A Handbook of Statistical Analyses using Stata

Second Edition

A Handbook of Statistical Analyses using Stata
Second Edition

Sophia Rabe-Hesketh
Brian Everitt

CHAPMAN & HALL/CRC

Boca Raton London New York Washington, D.C.

Library of Congress Cataloging-in-Publication Data

Rabe-Hesketh, S.
A handbook of statistical analyses using Stata / Sophia Rabe-Hesketh, Brian Everitt.—2nd ed.
p. cm.
Rev. ed. of: Handbook of statistical analysis using Stata. c1999.
Includes bibliographical references and index.
ISBN 1-58488-201-8 (alk. paper)
1. Stata. 2. Mathematical statistics—Data processing. I. Everitt, Brian.
II. Rabe-Hesketh, S. Handbook of statistical analysis using Stata. III. Title.
QA276.4 .R33 2000
519,5′0285′5369—dc21 00-027322
CIP

International Standard Book Number 1-58488-201-8
Library of Congress Card Number 00-027322
Printed in the United States of America 1 2 3 4 5 6 7 8 9 0
Printed on acid-free paper

Preface

Stata is an exciting statistical package which can be used for many standard and non-standard methods of data analysis. Stata is particularly useful for modeling complex data from longitudinal studies or surveys and is therefore ideal for analyzing results from clinical trials or epidemiological studies. The extensive graphic facilities of the software are also valuable to the modern data-analyst. In addition, Stata provides a powerful programming language that enables 'taylor-made' analyses to be applied relatively simply. As a result, many Stata users are developing (and making available to other users) new programs reflecting recent developments in statistics which are frequently incorporated into the Stata package.

This handbook follows the format of its two predecessors, *A Handbook of Statistical Analysis using S-Plus* and *A Handbook of Statistical Analysis using SAS*. Each chapter deals with the analysis appropriate for a particular set of data. A brief account of the statistical background is included in each chapter but the primary focus is on how to use Stata and how to interpret results. Our hope is that this approach will provide a useful complement to the excellent but very extensive Stata manuals.

We would like to acknowledge the usefulness of the Stata Netcourses in the preparation of this book. We would also like to thank Ken Higbee and Mario Cleves at Stata Corporation for helping us to update the first edition of the book for Stata version 6 and Nick Cox for pointing out errors in the first edition. This book was typeset using LaTeX.

All the datasets can be accessed on the inernet at the following web-sites:

- http://www.iop.kcl.ac.uk/IoP/Departments/BioComp/stataBook.stm
- http://www.stata.com/bookstore/statanalyses.html

S. Rabe-Hesketh
B. S. Everitt
London, December 99

To my parents, Brigit and George Rabe
Sophia Rebe-Hesketh

To my wife, Mary Elizabeth
Brian S, Everitt

Contents

CHAPTER 1

A Brief Introduction to Stata

1.1 Getting help and information

Stata is a general purpose statistics package which was developed and is maintained by Stata Corporation. There are several forms of Stata, "Intercooled Stata", its shorter version "Small Stata" and a simpler to use (point and click) student package "StataQuest". There are versions of each of these packages for Windows (3.1/3.11, 95, 98 and NT), Unix platforms, and the Macintosh. In this book we will describe Intercooled Stata for Windows although most features are shared by the other versions of Stata.

The Stata package is described in seven manuals (*Stata Getting Started*, *Stata User's Guide*, *Stata Reference Manuals 1-4* and the *Stata Graphics Manual*) and in Hamilton (1998). The reference manuals provide extremely detailed information on each command while the User's guide describes Stata more generally. Features which are specific to the operating system are described in the appropriate *Getting Started* manual, e.g. *Getting started with Stata for Windows*.

Each Stata-command has associated with it a help-file that may be viewed within a Stata session using the help facility. If the required command-name for a particular problem is not known, a list of possible command-names for that problem may be obtained using *search*. Both the help-files and manuals refer to the reference manuals by "[R] command name", to the User's Guide by "[U] chapter number and name", and the graphics manual by "[G] name of entry".

The Stata web-page (http://www.stata.com) contains information on the Stata mailing list and internet courses and has links to files containing extensions and updates of the package (see Section 1.10) as well as a "frequently asked questions" (FAQ) list and further information on Stata.

The Stata mailing list, *Statalist*, simultaneously functions as a technical support service with Stata staff frequently offering very helpful responses to questions. The Statalist messages are archived at:
http://www.hsph.harvard.edu/statalist

Internet courses, called *netcourses*, take place via a temporary mailing list for course organizers and "attenders"; Each week, the course organizers send out lecture notes and exercises which the attenders can discuss with each other

until the organizers send out the answers to the exercises and to the questions raised by attenders.

1.2 Running Stata

When Stata is started, a screen opens as shown in Figure 1.1 containing four windows labeled:

- Stata Command
- Stata Results
- Review
- Variables

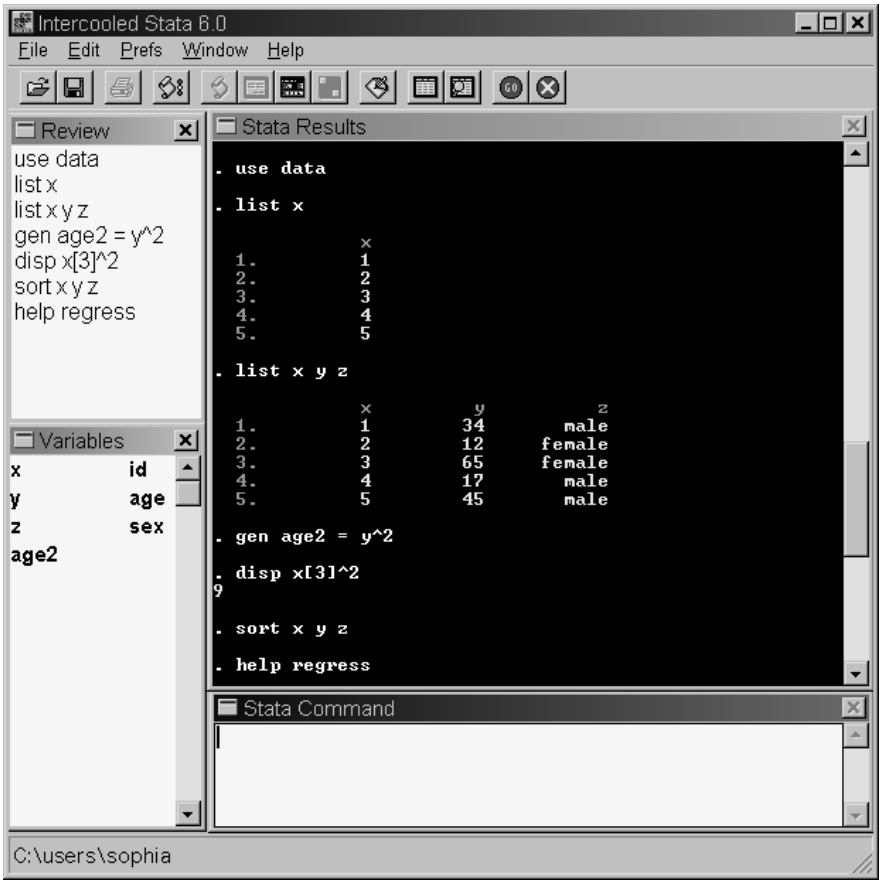

Figure 1.1 *Stata windows.*

A command may be typed in the Stata Command window and executed by pressing the *Return* (or *Enter*) key. The command then appears next to a full stop in the Stata Results window, followed by the output. If the output is longer than the Stata Results window, `--more--` appears at the bottom of the screen. Typing any key scrolls the output forward one screen. The scroll-bar may be used to move up and down previously displayed output. However, only a certain amount of past output is retained in this window. It is therefore a good idea to open a log-file at the beginning of a stata-session. Press the button  , type a filename into the dialog box and choose **Open**. If the file-name already exists, another dialog opens to allow you to decide whether to overwrite the file with new output or whether to append new output to the existing file. The log-file may be viewed during the Stata-session and is automatically saved when it is closed. A log-file may also be opend and closed using commands:

```
log using filename, replace
log close
```

Stata is ready to accept new commands when the prompt (a period) appears at the bottom of the screen. If Stata is not ready to receive new commands because it is still running or has not yet displayed all the current output, it may be interrupted by holding down *Ctrl* and pressing the *Pause/Break* key or by pressing the red **Break** button  .

A previous command can be accessed using the *PgUp* and *PgDn* keys or by selecting it from the Review window where all commands from the current Stata session are listed. The command may then be edited if required before pressing *Return* to execute the command. In practice, it is useful to build up a file containing the commands necessary to carry out a particular data-analysis. This may be done using Stata's **Do-file Editor**. The editor may be opened by clicking  . Commands that work when used interactively in the command window can then be copied into the editor. The do-file may be saved and all the commands contained in it may be executed by clicking  in the do-file editor or using the command

```
do dofile
```

A single dataset may be loaded into Stata. As in other statistical packages, this dataset is generally a matrix where the columns represent variables (with names and labels) and the rows represent observations. When a dataset is open, the variable names and variable labels appear in the Variables window. The dataset may be viewed as a spread-sheet by opening the **Data Browser** with the  button and edited by clicking  to open the **Data Editor**. See Section 1.3 for more information on datasets.

Most Stata commands refer to a list of variables, the basic syntax being

```
command varlist
```

For example, if the dataset contains variables `x` `y` and `z`, then

```
list x y
```

lists the values of `x` and `y`. Other components may be added to the command, for example adding `if exp` after `varlist` causes the command to process only those observations satisfying the logical expression `exp`. Options are separated from the main command by a comma. The complete command structure and its components are described in Section 1.4.

Help may be obtained by clicking on **Help** which brings up the dialog box shown in Figure 1.2. To obtain help on a Stata command, assuming the com-

Figure 1.2 *Dialog box for help.*

mand name is known, select **Stata Command...**. To find the appropriate Stata command first, select **Search...**. For example, to find out how to fit a Cox regression, we can select **Search...**, type "survival" and press **OK**. This gives a list of relevant command names or topics for which help-files are available. Each entry in this list includes a green keyword (a hyperlink) that may be selected to view the appropriate help-file. Each help-file contains hyperlinks to other relevant help-files. The search and help-files may also be accessed using the commands

```
search survival
help cox
```

except that the files now appear in the Stata Results window where no hyperlinks are available.

The selections **News**, **Official Updates**, **STB and User-written Programs** and **Stata Web Site** all enable access to relevant information on the Web providing the computer is connected to the internet (see Section 1.10 on keeping Stata up-to-date).

Each of the Stata windows may be resized and moved around in the usual way. The fonts in a window may be changed by clicking on the menu button  on the top left of that window's menu bar. All these setting are automatically saved when Stata is exited.

Stata may be exited in three ways:

- click into the **Close** button  at the top right hand corner of the Stata screen
- select the **File** menu from the menu bar and select **Exit**
- type `exit, clear` in the Stata Commands window and press *Return*.

1.3 Datasets in Stata

1.3.1 Data input and output

Stata has its own data format with default extension *.dta*. Reading and saving a Stata file are straightforward. If the filename is *bank.dta*, the commands are

```
use bank
save bank
```

If the data are not stored in the current directory, then the complete path must be specified, as in the command

```
use c:\user\data\bank
```

However, the least error prone way of keeping all the files for a particular project in one directory is to change to that directory and save and read all files without their pathname:

```
cd c:\user\data
use bank
save bank
```

When reading a file into Stata, all data already in memory need to be cleared, either by running `clear` before the `use` command or by using the option `clear` as follows:

```
use bank, clear
```

If we wish to save data under an existing filename, this results in an error message unless we use the option `replace` as follows:

```
save bank, replace
```

If the data are not available in Stata format, they may be converted to Stata format using another package (e.g. Stat/Transfer) or saved as an ASCII file (although the latter option means loosing all the labels). When saving data as ASCII, missing values should be replaced by some numerical code.

There are three commands available for reading different types of ASCII data: `insheet` is for files containing one observation (on all variables) per line with variables separated by tabs or commas, where the first line may contain the variable names; `infile` with varlist (free format) allows line breaks to occur anywhere and variables to be separated by spaces as well as commas or tabs; and `infile` with a dictionary (fixed format) is the most flexible command. Data may be saved as ASCII using `outfile` or `outsheet`.

Only one dataset may be loaded at any given time but a dataset may be merged with the currently loaded dataset using the command `merge` or append to add observations or variables.

1.3.2 Variables

There are essentially two types of variables in Stata, string and numeric. Each variable can be one of a number of storage types that require different numbers of bytes. The storage types are byte, int, long, float, and double for numeric variables and str1 to str80 for string variables of different lengths. Besides the storage type, variables have associated with them a name, a label, and a format. The name of a variable `y` can be changed to `x` using

```
rename y x
```

The variable label can be defined using

```
label variable x "cost in pounds"
```

and the format of a numeric variable can be set to "general numeric" with two decimal places using

```
format x %7.2g
```

Numeric variables

Missing values in numeric variables are represented by dots only and are interpreted as very large numbers (which can lead to mistakes). Missing value codes may be converted to missing values using the command `mvdecode`. For example,

```
mvdecode x, mv(-99)
```

replaces all values of variable `x` equal to −99 by dots and

```
mvencode x, mv(-99)
```

changes the missing values back to −99.

Numeric variables can be used to represent categorical or continuous variables including dates. For categorical variables it is not always easy to remember which numerical code represents which category. Value labels can therefore be defined as follows:

```
label define s 1 married 2 divorced 3 widowed 4 single
label values marital s
```

The categories can also be recoded, for example

```
recode marital 2/3=2 4=3
```

merges categories 2 and 3 into category 2 and changes category 4 to 3.

Dates are defined as the number of days since 1/1/1960 and can be displayed using the date format `%d`. For example, listing the variable `time` in `%7.0g` format gives

```
list time
```

	time
1.	14976
2.	200

which is not as easy to interpret as

```
format time %d
list time
```

	time
1.	01jan2001
2.	19jul1960

String variables

String variables are typically used for categorical variables or in some cases for dates (e.g. if the file was saved as an ASCII file from SPSS). In Stata it is generally advisable to represent both categorical variables and dates by numeric variables, and conversion from string to numeric in both cases is straightforward. A categorical string variable can be converted to a numeric variable using the command `encode` which replaces each unique string by an integer and uses that string as the label for the corresponding integer value. The command `decode` converts the labeled numeric variable back to a string variable.

A string variable representing dates can be converted to numeric using the function `date(string1, string2)` where `string1` is a string representing a

date and `string2` is a permutation of `"dmy"` to specify the order of the day, month and year in `string1`. For example, the commands

```
display date("30/1/1930","dmy")
```

and

```
display date("january 1, 1930", "mdy")
```

both return the negative value -10957 because the date is 10957 days before 1/1/1960.

1.4 Stata commands

Typing `help language` gives the following generic command structure for most Stata commands.

```
[by varlist:] command [varlist] [=exp] [if exp] [in range] [weight]
                                       [using filename]  [, options]
```

The help-file contains links to information on each of the components, and we will briefly describe them here:

[**by varlist:**] instructs Stata to repeat the command for each combination of values in the list of variables `varlist`.

[**command**] is the name of the command and can be abbreviated; for example, the command `display` can be abbreviated as `dis`.

[**varlist**] is the list of variables to which the command applies.

[**=exp**] is an expression.

[**if exp**] restricts the command to that subset of the observations that satisfies the logical expression `exp`.

[**in range**] restricts the command to those observations whose indices lie in a particular `range`.

[**weight**] allows weights to be associated with observations (see Section 1.6).

[**using filename**] specifies the filename to be used.

[**options**] are specific to the command and may be abbreviated.

For any given command, some of these components may not be available, for example `list` does not allow [`using filename`]. The help-files for specific commands specify which components are available, using the same notation as above, with square brackets enclosing components that are optional. For example, `help log` gives

```
log using filename [, noproc append replace ]
```

implying that `[by varlist:]` is not allowed and that `using filename` is required whereas the three options `noproc`, `append` or `replace` are optional.

The syntax for `varlist`, `exp` and `range` is described in the next three subsections, followed by information on how to loop through sets of variables or observations.

1.4.1 Varlist

The simplest form of `varlist` is a list of variable names separated by spaces. Variable names may also be abbreviated as long as this is unambiguous, i.e. `x1` may be referred to by `x` only if there is no other variable starting on x such as `x` itself or `x2`. A set of adjacent variables such as `m1`, `m2` and `x` may be referred to as `m1-x`. All variables starting on the same set of letters can be represented by that set of letters followed by a wild card *, so that `m*` may stand for `m1 m6 mother`. The set of all variables is referred to by `_all`. Examples of a `varlist` are

```
x y
x1-x16
a1-a3 my* sex age
```

1.4.2 Expressions

There are logical, algebraic and string expressions in Stata. Logical expressions evaluate to 1 (true) or 0 (false) and use the operators `<` and `<=` for "less than" and "less than or equal to" respectively and similarly `>` and `>=` are used for "greater than" and "greater than or equal to". The symbols `==` and `~=` stand for "equal to" and "not equal to", and the characters `~`, & and `|` represent "not", "and" and " or" respectively, so that

```
if (y~=2&z>x)|x==1
```

means "if y is not equal to two and z is greater than x or if x equals one". In fact, expressions involving variables are evaluated for each observation so that the expression really means

$$(y_i \neq 2 \& z_i > x_i) | x_i == 1$$

where i is the observation index.

Algebraic expressions use the usual operators `+ - * /` and `^` for addition, subtraction, multiplication, division, and powers respectively. Stata also has many mathematical functions such as `sqrt()`, `exp()`, `log()`, etc. and statistical functions such as `chiprob()` and `normprob()` for cumulative distribution functions and `invnorm()`, etc. for inverse cumulative distribution functions. Pseudo-random numbers may be generated using `uniform()`. Examples of algebraic expressions are

```
y + x
(y + x)^3 + a/b
invnorm(uniform())+2
```

where `invnorm(uniform())` returns a (different) sample from the standard normal distribution for each observation.

Finally, string expressions mainly use special string functions such as `substr(str,n1,n2)` to extract a substring starting at `n1` for a length of `n2`. The logical operators `==` and `~=` are also allowed with string variables and the operator `+` concatinates two strings. For example, the combined logical and string expression

```
("moon"+substr("sunlight",4,5))=="moonlight"
```

returns the value 1 for "true".

For a list of all functions, use `help functions`.

1.4.3 Observation indices and ranges

Each observation has associated with it an index. For example, the value of the third observation on a particular variable `x` may be referred to as `x[3]`. The macro `_n` takes on the value of the running index and `_N` is equal to the number of observations. We can therefore refer to the previous observation of a variable as `x[_n-1]`.

An indexed variable is only allowed on the right hand side of an assignment. If we wish to replace `x[3]` by 2, we can do this using the syntax

```
replace x=2 if _n==3
```

We can refer to a range of observations using either `if` with a logical expression involving `_n` or, more easily by using `in range`, where `range` is a range of indices specified using the syntax `f/l` (for "first to last") where `f` and/or `l` may be replaced by numerical values if required, so that `5/12` means "fifth to twelfth" and `f/10` means "first to tenth" etc. Negative numbers are used to count from the end, for example

```
list x in -10/l
```

lists the last 10 observations.

1.4.4 Looping through variables or observations

Explicitly looping through observations is often not necessary because expressions involving variables are automatically evaluated for each observation. It may however be required to repeat a command for subsets of observations and this is what `by varlist:` is for. Before using `by varlist:`, however, the data must be sorted using

```
sort varlist
```

where varlist includes the variables to be used for `by varlist:`. Note that if `varlist` contains more than one variable, ties in the earlier variables are sorted according to the next variable. For example,

```
sort school class
by school class: summ test
```

give the summary statistics of `test` for each class. If `class` is labeled from 1 to n_i for the ith school, then not using `school` in the above commands would result in the observations for all classes labeled 1 to be grouped together.

A very useful feature of `by varlist:` is that it causes the observation index `_n` to count from 1 within each of the groups defined by the unique combinations of the values of varlist. The macro `_N` represents the number of observations in each group. For example,

```
sort group age
by group: list age if _n==_N
```

lists `age` for the last observation in each group where the last observation in this case is the observation with the highest age within its group.

We can also loop through a set variables or observations using `for`. For example,

```
for var v*: list X
```

loops through the list of all variables starting on `v` and applies the command `list` to each member `X` of the variable list. Numeric lists may also be used. The command

```
for num 1 3 5: list vX
```

lists `v1`, `v3` and `v5`. Numeric lists may be abbreviated by "first(increment)last", giving the syntax `1(2)5` for the list `1 3 5` (not an abbreviation in this case!). The `for` command can be made to loop through several lists (of the same length) simultaneously where the "current" members of the different lists are referred to by `X`, `Y`, `Z`, `A`, `B` etc. For example, if there are variables `v1`, `v2`, `v3`, `v4`, and `v5` in the dataset,

```
for var v1-v5 \num 1/5: replace X=0 in Y
```

replaces the ith value of the variable `vi` by 0, i.e., it sets vi[i] to 0. Here, we have used the syntax "first/last" which is used when the increment is 1. See `help for` for more information, for example on how to construct nested loops.

Another method for looping is the `while` command which is described in Section 1.9 on programming but may also be used interactively.

1.5 Data management

1.5.1 Generating variables

New variables may be generated using the commands `generate` or `egen`. The command `generate` simply equates a new variable to an expression which is evaluated for each observation. For example,

```
generate x=1
```

creates a new variable called `x` and sets it equal to one. When `generate` is used together with `if exp` or `in range`, the remaining observations are set to missing. For example,

```
generate percent = 100*(old - new)/old if old>0
```

generates the variable `percent` and set it equal to the percentage decrease from `old` to `new` where `old` is positive and equal to missing otherwise. The function `replace` works in the same way as `generate` except that it allows an existing variable to be changed. For example,

```
replace percent = 0 if old<=0
```

changes the missing values in `percent` to zeros. The two commands above could be replaced by the single command

```
generate percent=cond(old>0, 100*(old-new)/old, 0)
```

where `cond()` evaluates to the second argument if the first argument is true and to the third argument otherwise.

The function `egen` provides an extension to `generate`. One advantage of `egen` is that some of its functions accept a variable list as an argument, whereas the functions for `generate` can only take simple expressions as arguments. for example, we can form the average of 100 variables `m1` to `m100` using

```
egen average=rmean(m1-m100)
```

where missing values are ignored. Other functions for `egen` operate on groups of observations. For example, if we have the income (variable `income`) for members within families (variable `family`), we may want to compute the total income of each member's family using

```
egen faminc = sum(income), by(family)
```

An existing variable can be replaced using egen functions only by first dropping it using

```
drop x
```

Another way of dropping variables is using `keep varlist` where `varlist` is the list of all variables not to be dropped.