STATISTICS/DATA ANALYSIS

STATA™

REFERENCE MANUAL

The Computing Resource Center
10801 National Boulevard
Los Angeles, California 90064
Copyright ©1984

STATISTICS/DATA ANALYSIS

STATA™

REFERENCE MANUAL

The Computing Resource Center
10801 National Boulevard
Los Angeles, California 90064

Copyright ©1984

First Edition (November 1984)

Version 1.0

Copyright © Computing Resource Center 1984

Computing Resource Center provides this manual "as is" without warranty of any kind, either express or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. CRC may make improvements and/or changes in the product(s) and the program(s) described in this manual at any time and without notice.

This manual contains proprietary information protected by copyright. All rights are reserved. No part of this manual may be photocopied or reproduced in any form without prior written consent from Computing Resource Center.

The software described in this manual is furnished under a license agreement or nondisclosure agreement. The software may be copied only in accordance with the terms of the agreement. It is against the law to copy STATA onto cassette tapes, disk, diskette, or any other medium for any purpose other than backup or archival purposes.

STATA is a trademark of Computing Resource Center. IBM is a trademark of International Business Machines Corporation.

How to Use this Manual

Two manuals accompany the STATA program - the STATA User´s Guide and the STATA Reference Manual. The STATA User´s Guide gives an overview of STATA. If you are new to STATA, you should consult the User´s Guide for information on its features.

This document is the STATA Reference Manual. It provides a technical reference to every feature of STATA. Each STATA command is presented in alphabetical order. The syntax and function of the command, along with all its options, are explained. The Reference Manual is, in essence, STATA´s dictionary. It will not teach you how to "speak" STATA, but will refresh and augment your knowledge of particular STATA commands and options.

Contents

STATA Fundamentals

How to Start and Stop STATA

Before using STATA for the first time you must install it on your PC (See Appendix F. Installation Instructions).

Your computer must be turned on, DOS must be loaded, and the STATA.EXE file must be available on the current drive and directory. To begin to run STATA, type "STATA" and press the Return key. Something like the following will appear on your monitor:

```
A>stata

==S=T=A=T=A==1.0== Copyright (C) 1984 by ==C=R=C==


. _
```

The final line contains a period (.) and the cursor (indicated by _). The period is the STATA prompt. It indicates that STATA is running and is ready to accept your next command.

To exit from STATA and return to DOS, use the **exit** command. If the data set in memory has changed since the last time it was saved, you must type **exit, clear** to leave STATA. This prevents you from accidentally losing your data.

You may optionally include any STATA command on the invocation line. For instance, typing **stata run profile** in response to the DOS prompt will start STATA and execute the do file "profile.do". This method of starting STATA is useful if you wish to have STATA perform a certain "set-up" procedure every time you enter it.

Keyboard Entry

In addition to the normal typewriter keys, certain keys provide editing functions in STATA. These functions affect the current line, that is, the command you are in the process of typing. The STATA editing keys are:

Esc deletes the line and lets you start over.

Backspace (the key to the right of =) backs up and deletes one character.

Left-Arrow (4 on the numeric keypad) backs up one character but leaves the character on the screen. You may type over the character, or use the Right-Arrow to space over it, leaving it in place.

Right-Arrow (6 on the numeric keypad) moves the cursor forward one position, retrieving the previous character typed (from Backspace or Left-Arrow or from the previous line).

Ctrl-Right-Arrow (hold down Ctrl and press 6 on the numeric keypad) moves the cursor to the end of the line, retrieving the previously typed characters (from Backspace or Left-Arrow or from the previous line).

Ctrl-Break (hold down Ctrl and press Break) tells STATA to cancel what it is doing and to return control to the keyboard as soon as possible. You may press Ctrl-Break at any time.

Input Prompt and Return Message

STATA prompts you to enter a command with a single period on the left hand side of the screen. After you enter a command, STATA produces any output generated by the command and displays a return message. For instance:

```
. drop _all
R; T=0.16 15:52:10
```

The user typed **drop _all** after the period prompt, and STATA responded with the message "R; T=0.16 15:52:10". The "R;" indicates that STATA successfully completed the command. The "T=0.16" shows the amount of time, in seconds, it took STATA to perform the command (timed from the point at which the user pressed Return to the time at which STATA typed its return message). In addition, STATA shows the time-of-day using a 24-hour clock. This command completed at 3:52 p.m.

When an error occurs, STATA produces an error message and a return code. For instance:

```
. list myvar
No variables defined
R(111); T=0.11 15:52:25
```

After typing **drop _all**, the user asked STATA to list the variable named myvar. STATA responded with the message "No variables defined" and a slight variation on the return message. The number after the R (in this case 111) is called the return code. Errors are numbered in STATA. You need not keep track of the numbers since STATA always displays a message describing the problem. However, if you want more information you can look up the return code in Appendix C, Messages and Return Codes.

The Elements of STATA

Numbers

A number may contain a sign, an integer part, a decimal point, a fraction part, an **e** or **E**, and a signed integer exponent. Numbers may not contain commas; for example, the number 1,024 must be typed as 1024 (or 1024. or 1024.0 or ...). The following are examples of valid numbers:

```
5
-5
5.2
.5
5.2e+2
5.2e-2
```

A number can also take on the special value "missing", denoted by a single period (.). You may specify a missing value any place you may specify a number. Do not place the period in double quotes or STATA will interpret it as a string. Missing values differ from ordinary numbers in one respect: any arithmetic operation on a missing value yields a missing value.

Technical Note: Numbers can be stored in one of four variable types: **int**, **long**, **float** (the default), or **double**. **ints** are stored in 2 bytes, **longs** and **floats** in 4 bytes, and **doubles** in 8 bytes. **ints** may contain any number between -32,768 and 32,766 inclusive, and missing values are stored as 32,767. **longs** may contain any number between -2,146,483,648 and 2,147,483,646 inclusive, and missing values are stored as 2,147,483,647. **floats** may contain any number between +/-10^-37 and +/-10^37, and missing values are stored as 2^128. **doubles** may contain any number between +/-10^-99 and +/-10^99, and missing values are stored as 2^333.

Do not confuse the term integer, which is a characteristic of a number, with **int**, which is a storage type. For instance, 5 is an integer no matter how it stored. Thus if you read that an argument is required to be an integer that does not mean that it must be stored as an **int**.

Names

A name is a string of one to eight letters (A-Z and a-z), digits (0-9), and underscore (_). STATA reserves the names **double**, **float**, **if**, **in**, **int**, **long**, **using**, **with**, **_all**, **_b**, **_coef**, **_n**, **_N**, **_pi**, and **_rc**. You may not use these reserved names for your variables. The first character of a name must be either a letter or an underscore. We recommend, however, that you do not begin your variable names with an underscore. All STATA built-in variables begin with an underscore, and we reserve the right to incorporate new _variables freely.

STATA respects case, that is, **myvar**, **Myvar**, and **MYVAR** are distinct names in STATA.

Raw Data

Raw data (**data**) is a rectangular table of numeric values where each row is an observation on all the variables and each column contains all the observations on a single variable. Observations are numbered sequentially from 1 to **_N**. The following example of **data** contains the first five odd and the first five even numbers:

Elements

	odd	even
1.	1	2
2.	3	4
3.	5	6
4.	7	8
5.	9	10

The observations are numbered 1 to 5 and the variables are named **odd** and **even**.

Cross-Product

A data set can be stored as a cross-product (**xp**) rather than as **data**. Define **X** to be a raw data matrix augmented on the left with a system variable (**_cons**) every element of which is equal to one. The **xp** form of this data is the matrix inner product **X′X.** For the example above, the corresponding **xp** form is:

	_cons	odd	even
1.	5	24	29
2.	24	156	180
3.	29	180	209

Several STATA commands, such as **regress** and **correlate**, execute more rapidly if the data set is stored in **xp** form. In addition, STATA can process a data set with an unlimited number of observations if the data set is stored in **xp** form. The **convert** command can be used to transform **data** format data sets to **xp** data sets. Note that not all STATA commands can be used on **xp** data sets. See the description of the **convert** command for more details.

It is also possible to enter a cross-product directly into STATA (using **input** or **infile**) and then to direct STATA to interpret the data as a cross-product via the **set contents xp** command. If you use this method, you must be careful that the data you input conforms exactly to the description of **X′X** given above. For example, the value in the first row and first column must contain the number of observations in the data set. When you **input** or **infile** your data, you may give the first variable any name except **_cons**. The **set contents xp** command automatically renames the first variable **_cons** if the data set meets the requirements of a cross-product. **_cons** is the special name STATA reserves for the first column of a cross-product. (Note: As a convenience, STATA allows you to set one, but not both, of a pair of corresponding off-diagonal elements of a cross-product to missing value. The non-missing value will be automatically copied over the missing value by **set contents xp**.)

Technical Note: When STATA creates a cross-product, it stores all the data as **doubles**. If you create your own cross-product directly, we recommend that you do likewise. However, you may use any variable types you wish.

Language Syntax

With few exceptions, the basic language syntax is:

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

where square brackets denote optional qualifiers. In this diagram, "varlist" denotes a list of variable names, "command" denotes a STATA command, "exp" denotes an algebraic expression, "range" denotes an observation range, and "options" denotes a list of options.

Most commands that take a subsequent varlist do not require one to be explicitly typed. If no varlist appears, these commands assume a varlist of **_all**, the STATA shorthand for indicating all the variables in the data set. In commands that alter or destroy data, STATA always requires that the varlist be specified explicitly.

The **by** varlist: prefix and the **if** exp and **in** range qualifiers are described completely in the Command Reference section of this manual. Briefly, the **by** varlist: prefix causes STATA to repeat a command for each subset of the data for which the values of the variables in the varlist are equal. The **if** exp qualifier restricts the scope of the command to those observations for which the value of the expression is non-zero. The **in** range qualifier restricts the scope of the command to a specific observation range.

The **=exp** phrase serves two different functions. In the **generate** and **replace** commands, =exp specifies the values to be assigned to a variable. In other STATA commands, =exp is used to indicate the weight to attach to each observation. In these latter commands, failing to specify a weight is equivalent to specifying **=1**.

Many commands take command specific options. These are described along with each command in the Command Reference section of this manual.

STATA treats any line starting with a "*" as a comment and ignores it.

Abbreviation Rules

Command, variable, and option names may be abbreviated to the shortest string of characters that uniquely identifies them. For instance, there are four commands that start with the letter "r": **regress**, **rename**, **replace**, and **run**. Therefore **regress** may be abbreviated as **regres**, **regre**, **regr**, or **reg**. It may not be abbreviated as **re** since this string does not distinguish **regress** from **rename** and **replace**.

There is one exception to the abbreviation rule: if a command or option alters or destroys data, then the command or option name must be spelled out completely. For example, the **drop** command may not be abbreviated.

varlists

A varlist is a list of variable names. The variable names in a varlist refer exclusively either to new (not yet created) variables or to existing variables.

In lists of existing variable names, variable names may be repeated in the varlist. The variable names may also be abbreviated. A "*" may be appended to a partial variable name to indicate all variables that start with that letter combination. For example, if the variables poplt5, pop5to6, and pop18p are in your data set, you may type pop* as a shorthand way to refer to all three variables. You may also place a dash (-) between two variable names to specify all the variables stored between the two listed variables inclusive. (The **describe** command lists variables in the order in which they are stored.)

In lists of new variables, no variable names may be repeated or abbreviated in the varlist. You may specify a dash (-) between two variable names that have the same letter prefix and that end in numbers. This form of the dash notation indicates a range of variables in ascending numerical order. For instance, typing "v1-v4" is equivalent to typing "v1 v2 v3 v4".

In lists of new variables, you may type the name of a storage type before the variable name to force a storage type other than the default. The storage types are **int**, **long**, **float** (the default), and **double**. For instance, the list "var1 int var2 var3" specifies that var1 and var3 are to be given the default storage type, and var2 is to be stored as an **int**. You may use parentheses to bind a list of variable names. The list "var 1 int(var2 var3)" specifies that both var2 and var3 are to be stored as **ints**.

In lists of new variables, you may also append a colon and a value label name. For instance, "var1 var2:myfmt" specifies that the variable var2 is to be associated with the value labels stored under the name myfmt. This has the same effect as typing the list "var1 var2" and then subsequently giving the command

label values var2 myfmt

The advantage of specifying the value label association with the colon notation is that the value labels can then be used by the current command. (See the descriptions of the **input** and **infile** commands for further explanations of using the colon notation.)

Expressions

STATA includes a complete expression parser. Algebraic expressions are specified in a natural way using the standard rules of hierarchy. For instance, **myvar+2/othvar** is interpreted as **myvar+(2/othvar)**. You may use parentheses freely to force a different order of evaluation.

Operators

The arithmetic operators in STATA are: + (addition), - (subtraction), * (multiplication), / (division), ^ (raise to a power), and the prefix - which indicates negation. Any arithmetic operation on a missing value or any impossible arithmetic operation (such as division by zero) yields a missing value.

The relational operators in STATA are: > (greater than), < (less than), >= (greater than or equal), <= (less than or equal), == (equal), and ~= (not equal). Note that the relational operator for equality is a pair of equal signs. This convention distinguishes relational equality from the =exp phrase.

Relational expressions are either true (denoted by **1**) or false (denoted by **0**). Relational operations are performed after all arithmetic operations. Thus the expression **(3>2)+1** is equal to **2** while **3>2+1** is equal to **0**. Missing values may appear in relational expressions. The expression **x==.** is true (equal to **1**) if x is missing and false (equal to **0**) otherwise. A missing value is greater than any non-missing value.

The logical operators in STATA are: & (and), | (or), and ~ (not). On input, the logical operators interpret any non-zero value (including missing value) as true and zero as false. Like the relational operators, they return the value **1** for true and **0** for false. For example, the expression **5 & .** is equal to **1**. Logical operations, except for **~**, are performed after all arithmetic and relational operations; the expression **3>2 & 5>4** is interpreted as **(3>2)&(5>4)** and is equal to **1**.

The order of evaluation (from first to last) of all the operators is: **-** (negation), **~**, **^**, **/**, *****, **-** (subtraction), **+**, **~=**, **>**, **<**, **<=**, **>=**, **==**, **&**, **|**.

Functions

Functions may appear in expressions. Functions are indicated by the function name, an open parenthesis, an expression or expressions separated by commas, and a close parenthesis. For example, the square root of a variable named x is specified by typing **sqrt(x)**. All functions return missing values when given missing values as arguments or when the result is undefined.

The mathematical functions in STATA are: **abs(**x**)** (absolute value), **atan(**x**)** (arc-tangent returning radians), **cos(**x**)** (cosine of radians), **exp(**x**)** (exponent), **mod(**x,y**)** (the modulus of x with respect to y), **sin(**x**)** (sine of radians), and **sqrt(**x**)** (square root).

The statistical functions in STATA are: **chiprob(**df,x**)** (the cumulative chi-square with df degrees of freedom and value x), **fprob(**df1,df2,f**)** (the cumulative F-distribution with df1 numerator and df2 denominator degrees of freedom), **invnorm(**p**)** (the inverse cumulative normal), **normprob(**z**)** (the cumulative normal), and **tprob(**df,t**)** (Student´s cumulative t-distribution with df degrees of freedom).

STATA includes a random number function, **uniform()**, which takes no arguments (although you must include the open and close parentheses). It produces uniformly distributed pseudo-random numbers over the open interval zero to one. Every time STATA is started, **uniform()** produces the same sequence of numbers. The seed value, and hence the sequence of pseudo-random numbers, can be changed with the **set seed** command.

STATA also includes the following special functions:

autocode(x,ng,xmin,xmax) partitions the interval from xmin to xmax into ng equal length intervals and returns the upper bound of the interval which contains x. This function is an automated version of **recode()** (see below). The algorithm for **autocode** is

```
if (x==. | ng==. | xmin==. | xmax==. | ng<=0
   | xmin>=xmax) then return .
otherwise
   for i=1 to ng-1
     xmap=xmin+i*(xmax-xmin)/ng
     if x<=xmap then return xmap
   end
   otherwise
      return xmax
```

cond(x,a,b) returns a if x evaluates to true (not **0**) and b if x evaluates to false (**0**). For example,

generate maxinc=cond(inc1>inc2,inc1,inc2)

creates the variable maxinc as the maximum of inc1 and inc2.

float(x) returns the value of x rounded to **float**. Although you may store your variables as **double**,

float, **long**, or **int**, STATA converts all numbers to **double** before performing any calculations. As a consequence, difficulties can arise when comparing numbers that have no finite digit binary representation. For example, if the variable x is stored as a **float** and contains the value 1.1 (a repeating decimal in binary) the expression **x==1.1** will evaluate to false because the literal **1.1** is the **double** representation of 1.1 which is different than the **float** representation stored in x. The expression **x==float(1.1)** will evaluate to true, because the **float** function converts the literal **1.1** to its **float** representation before it is compared to x.

group(x) creates a categorical variable that divides the data into x as near equally sized subsamples as possible, numbering the first group 1, the second 2, and so on.

int(x) returns the integer obtained by truncating x.

max(x1,x2,...,xn) returns the maximum of x1, x2, ..., xn. Missing values are ignored. If all the arguments are missing, missing is returned.

min(x1,x2,...,xn) returns the minimum of x1, x2, ..., xn. Missing values are ignored. If all the arguments are missing, missing is returned.

recode(x,x1,x2,...,xn) returns missing if x is missing, x1 if x<=x1, x2 if x<=x2, ..., or xn if is greater than x1, x2, ... , x(n-1).

sign(x) returns missing if x is missing, -1 if x<0, 0 if x==0, and 1 if x>0.

sum(x) returns the running sum of x, treating missing values as zero. For example, following the command

```
generate y = sum(x)
```

the i-th observation on y contains the sum of the first through i-th observations on x.

System Variables (_variables)

Expressions may also contain _variables (pronounced "underscore variables"). These are built-in, system variables that are created and updated by STATA. They are called _variables because their names all begin with the underscore character (_).

The _variables in STATA are:

_coef[varname] (synonym: **_b**[varname]) contains the value (to machine precision) of the coefficient on varname from the last most recent regression.

_cons is always equal to the number **1**.

_n contains the number of the current observation.

_N contains the total number of observations in the data set.

_pi contains the value of pi to machine precision.

_pred contains the predicted values of the dependent variable from the most recent regression. The predictions are formed using the current values of the regressors which may not be the same as the values they contained when the regression was run. As a result, **_pred** can be used to calculate forecasts and other out-of-sample predictions. For instance, you may **use** one data set, run a regression, then **use** another data set and make predictions using **_pred**.

Elements

_rc contains the value of the return code from the most recent **capture** command.

Explicit Subscripting

Individual observations on variables can be referenced by subscripting the variables. Explicit subscripts are specified by following a variable name with square brackets that contain an expression. The result of the subscript expression is truncated to an integer and the value of the variable for the indicated observation is returned. If the value of the subscript expression is less than **1** or greater than **_N**, a missing value is returned.

As an example, the lagged value of a variable x can be **generated** by

```
generate xlag = x[_n-1]
```

The first observation on xlag is equal to missing value.

When a command is preceded by the **by** varlist: prefix, subscript expressions and the _variables **_n** and **_N** are evaluated relative to the subset of the data currently being processed. For example, in the data set

```
      bvar  oldvar
  1.    1     1.1
  2.    1     2.1
  3.    1     3.1
  4.    2     4.1
  5.    2     5.1
```

the command

```
by bvar: gen newvar=oldvar[1]
```

will produce

```
        bvar  oldvar    newvar
  1.     1      1.1       1.1
  2.     1      2.1       1.1
  3.     1      3.1       1.1
  4.     2      4.1       4.1
  5.     2      5.1       4.1
```

Label Values

You may use labels in an expression in place of the numeric values with which they are associated. To use a label in this way, type the label in double quotes followed by a colon and the name of the value label. For instance, if the value label yesno associates the label "yes" with **1** and the label "no" with **0**, then **"yes":yesno** is evaluated as **1**. If the double quoted label is not defined in the indicated value label, or the value label itself is not found, a missing value is returned. Thus, the expression **"maybe":yesno** is evaluated as a missing value.

STATA Command Reference

This chapter contains reference information on every STATA command. The commands are presented in alphabetical order. To improve readability, the information is presented in a standardized format to improve the readability of the reference information described below.

Syntax

The reference material begins with a complete syntax diagram of the command. Items in boldface should be typed exactly as they appear in the syntax diagram (subject to STATA's abbreviation rules, of course). Syntax diagrams employ the following symbols:

#	Indicates a literal number; e.g., 5
[]	Anything enclosed in brackets is optional
{}	At least one of the elements enclosed in braces must appear
|	The vertical bar separates alternatives
%fmt	Any STATA format, e.g., 8.2f
exp	Any STATA expression, e.g., (5+varname)/2
filename	A DOS filename (may include DOS path), e.g., b:myfile.dta
newvar	A variable that will be created by the current command
oldvar	A previously created STATA variable
options	A list of options
range	An observation range; e.g. 5/1
"string"	Any string of characters
varlist	A list of variable names
varname	A variable name
xvar	The variable to be displayed on the horizontal axis
yvar	The variable to be displayed on the vertical axis

Purpose

A brief description of the purpose of the command.

Remarks

This section contains any necessary amplifications or qualifications.

Output

If the command displays any output, it is described in this section.

Options

If the command takes options, they are explained here.

Example

For all but the most trivial commands, one or two examples are presented. These examples list a partial STATA session in which the command is used. Explanatory text accompanies the listing when necessary.

You may notice that the execution times of the STATA commands vary a great deal from example to example. This is because the examples were run on a variety of computers which are configured in different ways. If you want to make STATA run as fast as possible, we recommend that you install an 8087 Math Coprocessor. You might also configure a memory disk and place STATA on it, but be sure to leave at least 256K of memory available.

Reference

Many of the examples make use of common data sets described in Appendix A.

#command   command_arguments

Purpose

The #commands are the STATA pre-processor commands. See specific #command for details.

Remarks

#commands may be entered whenever STATA issues a period prompt, including during prompting of **input** and **modify**. #commands are instructions to the STATA pre-processor rather than to STATA itself, and they affect how STATA treats the terminal and line-input buffer.

#commands do not generate a return code, nor do they generate ordinary STATA errors. The only error message associated with #commands is "unrecognized #command".

#delimit {cr | ;}

Purpose

The **#delimit** command resets the character that marks the end of a command.

Remarks

STATA begins to process a command as soon as the delimiter is typed. When a STATA session begins, the delimiter is the carriage return, also called the Return key. It is sometimes convenient, particularly in do-files, to use a semicolon rather than a carriage return to delimit commands. The characters "**cr**" must be typed to reset the delimiter to the Return key.

Note that a semicolon appearing in a double-quoted string is not interpreted as a delimiter even if the semicolon is currently used as the delimiter. Also, whenever a new do-file is invoked, the delimiter is reset to **cr**. At the end of the do-file, the previous delimiter is restored automatically.

Example

The command

#delimit ;

causes STATA to treat the semicolon as the delimiter.

The command

```
#delimit cr
```

causes STATA to treat the carriage return as the delimiter.

#review [# [#]]

Purpose

The **#review** command displays the last few lines typed at the terminal.

Remarks

If no arguments follow **#review**, the last five lines typed at the terminal are displayed. The first argument specifies the the number of lines to be reviewed, so that **#review 10** displays the last ten lines typed. The second argument specifies the number of lines to be displayed, so that **#review 10 5** displays five lines, starting at the tenth previous line. The last line displayed by **#review** is left in the line input buffer and may be edited.

STATA reserves a buffer for **#review** lines and stores as many previous lines in the buffer as will fit, rolling out the oldest line to make room for the newest line. Requests to **#review** lines no longer stored will be ignored. Only lines typed at the terminal are placed in the **#review** buffer.

Example

```
. #review
  use hoel
  * comments go into the #review buffer too
  describe
  tabulate mar ed =number
  tabulate mar ed =number, chi2
. #review 2
  tabulate mar ed =number
  tabulate mar ed =number, chi2
. #review 2 1
  tabulate mar ed =number
```

---

append **using** filename

Purpose

The **append** command appends a STATA format data set stored on disk to the end of the data set currently in memory. If "filename" is specified without an extension, ".dta" or ".xp" is assumed (depending on whether the current data set is **data** or **xp**).

Remarks

The disk data set must be a STATA format data set, that is, it must have been created with the **save** command. If the disk file was encoded, the current encode key must be set appropriately (see **set encode** for details). The disk data set also must be of the same type (either **data** or **xp**) as the data set in memory. When **xp** data sets are appended, a new variable, called **_append**, is created. This variable can serve as a dummy variable to indicate a separate constant for the appended data set.

Technical Note: **xp** data sets are appended by name, that is, if there is a variable named x1 in both data sets, they will be joined (summed) regardless of their respective positions in the data set. If a variable exists only in one or the other data set the **xp** information will be copied across unchanged, thus effectively creating an interaction of that variable with a dummy variable reflecting the sample.

Options

nolabel prevents STATA from copying labels from the disk data set into the data set in memory. In no event do labels from the disk data set replace labels already in memory.

Example

In this example, a data set containing the sixth through eighth even numbers is appended to a data set containing the first five odd numbers.

```
. use even.dta
(6th through 8th even numbers)
R; T=4.99 13:54:23

. list

        number      even
  1.         6.      12.
  2.         7.      14.
  3.         8.      16.
R; T=2.14 13:54:27

. use odd.dta
(First five odd numbers)
R; T=5.21 13:54:42

. list

        number       odd
  1.         1.       1.
  2.         2.       3.
  3.         3.       5.
  4.         4.       7.
  5.         5.       9.
R; T=2.47 13:54:47
```

```
. append using even.dta
R; T=3.35 13:55:00

. list

       number        odd       even
  1.        1.        1.          .
  2.        2.        3.          .
  3.        3.        5.          .
  4.        4.        7.          .
  5.        5.        9.          .
  6.        6.         .        12.
  7.        7.         .        14.
  8.        8.         .        16.
R; T=3.13 13:55:07
```

In the next example, **xp** data sets are appended.

```
. use oddeven.dta
R; T=6.81 16:14:10

. list

          odd       even
  1.        1.        2.
  2.        3.        4.
  3.        5.        6.
  4.        7.        8.
  5.        9.       10.
R; T=4.06 16:14:16

. convert
(obs=5)
Varname | Mean  Std. Dev.   Min.   Max.
--------+-------------------------------
     odd| 5.    3.162278     1.      9.
    even| 6.    3.162278     2.     10.

R; T=10.44 16:14:30
```

```
. describe
Contains crossproduct (xp)
Vars:     3 (max=  100)
  1. _cons          double %10.0g
  2. odd            double %9.0g
  3. even           double %9.0g
Note:  Data has changed since last save
R; T=2.75 16:15:02

. list

            _cons          odd          even
  1.           5.          25.           30.
  2.          25.         165.          190.
  3.          30.         190.          220.
R; T=3.57 16:15:07

. save oddeven.xp
File oddeven.xp saved
R; T=13.29 16:15:26

. use moredata.dta, clear
R; T=5.99 16:15:36

. list

             odd        even
  1.         11.         12.
  2.         13.         14.
  3.         15.         16.
R; T=3.79 16:15:43

. convert
(obs=3)
Varname | Mean   Std. Dev.  Min.    Max.
--------+---------------------------------
     odd| 13.         2.     11.      15.
    even| 14.         2.     12.      16.

R; T=10.65 16:15:55
```

```
. describe
Contains crossproduct (xp)
Vars:     3 (max=  100)
  1. _cons             double %10.0g
  2. odd               double %9.0g
  3. even              double %9.0g
Note:  Data has changed since last save
R; T=3.62 16:16:01

. list

             _cons          odd          even
  1.            3.          39.           42.
  2.           39.         515.          554.
  3.           42.         554.          596.
R; T=3.95 16:16:05

. append using oddeven.xp
R; T=6.26 16:16:29

. describe
Contains crossproduct (xp)
Vars:     4 (max=  100)
  1. _cons             double %10.0g
  2. odd               double %9.0g
  3. even              double %9.0g
  4. _append           double %10.0g
Note:  Data has changed since last save
R; T=3.35 16:16:34

. list

        _cons     odd    even _append
  1.       8.     64.     72.       5.
  2.      64.    680.    744.      25.
  3.      72.    744.    816.      30.
  4.       5.     25.     30.       5.
R; T=4.50 16:16:39
```

---

beep

Purpose

The **beep** command causes the computer to emit a single beep.

Remarks

This command can be used in do-files to signal that a time-consuming task is completed.

Example

```
. beep
R; T=2.14 14:57:41
```

(The computer emits a single beep)

---

by varlist: STATA_command

Purpose

The **by** prefix causes a STATA command to be repeated for each unique set of values of the variables in varlist. The data set must already be **sort**ed by the varlist.

Remarks

by is an optional prefix to perform STATA_command for each group of observations for which the values of variables in the "varlist" are the same. During each iteration, the values of the system variables **_n** and **_N** are set relative to the first observation in the by-group. The "**in** range" modifier cannot be used in conjunction with "**by** varlist:" because ranges specify absolute rather than relative observation numbers.

The results of STATA_command will be the same as if you formed separate data sets for each group of observations, **save**d them, **use**d each separately, and issued STATA_command.

by may not be used with **xp** data sets.

Example

```
. use hoel.dta
R; T=5.17 19:13:32
```

```
. sort marriage
R; T=1.04 19:13:36

. describe
Contains data
 Obs:      12 (max=  609)
Vars:       3 (max=  100)
  1. marriage       float  %9.0g  marlbl
  2. educ           float  %9.0g  edlbl
  3. number         float  %9.0g
Sorted by:  marriage
Note:  Data has changed since last save
R; T=0.99 19:13:38

. list, nolabel

      marriage        educ      number
  1.         1.         2.         17.
  2.         1.         3.         11.
  3.         1.         1.         18.
  4.         2.         3.         10.
  5.         2.         2.         28.
  6.         2.         1.         29.
  7.         3.         1.         70.
  8.         3.         3.         11.
  9.         3.         2.         30.
 10.         4.         3.         20.
 11.         4.         2.         41.
 12.         4.         1.        115.
R; T=3.73 19:13:45

. by marriage: summarize number

-> marriage= Very Low
 varname| Obs       Mean     Std. Dev.   Min.    Max.
--------+-----------------------------------------------
  number|    3  15.333333  3.7859389    11.     18.
```

```
by

-> marriage=      Low
 varname| Obs       Mean     Std. Dev.  Min.   Max.
--------+-----------------------------------------
  number|   3  22.333333  10.6926766  10.    29.

-> marriage=     High
 varname| Obs       Mean     Std. Dev.  Min.   Max.
--------+-----------------------------------------
  number|   3        37.  30.1164407  11.    70.

-> marriage= Vry High
 varname| Obs       Mean     Std. Dev.  Min.   Max.
--------+-----------------------------------------
  number|   3  58.666667  49.9032397  20.   115.

R; T=10.98 19:14:11
```

capture STATA_command

Purpose

The **capture** command executes STATA_command and issues a return code of zero. The actual return code generated by STATA_command is stored in the system variable **_rc**.

Remarks

The **capture** command is useful in do-files where any non-zero return code automatically terminates the do-file. By preceding sensitive commands with the word **capture**, the do-file can respond appropriately to any situation by conditioning the remaining actions on the value of **_rc**.

Example

```
. drop _all
R; T=0.16 09:05:49

. list myvar
No variables defined
R(111); T=0.22 09:06:33

. capture list myvar
No variables defined
R; T=0.11 09:07:06

. display =_rc
      111.
R; T=0.28 09:07:39
```

---

confirm **existence** [string]
variable [varlist]
newvariable [varlist]

Purpose

The **confirm** command checks on the existence of a string or the status of varlist. The **confirm existence** command issues a return code of zero if the string is non-null and a non-zero return code if the string is null. The **confirm variable** command issues a return code of zero if the variables in the varlist already exist in the current data set and a non-zero error code if any of the variables do not already exist. The **confirm newvariable** command issues a return code of zero if none of the variables in the varlist already exists in the current data set and all the names in the variable list are legal STATA variable names. If either of these conditions fails, the command returns a non-zero return code. Note that the **confirm newvariable** command ignores the STATA rules for abbreviating variable names. A variable is considered to be new unless its name exactly matches the complete name of an existing variable.

Remarks

The **confirm** command is most useful in do-files, particularly in conjunction with the **capture** command. It can be used, for example, to insure that necessary macros are non-null before commands using the macros are given.

Example

```
. macro define not_null "This macro is not null"
R; T=2.47 14:56:37

. macro define null ""
R; T=1.48 14:56:46

. macro list
null:
not_null:  This macro is not null
R; T=1.86 14:56:51

. confirm existence %not_null
R; T=1.54 14:57:04

. confirm existence %null
R(6); T=0.05 14:57:10

. confirm existence null
R; T=0.05 14:57:15

. confirm existence
R(6); T=0.05 14:57:18
```

---

convert [varlist] [**in** range]
[**if** exp] [=exp]

Purpose

The **convert** command converts a standard (**data**) STATA format data set into a cross product (**xp**) STATA format data set. **convert** may not be abbreviated.

Remarks

This command converts the data set in memory into an **xp** data set. The original data set in memory is destroyed. A system variable called **_cons** is created. This new variable forms the first column (and row) of the cross product matrix. Its first element is the weighted number of observations. The remaining elements are the weighted sums of the other variables in the matrix. All columns in the cross product matrix have type **double** regardless of the original types of the variables comprising the data.

The **convert** command is useful for reducing large data sets to a manageable size and for increasing the speed of the **correlate**, **summarize**, and **regress** commands. STATA **xp** data sets can be appended, allowing regressions to be run on data sets that would normally be too large to fit in the available memory.

STATA will not permit the conversion of a data set whose contents have been changed during the current STATA session unless the **clear** option is specified.

If "=exp" appears, then the expression is used to weight the cross product matrix. The weight defined by the expression is normalized to sum to the number of non-missing observations. Each observation is multiplied by the square root of the normalized weight before being added to the cross product matrix. (Note: the technique used for weighting does not require the calculation of the square roots, hence, the numerical inaccuracy inherent in taking square roots is avoided.)

Some STATA commands are not allowed when the data set in memory is an **xp** data set. These are **by**, **convert**, **count**, **expand**, **if**, **in**, **input**, **merge**, **modify**, **plot**, **replace**, **sort**, and **tabulate**. There are other restrictions as well. Only linear functions of the existing variables can be **generated**. Also, **list** displays the cross product matrix, not the original values of the variables.

convert automatically repartitions memory so the cross-product matrix will fit (for details see Appendix B, Memory Management in STATA). It is not necessary to understand the details if you remember two rules of thumb. After using **convert**, if you are going to use a raw data set, first type:

```
drop _all
set maxvar 99
```

If you are going to **use** a cross-product data set for the first time during a STATA session or after applying the first rule of thumb, type:

```
drop _all
set maxvar 100 lrecl 800
```

Even if you do not follow these rules it will probably not matter; but following them will avoid error messages when you are dealing with large data sets. If you choose to ignore them then you may receive an error message telling you that there is insufficient space to perform your request, and you will have to type these suggested statements then.

Output

Summary statistics are displayed for all the variables in the cross product matrix. This display is identical to that produced by the **summarize** command.

Options

clear allows the current data set to be converted even if the data set has been changed during the current STATA session.

noformat displays the summary statistics in **g** format regardless of the display format previously specified. The formats used differ across variable types as follows:

int	**%8.0g**
long	**%10.0g**
float	**%9.0g**
double	**%10.0g**

nomeans suppresses the display of the summary statistics.

noscale prevents the normalization of the weight.

Example

```
. use odd.dta
R; T=4.89 14:21:56

. convert, nomeans
(obs=5)
R; T=5.49 14:22:23
```

```
. describe
Contains crossproduct (xp)
Vars:     3 (max=  100)
  1. _cons         double %10.0g
  2. number        double %9.0g
  3. odd           double %9.0g
Note:  Data has changed since last save
R; T=1.86 14:22:30

. list

          _cons     number        odd
  1.         5.        15.        25.
  2.        15.        55.        95.
  3.        25.        95.       165.
R; T=2.75 14:22:36
```

[**by** varlist:] **correlate** [varlist]
[=exp] [**in** range] [**if** exp]

Purpose

The **correlate** command calculates and displays the correlation matrix for a group of variables. If no varlist is given, **correlate** calculates the correlation matrix for all the variables in the data set.

Remarks

If =exp appears, then the expression is used to weight the correlation matrix. The weight defined by the expression is normalized to sum to the number of non-missing observations. Each observation is effectively multiplied by the square root of the normalized weight before the correlations are calculated.

Output

The correlation matrix is displayed as a lower triangular matrix. Optionally, summary statistics for each of the variables can be displayed.

Options

means causes summary statistics to be calculated and displayed for each variable in the correlation matrix.

noformat displays the summary statistics in g format regardless of the display format previously specified. The formats used differ across variable types as follows:

int	**%8.0g**
long	**%10.0g**
float	**%9.0g**
double	**%10.0g**

noscale suppresses the normalization of the weight.

covariance displays the covariances rather than correlation coefficients.

_coef displays the correlations between the coefficients of the last regression.

Example

```
. drop _all
R; T=0.60 14:46:15

. set obs 5
obs was 0, now 5
R; T=0.11 14:46:16

. generate number = _n - 3
R; T=2.74 14:46:19

. generate square = number * number
R; T=1.65 14:46:23

. generate newvar = 5*number - square/3
R; T=1.93 14:46:27
```

```
. list

          number      square      newvar
  1.         -2.          4.   -11.33333
  2.         -1.          1.   -5.333333
  3.          0.          0.          0.
  4.          1.          1.    4.666667
  5.          2.          4.    8.666667
R; T=5.33 14:46:35

. correlate
(obs=5)
        |   number    square    newvar
--------+------------------------------
  number|   1.0000
  square|   0.0000    1.0000
  newvar|   0.9969   -0.0786    1.0000
R; T=6.70 14:46:45
```

[by varlist:]   **count**   **[in range] [if exp]**

Purpose

The **count** command counts the number of observations that satisfy the specified conditions. If no conditions are specified, **count** displays the number of observations in the data set.

Example

```
. use hoel.dta
(Data for Hoel's textbook)
R; T=2.91 20:26:37

. count
   12
R; T=0.66 20:26:45

. by educ: count if number>30

-> educ=  College      2
-> educ=     H.S.      1
-> educ=   < H.S.      0
R; T=1.48 20:26:47
```

---

describe [{varlist | **using** filename}]

Purpose

The **describe** command produces a summary of the contents of a STATA format data set.

Remarks

If **describe** is typed without any operands, then the contents of the current STATA data set are summarized. This summary includes some general information and a description of every variable in the data set. If a varlist is specified, the general information is omitted, and only the variables in the varlist are described. If "**using** filename" is specified, then the contents of the disk data set "filename" are summarized. "filename" must be in STATA format, that is, it must have been created by the **save** command. If "filename" is specified without an extension, ".dta" is assumed.

Output

In the first line of the display, the data set type (**data** or **xp**) is indicated. If the data set is labeled, the label is also displayed on the first line. Next, the current and maximum allowed numbers of observations and variables are listed. A description of the variables follows.

For each variable, the name is listed followed by its type (**int**, **long**, **float**, or **double**) and display format. If the variable is associated with a set of value labels, the name of the value label list is displayed after the display format. (In the example below, the variables "marriage" and "educ" are associated with the value label lists "marlbl" and "edlbl", respectively.) The variable label, if any, is displayed on the far right.

Two additional general items are displayed. The sort order is listed, and, if the data set has changed since it was **used**, this fact is noted.

Options

short causes STATA to suppress the information on each variable. Only the general information is displayed.

Example

```
. use hoel.dta
(Data from Hoel´s textbook)
R; T=0.55 13:04:19

. describe
Contains data                                     Data from Hoel´s textbook
 Obs:     12 (max=  151)
Vars:      3 (max=  100)
  1. marriage     float  %9.0g     marlbl   Marriage-adjustment score
  2. educ         float  %9.0g     edlbl    Level of education
  3. number       float  %9.0g              Number of cases
Sorted by:  educ  marriage
R; T=1.10 13:04:21

. describe marriage number
  1. marriage     float  %9.0g     marlbl   Marriage-adjustment score
  3. number       float  %9.0g              Number of cases
R; T=0.44 13:04:21
```

```
. convert
(obs=12)
Varname |         Mean    Std. Dev.         Min.         Max.
--------+-----------------------------------------------------
marriage|          2.5     1.167748           1.           4.
    educ|           2.     .8528029           1.           3.
  number|     33.33333     30.67819          10.         115.

R; T=2.96 13:04:24

. describe
Contains crossproduct (xp)                     Data from Hoel´s textbook
Vars:     4 (max=  100)
  1. _cons          double %10.0g
  2. marriage       double %9.0g               Marriage-adjustment score
  3. educ           double %9.0g               Level of education
  4. number         double %9.0g               Number of cases
Note:  Data has changed since last save
R; T=1.15 13:04:26
```

```
. drop _all
R; T=0.11 13:04:26

. describe
Contains data
 Obs:      0 (max=  151)
Vars:      0 (max=  100)
Sorted by:
R; T=0.55 13:04:27

. describe using hoel.dta
Contains data                                        Data from Hoel´s textbook
Obs:    12  Var:   3
  1. marriage     float  %9.0g    marlbl   Marriage-adjustment score
  2. educ         float  %9.0g    edlbl    Level of education
  3. number       float  %9.0g             Number of cases
Sorted by: educ marriage
R; T=1.16 13:04:28
```

---

dir DOS_file_specification

Purpose

The **dir** command is the STATA equivalent of the DOS DIR command. It displays the names of the files in a directory.

Remarks

There are two differences between the STATA and the DOS **dir** commands. First, the DOS /P option is unnecessary; STATA always pauses when the screen is full. Second, you must include a file specification when using the **dir** command. To see all the files in a directory, give the command

```
dir *.*
```

Output

The output of the STATA **dir** command is roughly the same as that produced by the DOS DIR command.

Options

wide produces the same effect as specifying /W with the DOS DIR command: four filenames, rather than one, are displayed on each line.

dir

Example

```
. dir *.doc

RCORREL.DOC     2.5k   8/10/84 18:35
RAPPEND.DOC     1.8k   8/10/84 17:56
RDELIM.DOC      0.9k   8/10/84 17:54
RBEEP.DOC       0.5k   8/10/84 17:59
RCONVERT.DOC    4.4k   8/10/84 18:34
RBY.DOC         3.0k   8/10/84 19:19
RIF.DOC         1.8k   8/10/84 19:37
RIN.DOC         1.7k   8/10/84 19:37
RCOUNT.DOC      1.6k   8/10/84 20:29
RDES.DOC        3.9k   8/12/84 13:25
R; T=1.82 13:33:47
```

---

discard

Purpose

The **discard** command eliminates information stored by the most recent **regress** command. It makes more space available for adding data to the current data set. Note that the **test** command and the **regress** command without operands will not work after a **discard.** In addition, the contents of **_pred** and **_coef** are destroyed by the **discard** command.

Remarks

You will rarely, if ever, need this command. Use **discard** after the message "Insufficient memory" if you are willing to part with previous regression results.

---

display [[=exp | "string"]
[, [=exp | "string"] [, ...]]] [**if** exp]

Purpose

The **display** command displays strings and/or the values of scalar expressions.

Remarks

This command combines the functions of an electronic calculator with the ability to label results. STATA variables may appear in the expression. If these variables are not explicitly subscripted, their values for the first observation are used.

Output

The value of any expression is displayed as a floating point number.

Example

```
. display = 5
        5.
R; T=0.83 14:09:00

. display = 5/3
 1.6666667
R; T=0.16 14:09:06
```

```
. display = int(5/3)
        1.
R; T=0.16 14:09:15

. display = sqrt(2)
 1.4142136
R; T=1.27 14:09:27

. use odd.dta
R; T=4.89 14:08:38

. display = odd[4] - number[4]
        3.
R; T=0.83 14:09:58

. display = odd
        1.
R; T=0.11 14:10:05

. display "2+2 is ", =2+2
2+2 is         4
R; T=0.44 14:11:01
```

---

do filename [parameter_list]

Purpose

The **do** command causes a disk file containing STATA commands to be executed. The commands in "filename" are executed as though they were currently being typed. The disk file is called the "do-file". If no extension is specified with the filename, ".do" is assumed.

Remarks

The name of the disk file may be followed by up to 20 parameters. In the do-file, these parameters can be referenced by the macro names _1 through _20, where _n refers to the n-th parameter in the typed list. These macro names replace the current _1 through _20, if they are defined. The current _1 through _20 are restored when the execution of the do-file is completed. In fact, all macros that begin with the underscore character (_) are local to a do-file, that is, their previous definitions are ignored during the execution of the do-file. At the termination of the do-file, these previous definitions are all restored.

A do-file completes execution when (1) the end of the file is reached, (2) an **exit** is executed, or (3) when an error (non-zero return code) occurs. In the last case, the remaining commands in the do-file are abandoned. If STATA senses a Ctrl-Break typed at the keyboard it responds as though an error occurred. Thus, do-files can be interrupted.

Do-files operate on the current data set. If a do-file terminates as the result of an error, you should **describe** the current data set to be sure that the do-file did not "leave behind" any temporary variables.

A do-file can contain **do** commands; that is, do-files can be nested. STATA will not allow do-files to be nested more than five deep.

Output

The commands inside the do-file produce their usual output. At the completion of a do-file, the message "end of do file" and a return message are displayed. The time reported in this return message is the execution time for the complete do-file.

Options

delay causes the commands in the do-file to be displayed at a slower speed than normal. This allows them to be read before they are executed. The output produced by each command is displayed at its normal speed.

nostop causes STATA to ignore non-zero return codes during the execution of a do-file; that is, a non-zero return code will not terminate the execution of a do-file if **nostop** is specified.

Example

```
. type load.do
 use hoel.dta
 count if number>18

R; T=0.66 14:17:11
```

```
. do load

. use hoel.dta
(Data from Hoel´s textbook)
R; T=0.55 14:17:14

. count if number>18
    7
R; T=0.72 13:01:28

.
end of do file
R; T=2.10 14:17:18
```

---

drop varlist

[**by** varlist:] **drop** **in** range [**if** exp]

[**by** varlist:] **drop** **if** exp [**in** range]

Purpose

The **drop** command eliminates variables or observations from the current data set. **drop** may not be abbreviated.

Remarks

The entire data set can be cleared by typing **drop _all**. Value labels and macros are unaffected by the **drop** command. (See **label drop _all** and **macro drop _all**.)

Example

```
. use hoel.dta
(Data from Hoel´s textbook)
R; T=0.55 14:45:11

. drop marriage
R; T=0.22 14:45:16

. drop in 3/7
(5 observations deleted)
R; T=0.17 14:45:18
```

```
. list

          educ     number
  1.   College        18.
  2.   College        29.
  3.      H.S.        41.
  4.    < H.S.        11.
  5.    < H.S.        10.
  6.    < H.S.        11.
  7.    < H.S.        20.
R; T=1.43 14:45:19

. drop _all
R; T=0.17 14:45:20

. describe
Contains data
 Obs:     0 (max=  605)
Vars:     0 (max=  100)
Sorted by:
R; T=0.55 14:45:21
```

---

exit [**if** exp]

Purpose

The **exit** command causes STATA to terminate processing and return control to DOS. If the data set in memory has changed since the last **save** command, you must specify the **clear** option before STATA will allow you to **exit**.

exit is also used to stop a do-file and return control to the caller whether that be a previous do-file or the keyboard.

Options

clear permits you to **exit** even if the current data set has not been **saved**.

Example

```
. exit
No - data in memory would be lost
R(4); T=0.39 10:19:21

. exit, clear

A>
```

---

expand =exp [**in** range] [**if** exp]

Purpose

The **expand** command replaces each observation in the current data set with n copies of the observation, where "n" is equal to the integer part of the required expression. If the expression is less than one (1.0) or equal to missing value, then it is interpreted as if it were one and the observation is retained but not duplicated.

Remarks

If there is not enough memory available to expand the data set, a warning message is displayed and the data set remains in its original form.

Output

The number of observations added to the data set is displayed.

Options

clear permits the data set to be **expand**ed even if the data set has changed during the current STATA session.

Example

```
. use hoel.dta
R; T=6.76 15:58:48

. tabulate educ = number
       educ|  Freq.  Percent   Cum.
-----------+-----------------------
   College |   232    58.00   58.00
      H.S. |   116    29.00   87.00
    < H.S. |    52    13.00  100.00
-----------+-----------------------
      Total|   400   100.00
R; T=2.36 15:59:18

. expand = number
(388 observations created)
R; T=5.38 15:59:30

. describe
Contains data
 Obs:    400 (max=  721)
Vars:      3 (max=   99)
  1. marriage      float  %9.0g    marlbl
  2. educ          float  %9.0g    edlbl
  3. number        float  %9.0g
Sorted by:  educ  marriage
Note:  Data has changed since last save
R; T=2.47 15:59:35

. tabulate educ
       educ|  Freq.  Percent   Cum.
-----------+-----------------------
   College |   232    58.00   58.00
      H.S. |   116    29.00   87.00
    < H.S. |    52    13.00  100.00
-----------+-----------------------
      Total|   400   100.00
R; T=2.81 15:59:52
```

---

format varlist %fmt

Purpose

The **format** command allows you to specify the display format of variables in the current data set. The internal precision of the variables is unaffected.

Remarks

Formats are denoted by a leading percent sign (%) followed by the string "#.#", where # stands for an integer. The first integer specifies the width of the format. The second integer, which must be less than or equal to the first, specifies the number of digits that are to follow the decimal point. A character denoting the format type (**e**, **f**, or **g**) is then listed. As an example, the string **%9.2f** specifies an **f** format that is nine characters wide and has two digits following the decimal point.

By default, every variable is given a **%#.0g** format, where # is large enough to display the largest number of the variable′s type. The **g** format is really a complicated set of formatting rules that attempts to present values in as readable a fashion as possible without sacrificing precision. The **g** format changes the number of decimal places displayed whenever it improves the readability of the current value. The number after the decimal point specifies the minimum number of digits that are to follow the decimal point. For instance, the number 1.1 would be displayed as "1.1" in **%9.0g and %9.1g**, and as "1.10" in **%9.2g**.

Under the STATA **f** format, values are always displayed with the same number of decimal places, even if this causes a loss in (the displayed) precision. Thus, the **f** format is similar to the FORTRAN F format. The only difference is that the width of the STATA **f** format is temporarily increased whenever a number is too large to be displayed in the specified format.

The **e** format is similar to the FORTRAN E format. Every value is displayed as a leading digit (with a minus sign, if necessary) followed by a decimal point, the specified number of digits, the letter "E", a plus or minus sign, and the power of ten (modified by the preceding sign) that multiplies the displayed value. When the **e** format is specified, the width must exceed the number of digits that follow the decimal point by at least seven. This space is needed to accommodate the leading sign and digit, the decimal point, the "E", and the signed power of ten.

Example

```
. describe
Contains data
 Obs:      0 (max=  605)
Vars:      0 (max=  100)
Sorted by:
R; T=0.49 14:56:58

. set obs 5
obs was 0, now 5
R; T=0.11 14:56:58

. generate e_fmt=sqrt(7.85*_n)*(1+(999999*
             (_n==2))+(-1.000001*(_n==4)))
R; T=1.21 14:57:00
```

format

```
. generate f_fmt = e_fmt
R; T=0.16 14:57:01

. generate g_fmt = e_fmt
R; T=0.17 14:57:01

. format e_fmt %9.2e
R; T=0.05 14:57:01

. format f_fmt %9.2f
R; T=0.11 14:57:02

. describe
Contains data
 Obs:      5 (max=  605)
Vars:      3 (max=  100)
  1. e_fmt         float  %9.2e
  2. f_fmt         float  %9.2f
  3. g_fmt         float  %9.0g
Sorted by:
Note:  Data has changed since last save
R; T=0.99 14:57:03

. list

         e_fmt       f_fmt       g_fmt
  1.  2.80E+00        2.80    2.801785
  2.  3.96E+06  3962322.50    3962323.
  3.  4.85E+00        4.85    4.852834
  4. -5.60E-06       -0.00   -5.60E-06
  5.  6.26E+00        6.26    6.264982
R; T=1.54 14:57:05
```

```
[by varlist:]   generate   newvar = exp
          [in range] [if exp]
```

Purpose

The **generate** command creates a new STATA variable. The values of the variable are specified by "=exp".

Remarks

The **generate** command cannot be used to change the values of an existing variable. Use the **replace** command for this purpose.

Note that "newvar" can be specified as

```
[type] new_varname[:label_name]
```

where "type" is **int**, **long**, **float**, or **double**. If no type is specified, then **float** is assumed (or the type specified in the **set type** command). The optional ":label_name" associates a value label with the new variable. This value label need not be defined at the time the **generate** command is given.

Output

If any missing values are generated, the number generated are displayed. If no message is presented, then no missing values were produced.

generate

Example

```
. describe
Contains data
 Obs:      0 (max=  605)
Vars:      0 (max=  100)
Sorted by:
R; T=0.44 15:39:04

. set obs 5
obs was 0, now 5
R; T=0.17 15:39:04

. generate var1 = _n
R; T=0.22 15:39:04

. generate var2 = 15.88*sqrt(var1)/_N
R; T=0.71 15:39:05

. list

          var1        var2
  1.        1.       3.176
  2.        2.    4.491542
  3.        3.    5.500993
  4.        4.       6.352
  5.        5.    7.101752
R; T=1.15 15:39:07
```

---

help [command name]

Purpose

The **help** command displays help information on the specified command or topic. If **help** is not followed by a command or topic name, the list of topics for which help is available is listed.

Remarks

Unless you have previously given the command **set help** filename, the file STATA.HLP must be in the current directory or the **help** command will not work. Users short on disk space may delete STATA.HLP.

Output

The help information starts with a syntax diagram of the specified command. If the command takes any options, they are listed. A brief description of the purpose of the command is also displayed.

STATA_command **if** exp

Purpose

The **if** exp qualifier restricts the scope of a STATA command to those observations for which the value of "exp" is non-zero. "exp" may be any STATA expression.

Remarks

if may not be used with **xp** data sets.

Example

```
. use hoel.dta
(Data from Hoel´s textbook)
R; T=5.16 19:27:08

. list number if marriage==educ

        number
  1.       18.
  6.       28.
 11.       11.
R; T=1.64 19:27:23

. list if number>40, nolabel

      marriage        educ      number
  3.        3.          1.         70.
  4.        4.          1.        115.
  8.        4.          2.         41.
R; T=0.88 19:27:26
```

STATA_command **in** range

Purpose

The **in** range qualifier restricts the scope of a STATA command to the observations specified by "range".

Remarks

A STATA range specification takes the form #1[/#2] where #1 and #2 are integers such that #1 is less than or equal to #2. The first and last observations in the data set may be denoted by **f** and **l** (letter ell), respectively. A range specifies absolute observation numbers within a data set. As a result, the **in** modifier cannot be used when a command is preceded by the "**by** varlist:" prefix.

in may not be used with **xp** data sets.

Example

```
. use odd.dta
R; T=5.00 19:33:34
```

in

```
. list in f/l

          number          odd
  1.          1.           1.
  2.          2.           3.
  3.          3.           5.
  4.          4.           7.
  5.          5.           9.
R; T=1.43 19:33:45

. list in 3

          number          odd
  3.          3.           5.
R; T=0.93 19:33:50

. summarize in 3/l

 varname|  Obs   Mean   Std. Dev.   Min.     Max.
--------+-------------------------------------------
  number|   3     4.        1.       3        5.
     odd|   3     7.        2.       5.       9.
R; T=2.26 19:33:57
```

---

infile varlist
[**_skip**[(#)] [varlist [**_skip**[(#)] ...]]]
using filename [**in** range] [**if** exp]

Purpose

The **infile** command reads into memory a disk data set that is not in STATA format. The data can then be saved as a STATA format data set. The original disk data set is unchanged. If "filename" is specified without an extension, ".raw" is assumed.

Remarks

There must not be a data set already in memory when the **infile** command is given.

The non-STATA disk data set must either be in free format or comma-separated-value format. In free format, data are separated by one or more "white space" characters. White space characters are blanks, tabs, or "newlines" (carriage return/line feed combinations). Missing values are indicated by single periods (.). In comma-separated-value format, data are separated by commas. You may intermix comma-separated-value and free format. Missing values may also be indicated by multiple commas which serve as place holders. In either format, a single observation can span any number of input lines. String variables may be enclosed in single or double quotes. If the string contains imbedded white space or any characters besides letters, digits, and underscore, it must be enclosed in one or other type of quotes.

infile

All the variables specified in the **infile** command are new variables, that is, they are created by the **infile** command. The syntax for a new variable is

```
[type] new_varname[:label_name]
```

By default, variables created by **infile** are of type **float**. This default can be overridden by preceding the variable name with a type name (**int**, **long**, **float**, or **double**), or by the **set type** command.

A list of variables placed in parentheses will be given the same type. For example,

```
double (first_var second_var ... last_var)
```

causes "first_var second_var ... last_var" to all be type **double**.

There is also a shorthand syntax for variable names with numeric suffixes. For example, the varlist

```
var1-var4
```

is equivalent to specifying

```
var1 var2 var3 var4
```

The **infile** command can handle non-numeric data in several ways. If the non-numeric data are unexpected, then a warning message is issued and the variable is set to missing value for that observation. **infile** can be directed to expect non-numeric data by typing a colon (:) and the name of a value label after the variable name and optionally including the **automatic** option. (See the description of the **label** command for an explanation of value labels.) For example, the command

```
infile varname:lblname using diskfile
```

causes **infile** to assign values to varname based on the value labels stored under the name "lblname". (If some or all of the observations contain numerical values, they are stored in the usual way.) The label modifier may be combined with the range notation and the type modifier, so that

```
**long** var1-var4:lblname
```

is equivalent to specifying

```
**long** var1:lblname ... **long** var4:lblname
```

Value labels can also be created by **infile.** The command

```
infile varname:lblname using diskfile, automatic
```

causes **infile** to assign an integer to each unique string it reads. The resulting value label is stored under the name "lblname".

Specifying **_skip** as a variable name directs STATA to ignore the variable in that location, that is, it is not added to the data set being created in memory. This feature makes it possible to extract manageable subsets from large disk data sets. A number of contiguous variables can be skipped by specifying **_skip(#)** as a variable name, where # is the number of variables to ignore.

Subsets of observations can be extracted by specifying an "**if** exp". It is important to remember that the system variables **_n** and **_N** refer to the observation number and sample size, respectively, of the data set in memory - not of the disk data set. Use the "**in** range" modifier to refer to observation numbers within the disk data set.

Options

automatic causes STATA to create value labels from the non-numeric data it reads.

byvariable(#) specifies that the external data file is organized by variables rather than by observations. In other words, all the observations on the first variable appear, then all the observations on the second variable, and so on. **infile** needs to know the number of observations in order to read the data properly. You specify the number in the parentheses following **byvariable**. Alternatively, you can mark the end of one variable's data and the beginning of another's by placing a semicolon (;) in the raw data file. You may then specify a number larger than the number of observations in the data set and leave it to **infile** to figure out how many observations there really are. This method can also be used to read unbalanced data.

Example

```
. type oddeven.raw
1  2
3  4
5  6
7  8
9 10

R; T=0.66 11:45:32

. infile odd even using oddeven.raw
(5 observations read)
R; T=0.76 11:45:33
```

```
. describe
Contains data
 Obs:      5 (max=  605)
Vars:      2 (max=  100)
  1. odd             float  %9.0g
  2. even            float  %9.0g
Sorted by:
Note:  Data has changed since last save
R; T=0.99 11:45:34

. list

            odd       even
  1.         1.         2.
  2.         3.         4.
  3.         5.         6.
  4.         7.         8.
  5.         9.        10.
R; T=1.15 11:45:36
```

---

input [varlist]

Purpose

The **input** command allows you to type data directly into the data set in memory.

Remarks

After the **input** command is entered, STATA lists the variable names in the order in which they are to be entered. Then STATA prompts you with observation numbers. You must respond by typing a list of values, one for each variable. Missing values may be indicated with a period (.). You may terminate data entry at any time by typing **end** in response to the observation number prompt.

If there are no data in memory when you enter the **input** command, or if you type **input** without a varlist, STATA will prompt you for new observations until you type **end**. If you are adding a new variable or variables to the current data set, STATA will automatically terminate data entry when you have entered values for as many observations as are in the current data set.

STATA has a very flexible syntax for specifying new variables and lists of new variables. See the description of the **infile** command for a complete explanation of this syntax.

By default, variables created by **input** are of type **float** (or of the type specified by **set type** if you have changed it). This default can be overridden by preceding the variable name (or bound list of variable names) with a type name (**int**, **long**, **float**, or **double**).

If a variable name is followed by a colon (:) and the name of a value label, the value label is associated with the variable. (This feature is explained in detail in the description of the **infile** command.) If the **label** option is specified, the value labels may be typed instead of the values. If the **automatic** option is specified, then value labels are created as the data are typed.

Options

automatic causes STATA to create value labels as non-numeric data are typed.

label allows you to type the value labels instead of the values for variables associated with a value label name.

Example

```
. drop _all
R; T=0.55 12:31:14

. input number odd

        number        odd
  1. 1 1
  2. 2 3
  3. 3 5
  4. end
R; T=14.77 12:31:36
```

```
. input even

          even
  1. 2
  2. 4
  3. 6
R; T=4.23 12:31:42

. input

        number        odd       even
  4. 4 7 8
  5. 5 9 10
  6. end
R; T=10.17 12:31:55

. list

        number        odd       even
  1.         1.        1.         2.
  2.         2.        3.         4.
  3.         3.        5.         6.
  4.         4.        7.         8.
  5.         5.        9.        10.
R; T=1.37 12:31:57
```

```
                label   label_command

 "label_command" can be any of the following:

   data    "label"
   define    label_name # "label" [# "label" ... ]
   dir
   drop    list of label_names
   list    list of label_names
   save    list of label_names using filename
   values    varname label_name
   variable    varname "label"
```

Purpose

The **label** command is a collection of functions that define, list, associate, and drop labels for data sets, variables, and the values of variables.

Remarks

There are three kinds of STATA labels: data labels, variable labels, and value labels. Data labels are 32 character (maximum) labels that are assigned to STATA data sets. They are displayed whenever a labeled data set is **used** or **described**. To assign a label to a data set, enter the command

```
label data "label"
```

Note that only labels with imbedded "white space" or non-alphabetic and numeric characters (such as "+") need to be enclosed with double quotes. The white space characters are blanks, tabs, or "newlines" (carriage return/line feed combinations).

Variable labels are 32 character (maximum) labels that are associated with particular variables. These labels are displayed whenever the variable is **described** and are used by various other commands to label output. To assign a label to a variable, enter the command

label variable varname "label"

A value label is a list of up to 255 labels each of which is associated with a numeric value, which must be an integer. To define a value label, enter the command

label define label_name # "label" [# "label" ...]

Value label names follow the same naming conventions as STATA variable names.

Value labels have no effect until they are associated with a variable or variables. To associate a value label with a variable, enter the command

label values varname label_name

(Although only integer values may be labeled, the variable need not be stored as an **int**.) Once a value label is associated with a variable, the labels are displayed instead of the numeric values in all STATA output. If an observation contains a value for which no label is defined, then the value is displayed. The same list of value labels may be associated with more than one variable. If a list of value labels is associated with a variable, then the label_name appears just before the variable label in the **describe** output.

The label_names of all currently defined value labels can be displayed by entering the command

label dir

The contents of value label lists (the numeric codes and associated labels) can be displayed by entering the command

label list list of label_names

Value labels that are no longer needed can be eliminated by entering the command

label drop list of label_names

If the name **_all** appears in place of a list of label names, then the **label list** or **label drop** command operates on all the value labels that are currently defined.

Labels are automatically stored with your data set when you **save** it. Conversely, the **use** command drops all labels before it loads a new data set. You may occasionally wish to move a value label from one data set to another. You can do this typing:

label save list of label names **using** filename

which creates a do-file containing a **label define** command for each label in the list. If you do not specify an extension on the filename, ".do" will be assumed. You can then **use** the data set to which you wish to add the label(s), and **do** filename.

All labels are stored in the same area of memory as the data set. As labels are added, STATA silently readjusts the maximum number of observations that can be loaded. As a result, it is good practice to drop unused value labels. Also, **describe** your data set occasionally to make sure there is sufficient space available before increasing the number of observations.

Example

```
. drop _all
R; T=0.55 12:37:19

. input odd even

             odd        even
  1. 1 2
  2. 3 4
  3. 5 6
  4. 7 8
  5. 9 10
  6. end
R; T=1.04 12:37:21

. describe
Contains data
 Obs:     5 (max=  605)
Vars:     2 (max=  100)
  1. odd          float  %9.0g
  2. even         float  %9.0g
Sorted by:
Note:  Data has changed since last save
R; T=0.88 12:37:22

. list

             odd        even
  1.          1.          2.
  2.          3.          4.
  3.          5.          6.
  4.          7.          8.
  5.          9.         10.
R; T=1.05 12:37:23
```

```
. label data "Odd and even numbers"
R; T=0.06 12:37:24

. label variable odd "Odd numbers"
R; T=0.05 12:37:24

. label variable even "Even numbers"
R; T=0.06 12:37:24

. label dir
R; T=0.11 12:37:25
```

```
. label define oddlbl 1 "One" 3 "Three" 5 "Five" 7 "Seven" 9 "Nine"
R; T=0.11 12:37:25

. label define evenlbl 2 "Two" 4 "Four" 6 "Six" 8 "Eight" 10 "Ten"
R; T=0.17 12:37:26

. label dir
evenlbl
oddlbl
R; T=0.22 12:37:27

. label values odd oddlbl
R; T=0.06 12:37:27

. label values even evenlbl
R; T=0.11 12:37:27
```

```
. describe
Contains data                              Odd and even numbers
 Obs:     5 (max=  604)
Vars:     2 (max=  100)
  1. odd          float  %9.0g    oddlbl   Odd numbers
  2. even         float  %9.0g    evenlbl  Even numbers
Sorted by:
Note:  Data has changed since last save
R; T=0.99 12:37:28

. list

           odd       even
  1.       One        Two
  2.     Three       Four
  3.      Five        Six
  4.     Seven      Eight
  5.      Nine        Ten
R; T=1.05 12:37:30
```

```
. label drop oddlbl
R; T=0.16 12:37:30

. label dir
evenlbl
R; T=0.16 12:37:31

. describe
Contains data                              Odd and even numbers
 Obs:     5 (max=  604)
Vars:     2 (max=  100)
  1. odd          float  %9.0g    oddlbl   Odd numbers
  2. even         float  %9.0g    evenlbl  Even numbers
Sorted by:
Note:  Data has changed since last save
R; T=0.99 12:37:32
```

```
. list

           odd       even
  1.        1.        Two
  2.        3.       Four
  3.        5.        Six
  4.        7.      Eight
  5.        9.        Ten
R; T=1.10 12:37:33
```

[**by** varlist:] **list** [varlist]
[**in** range] [**if** exp]

Purpose

The **list** command displays the values of variables in the current data set. If no varlist is specified, then the values of all the variables are displayed.

Output

The names of the specified variables are displayed across the top of the screen. The values are displayed under the variable names and the observation number is displayed at the far left of the screen.

Options

nolabel causes the numeric codes to be displayed rather than the value labels.
noobs suppresses the printing of observation numbers.

Example

```
. use hoel.dta
(Data from Hoel´s textbook)
R; T=0.55 12:39:06
```

```
. list

      marriage       educ     number
  1.  Very Low    College        18.
  2.       Low    College        29.
  3.      High    College        70.
  4.  Vry High    College       115.
  5.  Very Low       H.S.        17.
  6.       Low       H.S.        28.
  7.      High       H.S.        30.
  8.  Vry High       H.S.        41.
  9.  Very Low     < H.S.        11.
 10.       Low     < H.S.        10.
 11.      High     < H.S.        11.
 12.  Vry High     < H.S.        20.
R; T=2.75 12:39:13

. list, nolabel

      marriage       educ     number
  1.        1.         1.        18.
  2.        2.         1.        29.
  3.        3.         1.        70.
  4.        4.         1.       115.
  5.        1.         2.        17.
  6.        2.         2.        28.
  7.        3.         2.        30.
  8.        4.         2.        41.
  9.        1.         3.        11.
 10.        2.         3.        10.
 11.        3.         3.        11.
 12.        4.         3.        20.
R; T=2.92 12:39:21

. by educ: list number if number>30, noobs

-> educ=  College
   number
      70.
     115.
```

```
list

-> educ=     H.S.
   number
      41.

-> educ=   < H.S.
   number

R; T=1.70 12:39:54
```

macro macro_command

"macro_command" can be any of the following:

```
define   macro_name "string"
dir
drop     list of macro_names
list     list of macro_names
```

Purpose

The **macro** command assigns strings to designated macro names. Before executing any STATA commands, all macro names are replaced with their associated strings.

Remarks

A STATA macro is a name that has been associated with a string using the STATA **macro** command. The rules for naming macros are identical to the rules for naming variables. When a macro name is referenced, it must be preceded by a percent sign (%). For example:

macro define mname "string"

followed by the command

... **%mname** ...

will cause STATA to execute the command

... string ...

Macros may be indirectly referenced. For instance, if the macro iname contains "mname", and the macro mname contains "string", then "... %%iname ..." is interpreted as "... string ...". The results of macro substitution may be joined with the following text using the join character forward single quote (`). If the macro drive contains "b:" then

... **%drive`myfile.dta** ...

is interpreted as

... b:myfile.dta ...

The **macro** command can also associate a string with one of the ten function keys (labeled "F1" through "F10"). The variables **_1** through **_10** implicitly refer to the function keys. Thus, the macro command

macro define _1 "This is key F1"

will cause the string "This is key F1" (without the double quotes) to be typed every time the F1 function key is pressed. Names of the form "_#" where "#" is an integer greater than ten can also be defined as macros. However, these macros are not associated with any keys on the keyboard.

In a do-file, variables of the form "_#" are interpreted as parameters of the **do** command. For example, if a do-file is executed with the command

do do-file_name word1 word2 word3 ...

then, during the execution of the do-file, **%_1** is replaced with word1, **%_2** is replaced with word2, and so on. All macros that begin with an underscore (_) are local to a do-file in that their meanings inside the do-file are independent of their definition outside the do-file. The original definitions of all "_" macros are restored at the completion of the do-file.

In the **macro drop** and **macro list** commands, the word **_all** can be used in place of the list of macro names to indicate that the command should operate on all currently defined macros.

Example

```
. macro define usehoel "use hoel.dta, clear"
R; T=2.86 14:13:02

. macro list
usehoel:   use hoel.dta, clear
R; T=2.09 14:13:07

. %usehoel
(Data from Hoel´s textbook)
R; T=6.48 14:13:15

. macro drop usehoel
R; T=2.80 14:13:21

. macro list
R; T=1.81 14:13:25

. type command.do
%command %_1 %_2, %_3

. macro define command "tabulate"
R; T=2.75 14:13:39
```

```
macro

. do command.do marriage

. %command %_1 %_2, %_3
    marriage|      Freq.     Percent        Cum.
------------+-----------------------------------
   Very Low |          3       25.00       25.00
        Low |          3       25.00       50.00
       High |          3       25.00       75.00
   Vry High |          3       25.00      100.00
------------+-----------------------------------
      Total |         12      100.00
R; T=5.39 14:13:53

.
end of do file
R; T=16.20 14:13:57
```

---

merge [varlist] **using** filename

Purpose

The **merge** command joins corresponding observations from the data set currently in memory (called the "master" data set) and from a STATA format data set stored on disk (called the "using" data set) into single observations. If the using filename is specified without an extension, ".dta" is assumed. The data set that results from **merge** replaces the master data set. The using data set is not changed.

Remarks

The using data set must be a STATA format data set, that is, it must have been created with the **save** command. If the file was encoded, the current encoding key must be set appropriately (see **set encode** for details).

Two kinds of merges can be performed. If no varlist is specified, a one-to-one merge is performed. If a varlist is specified, a match-merge is performed.

In a one-to-one merge, the first observation in the master data set is joined with the first observation in the using data set, the second observation in the master data set is joined with the second observation in the using data set, and so on. If a variable name occurs in both the master and the using data sets, the joined observation takes the variable′s value from the master data set. When the master and using data sets contain different numbers of observations, missing values are joined with the remaining observations from the longer data set.

In a match-merge, observations are joined if the values of the variables in the varlist are the same. To perform a match-merge, all variables in the varlist must appear in both the master and the using data set. In addition, both data sets must be sorted in the order of the varlist.

A match-merge proceeds by taking an observation from the master data set and one from the using data set and comparing the values of the variables in the varlist. If the varlist values match, then the observations are joined in the same way as in a one-to-one merge.

If the varlist values do not match, the observation from the "earlier data set" (the data set whose varlist value comes first in the sort order) is joined with a pseudo-observation from the "later data set" (the other data set). All the variables in the pseudo-observation contain missing values. The actual observation from the later data set is retained and compared to the next observation in the earlier data set.

The master and/or using data sets may have multiple observations with the same varlist value. These multiple observations are joined sequentially, as in a one-to-one merge. If the data sets have an unequal number of observations with the same varlist value, the last such observation in the "shorter" data set is replicated until the number of observations is equal.

The **merge** command adds a new variable, called **_merge**, to the master data set. This variable is coded according to the table below.

1. This observation occurred only in the master data set.
2. This observation occurred only in the using data set.
3. This observation is the result of joining an observation from the master data set with one from the using data set.

Options

nolabel prevents copying of labels from the disk data set into the current data set.

Example

```
. use merge1.dta, clear
(Merge data #1)
R; T=6.38 14:50:45

. list

            odd       even     negodd
  1.         1.         2.        -1.
  2.         3.         4.        -3.
  3.         5.         6.        -5.
  4.         7.         8.        -7.
  5.         9.        10.        -9.
R; T=3.96 14:50:51

. use merge2.dta, clear
(Merge data #2)
R; T=6.71 14:51:00
```

```
. list

             odd          even     negeven
  1.          7.            8.         -8.
  2.          9.           10.        -10.
  3.         11.           12.        -12.
  4.         13.           14.        -14.
R; T=4.45 14:51:07

. use merge1.dta, clear
(Merge data #1)
R; T=6.42 14:51:20

. merge odd using merge2.dta
R; T=5.55 14:51:28

. list

          odd      even    negodd    negeven    _merge
  1.       1.        2.       -1.          .        1.
  2.       3.        4.       -3.          .        1.
  3.       5.        6.       -5.          .        1.
  4.       7.        8.       -7.        -8.        3.
  5.       9.       10.       -9.       -10.        3.
  6.      11.       12.         .       -12.        2.
  7.      13.       14.         .       -14.        2.
R; T=5.77 14:51:36

. use merge1.dta, clear
(Merge data #1)
R; T=6.60 14:51:48

. merge using merge2.dta
R; T=5.50 14:51:55

. list

          odd      even    negodd    negeven    _merge
  1.       1.        2.       -1.        -8.        3.
  2.       3.        4.       -3.       -10.        3.
  3.       5.        6.       -5.       -12.        3.
  4.     7.      8.     -7.     -14.      3.
  5.     9.     10.     -9.        .      1.
R; T=5.17 14:52:03

. use merge2.dta, clear
(Merge data #2)
R; T=6.81 14:52:12

. merge odd using merge1.dta
R; T=5.50 14:52:20

. list

        odd    even  negeven  negodd  _merge
  1.     7.      8.      -8.     -7.      3.
  2.     9.     10.     -10.     -9.      3.
  3.    11.     12.     -12.       .      1.
  4.    13.     14.     -14.       .      1.
  5.     1.      2.        .     -1.      2.
  6.     3.      4.        .     -3.      2.
  7.     5.      6.        .     -5.      2.
R; T=5.55 14:52:28

. use merge2.dta, clear
(Merge data #2)
R; T=6.81 14:52:37

. merge using merge1.dta
R; T=5.71 14:52:45

. list

        odd    even  negeven  negodd  _merge
  1.     7.      8.      -8.     -1.      3.
  2.     9.     10.     -10.     -3.      3.
  3.    11.     12.     -12.     -5.      3.
  4.    13.     14.     -14.     -7.      3.
  5.     9.     10.        .     -9.      2.
R; T=5.32 14:52:52
```

[**by** varlist:] **modify** [varlist] [**in** range] [**if** exp]

Purpose

The **modify** command allows you to alter the values of existing variables for particular observations.

Remarks

The **replace** command performs the same function as the **modify** command. It is easier to use the **modify** command if there are only a few values to change, or if the changes cannot be written as an expression.

After the **modify** command is entered, STATA prompts you with the variable name, observation number, and current value. At this point, you type in the correct value. STATA reprompts you with the corrected value. If you are satisfied with the correction, press the Return key; otherwise type a new value. You can terminate the **modify** command at any time by typing **end** instead of a new value.

Options

automatic causes STATA to create value labels from non-numeric data.
nolabel causes the numeric values of the variables to be displayed rather than any associated value labels.

Example

```
. list

           odd       even
  1.        1.         2.
  2.        3.         4.
  3.       -8.         6.
  4.        7.         8.
  5.        9.        10.
R; T=3.46 15:00:26

. modify odd in 3

           odd
  3.       -8.
       odd =       -8.
       odd = . 6
       odd =        6.
       odd = . 5
       odd =        5.
       odd = .
R; T=83.21 15:01:57
```

---

more

Purpose

The **more** command causes STATA to display the string "--more--" and pause until any key is depressed.

Remarks

The **more** command is useful in do-files to keep information from scrolling off the screen before it can be read. If Ctrl-Break response to the "--more--" prompt, an error is generated. Since do-files terminate on any error, this feature provides a way of halting the execution of a do-file. The "--more--" prompt is not echoed to the spool file.

---

outfile [varlist] **using** filename
[**in** range][**if** exp]

Purpose

The **outfile** command writes data to an external disk file. This new file is not in STATA format, although it can be read back by STATA using **infile.** If "filename" is specified without an extension, ".raw" is assumed.

Remarks

The **outfile** command enables data to be sent to a disk file for processing by a non-STATA program. Each observation is written as a single "record"; that is, a carriage return/line feed combination is written to the disk file after each observation. The values of the variables are written using their current display formats. A single blank space separates each value.

Options

comma causes STATA to write the disk data set in comma-separated-values format. In this format, values are separated by commas instead of blanks. Missing values are written as two consecutive commas.

nolabel causes STATA to write the numeric values of labeled variables. The default is to write the labels enclosed in double quotes.

replace permits the **outfile** command to overwrite an existing data set. **replace** may not be abbreviated.

Example

```
. use hoel.dta
(Data from Hoel´s textbook)
R; T=1.04 12:49:06

. outfile using ex1.raw, nolabel
R; T=1.27 12:49:33

. type ex1.raw
       1.          1.          18.
       2.          1.          29.
       3.          1.          70.
       4.          1.         115.
       1.          2.          17.
       2.          2.          28.
       3.          2.          30.
       4.          2.          41.
       1.          3.          11.
       2.          3.          10.
       3.          3.          11.
       4.          3.          20.

R; T=5.16 12:49:46

. outfile using ex2.raw, nolabel comma
R; T=1.26 12:49:53

. type ex2.raw
1.,1.,18.
2.,1.,29.
3.,1.,70.
4.,1.,115.
1.,2.,17.
2.,2.,28.
3.,2.,30.
```

outfile

```
4.,2.,41.
1.,3.,11.
2.,3.,10.
3.,3.,11.
4.,3.,20.

R; T=2.04 12:49:58
```

```
[by varlist:] plot yvar1 [yvar2 [...]] xvar
              [in range] [if exp]
```

Purpose

The **plot** command produces a scatter diagram for the variables yvar1 and xvar. If more than one yvar is specified, a single diagram is produced that overlays the plot of each yvar against xvar. No more than nine yvars may be specified.

Remarks

The **plot** command displays a "line printer plot" which is a scatter diagram drawn using characters available on an ordinary typewriter or line printer. As a result, this scatter diagram can be displayed on any monitor and printed on any printer. The diagram necessarily has a rougher appearance than one designed to be displayed on a graphics monitor.

Output

STATA displays a scatter diagram of yvar against xvar. Each point is plotted with an asterisk (*). The minimum and maximum values of yvar and xvar are marked and the variable names are displayed along the axes. When more than one yvar is specified, the first yvar is plotted with the letter "A", the second with the letter "B", and so on. When more than one variable is plotted at the same point, that point is plotted with an asterisk (*).

Options

columns(#) specifies the column width of the plot. The number specified must lie between 30 and 133. The default is 75. Note that **columns** is specified as 10 larger than the actual width of the plot. This extra space is used to label the diagram.

encode plots points that occur more than once in the data with the number of occurrences. If a point occurs only once, it is plotted as usual with an asterisk (*). Points that occur twice are plotted with the numeral two (2). Points that occur three times are plotted with the numeral three (3), and so on. Points that occur ten times are plotted with "A", eleven with "B", and so on, until "Z". The letter "Z" is used subsequently. **encode** may not be specified if there is more than one yvar.

hlines(#) causes a horizontal line of dashes (-) to be drawn across the diagram every #-th line where "#" is a number between 0 and the line height of the plot. Specifying "#" as 0, which is the default, results in no horizontal lines.

lines(#) specifies the line height of the plot. The number specified must lie between 10 and 80. The default is 23. Note that **lines** is specified as 3 larger than the number of lines occupied by the plot. This extra space is used to label the diagram.

vlines(#) causes a vertical line of vertical bars (|) to be drawn on the diagram every #-th column where "#" is a number between 0 and the column width of the plot. Specifying "#" as 0, which is the default, results in no vertical lines.

plot

Example

```
. drop _all
R; T=0.17 14:10:25

. set obs 21
obs was 0, now 21
R; T=0.16 14:10:29

. generate x = _n - 11
R; T=0.33 14:10:38

. generate y = x * x
R; T=0.44 14:10:45

. plot y x, lines(16) columns(50)

    100. +
         | *                                      *
         |
         |
         |  *                                   *
         |
    y    |    *                               *
         |
         |      *                           *
         |        *                       *
         |          *                   *
         |
         |            * *           * *
      0. +                * * * * *
          +---------------------------------------+
             -10.           x                10.

R; T=3.79 14:11:07
```

```
. plot y x, lines(16) columns(50) hlines(5) vlines(10)

    100. +           |         |         |
         | *         |         |         |        *
         |           |         |         |
         |           |         |         |
         | -*--------+---------+---------+------*-
         |           |         |         |
    y    |    *      |         |         |    *
         |           |         |         |
         |      *    |         |         |  *
         | -------*--+---------+---------+*-------
         |          *|         |        *|
         |           |         |         |
         |           |* *      |    * *  |
      0. +           |    * * *|* *      |
          +---------------------------------------+
             -10.           x                10.

R; T=4.18 14:11:51

. generate newx = abs(x)
R; T=0.38 14:12:04
```

plot

```
. plot y newx, lines(16) columns(50) encode
```

```
    100. +
         |                                        2
         |
         |
         |                                    2
         |
    y    |                                2
         |
         |                            2
         |                        2
         |                    2
         |
         |            2   2
      0. + *  2   2
          +---------------------------------------+
               0.          newx               10.

R; T=3.78 14:12:33
```

query

Purpose

The **query** command displays the settings of various system parameters.

Remarks

The parameters displayed by the **query** command can be changed by the **set** command. See the **set** command for a complete description of each parameter.

Example

```
. query
Displ: linesize= 79  pagesize= 23
Spool: linesize= 79  pagesize=  0
Spool file query.ex, proc, open; spooling on
help file stata.hlp
prefix ""
beep off; type=float; more=0; encode: ""
R; T=0.38 11:32:59
```

```
[by varlist:]    regress    [varname [varlist1
                 [(varlist2)]]
       [= exp] [in range] [if exp] ]
```

Purpose

The **regress** command regresses "varname" against "varlist1". If "(varlist2)" appears, it indicates a list of instrumental variables.

Remarks

The **regress** command performs linear multivariate regression. By defining the appropriate dummy variables, ANOVA and general linear models can be estimated using **regress**. If **regress** is typed with no arguments, the results of the last regression are re-displayed. If only one variable is specified, that variable is regressed on a constant.

Output

The **regress** command produces a variety of summary statistics and a table of regression coefficients. The summary statistics include the number of observations, the sum of the weights if the regression is weighted, an ANOVA table for the regression model, the F-statistic and marginal significance level for the hypothesis that all coefficients (except the constant) are zero, the R-square and adjusted R-square statistics and the square root of the mean squared residual. The coefficient table includes the estimated coefficients and their standard errors, the associated t-statistics and their marginal significances, and the mean for each variable. If there is a constant in the model, it is listed under the variable name "_cons".

The **regress** command stores some of its calculations in memory and creates two system variables, **_coef** and **_pred**. The stored calculations are used by the **test** command. These stored calculations also enable you to review the most recent regression by typing **regress** with no arguments or qualifiers.

_coef is STATA´s name for the coefficients from the most recent regression. **_coef** is indexed by name. For example, if the most recent regression contained a regressor called "educ", then **_coef[educ]** will return the coefficient on "educ". **_pred** produces predicted values using the coefficients from the most recent regression. Like **_coef**, **_pred** is keyed to variable names. Thus, changing the contents of the variable "educ" after running the regression changes the values returned by **_pred** (if "educ" was an explanatory variable in the most recent regression). You can use this feature to run a regression, **use** another data set, and then make out-of-sample predictions. The only restriction is that the explanatory variables must have the same name. (If they do not, you can use **rename** to change them.) There are no restrictions on variable order or storage type. You may run a regression on **xp** and then use the **data** to make predictions or vice-versa.

To generate the residuals from a regression, use

generate resid=**varname-_pred**

Options

hascons indicates that a user defined constant or its equivalent is specified in the list of right hand side variables.

means causes STATA to **summarize** the variables in the regression before displaying the regression results.

noconstant suppresses the constant term (or intercept) in the regression.

noformat displays the summary statistics in **g** format regardless of the display format previously specified.

nooutput suppresses all regression output to the screen. This is useful for quickly defining **_coef[]** and **_pred**.

noscale suppresses the normalization of the weight.

Example

```
. use census
(Census Data)
R; T=1.48 13:46:39

. generate medage2 = medage*medage
R; T=0.88 13:46:52
```

```
. regress drate medage medage2
(obs=50)

   Source|     SS         df       MS            Number of obs =      50
---------+------------------------------         F(  2,    47) =   27.94
    Model|  4548.84187     2  2274.42094         Prob > F      =  0.0000
 Residual|  3825.65813    47  81.3969814         R-square      =  0.5432
---------+------------------------------         Adj R-square  =  0.5237
    Total|     8374.50    49  170.908163         Root MSE      =   9.022

 Variable|  Coefficient    Std. Error      t     Prob > |t|      Mean
---------+--------------------------------------------------------------
    drate|                                                         84.3
---------+--------------------------------------------------------------
   medage|     21.60568      13.38839     1.614     0.113         29.54
  medage2|     -.271548      .2270073    -1.196     0.238       875.422
    _cons|    -316.2128      197.4148    -1.602     0.116            1.
---------+--------------------------------------------------------------
R; T=10.98 13:47:11

. generate pop=poplt5+pop517+pop18p
R; T=0.94 13:48:04
```

```
. regress drate medage medage2 =pop
(sum of wgt is   2.2591E+08)
(obs=50)

  Source|       SS        df       MS            Number of obs =      50
---------+---------------------------            F(  2,    47) =   37.83
   Model|  3290.73747      2  1645.36874         Prob > F      =  0.0000
Residual|  2044.28169     47  43.4953552         R-square      =  0.6168
---------+---------------------------            Adj R-square  =  0.6005
   Total|  5335.01916     49  108.877942         Root MSE      =  6.5951

Variable| Coefficient     Std. Error       t     Prob > |t|      Mean
---------+--------------------------------------------------------------
   drate|                                                      87.34306
---------+--------------------------------------------------------------
  medage|    9.375407      11.65003     0.805     0.425        30.11047
 medage2|   -.0731833       .190467    -0.384     0.703        909.3716
   _cons|   -128.4041      177.9866    -0.721     0.474              1.
---------+--------------------------------------------------------------
R; T=13.24 13:48:26
```

```
. * instrumental variable example, if qualifier
. * removes Nevada from the data:
. regress dvcrate mrgrate (medage medage2) if mrgrate<1000
(obs=49)

  Source|       SS          df       MS                 Number of obs =      49
--------+-----------------------------------            F(  1,    47) =    7.08
   Model|  4201.65955        1  4201.65955              Prob > F      =  0.0106
Residual|  6733.40167       47  143.263865              R-square      =  0.1310
--------+-----------------------------------            Adj R-square  =  0.1125
   Total|  10935.0612       48  227.813776              Root MSE      =  11.969

Variable| Coefficient     Std. Error        t      Prob > |t|         Mean
--------+-----------------------------------------------------------------------
 dvcrate|                                                            54.2449
--------+-----------------------------------------------------------------------
 mrgrate|    .4382313       .1646469      2.662      0.011          106.7347
   _cons|    7.470411       17.65653      0.423      0.674                1.
--------+-----------------------------------------------------------------------
R; T=12.36 13:53:59
```

rename old-varname new-varname

Purpose

The **rename** command changes the name of an existing variable. The contents of the variable are unchanged.

Example

```
. use hoel.dta
R; T=0.55 13:14:49

. rename educ school
R; T=0.16 13:14:58

. describe
Contains data
 Obs:     12 (max=  590)
Vars:      3 (max=  100)
  1. marriage      float  %9.0g    marlbl
  2. school        float  %9.0g    edlbl
  3. number        float  %9.0g
Sorted by:  school  marriage
Note:  Data has changed since last save
R; T=1.21 13:15:01
```

[**by** varlist:] **replace** oldvar=exp
[**in** range][**if** exp]

Purpose

The **replace** command changes the contents of an existing variable. The command name **replace** cannot be abbreviated.

Remarks

The **replace** command is identical to the **generate** command except that it operates only on existing variables while **generate** operates only on new variables. If "**in** range" or "**if** exp" is specified, the observations of oldvar that are not **replaced** retain their original values.

Output

The number of changes actually made to the data.

Example

```
. use hoel.dta
(Data from Hoel´s textbook)
R; T=0.50 13:15:49

. list number

        number
  1.       18.
  2.       29.
  3.       70.
  4.      115.
  5.       17.
  6.       28.
  7.       30.
  8.       41.
  9.       11.
 10.       10.
 11.       11.
 12.       20.
R; T=1.59 13:15:59

. replace number = log(number)
(12 changes made)
R; T=0.88 13:16:08

. list number

        number
  1.  2.890372
  2.  3.367296
  3.  4.248495
  4.  4.744932
  5.  2.833213
  6.  3.332205
  7.  3.401197
  8.  3.713572
  9.  2.397895
 10.  2.302585
 11.  2.397895
 12.  2.995732
R; T=1.48 13:16:12
```

---

run filename [parameter_list]

Purpose

The **run** command is identical to the **do** command except that a **set output error** command is implied at the start of the **run** command. The previous output level is restored at the end of the do-file.

Options

The **run** command takes the same options as the **do** command.

---

save filename

Purpose

The **save** command stores the data set currently in memory as a disk data set with the name "filename". If no file extension is specified, ".dta" is used if the data set is **data** and ".xp" is used if the data is **xp**.

Remarks

The **save** command stores data sets in a special STATA format that is readable only by STATA. If the encoding key is set, the file will be encoded (see **set encode** for details). The **use** command brings a **saved** data set back into memory.

The **save** command stores all the information about a data set including the variable names, types, display formats, sort order, and all labels associated with the data set.

Options

nolabel omits value labels from the **saved** data set. However, the associations between variables and value label names are saved along with the data set label and any variable labels.

replace permits the **save** command to overwrite an existing "filename". **replace** may not be abbreviated.

---

set beep {**on** | **off**}

contents {**data** | **xp**}

display {**linesize** | **pagesize**} **#**

encode ["string"]

help filename

maxobs # [lrecl #]

maxvar # [lrecl #]

more #

obs #

output {**proc** | **inform** | **error**}

prefix [string]

seed #

spool {**linesize** | **pagesize**} **#**

type {**int** | **long** | **float** | **double**}

Purpose

The **set** command specifies values of STATA system parameters.

set

Remarks

A variety of system parameters can be specified by the **set** command. Their current values can be obtained during a STATA session by typing **query.** Each system parameter is described below.

set beep on|off indicates whether the computer should emit a beep at the completion of each command. The default is **off.**

set contents data|xp specifies the interpretation of the current data set. When the interpretation is changed from **data** to **xp**, STATA checks to see that the current data set is a valid **xp** data set. When **set contents xp** is typed STATA automatically renames the first variable in the data set **_cons.** When **set contents data** is typed STATA automatically renames **_cons** to **_user.**

set display|spool linesize|pagesize # controls the dimensions of STATA output. The **linesize** parameter indicates the number of characters that can be placed on one line. The **pagesize** parameter has no effect on spool file output. For the **display** it serves to indicate the number of lines that can be displayed before a "--more--" condition should arise.

set encode ["string"] causes **save** and **use** commands to encode and decode the data set using "string" as the key. Use **set encode** with no arguments to turn encryption off.

set help filename specifies the file to be used by the **help** command. The default filename is "STATA.HLP". You can use this command to place the help file on a different drive than the logged drive and/or in a different directory than the current directory.

set maxobs|**maxvar** # specifies either the maximum number of observations or variables in the current STATA data set. When one of these parameters is specified, the other is automatically set to the maximum value consistent with the amount of RAM in your computer. The value for **maxvar** can be attained only if all the variables in the data set have type **int** (unless you specify **lrecl**). For more information on how STATA stores variables, consult Appendix B, Memory Management in STATA.

set more # specifies the number of seconds that the output is halted when the "--more--" message is displayed. If "#" is 0 (the default), then output is halted until a key is pressed.

set obs # changes the number of observations in the current data set. "#" must be at least as great as the current number of observations. If there are variables in memory, the values of all new observations are set to missing value.

set output specifies the output to be displayed. The default, **proc**, means that all output, including procedure (command) output is displayed. **inform** suppresses procedure output but displays informative messages, such as the return message. **error** suppresses all output except error messages. **error** is useful for do-files that you wish to run "silently". **inform** is most useful for the **by** varlist: **generate** construct when you do not want the details on the missing values generated for each by group. The current **set output** level is not shown by **query** since **query**´s output will only be displayed if the current output level is **proc**.

set prefix [string] defines a string that will be appended in front of all filenames that do not start with an explicit drive indication, or that are device references, or that start with a backslash (\). For instance, if string is **b:**, then the command **use myfile** is interpreted as **use b:myfile.** This provides a convenient way to have all STATA disk input and output performed on a directory other than the current directory. The **help** filename is unaffected by the prefix.

set seed # initializes the random number seed for the **uniform()** function. It should be specified as a large, odd, positive integer. If a negative number is specified, it will be made positive. An even number will be made odd. STATA always initializes the seed to 1001, and so will always produce the same sequence of random numbers unless you re-initialize the seed.

set type int|long|float|double specifies the default type to be assigned to all new variables. The initial default is **float.**

---

sort varlist [**in** range]

Purpose

The **sort** command arranges the observations of the current data set in ascending order of the values of the variables in "varlist". Missing values are interpreted to be larger than any other number, and so are placed last. The data set is marked as being sorted by "varlist" unless "**in** range" is specified.

Remarks

The sorting technique used by the **sort** command is very fast. A side effect of this technique is that the order of variables not included in varlist is not maintained. If it is desired to maintain the order of additional variables, include them at the end of the varlist.

The worst case for the STATA sort algorithm is a varlist that is already sorted. As a safeguard, the **sort** command checks to see if the data set is already sorted in either ascending or descending order before it begins. This check fails to detect almost-sorted data sets, however. If you believe your data is almost-sorted, you may wish to deliberately randomize your data before sorting by first **sort**ing on a random number. (You can **generate** a random variable with the **uniform()** function.)

Some timings (performed on an XT with an 8087 using a memory disk) illustrate this point. Sorting 500 randomly ordered observations takes roughly 30 seconds. Sorting 500 observations that are already in ascending order takes 3.57 seconds. Sorting 500 observations that are in descending order takes 5.54 seconds. Sorting 500 observations that are in ascending order except for one interchange (e.g., 1, 3, 2, 4, 5, 6, ...) takes 463.17 seconds. The following sequence of statements, which produce exactly the same result, takes only 63.47 seconds:

```
generate random=uniform()
sort random
sort almost
drop random
```

Example

```
. use hoel.dta
(Data from Hoel´s textbook)
R; T=0.55 11:45:52

. sort educ
R; T=0.33 11:46:00

. list

      marriage        educ     number
  1.  Vry High     College       115.
  2.      High     College        70.
  3.  Very Low     College        18.
  4.       Low     College        29.
  5.  Vry High        H.S.        41.
  6.       Low        H.S.        28.
  7.  Very Low        H.S.        17.
  8.      High        H.S.        30.
  9.  Vry High      < H.S.        20.
 10.  Very Low      < H.S.        11.
 11.       Low      < H.S.        10.
 12.      High      < H.S.        11.
R; T=2.74 11:46:03
```

```
. sort marriage number
R; T=0.39 11:46:04

. list

       marriage          educ        number
  1.  Very Low        < H.S.           11.
  2.  Very Low          H.S.           17.
  3.  Very Low       College           18.
  4.       Low        < H.S.           10.
  5.       Low          H.S.           28.
  6.       Low       College           29.
  7.      High        < H.S.           11.
  8.      High          H.S.           30.
  9.      High       College           70.
 10.  Vry High        < H.S.           20.
 11.  Vry High          H.S.           41.
 12.  Vry High       College          115.
R; T=2.75 11:46:07
```

---

spool {**using** filename |
on | **off** | **close**}

Purpose

The **spool** command echoes a copy of the current STATA session to a file.

Remarks

spool creates a log of all or part of a STATA session. This log is stored as an ordinary DOS file. It can be edited and/or printed after the end of the STATA session.

To initiate spooling, give the command

spool using filename

where "filename" is the name under which you wish to store the session log. If no extension is specified with "filename", ".spl" is used. You may also specify device names, for instance, **aux:**, **com1:**, **prn:**, and **lpt1:**. Unlike DOS, STATA requires the : on the end of the device name. Thus, to **spool** directly to the printer, give the command

spool using prn:

We recommend, however, **spool**ing to disk files since you can then print multiple copies of the output or edit it using your word processor. To temporarily halt spooling, give the command

spool off

Spooling can be resumed with

spool on

Give the command

spool close

to terminate spooling and to save the file containing the session log.

Output

The log contains everything that appears on the screen during the STATA session. Commands and their associated output appear just as they were initially displayed. However, the "--more--" message that appears when output to the screen is halted is not sent to the spooled file.

The linesize of the spooled file may be **set** independently of the linesize of the display screen. This feature is useful for creating oversize plots. For more details, see the description of the **set** command.

Options

Options for the **spool** command may only be specified when the command

spool using filename

is given.

noproc causes STATA to spool only the characters you type. No output of any kind (including return and error messages) is sent to the spooled file. This option offers an easy way to generate a do-file.

replace directs STATA to allow spooling to overwrite an existing file. **replace** may not be abbreviated

[**by** varlist:] **summarize** [varlist]
[=exp] [**in** range] [**if** exp]

Purpose

The **summarize** command calculates and displays a variety of univariate summary statistics. If no varlist is indicated, then summary statistics are calculated for all the variables in the current data set.

Remarks

If "=exp" is specified, the expression is used to weight the data. Each observation is multiplied by the value of the weighting expression before the summary statistics are calculated. In other words, the weighting expression is interpreted as the discrete density of each observation.

Output

The **summarize** command can produce two different sets of summary statistics. Normally, the summary statistics are the number of non-missing observations, the mean and standard deviation, and the minimum and maximum values for each variable. If the **detail** option is specified, the same information is presented along with the variance, skewness, and kurtosis. The four smallest and four largest values are listed instead of just the minimum and maximum. The following percentiles are also listed: 1%, 5%, 10%, 25%, 50% (the median), 75%, 90%, 95%, and 99%.

Options

detail produces the additional statistics described above.

noformat displays the summary statistics in **g** format regardless of the display format previously specified.

noscale suppresses the normalization of the weight.

Example

```
. use census
(Census Data)
R; T=1.54 15:19:44

. summarize

 varname|  Obs         Mean   Std. Dev.         Min          Max
--------+-------------------------------------------------------
 mrgrate|    50      133.16  188.095976         75.        1428.
 dvcrate|    50       56.62  22.4770995         29.         173.
   state|    50       29.32  15.7822427          1.          56.
   drate|    50        84.3  13.0731849         40.         107.
  poplt5|    50   326277.78  331585.142      35998.     1708400.
  pop517|    50    945951.6  959372.831      91796.     4680558.
  pop65p|    50    509502.8  538932.376      11547.     2414250.
  pop18p|    50  3245920.06  3430531.31     271106.    17278944.
  medage|    50       29.54  1.69344465  24.2000008   34.7000008
  dvcmrg|    50  .501854873  .116144019   .12114846   .770114958
division|    50        5.12  2.56061217          1.           9.
  region|    50        2.66  1.06157373          1.           4.
R; T=9.67 15:20:11
```

```
. summarize drate, detail

                                Death Rate
-------------------------------------------------------------------
      Percentiles                Smallest
  1%        40.                     40.
  5%        55.                     50.
 10%      68.5                      55.      Obs                 50
 25%        79.                     65.      Sum of Wgt.         50.

 50%      85.5                               Mean              84.3
                                 Largest     Std. Dev.   13.0731849
 75%        93.                     99.
 90%        98.                    100.      Variance    170.908163
 95%       100.                    104.      Skewness   -1.19328432
 99%       107.                    107.      Kurtosis    4.99267571

R; T=7.53 15:20:30
```

```
[by varlist:]  tabulate  varlist
   [= exp][in range] [if exp]
```

Purpose

The **tabulate** command produces one-way and two-way tables of frequency counts.

Remarks

For each value of a specified variable (or set of values for a pair of variables), **tabulate** reports the number of observations with that value. In other words, **tabulate** reports the frequency of occurrence of each value. There must be at least one variable and at most two variables included in the varlist. N-way tables can be calculated by preceding the **tabulate** command with the **by** varlist: prefix.

If "=exp" is specified, the expression is used to weight the variables. This weight is interpreted as a replication (number of cases) count.

The **generate** option produces a set of dummy variables indicating each level of the **tabulate**d variable. These dummy variables can be interacted and used with the **regress** command to estimate ANOVA and general linear models.

Output

The **tabulate** command displays the values of the specified variable(s) in ascending order. For one-way tables, frequency of occurrence and relative frequency of each value is listed along with the percentage of the data set for which the variable is at least as large as this value. For two-way tables, the frequency of occurrence of each pair of values is reported in tabular format.

Options

cell displays the relative frequency of each cell in a two-way table.

chi2 causes STATA to calculate and display the chi-squared statistic for the hypothesis that the rows and columns of a two-way table are independent.

column displays in each cell of a two-way table the relative frequency of that cell within its column.

generate(name) creates a set of dummy variables that indicate each value of the variable. This option cannot be used when two variables are **tabulated**.

nofreq suppresses the printing of the frequencies.

plot produces a bar chart of the relative frequencies. This option cannot be used when two variables are **tabulated**.

row displays in each cell of a two-way table the relative frequency of that cell within its row.

Example

```
. use hoel.dta
(Data from Hoel's textbook)
R; T=0.55 12:03:54

. tabulate educ
    Level of|
   Education|       Freq.      Percent          Cum.
------------+--------------------------------------
     College |          4        33.33         33.33
        H.S. |          4        33.33         66.67
      < H.S. |          4        33.33        100.00
------------+--------------------------------------
       Total |         12       100.00
R; T=2.15 12:04:01
```

```
. tabulate educ = number, plot
    Level of|
   Education|      Freq.
------------+------------+--------------------------------------------------
     College |        232 |*************************************************
        H.S. |        116 |******************
      < H.S. |         52 |***
------------+------------+--------------------------------------------------
       Total |        400
R; T=3.08 12:04:04
```

```
. tabulate educ marriage = number, row chi2

   Level of| Marriage Adjustment Score ->
  Education| Very Low        Low       High  Vry High |     Total
-----------+----------------------------------------+----------
   College |       18         29         70       115 |       232
           |     7.76      12.50      30.17     49.57 |    100.00
-----------+----------------------------------------+----------
      H.S. |       17         28         30        41 |       116
           |    14.66      24.14      25.86     35.34 |    100.00
-----------+----------------------------------------+----------
    < H.S. |       11         10         11        20 |        52
           |    21.15      19.23      21.15     38.46 |    100.00
-----------+----------------------------------------+----------
      Total|       46         67        111       176 |       400
           |    11.50      16.75      27.75     44.00 |    100.00

            chi2(6)= 19.9426    Prob>chi2=0.003
R; T=10.65 13:38:08
```

```
. tabulate educ, nofreq generate(ed)
R; T=0.82 12:04:13

. describe
Contains data                                    Data from Hoel´s textbook
 Obs:    12 (max=  604)
Vars:     6 (max=  100)
  1. marriage     float  %9.0g    marlbl   Marriage-adjustment score
  2. educ         float  %9.0g    edlbl    Level of education
  3. number       float  %9.0g             Number of cases
  4. ed1          int    %8.0g             educ==College
  5. ed2          int    %8.0g             educ==H.S.
  6. ed3          int    %8.0g             educ==< H.S.
Sorted by:  educ  marriage
Note:  Data has changed since last save
R; T=1.65 12:04:15
```

```
. list educ ed1 ed2 ed3

            educ         ed1         ed2         ed3
  1.     College          1.          0.          0.
  2.     College          1.          0.          0.
  3.     College          1.          0.          0.
  4.     College          1.          0.          0.
  5.        H.S.          0.          1.          0.
  6.        H.S.          0.          1.          0.
  7.        H.S.          0.          1.          0.
  8.        H.S.          0.          1.          0.
  9.      < H.S.          0.          0.          1.
 10.      < H.S.          0.          0.          1.
 11.      < H.S.          0.          0.          1.
 12.      < H.S.          0.          0.          1.
R; T=3.41 12:04:19
```

---

test exp=exp

Purpose

The **test** command tests linear hypotheses about the most recent regression.

Remarks

The **test** command performs F-tests of linear restrictions applied to the most recent regression. Multiple hypotheses can be tested by issuing multiple **test** commands and specifying the **accumulate** option.

Output

The **test** command echoes the hypotheses being tested. In addition, the F-value for the test and the probability, under the null hypothesis, of randomly drawing a value higher than the computed F-value are displayed.

Options

accumulate allows a hypothesis to be tested jointly with the hypotheses previously tested.

notest suppresses the output. This option is useful when you are only interested in the joint test of a number of hypotheses.

Example

```
. use census
(Census Data)
R; T=1.65 14:21:23

. generate medage2=medage*medage
R; T=0.88 14:21:36

. * please see example under regress for regression
. * the nooutput option suppresses the output here
. regress drate medage medage2, nooutput
(obs=50)
R; T=2.63 14:21:46

. test medage=0

 ( 1)  medage = 0.0

       F(  1,    47) =    2.60
            Prob > F =    0.1133

R; T=2.09 14:21:56

. test medage2=0, accumulate

 ( 1)  medage = 0.0
 ( 2)  medage2 = 0.0

       F(  2,    47) =   27.94
            Prob > F =    0.0000

R; T=2.03 14:22:05
```

```
. test medage=1, accumulate

 ( 1)  medage = 0.0
 ( 2)  medage2 = 0.0
 ( 3)  medage = 1.0
       Constraint 3 dropped

       F(  2,    47) =    27.94
            Prob > F =    0.0000

R; T=2.47 14:22:13

. test medage2=0

 ( 1)  medage2 = 0.0

       F(  1,    47) =     1.43
            Prob > F =    0.2376

R; T=1.92 14:22:21

. test 2*(medage+medage2/4)=(medage-medage2)/4

 ( 1)  1.75 medage + .75 medage2 = 0.0

       F(  1,    47) =     2.61
            Prob > F =    0.1126

R; T=2.53 14:22:47
```

type filename

Purpose

The **type** command lists the contents of a file stored on disk. This command is identical to the DOS TYPE command.

Example

```
. type type.do
use hoel.dta
describe
list

R; T=0.49 12:07:41
```

---

use filename

Purpose

The **use** command loads a STATA format data set from a disk file into memory. If no extension is specified with "filename", ".dta" is assumed.

Remarks

The data set specified by "filename" must be in STATA format, that is, it must have been created by **save**. If the file was encoded, the current encoding key must be set appropriately. See the description of the **set encode** command for details.

Output

If the data set has a data set label, it is displayed.

Options

clear permits the data set to be loaded even if there is a STATA data set currently in memory.
nolabel prevents value labels in the **save**d data set from being loaded. However, associations between variables and value label names are loaded.

Appendices

Description of Data Sets

census.dta

```
. describe
Contains data                                   Census Data
 Obs:     50 (max=  716)
Vars:     12 (max=   99)
  1. mrgrate      long   %10.0g                 Marriages per 100,000
  2. dvcrate      long   %10.0g                 Divorces per 100,000
  3. state        int    %8.0g     fips
  4. drate        long   %10.0g                 Death Rate
  5. poplt5       long   %10.0g                 Pop. < 5 yrs
  6. pop517       long   %10.0g                 Pop. 5<=age<=17
  7. pop65p       long   %10.0g
  8. pop18p       long   %10.0g                 Pop. 18+
  9. medage       float  %9.0g                  Median Age
 10. dvcmrg       float  %9.0g
 11. division     int    %8.0g     division Census Division
 12. region       int    %8.0g     region   Census Region
Sorted by:  state
R; T=2.42 13:13:56
```

```
. summarize

 varname|  Obs         Mean    Std. Dev.          Min          Max
--------+---------------------------------------------------------
 mrgrate|   50       133.16   188.095976          75.        1428.
 dvcrate|   50        56.62   22.4770995          29.         173.
   state|   50        29.32   15.7822427           1.          56.
   drate|   50         84.3   13.0731849          40.         107.
  poplt5|   50    326277.78   331585.142       35998.     1708400.
  pop517|   50     945951.6   959372.831       91796.     4680558.
  pop65p|   50     509502.8   538932.376       11547.     2414250.
  pop18p|   50   3245920.06   3430531.31      271106.    17278944.
  medage|   50        29.54   1.69344465   24.2000008   34.7000008
  dvcmrg|   50   .501854873   .116144019    .12114846   .770114958
division|   50         5.12   2.56061217           1.           9.
  region|   50         2.66   1.06157373           1.           4.
R; T=9.72 13:14:08
```

hoel.dta

```
. describe
Contains data                                         Data from Hoel´s textbook
 Obs:     12 (max=  151)
Vars:      3 (max=  100)
  1. marriage      float  %9.0g    marlbl   Marriage-adjustment score
  2. educ          float  %9.0g    edlbl    Level of education
  3. number        float  %9.0g             Number of cases
Sorted by:  educ  marriage
R; T=1.10 13:04:21
```

```
. list

      marriage      educ     number
  1.  Very Low   College       18.
  2.       Low   College       29.
  3.      High   College       70.
  4.  Vry High   College      115.
  5.  Very Low      H.S.       17.
  6.       Low      H.S.       28.
  7.      High      H.S.       30.
  8.  Vry High      H.S.       41.
  9.  Very Low    < H.S.       11.
 10.       Low    < H.S.       10.
 11.      High    < H.S.       11.
 12.  Vry High    < H.S.       20.
R; T=2.75 12:39:13
```

```
. list, nolabel

          marriage         educ      number
  1.         1.          1.          18.
  2.         2.          1.          29.
  3.         3.          1.          70.
  4.         4.          1.         115.
  5.         1.          2.          17.
  6.         2.          2.          28.
  7.         3.          2.          30.
  8.         4.          2.          41.
  9.         1.          3.          11.
 10.         2.          3.          10.
 11.         3.          3.          11.
 12.         4.          3.          20.
R; T=2.92 12:39:21
```

oddeven.dta

```
. describe
Contains data
 Obs:      5 (max=  610)
Vars:      2 (max=  100)
  1. number          float  %9.0g
  2. odd             float  %9.0g
Sorted by:
R; T=0.55 14:22:01

. list

          number          odd
  1.          1.           1.
  2.          2.           3.
  3.          3.           5.
  4.          4.           7.
  5.          5.           9.
R; T=2.42 14:22:06
```

Appendix B

Memory Management in STATA

STATA stores the current data set in memory. As a result, STATA runs quickly and changes made to the current data set do not affect any copies of the data stored on disk. If the power fails or the computer is accidentally re-booted, only the contents of memory are lost. No disk files are left open (with the possible exception of **spool**), and no pointers needed to make the disk files intelligible are stored in memory.

The price you pay for these features is the requirement that the current data set be able to fit in memory. When STATA is started, the program examines your PC, determines how much memory is available, and then lays claim to all of it. Thus the size of the largest data set that you can use depends upon the hardware configuration of your PC.

Data sets have two dimensions: the number of variables and the number of observations. On start-up, STATA sets the maximum number of variables to 99. This setting leaves space for a maximum of 721 observations on a 256K PC or 2,573 observations on a 640K PC (the version you have may differ slightly from these numbers).

If you desire, you may trade off variables for observations or observations for variables by using the **set maxvar** or **set maxobs** command. Setting **maxvar** to 50, for instance, makes it possible to have 1,416 observations on a 256K PC and 5,047 observations on a 640K PC (again, your numbers may differ). Setting **maxvar** to 25 allows 2,779 observations on 256K or 9,900 on a 640K PC. In practice, the PC is just not fast enough to make working with 9,000 observations in an interactive environment pleasant. It is feasible though.

When it is necessary to pack as much data into memory as possible, you may need to reset the **lrecl** parameter. Recall that there are four variable types in STATA (**int**, **long**, **float**, and **double**) and that a single observation of each variable type requires a different amount of memory: an **int** requires 2 bytes, a **long** or **float** requires 4 bytes, and a **double** requires 8 bytes. The amount of memory that is available for storing a single observation of all variables is equal to **lrecl.**

Normally, STATA sets **lrecl** to 2***maxvar**, thus **lrecl** is equal to 198 bytes when the program is started. Since only **ints** are 2 bytes long, the current data set can contain 99 variables only if they are all **ints**. However, the data set may contain up to 49 **longs** or **floats** (4 bytes times 49 variables = 196) or up to 24 **doubles**. Variable types can be combined freely in the same data set as long as the sum over the variable types of the number of variables of each type times the storage requirement for each type is less than or equal to **lrecl** bytes. For example, with **lrecl**=198 and **maxvar**=99, it is possible to store 19 **ints**, 20 **float**, and 10 **doubles** (19*2 + 20*4 + 10*8 = 198 = **lrecl**). It is impossible to add more variables to this data set even though the data set contains only 69 variables and **maxvar** is equal to 99.

If all the variables you use are **floats**, you may want to set **lrecl** to 4***maxvar** instead of 2***maxvar.**

```
set maxvar 50 lrecl 200
```

partitions memory so that it can contain a maximum of 50 variables, but each observation may now occupy 200 bytes. If **lrecl** were not specified, it would automatically be set to 2*50=100. Setting the **lrecl**=200 makes it possible to store up to 50 **floats** or **longs**, or 25 **doubles**.

You may only specify **lrecl** on a **set maxobs** or **set maxvar** command. Since these commands can only be given when the current data set is empty, feel free to experiment. The worst that can happen is that you will receive the message "system limit exceeded - see manual", which indicates your request was ignored either because there is insufficient memory on your PC or because you attempted to set **lrecl** to less than 2***maxvar** or more than 8***maxvar.**

Messages and Return Codes

This appendix describes the return codes and messages produced by STATA.

1

You pressed Ctrl-Break. This is not considered an error.

3 no data set in use

You attempted to perform a command (such as merging or appending) which requires some data in memory. There are no data in memory.

4 No - data in memory would be lost

You attempted to perform a command which would substantively alter or destroy the data and the data have not been **saved** (or have not been **saved** since the last change). If you wish to continue anyway, add the **clear** option to the end of the command. Otherwise, **save** the data first.

5 master data not sorted
using data not sorted

Both the data set in memory and the data set on disk must be sorted by the variables specified in the varlist of **merge** before they can be **merged**. If the master data set is not sorted, **sort** it. If the using data set is not sorted, **use** it, **sort** it, and then **save** it.

6

Return code when string does not exist from **confirm existence**.

18 you must start with an empty data set
The command (e.g., **infile**) requires that no data be in memory. You may not append to existing data using **infile.** Instead, **drop _all**, **infile** the new data, and then **append** the previously existing data.

100 _____ required
Certain commands require a varlist, or an **in** range, or other elements of the language. The message specifies the required item that was missing from the command you gave. See the command's syntax diagram.

101 _____ not allowed
Certain commands do not allow an **if** exp, or other elements of the language. The message specifies which item in the command is not allowed. See the command's syntax diagram. You may not specify **in** or **if** when using **xp** data sets even when the command syntax would otherwise allow it.

102 too few variables specified
The command requires more variables than you specified. For instance, **plot** requires at least two variables. See the syntax diagram for the command.

103 too many variables specified
The command does not allow as many variables as you specified. For example, **tabulate** takes only one or two variables. See the syntax diagram for the command.

104 nothing to input
You gave the **input** command with no arguments. STATA will input onto the end of the data set, but there is no existing data set in this case. You must specify the variable names on the **input** command.

110 _____ already defined

The variable or value label has already been defined and you attempted to redefine it. This occurs most often with **generate.** If you really intend to replace the values, use **replace.** For value labels, if you intend to replace the label, first give the command **label drop** name.

111 _____ not found

no variables defined

The variable does not exist. You may have mistyped the variable´s name.

_____ not found in using data

You specified a varlist with **merge**, yet the variables on which you wish to merge are not found in the using data set, so the **merge** is not possible.

help for _____ not found

You requested **help** on a topic not found in the on-line help file. Type **help** for a menu of help items.

_____ ambiguous abbreviation

You gave a variable name an ambiguous abbreviation; the abbreviation could indicate more than one variable. Use a non-ambiguous abbreviation.

120 invalid %format

You specified an invalid %format. See **format**.

130 expression too long

You specified an expression that is too long for STATA to process. Break the expression into smaller parts.

131 not possible with test

Your requested **test** is non-linear in the variables. **test** tests only linear hypotheses.

132 Too many ´(´ or ´[´
Too many ´)´ or ´]´

You specified an expression with unbalanced parentheses or brackets.

133 unknown function _____()

You specified a function that is unknown to STATA. See "Expressions". Alternatively, you may have meant to subscript a variable, and accidentally used parentheses rather than square brackets.

190 request may not be combined with by

Certain commands may not be combined with **by**, and you constructed such a combination. See the syntax diagram for the command.

in may not be combined with by

in may never be combined with **by**. See description under **by**.

198 invalid syntax
_____ invalid
range invalid
_____ must be between ___ and ___
_____ invalid obs no
Obs nos. out of range
invalid filename
_____ invalid varname
_____ invalid name
multiple by´s not allowed
_____ found where number expected
on or off required

All items in this list indicate invalid syntax. These errors are often, but not always, due to typographical errors. STATA attempts to provide you with as much information as it can. Review the syntax diagram for the designated command.

In giving the message "invalid syntax" STATA is not very helpful. Errors in specifying expressions often result in this message.

199 unrecognized command

STATA failed to recognize the command, probably due to a typographical or abbreviation error.

201 may not drop _cons
may not rename _cons

_cons is a special STATA variable name designating the sum vector of a cross-product (**xp**). You are not allowed to **drop** or **rename** it.

202 invalid xp (___) - see manual

You attempted to designate the data in memory as an **xp** data set, but the criteria are not met. The number in parentheses explains which criterion was not met.
1: the matrix is not square.
2: the number of observations stored in the (1,1) element is less than or equal to zero.
3: The number of observations stored in the (1,1) element is not an integer.
4: Two symmetric, off diagonal elements are both equal to missing values.
5: Two symmetric, off diagonal elements are not equal.

210 request not possible on xp data

You requested something that would be possible on **data** but is not possible on **xp**. Non-linear expressions or commands such as **tabulate** and **plot** are likely suspects.

211 request requires xp data
You attempted to designate the **data** in memory as **data**, which it is already.

301 last regression not found
You typed **regress** without arguments, performed a **test** on the last regression, or attempted to use **_pred**, but there is no previous regression.

601 file _____ not found
The filename you have specified cannot be found. Perhaps you mistyped the name, or it may be on another diskette or directory.

602 file _____ already exists
You attempted to write over a file that already exists. STATA will never let you do this accidentally. If you really intend to overwrite the previous file, reissue the last command specifying the **replace** option.

603 file _____ could not be opened
The file, while found, failed to open properly. This error is unlikely to occur. You will have to review the DOS manual to determine why it occurred in this case.

604 spool already open
You attempted to open a **spool** file when one is already open. Perhaps you forgot you have the file open or forgot to close it.

610 file _____ not STATA format
The designated file is not a STATA format file. This occurs most frequently with **use**, **append**, and **merge**. You probably typed the wrong filename. Alternatively, you may ave the **encode** key set incorrectly.

699 "error writing file"

A fatal (for the file, not STATA) error occurred while writing the file. The file is now closed and STATA has given up. Review the DOS manual to determine why this happened.

900 no room to add more observations

There is no room in the current partition to add more observations; you are already at the maximum. See Appendix B, Memory Management in STATA. You might **save** the data, repartition, and then **use** it again.

901 no room to add more variables

There is no room in the current partition to add more variables. See 900 above.

902 no room to add more variables due to lrecl

There is no room in the current partition to add more variables due to an **lrecl** shortage. See 900 above.

920 too many macros

You specified a line containing too many macros, and after expansion of the macros the line exceeds 1000 characters. The line was ignored.

950 insufficient memory

There is insufficient memory in your PC to carry out the request. Consider dropping value labels, variable labels, or macros.

1000 system limit exceeded

In most cases you attempted to **set maxvar** or **set maxobs** outside limits that are physically possible on your PC. You will also receive this message if you specify the **lrecl** parameter with a value less than 2***maxvar** or more than 8***maxvar**. See

Appendix B, Memory Management in STATA. In all cases the request has been ignored and the partition was not changed.

You will also get this message if you attempt to go beyond any of STATA's preset limits, for instance, by specifying an expression with more than 200 operators or more than 50 constants or more than five **sum()** functions.

9xxx Various messages, all indicating system failure. You should never see such a message. If one occurs, **exit** STATA immediately and report the problem.

Other Messages

--more--

STATA never allows output to scroll off the screen (unless you explicitly **set display linesize 0** or **set more** to some small number). When --more-- occurs STATA waits for you to press any key on the keyboard before continuing. If you hold down the Ctrl key and press Break, STATA will terminate the command as soon as possible. If you have **set more** to some number other than **0**, STATA waits that many seconds and then acts as if you tapped some key other than Ctrl-Break.

Note: ___ missing values generated

The command resulted in the creation of the indicated number of missing values. Missing values occur when a mathematical operation is performed on a missing value, or when a mathematical operation is infeasible.

Note: File ____ not found

You specified the **replace** option on a command, yet no such file was found. The file was saved anyway.

Note: _____ is __ in using data but will be __ now

Occurs during **append** or **merge.** The first blank is filled in with a variable name and the second and third blanks with a storage type. For instance, you might receive the message "myvar is float but will be int now", meaning that myvar is already of type of **int** in the master data set, but that in the using data set a variable of the same name is found of type **float.** Thus, truncation could occur as the **using** data is copied into the master. You will only receive this message if truncation or rounding might occur.

label ____ already defined

Occurs during **append** or **merge.** The using data had a label definition for one of its variables, and a label was stored with the data set. A label with the same name was already defined. **append** and **merge** never replace data or labels. Thus, you are warned that the label already existed, and the previous definition was retained.

Note: hascons false

You specified the **hascons** option on **regress,** yet an examination of the data revealed that there is no effective constant in your varlist. STATA added a constant to the regression.

Appendix D

Methods and Formulae

Notation

Variables printed in lowercase and not boldfaced (e.g., x) are scalars. Variables printed in lowercase and boldfaced (e.g., $\mathbf{w}$) are column vectors. Variables printed in uppercase and boldfaced (e.g., $\mathbf{X}$) are matrices.

$\mathbf{1}$ is a column vector of 1′s.

$\mathbf{v}$ is a column vector of weights specified by the user. If no weights are specified, $\mathbf{v}=\mathbf{1}$.

$\mathbf{w}$ is a column vector of normalized weights. If no weights are specified or **noscale** was specified, $\mathbf{w}=\mathbf{v}$. Otherwise, $\mathbf{w}=(\mathbf{v}/(\mathbf{1}'\mathbf{v}))*(\mathbf{1}'\mathbf{1})$.

n is the effective number of observations, defined as $\mathbf{1}'\mathbf{w}$.

$\mathbf{x}$ is the vector of observations on the variable specified by the user.

Element-by-element multiplication of a vector is indicated by ".". For instance, $\mathbf{x}.\mathbf{x}$ denotes the column vector of the squares of $\mathbf{x}$.

$\mathbf{w}[i]$ is a scalar, the i-th element of the vector $\mathbf{w}$.

Summary Statistics

The number of observations is n. The sum of the weights is $\mathbf{1}'\mathbf{v}$.

Define $m1=\mathbf{1}'\mathbf{x}/n$, $m2=\mathbf{x}'\mathbf{x}/n$, $m3=\mathbf{x}'(\mathbf{x}.\mathbf{x})/n$, and $m4=\mathbf{x}'(\mathbf{x}.\mathbf{x}.\mathbf{x})/n$.

The mean: $m=m1$.

The variance: v=(m2-m1*m1)n/(n-1).

The standard deviation: s=sqrt(v).

The skewness: k=m3/sqrt(m2*m2*m2).

The kurtosis: u=m4/(m2*m2)

The p-th percentile is defined as follows: Let P=p/100*n. Let **W**[i]=**w**[1]+...+**w[i]. Find the first** index i such that **W**[i]>P. If **W**[i-1]==P then the percentile is (**w**[i-1]**x**[i-1]+**w**[i]**x**[i])/(**w**[i-1]+**w**[i]). Otherwise, the percentile is **x**[i].

Regression Statistics

The number of observations is n. The sum of the weights is **1′v**. Define c=1 if there is a constant in the regression and zero otherwise. Define k=number of right hand side (rhs) variables (including the constant).

Let **X** denote the matrix of observations on the rhs variables, **y** denote the vector of observations on the left hand side (lhs) variable, and **Z** denote the matrix of observations on the instruments. If the user specifies no instruments, then **Z=X.** In the following formulae, if the user specifies weights then **X′X**, **X′y**, **y′y**, **Z′Z**, **Z′X**, and **Z′y** are replaced by **X′DX**, **X′Dy**, **y′Dy**, **Z′DZ**, **Z′DX**, and **Z′Dy**, respectively, where **D** is a diagonal matrix whose diagonal elements are the elements of **w**. We suppress the **D** below to simplify the notation.

If no instruments are specified define **A** as **X′X** and **a** as **X′y**. Otherwise, define **A** as **X′Zinv(Z′Z)(X′Z)′** and **a** as **X′Zinv(Z′Z)Z′y** where inv() denotes the matrix inverse operator.

Appendix D

The coefficient vector **b** is defined as inv(**A**)**a**.

The total sum of squares tss equals **y′y** if there is no intercept and **y′y**-((**1′y**)(**1′y**)/n) otherwise. The degrees of freedom are
n-c.

The error sum of squares ess is defined as **y′y**-2**bX′y**+**b′X′Xb** if there is no intercept and as **y′y**-**b′X′y** otherwise. The degrees of freedom are n-k.

The model sum of squares mss is defined as tss-ess. The degrees of freedom are k-c.

The mean square error mse is defined as ess/(n-k).

The root mean square error rmse is defined as sqrt(mse).

The F-statistic fstat of k-c and t-k degrees of freedom is defined as (mss/(k-c))/mse if no instruments are specified. If instruments are specified and c=1 then fstat is defined as (**b**-**c**)′**A**(**b**-**c**)/(mse*(k-1)), where **c** is a vector of k-1 zeros and kth element **1**′y/n. Otherwise, fstat is undefined (missing value). (In this case you may use the **test** command to construct any F-test you wish.)

The R-square rsq is defined as 1-ess/tss if no instruments are specified and otherwise as 1/(1+(1/fstat)(n-k)/(k-1)).

The Adjusted R-square rbarsq is defined as 1-(1-rsq)(n-c)/(n-k).

The Standard error **s**[i] of **b**[i] is defined as sqrt(mse*(inv(A))[i,i]).

The t-statistic is defined as **b**[i]/**s**[i].

Hypothesis Testing

Let **Rb=r** denote the set of q linear hypotheses to be tested jointly.

The constrained estimate **q** is defined **b**+inv(**A**)**R**′ inv(**R**inv(**A**)**R**′)(**r**-**Rb**) and the corresponding F-statistic is (1/mse)(**b**-**q**)′**A**(**b**-**q**).

Contingency Table

Let the contingency table have r rows and c columns, and let n[i,j] represent the number of observations in the i-th row and j-th column. Let n[i,.] represent the total number of observations in row i, n[.,j] the total number of observations in column j, and n[.,.] the total number of observations.

The chi-square statistic with (r-1)(c-1) degrees of freedom is defined as the sum over all cells of

(n[i,j]-n[i,.]n[.,j]/n)^2/((n[i,.]n[.,j])/n[.,.])

Appendix E

Hardware Requirements

STATA runs on an IBM PC, PC/XT, PC/AT, or equivalent. STATA requires a computer with at least 256K of memory and two double density/double sided diskette drives or a single diskette and a fixed disk. Additional memory allows STATA to handle larger data sets.

If the computer includes an 8087 math co-processor STATA will use it. Some comparative timings, performed on an XT using a memory drive, are:

	w/o 8087	8087
set obs 7000		
gen n=_n	21.97	20.36
gen new=n*n	78.65	48.54
summarize	110.62	46.10
gen rand=uniform()	31.91	30.36
reg n new rand	242.72	79.84
Time to convert 50 observations on 26 variables	73.71	24.70
Sort 100 random variables	106.50	72.59
Take **log** of 500 numbers	20.48	5.50
Take **invnorm** of 500 numbers	181.19	23.92

Thus, heavily math dependent routines (.e.g, **summarize** and **regress**) run approximately 2 to 3 times faster. Complicated mathematical functions such as **invnorm()** speed up by a factor approaching 8.

Installation Instructions

The distribution diskette contains a file named INSTALL.DOC that tells you how to install STATA on your computer. Boot the system as you ordinarily would. If you have two diskette drives and no hard disk, place the distribution diskette in drive B. If you have a fixed disk place the distribution diskette in drive A. In the text below, lowercase **d** stands for the letter of the drive containing the distribution diskette.

Type:

TYPE d:INSTALL.DOC

and then follow the instructions. The distribution diskette may also contain updates to this manual. You can determine if there are by typing:

DIR d:*.UPD

The message "File not found" indicates there are no updates; otherwise you will obtain a list of one or more files. You can review the updates on the screen by using the DOS TYPE command. You may use the DOS PRINT command to make a printed copy for inclusion in this notebook.