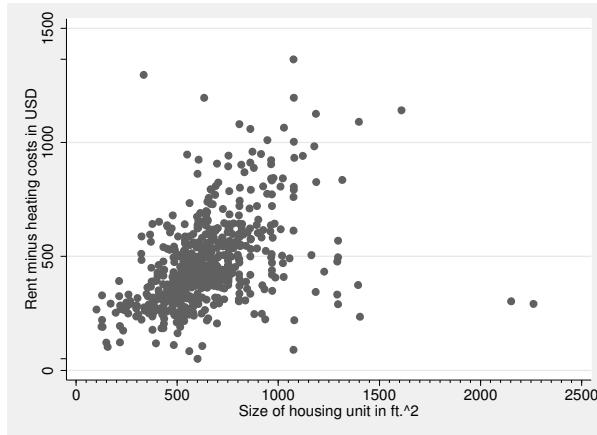
Tick lines

Tick lines are the little lines between the axis and the axis labels; they are usually drawn by Stata near the labels. You can modify tick lines with the following options:

- `xlabel()` for tick lines along the *x* axis and `ylabel()` for the *y* axis.
- `xmtick()` for small tick lines along the *x* axis and `ymtick()` for the *y* axis.

To specify the number of tick lines, you can use the three possibilities we introduced within the previous section: a list of numbers, an approximate number of tick lines, or keywords. For small tick lines, you need two pound signs to specify the number of small tick lines, and the number you specify refers to the number of small tick lines between each of the larger tick lines.

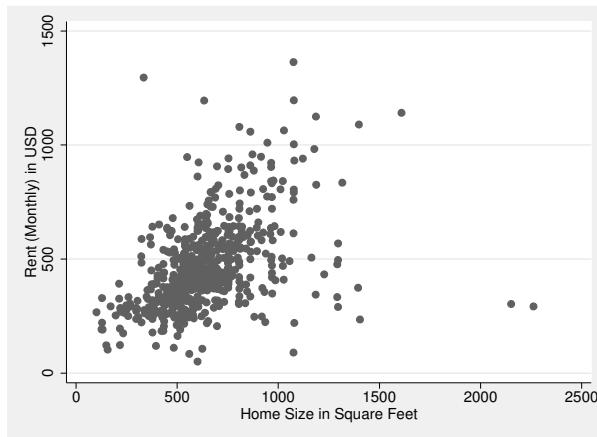
```
. scatter rent size, ytick(minmax) xmtick(##10)
```



Axis titles

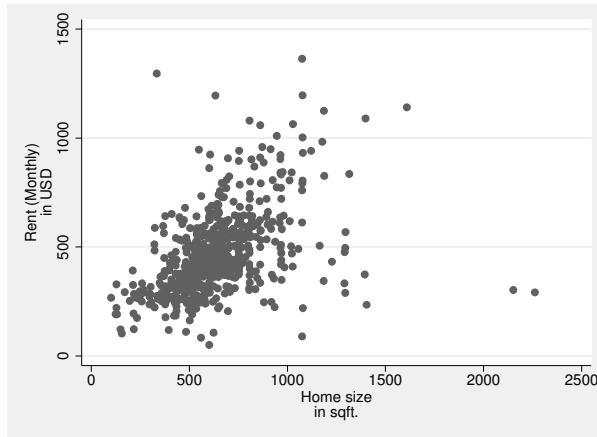
You can provide every axis in a graph with a title, one or more lines long, by using the `xtitle()` and `ytitle()` options. You type the title, set in quotation marks, within the parentheses.

```
. scatter rent size, ytitle("Rent (Monthly) in USD")
> xtitle("Home Size in Square Feet")
```



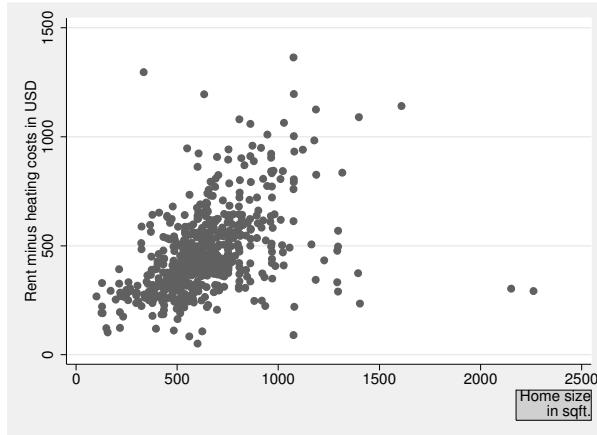
When specifying an axis title with more than one line, set each line in quotation marks separately. The first piece of quoted text will appear on the first line, the second on the second line, and so on.

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



You can easily modify the shape, appearance, and placement of the title by using textbox options in the parentheses of the `ytitle()` and `xtitle()` options. For detailed information, type `help textbox_options` or see [G-3] *textbox_options*. The example below is primarily a teaser:

```
. scatter rent size, xtitle("Home size" "in sqft.", placement(east)
> box justification(right))
```



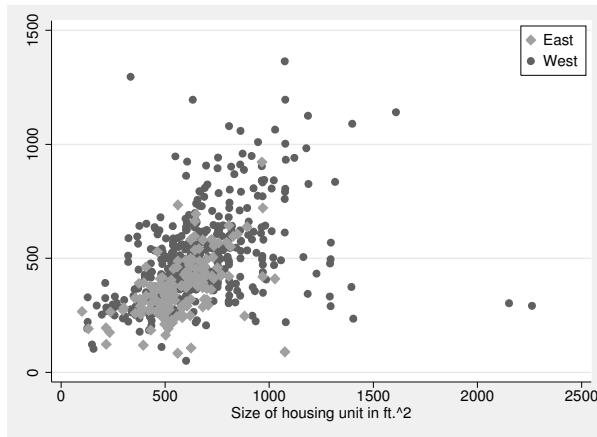
The legend

If a graph contains more than one *y* variable with visible markers, Stata automatically creates a legend that displays the variable names or labels used in a given data area.

You can edit the legend by using the `legend()` option, in which you specify the content and position of the legend. Here we will limit ourselves to specifying the position. However, there are many other things you can add to a legend and many other options for controlling how legends look; see `help legend_option` for the details.

First, take a look at the graph resulting from the following command before we consider the individual options more closely (see page 122 for the creation of variables `rent_w` and `rent_e`):

```
. scatter rent_w rent_e size, legend(cols(1) ring(0) position(1)
> order(2 "East" 1 "West"))
```



In this example, we made four changes to the legend. First, we specified that all entries in the legend will be displayed in one column by using the `cols(1)` option, where the number of columns that the legend should have is put in the parentheses. You can specify the number of rows by using the `rows()` option.

Then we changed the position of the legend by using the `ring()` option. The legend is now located inside rather than outside the plot region. The ring position 0 assigns a position inside the plot region; a ring position more than 0 means a position outside the plot region.

Next we have set the position of the legend inside the plot region. The number specified in the `position()` option refers to the position of numbers on a clock face, as we discussed earlier (see section 6.3.3).

Finally, we changed the order of the entries in the legend. By default, the first data region is listed first in the legend, but this could be changed with the option `order()`. In our example, we stated `order(2 1)` so that the legend starts with the second data region before the first. We also placed text behind the number for each data region, which is used as text inside the legend. This way to specify labels for the data region is somewhat discouraged in [G-3] ***legend_options*** although it is very convenient.

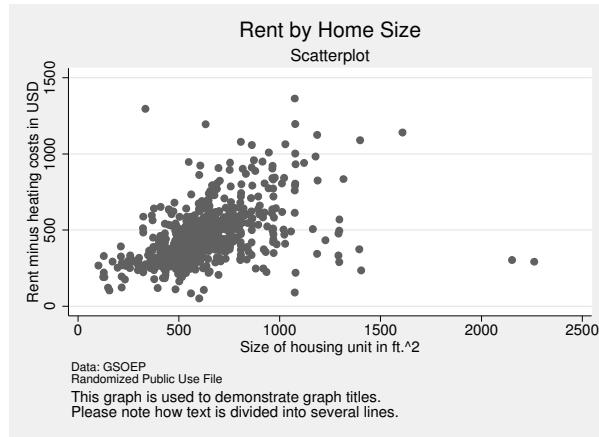
Graph titles

The following options allow you to control the title and other label elements:

- `title()` for the graph title;
- `subtitle()` for a subtitle for the graph;
- `note()` for some explanatory text in a small font, which is frequently used for references or bibliographies; and
- `caption()` for text beneath the graph, which can make the graph easier to interpret.

In each case, you enter text in the parentheses. If the text consists of more than one line, each line must be entered in a separate set of quotation marks, as described in section 6.3.4. Here is an example:

```
. scatter rent size, title("Rent by Home Size") subtitle("Scatterplot")
> note("Data: GSOEP" "Randomized Public Use File")
> caption("This graph is used to demonstrate graph titles."
> "Please note how text is divided into several lines.")
```



You can use several suboptions to change the appearance of the title; for more information, see [G-3] `title_options` and [G-3] `textbox_options`.

6.4 Multiple graphs

In Stata, you can create multiple graphs in several different ways. By “multiple graphs”, we mean graphs that consist of different graph parts, in particular,

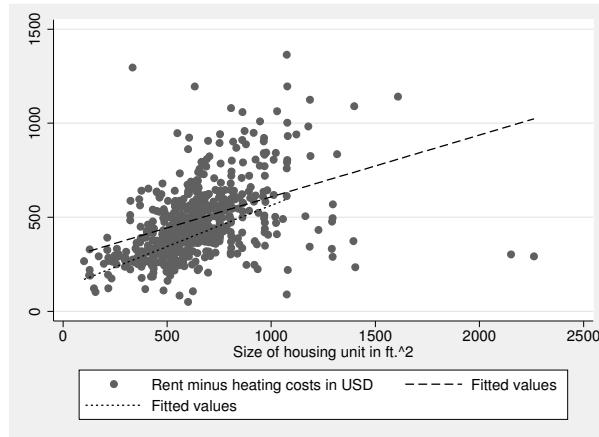
- `twoway` graphs that are plotted on top of each other,
- graphs that are broken out with the `by()` option and are then displayed together, and
- varying graphs that are combined using the `graph combine` command.

We will now quickly introduce these three types of graphs.

6.4.1 Overlaying many twoway graphs

You can overlay as many types of `twoway` graphs as you want in the same coordinate system. In the following example, three graphs are placed on top of each other: a scatterplot; a *linear fit* (or *regression line*; see chapter 9) for the same data, but restricted to the old federal states in West Germany (`rent_w`); and finally, a linear fit that is restricted to the new federal states in East Germany (`rent_e`).

```
. twoway || scatter rent size || lfit rent_w size || lfit rent_e size
```



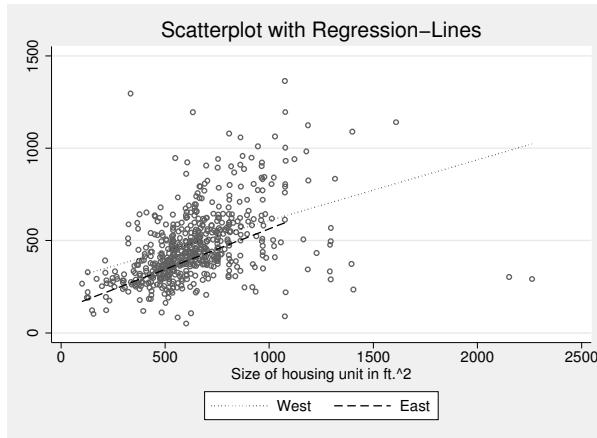
To overlay `twoway` graphs, you consolidate the graphs in a single `twoway` command, separated by parentheses or two vertical lines. In this book and our own work, we use two vertical lines because there are already so many parentheses in the graph syntax. The two vertical lines are particularly readable in do-files containing the individual graphs one after another with line breaks commented out (see section 2.2.2).

If you are combining several twoway graphs, you can specify options that correspond to the respective graph types, as well as twoway options that apply to all the graphs to be combined. Generally, the syntax for overlaid twoway graphs is as follows:

```
twoway
|| scatter varlist, scatter_options
|| lfit varlist, lfit_options
|| plottype varlist, plottype_options,
|| twoway_options
```

The first and the last two vertical bars are superfluous. However, we tend to use them for long graph commands to enhance readability. This syntax structure can be illustrated with an example (but note that commands of that length are often better typed into the Do-file Editor than into the command line):

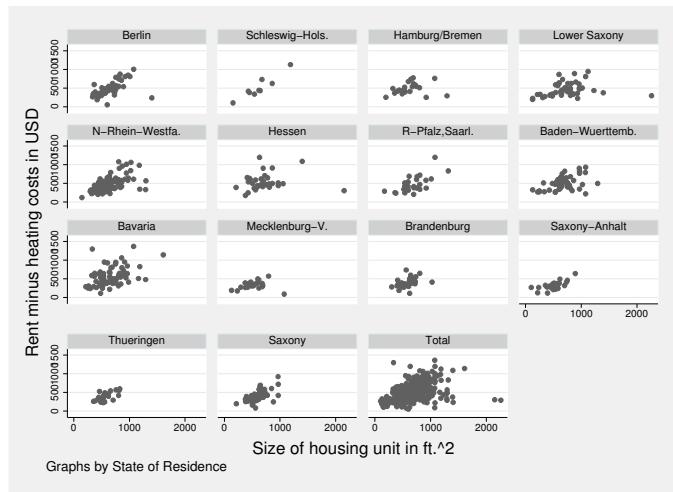
```
. twoway || scatter rent size, msymbol(oh)
> || lfit rent_w size, clpattern(dot)
> || lfit rent_e size, clpattern(dash)
> || , title("Scatterplot with Regression-Lines") legend(order(2 "West" 3 "East"))
```



6.4.2 Option by()

The `by()` option displays separate graphs for each group defined by the variable in the parentheses. If more than one variable is entered in the parentheses, graphs are provided for every combination of the chosen variables. If you also specify the `total` suboption, another graph is displayed without separating it by group. Other suboptions control the positioning (for example, `rows()` and `cols()`), the display or omission of individual axes (for example, `[no]ixaxes`), or the appearance of the margins between the individual graphs. For the list of suboptions, see `help by_option` or [G-3] **by_option**. One example should be enough at this point:

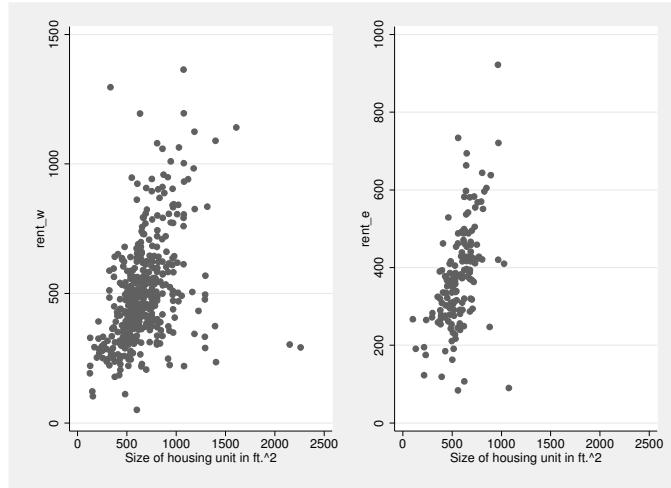
```
. scatter rent size, by(state, total)
```



6.4.3 Combining graphs

Stata allows you to combine as many graphs as you want into a joint graph. To do this, you first save the individual graphs and then combine them using `graph combine`. We will demonstrate this using a display of `rent` by `size`, separated by respondents from East and West Germany:

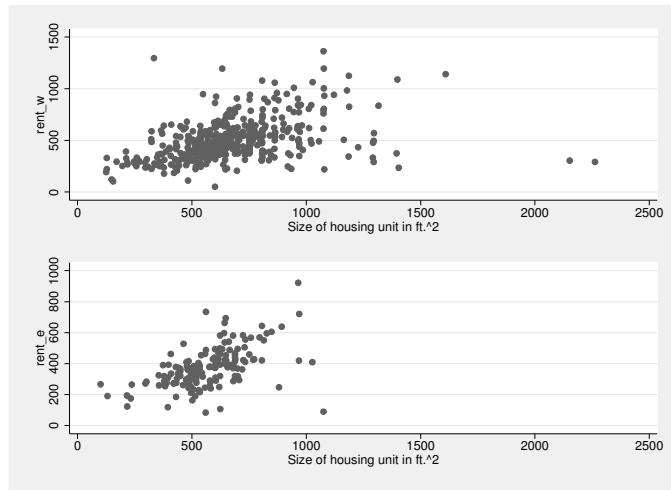
```
. scatter rent_w size, name(west, replace)
. scatter rent_e size, name(east, replace)
. graph combine west east
```



To save both graphs, we use the `name()` option, which specifies the name under which the graph will be saved in the computer's memory. The `replace` suboption tells Stata to delete any graphs already saved under this name. We then combine the two graphs using `graph combine`.

The `graph combine` command has a series of options for controlling how the combined graph is to be displayed. To begin with, it is important to set the number of rows and columns in the combined graph. The individual graphs are placed in the combined graph in rows and columns in a matrix-like fashion. The positioning of the individual graphs depends on how many rows and columns the matrix has. In the matrix above, one row and two columns were used. Here you will see what happens if we instead use two rows and one column:

```
. graph combine west east, rows(2)
```

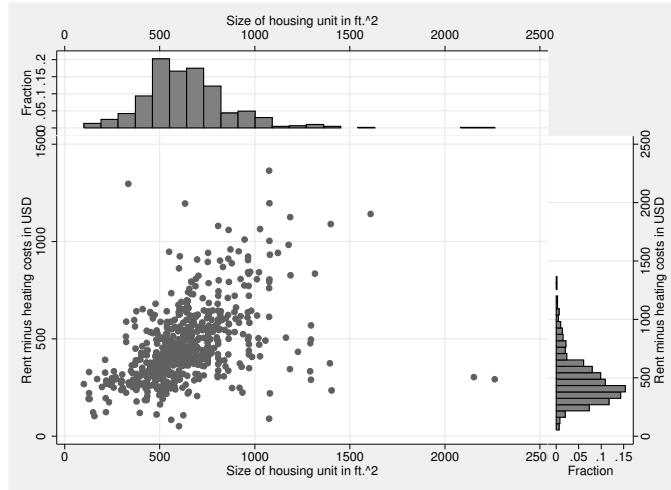


The number of individual graphs you can put in a multiple graph is limited only by printer and screen resolutions. If the main issue is the readability of the labels, you can increase their size with the `iscale()` option. The default font size decreases with every additional graph. With `iscale(1)`, you can restore the text to its original size; `iscale(*.8)` restores the text to 80% of its original size.

If you want the individual graph parts of the combined graph to have different sizes, you will have to save the graphs with different sizes before combining them. However, you cannot use the `xsize()` and `ysize()` options discussed in section 6.3.2, because these sizes are not taken into account by `graph combine`. Instead, you will have to use the forced-size options `fysize()` and `fxsize()`, which tell Stata to use only a certain percentage of the available space. For example, the `fxsize(25)` option creates a graph that uses only 25% of the width of the available space; correspondingly, a graph created using the `fysize(25)` option uses only 25% of the available height.

Here is a slightly more advanced example of `graph combine` using the forced-size options.

```
. twoway scatter rent size, name(xy, replace)
> xlabel(, grid) ylabel(, grid gmax)
. twoway histogram size, name(hx, replace) fraction
> xscale(alt) xlabel(, grid gmax) fysize(25)
. twoway histogram rent, fraction name(hy, replace) horizontal
> yscale(alt) ylabel(0(500)2500, grid gmax) fxsize(25)
. graph combine hx xy hy, imargin(0 0 0 0) hole(2)
```



For more details on creating such graphs, see [G-2] **graph combine**. The Graph Editor has further capabilities regarding the positioning of graph elements. The repositioning is best done with the Grid Edit Tool. A description of its functionality can be found in [G-1] **graph editor**.

6.5 Saving and printing graphs

To print a Stata graph, you type

```
. graph print
```

The graph displayed in Stata's Graph window is then printed directly. If you create many graphs in a do-file, you can also print them with that do-file by typing the `graph print` command after every graph command. The `graph print` command also has many options that can be used to change the printout of a graph (see `help pr_options`).

You can also print graphs that are not (or are no longer) in the Graph window, but you must first save the graph to memory or to a file on the hard drive. You already learned how to save graphs to memory in the previous section. There you

saved numerous graphs with the `name()` option, which we then combined with the `graph combine` command. To print out these graphs, you first display them with `graph display` and then print them using `graph print`. Let us try this with the graph saved as `east` above (see page 150):

```
. graph display east
. graph print
```

Saving graphs to memory is often useful, but they are lost when you close Stata. To print graphs that you created a few days ago, you must have saved them as a file by using the `saving()` option. It works the same way as the `name()` option except that it saves the file to the hard drive. You type the `filename` under which the file is to be saved in the parentheses. By default, Stata uses the `.gph` file extension. To overwrite an existing file of the same name, include the `replace` option in the parentheses. The option `replace` is commonly used when creating graphs with do-files.

Be careful when you use the Graph Editor. When you exit the Graph Editor, you will be asked if you want to save the changes you made to the graph. In most cases, you would want to answer yes. If you do so, you should make sure that you use a *new* filename. If you save the graph with its old name, you might be in trouble next time you run the do-file that created the graph originally. Running the do-file will re-create the original graph and therefore overwrite the changes you made with the Graph Editor. You would need to start over.

All saved files can be, at a later point in time, printed or edited with the Graph Editor. To do so, you simply call the saved graph on the screen with the command `graph use` and start printing or editing. The command

```
. graph combine hx xy hy, hole(2) imargin(0 0 0) saving(combined, replace)
```

saves the graph under the name `combined.gph` in the current working directory. If a file with the same name already exists in the working directory, it is overwritten because you specified `replace`. You can now close Stata, shut down the computer, or display another graph ...

```
. graph display east
```

... and regardless of what you do, you can print the graph saved in a file by typing

```
. graph use combined
. graph print
```

Finally, you will usually be exporting a Stata graph to a word processing program or presentation program rather than printing it. For this, you can use the much-loved *copy-and-paste* procedure, where you first copy the graph displayed in a Graph window and then paste it into the respective document. If you are dealing with several graphs, it is better to save the graph in a suitable file format on the hard drive and then import to the desired program when needed.⁵

5. When doing this and working with do-files, it is a good idea to document your work (see chapter 2).

To save a graph in a different file format, use the `graph export` command. You type the filename with the appropriate file extension after the command. Table 6.1 lists the formats that are available to you.

Table 6.1. Available file formats for graphs

Extension	File format	Restriction
.ps	PostScript	
.eps	Encapsulated PostScript	
.wmf	Windows Metafile	Windows
.emf	Windows Enhanced Metafile	Windows
.pict	Mac Picture Format	Mac
.pdf	Portable Document Format	Windows/Mac
.png	Portable Network Graphics	
.tif	Tagged-Image File Format	

Microsoft applications normally handle files saved in WMF or EMF formats well. The same applies to most other software that runs under Windows operating systems. PostScript and Encapsulated PostScript work well on Unix systems, and you should use them if you write reports using L^AT_EX. Mac users will usually prefer PDF or PICT files. In any case, if you want to save the graph in the Graph window as a WMF file called `mygraph1.wmf`, use

```
. graph export mygraph1.wmf
```

Use the other file formats as you wish.

6.6 Exercises

1. Get data from the National Health and Nutrition Examination Study (NHANES) by using the following command:


```
. webuse nhanes2.dta, clear
```
2. Using the NHANES data, produce a scatterplot of weight in kg by height in cm. Use hollow circles as the marker symbol.
3. Change the title of the vertical axis to “Weight (in kg)”, and add a note for the data source of your graph.
4. Add reference lines to indicate the arithmetic means of weight and heights.
5. Add an axis label to explain the meaning of the reference lines.
6. Use blue marker symbols for male observations and pink marker symbols for female observations, and construct a self-explanatory legend. Remove the reference lines.

7. Plot the data for men and women separately, and produce a common figure of both plots placed on top of each other. Take care that the note on the data source does not appear twice in the figure.
8. Construct a graph similar to the previous one but this time with reference lines for the gender-specific averages of weight and height.
9. Create a variable holding the body mass index [BMI; see (5.1) on page 112], and classify the observations according to the table on page 112. Change the previous graph so that the colors of the marker symbols represent the categorized BMI.
10. Add the unique person identifier (`samp1`) to the symbols for the male and the female observations with the highest BMI.
11. Export your graphs so that they can be imported into your favorite word processing program.

7 Describing and comparing distributions

So far, we have dealt with the basic functionality of Stata. We have used data mainly to show you how Stata works. From now on, we will do it the other way around: we will show you how to use Stata as a tool to analyze data *and* understand the analysis.

We begin with the process of describing distributions. An example for such a description is the presentation of political election returns on television. Consider the announcement that a candidate for the presidency is elected by, say, 80% of American voters. You will definitely regard this description as interesting. But why? It is interesting because you know that this candidate has won the election and that he will be the next president of the United States. Moreover, you know that 80% of the votes is quite a success because you know that previous elections have never seen such a result.

Now consider the following description of a distribution: In Baden-Württemberg, a state in the southwest of Germany, 28% of the inhabitants live in single-family houses. What do you think about this information? You might wonder what the proportion would be in the area where you live—which means you need some knowledge about the distribution to figure out if 28% is high or low.

Generally speaking, a description of a single distribution is satisfying as long as we know something *a priori* about the distribution. Here we can compare the actual distribution with our knowledge. If we do not have such knowledge, we need to collect information about the distribution from somewhere else and compare it with the distribution. A full description of a distribution therefore often involves comparing it with a comparable distribution.

There are many ways to describe distributions. One criterion—not exactly a statistical criterion, but in practical applications quite an important one—for choosing the description is the number of categories. Distributions with just a few categories can often be fully described with tables, but there are also some graphical tools for them. Describing distributions with many categories is more difficult. You will often evaluate statistical measures, but in most cases, graphical tools are better.

In what follows, we will first distinguish between distributions with few categories and those with many categories. Then we will treat those two cases separately.

To follow this chapter, load our example dataset:¹

```
. use data1, clear
```

7.1 Categories: Few or many?

Let us begin with some vocabulary. In what follows, we use the letter n for the number of observations; the uppercase letter Y for a variable; and $y_i, i = 1, \dots, n$, to denote the values of the variable. The value of the variable Y for the first observation is y_1 , for the second observation is y_2 , etc. Take our dataset `data1.dta` as an example. `data1.dta` has $n = 5,411$. One of the variables in it (`pib`) contains the party affiliation of each respondent. If we choose this variable as Y , we can look at the values y_1, \dots, y_{5411} with

```
. list pib
```

Obviously, this list generally is not the information we were looking for; we are simply overwhelmed by the amount of information scrolling by. You should break the list as described on page 8.

To get a better understanding, you can begin by finding out how many different numbers occur in Y . We will call these numbers *categories* and denote them with a_j . Thus j is an index that runs from the lowest to the highest category ($j = 1, \dots, k$). The number of different categories of a variable usually is much smaller than the number of observations. In Stata, we can find out the number of categories with `inspect`. For example, we get the number of categories for party affiliation as follows:

```
. inspect pib
pib: Political party supported
          Number of Observations
          Total    Integers   Nonintegers
#           Negative      -        -
#           Zero         -        -
#           Positive     2101     2101     -
#           Total        2101     2101     -
# . # . . . Missing       3310
1          8          5411
(8 unique values)
pib is labeled and all values are documented in the label.
```

Party identification in our dataset has $k = 8$ categories (`(8 unique values)`). All categories are positive integers (`Integers`) between 1 and 8. There are only 2,101 respondents with one of those eight categories. The rest have another category: the missing value.²

We will distinguish between variables with few categories and variables with many categories by setting the threshold at approximately 6–10 categories. But do not take

1. Please make sure your working directory is `c:\data\kk3`; see page 3.
2. Read more on missings in section 5.5, as well as on pages 11 and 413.

this threshold too literally. With graphical tools, you can take “few” much more broadly than you can with tables. Often you will get a good description of a variable with few categories by using a technique designed for variables with many categories and vice versa. The most important criterion for describing a variable is that the main properties of the distribution stand out. You should try out different approaches.

7.2 Variables with few categories

7.2.1 Tables

Frequency tables

The most important way to describe a distribution with few categories is with a one-way frequency table, which lists the absolute and relative frequencies of all categories a_j of a variable. The absolute frequency n_j is the number of observations in the category a_j . The relative frequencies f_j are the ratios of the absolute frequencies to the entire number of observations:

$$f_j = \frac{n_j}{n}$$

In what follows, we will often use the word “proportion” for relative frequency.

In Stata, one-way frequency tables are produced with the command `tabulate`, which can be abbreviated to `tab` or even `ta`. You specify the variable for which the table is displayed by typing the variable name after the command. The following command shows the one-way frequency table for partisanship:

<code>. tabulate pib</code>		Freq.	Percent	Cum.
Political party supported				
SPD	733	34.89	34.89	
CDU/CSU	795	37.84	72.73	
FDP	144	6.85	79.58	
Greens/B90	211	10.04	89.62	
Linke	145	6.90	96.53	
DVU, Rep., NPD	32	1.52	98.05	
Other parties	29	1.38	99.43	
Several parties	12	0.57	100.00	
Total	2,101	100.00		

The first column of this frequency table shows the different categories a_j of party identification (Germany has a *multiparty system*). The second column shows the absolute frequencies n_j , and the third column shows the proportion as a percentage ($f_j \times 100$). The final column is for the cumulated relative frequencies, which we will not discuss here.

To illustrate how to interpret all those figures, suppose that the answers to the question about partisanship are votes in an election for a “parliament of respondents”. The Social Democratic Party (SPD) has won $n_1 = 733$ votes, or $f_1 = 733/2101 = 34.89\%$. Five parties have been chosen by more than 5% of the respondents. No single party has won more than 50% of the votes. But together with the Christian Democrats (CDU/CSU), the Social Democrats have won $733 + 795 = 1528$ votes (about 73%) and might therefore form a coalition cabinet.

The frequency table has been calculated for 2,101 observations, but the entire dataset has 5,411 observations. The difference stems from observations with missing values. In our dataset, people who have not answered a question are set to the missing value for that variable. By default, the `tabulate` command excludes missing values, so the above table is produced only for respondents who have answered the question on party identification. To include the missing values, use the option `missing`:

```
. tabulate pib, missing
```

In the following tables and graphs, we will usually not explicitly include missing values. Some of the commands we will describe allow the inclusion of missing values with the option `missing`, just like the example above. If not, you can always replace missing values with numeric values (see section 5.5).

More than one frequency table

Using `tabulate` with two variables will give you a two-way frequency table. To generate more than one one-way frequency table with a single command, you need `tab1`, which generates a frequency table for each variable of a *varlist*. Here are some examples:

```
. tab1 pib
. tab1 pib sex
. tab1 pi*, missing
. tab1 sex - emp
```

Comparing distributions

Examining a single distribution is rarely useful. Often you need to compare a given distribution with the same distribution from some other time, population, or group.

If you have sampled your distribution in different groups, you can do this by simply producing a one-way table for each of the groups by using the `by` prefix. For example, the following command produces a one-way table for men and women separately:

```
. by sex, sort: tabulate pib
```

We have compared one distribution (the distribution of party identification) based on different levels of another distribution (gender). A more technical way to say this is that we have shown the distribution of party identification *conditioned* on gender.

A more illustrative way to show a distribution conditioned on another distribution is with a two-way table, also known as a cross table or contingency table. A two-way table displays the distribution of one variable, say, Y , for the categories of another variable, say, X , side by side.

To produce such a two-way table with Stata, you simply include a second variable name in the *varlist* after `tabulate`. Here is an example—the two-way frequency table of party identification by gender:

Political party supported	Gender		Total
	Male	Female	
SPD	381	352	733
CDU/CSU	415	380	795
FDP	85	59	144
Greens/B90	86	125	211
Linke	88	57	145
DVU, Rep., NPD	25	7	32
Other parties	19	10	29
Several parties	5	7	12
Total	1,104	997	2,101

The first variable forms the rows of the table, and the second variable forms the columns. The body of the table shows the distribution of party identification for each level of the gender variable.³ For example, in the “male” column, there are 381 SPD supporters and 415 CDU/CSU supporters. In the “female” column, there are 352 SPD supporters and 380 CDU/CSU supporters. You have already seen these numbers in response to the previously entered command by `sex`, sort: `tabulate pib`.

In addition to the number of observations with specific variable combinations, you also find the row and column sums of those numbers. The rightmost column shows the overall distribution of party identification, which you have already seen as a result of the command `tabulate pib` (page 159). The bottom row displays the same for `gender`.

Although it is easy to understand the meaning of the numbers in the two-way frequency table above, you should not use such tables to compare distributions between different groups. Instead, you should use the proportions within each group, sometimes called the conditional relative frequencies, for the comparison. Under the condition that $X = a_j$, the conditional relative frequency of the variable Y is calculated by dividing each absolute frequency by the number of observations within the group $X = a_j$. Thus, for our example, when `sex==men`, the conditional relative frequency distribution of party identification is given by dividing the number of male supporters of each party by the total number of men.

3. Or the other way around, the distribution of gender for each level of party identification.

Stata calculates the conditional proportions in two-way tables with the options `row` and `column`. The proportions of party identification conditioned on gender are calculated with the option `column`, giving us the proportions for the row variable conditioned on the column variable. Here is an example where we also use the option `nofreq` to suppress the output of the absolute frequencies:

Political party supported	Gender		Total
	Male	Female	
SPD	34.51	35.31	34.89
CDU/CSU	37.59	38.11	37.84
FDP	7.70	5.92	6.85
Greens/B90	7.79	12.54	10.04
Linke	7.97	5.72	6.90
DVU, Rep., NPD	2.26	0.70	1.52
Other parties	1.72	1.00	1.38
Several parties	0.45	0.70	0.57
Total	100.00	100.00	100.00

The numbers shown in this table are the same as those calculated with the command by `sex`, `sort`: `tabulate pib` on page 160. As you can see, 8% of German men support the Greens, while 13% of German women do. That is, German women lean somewhat more to the Greens than do German men. You can see that men lean somewhat more toward the FDP and the far right parties like the DVU than do women.

In addition to the proportions conditioned on the column variable, you can also calculate the proportions of the column variable conditioned on the row variable by using the `row` option. You can use that option in addition to, or instead of, the `column` option. You could use any of the following commands:

```
. tabulate pib sex, row
. tabulate pib sex, row column
. tabulate pib sex, row nofreq
. tabulate pib sex, row column nofreq
```

Summary statistics

The `tabulate` command has options for calculating overall statistics for the differences in the distributions between groups. We will not explain these calculations here, but we will simply list the options and the names of the statistics as an overview. For a complete description, see the cited references. The formulas can be found in [R] `tabulate twoway`:

- `chi2`: Pearson's chi-squared (Pearson 1900)
- `gamma`: Goodman and Kruskal's gamma (Agresti 1984, 159–161)
- `exact`: Fisher's exact test (Fisher 1935)

- `lrchi2`: likelihood-ratio chi-squared test (Fienberg 1980, 40)
- `taub`: Kendall's tau-b (Agresti 1984, 161–163)
- `V`: Cramér's V (Agresti 1984, 23–24)

More than one contingency table

`tabulate` allows up to two variable names. If you list three variable names, you will get an error message. There are two reasons why you might want to try `tabulate` with more than two variable names: to produce a three-way table or to produce more than one two-way table with a single command. You can produce three-way frequency tables using the `by` prefix (section 3.2.1) or the `table` command (section 7.3.2).

To produce more than one two-way table with a single command, use `tab2`, which produces two-way tables for all possible combinations of the variable list. Therefore, the command

```
. tab2 pia pib pic sex
```

is equivalent to typing

```
. tabulate pia pib
. tabulate pia pic
. tabulate pia sex
. tabulate pib pic
. tabulate pib sex
. tabulate pic sex
```

However, in many cases, you will want to display only some of the tables. For example, you might want to tabulate each of the three party-identification variables conditioned on gender. In this case, using a `foreach` loop (section 3.2.2) would be more appropriate.

7.2.2 Graphs

During data analysis, graphs are seldom used for displaying variables with just a few categories because tables are usually sufficient. However, in presentations, you will often see graphs, even for variables with few categories. Most often these are special types of histograms or atypical uses of bar charts or pie charts. Stata can be used for all of these presentation techniques. Stata also produces dot charts (Cleveland 1994, 150–153), which are seldom used but powerful.

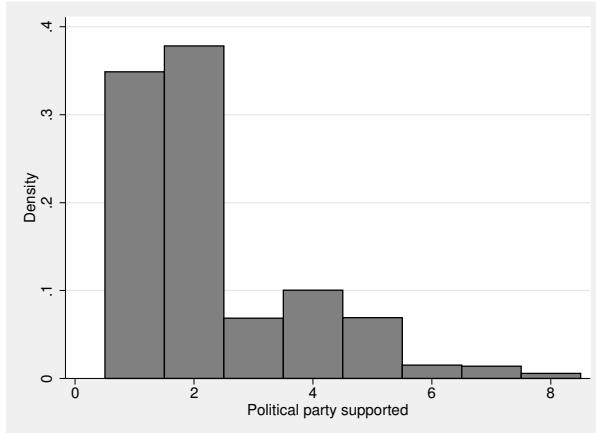
Before we explain the different chart types, we emphasize that we will use only very simple forms of the charts. To learn how to dress them up more, refer to chapter 6.

Histograms

In their basic form, histograms are graphical displays of continuous variables with many outcomes (see section 7.3.3). In this form, histograms plot the frequencies of groups or intervals of the continuous variable. Grouping the continuous variable generally requires that you choose an origin and the width of the intervals.

When dealing with variables with just a few categories, you do not need to make this choice. Instead, you can plot the frequency of each category with the `histogram` command and the `discrete` option.⁴ First, we draw a histogram of the variable for party identification (`pib`):

```
. histogram pib, discrete
```



What do we see here? Essentially, we see some rectangles, or *bars*, each of which represents one category of the variable for party identification. Namely, the first bar represents the first category of party identification (the SPD), the second bar represents the second category (the Christian Democratic Union or CDU), and so on. Because party identification has eight categories, there are eight bars in the graph.

The height of the bars varies. According to the label on the *y* axis, the height of the bars represents the density. In the special case of the histograms discussed here, this density equals the proportions. You can use the option `fraction` to see proportions instead of densities,

```
. histogram pib, discrete fraction
```

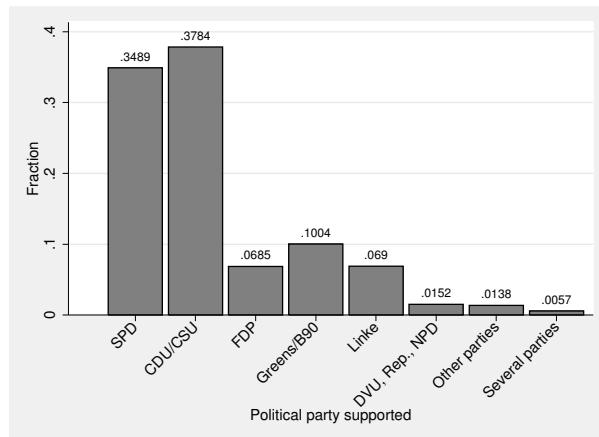
but this changes only the label on the *y* axis, not the numbers or the relative height of the bars. In the more general case, the density is not equal to the proportion (see section 7.3.3).

4. The name `discrete` arises from the fact that variables with few categories are usually discrete. Especially in small samples, continuous variables can also have few categories. Here it makes sense to specify `discrete` even for continuous variables.

The height of the bar represents the proportion of the category. The higher the bar, the higher the proportion of supporters for the party represented by the bar. You can see that the CDU/CSU supporters are most frequent in the data, followed by the SPD supporters, and so on. In fact, you see the same results as those in the table in section 7.2.1, but this time graphically.

The graph is a bit hard to read because we do not know from the display which bar represents which party. Therefore, you should use the option `xlabel`, which we described in section 6.3.4. Moreover, there are two options specific to histograms, which we introduce here: `gap()` and `addlabels`. With nominal scaled variables, you might prefer a display with gaps between the bars. This can be achieved with the option `gap()`, where a number inside the parentheses specifies the percentage to reduce the width of the bar. For example, `gap(10)` reduces the width of the bar by 10%, creating a gap between the bars. Strictly speaking, with `gap()`, your graph can no longer be called a histogram. Finally, some people like to have numbers with the exact proportions above the bars. The `addlabels` option is their friend. Let us use these options now:

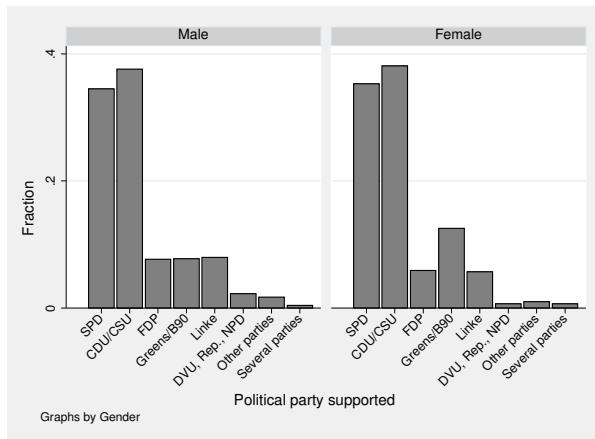
```
. histogram pib, discrete fraction gap(10) addlabels
> xlabel(1(1)8, valuelabel angle(45))
```



Conditional distributions can be shown using the `by()` option, which we have already described in section 6.4.2. The argument is the name of the variable on which you want to condition your distribution. For example,

```
. histogram pib, discrete fraction by(sex) gap(10)
> xlabel(1(1)8, valuelabel angle(45))
```

displays the distribution of party identification for men and women separately:



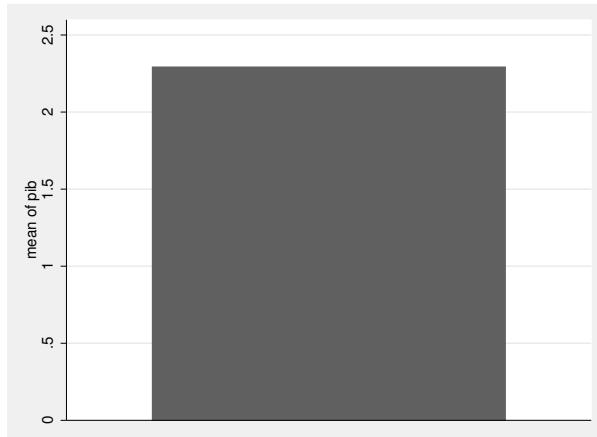
Again we see that among the GSOEP respondents, women lean more toward the Greens than do men. Beyond this difference, the overall answer pattern is quite similar for men and women.

Bar charts

Bar charts and histograms are frequently confused because both are often used to show proportions of discrete variables, although they are not intended for that task. Histograms are intended to show proportions of continuous variables, and bar charts are intended to display summary statistics of one or more variables.

Using techniques in an unintended manner can lead to surprises, and that often happens with bar charts. Look what happens if you naïvely try to use a bar chart to display the proportion of party affiliation:

```
. graph bar pib
```



As we mentioned, bar charts are intended to display summary statistics of one or more variables. This is done by plotting a bar with a height proportional to the size of the summary statistic for each variable to be plotted. Stata allows you to plot a variety of summary statistics with a bar chart, for example, the mean, the number of nonmissing observations, and the sum. The default is the mean, so the bar in our bar chart represents the mean of party affiliation, which in this case is useless or, at any rate, a waste of space.

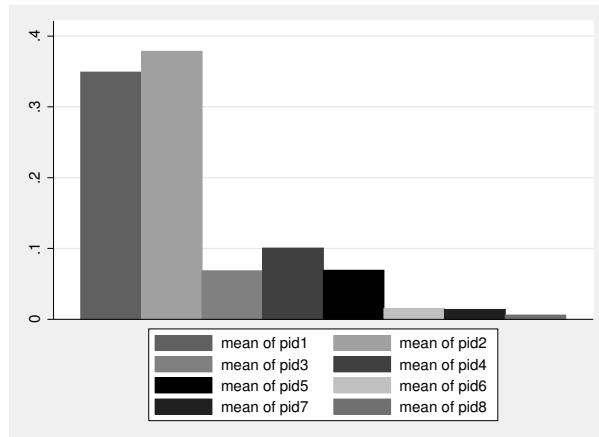
To get what we want, we need to generate what we will call a set of dummy variables. Generally, dummy variables are variables with values of 0 and 1. For example, we could create a dummy variable that is 1 for all respondents who favor the SPD and 0 for all others (see page 80 for details on generating dummy variables). The mean of this dummy variable would be equal to the proportion of SPD partisans. Therefore, if we used this dummy variable in our bar chart, we would get a bar with height equal to the proportion of SPD partisans, which would not be useless but would still be a waste of space. We also need bars for all the other parties, so we need to define a dummy variable for each category of party affiliation and use all of these dummies in our bar chart.

The easiest way to generate such a set of dummy variables is to use the `generate()` option of the `tabulate` command. Typing

```
. tabulate pid, generate(pid)
```

produces the dummy variable `pid1` for the first category of party identification, `pid2` for the second, `pid3` for the third, and so on. This set of dummy variables can be used for the bar chart:

```
. graph bar pid*
```



This bar chart displays the proportions of each category of party identification—much like the histogram in the previous section. Now you know why bar charts and histograms are so frequently confused. But be aware that we have used both techniques in a somewhat nonstandard way, by graphing a variable that is not continuous.

To show conditional distributions, you can choose the option `over()` or `by()`. Put the name of the variable on which you want to condition your distribution inside the parentheses. The option `over()` displays the proportion of party identification conditioned on its argument in one graph. The `by()` option gives you one graph for each distribution, arranged side by side in one display.

Pie charts

In the mass media and certain business presentations, you often see pie charts used to present distributions of variables with few categories. In the literature on graphical perception, pie charts are often criticized because the reader needs to decode sizes and angles to interpret a pie chart, which is not easy for humans to do (Cleveland 1994, 262–264). We share this critical view of pie charts, so we will keep our description of them short.

In Stata, pie charts are implemented much like bar charts. Stata pie charts show slices of a pie with a size proportional to a summary statistic. Consequently, we again need to produce dummy variables to show the proportion of the categories. To compare distributions, you must use the option `by()`:

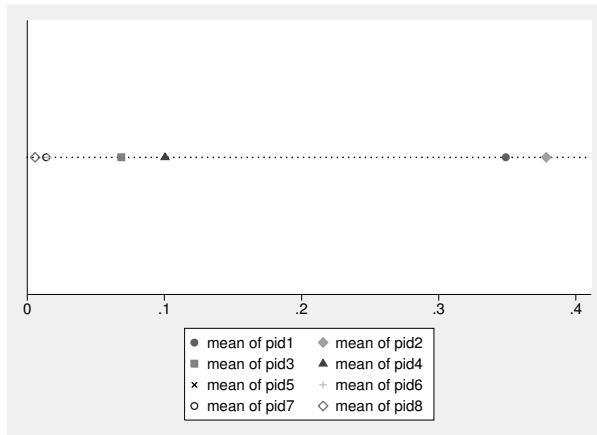
```
. graph pie pid*, by(sex)
```

Dot charts

Dot charts were introduced by Cleveland (1984) as a graphical method to display data with labels. Dot charts have shown their strength in experiments on graphical perception—not only for variables with few categories but also for variables with many categories. However, dot charts are seldom used either in scientific literature or in the mass media. One reason for this is that dot charts are rarely implemented in statistical packages or spreadsheet programs.

In principle, dot charts are similar to bar charts except that dot charts use a dot to display the value of a statistic instead of a bar. Thus dot charts are implemented like bar charts in Stata. You can replace the graph subcommand `bar` with `dot` to produce a dot chart. As in bar charts, however, you need to use dummy variables to show relative frequencies. Here is a first attempt:

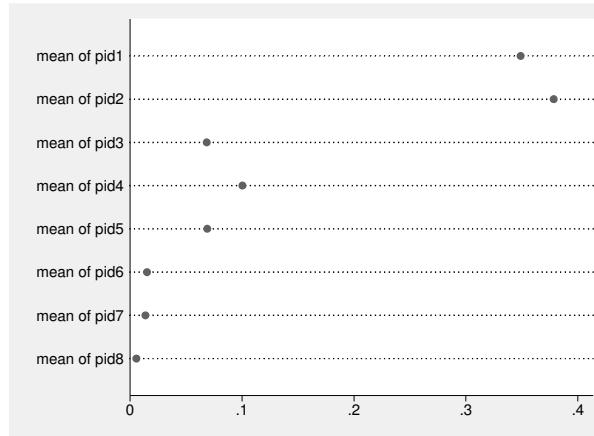
```
. graph dot pid*
```



This is not a good-looking display. But before refining the graph, you should understand what you see. The graph shows a marker symbol, on one dotted line, for each summary statistic. The rightmost marker symbol (the filled diamond) shows the mean of the variable `pid2`, which happens to be the proportion of respondents leaning toward the CDU/CSU. The other marker symbols can be interpreted similarly.

You might get a nicer graph if you put each symbol on a different line. We will do so presently, but first note from the previous graph that you *can* put more than one symbol on one line and that this is possible only because we have used symbols instead of bars. Later on, when describing the distributions of continuous variables, we will use this feature—let us call it *superposition*—to compare distributions between groups. But for now, we need to find a way to put the markers on different lines. This is most easily done using the option `ascategory`, which specifies that the variables listed in `varlist` be treated as a grouping variable. See how it works:

```
. graph dot pid*, asccategory
```



Using this as a starting point, we can immediately use the option `over()` or `by()` to compare the distribution between groups:

```
. graph dot pid*, asccategory over(sex)
. graph dot pid*, asccategory by(sex)
```

7.3 Variables with many categories

You can describe distributions of variables with many categories by using tables, summary statistics, or graphs. To display such variables with tables, you need to group the variables first. Therefore, we begin by showing you the nuts and bolts of grouping variables in section 7.3.1. In section 7.3.2, we explain the most common summary statistics and show you how to use them to describe and compare distributions, both with tables and graphically. Finally, we show you some graphical methods intended especially for the description of distributions with many categories in section 7.3.3.

7.3.1 Frequencies of grouped data

Variables with many categories usually cannot be described in tables. If you type

```
. tabulate income
```

you will immediately understand why: such tables can become huge, and the frequencies for each category are likely to be too small to provide any useful information. One solution is to group such variables, that is, recode the variable by assigning the information on some similar categories to one category. Taking `income` as an example, this could mean using the frequencies of the intervals [0, 1000), [1000, 2000), etc., instead of the frequencies for every category of income. This would lead to a frequency distribution of grouped data.

You can arrive at a description of grouped data in two steps:

1. Generate a new variable that is a grouped version of the original variable.
2. Display the new variable with techniques for the description of variables with few categories.

You can generate the grouped version of the variable with the methods described in chapter 5, as well as some specific tools for this task. But first we would like to make some general remarks on grouping variables.

Some remarks on grouping data

Grouping variables leads to a loss of information, which may emphasize or hide certain properties of distributions. Grouping variables therefore is an important step of data analysis. You should carefully decide how to group your variables and inspect the consequences of your decision. There are suggested rules for grouping variables, for example, “Always use intervals of equal width” or “Do not group nominal scaled variables”.

However, in practice such rules are not helpful. The research goal always determines whether the loss of information due to a specific grouping is problematic. If the goal is to detect data errors, group the moderate values into one category and leave the extreme values as they are. If the goal is to compare the distribution of income of unemployed people between Germany and France, you might want to group all incomes above a specific level into one category and form finer intervals for lower incomes.

The same is true for nominal scaled variables. In a very large population sample of German inhabitants, the variable for nationality can easily include 20–30 categories. For most data analysis, you will want to group that variable, for example, by differentiating only between Germans and foreigners. However, whether that or any grouping procedure will be useful depends on the research topic.

Special techniques for grouping data

In grouping variables, we need to differentiate between nominal scaled variables and other variables. For nominal scaled variables—such as `nation`—the order of the categories has no special meaning. To group such variables, you need to decide if any category can be grouped with another. Therefore, in performing the grouping, you are confined to the techniques described in chapter 5. For variables with categories that have a quantitative meaning, where observations with low values are also low in a real sense, you can always say that *neighboring* values are similar to some extent. Therefore, you can often refer to entire ranges of categories, and Stata has a set of special techniques for that task.

Grouping by quantiles

Sometimes, you will want to group a distribution into intervals that contain nearly the same numbers of observations. We call this grouping by quantiles (see also section 7.3.2). In Stata, you can group by quantiles with the command `xtile`. To generate a new variable with, say, four groups with almost the same number of respondents, you simply type

```
. xtile inc_4 = income, nquantiles(4)
```

If you had used the option `nquantiles(10)`, the variable `inc_4` would have contained 10 intervals with nearly the same number of respondents, and so on.

Because the new variable has only four categories, you can simply use the techniques described in the last section, for example,

```
. tabulate inc_4
```

Grouping into intervals with same width

Grouping variables by quantiles leads to variables having almost the same number of observations in each category, whereas the widths of the intervals of each category differ. That is, the difference between the upper bound of a category, c_j , and the upper bound of the category below, c_{j-1} , is not constant for all categories.

Instead of grouping by quantiles, you can also try to get intervals with the same width for each class. For example, you could group income so that all respondents with income between \$0 and \$5,000 form the first interval, respondents with income between \$5,001 and \$10,000 form the second interval, and so on, until the maximum income is reached.

More generally, you group the values of a variable into k categories with $d_j = c_j - c_{j-1}$ equal for all categories using either the `recode()` function (see page 84) or the `autocode()` function.

Let us begin with an example of the `recode()` function. With that function, the grouping of income into intervals with equal width can be done like this:

```
. generate inc_g1 = recode(income,5000,10000,15000,20000,25000,30000,60000)
```

This generates the new variable `inc_g1`, which becomes 5,000 for all observations with an income of €5,000 or below (that is, between €0 and €5,000), 10,000 for all observations above €5,000 up to (and including) €10,000, and so on. All nonmissing observations with an income above €30,000 become €60,000 on `inc_g1`.

The one-way frequency table of the grouped income variable `inc_g1` can be displayed as usual with

```
. tabulate inc_g1
```

The `autocode()` function is a sort of shorthand for the `recode()` function if you want to have intervals with equal widths. The syntax of the `autocode()` function is

```
autocode(exp,k,min,max)
```

The `autocode()` function internally splits the interval from `min` to `max` into `k` intervals with equal widths and returns the upper limit of the interval that contains the value of the expression `exp`. To group the variable `income` into 13 intervals, you can use the `autocode()` function as follows:

```
. generate inc_g2 = autocode(income,13,0,12500)
```

Grouping into intervals with arbitrary widths

You can group a variable into intervals with arbitrary widths by simply imposing the upper limits of the arbitrary intervals on the list of numbers in the `recode()` function:

```
. generate inc_g3 = recode(income,2000,10000,20000,40000,80000,200000)
. tabulate inc_g3
```

7.3.2 Describing data using statistics

Summary statistics are often used to describe data with many categories. With summary statistics, you can describe distributions parsimoniously. Generally, you will distinguish between summary statistics for the position and for the dispersion of distributions. Summary statistics for the position describe which values the observations *typically* have, and summary statistics for the dispersion describe how different the values for the observations are. Generally, you should use at least one summary statistic of each type to describe a distribution.

We will begin by briefly describing the most important summary statistics for distributions. Then we will present two commands for calculating these statistics: `summarize` and `tabstat`.

Important summary statistics**The arithmetic mean**

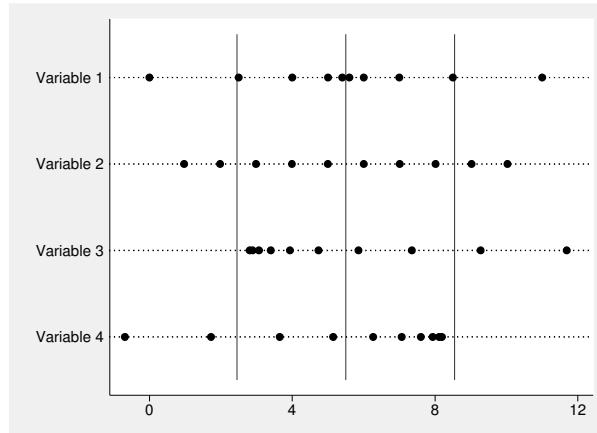
The most common summary statistic for determining the central position of a distribution is the arithmetic mean, often simply called the average. The arithmetic mean is a summary statistic for determining the position of variables with interval-level scales or higher, that is, for variables like income, household size, or age. You can also use the arithmetic mean for dichotomous variables that are coded 0 and 1 (dummy variables) because the arithmetic mean of such variables gives the proportion of observations that are coded 1. Sometimes, you might also use the arithmetic mean for ordinal variables, such as life satisfaction or the intensity of party identification. But strictly speaking, this approach is wrong, and you should not take the resulting numbers too seriously. You definitely should not use the arithmetic mean for categorical variables that take on more than two values, such as marital status or type of neighborhood, because the result makes no sense.

The standard deviation

The most common summary statistic for determining the dispersion of a distribution is the standard deviation. The standard deviation can be thought of as the average distance of the observations from the arithmetic mean. This interpretation is not entirely correct but may give you an idea of the notion. To calculate the standard deviation, you first need to calculate the arithmetic mean. This means that you can calculate the standard deviation only for variables for which you can calculate the arithmetic mean.

In some ways, the arithmetic mean and the standard deviation can be seen as sibling measures for describing a distribution. Usually, when you describe the position of the distribution with the arithmetic mean, you will use the standard deviation to describe the dispersion of the distribution. But the description of a distribution in terms of its mean and standard deviation can be problematic.

To understand why, take a look at the data in figure 7.1. Each line in this graph gives you the values of one of four artificial distributions. As you see, the distributions of all four variables are different. The first variable has what statisticians call a normal distribution, the second variable has a uniform distribution, and the third and fourth variables are skewed to the right and to the left, respectively. Nevertheless, all four variables share the same summary statistics: they have a mean of 5.5 and a standard deviation of 3.05. Given only the description with summary statistics, you might have concluded that all four distributions are identical.



Source: grmeans.do

Figure 7.1. Distributions with equal averages and standard deviations

To guard against such misinterpretations, you should consider other summary statistics (or describe your data graphically).

Quantiles

Quantiles are the most important companions to the arithmetic mean and the standard deviation. The p -quantile ($x_{[p]}$) splits a distribution into two parts such that the first part contains $p \times 100$ percent of the data and the second part contains $(1 - p) \times 100$ percent. In particular, the 0.5-quantile—the median—separates the data so that each part contains 50% of the observations.

To calculate the quantiles, you use the position i of each observation in the sorted list of a distribution. The p -quantile is the value of the first observation with position $i > np$. If there is an observation with $i = np$, you use the midpoint between the value of that observation and the value of the following observation. To find the 0.5-quantile of a distribution with 121 valid observations, you need to search for the 61st ($121 \times 0.5 = 60.5$) observation in the sorted list and use the value of the distribution for this observation. For 120 observations, you would choose a value between the values of the 60th and 61st observations.

To find the quantiles, you use the position of the observations in the sorted data. The values of the categories are of interest only insofar as they determine the order of the sorted list. It does not matter if the highest value is much higher than all others or just a bit higher. In this sense, quantiles are *robust* against outliers.

In practice, the most important quantiles are the *quartiles*: the quantiles with $p = 0.25$ (first quartile), $p = 0.5$ (*median* or second quartile), and $p = 0.75$ (third quartile). From the quartiles, you can also learn something about the skewness and the dispersion of a distribution.

- If the distances of the first and third quartiles from the median are equal or almost equal, the distribution is symmetric. If the first quartile is closer to the median than the third quartile is, we say the distribution is skewed to the right. If it is the other way around, we say the distribution is skewed to the left.
- The difference between the third quartile and the first quartile is called the *interquartile range*. This value tells us the range of values that are the middle 50% of the observations.

By the way, the quartiles for the distributions from figure 7.1 are shown in table 7.1. The results clearly show the different shapes of the distributions.

Table 7.1. Quartiles for the distributions

Variable	1st Quartile	Median	3rd Quartile
1	4.00	5.50	7.00
2	2.98	5.50	8.02
3	3.07	4.33	7.35
4	3.65	6.67	7.93

The `summarize` command

You already know the most important command for calculating summary statistics: `summarize`. In its basic form, `summarize` calculates the mean and the standard deviation:

<code>. summarize income</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
income	4779	20540.6	37422.49	0	897756

From this output, you can see that the mean monthly income of the German population in 2009 was €20,540.60 and that (roughly speaking) the difference between the observed income and the mean income was about €37,422.49 on average.

In addition to the mean and the standard deviation, `summarize` also reports the minimum (lowest value) and the maximum (highest value) of the distribution. In our example, the lowest income of all respondents is €0 and the highest income is €897,756.

`summarize` also calculates a set of quantiles if you specify the `detail` option. Note that quantiles are called percentiles in the output because the value of p is expressed in percent. In the output below, you can now see that while the mean is €20,540, the median of the income distribution is €12,424, much lower than the mean. This indicates a right-skewed distribution, also indicated by a positive value for skewness.

Individual Labor Earnings			
	Percentiles	Smallest	
1%	0	0	
5%	0	0	
10%	0	0	Obs 4779
25%	0	0	Sum of Wgt. 4779
50%	12424		Mean 20540.6
		Largest	Std. Dev. 37422.49
75%	30998	710920	
90%	46972	749421	Variance 1.40e+09
95%	62400	869446	Skewness 11.53857
99%	100341	897756	Kurtosis 213.8692

The `tabstat` command

The `tabstat` command also displays and calculates summary statistics. It is a generalization of `summarize`, because it allows you to specify a list of statistics to be displayed using the option `statistics()`. Inside the parentheses you specify the names of the statistics to be displayed. The default is `statistics(mean)`, which displays the arithmetic mean, but you can also use other statistics and even multiple statistics. For example, typing

```
. tabstat income, statistics(count mean sd min max)
```

displays the number of nonmissing observations (`count`), the mean (`mean`), the standard deviation (`sd`), the minimum (`min`), and the maximum (`max`)—replicating the output of `summarize`.

The following command shows the minimum, the three quartiles (`p25 p50 p75`), and the maximum—what is sometimes referred to as the “five-number summary” of a distribution:

<code>. tabstat income, statistics(min p25 p50 p75 max)</code>					
variable	min	p25	p50	p75	max
income	0	0	12424	30998	897756

For a list of names for statistics you can calculate with `tabstat`, see `help tabstat`.

Comparing distributions using statistics

Stata provides several tools for comparing different distributions through summary statistics:

- a combination of `summarize` with the prefix `by`,
- the `summarize()` option of `tabulate`,
- the `by()` option of `tabstat`,
- the `table` command, and
- graphs of summary statistics.

We will not go into the combination of `summarize` and the `by` prefix because we have already described this tool in section 3.2.1. Most likely, you have already used `summarize` with `by` several times by now. If not, you should read section 3.2.1 now. This combination is easy to use and quite powerful: you can use it to obtain most of the results of the other tools we have described. The main advantage of the other tools is the attractive arrangement of the results.

For those already familiar with inferential statistics, we should point out that Stata has a variety of built-in commands. Among the frequently used built-in commands are `ttest` to test the equality of means, `prtest` to test the equality of proportions, and `ranksum` to test the hypothesis that two independent samples are from populations with the same distribution. A general introduction to inference statistics is given in chapter 8.

The `summarize()` option of `tabulate`

The `summarize()` option of `tabulate` is used to display the arithmetic means and standard deviations of a distribution, conditioned on the values of one or two other variables. For example, to show the average and standard deviation of income conditioned on gender, you can use

Gender	Summary of Individual Labor Earnings		
	Mean	Std. Dev.	Freq.
Male	28190.753	47868.242	2320
Female	13322.888	21286.438	2459
Total	20540.6	37422.487	4779

The `summarize()` option includes the means and standard deviations of the variable to be summarized. This tells you that the mean income for men is about €28,190.75 and the mean income for women is about €13,322.89.

You can also use the `summarize()` option in two-way tables. The advantages of comparing distributions this way may be even more convincing. Suppose that you want to know the income inequality between men and women in each German state separately—you want to know the mean income conditioned on gender and state. In this case, you might produce a two-way table containing the mean of income in each of its cells:

State of Residence	Means of Individual Labor Earnings		
	Gender		Total
	Male	Female	
Berlin	18410.298	22315.633	20403.646
Schleswig	31986.029	19552	25905.153
Hamburg/B	24720.95	15341.66	19375.763
Lower Sax	26120.599	11564.005	18404.809
N-Rhein-W	29157.271	13037.589	21129.477
Hessen	40825.28	14387.923	27648.834
R-Pfalz,S	28233.652	11271.49	19462.123
Baden-Wue	36559.568	13231.973	24312.58
Bavaria	33366.952	14349.809	23314.215
Mecklenbu	19170.673	13481.516	16076.57
Brandenbu	19242.543	12862.092	15852.929
Saxony-An	19745.641	10923.757	15250.681
Thueringe	22546.186	9984.5391	15732.085
Saxony	16729.839	11092.707	14010.919
Total	28190.753	13322.888	20540.6

We used the options `nostandard` and `nofreq` to suppress the output of standard deviations and frequencies.

The statistics and by() option of tabstat

The `summarize()` option of `tabulate` allows you to compare only means and standard deviations of a distribution between different groups. `tabstat` is statistically more flexible. You can use the `by()` option of `tabstat` to specify that the statistics be displayed separately for each unique value of a variable given within the parentheses. Therefore, to produce the five-number summary of income conditioned on gender, you could type

. tabstat income, statistics(count q max) by(sex)					
Summary for variables: income					
by categories of: sex (Gender)					
sex	N	p25	p50	p75	max
Male	2320	270.5	22142.5	39884	897756
Female	2459	0	6761	21787	612757
Total	4779	0	12424	30998	897756

You can use only one variable name in the parentheses of the `by()` option. The `table` command is one way to overcome this restriction.

The `table` command

The `table` command is a generalization of the techniques we have described so far. With `table`, you can display many types of statistics, including means, standard deviations, and arbitrary quantiles, in tables having up to seven dimensions. In this respect, the advantages of `table` are obvious, but it is not as fast and its syntax is slightly more complicated.

The syntax of `table` has two parts: one to format the table and one to specify the contents of the cells. As with `tabulate`, the table can be laid out simply in rows,

Worried about economic development	Freq.
Very concerned	2,404
Somewhat concerned	2,623
Not concerned at all	298
Refusal	20

or in rows and columns,

Worried about economic development	Gender	
	Male	Female
Very concerned	1,154	1,250
Somewhat concerned	1,238	1,385
Not concerned at all	153	145
Refusal	8	12

However, in `table` you can also specify a third variable, which defines a *supercolumn*. The supercolumn works as if you had specified `tabulate` with a `by` prefix except that the different tables are displayed side by side. This way, you get a three-way table:

Worried about economic development	Status of Employment and Gender							
	full time		part time		irregular		not emplo	
	Male	Female	Male	Female	Male	Female	Male	Female
Very concerned	636	322	18	258	27	98	442	563
Somewhat concerned	654	338	36	258	32	104	468	649
Not concerned at all	52	31	5	23	5	7	78	74
Refusal	4	4		1		2	4	5

Finally, you can specify up to four *superrows*, which work like supercolumns except that the different parts of the table are displayed one below each other—but still in one table. Superrows are specified as a variable list with the option `by()`:

```
. table wor01 sex emp, by(mar)
```

Marital Status of Individual and Worried about economic development	Status of Employment and Gender							
	- full time -		- part time -		- irregular -		- not emplo -	
	Male	Female	Male	Female	Male	Female	Male	Female
Married								
Very concerned	423	144	10	195	15	74	307	313
Somewhat concerned	408	145	19	195	11	63	272	361
Not concerned at all	26	12	2	16	1	2	30	26
Refusal	3							3
Single								
Very concerned	136	110	4	18	12	10	62	66
Somewhat concerned	185	135	15	30	20	30	120	113
Not concerned at all	21	16	3	6	3	5	31	24
Refusal	1	4		1		1	4	1
Widowed								
Very concerned	2	8		8		3	39	125
Somewhat concerned	5	5		6	1	4	44	125
Not concerned at all				1			8	15
Refusal								1
Divorced								
Very concerned	62	52	3	31		10	26	53
Somewhat concerned	44	41	2	24		5	28	46
Not concerned at all	4	2			1		6	6
Refusal						1		
Separated								
Very concerned	13	7	1	6		1	8	6
Somewhat concerned	12	12		3		2	4	4
Not concerned at all	1	1					3	3
Refusal								
Refusal				1				
Very concerned								
Somewhat concerned								
Not concerned at all								
Refusal								

The table forms the groups on which we want to condition our variable of interest. But to describe the variable, we still need two pieces of information:

- the variable to be described and
- the summary statistics by which we provide the description.

Both pieces of information are specified with the option `contents()`. Within the parentheses of `contents()`, you first specify the name of the summary statistic and then the name of the variable. Specifying `contents(mean income)`, for example, would fill the cells of the table with the arithmetic mean of income for each of the groups of the table.

As an example, we will reproduce some of the tables we produced earlier. With `table`, the comparison of income conditioned on gender from page 178 would look like this:

```
. table sex, contents(count income mean income sd income)
```

Accordingly, you would get the mean income conditioned on gender and state (page 179) by typing

```
. table state sex, contents(mean income)
```

What if you distrust the mean and want to use quartiles instead?

```
. table sex, contents(p25 income p50 income p75 income)
```

Suppose that you want to compare the monthly rent among apartments having different features. How much do the monthly rents of apartments having central floor heat, a balcony or a terrace, and a basement differ from those without those amenities?

```
. table eqphea eqpter eqpbas, contents(mean rent)
```

Dwelling has central floor heat	Dwelling has basement and Dwelling has balcony/terrace		
	Yes	No	Refusal
Yes	656.083252	499.0514221	705.333313
No	494.868866	469.5681763	
Refusal			

Dwelling has central floor heat	Dwelling has basement and Dwelling has balcony/terrace		
	Yes	No	Refusal
Yes	626.4736938	366.8153992	
No	434	278.375	375
Refusal			

Dwelling has central floor head	Dwelling has basement and Dwelling has balcony/terrace		
	Yes	No	Refusal
Yes	320	401	
No			
Refusal	638		

In this table, you can see that an apartment with the listed amenities rents for € 656 on average, whereas the apartment without the amenities rents for € 278. However, when interpreting results from multidimensional tables like the above, you should also look at the number of observations on which the averages within each cell are based:

```
. table eqpheha eqptter eqpbas, contents(mean rent n rent)
```

Dwelling has central floor head	Dwelling has basement and Dwelling has balcony/terrace		
	Yes	No	Refusal
Yes	656.083252 1,574	499.0514221 525	705.333313 3
No	494.868866 61	469.5681763 44	
Refusal		0	

Dwelling has central floor head	Dwelling has basement and Dwelling has balcony/terrace		
	No		
Yes	626.4736938 76	366.8153992 65	
No	434 2	278.375 8	375 1
Refusal			

Dwelling has central floor head	Dwelling has basement and Dwelling has balcony/terrace		
	Yes	Refusal	No
Yes	320 1		401 1
No			
Refusal	638 1		

The more cells a table has, the more important it is to check that the number of observations is large enough to get reliable estimates of the summary statistic. With a small number of observations, results are sensitive to outliers. As we can see now, the rent of € 278 in the table above is based on just eight observations.

Our examples do not fully explore the range of possibilities from using tables. There are further available summary statistics, and there are many tools for improving the appearance of the table; see [R] **table** and **help table** for further insights.

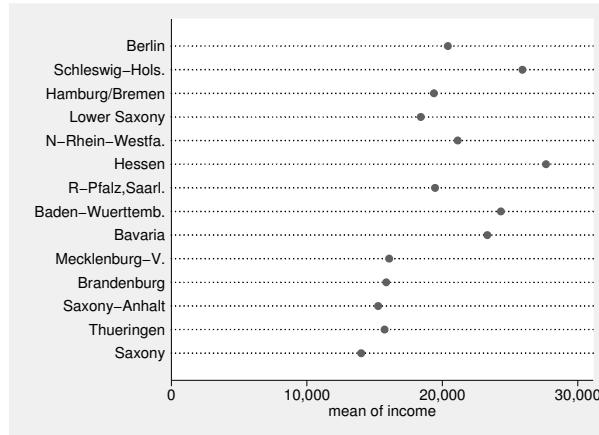
Graphical displays of summary statistics

Stata has three special tools for displaying summary statistics graphically: bar charts, dot charts, and pie charts. Technically, all three work similarly in Stata, but dot charts have been shown to be the most powerful way to graphically display statistical information (Cleveland 1994, 221–269). So we concentrate on the presentation of dot charts. For basic bar or pie charts, it is usually enough to replace the **graph** subcommand **dot** with **bar** or **pie**, respectively.

In our first example, we compare the arithmetic mean of income among the German states:

```
. graph dot (mean) income, over(state)
```

The command begins with **graph dot**, which is the command for making a dot chart. Then we indicate our choice of summary statistic inside the parentheses. Because the arithmetic mean is the default, we could have left **(mean)** out. For a complete list of available summary statistics, refer to **help graph.dot**. After indicating the summary statistic, we entered the name of the variable we want to describe. Finally, inside the parentheses of the option **over()**, we put the name of the variable on which we want to condition the summary statistic. Here is the result:



The graph shows a dot for the mean income of each German state. It can be seen rather quickly that Schleswig-Holstein and Hessen have the highest mean incomes of all German states and that Saxony has the lowest. After some inspection, German readers will see that the average income in the five East German states (the former Communist ones) is lower than in West Germany.

However, the graph is hardly optimal—even if it is not meant for publication. In our graph, we have followed the rule “Alabama first”, which is a well-known rule for displaying data *badly* (Wainer 1984). “Alabama first” means that you display your data in alphabetical or some other order that is unrelated to the content. If you do not want to display data badly, you should give your data a meaningful order. You can do this most easily by specifying the `sort()` suboption within the `over()` option. Inside the parentheses of `sort()`, you specify a sort order, either by the contents of a variable or by a function of these contents. We choose to have our data sorted by the average income: `sort((mean) income)`. `sort()` is a suboption under `over()`, not an option under `graph dot`, so you place it within the `over()` option.

```
. graph dot (mean) income, over(state, sort((mean) income))
```

To condition the summary statistic on more than one variable, we can use the `by()` option. Another method specific to dot charts is *superposition*, which we use in our next example to show the mean income in the different states conditioned on gender.

With dot charts, you can plot more than one summary statistic on one line. If you enter a second variable, a second marker symbol appears on each line, representing the second variable. To show conditional means for men and women separately, you therefore need two income variables: one for men and one for women. You can generate these variables easily with

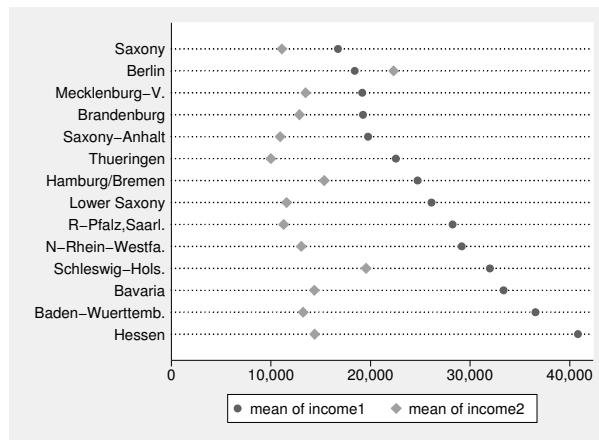
```
. generate inc1 = income if sex == 1
. generate inc2 = income if sex == 2
```

or in one command with

```
. separate income, by(sex)
```

Now you can use the new *separated* variables for the dot chart instead of the original one. We include the `sort()` option on the `graph` command to sort the states by the mean income of males, but you could also use the mean income of females, the overall mean income, or any other useful order. Your research goals, however, determine what is useful.

```
. graph dot (mean) income1 income2, over(state, sort((mean) income1))
```



7.3.3 Graphs

Summary statistics describe variables by highlighting a certain property of the distribution and excluding most information about it according to certain *assumptions*. For example, the description of a distribution with the mean and the standard deviation excludes, among many other things, all information about the skewness of the distribution. If all the distributions are symmetric (or at least equally skewed), the means and the standard deviations provide useful descriptions for comparing them. But if the assumptions underlying a summary statistic are not true, the summary statistic does not provide an informative description.

This section provides an overview of techniques for describing data that require fewer assumptions than those we have previously mentioned. Fewer assumptions mean that more information must be communicated, and for this we use graphs. But these graphs differ from those discussed earlier, which we used to display summary statistics that were calculated under certain assumptions. Here we use graphs to describe the distribution with as few assumptions as possible.

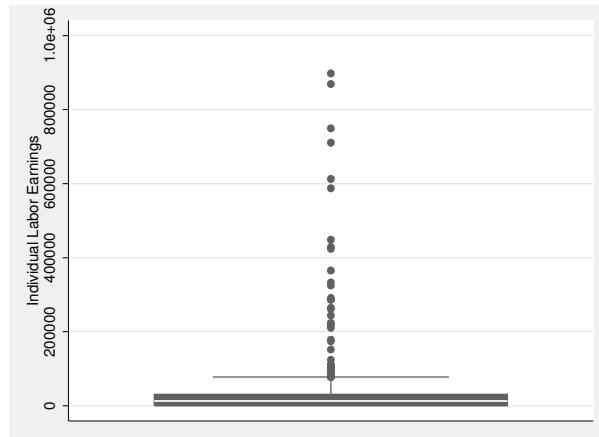
Box plots

In section 7.3.2, we introduced the five-number summary of distributions, which uses the minimum, maximum, and quartiles to describe the data. Box plots can be seen as graphical representations of the five-point description with some enhancements.

Box plots in general are composed of a *box*, two *whiskers*, two *fences*, and some marker symbols. The lower border of the box is the first quartile; the upper border is the third quartile. The line in the middle of the box is the median. The height of the box therefore shows the interquartile range. An upper whisker extends from the third quartile to the value corresponding to the third quartile percentile plus 1.5 times the interquartile range. Likewise, a lower whisker extends from the first quartile to the value corresponding to the first quartile minus 1.5 times the interquartile range.

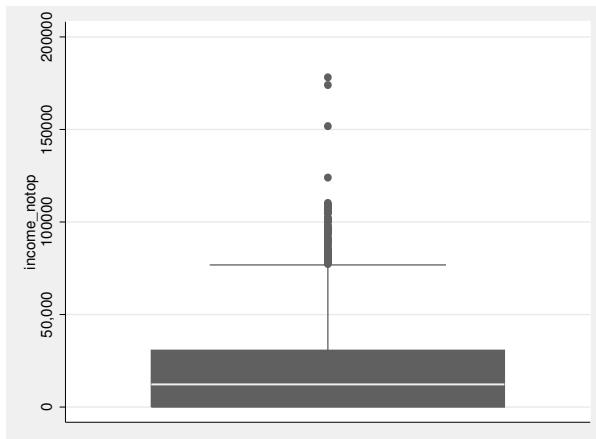
Creating a box plot with the `income` variable, you can hardly see the box:

```
. graph box income
```



The resulting graph is dominated by a series of marker symbols that line up from the top to almost the bottom of display. These marker symbols show the highest values of income, and they dominate the display because they are so much higher than most other values. To us, this characteristic of the income-distribution is important information, and it is good that the figure makes it stand out. However, the extreme values of income also suppress the other features of box plots and make it harder to explain the remaining graphs in this chapter. We therefore use a modified income variable that contains only income values below 200,000. Nevertheless, we encourage you to try the commands with both variables to see how outliers change the graphical features.

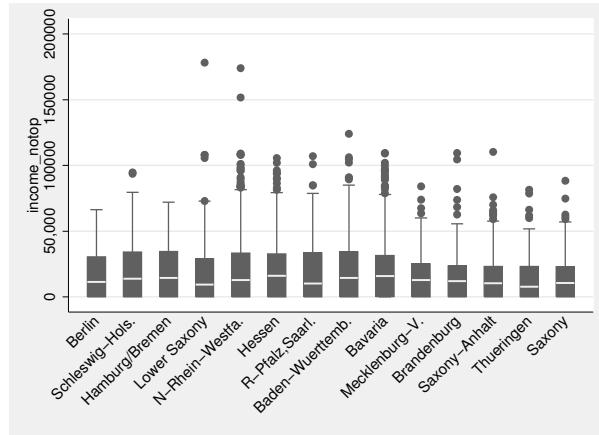
```
. generate income_notop = income if income < 200000
. graph box income_notop
```



Box plots provide much information about a distribution. You can directly infer the position of the distribution from the position of the box along the y axis. Fifty percent of the observations have values between the lower and upper bounds of the box. You can infer the dispersion of the distribution by the size of the box, by the length of the whiskers, and by the positions of the outliers. Moreover, you can see the symmetry or skewness of the distribution: in symmetric distributions, the distances between the median and the lower and upper bounds of the box are equal. Also the whiskers should be of equal length, and the outliers above and below should be equally spaced. If the outliers, the whiskers, or the box are squeezed at the bottom, the distribution is skewed to the right. If they are squeezed at the top, it is skewed to the left. Here the distribution of income is skewed to the right.

Box plots can be used effectively to compare distributions across different groups using the option `over()` or `by()`. You simply put the names of the variables on which you want to condition your distribution inside the option's parentheses. Here is the distribution of income conditioned on state. Because the value labels for `state` are rather long, we use `label(angle(45))` within `over()` to avoid overlapping.

```
. graph box income_notop, over(state, label(angle(45)))
```



Histograms

The most common graphical display for distributions with many categories is the histogram, which is a graphical display of grouped frequency tables. In histograms, a rectangle of height

$$\hat{f}_j = \frac{n_j}{d_j} \quad (7.1)$$

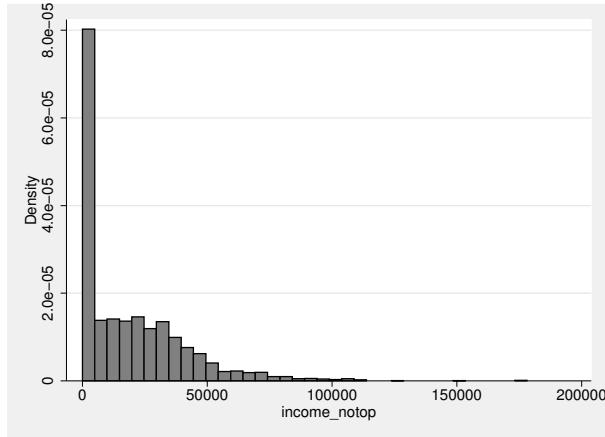
and of width d_j is drawn for each interval j of the grouped distribution, where n_j is the absolute frequency and $d_j = c_j - c_{j-1}$ is the width of the interval. The quantity \hat{f} is called the *density*. The density \hat{f}_j is not the relative frequency f_j . Relative frequencies are always between 0 and 1, whereas densities can be any positive real number. In histograms, the areas of the rectangles are proportional to the relative frequencies of the intervals, whereas their height is proportional to the data density within those intervals.

The implementation of histograms in Stata is confined to a special case: Stata draws histograms where the widths of all rectangles are equal. In other words, Stata draws histograms of distributions that are grouped with `autocode()` (see page 173). For this reason, the heights of the rectangles can also be interpreted as proportions within the intervals. The taller the rectangle, the larger the proportion within the corresponding interval.⁵

5. The ado-package `eqphistogram`, described by Cox (2004), implements histograms that may have rectangles of different widths. For more on installing external ado-packages, see section 13.3.

You obtain a histogram by using the `histogram` command. The command is easy to use: you just specify the command and the name of the variable you want to display. The graph again shows the skewed income distribution. We also see that there are some very low values that are quite frequent.

```
. histogram income_notop
```



Often the histograms you get from this simple command will suffice. But be careful, as the appearance of histograms depends on your choice of the origin and the width of the displayed intervals. If you do not specify them, Stata chooses for you. Stata always chooses the minimum value of the distribution as the origin and infers the width of the intervals from a formula for the *optimal* number of bars to be displayed. For our distribution, Stata decided to use 37 bars. Then Stata internally grouped the income variable into 37 intervals with equal widths and drew bars with heights proportional to the proportion within these intervals. You can do this on your own by using the `autocode()` function and `histogram` with the `discrete` option:

```
. generate inc37 = autocode(income_notop,37,0,200000)
. histogram inc37, discrete
```

The optimal number of groups for histograms is debatable (Emerson and Hoaglin 1983). The goal is to generate a histogram that shows all the important features of the distribution but ignores the unimportant ones. As rules of thumb for the optimal number of groups, $10 \log_{10} n$, $2\sqrt{n}$, and $1 + \log_2 n$ have been proposed. For the 4,757 valid observations on `income_notop`, these rules propose histograms with 37, 138, and 13 groups, respectively. Stata uses

$$\min(\sqrt{n}, 10 \log_{10} n)$$

which generates 37 bars.

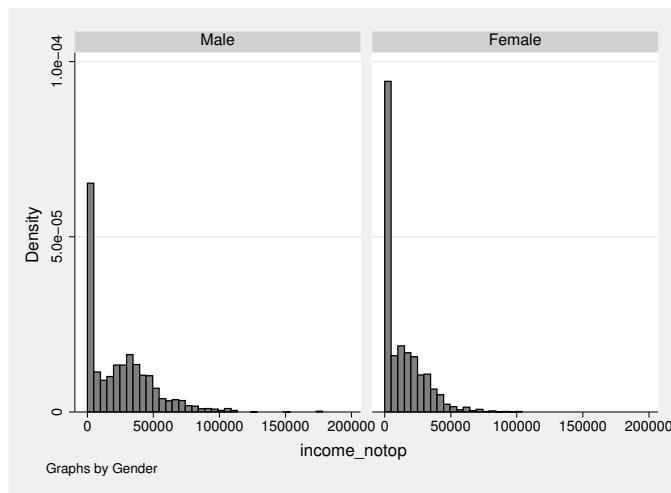
As we have said, the appearance of histograms depends on the width of the intervals, which is a function of the number of bars. With the `bin()` option, you can choose the number of bars yourself by indicating the number of bars you want inside the parentheses.

It is always a good idea to try out some other numbers. Here are the commands to generate histograms with 13 and 138 bars:

```
. histogram income_notop, bin(13)
. histogram income_notop, bin(138)
```

You can compare different distributions with histograms by using the `by()` option. The following example shows the income distribution conditioned on gender:

```
. histogram income_notop, by(sex)
```



We do not want to interpret these graphs here. Instead, we remind you that the appearance of histograms depends on the origin and width of the bars. Moreover, the necessary grouping of data often leads to surprising results—take a look at the histograms of income by state, for example. A graphical technique to circumvent some of these problems uses kernel density estimators, which we describe below.

Kernel density estimation

When displaying a distribution with a histogram, we are interested in the density at arbitrary points of the distribution. That is, we are interested in the density of persons with a value of about x . Histograms approximate the density by choosing an interval that contains x and then calculating the proportion of persons within this interval. You can see a disadvantage of this procedure in figure 7.2.⁶

6. We took the figure with permission from Fahrmeir et al. (1997, 99).

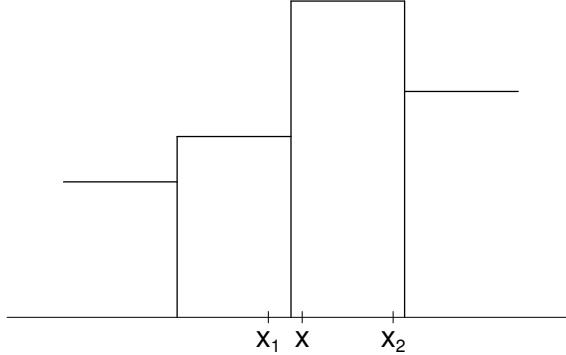


Figure 7.2. Part of a histogram

grfahrmeir.do

The figure displays a part of a histogram. There are three values of interest in the figure, x_1 , x_2 , and x . Suppose that you are interested in the density of the distribution at point x . The height of the bar at point x is based on the proportion of observations within the interval around x . Clearly, observations with a value of x_1 are not counted in calculating the height of the bar at point x . But the observations at point x_2 , which are farther away from x than x_1 is, are counted. To calculate the proportion of observations with a value of about x , it would be better to use an interval that has x as its midpoint. That is, we should use an interval that ranges from $x - h$ to $x + h$. The density at x would be the proportion within that interval divided by its width ($2h$).

We can do the same for any arbitrary value of a distribution. If we graph each of these densities along the y axis, we get what is called a *sliding histogram*. Sliding histograms are special cases of kernel density estimators, which generally estimate the density at x with

$$\hat{f}_{(x)} = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x-x_i}{h}\right) \quad (7.2)$$

In the case of a sliding histogram, we define

$$K(z) = \begin{cases} \frac{1}{2} & \text{if } |z| < 1 \\ 0 & \text{otherwise} \end{cases}$$

This looks rather complicated but it is not. To fully understand this formula, consider circumstances where $z = |(x - x_i)/h|$ is smaller than 1. Assume that you want to

calculate the density at, say, 6. Further assume that a value is approximately 6 when it is not more than 2 above or below 6; that is, the quantity h is 2. Finally, assume that your distribution has values from 0 to 10.

Substitute $x_i = 9$ into $(x - x_i)/h$:

$$z = \frac{6 - 9}{2} = -1.5$$

For $x_i = 9$, the absolute value of z is greater than 1. Therefore, K becomes 0. If you try other values for x_i , you will see that $|(x - x_i)/h|$ becomes lower than 1 whenever x_i is inside the interval from $6 - h$ to $6 + h$. As a result, K takes the value 1/2 for all values within that interval and takes the value 0 elsewhere. The sum in (7.2) is therefore

$$\frac{\text{Number of observations within } [x - h, x + h]}{2}$$

If you denote the numerator—the absolute frequency within the interval j —with n_j , you can write (7.2) as

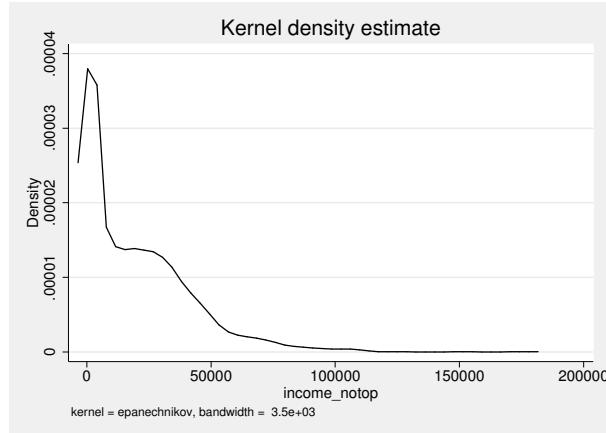
$$\begin{aligned}\hat{f}_{(x)} &= \frac{1}{nh} \times \frac{n_j}{2} \\ &= \frac{f_j}{2h} = \frac{f_j}{d_j}\end{aligned}$$

Equation (7.2) therefore is just a slightly more complicated way to express (7.1).

Equation (7.2) is nice because of $K(z)$, which is the kernel from which the name “kernel density estimate” stems. Kernel density estimates are a group of estimates for the density at point x , which are all calculated with (7.2). They differ only in the definition of $K(z)$. The rectangular kernel used for sliding histograms is only one case. Its main feature is that all observations within the observed interval are equally important for estimating the density. More common are kernels that treat observations closer to x as more important. However, with many observations, the different kernel density estimates do not differ very much from each other.

In Stata, kernel density estimates are implemented in the command **kdensity**. For the easiest case, simply enter the name of the variable for which you want to estimate the densities. Here we generate a graph displaying densities that were calculated with an *Epanechnikov* kernel:

```
. kdensity income_notop
```



There are eight different kernels available. They are listed in the online help and are more fully described in [R] **kdensity**. The **kernel(rectangle)** option, for example, calculates the densities with the rectangular kernel:

```
. kdensity income_notop, kernel(rectangle)
```

As stated, the choice of kernel typically does not affect the estimation of the density curve much. Both commands display the skewed income distribution.

Often the width of the interval ($2h$) used for the density estimation is more critical than the choice of kernel. If you do not provide a value, Stata calculates some *optimal* value, which some consider to be slightly too high for skewed or multimodal distributions. The **bwidth()** option can be used to change the interval half-width (h). If you scale down the optimal value to, say, $h = 1,000$, you will get a less smooth display:⁷

```
. kdensity income_notop, bwidth(1000)
```

To compare distributions with kernel density estimators, you can draw the curves for different distributions on one graph. You can do this by applying the **graph twoway** plottype **kdensity**.

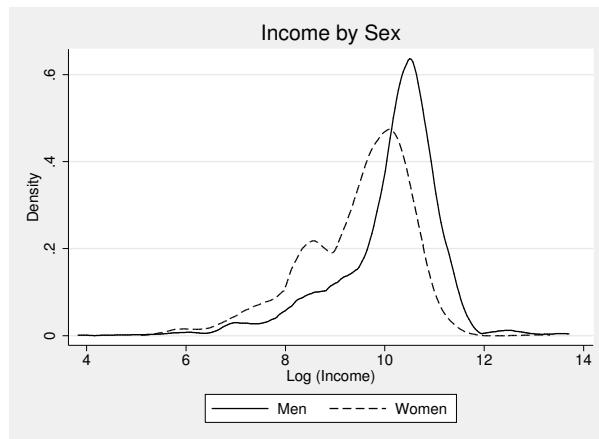
Let us demonstrate this by comparing the logarithmic income of men and women. First, we create the logarithmic income:

```
. generate linc = log(income)
```

7. See Marron (1988) for a discussion of different algorithms for selecting the interval width.

We then plot the log-income densities for men and women by applying an `if` qualifier to the `kdensity` plot of `graph twoway`. See chapter 6 for a detailed explanation of the other options used here:

```
. graph twoway || kdensity linc if sex == 1
> || kdensity linc if sex == 2
> || , title(Income by Sex) ytitle(Density) xtitle(Log (Income))
> legend(order(1 "Men" 2 "Women"))
```



Overlaying density curves allows you to compare different distributions more easily than is possible with histograms: the density curve for men is steeper and higher than the one for women. Moreover, the level of log income with the highest density is farther to the right for men than for women. Thus the incomes of women are somewhat more uniformly distributed at the lower end of the income scale, whereas the incomes of men are more concentrated around the average.⁸

Unlike histograms, kernel density estimates are not affected by the selection of the origin. And they do look much nicer than histograms. However, kernel density estimates share one disadvantage with histograms: they are affected by the selection of the interval width. A graphical technique that attempts to circumvent those problems is the quantile plot, which is discussed below.

Quantile plot

Quantile plots display marker symbols for each observation in the dataset, using the value of the observation as the y coordinate and the cumulative proportion of observations in the sorted list of values at that point as the x coordinate. The symbols form a curve from which we can infer the shape of the distribution, relative densities, repeated values, and outliers.

8. Be careful in interpreting the graph. It is useful for comparing the *forms* of the curves, but do not attach significance to the *difference* between them, because that can lead to an erroneous interpretation (Cleveland 1994, 227–230).

An example will help make things clearer. Take a look at the following data on the birth years of 10 people:

```
. preserve
. use qplot, clear
. list
```

We have sorted the data by birth year, making it easier to explain. The proportion of any observation is $1/10$.

To begin, look at the lowest value of birth year. This observation has the value $x_1 = 1901$, which is going to be the y coordinate of the first plot symbol. The x coordinate is the cumulative proportion up to this first observation, which is 0.1. The first marker symbol therefore is plotted at the position (0.1, 1901).

Now look at the second observation. This observation has the value $x_2 = 1902$, which again will be the y coordinate. The x coordinate is the cumulative proportion up to the second observation, which is $0.1 + 0.1 = 0.2$. Hence, the coordinate for the second marker symbol is (0.2, 1902).

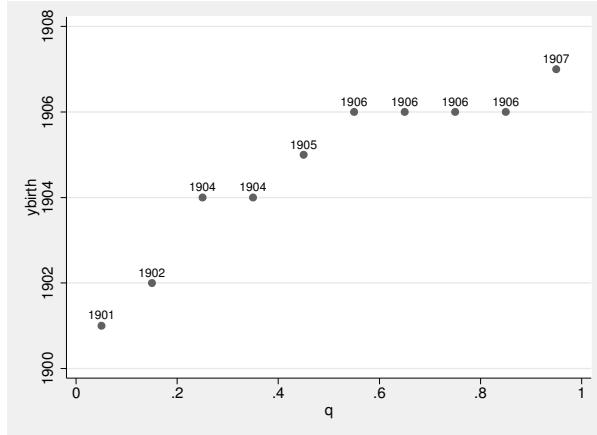
We could proceed like this for any other value of the birth year, but we should let Stata do the work for us. For that, we need a formula to calculate the x coordinates. We use $(i - 0.5)/n$, with i being the position of the observation in the sorted list of values and n being the number of observations. Subtracting 0.5 in the formula is not necessary—and we did not do it above—but is a convention among statisticians (Cleveland 1994, 137).

Here is the Stata command that calculates the formula and stores the results in a variable. If you have read chapter 5, you should be able to use it with no problem:

```
. generate q = (_n - .5)/_N
. list
```

To generate a quantile plot, you graph both quantities in a two-way graph:

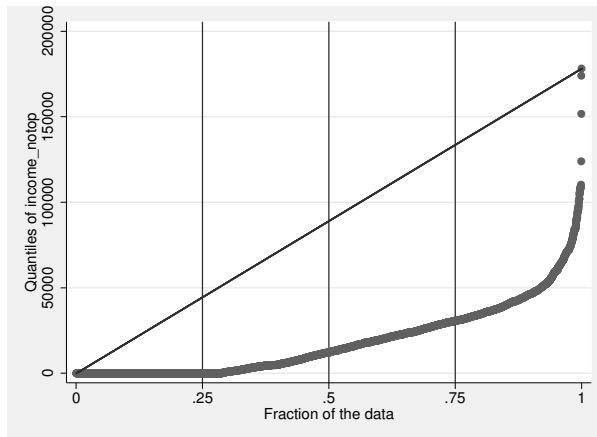
```
. scatter ybirth q, mlabel(ybirth) mlabposition(12)
```



There are four observations for 1906. The marker symbols for these observations form a horizontal line. That may not appear very impressive, but you should remember that equal values always form horizontal lines in quantile plots. Marker symbols that display low slope correspond to data regions with high density. That is, the steeper the curve formed by the marker symbols, the lower is the density of values in the corresponding data.

The power of quantile plots can be demonstrated more compellingly with a larger number of observations. Take the quantile plot of income in our main data as an example. To produce this plot, you can use the command `quantile`, which calculates the x coordinates and displays the graph in one step. Moreover, you can use most of the general options for two-way graphs, including `by()`, to compare distributions between groups.

```
. restore
. quantile income_notop, xline(.25 .5 .75)
```



This quantile plot has four regions. The first region is a flat line at 0 showing that `income` has many observations with a value of 0. The second region is a region with relatively high density that starts with incomes just above €0 up to about €50,000. Above there is a region with low and declining density until around €120,000. Finally, there are four extreme values with even higher incomes. Because the density is high for low values and low for high values, the distribution is skewed to the right. You can see this also from the fact that all symbols are below—to the right of—the main diagonal. The distribution would be skewed to the left if they had been above the main diagonal.

A main advantage of quantile plots is that they provide a marker symbol for every observation. It is therefore possible to identify repeated values (all flat regions) and single outliers. Another feature of the plot is that you can read off arbitrary quantiles from the quantile plot. The x coordinate is the p -value of the quantile, so the 0.25 quantile—the first quartile—of income is the y -value that corresponds to 0.25 on the x axis. It is 0 here. The 0.5 quantile or median is around €12,000.

Finally, here are two further examples for quantile plots. First, a quantile plot for a uniformly distributed random variable:

```
. set seed 42
. generate r1 = runiform()
. quantile r1
```

Second, a quantile plot for a normally distributed random variable:

```
. generate r2 = rnormal()
. quantile r2
```

Comparing distributions with Q–Q plots

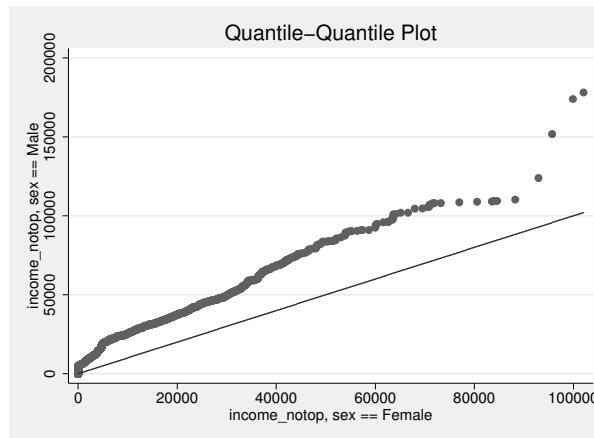
A *Q–Q plot*, or *quantile–quantile plot*, is a graphical method for comparing two distributions. Q–Q plots are very simple when both distributions have the same number of observations. In that case, you sort the values of both distributions and then plot the lowest value of the first distribution against the lowest value of the second, and so on. Usually, there are unequal numbers of observations, which is slightly more complicated, but the general idea stays the same. In the first step, you calculate the quantity $(i - 0.5)/n$ for each observation of the distribution with fewer observations and then calculate the same quantiles for the other distribution. Finally, you plot the lowest value of the smaller dataset against the quantile of the larger dataset that corresponds to the lowest value of the smaller dataset, and so forth. Thus in the case of unequal numbers of observations, there are as many marker symbols as there are values in the smaller of the two datasets.

To compare the income of men and women with a Q–Q plot, you need to generate separate income variables for men and women. You can do this by using the `separate` command shown in section 7.3.2.

```
. separate income_notop, by(sex)
```

To generate the Q–Q plot comparing these two distributions, simply enter

```
. qqplot income_notop1 income_notop2
```



We can see that most data points are to the left of the main diagonal. This means that the values of the variable on the *y* axis usually are higher than those of the variable on the *x* axis. Here this means that men earn more than women. We also see that income inequality becomes particularly strong for incomes above €100,000.

7.4 Exercises

1. Get data from the National Health and Nutrition Examination Study (NHANES) by using the following command:

```
. webuse nhanes2, clear
```

2. Produce a frequency table of health status (`hlthstat`).
3. Produce one-way frequency tables of the following variables with just one command: `psu`, `region`, `smsa`, `sex`, and `race`.
4. Investigate whether men or women in this subsample are of better health by using a cross-tabulation of gender and health status.
5. Investigate how health differs between races by using cross-tabulation and the chi-squared statistic.
6. Produce a table holding the mean weights of observations by race and gender. Do not show any value other than the mean weights.
7. Produce a dot plot for the figures of the previous problem.
8. Produce a histogram of weight with a normal overlay.
9. Produce a kernel density plot of weight with a normal curve overlay.
10. Compare the distribution of the weights between whites and blacks by using a Q–Q plot. Do this for men and women separately and show both plots in one figure.
11. Use a box-and-whisker plot to compare the distribution of the systolic (`bpsystol`) and diastolic (`bpdiast`) blood pressure across race and gender.

8 Statistical inference

So far we have explored distributions, examined data through tables and graphs, and compared subgroups in the datasets we provided. Any statement we made about differences between subgroups or the distribution of certain variables was with respect to the data we observed. The term “statistical inference” is used for the process of drawing conclusions from an observed dataset to something that is more than just the observed data. There are two aspects of inference we discuss here: descriptive inference and causal inference.

Descriptive inference takes place when you calculate a statistic using data from a sample and then use this information to make estimates about the same statistic in the population from which the (random) sample was drawn. Take as an example the income inequality between men and women. We have shown before that in our data, male earnings are on average much higher than female earnings. We arrived at this conclusion using a statistic—namely, the difference in average earnings of men and women—calculated from a sample. What does this value indicates about the difference in earnings between all German men and women? An answer to this question can be given using the techniques discussed in section 8.2.

In other settings, a different logic is needed when we think about inference. For example, during the sinking of the *Titanic*, women and children queued up for places in lifeboats. It later turns out that 97% of the first-class women and children survived, while this was the case for only 42% of the third-class women and children. You might infer from this result that the order in the queue was created by some causal process related to the social status of the passengers (that is, the crew locked the doors between the lower cabins and the upper deck). However, before you make this inference, you should check whether the observed difference in the survival rate could also arise by chance. That is, you should wonder, What are the odds of this result if the order of the queue would have been randomly assigned? Section 8.3 discusses such causal inference in more detail.

Both types of statistical inference deal with random processes. Random sampling is a prerequisite for descriptive inference, and causal inference builds on the assumption that the data are created by a mixture of both a systematic and a random process. To understand these random processes, it is very helpful to be able to re-create them. We therefore show in section 8.1 how to generate random numbers and random variables in Stata, how to draw samples, and how to build sampling distributions. Readers familiar with these steps can jump ahead to section 8.2 to learn how to estimate standard errors in Stata for various types of samples with complete and missing data.

8.1 Random samples and sampling distributions

To fully understand the following sections about sampling distributions and inference, it is good to have some hands-on experience in drawing random samples. Generating and applying random numbers is key in the process of random sampling; thus we will start out with a little detour on how to do just that in Stata. It might take you a bit to see why you need to know about random numbers; if that is the case, just bear with us and it will become clear.

8.1.1 Random numbers

Stata has a built-in random-number generator. The random-number generator can be accessed by various commands and functions. By far, the most important function for creating random numbers is `runiform()`, which returns random numbers in $[0, 1]$ (that is, between 0 and almost 1) from a uniform distribution. We can use this function to display just one randomly selected number:

```
. display runiform()
.76668427
```

In our case, the number is around 0.767. Most likely, you will see a different number on your screen; after all, this is a random number. However, we both could get the same value because Stata's random-number generator is not entirely random. In fact, the random numbers created are completely determined by an initial value, the random-number seed. If Stata uses the same seed on two different computers, it will always produce the same random numbers afterward. You can set the initial value of the random-number seed by using the command `set seed` together with a positive integer number. Let us use 42 and prove that we all get 0.14358038:

```
. set seed 42
. display runiform()
.14358038
```

Setting the random number seed to 42 does not mean that `runiform()` produces the value 0.144 all the time. It only means that the first number created by `runiform()` after having set the seed to 42 is always 0.144. If we issue the `display` command again, we will get a different number than we got the first time, but now we all will get the same different number, 0.99569016:

```
. display runiform()
.99569016
```

The function `runiform()` can be used to create random numbers for arbitrary intervals. For example, multiplying the random number by 6 leads to a random number between 0 and almost 6:

```
. display 6*runiform()
2.7913871
```

If you want integer numbers from 1 to 6, you can type

```
. display 1 + int(6*runiform())
6
```

which uses the function `int()` that returns the integer part of the number inside the parentheses. More generally, you can create random numbers from the interval $[a, b]$ (that is, $\geq a$ and $< b$) with $a + (b - a)r$, where r is the result of the `runiform()` function, and you can create integer numbers from the interval $[a, b]$ with $a + \text{int}\{(b - a + 1)r\}$. Here are two examples that show a random number and a random integer between 6 and 23:

```
. display 6 + (23-6) * runiform()
11.956807
. display 6 + int((23-6+1) * runiform())
15
```

8.1.2 Creating fictitious datasets

Stata not only allows you to draw single numbers at random but also allows you to create datasets with random variables. To do so, we start by creating a dataset with 1,000 empty observations:

```
. set obs 1000
obs was 0, now 1000
```

The command `set obs` creates observations. The number behind the command defines the numbers of observations to be created, here 1,000. We can add variables to those empty observations. For example, we can generate a random variable with integer numbers between 1 and 6 by using the expression from above together with `generate`:

```
. generate x1 = 1 + int(6*runiform())
```

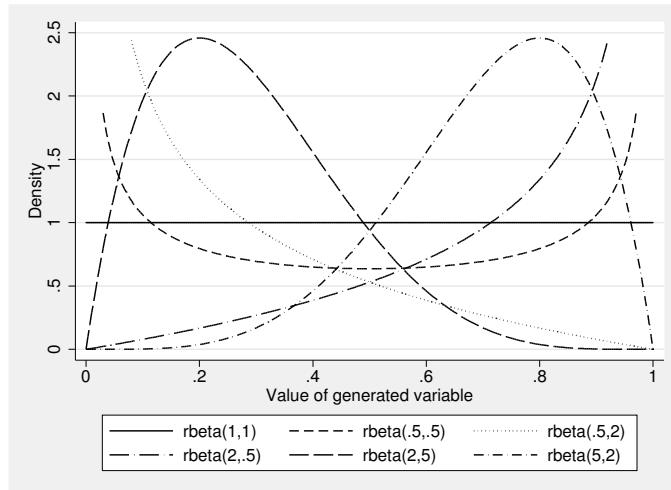
The new variable can be examined with

<code>. tabulate x1</code>				
x1	Freq.	Percent	Cum.	
1	154	15.40	15.40	
2	175	17.50	32.90	
3	183	18.30	51.20	
4	179	17.90	69.10	
5	141	14.10	83.20	
6	168	16.80	100.00	
Total	1,000	100.00		

or any of the other tools we described previously in this book. From the result above, we see that the new variable holds all possible integers between 1 and 6, and that all of these values appear approximately with a similar, yet not identical, frequency. The

frequencies are similar because the random-number generator produces values from a uniform distribution. The frequencies are not identical because they are created randomly. To the extent that Stata's random-number generator works well (it does!), the differences of the frequencies are due to chance.

It is possible to generate random variables with other distributions, too. A very powerful tool for this is the random-number function `rbeta(a,b)`, which creates random numbers from a Beta distribution. The function lets you specify two parameters, α and β . Depending on the value chosen, you can generate variables with the following distributions (among others):



Source: `grbeta.do`

Figure 8.1. Beta density functions for different settings of α and β

α	β	Shape	Example
1	1	uniform	<code>rbeta(1,1)</code>
< 1	< 1	U-shaped	<code>rbeta(.5,.5)</code>
1	> 1	decreasing	<code>rbeta(1,3)</code>
> 1	1	increasing	<code>rbeta(5,1)</code>
> 1	> α	skewed right	<code>rbeta(2,5)</code>
> β	> 1	skewed left	<code>rbeta(5,2)</code>

Figure 8.1 shows the density function of the Beta distributions for various settings of α and β . If you create random variables with the indicated settings, the distribution of the created variables will be similar to that shown in the figure.¹

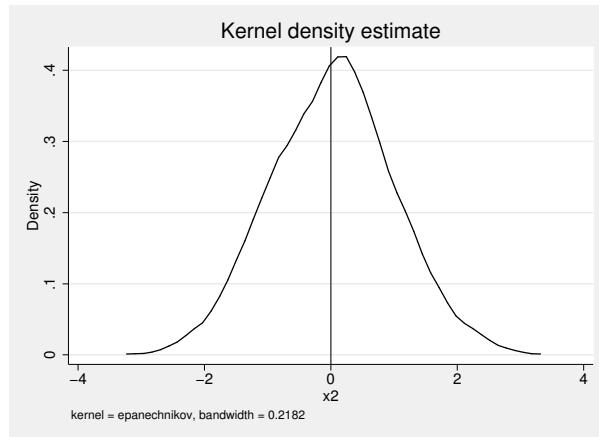
1. Examples mirror the examples on Wikipedia; see the permanent link
http://en.wikipedia.org/w/index.php?title=Beta_distribution&oldid=448380008.

Many other distributions can be created by applying specific transformations on random numbers between 0 and almost 1 in one way or another. One of the more complicated transformations uses the results of the function for the *inverse cumulative standard normal distribution*, `invnormal()`. If you apply this transformation on a variable with uniformly distributed values between 0 and almost 1,

```
. generate x2 = invnormal(runiform())
```

you end up with a variable that has a mean of approximately 0, a standard deviation of 1, and a bell-shaped distribution:

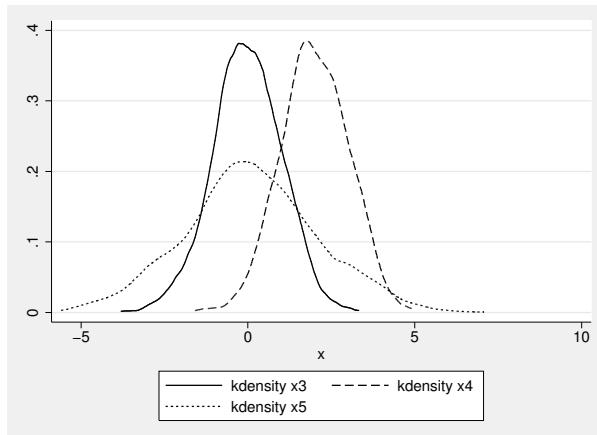
```
. kdensity x2, xline(0)
```



The bell-shaped distribution of our newly generated variable `x2` is what statisticians would describe as a “normal distribution”. Because it has 0 mean and standard deviation of 1, they would even talk of a “standard” normal distribution. Note that the curve is a little jerky because we only used 1,000 observations. Had we used many more, the graph would look much smoother.

Because variables with normal distributions are very important in statistics, Stata has the function `rnormal()` that allows you to directly create random numbers from a normal distribution. Without further arguments, the function creates random numbers from a standard normal distribution. If you put a number inside the parentheses, this number will become the mean of the random values to be created. Likewise, if you place two numbers inside the parentheses, the first number will become the mean, and the second number becomes the standard deviation of the numbers created. Here is an illustration for each of these uses:

```
. generate x3 = rnormal()
. generate x4 = rnormal(2)
. generate x5 = rnormal(0,2)
. twoway kdensity x3 || kdensity x4 || kdensity x5
```



So far, all of our examples have created values randomly while controlling certain aspects of the distributions. As a result, we got variables that have the intended distribution but are fairly unrelated among each other. However, it is also possible to create variables that are interrelated in some respect. You can, for example, create a fictitious variable `income` with a mean of \$10,000 and a standard deviation of \$15,000 if another fictitious variable, `men`, is 0; `income` can have a mean of \$20,000 and standard deviation of \$40,000 if `men` is 1. A simple way to achieve this is

```
. generate men = int(runiform()*2)
. generate income = rnormal(10000,15000) if !men
(497 missing values generated)
. replace income = rnormal(20000,40000) if men
(497 real changes made)
. tabulate men, sum(income)
```

men	Summary of income		
	Mean	Std. Dev.	Freq.
0	9631.7472	15630.947	503
1	22055.794	38172.359	497
Total	15806.499	29746.606	1000

That is to say, we have implemented a data-generating process in which the average income is about \$10,000 higher for men than for women, while individual incomes vary randomly around the means.

8.1.3 Drawing random samples

To understand how random numbers can be used to draw samples from a dataset, load `berlin.dta` into memory:²

```
. use berlin, clear
```

This file has 2,932,818 observations, one for each registered resident in Berlin in 2010 between the ages 18 and 89. The file contains one variable, the year in which the resident was born. Before we proceed, note that the average year of birth of the observations in `berlin.dta` is 1962, or to be precise, 1962.252:

<code>. summarize ybirth</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
ybirth	2932818	1962.252	17.8411	1921	1992

Now assume you want to randomly select 0.1% of the observations from that dataset. This can be done with the Stata command `sample` or with the already familiar function `runiform()`. While the command `sample` is not as flexible as the function `runiform()`, it is very easy to use. The command

```
. sample 0.1
(2929885 observations deleted)
```

deletes 2,929,885 randomly selected observations from the dataset so that 2,933 observations (approximately 0.1% of 2,932,818 observations) remain. Summarizing `ybirth` again, we realize that the average year of birth of the random sample is only slightly different from what it was before:

<code>. summarize ybirth</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
ybirth	2933	1962.371	17.73615	1921	1992

The same result can be achieved with the function `runiform()`. We start by loading the dataset again so that we have full access to our population:

```
. use berlin, clear
```

We then create the new variable `r` using the `runiform()` function. Because the values in the new variable are uniformly distributed, approximately 0.1% of the values created are below 0.001.

```
. generate r = runiform()
. count if r<=.001
2881
```

2. Make sure that your current working directory is `c:\data\kk3`; see page 3.

To draw a random sample, we can therefore drop all observations with values of the new variable `r` above 0.001. Or we just look at the mean of `ybirth` for the sample by using the `if` qualifier without deleting any observations from the dataset:

<code>. summarize ybirth if r<=0.001</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
<code>ybirth</code>	2841	1961.928	17.92092	1921	1992

Instead of drawing a sample based on a set percentage, we could also draw a sample of exactly, say, 2,930 observations from the file `berlin.dta`. With the three commands below, we create a uniformly distributed random variable, sort the cases by that variable so that they are now in random order, and mark the first 2,930 cases with the variable `srs`:

```
. replace r = runiform()
. sort r
. generate srs = _n<=2930
```

From

<code>. summarize ybirth if srs</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
<code>ybirth</code>	2930	1962.414	17.92464	1921	1992

we see that we have in fact selected 2,930 observations.

Either way, these sampling methods are called simple random sampling (SRS). Simple random samples are conceptually the most simple ones. With SRS, each observation has a known and equal probability for being selected into the sample of n/N , with n being the sample size and N being the size of the population. In our last example, this sampling probability is $2930/2932818 \approx 0.1\%$ for each observation of `berlin.dta`.

Another feature of SRS is that estimates of the population mean derived from simple random samples mean have a known *sampling distribution*. We will explain this somewhat technical feature in the next section. Before we start, we would like to mention that simple random samples are rare in practice because other sampling designs are more practical or more cost efficient (see section 8.2.2).

8.1.4 The sampling distribution

In section 8.1.3, if you look at the outputs of the `summarize` commands for the three different 0.1% simple random samples, you see that the means of `ybirth` differ slightly between the samples. Imagine now that you draw 500 simple random samples. What values would you get for the means of `ybirth` in all those samples, then?

Actually, you do not have to imagine that. You already learned in this book all the tools needed to simulate that process. See our example below. We create a loop of 500 iterations (see section 3.2). Inside the loop, we draw simple random samples from the

`berlin.dta` population as before, although this time we only select approximately 100 observations. We then summarize `ybirth` for the sample and collect the mean of each repetition in the new variable `smplmeans` alongside the existing data. Before you start typing the commands, note that they will take a while and that issuing the command `set more off` before starting the loop is convenient:

```
. generate smplmeans = .
. forvalues i = 1/500 {
.     summarize ybirth if runiform() < (100/_N), meanonly
.     replace smplmeans = r(mean) in `i'
. }
```

When the loop finishes, we end up with 500 observations stored in the variable `smplmeans`. Each of these 500 observations holds the arithmetic mean of one sample. Or to say it somewhat differently: `smplmeans` holds the distribution of the sample means (also known as the sampling distribution of the mean).

In sampling theory, the term “sampling distribution” is used to describe the distribution of a statistic (such as the arithmetic mean) in an infinite number of random samples. We would be sitting here a long time if we were to create an infinite number of random samples, but for the purpose of explaining the features and the use of sampling distributions, 500 repetitions is close enough to infinity. Thus let us take a look at the distribution of the 500 means stored in `smplmeans` by summarizing both the original variable `ybirth` and the means of the samples in `smplmeans`.

. summarize ybirth smplmeans					
Variable	Obs	Mean	Std. Dev.	Min	Max
ybirth	2932818	1962.252	17.8411	1921	1992
smplmeans	500	1962.236	1.785025	1957.325	1967.325

The mean of `ybirth`, 1962.252, is the average of the year of birth in the `berlin.dta` population (that is, the *true value*). Normally, we would not know this value but would like to estimate it using just one of our 500 samples. If we look at the average of the means of all of our 500 samples, we see that this average is very close to the population mean: 1962.236. This is not a coincidence. Probability theory shows that the mean of the sample means for an infinite number of simple random samples is equal to the population mean. The sample mean of a simple random sample is therefore said to be an unbiased estimate of the (normally unknown) population mean. If the mean of the sample means would differ from the population mean, the estimate would be biased.

However, being unbiased is not the only characteristic a method should have. This becomes clear by looking at the minimum and maximum values of the simulated sampling distribution. According to these values, at least one of the samples produced a mean of `ybirth`, 1957.325, that was more than five years below the true value, and yet another one produced a mean that was almost five years above the true value.

In practice, where we have just one sample, we are therefore faced with the question of how good our point estimate really is. Have we ascertained a sample that produced a point estimate that is close to the true value or one that is far off? In this situation, it

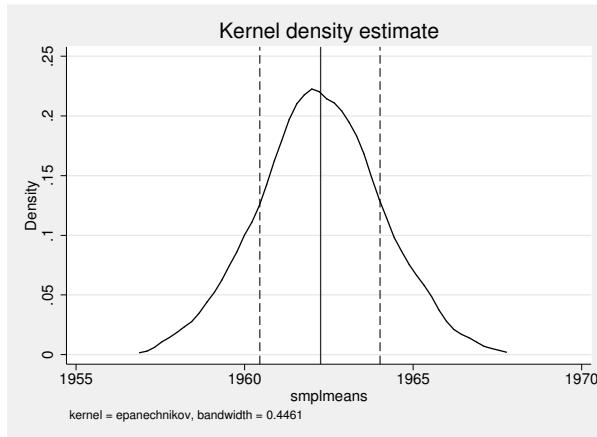
would be good if we knew how much, on average, the point estimate varies around the true mean. One can say that the more it varies, the less we should trust in our present point estimate. The sampling distribution provides us with a good measure for the level of trust that we can have in our observed point estimate: its standard deviation. For our simulated sampling distribution, the standard deviation is 1.785025, which means that the average year of birth estimated in simple random samples of approximately 100 observations from a population of 2,932,818 Berlin residents deviates, on average, 1.785025 years from the true value.

The standard deviation of the sampling distribution is commonly termed the standard error, distinguishing it from the standard deviation of a variable in a dataset. Generally, we wish to apply methods that have small standard errors because those methods make it more likely that the result ascertained in our present sample is close to the true value in the population. In statistics, the amount of trust that we can have in our point estimate is termed “efficiency.” A method that leads to point estimates with small standard errors is termed an efficient method, while those with large standard errors are inefficient. If it can be shown that no other method leads to a point estimate with smaller standard errors, the method is sometimes called the best method.

The standard error is the most fundamental concept of statistical inference. It is the basis for all other procedures of statistical inference, such as creating confidence intervals and significance tests. Estimating the standard error correctly is therefore the core problem of statistical inference. We will deal with this problem in the next section.

However, before we do this, we would like to take a more thorough look at the distribution of `smplmeans` by using a kernel density plot (see section 7.3.3), where we mark the true mean and the standard deviation with vertical lines (see section 4.1 for the meanings of `r(mean)` and `r(sd)`):

```
. summarize ybirth
. local truemean = r(mean)
. summarize smplmeans
. local XplusSE = r(mean) + r(sd)
. local XminusSE = r(mean) - r(sd)
. kdensity smplmeans, xline(`truemean') xline(`XplusSE' `XminusSE', lpattern(dash))
```



The figure shows that the means of the 500 samples vary symmetrically around the true mean of *ybirth*, whereby means close to the true value are more frequent than those far out. A large proportion of the sample means are within the boundaries of one standard error above and below the mean, marked here with dashed lines. More specifically, from

```
. generate limit1 = inrange(smplmeans,`XminusSE',`XplusSE') if !missing(smplmeans)
(2932318 missing values generated)
. tabulate limit1
```

limit1	Freq.	Percent	Cum.
0	155	31.00	31.00
1	345	69.00	100.00
Total	500	100.00	

we see that actually about 69% of all sample means are within one standard deviation above and below the mean; if we draw a larger number of random samples, that figure would be around 68%. Likewise, one can expect that about 95% of all sample means are within two standard deviations above and below the mean, and about 99% are within three standard deviations above and below the mean. If you look at the respective proportions among our 500 sample means, you will see that their distribution is very close to these numbers:

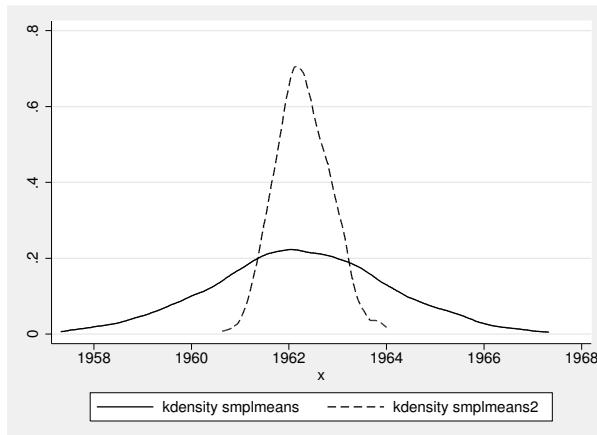
```
. summarize smplmeans
. forvalues i = 2/3 {
.     local Xplus`i`SE = r(mean) + `i`*r(sd)
.     local Xminus`i`SE = r(mean) - `i`*r(sd)
.     generate limit`i` = inrange(smplmeans, `Xminus`i`SE', `Xplus`i`SE')
>         if !missing(smplmeans)
.     tabulate limit`i'
. }
```

Overall, the sampling distribution of the sample means looks very much like a normal distribution. The central limit theorem shows us that for an infinite number of replications, the sampling distribution of sample means from simple random samples has a normal distribution. Or to say it in more technical terms, when we draw a *large* simple random sample of size n from a population, and we observe a characteristic that has a mean of μ and a standard deviation of σ in the population, then the sampling distribution has a normal distribution with a mean of μ and a standard deviation (that is, the standard error) of σ/\sqrt{n} :

$$N\left(\mu, \frac{\sigma}{\sqrt{n}}\right)$$

This is the case no matter what distribution the characteristic in the population has, though the further it is away from normal, the larger the sample size should be. If you would like to see what effect a different sample size has on the sampling distribution of `ybirth`, you can rerun the commands we used above (page 209) with a different sample size, and you can graph the new sampling distribution on top of the distribution of `smplmeans`. If we do this for sample sizes of approximately 1,000 observations, we see that the sampling distribution becomes much narrower:

```
. generate smplmeans2 =
. forvalues i = 1/500 {
.     summarize ybirth if runiform() < (1000/_N), meanonly
.     replace smplmeans2 = r(mean) in `i'
. }
. twoway || kdensity smplmeans || kdensity smplmeans2
```



The central limit theorem simplifies the estimation of the standard error for sampling means from simple random samples. However, it should be noted that this is not necessarily true for all kinds of point estimates, all kinds of samples, and all kinds of variables. Real statistical problems start here.

8.2 Descriptive inference

Knowing about sampling, sampling distributions, and standard errors is key to descriptive inference. As explained in the last section, the standard error is the standard deviation of the sampling distribution. As also explained in the last section, the sampling distribution is a theoretical concept that cannot be observed directly. What we observe in practice is one single sample. We therefore cannot measure the standard error using the standard deviation of means calculated in a huge number of independent samples; we need to estimate it from the single sample that we have. The only pieces of information that we have to do this are the values of the variables in our dataset and the information about how the sample was drawn.

The following sections explain how to use this information for the estimation of standard errors, and then go on to explain the inferences that can be made from the estimated standard errors.

To continue, please load `data1.dta` into memory:³

```
. use data1, clear
(SOEP 2009 (Kohler/Kreuter))
```

8.2.1 Standard errors for simple random samples

The dataset `data1.dta` contains information from a survey of 5,411 of the roughly 82 million German inhabitants. The respondents for the survey were selected by a sample that was *not* a simple random sample; however, for the sake of simplicity, we start as if it were a simple random sample. How could we use the information in `data1.dta` to estimate the standard error?

Following our earlier discussion of the central limit theorem, the standard error of the mean of some variable X is equal to the standard deviation of that variable in the population divided by the square root of the number of observations of the sample. Hence, we need the standard deviation of X in the population, which we do not know. However, if we take as our best guess the standard deviation of X in the sample, we can estimate the standard error in direct analogy to the central limit theorem with

$$\widehat{SE}(\bar{X}) = \frac{SD(X)}{\sqrt{n}} \quad (8.1)$$

where $SD(X)$ is the standard deviation of X in the sample (that is, our estimate for σ , the standard deviation of X in the population).

3. Make sure that your current working directory is `c:\data\kk3`; see page 3.

You can use this formula to obtain the standard error from the output of `summarize`:

```
. summarize ybirth
      Variable   Obs      Mean    Std. Dev.      Min      Max
ybirth        5411  1959.493   18.12642   1909     1992
. display r(sd)/sqrt(r(N))
.24641851
```

According to this output, the standard error of the average year of birth in simple random samples of size 5,411 from the entire German population is estimated to be 0.246; in other words, if we drew a large number of simple random samples on the German population of 2009, the mean year of birth in the sample would differ on average about 0.25 years from the mean year of birth in the population.

The estimate of the standard error for the sample mean is even more accessible in the output of the command `mean`. The command calculates the estimated standard errors of the sample mean for a list of variables. If specified without further options, the command displays the standard error assuming a simple random sample. Hence, in this simple case, the command reproduces the result of our manual calculation above:

```
. mean ybirth
Mean estimation                               Number of obs = 5411
                                               
      |      Mean    Std. Err. [95% Conf. Interval]
ybirth |  1959.493   .2464185    1959.01    1959.976
```

The command `mean` is an example of an estimation command. Stata has many such estimation commands, and what they have in common is that they all estimate standard errors of specific statistics (also known as point estimates). They also have in common that they default to estimating these standard errors under the assumption of simple random samples. This assumption is not always reasonable. In practice, most samples are not simple random samples because

- the sample was drawn with a different sampling method than SRS, or
- some sampling units dropped out of the sample either because they could not be contacted at home or declined to participate (unit nonresponse), or because they declined to answer a specific survey question (item nonresponse).

For the dataset in the file `data1.dta`, both mechanisms are present. In the following, we therefore discuss techniques to estimate a more realistic standard error.

8.2.2 Standard errors for complex samples

Earlier, we mentioned that practical reasons and cost considerations lead to sample designs that are not simple random samples. Samples that are not simple random samples we call complex samples. In the context of inference, three features about samples are particularly important because they affect the point estimates or their standard errors or both. These three features are clustering, stratification, and disproportional sampling probabilities. Each of these features can arise as a consequence of the sample design. It is therefore necessary to have at least some knowledge about various sampling methods, their sampling distribution, and how those affect the estimation of standard errors.

Typical forms of complex samples

Cluster sampling

Cluster samples are samples where we do not sample the individual units directly. Instead, we sample clusters of units, and within each selected cluster *all units* are observed. A typical example for cluster sampling is a sample of classrooms in schools, where we take observations on every student within the selected classroom. Another example is a sample of addresses where we observe all persons living at the selected address. Cluster samples are often used as alternatives to simple random sampling when no list of individual sampling units is available but a list of clusters is.

Because the observations within a cluster tend to be somewhat more similar than observations in different clusters, cluster samples tend to show what we call *clustering*.

Two-stage and multistage sampling

Two-stage sampling is very similar to cluster sampling. For cluster sampling, we observed each unit in a cluster. For two-stage sampling, we draw a second sample inside each selected cluster. The clusters of observations selected in the first step are commonly termed the *primary sampling units* (PSUs), while the units selected within these PSUs are the *secondary sampling units*. The PSUs can be large units like election districts or city blocks, but they can also be small units such as households. A secondary sampling unit for a sample of households could be an individual person who is selected within a household.

Because the observations within a PSU cluster tend to be somewhat more similar than observations in different PSUs, two-stage samples tend to show clustering.

However, two-stage samples also often show disproportional sampling probabilities; that is, the probability of being selected is not the same for everybody. This is because members of one-person households are sampled with certainty once the household was selected, while members of larger households might or might not be selected for the sample. More generally, the probability that a person from a selected household of size k is selected into the sample is $1/k$; hence members of large households have a

smaller probability of being included in the sample. To draw valid inferences about the population, it is therefore necessary to weight the data by the reciprocal value of the selection probability.

Likewise, multistage sampling is a generalization of two-stage sampling. Consider the following three-stage sample as an example: 1) select communities from a list of communities, 2) within the selected communities, select addresses from a list of addresses, and 3) within the selected addresses, select one person from each of the addresses. It is often the case that multistage sampling is combined with a cluster sample. In our example, this would be the case if we sample one person of all those living at a selected address.

Probability proportional to size

Often researchers are interested to have certain units in their sample because those units are important. For example, in establishment surveys, large establishments with lots of revenue are often selected with a certainty. Likewise, in multistage samples, certain cities might be selected with certainty because they are so large and researchers want at least some people from those cities in their samples. A frequently used technique to reflect the “importance” of certain large units is sampling with probability proportional to size (PPS). This technique requires information about the number of units inside each PSU.

Assume you have a list of all German communities with their number of inhabitants. Some of these communities are pretty big—Berlin, for example, had 3,442,675 inhabitants in 2009. Other communities are rather tiny—the smallest German community, Dierfeld in Rhineland-Palatine, had just eight inhabitants in 2009.⁴ Germany as a whole had 81,802,257 inhabitants in 2009. Hence, when selecting communities proportional to size, the selection probabilities of Berlin and Dierfeld should be 4.2% and $9.7 \times 10^{-6}\%$, respectively:

```
. display 3442675/81802257
.04208533
. display 8/81802257
9.780e-08
```

Now imagine that you select communities with probability proportional to size, and afterward you select, say, six inhabitants in each community. The overall selection probability for people from Berlin and Dierfeld would then be the product of the probability of selecting the respective community and the probability of selecting a person from the respective community. Hence,

```
. display 3442675/81802257 * 6/3442675
7.335e-08
. display 8/81802257 * 6/8
7.335e-08
```

4. The list of German communities can be downloaded from <http://www.destatis.de>.

This overall probability is the same for both people from Berlin and people from Dierfeld. It is possible—and even quite common in practice—that multistage samples use sampling with probability proportional to size for one of the stages.

As it should be clear from the discussion, PPS samples do not contain disproportional sampling probabilities, but they still show clustering.

Stratified sampling

Stratified sampling takes place if you first divide your population into specific subgroups (called *strata*) and then draw random samples for *each* of these subgroups. This sounds very similar to two-stage sampling but is fundamentally different. In stratified sampling, a sample is drawn for each stratum, while in two-stage sampling some but not all PSUs are selected. Stratified sampling is typical for samples on the population of the European Union, where random sampling is done in each country separately. However, it also happens frequently in samples for just one country, when the researchers wish to be sure that the sample contains enough observations from each stratum.

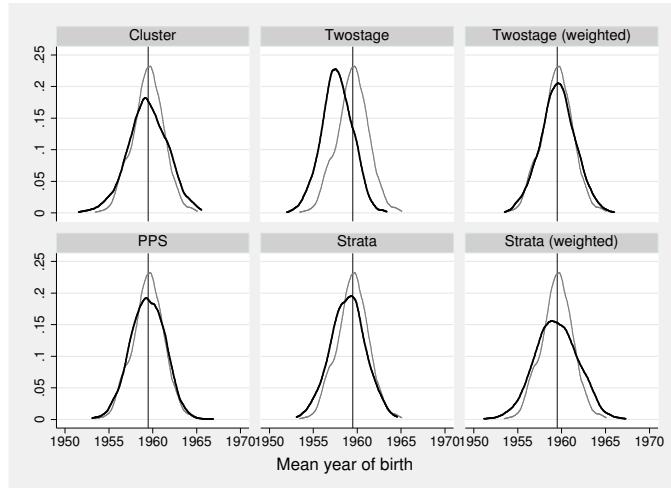
The German Socio-Economic Panel (GSOEP) data that we used for `data1.dta` is an example. The sample of the GSOEP was done in two strata separately: East Germany and West Germany. Then within West Germany, two more strata were used and samples were drawn separately for native and immigrant citizens.

Stratified sampling is not an alternative sampling method to the methods discussed so far. Instead, one may apply one of the methods discussed previously for each strata or even use different methods for each strata.

Sampling distributions for complex samples

We wrote above that clustering, stratification, and disproportionality may affect point estimates and their standard errors, and this is illustrated in figure 8.2. For the figure, we have drawn small subsamples of approximately 100 observations from the 5,411 observations of `data1.dta`—just like we did in section 8.1.3. However, this time, we did not draw simple random samples but a cluster sample, a two-stage sample, a PPS sample, and a stratified sample. For each of these sampling methods, we draw 1,000 samples, and then calculate the average year of birth in each of the samples. The figure displays the distribution of the averages (that is, the sampling distribution) for each sampling method by using a kernel density curve in comparison with the sampling distribution of a simple random sample of (approximately) the same size.

If a sampling method leads to an unbiased estimation, the peak of the curve will be approximately on the vertical line, which shows the average year of birth for the population of the 5,411 observations in `data1.dta`. The stronger the offset, the stronger the bias is. Similarly, the broader the curve (or the lower the peak), the lower the efficiency of the sampling method.



Source: `ansamples.do`

Figure 8.2. Sampling distribution of the mean year of birth for various sampling methods (black line) in comparison with the sampling distribution under SRS (gray line)

A few things are worth noting:

- The sampling distributions differ between the sampling methods. It is therefore *not* possible to estimate the standard error for all sampling methods with the same formulas.
- Two sampling methods lead to biased estimation of the population mean: two-stage samples (large bias) and stratified sampling (small bias).
- After applying weights, the bias disappears but at the cost of inefficiency (larger standard errors).
- Clustered samples often have larger standard errors, because people within the same cluster are more similar to each other. Thus with the same sample size, less information is obtained from a clustered sample compared with a simple random sample.
- Stratified samples have smaller standard errors than simple random samples when the variables of interest vary by strata. Here this was not the case. In practice, stratified samples are often also samples with unequal selection probabilities; thus through weighting, the gained efficiency is likely lost (which you see here in the graphs as well).

Using Stata's **svy** commands

For a general discussion of techniques to estimate standard errors for complex samples, we refer the interested reader to Valliant, Dever, and Kreuter (2012). But here we do want to show you how to obtain the correct standard errors for complex samples using Stata. Stata's default for estimating standard errors in complex surveys is Taylor linearization. There are other methods available—such as bootstrap, balanced repeated replication, or jackknife—but with most survey datasets and problems, you should be fine with the default; see Kreuter and Vaillant (2007) for other methods.

Stata has a suite of commands for dealing with complex samples. These commands work in two steps. The first step is to declare the dataset a complex sample. The second step is to prefix the relevant estimation commands with **svy**.

The command to declare the dataset a complex sample is **svyset**. The command is also used to designate variables that contain information about the survey design and to specify the default method for the estimation of the standard error. A simplified version of the syntax diagram of **svyset** is

```
svyset [psu] [weight] [, strata(varname) vce(vcetype)]
```

In this syntax diagram,

- [*psu*] refers to the name of the variable that holds the identifier for the primary sampling unit,
- [*weight*] is used to specify the name of the variable that contains the reciprocal value of the selection probability,
- **strata**(*varname*) is used to specify the name of the variable holding the identifier for the strata used in the sampling design, and
- **vce**(*vcetype*) selects one of several possible methods to estimate the standard error. Taylor linearization is the default, so we will just use that.

Let us demonstrate the specification of a complex sample with our **data1.dta**. To do this, you need some background on the GSOEP sampling design. The observations used in our dataset stem from five independently drawn samples for the following populations at various time points:⁵

5. Keep in mind that **data1.dta** is a subset of the original GSOEP and has been slightly altered to meet data privacy restrictions. In the original GSOEP, there are even more samples. The estimates also do not necessarily add up to the German population.

- A** West Germans (1984)
- B** Foreigners living in West Germany who immigrated from Turkey, Italy, Spain, former Yugoslavia, and Greece (1984)
- C** East Germans (1990)
- D** Immigrants from Eastern Europe (1994–1995)
- F** People living in Germany (2000)

The sample from which each observation originated is stored in the variable `sample`, and we use this variable to define the strata in `svyset`. All but one of the samples applied a two-stage sampling design with selection proportional to size. In a first stage, geographical clusters (election districts and communities depending on the sample) were selected; those form the PSUs. In our dataset, an identifier for the PSU is stored in the variable `psu`. In a second stage, households were selected from within those PSUs.

Now we are good to go with the setting except that the GSOEP sample D, immigrants, poses a problem: unlike all other samples, sample D did not apply any kind of multistage sampling. If there is a stratum with no or only one PSU in the dataset, Stata will not report standard errors. One option then is to combine the single-state stratum with another stratum in which multistage sampling was applied. We do this by combining sample D with sample B:

```
. generate sampleR = sample
. replace sampleR = 2 if sample==4
```

We then `svyset` our dataset as follows:

```
. svyset psu, strata(sampleR)
```

Once we have declared the sampling design of our sample, we can use `mean` with the `svy` prefix. In the output, you see the estimated mean and standard error and the sample design information. We now have four strata with a total of 515 PSUs that together hold our 5,411 cases.

```
. svy: mean ybirth
(running mean on estimation sample)
Survey: Mean estimation
Number of strata =        4          Number of obs     =    5411
Number of PSUs   =      515          Population size  =    5411
                                         Design df       =      511

```

	Linearized			
	Mean	Std. Err.	[95% Conf. Interval]	
ybirth	1959.493	.3750048	1958.756	1960.23

As is often the case, stratified sampling involves unequal selection probabilities. The GSOEP is therefore delivered with the variable `design`, which holds the reciprocal value of the selection probabilities of each respondent. In `data1.dta`, that variable was renamed `dweight`. We can overwrite the old `svyset` specifications by issuing a new `svyset` command that takes those weights into account:

```
. svyset psu [pweight=dweight], strata(sample)
  pweight: dweight
  VCE: linearized
  Single unit: missing
  Strata 1: sample
  SU 1: psu
  FPC 1: <zero>
. svy: mean ybirth
(running mean on estimation sample)
Survey: Mean estimation
Number of strata =      4          Number of obs     =    5411
Number of PSUs   =    515          Population size  =  60039280
Design df        =         511

```

	Linearized		
	Mean	Std. Err.	[95% Conf. Interval]
ybirth	1958.894	.4092376	1958.09 1959.698

In the results, you see a slight change in the point estimates, an increase in the estimated standard errors, and in the upper right corner we see the estimated population size from which this sample was drawn.

To see the effect of the survey settings, compare the results above with the results that ignored the sample design (page 214). The standard error for `ybirth`, taking the design information into account, is 0.409, while ignoring the sample design led to an estimate of 0.246. Hence, the standard error of the stratified two-stage cluster sample is $0.409/0.246 = 1.663$ times larger than the standard error of ignoring the clustering design.

Generally, the result of dividing the standard error of a complex sample by the standard error of an equally sized simple random sample is called the design effect (DEFT). After having used an estimation command with the `svy` prefix, you can always issue the command

```
. estat effects
```

	Linearized			
	Mean	Std. Err.	DEFF	DEFT
ybirth	1958.894	.4092376	2.71392	1.6474

to obtain this design effect. The command also shows a second design effect, DEFF, which is the square of DEFT. The number obtained here, 2.714, means that the sample

size of the cluster sample should be almost three times larger than the sample size of a simple random sample to get the same efficiency for the sample mean of year of birth. Note that design effects of 1 would mean that the complex sample yields the same standard error as the simple random sample. Note also that these are huge design effects, meaning that all statements about the precision of the estimates, confidence intervals, or significance tests (see section 8.2.4) are pointless if they do not take into account the complex nature of the sample.

8.2.3 Standard errors with nonresponse

So far we have only dealt with situations where the sample was drawn as intended. A sample was designed, the sampling units were drawn, and observations were taken from the selected units. However, in real world situations, you cannot always take observations from sampled units, especially if you take your observations using a survey. Quite often, the target respondents decline to participate in a survey. Others are simply not available at the address you have selected. Last but not least, many selected units participate in the survey, but decline to answer specific questions.

If selected units drop out of the sample for some reason, we call that *unit nonresponse*. If we miss information on a particular variable, it is called *item nonresponse*. Let us start by discussing unit nonresponse.

Unit nonresponse and poststratification weights

Unit nonresponse decreases the number of observations that can be used to estimate a particular statistic. The denominator in the right-hand side of (8.1) on page 213 therefore becomes smaller so that the standard error increases. Even more problematic is unit nonresponse that is correlated with the statistic we want to ascertain. It is quite common, for example, that persons above a specific age participate less frequently in a survey than those below that age, for a variety of reasons. As a consequence, the proportion of people above the specific age is lower in the sample than in the population. If this is the case, the average year of birth calculated in the sample would be too low.

The most common way to deal with such problems of unit nonresponse is with poststratification weights. In section 8.2.2, we showed that the selection probabilities of two-stage samples are often not equal for all units of the population. To deal with that problem, we created a variable to hold the reciprocal value of the selection probability and we calculated our statistic using these design weights. Very much the same could be done for unit nonresponse. For example, if we knew from somewhere persons aged 70 and above participate with a probability of, say, 50% in a survey, while those below 70 participate with a probability of 75%, we could create a poststratification weight that holds the reciprocal values of these participation probabilities:

```
. generate postweights = 1/0.5 if (2009-ybirth) >= 70 & !missing(ybirth)
. replace postweights = 1/0.75 if (2009-ybirth) < 70
```

The multiplication of the variable `postweight` with the design weight (the variable holding the selection probabilities due to the sampling method) is then the reciprocal value of the probability of being observed. Using the result of this multiplication as the sampling weight corrects for the differences in the selection probability and the differences in the participation probabilities:

```
. generate bothweights = dweight * postweights
. mean ybirth [pweight = bothweights]
```

An obvious problem with poststratification weights is that the participation probability is not observable. Unlike the selection probability, which follows directly from the sampling method, the participation probability hinges in an unknown way on characteristics of the target person and of the specific situation in which the target person is contacted.

In this situation, we need to estimate the participation probability with the limited information that we have about all this. There is a large body of research on how exactly this can be done, but this is beyond the scope of this book. The good news is that many dataset providers use the methods proposed by this research field and add weighting variables into the dataset. This is also true for the GSOEP, which contains the variable `w1110109` that holds the reciprocal value of the overall observation probability for the survey year 2009. For the sake of simplicity, we have renamed that variable to `xweights`. Here is how you can use this to set the survey:

```
. svyset psu [pweight=xweights], strata(sample)
    pweight: xweights
    VCE: linearized
Single unit: missing
Strata 1: sample
    SU 1: psu
    FPC 1: <zero>
. svy: mean ybirth, noheader
(running mean on estimation sample)
```

	Linearized			
	Mean	Std. Err.	[95% Conf. Interval]	
ybirth	1959.616	.5040505	1958.626	1960.606

Item nonresponse and multiple imputation

The consequences of item nonresponse are essentially the same as for unit nonresponse: standard errors are increased because of the loss of observations and the statistics from the variables we analyze are biased if the probability of nonresponse is correlated with the variable that we want to analyze. The technique to solve the problem is different, however. This is because for those respondents who have declined to answer a survey question, we have information from their answers to other questions. This information can be used to impute the missing values.

To start with, let us create some fictitious reality. We generate a new variable that is equal to `ybirth` except that it has some missing values (that is, item nonresponse). The missing values are created such that the likelihood of item nonresponse is higher the older the respondent is. Item nonresponse is also higher for women than for men. We do this by generating a copy of `ybirth` and then replacing some values in this copy to missing. Specifically, `ybirth` becomes missing if year of birth is smaller than a right-skewed random variable between 1910 and 1991, whereby the amount of skewness is different for men and women:

```
. generate ybirthmis = ybirth
. replace ybirthmis = . if sex == 1 & (ybirth < (1910 + 81 * rbeta(1,3)))
. replace ybirthmis = . if sex == 2 & (ybirth < (1910 + 81 * rbeta(1,6)))
```

Take a look at the averages of `ybirth` for men and women, and for those missing versus not missing:

```
. generate missing = missing(ybirthmis)
. tabulate sex missing, sum(ybirth) nost
      Means and Frequencies of Year of birth
```

Gender	missing		Total
	0	1	
Male	1962.5009 2280	1940.4492 305	1959.899 2585
Female	1960.4128 2706	1930 120	1959.1214 2826
Total	1961.3676 4986	1937.4988 425	1959.4929 5411

The above output reveals that we set 425 values to missing. Those with missing are older than those with not missing, and this effect is stronger among women than among men. If we use `ybirthmis` instead of `ybirth` to estimate the average year of birth of the German population, we obtain a result that is more than two years too high:

```
. svy: mean ybirthmis, noheader
(running mean on estimation sample)
```

	Linearized			
	Mean	Std. Err.	[95% Conf. Interval]	
ybirthmis	1961.601	.4897198	1960.638	1962.563

How can we correct the result? One idea is to replace the missing values in `ybirthmis` with some plausible year of birth. The first plausible value one might think of is the average year of birth taken from those observations that we actually have observed. Because we have estimated this value in our last command, we can use its saved result to create a new version of year of birth. This new version contains the values of `ybirthmis` if this variable is not missing and is equal to the estimated year of birth otherwise. Note as an aside that the stored result of mean can be accessed with `_b[ybirthmis]`:

```
. generate imputed1 = cond(!missing(ybirthmis), ybirthmis, _b[ybirthmis])
```

Conforming to the terminology used in [MI] **intro substantive**, we will call the data in which the missing values have been replaced with a plausible value an *imputation*. If we use our imputation (the variable `imputed1`) to reestimate the population mean of year of birth for Germany in 2009, it turns out that the estimate does not change and the standard error decreases substantially:

```
. svy: mean ybirth imputed1, noheader  
(running mean on estimation sample)
```

	Linearized			
	Mean	Std. Err.	[95% Conf. Interval]	
ybirth	1959.616	.5040505	1958.626	1960.606
imputed1	1961.601	.451904	1960.713	1962.488

Therefore, while this single imputation does not help in fixing the bias of the estimation, it underestimates the standard errors and overstates the precision of our analysis. This is so because we treated the imputed values just like the observed values even though they are really just wild guesses. What happens if we choose not just one plausible value for the missings in `ybirthmis` but different ones for various groups? For example, our dataset contains the variable `heval`, the current self-rated health status:

Current Self-Rated Health Status	Freq.	Percent	Cum.
Very good	422	7.91	7.91
	2,034	38.14	46.05
	1,889	35.42	81.47
	780	14.63	96.10
	208	3.90	100.00
Total	5,333	100.00	

You might have the idea that those who rate their health as being bad tend to be older than those who consider their health to be very good. In fact, if you look at

Current Self-Rated Health Status	Summary of ybirthmis		
	Mean	Std. Dev.	Freq.
Very good	1972.7119	13.805401	413
Good	1966.6378	15.082601	1952
Satisfact	1956.8227	16.132137	1703
Poor	1951.7997	15.783975	679
Bad	1944.3313	14.617787	163
Refusal	1963.3	19.385275	10
Does not	1992	0	66
Total	1961.3676	17.266425	4986

you see that the former are born in 1944 on average, while the latter are born on average almost 30 years later. It seems plausible, therefore, to impute the value 1945 for all those who rate their health as being bad, and impute the value 1972 for all those who regard their health as being very good. This can be done by

```
. egen mean = mean(ybirthmis), by(heval)
. generate imputed2 = cond(!missing(ybirthmis),ybirthmis,mean)
```

Reestimating the average year of birth with this second imputation reveals that the estimated mean slightly moves in the right direction while the standard error remains too low:

```
. svy: mean ybirth imputed1 imputed2, noheader
(running mean on estimation sample)
```

	Linearized			
	Mean	Std. Err.	[95% Conf. Interval]	
ybirth	1959.616	.5040505	1958.626	1960.606
imputed1	1961.601	.451904	1960.713	1962.488
imputed2	1961.186	.458776	1960.284	1962.087

Generally, the bias in the estimation of the mean will disappear to the extent that the imputed plausible values are correct on average. Hence, if you impute group averages of observed variables that are highly correlated with year of birth, your plausible values get better and better. You might even use more than just one variable for the imputation step, either by calculating averages of combinations of groups or by applying a variant of the models that we describe in chapters 9 and 10.

The problem of deflated standard errors is slightly more difficult to deal with. The standard error decreases because our statistical analysis gives the imputed values the same level of confidence as the observed values. To solve the problem, we must tell our analysis that imputed values should not be taken as seriously as the observed values. This is done with *multiple imputation*.

Multiple imputation is an advanced topic that is way beyond the scope of this book. In what follows, we provide just a broad overview on the general idea of it. This is primarily to tease your interest. We recommend reading [MI] **intro substantive** and some of the literature cited therein before actually using multiple imputation for real research. The general idea of multiple imputation is, however, easy to understand.

As said above, our singular imputed values were treated with the same confidence as the observed values. A first step to decrease the level of confidence could be to add some random noise to the group average that we impute. Building on the group averages in the variable mean, we can add a normally distributed random value to each imputed value:

```
. generate imputed3 = cond(!missing(ybirthmis),ybirthmis,mean+rnormal())
```

Before we proceed, we confirm that adding randomness does not really change much the estimates of both the mean and the standard errors:

```
. svy: mean ybirth imputed1 imputed2 imputed3, noheader  
(running mean on estimation sample)
```

	Linearized			
	Mean	Std. Err.	[95% Conf. Interval]	
ybirth	1959.616	.5040505	1958.626	1960.606
imputed1	1961.601	.451904	1960.713	1962.488
imputed2	1961.186	.458776	1960.284	1962.087
imputed3	1961.185	.4591513	1960.283	1962.087

The reason for the small change is that we have on average imputed the same values as before, and our analysis still treats the imputed values like the observed ones. However, now that we have added random noise to the imputation, we can replicate the imputation step several times. In each replication, the results will differ slightly. Let us try that out with five replicates:

```
. forvalue i = 1/5 {
    2.         generate mi`i` = cond(!missing(ybirthmis),ybirthmis,mean+rnormal())
    3. }
. svy: mean mi1-mi5, noheader
(running mean on estimation sample)
```

	Linearized			
	Mean	Std. Err.	[95% Conf. Interval]	
mi1	1961.19	.4586463	1960.289	1962.091
mi2	1961.18	.4595625	1960.277	1962.082
mi3	1961.191	.4586397	1960.29	1962.092
mi4	1961.178	.4591674	1960.276	1962.08
mi5	1961.186	.4587945	1960.284	1962.087

We now discover that the estimation of average year of birth is uncertain not only because we ascertained it in a random sample—this uncertainty is expressed in the standard errors shown in each line of the output—but also because of the uncertainty in the imputed values, shown by the slight differences in the means themselves. The more the various means differ, the higher is the uncertainty due to the imputation. To estimate the entire uncertainty in the estimation of the mean year of birth, we should consider both sources of uncertainty. How this could be done is answered in the statistical literature on multiple imputation (Rubin 1987; Schafer 1997), and we will not deal with that here. However, what we will do is show a very simple example for how Stata does all the nuts and bolts for you.

Multiple imputation has two parts. First, you instruct Stata to create multiple sets of imputed variables (“imputation”). Second, you perform a statistical analysis on each of the imputations, and let Stata come up with point estimates and standard errors (“analysis”).

The imputation is further divided into three steps. First, you must instruct Stata how the multiple imputations should be organized. You have two choices: wide or long. Style long has various variants. In our case, the choice is not important, so we use `mlong`, which seems to be preferable in situations where all styles are possible (see [MI] **styles** for details).

```
. mi set mlong
```

The second step of the imputation is to instruct Stata which variables should be imputed. In our example, this is just `ybirthmis`. To register this variable as imputed, we type

```
. mi register imputed ybirthmis
```

During the registration step, it is also advisable to register “regular” variables. These are variables that are not imputed and whose values do not depend in any way on the variables to be imputed. In our example, all variables of the dataset except `ybirthmis` are regular. Hence,

```
. mi register regular persnr-xweights
```

If we had created an age variable from `ybirthmis`, the dataset would have contained a nonregular variable. Because age is a function of the imputed variable, we would have registered age as *passive* by using `mi register passive age`.

The creation of the multiple imputations is the third and final step of the imputation. For this, you have to instruct Stata how to create the plausible values: Which variables should be used, which technique should be applied, and how many imputations should be performed? The command `mi impute` lets us specify all of this in one command. Our example below uses the predicted values of a linear regression of `ybirthmis` on `sex` and dummy variables of `heval` (see chapter 9) to create the plausible values for the missings in `ybirthmis`. The option `add(20)` instructs Stata to create 20 imputations. Note also that we changed the missings in `heval` to a valid value before we actually used `mi impute`.⁶

```
. generate hevalR = heval
(78 missing values generated)
. replace hevalR = 6 if missing(heval)
(78 real changes made)
. mi impute regress ybirthmis = sex i.hevalR, add(20)
Univariate imputation                               Imputations =      20
Linear regression                                     added =          20
Imputed: m=1 through m=20                           updated =          0

```

Variable	Observations per m			
	Complete	Incomplete	Imputed	Total
ybirthmis	4986	425	425	5411

(complete + incomplete = total; imputed is the minimum across m
of the number of filled-in observations.)

Once `mi impute` is finished, the first part of multiple imputation is done. The output informs us that `mi impute` has created 20 imputations of `ybirthmis`. It also shows that in each imputation, 425 missing values were imputed. The imputed values have been written below the original data into the variable `ybirthmis` so that the dataset now contains $5411 + 20 \times 425 = 13911$ observations. `mi impute` has also created the new variables `mi_m`, `mi_id`, and `mi_miss`, which are necessary for performing an analysis with the multiply imputed dataset.

6. We could have also used the option `force` of `mi impute` to deal with the missings in `heval`. In this case, the information that these persons were not willing to answer the question about health would have been discarded from the imputation.

A multiple imputation analysis is as simple as using one of the commands mentioned in [MI] **estimation** behind the prefix [MI] **mi estimate**. Let us do that with our standard example:

<code>. mi estimate: svy: mean ybirthmis, noheader</code>	
Multiple-imputation estimates	Imputations = 20
Survey: Mean estimation	Number of obs = 5411
Number of strata = 4	Population size = 80858869
Number of PSUs = 515	
	Average RVI = 0.0303
	Largest FMI = 0.0296
	Complete DF = 511
DF adjustment: Small sample	DF: min = 483.17
	avg = 483.17
Within VCE type: Linearized	max = 483.17
<hr/>	
	Mean Std. Err. [95% Conf. Interval]
ybirthmis	1961.194 .4713432 1960.268 1962.12

Realize that the results of the point estimate (the mean of `ybirthmis`) is not any better than the results that we gained with the single imputation methods before. Note, however, that the standard error is now substantially closer to the value we gained for the original variable without missings (`ybirth`).

As with single imputation, the quality of the point estimate depends on the quality of the imputed plausible values, and this largely depends on the information used for finding these values. It is therefore quite helpful to have good knowledge about why missing values on your variables arise. In our example, the missingness was caused by an interaction term, which should be modeled here; how interaction terms are modeled will be explained in section 9.4.2. More importantly though is that the missing values are a function of the variable of interest. Thus unless the variables we use in the imputation models are perfect proxy variables for `ybirth`, it is likely that the values are not missing at random, even after conditioning on the covariates in the model.

8.2.4 Uses of standard errors

In the previous section, you learned how to estimate the standard errors for a given sampling method. In this section, we assume that you obtained the correct standard error, and so we introduce confidence intervals and significance tests as further uses of standard errors.

Confidence intervals

Like the standard error, a confidence interval is a way to express the precision or reliability of an estimated statistic. To understand confidence intervals, remember that, if we estimate the mean of some variable for an infinite number of random samples, these means have a normal distribution. You have seen an approximation of that normal distribution in section 8.2.

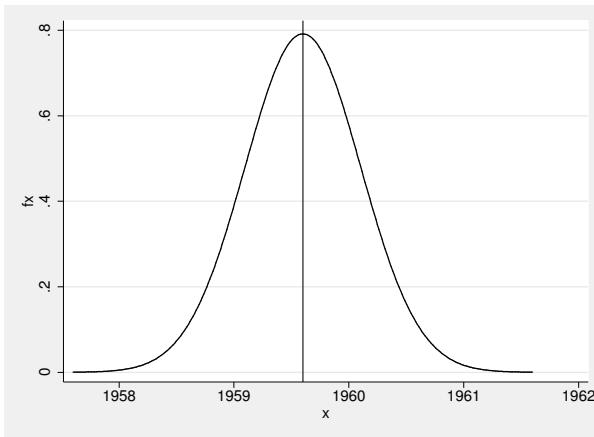
The density of values of a normal distribution is given by the *probability density function of the normal distribution*, which is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the unknown mean of all the sample means, and σ is the unknown standard deviation of all the sample means ($\pi \approx 3.142\dots$).

We discussed before that the best guess for the unknown parameters μ and σ that we have from our one-shot sample are the sample means and its standard error. Inserting these values in the formula above, we can plot the above function using the two-way plottype `function`. Let us use the values 1959.6 and 0.504 from page 223 to produce an imagination of the sampling distribution of sample means for the GSOEP.

```
. twoway function
> fx = 1/sqrt(2*_pi*.504^2) * exp((-x-1959.6)^2/(2*.504^2)),
> range(1957.6 1961.6) xline(1959.6)
```



Based on our guesses for μ and σ , the sampling distribution suggests that the average year of birth in the population is somewhere between 1958 and 1961. It must be clear, however, that this statement is only correct to the extent that our guess for μ is correct in the sense that the mean and standard deviation of guesses for an infinite number of samples is equal to μ and σ , respectively. In this case, it is also true that 95% of the intervals from $\hat{\mu} - 1.96 \times \hat{\sigma}$ to $\hat{\mu} + 1.96 \times \hat{\sigma}$, with $\hat{\mu}$ and $\hat{\sigma}$ being the guesses for

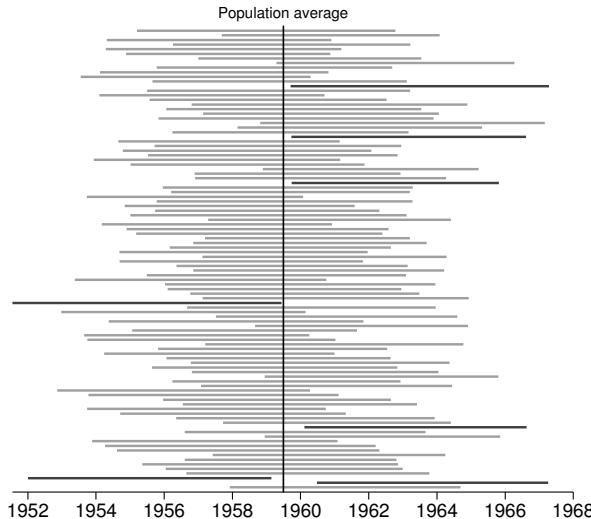
the respective population parameters in each replication, include the true population parameter. In our example, we can be therefore 95% confident that Germany's average year of birth is between

```
. display 1959.6 - 1.96*0.504
1958.6122
```

and

```
. display 1959.6 + 1.96*0.504
1960.5878
```

The interval between these two numbers is called the 95% confidence interval. Note as an aside that the statement that 95% of the intervals formed this way include the population mean is not the same as saying that there is a 95% probability of the population mean falling inside this interval. Only the former statement is correct. Figure 8.3 demonstrates the difference between these two statements. For the figure, we have drawn 100 simple random samples with 100 observations from the population of `data1.dta`. In each of the samples, we have estimated the 95% confidence interval. The figure shows each confidence interval as a horizontal bar. The seven highlighted confidence intervals did not include the true value. Also these are 7% of the confidence intervals; in the long run only 95% of the intervals will include the true value, while 5% will not.



Source: `grci.do`

Figure 8.3. One hundred 95% confidence intervals

Significance tests

When doing data analysis with sample data, you are often faced with the question of whether an observed difference between two groups happened by chance due to sampling variability or if you can be pretty sure there is a difference in the population as well. A similar question might arise when you estimate a regression coefficient, and you wonder if the effect you observe for a covariate is different from 0 in the population. Or very simply, you estimate a mean for a variable in your sample and you wonder what inference to make out of that mean. In each of these settings and many more, you are applying a significance test. In the context of descriptive inference, a significance test proposes that some parameter is 0 in the population, and calculates how probable it is that we get a value as large or larger in our sample if that were indeed the case. If this probability is below some threshold (we will come to that), it is claimed that the parameter is significantly different from 0, or just *significant*.

Before we introduce significance tests, we must wave a warning flag: In applied research, the results of significance tests are often interpreted with statements claiming that this or that is significant, which sounds as if we found something important. There is a sizable body of literature that criticizes the misuse of significance tests among scientists from various disciplines (among others Selvin [1970]; Läärä [2009]). However, not all significant results are important and vice versa. We therefore urge you not to base any substantive statement on significance tests alone. Some of our examples below will show this very clearly.

To illustrate a common significance test, we will compute a value that represents the difference in the age of the head of the household and the age of the partner of the head of the household (where such a partner exists). The partner of the head of the household is identified by the variable `rel2head` and coded with a value of 2 to specify that the person is the partner of the head of the household. We can create a variable that is the difference in the age of the head of the household and the age of his or her partner by using the `generate` command below.⁷

```
. use data1, clear
(SOEP 2009 (Kohler/Kreuter))
. by hhn2009 (rel2head), sort: generate agediff=ybirth-ybirth[_n-1] if
> rel2head==2
(3739 missing values generated)
. summarize agediff
Variable |   Obs      Mean    Std. Dev.     Min      Max
agediff | 1672  1.358852  5.168304    -27      33
```

According to our data, there is on average more than one year difference between the heads of households and their partners. But we see quite a bit of variation around that average and a range of partners from 27 years younger to 33 years older.

7. For the use of subscripts, see section 5.1.5.

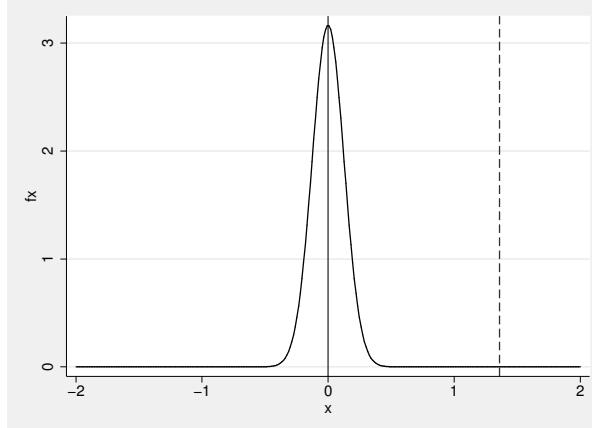
Now we formulate the null hypothesis: “The average age difference between heads of households and their partners in 2009 is 0.” We assume that this hypothesis is true! Which values for the average age difference would then be observed in our GSOEP data?

We have learned that the sample means of an infinitive number of samples have a normal distribution around the true mean. Hence, if our null hypothesis were true, the sample means should vary normally around 0. How much do they vary? The best answer to this question is the standard error for the sample that we have. For the moment, we (incorrectly) assume our sample was drawn as a simple random sample and estimate the standard errors with

Mean estimation		Number of obs = 1672		
		Mean	Std. Err.	[95% Conf. Interval]
agediff		1.358852	.126395	1.110942 1.606761

Using this answer, we can draw the expected distribution of sample means for an infinite number of samples in direct analogy to the last section. The hypothesized population value is again indicated with a solid line, and our observed sample value is indicated with a dashed line.

```
. twoway function
> fx = 1/sqrt(2*_pi*.126^2) * exp((-x-0)^2/(2*.126^2)),
> range(-2 2) xline(0) xline(1.359, lpattern(dash))
```



You can see from this graph that if the average age difference of heads of households and their partners in Germany in 2009 were 0, most of the samples (of the same size we have here) would have a mean difference very close to 0, almost all of them between -1 and 1 . The average we observed here—indicated by the dashed line—is already very unlikely. And we can be pretty sure that we should have observed a much lower mean if

the true mean were indeed 0. How sure is pretty sure? The fact that this is a probability density function allows us to give a numerical answer to this question. The probability to observe (as we do) a value of `agediff` of 1.359 or higher is equal to the area to the right of the dashed line underneath the probability density function of the observed value.

Stata has a built-in function that does such calculations for you: `normal(z)`. The function displays the probability to observe a value below z for the case of a *standard normal distribution* (that is, a normal distribution with a standard deviation of 1). It is helpful to try out the function for various values of z . Starting with $z = 0$,

```
. display normal(0)
.5
```

we see that 50% of the values of a standard normal distribution are lower than 0, or stated differently, the probability of getting values below 0 is 50%. Likewise, the probabilities of values below -1 and -2 are 15.9% and 2.3%, respectively:

```
. display normal(-1)
.15865525
. display normal(-2)
.02275013
```

We can also use the function to find out the probability of values above the chosen value. Because all values that are not below the chosen value must be greater than it, we get this probability by subtracting the result of `normal()` from 1:

```
. display 1 - normal(-2)
.97724987
. display 1 - normal(2)
.02275013
```

In a standard normal distribution, the probability of values above -2 is approximately 97.7%, and the probability of values above 2 is only 2.3%.

How can we use the function `normal()` to get the probability of a value below z in a sampling distribution that does not have a standard deviation of 1? For this, you must know that it is possible to rescale a distribution to a distribution with a standard deviation of 1 by dividing each value of the distribution by its standard deviation. In our case, we have observed only one value of our sampling distribution, 1.359, and we have estimated the standard deviation of the sampling distribution (that is, the standard error) to be 0.126. If we divide 1.359 by 0.126, we obtain approximately 10.786:

```
. display 1.359/0.126
10.785714
```

This is the value of the mean transformed to a distribution with a standard deviation of 1; note that the result of the division of a point estimate by its standard error is commonly termed the *test statistic*. The test statistic can be used instead of the observed statistic. In our example, this means that instead of checking the probability

of values below 1.359 in a normal distribution with a standard deviation of 0.126, we can now obtain the probability of values below 10.786 in a standard normal distribution by using `normal()`:

```
. display normal(1.359/0.126)
1
```

This means that the probability of observing a value below 10.786 in a standard normal distribution is very close to 1; and therefore, the probability of observing a value below 1.359 in a normal distribution with a standard deviation of 0.126 is also very close to 1. Likewise, the probability of observing a value above 1.359 is

```
. display 1 - normal(1.359/0.126)
0
```

meaning that it is extremely unlikely to observe a value as big as the one we have observed when the true value is 0. Hence, we ought to reject the null hypothesis that heads of households and their partners in the German population of 2009 have on average the same age.

By the way, it is common practice to reject the null hypothesis if the probability of observing a test statistic as high as the one observed is below 5%. In this case, authors often write that the observed value is “significant at the 5% level”.

Notice though the role of the sample size for that conclusion. If you had observed the same mean and standard deviation as you did here for a simple random sample of size 30, your estimated standard error would be $5.168/\sqrt{30} = 0.944$. Hence, the test statistic would become $1.359/0.944 = 1.44$, and the probability to observe such a value given the null hypothesis would be 0.075:

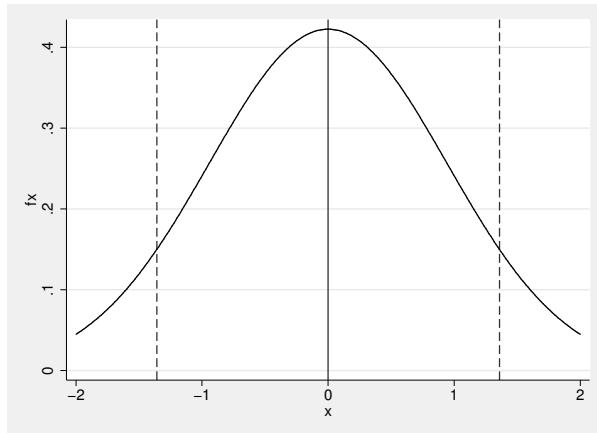
```
. display 1 - normal(1.359/.944)
.07498766
```

Although this probability is still pretty low, it is much higher than for a larger sample and no longer significant at the 5% level. You learn from this that the same values for the mean and the standard deviation can be significant in one sample and not significant in another.

What we have just performed was a one-sided test that calculates the probability of a value above z . More common, however, is a two-sided test that calculates the probability of a value below $-z$ or above $+z$. You would do a two-sided test if you had no prior hypothesis about the direction this age difference would go.

Using the example above with a sample size of 30, the probability of observing a value between -1.359 and 1.359 is equal to the area underneath the probability density function in the range between -1.359 and 1.359 .

```
. twoway function
> fx = 1/sqrt(2*_pi*.944^2) * exp((-x)^2/(2*.944^2)),
> range(-2 2) xline(0) xline(-1.359 +1.359, lpattern(dash))
```



Notice how much wider the function is now that we have a sample size of 30. To get a numerical estimate for the area in between the two dashed lines, we need to subtract the area below -1.359 from the area below 1.359 . In Stata terms:

```
. display normal(1.359/0.944) - normal(-1.359/0.944)
.85002468
```

If the average age difference between the heads of household and their partners were 0, the probability of observing a value between -1.359 and 1.359 in a sample of size 30 would be 0.85. To get the probability of observing values outside the range of $[-1.359, 1.359]$, you would take 1 minus the probability of values inside $[-1.359, 1.359]$,

```
. display 1 - (normal(1.359/0.944) - normal(-1.359/0.944))
.14997532
```

which is the same as two times the probability of values below -1.359 :

```
. display 2 * normal(-1.359/0.944)
.14997532
```

For those smaller samples, the probability would be about 0.15; to say it differently, in 15% of the sample means, the absolute age difference would be as large or larger than what we observed here, even if the true age difference is 0. In this situation, we would fail to reject the null hypothesis at the conventional 5% level. Notice as an aside that if you follow the conventional 5% level, significance of a two-sided test is reached

whenever the absolute value of the test statistic is above 1.96 or when the observed point estimate is near doubly as high as its standard error.

Before turning to the more frequently used two-group mean comparison test, we want to show how to perform the test for complex samples. For this, we estimate the standard error as shown in section 8.2.2

```
. generate sampleR = sample
. replace sampleR = 2 if sample==4
(150 real changes made)
```

We then go on as shown:

```
. svyset psu [pweight = xweight], strata(sampleR)
(output omitted)
. svy: mean agediff
(output omitted)
```

To avoid rounding errors, we use saved results for calculating the test statistic (see chapter 4):

```
. display 2*normal(-_b[agediff]/_se[agediff])
5.865e-13
```

Two-group mean comparison test

Having been through the steps of testing the sample mean against the null hypothesis, testing the difference between two sample means is straightforward. The key difference between the two settings is that now both sample means have a distribution that needs to be taken into account when testing. In a two-group mean comparison test, the null hypothesis is that “the difference of the means between group A and group B is 0.” We could, for example, be interested in any age difference in the population of East and West Germans. After all, there were many years when these two parts of Germany were separate countries with different health care systems, different family policies, and other factors related to the demographic development of a country. On the other hand, there has been some geographic mobility since the reunification, and the countries have been subject to the same political system for more than 20 years.

So let us first see if there is an age difference between East and West Germans in our sample:

```
. generate east = state >= 11 if !missing(state)
. tabulate east, sum(ybirth)
      Summary of Year of birth
      Mean   Std. Dev.    Freq.
east
  0 | 1959.8282  18.140227    4068
  1 | 1958.4773  18.053304    1343
  Total | 1959.4929  18.126422    5411
```

Here we see that people living in the former Western part of Germany are on average 1.351 years older than those living in the East. How likely is it that we observe such a difference if East and West Germans had the same age in the population? To answer this question, we have to estimate the standard error for both the East Germans' and the West Germans' average year of birth. For simplicity, let us ignore the complex nature of our sample for a moment and estimate the standard error for both groups with the **mean** command:

```
. mean ybirth, over(east)
Mean estimation                               Number of obs      =      5411
                                                0: east = 0
                                                1: east = 1


| Over   | Mean     | Std. Err. | [95% Conf. Interval] |          |
|--------|----------|-----------|----------------------|----------|
| ybirth | 1959.828 | .2844148  | 1959.271             | 1960.386 |
|        | 1958.477 | .4926276  | 1957.512             | 1959.443 |


```

The mean comparison test starts by stating the null hypothesis: “The difference between East and West Germans’ years of birth was 0 in Germany in 2009.” We then ask what values we would observe in a sample such as ours if this were indeed the case. To answer this, we need the standard error for the difference between the two means. If we assume that both groups, East and West Germans, are independently sampled, the standard error of the difference between the two groups can be estimated with

$$\widehat{SE}(\bar{x}_A - \bar{x}_B) = \sqrt{SE(\bar{x}_A)^2 + SE(\bar{x}_B)^2} \quad (8.2)$$

Using the numbers above, we can apply this formula in Stata:

```
. display sqrt(.284^2 + .493^2)
.56895079
```

After having obtained the standard error we can calculate how probable it is to observe a value as big as our observed difference of 1.351 if the true difference in the population were 0. With a small risk of rejecting the null hypothesis too early, we can apply a simplified command for a two-sided significance:

```
. display 2 * normal(-1.351/.493)
.0061371
```

If East and West Germans are of equal age in the population, the probability of observing an absolute difference in year of birth of 1.351 or higher would be 0.6%. This probability is very low, and it would be common practice to reject the null hypothesis.

We said above that we are risking rejecting the null hypothesis too early. The reason for this is that sampling distributions of differences between sample means are not normally distributed but are Student’s *t* distributed. The Student’s *t* distribution

is very similar to the normal distribution, but it is a little wider when the sample size is below 30 observations. In our case, this does not make a difference; in the more general case, you might want to test the difference of sample means using a Student's *t* distribution. The easiest way to do this is with the command `ttest`. Above we did not assume that the sampling distributions were the same in East and West Germany. Therefore, we perform `ttest` with the option `unequal`.

```
. ttest ybirth, by(east) unequal
Two-sample t test with unequal variances

```

Group	Obs	Mean	Std. Err.	Std. Dev.	[95% Conf. Interval]
0	4068	1959.828	.2844148	18.14023	1959.271 1960.386
1	1343	1958.477	.4926276	18.0533	1957.511 1959.444
combined	5411	1959.493	.2464185	18.12642	1959.01 1959.976
diff		1.350881	.5688354		.2353979 2.466365

```

diff = mean(0) - mean(1)                                t =    2.3748
Ho: diff = 0                                         Satterthwaite's degrees of freedom = 2301.37
Ha: diff < 0                                         Ha: diff != 0                               Ha: diff > 0
Pr(T < t) = 0.9912          Pr(|T| > |t|) = 0.0176          Pr(T > t) = 0.0088

```

The upper part of the output displays means and standard errors for both subgroups, here East and West Germany, as well as the "combined" mean and standard error. The row starting with `diff` shows the difference between the East and West German averages and the standard error of that difference. Underneath the table, results from three significance tests can be found. The two-sided test that we performed earlier is the one in the middle. As you can see, the result of `ttest` is identical to our result above—which is always the case if the sample sizes of both groups are much higher than 30.

So far, our example has assumed that the sample for both East and West Germany was a simple random sample. To account for the complex sample structure, we can use the command `test` after applying `mean` with the `svy` prefix. Here is an example:

```
. svy: mean ybirth, over(east)
(running mean on estimation sample)

Survey: Mean estimation

Number of strata =        4      Number of obs     =    5411
Number of PSUs   =      515      Population size  = 80858869
Design df       =      511

0: east = 0
1: east = 1



| Over   | Linearized |           |                      |                   |
|--------|------------|-----------|----------------------|-------------------|
|        | Mean       | Std. Err. | [95% Conf. Interval] |                   |
| ybirth | 0          | 1959.832  | .5656418             | 1958.721 1960.943 |
|        | 1          | 1958.493  | 1.085174             | 1956.361 1960.625 |


.  

.test [ybirth]0 - [ybirth]1=0  

Adjusted Wald test  

( 1) [ybirth]0 - [ybirth]1 = 0  

F( 1, 511) = 1.18  

Prob > F = 0.2771
```

The command `test` performs a adjusted Wald test of a specified expression. In the command above, our expression used the saved results of `mean`, where `[ybirth]0` refers to the mean of `ybirth` for West Germany, and `[ybirth]1` refers to the mean of `ybirth` for East Germany. Hence, we ask the command to test whether the difference between those two saved results could be 0 in the population. The results show that now we can no longer reject the null hypothesis. The correct standard errors are much larger than the naïve estimates, and consequently, the difference is no longer significantly different from 0.

The `test` command is not restricted to just testing the null hypothesis of the population difference being 0. We can insert other values as well. Were your null hypothesis, for example, that West Germans are on average two years younger than East Germans, you would specify the test as follows:

```
. test [ybirth]0 - [ybirth]1=-2
Adjusted Wald test
( 1) [ybirth]0 - [ybirth]1 = -2
F( 1, 511) = 7.36
Prob > F = 0.0069
```

8.3 Causal inference

When we, the authors, designed the contents of this chapter, we had a hard time deciding whether it should include a section on causal inference. The reason is that causal inference, really, is beyond the scope of this book. There is so much to be said on causality—what it is and how it can be observed—that a short introduction might perhaps do more harm than good.

On the other hand, some highly practical consequences hinge on the distinction between descriptive and causal inference, even for a person new to data analysis. For example, do we need significance tests when we observe an entire population? Do we need weights when we estimate statistical models? These questions cannot be answered without reference to the difference between descriptive and causal inference. Without making clear what causal inference really means, we risk that readers think “causality” when they really have just description. So in the end, we decided to go with this section. Before we start, we wish to recommend some helpful literature on causal inference: King, Keohane, and Verba (1994); Winship and Morgan (1999); Imai, King, and Stuart (2008); Stuart (2010); and last but not least, Berk (2004).

8.3.1 Basic concepts

Could this have happened by chance? This is a key question in inference problems. In descriptive inference problems we often ask whether “chance” could have produced a sample that shows this value even if the value is 0 in the population. In causal inference problems, we ask an even more fundamental question: Could chance have produced this relationship between what I think is the cause and what I see as the outcome? Or to phrase it somewhat differently: Is there a mechanism that created the data I observed that is not random? Thus to understand causal inference, you need a basic understanding of the data-generating process. You also need to know about *counterfactuals*, and what it means when we colloquially say “everything else being equal”.

Data-generating processes

Please load `titanic.dta` from our data package:⁸

```
. use titanic, clear
(Death Rates for an Unusual Episode (Dawson 1995))
```

The dataset contains information on age, gender, class, and survival of all 2,001 persons on board the *Titanic*.⁹ Note that the observations in this dataset are not a random sample of a larger population. Neither the crew nor the passengers are selected

8. Make sure that your working directory is `c:\data\kk3`; see page 3.

9. The dataset was collected by the British Board of Trade in their investigation of the sinking and made publicly available by Dawson (1995). The creation of our Stata dataset is documented in `crtitanic.do` in our data package.

by any known sampling method. Also there is no larger population of *Titanic* crews and *Titanic* passengers, thus all we observe are the persons on board. Hence, there is no need for descriptive inference here. Descriptive statements about the dataset are just statements about the people on the *Titanic*.

Nevertheless, if you are interested in causes or significant correlates of survival in the sinking of the *Titanic*, there is still a need for causal inference. To see why, let us create some fictitious reality using Stata's random-number generator. The following commands create the variable `fsurvived` by randomly assigning the values 0 (died) and 1 (survived) to observations. Because we want to end up with approximately 30% of observations survived, we draw the random numbers from a Beta distribution with `rbeta(2,3)` (see section 8.1.2):

```
. set seed 731
. generate fsurvived = int(rbeta(2,3)*2)
```

Suppose that the variable `fsurvived` comprised real data about whether persons survived the sinking of the *Titanic*. It would then be interesting to see whether there are differences in the survival rates between the various passenger classes. In fact, from the output of

Class of passenger	fsurvived		Total
	0	1	
Crew	69.38	30.62	100.00
First	66.77	33.23	100.00
Second	70.88	29.12	100.00
Third	70.40	29.60	100.00
Total	69.51	30.49	100.00

we see that first-class passengers survived more frequently than the third-class passengers.

Does this mean that the ticket a passenger purchased influenced his or her survival? Well, not in the data we just created. Our survival indicator was created through a random process. There is no link at all between passenger class and the process that generated the data in the variable `fsurvived`. The random process that generated the data happened to create an association between `fsurvived` and `class`.

Now in the real world, we do not know what generated the data. Thus the question is whether an association we see could have happened (with a decent likelihood) because of a random data-generating process or whether it is systematically linked to some key variable of interest (`class`, in our case). Distinguishing these two possible elements of the data-generating process is the task of causal inference.

Now let us define a bit more systematically what we mean by *causality*.

Counterfactual concept of causality

In statistics, causality is commonly defined using the counterfactual concept of causality. The initial development of this framework is usually credited to Neyman, Iwaszkiewicz, and Kolodziejczyk (1935) and Rubin (1974, 1979, 1980). Sometimes the framework is referred to as the Rubin causal model (Holland 1986). According to this concept, a causal effect of some treatment T is the difference between an outcome Y^T of a specific research unit i if that unit experiences the treatment and the outcome Y^C of the same unit if that unit experiences the control condition. Formally,

$$\delta_i = Y_i^T - Y_i^C$$

where δ_i is the causal effect. Both of these outcomes are *potential* outcomes, of which only one can be observed.

Consider Jack Dawson, a fictional third-class passenger of the *Titanic* who died during the sinking in the Hollywood movie “*Titanic*”. The movie made us believe that third-class passengers were kept from entering the lifeboats in favor of first- and second-class passengers. Therefore, the question arises whether Jack died because he had a third-class ticket. From the standpoint of Holland’s (1986) counterfactual conception of causality, this would be the case if there were a difference between Jack as a third-class passenger and the same Jack as a first- or second-class passenger. Because Jack cannot be at the same time both a third-class and a first- or second-class passenger, we can only observe one of the potential outcomes.

However, even though we cannot observe what would have happened to Jack had he purchased a different ticket, we can observe what did happen to others who did have a first- or second-class ticket. We can observe, for example, that the first-class passenger Rose DeWitt Bukater survived the sinking of the *Titanic*. But Rose is not just a first-class passenger, she is also a beautiful young lady. According to the rule “women and children first”, she should have survived even if she were a third-class passenger.

So what we really need is to find some first-class passenger who is more similar to Jack, ideally identical in anything that could affect the chance of survival except for having a different ticket class. At the very least, the person should be a man. The person should also be of similar age because children take precedence on the lifeboats and very old persons might have died during the travel anyway. Assume that we found Cal Hockley, a male first-class passenger of similar age to Jack. We can then compare Jack’s outcome—whether he died during the sinking or not—with Cal’s outcome by using a little variation of the formula above:

$$\delta_{\text{Cal}, \text{Jack}} = Y_{\text{Jack}} - Y_{\text{Cal}} \tag{8.3}$$

Using the values 0 and 1 for the outcomes of survived and died, respectively, we can get three different values for $\delta_{\text{Cal},\text{Jack}}$:

- 0 if Cal and Jack either both died or both survived.
- 1 if Jack died and Cal survived.
- -1 if Jack survived and Cal died.

We might want to use the value of $\delta_{\text{Cal},\text{Jack}}$ as an estimate for the causal effect δ_i . In this case, we would interpret the value of 1 as evidence that traveling on the *Titanic* as third-class passenger causally decreased the odds of surviving; -1 would mean the opposite; and 0 would mean that classes do not matter.

Quite obviously, however, the value of $\delta_{\text{Cal},\text{Jack}}$ is not sufficient as an estimate for δ_i . There are various reasons for this, but at this stage we consider only one of them: $\delta_{\text{Cal},\text{Jack}}$ might have occurred more or less by chance. That is to say, if we had chosen two other persons with the same characteristics to observe an equivalent of $\delta_{\text{Cal},\text{Jack}}$, we might have obtained a different value, leading us to different conclusions.

One way to protect ourselves against making too much out of accidental findings is to find many observations of both types, the Jack type and the Cal type. Let us call the observations of these types the treatment group and the control group. If we have such data, we can obtain the average outcome for both groups and compare the two results. This would lead us to the following estimate for the causal effect:

$$\hat{\delta} = \bar{Y}^T - \bar{Y}^C$$

With this, we would estimate what has been called the *average treatment effect* (ATE). However, the validity of the estimated ATE crucially hinges on the comparability of the treatment and control groups. Having comparable groups is often a serious challenge for social science researchers. Generally, the treatment and control groups are comparable only if they differ in whether they got the treatment and in factors that are, taken together, unrelated to the outcome. Such comparability can be achieved quite well in randomized trials, where cases are randomly assigned to the treatment and control groups.¹⁰

In the social sciences, however, it is often not possible for researchers to assign cases to the treatment of interest. Instead, there is often no other possibility than to observe cases that, for some reason, happen to belong to a treatment group and to compare them with cases that do not belong to a treatment group. In such observational studies, the comparability of the treatment and control groups must be assured by statistically controlling for the reasons responsible for belonging to the treatment group or the control group. This often requires the use of advanced statistical techniques, such as fixed-effects models for panel data (Halaby 2004; Allison 2009), matching methods

10. See Stuart et al. (2011) for ways to assess the generalizability of the results of randomized trials.

(Stuart 2010), instrumental-variables regression (Baum, Schaffer, and Stillman 2007), or regression discontinuity models (Nichols 2007).

In the next subsection, we provide an empirical example that shows the logic of causal inference using the *Titanic* dataset. The small number of variables in it, and their categorical nature, allow us to demonstrate the overall reasoning for a subgroup of interest without estimating the ATE.

8.3.2 The effect of third-class tickets

From

Summary statistics: mean by categories of: class (Class of passenger)			
class	survived	men	adult
Crew	.239548	.9740113	1
First	.6246154	.5538462	.9815385
Second	.4140351	.6280702	.9157895
Third	.2521246	.7223796	.888102
Total	.323035	.7864607	.9504771

we learn that 62% of the first-class passengers survived, but only 25% of the third-class passengers did. The difference between both groups is 37%. However, we also see that the treatment group (third-class passengers) and the control group (first-class passengers) differ in other characteristics related to survival: Third-class passenger are more frequently men and less frequently adults. Before performing causal inferences, you should make sure that the treatment and control groups are equivalent in all characteristics that are not an effect of the treatment itself. For our example, one way to proceed would be to compare only male adults from both groups:

Summary for variables: survived by categories of: class (Class of passenger)	
class	mean
Crew	.2227378
First	.3257143
Second	.0833333
Third	.1623377
Total	.2027594

The survival rate for adult men is smaller than the total average, and more importantly, the difference between first-class and third-class passengers reduced to 16%. If we are willing to accept that there are no other differences between the treatment and control groups, we might consider this value to be an acceptable approximation of the ATE. Now consider the inferential research question from the introduction of this

chapter: How likely is it that the difference of 16% in survival rates between first-class and third-class passengers could arise if there were no systematic process at stake?

Well, if the data were really generated by a completely nonsystematic process, the difference between arbitrary groupings should be inside the borders that pure random fluctuation allows. This brings us back to the techniques and concepts discussed in section 8.2. The only difference here is that we are dealing with a proportion instead of an arithmetic mean. To estimate the standard error of a proportion, we use the e-class command **proportion**. From

```
. proportion survived if men & adult, over(class)
Proportion estimation          Number of obs      =     1667
No: survived = No
Yes: survived = Yes
Crew: class = Crew
First: class = First
Second: class = Second
Third: class = Third
```

Over	Proportion	Std. Err.	[95% Conf. Interval]
No			
Crew	.7772622	.0141801	.7494495 .8050749
First	.6742857	.0355276	.6046023 .7439691
Second	.9166667	.0213873	.8747178 .9586156
Third	.8376623	.0171749	.8039757 .871349
Yes			
Crew	.2227378	.0141801	.1949251 .2505505
First	.3257143	.0355276	.2560309 .3953977
Second	.0833333	.0213873	.0413844 .1252822
Third	.1623377	.0171749	.128651 .1960243

we see that random processes would let average survival rates of adult male first-class passengers vary between 25.6% and 39.5%. Correspondingly, for adult male third-class passengers, the survival rates vary between 12.9% and 19.6%:

Because the two intervals do not overlap, we can be pretty sure that the observed difference cannot just be due to random fluctuations. To estimate how sure we can be, we can feed the values from the last output into the **test** command:

```
. test [Yes]First==[Yes]Third
( 1) [Yes]First - [Yes]Third = 0
      F( 1, 1666) =    17.14
      Prob > F =    0.0000
```

The probability of observing a difference of 16% in the survival rates of adult male first- and third-class passengers when there are only random processes going on is 0. Hence, we can be pretty sure that something “nonrandom” is going on.

So far, we have only tested the difference of the survival rates between first- and third-class passengers for adult males. We might want to continue testing the same difference for the other groups—female adults, male children, and female children—as well. If each of these groups shows similar results, we would be even more convinced that the difference of the survival rates of first- and third-class passengers is not due to chance.

With just a couple of (categorical) covariates, it is easy to fully adjust for differences in their distributions across treatment and control groups, as we did here for gender and adult. When there are more covariates, or some that are continuous, this sort of direct adjustment becomes more difficult. In this case, regression models or propensity score matching methods can be useful.¹¹ We recommend reading Stuart (2010), Nichols (2007), and Abadie et al. (2004) before applying these methods.

8.3.3 Some problems of causal inference

The example of the last section was performed on a dataset of the entire population about which we made our inferential statement. Hence, there is no inference involved from a sample to a population. The inference we used is instead about a specific data-generating process. Specifically, we tested whether the data-generating process was a random process. The results showed that something else must have generated the data.

From the results of our example, we are inclined to say that third-class passengers were hindered somehow from getting a place in a lifeboat. However, before you make such a strong causal statement, you should remember that causal inference rests on the crucial assumption that the treatment and the control groups are equivalent. This requirement can be well secured in experiments where the researcher decides who is getting the treatment and who is not. In observational data, however, this is much more difficult to accomplish.

Take the *Titanic*'s first- and third-class passengers as an example. It was clearly not the principal investigator of the dataset (the British Board of Trade) that decided who got a first- and a third-class ticket. Contrarily, it was a complicated social process. First-class tickets were predominantly bought by rich persons for obvious reasons. It is known that a much higher proportion of third-class passengers were foreigners without mastery of English; therefore, it is also plausible that many third-class passengers wanted to emigrate from their home countries to overcome serious economic hardship.

Quite obviously, first- and third-class passengers not only differed in their gender and age proportions but also on various other dimensions. And at least some of these other dimensions are also related to survival: poor people tend to be less healthy, which might have decreased the odds of surviving in the cold water in comparison with rich people; the foreigners in the third-class might have not understood the announcements

11. Currently, there is no built-in Stata routine for matching; however, several user-written commands do exist. See chapter 13 for details about how to find and install user-written commands. For a general collection of matching software, see
<http://www.biostat.jhsph.edu/~estuart/propensitiescoresoftware.html>.

made by crew; and what about the ability to swim? Do we know something about how the ability to swim was distributed among social strata in the early twentieth century?

All of these mechanisms for surviving are related to the causal process that forced passengers to buy first- or third-class tickets but are not related to the class of the ticket itself. Restricting the analysis to adult males would be therefore not enough, and we must make the treatment and control groups much more equivalent before we are allowed to talk about causality.

The statistical models discussed in the next chapters are one way to match treatment and control groups. When it comes to causal inference, the general idea of these models is to—theoretically—assume a model for the data-generating process and to estimate parameters of this model under the assumption that the theoretical model is correct. The advantage of this modeling approach is that if the theoretical model really is correct, you do not have to care about sampling and all the materials discussed in section 8.2.

A frequent critique to this approach is, however, that the assumed models are overly simplistic. The modern reaction to this critique is to assume more-complicated models for the data-generating process, although Berk (2004) showed that many of these models also come with assumptions that cannot be easily justified. We cannot decide the issue here, but we strongly recommend that you read the literature cited at the beginning of this section before dealing with the concepts of causality.

Before moving on to statistical models themselves, we would like to stress that the statistical models discussed in the next chapters are not only used for causal analysis but also can be used as devices for parsimonious descriptions of a complicated empirical reality. If used that way, there are far fewer concerns about correctness of the models. It then becomes more of a question about whether the description is interesting or useful. However, if you want to describe an empirical reality using empirical data, there must be a known relationship between the observed data and the empirical reality. If the data are from a sample, we must take into account how the sample was drawn. In this case, we are back at the methods shown in section 8.2.

In practice, researchers are often faced with a situation that comprises both types of statistical inference. This happens when researchers try to make causal inferences from samples. There is an ongoing debate on how many techniques used for descriptive inference are also necessary for causal inference. The general background of this debate is that valid causal inference always hinges on a correctly specified model of the data-generating process. If that model really is correct, the techniques used for descriptive inference are superfluous. Others argue that there is never just one data-generating process for all members of a population. They argue that one can at best describe a population average of many different data-generating processes; hence they need the tools for statistical inference. Obviously, this is not the place to continue the debate among *describers* and *modelers* (Groves 1989).

8.4 Exercises

1. Create a dataset with 1,000 observations and the following variables:
 - `x1`, a variable that holds uniformly distributed random integer numbers between 1 and 10.
 - `x2`, a dichotomous variable with values 0 and 1, where the frequencies of both values are approximately equal.
 - `x3`, a dichotomous variable with values 0 and 1, where the proportion of values of 1 are approximately 70%.
 - `y`, a continuous variable that is the sum of `x1`, `x2`, `x3`, and a value drawn randomly from a standard normal distribution.
2. Compare the distribution of `y` between values of `x1`, `x2`, and `x3`, respectively.
3. Load `data1.dta` into memory. Create a simple random sample of 1,000 observations from the dataset without destroying the dataset. Calculate the mean of the `income` variable in your sample.
4. Repeat the last task 100 times in a row. Display the distribution of the means of `income` in the 100 samples by using `kdensity`.
5. Estimate the standard error of the mean of `income` with the 100 sample means of `income`.
6. Estimate the standard error of the mean of `income` with only one of the 100 random samples.
7. Estimate the standard error of `income` with the entire dataset. Thereby assume that the observations of `data1.dta` were selected by simple random sampling.
8. Reestimate the standard error of the mean of `income` assuming the following sampling designs:
 - Stratified sampling with strata being the states.
 - Two-stage stratified sampling, with strata being the states and PSU being the households.
 - Two-stage stratified sampling, with strata being the states and PSU as identified by variable `psu`.
 - Two-stage stratified sampling that has unequal sampling probabilities, with strata being the states, PSU as identified by variable `psu`, and the reciprocal value of the sampling probability stored on `xweights`.
 - Sampling of the GSOEP as described on page 220.
9. Reestimate the standard error of the mean of `income` using 30 multiple imputations for the missing values of `income`. Thereby use the variables `sex`, `ybirth`, `yedu`, `egp`, and `rooms` to find plausible values of `income`.

10. Reload the original `data1.dta`. Then analyze how far the difference of income between men and women might be due to sampling fluctuation. Thereby apply each of the following settings:
 - Simple random sampling
 - Sampling of the GSOEP as described on page 220.
11. Perform the tests of the previous exercise by restricting the analysis on the single (variable `mar`) full-time employed respondents (variable `emp`) with maturity qualification (variable `edu`).

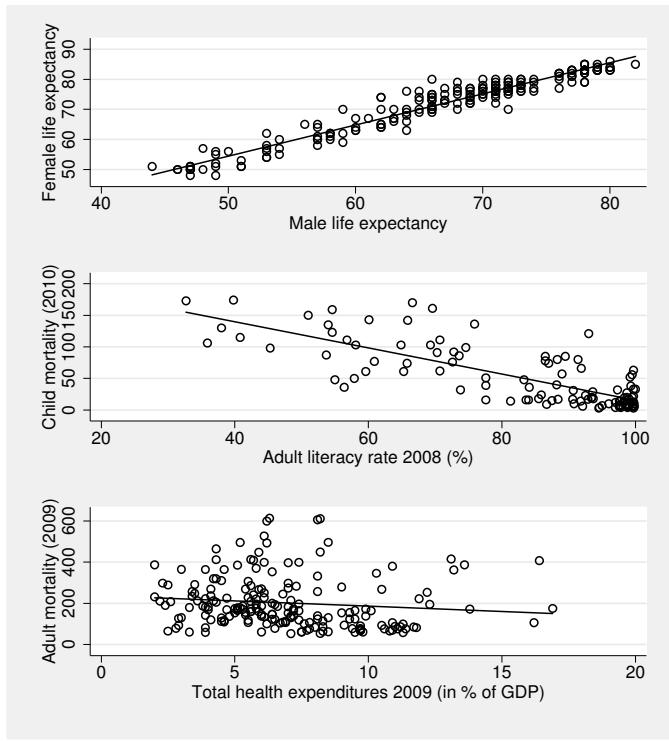
9 Introduction to linear regression

In the last chapter, we described the distributions of a few different variables for various subgroups. For example, we compared the distributions of income and political party affiliation for men and women using several techniques. One of those techniques was cross-tabulation, which we used to examine the relative frequencies of votes cast for different groups formed by the values of a second variable—gender, in this case (page 162). Applying a different technique to the income variable, we compared the distribution of income for men and women using statistics such as means, quantiles, and standard deviations (page 178). In other words, we looked at how income depends on gender. Therefore, `income` was our *dependent* variable, and `sex` was our *independent* variable.

The techniques described in chapter 7 provide a reasonably good representation of your data if you want to compare the distribution of one variable for a few different subgroups formed by a second variable. However, if you are interested in the relationship between two variables with many categories, a scatterplot may be more useful. A scatterplot is a graphical representation of the joint distribution of two variables. When you draw the scatterplot, each observation is plotted in two-dimensional space (along two axes). The coordinates of each point are the values of the variables for that particular observation. The values of the independent variable are graphed on the x axis, whereas the values of the dependent variable are graphed on the y axis.

Three examples of scatterplots can be seen in figure 9.1.¹ The first scatterplot shows data from 193 nations on the life expectancy at birth for females plotted against the life expectancy at birth for males. The dots are distributed from the lower-left corner to the upper-right corner. This distribution suggests that high life expectancies for males go along with high life expectancies for females. Cases such as these are called *positive relationships*.

1. The data for these examples are taken from the Global Health Observatory Data Repository of the World Health Organization (<http://apps.who.int/ghodata/#>). The dataset with the variables used for the figure is included in the data package of this book (`who2009.dta`).



Source: grscatter.do

Figure 9.1. Scatterplots with positive, negative, and weak correlation

The second scatterplot depicts the relationship between infant mortality and the adult literacy rate. There we find the data points for 110 nations spreading out from the upper-left corner to the lower-right corner. This means that when the literacy rate in a country is higher, the observed infant mortality rate is lower. This is called a *negative relationship*.

The third scatterplot shows the relationship between the adult mortality rate and the total health expenditures as a percentage of the gross domestic product. Here the observations from 191 different countries are distributed fairly evenly over the entire diagram. The relationship between health expenditures and lost healthy years is therefore not obvious. We can, at best, find a weak (negative) relationship.

All three graphs contain a solid line that summarizes the relationship between the two variables and is called a *regression line*. In the first scatterplot example, the dots are close to the regression line; there we have a *strong correlation*. In contrast, widely scattered clouds of dots, as in the third example, indicate a *weak correlation*. One way to measure the strength of the correlation is with Pearson's correlation coefficient r . A Pearson's correlation coefficient of 0 means that no relationship can be observed

between the two variables. Both -1 and $+1$ represent the strongest possible observed relationships, with -1 indicating a negative relationship and $+1$ indicating a positive relationship.

Creating scatterplots for different values of r is useful for getting an idea of the relationship. You can practice doing so by using a small demonstration we wrote for Stata.² Type

```
. do cplot 0.5
```

and you will see a scatterplot of two variables whose correlation coefficient is 0.5. You can vary the strength of the relationship by changing the number you enter for r after the `do cplot` command.

Regardless of the strengths of correlation, there is not necessarily a causal relationship between the variables. In figure 9.1, the life expectancy of women is not caused by the life expectancy of men. You can instead think of a common cause for both of them. You could hypothesize on the causal link between literacy and infant mortality, but neither scatterplots nor regression lines can test such an assumption (King, Keohane, and Verba 1994; Berk 2004).

A simple linear regression analysis aims to characterize the relationship between one dependent variable and one independent variable with a line. A straightforward generalization of this is multiple linear regression analysis, which characterizes the relationship between one dependent and *more than one* independent variables. The term *multivariate regression* is reserved for a technique for more than one *dependent* variables.

We begin by outlining the basic principle behind simple linear regression in section 9.1. We then extend the model to deal with multiple independent variables in section 9.2. Linear regression analysis requires us to make several assumptions, and section 9.3 introduces techniques to check those assumptions. Refinements of the basic model are the subject of section 9.4. In section 9.5, we show different ways to display regression results and discuss alternative methods of computing standard errors and other extensions of the linear regression model in section 9.6.

Although we will explain some of the statistical background, our main purpose is to show you how to perform regression analysis with Stata. You will need to do more reading to gain a full understanding of regression analysis. Books that work well with our approach are Hamilton (1992) and Fox (1997). You should also read Berk (2004) for a critical discussion of common mistakes.

2. Make sure that your current working directory is `c:\data\kk3`; see page 3.

9.1 Simple linear regression

9.1.1 The basic principle

Here we will introduce terms such as *ordinary least squares*, *residual sum of squares*, *predicted values*, and *regression*. If you are already familiar with these terms, you may skip this section.

The basic principle of all regression models is straightforward. To describe the relationship between your variables, you are looking for an equation that allows you to predict the values of a dependent variable as well as possible with the help of one or more independent variables. Consider the following example.

You have a hunch that the size of someone's home is determined by his or her net income. You believe that the higher someone's income is, the larger the home will be. At the same time, you know that homes are not of zero size if someone's income is zero. Hence, homes should have a certain minimum size. You could formalize your suspicion about the relationship between income and home size with the aid of a simple equation (let us use the Lopez family for our example):

$$\text{home size}_{\text{Lopez}} = \beta_0 + \beta_1 \times \text{income}_{\text{Lopez}} + \epsilon_{\text{Lopez}}$$

Or you could use symbols instead of text

$$y_{\text{Lopez}} = \beta_0 + \beta_1 x_{\text{Lopez}} + \epsilon_{\text{Lopez}} \quad (9.1)$$

where y and x could be symbols for any variable. In what follows, we will consistently use y for home size and x for income.

Our equation calculates the home size of the Lopez family, where the Lopez family is an arbitrary household in our dataset. The first term, β_0 , is the home size when the household income is zero. To this term, we add the term $\beta_1 x_{\text{Lopez}}$. The crucial part here is the parameter β_1 , which tells us how many square feet each additional Euro of household income can buy. If you multiply β_1 with the actual income of the Lopez family, you will get an estimate of how much larger the Lopez family home is compared to the home size for a family that has zero income.

Now you might argue that income is not the only variable that affects home size. For example, family size or the ages of family members might play a role, as well. You may, in fact, come up with several factors that could affect home size. If you do not know all the factors, the home size y , your calculation using the above equation, will always deviate from the observed values. This deviation is called the *error term*. In (9.1), the error term is indicated by ϵ_{Lopez} for the Lopez family.

Equation (9.1) above is just another way to write down the hunch that an individual's home size depends on a minimum home size, some quantity that rises with income and some other factors. In practice, we do not know the value of those parameters. We can only speculate how many square feet the home size for a family with zero income β_0 is and how large the quantity β_1 is that must be added to the home size with each additional Euro of income. Estimating those regression parameters (as β_0 and β_1 are often called) is the aim of regression analysis.

Let us assume you would, based on your past housing experience, make a rough estimate for the minimum home size and the effect of income.³ Based on those estimates, you could go ahead and predict the square footage of other people's places knowing nothing about those people except their income. To do so, you would replace x with their respective income in the formula

$$\hat{y}_i = b_0 + b_1 \times x_i \quad (9.2)$$

where b_0 and b_1 symbolize your estimates for the parameters β_0 and β_1 . (In statistics, Greek symbols commonly refer to unknown parameters, and Roman letters indicate their estimates that one has derived from empirical sources.) The subscript i indicates that we are now dealing with values from several households, not just the Lopez family. The estimated regression coefficients b_0 and b_1 do not have the subscript i . They are constant across individuals. Also compared with (9.1), the error term ϵ is missing, which means we ignore all factors besides income that could influence square footage. Instead, the predicted home size \hat{y}_i in (9.2) has a *hat* over y_i , symbolizing that it is an estimate.

Because you did not include any of the other factors beyond income that could influence the households' housing choices, the home sizes you predicted using the above equation will deviate from the actual values of these people's homes. If we know an individual's actual home size y_i and the predicted size of his or her home \hat{y}_i , we can compute the difference:

$$e_i = y_i - \hat{y}_i \quad (9.3)$$

This deviation e_i between the actual value and the predicted value is called the *residual*. Looking at the equations, we can say that our prediction will get better the smaller e_i is. Likewise, we could say that—given that our model is correct—the closer our estimated coefficients are to the true values of the parameters, the smaller e_i tends to get. Hence, it is reasonable to replace simple guessing based on day-to-day experience with statistical techniques. You can use several techniques to get estimates for the regression parameters. Here we will limit ourselves to the one that is the simplest and in wide use: ordinary least squares (OLS).

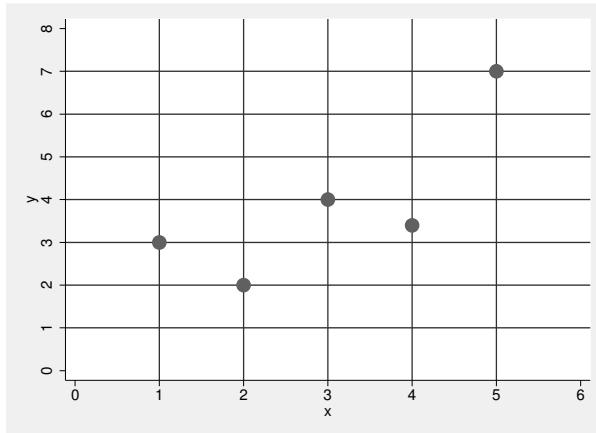
3. Maybe you would say that houses are at least 600 square feet and people would spend roughly one-third of their net income on housing. If one square foot costs €5, people could afford an additional square foot with every €15 of additional income, or in other words, each additional Euro of income would increase the home size by $1/15 \text{ ft}^2$.

OLS is based on a simple rule: make the difference between the predicted and observed values as small as possible. To put it differently, we want to choose the parameters that minimize the sum of the squared residuals. Thinking back on what we said earlier, that the error term in (9.1) includes all other effects that influence home size, it might seem funny to you that we try to find values for b_0 and b_1 that minimize the residuals. After all, there could be other important variables, like household size, that we do not include in our model. This is true! Trying to minimize the residuals implicitly makes the assumption that the error term is 0 in expectation. We will come back to this assumption in section 9.3.1.

To understand what minimizing the sum of the squared residuals means, look at the scatterplot in figure 9.2. Try to find a line that depicts the relationship between the two variables. You will find that not all points lie on one line. You might try to draw a line among the points so that the vertical distances between the points and the line are as small as possible. To find these distances, you might use a ruler.

The goal was to minimize the differences across all points, so looking at one of the distances will not provide you with enough information to choose the best line. What else can you do? You could try adding the distances for all the points. If you did this, you would notice that negative and positive distances could cancel each other out. To find a way around this problem, you might use the squared distances instead.

If you drew several lines and measured the distances between the points and every new line, the line with the smallest sum of squared distances would be the one that reflects the relationship the best. This search for the line with the best fit is the idea behind the OLS estimation technique: it minimizes the sum of squared residuals (e_i^2). The points on the line represent the predicted values of \hat{y}_i for all values of X . If your model fits the data well, all points will be close to the straight line, and the sum of the squared residuals will be small. If your model does not fit the data well, the points will be spread out, and the sum of the squared residuals will be relatively large.



Source: `grreg1.do`

Figure 9.2. Exercise for the OLS principle

We have prepared a small demonstration of the OLS solution to the regression problem in figure 9.2. Typing

```
. do grreg2.do
```

causes figure 9.2 to be displayed with the regression line.

We can also present the OLS principle a bit more formally. We are looking for those parameters (b_0 and b_1) in (9.3) for which the sum of the squared residuals (the residual sum of squares [RSS]) is at a minimum. Those parameters are the y -axis intercepts and the slopes of the lines we drew. A search for the best fit using a trial-and-error technique like the one described above would be time consuming. Using mathematical techniques to minimize the RSS is an easier way to find our parameters that more reliably leads to the correct solution. Mathematically, the RSS can be written as the difference between the observed and predicted values:

$$\text{RSS} = \sum_{i=1}^n e_i^2 = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (9.4)$$

Substituting for \hat{y}_i , we can write the above equation as

$$\text{RSS} = \sum e_i^2 = \sum (y_i - b_0 - b_1 x_i)^2 \quad (9.5)$$

Now that we have defined the RSS mathematically, we can use the OLS technique to minimize it.⁴ This means that we must find values for b_0 and b_1 for which (9.5) is as small as possible. To do this, we can take the first partial derivatives of (9.5) with respect to b_0 and b_1 , set them equal to 0, and solve for b_0 and b_1 . At this point, it is not particularly important that you be able to take the derivative yourself. The entire technique is nothing more than a search for the minimum of a function with two unknowns.

If on the other hand, you wish to review the high school and college math necessary for taking partial derivatives, you can find a helpful review in Hagle (1996, 38–58).⁵

Before we continue with the mathematics, we will show you how to compute a regression with Stata, and you will see how easy and helpful it is to use statistical packages for these kinds of computations. But be careful: despite the simplicity of the computational work, you must always think carefully about what exactly you are doing. In this chapter, we will look at substantive problems caused by naïvely applying these regression techniques.

9.1.2 Linear regression using Stata

Here we will explain how to fit a linear regression model with Stata. In the previous subsection, we voiced a suspicion that home size is influenced by net household income. You might now be interested in a specification of this relationship. A good place to begin would be with a linear regression of home size (`size`) on net household income (`hhinc`). The Stata command you will need to perform your regression is pretty simple:

-
- 4. The exact mathematical procedure for this technique has been presented in several different ways. For fans of a graphic interpretation, we recommend starting with Cook and Weisberg (1999) or Hamilton (1992).
 - 5. To reconstruct the transformations used in finding values for b_0 and b_1 for which RSS is at a minimum, you can do so as follows:

$$\frac{\partial \text{RSS}}{\partial b_0} = -2 \sum y_i + 2nb_0 + 2nb_1 \sum x_i$$

If you set this partial derivative equal to 0 and solve for b_0 , you will get

$$b_0 = \bar{y} - b_1 \bar{x}$$

Following the same principle, you can find the first partial derivative with respect to b_1 :

$$\frac{\partial \text{RSS}}{\partial b_1} = -2 \sum y_i x_i + 2b_0 \sum x_i + 2b_1 \sum x_i^2 = 0$$

Now you replace b_0 with $\bar{y} - b_1 \bar{x}$. After a few transformations, you end up with

$$b_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

You can find a more detailed presentation of this derivation in Hamilton (1992, 33).

. use data1, clear (SOEP 2009 (Kohler/Kreuter))						
. regress size hhinc						
Source	SS	df	MS		Number of obs =	5407
Model	263141566	1	263141566		F(1, 5405) =	1613.34
Residual	881575997	5405	163103.792		Prob > F =	0.0000
Total	1.1447e+09	5406	211749.457		R-squared =	0.2299
					Adj R-squared =	0.2297
					Root MSE =	403.86
size	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
hhinc	.0082545	.0002055	40.17	0.000	.0078516	.0086574
_cons	800.9835	9.404917	85.17	0.000	782.5461	819.4209

Clearly, the command consists of the `regress` statement and a list of variables. The first variable is the dependent variable, and the second is the independent variable. The output contains three different sections: the table of ANOVA results in the upper left corner, the model fit table in the upper right corner, and the table of coefficients in the bottom half of the output.

The table of coefficients

The following description is confined to the coefficients in the first column of the table of coefficients. For the meaning of the other columns, please refer to chapter 8. We would like to stress that if you are interested in inference statistics for the commands shown in this chapter, you need to `svyset` the dataset as shown on page 220 and to prefix all regression commands with `svy`.

At the bottom of the table in the column labeled `Coef.`, you will find the estimated regression coefficients—that is, the values for b_0 and b_1 —from (9.3).

To the right of the estimated regression coefficients are their standard errors, significance tests, and 95% confidence intervals. These statistics are used to evaluate the accuracy of the estimated coefficients (see chapter 8).

The value for b_0 is written in the regression output row labeled `_cons`. b_0 is 800.984 in this example. According to this model, the predicted home size for a family with no (zero) income is 801 square feet. The value for b_1 is displayed in the row that begins with `hhinc` and is about 0.008. According to the regression model, the home size will increase by about 0.008 ft² with every additional Euro in the annual household income.

Assuming that the Lopez family has a net annual income of €36,749, you can use (9.1) to estimate how big the family's home might be:

$$\hat{y}_{\text{Lopez}} = 800.984 + 0.008 \times 36,749$$

You can calculate this amount directly within Stata using the `display` command, much as you would use a pocket calculator. Type

```
. display 800.984 + .008 * 36749
1094.976
```

If you use the numbers displayed in the table of coefficients, you must deal with two problems: 1) typing numbers by hand often leads to mistakes and 2) the figures in the output have been rounded. For computations like the one above, we recommend using the results saved internally by Stata (see chapter 4). Commands that fit regression models are considered to be e-class commands in Stata, so you can look at the saved results with the command `ereturn list`. If you do this, you might find yourself searching in vain for the estimated regression coefficients because they are all stored in a matrix named `e(b)`. The easiest way to access the values contained in this matrix is to use the construction `_b[varname]`, where *varname* is replaced by the name of either an independent variable or the constant (`_cons`).

The computation for the Lopez family would then look like this:

```
. display _b[_cons]+_b[hhinc]*36749
1104.3281
```

This number differs a bit from the number we computed above because the results saved by Stata are accurate to about the 16th decimal place. You can see the effect of raising income by €1 on home size. If you enter €36,750 instead of €36,749 as the value for income, you will see that the predicted value for home size increases by $b_1 = b_{\text{hhinc}} = 0.0082545 \text{ ft}^2$.

You might not be interested in an estimated home size for a family with a certain income but instead in the actual home sizes of all families in our data who have that income. To see the sizes of the homes of all the families with a net household income of €36,749, you could use the following command:

```
. list size hhinc if hhinc==36749
```

	size	hhinc
1756.	1076	36749
1757.	1072	36749
1758.	1072	36749

As you can see here, the predicted home size of $1,104 \text{ ft}^2$ is not displayed, but values between $1,072 \text{ ft}^2$ and $1,076 \text{ ft}^2$ appear instead. The observed values of y_i differ from the predicted values \hat{y}_i . These differences are the residuals.

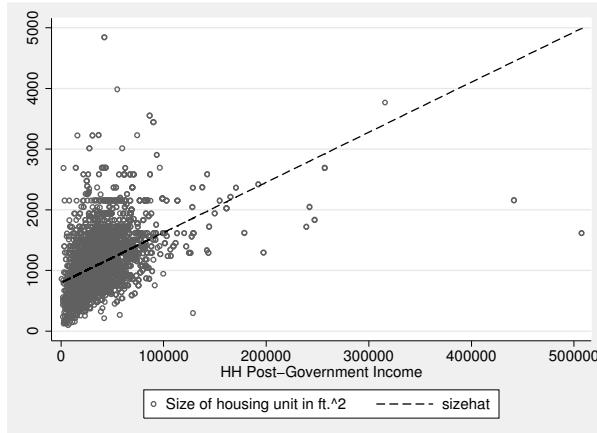
If you want to compute the predicted values for every household in your dataset, you could use the saved estimates of the regression coefficients.⁶ To compute the predicted values this way, you would type⁷

```
. generate sizehat=_b[_cons]+_b[hhinc]*hhinc
```

This is the same principle that was used in the previous `display` command, except that the home size is predicted not only for the Lopez family but for all families. The result of this computation is stored in the `sizehat` variable. We use the suffix `hat` to indicate that this is a predicted variable.⁸

Predicted values are very helpful for interpreting the results of regression models. For instance, you can draw a scatterplot with `size` against `hhinc`, overlaid by a line plot of the predicted values (`sizehat`) against `hhinc` (that is, the regression line).

```
. scatter size hhinc, msymbol(oh) || line sizehat hhinc, sort
```



Note as an aside that the `twoway` plottype `lfit` performs the regression analysis and the creation of predicted values in the background. You can therefore also get the above figure directly with

```
. scatter size hhinc, msymbol(oh) || lfit size hhinc
```

6. You can use the saved regression coefficients anywhere Stata expects an expression; see section 3.1.5.

7. After entering this command, you will get a warning that some missing values have been generated. Those missing values are for all the families for whom the dataset contains no income information.

8. We placed a “hat” (circumflex) on y in the above equations to indicate a predicted value (\hat{y}) as opposed to a value actually measured for a certain family (y).

Because predicted values are so important for interpreting regression models, Stata has two convenience commands—`predict` and `margins`—that let you deal with predicted values. The command `predict`, to start with, is just an easier way to get a variable with the predicted values. If you type `predict` followed by a variable name, Stata will store the predicted values in a new variable with the specified name. Here we use `yhat` as the variable name:

```
. predict yhat
```

The new variable `yhat` contains the same values as the `sizehat` variable. If you want to convince yourself that this is the case, type `list sizehat yhat`. Because it is used after estimation, the `predict` command is called a *postestimation* command.

`margins` is yet another postestimation command. In its simplest form, it shows the averages of predicted values. If you type

```
. margins
Predictive margins                               Number of obs =      5407
Model VCE    : OLS
Expression   : Linear prediction, predict()

```

	Delta-method				
	Margin	Std. Err.	z	P> z	[95% Conf. Interval]
_cons	1107.638	5.492295	201.67	0.000	1096.873 - 1118.403

without any options, you get the average of the predicted values for home size along with its inference statistic. Being the same as the average of the dependent variable itself, this number is fairly uninteresting. However, `margins` also can show the averages of the predicted values at specified levels of the independent variables. For example, the command

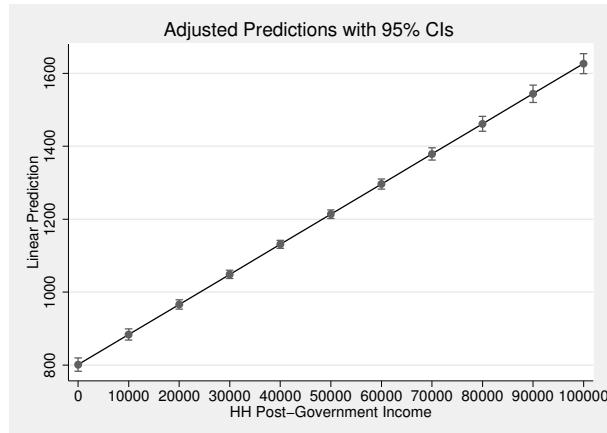
```
. margins, at(hhinc=(0(10000)100000)) vsquish
Adjusted predictions                                         Number of obs = 5407
Model VCE : OLS
(output omitted)
```

	Delta-method					
	Margin	Std. Err.	z	P> z	[95% Conf. Interval]	
_at						
1	800.9835	9.404917	85.17	0.000	782.5502	819.4168
2	883.5285	7.829203	112.85	0.000	868.1835	898.8734
3	966.0735	6.525876	148.04	0.000	953.283	978.8639
4	1048.618	5.685452	184.44	0.000	1037.475	1059.762
5	1131.163	5.523436	204.79	0.000	1120.338	1141.989
6	1213.708	6.094179	199.16	0.000	1201.764	1225.653
7	1296.253	7.226089	179.39	0.000	1282.091	1310.416
8	1378.798	8.702896	158.43	0.000	1361.741	1395.856
9	1461.343	10.3784	140.81	0.000	1441.002	1481.685
10	1543.888	12.1708	126.85	0.000	1520.034	1567.743
11	1626.433	14.0354	115.88	0.000	1598.924	1653.942

shows the averages of the predicted house sizes for household incomes in increments of 10,000 between the values 0 and 100,000. To achieve this, we specified a *numlist* (see section 3.1.7) for the values of *hhinc* inside the option *at()* (*vsquish* just saves us space). The output shows that the average predicted house size is around 801 ft² for households without income (the lowest *at-level*), while it is around 1,626 ft² for household incomes of €100,000 (the highest *at-level*).

The command *marginsplot* displays the results of the *margins* command graphically. When typing *marginsplot* immediately after *margins*, you will get the same regression line as shown in the previous figure, however, this time the 95% confidence intervals around the predictions are by default included:

```
. marginsplot
```



The last figure was nice and simple to obtain. You will learn later on that `margins` and `marginsplot` are also very helpful for more complicated regression models (see section 9.5).

Despite its use for the interpretation of regression models, predicted values are also used to calculate the values of the residuals. They can be obtained by generating the differences between the observed and predicted values:

```
. generate resid1=size-sizehat
```

This difference is nothing more than the distance you measured between each point and the line in the figure on page 259.

You can also compute the residuals by using the `predict` postestimation command with the `residuals` option and specifying a variable name (here `resid2`):⁹

```
. predict resid2, resid
```

The table of ANOVA results

ANOVA is short for analysis of variance. We use the term “table of ANOVA results” to describe the upper left section of the Stata regression output, where you will find the variation in the dependent variable divided into an explained portion and an unexplained portion. For handy reference, we reproduced the table of ANOVA results that you have already seen on page 261:

Source	SS	df	MS
Model	263141566	1	263141566
Residual	881575997	5405	163103.792
Total	1.1447e+09	5406	211749.457

We can learn a bit more about the table of ANOVA results by using a fictional example. Say that you are asked to predict the size of an home belonging to a student named Paul. If you do not know anything about Paul, you might answer that his home is as big as the average student home. Here your guess would be reasonable because the average home size is the value with which you get the smallest squared error.

Table 9.1 lists the home sizes and household sizes of three students in a hypothetical city. The average student home size in that city is 590 ft^2 , which we calculated using data for all the students in the city, not just the ones listed here.¹⁰

9. Please resist the temptation to set `e` as a name for the residuals. The name `e` may, in principle, be a valid variable name, but using it might lead to confusion if scientific notation is used for numbers. See section 5.1.1 for a list of variable names you should avoid.

10. We got the idea of using a table like this one from Hair et al. (1995).

Table 9.1. Apartment and household size

	Apt. size	City	Diff.	HH size	Estim.	Residual
Paul	430	590	-160	1	480	-50
John	590	590	0	2	640	-50
Ringo	860	590	+270	3	800	+60

If you use 590 ft^2 to estimate the size of Paul's home, you end up with a number that is 160 ft^2 too high.¹¹ If you use the mean to estimate the other students' home sizes, then in one case you make a correct prediction and in the other case you underestimate the student's home size by 270 ft^2 . If you take the squares of these differences and sum them, the result is a total squared deviation of $98,500 \text{ ft}^4$. This number is usually called the *total sum of squares* (TSS). In general,

$$\text{TSS} = \sum (y_i - \bar{y})^2$$

This corresponds to the expression you find in the numerator of the formula for the variance (s^2). The TSS is therefore sometimes also called the *variation*.

Maybe you should not make your prediction using only the mean. You might wish to use other information you have about the students. After all, it is reasonable to assume that the size of the home increases with the number of people living there. If all the students you know have bedrooms that are about 160 ft^2 , you might think this number holds true for most other students. So the home would have to have at least 160 ft^2 for each of the students living there, but it is likely to be even larger. A home usually has at least one bathroom and a kitchen, and you might think that together they take up about 320 ft^2 . You might describe this hypothesis using the equation below:

$$y_i = 320 + 160x_i$$

You could use that model to compute an home size for each household size. If you did this, you would calculate the difference between the actual home size and the home size you predicted with your model; this is the amount displayed in the last column of the table. To compare these differences with the TSS we calculated above, you would have to square these deviations and sum them. If you did this, you would have calculated the RSS we introduced in section 9.1.1. For your hypothesis, the value of RSS is 8,600.

¹¹ In table 9.1, the difference between the observed value and the predicted mean is calculated as follows: $430 - 590 = -160$.

If you subtract the RSS from the TSS, you get the *model sum of squares* (MSS), which indicates how much you have been able to improve your estimation by using your hypothesis:

$$\begin{array}{rcl} \text{TSS} & = & 98,500 \\ -\text{RSS} & = & 8,600 \\ \hline = \text{MSS} & = & 89,900 \end{array}$$

The squared residuals that you get when you use household size to predict home size are about 89,900 smaller than the ones you got without taking this knowledge into account. That means that the actual home sizes are much closer to your predicted values when you use household size in making your prediction.

Therefore, the MSS can be regarded as a baseline to measure the quality of our model. The higher the MSS, the better are your predictions compared with the prediction based solely on the mean. The mean can be regarded as the standard against which to judge the quality of your prediction.

In the ANOVA part of the regression output, you will find information about the MSS, RSS, and TSS in the column labeled **SS**. The first row of numbers (**Model**) describes the MSS, the second (**Residual**) describes the RSS, and the third (**Total**) describes the TSS. If you look at the output on page 266, you will see that our RSS is 881,575,997. The sum of the squared residuals taking the mean as the estimate (TSS) is 1.145×10^9 , and the difference between these two quantities (MSS) is 263,141,566.

The column labeled **df** contains the number of degrees of freedom,¹² which equals the number of unknowns that can vary freely. For the MSS, the number of degrees of freedom is just the number of independent variables included in the model, that is, $k - 1$, where k is the number of regression coefficients (the constant and all independent variables). The number of degrees of freedom for the RSS is $n - k$, where n is the number of observations. The number of degrees of freedom for the TSS is $n - 1$. The last column contains the average sum of squares (MS). You may want to compute these numbers yourself by dividing the first column by the second column (the number of degrees of freedom).

The model fit table

Here is the model fit table from page 261:

Number of obs	=	5407
F(1, 5405)	=	1613.34
Prob > F	=	0.0000
R-squared	=	0.2299
Adj R-squared	=	0.2297
Root MSE	=	403.86

12. For a well-written explanation of the concept of degrees of freedom, see Howell (1997, 53).

Earlier, we showed that the MSS tells you how much the sum of the squared residuals decreases when you add independent variables to the model. If you were looking at models with different independent variables, you might want to compare the explanatory power of those models using the MSS. You could not, however, use the *absolute* value of the MSS to do so. That value depends not only on the quality of the model but also on how much variation there was in the first place as measured by TSS.

To compare models, you must look at how much the model reduces the squared residuals relative to the total amount of squared residuals. You can do this using the coefficient of determination, or R^2 :

$$R^2 = \frac{\text{MSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}} = 1 - \frac{\sum e_i^2}{\sum (y_i - \bar{y})^2}$$

R^2 represents the squared residuals that are explained by the model as a share of the total squared residuals. When we say that the model explains a portion of the residuals, we mean that portion of the residuals of the model without independent variables that disappears when we use a model *with* independent variables. For this reason, R^2 is called the *explained variation* or the *explained variance*. You will find this statistic in the model fit table of the Stata output, where it is called R-squared.

Here $R^2 = 0.23$, meaning that household size (the independent variable in our model) explains 23% of the variation in home size.

R^2 is a useful indicator of a model's explanatory power, but it should not be considered in isolation. Unfortunately, people often evaluate the quality of a regression model only by looking at the size of R^2 , which is not only invalid but also dangerous. In section 9.3, we will show you why.

One alternative to R^2 is the root mean squared error (MSE), which is the square root of the average residual of the model from the table of ANOVA results:

$$\text{root MSE} = \sqrt{\frac{\text{RSS}}{n - k}}$$

This statistic is easy to interpret, because it has the same units as the dependent variable. In our example, a root MSE of 403.86 can be interpreted as showing that we are, on average for our data, about 404 ft² off the mark in predicting a respondent's home size with our model. (This interpretation is not completely correct because it is not a literal average. After all, $\sqrt{\sum e_i^2} \neq \sum e_i$. But the above interpretation seems justified to us.)

There are two further rows of the model fit table that we should talk about here: the rows labeled “ $F(1, 5405)$ ” and “Prob > F ” (for “Adj R-squared”, see page 274). The values in these rows are included because we are using a sample to test our regression model and therefore want some measure of its significance (see chapter 8). The F value is calculated using the following equation:

$$F = \frac{\text{MSS}/(k - 1)}{\text{RSS}/(n - k)}$$

This F statistic is the ratio of the two values in the third column of the ANOVA table. It is F distributed and forms the basis of a significance test for R^2 . The value of F is used to test the hypothesis that the R^2 we estimated from our sample data is significantly different from the population value of 0.¹³ That is, you want to estimate the probability of observing the reduction in RSS in the model if, in fact, the independent variables in the model have no explanatory power.¹⁴ The value listed for “Prob > F ” gives the probability that the R^2 we estimated with our sample data will be observed if the value of R^2 in the population is actually equal to 0.

9.2 Multiple regression

Load `data1.dta` into working memory:

```
. use data1, clear
```

Earlier, we introduced linear regression with one independent variable. A multiple regression is an extension of the simple linear regression presented in that section. Unlike simple regression, you can use several independent variables in a multiple regression. Analogous to (9.1), the model equation of the multiple linear regression is

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_{K-1} x_{K-1,i} + \epsilon_i \quad (9.6)$$

The equation for the simple linear regression has been extended with more X variables and the attendant regression coefficients. You might want to use a model like this for two reasons.

In section 9.1.2, you fit a simple linear regression model of the home size on household income. You were able to explain 23% of the variation in home size with this regression, and the average error in predicting home size was 404 ft². If you want to maximize the predictive power of our model, there is no reason to be satisfied with the performance of this simple model. You could improve the predictive power of our model by including other variables. This would be one reason for using a regression with more than one independent variable.

13. A description of this relationship can be found in Gujarati (2003, 253–260).

14. This F test is often called a test of the null hypothesis—that all coefficients but the constant are 0 (Gujarati 2003, 257). Incidentally, the confidence intervals might *not* contain the value 0, but the overall model may nevertheless not be significant.

A second reason is a bit more complicated. In the previous section, we study how house size co-varies with household income. You have, however, reason to assume that household income is related to the size of the household, because more family members might contribute to the overall household income. At the same time, it is reasonable to assume that households with more members need more space than those with fewer members. Thus the regression coefficient that we estimate for household income may include the effect of household size. It is therefore sensible to ask what the effect of household income net of the effect of household size would be. This question could be answered by including another variable, household size, into the model.

We will show you how to fit a multiple linear regression model in Stata and then interpret the estimated regression coefficients. Then we will present some computations that are specific to this kind of regression. Finally, we will illustrate what is meant by the formal phrase “controlling for” when it is used for the interpretation of coefficients in multiple regression models (section 9.2.3).

9.2.1 Multiple regression using Stata

The Stata command for computing a multiple regression is the same as that for simple linear regression. You just enter more independent variables at the end of the list of variables; the order in which you enter them does not matter. You can apply the general rules for lists of variables (page 43), but remember that the dependent variable is always the first one in your list.

The output for a multiple linear regression resembles the one for a simple linear regression except that for each additional independent variable, you get one more row for the corresponding estimated coefficient. Finally, you obtain the predicted values using `predict` and `margins` as you did earlier.

For example, say that you want to fit a regression model of home size that contains not only household size and household income but also a location variable for the difference between East and West Germany and an ownership variable indicating owned and rented living space. To do this, you will need to recode some of the variables:¹⁵

```
. generate owner = renttype == 1 if !missing(renttype)
. generate east = state>=11 if !missing(state)
```

15. See chapter 5 if you have any problems with these commands.

Now you can fit the regression model:

. regress size hhinc hysize east owner									
Source	SS	df	MS	Number of obs = 5407 F(4, 5402) = 1075.80 Prob > F = 0.0000 R-squared = 0.4434 Adj R-squared = 0.4430 Root MSE = 343.44					
Model	507558816	4	126889704						
Residual	637158747	5402	117948.676						
Total	1.1447e+09	5406	211749.457						
size	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]				
hhinc	.0046358	.0001956	23.70	0.000	.0042523	.0050193			
hysize	83.56691	4.396585	19.01	0.000	74.94783	92.18599			
east	-106.6267	10.88597	-9.79	0.000	-127.9676	-85.28581			
owner	366.1249	9.889078	37.02	0.000	346.7383	385.5115			
_cons	550.58	12.39905	44.41	0.000	526.2729	574.8872			

You interpret the estimated coefficients in a multiple regression model just as you do in the simple linear regression. The only difference is that the b coefficients are now estimated *controlling* for the effect of all the other independent variables. We will discuss the meaning of that phrase in section 9.2.3. For now, we will confine ourselves to once again illustrating the formal interpretation of the coefficients.

The regression coefficients reflect the average change in the size of the home as the independent variable in question increases by one unit, holding all other independent variables constant. The estimated coefficient might, for example, be interpreted as saying that “with each additional Euro of household income, the predicted size of the home increases by an average of about 0.005 ft^2 ”. Similarly, the predicted home size increases by an average of about 83.567 ft^2 for each additional person in the household.

The variables `east` and `owner` are dummy variables, or variables that have only two categories denoted by the values 0 and 1.¹⁶ In principle, you interpret these variables just as you interpret all the other variables. For example, let us look at the `owner` variable, which has a value of 0 for all renters and 1 for all owners. For each unit by which the `owner` variable increases, the home increases by an average of about 366 ft^2 . Because a dummy variable can be increased by one unit only once, we could also say, “Owners live in homes that are, on average, about 366 ft^2 larger than the ones in which renters live.” Likewise, the homes in East Germany are, on average, around 107 ft^2 smaller than the homes in West Germany.

The regression constant indicates how large a home is whose observation has a value of 0 for all variables included in the model. In the example, this value would refer to home size for West Germany households with no household income and no household members. This is clearly useless information. A more sensible interpretation for the constant can be reached by subtracting from all values of a continuous variable the average value of that variable. In doing so, that new *centered* version of the variables will have a mean of 0, and the constant term of the regression will then refer to observations

16. There are other possibilities for coding binary variables (Aiken and West 1991, 127–130).

with the average value of the continuous variables. If we center the two continuous variables of our last example by applying the methods described in chapter 4,

```
. summarize hhsiz
. generate c_hhsiz = hhsiz - r(mean)
. summarize hhinc
. generate c_hhinc = hhinc - r(mean)
```

and rerun the regression model using these centered variables,

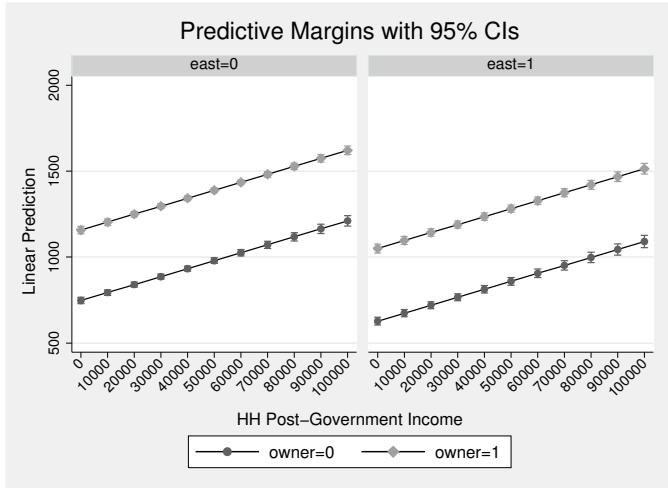
```
. regress size c_hhinc c_hhsiz east owner, noheader
```

size	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
c_hhinc	.0046358	.0001956	23.70	0.000	.0042523 .0050193
c_hhsiz	83.56691	4.396585	19.01	0.000	74.94783 92.18598
east	-106.6267	10.88597	-9.79	0.000	-127.9676 -85.28581
owner	366.1249	9.889078	37.02	0.000	346.7383 385.5115
_cons	940.9296	7.511176	125.27	0.000	926.2047 955.6546

we notice that all estimated coefficients remain the same except the constant. The predicted house size of renters from West Germany with an average income and average household size is 941 ft².

Postestimation commands can also be used after multiple regression models. The commands `margins` and `marginsplot` provide a neat way to illustrate the model results. In particular, when interpreting complicated models, such illustrations can be quite helpful. Here we use `margins` again to calculate the mean predicted values for various values of `hhinc` in increments of 10,000 between 0 and 100,000, this time separately for all combinations of `east` and `owner`. We then use `marginsplot` to show the results graphically. Before we do all that, we refit the model with the original variables because it is neither necessary nor advantageous to use centered variables for the graphical display.

```
. regress size hhinc hhsiz east owner
. margins, at(hhinc=(0(10000)100000)) over(owner east)
. marginsplot, by(east) xlabel(,angle(forty_five))
```



In this figure, the slope of each line represents the influence of the first variable mentioned in the option `at()` of `margins`—household income—on household size.¹⁷ The steeper the slope, the stronger is the estimated effect of household income on house size. Inside each plot, the difference between the two lines represents the estimated influence of house ownership. The bigger the difference between the two lines, the stronger is the effect of house ownership on house size. Finally, the difference between the two corresponding lines of each panel represents the difference in the average house sizes of West and East Germany.

Note that what you see in these graphs are the estimated effects according to the specified model. For example, we did not allow for a difference in the effect of house ownership on house size between East and West Germany. If you suspect that, for example, in East Germany ownership has less effect on the house size or that effects of income on house size are stronger in West Germany than in East Germany, you would need to specify this explicitly in the model through interaction effects (see section 9.4.2).

9.2.2 More computations

Adjusted R²

In adding the two dummy variables and the household size variable to our regression model, you have increased R^2 from 23% to 44%. This is an obvious improvement in the explanatory power of our model, but you need to put this improvement in perspective:

¹⁷. This figure is an example of a *conditional-effects plot*; read more about conditional-effects plots in section 9.5.3.

R^2 almost always increases if you add variables to the model.¹⁸ The effect of these additional variables on R^2 is offset by the effect of additional observations. Having more observations tends to result in a lower R^2 than you would obtain by fitting the same model with fewer observations. You can safeguard against misleading increases in R^2 by making sure that you have enough observations to test your model. In the example above, the ratio between observations and independent variables that was used in the model is quite favorable. However, if you intend to work with only a small number of observations (for example, if your dataset comprises country-level information for European countries) and you use many independent variables, R^2 will quickly become an unreliable measure.¹⁹

Perhaps it will be easier to understand why a small number of observations leads to a higher R^2 if you imagine a scatterplot with two points. These two points can be easily connected by a line, which is the regression line. Now you have explained all the variance, because there are no distances left between either of the points and the line. But does this mean that the two variables for which you made the scatterplot are really related to each other? Not necessarily. Imagine that you plotted the gross national products of Great Britain and Germany against the lengths of their coasts and drew a regression line. You would be able to explain the difference between the gross national products of Germany and Great Britain “perfectly”; at the same time, you would be forced to leave the scientific community.

Given the effects of the number of observations and the number of independent variables on R^2 , you may want a more meaningful measure of your model’s explanatory power. The adjusted R^2 (**Adj R-squared**) results from a correction that accounts for the number of model parameters k (everything on the right side of your equation) and the number of observations (Greene 2012, 35)

$$R_a^2 = 1 - \frac{n-1}{n-k} (1 - R^2)$$

where k is the number of parameters and n is the number of observations. As long as the number of observations is sufficiently large, the adjusted R^2 will be close to R^2 . Here they differ only in the fourth decimal place.

18. The only situation in which R^2 does not increase when you add variables is when the coefficient of the additional variables is exactly equal to 0. In practice, this case is almost never observed.

19. You will find a list of problems related to the use of R^2 in Kennedy (2008, 26–28).

Standardized regression coefficients

In our regression model, the estimated coefficient for household size is much larger than the one for household income. If you look only at the absolute size of the estimated coefficients, you might be tempted to assume that the household size has a larger influence on home size than does household income. But you will recognize that the estimated coefficients reflect how much a dependent variable changes if the independent variable is changed by one unit. Thus you are comparing the change in home size if household income increases by €1 with the change in home size if the size of the household increases by one person!

To compare the effects of variables measured in different units, you will often use the standardized form of the estimated regression coefficients (b_k^*), which are calculated as follows

$$b_k^* = b_k \frac{s_{X_k}}{s_Y} \quad (9.7)$$

where b_k is the estimated coefficient of the k th variable, s_Y is the standard deviation of the dependent variable, and s_{X_k} is the standard deviation of the k th independent variable.

The standardized estimated regression coefficients are often called beta coefficients, which is why you use the `beta` option with the `regress` command to display them. If you want to reexamine the estimated coefficients of your last model in standardized form, you can redisplay the results (with no recalculation) by typing `regress, beta`. If you type this command, Stata displays the standardized (`beta`) coefficients in the rightmost column of the table of coefficients:²⁰

<code>. regress, beta noheader</code>					
size	Coef.	Std. Err.	t	P> t	Beta
hhinc	.0046358	.0001956	23.70	0.000	.2692651
hhsiz	83.56691	4.396585	19.01	0.000	.2115789
east	-106.6267	10.88597	-9.79	0.000	-.1001275
owner	366.1249	9.889078	37.02	0.000	.3972494
_cons	550.58	12.39905	44.41	0.000	.

The beta coefficients are interpreted in terms of the effect of standardized units. For example, as household income increases by one standard deviation, the size of the home increases by about 0.27 standard deviation. In contrast, a one-standard-deviation increase in household size leads to an increase in home size of about 0.21 standard deviation. If you look at the beta coefficients, household income has a stronger effect on home size than does the size of the household.

20. The `noheader` option suppresses the output of the ANOVA table and the model fit table.

Understandably, using the standardized estimated regression coefficients to compare the effect sizes of the different variables in a regression model is quite popular. But people often overlook some important points in doing so:

- You cannot use standardized regression coefficients for binary variables (for example, dummy variables like `east` and `owner`). Because the standard deviation of a dichotomous variable is a function of its skewness, the standardized regression coefficient gets smaller as the skewness of the variable gets larger.²¹
- If interaction terms are used (see section 9.4.2), calculating b_k^* using (9.7) is invalid; if interactions are included in your model, you cannot interpret the beta coefficients provided by Stata. If you want to study effect sizes with beta coefficients that are appropriate for interactions, you must transform all the variables that are part of the interaction term in advance, using a z standardization (Aiken and West 1991, 28–48).
- You should not compare standardized regression coefficients estimated with different datasets, because the variances of the variables will likely differ among those datasets (Berk 2004, 28–31).

9.2.3 What does “under control” mean?

The b coefficients from any regression model show how much the predicted value of the dependent variable changes with a one-unit increase in the independent variable. In a multiple regression model, this increase is calculated *controlling* for the effects of all the other variables. We see the effect of changing one variable by one unit while holding all other variables constant. Here we will explain this concept in greater detail by using a simpler version of the regression model used above. Here only the estimated regression coefficients are of interest to us:

<code>. regress size hhinc hysize, noheader</code>						
size	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
hhinc	.0064471	.0002138	30.16	0.000	.006028	.0068662
hysize	106.8913	4.904582	21.79	0.000	97.2763	116.5062
_cons	589.1072	13.26019	44.43	0.000	563.1119	615.1025

Look for a moment at the estimated coefficient for household income, which differs from the coefficients we estimated both for the simple model (page 261) and for the multiple model (page 272). What is the reason for this change? To find an answer,

21. To make this point clear, we wrote a small do-file demonstration: `anbeta.do`. This program fits 1,000 regressions with a dichotomous independent variable that takes on the values 0 and 1. In the first regression, no observation has a value of 1 for the independent variable. In each additional regression, the number of observations where $X = 1$ increases by 1 until the last regression, where all cases have the value 1 for the independent variable. A figure is drawn with the beta coefficients from each of those 1,000 regressions.

you need to estimate the coefficient for household income in a slightly different way. To begin, compute the residuals of the regression of home size on household size:

```
. regress size hsize, noheader
(output omitted)
. predict e_fs, resid
```

When you do this, you create a new variable that stores the residuals: `e_fs`. Before continuing, you should give some serious thought to the meaning of those residuals.

We suggest that the residuals reflect the size of the home adjusted for household size. The residuals reflect that part of the home size that has nothing to do with household size. You could also say that the residuals are that part of the information about home size that cannot already be found in the information about the household size.

Now compute the residuals for a regression of household income on household size:

```
. regress hhinc hsize, noheader
(output omitted)
. predict e_hh, resid
(4 missing values generated)
```

These residuals also have a substantive interpretation. If we apply the above logic, they reflect that part of household income that has nothing to do with household size. They therefore represent household income *adjusted for* household size.

Now fit a linear regression of `e_fs` on `e_hh`.

. regress e_fs e_hh, noheader						
	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
e_hh	.0064471	.0002138	30.16	0.000	.006028	.0068662
_cons	.2920091	5.265751	0.06	0.956	-10.03098	10.615

Take a close look at the *b* coefficient for `e_hh`, which corresponds to the coefficient in the multiple regression model you estimated above.²² If you interpreted this estimated coefficient in the same way as one from a simple linear regression, you might say that home size, adjusted for household size, increases about 0.006 ft^2 with each additional Euro of yearly household income, adjusted for household size. The same interpretation holds true for the coefficients in the multiple regression model. The regression coefficients in the multiple regression model therefore reflect the effect of the independent variable in question on the dependent variable, adjusted for the effect of all other independent variables. This is what “controlling for” means.

22. Differences are due to rounding errors.

9.3 Regression diagnostics

It is so easy to fit a multiple regression model using modern statistical software packages that people tend to forget that there are several assumptions behind a multiple regression; if they do not hold true, these assumptions can lead to questionable results. These assumptions are called Gauss–Markov assumptions.²³

We will describe each of the Gauss–Markov assumptions in detail in sections 9.3.1, 9.3.2, and 9.3.3, respectively. To illustrate the importance of the underlying assumptions, open the data file `anscombe.dta` and fit the following regression models:²⁴

```
. use anscombe, clear
. regress y1 x1
. regress y2 x2
. regress y3 x3
. regress y4 x4
```

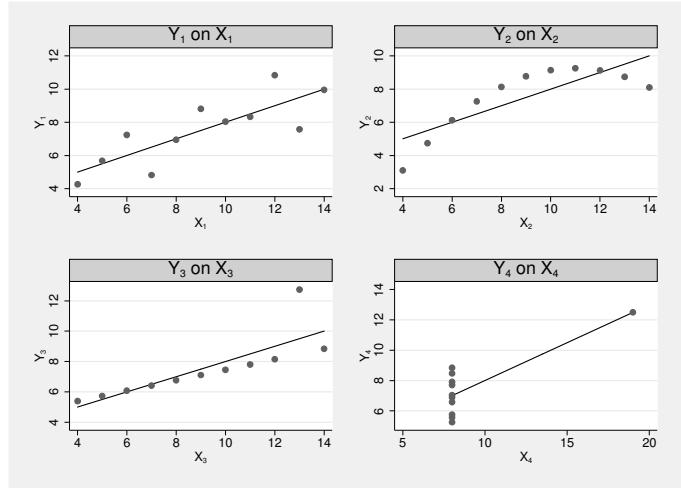
The estimated results for each regression model are the estimated coefficients, the variance of the residuals (RSS), and the explained variance R^2 . The results of these four models are nearly identical. For each of them, the R^2 rounds to 0.67. Further, the constant (or intercept) is 3, and the slope of the regression line is 0.5. If you did not know about the regression assumptions or regression diagnostics, you would probably stop your analysis at this point, supposing that you had a good fit for all models.

Now draw a scatterplot for each of these variable combinations, and then consider which model convinces you and which one does not; you can do this by typing the commands `scatter y1 x1`, `scatter y2 x2`, etc., one after the other. We actually used `granscomb1.do` to produce the graphs.

23. If you are already familiar with the Gauss–Markov assumptions and how to check them, you might want to get a quick overview of regression diagnostics within Stata by typing `help regress postestimation`.

24. The data file was created by Anscombe (1973).

The scatterplots in figure 9.3 show, without a doubt, that there is good reason to be cautious in interpreting regression results. Looking at just the R^2 or the estimated coefficients can be misleading!



Source: granscomb1.do

Figure 9.3. The Anscombe quartet

Now we want to show you how to check the Gauss–Markov conditions and correct any violations of them. Most of the diagnostic techniques we present are graphical, so you will need to understand the basics of the Stata `graph` command (see chapter 6). For an overview of various graphical diagnostic techniques, see Cook and Weisberg (1994). See Berk (2004, chap. 9) for a discussion on the limitations and potential hazards of using regression diagnostics.

9.3.1 Violation of $E(\epsilon_i) = 0$

The OLS estimation of the parameters of a regression model is based on the assumption that the expected value of the error terms in (9.1) and (9.6) is equal to 0, or formally: $E(\epsilon_i) = 0$.

To understand this assumption, you must first understand the meaning of an *expected value*. Consider a situation in which you measure the size of the Lopez's home from (9.1) over and over again. Sure your measures will be fairly similar in each replication, but not quite identical. At the end of the day, it might be sensible for you to calculate the average across all repeated measures. This average value over an infinite number of hypothetical replications of an experiment is called the expected value; in our example, it would be the expected value of home size.

In $E(\epsilon_i) = 0$, we deal with the expected value of the error term of our regression model. As we said before, the error term comprises all factors that influence the values of the dependent variable beyond the influence of the independent variables on our regression model. Hence, $E(\epsilon_i) = 0$ means that the average influence of all of these factors is 0 when we apply the model over and over again. Or, all influences that are not part of the model cancel out each other out in the long run.

If you estimate the parameters of a model by making such a simplifying assumption, you might ask yourself what happens when the assumption fails. The plain and simple answer is that your estimates of the regression coefficients will be biased. It is therefore important to verify that the expected value of the error term is indeed 0. All the problems that showed up in the Anscombe quartet are due to violations of this assumption.

To avoid biased estimates of the regression coefficients, you should always check the underlying assumptions. Checking $E(\epsilon_i) = 0$ is particularly important because its violation leads to biased estimators. The assumption will be violated if

- the relationship between the dependent and independent variables is nonlinear,
- some outliers have a strong effect on the estimated regression coefficients, or
- some influential factors have been omitted that are in fact correlated with the included independent variables.

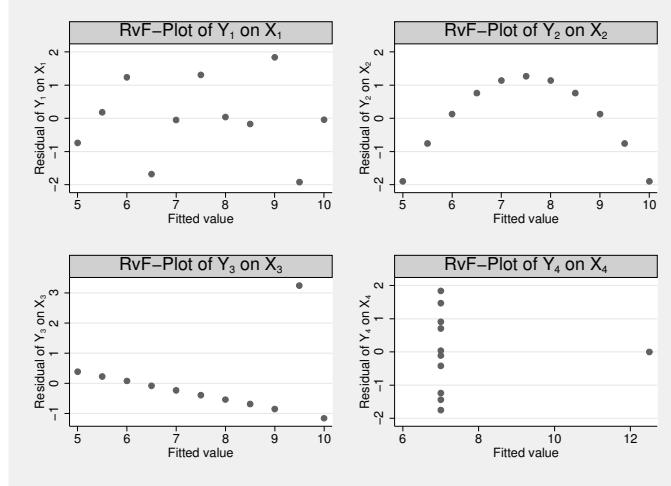
There are special techniques for testing each of the problems named above. You can see all three possible causes using a residual-versus-fitted plot, which is a scatterplot of the residuals of a linear regression against the predicted values. For the regression fit last, you could build the plot by typing

```
. regress y4 x4
. predict yhat
. predict resid, resid
. scatter resid yhat
```

or more simply by using the `rvfplot` command, which generates one of the specialized statistical graphs mentioned in section 6.2.

```
. rvfplot
```

With `rvfplot`, you can use all the graphic options that are available for scatterplots. Figure 9.4 shows the residual-versus-fitted plots for all regressions in the Anscombe example.



Source: `granscomb2.do`

Figure 9.4. Residual-versus-fitted plots of the Anscombe quartet

In these graphs, the mean of the residuals is by definition always equal to 0. In a regression model, the regression coefficients are estimated so that the mean of the sample residuals is equal to 0. To fulfill the assumption that $E(\epsilon_i) = 0$, not only must the overall mean of the residuals be 0 but also the mean of the residuals must be 0 *locally*, meaning the mean of the residuals is 0 for any slice of the x axis. This is true only for the first and the last regression model.

In a regression with only one independent variable, violations of the regression assumptions can be seen with a simple scatterplot of the dependent variable against the independent variable. The advantage of the residual-versus-fitted plot is that it also applies to regression models with more than just one independent variable.

In practice, a violation of $E(\epsilon_i) = 0$ is usually not as obvious as it is in the Anscombe data. Therefore, we will now introduce some special diagnostic tools for determining which of the three possibilities might be causing the violation of this assumption.

Linearity

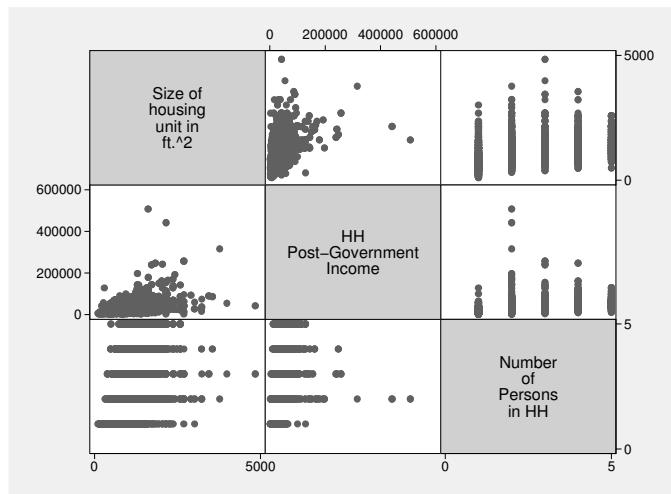
To understand the following examples, you might want to start with a regression of home size on household income and household size using the German Socio-Economic Panel (GSOEP) data:

```
. use data1, clear
. regress size hhinc hysize
```

One of the most important requirements for a linear regression is that the dependent variable can indeed be described as a linear function of the independent variables. To examine the functional form of the relationship, you should use nonparametric techniques, where you try to have as few prior assumptions as possible. A good example is a scatterplot reflecting only some general underlying assumptions derived from perception theory.

You can use a scatterplot matrix to look at the relationships between all variables of a regression model. Scatterplot matrices draw scatterplots between all variables of a specified variable list. Here is an example:

```
. graph matrix size hhinc hysize
```

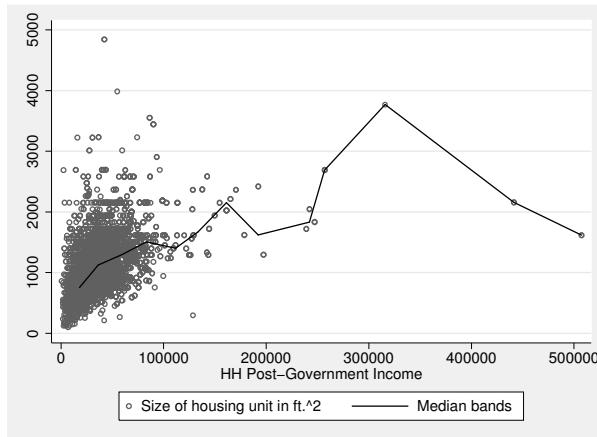


In each plot, the variable to the side of the graph is used as the Y variable, and the variable above or below the graph is used as the X variable. In the first line of the figure are scatterplots of home size against all the independent variables of the regression model.

However, scatterplots often show the functional form of a relationship only for small sample sizes. If you deal with larger sample sizes, you will need more information to improve the scatterplot. For this purpose, Stata allows you to overlay scatterplots with a scatterplot smoother (Fox 2000).

One example of a scatterplot smoother is the *median trace*. To construct a median trace, you divide the variable plotted on the x axis of a two-way plot into strips and calculate the median for each strip. Then the medians are connected with straight lines. In Stata, you get the median trace as plottype `mband` of two-way graphs. The `bands(k)` option of this plottype is used to decide the number of strips into which the x axis should be divided. The smaller the number of bands, the smoother the line.

```
. scatter size hhinc, ms(oh) || mband size hhinc, bands(20) clp(solid)
```



At first glance, the figure shows that there are many outliers on both variables, household income and home size. These observations might heavily influence the regression result, but we will deal with that in the next section. Concentrating on the area of household incomes below €100,000, we find a slight indication of curvilinearity. This is a first clue to a violation of the linearity assumption. However, you must be aware that the functional form of a relationship between two variables may change once we control for the influence of other variables in a multiple regression model.

One clue about the relationship between one independent variable (for example, household income) and the dependent variable (home size) if you control for other independent variables (such as household size) is given by plotting the residuals against the independent variables.²⁵ But plotting the residuals against one of the independent variables does not indicate the exact shape of any curvilinearity. For example, a U-shaped relationship and a logarithmic relationship might produce the same plot under certain circumstances (Berk and Both 1995).²⁶

25. When the sample size becomes large, it is reasonable to use a scatterplot smoother.

26. You must distinguish between these two kinds of relationships: if there is a U-shaped relationship, you must insert a quadratic term, whereas it might be sufficient to transform the dependent variable if there is a logarithmic relationship (see section 9.4.3).

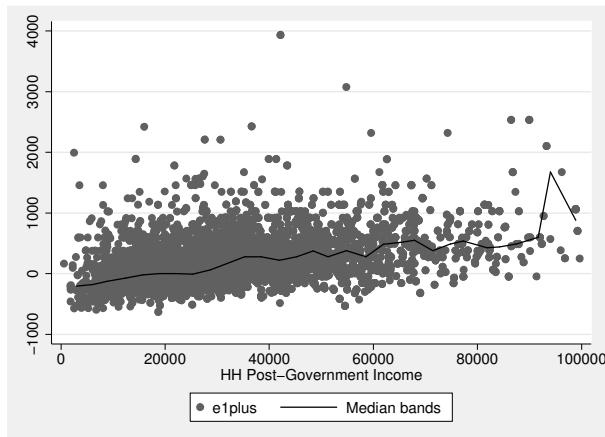
The component-plus-residual plots—also known as partial residual plots—are a modification of the plot just described: they allow the determination of the functional form of the relationship. Within the component-plus-residual plots, instead of using the residual, the product of the residual and the linear part of the independent variable are plotted against the other independent variables. What this means is shown in the following example.

To examine the potential nonlinearity between home size and household income in the multiple regression model of home size on household income and household size, first save the residuals as `e1`:

```
. predict e1, resid
```

Then you can add the linear part of household income to the saved residuals and plot the resulting number against household income. Because we do not want to deal with the extreme values of household income here, we just use the observations below household incomes of €100,000:

```
. generate e1plus = e1 + _b[hhinc]*hhinc
. scatter e1plus hhinc || mband e1plus hhinc, bands(30) || if hhinc < 100000
```



We infer from the graph that home size increases linearly with household income.

You would end up with the same result if you used the command `cprplot` after you fit your model using `regress`; this postestimation command will run the same procedure for any independent variable of your choice.²⁷ After `cprplot`, you enter the name of the independent variable for which you want to create the variable.

```
. regress size hhinc hhsiz
. cprplot hhinc, msopts(bands(20))
```

²⁷. You will also find the augmented component-plus-residual plot from Mallows (1986): `acprplot`. Also, instead of the median trace used here, you could use the locally weighted scatterplot smoother (LOWESS) (Cleveland 1994, 168). Then you would use the option `lowess`.

The command `cprplot` is very convenient. However, it does not allow an `if` qualifier. It is therefore good to know how to create the plot “by hand”.

Potential solutions

In our example, the relationships seem to be linear. In the presence of nonlinear relationships, you need to transform the independent variables involved or include more quadratic terms in the equation; see section 9.4.3.

Influential cases

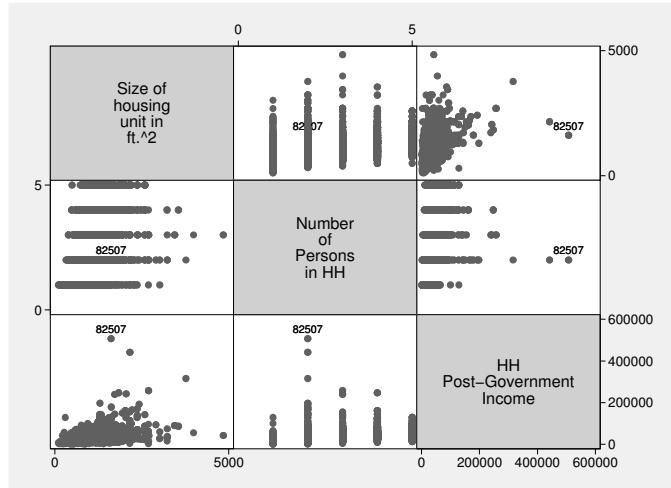
Influential cases are observations that heavily influence the results of a regression model. Mostly, these are observations that have unusual combinations of the regression variables included in the model (multivariate outliers). As an example, think of a person with a huge income living in a very small home.

It may not be possible to detect multivariate outliers in a bivariate scatterplot. Observations that show up as outliers in one scatterplot might in fact turn out to be normal if you controlled for other variables.

If, for example, the person mentioned had been interviewed in a secondary residence, the small home size is less surprising. Thus it is often possible to *explain* multivariate outliers. Then the solution for this problem is to include a variable in the regression model that captures the explanation. Here you would have to include in the regression model a variable that indicates whether this is the primary or secondary residence.

You can find signs of influential cases by using a scatterplot matrix that is built from the variables included in the regression model. Because each data point of one of these scatterplots lies on the same row or column as that of the other scatterplot, you can locate conspicuous observations over the entire set of scatterplots (Cleveland 1993, 275). Our example illustrates this with the help of one observation, which we have highlighted.

```
. generate str label = string(hhnr2009) if hhinc == 507369
. graph matrix size hhsiz hhinc, mlab(label)
```



A more formal way to discover influential cases is to use DFBETAs. The computation of DFBETAs has a simple logic: First, you fit a regression model, and then fit it again with one observation deleted. Then you compare the two results. If there is a big difference in the estimated coefficients, the observation that was excluded in the second computation has a big influence on the coefficient estimates. You then repeat this technique for each observation to determine its influence on the estimated regression coefficients. You compute this for each of the k regression coefficients separately. More formally, the equation for computing the influence of the i th case on the estimation of the k th regression coefficient is

$$\text{DFBETA}_{ik} = \frac{b_k - b_{k(i)}}{s_{e(i)} / \sqrt{\text{RSS}_k}}$$

where b_k is the estimated coefficient of variable k , $b_{k(i)}$ is the corresponding coefficient without observation i , and $s_{e(i)}$ is the standard deviation of the residuals without observation i . The ratio in the denominator standardizes the difference so that the influences on the estimated coefficients are comparable (Hamilton 1992, 125).

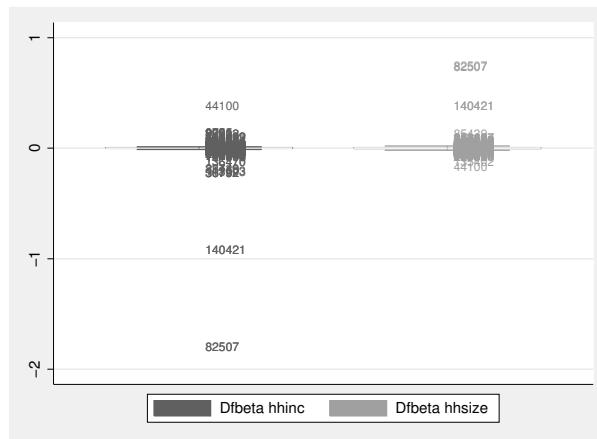
In Stata, you compute values for DFBETA_{ik} with the **dfbeta** command. You enter this command after the regression command, with a variable list in which you specify the coefficients for which you want to view the change. If you do not specify a variable list, all coefficients are used. The results of the command **dfbeta** are stored in variables whose names begin with **_dfbeta**.

Typing

```
. regress size hhinc hhsiz
. dfbeta
```

generates two variables: `_dfbeta_1` and `_dfbeta_2`. Both variables contain, for each observation, its influence on the estimated regression coefficient. If there are indeed influential cases in your dataset, you can detect them by using box plots with marker labels for the outliers:

```
. graph box _dfbeta*, marker(1, mlab(hhn2009) ms(i))
> marker(2, mlab(hhn2009) ms(i))
```



There are two data points that decrease the estimated coefficients for household income very much. We note that are no data points acting in the opposite direction. Moreover, there are also two data points that increase the estimated coefficients of household size, without having other data points balancing them out. Looking at the marker labels, we realize that the data points that decrease the coefficient of household income stem from the same households as those increasing the coefficient of household size. We consider observations from those households to be conspicuous.

More formally, values of $|DFBETA| > 2/\sqrt{n}$ are considered large (Belsley, Kuh, and Welsch 1980, 28).²⁸ In our model, several observations exceed this boundary value. With

```
. foreach var of varlist _dfbeta* {
.   list persnr `var' if (abs(`var') > 2/sqrt(e(N))) & !missing(`var')
. }
```

you obtain a list of these observations.²⁹

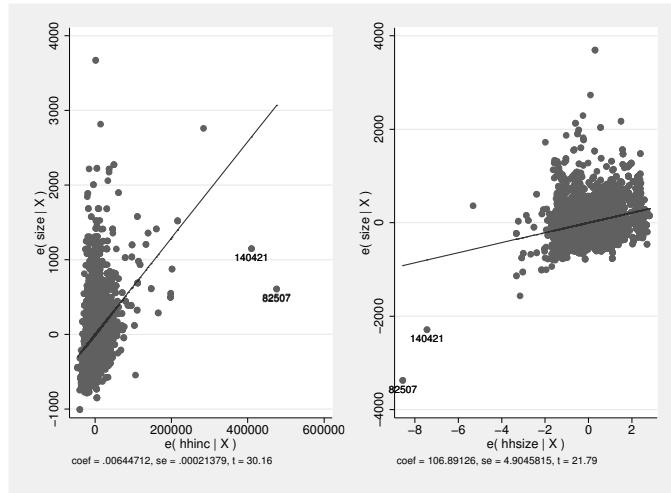
28. Other authors use 1 as the boundary value for DFBETA (Bollen and Jackman 1990, 267).

29. The command `foreach` is explained in section 3.2.2. The expression `abs()` is a general Stata function that returns the absolute value of the argument included in the parentheses (see section 3.1.6). Finally, `e(N)` is the number of observations included in the last regression model fit (see chapter 4).

Another way to detect outliers is to use the added-variable plot (partial regression plot). To create the added-variable plot of the variable X_1 , you first run a regression of Y on all independent variables except X_1 . Then you run a regression of X_1 on all remaining independent variables. You then save the residuals of both regressions and plot them against each other.³⁰

In Stata, you can also create added-variable plots by using the postestimation command `avplot` or `avplots`. `avplot` creates the added-variable plot for one explicitly named independent variable, whereas `avplots` shows all possible plots in one graph. In our example, we also highlight observations from the household that we considered to be suspicious above:

```
. generate suspicious = string(hhnr2009) if inlist(hhnr2009,82507,140421)
. avplots, mlab(suspicious) mlabpos(6)
```

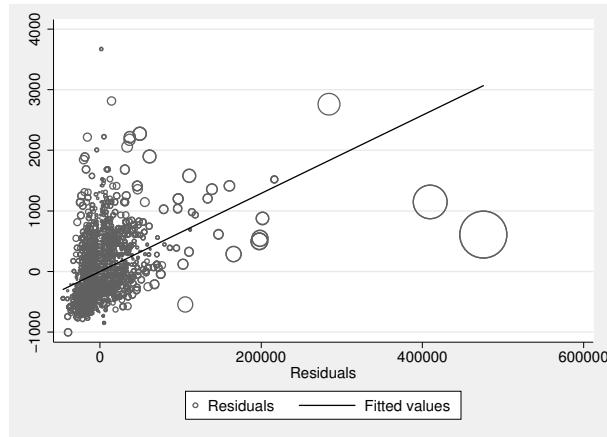


In these plots, points that are far from the regression line are *multivariate outliers*. These kinds of observations have more potential to influence the regression results. Here some observations are conspicuous in that household income is higher than you would assume by looking at the values of the remaining variables. In the plots shown here, we see that the observations from the conspicuous households cause concern yet again.

Hamilton (1992, 128–129, 141) recommends using an added-variable plot where the size of the plot symbol is proportional to DFBETA. To do this, you must create the plot yourself. In the multiple linear regression we used above, you would create such a plot for household income as follows:³¹

30. The logic behind added-variable plots corresponds to the way the b coefficients are interpreted in a multiple regression model (see section 9.2.3). A scatterplot of the residuals that were created there would be an added-variable plot.
31. For this example, we use the variable `_dfbeta_1`, which we created on page 288. The x axis of this graph stems from the variable label created automatically by the command `predict`. If you want to change these labels, see section 6.3.4.

```
. regress size hsize
. predict esize, resid
. regress hhinc hsize
. predict ehhinc, resid
. generate absDF = abs(_dfbeta_1)
. graph twoway || scatter esize ehhinc [weight = absDF], msymbol(oh)
> || lfit esize ehhinc, clp(solid)
```



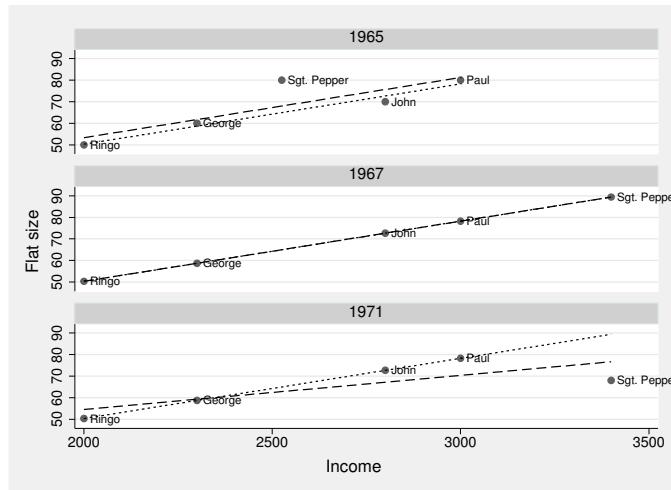
In Stata graphs, you can control the size of the plot symbol by using *weights*. Here it is not important what kind of weights (*fweights* or *aweights*, for example) you use. In this example, you must pay attention to possible negative values of DFBETA, so you can compute the absolute values of DFBETA first and use these values for weighting.³²

The previous figure shows that the multivariate outliers identified before have an appreciable influence on the regression line. More generally, we seem to find the influential cases mainly in the upper region of income, regardless of the other variables. Those few observations with high income have a disproportionately strong influence on the regression result.

So far, the impact of single observations has been examined separately for the different coefficients. If you have many independent variables, you will find it more complicated to interpret the many DFBETA values. With Cook's *D*, you have a statistic available that estimates the effect of one observation on all regression coefficients simultaneously (Fox 1991, 84) and hence the influence of one observation on the entire regression model.

³² You will find some general remarks about weights in section 3.3.

The idea behind Cook's D is that the influence of one observation on the regression model is composed of two aspects: the value of the dependent variable and the combination of independent variables. An influential case has an unusual value on Y and an unusual combination of values on the X s. Only if both aspects are present will the estimated coefficients be strongly affected by this observation. The graphs in figure 9.5 clarify this. The graphs present scatterplots of home size against the income of five Englishmen in 1965, 1967, and 1971.



Source: [grbeatles.do](#)

Figure 9.5. Scatterplots to picture leverage and discrepancy

In the first scatterplot, which shows the year 1965, Sgt. Pepper has an extraordinarily large home given his income. Sgt. Pepper's income is, however, anything but extraordinary: it is equal to the mean net income of the five Englishmen. We draw two regression lines in this picture. The dotted line is the regression line that results from a regression without Sgt. Pepper. When Sgt. Pepper is included in the regression, the regression line is shifted upward. There is no change in the slope of the line (the b coefficient of income).

In the scatterplot for 1967, Sgt. Pepper has an extraordinarily high income. The size of his home corresponds exactly, however, to the square footage we would expect from our model. Sgt. Pepper therefore has an extraordinarily large value of X but, given this value for X , a quite common Y value. The regression lines that result from the regressions with and without Sgt. Pepper are identical in this case.

In the scatterplot for 1971, Sgt. Pepper has an extraordinarily high income and, for this income, an extraordinarily small home. Here both aspects mentioned above are present. Accordingly, the regression line changes.³³

The idea that the effect of a certain point is determined by the extreme values of X and Y can be described mathematically as

$$\text{influence} = \text{leverage} \times \text{discrepancy} \quad (9.8)$$

where the leverage signifies how extraordinary the combination of the X values is (as in the second scatterplot) and the discrepancy signifies how extraordinary the Y value is (as in the first scatterplot). Because leverage and discrepancy are multiplied, the influence of any given observation is equal to 0 if one or both aspects are 0.

To compute the influence as shown in (9.8), you need some measures of the leverage and the discrepancy. First, look at a regression model with only one independent variable. Here the leverage of a specific observation increases with its distance from the mean of the independent variable. Therefore, a measure of the leverage would be the ratio of that distance to the sum of the distances of all observations.³⁴

When there are several independent variables, the distance between any given observation and the centroid of the independent variables is used, controlling for the correlation and variance structure of the independent variables (see also Fox [1997, 97]). In Stata, you obtain the leverage value for every observation by using the `predict lev, leverage` command after the corresponding regression. When you type that command, Stata saves the leverage value of every observation in a variable called `lev`.

To measure the discrepancy, it seems at first obvious that you should use the residuals of the regression model. But this is not in fact reasonable. Points with a high leverage pull the regression line in their direction, and therefore they may have small residuals. If you used residuals as a measure of discrepancy in (9.8), you might compute small values for the influence of an observation, although the observation changed the regression results markedly.³⁵

Hence, to determine the discrepancy, you need a statistic that is adjusted for the leverage. The standardized residual e'_i is such a statistic. You can obtain the values of the standardized residuals by using the `predict varname, rstandard` command, which you can enter after a regression.³⁶

33. Think of the regression line as a seesaw, with the support at the mean of the independent variable. Points that are far away from the support and the regression line are the most influential.

34. Specifically,

$$h_i = \frac{1}{n} + \frac{(x_i - \bar{x})^2}{\sum_{j=1}^n (x_j - \bar{x})^2}$$

35. This can be demonstrated with the fourth graph in the Anscombe quartet (page 280). If you computed the influence of this outlier with (9.8) and thereby used the residuals as a statistic for discrepancy, the influence of this outlier would be equal to 0.

36. You can choose any variable name for `varname`.

After finding a statistic for both discrepancy and leverage, you can multiply the two statistics together in accordance with (9.8). But you should provide an appropriate weight for which to multiply each value. We leave that task to the statisticians. Cook (1977) suggested the following computation:

$$D_i = \underbrace{\frac{h_i}{1 - h_i}}_{\text{leverage}} \times \underbrace{\frac{e_i'^2}{k + 1}}_{\text{discrepancy}}$$

Here e_i' is the standardized residual and h_i is the leverage of the i th observation.³⁷ Values of Cook's D that are higher than 1 or $4/n$ are considered large. Schnell (1994, 225) recommends using a graph to determine influential cases. In this graph, the value of Cook's D for each observation is plotted against its serial number within the dataset, and the threshold is marked by a horizontal line.

To construct this graph, you must first compute the values for Cook's D after the corresponding regression. You do this with the option `cooksdi` of the postestimation command `predict`:

```
. regress size hysize hhinc
. predict cooksdi, cooksdi
```

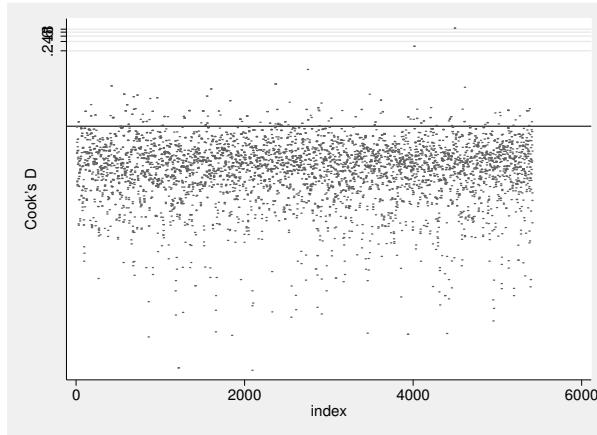
Then you save the threshold in a local macro (`max`) using the number of observations in the last regression model, which is stored by Stata as an internal result in `e(N)` (see chapter 4 and section 12.2.1):

```
. local max = 4/e(N)
```

37. There is a useful teaching tool you can see by typing the command `regpt`, which is taken from an ado-file programmed by the Academic Technology Services of the University of California–Los Angeles. To learn more about ado-files, see chapter 12; to learn more about ado-files provided over the Internet, see chapter 13.

Now you build a variable `index`, which contains the serial observation number, and use this variable as the x axis on our graph. Next construct the graph with a logarithmic y axis:

```
. generate index = _n
. scatter cooksd index, yline(`max') msymbol(p) yscale(log)
```



The figure shows many observations that are above the critical value, especially those with a comparatively high income:

```
. generate bigcook = cooksd > `max'
. tabulate bigcook, summarize(hhinc)

```

bigcook	Summary of HH Post-Government Income		
	Mean	Std. Dev.	Freq.
0	35570.493	20567.621	5195
1	75854.632	79519.082	212
Total	37149.971	26727.972	5407

In summary, the analyses you ran in this section show a clear finding: using these diagnostic techniques, you found observations from two households with high incomes to be conspicuous. The results of the model are much more strongly affected by these observations than by all the other observations with low, medium, or high (but not very high) income. Read on to learn what you might do with those high influential data points.

Potential solutions

You may wonder what to do when influential observations are present. If an influential case can be attributed unquestionably to a measurement error, you should either correct the error or delete the observation from the file. If influential observations result from extreme values of the dependent variable, it is reasonable to use median regression (section 9.6.1).

Almost always, however, influential observations result from an incompletely specified model. Exceptional cases are in this case exceptional only because our theory explains them insufficiently. As in our example, where observations with a high income influence the regression extraordinarily, you should ask if another factor that is typically related to high (or to low) income influences home size. With *right-skewed* distributions, such as that of income, you may want to change the model to use the logarithm of household income instead of household income itself. In the current context, this means that household income is supposed to be in a logarithmic relationship to home size: the higher the household income gets, the smaller the change in home size with each additional Euro of household income.

If none of those solutions fits your needs, you may want to drop the highly influential observations from your dataset and run the regression again. If you do so, however, you have to report both results, the one with and the one without the highly influential cases. We would not trust any substantial statement that is visible in just one of these results.

Omitted variables

Variables are called *omitted variables* or *omitted factors* if they influence the dependent variable and are at the same time correlated with one or more of the independent variables of the regression model. Strictly speaking, nonlinear relationships and influential cases are omitted factors, too. In the first case, you may have overlooked the fact that an independent variable does not have the same influence on the dependent variable throughout the range of the dependent variable. In the second case, you may have neglected to model your theory adequately or overlooked a mechanism that would explain the outliers.

To figure out which variables have been omitted, you can begin by graphing the residuals against all variables that are not included in the model. But this is obviously possible only for those variables that are included in the data file. Even if these graphs show no distinctive features, there still may be a problem. This diagnostic tool is therefore necessary but not sufficient.

Identifying omitted factors is, first of all, a theoretical problem. Thus we warn against blindly using tools to identify omitted variables.

Multicollinearity

In trying to include all important influential factors in the model, there is another risk called *multicollinearity*. We will introduce an extreme case of multicollinearity in section 9.4.1 when we discuss how to include categorical independent variables in regression models. If there is a perfect linear relationship between two variables of the regression model,³⁸ Stata will exclude one of them when calculating the model.

But even if the two variables are not a perfect linear combination of each other, some problems can arise: the standard errors of the estimated coefficients might increase, and there might be an unexpected change in the size of the estimated coefficients or their signs. You should therefore avoid including variables in the regression model haphazardly. If your model fits the data well based on R^2 but nevertheless has a few significant estimated coefficients, then multicollinearity may be a problem.

Finally, you can use the `estat vif` command to detect multicollinearity after regression. This command gives you what is called a *variance inflation factor* for each independent variable. See Fox (1997, 338) for an interpretation and explanation of this tool.

9.3.2 Violation of $\text{Var}(\epsilon_i) = \sigma^2$

The assumption that $\text{Var}(\epsilon_i) = \sigma^2$ requires that the variance of the errors be the same for all values of the independent variables. This assumption is called homoskedasticity, and its violation is called heteroskedasticity. Unlike the violation of $E(\epsilon_i) = 0$, heteroskedasticity does not lead to biased estimates. But when the homoskedasticity assumption is violated, the estimated coefficients of a regression model are not efficient. With inefficient estimation, there is an increasing probability that a particular estimated regression coefficient deviates from the true value for the population. That is, heteroskedasticity causes the standard errors of the coefficients to be incorrect, and that obviously has an impact on any statistical inference that you perform.

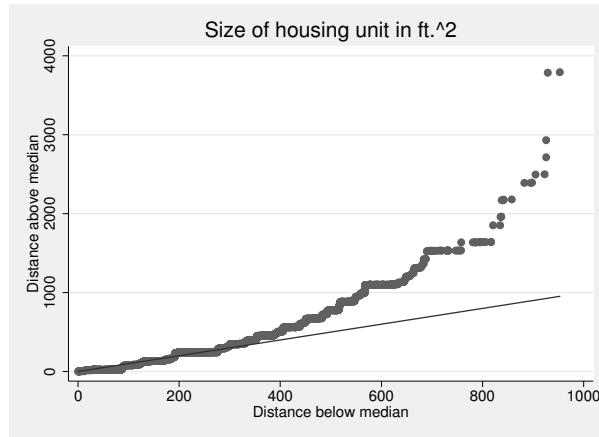
There are many possible reasons for heteroskedasticity. Frequently, you find heteroskedasticity if the dependent variable of your regression model is not symmetric. To test the symmetry of variables, you will find the graphical techniques described in section 7.3.3 to be very useful.

Stata has a special technique for checking the symmetry of a distribution, called a *symmetry plot* (Chambers et al. 1983, 29). To construct a symmetry plot, you first determine the median. Then you compute the distances between the observations next in size and the median. In a symmetry plot, you plot these two quantities against each other. You do the same with the next observation, and so on. If all distances are the same, the plot symbols will lie on the diagonal. If the distances of the observations above the median are larger than those below, the distribution is right-skewed. If the reverse is true, the distribution is left-skewed.

³⁸ For example, $x_1 = 2 + x_2$.

In Stata, the `sympplot` command graphs a symmetry plot of a given variable. Here we graph the symmetry plot for home size:

```
. sympplot size
```



The figure shows an obviously right-skewed distribution of the home size variable. With this kind of distribution, there is risk of violating the homoskedasticity assumption.

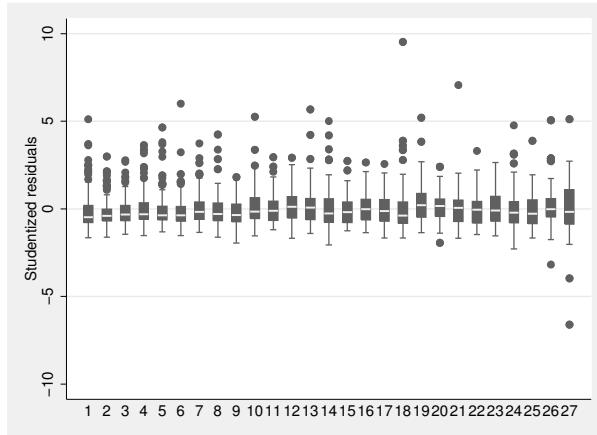
The residual-versus-fitted plot (Cleveland 1994, 126) is the standard technique for examining the homoskedasticity assumption. We want to introduce one variation of this plot that emphasizes the variance of the residuals. You therefore divide the x axis into k groups with the same number of observations and then draw a box plot of the Studentized residuals for each group.

To do this, you again run the regression model to get the predicted values and the Studentized residuals:

```
. regress size hhinc hsize
. predict yhat
. predict rstud, rstud
```

For this example, we chose the number of groups used for the x axis so that each box plot contains roughly 200 observations. In the graph, you can see that there is a slight increase in the variance of residuals.

```
. local groups = round(e(N)/200,1)
. xtile groups = yhat, nq(`groups')
. graph box rstud, over(groups)
```



Potential solutions

Often you can transform the dependent variable to remove heteroskedasticity. The transformation should end in a symmetric variable. For right-skewed variables, a logarithmic transformation is often sufficient. Also the `boxcox` command allows you to transform a variable so that it is as symmetric as possible. You will find more discussion of the Box–Cox transformation in section 9.4.3.

If transforming the dependent variable does not remove heteroskedasticity in the regression model, you cannot use the standard errors of the estimated coefficients (because they are given in the regression output) for a significance test. If you are nevertheless interested in a significance test, you might want to try the `vce(robust)` option in the regression command. When you use this option, the standard errors are computed so that homoskedasticity of the error terms need not be assumed. When you use the `svy` prefix to analyze data from a complex sample (as discussed in section 8.2.2), this implies robust estimates of the standard errors that do not assume homoskedasticity; so this assumption is not required in such situations.

9.3.3 Violation of $\text{Cov}(\epsilon_i, \epsilon_j) = 0, i \neq j$

What $\text{Cov}(\epsilon_i, \epsilon_j) = 0, i \neq j$, means is that the errors are not correlated. The violation of this assumption is often called *autocorrelation* or *correlated response variance*.

Typical violations of no correlation happen in the context of the complex samples discussed in section 8.2.2, when respondents from one primary sampling unit differ systematically from respondents in another. This can be further accentuated when there is just one interviewer per primary sampling unit. Consider, for example, that you have surveyed home size by letting the interviewers estimate the size instead of asking the respondent. Here it is reasonable to assume that some of the interviewers tend to overestimate the sizes of the homes, whereas others tend to underestimate them. All the observations from one interviewer should be similar in over- or underestimating home size. A similar situation occurs if all people in a household are interviewed. Here, as well as in the above, there may be factors within the unobserved influences (ϵ_i) that are the same for all members of a household.

Autocorrelation leads to inefficient estimation of the coefficients. This means that the standard errors shown in the output of regress are too small, that the confidence intervals are too narrow, and that conclusions of statistical significance may be declared inappropriately.

In most instances, the wrong standard errors can be corrected by using the `svy` prefix, as explained in section 8.2.2. Here are the results of the regression after applying the settings from page 220:

```
. generate sampleR = sample
. replace sampleR = 2 if sample==4
(150 real changes made)
. svyset psu [pweight=xweights], strata(sampleR)
    pweight: xweights
    VCE: linearized
Single unit: missing
Strata 1: sampleR
    SU 1: psu
    FPC 1: <zero>
```

. svy: regress size hhinc hysize (running regress on estimation sample)		Survey: Linear regression			
		Number of strata = 4	Number of obs = 5407		
		Number of PSUs = 515	Population size = 80645043		
		Design df = 511	F(2, 510) = 163.75		
		Prob > F = 0.0000	R-squared = 0.2761		

size	Linearized					
	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
hhinc	.0062389	.0013269	4.70	0.000	.0036321	.0088457
hysize	99.84128	14.02799	7.12	0.000	72.28165	127.4009
_cons	586.8209	27.86148	21.06	0.000	532.0838	641.558

If you compare these results with those computed without the `svy` prefix (page 277), you will find that the confidence intervals of all coefficients have become much larger now. This is often the case in complex surveys. You should therefore normally not trust the inference statistics without respecting the complex nature of the sample.

Autocorrelation is also a key concept in time-series analysis, because successive observations tend to be more similar than observations separated by a large time span (serial autocorrelation). The Durbin–Watson test statistic has been developed for time-series analysis; in Stata, it is available using the `estat dwatson` command after regression. However, you must define the data as a time series beforehand.³⁹

9.4 Model extensions

Here we will introduce three extensions to the linear model you have seen so far. These extensions are used for categorical independent variables, interaction terms, and modeling curvilinear relationships.

39. Because we do not discuss time-series analysis in this book, we refer you here to the online `help tsset` and to the manual entry [U] **26.14 Models with time-series data**.

9.4.1 Categorical independent variables

Be careful when including a categorical variable with more than two categories in the regression model. Take, for example, marital status. The variable `mar` has five categories, namely, married, single, widowed, divorced, and separated:

Marital Status of Individual	Freq.	Percent	Cum.
Married	3,080	56.93	56.93
Single	1,394	25.77	82.70
Widowed	400	7.39	90.09
Divorced	449	8.30	98.39
Separated	87	1.61	100.00
Total	5,410	100.00	

It would not make sense to include marital status in the same way as all other independent variables; that would be like assuming that going from being married to single has the same effect on home size as going from widowed to divorced. However, you would assume this implicitly when a categorical variable with several categories is included in a regression model without any changes. What you need instead are contrasts between the individual categories.

Let us say that you want to include a variable that differentiates between married and unmarried respondents. To do so, you can create a dichotomous variable with the response categories 0 for not married and 1 for married.

```
. generate married = mar == 1 if !missing(mar)
```

You can interpret the resulting b coefficient for this variable just as for other dummy variables; accordingly, you could say that married respondents live on average in a space that is b square feet bigger than the space where unmarried people live. All other contrasts can be built in the same way:

```
. generate single = mar == 2 if !missing(mar)
. generate widowed = mar == 3 if !missing(mar)
. generate divorced = mar == 4 if !missing(mar)
. generate separated = mar == 5 if !missing(mar)
```

Each contrast displays the difference between respondents with one particular marital status and all other respondents. In the following regression, we use the contrasts as independent variables; however, we leave out one of them—**widowed**:

. regress size hhinc hhsiz married single divorced separated, noheader						
size	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
hhinc	.0063692	.000213	29.90	0.000	.0059515	.0067868
hhsiz	107.6239	5.178204	20.78	0.000	97.47256	117.7753
married	-61.45118	21.82667	-2.82	0.005	-104.2403	-18.6621
single	-166.3739	22.7343	-7.32	0.000	-210.9423	-121.8055
divorced	-159.533	26.5044	-6.02	0.000	-211.4923	-107.5737
separated	-153.5253	45.66446	-3.36	0.001	-243.0461	-64.00455
_cons	683.6544	20.35964	33.58	0.000	643.7413	723.5675

The reason for dropping one of the contrasts is that of the five new dummy variables, only four are needed to know the marital status of each person. If four dummies indicate a person is neither married, nor single, nor widowed, nor divorced, then the person must be separated from his or her spouse. The fifth dummy variable tells you nothing that cannot be gleaned from the other four because there are only five possibilities. Computationally, it is not even possible to estimate coefficients on all five dummies in addition to the constant term because those five dummies sum to 1 and hence are perfectly correlated with the constant term.

The category whose dummy variable is not included in the model is commonly termed the *reference category* or *base category*. For the interpretation of the model, it is very important to know which is the base category. To understand this, remember that the constant represents the predicted value for respondents with 0 on all covariates. In our case, these are respondents who are widowed (and have no household income and zero household size), because the widowed dummy was not included in the model. The predicted home size of persons with a different family status differs by the amount of the corresponding *b* coefficient. Therefore, married respondents have, on average, a home size that is 61.451 ft² smaller than those of respondents who are widowed. Even smaller are the homes of single respondents—about 166.374 ft² smaller than those of widowed respondents. All other estimated coefficients are interpreted accordingly.

The coefficients you estimated may be somewhat surprising. It would be more reasonable if married respondents have on average larger home sizes than those who are widowed, other things constant; however, according to the model, the opposite is true. Upon closer inspection, though, the results are more reasonable. Widowed respondents typically live alone, which means their household size is 1. Whereas married couples have an average home size 61.451 ft² less than widowed people, holding all other factors constant, the *hhsiz* variable's estimated coefficient of 107.624 implies that a married couple with no children has, in fact, a home that is an average of 46.173 ft² larger than a widowed person living alone.

Categorical variables such as *mar* are frequently used in regression models. Stata therefore provides two easy ways to deal with them, namely, the option *gen(newvar)* of

`tabulate` and the factor-variables notation ([U] 11.4.3 Factor variables). The former instructs Stata to automatically construct dummy variables for the categories shown in the table. Here we use this technique to create the variables `mar_1` to `mar_5`:

```
. tabulate mar, generate(mar_)
```

You can include those in your regression model. It is usually a good idea to decide on a contrast that will be left out and used as a comparison category, but which one you use does not affect the results substantively.

Factor-variable notation provides an extended way to specify variable lists and is allowed for most estimation and postestimation commands. Factor-variable notation can also be used to add categorical variables to regression models. The basic idea here is that you mark a variable as categorical by putting the operator `i.` in front of it. If you do this with an independent variable in a regression model, Stata includes this variable as a set of dummy variables in the model. It therefore creates virtual variables in the background, chooses the base category, and estimates the model using the set of virtual dummy variables. Here is an example with marital status (`mar`) and the social class according to (Erikson and Goldthorpe 1992) (`egp`).⁴⁰

```
. regress size hhinc hysize i.mar i.egp, noheader
```

size	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
hhinc	.0061854	.0002441	25.34	0.000	.0057068 .006664
hysize	113.7141	5.988777	18.99	0.000	101.9733 125.4548
mar					
2	-103.2019	14.60338	-7.07	0.000	-131.8312 -74.57252
3	58.4887	22.35011	2.62	0.009	14.67218 102.3052
4	-82.69869	20.54655	-4.02	0.000	-122.9794 -42.41798
5	-95.12313	42.51988	-2.24	0.025	-178.4817 -11.76454
egp					
2	-57.52079	24.52347	-2.35	0.019	-105.5981 -9.443481
3	-67.02651	29.92765	-2.24	0.025	-125.6985 -8.354513
4	-25.83427	28.39558	-0.91	0.363	-81.5027 29.83415
5	31.1338	32.66333	0.95	0.341	-32.9014 95.169
8	-81.56398	26.90431	-3.03	0.002	-134.3088 -28.81912
9	-105.6007	25.78623	-4.10	0.000	-156.1536 -55.04781
15	-171.8781	30.35601	-5.66	0.000	-231.3899 -112.3663
18	-43.00595	24.4626	-1.76	0.079	-90.96394 4.95203
_cons	668.938	28.76018	23.26	0.000	612.5547 725.3212

40. The categories of `egp` are not numbered consecutively because some of them had to be combined for reasons of data confidentiality.

A number of possibilities exist that let you fine-tune the way factor variables are handled in regression models. To start with, the operator `ib.` lets you control the base category. If you type

```
. regress size hhinc hhsiz ib4.mar ib(#3).egp
```

Stata will choose the category with the value 4 as base category for `mar` and the category with the third highest value for `egp`. Other possibilities are `ib(first)` for the smallest value, `ib(last)` for the largest value, and `ib(freq)` for the most frequent value.

It is also possible to select a subset of categories for the inclusion into the model. This is done by directly placing a number or a *numlist* behind the `i.` operator. With

```
. regress size hhinc hhsiz i2.mar i(1/5).egp
```

there will be a virtual dummy variable for single versus anything else in the regression model, and there will be four virtual dummies for the categories 2, 3, 4, and 5 of `egp` versus anything else.

Finally, you can apply factor-variable operators to groups of variables by putting variable names in parentheses. This is useful if you have several categorical variables in your regression model. We provide an example for `mar` and `egp`:

```
. regress size hhinc hhsiz i.(mar egp)
```

9.4.2 Interaction terms

Interaction terms are used to investigate whether the influence of one independent variable differs systematically with the value of another independent variable. To discuss interaction terms, we return to our analysis of the gender–wage gap from chapter 1. There we tried to explain personal income by gender and employment status for all respondents who are not unemployed. Specifically, we fit the following model:

```
. use data1, clear
. generate men = sex==1
. replace emp = . if emp==5
. replace income = . if income == 0
. regress income men i.emp
```

The analysis showed that women earn less on average than men and that this difference can be only partly explained by the differences in full-time, part-time, or irregular occupation between men and women.

Assume that you hypothesize that income depends not only on the respondents' gender and occupational status but also on their age. Here you should include age in your regression model. Say that you also assume that the gender–wage gap is stronger the older people are. There are at least two good reasons for this assumption:

- Women interrupt their occupational careers for child care more frequently than men. Those interruptions are accompanied with difficulties returning to the labor market at an adequate level. As a result of the corresponding mismatch between women's qualifications and the demands of their jobs, women gradually fall behind during a lifetime. The gender–wage gap should be therefore particularly strong at the end of careers.
- In the past, Germany's families were dominated by the male breadwinner model, which led women to invest less in education and occupational careers. However, this role model has changed. Modern women have the same or even better education than men, and they are far less ready to give up their careers for child care. It is therefore sensible that the gender–wage gap is particularly strong among the older women who grow up during the domination of the male breadwinner model.

Both arguments could lead us to hypothesize that the effect of gender on income increases with age. Such effects that vary with values of a third variable are called *interaction effects*, which you can include in a regression model by multiplying the relevant variables or by using factor-variable notation. We start by illustrating the hard way of creating interactions (that is, multiplying the relevant variables).

In our example, gender and age are the relevant variables for the interaction effect. To extend the regression model, you first need to create a variable for age using year of birth (*ybirth*) and the date of the interview:

```
. generate age = 1997 - ybirth
```

It is advantageous to center continuous variables, such as length of education or age, before they are included in a regression model, for several reasons. This is especially true in the presence of interaction terms (Aiken and West 1991).

To center a variable, subtract the mean from each value. However, you need to compute the mean only for those respondents who will later be included in the regression model. To do this, we use the function `missing()` with more than one argument. The function then returns 1 if at least one of the arguments inside the parentheses is missing and returns 0 otherwise:

```
. summarize age if !missing(income,emp,men,age)
. generate age_c = age - r(mean)
```

To build the interaction term, multiply both variables that are part of the interaction:

```
. generate menage = men * age_c
```

You can now extend the linear regression model to include `men` and `age_c` as well as the interaction term, `menage`, that you just created:

<code>. regress income i.emp men age_c menage</code>									
Source	SS	df	MS	Number of obs = 2886 F(5, 2880) = 78.38 Prob > F = 0.0000 R-squared = 0.1198 Adj R-squared = 0.1182 Root MSE = 39127					
Model	5.9995e+11	5	1.1999e+11						
Residual	4.4090e+12	2880	1.5309e+09						
Total	5.0089e+12	2885	1.7362e+09						
income	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]				
emp									
2	-16120.07	2076.971	-7.76	0.000	-20192.57	15424.06	-12047.57		
4	-29188.95	2669.66	-10.93	0.000	-34423.58	374.5461	-23954.31		
men	12151.78	1668.855	7.28	0.000	8879.512	667.9356			
age_c	190.1975	94.01761	2.02	0.043	5.848915				
menage	412.3474	130.3497	3.16	0.002	156.7592				
_cons	31363.58	1408.624	22.27	0.000	28601.57		34125.6		

Let us interpret each of the fitted coefficients of this model. We start with the simple ones. The constant, as always, shows the predicted value for those observations that have 0 values on all independent variables in the model: the estimated average income of full-time employed women of average age is €31,364. The interpretation of the coefficients for categories 2 and 4 of employment status is also straightforward: part-time employed respondents have, on average, €16,120 less income than the full-time employed respondents, and those who are irregularly employed earn €29,189 less than the full-time employed.

Now on the interaction terms. For ease of interpretation, we focus on full-time employed persons. That is, we set all empty dummy variables to 0 and get the following formula for calculating the predicted values from the above regression model:

$$\hat{y}_i = 31364 + 12152 \times \text{men} + 190 \times \text{age_c} + 412 \times \text{men} \times \text{age_c}$$

where the numbers correspond to the respective estimated regression coefficients. Let us use this formula to calculate the predicted values for women with an average age. Bypassing rounding of estimated coefficients, we can do this with Stata:

```
. display _b[_cons] + _b[men]*0 + _b[age_c]*0 + _b[menage]*0*0
31363.583
```

Because the values for both variables, `men` and `age`, are 0 for women with average age, all coefficients except the constant drop out of the equation. Hence, the predicted value is the constant, that is, €31,364.

Now let's choose male respondents of average age:

```
. display _b[_cons] + _b[men]*1 + _b[age_c]*0 + _b[menage]*1*0
43515.367
```

Note that neither the coefficient for centered age nor the coefficient for the interaction term contributes to the result. Compared with the last result, the predicted value is only increased by the coefficient for `men`, which is one of the two constituent effects of the interaction. The interpretation of this constituent effect is this: At average age, the average earnings of men are around €12,152 higher than the average earnings of women.

We now turn to the second constituent effect of the interaction. Calculate the predicted values for female respondents who are one year older than the average age:

```
. display _b[_cons] + _b[men]*0 + _b[age_c]*1 + _b[menage]*0*1
31553.781
```

Here the coefficient for `men` and for the interaction term is dropped from the equation. Compared with females at average age, the predicted value of females that are one year older is increased by the coefficient of the constituent effect of age. Hence, for females, an increase in age of one year corresponds to an increase in average income of around €190.

Finally, let us consider male respondents who are one year older than the average:

```
. display _b[_cons] + _b[men]*1 + _b[age_c]*1 + _b[menage]*1*1
44117.912
```

This time, all coefficients are being used. If we compare the predicted values of male respondents of average age with those one year older, we see that they differ by the amount of the constituent effect of age *and* by the amount of the interaction term. Therefore, the effect of age on income is on average €412 stronger for men than for women, or stated differently, male income increases faster with age than does female income.

There is also a second interpretation of the interaction term. To see this, compare the predicted values of female respondents who are one year older than the average with male respondents of the same age. The two values differ by the amount of the constituent effect for `men` and by the interaction term. Therefore, the effect of gender is on average €412 stronger when we compare men and women one year above the average age than when we compare men and women of average age. The same happens with each increase of age by one unit. Hence, the gender–wage gap gets stronger the older the respondents are.

We wrote above that interaction terms could be specified by generating variables that are the product of existing variables or by using factor-variable notation. For the remainder of this section, we show some examples of factor-variable notation.

With factor-variable notation, if you place `#` between two variable names, it means that the two variables are interacted, whereby both variables are considered to be categorical. That is to say, the command

```
. regress income i.sex i.emp sex#emp
```

specifies a regression model that includes one virtual dummy variable for the two categories of `sex`, two virtual dummy variables for the three categories of `emp`, and two virtual product variables of each product of the gender category with the two employment categories.

You can further simplify the command by using `##` instead of `#`. With two `#` operators, we specify the interaction effects and the virtual dummy variable for the constituent variables. Therefore, the command above can be rewritten as

```
. regress income sex##emp
```

By default, variables connected with `##` or `#` are treated as categorical. Stata therefore always creates virtual dummy variables for each category of the variable and produces all pairwise products of these virtual dummies for the interaction. If an interaction involves a continuous variable, we must make this explicit by using the `c.` operator. Here we use that operator to specify the model that we set up the hard way above:

```
. regress income i.emp c.men##c.age_c
```

Note here that we have also prefixed the variable `men` with the `c.` operator. This is possible because `men` is a dummy variable with values 0 and 1. It does not matter whether you treat such variables as categorical or continuous; we prefer the latter because it makes a cleaner output.

Factor-variable notation makes it very easy to specify much more complicated models. To give you an impression, here is a model that includes the constituent effects and all two- and three-way interactions of the variables for gender, age, and years of education:

```
. regress income i.emp c.men##c.age##c.yedu
```

Here is a model with all two-way interactions between gender and each of the variables `emp`, `yedu`, and `age`:

```
. regress income c.men##(c.age c.yedu i.emp)
```

While factor-variable notation makes specification of complicated models real easy, interpretation of such models remains cumbersome. Conditional-effects plots are very helpful for such models. Section 9.5.3 shows examples of a conditional-effects plot for some of the models specified here.

9.4.3 Regression models using transformed variables

There are two main reasons to use transformed variables in a regression model:

- The presence of a nonlinear relationship
- A violation of the homoskedasticity assumption

Depending on the reason you want to transform the variables, there are different ways to proceed. In the presence of a nonlinear relationship, you would (normally) transform the *independent* variable, but in the presence of heteroskedasticity, you would transform the *dependent* variable. We will begin by explaining how to model nonlinear relationships and then how to deal with heteroskedasticity (see also Mosteller and Tukey [1977]).

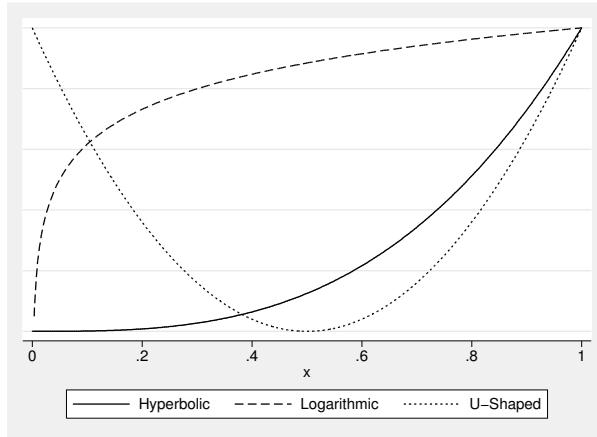
Nonlinear relationships

We introduced regression diagnostic techniques for detecting nonlinear relationships in section 9.3.1. Often, however, theoretical considerations already provide enough reason to model a nonlinear relationship: think about the correlation between female literacy rate and birth rate. You would expect a negative correlation for these two variables, and you would also expect the birth rate not to drop linearly toward 0. Rather, you would expect birth rate to decrease with an increase in literacy rate to levels of around one or two births per woman.

Nonlinear relationships occur quite often when income is used as an independent variable. For many relationships, income changes in the lower range of income have more impact on the dependent variable than income changes in the upper part of the income distribution. Income triples with a change from \$500 to \$1,500, whereas the increase is only 10% for a change from \$10,000 to \$11,000, although the dollar increase is in both cases \$1,000.

When modeling nonlinear relationships, you first need to know or at least hypothesize a functional form of the relationship. Here you need to distinguish among three basic types of nonlinear relationships: logarithmic, hyperbolic, and U-shaped. Stylized versions of these relationships can be produced with the two-way plottype function (see [G-2] **graph twoway function**). We thereby take advantage of locals (see sections 4.1 and 12.2.1) to make the idea of the command stand out more:

```
. local opt1 yaxis(1) yscale(off axis(1))
. local opt2 yaxis(2) yscale(off axis(2))
. local opt3 yaxis(3) yscale(off axis(3))
. graph twoway || function y = x^3, `opt1'
> || function y = ln(x), `opt2' || function y = (-1) * x + x^2, `opt3'
> || , legend(order(1 "Hyperbolic" 2 "Logarithmic" 3 "U-Shaped") rows(1))
```



In logarithmic relationships, the dependent variable increases with increasing values of the independent variable. However, with increasing values of the independent variable, the increase in the dependent variable levels off. In hyperbolic relationships, the relationship is reversed, because the dependent variable increases only moderately at the beginning and increases with increasing values of the independent variable. In U-shaped relationships, the sign of the effect of the independent variable changes.

All three basic types can occur in opposite directions. For logarithmic relationships, this would mean that the values decrease rapidly at the beginning and more slowly later on. For hyperbolic relationships, the values drop slowly at the beginning and rapidly later on. For U-shaped relationships, the values first decrease and increase later on, or vice versa. In practice, logarithmic relationships occur often.

To model logarithmic relationships, you first form the log of the independent variable and replace the original variable in the regression model with this new variable. A strong logarithmic relationship can be found between each country's gross domestic product and infant mortality rate. The file `who2009.dta` contains these data.⁴¹

```
. use who2009.dta, clear
. scatter mort_child gdp
```

41. The example was taken from Fox (2000) using updated data from the Global Health Observatory Data Repository of the World Health Organization (<http://apps.who.int/ghodata/#>).

You can model this logarithmic relationship by first creating the log of the X variable,

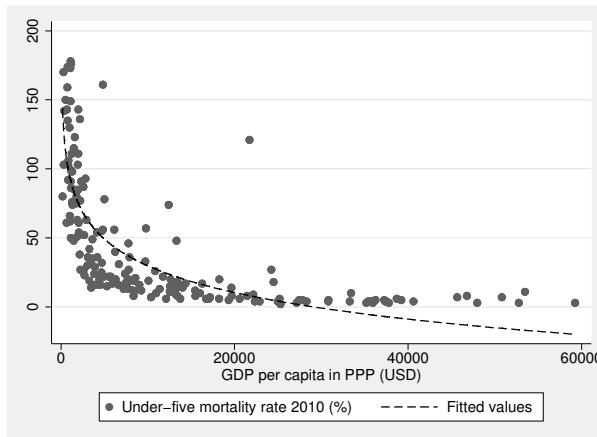
```
. generate loggdp = log(gdp)
```

and then using this variable instead of the original X variable:

```
. regress mort_child loggdp
. predict yhat
```

You can see the logarithmic relationship between the predicted value of the regression model $yhat$ and the untransformed independent variable:

```
. scatter mort_child gdp || line yhat gdp, sort
```

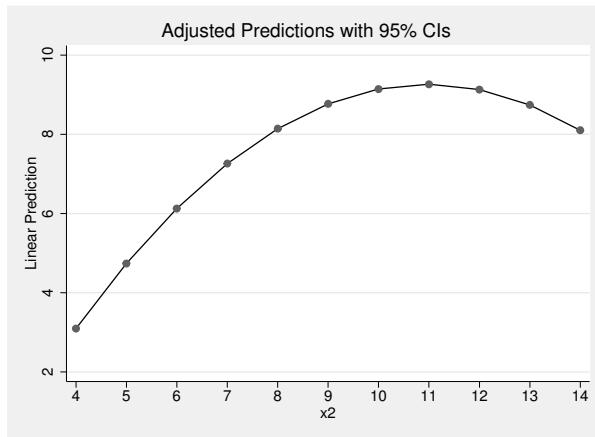


You use a similar procedure to model hyperbolic relationships except that now you square the original variable instead of taking its logarithm. Here the original variable is also replaced by the newly transformed variable.⁴²

The situation is different when you are modeling a U-shaped relationship. Although you still square the independent variable, the newly generated variable does not replace the original variable. Instead, both variables will be used in the regression model. A U-shaped relationship is one of the examples in *Anscombe's quartet* on page 280. Including the quadratic term will allow you to model this correlation perfectly:

42. Examples for hyperbolic relationships are rare in the social sciences. The salary of a Formula 1TM driver could possibly show a hyperbolic relationship to the number of Grand Prix victories.

```
. use anscombe, clear
. regress y2 c.x2##c.x2
. margins, at(x2=(4(1)14))
. marginsplot
```



Further discussion of quadratic terms can be found in section 9.5.3 If you are thinking of using transformations of the independent variables, see Cook and Weisberg (1999, chap. 16) for some precautions.

Eliminating heteroskedasticity

In section 9.3.2, we discussed skewed dependent variables as one of the possible causes of heteroskedasticity. Here you would need to transform the dependent variable to remove heteroskedasticity. The interpretation of the regression model changes when you include a transformed variable. Transforming the dependent variable leads to a nonlinear relationship between the dependent and *all* independent variables (Hair et al. 1995, 75).

The aim in transforming a variable is to obtain a fairly symmetric or normal dependent variable. Remember the following rule of thumb: If the distribution is wide, the inverse of the variable is a useful transformation ($1/Y$). If the distribution is skewed to the right (such as home size in our example), then taking the log is reasonable; you can take the square root if the distribution is skewed to the left (Fox 1997, 59–82).

Aside from following these rules, you can use the Stata command `bcskew0`, which uses a Box–Cox transformation that will lead to a (nearly) unskewed distribution.⁴³

```
. use data1, clear
. bcskew0 bcsize = size
```

43. Make sure that the variable used in `bcskew0` does not include negative values or the value 0.

The residual-versus-fitted plot (page 281) tells you something about the type of transformation necessary. If the spread of the residuals increases with increasing values in the predicted variable, the inverse of Y is a better dependent variable. If there is a decreasing spread of the residuals with increasing values in the predicted variable, you will want to replace the original dependent variable Y with the square root of Y (Hair et al. 1995, 70).

9.5 Reporting regression results

Sometimes, you need to report the results of your data analysis to a larger audience—in the form of an article in a journal, as a chapter in a book, on a webpage, or as a presentation at a conference. In all of these instances, you are not well advised if you show Stata output. Instead, you should find a device that makes your main results stand out.

In this section, we introduce three ways to report results of regression analysis: tables of similar regression models, plots of regression coefficients, and conditional-effects plots.

9.5.1 Tables of similar regression models

In scientific journals, regression models are almost always presented in tables that look like table 9.2. Such tables concatenate several regression models in one table. The various regression models are similar in that they all use the same dependent variable. Moreover, they are similar in the respect of the independent variables. Either they all apply the same set of independent variables but differ in the observations used or they are nested like our example in table 9.2. In both instances, the cells of the tables contain the estimated regression coefficient and some statistic to show the uncertainty of the results—for example, the standard error. Moreover, it is very common to use asterisks to express the level of significance of the estimated coefficients. The last few lines of the table are usually reserved for some overall information about the regression model as a whole. At the very minimum, this should be the number of observations. Very often, you also see some measure for the goodness of fit.

To produce tables such as table 9.2, you need `estimates store` and `estimates table`. The command `estimates store` saves the results of a regression model under a specified name in the computer's memory. The command `estimates table` displays the results of one or more saved regression models in one table.

Table 9.2. A table of nested regression models

	(1)	(2)	(3)
Men	20905.7*** (1532.0)	12735.6*** (1657.0)	12859.2*** (1657.0)
Age in years	321.0*** (69.33)	414.9*** (65.76)	270.9** (95.18)
Part-time		-15113.1*** (2049.5)	-14788.3*** (2054.1)
Irregular		-25938.4*** (2693.5)	-25813.3*** (2692.5)
Years of education		3732.6*** (263.8)	3693.3*** (264.3)
Men			275.0* (131.5)
Constant	21234.9*** (1093.2)	30788.6*** (1400.8)	30593.7*** (1403.1)
<i>R</i> ²	0.070	0.174	0.176
Observations	2814	2814	2814

Standard errors in parentheses.

* $p < 0.05$, ** $p < 0.01$, *** $p < 0.001$ Source: `anincome.do`

Here we show how to use the two commands to produce a table that is already quite similar to table 9.2. We start estimating the first regression model and store it under the name `model1`:

```
. use data1, clear
. generate men = sex==1
. generate age = 2009 - ybirth
. summarize age, meanonly
. replace age = age-r(mean)
. replace income = . if income == 0
. regress income men age
. estimates store model1
```

We then proceed by estimating the second and third models,

```
. replace emp = . if emp==5
. regress income men age i.emp yedu
. estimates store model2
. regress income men age i.emp yedu i.men#c.age
. estimates store model3
```

and finally we use `estimates table` to show all three models.

```
. estimates table model1 model2 model3
```

Variable	model1	model2	model3
men	17641.994	12812.363	14351.971
age	470.2667	408.99101	264.19825
emp			
2		-14843.926	-14523.981
4		-25919.194	-25760.172
yedu		3748.8247	3708.9057
men#c.age			274.3864
1			
_cons	22827.3	-13121.708	-13569.714

This shows only the fitted coefficients of the three models. Several options of `estimates table` let us customize the table. The option `se`, to start with, adds standard errors to the output. The option `stats()` lets you specify any scalar mentioned in `ereturn list` to be printed below the table, and option `label` displays variable labels—if given—instead of variable names:

```
. estimates table model1 model2 model3, se stats(r2 N) label
```

Variable	model1	model2	model3
men	17641.994	12812.363	14351.971
age	1375.8066	1668.1342	1825.9869
	470.2667	408.99101	264.19825
	54.766644	66.414642	96.502912
emp			
2		-14843.926	-14523.981
4		2071.4198	2075.9848
Number of Years of Edu-n		-25919.194	-25760.172
		2732.7014	2732.1836
		3748.8247	3708.9057
		265.40999	265.95661
men#c.age			274.3864
1			132.74773
Constant	22827.3	-13121.708	-13569.714
	1050.1051	3641.9074	3646.2212
r2	.06595712	.17230189	.17356788
N	3410	2796	2796

legend: b/se

You can also use the option `star` for expressing significance with stars; however, this is not possible together with option `se`. If you want to produce a table that reports both

standard errors and asterisks, you need one of the following user-written commands: `esttab` (by Ben Jann), `outreg` (by John Luke Gallup), or `outreg2` (by Roy Wada),⁴⁴ which also provide the possibility to produce the table in L^AT_EX, HTML, or Microsoft Word format. Table 9.2 was created with `esttab`.

To concatenate models that apply the same set of independent variables but differ in the observations used, the basic principles remain the same. Consider that you want to compare the regression result of the last model between East and West Germany. We start estimating and storing the models,

```
. generate east = state > 10 & !missing(state)
. regress income men age i.emp yedu i.men#c.age if east
. estimates store east
. regress income men age i.emp yedu i.men#c.age if !east
. estimates store west
```

and then we use `estimates table` to show the saved models:

```
. estimates table east west
```

Variable	east	west
men	4510.6336	16660.102
age	228.2834	313.68134
emp		
2	-9496.7306	-16778.626
4	-19926.376	-27800.59
yedu	2553.4702	4292.6477
men#c.age		
1	-53.957075	349.83153
_cons	-6451.7335	-17663.455

9.5.2 Plots of coefficients

Plots of coefficients are used in two typical situations. The first situation arises if the regression model contains a categorical variable with many categories. A plot of all the coefficients for each of those categories usually makes the main results much clearer. In the following, we propose a solution using `marginsplot` for this case. The second situation arises when the model is fit for many different groups. In this case, one often wants to show a specific estimated coefficient for each of the groups. Here `marginsplot` does not help. Instead, we propose the `resultsset` approach (Newson 2003).

44. Read section 13.3 to learn how to install user-written commands.

We start with the first situation. Consider the following regression of household income on social class (egp), household size, and year of birth:

. regress hhinc i.egp hhsizex ybirth						
Source	SS	df	MS	Number of obs = 4788		
Model	8.7850e+11	10	8.7850e+10	F(10, 4777) = 176.34		
Residual	2.3798e+12	4777	498175319	Prob > F = 0.0000		
Total	3.2583e+12	4787	680651461	R-squared = 0.2696		
				Adj R-squared = 0.2681		
				Root MSE = 22320		
hhinc	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
egp						
2	-11053.99	1445.191	-7.65	0.000	-13887.23	-8220.75
3	-16131.8	1758.446	-9.17	0.000	-19579.17	-12684.44
4	-20014.95	1658.193	-12.07	0.000	-23265.77	-16764.12
5	-5479.182	1936.01	-2.83	0.005	-9274.653	-1683.711
8	-21327.39	1565.434	-13.62	0.000	-24396.36	-18258.42
9	-24828.94	1485.18	-16.72	0.000	-27740.57	-21917.3
15	-32172.19	1734.536	-18.55	0.000	-35572.68	-28771.7
18	-29419.26	1548.131	-19.00	0.000	-32454.31	-26384.21
hhsizex	8801.217	314.4606	27.99	0.000	8184.729	9417.705
ybirth	-219.6513	29.37816	-7.48	0.000	-277.246	-162.0566
_cons	465140.1	57542.31	8.08	0.000	352330.7	577949.6

The output shows eight coefficients for the variable `egp`. Because there are many coefficients, showing the results graphically—for example, as shown in figure 9.6—could be helpful.

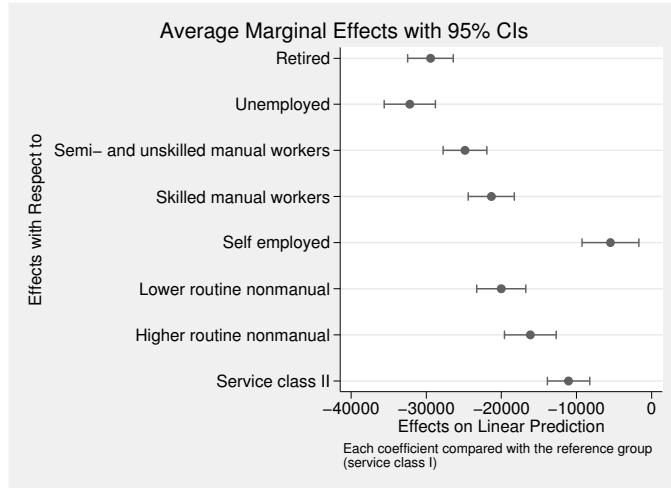


Figure 9.6. Example of a plot of regression coefficients with 95% confidence intervals

But how do we get there? To use `marginsplot` to show the results, we have to apply `margins` first. So far, we have used `margins` to show averages of predicted values for specified combinations of the independent variables. Here we want to show the estimated coefficients themselves. It is therefore necessary to trick `margins` into showing the coefficients instead of averages of predicted values. For linear regression models, this is possible with the option `dydx(varlist)`. The option, in general, shows the average marginal effect of the *varlist* in the parentheses (that is, the average slopes of the regression line). For linear models without interaction terms,⁴⁵ this is equal to the regression coefficient.

Average marginal effects						
	Delta-method			Number of obs = 4788		
	dy/dx	Std. Err.	z	P> z	[95% Conf. Interval]	
egp						
2	-11053.99	1445.191	-7.65	0.000	-13886.51	-8221.468
3	-16131.8	1758.446	-9.17	0.000	-19578.29	-12685.31
4	-20014.95	1658.193	-12.07	0.000	-23264.94	-16764.95
5	-5479.182	1936.01	-2.83	0.005	-9273.691	-1684.673
8	-21327.39	1565.434	-13.62	0.000	-24395.59	-18259.2
9	-24828.94	1485.18	-16.72	0.000	-27739.84	-21918.04
15	-32172.19	1734.536	-18.55	0.000	-35571.82	-28772.56
18	-29419.26	1548.131	-19.00	0.000	-32453.54	-26384.98

Note: dy/dx for factor levels is the discrete change from the base level.

Once you get `margins` to show the values needed, `marginsplot` will show the results graphically. We use the command with the option `horizontal` to give the long labels more space, the option `recast(scatter)` to use the plottype `scatter` instead of the default `connected` to show the coefficients, and the option `note()` to explain that each coefficient is compared with the reference group (here service class I). The command below will return the graph shown in figure 9.6.

```
. marginsplot, recast(scatter) horizontal
> ylabel(2 "Service class II" 3 "Higher routine nonmanual"
> 4 "Lower routine nonmanual" 5 "Self employed" 6 "Skilled manual workers"
> 7 "Semi- and unskilled manual workers" 8 "Unemployed" 9 "Retired")
> note("Each coefficient compared with the reference group (service class I)")
```

We now turn to the second situation. Consider that you wish to compare the gender-wage gap at mean age between the German states. Therefore, you have calculated 14 regression models—one for each category of the variable `state`—and you want to graphically display the coefficients of gender of all those models.

45. With interaction terms, you need to fix one constituent variable to the base category by using `at()` to get the coefficients of the other constituent variable.

Creating a graph of the coefficients requires two steps. The first step is to construct a *resultset* (Newson 2003). In the second step, the data in the *resultset* are then plotted using standard Stata graph commands.

A *resultset* is a Stata dataset that contains the results of one or more statistical analyses. For our example, we wish to arrive at a dataset with the variables **state** for the names of the German states, **b** for the estimated regression coefficients, and **se** for the standard errors of the estimated coefficients. Each observation of our *resultset* is thus a German state.

Such a *resultset* can be produced with the prefix command **statsby**. The **statsby** prefix works similarly to the **by** prefix: It performs a command behind the colon for each group of a by-list. However, unlike the **by** prefix, it does not show the results of the command behind the colon but instead stores statistics produced by that command in a dataset. In its most common use, the syntax of **statsby** requires the name of the statistics that should be saved, the specification of the by-list, and the option **clear** because **statsby** destroys the data in memory. The simplified syntax diagram of the command is

```
statsby stat-list, by(varlist) clear: command
```

where *stat-list* contains names of saved results and *command* can be any Stata command that produces saved results. Let us use the command to save the estimated coefficient for variable **men** of the income regression in the last section. Note that we create a string variable for state before we apply **statsby**; this is not necessary but turns out to be helpful later on.

```
. decode state, generate(ststr)
. statsby _b[men] _se[men], by(ststr) clear:
> regress income men age i.emp yedu menage
```

From

```
. describe
Contains data
obs: 14
vars: 3
size: 336
statsby: regress

variable name  storage  display  value
           type   format    label   variable label
ststr        str16  %16s      State of Residence
_stat_1      float   %9.0g    _b[men]
_stat_2      float   %9.0g    _se[men]

Sorted by:
Note: dataset has changed since last saved
```

we learn that the command has destroyed our dataset. Our new dataset has 14 observations and variables **ststr**, **_stat_1**, and **_stat_2**, whereby **ststr** is the string variable

for state, and `_stat_1` and `_stat_2` hold the estimated coefficients of `men` and their standard errors, respectively. We can use the standard errors to create the 95% confidence intervals as shown on page 231:

```
. generate lb = _stat_1 - 1.96 * _stat_2
. generate ub = _stat_1 + 1.96 * _stat_2
```

We now go on to plot the coefficients and their standard errors. In the graph, we want to sort the states according to their observed gender–wage gap. Therefore, we sort the dataset by `_stat_1` and create the new variable `state` according to the sort order.

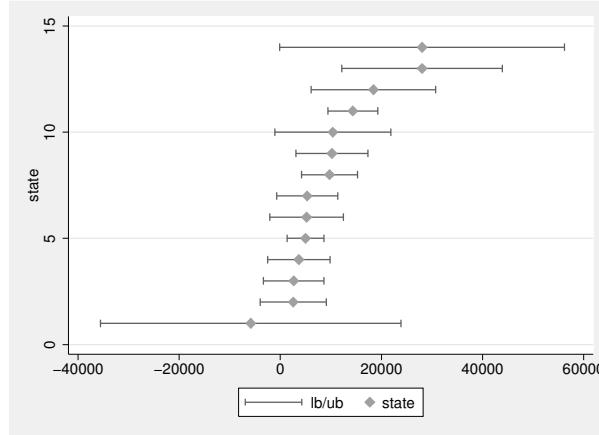
```
. sort _stat_1
. generate state = _n
```

We assign a value label to the sorted `state` variable and define the new label `state` by using a `forvalues` loop. The label holds the name of the state at each position. Note that this is possible because we have used `state` as a string variable in `statsby`.⁴⁶

```
. forvalues i = 1/14 {
. local state_name = ststr[`i`]
. label define state `i` "`state_name`", modify
. }
. label value state state
```

Having done all this, let us try a first sketch for the figure:

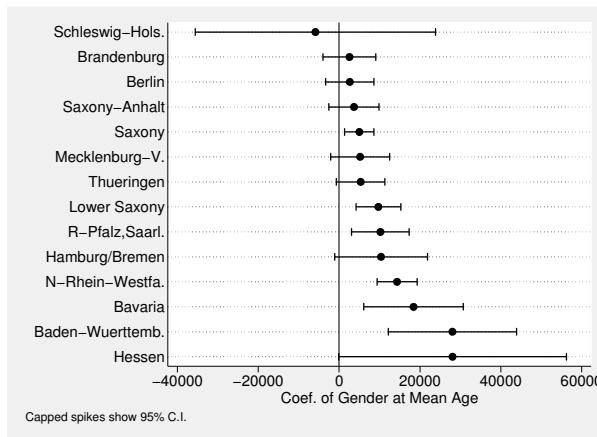
```
. twoway rcap lb ub state, horizontal || scatter state _stat_1
```



46. The last two steps can be automated using the user-written `egen` function `axis()` from the package `egenmore`. The package is available in the Statistical Software Components (SSC) archive (see chapter 13).

This already looks pretty much as desired. However, using some standard graph and plot options, the figure gets much nicer still (see 7.2.2):

```
. twoway || rcap lb ub state, horizontal lcolor(black)
> || scatter state _stat_1, ms(0) mcolor(black)
> || , yscale(reverse) ylab(1/14, valuelabel angle(0) gstyle(dot) notick)
> ytitle("") xline(0) xtitle(Coef. of Gender at Mean Age)
> legend(off) note("Capped spikes show 95% C.I.", span)
```



9.5.3 Conditional-effects plots

Conditional-effects plots show various regression lines for specified combinations of the independent variables of a multiple regression in one display. The effect of the independent variables can be read from the distances between the various lines and their respective slopes. These plots are especially useful when the effect of a variable cannot be fully grasped from the regression coefficients alone, that is, in the presence of interaction terms and transformed independent variables.

We have already shown an example of a conditional-effects plot in section 9.2.1. From there, we also see the basic principle of their construction: We first estimate a regression model, we then apply the postestimation command `margins` to calculate the (average) predicted values for various combinations of the independent variables, and we finally use `marginsplot` to produce the graph. In what follows, we apply this basic principle for models with interaction terms and transformed variables.

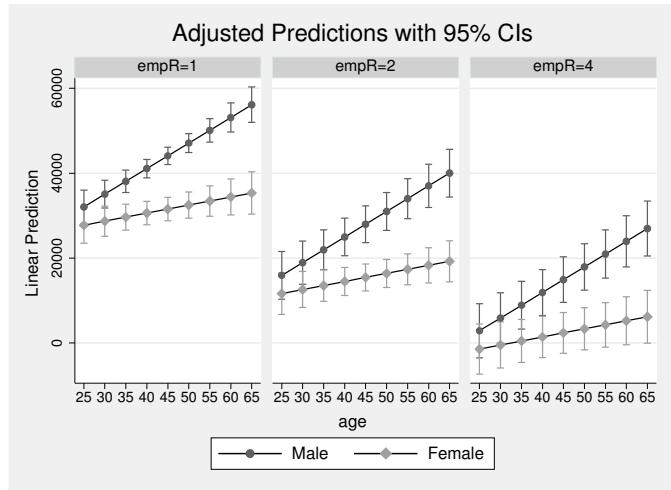
Let us start with the models for personal income described in section 9.4.2. The first model fit used an interaction between gender and age. Using factor-variable notation, the model can be reproduced as follows:

```
. use data1, clear
. generate age = 2009 - ybirth
. generate incomeR = income if income > 0
. generate empR = emp if emp!=5
. regress incomeR i.empR i.sex##c.age
```

(Note as an aside that there is no need to center variables if you do not report the coefficients but show the results with conditional-effects plots only.)

To produce the conditional-effects plot, we first apply `margins`. We set up the option `at()` such that the (average) predicted values are calculated for all combinations of employment status, gender, and various values for age. Subsequently, we use `marginsplot` to show the results of the `margins`.

```
. margins, at(age=(25(5)65) sex=(1,2) empR=(1,2,4))
. marginsplot, bydim(empR) byopt(rows(1))
```

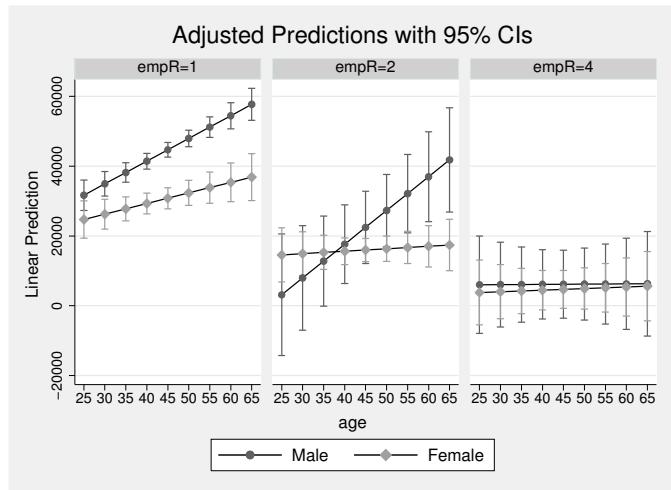


Unless stated differently, the first variable in the option `at()` of `margins` becomes the x variable in the conditional-effects plot. Predicted values are then displayed as lines for each combination of the values of the other variables in that option. The option `bydim(varlist)` is used to display one graph of each by-dimension in juxtaposition.

The figure shows that the effect of age (the slope of the line) is stronger for men than for women. We also see that the gender–wage gap becomes larger with age. Finally, we see that full-time employment leads to higher income than part-time and irregular employment.

We have seen all this already from the regression analysis in section 9.4.2, but the results are much easier to depict from the figure than from the regression table. And this advantage of the conditional-effects plot rises when the model becomes more complicated. We illustrate this point by allowing the interaction term of gender and age to vary with employment status, that is, by adding a three-way interaction:

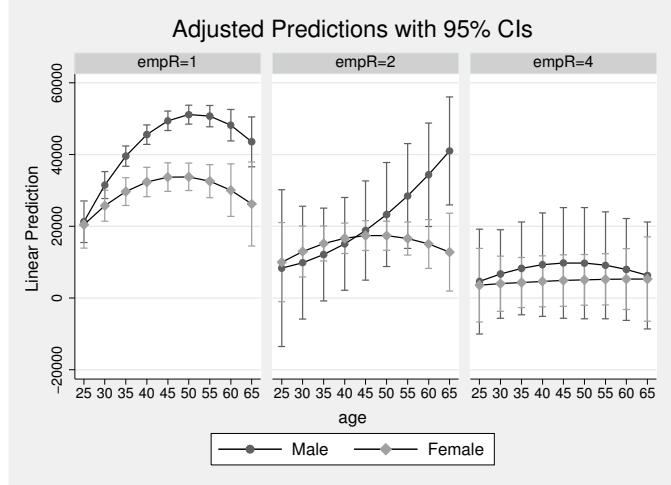
```
. regress incomeR i.empR##i.sex##c.age
. margins, at(age=(25(5)65) sex=(1,2) empR=(1,2,4))
. marginsplot, bydim(empR) byopt(rows(1))
```



The figure quite clearly shows that the gender–wage gap increases with age, although this is not visible to the same extent among the full-time employed and the part-time employed. Moreover, the effect is absent among those irregularly employed.

Let us finally extend our example by adding a quadratic age term to the model. Such quadratic age terms can be seen as an interaction term of a variable with itself, meaning that the effect of the variable varies with the value of the variable itself. Quadratic age terms are commonly used to model U-formed relationships. To make the model even more complicated, we allow for an interaction of the quadratic age term with the other variables in the model.

```
. regress incomeR i.empR##i.sex##c.age##c.age
. margins, at(age=(25(5)65) sex=(1,2) empR=(1,2,4))
. marginsplot, bydim(empR) byopt(rows(1))
```



It is pretty much impossible to interpret a model such as the above based on the coefficients alone. The conditional-effects plot provides helpful visualization.

9.6 Advanced techniques

In addition to multiple linear regression, there are several related models that can be fit in Stata. We cannot explain all of them in detail. However, a few of these models are so common that we want to describe the general ideas behind them. Each model is explained in detail in the *Stata Reference Manual*, where you will also find selected literature on the model.

9.6.1 Median regression

A median regression is quite similar to the OLS regression we talked about earlier. Whereas the sum of the squared residuals $\sum (\hat{y}_i - y_i)^2$ is minimized in OLS regression, the sum of the absolute residuals $\sum |\hat{y}_i - y_i|$ is minimized when applying median regression. Squaring residuals in OLS means that large residuals are more heavily weighted than small residuals. This property is lost in median regression, so it is less sensitive to outliers than OLS regression.

Median regression takes its name from its predicted values, which are estimates of the median of the dependent variable conditional on the values of the independent variables. In OLS, the predicted values are estimates of the conditional means of the dependent variable. The predicted values of both regression techniques, therefore, describe a measure of a certain property—the central tendency—of the dependent variable.

Stata treats median regression as a special case of a quantile regression. In quantile regression, the coefficient is estimated so that the sum of the weighted (that is, multiplied by the factor w_i) absolute residuals is minimized.

$$\min \sum (|y_i - \hat{y}_i| \times w_i)$$

Weights can be different for positive and negative residuals. If positive and negative residuals are weighted the same way, you get a median regression. If positive residuals are weighted by the factor 1.5 and negative residuals are weighted by the factor 0.5, you get a third-quartile regression, etc.

In Stata, you estimate quantile regressions by using the `qreg` command. Just as in any other Stata model command, the dependent variable follows the command, and then you specify the list of independent variables; the default is a median regression.

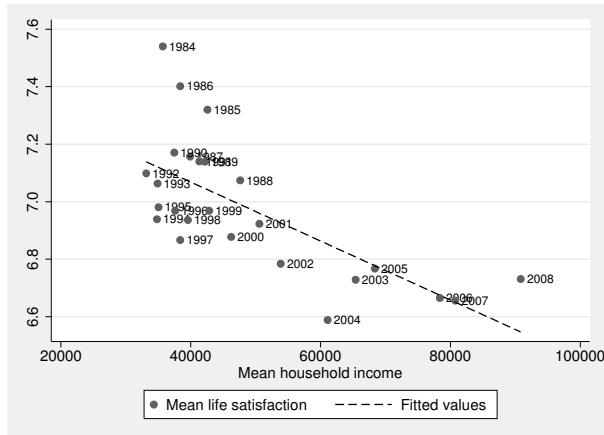
For this, use the dataset `data2agg.dta`, which contains the mean life satisfaction and the mean income data from the German population from 1984 to 2008.⁴⁷

```
. use data2agg, clear
```

⁴⁷. We used this small dataset to exemplify the effect of median regression. Be aware that working with aggregate data is prone to ecological fallacy (Freedman 2004).

First, take a look at a scatterplot with the regression line of the mean life satisfaction on the mean income:

```
. twoway scatter lsat hhinc, mlab(wave) || lfit lsat hhinc
```



In this graph, the data for 1984 might influence the regression results more than any other data point. Now estimate a median regression,

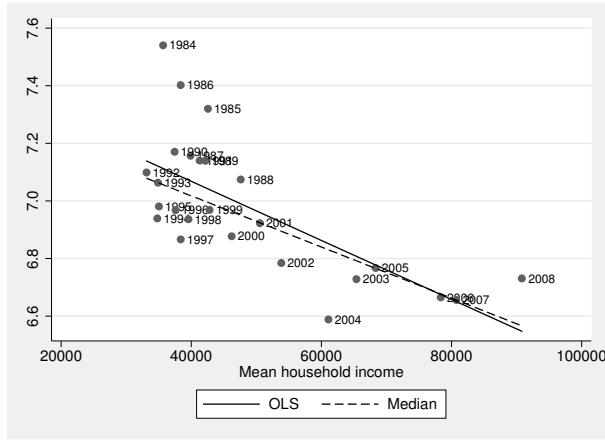
```
. qreg lsat hhinc
Iteration 1: WLS sum of weighted deviations = 3.1802261
Iteration 1: sum of abs. weighted deviations = 3.1024316
Iteration 2: sum of abs. weighted deviations = 3.0970802
Median regression                                         Number of obs =      25
  Raw sum of deviations 4.590568 (about 6.9682765)
  Min sum of deviations 3.09708                         Pseudo R2      =     0.3253

```

lsat	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
hhinc	-8.87e-06	3.55e-06	-2.50	0.020	-.0000162 -1.53e-06
_cons	7.37277	.1859772	39.64	0.000	6.988047 7.757493

and compare the predicted values of the median regression with the standard linear fit of the OLS regression:

```
. predict medhat
. graph twoway || scatter lsat hhinc, mlab(wave)
> || lfit lsat hhinc, lpattern(solid)
> || line medhat hhinc, sort lpattern(dash)
> ||, legend(order(2 "OLS" 3 "Median"))
```



The regression line of the median regression is not as steep as the standard regression line because the median regression is more robust to extreme data points, such as those from 1984.

9.6.2 Regression models for panel data

Panel data, or cross-sectional time-series data, contain repeated measures of the same individuals over time. An example of panel data is the GSOEP. In the GSOEP, about 12,000 persons have been asked identical questions every year since 1984. That is, the GSOEP measures the *same* variables for the *same* respondents at *different* points in time. Panel data, however, do not arise only from such panel surveys. The same data structure is also present if you have collected certain macroeconomic indices in many different countries over time, or even data about certain features of political parties over time. Really, what defines panel data is that the *same* entities are observed at different times. In the remaining section, we will use the term “individuals” for these entities.

In Stata, all the commands that deal with panel data begin with the letters `xt`, and these commands are described in the *Longitudinal-Data/Panel-Data Reference Manual*. A list of the `xt` commands can be found by typing `help xt`. Among the `xt` commands are some of the more complex models in the statistical universe, which we will not describe here. Instead, we will help you understand the thinking behind the major approaches to analyzing panel data together with examples of how you can use these approaches in Stata.⁴⁸

Before we describe the statistical models, we need to say a word about data management. All Stata commands for panel analysis require a panel dataset that is in long format, so the next section describes how to put your data in this format. Then we will explain fixed-effects models and error-components models.

From wide to long format

Panel data can be stored in wide format or in long format. In wide format, the observations of the dataset are the individuals observed and the variables are their characteristics at the respective time points. For example, if we ask four individuals, say, John, Paul, George, and Ringo, about their life satisfaction in 1968, 1969, and 1970, we can store their answers in wide format by making a dataset with four observations—namely, John, Paul, George, and Ringo—and three variables reflecting life satisfaction in 1968, 1969, and 1970, respectively (see table 9.3). However, the same information can also be stored in long format, where the observations are the individuals *at a specific point in time* and the variables are the observed characteristics. Hence, in our example, there would be three observations for John—one for 1968, one for 1969, and one for 1970—three observations for Paul, etc. The information on life satisfaction would be in one variable. To keep the information about the timing, we would need a new variable for the year of observation.

48. For more information, see Baltagi (2008); Baum (2010); Hardin and Hilbe (2003); Diggle, Liang, and Zeger (2002); Wooldridge (2010); and the literature cited in [XT] `xtreg`.

Table 9.3. Ways to store panel data

Wide format				Long format		
i	X_{1968}	X_{1969}	X_{1970}	i	year	X
John	7	8	5	John	1968	7
Paul	5	2	2	John	1969	8
George	4	3	1	John	1970	5
Ringo	8	8	6	Paul	1968	5
				Paul	1969	2
				Paul	1970	2
				George	1968	4
					:	:
						:
				Ringo	1970	6

Stata's `xt` commands generally expect panel data in long format. It is, however, more common for dataset providers to distribute panel data in wide format.⁴⁹ You will often need to reshape your dataset from wide to long.

An example of panel data in wide format is `data2w.dta`. Please load this dataset to follow our example of changing from wide format to long format:

```
. use data2w, clear
```

This file contains information on year of birth, gender, life satisfaction, marital status, individual labor earnings, and annual work hours of 4,710 respondents (individuals) from the GSOEP. The individuals were observed every year between 1984 and 2009. Therefore, with the exception of the time-invariate variables gender and year of birth, there are 26 variables for each observed characteristic. If you look at the file with

49. For very large panel studies, such as the GSOEP, the American Panel Study of Income Dynamics, or the British Household Panel Study, the situation tends to be even more complicated. These data are often distributed in more than one file. You would need to first combine these files into one file. In section 11.4, we show you how to do this using an example from the GSOEP, resulting in a dataset in wide format.

```
. describe
Contains data from data2w.dta
    obs:        4,710                               SOEP 1984-2009 (Kohler/Kreuter)
    vars:        113                                13 Feb 2012 17:08
    size:     1,102,140

variable   storage  display      value
name      type    format    label      variable label
persnr    long    %12.0g
hhnr      long    %12.0g
sex       byte    %20.0g    sex
ybirth    int     %8.0g
mar1984   byte    %29.0g    mar
mar1985   byte    %29.0g    mar
mar1986   byte    %29.0g    mar
mar1987   byte    %29.0g    mar
mar1988   byte    %29.0g    mar
mar1989   byte    %29.0g    mar
mar1990   byte    %29.0g    mar
mar1991   byte    %29.0g    mar
mar1992   byte    %29.0g    mar
mar1993   byte    %29.0g    mar
mar1994   byte    %29.0g    mar
mar1995   byte    %29.0g    mar
mar1996   byte    %29.0g    mar
Never changing person ID
Original household number
Gender
Year of birth
* Marital Status of Individual
(output omitted)
```

you will see that the variable names of the file have a specific structure. The first part of the variable names, namely, `lsat`, `mar`, `hour`, and `inc`, refers to the content of the variable, whereas the second part refers to the year in which the variable has been observed. Using this type of naming convention makes it easy to reshape data from wide to long.

Unfortunately, in practice, variable names rarely follow this naming scheme. Even the variables in the GSOEP do not. For your convenience, we have renamed all the variables in the dataset beforehand, but generally you will need to do this on your own using the `rename` and `renpfix` commands. Renaming all the variables of panel data in wide format can be quite cumbersome. If you need to rename many variables, you should review the concepts discussed in sections 3.2.2 and 12.2.1.⁵⁰

The command for changing data between wide and long is `reshape`. `reshape long` changes a dataset from wide to long, and `reshape wide` does the same in the other direction. Stata needs to know three pieces of information to reshape data:

- the variable that identifies the individuals in the data (that is, the respondents),
- the characteristics that are under observation, and
- the times when the characteristics were observed.

50. The user-written Stata command `soepren` makes it easier to rename GSOEP variables. The command is available on the SSC archive; for information about the SSC archive and installing user-written commands, see chapter 13.

The first piece of information is easy to obtain. In our example data, it is simply the variable `persnr`, which uniquely identifies each individual of the GSOEP. If there is no such variable, you can simply generate a variable containing the running number of each observation (see section 5.1.4).

The last two pieces of information are coded in the variable names. As we have seen, the first part of the variable names contains the characteristic under observation and the second part contains the time of observation. We therefore need to tell Stata where the first part of the variable names ends and the second part begins. This information is passed to Stata by listing the variable name stubs that refer to the characteristic under observation. Let us show you how this works for our example:

```
. reshape long hhinc lsat mar whours, i(persnr) j(wave)
```

First, option `i()` is required. It is used to specify the variable for the individuals of the dataset. Second, look at what we have specified after `reshape long`. We have listed neither variable names nor a *varlist*. Instead, we have specified the name stubs that refer to the characteristic under observation. The remaining part of the variable names is then interpreted by Stata as being information about the time point of the observation. When running the command, Stata strips off the year from the variables that begin with the specified name stub and stores this information in a new variable. Here the new variable is named `wave`, because we specified this name in the option `j()`. If we had not specified that option, Stata would have used the variable name `_j`.

Now let us take a look at the new dataset.

```
. describe
Contains data
    obs:           122,460                               SOEP 1984-2009 (Kohler/Kreuter)
    vars:            14
    size:        4,408,560

variable   storage   display   value
name      type      format   label   variable label
persnr    long      %12.0g
wave      int       %9.0g
hhnr      long      %12.0g
sex       byte     %20.0g
          sex
ybirth    int       %8.0g
          year of birth
mar       byte     %29.0g
          mar
hhinc    long      %10.0g
whours   int       %12.0g
lsat     byte     %32.0g
          scale11 *
sample   byte     %25.0g
          sample
intnr    long      %12.0g
strata   int       %8.0g
psu     long      %12.0g
dweight  float    %12.0g
          Design weights
          * indicated variables have notes

Sorted by: persnr wave
Note: dataset has changed since last saved
```

The dataset now has 14 variables instead of 113. Clearly there are still 4,710 *individuals* in the dataset, but because the observations made on the several occasions for each individual are stacked beneath each other, we end up with 122,460 *observations*. Hence, the data are in long format, as they must be to use the commands for panel data. And working with the `xt` commands is even more convenient if you declare the data to be panel data. You can do this with the `xtset` command by specifying the variable that identifies individuals followed by the variable that indicates time:

```
. xtset persnr wave
```

After reshaping the data once, reshaping from long to wide and vice versa is easy:

```
. reshape wide  
. reshape long
```

Fixed-effects models

If the data are in long format, you can now run a simple OLS regression. For example, if you want to find out whether aging has an effect on general life satisfaction, you could run the following regression:

		SS	df	MS		
Model		1666.28069	1	1666.28069	Number of obs =	86680
Residual		290617.155	86678	3.35283641	F(1, 86678) =	496.98
Total		292283.435	86679	3.37202131	Prob > F =	0.0000
					R-squared =	0.0057
					Adj R-squared =	0.0057
					Root MSE =	1.8311
		Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
lsat						
age		-.0086023	.0003859	-22.29	0.000	-.0093586
_cons		7.301021	.0187129	390.16	0.000	7.264344
						7.337698

From this regression model, you learn that life satisfaction tends to decrease with age. However, after having read this chapter, you probably do not want to trust this regression model, particularly because of omitted variables. Should you control the relationships for quantities like gender, education, and the historical time in which the respondents grew up?

Now let us imagine that you include a dummy variable for each GSOEP respondent. Because there are 4,710 individuals in the dataset, this would require a regression model with 4,709 dummy variables, which might be overwhelming to work with. But for small datasets like those presented in table 9.3, this is not a problem. So let us deal with these data for now.

```

. preserve
. use beatles, clear
(Kohler/Kreuter)
. describe
Contains data from beatles.dta
    obs:                 12
    vars:                  4
    size:                 60
                                Kohler/Kreuter
                                13 Feb 2012 17:08

```

variable name	storage type	display format	value label	variable label
persnr	byte	%9.0g		Person
time	int	%9.0g		Year of observation
lsat	byte	%9.0g		Life satisfaction (fictive)
age	byte	%9.0g		Age in years

Sorted by:

This dataset contains the age and (artificial) life satisfaction of four Englishmen at three points in time in long format. The command

. regress lsat age					
Source	SS	df	MS		
Model	13.460177	1	13.460177	Number of obs = 12	
Residual	68.2064897	10	6.82064897	F(1, 10) = 1.97	
Total	81.6666667	11	7.42424242	Prob > F = 0.1904	
				R-squared = 0.1648	
				Adj R-squared = 0.0813	
				Root MSE = 2.6116	
lsat	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
age	.6902655	.4913643	1.40	0.190	-.4045625 1.785093
_cons	-14.32153	13.65619	-1.05	0.319	-44.74941 16.10635

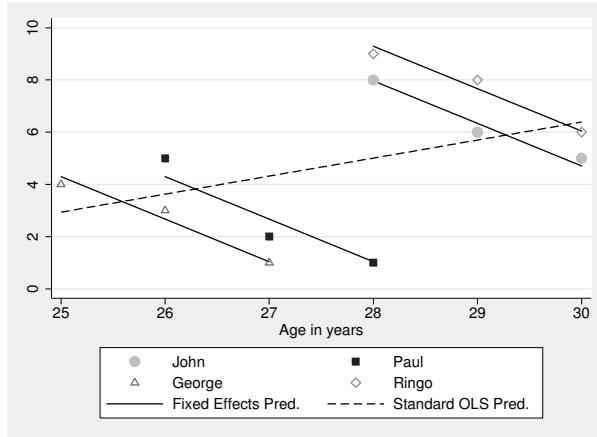
mirrors the regression analysis from above, showing a slight insignificant, positive effect of age on life satisfaction. Incorporating dummy variables for each individual of the dataset into this regression is straightforward.

. regress lsat age i.persnr						
Source	SS	df	MS	Number of obs = 12 F(4, 7) = 90.95 Prob > F = 0.0000 R-squared = 0.9811 Adj R-squared = 0.9703 Root MSE = .4693		
Model	80.125	4	20.03125			
Residual	1.54166667	7	.220238095			
Total	81.6666667	11	7.42424242			
lsat	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]	
age	-1.625	.165921	-9.79	0.000	-2.017341	-1.232659
persnr						
2	-6.916667	.5068969	-13.65	0.000	-8.115287	-5.718046
3	-8.541667	.6281666	-13.60	0.000	-10.02704	-7.056289
4	1.333333	.383178	3.48	0.010	.4272613	2.239405
_cons	53.45833	4.81933	11.09	0.000	42.06243	64.85424

Now it appears that age has a strong negative effect on life satisfaction. The sign of the age effect has reversed, and we will soon see why. But let us first say something about the individual dummies. The estimated coefficients of the individual dummies reflect how strongly the life satisfaction of the four Englishmen differs. You can see that persons 1 and 4 have a much higher life satisfaction than persons 2 and 3. You do not know *why* these people differ in their life satisfaction; the differences are not surprising because different people perceive life differently. Maybe they live in different neighborhoods, have different family backgrounds, grew up under different circumstances, or just have different habits about answering odd questions in population surveys. What is important here is that because you put individual dummies into the regression model, you have reliably controlled for any differences between the persons. In this sense, the estimated coefficient for age cannot be biased because we omitted stable characteristics of these persons. It is a pure aging effect, which rests solely on the development of the life satisfaction during the aging process of these four men.

This interpretation of the estimated age coefficient can be illustrated with the following plot.

```
. predict yhat
. separate yhat, by(persnr)
. separate lsat, by(persnr)
. graph twoway || line yhat? age, clstyle(p1..)
> || lfit lsat? age, lpattern(dash)
> || scatter lsat? age, ms(0 S Th Dh) msiz(*1.5)
> || , legend(order(6 "John" 7 "Paul" 8 "George" 9 "Ringo"
> 1 "Fixed Effects Pred." 5 "Standard OLS Pred."))
>
```



The plot is an overlay of a standard scatterplot with different markers for each person in the dataset (`scatter lsat? age`), a conditional-effects plot of the regression model with the person dummies (`line yhat? age`), and a simple regression line for all the data (`lfit lsat age`). If you look at the markers for each person separately, you will find that the life satisfaction decreases as the person gets older. At the same time, however, Ringo and John, the two oldest people in the dataset, have a higher life satisfaction than Paul and George. If we do not control for this, differences between people contribute to the age effect. The age effect of the simple OLS regression just shows that the older people have a higher life satisfaction than the younger ones. After we control for the personal differences, the only variation left is that within each person, and the age effect reflects the change in life satisfaction as each person gets older.

Because the regression model with person dummies restricts itself on the variation within each person, the model is sometimes called the *within estimator*, covariance model, individual dummy-variable model, or *fixed-effects model*.

Whereas the derivation of the fixed-effects model is straightforward, the technical calculation of the model in huge datasets is not. The problem arises because the number of independent variables in regression models is restricted to 800 in Stata/IC and to 11,000 in Stata/MP and Stata/SE. Therefore, you cannot estimate a fixed-effects model by incorporating individual dummies in datasets with more than 800 or 11,000 individuals, respectively.

Fortunately, in the linear regression model, you can use an algebraic trick to fit the fixed-effects model, anyway. And even more fortunately, Stata has a command that does this algebraic trick for you: `xtreg` with the option `fe`. Here you can use the command for our small example data,

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

which reproduces the estimated age coefficient of the model with dummy variables exactly. You do not need to list the dummy variables in this command. Instead, you either `xtset` your data or else specify the name of the variable, which identifies the individuals in the option `i()`.

The same logic applies if you want to fit the fixed-effects model for larger datasets. Therefore, you can also use the same command with our previously constructed dataset. Because you have already used the command `xtset` above (see page 332), you do not need to specify the `i()` option.

```
. restore
. xtreg lsat age, fe
Fixed-effects (within) regression
Group variable: persnr
Number of obs      =     86680
Number of groups   =      4709
R-sq: within      =  0.0200
          between    =  0.0055
          overall    =  0.0057
Obs per group: min =       6
                  avg =    18.4
                  max =    26
F(1,81970)        =  1669.61
Prob > F         =  0.0000
corr(u_i, Xb)    = -0.3028


```

	Coef.	Std. Err.	t	P> t	[95% Conf. Interval]
lsat					
age	-0.0325964	.0007977	-40.86	0.000	-.03416 -.0310328
_cons	8.398466	.0367987	228.23	0.000	8.326341 8.470591
sigma_u	1.287599				
sigma_e	1.4066128				
rho	.45591217		(fraction of variance due to u_i)		

F test that all u_i=0: F(4708, 81970) = 13.79 Prob > F = 0.0000

The values of the estimated coefficients for the 4,709 dummy variables are not shown in the output and were not estimated. But the coefficient for age in the model is fit as if the dummy variables were present. The fixed-effects model controls for all time-invariant differences between the individuals, so the estimated coefficients of the fixed-effects models cannot be biased because of omitted time-invariant characteristics. This feature makes the fixed-effects model particularly attractive.

One side effect of the features of fixed-effects models is that they cannot be used to investigate time-invariant causes of the dependent variables. Technically, time-invariant characteristics of the individuals are perfectly collinear with the person dummies. Substantively, fixed-effects models are designed to study the causes of changes within a person. A time-invariant characteristic cannot cause such a change because it is constant for each person.

9.6.3 Error-components models

Let us begin our description of error-components models with the simple OLS regression:

```
. regress lsat age
```

This model ignores the panel structure of the data and treats data as cross-sectional. From a statistical point of view, this model violates an underlying assumption of OLS regression, namely, the assumption that all observations are independent of each other. In panel data, you can generally assume that observations from the same individual are more similar to each other than observations from different individuals.

In observing the similarity of the observations from one individual, you might say that the residuals of the above regression are correlated. That is, an individual with a high positive residual at the first time of observation should also have a high positive residual at the second time point, etc.

Let us show you that the residuals of the above regression model are in fact correlated. First, we calculate the residuals from the above regression model,

```
. predict res, resid
```

and then we change the dataset to the wide format. Because we have generated a new variable in the long format since last using `reshape`, we cannot just type `reshape wide`; instead, we need to use the full syntax:

```
. reshape wide lsat mar whours hhinc age res, i(persnr) j(wave)
```

You end up with 26 variables containing the residuals for each individual for every year. These variables can be used to construct a correlation matrix of the residuals. We will display this correlation matrix only for the residuals from the 1980s:

```
. correlate res198?
(obs=2248)
```

	res1984	res1985	res1986	res1987	res1988	res1989
res1984	1.0000					
res1985	0.5122	1.0000				
res1986	0.4437	0.4755	1.0000			
res1987	0.3801	0.4244	0.5020	1.0000		
res1988	0.3674	0.3946	0.4590	0.5240	1.0000	
res1989	0.3665	0.3914	0.4005	0.4843	0.5268	1.0000

The residuals are in fact highly correlated. Let us now define this correlation matrix as $\mathbf{R}_{t,s}$:

$$\mathbf{R}_{t,s} = \begin{pmatrix} 1 & & & \\ r_{e_{i2}, e_{i1}} & 1 & & \\ \vdots & \vdots & \ddots & \\ r_{e_{iT}, e_{i1}} & r_{e_{iT}, e_{i2}} & \dots & 1 \end{pmatrix}$$

As we said, in computing the simple OLS regression on panel data, you assume among other things that all correlations of this correlation matrix are 0, or more formally,

$$\mathbf{R}_{t,s} = \begin{cases} 1 & \text{for } t = s \\ 0 & \text{otherwise} \end{cases}$$

As we have seen, this assumption is not fulfilled in our example regression. Hence, the model is not correctly specified. This is almost always the case for panel data. With panel data, you should expect correlated errors. In error-components models, you can therefore hypothesize about the structure of $\mathbf{R}_{t,s}$. Probably the simplest model after the simple regression model is the random-effects model:

$$\mathbf{R}_{t,s} = \begin{cases} 1 & \text{for } t = s \\ \rho & \text{otherwise} \end{cases}$$

Here the hypothetical structure of $\mathbf{R}_{t,s}$ is that observational units are more similar to each other over time than observations across observational units. The Stata command for random-effects models is `xtreg` with the option `re`.

```
. reshape long
. xtreg lsat age, re
```

Another reasonable assumption for the correlation structure of the residuals might be that the similarity between observations within each observational unit is greater the shorter the elapsed time between the observations. This structure can be imposed using an AR(1) correlation matrix:⁵¹

$$\mathbf{R}_{t,s} = \begin{cases} 1 & \text{for } t = s \\ \rho^{|t-s|} & \text{otherwise} \end{cases}$$

Different structures for the correlation matrix allow for a nearly infinite number of model variations. All of these variations can be estimated using the `xtgee` command with the `corr()` option for specifying predefined or customized correlation structures. Typing

51. AR is short for autoregression.

```
. xtgee lsat age, corr(exchangeable)
```

specifies the random-effects model and produces results similar to those from `xtreg, re`.⁵² Typing

```
. xtgee lsat age, corr(ar1)
```

produces a model with an AR(1) correlation matrix. Typing

```
. xtgee lsat age, corr(independent)
```

produces the standard OLS regression model described at the beginning of this section.

You can interpret the estimated coefficients of error-components models just like the estimated coefficients of a simple OLS regression model. But unlike in the simple OLS model, in an error-components model, if the error structure is correctly specified, the estimates are more accurate. Because the estimated coefficients are based on variations within and between the individuals, you should have no problem investigating the effects of time-invariant independent variables on the dependent variable. Unlike in the fixed-effects model, the estimated coefficients can be biased because of omitted time-invariant covariates.

9.7 Exercises

1. Set up a regression model with `data1.dta` to test the hypothesis that household income (`hhinc`) increases general life satisfaction (`lsat`). How many observations are used in your model? What is the estimated life satisfaction for respondents with a household income of €12,000? How does the predicted life satisfaction change when household income increases by €6,000?
2. Check whether the correlation between general life satisfaction and household income is due to factors that are influences of life satisfaction or correlated with income (like gender, age, education, employment status). How does the effect of household income change when you take these other variables into account?
3. Check your model for influential cases. List the three observations with the highest absolute value of the DFBETA for household income.
4. Check your model for nonlinearities. Discuss your findings. Go through the following three steps and see how your model improves.
 - Recalculate your regression model with log household income instead of household income and add a squared age variable to the regression model.
 - Reiterate the regression diagnosis for the new model. What would you further do to improve your model?
 - Reiterate the process until you are satisfied.

⁵². `xtgee, corr(exchangeable)` and `xtreg, re` produce slightly different results because of implementation details that are too technical to discuss here. In practice, the results are usually quite similar.

5. Check the homoskedasticity assumption for your final model. What do you conclude? Recalculate the model with robust standard errors, if necessary.
6. Sociologist Ronald Inglehart has proposed the theory that younger generations are less materialistic than older generations.
 - Which testable implication has this hypothesis for your regression model?
 - Change your regression model to test this implication. Is Inglehart correct?
 - Create a conditional-effects plot to illustrate the results of your analysis of Inglehart's hypotheses.

10 Regression models for categorical dependent variables

Researchers in the social sciences often deal with categorical dependent variables, whose values may be dichotomous (for example, rented apartment: yes or no), nominal (party identification: CDU, SPD, or Green Party), or ordinal (no concerns, some concerns, strong concerns). Here we will present several procedures used to model variables such as these by describing a procedure for dealing with dichotomous dependent variables: logistic regression.

Logistic regression is most similar to linear regression, so we will explain it as an analogy to the previous chapter. If you have no experience or knowledge of linear regression, first read chapter 9 up to page 266.

As in linear regression, in logistic regression a dependent variable is predicted by a linear combination of independent variables. A linear combination of independent variables can look like this:

$$\beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_{K-1} x_{K-1,i}$$

Here x_{1i} is the value of the first independent variable for interviewee i , x_{2i} is the respective value of the second independent variable, and so on. The regression parameters $\beta_1, \beta_2, \dots, \beta_{K-1}$ represent the weights assigned to the variables.

We did not say, however, that the mean of the dependent variable y_i is equal to that linear combination. In contrast to linear regression, in logistic regression you must consider a particular transformation of the dependent statistic. Why such a transformation is required and why linear regression is inappropriate are explained in section 10.1, whereas the transformation itself is explained in section 10.2.1.

Section 10.2.2 explains the method by which the logistic regression coefficients are estimated. Because this explanation is slightly more difficult and is not required to understand logistic regression, you can skip it for now.

Estimating a logistic regression with Stata is explained in section 10.3. Then we discuss methods of verifying the basic assumptions of the model in section 10.4. The procedure for verifying the joint significance of the estimated coefficients is discussed in section 10.5, and section 10.6 demonstrates a few possibilities for refining the modeling of correlations.

For an overview of further procedures—in particular, procedures for categorical variables with more than two values—see section 10.7.

As with linear regression (chapter 9), you will need to do some additional reading if you want to fully understand the techniques we describe. We suggest that you read Hosmer and Lemeshow (2000) and Long (1997).

10.1 The linear probability model

Why is linear regression not suitable for categorical dependent variables? Imagine that you are employed by an international ship safety regulatory agency and are assigned to take a closer look at the sinking of the *Titanic*. You are supposed to find out whether the seafaring principle of “women and children first” was put into practice or if there is any truth in the assumption made by the film *Titanic*, in which the first-class gentlemen took places in the lifeboats at the expense of the third-class women and children.

For this investigation, we have provided you with data on the sinking of the *Titanic*.¹ Open the file by typing²

```
. use titanic2
```

and before you continue to read, make yourself familiar with the contents of the dataset by using the commands

```
. describe
. tab1 _all
```

You will discover that the file contains details on the age (`age`), gender (`men`), and passenger class (`class`) of the *Titanic*'s passengers, as well as whether they survived the catastrophe (`survived`).

To clarify the disadvantages of using linear regression with categorical dependent variables, we will go through such a model. First, we will investigate whether children really were rescued more often than adults. What would a scatterplot look like where the Y variable represents the variable for survival and the X variable represents age? You may want to sketch this scatterplot yourself.

The points can be entered only on two horizontal lines: either at the value 0 (did not survive) or at 1 (survived). If children were actually rescued more often than adults, the number of points on the 0 line should increase in relation to those on the 1 line the farther to the right that you go. To check whether your chart is correct, type

1. The dataset was collected by the British Board of Trade in their investigation of the sinking and made publicly available by Dawson (1995). For teaching purposes, we have changed the original dataset in that we have created a fictional age variable, because the original set differentiates merely between adults and children. Among the files you installed in the steps discussed in the *Preface* is the do-file we used to create the dataset (`crtitanic2.do`).

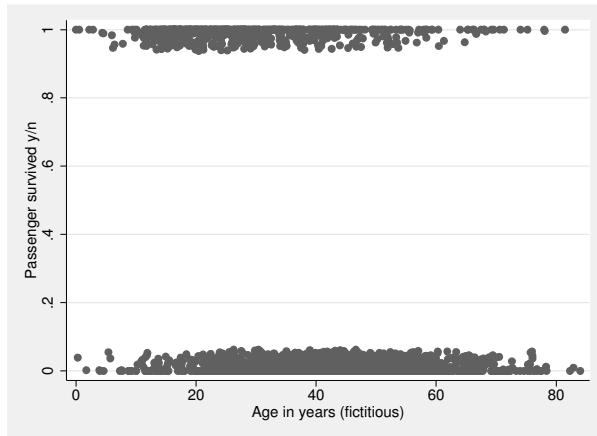
2. Please make sure that your working directory is `c:\data\kk3`; see page 3.

```
. scatter survived age
```

This diagram is not particularly informative, because the plot symbols are often directly marked on top of each other, hiding the number of data points.

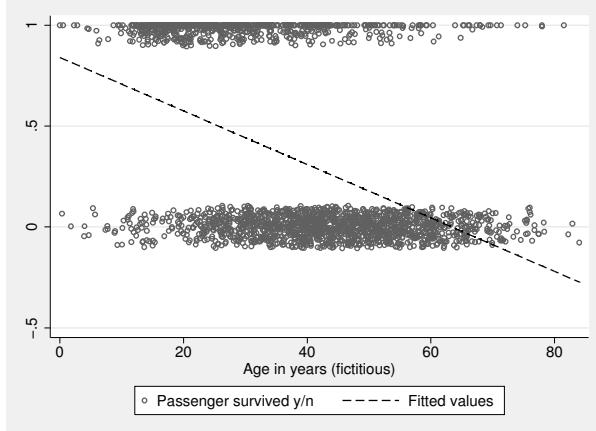
With the help of the `scatter` option `jitter()`, you can produce a more informative diagram. `jitter()` adds a small random number to each data point, thus showing points that were previously hidden under other points. Within the parentheses is a number between 1 and 30 that controls the size of the random number; you should generally use small numbers if possible.

```
. scatter survived age, jitter(10)
```



On examining the chart, you might get the impression that there is a negative correlation between ages and survival of the *Titanic* disaster. This impression is confirmed when you draw the regression line on the chart (see also section 9.1.2):

```
. regress survived age
. predict yhat
. scatter survived age, jitter(10) ms(oh) || line yhat age, sort
```



The chart reveals one central problem of linear regression for dichotomous dependent variables: the regression line in the illustration shows predicted values of less than 0 from around the age of 60 onward. What does this mean with regard to the content? Remind yourself of how the predicted values of dichotomous dependent variables are generally interpreted. Until now, we have understood the predicted values to be the estimated average extent of the dependent variables for the respective combination of independent variables. In this sense, you might say, for example, that the survival of a 5-year-old averages around 0.77. This is an invalid interpretation if you consider that passengers can only survive or not survive; they cannot survive just a little bit.

However, the predicted value of the dichotomous dependent variable can also be interpreted in a different way. You need to understand what the arithmetic mean of a dichotomous variable with the values of 0 and 1 signifies. The mean of the variable `survived`, for example, is 0.323. This reflects the share of passengers who survived.³ So, we see that the share of survivors in the dataset amounts to around 32%, or in other words, the probability that you will find a survivor in the dataset is 0.32. In general, the predicted values of the linear regression are estimates of the conditional mean of the dependent variable. Thus you can use the probability interpretation for every value of the independent variable: the predicted value of around 0.77 for a 5-year-old means a predicted probability of survival of 0.77. From this alternative interpretation, the linear regression model for dichotomous dependent variables is often called the linear probability model or LPM (Aldrich and Nelson 1984).

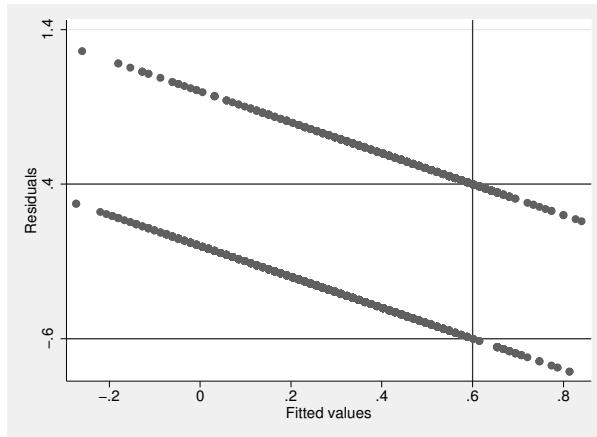
How can you interpret the negative predicted values for passengers over 60 with the help of the probability interpretation? In fact, you cannot, because according to the mathematical definition of probabilities, they must be between 0 and 1. Given

³. You can confirm this for yourself by typing `tab survived`.

sufficiently small or large values of the X variable, a model that uses a straight line to represent probabilities will, however, inevitably produce values of more than 1 or less than 0. This is the first problem that affects ordinary least-squares (OLS) regression of dichotomous variables.⁴

The second problem affects the homoskedastic assumption of linear regression that we introduced in section 9.3.2. According to this assumption, the variance of errors for all values of X (and therefore all values of \hat{Y}) should be constant. We suggested that the scatterplot of the residuals against the predicted values indicated a possible violation of this assumption. You can achieve a graph such as this for our linear probability model by typing

```
. predict r, resid
. scatter r yhat, yline(-.6 .4) ylab(-.6 .4 1.4) xline(.6)
```



Here you can observe that only two possible residuals can appear for every predicted value. Less apparent is that both of these residuals result directly from the predicted values. If a survivor (`survived = 1`) has a predicted value of 0.6 because of her age, she will have a residual of $1 - 0.6 = 0.4$. If you predict a value of 0.6 for an individual who did not survive (`survived = 0`), you will receive a value of $0 - 0.6 = -0.6$.

Thus the residuals are either $1 - \hat{y}_i$ or $-\hat{y}_i$. The variance of the residuals is $\hat{y}_i \times (1 - \hat{y}_i)$ and is therefore larger as the predicted values approach 0.5. The residuals of the linear probability model are therefore by definition heteroskedastic so the standard errors of the estimated coefficients will be wrong.

In conclusion, although a linear regression with a dichotomous dependent variable is possible, it leads to two problems. First, not all predicted values can be interpreted, and second, this model does not allow for correct statistical inference. To avoid these

4. In practice, this is a problem of little importance when predicted values of more than 1 or less than 0 do not appear for real values of the independent variables. However, using a model that would prevent such impossible probabilities from the start seems sensible.

problems, you need a model that produces probabilities only between 0 and 1 and relies on assumptions that are maintained by the model. Both are fulfilled by logistic regression, the basic principles of which we will introduce now.

10.2 Basic concepts

10.2.1 Odds, log odds, and odds ratios

Earlier we found that the linear OLS regression of dichotomous dependent variables can produce unwanted predicted values. This is clearly because we attempted to represent values between 0 and 1 with a straight line. The values estimated with a linear regression are basically not subject to any restrictions.

This means that, theoretically, values between $-\infty$ and $+\infty$ may emerge. Therefore, regression models that are based on a linear combination should use only dependent variables whose range of values are equally infinite.

Because the range of values for probabilities lies between 0 and 1, they are not suited to be estimated with a linear combination. An alternative is the logarithmic chance, which we will explain using the *Titanic* data from the previous section.

We previously received indications that children had a higher chance of survival than adults did. Now we want to investigate whether women were more likely to survive than men. You can obtain an initial indication of the chance of survival for women and men through a two-way tabulation between `men` and `survived`:

```
. tabulate men survived, row
```

Key			
frequency row percentage			
Male passenger y/n	Passenger survived y/n		
	No	Yes	Total
No	126 26.81	344 73.19	470 100.00
Yes	1,364 78.80	367 21.20	1,731 100.00
Total	1,490 67.70	711 32.30	2,201 100.00

In section 7.2.1, we interpreted tables like this using row or column percentages. By using the available row percentages, we determine that the overall share of survivors was around 32%, whereas that of the women was about 50 percentage points higher than that of the men (73% compared with 21%). You can do a similar comparison by dividing the number of survivors by the number of dead. For the women, this ratio would be 344:126.

```
. display 344/126
2.7301587
```

You will get the same number⁵ if you divide the proportional values (in this case, the row percentages):

```
. display .7319/.2681
2.7299515
```

You can interpret these ratios as follows: for women, the estimated probability of surviving is almost three times as high as the estimated probability of dying. The estimated probability of dying is around one-third ($1 : 2.73 = 0.366$) the estimated probability of surviving. In practice, we would say that the estimated odds of surviving are generally around 2.73 to 1, whereas the estimated odds of dying are around 1 to 2.73.

In general, this relationship can be written as

$$\text{odds}_{\text{surviving}} = \frac{\text{Probability}_{\text{surviving}}}{\text{Probability}_{\text{dying}}}$$

or slightly shorter by using symbols instead of text:

$$\text{odds} = \frac{\Pr(Y = 1)}{1 - \Pr(Y = 1)} \quad (10.1)$$

The probabilities of survival, $\Pr(Y = 1)$, and dying, $\Pr(Y = 0)$, can be found, respectively, in the numerator and the denominator. Because the only two alternatives are surviving or dying, their probabilities sum to 1, so we replace $\Pr(Y = 0)$ with $1 - \Pr(Y = 1)$.

You can also estimate the chance of survival for men. Their odds of survival are considerably lower than those of the women: $367/1364 = 0.269$. This means that for men, the estimated probability of survival stands at 0.269:1; men are 3.72 times more likely to be among the victims.

⁵. The deviations are due to roundoff error.

Of course, you can compare the odds of survival for men and women using a measured value. For instance, you can compare the estimated chance of survival for men with that of women by dividing the odds for men by the odds for women:

```
. display .269/2.73
.0985348
```

This relationship is called the *odds ratio*.

Here we would say that the odds of survival for men are 0.099 times, or one-tenth, the odds for women. Apparently, the principle of *women and children first* appears to have been adhered to. Whether this appearance actually holds is something that we will investigate in more detail in section 10.6.2.

However, first we should consider the suitability of using odds for our statistical model. Earlier, we looked at the probabilities of surviving the *Titanic* catastrophe by passenger age. We found that predicting these probabilities with a linear combination could result in values outside the definition range of probabilities. What would happen if we were to draw upon odds instead of probabilities?

In the first column of table 10.1, we list several selected probability values. You will see that at first the values increase slowly, then rapidly, and finally slowly again. The values are between 0 and 1. If we presume that the values represent the estimated chance of survival for passengers of different ages on the *Titanic*, the first row would contain the group of the oldest passengers with the lowest chance of survival, and the bottom row would contain the group of the youngest passengers with the highest chance of survival. Using (10.1), you can calculate the odds that an individual within each of these groups survived the *Titanic* catastrophe. Furthermore, imagine that each of these groups contains 100 people. Because the first group has a probability of 0.01, 1 person out of 100 should have survived, a ratio of 1 to 99 (1:99). If you calculate 1/99, you get 0.010. You can perform this calculation for each row in the table. The values of the odds lie between 0 and $+\infty$; odds of 0 occur if there are no survivors within a specific group, whereas odds of $+\infty$ occur when everyone in a large group survives. If the number of survivors is equal to the number of victims, we get odds of 1.

Table 10.1. Probabilities, odds, and logits

$\Pr(Y = 1)$	odds = $\frac{\Pr(Y=1)}{1-\Pr(Y=1)}$	$\ln(\text{odds})$
0.01	1/99 =	0.01
0.03	3/97 =	-3.48
0.05	5/95 =	-2.94
0.20	20/80 =	-1.39
0.30	30/70 =	-0.85
0.40	40/60 =	-0.41
0.50	50/50 =	0
0.60	60/40 =	0.41
0.70	70/30 =	0.85
0.80	80/20 =	1.39
0.95	95/5 =	2.94
0.97	97/3 =	3.48
0.99	99/1 =	4.60

Odds are therefore *slightly* better suited than probabilities to be estimated with a linear combination. No matter how high the absolute value is when predicting with a linear combination, it will not be outside the definition range of the odds. However, a linear combination also allows for negative values, but negative odds do not exist. You can avoid this problem by using the natural logarithm of the odds. These values, called *logits*, are displayed in the last column of the table.

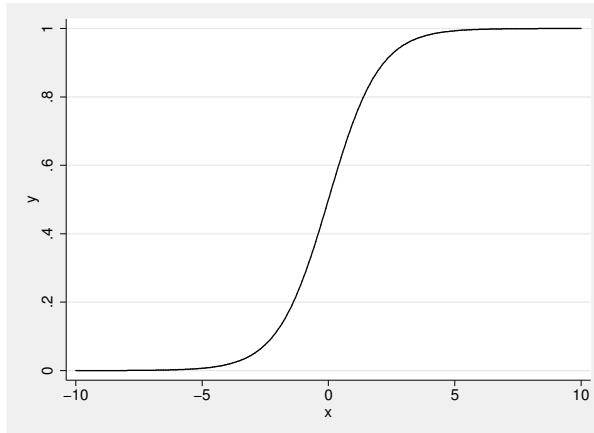
Now look at the values of the logits more closely. Although the odds have a minimum boundary, the logarithmic values have no lower or upper boundaries. The logarithm of 1 is 0. The logarithm of numbers less than 1 results in lower figures that stretch to $-\infty$ as you approach 0. The logarithm of numbers over 1 stretch toward $+\infty$. Note also the symmetry of the values. At a probability of 0.5, the odds lie at 1:1 or 50:50; the logarithmic value lies at 0. If you look at the probabilities above and below 0.5, you will see that at equal intervals of probabilities of the odds' logarithm, only the algebraic sign changes.

The logit is not restricted and has a symmetric origin. It can therefore be represented by a linear combination of variables and hence is better suited for use in a regression model. Unfortunately, the logit is not always easy to interpret. Your employers are unlikely to understand you if you tell them that the logarithmic chance of survival of a male *Titanic* passenger is -1.31 , whereas that of a female passenger is $+1.00$. However, you can convert the values of the logits back into probabilities

$$\Pr(Y = 1) = \frac{e^L}{1 + e^L} \quad (10.2)$$

where L is logit and e is Euler's constant ($e \approx 2.718$). A functional graph of this transformation can be drawn as follows:

```
. twoway function y=exp(x)/(1+exp(x)), range(-10 10)
```



In this graph, we see another interesting characteristic of logits: although the range of values of the logits has no upper or lower boundaries, the values of the probabilities calculated from the logits remain between 0 and 1. For logits between around -2.5 and 2.5 , the probabilities increase relatively rapidly; however, the closer you get to the boundary values of the probabilities, the less the probabilities change. The probabilities asymptotically approach the values 0 and 1, but they *never* go over the boundaries. From this, we can deduce that on the basis of a linear combination, predicted logits can always be converted into probabilities within the permitted boundaries of 0 and 1.

To summarize, the logarithmic chance is well suited to be estimated with a linear combination and can therefore be used in a regression model. The equation for such a model could be

$$L_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_{K-1} x_{K-1,i} \quad (10.3)$$

This is called the logistic regression model or logit model. The formal interpretation of the β coefficients of this model is identical to that of the linear regression (OLS). When an X variable increases by one unit, the predicted values (the logarithmic odds) increase by β units.

Before we use logistic regression, let us examine the procedure for estimating the β coefficients of (10.3). For the linear regression, we used the OLS procedure for estimation. For the logistic regression, we instead use the process of maximum likelihood. The logic of this process is somewhat more complex than that of OLS, even though the basic principle is similar: you look for the β coefficients that are optimal in a certain respect. We will explain this process in detail in the following section. However, you do not need to work through the example to understand the section that follows it!

10.2.2 Excursion: The maximum likelihood principle

In discussing linear regression, we explained the OLS process used to determine the b coefficients, which are estimators of the β coefficients. In principle, you could calculate the logarithmic odds for each combination of the independent variables and use these in an OLS regression model. Nevertheless, for reasons we will not explain here, a procedure such as this is not as efficient as the process of estimation applied in logistic regression, a process called the maximum likelihood principle.⁶ Using this technique, you can determine the b coefficients so that the proportionate values you observed become maximally probable. What does this mean? Before we can answer this question, we need to make a little detour.

On page 344, we informed you that 32.3% of the *Titanic* passengers survived. Suppose that you had determined this figure from a sample of the passengers. Here you could ask yourself how likely such a percentage may be, when the true number of the survivors amounts to, say, 60% of the passengers? To answer this question, imagine that you have selected one passenger from the population. If 60% of the passengers survived, the estimated probability that this passenger will be a survivor is 0.6, and the estimated probability that he or she will be a victim is 0.4. Now select a second passenger from the population. Whether this person is a survivor or a victim, the estimated probabilities remain the same (sampling with replacement).

In figure 10.1, we have conducted all possible samples with three observations. We obtained $2^n = 2^3 = 8$ samples with $n = 3$. In the first sample, we observed only survivors (S). The probability that a sample randomly selects three survivors is $0.6 \times 0.6 \times 0.6 = 0.6^3 = 0.216$. In the second, third, and fifth samples, we observed two survivors and one victim (V). Each of these samples has probability $0.6 \times 0.6 \times 0.4 = 0.6^2 \times 0.4^1 = 0.144$. In total, the probability of such a sample is $0.144 \times 3 = 0.432$. The probabilities of samples 4, 6, and 7 are each $0.6 \times 0.4 \times 0.4 = 0.6 \times 0.4^2 = 0.096$. In total, the probability of these samples is therefore $0.096 \times 3 = 0.288$. Finally, there is sample 8, where the probability lies at $0.4 \times 0.4 \times 0.4 = 0.4^3 = 0.064$. If, from the samples given in the mapping, we ask how likely it is that one of three survives, the answer is that it is as likely as samples 4, 6, and 7 together, that is, 0.288.

6. Andreß, Hagenaars, and Kühnel's (1997, 40–45) introduction to the maximum likelihood principle served as a model for the following section.

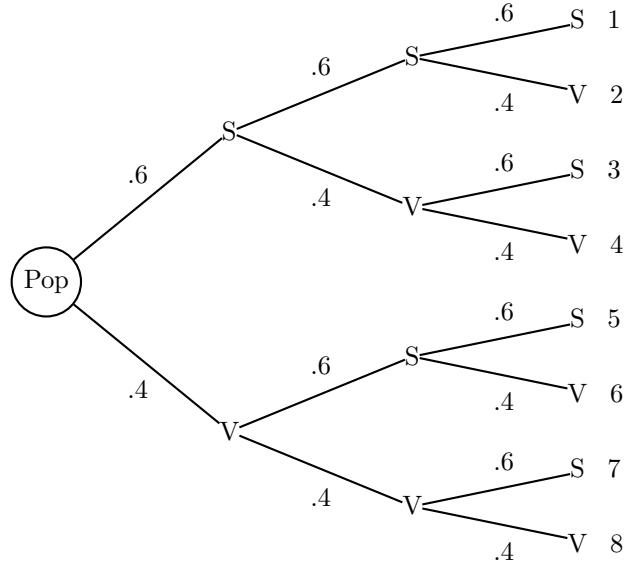


Figure 10.1. Sample of a dichotomous characteristic with the size of 3

Generally, the probability of observing h successes in a sample of size n is

$$\Pr(h|\pi, n) = \binom{n}{h} \pi^h (1 - \pi)^{n-h} \quad (10.4)$$

where π defines the probability of a positive outcome in the population. The term $\binom{n}{h}$ stands for $n!/h!(n-h)!$. It enables us to calculate the number of potential samples in which the dichotomous characteristic appears n times. In Stata, the probability of samples 4, 6, and 7 in our mapping can be calculated with this command:

```
. display comb(3,1) * .6^1 * .4^2
.288
```

In practice, we are usually not interested in this figure; instead our attention is on π , the characteristic's share in the population. Although π is unknown, we can consider what value of π would make the given sample most probable. For this, we can use various values for π in (10.4) and then select the value that results in the highest probability. Formally, this means that we are searching for the value of π for which the likelihood

$$\mathcal{L}(\pi|h, n) = \binom{n}{h} \pi^h (1 - \pi)^{n-h} \quad (10.5)$$

is maximized. We can forgo a calculation of $\binom{n}{h}$, because this term remains constant for all values of π . The likelihood is calculated with the same formula as in (10.4). If (10.4) is evaluated for all possible values of h , then the probabilities sum to 1, but this is not the case for the values of \mathcal{L} and all possible values of π . Therefore, we must differentiate between likelihood and probability.

You can do this for sample 2 from figure 10.1 (two survivors and one victim) by creating an artificial dataset with 100 observations:

```
. clear
. set obs 100
```

Now generate the variable `pi` by rendering a series of possible values for π :

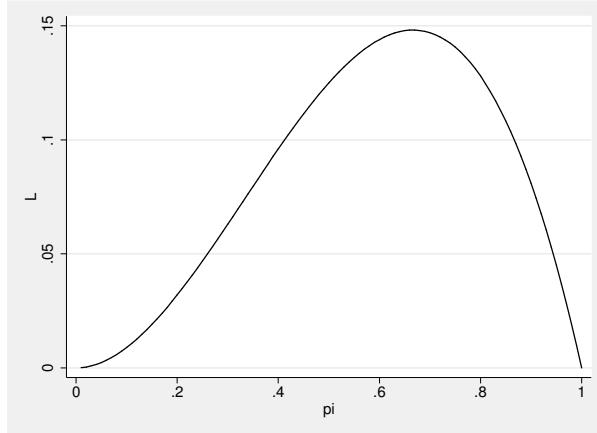
```
. generate pi = _n/100
```

Because h and n are known from the sample, you can calculate the likelihood for the various values of π :

```
. generate L = pi^2 * (1 - pi)^(3-2)
```

With the help of a graph, you can then analyze which π results in a maximal likelihood:

```
. line L pi, sort
```



The maximum of the likelihood lies around $\pi = 0.66$. This is the maximum likelihood estimate of the share of survivors from the population, given that the sample contains two survivors and one victim.

How can you estimate the β coefficients of our regression model with the maximum likelihood principle from (10.3)? The answer is simple. Instead of directly inserting the values for π , you can calculate π with the help of our regression model. Now insert (10.3) in (10.2):

$$\Pr(Y = 1) = \frac{e^{\beta_0 + \beta_1 x_{1i} + \dots + \beta_{K-1} x_{(K-1)i}}}{1 + e^{\beta_0 + \beta_1 x_{1i} + \dots + \beta_{K-1} x_{(K-1)i}}}$$

Also insert (10.3) in (10.5):

$$\begin{aligned}\mathcal{L}(\beta_k | f, n, m) &= \Pr(Y = 1)^h \times \{1 - \Pr(Y = 1)\}^{n-h} = \\ &\left(\frac{e^{\beta_0 + \beta_1 x_{1i} + \dots + \beta_{K-1} x_{K-1,i}}}{1 + e^{\beta_0 + \beta_1 x_{1i} + \dots + \beta_{K-1} x_{K-1,i}}} \right)^h \times \left(1 - \frac{e^{\beta_0 + \beta_1 x_{1i} + \dots + \beta_{K-1} x_{K-1,i}}}{1 + e^{\beta_0 + \beta_1 x_{1i} + \dots + \beta_{K-1} x_{K-1,i}}} \right)^{n-h}\end{aligned}$$

After doing this, you can attempt to maximize this function by trying out different values of β_k . However, like with OLS regression, it is better to reproduce the first derivative from β_k and to set the resulting standard equation as 0. The mathematical process is made easier when the log likelihood—that is, $\ln \mathcal{L}$ —is used. You will not find an analytical solution with this model, unlike with linear OLS regression. For this reason, iterative algorithms are used to maximize the log likelihood.

We have introduced the maximum likelihood principle for logistic regression with a dichotomous dependent variable. In principle, we can apply it to many different models by adapting (10.5) to reflect the distributional assumptions we wish to make. The resulting likelihood function is then maximized using a mathematical algorithm. Stata has a command called `m1` to do this, which is described in detail in Gould, Pitblado, and Poi (2010).

10.3 Logistic regression with Stata

Let us now set aside our *Titanic* example for an alternative. Say that you assumed that when the age and household income of a surveyed individual increases, the probability of living in an apartment or house they own also increases. Also you expect that the share of individuals who own their own residences⁷ differs between West Germany and East Germany.

Now let us load our dataset `data1.dta`.

```
. use data1, clear
```

To check your assumptions, you can calculate a logistic regression model of residence ownership against household income, age, and an East–West variable.

Stata has two commands for fitting logistic regression models: `logit` and `logistic`. The commands differ in how they report the estimated coefficients. `logit` reports the actual b 's in (10.3), whereas `logistic` reports the odds ratios discussed previously. Because we have emphasized using a linear combination of variables to explain the dependent variable, we will focus on `logit` and show you how to obtain odds ratios after estimation. Some researchers, particularly biostatisticians and others in the medical field, focus almost exclusively on odds ratios and therefore typically use `logistic` instead. Regardless of how the estimated coefficients are reported, both commands fit the same underlying statistical model.

7. In the following, we will refer to living in an apartment or house that the individual owns as *residence ownership*. In this respect, children may also be considered to “own” housing.

At least one category of the dependent variable must be 0, because `logit` takes a value of 0 to represent failure and any other value to represent success. Normally, you use a dependent variable with the values 0 and 1, where the category assigned the value of 1 means success. Here the variable `owner` should be generated with the values of 1 for house owner and 0 for tenant, as follows:⁸

```
. generate owner = renttype == 1 if !missing(renttype)
```

We generate the East–West variable analogously, as we did previously for our linear regression model (page 271):

```
. generate east = inrange(state,11,16) if !missing(state)
```

To control for age, we generate an age variable from the year-of-birth variable, and we center this variable.

```
. generate age = 2009-ybirth
. summarize age, meanonly
. generate age_c = age-r(mean)
```

Because household income is so heavily skewed to the right and we expect that the effect of income on house ownership decreases when household income becomes very large, we create a new variable that is log base-2 of household income.

```
. generate loghhinc = log(hhinc)/log(2)
```

Note that the value 10 of `loghhinc` is equal to a yearly household income of approximately €1,000 ($2^{10} = €1,024$), and 11 is equal to an household income of approximately €2,000.⁹ Every unit increase on `loghhinc` is equal to doubling yearly household income.

8. For details on this command, see page 80. The command `label list` determines the assignment of the values to labels (section 5.6).

9. There is a simple trick to estimate roughly the value of exponents of 2. For this, you need to know that 2^{10} is approximately 1,000. Then 2^{11} is $2^{10} \times 2^1 \approx 1,000 \times 2 = 2,000$. Likewise, $2^{15} \approx 1,000 \times 32 = 32,000$.

Now we are ready to fit the logistic regression model:

. logit owner age_c loghhinc east					
Iteration 0: log likelihood = -3739.5754					
Iteration 1: log likelihood = -3247.7417					
Iteration 2: log likelihood = -3246.6989					
Iteration 3: log likelihood = -3246.6983					
Iteration 4: log likelihood = -3246.6983					
Logistic regression	Number of obs = 5407				
	LR chi2(3) = 985.75				
	Prob > chi2 = 0.0000				
Log likelihood = -3246.6983	Pseudo R2 = 0.1318				
owner	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]
age_c	.0228959	.0017586	13.02	0.000	.019449 .0263428
loghhinc	1.157434	.0438131	26.42	0.000	1.071562 1.243306
east	.088873	.0692819	1.28	0.200	-.046917 .2246631
_cons	-17.17416	.6571883	-26.13	0.000	-18.46223 -15.8861

The results table is similar to the one from linear regression. At the bottom of the output is the coefficient table, which contains the estimated coefficients for the independent variables and the constant along with their standard errors, significance tests, and confidence intervals. At the top left is the iterations block with results that are related to the maximum likelihood algorithm. At the top right, we see a block describing the model fit. We will discuss each of these blocks along the lines of our explanation of linear regression.

10.3.1 The coefficient table

The following description focuses on the coefficients in the first column of the table of coefficients. For the meaning of the other columns, please refer to chapter 8. We like to stress that if you are interested in inferential statistics for the commands shown in this chapter, you need to `svyset` the dataset as shown on page 220 and to prefix all regression commands with `svy`.

The b coefficients can be found in the first column of the coefficient table.¹⁰ The b coefficients indicate how the predicted values change when the corresponding independent variables increase by one unit, just like in linear regression, although here the predicted values are the logarithmic odds of success, not the mean of the dependent variable. For example, you would interpret the estimated regression coefficient of `age_c` as follows: the logarithmic odds of residence ownership rise on average by 0.023 if age increases by 1 year. Likewise, the log odds of residence ownership increase by 1.157 if log household income increases by one unit (that is, doubled).

10. In the second column, you will find the standard errors of the regression coefficients, which will help you calculate significance tests and confidence interval limits; see chapter 8. In logistic regression, you usually evaluate the significance of the coefficients using a likelihood-ratio test (section 10.5).

From the estimated regression coefficient of `east`, we can say that with every one-unit increase of the variable `east`, the estimated logarithmic chance of residence ownership increases on average by 0.089. Because `east` can increase by one unit only once, we might instead say that East Germans have, on average, a 0.089 bigger estimated logarithmic chance of residence ownership than West Germans. The estimated regression constant provides the predicted value for those individuals surveyed for whom all independent variables show the value 0. Because we centered the age variable, the value 0 on `age_c` corresponds to the mean of age; because we use the log base-2 of household income, the value 0 on `loghhinc` means $2^0 = €1$. Hence, the interpretation of the regression coefficient is that the predicted logarithmic chance of residence ownership for West German individuals with mean age and household income of €1 is -17.174 .

Because these changes in the logarithmic chance of positive outcome are hard to understand, we will discuss some alternative interpretations below.

Sign interpretation

As a first step, consider just the signs and relative sizes of the estimated coefficients. A positive sign means that the estimated probability or chance of residence ownership increases with the respective independent variable, whereas a negative sign means that it decreases. Here the estimated probability of house ownership increases with age and household income, and is higher in the East than it is in the West.

Interpretation with odds ratios

Using the model equation, we want to calculate the predicted logarithmic chance of a West German with a log household income of 15 (that is, $2^{15} \approx €32,000$). Using the saved coefficients as shown in section 9.1.2, this can be done as follows:

```
. display _b[_cons] + _b[loghhinc]*15
.18734942
```

The estimated logarithmic odds of house ownership for West Germans at average age with a household income of approximately €32,000 is around 0.187.

By calculating the exponential of the regression function, you can convert the logarithmic odds to odds:

```
. display exp(_b[_cons] + _b[loghhinc]*15)
1.2060486
```

The odds that households with a log household income of 15 have house ownership is around 1.206.

Similarly, you can calculate the predicted odds for those who have a log household income that is one unit larger:

```
. display exp(_b[_cons] + _b[loghhinc]*16)
3.8373555
```

A household with a log household income of 16 has a substantially larger odds of residence ownership than the household with a log household income of 15. We can use the *odds ratio* (page 348) to compare the two odds. Here it amounts to

```
. display exp(_b[_cons] + _b[loghhinc]*16)/exp(_b[_cons] + _b[loghhinc]*15)
3.1817586
```

This means that if the log household income increases by one unit (doubles), a person is around 3.182 times as likely to own his or her residence. Increasing log household income by two units increases the odds of owning a residence by around $3.182 \times 3.182 = 10.125$. Odds ratios work in a multiplicative fashion.

You can reduce the complexity of calculating odds ratios if you consider that to determine the odds ratios, you must first calculate the odds for a particular value X and then for the value $X + 1$. After that, you divide both results by each other, which can be presented as follows:

$$\widehat{\text{odds ratio}} = \frac{e^{b_0+b_1(X+1)}}{e^{b_0+b_1X}} = \frac{e^{b_0+b_1X} e^{b_1}}{e^{b_0+b_1X}} = e^{b_1}$$

You can therefore obtain the estimated odds ratio simply by computing the exponential of the corresponding b coefficient.

Many logistic regression users prefer the interpretation of results in the form of the odds ratio. For this reason, Stata also has the command `logistic` and the `logit` option `or` that both directly report the estimated odds ratios. If you have already fit your model using `logit`, you can type in `logistic` or `logit`, `or` to redisplay the output in terms of odds ratios:

<code>. logit, or</code>						
Logistic regression						
Number of obs = 5407						
LR chi2(3) = 985.75						
Prob > chi2 = 0.0000						
Pseudo R2 = 0.1318						
owner	Odds Ratio	Std. Err.	z	P> z	[95% Conf. Interval]	
age_c	1.02316	.0017994	13.02	0.000	1.019639	1.026693
loghhinc	3.181759	.1394028	26.42	0.000	2.919937	3.467057
east	1.092942	.0757211	1.28	0.200	.9541666	1.251901
_cons	3.48e-08	2.29e-08	-26.13	0.000	9.59e-09	1.26e-07

Probability interpretation

The third possibility for interpreting the coefficient is provided by (10.2), which we used to show you how to convert predicted logits into predicted probabilities. For example, to compute the estimated probability of residence ownership for mean-aged West Germans with household incomes of around €32,000, you could type

```
. display exp(_b[_cons] + _b[loghhinc]*15)/(1 + exp(_b[_cons] + _b[loghhinc]*15))
.54670084
```

The `predict` command lets you generate a new variable that contains the predicted probability for every observation in the sample. You just enter the `predict` command along with the name of the variable you want to contain the predicted probabilities.

```
. predict Phat
```

Here the name `Phat` indicates that this variable deals with predicted probabilities. You can also calculate the predicted logits with the `xb` option of the `predict` command.

`margins` provides another possibility to access the predicted probabilities. In the following example, we use the command to calculate the predicted probabilities of residence ownership for West German citizens of mean age with log household incomes between 13 and 18 (the option `noatlegend` saves space):

```
. margins, at(loghhinc=(13(1)18) age_c=0 east=0) noatlegend
Adjusted predictions                                         Number of obs     =      5407
Model VCE       : OIM
Expression     : Pr(owner), predict()
```

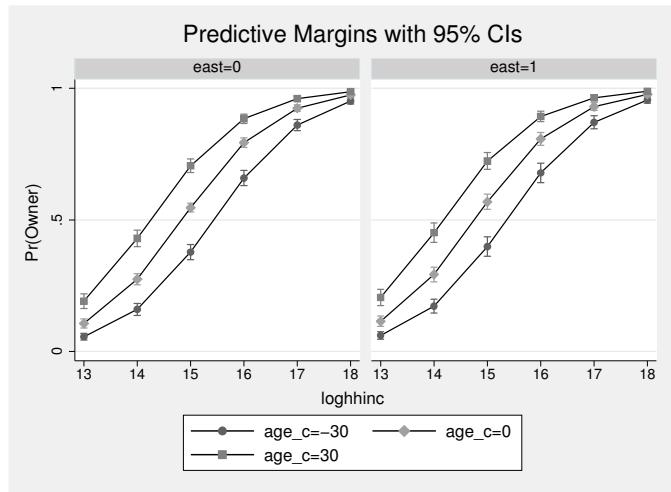
	Delta-method					
	Margin	Std. Err.	z	P> z	[95% Conf. Interval]	
-at						
1	.1064508	.0088783	11.99	0.000	.0890497	.1238519
2	.2748636	.0109781	25.04	0.000	.2533469	.2963803
3	.5467008	.0085653	63.83	0.000	.5299132	.5634885
4	.7932755	.0092688	85.59	0.000	.775109	.811442
5	.9242971	.0066506	138.98	0.000	.9112623	.937332
6	.9749045	.0033467	291.30	0.000	.9683451	.981464

In this output, the first row shows the predicted probability of residence ownership for West German citizens with mean age and yearly log household income of 13 (\approx €8,000). Not surprisingly, this probability is fairly small (around 10.6%). The following rows in the table show the predicted probabilities for the corresponding persons with log households incomes of 14, 15, ..., 18. We see that the predicted probabilities increase when log household income becomes larger. The predicted probability of residence ownership for average-aged West German citizens with a log household income of 18 (\approx €256,000) is almost 100%.

One difficulty in interpreting predicted probabilities is that they do not increase at the same rate for each unit increase in an independent variable. You can see that very well from the table above. If you are comparing mean-aged West Germans with log household income of 13 with those having a household income of 14, the predicted probability of residence ownership increases by around $0.27 - 0.11 = 0.16$. However, if you compare the last two lines, that is, persons with log household incomes of 17 and 18, the predicted probability increases by only $0.97 - 0.92 = 0.05$. We see that an increase of log household income by one unit does not lead to a constant change in the predicted probability.

One solution would be to show the predicted probabilities by means of conditional-effects plot (see section 9.5.3). Like with linear regression, the combination of `margins` and `marginsplot` lets you easily create these plots. Here is an example that shows the effects of all independent variables in the model:

```
. margins, at(loghhinc=(13(1)18) age_c=(-30(30)30)) by(east)
(output omitted)
. marginsplot, by(east)
Variables that uniquely identify margins: loghhinc east
```



The graph shows that the increase in predicted probabilities is not constant over log household income values. Depending on income, the predicted probability of residence ownership will rise either rapidly or slowly. It is also visible that the effect of age on the predicted probability of residence ownership is larger for log household incomes around 14 than for log household incomes above 16. Finally, we see that West and East Germans do not differ much in their predicted probability of residence ownership.

Average marginal effects

As we have seen, the effects of the independent variables on the probability of success is not constant across their levels. Generally, the effects of independent variables are stronger when the predicted probabilities are close to 50% and weaker when the predicted probabilities approach 0% or 100%. Unlike in linear regression, there is therefore not an obvious way to express the influence of an independent variable on the dependent variable with one single number. So-called *average marginal effects* are a reasonable candidate, however.

To introduce average marginal effects, we need to introduce *marginal effects* first. A marginal effect is the slope of a regression line at a specific point. With linear regression, the slope of the regression line is just the regression coefficient. However, as you can see in the conditional-effects plot above, the slope of the regression line—and hence the marginal effect—is not constant in the case of logistic regression. It is therefore only possible to calculate the slope of the regression line at specific points. This can be done with the option `dydx(varlist)` of `margins`. For example, the command

Conditional marginal effects						Number of obs = 5407
	dy/dx	Std. Err.	z	P> z	[95% Conf. Interval]	
Model VCE : OIM						
Expression : Pr(owner), predict()						
dy/dx w.r.t. : age_c loghhinc east						
age_c	.0057079	.0004387	13.01	0.000	.0048481	.0065678
loghhinc	.2885477	.0109437	26.37	0.000	.2670983	.309997
east	.022156	.0172725	1.28	0.200	-.0116974	.0560094

shows the slopes of the regression lines for all independent variables at their means. Holding all variables at their averages, the predicted probabilities of residence ownership increase by 0.006 with age, by 0.022 with East Germany, and by 0.289 with log household income.

If you fix the covariates at other points, the marginal effects change as well. Take as an example the marginal effects for East Germans at an age of 20 years above the average and a log household income of 15:

Conditional marginal effects							Number of obs = 5407
	Model VCE : OIM			Expression : Pr(owner), predict()			dy/dx w.r.t. : age_c loghhinc east
	Delta-method						
	dy/dx	Std. Err.	z	P> z	[95% Conf. Interval]		
age_c	.0050171	.0003387	14.81	0.000	.0043531	.005681	
loghhinc	.2536225	.0096303	26.34	0.000	.2347475	.2724976	
east	.0194743	.0148274	1.31	0.189	-.0095869	.0485354	

Because the marginal effects differ with the levels of the covariates, it makes sense to calculate the average of all marginal effects of the covariate patterns observed in the dataset. This is the average marginal effect. You get the average marginal effects by simply stating the command without specifying the values for which the marginal effects should be obtained.

Average marginal effects							Number of obs = 5407
	Model VCE : OIM			Expression : Pr(owner), predict()			dy/dx w.r.t. : age_c loghhinc east
	Delta-method						
	dy/dx	Std. Err.	z	P> z	[95% Conf. Interval]		
age_c	.0047528	.0003443	13.80	0.000	.004078	.0054277	
loghhinc	.2402647	.0066963	35.88	0.000	.2271402	.2533891	
east	.0184486	.0143741	1.28	0.199	-.009724	.0466212	

10.3.2 The iteration block

In the upper left part of the `logit` output (see page 356) are several rows beginning with the word *iteration*. This sort of output is typical for models whose coefficients are estimated by maximum likelihood. As we mentioned in our discussion of this procedure, when you use the maximum likelihood principle, there is typically no closed-form mathematical equation that can be solved to obtain the b coefficients. Instead, an iterative procedure must be used that tries a sequence of different coefficient values. As the algorithm gets closer to the solution, the value of the likelihood function changes by less and less.

The first and last figures of the iteration block are in some respects similar to the figures given in the ANOVA block of the linear regression (see section 9.1.2), which contained figures for the total sum of squares (TSS), the residual sum of squares (RSS), and the model sum of squares (MSS). TSS is the sum of the squared residuals from predicting all the values of the dependent variables through arithmetic means. RSS is the sum of the squared residuals from the regression model, and MSS is the difference between TSS and RSS. MSS thus represents how many fewer errors we make when using the regression model instead of the mean for predicting the dependent variable.

In the logistic regression model, the residuals used to estimate the regression coefficients cannot be interpreted in the same way as with linear regression. Two values of the likelihood function are of particular interest, namely, the first and the last. The first likelihood shows how probable it is that all β coefficients of the logistic regression apart from the constant term equal 0 (\mathcal{L}_0).¹¹ The last likelihood represents the maximized value. The larger the difference between the first and last values of the log likelihood, the stronger the advantage of the model with independent variables compared with the null model. In this sense, you can consider TSS analogous to \mathcal{L}_0 , RSS to \mathcal{L}_K , and MSS to $\mathcal{L}_0 - \mathcal{L}_K$.

Other than the first and last log likelihoods, the rest of the figures in the iteration block are of little interest, with one exception. Sometimes the maximum likelihood process delivers a solution for the coefficients that is not optimal. This may occur if the domain where you are searching for the coefficients is not concave or flat. This is a somewhat technical issue, so we will not delve further into it. However, many iterations may indicate a difficult function to maximize, though it is difficult to say how many iterations are too many. You should generally expect more iterations the more independent variables there are in your model.

In general, the logistic regression model's likelihood function is “well behaved”, meaning that it is relatively easy to maximize. However, if you do have problems obtaining convergence, you may want to remove a few independent variables from your specification and try again.

10.3.3 The model fit block

R^2 was used to assess the fit of a linear regression model. R^2 is so commonly used because it has, on one hand, clear boundaries of 0 and 1, and on the other, a clear interpretation of the *share of explained variance*. There is no comparable generally accepted measured value for logistic regression. Instead, many different statistics have been suggested, some of which we will introduce here.

One such measure of fit is reported in the model fit block of `logit`: the pseudo- R^2 (p^2). Nevertheless, it is already a mistake to speak of the pseudo- R^2 . There are

11. This is true for Stata's `logit` and `logistic` commands. However, other maximum likelihood commands use alternative starting values, and then the iteration-zero log likelihood is *not* the value obtained when all the slope parameters are set to 0.

various definitions for pseudo- R^2 (Veall and Zimmermann 1994; Long and Freese 2006). Therefore, you should always indicate which pseudo- R^2 you are referring to. The one reported by Stata is that suggested by McFadden (1973), which is why we refer to it as p_{MF}^2 .

McFadden's p_{MF}^2 is defined in a way that is clearly analogous to the R^2 in linear regression (recall that $R^2 = \text{MSS}/\text{TSS} = 1 - \text{RSS}/\text{TSS}$). p_{MF}^2 is defined as

$$p_{MF}^2 = \frac{\ln \mathcal{L}_0 - \ln \mathcal{L}_K}{\ln \mathcal{L}_0} = 1 - \frac{\ln \mathcal{L}_K}{\ln \mathcal{L}_0}$$

where \mathcal{L}_0 is the likelihood from the model with just a constant term and \mathcal{L}_K is the likelihood of the full model. As is the case in R^2 , p_{MF}^2 lies within the boundaries of 0 and 1; however, interpreting the content is disproportionately more difficult. “The higher, the better” is really all you can say of p_{MF}^2 . In our example (page 356), the value of p_{MF}^2 around 0.08 is what most people would say is rather small.

Besides McFadden's pseudo- R^2 , the likelihood-ratio χ^2 value (χ_L^2) is another indicator of the quality of the overall model. It too is based on the difference between the likelihood functions for the full and constant-only models. However, unlike p_{MF}^2 , this difference is not standardized to lie between 0 and 1. It is defined as

$$\chi_L^2 = -2(\ln \mathcal{L}_0 - \ln \mathcal{L}_K)$$

χ_L^2 follows a χ^2 distribution, and as with the F value in linear regression, you can use χ_L^2 to investigate the hypothesis that the independent variables have no explanatory power or, equivalently, that all the coefficients other than the constant are 0. The probability of this hypothesis being true is reported in the line that reads “Prob > chi2”. Here it is practically 0. Therefore, we can assume that at least one of the two β coefficients in the population is not 0. As is the case in the linear regression F test, rejection of this null hypothesis is not sufficient for us to be satisfied with the results.

As with linear regression, you should not judge a model's suitability purely by the measured values within the model fit block, especially in logistic regression, because there is no single generally accepted measured value for doing so. Therefore, we will discuss other measures that are not reported in the output.

Classification tables

The fit of the linear regression model was assessed primarily on the basis of the residuals ($y - \hat{y}$). In logistic regression, one way to assess fit is with a classification table, in which every observation is assigned one of the two outcomes of the dependent variable. The positive outcome is normally assigned when the model predicts a probability of more than 0.5, whereas an observation is assigned a negative outcome if a probability of less than 0.5 is predicted. For example, you could do this manually, assuming you have already created the variable Phat containing the predicted probabilities, by typing

```
. generate ownerhat = Phat >= .5 if !missing(Phat)
```

The classified values so generated are typically presented in a classification table, a simple cross-classified table containing the classified values and the original values:

```
. tabulate ownerhat owner, cell column
```

Key	
	frequency
	column percentage
	cell percentage
ownerhat	owner
	0 1
0	1,556 707 60.92 24.78 28.78 13.08
1	998 2,146 39.08 75.22 18.46 39.69
Total	2,554 2,853 100.00 100.00 47.24 52.76
	Total

The sensitivity and the specificity of the model are commonly used by people in the medical profession. Sensitivity is the share of observations classified as residence owners within the observations who actually do own their residences. Specificity is the share of observations classified as tenants among those who are actual tenants. Here the sensitivity is 75.22% and the specificity is 60.92%.

The count R^2 is commonly used in the social sciences. It deals with the share of overall correctly predicted observations, which you can determine by adding the overall shares in the main diagonal of the above-generated table. However, it is easier to use the `estat classification` command, which you can use to generate the table in a different order, as well as derive the sensitivity, specificity, count, R^2 , and other figures:

True			
Classified	D	~D	Total
+	2146	998	3144
-	707	1556	2263
Total	2853	2554	5407

Classified + if predicted $\text{Pr}(D) \geq .5$
 True D defined as owner != 0

Sensitivity	$\text{Pr}(+ D)$	75.22%
Specificity	$\text{Pr}(- \sim D)$	60.92%
Positive predictive value	$\text{Pr}(D +)$	68.26%
Negative predictive value	$\text{Pr}(\sim D -)$	68.76%
False + rate for true ~D	$\text{Pr}(+ \sim D)$	39.08%
False - rate for true D	$\text{Pr}(- D)$	24.78%
False + rate for classified +	$\text{Pr}(\sim D +)$	31.74%
False - rate for classified -	$\text{Pr}(D -)$	31.24%
Correctly classified		68.47%

The classification table shows that we have a total of 3,144 observations classified as 1. For 2,146 observations, this corresponds to the true value, but for 998 it does not. We have assigned the value 0 to 2,263 observations, which turned out to be correct for 1,556 of the observations. In total, we correctly classified $R^2_{\text{count}} = (2146 + 1556)/5407 = 68.47\%$ of the observation, as shown in the final line of the table.

Overall, we might think that our model looks good. However, you can also correctly classify some cases without knowing about the independent variable. If you know only the distribution of the dependent variable, you will make fewer errors if you assign all observations to the most-frequent category. If you were to predict that all the observations are tenants, you would already be correct in $2853/5407 = 52.77\%$ of the cases. By comparing the correct classification obtained from the marginal distribution with the correct classification obtained with the knowledge of the independent variable, you can calculate the adjusted count R^2 (Long 1997, 108):

$$R^2_{\text{AdjCount}} = \frac{\sum_j n_{jj} - \max_c(n_{+c})}{n - \max_c(n_{+c})}$$

where n_{+c} is the sum of column c and $\max_c(n_{+c})$ is the column with the higher value of n_{+c} . $\sum_j n_{jj}$ is the sum of cases in the main diagonal of the classification table, that is, the amount of correctly classified cases. Here we receive an R^2_{AdjCount} of

```
. display ((2146 + 1556) - 2853)/(5407 - 2853)
.33241973
```

This means that when predicting with a model that includes independent variables, our error rate drops by 33.24% compared with prediction based solely on the marginal

distribution of the dependent variable. You can receive the adjusted count R^2 and other model fit–measured values through Scott Long and Jeremy Freese’s ado-package **fitstat**, available from the SSC archive (see section 13.3.2).

Two further common fit statistics, the Akaike information criterion and the Bayesian information criterion, are available with the **estat ic** command.

Pearson chi-squared

A second group of fit statistics is based on the Pearson residuals. For you to understand these, we must explain the term *covariate pattern*, which is defined as every possible combination of a model’s independent variables. In our example, this is every possible combination of the values of log household income, age, and region. Every covariate pattern occurs m_j times, where j indexes each covariate pattern that occurs. By typing

```
. predict cpatt, number
. sort east age_c loghhinc
. list east age_c loghhinc cpatt, sepby(loghhinc)
```

you can view the index number representing the covariate pattern of each observation.

The Pearson residuals are obtained by comparing the number of successes having covariate pattern j (y_j) with the predicted number of successes having that covariate pattern ($m_j \hat{P}_j$, where \hat{P}_j is the predicted probability of success for covariate pattern j). The Pearson residual is defined as

$$r_{P(j)} = \frac{(y_j - m_j \hat{P}_j)}{\sqrt{m_j \hat{P}_j (1 - \hat{P}_j)}}$$

(Multiplying \hat{P}_j by the number of cases with that covariate pattern results in the predicted number of successes in pattern j .) Unlike residuals in linear regression, which are in general different for each observation, the Pearson residuals for two observations differ only if those observations do not have the same covariate pattern. Typing

```
. predict pres, resid
```

generates a variable containing the Pearson residuals. The sum of the square of this variable over all covariate patterns produces the Pearson chi-squared statistic. You can obtain this statistic by typing

```
. estat gof
Logistic model for owner, goodness-of-fit test
    number of observations =      5407
    number of covariate patterns =   5231
        Pearson chi2(5227) =      6400.84
        Prob > chi2 =            0.0000
```

This test is for the hypothesis of the conformity of predicted and observed frequencies across covariate patterns. A small χ^2 value (high p -value) indicates small differences between the observed and the estimated frequencies, whereas a large χ^2 value (low p -value) suggests that the difference between observed and estimated values cannot be explained by a random process. Be careful when interpreting the p -value as a true “significance” level: a p -value less than 0.05 may indicate that the model does not represent reality, but values greater than 5% do not necessarily mean that the model fits the data well. A p -value of, say, 6% is still fairly small, even though you cannot formally reject the null hypothesis that the difference between observed and estimated values is completely random at significance levels below 6%.

The χ^2 test is unsuitable when the number of covariate patterns (here 5,231) is close to the number of observations in the model (here 5,407). Hosmer and Lemeshow (2000, 140–145) have therefore suggested modifying the test by sorting the data by the predicted probabilities and dividing them into g approximately equal-sized groups. They then suggest comparing the frequency of the observed successes in each group with the frequency estimated by the model. A large p -value indicates a small difference between the observed and the estimated frequencies.

You can obtain the Hosmer–Lemeshow test by using `estat gof` together with the `group()` option. Enter the number of groups into which you want to divide the data parentheses; $g = 10$ is often used.

```
. estat gof, group(10)
Logistic model for owner, goodness-of-fit test
  (Table collapsed on quantiles of estimated probabilities)

    number of observations =      5407
    number of groups =          10
    Hosmer-Lemeshow chi2(8) =    19.76
    Prob > chi2 =            0.0113
```

10.4 Logistic regression diagnostics

We now discuss two methods to test the specification of a logistic regression model. First, logistic regression assumes a linear relationship between the logarithmic odds of success and the independent variables. Thus you should test the validity of this assumption before interpreting the results.

Second, you need to deal with influential observations, meaning observations that have a strong influence on the results of a statistical procedure. Occasionally, these outliers, as they are also known, turn out to be the result of incorrectly entered data, but usually they indicate that variables are missing from the model.

10.4.1 Linearity

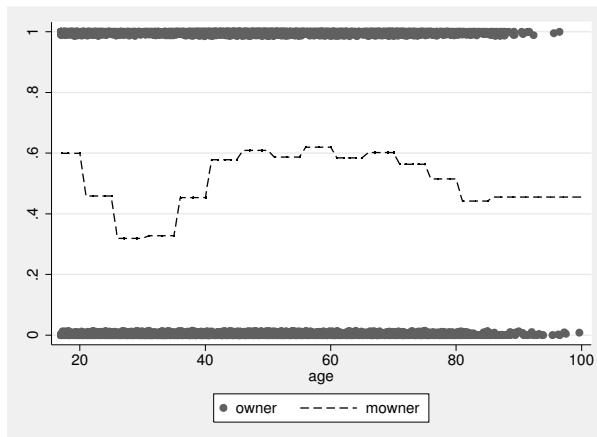
We used graphical analyses to discover nonlinear relationships in the linear regression model, and we used smoothing techniques to make the relationship more visible. You can also use certain scatterplots for logistic regression, but you should consider two issues. First, the median trace used in linear regression as a scatterplot smoother does not work for dichotomous variables because the median can take values of only 0 and 1.¹² Second, the functional form of the scatterplot does not have to be linear, because linearity is assumed only with respect to the logits. The functional form between the probabilities and the independent variable has the shape of an S (see the graph on page 350).

You may use a local mean regression as the scatterplot smoother instead of the median trace. Here the X variable is divided into bands in the same way as for the median trace, and the arithmetic mean of the dependent variable is calculated for each band. These means are then plotted against the respective independent variable.

Ideally, the graph should show the local mean regression to have an S-shaped curve like the illustration on page 350. However, the graph often only depicts a small section of the S-shape, so if the band means range only from about 0.2 to 0.8, the mean regression should be almost linear. U-shaped, reverse U-shaped, and other noncontinuous curves represent potential problems.

Stata does not have a specific command for simple local mean regression, but you can do it easily nonetheless:¹³

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



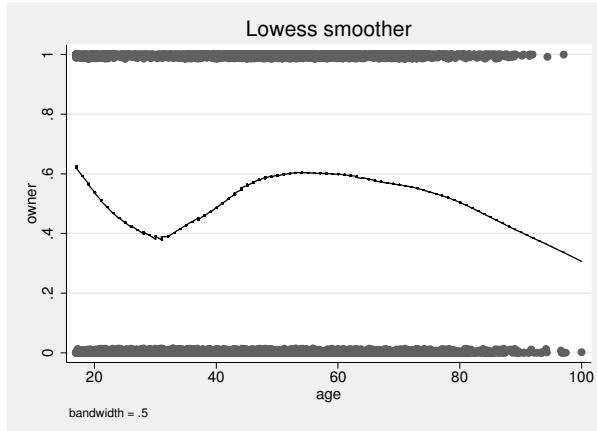
12. The value 0.5 can occur if there is an equal number of 0 and 1 values.

13. For the function `autocode()`, see page 173. For the command `egen`, see section 5.2.2 on page 92.

In this graph, the mean of residence ownership first decreases with age, then increases until the age of 40, and then remains constant until dropping with the oldest individuals surveyed. This correlation certainly does not match the pattern assumed by logistic regression.

Cleveland's (1979) locally weighted scatterplot smoother (LOWESS)¹⁴ is an alternative that is often better for investigating functional forms. You can use this smoother with the `twoway` plottype `lowess` or with the statistical graph command `lowess`. We will not discuss the calculation of this smoother, but we refer you to the excellent explanation of the logic behind LOWESS in Cleveland (1994). You can adjust the level of smoothing by specifying a value between 0 and 1 in the `bwidth()` option, with higher numbers specifying increased smoothing. And LOWESS is a computationally intensive process, so it may take some time to display the following graph on your screen:

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



This graph also displays the double U-shaped correlation between residence ownership and age. The middle-age groups have a higher predicted probability of residence ownership than the upper- and lower-age groups. The youngest individuals surveyed, who presumably still live with their parents, are also likely to live in their own houses or apartments.

Both graphs show a correlation that contradicts the S-shaped correlation required by logistic regression. As with linear regression, U-shaped relationships can be modeled through the generation of polynomials.

14. The process has recently also become to be known as *loess*. We use the older term, because it corresponds to the name of the Stata plottype.

Nevertheless, before you do this, check whether the U-shaped relationship is still visible after controlling for household income. You can do this by replacing the age variable in the regression with a set of dummy variables (see section 9.4.1.):¹⁵

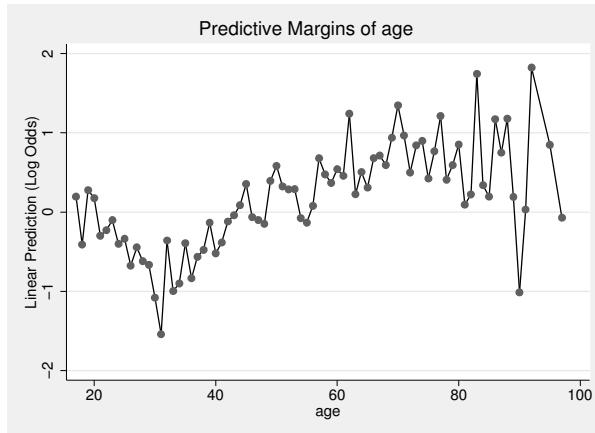
```
. logit owner i.age loghhinc east
```

Fitting this model yields a huge number of coefficients, namely, one for each category of age. Because some of the age categories have very few cases, we run into the problem of high discrimination, indicated by a series of warnings about dropped cases. Some b coefficients can therefore not be estimated. In our particular case, there is nothing to worry about.¹⁶ Each b coefficient indicates how much higher the predicted logarithmic chance of residence ownership is for the respective age group compared with the youngest surveyed individual. When the correlation between age and (the predicted logarithmic chance of) residence ownership is linear, the age b coefficients should increase continuously and steadily. The simplest way to evaluate this is to graphically show the effect of age on the logarithmic chance of residence ownership.

To obtain a plot of the estimated age effects, we again use a combination of `margins` and `marginsplot`.

By default, `margins` produces averages of the predicted probabilities. To obtain the average marginal effects on the linear prediction (that is, the estimated logarithmic chance), we additionally state the option `predict(xb)` with `margins`. We also use the `marginsplot` option `noci` to suppress the display of the confidence intervals.

```
. margins age, predict(xb)
. marginsplot, noci
```

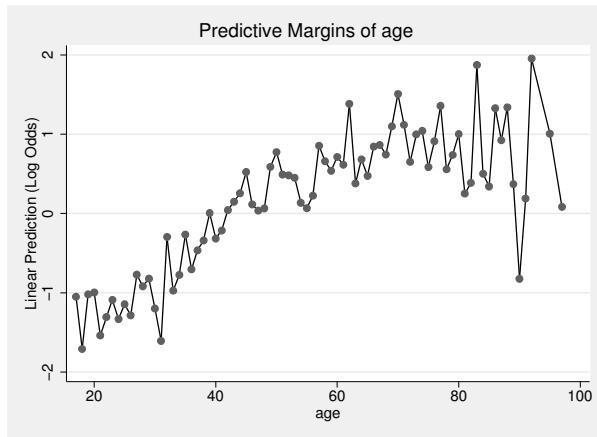


15. For the following process, see Hosmer and Lemeshow (2000, 90). Fox (1997) demonstrates a process related to the component-plus-residual plot (page 285).

16. For further discussion of this problem, see
http://www.ats.ucla.edu/stat/mult_pkg/faq/general/complete_separation_logit_models.htm.

The graph shows by and large the same curvilinear relationship between residence ownership and age as the bivariate figures above. However, if we add dummy variables for the categories of the relationship to the household head (`rel2head`), the decrease at the beginning of the plot vanishes:

```
. logit owner i.age loghhinc east i.rel2head
. margins age, predict(xb)
. marginsplot, nocl
```



When we control for the relationship to the household head, the higher probability of residence ownership of children living at their parents' house is captured by the respective category of the `rel2head` variable; the estimated coefficients of age are therefore net of that nuisance.

The insertion of `rel2head` does not solve the problem of decreasing probability at high ages, however. We will deal with this in section 10.6.1.

10.4.2 Influential cases

Influential data points are observations that heavily influence the b coefficients of a regression model. That is, if we were to remove an influential data point and then refit our model, our coefficient estimates would change by more than a trivial amount. As explained on page 290, influential observations are observations that exhibit an unusual combination of values for the X variable (leverage), as well as an unusual characteristic (given the X values) of the Y variable (discrepancy). Correspondingly, the measured value of Cook's D is calculated by multiplying leverage and discrepancy.

This concept is somewhat more problematic in logistic regression than it is in linear regression, because you can measure only the approximate leverage and discrepancy (Fox 1997, 459). In Stata, you can approximate the leverage values by typing

```
. logit owner age_c loghhinc east i.rel2head
. predict leverage, hat
```

You must refit the original model with `age_c`, `loghhinc`, and `east` as independent variables, because `predict` always refers to the last model fit, and we just fit a model including dummy variables to control for age. Because we found out that controlling for `rel2head` is reasonable, we keep this variable in the model. After getting the predicted leverage values, you can obtain the standardized residuals as an approximation to discrepancy by typing

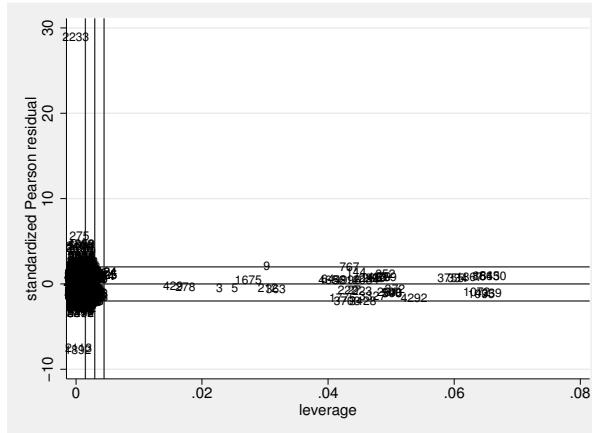
```
. predict spres, rstandard
```

In logistic regression, the standardized residuals for observations having the same covariate pattern are identical. The same also applies for the leverage values. To isolate those covariate patterns having high leverage and discrepancy values, you can produce a graph that compares the standardized residuals with the leverage values. Fox (1997, 461) uses a diagram with vertical lines at the mean of the leverage values and at two and three times the mean. To produce this graph, we first calculate the mean of the variable leverage. We save the mean, as well as its doubled and tripled values, in the local macros ‘`a`’, ‘`b`’, and ‘`c`’ (see chapter 4) to later use them as vertical lines in the graph.

```
. summarize leverage
. local a = r(mean)
. local b = 2 * r(mean)
. local c = 3 * r(mean)
```

Next we generate the graph with the standardized residuals against the leverage values. To generate vertical lines, we use the `xline()` option. We use the number of covariate patterns as the plot symbol. We must therefore create a new variable (`cpatt2`) holding the covariate pattern because the `cpatt` variable that we created on page 367 does not identify the covariate pattern of the model with `rel2head`.

```
. predict cpatt2, number
. scatter spres leverage, xline(`a' `b' `c') yline(-2 0 2)
> mlabel(cpatt2) mlabpos(0) ms(i)
```



No covariate pattern is particularly conspicuous; that is, there is no covariate pattern with both high leverage *and* high discrepancy. However, there is pattern 2233 with very large positive discrepancy and two others with outstanding negative discrepancies. There are also several patterns with leverages higher than three times the mean. When listing the observations with high absolute discrepancy with

```
. list cpatt2 hhnr2009 owner age hhinc east rel2head if abs(spres)>6 &
> !missing(spres)
```

	cpatt2	hhnr2009	owner	age	hhinc	east	rel2head
1462.	1892	140421	0	41	441428	0	Partner
1649.	2113	140421	0	43	441428	0	Head
1748.	2233	156321	1	45	583	0	Head

we see that two of these patterns stem from observations of the same household that have huge household incomes but do not own a residence. Pattern 2233 stems from an observation who owns a residence but has an incredible low yearly household income. If we had the chance to check the original questionnaires of these observations, we should definitely do so. Perhaps there was an error during data entry.

We might also want to analyze further the covariate patterns with high leverage. This could be done, for example, with

```
. generate high = leverage > `c' if !missing(leverage)
(4 missing values generated)
. tabstat owner age hhinc east, by(high)
Summary statistics: mean
by categories of: high


| high  | owner    | age      | hhinc    | east     |
|-------|----------|----------|----------|----------|
| 0     | .5273474 | 49.66903 | 37103.64 | .2462199 |
| 1     | .56      | 33.28    | 42113.8  | .48      |
| Total | .5276493 | 49.51748 | 37149.97 | .2483817 |


```

The high leverage patterns tend to be comparably young East Germans with rather high incomes.

In linear regression, the influence of individual observations on the regression result is determined by Cook's D (see section 9.3.1), which involves multiplying the leverage and discrepancy. An analogous measured value for logistic regression is

$$\Delta\beta = \underbrace{\frac{r_{P(j)}^2}{(1 - h_j)^2}}_{\text{Discrepancy}} \times \underbrace{h_j}_{\text{Leverage}}$$

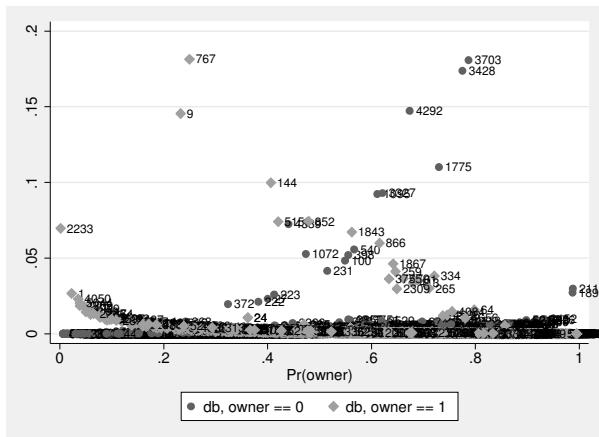
where h_j is the value for the leverage. In Stata, you can obtain this value with

```
. predict db, dbeta
```

as a variable under the name `db`. A scatterplot of $\Delta\beta$ against the predicted probabilities is often used, in which observations with success as the outcome are displayed in a different color or symbol than those of failure. The `separate` command is particularly useful for the latter:¹⁷

17. For an explanation of `separate`, type `help separate`. The variable `Phat` was generated on page 359 with `predict Phat`.

```
. separate db, by(owner)
. scatter db0 db1 Phat, mlab(cpatt2 cpatt2)
```



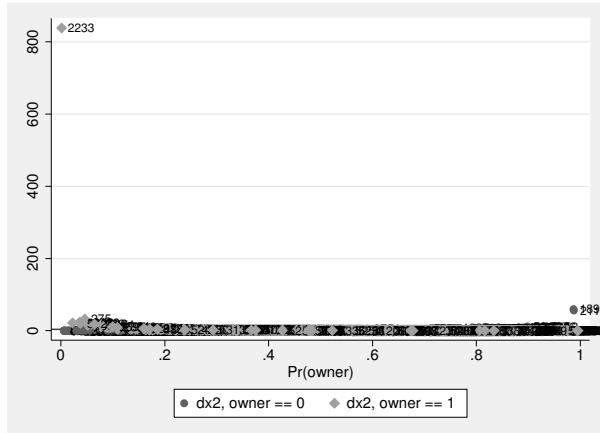
The curve from the bottom left to the top right consists of all tenants, whereas the curve that slopes downward from the top left to the bottom right consists of all residence owners. Far left in the plot, we find again covariate pattern 2233. It stands out for being a residence owner, despite having a very low probability for it. There are also a number of observations that strongly influence the regression result despite having only intermediate predicted probabilities. If we look at those observations with

```
. list hhn2009 owner age hhinc east rel2head if abs(db)>.1 & !missing(db)
```

we do not find anything particularly conspicuous among those covariate patterns—except, perhaps, that they all stem from observations who do not belong to the core family. Should we restrict our model to household heads and their partners?

We postpone the answer to that question and go on introducing a further statistic for influential observations. As shown in section 10.3.3, the sum of the squared Pearson residuals is a measure of the deviation of the predicted values from the observed values. The contribution of each covariate pattern to this measure matches the square of the Pearson residual. If you divide this contribution by $1 - h_j$, you get $\Delta\chi^2_{P(j)}$, which indicates the change in the Pearson chi-squared statistic when the covariate pattern j is removed from the dataset. The scatterplot of $\Delta\chi^2_{P(j)}$ against the predicted probabilities is well suited to the discovery of covariate patterns that are hard to predict through the model. Here it would be useful to enter Hosmer and Lemeshow's raw threshold value of $\Delta\chi^2_{P(j)}$ into the graph (Hosmer and Lemeshow 2000, 163):

```
. predict dx2, dx2
. separate dx2, by(owner)
. scatter dx20 dx21 Phat, yline(4) mlabel(cpatt2 cpatt2)
```



Once again, the conspicuous covariate pattern 2233 stands out. As a first measure, we should therefore deal with the observation that produced this covariate pattern. As we have seen before, the observation has a household income of only €583 per year, which is good for buying only around 13 Big Macs per month but nothing else. The information on household income for this observation is highly untrustworthy, and this seems to be a good reason to remove the observation from the estimation sample:

```
. replace loghhinc = .c if cpatt2==2233
```

However, in the more general case, you should be extremely reluctant to remove observations from the dataset. Normally, if you can eliminate data errors, you should determine if a variable important to the model was left out. This could be a subgroup for which the assumed correlation between age, household income, region, and residence ownership does not hold.

10.5 Likelihood-ratio test

In section 10.3.3, we showed you how to calculate χ^2_L . That statistic compares the likelihood of the fitted model with that of a model in which all the coefficients other than the constant are set to 0. A large value of χ^2_L indicates that the full model does significantly better at explaining the dependent variable than the constant-only model.

You can apply the same principle to determine whether the addition of more independent variables achieves a significant increase in the explanatory power of our model compared with a null model with fewer independent variables. For example, you can ask whether the fit of a model on residence ownership against household income increases if we include an age variable. To answer this question, you can carry out a calculation that is analogous to the test of the overall model by again using -2 times the difference between the log likelihood of the model without age ($\ln \mathcal{L}_{\text{without}}$) and the log likelihood of the model with age ($\ln \mathcal{L}_{\text{with}}$):

$$\chi^2_{\mathcal{L}(\text{Diff})} = -2(\ln \mathcal{L}_{\text{without}} - \ln \mathcal{L}_{\text{with}})$$

Like $\chi^2_{\mathcal{L}}$, this test statistic also follows a χ^2 distribution, in which the degrees of freedom is the difference in the number of parameters between the two models.

You can easily calculate $\chi^2_{\mathcal{L}(\text{Diff})}$ in Stata by using the `lrtest` command. Here we want to investigate the significance of the joint contribution of the `rel2head` dummy variable. First, we calculate the model with the variable we want to investigate:

```
. logit owner age_c loghhinc east i.rel2head
```

We store this model internally using the command `estimates store`, and we name the model `full`:

```
. estimates store full
```

Now we calculate the reduced model.

```
. logit owner age_c loghhinc east
```

Then you can use `lrtest` to test the difference between this model and the previously stored model. You can simply list the name of the stored model (`full`) and, optionally, the name of the model against which it should be compared. If you do not specify a second name, the most recent model is used:

```
. lrtest full
Likelihood-ratio test                               LR chi2(4)      =     109.12
(Assumption: . nested in full)                     Prob > chi2 =    0.0000
```

Negative two times the difference of the log likelihood of the full model and the reduced model is 109.12. The probability of receiving a $\chi^2_{\mathcal{L}(\text{Diff})}$ value of that size or higher in our sample is very small when the coefficient of `rel2head` in the population is 0. You can therefore be fairly certain that the joint effect of all categories of `rel2head` is not 0 in the population. However, this statistic reveals nothing about the degree of influence of `rel2head` on residence ownership; for that, you need to consider the estimated coefficient on age.

When using the likelihood-ratio test, only models that are nested can be compared with one another. This means that the full model must contain all the variables of the reduced model. Furthermore, both models must be calculated using the same set of observations. The latter may be problematic if, for example, some observations in your full model must be excluded because of missing values, while they may be included in the reduced model if you leave out a variable. In such cases, Stata displays a warning message (“observations differ”).

If you wish to compare models not fit to the same sets of observations, an alternative is to use *information criteria* that are based on the log-likelihood function and are valid even when comparing nonnested models. Two of the most common information criteria are the Bayesian information criterion and Akaike’s information criterion, which are obtained through the `estat ic` command mentioned earlier (page 367). An excellent introduction to the statistical foundations of these indices is provided by Raftery (1995).

10.6 Refined models

As with the linear regression model, the logistic regression model might be expanded in various ways to investigate more complicated research questions. In what follows, we will discuss nonlinear relations and investigate varying correlations between subgroups (interaction effects).

10.6.1 Nonlinear relationships

During the diagnosis of our regression model, we saw signs of a U-shaped association between age and the logarithmic chance of residence ownership (section 10.4.1). In this respect, U-shaped associations are only one form of nonlinear relationships. Logarithmic or hyperbolic relationships can also occur. The model assumption of logistic regression is violated only if these relationships appear between the logits and the independent variables. With respect to probabilities, logarithmic or hyperbolic relationships are to a certain extent already taken into account by the S-shaped distribution of the logit transformation.

There are many ways to account for nonlinear relationships. If you have an assumption as to why older people are less likely to own a residence than middle-aged people, you should incorporate the variable in question into the regression model. If, for example, you suspect that the observed decline is a consequence of older people moving into nursing homes to receive full-time care, you might want to introduce some type of variable indicating whether a person is in poor health.

Another way of controlling for a nonlinear relationship is to categorize the independent variable into several groups and use a set of dummy variables instead of the continuous variable. We discussed a strategy like this on page 371. A more common strategy is to use transformations or polynomials of the independent variables. The rules for linear regression apply here, too: for hyperbolic relationships, the X variable

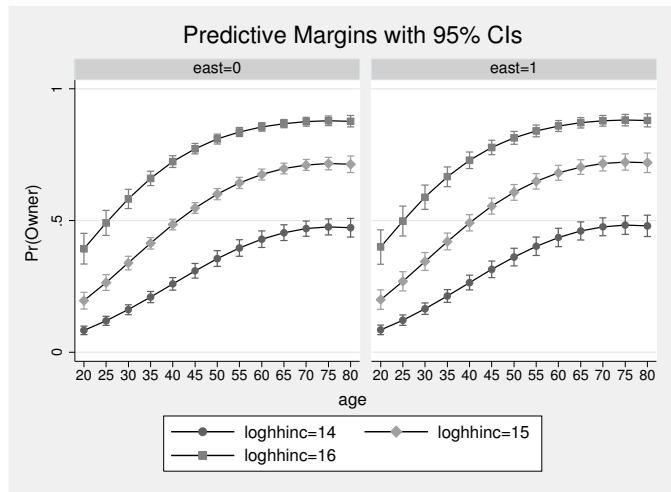
is squared, and for logarithmic relationships, the logarithm of the X variable is used. For U-shaped relationships, we use the squared X variable in addition to the original X variable.

However, instead of generating a squared version of the continuous variable for age, we recommend that you use factor-variable notation (see [U] **11.4.3 Factor variables**) to specify the interaction of age with itself:

. logit owner c.age##c.age loghhinc east i.rel2head, nolog						
Logistic regression						
Number of obs = 5406						
LR chi2(8) = 1156.28						
Prob > chi2 = 0.0000						
Pseudo R2 = 0.1546						
Log likelihood = -3160.7975						
owner	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
age	.1219795	.0129222	9.44	0.000	.0966525	.1473064
c.age#c.age	-.0008032	.0001157	-6.94	0.000	-.0010299	-.0005765
loghhinc	1.073894	.045947	23.37	0.000	.98384	1.163949
east	.0272748	.0708097	0.39	0.700	-.1115096	.1660592
rel2head						
2	.105434	.0684178	1.54	0.123	-.0286625	.2395305
3	1.848891	.1554012	11.90	0.000	1.54431	2.153471
4	.3396665	.5262689	0.65	0.519	-.6918015	1.371134
5	.8898796	.4800361	1.85	0.064	-.0509738	1.830733
_cons	-19.99336	.7447019	-26.85	0.000	-21.45295	-18.53377

It would be best to display the results of this regression model in a conditional-effects plot. In such a plot, we can inspect the curvilinearity of the correlation between age and the probability of home ownership that is induced both by the general functional form of the logistic regression and by the significant `age#age` coefficient.

```
. margins, at(age = (20(5)80) loghhinc=(14(1)16)) over(east)
. marginsplot, by(east)
```



10.6.2 Interaction effects

Let us continue our investigation into the *Titanic* catastrophe (see section 10.1). You want to see whether the seafaring principle of *women and children first* was put into practice or whether, as shown in the film *Titanic*, the first-class gentlemen took their places in the lifeboats at the expense of the third-class women and children.

The above research question can be translated into expectations about the effects of logistic regression models. At the outset, *women and children first* means that, controlling for age and gender, the passenger class should not have any effect on the chance of surviving, while this should not be the case under the Hollywood movie hypothesis. However, the two principles go a little bit further. To start with, *women and children first* implies that

1. women and children should have a higher probability of surviving regardless of their passenger class, and
2. gender should not have an effect for children.

The Hollywood movie hypothesis, on the other hand, means that

1. the effect of gender might differ between social classes, either because male first-class men survive more often or because lower-class women survive less often.

To decide which of the two principles was used, we must therefore also analyze whether the effect of gender varies with social class and whether the effect of gender varies with age. Effects of independent variables that vary between subgroups are called interaction effects.

Now let us load the original dataset:¹⁸

```
. use titanic, clear
```

This dataset contains the dummy variables `survived`, `adult`, and `men`, as well as the categorical variable `class` with categories for crew (0), first-class passengers (1), second-class passengers (2), and third-class passengers (3).

To check whether social class had an effect on the chance of surviving, we can set up a multiple logistic regression of `survived` on the categories of `class` controlling for the dummy variables for `men` and `adult`. To use the independent variable `class` in the regression model, you create dummy variables for each of its categories or you apply factor-variable notation. If you apply factor-variable notation, Stata's default is to choose the value with the lowest level as the base category. Here we want to choose the third-class passengers (`class==3`) as the base category. This can be done with the `ib`. operator:

Logistic regression						
Number of obs = 2201						
LR chi2(5) = 559.40						
Prob > chi2 = 0.0000						
Pseudo R2 = 0.2020						
Log likelihood = -1105.0306						
survived	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
adult	-1.061542	.2440257	-4.35	0.000	-1.539824	-.5832608
men	-2.42006	.1404101	-17.24	0.000	-2.695259	-2.144862
class						
0	.9200861	.1485865	6.19	0.000	.6288619	1.21131
1	1.777762	.1715666	10.36	0.000	1.441498	2.114027
2	.7596673	.1763533	4.31	0.000	.4140211	1.105313
_cons	1.327618	.2480381	5.35	0.000	.8414719	1.813763

According to the signs associated with the `adult` and `men` coefficients, adults were less likely to survive than children, and men were less likely to survive than women. So far, this supports the principle of *women and children first*. However, it also becomes apparent that the first-class passengers have the largest estimated chance of survival compared with the rest. The third-class passengers had the smallest chances of survival; in fact, their chances of survival were even smaller than those of the crew. In conclusion, you can state that women and children were indeed favored for rescue, but apparently passenger class also played a role.

18. The data were collected by the British Board of Trade in their investigation of the sinking of the *Titanic* and made publicly available by Dawson (1995).

Notice as an aside that all coefficients would be considered significant by usual standards. However, the significance tests do not refer to a population but to a data-generating principle. Basically, the test asks whether the observed pattern could have been created just by chance. The answer to this? No, this is very unlikely (see section 8.3).

Checking for the interaction terms could be also done with factor-variable notation:

. logit survived i.men##i.adult i.men##ib3.class, nolog						
Logistic regression						
Number of obs = 2201						
LR chi2(9) = 634.70						
Prob > chi2 = 0.0000						
Pseudo R2 = 0.2292						
Log likelihood = -1067.3785						
survived	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
1.men	-.226458	.4241198	-0.53	0.593	-1.057718	.6048015
1.adult	-.1803451	.3617946	-0.50	0.618	-.8894496	.5287593
men#adult						
1 1	-1.358117	.4551	-2.98	0.003	-2.250096	-.4661373
class						
0	2.089406	.6381427	3.27	0.001	.8386694	3.340143
1	3.753608	.5298566	7.08	0.000	2.715108	4.792108
2	2.13911	.3295635	6.49	0.000	1.493178	2.785043
men#class						
1 0	-1.562327	.6562461	-2.38	0.017	-2.848546	-.2761087
1 1	-2.665672	.5673195	-4.70	0.000	-3.777598	-1.553746
1 2	-2.327067	.4140798	-5.62	0.000	-3.138648	-1.515485
_cons	-.0119411	.3362455	-0.04	0.972	-.6709701	.647088

Before starting to interpret the output, you should be aware of one pitfall of logistic regression or, more generally, of all nonlinear probability models: You cannot really compare the estimated coefficients between groups, and interpretation of the estimated coefficients of interaction terms are therefore error-prone (Allison 1999). The problem basically is that the estimated coefficients of logistic regressions are identified up to an unknown scale parameter and this parameter could be different between groups. To the best of our knowledge, there is no single *best* solution to this problem, although the statistical literature provides a number of techniques for specific situations. We cannot go into the details of all those solutions here, but we strongly recommend that you consider the literature before interpreting interaction terms of logistic regression (see Breen, Karlson, and Holm [2011], Williams [2009], and Wooldridge [2010]).

Let us use average marginal effects as one possible solution. The following shows the average marginal effects of gender for categories of `adult` and `class`:

		Delta-method					
		dy/dx	Std. Err.	z	P> z	[95% Conf. Interval]	
1.men	adult						
	0	-.2509003	.0814881	-3.08	0.002	-.410614	-.0911865
	1	-.5502979	.0327675	-16.79	0.000	-.6145209	-.4860748
	class						
	0	-.6304944	.0713626	-8.84	0.000	-.7703626	-.4906262
	1	-.6201961	.0376232	-16.48	0.000	-.6939362	-.546456
	2	-.7395926	.0408843	-18.09	0.000	-.8197242	-.6594609
	3	-.2949521	.0401417	-7.35	0.000	-.3736283	-.2162758

Note: dy/dx for factor levels is the discrete change from the base level.

The predicted probability of surviving is on average around 25 percentage points lower for male children than for female children. For adults, the difference in the predicted probability of surviving for men is 55 percentage points lower than for women. Thus as expected by *women and children first*, the gender effect is stronger for adults than for children, although the gender effect is not zero among children.

We also see how the gender effect co-varies with class. Obviously, the survival rates of men and women are much more equal among third-class passengers than among all others, which fits the expectation that third-class women did not find a place in the lifeboats. Hence, we found indications of both data-generating principles, *women and children first* and the Hollywood movie mechanism.

Calculating average marginal effects can be a reasonable technique for interpreting interaction terms of logistic regression models, but they cannot be used in all situations. The reason is that the size of the average marginal effects depends on the observed distributions of the independent variables. Strictly speaking, the average marginal effects are only comparable if the distributions of the independent variables are also comparable. As a principle, we would be particularly reluctant to interpret average marginal effects between groups formed from various population samples, because those distributions might be affected by sample selection or other influences.

10.7 Advanced techniques

Stata allows you to fit many related models in addition to the logistic regression we have described above. Unfortunately, we cannot show them in detail. However, we will describe the fundamental ideas behind some of the most important procedures. For

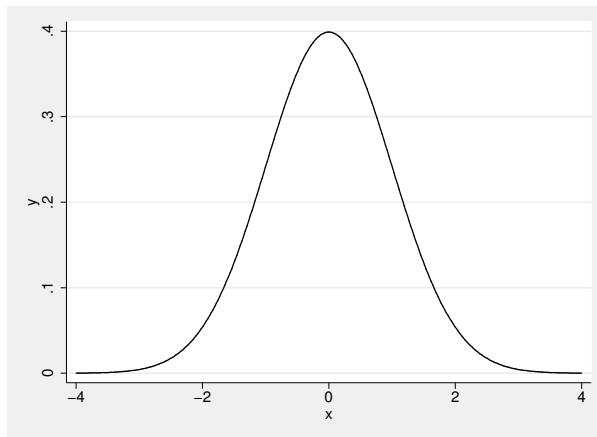
further information, we will specifically refer you to the entry in the *Stata Reference Manual* corresponding to each command. There you will also find references to the literature.

10.7.1 Probit models

In the logistic regression model, we attempted to predict the probability of a success through a linear combination of one or more independent variables. To ensure that the predicted probabilities remained between the limits of 0 and 1, the probability of the success underwent a logit transformation. However, using the logit transformation is not the only way to achieve this. An alternative is the probit transformation used in probit models.

To get some idea of this transformation, think of the density function of the standard normal distribution:

```
. twoway function y = 1/sqrt(2*_pi) * exp(-.5 * x^2), range(-4 4)
```



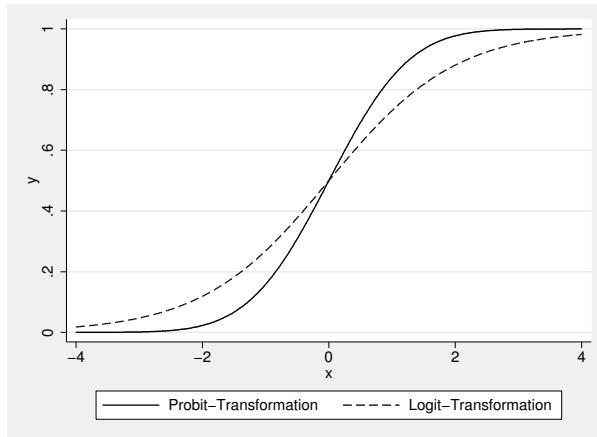
You can interpret this graph in the same way as a histogram or a kernel density estimator (see section 7.3.3); that is, for this variable, the values around 0 occur most often, and the larger or smaller they become, the more rarely they occur.

Suppose that you randomly selected an observation from the variable X . How large would the probability be of selecting an observation that had a value of less than -2 ? Because values under -2 do not occur very often in the X variable, the intuitive answer is, “not very large”. If you want to know the exact answer, you can determine the probability through distribution-function tables for standard normal distribution or through the Stata command

```
. display normal(-2)
.02275013
```

The probability of selecting an observation with a value less than or equal to -2 from a standard normal variate is therefore 0.023. You can repeat the same calculation for any value of X and then show the results graphically. This results in the cumulative density function for the standard normal distribution Φ depicted in the following graph:

```
. twoway || function y = normal(x), range(-4 4)
> || function y = exp(x)/(1+exp(x)), range(-4 4)
> ||, legend(order(1 "Probit-Transformation" 2 "Logit-Transformation"))
```



The function shows an S-shaped curve, similar to the probabilities assigned to the logits, which we have also included in the graph.

As with the logit transform you used with logistic regression, the normal distribution function can also be used to transform values from $-\infty$ and $+\infty$ into values between 0 and 1. Correspondingly, the inverse of the distribution function for the standard normal distribution (Φ^{-1}) converts probabilities between 0 and 1 for a success [$\Pr(Y = 1)$] into values between $-\infty$ and $+\infty$. The values of this probit transformation are thus also suited to be estimated with a linear model. This yields the probit model:

$$\Phi^{-1}\{\Pr(Y = 1)\} = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \cdots + \beta_{K-1} x_{K-1,i} \quad (10.6)$$

You can estimate the b coefficients of this model through maximum likelihood. You can interpret the estimated coefficients the same way as in logistic regression, except that now the value of the inverse distribution function of the standard normal distribution increases by b units instead of the log-odds ratio increasing by b units for each one-unit change in the corresponding independent variable. Using the distribution function for the standard normal distribution, you can then calculate probabilities of success. Usually, the predicted probabilities of probit models are nearly identical to those of logistic models, and the estimated coefficients are often about 0.58 times the value of those of the logit models (Long 1997, 49).

The Stata command used to calculate probit models is `probit`. For example, you can refit the previous model (see page 383) using `probit` instead of `logit`:

Probit regression						
		Number of obs = 2201				
		LR chi2(5) = 556.83				
		Prob > chi2 = 0.0000				
		Pseudo R2 = 0.2011				
Log likelihood = -1106.3142						
survived	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
men	-1.44973	.0808635	-17.93	0.000	-1.608219	-1.29124
adult	-.5803382	.1377535	-4.21	0.000	-.85033	-.3103463
class						
1	.5399101	.0951552	5.67	0.000	.3534092	.7264109
2	-.0898158	.1028857	-0.87	0.383	-.2914681	.1118364
3	-.4875252	.0800342	-6.09	0.000	-.6443893	-.3306611
_cons	1.277019	.1648489	7.75	0.000	.9539214	1.600117

See [R] `probit` for more information on this model.

10.7.2 Multinomial logistic regression

Multinomial logistic regression is used when the dependent variable exhibits more than two categories that cannot be ranked. An example for this would be party preference with values for the German parties CDU, SPD, and all other parties.

The main problem with using multinomial logistic regression is in the interpretation of the estimated coefficients, so this will be the focus point of this section. Nevertheless, to understand this problem, you must at least intuitively grasp the statistical fundamentals of the process. These fundamentals will be discussed shortly (cf. Long 1997).

In multinomial logistic regression, you predict the probability for every value of the dependent variable. You could initially calculate a binary logistic regression¹⁹ for every value of the dependent variable. Here you could calculate three separate logistic regressions: one with the dependent variable CDU against non-CDU, one with the dependent variable SPD against non-SPD, and one with the dependent variable for the other parties against the CDU and SPD together:

19. To differentiate it from multinomial logistic regression, we call the logistic regression of a dichotomous dependent variable a binary logistic regression.

$$\begin{aligned}\ln \frac{\Pr(Y = \text{CDU})}{\Pr(Y = \text{not-CDU})} &= \beta_0^{(1)} + \beta_1^{(1)}x_{1i} + \beta_2^{(1)}x_{2i} + \cdots + \beta_{K-1}^{(1)}x_{K-1,i} \\ \ln \frac{\Pr(Y = \text{SPD})}{\Pr(Y = \text{not-SPD})} &= \beta_0^{(2)} + \beta_1^{(2)}x_{1i} + \beta_2^{(2)}x_{2i} + \cdots + \beta_{K-1}^{(2)}x_{K-1,i} \\ \ln \frac{\Pr(Y = \text{Sonst.})}{\Pr(Y = \text{not-Sonst.})} &= \beta_0^{(3)} + \beta_1^{(3)}x_{1i} + \beta_2^{(3)}x_{2i} + \cdots + \beta_{K-1}^{(3)}x_{K-1,i}\end{aligned}$$

The superscript in parentheses means that the β coefficients differ between the individual regression equations: $\beta_k^{(1)} \neq \beta_k^{(2)} \neq \beta_k^{(3)}$. To simplify the notation, we refer to $\beta_1^{(1)} \dots \beta_{K-1}^{(1)}$ as $\mathbf{b}^{(1)}$ and refer to the sets of b coefficients from the other two equations as $\mathbf{b}^{(2)}$ and $\mathbf{b}^{(3)}$, respectively.

Each of the unconnected regressions allows for a calculation of the predicted probability of every value of the dependent variable. These predicted probabilities do not all add up to 1. However, they should, as one of the three possibilities—SPD, CDU, or other—must occur.²⁰

Therefore, it is sensible to jointly estimate $\mathbf{b}^{(1)}$, $\mathbf{b}^{(2)}$, and $\mathbf{b}^{(3)}$ and to adhere to the rule that the predicted probabilities must add up to 1. However, it is not possible to estimate all three sets of coefficients. To do so, you must constrain one of the coefficient vectors to be equal to a fixed value, 0 being by far the most common choice. After making such a normalization, you can estimate the remaining coefficients by using the maximum likelihood principle. Which one of the three sets of coefficients you constrain to be 0 does not matter. By default, Stata's `mlogit` command constrains the coefficient vector corresponding to the most frequent outcome.

Let us show you an example of interpreting the estimated coefficients. Please load `data1.dta`:

```
. use data1, clear
```

Now generate a new variable for party choice that leaves aside all parties that are not in the German parliament and generate an age variable:

```
. generate party:pib = pib if pib < 6
. generate age = 2009 - ybirth
```

The Stata command for multinomial logistic regression is `mlogit`. The syntax for the command is the same as for all estimation commands; that is, the dependent variable follows the command and is in turn followed by the list of independent variables. With the `baseoutcome()` option, you can select the equation for which the b coefficients are set to 0.

20. Here you disregard the possibility of no party preference. If you did not, you would have to calculate a further regression model for this alternative. The predicted probabilities for the four regression should then add up to 1.

Let us calculate a multinomial logistic regression for party preference against education (in years of education) and year of birth. Here the b coefficients of the equation for the SPD are set at 0:

. mlogit party yedu age, base(1) nolog						
Multinomial logistic regression						
Number of obs = 1948						
LR chi2(8) = 193.62						
Prob > chi2 = 0.0000						
Pseudo R2 = 0.0371						
Log likelihood = -2515.1866						
party	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]	
SPD	(base outcome)					
CDU_CSU						
yedu	.0145588	.0195694	0.74	0.457	-.0237966	.0529141
age	.0098657	.0032362	3.05	0.002	.0035228	.0162086
_cons	-.6472421	.3364861	-1.92	0.054	-1.306743	.0122586
FDP						
yedu	.1797254	.0308604	5.82	0.000	.1192401	.2402108
age	-.0059135	.0059058	-1.00	0.317	-.0174887	.0056618
_cons	-3.61921	.5720866	-6.33	0.000	-4.740479	-2.497941
Greens_B90						
yedu	.2504939	.0275987	9.08	0.000	.1964014	.3045864
age	-.021925	.005449	-4.02	0.000	-.0326048	-.0112451
_cons	-3.478488	.5036535	-6.91	0.000	-4.465631	-2.491346
Linke						
yedu	.1104863	.0318876	3.46	0.001	.0479877	.1729849
age	.0146143	.0058458	2.50	0.012	.0031568	.0260718
_cons	-3.828885	.5941485	-6.44	0.000	-4.993395	-2.664375

In contrast to binary logistic regression, the coefficient table is split into several parts. The first panel contains the estimated coefficients of the equation for the CDU/CSU, followed by the estimated coefficients for FDP, and so on. There is a panel of coefficients for each category of *party* except for the base category.

As a result of setting $\mathbf{b}^{(\text{SPD})} = 0$, you can interpret the estimated coefficients of the other two equations in relation to the SPD supporters. By this, we mean that estimated coefficients in the equation for the CDU/CSU indicate how much the predicted logarithmic chance of preferring the CDU/CSU and not the SPD changes when the independent variables increase by one unit. The equation for the FDP indicates changes in the predicted logarithmic chance of preferring the FDP and not the SPD, and so on.

Interpreting the estimated coefficients for a multinomial logistic regression is not as easy as for binary logistic regression, because the sign interpretation cannot be used. The positive sign for length of education in the CDU/CSU equation does not necessarily mean that the predicted probability of a preference for the CDU/CSU increases with education. We can demonstrate this with the estimated coefficient for the variable *yedu* from the equation for the CDU/CSU. Writing the probability of preferring the CDU/CSU

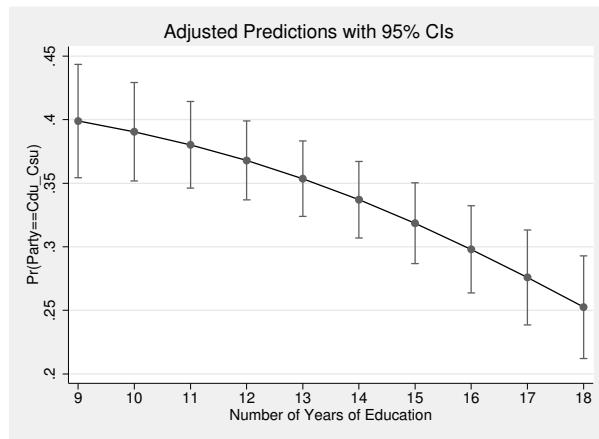
as $P_{\text{CDU/CSU}}$ and the probability of preferring the SPD as P_{SPD} , the b coefficient for yedu in the CDU/CSU equation can be written as

$$\begin{aligned} b_{\text{yedu}}^{(\text{CDU/CSU})} &= \ln \left(\frac{\hat{P}_{\text{CDU/CSU}|\text{yedu+1}}}{\hat{P}_{\text{SPD}|\text{yedu+1}}} \right) - \ln \left(\frac{\hat{P}_{\text{CDU/CSU}|\text{yedu}}}{\hat{P}_{\text{SPD}|\text{yedu}}} \right) \\ &= \ln \left(\frac{\hat{P}_{\text{CDU/CSU}|\text{yedu+1}}}{\hat{P}_{\text{CDU/CSU}|\text{yedu}}} \right) / \left(\frac{\hat{P}_{\text{SPD}|\text{yedu}}}{\hat{P}_{\text{SPD}|\text{yedu+1}}} \right) \end{aligned}$$

The b coefficient for years of education in the equation for the CDU/CSU, on one hand, depends on the change in the predicted probability of CDU/CSU preference with the years of education. On the other hand, it also depends on the respective change in predicted probability for choosing the SPD. In contrast to the binary logit model, in the multinomial logit model, the change in the predicted probability of SPD preference does not completely depend on the change in the probability of CDU/CSU preference. In this respect, the b coefficient can be solely, mainly, or partly dependent on the probability relationship in the base category.

To avoid misinterpreting the multinomial logistic regression, we recommend that you use average marginal effects or the conditional-effects plot for the predicted probabilities. The plot can be created with `margins` and `marginsplot`, as usual. However, when using `margins`, we have to specify the equation on which the command should be applied. We use `predict(outcome(2))` to specify that we want the predicted probabilities for outcome 2, the CDU/CSU. We also fix the age to 40.

```
. margins, at(yedu=(9(1)18) age=40) predict(outcome(2))
. marginsplot
```



Note that the predicted probability of preferring the CDU/CSU decreases with years of education despite the positive effect of `yedu` in the respective equation.

10.7.3 Models for ordinal data

Models for ordinal data are used when the dependent variable has more than two values that can be ordered. An example would be the question regarding concerns about the increase of crime, which respondents could answer with “no concerns”, “moderate concerns”, or “strong concerns”. In a dataset, these could be assigned respective values of 0, 1, and 2, or equivalently, 0, 10, and 12. The difference between two consecutive categories is immaterial—all that matters is that the outcomes can be ordered.

In principle, there are two strategies available for modeling ordinal dependent variables. The first uses multinomial logistic regression, whereby certain constraints are imposed upon the coefficients (stereotype model). The second strategy generalizes binary logistic regression for variables with more than two values (proportional-odds model). Anderson (1984) discusses the underlying assumptions for both models.

The logic behind the stereotype model is simple. In multinomial logistic regression, every value of the dependent variable has its own set of coefficients. The effect of age in the regression model on page 389 was positive on the predicted chance of preferring the CDU/CSU (and not the SPD), and at the same time the effect was negative on the predicted chance of preferring the FDP (and not the SPD). If the dependent variable indicates the presence of ranking, you would normally not expect a directional change in the effects. For example, consider the variable for concerns about increasing crime (`wor09`), which contains the values 1 for strong concerns, 2 for moderate concerns, and 3 for no concerns. First, calculate a multinomial logistic regression for this variable against the length of education. Before you do this, however, you should reverse code the variable `wor09` so that large values stand for strong concerns:

```
. generate worries = 4 - wor09
. mlogit worries yedu, base(1)
```

You will get an estimated coefficient of around -0.116 in the equation for moderate concerns and -0.232 in the equation for strong concerns. The direction of the effects does not change here. This should come as little surprise, because if education reduces the chance of having moderate concerns (and not of having no concerns), it should also reduce the chance of having strong concerns (and not of having no concerns). However, if you calculate a multinomial logistic regression, this assumption is ignored. Nevertheless, you can include such assumptions in the model by imposing constraints on the b coefficients.

Using constraints, you can impose certain structures for the b coefficients before calculating a model. You could, for example, require that education reduces the chance of having moderate concerns (and not of having no concerns) to the same extent that it does for having strong concerns (and not of having moderate concerns). Here the coefficient of education for strong concerns would have to be exactly twice as large as the coefficient of education for moderate concerns. With the `constraint` command, you can set this structure for the `mlogit` command. With

```
. constraint define 1 [3]yedu = 2*[2]yedu
```

you define constraint number 1, which states that the coefficient of the variable `yedu` in the third equation must be twice as large as the coefficient of the variable `yedu` in the second equation. You impose the constraint by specifying the `constraints()` option of the `mlogit` command. Here you would enter the number of the constraint you wish to use in the parentheses.

```
. mlogit worries yedu, base(1) constraints(1)
```

If you calculate this model, you will discover that it is almost identical to the previous model. However, it is far more economical, because in principle only one education coefficient has to be estimated. The other estimate is derived from the ordinal structure of the dependent variable and our assumption that education proportionately increases concerns.

Establishing specific constraints that take into account the ordinal structure of the dependent variable is one way of modeling the ordinal dependent variable. Nevertheless, the constraint is just one example of many alternatives. See [R] **slogit** for more information about this model.

The proportional-odds model follows a different approach: the value of the ordinal variable is understood as the result of categorizing an underlying metric variable. Here you could assume that answers in the `worries` variable provide only a rough indication of the attitudes toward the increase in crime. The attitudes of people probably vary between having infinitely many concerns and no concerns whatsoever, so they might take any value between; that is, attitude is actually a continuous variable E . Instead of observing E , however, all you see are the answers reported on the survey—no concerns, moderate concerns, or strong concerns. Because you have three outcomes in the model, there must also exist two points, κ_1 and κ_2 , that partition the range of E into the three reported answers. That is, if $E < \kappa_1$, then the person reported no concerns; if $\kappa_1 \leq E \geq \kappa_2$, the person reported moderate concerns; and if $E > \kappa_2$, the person reported strong concerns.

Remember the predicted values (\hat{L}) of the binary logistic regression. These values can take on any values from $-\infty$ to $+\infty$. In this respect, you could interpret these predicted values as the unknown metric attitude E . If you knew the value of κ_1 and κ_2 by assuming a specific distribution for the difference between E and \hat{L} , you could determine the probability that each person reported each of the three levels of concern. The proportional-odds model estimates the b 's in the linear combination of independent variables as well as the cutpoints needed to partition the range of E into discrete categories.

An example may clarify this concept. The command for the *proportional odds* model in Stata is `ologit`. The syntax of the command is the same as that for all other model commands: the dependent variable follows the command and is in turn followed by the list of independent variables. We will calculate the same model as above:

```
. ologit worries yedu
Iteration 0:  log likelihood = -4947.8488
Iteration 1:  log likelihood = -4843.702
Iteration 2:  log likelihood = -4843.087
Iteration 3:  log likelihood = -4843.0868
Ordered logistic regression
Number of obs      =      5002
LR chi2(1)        =     209.52
Prob > chi2       =    0.0000
Pseudo R2         =    0.0212
Log likelihood = -4843.0868
```

worries	Coef.	Std. Err.	z	P> z	[95% Conf. Interval]
yedu	-.1496667	.0104568	-14.31	0.000	-.1701617 -.1291717
/cut1	-3.590283	.136148			-3.857128 -3.323438
/cut2	-1.066422	.1241718			-1.309794 -.8230495

The predicted value of this model for respondents with 10 years of education is $S_{10} = -0.150 \times 10 \approx -1.50$. The value for κ_1 and κ_2 are provided beneath the coefficient table. The predicted probability that respondents with a predicted value of -1.50 are classified as individuals with no concerns matches the probability of $-1.50 + u_j \leq -3.590$, or in other words, the probability that $u_j \leq -2.09$. If you assume that the error term follows the logistic distribution, the predicted probability is $1 - 1/(1 + e^{-2.09}) \approx 0.11$.

For more information on ordered logistic regression in Stata, see [R] **ologit**.

10.8 Exercises

- Download a subset of the 1988 National Longitudinal Survey by typing

```
. webuse nlsw88, clear
```

- Create a regression model where **union** is the dependent variable. Create and use the following independent variables for your model:

- **tenure** (centered)
- **age** (centered)
- **collgrad**
- **race**

- Calculate the predicted values for all cases.
- Request the display of
 - the predicted odds of being unionized for respondents with mean age and mean tenure without a college diploma.
 - the predicted odds of being unionized for black respondents with mean age and mean tenure with a college diploma.

- c. the estimated odds ratio of being unionized for college graduates versus non-college graduates.
 - d. the odds ratio for all covariates using the `logistic` command.
 - e. the probability of being unionized for respondents with mean age, mean tenure, and a college diploma.
5. Predict the probability of being unionized as a function of tenure for respondents with mean age and without a college diploma. Display these probabilities graphically.
 6. Predict the respective probability for mean-aged respondents with a college diploma. Add a line for these probabilities to the graph from the last problem.
 7. Investigate the functional form of the effect of tenure. Does the relationship appear linear?
 8. Generate a classification table manually and then using the built-in function.
 9. Perform a likelihood-ratio test between the full model including age and a reduced model that excludes age. What do you conclude from this test?
 10. Produce a plot of $\delta\chi^2$ by predicted probabilities. Label the marker symbol with the covariate pattern. Describe the problem your model suffers from most.
 11. Investigate the correlation of high influence points with the industrial branch. What do you conclude? See how your results change if you reformulate your model by including a variable for an industrial branch with three categories.

11 Reading and writing data

In previous chapters, we asked you to load some data into Stata. Usually, you just needed to type the command `use` followed by a filename, for example,¹

```
. use data1
```

In practice, however, reading data into Stata is not always that easy—either because the data you want to use are in a format other than Stata, such as SAS, SPSS, or Excel, or because they are not available as a machine-readable dataset.

After defining a rectangular dataset in section 11.1, we will explain how to import different forms of machine-readable data into Stata in section 11.2. Machine-readable data are data stored on a hard drive, a CD-ROM, a memory stick, a website, or any other medium that can be read by a machine. However, most types of machine-readable data, such as information in statistical yearbooks or questionnaires, are not in Stata format and therefore cannot be read into Stata with `use`. In section 11.3, we will discuss how to deal with data that are not yet machine readable. Then we will move on to data that are already in Stata format but spread across several data files. Section 11.4 shows you how to combine the information from different files into one rectangular dataset using data from the GSOEP database. In section 11.5, we will show you how to save the Stata files you created.

11.1 The goal: The data matrix

Before you start, look again at `data1.dta`, which we used in previous chapters:

```
. describe
```

1. Make sure that your working directory is `c:\data\kk3` before invoking the command (see page 3 for details).

As you can see, the dataset consists of 5,411 observations and 65 variables. To get an impression of the data, type

```
. browse
```

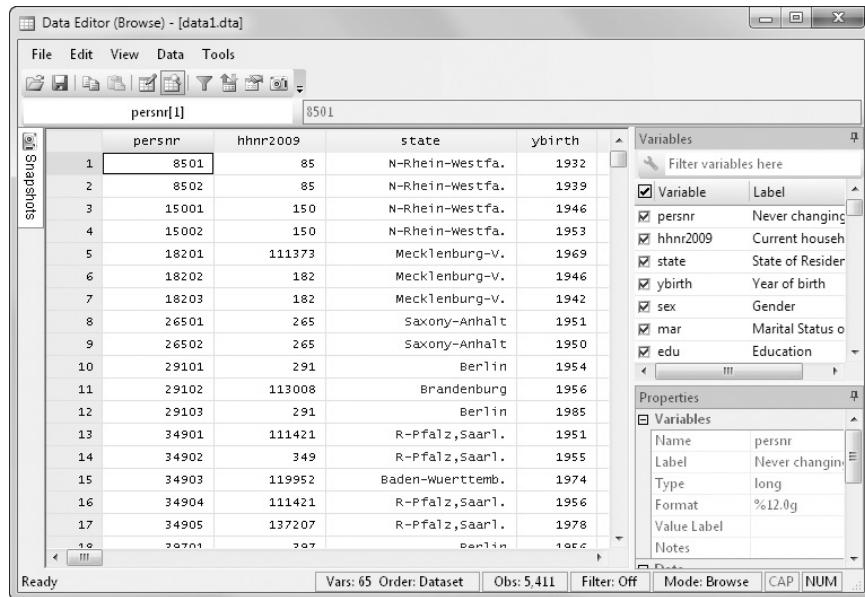


Figure 11.1. The Data Editor in Stata for Windows

This command opens the Data Editor (figure 11.1), which is a separate window with a table containing text or numerical cells. The gray headline (the first row in the table) shows the names of the variables. The first column, which is also gray, displays an observation number for each observation. Correspondingly, each row in this table contains information on one of the 5,411 interviewed persons (cases, observations). Each column in the table contains an entry for each of the 65 variables for all 5,411 persons. We call such a table a data matrix.

The first white column displays the personal identification number, which is unique for each interviewed person. If you look in the fourth column of the row for the person identified by the number 8501, you will find that this person was born in 1932. In column 12, the information about the income of person 8501 is missing. Instead of a specific value, the Data Editor displays a dot representing a missing value. Because dots are used in all empty cells (missings), the data matrix is rectangular; that is, the matrix has the same number of observations for all variables and the same number of variables for all observations.

Now you have seen what the structure of the data should resemble. In the following sections, you will learn how to put your data into such a structure. At this point, you can close the Data Editor: press *Alt+F4* or use the mouse to close the window, as appropriate in your operating system.

11.2 Importing machine-readable data

Statistical offices and other research institutions produce large amounts of data on machine-readable media (for example, CD-ROMs) or on the Internet.² You can use these data for secondary data analyses.³

However, such datasets, or data that you might get from a collaborator, are often not available in Stata format. Often you will be confronted with data from other statistical packages (SAS, SPSS, and R), databases (Access, Oracle, and MySQL), or spreadsheet programs (Excel and OpenOffice Calc). The origin of a new file usually can be determined from the filename extensions. Most software automatically assigns a specific extension to the filename that the user has chosen while saving a file; Microsoft Word, for example, adds the extension `.docx`. Hence, if we get a file with the extension `.docx`, we know that this file is from Microsoft Word. Table 11.1 lists some extensions of frequently used software for statistical analysis:

Table 11.1. Filename extensions used by statistical packages

Extension	Origin
<code>.dta</code>	Stata
<code>.odf</code>	OpenOffice Calc
<code>.por</code>	SPSS (portable)
<code>.rda, .rdata</code>	R
<code>.sas7bdat</code>	SAS dataset
<code>.sav</code>	SPSS
<code>.xls, .xlsx</code>	Excel
<code>.xpt</code>	SAS (portable)

The file types listed in table 11.1 share the characteristic that they are all binary file formats. Although some of these files can be read directly into Stata, they are designed to be read particularly by their originating software. If Stata cannot read these files directly, you need to transform them to another format before you can load them into Stata.

If the new file has none of the above extensions, chances are that they are plain ASCII files. Unlike the binary file types mentioned in table 11.1, plain ASCII files do not have a standard filename extension (typical extensions are `.txt`, `.tsv`, `.csv`, `.raw`,

2. A list of data archives can be found at http://www.ifdo.org/wordpress/?page_id=35.

3. Analyses are called secondary if they are carried out using data collected by others.

.out, and .asc). Within Stata, the easiest way to determine that a new file is in ASCII format is to use the Stata command `type filename`. With this command, the contents of the specified file are displayed in the Results window, without loading the file into memory. You can see the difference between a binary file and an ASCII file by typing the following commands:

```
. type popst1.dta
. type popst1.xls
. type popst1.raw
```

The output of the first two commands is rather confusing. Only the output of `type popst1.raw` is readable for humans.⁴ This is a good indication that it is a plain ASCII file, meaning a file without program-specific control characters. Plain ASCII files can be loaded directly into Stata.

11.2.1 Reading system files from other packages

Reading Excel files

Many freely accessible datasets are available in Excel format. Stata therefore provides an import filter to read those files into Stata. However, when importing Excel-formatted files into Stata, a few things are not as straightforward as one would think. Figure 11.2 shows an example of an Excel file loaded into OpenOffice Calc. The dataset is arranged in a data matrix very similar to the arrangement of data in Stata's browser window (see figure 11.1). However, there are differences:

- The first line of the Excel data matrix has a line explaining the contents of each column. Excel files usually have at least one such line, if not several lines. How this first line and subsequent lines are read into Stata will affect your resulting dataset.
- If you open the contents of an Excel file with Excel (or OpenOffice Calc), the data are viewed as a table within a large table (the worksheet). It is thereby possible that the worksheet contains more than one such table. In figure 11.2, there might be another table in the cells from AA-1 to AF-17, for example, and this would be visibly indistinguishable from the case where we just have data in the area from A-1 to F-17. To Stata, the empty rows and columns between the different subtables are variables and observations with missing values.
- A particular cell in Excel might contain a number created from other entries in the same table through a formula, for example, a summary of all values in the table. In these cases, Stata imports the number, not the formula.

4. All files of the data package with filenames starting with `pop` contain information on the number of inhabitants and the size of various German regions in 2010. All files have been created with the do-file `cropd.do` from the list of German communities provided by the German Statistical Office (<http://www.destatis.de>).

- Excel's data type is cell based. You cannot infer from a cell containing a number that all cells in the same column contain numbers as well. Some values displayed as numbers might be formatted as a string of characters.
- Excel files often contain more than one worksheet. The file `popst1.xls` contains, for example, the worksheet **Absolute** with absolute numbers and the worksheet **Percent** with relative frequencies.

A	B	C	D	E	F	G
1 State (name)	Area in km ²	Population	Population (male)	Population (female)	Population density	
2 Baden-Württemberg	35741.5400617123	10744921	5285946	5458975	300.6283569336	
3 Bayern	68025.4099766016	12510331	6136004	6374327	183.9067382813	
4 Berlin	891.5399780273	3442675	16886256	1756419	3861.4926757813	
5 Brandenburg	29481.9600570202	2511525	1244101	1267424	65.1885375977	
6 Bremen	404.2800140381	661716	322227	339489	1636.7764892578	
7 Hamburg	755.1599731445	1774224	866623	907601	2349.4677734375	
8 Hessen	20787.7200107574	6061951	2970776	3091175	291.612121582	
9 Mecklenburg-Vorpommern	23189.0200080872	1651216	818119	833097	71.2068023682	
10 Niedersachsen	46510.3399454355	7928815	3894627	4034188	170.4742431641	
11 Nordrhein-Westfalen	34087.9700431824	17872763	8719694	9153069	524.3129272461	
12 Rheinland-Pfalz	19847.2600201368	4012675	1970665	2042010	202.1777801514	
13 Saarland	2568.6599926949	1022585	497605	524980	398.1005554199	
14 Sachsen	18419.6400022507	4168732	2039451	2129281	226.3199462891	
15 Sachsen-Anhalt	20448.7900342345	2356219	1153749	1202470	115.2253494263	
16 Schleswig-Holstein	15699.6899659036	2832027	1387049	1444978	180.3874511719	
17 Thüringen	16172.4899902344	2249882	1110714	1139168	139.1178436279	
18						

The screenshot shows the 'Absolute' sheet of the 'popst1.xls' file in OpenOffice.org Calc. The table contains 18 rows of data, each representing a German state with its name, area, population, and density. The columns are labeled A through G. The data is presented in a clean, tabular format with clear column headers.

Figure 11.2. Excel file `popst1.xls` loaded into OpenOffice Calc

Now that you know these peculiarities, we introduce the command `import excel`, Stata's import filter for Excel files. In its simplest form, the command expects just a filename. Let us give this a try:

```
. import excel popst1, clear
. describe
Contains data
obs:           17
vars:          6
size:        1,751
variable   storage  display    value
          name     type   format   label   variable label
A          str22  %22s
B          str17  %17s
C          str10  %10s
D          str17  %17s
E          str19  %19s
F          str18  %18s
Sorted by:
Note: dataset has changed since last saved
```

The Excel worksheet is imported as a Stata dataset with 6 variables and 17 observations. The dataset contains all nonempty cells of the *first* worksheet of the Excel file. The variable names A, B, ..., F were inherited from the column addresses of the Excel file. Most importantly, all variables in the Stata dataset are string variables (see section 5.3). The reason is that if a column contains at least one nonnumerical text, the entire column is imported as a string variable. Because of the explanation line, the first cell of each column is not a number, and the columns are thus imported as string variables.

There are two ways to import `popst1.xls` in a form that is more suitable for being analyzed with Stata. The first way is to specify the option `firstrow` of `import excel`. With `firstrow`, Stata takes the contents of the first row of the Excel file as variable names. If the contents of the first row are not valid variable names, Stata creates valid variable names from the contents—basically by stripping out all characters that are not allowed in variable names—and moves the information of the first line into the variable labels:

```
. import excel popst1, clear firstrow
. describe
Contains data
obs:           16
vars:          6
size:        800
storage      display    value
variable name   type   format   label   variable label
Statename      str22  %22s      State (name)
Areainkm2     double  %10.0g    Area in km^2
Population     long   %10.0g    Population
Populationmale long   %10.0g    Population (male)
Populationfem~e long   %10.0g    Population (female)
Populationden~y double  %10.0g    Population density
Sorted by:
Note: dataset has changed since last saved
```

This creates a pretty nice dataset with 16 observations—one for each German state—and 5 numerical variables. If you prefer shorter variable names than the ones automatically created, the Stata command `rename` lets you change the names. We do not recommend changing the first line in the Excel file itself, because such an exercise would hamper the full reproducibility of your analysis.

The option `firstrow` does not necessarily prevent `import excel` from importing Excel columns into string variables. Particularly, `firstrow` does not work if there are two or more lines of explanations before the numerical values or if there is another string in a cell somewhere below the numerical values. Note that a column will be imported as a string variable if just one cell in the column contains a white space character (that is, a tab or a space). In all of these situations, the option `cellrange()` allows you to specify the area of the Excel worksheet that should be used in the Stata dataset. For example,

```
. import excel state area pop_total pop_male pop_female
> using popst1, clear cellrange(A2:E17)
```

instructs Stata to load only the cells between cell A2 and E17. Thus we have cut out the first line, all cells below row 17 (where there are no further data points), and all columns to the right of column E (we intentionally left out the values in column F). Also we specified a list of variable names directly after `import excel`, which means that we avoided having to rename the complicated variable names created by `firstrow`. However, now variable labels are not created automatically. The structure of the command has changed slightly, and the keyword `using` has to be placed in front of the filename.

So far, we have only dealt with the first worksheet of a workbook. To load the second worksheet, which we do in our final example, we need the option `sheet("sheetname")`:

```
. import excel popst1, clear sheet("Percent") firstrow
```

Reading SAS transport files

Stata has an import filter for reading system files in SAS transport format (SAS XPORT files, .xpt). You can read SAS XPORT files into Stata by using the command `import sasxport`. For example, the file `popst1.xpt` is imported into Stata with

```
. import sasxport popst1, clear
```

Reading other system files

To read all other system files into Stata, you can either use a data-conversion program or export the data as an ASCII file from the program in which they were saved. Then you can read them directly into Stata.⁵

We know of two programs that convert system files from one statistical package to another: Stat/Transfer by Circle Systems and DBMS/COPY from DataFlux, a subsidiary of SAS Institute. The advantage of using a conversion program is that you can keep variable and value labels that have been assigned to a data file and, in some cases, even keep missing-value definitions. But to import data into your system, you need not pay for an external program. As we said, you can always save data as an ASCII file and follow the description in the upcoming sections.

11.2.2 Reading ASCII text files

Stata has three commands for reading ASCII files: `infile`, `insheet`, and `infix`. The last two commands are simplified special cases of the `infile` command. Once you are at ease using `infile`, you can read in all ASCII files with no problems. We begin with the `insheet` command, which is very easy to use. However, because not all datasets can be read in using `insheet`, we will explain the use of `infile` in detail.

Reading data in spreadsheet format

In a simple case where data are in an ASCII data file, each observation is written into a separate row, and the columns (variables) are separated by commas or tabs. Spreadsheet programs usually export ASCII files of this type, which is why the format is also known as spreadsheet format. Windows files are often tab-delimited with the file extension `.txt` or comma-separated with the extension `.csv`. We have prepared a data file called `popst1.raw`:⁶

5. The user-written program `usespss` loads SPSS-format (.sav) datasets into Stata. See chapter 13 for information on how to install user-written commands.

6. We use the extension `.raw` because it is the Stata default extension for ASCII files. See section 3.1.8 for more information.

```
. type popst1.raw
Baden-Wuerttemberg,35742,10744921,5285946,5458975,301
Bayern,68025,,6136004,6374327,184
Berlin,892,3442675,1686256,1756419,3861
Brandenburg,29482,2511525,1244101,1267424,85
Bremen,404,661716,322227,339489,1637
Hamburg,755,1774224,866623,907601,2349
Hessen,20788,6061951,2970776,3091175,292
Mecklenburg-Vorpommern,23189,1651216,818119,833097,71
Niedersachsen,46510,7928815,3894627,4034188,170
Nordrhein-Westfalen,34088,17872763,8719694,9153069,524
Rheinland-Pfalz,19847,4012675,1970665,2042010,202
Saarland,2569,1022585,497605,524980,398
Sachsen,18420,4168732,2039451,2129281,226
Sachsen-Anhalt,20449,2356219,1153749,1202470,115
Schleswig-Holstein,15700,2832027,1387049,1444978,180
Thueringen,16172,2249882,1110714,1139168,139
```

The file `popst1.raw` contains the same information on the number of inhabitants and the sizes of the German states as the information presented in the Excel file in figure 11.2. Every row begins with the name of the state and continues with the numbers for the sizes and inhabitants. Every row in this file is a new observation, and each observation contains six pieces of information, separated by commas. In the second row, there seem to be only five values—the state name “Bayern” (Bavaria) and four numbers. But if you look closely, you will find two commas, one directly after the other. The value between is missing, so we do not know what the population was in Bavaria, but we do know that the value of the third entry is missing. There are still six pieces of information in the second row. This is an important point because the data must be rectangular. Each observation must appear in one row, and each row must contain the same number of entries, even if one of the values is missing.

A text file generated by a spreadsheet program may also separate variables by tabs instead of commas. For humans, however, tabs are difficult to distinguish from blanks. The option `showtabs` with the `type` command displays characters representing tabs:

```
. type popst5.raw
. type popst5.raw, showtabs
```

Datasets delimited with commas or tabs can be read into Stata using the command `insheet`. To read in `popst1.raw`, for example, you could type

```
. insheet using popst1.raw, clear
. describe
```

Following `insheet`, the keyword `using` is issued to specify the file to be read in. If the extension of a file is `.raw`, you can omit it. As you see in the output of `describe`, Stata automatically assigned six variable names, `v1–v6`, for the six columns. If you prefer other variable names, you can indicate these between `insheet` and `using`:

```
. insheet state area pop_total pop_male pop_female pop_dens using popst1, clear
```

When you enter variable names, Stata knows how to deal with numbers, so you can also use

```
. insheet state area pop1-pop4 using popst1.raw, clear
```

Many spreadsheet programs store column titles in the first row of spreadsheet files. If this is the case, `insheet` uses the first row as variable names and all other rows as observations. The file `popst2.raw` contains such a row. From

```
. insheet using popst2.raw, clear
```

you obtain the same dataset as the one you created with the previous command.

Although `insheet` is fairly easy to use, checking the state of your data is crucial. Different spreadsheet programs have different problems exporting ASCII files, so you need to be careful. In addition to the problems already mentioned in section 11.2, be aware of the following:

- A common problem is that commas are used in numbers as thousands separators or—depending on the country you work in—as decimal points. In both cases, one numeric value would be split by Stata into two or more variables.
- Sometimes missing values are coded as blanks or dots in the plain ASCII spreadsheet file. In this case, the respective column will be imported into Stata as string variables. With `insheet`, the characters separating cells should always be next to each other if the value between them is missing.
- If the individual variables in the ASCII file are separated by blanks or if characters other than commas or tabs were used to separate cells, you must use the `delimiter` option. See `help insheet` for details.
- You cannot use `insheet` if cells are not separated or if the values of any observations span across multiple rows. You need to use another Stata command called `infile`.

You can solve some of these problems by cleaning up your file with the Stata command `filefilter` or by loading the dataset with the `infile` command. Unless it is fully automated, we do not recommend repairing the text file with a text editor.

Reading data in free format

Sometimes ASCII files are in free format, which means that individual variables are separated by blanks, tabs, commas, or line breaks. Take `popst3.raw` as an example:

```
. type popst3.raw
Baden-Wuerttemberg
    35742 10744921  5285946  5458975      301
Bayern          68025 12510331  6136004  6374327      184
Berlin          892  3442675  1686256  1756419     3861
Brandenburg    29482 2511525 1244101 1267424       85
Bremen           404  661716  322227  339489    1637
Hamburg          755 1774224  866623  907601    2349
Hessen           20788 6061951 2970776 3091175      292
Mecklenburg-Vorpommern  23189 1651216  818119  833097       71
Niedersachsen   46510 7928815 3894627 4034188      170
Nordrhein-Westfalen  34088 17872763 8719694 9153069      524
Rheinland-Pfalz  19847 4012675 1970665 2042010      202
Saarland          2569 1022585  497605  524980      398
Sachsen           18420 4168732 2039451 2129281      226
Sachsen-Anhalt   20449 2356219 1153749 1202470      115
Schleswig-Holstein  15700 2832027 1387049 1444978      180
Thueringen        16172 2249882 1110714 1139168      139
```

Here the information for the state Baden-Würtemberg is for some reason spread over two rows. Unlike in the spreadsheet format, observations in the free format can be spread over several rows. This has an important implication: Stata can no longer automatically identify how many variables the dataset contains. You must enter this information.

ASCII files in free format are read in using the `infile` command. You indicate the number of variables by specifying a variable list. The file `popst3.raw`, for example, can be read with

```
. infile str22 state area pop_tot pop_male pop_female pop_dens using popst3.raw,
> clear
```

or

```
. infile str22 state area pop1-pop4 using popst3.raw, clear
```

After specifying `infile`, you enter a variable list, followed by `using` and a file name. From the specified variable list, Stata can infer that there are six variables to read. Therefore, a new observation starts every sixth entry.⁷ One problem reading in `popst3.raw` is the first variable, which contains the name of the state. Because this variable contains text, it must be marked as a string variable (`str` in the variable list). In doing so, you must specify the maximum number of letters the variable may contain. Here because “Mecklenburg-Vorpommern” is the longest and therefore limiting element, with 22 characters, we use `str22` as the storage type. However, instead of counting the letters, you will usually find it easier to allow for more space than necessary and later optimize the dataset using `compress`.

In this form of the `infile` command, everything that is not a blank, tab, comma, or line break is read as the value of a variable until one of these characters appears. This logic prohibits the use of blanks within string variables and is a common source of error messages. In `popst4.raw`, for example, the hyphen in “Mecklenburg-Vorpommern” has been erased. If you repeat the last command, inserting `popst4.raw` as the filename, you get the following:

```
. infile str22 state area pop1-pop4 using popst4.raw, clear
`Vorpommern' cannot be read as a number for area[8]
`Niedersachsen' cannot be read as a number for area[9]
`Nordrhein-Westfalen' cannot be read as a number for area[10]
`Rheinland-Pfalz' cannot be read as a number for area[11]
`Saarland' cannot be read as a number for area[12]
`Sachsen' cannot be read as a number for area[13]
`Sachsen-Anhalt' cannot be read as a number for area[14]
`Schleswig-Holstein' cannot be read as a number for area[15]
`Thueringen' cannot be read as a number for area[16]
(eof not at end of obs)
(16 observations read)
```

What happened? The moment you have a blank in an unexpected place, the allocation of values to variables shifts. “Mecklenburg” is read into the string variable `state`. The blank between “Mecklenburg” and “Vorpommern” is understood as the beginning of a new variable. Stata then tries to read “Vorpommern” as a numerical value of the variable `pop90`, fails, and reports this. This mistake is continued through all rows because `infile` attributes values to one observation until the variable list ends.

To avoid problems with strings containing blanks, you should enclose strings within quotation marks. Missing strings should appear as two consecutive quotes to keep your data rectangular.

7. The same logic applies to the spreadsheet format, which is a special case of the free format, so we can therefore use the above commands to read the files `popst1.raw` and `popst2.raw`.

Reading data in fixed format

Fixed-format data have no specific separating characters between variables. Instead, we know from external information what the numbers at certain positions of the file mean. The file `popst6.raw` is an example of a fixed-format dataset:

```
. type popst6.raw
    Baden-Wuerttemberg357421074492152859465458975 301
        Bayern680251251033161360046374327 184
        Berlin 892 3442675168625617564193861
    Brandenburg29482 251152512441011267424 85
        Bremen 404 661716 322227 3394891637
        Hamburg 755 1774224 866623 9076012349
        Hessen20788 606195129707763091175 292
    Mecklenburg-Vorpommern23189 1651216 818119 833097 71
        Niedersachsen46510 792881538946274034188 170
    Nordrhein-Westfalen340881787276387196949153069 524
        Rheinland-Pfalz19847 401267519706652042010 202
        Saarland 2569 1022585 497605 524980 398
        Sachsen18420 416873220394512129281 226
        Sachsen-Anhalt20449 235621911537491202470 115
    Schleswig-Holstein15700 283202713870491444978 180
    Thueringen16172 224988211107141139168 139
```

Here the individual variables cannot be separated unless you know, for example, that the variable `area` starts at the 23rd position of each row and ends at the 28th position.

For this type of file, the command `infile` must be used in combination with a dictionary. This technique can also be used for the other data formats. It is the most general way of entering data. The dictionary is an auxiliary file⁸ used to define the positions of the variables. Variable labels and comments can be inserted, and unimportant variables or rows can be omitted. A simple version of such an auxiliary file consists solely of a row with the name of the file containing the data and a simple list of variables to be read in. Here is an example:

```
----- begin: popst5kk.dct -----
1: dictionary using popst5.raw {
2:   state
3:   area
4:   pop_total
5:   pop_male
6:   pop_female
7:   pop_dens
8: }
```

----- end: popst5kk.dct -----

8. A dictionary can also be written directly before the rows of numbers in the ASCII data. Here, however, the ASCII data would be worthless for other statistical packages. Also writing the dictionary in an editor before rows of numbers is often difficult because many editors cannot read in large datasets. We therefore recommend setting up the dictionary as an external file.

This dictionary merely lists the variable names and, for string variables, the corresponding specifications. As the positions of the variables are not defined, this dictionary clearly cannot read in `popst6.raw`. But you can use it to read `popst5.raw`. To understand how dictionaries work, try that first—and we will develop the dictionary for `popst6.raw` later.

Type the above example in the Stata Do-file Editor (or any other editor) and save it as `popst5.dct`⁹ in your working directory. Windows users should make sure that the file is not accidentally saved as `popst5.dct.txt`. Then return to Stata and type the following command:

```
. infile using popst5.dct, clear
```

Unlike the command on page 405, this command does not contain a variable list. Furthermore, `using` is followed by the reference to the dictionary file instead of to the data file. You can even leave out the extension of the dictionary file, because Stata automatically expects a file with extension `.dct`. The reason is that the `infile` command is entered without a variable list. Here Stata automatically assumes that `using` will be followed by a dictionary.

In this example, this procedure not only seems but also is unnecessarily complicated. Nonetheless, the use of such a dictionary can be extremely useful. On the one hand, it allows a detailed description of the dataset; on the other hand, it is the only way to read fixed-format datasets.

The following syntax extract shows you some of the more important ways to design a dictionary. Stata allows some further options, but these are rarely necessary and will therefore not be described here. For an overview with examples, type the command `help infile2`.

```
[infile] dictionary [using filename] {
    * comments may be included freely
    _lrecl(#)
    _firstlineoffile(#)
    _lines(#)
    _line(#)
    _newline[(#)]
    _column(#)
    _skip[(#)]
    [type] varname [:lblname] [%infmt] ["variable label"]
}
```

9. You can also find the dictionary among the files you installed at the beginning under the name `popst5kk.dct`.

Let us start with the second-to-last row. Recall that all elements in square brackets are optional. The only required element is therefore a variable name. For every variable, you can also specify the storage type [*type*] (see section 5.7), give a variable label ["*variable label*"], and specify the width of the variable [% *infmt*] and the name of the value label [:*lblname*].

The most important additional element in the syntax for dictionaries is probably `_column(#)`. With `_column(#)`, you can mark at what point in your file a specific variable begins. You can determine the end of the variable by specifying the format. To specify that the variable `area` in the file `popst6.raw` begins in the 23rd column and has a width of five digits, you would type the following:

```
_column(23) area %5f
```

We have chosen `%5f` as the format type because the area numbers do not contain more than five characters. With the other variables, proceed correspondingly.

The three line options refer to the rows in your file. Using `_firstlineoffile`, you determine in which row your data begin. Some files might contain a title, references, or comments on the data collection, which you can skip by specifying the first row with real data. Using `_lines`, you can state how many rows constitute an observation. This is necessary for data in fixed format when an observation is spread over several rows. The availability of `_lines` is also helpful for data in free format if you do not want to read all the rows into Stata. You determine the rows from which to read the values using `_line`. The values following this option always refer to the rows within an observation.

Most likely, you will not need a dictionary file right away. However, it is important to know that this possibility exists and how to use it. Let us therefore show you a simple example of a dictionary file in `popst6kk.dct`. We recommend that you copy the dictionary file to `popst6.dct`, use your copy to read the data into Stata, and finally work through some alterations of the dictionary file. You will quickly learn that things are not nearly as complicated as they seem.

```
begin: popst6kk.dct
1: dictionary using popst6.raw {
2:   _column(1) str22 state %22s "State (name)"
3:   _column(23) area      %5f "Area in km^2"
4:   _column(28) pop_total %8f "Total Population"
5:   _column(36) pop_male  %7f "Male Population"
6:   _column(43) pop_female %7f "Female Population"
7:   _column(50) pop_dens  %4f "Population Density"
8: }
```

For data in fixed format, the `infix` command offers an easier, although less flexible, alternative to the `infile` command. The `infix` command allows you to specify the variable names and column positions directly within the command. Because datasets in fixed format without separating characters are less common now, we will not discuss this command further here but instead leave you with an example that reads in the first three variables of `popst6.raw`:

```
. infix str22 state 1-22 area 23-27 pop_total 28-35 using popst6.raw, clear
```

11.3 Inputting data

For one of the examples in discussing graphs, we used temperature data from 1779 to 2004. Parts of these data are listed in table 11.2. The table contains the average temperatures by year for July and December for the small city of Karlsruhe, Germany, from 1984 to 1990. Printed this way, the information is an example of data that are not machine readable.¹⁰ To analyze these data, you first need to input the data by hand using Stata's Data Editor or the `input` command.

Table 11.2. Average temperatures (in °F) in Karlsruhe, Germany, 1984–1990

Time	Yearly	July	December
1984	49.82	65.84	36.86
1985	48.92	68.54	40.28
1986	50.18	67.28	38.66
1987	49.64	67.10	38.12
1988	52.16	66.56	40.82
1989	52.16	68.90	38.48
1990	52.88	68.00	35.24

11.3.1 Input data using the Data Editor

We need to begin this section with no data in memory:

```
. clear
```

10. The data are in machine-readable form at <http://www.klimadiagramme.de/Europa/special01.htm>. Here, however, we prefer to stick to the printed table with its Fahrenheit information. The webpage contains an HTML table of monthly temperatures in degrees Celsius from 1779 to 2004. If you want to read such data into Stata, just copy the table to a powerful text editor, do some searching and replacing, and import the data into Stata with `insheet` or `infile`.

You can open the Data Editor by typing the command

```
. edit
```

Typing `edit` opens a window containing an empty data matrix. The upper left corner of the data matrix is highlighted; this is the currently active cell, and you can alter its contents. Above the table, to the left of an input field, you see “`var1[1]`”. Here you can type the value of the first variable (“`var1`”) for the first observation (“`[1]`”). If you press the *Down Arrow* key, the highlighting also shifts downward but remains within variable 1; however, the value within the square brackets changes. Every cell therefore corresponds to a new observation.

Now type, say, the value 1984 in the input field to the right of “`var1[1]`”. Then confirm your entry by pressing *Enter*. The number 1984 is written in the first cell, the second observation becomes the active cell, and the variable is given a name. You can now directly type the value for the second observation, for example, 1985.

Use the mouse to click on the cell in the first row of the second column and type a value (for example, 49.82) for “`var2[1]`” into the input field. Confirm this entry by pressing *Enter*. The dot that appears in the second row of column 2 is a placeholder for a missing value. As soon as a value is entered for an observation, the program adds missing values for the remaining rows that already contain data. This way, the dataset always stays rectangular. If you continue entering data into the second column, the missing values are simply overwritten.

Before we end this example, use the mouse to double-click on the gray field with the label `var1` at the top of the first column. This sets the focus to the Stata properties panel on the right-hand side of the Data Editor window. Here you can enter the name and label definitions of the highlighted variable. Type `year` in the first textbox and `Year of observation` in the second textbox.

Now close the Data Editor by pressing *Alt+F4* or clicking on the button that closes windows in your operating system. This will bring you back to the normal Stata window. You can now treat the dataset like any other Stata file.

11.3.2 The input command

Another method to enter data into Stata is the `input` command. The main advantage of this method, compared with entering the data using the Data Editor, is that it can be used in do-files, so you can replicate your steps.

To begin with, we will use the command `input` to enter just one observation of data from table 11.2. Begin by clearing Stata:

```
. clear
```

Start entering data using `input`. Type the command and list all variable names for which you want to enter values, for example,

```
. input year temp
```

Following this entry, the two variable names appear on the screen, together with the number 1:

```
year temp
1.
```

Here 1. is the entry request of the `input` command. You are asked to enter the values for the first observation. According to table 11.2, these are the numbers 1984 and 49.82. Type these numbers, separated by a blank, in the Command window and confirm your entry by pressing *Enter*. The entry request for the second observation, 2., appears on the screen. Now you could enter the second observation accordingly, but for now, just type

```
. end
```

which brings you back to the normal Stata prompt.

As we said, the main advantage of `input` is that you can easily use the command in do-files. Let us give this a try. Open the Stata Do-file Editor (or any other text editor) and produce the following do-file:

```
begin: crkatemp.do
1: clear
2: input year mean jul
3:    1984  49.82  65.84
4:    1985  48.92  68.54
5:    1986  50.18  67.28
6:    1987  49.64  67.1
7:    1988  52.16  66.56
8:    1989  52.16  68.9
9: end
10: exit
end: crkatemp.do
```

After saving the do-file under, say, `crkatemp.do`, you can run the file as usual. Running the do-file will produce the data:

```
. do crkatemp
```

You can also use `input` to add new variables or more observations to an existing dataset. To add observations, type `input` without a variable list. Stata will then automatically let you input further observations. Let us use this feature to add the temperatures for the year 1990, which we have forgotten in the do-file:

```
. input
. 1990 52.88  68
. end
```

To add another variable, type `input` with a new variable name. Here we add the average December temperatures to our dataset:

```
. input dec
. 36.86
. 40.28
. 38.66
. 38.12
. 40.82
. 38.48
. 35.24
```

Here you can omit typing `end`. Stata knows that the dataset contains seven observations and will not let you input more observations if you use `input` with a varlist for an existing dataset. However, if you need to reproduce your steps, a better place to add variables or observations to the dataset would probably be the do-file we produced above.

Which numbers to assign?

So far, we have assumed that the numbers or text you enter are from the printed data. If you collected the data yourself, or if the printed data you enter include text (nonnumerical information) that you would like to use in your analysis, or if data are missing in the sources, you will have to make some coding decisions, which we want to address briefly.

- First of all, make sure that your file has an identification variable. You saw in our file `data1.dta` that the variable `persnr` was an identification variable. For the temperature data we just entered, `year` would serve the same purpose. It is easy to forget to enter an identification variable, especially when the printed data or data you find on the Internet provide this information through sorting. Suppose that you enter grades for a class and, being lazy, skip the names of the students to avoid typing. That seems fine because the data are ordered alphabetically and you will always know who is who. However, once your data are entered into Stata, you can sort them in various ways, and the relevant mapping information will then be lost. This will become even more problematic when you combine different datasets (see section 11.4).
- An equally important point is how to code missing values. In our population data example, we left the missing information for the population value in Mecklenburg-Vorpommern blank, and once the data are read into Stata, a period will appear indicating the missing value. Other data you enter, however—especially survey data—may contain some information about the missing values, so do not throw away essential information. In an interview, for example, there may be various reasons why data are missing: the respondent was not willing to answer, the respondent did not know the answer, or filter questions were used, so the respondent was never asked a particular question. If you merely leave this information out and assign a period to a missing value, you will not be able to determine why the

value is missing when analyzing the data. For this reason, you should use codes for all variables that explain why a value is missing in a given case. In many datasets, 97 is used for “do not know”, 98 for “answer refused”, and 99 for “not applicable”. You cannot use this coding if the true answers can contain such numbers. But you can resolve this by using missing-value codes that always contain one digit more than the answer categories (for example, 997, 998, 999). This type of coding is unambiguous but has the disadvantage that it differs across variables. A uniform codification considerably simplifies dealing with missing values. A possible alternative would be to use negative values as codes. Usually, you will not have any negative values as possible answers, so you can use, for example, the values -1, -2, and -3 as missing-value codes for all variables. In Stata, you can code all of these values into 27 different missing-value codes: the usual period, as well as .a to .z.

- When you enter plain text (such as the name of a state in the population example, names of students, or job titles), you can either use codes when entering them or enter the text directly into Stata. Which of the two options you choose essentially depends on the question asked. If the number of possible answers is limited, you can easily assign a number to every answer. Write down the allocation while entering the data and merely enter the number in the data file. This procedure is easiest if you are entering the data by yourself. If several people are entering data simultaneously, make sure that the codes have been defined previously (and are known to everyone). On the other hand, if the answer context allows for an unforeseeable number of possibilities, compiling a list before entering the data will be difficult, if not impossible. Here we recommend that you enter the text directly into Stata and read section 5.3.
- Sometimes your data will contain dates—not just years, as in the temperature example, but days and months. In this case, we recommend that you include the century in the year. If you add data later and if the old and new data are to be joined, the dates must be unambiguous.
- Make sure that every variable contains only one *logical unit*, that is, one piece of information. Sometimes respondents are asked to mark all answer categories that apply to one question. Here each of the possible answers should be stored in a separate variable with a “yes” or “no” indicator. We recommend that you use variable names that display the content of these questions, as the variables `eqphea` to `eqpnqrj` do in our `data1.dta`.
- To avoid errors, we recommend that you enter data twice, merge the datasets afterward, and check for differences (see section 11.4). Sometimes time and financial constraints will often preclude such double entry. Then some other strategies are helpful. For example, when you enter data from a questionnaire that has no codes printed next to the boxes, copy the questionnaire on a transparency and write the codes behind the boxes on the transparency. When you enter the data, you can then simply place the transparency on top of the questionnaire.¹¹ Also

11. For details on designing questionnaires, see, for example, Fowler (1984).

do not try to modify data or make coding changes while you are entering the data. For instance, in our temperature example, there is no point in converting the temperatures from Fahrenheit to Celsius before you have finished entering the data. You can automate almost everything later within Stata (see chapter 5).

11.4 Combining data

Suppose that you want to analyze how life satisfaction develops over time by using data collected with different surveys at different times. Or suppose that you want to compare the life satisfaction of German and American citizens by using survey data collected in both countries. Finally, suppose that you want to control for population density in the regression of rents (from chapter 9) by using the dataset constructed above. In any of these examples, the information you need is spread over several files, so to perform your analyses, you need to combine the files into a rectangular dataset.

Here we explain how to combine datasets, using as examples several datasets from the German Socio-Economic Panel (GSOEP), which is the same source that we used to produce `data1.dta`. The GSOEP is a representative longitudinal study that has collected information annually on more than 28,000 households and nearly 70,000 persons since 1984. At present, the information from the GSOEP is stored in 326 different files and is therefore an excellent resource for demonstrating data-management tasks, both simple and difficult.¹²

To follow our examples, you need to understand the file structure in the GSOEP database. We will provide a brief overview in section 11.4.1.

Stata has three commands for combining different datasets: `merge`, `append`, and `joinby`. `joinby`, however, is needed only in exceptional cases; for more information, see [D] `joinby`. `merge` and `append` apply more generally. With `merge`, you add variables (columns) to a dataset, and in our examples, we will combine data from different points in time, as well as personal information and household information (see section 11.4.2). `append` adds observations (rows) to a dataset. In section 11.4.3, we will show by example how to combine data from the GSOEP with the U.S. Panel Study of Income Dynamics (PSID).

11.4.1 The GSOEP database

In a panel survey, the same people are interviewed several times, so it is not surprising that multiple datasets may contain information from the same persons. These datasets

12. Two user-written programs exist to ease the process of putting together the files of the GSOEP database. `PanelWhiz`, by John Haisken-DeNew, is a large-scale project for the data management of various panel datasets, including the GSOEP. `PanelWhiz` is downloadable from <http://www.panelwhiz.eu>. The other program, `soepuse` by Ulrich Kohler, offers far less functionality than `PanelWhiz` but is somewhat simpler to use. Read more about user-written programs in chapter 13.

are updated every year. Of course, the new information could have been written into the existing data file right away, but usually it is entered into a separate file for storage reasons. Our task later will be to add the variables that are spread over different files to one file.

The GSOEP is a household panel, which means that all persons aged 16 and older from randomly selected households are included. Moreover, not only data on the individuals in households are gathered but also characteristics of the household as a whole (household-level data). However, because the information about the entire household is the same for all persons within the household, these data are stored in separate files: one in which each observation is a household and another in which each observation is a person. The household information can be merged with the person data when needed. In one of our examples, we will merge household information (for example, the state in which the household lives) with the person data.

You will find household panels like the GSOEP in several countries. In fact, one of the first panels of this kind was the PSID, which is a U.S. panel study that has been running since 1968. In recent years, efforts have been made to make the collected data comparable, so that you can now combine, for example, PSID data and GSOEP data, which are translated into internationally comparable variables. These are also stored in separate files.

Our data package includes 78 of the original 326 files of the GSOEP database.¹³ To get an impression of the data structure, look at the files in the `c:\data\kk3\kksoep` directory by typing

```
. dir kksoep/
<dir> 10/14/04 08:36 .
<dir> 10/14/04 08:36 ..
26.6k 10/14/04 08:36 ahbrutto.dta
94.7k 10/14/04 08:36 ap.dta
201.1k 10/14/04 08:36 apequiv.dta
27.5k 10/14/04 08:36 bhbrutto.dta
86.4k 10/14/04 08:36 bp.dta
183.4k 10/14/04 08:36 bpequiv.dta
26.5k 10/14/04 08:36 chbrutto.dta
83.4k 10/14/04 08:36 cp.dta
(output omitted)
```

The data from the different data collection years (called waves) are stored in different data files. All information gathered in the first year (1984) of the GSOEP is written into files whose names start with the letter `a`; information from 1985 is written to files whose names start with `b`; and subsequent years start with successive alphabetical letters up to the last survey year we have here (2009), which is written to files whose names start with the letter `z`.

¹³. The data files that you downloaded contain only a few of the variables of the original GSOEP files. Also the number of respondents is reduced: our data are a 37.5% sample from the observations of the original database. In accordance with German regulations for data confidentiality, we randomized parts of the information in the data files.

For each year (or, equivalently, letter), you see three file types: `hbrutto`, `p`, and `pequiv`. The observations (rows) of the `hbrutto` files are households, which are indicated by the letter `h` on the second character in the filenames. These files contain information about the households that is known prior to the survey, for example, the state in which the household lives. Other types of household-level data of the GSOEP, which are not included among the files you installed in the *Preface*, contain information such as apartment size or monthly rent. More generally, all information that is the same for all persons within a household is stored in a household-level dataset. Storing this information for each respondent would be redundant and a waste of disk space.

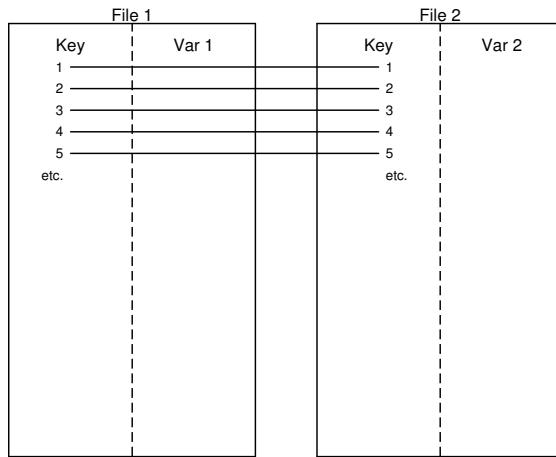
The observations (rows) of the two other file types represent persons, as indicated by the letter `p` in the filenames. In general, answers from persons to survey questions are in the `p` files. Other file types at the person level contain variables constructed out of the respondents' answers, such as indices or scales. A special sort of these generated variables is stored in the `pequiv` files, which contain variables that are comparable with variables in other panel studies.

Our data by no means represent all the information in the original GSOEP database. However, our explanation should be enough to give you an idea of its structure. In panel studies, you cannot collect data at every point in time from all respondents. You may lose contact with some respondents, some may refuse to answer the questionnaires, and some may die. At the same time, respondents may be added to the study, depending on how the survey is conducted. For example, one policy on follow-ups in the GSOEP is that if spouses get divorced and one of them moves into the household of another person, all persons of this new household will be interviewed as well. Also panel studies commonly draw refreshment samples from time to time or include more samples. Because of the German reunification, the GSOEP had one additional sample to collect information from respondents from the former German Democratic Republic and another sample in 1993–1994 to collect data on immigrants from eastern Europe. A refreshment sample for the entire sample was taken in 2002. Naturally, all of these mechanisms lead to changing numbers of observations for each survey year.

We will explain how to combine data from several file types and survey years.

11.4.2 The merge command

The `merge` command is used to add variables to a given dataset. `merge` joins corresponding observations from the dataset currently in memory with those from a Stata-format dataset stored on the disk. In the simplest case, which we will call a *1:1 match with rectangular data*, new variables are added for each observation, as shown in figure 11.3.



Source: grmerge.do

Figure 11.3. Representation of `merge` for 1:1 matches with rectangular data

The basic syntax of the `merge` command starts with the command itself, the specification of the merge type, a variable list, and the specification of the file that you want to merge with the file in memory:

`merge match_type varlist using filename ...`

We describe the command `merge` in four steps. We start with an example of using the command for the simplest case, a 1:1 match with rectangular data. We then proceed with a 1:1 match for nonrectangular data; here the major difficulty is keeping track of the observations in the new dataset. We then provide an example of performing a series of 1:1 matches. Finally, we merge individual data with aggregate data and vice versa to show examples for the match types m:1 and 1:m.

Merge 1:1 matches with rectangular data

Suppose that you want to use data from the GSOEP to investigate the relationship between general life satisfaction and hours worked in 2001. The information on life satisfaction is part of 2001's person data files:

```
. describe using kksoep/rp
Contains data                               PPFAD: 09/10/10 10:14:13-634
                                              DB09
obs:      7,704                                13 Feb 2012 17:08
vars:      7
size:    154,080

variable   storage  display   value
name      type     format   label   variable label
-----+-----+-----+-----+-----+
hhnr      long     %12.0g   Ursprungshaushaltsnummer
persnr    long     %12.0g   Unveraenderliche Personennummer
rhhnr     long     %12.0g   Haushaltsnummer 2001
rp111     byte    %20.0g   rp111    Allg. Parteienpraefenz
rp112     byte    %25.0g   rp112    Parteipraefenz
rp113     byte    %20.0g   rp113    Parteipraefenz-Intensitaet
rp13501   byte    %20.0g   rp13501  Lebenszufriedenh. gegenwaertig
```

Sorted by:

The number of hours worked is one of the variables that were generated for international comparability and stored in 2001's pequiv file:

```
. describe using kksoep/rpequiv
Contains data                               PPFAD: 09/10/10 10:14:13-634
                                              DB09
obs:      7,704                                13 Feb 2012 17:08
vars:      12
size:    292,752

variable   storage  display   value
name      type     format   label   variable label
-----+-----+-----+-----+-----+
hhnr      long     %12.0g   Ursprungshaushaltsnummer
persnr    long     %12.0g   Unveraenderliche Personennummer
rhhnr     long     %12.0g   Haushaltsnummer 2001
d1110401  byte    %29.0g   d1110401  Marital Status of Individual
d1110601  byte    %8.0g    Number of Persons in HH
d1110701  byte    %8.0g    Number of Children in HH
d1110901  float   %9.0g   Number of Years of Education
e1110101  int     %12.0g  Annual Work Hours of Individual
e1110301  byte    %20.0g  e1110301  Employment Level of Individual
i1110101  long    %10.0g  HH Pre-Government Income
i1110201  long    %10.0g  HH Post-Government Income
i1111001  long    %10.0g  Individual Labor Earnings
```

Sorted by:

To perform your analysis, you need to combine these two datasets so that identical respondents are combined into one row (observation) of the new dataset.

To combine the two datasets, you must understand the concepts of the key variable and the match type. Let us start with the match type. In our example, each of the observations (rows) of the two datasets `rp.dta` and `rpequiv.dta` represents respondents of the same survey. Hence, each of the 7,704 observations (respondents) in the file `rp.dta` has a corresponding observation in file `rpequiv.dta`. We are calling this a 1:1 match, so the match type to be used with `merge` is 1:1.

Knowing that each respondent in one file corresponds to one respondent in the second file is not enough to merge the two files together. To do so, you also have to indicate which observation of the two different files belong together. This information is specified using a key, a variable that is part of both datasets and that has a unique value for each observation. In the GSOEP dataset, `persnr` is such a key variable. Each person who was entered at least once in the GSOEP sample was assigned a personal identification number that does not change over time and is stored in the variable `persnr` in every GSOEP file containing personal information. Hence, you can use this variable as a key to match the right observations of the datasets with each other.

To combine two datasets, you begin by loading one of them into the computer's memory; we will use the term *master data* for this file. To use the `merge` command, you then specify the match type, followed by the name of the key variable, the word `using`, and the name of the second dataset. We call that dataset the *using* dataset.

Let us try it. Load `rp.dta` into the computer's memory and merge `rpequiv.dta` with `persnr` as a key. Remember that both datasets are stored in the subdirectory `kksoep` of `c:\data\kk3`. You therefore need to add the directory name to the filename:¹⁴

```
. use kksoep/rp, clear
(PPFAD: 09/10/10 10:14:13-634 DB09)
. merge 1:1 persnr using kksoep/rpequiv
      Result          # of obs.
      _____
      not matched           0
      matched            7,704  (_merge==3)
```

Let us take a look at the merged data file:

```
. describe, short
Contains data from kksoep/rp.dta
obs:           7,704                               PPFAD: 09/10/10 10:14:13-634 DB09
vars:            17                               13 Feb 2012 17:08
size:        300,456
Sorted by:  persnr
Note: dataset has changed since last saved
. describe, simple
hhnr      rp111      rp13501    d1110701   e1110301   i1111001
persnr     rp112      d1110401    d1110901   i1110101   _merge
rhhnr      rp113      d1110601    e1110101   i1110201
```

¹⁴. Remember from section 3.1.8 that it is not necessary to add the file extension `.dta` after `using`.

The new dataset has again 7,704 observations, and nine variables were added to the original file `rp.dta` (`d1110401` to `i1111001`). The variables `hhnr` and `rhhnr` were part of both datasets, so they are not added from the `using` dataset to the master dataset. Unless otherwise stated, a variable remains unchanged in the master data when the same variable is also found in the `using` data. Finally, there is a newly added variable, which has not been used in either of the two datasets: `_merge`. This is an indicator variable to check the merge process. A value 3 for `_merge` indicates that all observations from the master data were found in the `using` data. As you can see from

<code>. tabulate _merge</code>			
<code>_merge</code>	Freq.	Percent	Cum.
<code>matched (3)</code>	7,704	100.00	100.00
Total	7,704	100.00	

in our last `merge` command, every observation is in both files, which is why we said that the data are rectangular. You could see this same information in the output of the `merge` command. By default, Stata creates the `_merge` variable when you use `merge` because it is useful for looking closer at the results of the merge.

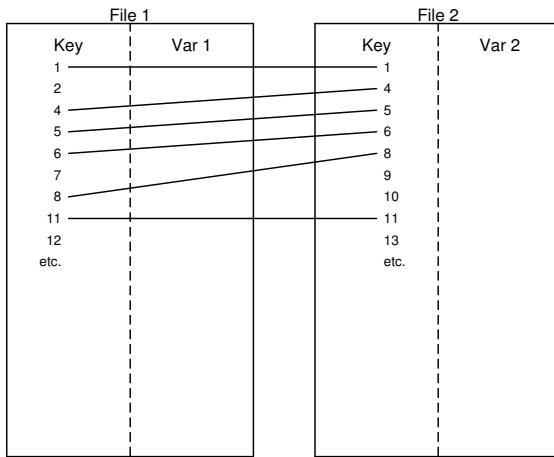
You must delete the variable `_merge` before running another `merge` command. After checking the results of your `merge` command, you should therefore

```
. drop _merge
```

Before we proceed, one more word about the key. As mentioned before, the key is used to ensure that each observation of the `using` data is merged to exactly the same observation in the master data. Therefore, the key must uniquely identify the observations of the datasets. However, it is not necessary for the key to be a single variable. Instead you can use a list of variables that jointly identify each observation by specifying a variable list between the match type and `using`.

Merge 1:1 matches with nonrectangular data

The files we merged in the last section had a special property: both files contained exactly the same observations, or, in other words, the data were rectangular. A slightly more complicated case arises if you have to merge nonrectangular data. Then one of the files contains observations that are not part of the other file and vice versa—a situation that is likely with panel data. Some respondents from an earlier panel might have refused to take part in the later year, whereas in the later year, some new respondents appear who had not participated in an earlier year (see figure 11.4 for a schematic).



Source: grmerge1.do

Figure 11.4. Representation of `merge` for 1:1 matches with nonrectangular data

If you merge two datasets with different numbers of observations, you need to keep track of the merging process. Suppose that you want to explore the development of life satisfaction and therefore combine data from 2001 with data collected at the beginning of the panel study. Life satisfaction in 1984 is stored in `ap`. From the command

we learn that this file contains information about 4,648 respondents, whereas the file produced so far contains 7,704 observations. What happens if we merge the two files? Technically, there are no difficulties in doing so. Simply merge the files as before:

```
. use kksoep/rp, clear
(PPFAD: 09/10/10 10:14:13-634 DB09)
. merge 1:1 persnr using kksoep/ap
      Result          # of obs.
      _____
      not matched           9,306
      from master           6,181  (_merge==1)
      from using            3,125  (_merge==2)
      matched                1,523  (_merge==3)
```

Now let us note some properties of the merged-data file. As expected, the new file contains variables from both files, and again the variable `_merge` has been added:

```
. describe, simple
hhnr      rhhnr      rp112      rp13501    ap5601    ap5603    _merge
persnr    rp111     rp113      ahhnr      ap5602    ap6801
```

However, the combined data file has 10,829 observations, neither just 4,648 from `ap.dta` nor just 7,704 from `rp.dta`.

```
. describe, short
Contains data from kksoep/rp.dta
obs:          10,829                               PPFAD: 09/10/10 10:14:13-634 DB09
vars:          22                                    13 Feb 2012 17:08
size:         508,963
Sorted by:
Note: dataset has changed since last saved
```

What happened? The answer can be seen from `_merge`:

<code>_merge</code>	Freq.	Percent	Cum.
master only (1)	6,181	57.08	57.08
using only (2)	3,125	28.86	85.94
matched (3)	1,523	14.06	100.00
Total	10,829	100.00	

The variable `_merge` has three values: 1, 2, and 3; that is, the new dataset contains three different types of observations. Here value 1 indicates that 6,181 observations were only part of the master data. As the file `rp.dta` has been the master data, these are respondents interviewed in 2001 but not in 1984. Value 2 indicates observations that are only in the `using` data. This means that for our example, 3,125 respondents were interviewed in 1984 who had not been interviewed in 2001.

The variable `_merge` allows us to fine-tune our dataset. Do we want to keep all the respondents or only those interviewed on both occasions? Here we proceed with the latter, which gives us what panel-data analysts call a *balanced* panel dataset:

```
. keep if _merge==3
```

Merging more than two files

Suppose that you want to add data on life satisfaction from every year to your dataset in memory—to merge not just two nonrectangular files but more than two. Merging more than two files is done by applying `merge` several times; if there are many files to merge, you might want to take advantage of a loop (section 3.2.2). However, the main problem with merging more than one nonrectangular dataset is to keep track of the observations being used in the merged dataset. Below you see one solution that works well with the GSOEP and many other large-scale panel datasets around the world.

Many large databases provide supplemental files to give more information about the observations in the dataset. In the GSOEP, the file `ppfad.dta` contains some general information about all persons for whom the entire GSOEP database has at least one piece of information. Such metadata commonly exist for all nonrectangular data kept in relational databases.

For GSOEP, the variables `anetto` to `znetto` of the file `ppfad.dta` indicate whether a specific person has been interviewed in a specific year and, if not, why not. With the help of these variables, you can precisely specify the observations to be kept. So you could use the information in `ppfad.dta` to first define the observation base and then merge the other files with the option `keep(3)`, which causes `merge` to keep only observations in the `using` data for which the variable `_merge` equals 3.

In our example below, we first construct a new variable `nwaves`, which counts how often a person has been interviewed so far; the variables `anetto`, `bnetto`, etc., have values between 10 and 19 if at least some information was observed in the given year for a specific respondent. We then drop observations with fewer than five interviews (which is an arbitrary number for our example).

After designing the observational database, we start adding variables from the various `p` files. Therefore, we set up a `foreach` loop over all single lowercase letters from “a” to “z”. Inside the loop, we apply the `merge` command as before except that we use the option `keep(3)` and `nogen`. The former keeps only those observations for which the variable `_merge` would equal 3; thus the observational database remains unchanged. The latter keeps the variable `_merge` from being generated; thus we do not need to drop it before we merge the next file.

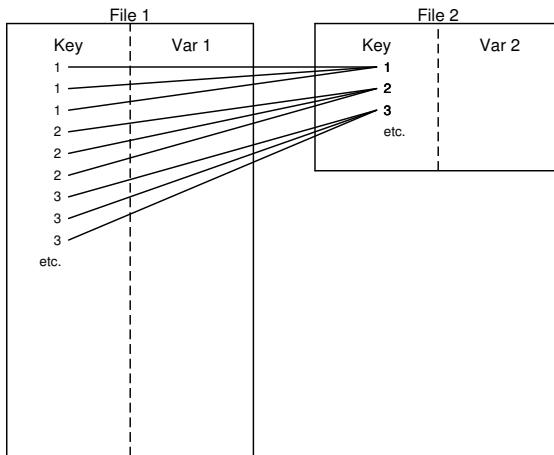
```
. use kksoep/ppfad, clear
. egen nwaves = anycount(?netto), values(10(1)19)
. drop if nwaves < 5
. foreach stub in a b c d e f g h i j k l m n o p q r s t u v w x y z {
.     merge 1:1 persnr using kksoep/`stub`p, keep(3) nogen
. }
```

Merging m:1 and 1:m matches

So far, we have merged datasets where the observations have the same meanings. That is, in any file we merged, an observation was a respondent. Now consider a case in which the meanings of the observations in the files are not identical. Such a case arises if you want to merge the person datasets of the GSOEP with the household data. Whereas the observations of the person datasets are individuals, the observations of the household dataset are households. The same case arises if you want to merge the population sizes used in section 11.2.2 with the person data from the GSOEP. Here the observations are states in one file and persons in the other.

To merge data with different observational levels, you first need to know how the observations of the different datasets are linked. This is quite obvious for the personal and household data of the GSOEP. The persons described by the person data live in the households of the household data. The same is true for persons and states: the persons represented in person data live in the states of the state data. In each case, the observations of any of those datasets somehow belong together. Second, you need to decide for which observational units the analysis should be performed after merging the files. Should the observations of the new data be persons, households, or states? This crucial decision guides the entire remaining process, and the answer depends entirely on your research question. Do you want to make a statement about persons, households, or states? In the following example, we will first merge households and states with the person data by keeping the data as person data. Afterward, we do the opposite.

The `hbrutto` file of the GSOEP has data on households. If we are to add the household data to person data, each person belonging to the same household should get the same information from the household data. Because in this situation one observation from the household data has to be added to many observations of the person data, we call that an `m:1` or a `1:m` match, depending on which of the two files is the master dataset. Figure 11.5 shows an `m:1` match, where the person data comprise the master dataset and the household data comprise the using dataset.



Source: grmerge2.do

Figure 11.5. Representation of `merge` for `m:1` matches

To merge datasets in which the observations form an `m:1` match, we specify `m:1` as match type. However, as the household data do not contain the variable `persnr`, this variable cannot be used as a key. Instead, the keys for our merging problem are the current household number. Each `hbrutto` file contains the household number of the current year, and this variable is also part of every person's data. To merge household information from, say, the 2009 household file `zhbrutto` with the 2009 person file `zp`, we thus specify

```
. use kksoep/zp, clear
(PPFAD: 09/10/10 10:14:13-634 DB09)
. merge m:1 zhhnr using kksoep/zhbrutto,
      Result          # of obs.
      _____
      not matched      0
      matched          5,306  (_merge==3)
      _____
. drop _merge
```

It does not matter which of the two files you specify as master and `using`. However, if you merge the files the other way around, you have to specify `1:m` as match type.

The same logic principally applies if we want to merge the population sizes in the file `popst1.dta` (see section 11.2) with our present dataset. However, as it is frequently the case, some difficulties arise with that example. The command

```
. describe using popst1
Contains data                                Population 2009
obs:                16                         13 Feb 2012 17:08
vars:                 6
size:            992

variable name   storage  display    value
          type   format   label      variable label
state_name      str22  %22s       State (name)
area           double  %8.0f     Area in km^2
pop_total       double  %8.0f     Population
pop_male        double  %8.0f     Population (male)
pop_female      double  %8.0f     Population (female)
pop_dens         float   %8.0f    Population density

Sorted by: state_name
```

tells us that `popst1.dta` contains the variable `state_name`, which holds the name of each German state, while the present dataset contains the numeric variable `zbula`, which also holds information about the state names. Unfortunately, however, both variables are different in several aspects. They have different names, different formats—string versus numeric—and slightly different information: `state_name` has the names of each German state while `zbula` does not distinguish between the states Rheinland-Pfalz and Saarland.

To merge the two datasets, we need to construct a state variable that is strictly comparable in both datasets. The variable should have the same name, the same format, and, most importantly, the same content. Because we cannot regain the difference between Rheinland-Pfalz and Saarland in our present dataset, we need to collapse the information of those two states in the file `popst1.dta`. Therefore, we save our present dataset as it is

```
. save tomerge, replace
```

and load `popst1.dta` into memory:

```
. use popst1, replace
```

We then apply `encode` to create the numerical variable `zbulia` from the string variable `state_name`. Defining a label before using `encode`, let us control the numerical code used for each state:

```
. label define zbulia 0 "Berlin" 1 "Schleswig-Holstein" 2 "Hamburg"
> 3 "Niedersachsen" 4 "Bremen" 5 "Nordrhein-Westfalen"
> 6 "Hessen" 7 "Rheinland-Pfalz" 8 "Baden-Wuerttemberg"
> 9 "Bayern" 10 "Saarland" 12 "Mecklenburg-Vorpommern"
> 13 "Brandenburg" 14 "Sachsen-Anhalt" 15 "Thueringen"
> 16 "Sachsen"
. encode state_name, generate(zbulia) label(zbulia)
```

This assigns the same numbers to each state as the numbers assigned in the other dataset. We use the code 10 for Saarland because this has not been used so far.

On the next step, we combine Rheinland-Pfalz and Saarland into one category:

```
. replace zbulia = 7 if zbulia == 10
```

We then sum up the area and population variables of the combined state using `collapse`:

```
. collapse (sum) area pop_total pop_male pop_female, by(zbulia)
```

We re-create the population density of all states, including the new unified state in the south east of Germany.

```
. generate pop_dens = pop_total/area
```

Finally, we only keep the second of the two observations from which we created the joined category:

```
. bysort zbulia: keep if _n==_N
```

Now we are ready to merge the dataset `tomerge.dta`, which we saved above, with our dataset in memory. This time, we merge many observations with each state, so we need to specify `1:m` as match type. However, before we do that, please save the present dataset for later use:

```
. save popst1V2
. merge 1:m zbulia using tomerge, assert(3) nogen
```

The option `assert()` is used here to check whether `_merge` is equal to 3 for all observations. You would get an error message if the assertion is not true. If the command runs without error, the creation of the variable `_merge` is not necessary, and we therefore suppress its creation by using the option `nogen`. The last two examples resulted in datasets with the respondents as units of analysis. Now for the other way around: suppose that you want to merge person data with household data, or person data with state data, and you want to end up with household or state data, respectively. Here the merging process involves deciding how the values associated with persons can be aggregated. Consider general life satisfaction. Each person in a household has his or her own general life satisfaction. If you want to end up with a household dataset, you

need to decide how this different information from one household can be summarized. It might make sense to consider the mean general life satisfaction of all persons from one household. Or you might use the general life satisfaction of the head of the household and merge this information with the household data. The answer once more depends on your research question. But after you decide, you must form a new aggregated dataset before you merge your data with the other dataset. The command for forming an aggregated dataset is **collapse**. For example, the command

```
. collapse (mean) zp15701, by(zhhnr)
```

calculates the means of the general life satisfaction from 2009 by current household number (**zhhnr**) and stores the result in a new dataset. This dataset is now no longer person data but household data. Merging this file with household data therefore becomes a 1:1 matching type:

```
. merge zhhnr using kksoep/zhbrutto
```

And, of course, the same applies if you want to form state data:

```
. collapse (mean) zp15701, by(zbula)
. merge 1:1 zbula using popst1V2
```

11.4.3 The append command

Adding observations to an existing dataset is straightforward. The **append** command adds a new dataset to the bottom of an existing dataset. This means that you extend the data matrix of your current dataset by one or several rows of observations. Identical variables are written one below the other, whereas new variables are added in new columns (see figure 11.6).

File 1		
Var 1	Var 2	
0	2002	
1	1876	
0	3000	
0	2130	
1	1000	
etc.	etc.	

File 2		
Var 1	Var 2	Var 3
0	1238	7
1	1500	9
etc.	etc.	etc.

Source: `grappend.do`

Figure 11.6. Representation of `append`

The structure and function of `append` are simple. The basic syntax is

`append using filename [filename ...] [, nolabel]`

Let us illustrate this with an example. We have mentioned before that the GSOEP database contains PSID-equivalence files, which are part of the Cross-National Equivalent File (CNEF) set assembled by the Cornell College of Human Ecology. The CNEF contains files with equivalently defined variables for panel studies from the United States, Germany, the United Kingdom, and Canada.¹⁵ Like those of the GSOEP, the data of the CNEF are also split into several files: one file for each year and each country. Among the files you installed, we have inserted one of them: a downsized version of the file with data from the 2007 U.S. PSID: `pequiv07kk.dta`. You might take a look at it with

15. The panel data distributed in the CNEF are the Panel Study of Income Dynamics (PSID), the German Socio-Economic Panel (GSOEP), the British Household Panel Study (BHPS), the Canadian Survey of Labor and Income Dynamics (SLID), and several others. See <http://www.human.cornell.edu/PAM/Research/Centers-Programs/German-Panel/cnef.cfm> for a more complete description of the CNEF.

```
. describe using pequiv07kk
Contains data
    obs:      22,102                               13 Feb 2012 17:08
    vars:        11
    size:   906,182

variable name  storage  display  value
          type  format  label  variable label
x1110111      long    %12.0g  person identification number
x1110207      int     %8.0g  hh identification number
d1110407      byte    %8.0g  marital status of individual
d1110607      byte    %8.0g  number of persons in hh
d1110707      byte    %8.0g  number of children in hh
d1110907      byte    %8.0g  number of years of education
e1110107      int     %8.0g  annual work hours of individual
e1110307      byte    %8.0g  employment level of individual
i1110107      double  %10.0g hh pre-government income
i1110207      double  %10.0g hh post-government income
i1110007      double  %10.0g individual labor earnings

Sorted by: x1110111
```

This file by and large contains the same variables as the file `xpequiv.dta` from the GSOEP. All variables have equivalent names in both files except for the unique person and household numbers, which are called `persnr` and `hhnr` in the GSOEP, but `x1110111` and `x1110207` in the PSID.

For a cross-national comparison between Germany and the United States, you need to combine the two files. You begin the process by loading one of the datasets into Stata. Which one you load does not matter, but we begin with the U.S. file `pequiv07kk`:

```
. use pequiv07kk, clear
```

Again we will use the term *master file* for the file in the working memory. You can examine that file using `describe`, and you will see that there are 22,102 observations and 11 variables in the data file. One of the variables is `e1110107`, the annual work hours of the respondents in the U.S. dataset.

Before you append another file to the master file, you should harmonize the variable names

```
. rename x1110111 persnr
. rename x1110207 xhhnr
```

and generate a variable that marks the observations of the master file. This variable will later help you to separate U.S. respondents from German respondents. We will do this by constructing a string variable that is “United States” for all observations in the current file:

```
. generate country = "United States"
```

Using `append`, you can now add the respective file of the GSOEP at the bottom of `pequiv07kk`. The data for Germany are stored in the subdirectory `kksoep`. The command contains the directory name and the file name (`kksoep/xpequiv`):

```
. append using kksoep/xpequiv
```

The new dataset has now 28,068 observations: 22,102 from the United States and 5,966 from Germany. Variables that are in both datasets can have valid values for all observations. On the other hand, variables that are part of only one of the two datasets get a missing value for any observation from the dataset without that variable. Therefore, the variable `country` created above is missing for all German respondents:

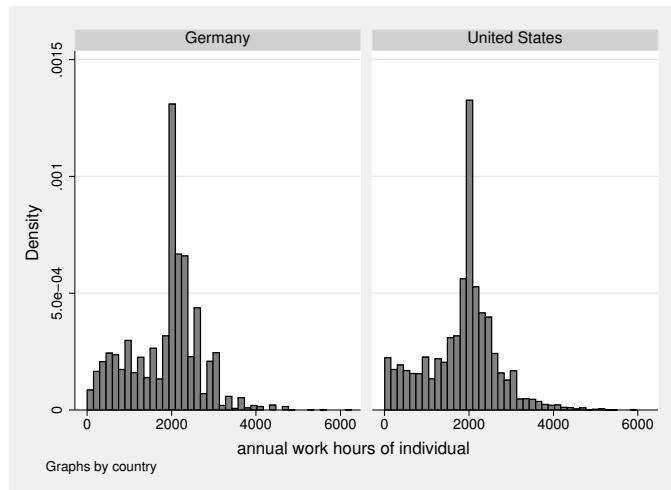
```
. describe, short
Contains data from pequiv07kk.dta
    obs:          28,068
    vars:           13
    size:      1,656,012
Sorted by:
    Note: dataset has changed since last saved
. tabulate country, missing


| country       | Freq.  | Percent | Cum.   |
|---------------|--------|---------|--------|
| United States | 5,966  | 21.26   | 21.26  |
|               | 22,102 | 78.74   | 100.00 |
| Total         | 28,068 | 100.00  |        |


```

After replacing the missing value in `country` with, say, “Germany”, you can use `country` to make the intended comparison:

```
. replace country = "Germany" if missing(country)
. histogram e1110107 if e1110107 > 0, by(country)
```



Two small points that might come in handy when working with `append`: First, we could have generated a numerical version of the `country` variable with the option `generate` of `append`. This was less practical for us because we use the country labels in the graphs. However, in many settings such a numerical variable will be sufficient. Second, you should know that you can `append` more than just one file within a single command by listing all relevant files behind `using`.

11.5 Saving and exporting data

You have taken a lot of trouble to construct a dataset with which you and your colleagues can easily work. Saving this dataset, on the other hand, is straightforward: with the command `save filename`, you save the dataset in memory with the name `filename.dta` as a Stata system file in the current working directory. With the option `replace`, you can overwrite a previous version of the same dataset.

Saving the data is therefore fairly easy. Just before saving, however, you should consider three issues:

1. If no data type is specified, variables are by default created as *floats* (see section 5.7). These variables often take up more storage space than is necessary. The command `compress` optimizes all variables with regard to their storage space, with no loss of information. You should therefore always use `compress` before saving.
2. Working with a dataset is often easier if the variables are in a certain order. The commands `order` and `aorder` can be used to order the variables in the dataset.
3. Like variables and values, the dataset can also be given a label. The corresponding command is `label data`. The data label is shown in the output of `describe` on the upper right. In `data1.dta`, for example, it is “SOEP 2009 (Kohler/Kreuter)”.

So far, we have focused on the different ways of reading data from other formats into Stata. At times, however, you will need to read a Stata dataset into another program. Of course, you can do this by using the software packages described in section 11.2.1. Stata also has export filters for Excel (`export excel`) and SAS XPORT Transport format files (`export sasxport`). However, because practically all statistical packages can read ASCII files, it is just as easy to save your Stata data to an ASCII file.

In Stata, you can save ASCII files in spreadsheet format and in free format. To save in spreadsheet format, you can use `outsheet`; to save in free or fixed format, you can use `outfile`. Both commands basically work the same way: after the command, you can specify a variable list followed by the code word `using` and the filename.

For both commands, the contents of string variables are written between quotes. If you do not want to use quotes, specify the `noquote` option. Important: Variables with value labels are saved as string variables unless you use the `nolabel` option to prevent this.

11.6 Handling large datasets

Stata always loads the entire dataset into the computer's working memory. On modern computers, this should not provide any problems, and Stata is quite good in managing working memory. However, if you find yourself faced with the error message "no room to add more variables", you should know a bit about working memory and the Stata commands `memory` and `query memory`. The first will indicate the current memory usage and the allocated memory. The second will provide ways to change the allocated memory.

11.6.1 Rules for handling the working memory

The working memory is part of the hardware of your computer. It consists of two parts, the physical and the virtual working memory. The physical working memory consists of special chips (RAM), which quickly remember and retrieve pieces of information. The virtual working memory is a file on the hard disk (a swap file), which in principle works like physical memory but much more slowly.

Stata loads its data into the working memory. Whether the data then end up in physical memory or the swap file is determined by the operating system. If the data end up in physical memory, the processor can access them quickly; if they end up in the swap file, it takes longer. You can even hear the access to the swap file because it will involve a lot of reading and writing on the hard disk.

For Stata to work efficiently, the datasets should be kept in physical memory if possible. Generally, the swap file is used only when the physical memory is full. The memory does not, however, contain only Stata data. The operating system needs part of the memory, and every program that is running uses memory. So, to prevent the first problem, close all applications that you do not really need.

Programs likely to use a lot of memory include Internet browsers and various office packages.

Closing applications, unfortunately, does not automatically eliminate the error message "no room to add more variables". This message appears if the working memory is too small to store another variable. Many Stata procedures create temporary variables, so that error message may appear even when you are not trying to create or load any new variables.

You can find out how much working memory you have from

```
. query memory
```

See `help memory` for more information on how Stata uses memory and on settings that you can tweak on your system.

11.6.2 Using oversized datasets

Ideally, the amount of working memory reserved would be large enough to easily fit the dataset and small enough to prevent the use of the swap file. For some datasets, it is impossible to comply with this rule. This is the case for datasets that are larger than the RAM. If you often work with such datasets, we have only one piece of advice: buy more RAM. If this is an exceptional situation, we suggest the following:

- Do you need all variables in the dataset? If not, load only the variables you need:


```
. use persnr hhnry income emp using data1, clear
```
- Do you need all observations in the dataset? If not, import only the observations you need:


```
. use data1 if state <= 9, clear
```
- Does your dataset take up more storage space than necessary? Try reading in your dataset a little at a time and optimizing it. To do so, you first import only specific observations or variables, optimize the storage space used by this partial dataset with `compress`, and save it under a new name. Repeat this procedure for the remaining variables or observations and join the different partial datasets using `merge` or `append` (section 11.4).¹⁶
- Does your dataset contain many identical observations? You should transform the dataset into a frequency-weighted dataset (page 66).

11.7 Exercises

- Obesity
 1. Point your browser to the interactive database on European Quality of Life of the European Foundation for the Improvement of Working and Living Conditions¹⁷. On this webpage, find the tables for the percentage of people who are very overweight. Create a Stata dataset holding this table.
 2. Merge this file with the information on the share of people who have strong underweight from the same website. Save this file for later use.
 3. Download data from the National Health and Nutrition Examination Study (NHANES) using the following command:

```
. webuse nhanes2, clear
```

16. If you do not know beforehand which variables or how many observations your dataset contains, see the dataset with `describe using data1.dta`. You can use the `describe` command even without loading the data into Stata's working memory. The only condition is that you add `using` to the command.

17. <http://www.eurofound.europa.eu/areas/qualityoflife/eurlife/index.php>

4. By using the definition of the European Foundation for the Improvement of Working and Living Conditions (ignoring the differences in the age of the observations), produce an aggregated dataset for the share of people who have strong overweight and strong underweight by region.
5. Produce an integrated dataset of the American figures and European figures from 2005.

- Income

1. Merge the files `ap` and `apequiv` of the subdirectory `kksoep` of the file package of this book.
2. Produce a dataset holding the “HH Post-Government Income” of all the GSOEP equivalence files (`kksoep/*pequiv`). Use only those respondents that have records in each of the data files (this is called *balanced panel design*).
3. Produce the same dataset as above, but this time use those respondents for whom there are records in at least 10 data files.
4. Merge the files `ahbrutto` and `ap`. Explore whether there is a difference when you reverse the order in which you merge the two data files.
5. Create a household file out of `ap`, that is, a data file where each household is one observational unit (row). Add `ahbrutto` to this file.

12 Do-files for advanced users and user-written programs

When working on a substantial data analysis, you will often want to reuse a previous command. For example, you might want to repeat your analysis or replicate it with different data, or your dataset might contain a series of similar variables that should be treated in the same way. Perhaps during your analysis, you are recalculating certain measured values or creating certain graphs for which there are no existing commands in Stata. In all of these situations, you can save work by learning the programming tools that we will introduce in this chapter.

We will distinguish among four programming tools: macros, do-files, programs, and ado-files. We will cover these in detail in section 12.2. In section 12.3, we will show you how you can use these tools to create your own Stata commands. However, before we start, we will give two examples of possible ways to use these tools.

12.1 Two examples of usage

Example 1

Imagine that you are working with one of our example datasets, the panel dataset `progex.dta`. The dataset contains the life satisfaction of 506 interviewees who participated in all survey waves of the German Socio-Economic Panel (GSOEP). The dataset is organized so that answers from every survey wave are in a different variable. You can view the structure of the dataset through the following commands:¹

```
. use progex  
. describe  
. list in 1
```

The variable for the respondent's life satisfaction appears 26 times in this file. You will find the data from 1984 in `ap6801`, those from 1985 in `bp9301`, those from 1986 in `cp9601`, and so on (see section 11.4.1 for an explanation of the structure of variable names in the GSOEP).

1. Make sure that your working directory is `c:\data\kk3`. More information on this can be found on page 3.

Imagine that you want to generate new variables, which differ between respondents with a life satisfaction below and above the yearly average life satisfaction. You would have to enter commands like the following for each year:

```
. label define lsat .a "No answer" 0 "below avarage" 1 "above avarage"
. summarize ap6801 if ap6801 > -1
. generate lsat1984:lsat = ap6801 > r(mean) if ap6801 > -1
. replace lsat1984 = .a if ap6801 == -1
. label variable lsat1984 "Life satisfaction above average (1984)"
```

Repeat these commands (except the first) for the answers from 1985, 1986, and so on. After 104 commands, you will have generated and labeled your variables. Naturally, you will not have created all of these commands interactively, but you will have entered the above block into a do-file. Once you have done this, you can simply copy the entire block 26 times and change the years in the variable names. Nevertheless, this still requires a lot of work and is comparatively error prone.

What would happen if you wanted to use the median as the differentiation point instead of average and if you wanted to put the missing category -1 into a category 3 for “other”? Or what if you wanted to separate those responses with a life satisfaction of 0 and 10, respectively?

You can probably guess what we are driving at. Overall, it seems slightly unwieldy to have a do-file that repeats only one set of commands. One solution for this problem can be found in the do-file `crlsat.do`, which you already downloaded. In this file, we used a `foreach`-loop (see section 3.2.2) together with more tricks that we will discuss in this chapter.

Example 2

Suppose that when carrying out logistic regressions, you would rather specify Aldrich and Nelson’s pseudo- R^2 (Aldrich and Nelson 1984, 57) than McFadden’s, which is the one calculated by Stata (McFadden 1973). To do this, you can calculate the value with the `display` command after a logistic regression:

```
. generate satisfied1984 = ap6801==10
. logit satisfied1984 ybirth
. display "p2 = " e(chi2)/(e(chi2) + e(N))
```

However, instead of writing the equation each time, you might write a small ado-file to accomplish this task. If you save the ado-file on your hard drive, in the future you can enter the name of the ado-file after the logistic regression to get Aldrich and Nelson’s pseudo- R^2 . Our version of this ado-file can be installed by typing²

```
. net install p2, from(http://www.stata-press.com/data/kk3/)
```

2. Notes on installing ado-files can be found in chapter 13.

Now please load our `data1.dta`.

```
. use data1, clear
```

12.2 Four programming tools

Here we discuss several features of Stata that you will find yourself using repeatedly. We begin with an introduction to local macros, which allow you to store results, strings, and other items. We then show you how to write your own commands and parse syntax. We end with discussions of do-files and ado-files.

12.2.1 Local macros

We have already dealt with local macros in chapter 4 and covered the long-term storage of internal results in various parts of the book. Here we want to introduce local macros as a more general concept.

A local macro contains strings of characters. When you define a local macro, a name is given to these characters. Once you have defined it, you can enter the name of the macro instead of the characters.

You can define a local macro with the command `local`. Let us use this command to define the local macro `a`:

```
. local a income yedu ybirth
```

This command assigns the string of characters `income yedu ybirth` to the macro name `a`.

If you wish to use the content of local macro `a`, you have to inform Stata that “`a`” is the name of a local macro and not the name of a variable. To do this, you place a single opening quotation mark (‘) before the name of the local macro and a single closing quotation mark (’) after it. On many American keyboards, the opening quote is found toward the top left (near the *Esc* key), whereas the closing quote is found toward the right, near the *Enter* key. On European keyboards, the position of both characters changes substantially from country to country. Often the opening quote is used to produce the French accent grave, which forces you to press the space key before the sign appears on the screen.

Whenever Stata comes across a local macro, the macro is immediately replaced by its contents. Whenever you have to type `income yedu ybirth`, you can instead simply type the macro name, for example,

<code>. summarize `a'</code>					
Variable	Obs	Mean	Std. Dev.	Min	Max
income	4779	20540.6	37422.49	0	897756
yedu	5039	11.80419	2.676028	8.7	18
ybirth	5411	1959.493	18.12642	1909	1992

With this command, Stata sees `summarize income yedu ybirth` once the macro has been expanded. If it does not, then you have probably used the wrong characters for identifying the local macro. Make sure that you have used the correct quotation marks around the macro name `a`.

Local macros enable you to save a lot of typing. However, remember three key points if you are using local macros for this purpose:

- The name of a local macro must be no more than 31 characters long.
- A local macro can contain up to 165,200 characters. This limit is for Stata/IC. For Small Stata, the limit is 8,681. For Stata/MP and Stata/SE, the limit is 1,081,511 characters.
- Local macros apply only within the environment in which they are defined. If you define a local macro in a do-file, the macro is accessible only in that do-file. If the do-file has finished, the local macro is no longer defined. If you interactively define a local macro, Stata will recognize it as long as you are working interactively. As soon as you start a do-file, you cannot use the interactively defined macro. You can use the interactively defined macro again only once the do-file has finished and you have begun working interactively again. One advantage of this is that you can use the same macro names in different contexts without having to worry about mixing them up.

Calculating with local macros

Besides using local macros to save on typing, you use them to carry out calculations. There are two options, which we show in this example:

```
. local m1 2+2
. local m2 = 2+2
```

The commands differ by the use of the equal sign. Both commands work and apparently lead to the same result:

```
. display `m1'
4
. display `m2'
4
```

However, there is one difference, which you will see if you embed the macro name in quotation marks. With quotation marks, `display` displays text, whereas without quotation marks, `display` shows the value of an expression (section 3.1.6). For this reason, the results above show the evaluation of the expression saved in the local macro. In both instances, the result was 4. But if you display the contents of the local macros as text, then you can see that the first macro contains the expression, whereas the second contains the result.

```
. display "`m1"
2+2
. display "`m2"
4
```

Note a crucial difference when multiplying

```
. display 2*m1
6
```

because $2 \times 2 + 2 = 6$ and

```
. display 2*m2
8
```

because $2 \times 4 = 8$.

Combining local macros

You can combine several local macros. Here are a few examples. Type these commands to get some practice and understanding of local macros. However, before typing these command lines, remember to check that your working directory (page 3) contains our do-files and datasets:

```
. local a dir *
. local b dta
. local c do
. `a``b'
. `a``c'
. local b `a``b'
. display ``b``
. `b'
```

Changing local macros

Imagine that you wanted to add the number 1 to an existing macro, in this case, `i`, which contains the number 5. The command is as follows:

```
. local i 5
. local i = `i' + 1
```

In the second command, the macro `i` on the right side of the equation must be embedded in the characters ‘ and ’, whereas the macro `i` on the left side must not. The reason is that ‘`i`’ is replaced by the number 5; the command is therefore converted into `local i = 5 + 1`. If you had enclosed the `i` on the left side in quotation marks, the command would have become `local 5 = 5+1`. This, in fact, would have created a new local macro with the name “5”.

Besides this minor glitch, it is not difficult to modify local macros. Here are some examples:

```
. local i 1
. local i = `i' + 10
. display `i'
11
. local i = `i' + `i'
. display `i'
22
. local i "i is `i'"
. display "`i'"
i is 22
```

You can redefine a local macro whenever you want with the contents of an expression, a string, or an extended macro function (see section 12.3.8). Two commonly used macro-extension operators add or subtract the number 1 from a macro, either immediately before Stata expands the macro or immediately thereafter. Consider, for example,

```
. local i 1
. display `i++'
1
. display `i'
2
```

Here we first define the local macro `i`. Afterward, we display `i` with an extension operator (`++`), which means that the macro is expanded, and then 1 is added to the macro; that is, the content of the macro is changed to 2. If you put the extension operator in front of the macro name, 1 is added to the macro before the macro is expanded. The same applies to the extension operator `--` for subtracting 1:

```
. display `++i'
3
. display `i--'
3
. display `--i'
1
```

12.2.2 Do-files

Say that you want your computer to display the words “hello, world”. To achieve this, you can type the following command:

```
. display "hello, world"
hello, world
```

You would have to repeatedly type this command every time you wanted to see “hello, world” displayed. Admittedly, this is not too much of a problem, because the command is not very long. Nevertheless, the command could be longer, so you might want to know how to save yourself some work.

Enter the following do-file and save it under the name `hello.do` in your working directory. As always, remember to write only the Stata commands. The solid line with the filename and the line number indicate that the commands are placed into a file:

```
----- begin: hello.do -----
1: display "hello, again"
2: exit
----- end: hello.do -----
```

Once you have done this, you can execute `hello.do` by typing

```
. do hello
```

We get the same result that we would get by entering `display "hello, world"`, but now we just have to type `do hello` to see the message again. Of course, a do-file will typically have many more lines of code.

12.2.3 Programs

Besides do-files, the command `program` offers a second option for accessing many commands at the same time. The way `program` works can be better illustrated through an example. Please type the commands below. The first line begins the definition of a program named `hello`; you may use any other name that it is not longer than 31 characters. When you are finished with the first line and press the *Enter* key, you will see that Stata displays 1. at the beginning of the next line. This is Stata’s way of prompting you to type the first command of the program. Please type the command without typing the number.

```
. program hello
 1. display "Hello, world"
 2. end
```

When you confirm this entry by pressing *Enter*, the prompt for the second command that the program should carry out appears on the screen. Here we want to end the program, so we type `end` in the Command window. This will now return you to the normal prompt.

Now that the program is defined, you can type `hello` in the Command window:

```
. hello
Hello, world
```

You entered `hello`, and Stata replied with “Hello, world”.

The `program` command defines programs. However, in contrast to do-files, these programs are stored in the computer’s memory and not in a file on the computer’s hard disk. If you type some word into the Stata Command window, Stata will look in the memory (see section 11.6) for something called the same as the word you just entered into the command line. Therefore, if you enter `hello` in the window, Stata will search the memory for a program called `hello` and will find the `hello` program you previously saved there with the `program` command. The Stata commands between `program` and `end` are then carried out.

In many respects, these programs are similar to the do-files you have already seen. However, there are some differences:

- Do-files are saved in a file on the computer’s hard drive, whereas programs are stored in memory.
- Do-files are not deleted when Stata is closed or the computer is shut down. Programs are lost when Stata is closed.
- A do-file is accessed by typing `do filename`, whereas a program is accessed by typing the program name without any command in front.
- Because Stata searches for programs in memory, programs must be loaded before they can be accessed. Do-files must be saved to the hard drive.
- Do-files display the results of Stata commands that have been carried out, as well as the commands themselves. Programs only display the results.

The most important difference between do-files and programs is that do-files remain available for long-term access. Programs, on the other hand, are available only during a Stata session. In the following section, we will therefore concern ourselves with the options for storing programs over a long period. Before we do this, we need to look at several typical problems that may arise when saving and accessing programs.

The problem of redefinition

Imagine that you want your computer to display “Hi, back” instead of “Hello, world” when you type `hello`. So you try reentering the program:

```
. program hello
hello already defined
r(110);
```

Stata knows that a program in memory is already named “hello” and does not allow you to overwrite it. First, you must delete the old version from the RAM before you can create the new version:

```
. program drop hello
. program hello
  1. display "Hi, back"
  2. end
. hello
Hi, back
```

The problem of naming

Imagine that you wanted the program to be called `q` instead of `hello`:

```
. program q
. display "Hello, world"
. end
. q
```

Surprisingly, this displays the settings of various Stata parameters. The reason is that the letter `q` is a shortcut for the Stata command `query`, which is a “built-in” Stata command. Stata searches for programs only when it has not found a built-in Stata command with the specified name. Thus you should never define a program with the same name as a built-in command.

To ascertain whether a command with a given name already exists, enter `which commandname`. If Stata replies with “Command `commandname` not found as either built-in or ado-file”, you may use the name for your program.

The problem of error checking

Stata checks the syntax of a program only when it has been executed:

```
. program hello2
  1. displai "Hello, world"
  2. end
```

Here `displai` was entered instead of `display`, so Stata will detect an error when executing the program:

```
. hello2
unrecognized command: displai
r(199);
```

Because the individual commands are not repeated on screen as they are in do-files, it is often hard to find an incorrectly entered command in lengthy programs. By using the command `set trace on`, you can instruct Stata to display a program's commands. This enables you to follow the program command by command while it is being executed and thus find the incorrect command. Lines that are to be executed begin with a hyphen. If a line in the program contains a macro, another line beginning with an equal sign shows the line with expanded macros. Unfortunately, the trace creates a large volume of output when it is dealing with lengthy programs. So do not forget to switch off the trace once you have found the error in a program: `set trace off`. See [P] `trace` or `help trace` for more information.

12.2.4 Programs in do-files and ado-files

Earlier we interactively stored a program by entering the respective commands in the Command window. One of the disadvantages of this process is that the stored programs are lost when the Stata session ends (at the latest). Another disadvantage is that you are unlikely to enter a lengthy program without typos, which will become apparent only when the program is executed. To correct the typo, you will have to reenter the entire program. If you want to store a program for a long time, then you have to enter the definition of the program in a do-file.

To help you learn how to define programs in do-files, you should rewrite the file `hello.do`, which you created on page 443, as follows:

```
begin: hello.do
1: program hello
2:   display "hello, again"
3: end
4: exit


---


end: hello.do
```

This do-file contains a new version of the `hello` program from the previous section. The difference is that our program should now display “hello, again”. More importantly, the definition of the program should now be written in a do-file. We have slightly indented the list of commands between `program` and `end`. This helps us to find the beginning and end of the program definition, but does not affect Stata in any way. Now that the program definition is written into the do-file, we do not have to rewrite the definition of the program every time we close Stata; executing the do-file is sufficient. Let us try it. Please save the do-file and execute it. If you have followed all our steps in section 12.2.3, then you should receive an error message:

```
. do hello
. program hello
hello already defined
r(110);
```

The reason for the error message is that Stata recognizes the names of the programs already loaded in memory and will not allow them to be overwritten. Because we have already interactively defined `hello`, it is currently being stored in memory. Therefore, you must delete this older version before you can create the new version. This is best done directly in the do-file through `capture program drop hello`. By using `program drop hello`, the `hello` program is deleted from memory; `capture` ensures that no error messages will follow if there is no such program (also see section 2.2.3). Change `hello.do` to

```
begin: hello.do
1: capture program drop hello
2: program hello
3:   display "hello, again"
4: end
5: exit
```

Save these changes and try it again:

```
. do hello
```

The program `hello` has not been executed, because it has been stored in memory only through the commands in the do-file. However, we can now execute the program interactively:

```
. hello
hello, again
```

We can also call the program directly in the do-file as follows:

```
begin: hello.do
1: capture program drop hello
2: program hello
3:   display "hello, again"
4: end
5: hello // <- Here we call the execution of the program
6: exit
```

Here the program is first defined and then executed. Give it a try:

```
. do hello
```

A further option for saving programs is offered by ado-files, which are also known as “ados”. How ados work becomes clearer if we go back one step. First, delete the program `hello` from memory:

```
. program drop hello
```

Then reload `hello.do` in your Editor. Remove the commands in `hello.do` related to deleting and calling up the program. The file `hello.do` should now look like this:

```
begin: hello.do
1: program hello
2:     display "hello, again"
3: end
4: exit
end: hello.do
```

Once you have saved it, the do-file can be executed with

```
. run hello
```

The command `run` is the same as the `do` command, the only difference being that after you enter `run`, the individual command lines do not appear on the screen. If you enter `run hello`, `hello.do` is quietly executed while the program `hello` is loaded in memory. After this, you can interactively execute the program:

```
. hello
hello, again
```

Now to the ado-files: If we save a do-file with the extension `.ado` instead of `.do`, the above-mentioned steps are both automatically carried out. Entering the name of the ado-file will suffice. Go ahead and try it. Save the last version of your do-file with the extension `.ado` under the name `hello.ado` and then enter the following:

```
. program drop hello
. hello
hello, again
```

It works. To understand why, let us take a closer look at the steps that take place after `hello`. In general, after a command has been entered, Stata takes the following steps:

1. Stata checks if `hello` is an internal command. If so, then Stata will have executed it. Because `hello` is not an internal command, it moves on to the next step.
2. Stata checks if `hello` is stored in memory. If so, the program would be executed. Because we deleted the `hello` program from memory shortly before entering the command `hello`, Stata would not find a program called `hello` in memory and would move to the next step.
3. Stata searches for the file `hello.ado`. Here Stata searches in various places on the hard drive, including the working directory. When Stata finds the file `hello.ado`, it essentially gives itself the command `run hello.ado` and subsequently checks if the program `hello` is now in memory. If so, as in this example, then the program is executed. If not, then Stata will display the error message “unrecognized command”. If the ado-file contains subprograms, Stata will also load any subprograms of `hello.ado` and make them available solely to `hello`.

Because of the second step, programs defined by ado-files must be deleted from memory before any potential changes to the program take effect. If you rewrite `hello.ado` as

```
----- begin: hello.ado -----
1: program hello
2:   display "hi, back again"
3: end
4: exit
----- end: hello.ado -----
```

and then type `hello`, at first nothing will have changed:

```
. hello
hello, again
```

You must first delete the old program from memory before the ado-file can be reimported. To do so, you can use the `program drop` command or the `discard` command. `discard` forces Stata to drop all automatically loaded programs from memory and reload them from disk the next time they are called; see `help discard` for details.

```
. discard
. hello
hi, back again
```

Let us summarize what we have learned so far: if we enter the command `hello` at the command prompt, Stata will find the program `hello` in memory and then execute it, or Stata will not find it. In the latter case, Stata searches for `hello.ado`, loads the program in memory, and executes it. The command `hello` therefore works just like a normal Stata command. In fact, the command `hello` is a normal Stata command: an “external” Stata command.

Stata generally differentiates between external and internal commands. We have already mentioned the idea of internal commands a couple of times, so we will now define them in more detail. Internal commands are commands programmed in the C programming language and compiled for various hardware platforms and operating systems. External commands, on the other hand, are the ado-files that we have just introduced you to: programs that are saved in files with the extension `.ado`. Nearly all Stata commands are ado-files. If you enter the command `adopath`, you will receive a list of directories that Stata searches for ado-files. You can view the contents of ado-files with any editor.

12.3 User-written Stata commands

As we said at the beginning, ado-files comprise nothing more than the definition of a program. If you enter the name of an ado-file in Stata, the program is stored in memory and then executed. In effect, an ado-file behaves just like every other Stata command; or, as we stated above, the ado-file is a normal Stata command.

To execute an ado-file, you must save it on the hard drive where Stata can find it, such as the personal ado-directory. The command `adopath` will indicate where the personal ado-directory can be found on your computer. In Windows, the personal ado-directory will generally be found at `c:\ado\personal` (see section 13.3.3).

Here, with the help of an example, we will briefly demonstrate how ados are programmed. We will not cover all the programming options available to you. Instead consider this section as a sort of ladder that allows you to climb a fruit tree. To pick the best fruit, you will have to choose the best branches by yourself. All the Stata commands and the tools described in the previous section are available for your use. The book *An Introduction to Stata Programming* by Baum (2009) and the Stata Internet courses NetCourse 151 and NetCourse 152 (section 13.1) will give you a detailed introduction to programming in Stata. Before you take the time to program your own Stata command, you should thoroughly check to see whether someone has already done the work. The Statalist archive and the sources of information listed in section 13.3.3 should help you do this. Nevertheless, it is certainly worthwhile learning a little bit about Stata programming. If you are using ados written by other users, it is especially useful to be able to “read” what is actually happening in Stata code. This is also true for the commands supplied with the program.

In what follows, we will show you step by step how to program a Stata command for superimposed kernel density estimates. We have discussed such graphs in section 7.3.3, and an example is printed on page 195. In modern versions of Stata, these graphs can be relatively easily produced by overlaying several layers of the `twoway` plottype `kdensity`. For didactic reasons, we will use the `twoway` plottype `connected`, which will allow us to talk about some important programming tools later.

If you look at the do-file `denscomp.do` below, you will see that `twoway connected` is used to show the results of the variables `fmen` and `fwomen`, which are created by the stand-alone version of `kdensity` in lines 5 and 6. In the two `kdensity` commands, the option `at()` is being used. The `at()` option specifies the values of a variable at which the densities should be estimated. Here they are taken from the variable `xhelp`, which was generated in lines 3 and 4. Those two commands generate a variable that holds the values of the log income variable at 50 equally spaced intervals between the minimum and the maximum of log income (see section 7.3.1).

```
begin: denscomp.do
```

```

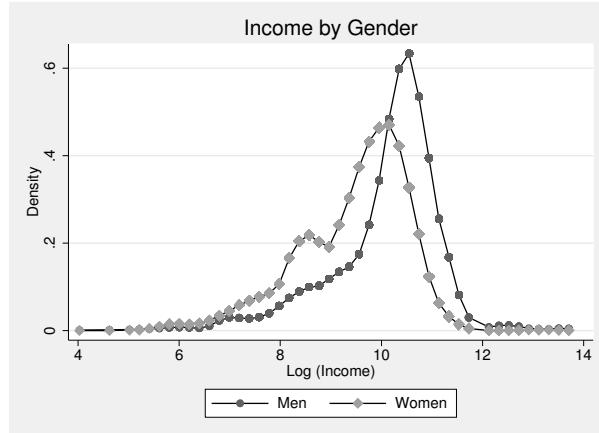
1: use data1, clear
2: generate linc = log(income)
3: summarize linc, meanonly
4: generate xhelp = autocode(linc,50,r(min),r(max))
5: kdensity linc if sex == 1, gen(fmen) at(xhelp) nodraw
6: kdensity linc if sex == 2, gen(fwomen) at(xhelp) nodraw
7: graph twoway connected fmen fwomen xhelp, ///
8:     title("Income by Gender") ytitle("Density") xtitle("Log (Income)") ///
9:     legend(order(1 "Men" 2 "Women")) sort
10: exit

```

```
end: denscomp.do
```

If you run this do-file, you will get a graph that connects the density values estimated at the 50 equally spaced intervals of log income for men and women of `data1.dta`. Starting from such a do-file, one might now try to generalize the commands so that they can be run with any variable from any dataset.

```
. run denscomp
```



12.3.1 Sketch of the syntax

The first step in programming a command is sketching the syntax, which includes choosing a name for the program and which elements of the Stata language (section 3.1) should be allowed. Regarding the name, you should use a name that does not exist already; the command `search, all` (page 475) is the most efficient way to find out whether a specific name already exists. Moreover, proper English words, such as `analyze`, `table`, or `predict`, and single-letter names should be avoided. Leaving this aside, you can create arbitrary names using combinations of lowercase letters, numbers, and the underscore. Our program is designed to compare densities; thus we choose the name `denscomp` and propose the following sketch of the syntax:

```
denscomp varname [ if ] [ in ], by(varname) [ at(#) twoway_options ]
```

Our command requires the specification of a variable name and the option `by(varname)`. To re-create our example above, we would specify `varname` as `linc`, the `by`-option as `by(sex)`, and the option `at()` with 50. The command allows `if` and `in` qualifiers and all options allowed with `graph twoway`.

12.3.2 Create a first ado-file

The generalization of the do-file starts with a first version of an ado-file. This first version is almost identical to the above `denscomp.do`. In fact, you can just save `denscomp.do` with the name `denscomp.ado`. Once you have done this, you can make the following changes:

1. Delete the `use` command and the command that generates the variables `linc`.
2. Type the command `program` `denscomp` in the first line of the file.
3. Type the command `version 12` in the second line of the file. You already know this command from the chapter on do-files. It indicates in which Stata version this program has been written. This guarantees that your program will work error free in later Stata versions.
4. Type the command `end` to indicate the end of the program in the next-to-last line of the file. The command `exit` ends the ado-file, not the program.

The preliminary version of `denscomp.ado` should now look like this:

```
----- begin: denscomp.ado -----
1: program denscomp
2: version 12
3: summarize linc, meanonly
4: generate xhelp = autocode(linc,50,r(min),r(max))
5: kdensity linc if sex == 1, generate(fmen) at(xhelp) nodraw
6: kdensity linc if sex == 2, generate(fwomen) at(xhelp) nodraw
7: graph twoway connected fmen fwomen xhelp, ///
8:     title(Income by Sex) ytitle(Density) xtitle(Log (Income)) ///
9:     legend(order(1 "Men" 2 "Women")) sort
10: end
11: exit
----- end: denscomp.ado -----
```

You should check to see that everything still works. Checks such as these should be made after every major step. Save this ado-file in your current working directory and then type `denscomp`.

If you run `denscomp`, you will now receive an error message:

```
. denscomp
xhelp already defined
r(110);
```

The reason is that the variables `xhelp`, `fmen`, and `fwomen` created by `denscomp.do` remain in the dataset. `denscomp.ado` tries to create them; however, because they already exist, you receive an error message. Therefore, you have to first delete `xhelp`, `fmen`, and `fwomen` before you run `denscomp.ado`:

```
. drop xhelp fmen fwomen
. denscomp
(2001 missing values generated)
```

If you did not run `denscomp.ado` above, you will probably receive this error message:

```
. denscomp
variable linc not found
r(111);
```

If you receive instead the error message “no variables defined”, then you have not loaded the dataset. In both cases, load `data1.dta` and create the `linc` variable (see section 7.3.3):

```
. use data1, clear
. generate linc = log(inc)
```

You can try running `denscomp` again after these steps:

```
. denscomp
```

The graph should now be displayed on your screen. We created the graph with one command; however, `denscomp` will re-create the graph only with the same variables and options.

12.3.3 Parsing variable lists

The next step in generalizing `denscomp.ado` is allowing the graph to be displayed for arbitrary variables.

You can achieve this with the `syntax` command. The `syntax` command is used to define the structure with which a user is to call our program. Here we assume that the program is accessed through the name of a command and a variable name. This setup is parsed to the program in line 3 of the following program snippet:

```
----- begin: denscomp.ado -----
1: program denscomp
2: version 12
3: syntax varname(numeric)
4: summarize `varlist', meanonly
5: generate xhelp = autocode(`varlist',50,r(min),r(max))
6: kdensity `varlist' if sex == 1, gen(fmen) at(xhelp) nodraw
7: kdensity `varlist' if sex == 2, gen(fwomen) at(xhelp) nodraw
8: graph twoway connected fmen fwomen xhelp ///
----- end: denscomp.ado -----
```

We will print only those parts of the ado-file that we talk about. However, we have printed the final version of `denscomp.ado` on page 465.

The `syntax` command has two functions: verifying input and parsing program arguments. `syntax` checks whether the user’s input makes sense and either produces an error message or parses the input so that the program can use it.

Let us clarify. Because we code `syntax varname`, Stata tries to interpret everything written behind `denscomp` as a variable name. If `denscomp` is typed without a variable name, Stata will issue an error message indicating that a variable name is required.

Moreover, Stata will check whether the user gave one or more variable names. Because we specified `syntax varname`, Stata will produce an error message if the user accidentally specified more than one variable name. If we had decided to allow the user to specify more than one variable name, we would have coded `syntax varlist`, with the effect that the user could have specified a list of variable names. Finally, because we specified `syntax varname(numeric)`, Stata will check whether the variable input by the user is a numeric variable or a string variable. If it is a string variable, Stata will issue an error message and exit the program immediately.

The second function of `syntax` is to parse program arguments given by the user. The `syntax` command in our example causes Stata to save the specified variable names in the local macro `varlist` when the program is called. With this in mind, we can use '`varlist`' in our program instead of the hard-coded variable name `linc`—see lines 4–7 of the program code.

We now call this program by typing `denscomp linc`. Go ahead and try it. You will first have to save `denscomp.ado` and then delete the old version of the program from memory with `discard` and delete the variables `xhelp`, `fmen`, and `fwomen`. Afterward, your program should run free of errors.

```
. discard
. drop xhelp fmen fwomen
. denscomp linc
```

12.3.4 Parsing options

These changes have made the program more generalized. However, the distributions of men and women are still being compared with each other. We should have the option of using the graph to compare other subgroups. We would like to have an option we could use to define the variable whose values will be compared. For example, this option could be called `by(varname)`. The question that remains is how to transfer these options to the program.

For this, we again turn to the `syntax` command. As indicated above, by using `syntax`, we are informing a program about the structure of the command. To enable the option `by(varname)`, we amend the `syntax` command as shown in line 3 below:

```
begin: denscomp.ado
3: syntax varname(numeric), by(varname)
3: syntax varname(numeric), by(varname)
4: summarize `varlist', meanonly
5: generate xhelp = autocode(`varlist',50,r(min),r(max))
6: kdensity `varlist' if `by' == 1, generate(fmen) at(xhelp) nodraw
7: kdensity `varlist' if `by' == 2, generate(fwomen) at(xhelp) nodraw
end: denscomp.ado
```

With these changes, the program `denscomp` now expects to be called with an option after the name of a numeric variable. We add the option `by` by placing a comma in the `syntax` command. Because we did not place the list of options (we refer to a list of

options even though currently only one option is allowed) in square brackets, we must specify the option. If we had used `syntax varname(numeric) [, by(varname)]`, the option would not have been required. Square brackets in `syntax` commands work just as described in the syntax diagram in the online help.

As it stands, we only allow one option: `by()`. Only one variable name is allowed inside the parentheses of `by()`. There is no restriction on numeric variables; hence, in this case, string variables are also allowed. As above, if we mistakenly type the name of a variable into `by()` that does not exist in the dataset, `syntax` will display an error message. Also `syntax` will place the variable name given in `by()` in the local macro called `by`. You can see how we use this local macro in lines 4 and 5 of our program snippet above.

You can apply the same logic for the `at()` option. In this option, we wish to specify the number of values for which the densities are estimated; that is, the number used in the second argument of the function `autocode()` in line 5 of the program. We type

```
----- begin: denscomp.ado -----
3: syntax varname(numeric), by(varname) [ at(integer 50) ]
4: summarize `varlist', meanonly
5: generate xhelp = autocode(`varlist',`at',r(min),r(max))
----- end: denscomp.ado -----
```

We used the keyword `integer` in the `at()` option because we want to allow the user to specify integer values. Because `at(integer 50)` is mentioned between square brackets, the option is not required. The number 50 is used whenever the user does not specify the option.

To try out these changes, save your file and type the following commands in Stata. You need to use the `discard` command to remove the current ado-file from memory. Only then will the changes become effective.

```
. discard
. drop xhelp fmen fwomen
. denscomp linc, by(sex) at(20)
```

You should now see the familiar graph, but this time with only 20 marker symbols. If not, carefully check your file for any errors. To determine which program line is faulty, you can use the command `set trace on`, which was described on page 446. A further help in locating errors is to display suspect program lines with `display`. To do so, you must modify the program with lines 6 and 8:

```
----- begin: denscomp.ado -----
5: generate xhelp = autocode(`varlist',`at',r(min),r(max))
6: display "generate xhelp = autocode(`varlist',`at',r(min),r(max))"
7: kdensity `varlist' if `by' == 1, generate(fmen) at(xhelp) nodraw
8: display "kdensity `varlist' if `by' == 1, generate(fmen) at(xhelp) nodraw"
----- end: denscomp.ado -----
```

With these lines, the program is called, and the output of `display` shows you the command line after macro expansion. This often helps to find typos. Once the program is error free, delete the `display` commands. During your search for errors, do not forget to type the commands `discard` and `drop xhelp fmen fwomen` after every change made to the ado-file.

12.3.5 Parsing if and in qualifiers

Before we refine our ado-file further, we would like to show you how to include `if` and `in` qualifiers in the program. Once again, our starting point is `syntax`:

```
————— begin: denscomp.ado —————
*-
3: syntax varname(numeric) [if] [in], by(varname) [ at(integer 50) ]
4: marksample touse
5: summarize `varlist' if `touse', meanonly
6: generate xhelp = autocode(`varlist',`at',r(min),r(max))
7: kdensity `varlist' if `by' == 1 & `touse', gen(fmen) at(xhelp) nodraw
8: kdensity `varlist' if `by' == 2 & `touse', gen(fwomen) at(xhelp) nodraw
————— end: denscomp.ado —————
```

Entering `[if]` and `[in]` lets you use `if` and `in` qualifiers. Because both of them are in square brackets, neither element is required. Again `syntax` saves the qualifiers that have been entered by the user in local macros—here, in `if` and `in`, respectively. In doing so, `syntax` saves the whole expression, including the keywords `if` and `in`, in the macro.

So far, we have enabled `if` and `in` qualifiers. Stata will not print an error message when the user invokes `denscomp` with an `if` qualifier. Now we must make our program exclude observations that do not satisfy those conditions. One solution is to use the `keep` command to delete all the observations for which the preconditions are not applicable and only execute the following commands for the remaining observations. A better solution would be to use `marksample` (see line 4). `marksample` generates a temporary variable; in our example, the name of this temporary variable is `touse` (short for: to use).³ In this variable, all cases to be used are assigned the value 1, and all other cases are assigned the value 0. `marksample` receives the information about which cases to use from the `if` and `in` qualifiers.

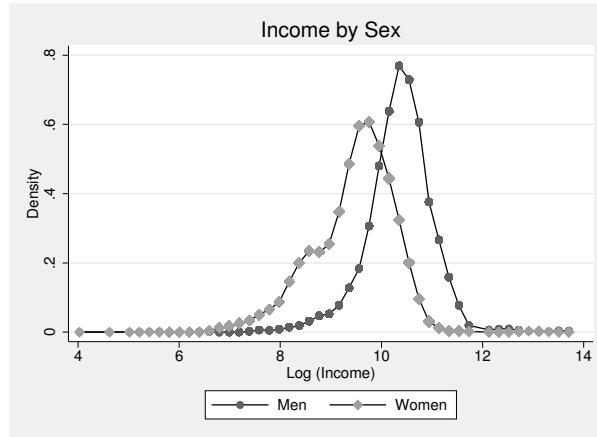
If the temporary variable `touse` has been defined by `marksample`, this variable can be used to restrict the statistical commands in the program to the observations marked with 1. You see examples of how to achieve this in lines 5, 7, and 8 of the program snippet above. Because `touse` is a temporary variable, it is placed between single quotation marks just like a local macro. In fact, `touse` is a local macro containing the name of the temporary variable. You do not need to specify `if 'touse'==1`, because `touse` contains only 0 and 1, which is Stata's way to express false and true (see section 3.1.6).

3. We discuss temporary variables in section 12.3.9 in more detail. Like local macros, they are cursory and available only in a given environment.

12.3.6 Generating an unknown number of variables

Before we continue, let us contemplate our situation. Save your current ado-file and type the following commands:

```
. discard
. drop xhelp fmen fwomen
. denscomp linc, by(emp)
```



This time, we have used employment status as the variable in the option `by()`. In principle, this should work.⁴ Nevertheless, something is not quite right here. The graph shows only two density curves, even though we were expecting one for every type of employment status, which would mean a total of four curves. According to the labeling, the graph still shows the income of men and women instead of the income of individuals with varying employment status.

Apparently, there is still some work to be done. We should start with the first problem: why are only two density curves being displayed instead of four? The cause lies in lines 7 and 8 of the program:

```
----- begin: denscomp.ado -----
%k
7: kdensity `varlist' if `by' == 1 & `touse', gen(fmen) at(xhelp) nodraw
8: kdensity `varlist' if `by' == 2 & `touse', gen(fwomen) at(xhelp) nodraw
----- end: denscomp.ado -----
```

These lines contain the calculations for the density of the values 1 and 2 for the variable given in `by()`. If we type the command with the option `by(emp)`, the local macro `by` is replaced by `emp`, and the densities for the first two values of employment status are displayed. There are no commands for any further categories.

4. If your program still will not work despite a thorough check for errors, then you can fall back on our file `denscomp.txt`. This contains the version of our program with which we generated the graph. You can use it by simply replacing your ado-file with the program definition in `denscomp.txt`.

The problem, however, is that the number of categories can vary with the variable specified in `by()`. This in turn means that we do not know how often we want the program to execute the `kdensity` command.

If we knew the number of categories, we could create a loop in our program that would execute `kdensity` for every variable in `by()` and stop when there are no remaining categories. Therefore, we must somehow find out how many categories the variable in `by()` has. The easiest way to get this information is to use the command `levelsof`.

In general, `levelsof` is used to look into a specific variable and store the existing distinct values of this variable in a local macro. The variable `emp`, for example, contains values of 1–5, although the value 3 is not used. The following command will store these values in the local macro `K`:

```
. levelsof emp, local(K)
. display "K"
```

Now let us see how you can use this in our program. You find the `levelsof` command in line 7 of the program excerpt below. We put `quietly` in front of the command. `quietly` can be used in front of any Stata command to suppress all of its output.

```
begin: denscomp.ado
8:
7: quietly levelsof `by' if `touse', local(K)
8: foreach k of local K {
9:     kdensity `varlist' if `by'==`k' & `touse', gen(fx`k') at(xhelp) nodraw
end: denscomp.ado
```

As you can see in line 7, we used `levelsof` to store the distinct values of the `by()` variable in the local macro `K`. Then we set up a `foreach` loop over all elements of the local macro `K`. Within the loop, `kdensity` is executed for each employment category. We changed the option `gen()` of the `kdensity` command in line 7. Instead of generating hard-coded variable names `fmen` and `fwomen`, we generate variables with names `fx1`, `fx2`, etc., according to the element name of the loop. That is, if we specify `denscomp` with the option `by(emp)`, we generate variables `fx1`, `fx2`, `fx4`, and `fx5` because there are no observations for which `emp` equals 3.

Next, because the number of `fx` variables is no longer fixed, we have to adapt the `graph` command accordingly. Thus we create a list with the names of the generated `fx` variables within the loop and then save it in a local macro called `yvars` (line 10):

```
begin: denscomp.ado
8:
8: foreach k of local K {
9:     kdensity `varlist' if `by'==`k' & `touse', gen(fx`k') at(xhelp) nodraw
10:    local yvars `yvars' `fx`k'
11: }
end: denscomp.ado
```

We can now include the local macro `yvars` in the `graph` command instead of `fmen` and `fwomen`:

```
begin: denscomp.ado
11:
12: graph twoway connected `yvars' xhelp, ///
13:      title(Income by Sex) ytitle(Density) xtitle(Log (Income)) ///
14:      legend(order(1 "Men" 2 "Women")) sort
end: denscomp.ado
```

Now it is time for another search for any errors:

```
. discard
. drop xhelp fmen fwomen
. denscomp linc, by(emp)
```

It looks a lot better. Nevertheless, it is far from perfect. In particular, the labeling of the graph remains a problem. We still have a bit of work ahead of us.

12.3.7 Default values

The first question is how to label the graph. We suggest that you make the default title “variable by by-variable”. “Density” would be suitable as the title of the *y* axis, and the title of the *x* axis should contain the variable name. The meaning of the data area would be explained in the legend by the categories of the by-variable.

Let us start with the title of the graph. A simple way of achieving the desired result is to change the `graph` command (line 13):

```
begin: denscomp.ado
11:
12: graph twoway connected `yvars' xhelp, ///
13:      title(`varlist' by `by') ytitle(Density) xtitle(Log (Income)) ///
14:      legend(order(1 "Men" 2 "Women")) sort
end: denscomp.ado
```

This, however, has the disadvantage of not being changeable by the user. To be able to change the title when calling up the program, we first have to include the corresponding option in the `syntax` statement at the beginning of the program:

```
begin: denscomp.ado
11:
3: syntax varname(numeric) [if] [in], by(varname) ///
4:   [ at(integer 50) TTitle(string asis) ]
end: denscomp.ado
```

With line 4, we allow the user to specify an option called `Title()`, where an arbitrary string can be specified, which is parsed inside the program `asis`; this means that quotation marks are part of the macro contents. The user now can enter a title when calling up the program. Because we do not want to be obliged to enter a title, we have written the option in square brackets. The two capital letters indicate the minimum abbreviation, so the user can specify `ti()` instead of `Title()`.

As was the case before, the **syntax** command saves everything within the parentheses in the local macro **title**. We can use this local macro in the title option of the **graph** command further down:

```
begin: denscomp.ado
13: local title = cond(`title' == "", `varlist' by `by'', `title')
14: graph twoway connected `yvars' xhelp, ///
15:      title(`title') ytitle(Density) xtitle(Log (Income)) ///
16:      legend(order(1 "Men" 2 "Women")) sort
end: denscomp.ado
```

Now we no longer have a default title, so we will add one. Add line 13 to the program:

```
begin: denscomp.ado
13: local title = cond(`title' == "", `varlist' by `by'', `title')
14: graph twoway connected `yvars' xhelp, ///
15:      title(`title') ytitle(Density) xtitle(Log (Income)) ///
16:      legend(order(1 "Men" 2 "Women")) sort
end: denscomp.ado
```

Two things require our comments in line 13: the function **cond(*x,a,b*)** and the compound quotes. We start with the former. The function **cond()** returns the second argument, *a*, if the expression in the first argument, *x*, is true; otherwise it returns the third argument (also see section 5.1.3). Consider the following simplified version of the **cond()** function in line 13: **cond("`title'" == "", ``varlist' by `by'', ``title'")**. Now think of a user who specified no title option. The first argument of the condition would become "" == "". This condition is true, so the second argument is returned: **local title = ``varlist' by `by''**. Hence, we will have a default title. If the user specifies **title(My title)**, the first argument becomes "My title" == "". This condition is false. Here Stata returns the third argument (**local title = "My title"**), which sets the title to the user-specified string.

Now for the compound quotes. Consider a user who includes **title("My" "title")** when invoking **denscomp**. This user wants the word "My" as the first line of the title, and "title" as the second. Again the first argument in the **cond()** function will be false. However, the simplified version of line 11 will become **local title = ""My" "title""**. The problem here is that Stata cannot distinguish the opening quotation in front of "My" from a closing quotation mark. In fact, Stata cannot distinguish opening and closing quotation marks at all. To differentiate between the beginning of a quotation and the end of a quotation, we use compound quotes. To start a quotation, we use '' (a single opening quotation, followed by a double quotation); to end a quotation, we use '' (a double quotation, followed by a single closing quotation). We recommend that you always use compound quotes inside programs. They never cause problems, but are often necessary.

The same procedure may be used for the other characteristics of the graph, for example, the title of the *y* and *x* axes:

```
----- begin: denscomp.ado -----
1: program denscomp
2: version 12
3: syntax varname(numeric) [if] [in], by(varname) ///
4:      [ at(integer 50) TTitle(string asis)      ///
5:          YTitle(string asis) XTitle(string asis) * ]
6: marksample touse
7: summarize `varlist' if `touse', meanonly
8: generate xhelp = autocode(`varlist',`at',r(min),r(max))
9: quietly levels of `by' if `touse', local(K)
10: foreach k of local K {
11:      kdensity `varlist' if `by'==`k' & `touse', gen(fx`k') at(xhelp) nodraw
12:      local yvars `yvars' fx`k'
13: }
14: local title = cond(`"`title'"' == `""', `"`varlist'"' by `by'"', `"`title'"')
15: local ytitle = cond(`"`ytitle'"' == `""', `"`Density'"', `"`ytitle'"')
16: local xtitle = cond(`"`xtitle'"' == `""', `"`varname'"', `"`xtitle'"')
17: graph twoway connected `yvars' xhelp, ///
18:      title(`title') ytitle(`ytitle') xtitle(`xtitle') sort `options'
19: end
----- end: denscomp.ado -----
```

In our `graph` command, we have not specified any other options, such as colors or marker symbols. Therefore, apart from the explicitly defined defaults, the graph resembles a standard Stata graph. At the same time, we want to be able to define all the graph's characteristics whenever we call the program. To do so, we have also specified the wildcard element in the `syntax` command (line 5). The star (*) means that the user can enter further options in addition to those explicitly allowed. These options are saved by `syntax` in the local macro `options` and can be included in the `graph` command as shown in line 18.

You should now try out these changes; type

```
. drop xhelp fx*
. discard
. denscomp linc, by(emp) ti(My Graph) xtitle(My X-Title) ytitle(My Y-Title)
```

12.3.8 Extended macro functions

Now all that is missing is an informative legend. This is a little tricky. By default, the variable labels of the data being plotted are used as the text in the legend. We can change this default setting by labeling the individual variables generated by `kdensity` with the appropriate caption. Ideally, the legend should inform us of the category of the variable in `by()`. We should therefore use the value label of the variable in `by()` as the text for the legend.

We can achieve this by using the extended macro function `label`. The general syntax for extended macro functions is

`local macroname: extended macro function`

Using a colon instead of an equal sign informs Stata that an extended macro function is to follow.

Here we add lines 10, 14, and 15 to the program, which concerns the `foreach` loop calculating the various density estimations. Also we slightly changed the command for displaying the graph (line 22).

```
————— begin: denscomp.ado —————
%<
10: local i 1
11: foreach k of local K {
12: kdensity `varlist' if `by'==`k' & `touse', gen(fx`k') at(xhelp) nodraw
13: local yvars `yvars' fx`k'
14: local label: label (`by') `k'
15: local order `order' `i++' `"\`label`"'"
16: }
%>
20: graph twoway connected `yvars' xhelp, title(`title') ytitle(`ytitle') xtitle(`xtitle') ///
21: legend(order(`order')) sort `options'
————— end: denscomp.ado —————
```

To customize the legend on a graph, we use the `legend()` option. In our sketch, we specified `legend(order(1 "Men" 2 "Women"))`, that is, a number and a label for each category in the `by`-variable. To generalize this for an arbitrary number of levels, we let the loop create the string inside the `order()` option.

Before the loop begins, the macros `label` and `order` are empty, and the macro `i` is 1. During the first iteration of the loop, the macro `label` is filled with the value label corresponding to the first level of the `by()` variable. The `order` macro is then set to 1 "`text`", where `text` is the text contained in `label`. Thereby, the macro `i` is incremented so that next time it will be equal to 2. In later iterations of the loop, the `order` macro is modified by adding elements of the form (`x "text"`), where `x` is the level of the `by()` variable and `text` is the value label corresponding to that level.

The local macro `order` can then easily be used for the definition of the legend in the `graph` command (line 22).

With this change, the first raw version of the ado-file would be ready. To check the situation so far, you should try the program again:

```
. discard
. drop xhelp fx*
. denscomp linc, by(sex)
. drop xhelp fx*
. denscomp linc, by(edu)
```

Before you work through the next section, you should make sure your program will run without errors, because locating errors will become more difficult in the next section. Try as many combinations of `graph` options, `if` and `in` qualifiers, and variables as you can.

12.3.9 Avoiding changes in the dataset

A program that generates variables is usually inconvenient. Users really do not enjoy deleting the variables generated by the previous command every time they use it. Therefore, an obvious solution would be to delete the newly generated variables at the end of every program. This is not much better either, because it assumes that the program will actually run until it finishes. If the user aborts the program or if it crashes and displays an error message, the variables generated until then will remain in the dataset.

The solution is to use temporary variables. You already came across a temporary variable when you were using `marksample`. Temporary variables are automatically removed when a program exits and are defined in two steps:

1. Declaring the variables' names
2. Generating the temporary variables

You declare temporary variables using the `tempvar` command, and you generate them using `generate`. However, the variable names are enclosed in single quotation marks, as is the case with local macros.

If, for example, you wanted to include the temporary variables `x1` and `x2` in a program, you could code

```
. tempvar x1 x2  
. generate `x1' = 1  
. generate `x2' = `x1'/_n
```

In our program snippet below, we declared the temporary variables for `xhelp` in front of the command that generates the variable (line 8). Because we do not know from the beginning how many variables holding the density estimations are created, these variables are declared as temporary variables inside the `foreach` loop (line 15). Once the temporary variables are declared, you should include all names of variables created by the program in ‘ and ’. Note the two single quotation marks around ‘`fx`k'`’ (lines 16 and 17). The first ‘ indicates the end of the local macro `k`, whereas the second ‘ indicates the end of the temporary variable. You also need to replace every instance of `fx`k'` in the program with ‘`fx`k'`’.

```

----- begin: denscomp.ado -----
%<
8: tempvar xhelp
9: summarize `varlist' if `touse', meanonly
10: generate `xhelp' = autocode(`varlist',`at',r(min),r(max))
%<
14: foreach k of local K {
15:   tempvar fx`k'
16:   kdensity `varlist' if `by'==`k' & `touse', gen(`fx`k') at(`xhelp') nodraw
17:   local yvars `yvars' `fx`k'
18:   local label: label (`by') `k'
19:   local order `order' `i++' `label' "
20: }
%<
26: graph twoway connected `yvars' `xhelp', ///
27:     title(`title') ytitle(`ytitle') xtitle(`xtitle') ///
28:     legend(order(`order')) sort `options'
----- end: denscomp.ado -----

```

After this step, a first beta version of the ado-file is ready, and figure 12.1 shows the entire code. We have inserted some blank lines to make the code a bit more readable. You may want to try the command with the following settings:

```

. discard
. denscomp linc, by(emp) ms(o..) mcolor(red green blue pink)
> lcolor(red green blue pink) ti("")

```

```

----- begin: denscomp.ado -----
1: program denscomp
2: version 12
3: syntax varname(numeric) [if] [in], by(varname) ///
4:           [ at(integer 50) TItle(string asis)      ///
5:             YTitle(string asis) XTitle(string asis) * ]
6: marksample touse
7:
8: tempvar xhelp
9: summarize `varlist' if `touse', meanonly
10: generate `xhelp' = autocode(`varlist',`at',r(min),r(max))
11:
12: quietly levelsof `by' if `touse', local(K)
13: local i 1
14: foreach k of local K {
15:   tempvar fx`k'
16:   kdensity `varlist' if `by'==`k' & `touse', gen(`fx`k') at(`xhelp') nodraw
17:   local yvars `yvars' `fx`k'
18:   local label: label (`by') `k'
19:   local order `order' `i++' `label'
20: }
21:
22: local title = cond(`title' == `",', `varlist' by `by', `title')
23: local ytitle = cond(`ytitle' == `",', `Density', `ytitle')
24: local xtitle = cond(`xtitle' == `",', `varname', `xtitle')
25:
26: graph twoway connected `yvars' `xhelp', ///
27:       title(`title') ytitle(`ytitle') xtitle(`xtitle') ///
28:       legend(order(`order')) sort `options'
29: end
----- end: denscomp.ado -----

```

Figure 12.1. Beta version of `denscomp.ado`

12.3.10 Help files

Once you are satisfied with your ado-file, you should write a help file. Help files are straightforward ASCII files displayed on screen by Stata when one types

```
. help command
```

For Stata to find the help file for a command, the help file must have the same name as the ado-file, albeit with the file extension `.sthlp`. Here the help file would be called `denscomp.sthlp`.

For your help file to have the same look and feel as common Stata help files, you can use the Stata Markup and Control Language (SMCL). In Stata, all output is in SMCL format, be it normal command output or output from invoking a help file. In SMCL, directives are used to affect how output appears. Say, for example, your `denscomp.sthlp` file includes the following text:

```

begin: denscomp.sthlp
1: {smcl}
2: {* January 17, 2012 @ 17:36:12 UK}{...}
4: {cmd:help denscomp}
5: {hline}
6:
7: {title:Title}
7: {p 4 8 2} {hi:denscomp} {hline 2} Comparing density estimates in one graph
8:
9: {p 8 17 2}
10: {cmd:denscomp}
11: {varname}
12: {ifin}
13: {cmd:,} {cmd:by()}{it:varname}{cmd:}
14: [{cmd:at()}{it:#}{cmd:} {it:twoway_options}]
15:
16: {title:Description}
17:
18: {p 4 4 2}
19: {cmd:denscomp} uses {helpb kdensity} to estimate densities of {it:varname}
20: for categories of {cmd:by()}.
21:
22: {title:Options}
23:
24: {p 4 8 2}{cmd:by()}{it:varname}{cmd:} is required. It is used to separate
25: density estimation for each category of {it:varname}.
26:
27: {p 4 8 2}{cmd:at()}{it:#}{cmd:} is an integer that controls at how many
28: levels the densities are estimated.
29:
30: {p 4 8 2}{it:twoway_options} are any of the options documented in
31: {manhelpi twoway_options G-3}, excluding {cmd:by()}. These include options
32: for titling the graph (see {manhelpi title_options G-3})
33: and for saving the graph to disk (see {manhelpi saving_option G-3}).
34:
35: {title:Example}
36:
37: {cmd:. denscomp linc, by(edu)}
38:
39: {title:Also see}
40:
41: Manual: {hi:[R] kdensity}
42:
43: Help: {manhelp kdensity R}

```

When you type `help denscomp`, line 19, “`denscomp` uses `kdensity` to estimate densities of `varname` for categories of `by()`”, will be displayed, with “`denscomp`” displayed in bold face and “`kdensity`” displayed in blue and clickable. By clicking on `kdensity`, you will open the help file for `kdensity` in the viewer window.

In [P] **smcl**, you will find further information on creating help files, as well as recommendations for organizing your help files.

12.4 Exercises

1. Create a local macro that holds the name of the directory in which you have stored the data package of this book.
2. In Stata, change to an arbitrary directory and load `data1.dta` into memory using your local macro.
3. Change to the directory in which you have stored our data package using your local macro.
4. Create a new dataset of all the GSOEP person files (`kksoep/*p`). Save this dataset for later use.
5. Rename the variables `anetto`, `bnetto`, ..., `snetto` into `netto1984`, `netto1985`, ..., `netto2002` with a loop.
6. Rename all variables for life satisfaction into `lsat1984`, `lsat1985`, ..., `lsat2002` with a loop.
7. Add to your last loop a command that keeps track of the original variable name using `note`. After reloading your dataset, rerun the loop for life satisfaction.
8. Save your loop as the program `mysoepren`. Run the program after having reloaded your dataset.
9. Store your program `mysoepren` in the file `mysoepren.ado` in the current working directory of your computer. Exit and relaunch Stata and rerun `mysoepren`.
10. Change `mysoepren.ado` such that you can parse an arbitrary list of variables to be renamed. Rerun the program for the life-satisfaction variables and the `netto*` variables.
11. Change `mysoepren.ado` such that you can use arbitrary numbers for the new variable names. Rerun the program for the life-satisfaction variables and the `netto*` variables using numbers 1–26 instead of 1984–2009.
12. Change `mysoepren.ado` such that you will get a confirmation if the number of variable names to be renamed and the number of new variable names match. Use the command to rename only the first five occurrences of the household number.

13 Around Stata

13.1 Resources and information

In addition to the manuals, several resources are available to help you learn Stata. The Stata webpage <http://www.stata.com> has the most complete and up-to-date versions of official resources, as well as links to a wealth of user-written resources. Of particular interest are the following:

- The *Stata Journal* (SJ):¹ A printed peer-reviewed quarterly journal containing articles about data analysis with Stata. The *Stata Journal* replaced the *Stata Technical Bulletin* (STB)². The *Stata Journal* also publishes Stata software for general use (see section 13.3.1).
- FAQs:³ A webpage with answers to frequently asked questions.
- The Statalist:⁴ An open Internet discussion forum where you can post questions about tricky Stata problems or general statistical techniques. Even simple questions will be posted and answered, so do not be shy. However, we recommend that you consult Stata's documentation and check out the Statalist archive⁵ before you post a question to Statalist. To participate, send an email to major-domo@hsphsun2.harvard.edu with "subscribe statalist" in the body of the message. The subject line remains empty. To unsubscribe, send an email with "unsubscribe statalist" to the same address. Participating in Statalist is free.
- Stata Bookstore:⁶ A collection of books about Stata, as well as an excellent selection of books regarding particular statistical techniques; many of these books use Stata for illustration. You can order the books online.
- Stata Press:⁷ The publisher of the Stata manuals, the *Stata Journal*, and many books on Stata. The Stata Press offers a webpage that also contains all datasets used in the manuals.
- Stata NetCourses:⁸ Internet courses offered and run by StataCorp. Once a week, the instructor will post lecture notes and assignments on a password-protected

1. <http://www.stata-journal.com>

2. <http://www.stata.com/support/stb/faq/>

3. <http://www.stata.com/support/faqs/>

4. <http://www.stata.com/statalist/>

5. <http://www.stata.com/statalist/archive/>

6. <http://www.stata.com/bookstore/>

7. <http://www.stata-press.com>

8. <http://www.stata.com/netcourse/>

website. The discussion between participants and instructors takes place on a message board at the NetCourse website. Instructors will respond to questions and comments usually on the same day as they are posted. NetCourses are available on beginner and advanced levels. There is a course fee for the NetCourses. Participation requires a valid Stata serial number.

- Additional links:⁹ Links to many other resources, including the impressive collection of Stata textbook and paper examples from UCLA Academic Technology Services (ATS).¹⁰

13.2 Taking care of Stata

In the past, a new Stata version has been released roughly every two years. However, this does not mean that there is no development between new releases. On the contrary, Stata updates are available on average every couple of months. These updates include improvements to existing commands, as well as more commands added since the last Stata version. Even if you are not interested in more commands, some of the changes may, for example, affect the overall speed of the program. We therefore recommend that you regularly update your copy of Stata. Updating Stata is free.

We distinguish between official and user-written Stata commands. Official Stata commands are distributed by StataCorp with the installation DVD and with the updates. Everyone who installed Stata will have the official Stata commands. User-written Stata programs will be only on those machines where the user has installed them. Here we will show you how to update the official commands. The next section will discuss how to install and update user-written commands.

Updating the official Stata installation takes place with the command `update`. To begin, please type

```
. update query
(contacting http://www.stata.com)

Update status
Last check for updates: 24 Apr 2012
New update available: 24 Apr 2012 (what's new)
Current update level: 04 Apr 2012 (what's new)

Possible actions
Install available updates (or type -update all-)
```

The command `update query` compares information on the installed files with the latest available version of these files. In our example, the most recent Stata is from 24 April 2012 while we have a Stata version from 04 April 2012. We should therefore update our installation. To do so, type

```
. update all
```

9. <http://www.stata.com/links/resources.html>
 10. <http://www.ats.ucla.edu/stat/stata/>

Stata will connect to the Internet, search for the required files, and save them at the right place on your machine. After successfully updating, you can see a list of changes by typing `help whatsnew`.

The procedure described above assumes that you have a direct connection to the Internet. If this is not the case for your computer, you can download the official update as a compressed zip archive.¹¹ Once you are at your machine, you unzip the archive with the appropriate program. However, you need to make sure that you keep the archive directory structure. Now you can proceed as described above, using the option `from()` of the `update` command. Just type the name of the directory that you used to unpack the zip archive name within the parentheses (for example, `update all, from(c:\temp)`).

13.3 Additional procedures

Any Stata user can write ados (see chapter 12), so many statisticians and others all over the world have written ados for statistical procedures that do not exist in Stata. Once an ado is written, any user can install it and use it just like any other command. Here we will show you how to install such user-written programs.

13.3.1 Stata Journal ado-files

We call ados published with the *Stata Journal* or the *Stata Technical Bulletin* SJ-ados and STB-ados, respectively. Stata itself knows about SJ-ados or STB-ados; the command `search` will provide information about their existence. SJ- and STB-ados can be downloaded for free from the Internet.

Say that you want to present the results of all the many regressions you run in a table similar to table 9.2 on page 314: you want to display the *t* statistic or the standard error in parentheses below each coefficient, you want to add stars to indicate their significance level, and you want to print summary measures like R^2 below the coefficients. To do so, you could of course use `estimates table` as shown in section 9.5.1, but this would not create the final table. If you are lucky, you might find a Stata command that does the work for you. One way to find such a command is to use the Stata command `search`. Try, for example,

```
. search regression output
```

11. <http://www.stata.com/support/updates/>

which will give you several hits. Among them are

```
(output omitted)
SJ-7-2 st0085_1 . . . . . Making regression tables simplified
(help estadd, estout, _eststo, eststo, esttab if installed) . B. Jann
Q2/07 SJ 7(2):227--244
introduces the eststo and esttab commands (stemming from
estout) that simplify making regression tables from stored
estimates
(output omitted)
```

The entry you see refers to issue 2 in volume 7 of the *Stata Journal*. This means that there is an article with the title “Making regression tables simplified” written by Ben Jann (2007). There is an entry number attached to the article, `st0085_1`, which we will discuss a little later. More important is the description “(`help estadd, estout ... if installed`)”. This line indicates that you are dealing with a program (or several programs) because only programs can be installed. From the short description of the program, we learn that in this version, some small bugs were fixed. Our example introduces additions (`eststo` and `esttab` to an already existing program, `estout`). The program itself simplifies the creation of regression tables from stored estimates. Sounds good! That is what we were looking for. You will learn more if you read the respective SJ article or if you install the program and read the online help.

The easiest way to install SJ- or STB-ados is to click on the entry number. This will open a window that guides you through the installation. However, if you want to install a program within your do-files, you can use the `net` commands. The installation happens in two steps:

1. The command `net sj volume-issue` will connect you to a web address of the *Stata Journal* holding all ados published in the specified issue. Likewise, the command `net stb issue` will do the same for STB-ados.
2. The command `net install pkgname` will install the program `pkgname`.

For example, to install the program `estout`, you would type

```
. net sj 7-2
http://www.stata-journal.com/software/sj7-2/
(no title)

DIRECTORIES you could -net cd- to:
..
          Other Stata Journals

PACKAGES you could -net describe-:
  gr0001_3      Update: Generalized Lorenz curves and related graphs
  st0085_1      Update: Making regression tables simplified
  st0097_1      Update: Generalized ordered logit/partial
                 proportional odds models for ordinal dependent variables
  st0122        Fit population-averaged panel-data models using
                 quasileast squares
  st0123        Maximum likelihood and two-step estimation of an
                 ordered-probit selection model
  st0124        Two postestimation commands for assessing confounding
                 effects in epidemiological studies
  st0125        Estimation of Nonstationary Heterogeneous Panels
  st0126        QIC criterion for model selection in GEE analyses
```

This will show a list of all available programs in volume 7, issue 2, of the SJ. Among others, you will find the package “Making regression tables simplified” with the entry “`st0085_1`”. This is the same entry number you saw with `search`, and it is the package name that you use to install the program:

```
. net install st0085_1
```

If you want to see a description of the package before you install it, you can type

```
. net describe st0085_1
```

13.3.2 SSC ado-files

Another source of user-written Stata commands is the Statistical Software Components (SSC) archive maintained by Boston College.¹² Many of the commands stored there are discussed on Statalist. In Stata, the `ssc install` command can be used to install them; for more information, type `help ssc`.

For example, in the SSC archive is a command called `soepuse` written by one of the authors of this book (U. Kohler). The command makes data retrievals from the GSOEP data base easy. Type

```
. ssc install soepuse
```

12. <http://ideas.repec.org/s/boc/bocode.html>

to install the command. Once the command has been installed, it is possible to update the command by typing

```
. adoupdate soepuse, update
```

Type

```
. adoupdate all, update
```

if you want to update all user-written commands.

13.3.3 Other ado-files

In addition to SJ-, STB-, and SSC-ados, there are many other user-written commands. You can get an idea of the variety of ados by following one of the links displayed when you type

```
. net from http://www.stata.com
```

Most of these ados can be installed with `net install`, so the only challenge is to actually find the right ado-file for your problem. However, before we discuss how to find the right ado-file, you should know how to install ados in case Stata's `net` facilities do not work. This can happen if

- the computer you work on is not connected to the Internet.
- you programmed your own ado-file.
- the author of the ado-file did not prepare the file appropriately. To detect an ado-package with `net from` and to install it with `net install`, there must be a table of contents file (`stata.toc`) and a package description file stored under the URL. If an author does not provide these files, you cannot use the `net` commands to install the ado-file.
- the `stata.toc` refers to packages on another computer. Then the particular file must be downloaded with `net get` and installed by hand, as explained below.

Usually, you can get the desired ado-packages outside Stata and install them by hand. Every ado-package should have at least two files: the ado-file itself and an accompanying help file. The ado-file has the extension `.ado`, and the help file has the extension `.sthlp` or `.hlp`. To install both of them, you copy the files in your personal ado-directory.

You can determine the location of your personal ado-directory by typing

```
. adopath
```

Official ados are saved in the first two directories (`UPDATES` and `BASE`). The third directory, called `SITE`, is meant for network administrators to install ados that should be accessible for all network users. The fourth directory, here indicated with “.”, refers

to your current working directory. Finally, you find the address of your personal ado-directory (**PERSONAL**), in which you can copy ados “by hand”. Ados you wrote yourself should be stored here. The most common place for the personal ado-directory under Windows is `c:\ado\personal`; however, this depends on the local installation. You must create `c:\ado\personal` by hand; this can be done using the Stata command `mkdir`.

Finally, the **PLUS** directory contains ados you have downloaded with Stata’s Internet facilities (for example, `net install` and `ssc install`). The `c:\ado\plus` directory is created by Stata when you first use `net install` or `ssc install`. **OLDPLACE** is the place where ados before Stata 7.0 were installed. The order of the path matches the order with which ados will be searched on your machine when you type a command (see section 12.2.4).

Now back to the interesting question: Where do you find the “right” ado-file? We have already discussed this above with respect to the SJ-ados. For SSC-ados, you can also browse the SSC archive with any Internet browser.¹³ Also you will find several links on the Stata website.¹⁴ The best way to find Stata ados, however, is by adding the option `all` to `search`. This searches in the online help, as well as the FAQs on the Stata website, the *Stata Journal*, and many other Stata-related Internet sources. You can try `search, all` with arbitrary keywords, known package names, or just author names. `search` displays results with the addresses of entries matching your search term. If you click on the address, you will be led through the installation process.

13.4 Exercises

1. Find out whether your Stata is up to date.
2. If necessary, update your Stata to the most recent version or explain to your local network administrator how he or she can update Stata.
3. Explore the changes of the most recent Stata update.
4. Install the following frequently used SJ-ados:
 - a. The value label utilities `labelup` and `labelrename` from SJ 5-2.
 - b. The package `dm75` of STB-53 for safe and easy matched merging.
 - c. The lean graphic schemes from SJ 4-3.
 - d. The enhanced graphing model diagnostic tools described in SJ 4-4.
5. Install the following ado-packages from the SSC archive:
 - a. `egenmore`
 - b. `fitstat`
 - c. `tostring`

13. <http://ideas.repec.org/search.html>

14. <http://www.stata.com/links/resources.html>

- d. `vclose`
 - e. `fre`
6. Find and install user-written packages for the following tasks:
 - a. Center metric variables
 - b. Draw thematic maps
 - c. Sequence analysis
 7. Starting from <http://www.stata.com/users/>, explore the ado-directories of the following users or institutions:
 - a. Nicholas J. Cox
 - b. Bill Rising
 - c. University of California in Los Angeles (UCLA)
 - d. Jeroen Weesie
 - e. Scott Long and Jeremy Freese
 - f. Ben Jann

References

- Abadie, A., D. Drukker, J. Leber Herr, and G. W. Imbens. 2004. Implementing matching estimators for average treatment effects in Stata. *Stata Journal* 4: 290–311.
- Agresti, A. 1984. *Analysis of Ordinal Categorical Data*. New York: Wiley.
- Aiken, L. S., and S. G. West. 1991. *Multiple Regression: Testing and Interpreting Interactions*. Newbury Park, CA: Sage.
- Aldrich, J. H., and F. D. Nelson. 1984. *Linear Probability, Logit, and Probit Models*. Newbury Park, CA: Sage.
- Allison, P. D. 1999. Comparing logit and probit coefficients across groups. *Sociological Methods and Research* 28: 186–208.
- . 2009. *Fixed Effects Regression Models*. Vol. 160 of *Quantitative Applications in the Social Sciences*. Thousand Oaks, CA: Sage.
- Anderson, J. A. 1984. Regression and ordered categorical variables (with discussion). *Journal of the Royal Statistical Society, Series B* 46: 1–30.
- Andreas, H.-J., J. A. Hagenaars, and S. Kühnel. 1997. *Analyse von Tabellen und Kategoriale Daten: Log-Lineare Modelle, Latente Klassenanalyse, Logistische Regression und GSK-Ansatz*. Berlin: Springer.
- Anscombe, F. J. 1973. Graphs in statistical analysis. *American Statistician* 27: 17–21.
- Baltagi, B. H. 2008. *Econometric Analysis of Panel Data*. 4th ed. New York: Wiley.
- Baum, C. F. 2009. *An Introduction to Stata Programming*. College Station, TX: Stata Press.
- . 2010. *An Introduction to Modern Econometrics Using Stata*. Rev. ed. College Station, TX: Stata Press.
- Baum, C. F., M. E. Schaffer, and S. Stillman. 2007. Enhanced routines for instrumental variables/generalized method of moments estimation and testing. *Stata Journal* 7: 465–506.
- Belsley, D. A., E. Kuh, and R. E. Welsch. 1980. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. New York: Wiley.

- Berk, K. N., and D. E. Both. 1995. Seeing a curve in multiple regression. *Technometrics* 37: 385–398.
- Berk, R. A. 2004. *Regression Analysis: A Constructive Critique*. Thousand Oaks, CA: Sage.
- Bollen, K. A., and R. W. Jackman. 1990. Regression Diagnostics: An Expository Treatment of Outliers and Influential Cases. In *Modern Methods of Data Analysis*, ed. J. Fox and S. Long, 257–291. Newbury Park: Sage.
- Breen, R., K. B. Karlson, and A. Holm. 2011. A reinterpretation of coefficients from logit, probit, and other non-linear probability models: Consequences for comparative sociological research. http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1857431.
- Chambers, J. M., W. S. Cleveland, B. Kleiner, and P. A. Tukey. 1983. *Graphical Methods for Data Analysis*. New York: Chapman and Hall.
- Cleveland, W. S. 1979. Robust locally weighted regression and smoothing scatterplots. *Journal of the American Statistical Association* 74: 829–836.
- . 1984. Graphical methods for data presentation: Full scale breaks, dot charts, and multibased logging. *American Statistician* 38: 270–280.
- . 1993. *Visualizing Data*. Summit, NJ: Hobart.
- . 1994. *The Elements of Graphing Data*. Rev. ed. Summit, NJ: Hobart.
- Cook, R. D. 1977. Detection of influential observations in linear regression. *Technometrics* 19: 15–18.
- Cook, R. D., and S. Weisberg. 1994. *An Introduction to Regression Graphics*. New York: Wiley.
- . 1999. *Applied Regression Including Computing and Graphics*. New York: Wiley.
- Cox, N. J. 2002a. Speaking Stata: How to face lists with fortitude. *Stata Journal* 2: 202–222.
- . 2002b. Speaking Stata: How to move step by: step. *Stata Journal* 2: 86–102.
- . 2004. Speaking Stata: Graphing distributions. *Stata Journal* 4: 66–88.
- . 2011. Speaking Stata: Fun and fluency with functions. *Stata Journal* 11: 460–471.
- Dawson, R. J. M. 1995. The “unusual episode” data revisited. *Journal of Statistics Education* 3.
- Dewald, W. G., J. G. Thursby, and R. G. Anderson. 1986. Replication in empirical economics: The *Journal of Money, Credit, and Banking* project. *American Economic Review* 76: 587–603.

- Diekmann, A. 1998. Die Bedeutung der Sekundäranalyse zur Kontrolle von Forschungsergebnissen. In *Statistik im Dienste der Öffentlichkeit*, ed. W. Haug, K. Armingeon, P. Farago, and M. Zürcher. Bern: Bundesamt für Statistik.
- Diggle, P. J., P. Heagerty, K.-Y. Liang, and S. L. Zeger. 2002. *Analysis of Longitudinal Data*. 2nd ed. New York: Oxford University Press.
- Emerson, J. D., and D. C. Hoaglin. 1983. Stem-and-leaf displays. In *Understanding Robust and Exploratory Data Analysis*, ed. D. C. Hoaglin, F. Mosteller, and J. W. Tukey, 7–32. New York: Wiley.
- Erikson, R., and J. H. Goldthorpe. 1992. *The Constant Flux: a Study of Class Mobility in Industrial Societies*. Oxford: Clarendon Press.
- Fahrmeir, L., R. Künstler, I. Pigeot, and G. Tutz. 1997. *Statistik. Der Weg zur Datenanalyse*. Berlin: Springer.
- Fienberg, S. E. 1980. *The Analysis of Cross-Classified Categorical Data*. Cambridge, MA: MIT Press.
- Fisher, R. A. 1935. The logic of inductive inference. *Journal of the Royal Statistical Society* 98: 39–82.
- Fowler, F. J. 1984. *Survey Research Methods*. Beverly Hills, CA: Sage.
- Fox, J. 1991. *Regression Diagnostics: An Introduction*. Newbury Park, CA: Sage.
- . 1997. *Applied Regression Analysis, Linear Models, and Related Methods*. Thousand Oaks, CA: Sage.
- . 2000. *Nonparametric Simple Regression: Smoothing Scatterplots*. Thousand Oaks, CA: Sage.
- Freedman, D. A. 2004. The ecological fallacy. In *Encyclopedia of Social Science Research Methods*, ed. M. Lewis-Beck, A. Bryman, and T. F. Liao. Thousand Oaks, CA: Sage.
- Gould, W., J. Pitblado, and B. Poi. 2010. *Maximum Likelihood Estimation with Stata*. 4th ed. College Station, TX: Stata Press.
- Greene, W. H. 2012. *Econometric Analysis*. 7th ed. Upper Saddle River, NJ: Prentice Hall.
- Groves, R. M. 1989. *Survey Errors and Survey Costs*. New York: Wiley.
- Gujarati, D. N. 2003. *Basic Econometrics*. 4th ed. New York: McGraw-Hill.
- Hagle, T. M. 1996. *Basic Math for Social Scientists: Problems and Solutions*, vol. 109. Thousand Oaks, CA: Sage.
- Hair, J. F., R. E. Anderson, R. L. Tatham, and W. C. Black. 1995. *Multivariate Data-Analysis: With Readings*. 4th ed. Englewood Cliffs, NJ: Prentice Hall.

- Halaby, C. N. 2004. Panel models in sociological research: Theory into practice. *Annual Review of Sociology* 30: 507–544.
- Hamilton, L. C. 1992. *Regression with Graphics: A Second Course in Applied Statistics*. Pacific Grove, CA: Brooks/Cole.
- Hardin, J. W., and J. M. Hilbe. 2003. *Generalized Estimating Equations*. New York: Chapman & Hall/CRC.
- Holland, P. W. 1986. Statistics and causal inference. *Journal of the American Statistical Association* 81: 945–960.
- Hosmer, D. W., and S. Lemeshow. 2000. *Applied Logistic Regression*. 2nd ed. New York: Wiley.
- Howell, D. C. 1997. *Statistical Methods for Psychology*. Belmont, CA: Duxbury.
- Huff, D. 1954. *How to Lie with Statistics*. New York: W. W. Norton.
- Imai, K., G. King, and E. A. Stuart. 2008. Misunderstandings between experimentalists and observationalists about causal inference. *Journal of the Royal Statistical Society, Series A* 171: 481–502.
- Jann, B. 2007. Making regression tables simplified. *Stata Journal* 7: 227–244.
- Kennedy, P. 2008. *A Guide to Econometrics*. 6th ed. New York: Wiley.
- King, G., R. O. Keohane, and S. Verba. 1994. *Designing Social Inquiry: Scientific Inference in Qualitative Research*. Princeton, NJ: Princeton University Press.
- Kish, L. 1965. *Survey Sampling*. New York: Wiley.
- Kreuter, F., and R. Vaillant. 2007. A survey on survey statistics: What is done and can be done in Stata. *Stata Journal* 7: 1–21.
- Lääärä, E. 2009. Statistics: Reasoning on uncertainty, and the insignificance of testing null. *Annales Zoologici Fennici* 46: 138–157.
- Long, J. S. 1997. *Regression Models for Categorical and Limited Dependent Variables*. Thousand Oaks, CA: Sage.
- . 2009. *The Workflow of Data Analysis Using Stata*. College Station, TX: Stata Press.
- Long, J. S., and J. Freese. 2006. *Regression Models for Categorical Dependent Variables Using Stata*. 2nd ed. College Station, TX: Stata Press.
- Mallows, C. L. 1986. Augmented partial residuals. *Technometrics* 28: 313–319.
- Marron, J. S. 1988. Automatic smoothing parameter selection: A survey. *Empirical Economics* 13: 187–208.

- McFadden, D. 1973. Conditional logit analysis of qualitative choice behavior. In *Frontiers in Econometrics*, ed. P. Zarempka, 105–142. New York: Academic Press.
- Mitchell, M. N. 2012. *A Visual Guide to Stata Graphics*. 3rd ed. College Station, TX: Stata Press.
- Mosteller, F., and J. W. Tukey. 1977. *Data Analysis and Regression: A Second Course in Statistics*. Reading, MA: Addison-Wesley.
- Newson, R. 2003. Confidence intervals and p-values for delivery to the end user. *Stata Journal* 3: 245–269.
- Neyman, J., K. Iwaszkiewicz, and S. Kolodziejczyk. 1935. Statistical problems in agricultural experimentation. *Supplement to the Journal of the Royal Statistical Society* 2: 107–180.
- Nichols, A. 2007. Causal inference with observational data. *Stata Journal* 7: 507–541.
- Pearson, K. 1900. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine, Series 5* 50: 157–175.
- Popper, K. R. 1994. *Logik der Forschung*. 10. Auflage. Tübingen: J.C.B. Mohr.
- Raftery, A. E. 1995. Bayesian model selection in social research. In *Sociological Methodology Volume 25*, ed. P. V. Marsden, 111–163. Oxford: Blackwell.
- Rubin, D. B. 1974. Estimating causal effects of treatments in randomized and nonrandomized studies. *Journal of Educational Psychology* 66: 688–701.
- . 1979. Using multivariate matched sampling and regression adjustment to control bias in observational studies. *Journal of the American Statistical Association* 74: 318–328.
- . 1980. Bias reduction using Mahalanobis-metric matching. *Biometrics* 36: 293–298.
- . 1987. *Multiple Imputation for Nonresponse in Surveys*. New York: Wiley.
- Schafer, J. L. 1997. *Analysis of Incomplete Multivariate Data*. Boca Raton, FL: Chapman & Hall/CRC.
- Schnell, R. 1994. *Graphisch gestützte Datenanalyse*. München u. Wien: Oldenbourg.
- Selvin, H. C. 1970. A critique of tests of significance in survey research. In *The Significance Test Controversy*, ed. D. E. Morrison and R. E. Henkel, 94–108. Chicago: Transaction Publishers.
- Stuart, E. A. 2010. Matching methods for causal inference: A review and a look forward. *Statistical Science* 25: 1–21.

- Stuart, E. A., S. R. Cole, C. P. Bradshaw, and P. J. Leaf. 2011. The use of propensity scores to assess the generalizability of results from randomized trials. *Journal of the Royal Statistical Society, Series A* 174: 369–386.
- Tufte, E. R. 2001. *The Visual Display of Quantitative Information*. 2nd ed. Cheshire, CT: Graphics Press.
- Valliant, R., J. Dever, and F. Kreuter. 2012. *Practical Tools for Designing and Weighting Survey Samples*. New York: Springer.
- Veall, M. R., and K. F. Zimmermann. 1994. Evaluating pseudo- R^2 's for binary probit models. *Quality and Quantity* 28: 150–164.
- Wainer, H. 1984. How to display data badly. *American Statistician* 38: 137–147.
- Williams, R. 2009. Using heterogeneous choice models to compare logit and probit coefficients across groups. *Sociological Methods and Research* 37: 531–559.
- Winship, C., and S. L. Morgan. 1999. The estimation of causal effects from observational data. *Annual Review of Sociology* 25: 659–707.
- Wissenschaftliche Dienste des Deutschen Bundestages, ed. 1998. *Die Mitglieder des Deutschen Bundestages. 1.-13. Wahlperiode. Alphabetisches Gesamtverzeichnis*. Bonn: Deutscher Bundestag (Materialien Nr. 127).
- Wooldridge, J. M. 2010. *Econometric Analysis of Cross Section and Panel Data*. 2nd ed. Cambridge, MA: MIT Press.

Author index

A

- Abadie, A. 248
Agresti, A. 162–163
Aiken, L. S. 31, 272, 277, 305
Aldrich, J. H. 344, 438
Allison, P. D. 245, 383
Anderson, J. A. 391
Anderson, R. G. 25
Anderson, R. E. 266, 312, 313
Andreß, H.-J. 351
Anscombe, F. J. 279

B

- Baltagi, B. H. 328
Baum, C. F. 246, 328, 450
Belsley, D. A. 288
Berk, K. N. 284
Berk, R. A. 242, 249, 255, 277, 280
Black, W. C. 266, 312, 313
Bollen, K. A. 288
Both, D. E. 284
Bradshaw, C. P. 245
Breen, R. 383

C

- Chambers, J. M. 296
Cleveland, W. S. 119, 121, 132, 163,
168, 169, 184, 195, 196, 285,
286, 296, 297, 370
Cole, S. R. 245
Cook, R. D. 260, 280, 293, 312
Cox, N. J. 58, 82, 91, 189, 476

D

- Dawson, R. J. M. 242, 342, 382
Dever, J. 219
Dewald, W. G. 25

- Diekmann, A. 25
Diggle, P. J. 328
Drukker, D. 248

E

- Emerson, J. D. 190

F

- Fahrmeir, L. 191
Fienberg, S. E. 163
Fisher, R. A. 162
Fowler, F. J. 414
Fox, J. 255, 283, 290, 292, 296, 310,
312, 371, 373
Freedman, D. A. 325
Freese, J. 364, 476

G

- Gallup, J. L. 316
Gould, W. 354
Greene, W. H. 275
Groves, R. M. 249
Gujarati, D. N. 270

H

- Hagenaars, J. A. 351
Hagle, T. M. 260
Hair, J. F. 266, 312, 313
Haïskens-DeNew, J. 415
Halaby, C. N. 245
Hamilton, L. C. 255, 260, 287, 289
Hardin, J. W. 328
Heagerty, P. 328
Hilbe, J. M. 328
Hoaglin, D. C. 190
Holland, P. W. 244
Holm, A. 383

Hosmer, D. W. 342, 368, 371, 376
 Howell, D. C. 268
 Huff, D. 132

I

Imai, K. 242
 Imbens, G. W. 248
 Iwaskiewicz, K. 244

J

Jackman, R. W. 288
 Jann, B. 316, 472, 476

K

Karlson, K. B. 383
 Kennedy, P. 275
 Keohane, R. O. 242, 255
 King, G. 242, 255
 Kish, L. 68
 Kleiner, B. 296
 Kohler, U. 415, 473
 Kolodziejczyk, S. 244
 Kreuter, F. 219
 Kuh, E. 288
 Kühnel, S. 351
 Künstler, R. 191

L

Leaf, P. J. 245
 Leber Herr, J. 248
 Lemeshow, S. 342, 368, 371, 376
 Liang, K.-Y. 328
 Long, J. S. 35, 342, 364, 366, 386,
 387, 476
 Läärä, E. 233

M

Mallows, C. L. 285
 Marron, J. S. 194
 McFadden, D. 364, 438
 Mitchell, M. N. 115
 Morgan, S. L. 242
 Mosteller, F. 309

N

Nelson, F. D. 344, 438

Newson, R. 316, 319
 Neyman, J. 244
 Nichols, A. 246, 248

P

Pearson, K. 162
 Pigeot, I. 191
 Pitblado, J. 354
 Poi, B. 354
 Popper, K. R. 25

R

Raftery, A. E. 379
 Rising, B. 476
 Rubin, D. B. 228, 244

S

Schafer, J. L. 228
 Schaffer, M. E. 246
 Schnell, R. 115, 293, 371
 Selvin, H. C. 233
 Stillman, S. 246
 Stuart, E. A. 242, 245, 246, 248

T

Tatham, R. L. 266, 312, 313
 Thursby, J. G. 25
 Tufte, E. R. 133
 Tukey, J. W. 309
 Tukey, P. A. 296
 Tutz, G. 191

V

Valliant, R. 219
 Veall, M. R. 364
 Verba, S. 242, 255

W

Wada, R. 316
 Wainer, H. 185
 Weesie, J. 476
 Weisberg, S. 260, 280, 312
 Welsch, R. E. 288
 West, S. G. 31, 272, 277, 305
 Williams, R. 383

Author index 485

Winship, C. 242
Wooldridge, J. M. 328, 383

Z

Zeger, S. L. 328
Zimmermann, K. F. 364

Subject index

Symbols

$\Delta\beta$	375–376
$\Delta\chi^2$	376–377
β	see regression, standardized coefficient
*	see do-files, comments
+	see operators
==	see operators
//	see do-files, comments
(factor-variable operator)307–308
(factor-variable operator)308
&	see operators
^	see operators
~	see operators
	see operators
	116
1:1 match	418–420
1:m match	425–429

A

Academic Technology Service..	see ATS
added-variable plot289–290
additive index81
adjusted count R^2366–367
adjusted R^2274–275
ado-directories474–475
ado-files	
basics447–449
programming449–465
aggregatesee collapse
Akaike's information criterion367
Aldrich–Nelson's p^2438
_all43
alphanumeric variablessee strings
analysis of variancesee regression

angle()	(axis-label suboption)141–142
ANOVAsee regression
Anscombe quartet279–280, 282, 311–312
anycount() (egen function)92–93
append429–432
arithmetic meansee average
arithmetical expressionssee expressions
ascategory (graph dot option)170
ASCII files397–398, 402–410
ATEsee average treatment effect
ATS470
augmented component-plus-residual plot285
autocode() (function)173
autocorrelationsee regression, autocorrelation
average9–10, 174, 177
average marginal effects361–362, 383
average partial effectssee average marginal effects
average treatment effect245–246
avplots289
aweight (weight type)66–67
axis,	
labels119–120, 139–142
scales131–133
titles119–120, 143–144
transformations132–133
b	
_b [name]75, 262
balanced panel data424
balanced repeated replication219

- bands()** (*twoway mband* option) 283–
 284
bar (graph type) 117–118, 121,
 166–168, 184
 bar charts 166–168, 184
 base category 302
 batch jobs see do-files
 Bayesian information criterion 367
bcskew0 312
 Bernoulli’s distribution see binomial
 distribution
 beta see regression, standardized
 coefficient
 beta distribution 204
 bias 209, 281
bin() (*histogram* option) 190–191
 binary variables see variables, dummy
 binomial distribution 351–352
 BLUE see Gauss–Markov assumptions
 book materials xxii–xxiii
 bookstore 469
 boolean expressions 50, 53–54
 bootstrap 219
box (graph type) 117–118, 187–189
 box plots 15–16, 187–189
 Box–Cox transformation 312
 break see commands, break
browse 396
 by prefix 12–13, 58–59, 86–91, 102,
 178
by() (graph option) 149
by() (tabstat option) 179–180
bysort 59
byte (storage type) 111
- C**
- c. (factor-variable operator) 308
 calculator see pocket calculator
caption() (graph option) 146
capture 34
 categories 158–159
 causality 244–246
cd 3–4
 center see variables, center
- central limit theorem 212–213
 chi-squared
- likelihood-ratio 163
 Pearson 162
- classification tables 364–367
 clock position 135–136
cmdlog 28–30
 CMYK 125
 CNEF 430
 coefficient of determination 269
collapse 86
 comma-separated values see
 spreadsheet format
- command + see commands, break
 command line see windows,
 Command window
- commands
- abbreviations 8, 42
 access previous 8
 break 8
 e-class 71
 end of commands 32
 external 41–42, 449
 internal 41–42, 449
 long see do-files, line breaks
 r-class 71–72
 search 16
 comments see do-files, comments
 component-plus-residual plot 285–286
 compound quotes 460
compress 433
 compute see generate
cond() (function) 84, 460
 conditional-effects plot 273–274, 360,
 380–381
 conditions see if qualifier
 confidence intervals 231–232
connect() (scatter option) 126–129
 connected (plottype) 126–129
 contingency table see frequency
 tables, two-way
- contract** 66
 Cook’s *D* 290–294
cooksd (predict option) 293

- correlation
- coefficient 254–255
 - negative 254
 - positive 253–254
 - weak 254
- count R^2 365–367
- Counterfactual concept of causality ...
..... see causality
- covariate see variables, independent
- covariate pattern 367
- cplot** 255
- cprplot** 285–286
- Cramér's V 163
- .csv see spreadsheet format
- Ctrl+Break** see commands, break
- Ctrl+C** see commands, break
- cumulated probability function see
probit model
- D**
- Data Editor 410–411
- data matrix 395–397
- data region 119
- data types see storage types
- data-generating process 242–243
- data1.dta** 5
- datasets 397–398
- ASCII files 402–410
 - combine 415, 417–432
 - describe 5–6
 - export 433
 - hierarchical 89–90, 425–429
 - import 398–402
 - load 4
 - nonmachine-readable 410–415
 - oversized 435
 - panel data 328–329
 - preserve 126
 - rectangular 396
 - reshape 328–332
 - restore 126
 - save 22, 433
 - sort 9
- date()** (function) 100–101
- dates
- combining datasets by 414
 - display format 99
 - elapsed 99–100
 - from strings 100–101
 - variables 98–102
- .dct (extension) 57
- decode** 319
- DEFF see design effect
- DEFT see design effect
- degrees of freedom see df
- delete see **erase**
- #delimit 32
- density 189, 191–193
- describe** 2, 5–6
- design effect 221–222
- destring** 95
- df 268
- dfbeta** 287–288
- diary 102–103
- dictionary 407–410
- dir** 4, 21
- directory
- change 3–4
 - contents 4, 21
 - working directory xxiii, 3–4, 57
- discard** 449
- discrepancy 292, 373
- discrete** (**histogram** option) 164
- display** 51, 441
- distributions
- describe** 157–199
 - grouped 171–173
- .do (extension) 57
- do** 21
- do-files
- analyzing 35
 - basics 20–21
 - comments 31–32
 - create 35–36
 - editors 20–21
 - error messages 21
 - execute 21
 - exit 34–35
 - from interactive work 25–30

- do-files, *continued*
 line breaks 32
 master 36–39
 organization 35–39
 set more off 33
 version control 33
doedit 20, 26
dot (graph type) 117–118, 169–170,
 184–186
 dot charts 169–170, 184–186
double (storage type) 103, 111
drop 6, 43
.dta (extension) 57
 dummy variables see variables,
 dummy
- E**
e() (saved results) 71–76
e(b) 75
e(V) 75
 e-class see commands, e-class
edit 410–411
 efficiency 209–210
egen 92–94
 elapsed dates see dates, elapsed
 elapsed time see time, elapsed
EMF 154
 Encapsulated PostScript see EPS
encode 95
 endogenous variable see variables,
 dependent
 enhanced metafile see EMF
 Epanechnikov kernel 194
EPS 154
erase 38–39
ereturn list 74
 error messages
 ignore 34
 invalid syntax 11
 error-components model 337–339
estat classification 365
estat dwatson 300
estat gof 368
estat ic 367, 379
estimates restore 75
estimates store 75
esttab 315
 Excel files 397–401
exit (in do-files) 34–35
 exit Stata 22
 exogenous variable see variables,
 independent
exp() (function) 54
expand 66
 expected value 280–281
 export see datasets, export
 expressions 51–55
 extensions 57
- F**
 F test 270
 factor variables 29, 303–304, 307–308,
 380, 382, 383
 FAQ 16, 469
filefilter 404
 filenames 56–57
 Fisher's exact test 162
 five-number summary 177, 187
 fixed format 407–410
 fixed-effects model 332–337
float() (function) 112
float (storage type) 111
foreach 59–62, 424–425, 458
levelsof 458
forvalues 62–63, 208–209
 free format 405–406
 frequencies
 absolute 159–160
 conditional 161–162
 relative 159–160
 frequency tables 14–15
 one-way 159–160
 two-way 160–163
 frequency weights see weights
function (plottype) 117–118
 functions 54–55, 82–85
fweight (weight type) 64–66
fxsize() (graph option) 151–152
fysize() (graph option) 151–152

G

- gamma coefficient 162
 Gauss curve ... see normal distribution
 Gauss distribution.....see normal distribution
 Gauss–Markov assumptions.....279
 GEE.....338–339
 generalized estimation equations ... see GEE
 generate 17–18, 77–91
 generate() (tabulate option)....168, 302–303
 gladder (statistical graph) 119
 .gph (extension).....57
 graph.....115–154
 graph region 119, 130
 graphs
 3-D 119
 combining 150–152
 connecting points.....127–128
 editor..120–121, 137–138, 152–153
 elements.....119–121
 export.....153–154
 multiple.....147–152
 overlay 147–148
 print 152–153
 titles.....146
 types 116–119
 weights 290
 grid lines.....133, 141–142
 grouping
 by quantiles.....172
 intervals with arbitrary width..173
 intervals with same width 172–173
 GSOEP.....xxiii, 5, 106, 108, 219–220, 223, 327, 415–417, 430

H

- help 16–17
 help files.....465–467
 histogram.....164–166, 190–191
 histogram (plottype) 117–118

homoskedasticitysee regression, homoskedasticity

- Hosmer–Lemeshow test 368
 Huber/White/sandwich estimator ..298

I

- i. (factor-variable operator)..303–304, 382
 if qualifier 11, 48–51
 imargin() (graph combine option).... 151
 importsee datasets, import
 importing see data, import
 in qualifier 9, 47–48
 inference
 causal 201, 242–249
 descriptive.....201, 213–241
 statistical.....201–249
 infile.....405–410
 influential cases.....286–295, 372–377
 inlist() (function) 55, 83–84
 input 411–413
 inputting data 410–415
 inrange() (function) 55, 83–84
 insheet 403–404
 inspect 158
 int() (function) 202–203
 interaction terms 304–308, 381–384
 invnormal() (function) ... 80, 204–205
 irecode() (function).....84–85
 iscale() (graph combine option).... 151
 item nonresponse see nonresponse
 iteration block 362–363

J

- jackknife 219
 jitter() (scatter option).....343

K

- kdensity.....194, 205, 210–211
 kdensity (plottype) 194–195
 Kendall's tau-b.....163
 kernel density estimator.....191–195
 key variable.....420–421

L

- label data** 433
- labels,
 - and values 110–111
 - datasets 433
 - display 110–111
 - values 18–19, 108–109
 - variables 18, 108
- legend 119–120, 144–145
- leverage 292, 373
- lfit** (plottype) 147
- likelihood 352
- likelihood-ratio χ^2 364
- likelihood-ratio test 377–379
- limits 3
- line** (plottype) 121, 126–129
- linear probability model see regression, LPM
- linear regression see regression
- linearity assumption 283–286, 369–372
- list** 7–8
- ln()** (function) 54
- local** 73, 439–443
- local macro see macro
- local mean regression 369–370
- loess see LOWESS
- log()** (function) 79–80
- log** (scale suboption) 132–133
- log files
 - finish recording 34
 - interrupt recording 28–29
 - log commands 27–30
 - SMCL 33–34
 - start recording 33–34
- logarithm 79–80
- logical expressions see expressions
- logistic** 358
- logistic regression,
 - coefficients 356–360
 - command 354–356
 - dependent variable 346–350
 - diagnostic 368–377
 - estimation 351–354
 - fit 363–368

- logit** 354–356
- logit model see logistic regression
- logits 349–350
- loops
 - foreach** 59–62
 - forvalues** 62–63
- lower()** (function) 95
- LOWESS 285, 370
- lowess** (plottype) 370
- lowess** (statistical graph) 370
- LPM see regression, LPM

M

- m:1** match see **1:m** match
- macro
 - extended macro functions 461–462
 - local 73–74, 439–443
- manuals xxiv
- margin()** (graph option) 130–131
- marginal effects 361–362
- margins** 264–266, 273–274, 318, 359, 371–372, 380–381, 383
- marginsplot** 265–266, 273–274, 318, 360, 371–372, 380–381
- marker,
 - colors 125
 - labels 119, 134–136
 - options 123
 - sizes 126
 - symbols 119, 123–125
- master data 420
- match see datasets, combine
- match type 420
- matrix** (graph type) 117–118, 283, 286–287
- maximum 10, 176–177
- maximum likelihood
 - principle 351–354
 - search domain 363
- mband** (plottype) 283–284
- mdy()** (function) 99
- mdyhms()** (function) 104
- mean see average
- mean** 214, 220–221, 239

- mean comparison test 238–241
- median 176–177
- median regression 295, 324–327
- median-trace 283–284
- memory see RAM
- merge** 417–429
 - _merge** (variable) 420–421, 423–424
- metadata 424
- mi** 228–230
- mi()** see **missing()**
- minimum 10, 176–177
- minutes()** (function) 104–105
- missing** (tabulate option) 160
- missing values see **missings**
- missing()** (function) 54–55, 305
- missings**
 - coding 413–414
 - definition 7–8
 - encode 107
 - in expressions 50–51
 - set 11–12, 105–106
- ML** see maximum likelihood
- mlabel()** (**scatter** option) 134–136
- mlabposition()** (**scatter** option)
 - 135–136
- mlabsize()** (**scatter** option) 135–136
- mlabvposition()** (**scatter** option) ...
 - 136
- MLE** see maximum likelihood
- mlogit** 388–389
- more off 33, 209
- MSS 268
- multicollinearity 296, 302
- multinomial logistic regression 387–390
- multiple imputation 223–230
- mvdecode** 12, 106
- mvencode** 107

- N**
- _N** 87–88, 102
- _n** 87, 102
- net install** 472–473
- NetCourses 469–470

- O**
- observations
 - definition 5–6
 - list 7–8
- odds 347–349
- odds ratio 347–348, 357–358
- odds-ratio interpretation 357–358
- OLS** 258–260
- operators 52–53
- options 13–14, 45–47
- order** 433
- ordered logistic regression see proportional-odds model
- ordinal logit model see proportional-odds model
- ordinary least squares see OLS
- outreg** 315
- outreg2** 315

- P**
- package description 474
- panel data see datasets, panel data
- PanelWhiz 415
- partial correlation see regression, standardized coefficient
- partial regression plot see added-variable plot
- partial residual plot see component-plus-residual plot
- PDF 154

Pearson residual 367–368
 Pearson- χ^2 367–368
 percentiles see quantiles
 PICT 154
pie (graph type) 117–118, 168–169,
 184
 pie charts 168–169, 184
 plot region 119, 130–131
plotregion() (graph option) 130–
 131
 PNG 154
 pocket calculator 51
 portable document format see PDF
 post-stratification weights see weights
 PostScript see PS
ppfad.dta 424
 PPS see sample, PPS
predict 264
 predicted values 263–266
 Pregibon's $\Delta\beta$ see $\Delta\beta$
preserve 126
 primary sampling unit 215, 220
 probability density function 231
 probability interpretation 359, 360
 probability proportional to size see
 sample, PPS
probit 387
 probit model 385–387
program define 443–447
program drop 445
 programs
 and do-files 444
 debugging 445–446, 455–456
 defining 443–444
 in do-files 446–447
 naming 445
 redefining 445
 syntax 453–456, 459–461
 proportional-odds model 392–393
 PS 154
 pseudo R^2 363–364
 PSID 416, 430
 PSU see primary sampling unit
pwd 21, 57
pweight (weight type) 67–68

Q
 quantile plot 195–198
 quantile regression 325
 quantiles 175–177
 quartiles 176–177
 quietly 458
 Q–Q plots 199

R
 r see correlation coefficient
r() (saved results) 71–72
r(max) (saved result) 71–72
r(mean) (saved result) 62, 71–72
r(min) (saved result) 71
r(N) (saved result) 71–72
r(sd) (saved result) 71–72
r(sum) (saved result) 71
r(sum_w) (saved result) 71
r(Var) (saved result) 71–72
 r-class see commands, r-class
 R^2 269
 RAM 3, 434–435
 random numbers 61, 80, 202–203
 random-effects model 338
range() (scale suboption) 131–132
 RAW 402–403
.raw (extension) 57
 raw data see RAW
rbeta() (function) 204, 243
 recode see variables, replace
recode() (function) 84–85, 173
recode 91–92
 reference lines 119, 133–134
regress 19, 260–261
 regression
 ANOVA table 266–268
 autocorrelation 299–300
 coefficient 261–266, 272–274
 command 260–261, 271–272
 control 277–278
 diagnostics 279–300
 fit 268–270
 homoskedasticity 296–298
 linear 19, 253–340
 LPM 342–346

regression, *continued*

- multiple 270–271
- nonlinear relationships 309–312
- omitted variables 295–296
- panel data 327–339
- residuals 266
- simple 256–260
- standardized coefficient 276–277
- with heteroskedasticity 312–313
- replace** 17–18, 77–91
- reshape** 330–332
- residual** (*predict* option) 266
- residual**
 - definition 257
 - sum 259–260, 267–268
- residual sum of squares see RSS
- residual-versus-fitted plot 281–282, 297–298, 312–313, 345
- response variable see variables, dependent
- restore** 126
- Results window .. see windows, Results window
- resultset** 319
- return list** 72
- reverse** (scale suboption) 132–133
- Review window .. see windows, Review window
- RGB** 125
- rnormal()** (function) 205–206
- root MSE 269
- round()** (function) 82–83
- rowmiss()** (*egen* function) 93
- RSS 267–268
- rstudent** (*predict* option) 297
- Rubin causal model see causality
- runiform()** (function) 61, 80, 202–203, 207
- running counter 87
- running sum 88–90
- rvfplot** 281–282

S

sample

- PPS 216–217

sample, *continued*

- SRS 208, 213–215
- cluster 215
- multistage 216
- stratified 217
- two-stage 215–216
- sample** 207
- sampling distribution 208–213, 217–218
- sampling probability 208, 215–217
- sampling weights see weights
- SAS files 397, 402
- save** 22, 433
- saved results 62, 71–74, 262–263
- scatter** (*plottype*) 117–118, 121
- scatterplot 253–254
- scatterplot matrix 283, 286–287
- scatterplot smoother 283–284
- _se** [*name*] 75
- search** 475
- secondary sampling unit 215
- selection probability see sampling probability
- sensitivity 365–366
- separate** 199
- set obs** 203
- set seed** 202
- sign interpretation 357
- significance test 233–241
- simple random sample see sample, SRS
- SJ** 16, 469
- SJ-ados** 471–473
- SMCL** 33–34, 466–467
- .**smcl** (extension) 57
- SOEP** see GSOEP
- soepuse** 415
- sort** 9
- sort** (*scatter* option) 128–129
- specificity 365–366
- spreadsheet format 402–404
- SPSS files 397, 402
- sqrt()** (function) 54
- SRS see sample, SRS
- SSC** 473

- T**
- tab-separated values... see spreadsheet format
 - tab1** 160
 - tab2** 163
 - table** 180–184
 - tabstat** 177
 - tabulate** 159–163
 - tagged-image file format see TIFF
 - tau-b** see Kendall's tau-b
 - Taylor linearization 219
 - tempvar** 463–465
 - test** 240–241
 - test statistic 235–236
 - text()** (two-way option) 136
 - textbox options 144
 - tick lines 119–120, 142–143
 - TIFF** 154
 - time
 - display-format 104
 - elapsed 102–105
 - from strings 104
 - title()** (graph option) 146
 - total sum of squares see TSS
 - total variance see TSS
 - trace** 446
 - TSS** 267
 - ttest** 240
 - two-way table see frequency tables, two-way
 - twoway** (graph type) 117–118
 - .txt** (extension) 57
- U**
- U-shaped relationship 284, 311–312
 - uniform()** (function)... see **runiform()** (function)
 - uniform distribution 202–204
 - unit nonresponse see nonresponse
 - update** 470–471
 - updating Stata 470–471
 - upper()** (function) 95
 - use** 4
 - using** 56–57
 - using data 420

V

- V* see Cramér's *V*
 value labels see labels, values
valuelabel (axis-label suboption)
 141–142
 variable list see variables, varlist
 variables
 [_all](#) 43
 allowed names 78–79
 categorical 301, 341
 center 29, 62, 72, 272–274, 305
 definition 5–6
 delete 6, 43
 dependent 253
 dummy 80–81, 167–168, 272,
 301–304
 generate 17–18, 77–107
 group 171–173
 identifier 413
 independent 253
 multiple codings 414
 names 107–108
 ordinal 391
 replace 17–18, 77–107
 temporary 463–465
 transformations 119, 286, 295,
 298, 308–313
 varlist 8, 43–45
 Variables window see windows,
 Variables window
 variance see standard deviation
 variance of residuals see RSS
 variation 267
 varlist see variables, varlist
vce(robust) 298
version 33
view 33–34

W

- webuse** xxiv
 weights 63–68, 221–223
 whisker 187
 wildcards 44
 windows
 change 21

windows, *continued*

- Command window 1
 font sizes 2
 preferences 2
 Results window 1
 Review window 1, 8
 scroll back 4
 Variables window 1
 windows metafile see WMF
 WMF 154
 working directory see directory,
 working directory

X

- xlabel()** (twoway option) 139–142
xline() (twoway option) 133–134
.xls see Excel files
xtick() (twoway option) 142–143
xscale() (twoway option) 131–133
xsize() (graph option) 130
xt commands 328
xtgee 338–339
xtick() (twoway option) 142–143
xtitle() (twoway option) 143–144
xtreg 336–338

Y

- ylabel()** (twoway option) 139–142
yline() (twoway option) 133–134
ytick() (twoway option) 142–143
yscale() (twoway option) 131–133
ysize() (graph option) 130
ytick() (twoway option) 142–143
ytitle() (twoway option) 143–144

Z

- z* transformation 235
 zip archive xxiii